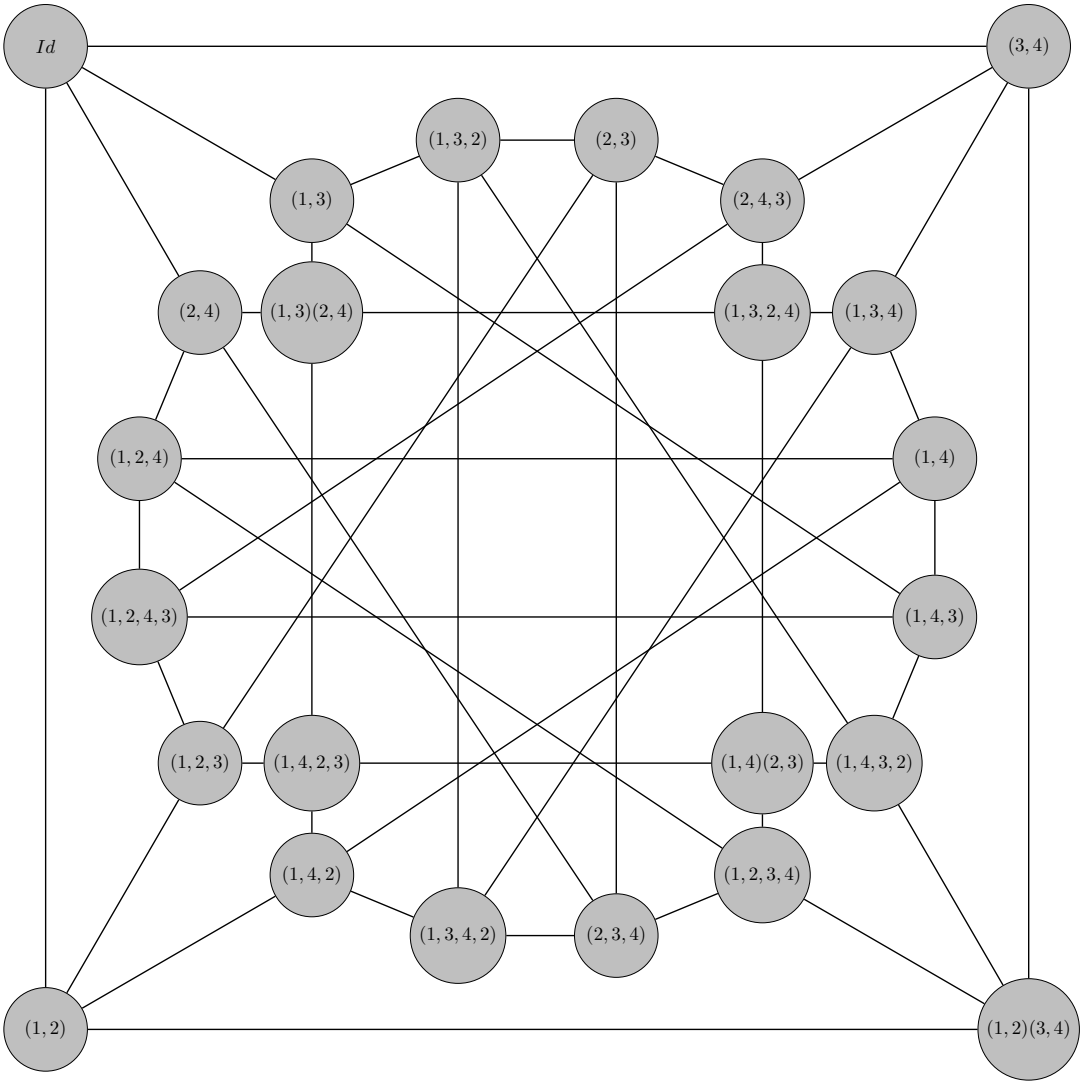
Els secrets dels trencaclosques de permutacions

Programació i teoria de grups

Turing i Taylor

29 de novembre



Agraïments

La realització d'aquest treball ha estat possible i s'ha fet molt més amena gràcies a una sèrie de persones que ens han ajudat i ens han oferit el seu suport.

En primer lloc, volem agrair a la nostra tutora. Ella ens va orientar cap a l'estudi de la teoria de grups i ens ha brindat el seu suport durant tot el treball, li agraïm també la seva paciència i dedicació. No podem no mencionar el matemàtic David Joyner i el també matemàtic Herbert Kociemba, tots dos van estar oberts a respodre els nostres missatges i a prendre'ns seriosament tot i que encara no som alumnes d'universitat. Finalment, volem agrair a les nostres famílies, i sobretot, als nostres pares, que en ocasions ens han oferit la seva ajuda amb els seus coneixements d'informàtica i matemàtiques.

Resumen

Hemos hecho un estudio de los juegos de permutaciones a través de la teoria de grupos con el objetivo de crear programas capaces de resolver el juego del 15, las torres de Hanoi o el Loopover de manera óptima. También hemos inventado nuestro propio juego de permutaciones y unas variantes adicionales (la de la tabla cuadrada se corresponde con el Loopover clásico). El proyecto se ha enfocado hacia la teoría de grupos porque, en un principio, partíamos de la hipótesis de que esta área de las matemáticas podía ayudarnos a comprender las particularidades de los juegos de permutaciones y a hallar heurísticas útiles para resolver el juego óptimamente. Nuestra herramienta de resolución se ha creado con el lenguaje de programación Python y con el sistema algebraico computacional SAGE. Finalmente hemos podido concluir que nuestra hipótesis era correcta, ya que combinar nociones de teoría de grupos (sobre todo grupos de permutaciones) y de programación ha sido de mucha utilidad durante la invención y programación de nuestro juego. Aun así, todavía se puede profundizar más en el proyecto ya que tanto el álgebra computacional como la programación orientada a juegos están en constante investigación y desarrollo.

Abstract

We have done a research about permutation puzzles through group theory with the objective of creating a program able to solve the 15 puzzle game, the Tower of Hanoi or the Loopover optimally. We have also invented our own permutation puzzle and some additional variations (the one of the square grid corresponds to the classical Loopover). The project has focused towards group theory because, initially, we started from the hypothesis that this mathematics area could help us to understand the particularities of permutation puzzles and to find useful heuristics to solve our game optimally. Our solving tool has been created with the programming language Python and the computational algebraic system SAGE. Finally, we have been able to conclude that our hypothesis was correct, because combining group theory (mainly permutation groups) and programming notions has been really useful during the invention and programming of our puzzle. Even so, it is still possible to delve deeper into the project since both computational algebra and game-oriented programming are in constant research and development.

Índex

Part I

Introducció

Qui sap què és un cub de Rubik? Qui sap quines matemàtiques té al darrere? Aquestes dues preguntes tenen dues respostes antagòniques: mentre que gairebé tothom sap què és un cub de Rubik, quasi ningú no coneix les matemàtiques que amaga. Aquesta última resposta ens va incitar a descobrir-ho, i són els descobriments i avenços que hem fet relacionats amb el cub de Rubik i altres puzzles similars allò que trobareu plasmat en aquest treball de recerca.

Partíem de la hipòtesi que la resolució d'aquesta mena de trencaclosques és modelable amb alguna teoria matemàtica. La tutora del treball ens va orientar envers la teoria de grups i el llenguatge de programació Python, que permet desenvolupar cerques exhaustives de solucions. Crèiem que la teoria de grups ens ajudaria a desenvolupar heurístiques o estratègies de solució que podríem aplicar a un llenguatge de programació per tal de filtrar les cerques i trobar solucions amb el mínim de moviments possibles; a més, la programació és un àmbit que trobàvem adient per aplicar els coneixements apresos. En aquest treball també hi trobareu el resultat obtingut i el camí que hem hagut de fer per poder-hi arribar. Les nostres hipòtesis inicials eren, per tant, que la teoria de grups ens permetria conèixer matemàticament el funcionament d'aquest tipus de trencaclosques i que ens ajudaria a crear un programa informàtic que en resolgués algun.

Els objectius d'aquest treball de recerca són, doncs, ampliar els nostres coneixements matemàtics per tal de poder veure'ls reflectits en un programa informàtic que resolgui un trencaclosques matemàtic de manera òptima. S'ha intentat, també, que aquest programa no sigui excessivament concret sinó ampli, per poder treballar així tot el ventall de conceptes apresos en l'etapa anterior en una sola peça, de manera que quedin reflectits els coneixements sobre grups, permutacions, etc., que es treballen al llarg del treball.

Un treball de recerca, però, no sorgeix només a partir d'una pregunta i un objectiu. Cal una predisposició de l'alumne envers l'àrea de coneixement afectada, i també la convicció que la manera com es treballen els diferents aspectes dins el treball és l'adequada. Per què, doncs, hem triat trencaclosques matemàtics semblants al cub de Rubik i no al sudoku? I per què creiem que l'aplicació informàtica és la més apropiada?

D'entrada, cal ressaltar que als autors ens agraden molt els jocs de tota mena, i no només això: creiem que són una eina didàctica molt útil que pot servir per aprendre coneixements d'àrees molt diverses. Ara bé, si hi ha una branca del coneixement que manté una relació especial amb jocs de tot tipus aquesta és, sens dubte, la branca de les matemàtiques. Molts passatemps i jocs de taula poden servir de punt de partida per treballar conceptes matemàtics com ara la probabilitat, la combinatòria o la teoria de grups, i molts es desenvolupen a partir d'alguna mena de suport geomètric: graelles de triangles, de quadrats, hexagonals… Les matemàtiques són, doncs, un pilar fonamental del funcionament dels passatemps i jocs més variats.

En aquest treball ens hem centrat en aquells trencaclosques matemàtics el funcionament dels quals es basa en moviments seqüencials. Es comença en una certa posició i l'objectiu és, fent successius moviments, arribar a una posició final. Per això, com es veurà en el cos del treball, les posicions i moviments es poden expressar com a permutacions que conformen un grup matemàtic. El fet que es puguin expressar d'aquesta manera va ser, també, una de les motivacions a l'hora de començar aquest projecte i un fet decisiu a l'hora de decantar-nos per aquesta mena de jocs i no per una altra. El perquè de rellevància que creiem que té la teoria de grups s'explica a continuació.

La teoria de grups és un tema molt interessant que, tot i això, no té cabuda en els currículums de matemàtiques de batxillerat i que per tant és completament desconegut pels alumnes. Volíem, doncs, ampliar les nostres perspectives matemàtiques i apropar, a qui hi estigués interessat, un àmbit que és desconegut per la majoria. Aquesta labor d'acostament i reconeixement de la teoria de grups, mostrant també les seves aplicacions d'una manera divertida i que es pugui relacionar amb jocs que tothom ha experimentat (tot i que potser no aprofundint-hi matemàticament) és un altre objectiu del treball.

A més del didactisme dels jocs i del seu component lúdic, un altre avantatge que tenen és que es poden fer programes informàtics que permeten jugar-hi sense necessitat de tenir el joc físicament. Des de ja fa uns anys, és difícil trobar un ordinador que no dugui incorporats diferents programes per jugar a jocs com el solitari o el go, i actualment és possible jugar a una multitud de jocs com ara els escacs o l'Scrabble en línia amb altres jugadors o, fins i tot, contra el programa mateix.

Els ordinadors i programes actuals poden processar quantitats ingents d'informació en un interval de temps molt petit, i és per això que són molt útils per a resoldre problemes matemàtics feixucs o treballar amb nombres descomunals i immanejables.

Per això, és una eina de treball imprescindible per a qualsevol científic o matemàtic del dia d'avui, i això fa que actualment els exercicis matemàtics que s'acostumen a fer a les escoles i instituts de resoldre equacions per resoldre equacions i factoritzar polinomis per factoritzar polinomis no acabin de tenir sentit.

És clar que, a les escoles, els alumnes no han de tenir una actitud passiva sinó activa, però quan això deriva, com passa sovint, en la resolució d'equacions com a fi en si mateixa, els alumnes s'obliden de què significa allò que estan fent i de quina utilitat pot tenir. La informàtica i la programació poden servir per, quan els alumnes ja han entès com es fa una cosa en qüestió, permetre'ls veure més ràpidament què és allò que està passant i proposar-los reptes que sense informàtica no podrien resoldre. Un exemple d'això és *GeoGebra*[1], que pot servir per representar funcions complicades amb precisió i entendre'n el comportament amb facilitat. Nosaltres mateixos el fem servir sovint i trobem que, aplicant-lo de manera adequada, pot ser una gran eina de treball que ajudi a la comprensió d'alguns continguts i millori la predisposició dels alumnes de cara a estudiar d'una manera diferent. També pot ajudar a trobar un sentit o una aplicació a allò que s'està treballant. Al cap i a la fi, si es tenen aquests recursos, per què no fer-los servir?

En aquest treball, doncs, hem utilitzat la programació perquè s'adapta bé als jocs, permet treballar amb nombres i matrius feixugues i ens permet veure el resultat d'aplicar la teoria de grups a alguns trencaclosques matemàtics, cosa que, si no, seria complicada. Concretament, els llenguatges que hem utilitzat han estat Python i SAGE. Python és un dels llenguatges de programació més populars arreu del món, de programari lliure, i fou inventat per Guido von Rossum, el 1991[2]. És molt versàtil i fàcil d'utilitzar, i per això és un bon llenguatge per a aquells que s'inicien en el món de la programació. D'altra banda, SAGE és un sistema algebraic computacional[3], la primera versió del qual va ser publicada el 2005 per, entre d'altres, William A. Stein[4]. Un dels seus objectius era oferir una alternativa de codi obert a altres sistemes d'àlgebra computacional amb propietari i codi tancat. Està escrit en Python i Cython[5], i proporciona una interfície en Python des de

[1] *GeoGebra* és un programa de geometria dinàmica en línia, que combina geometria, àlgebra i càlcul. Vegeu https://www.geogebra.org/.

[2] La seva pàgina web és https://www.python.org/.

[3] Un sistema algebraic computacional és un programa amb la capacitat de manipular expressions matemàtiques de manera semblant a les calculadores tradicionals. A diferència d'aquestes, però, poden treballar simbòlicament i no només numèricament.

[4] La seva pàgina web és https://www.sagemath.org/.

[5] Cython és un llenguatge de programació, compilador optimitzador de Python. Vegeu https://cython.org/.

la qual es pot accedir a diversos paquets (per exemple, GAP[6] per a la teoria de grups). Per accedir tant a Python com a SAGE, hem utilitzat bàsicament SageMathCloud[7] (també anomenat CoCalc), una plataforma integrant del projecte SAGE en el núvol que permet compartir documents amb altres usuaris. Justament per això, ens ha resultat molt útil.

L'estructura del treball, així com també la metodologia giren entorn, doncs, a dos blocs interrelacionats. En primer lloc, hi ha una part important on es treballen els diversos conceptes matemàtics necessaris per al nostre propòsit. En aquest apartat s'ha obtingut la informació mitjançant la cerca en llibres i alguna pàgina web, alguns de caire matemàtic més general i d'altres orientats més específicament al vessant matemàtic dels trencaclosques estudiats. I, en segon lloc, hi ha una segona part dedicada al procés de creació del nostre programa i de caire, així doncs, pràctic. El codi dels programes desenvolupats en aquesta segona part es troba a l'annex, així com també en una pàgina web feta pels autors (https://sites.google.com/insperecalders.cat/trencaclosques-permutacions) que conté recursos relacionats amb el treball. S'ha de dir, però, que ja en la primera part s'introdueixen exemples de programació amb SAGE, per tal de, en aquesta segona part, tenir-ne una petita noció inicial. Tanquen el treball les conclusions que n'hem pogut extreure i la bibliografia consultada per elaborar-lo.

Esperem que aquest treball us resulti tan interessant com ho ha sigut per a nosaltres a l'hora d'elaborar-lo i que gaudiu i aprengueu moltes coses tot llegint-lo.

[6] GAP és un acrònim de *Groups, Algorithms and Programming*, i és un sistema algebraic computacional. Vegeu https://www.gap-system.org/.

[7] La seva pàgina web és https://cocalc.com/.

Part II

Continguts teòrics

En aquesta part del treball es presenten els continguts teòrics treballats, que ens permetran entendre matemàticament el funcionament dels trencaclosques de permutacions i que són necessaris per tal de poder desenvolupar un programa com el desitjat.

1 Trencaclosques de permutacions

Com ja s'ha comentat, en aquest treball ens centrem en l'estudi d'un tipus concret de trencaclosques des d'una perspectiva matemàtica. Existeixen moltes i diverses classificacions de les diverses menes de trencaclosques, però nosaltres hem triat seguir una classificació proposada per David Joyner, que els anomena "trencaclosques de permutacions" [1].

Un trencaclosques de permutacions, doncs, és un trencaclosques que compleix les següents característiques:

- És un joc per a una sola persona.
- L'objectiu és, partint d'una posició desordenada, arribar a una certa ordenació de les peces mitjançant seqüències de moviments.
- En cada posició hi ha un nombre finit de moviments possibles, els quals no depenen de l'existència de moviments anteriors (com sí que passa, per exemple, en els escacs, en els quals el rei no es pot enrocar si s'ha mogut prèviament).
- Es pot predir quina serà la posició després d'haver fet un cert moviment.
- Tot moviment s'ha de poder "desfer", és a dir, després d'haver fet un moviment es pot tornar a la posició prèvia amb un altre moviment.

Sabent això, podem veure que s'inclouen en aquest grup de trencaclosques jocs tan coneguts (i dels quals es parlarà a continuació) com les Torres de Hanoi, el joc del 15 o el cub de Rubik, mentre que en són exclosos d'altres, com ara el joc en línia 2048[8]

[8] Es tracta d'un videojoc desenvolupat per Gabrielle Cirulli el 2014. Consisteix en una graella 4×4, les caselles de la qual es van omplint amb nombres a mesura que es desenvolupa la partida. Es comença amb el número 2 en dues d'aquestes caselles. S'han de fer lliscar cap amunt, cap avall o cap als costats mitjançant les fletxes del teclat totes les caselles plenes per tal de fer-les xocar. Quan dos nombres iguals entren en

(ja que ni l'objectiu és arribar a una posició concreta ni es poden predir els moviments possibles posteriors a cada moviment) o els sudokus[9] (perquè no es tracta d'arribar a una posició coneguda sinó de descobrir-la i això no s'aconsegueix amb seqüències de moviments de cap mena de peces). Gràcies a aquest funcionament dels trencaclosques de permutacions i com ja indica el seu nom, és molt senzill expressar una certa situació o moviment mitjançant les permutacions.

Seguidament, doncs, ens fixarem en els exemples anteriors per tal d'acabar de fer-nos una idea d'aquesta mena de trencaclosques.

1.1 Alguns exemples de trencaclosques de permutacions

1.1.1 Les Torres de Hanoi

Es tracta d'un enigma plantejat el 1883 pel matemàtic francès Édouard Lucas. Com es veu a sota, en el problema hi ha tres columnes, la primera de les quals conté un cert nombre de discs apilats, cadascun més petit que l'anterior. Es tracta d'aconseguir moure tots els discs a l'última columna mitjançant una sèrie de moviments. En cada moviment es pot moure només el disc superior d'un cert pal fins a un altre pal, deixant-lo sobre de tots els discs que aquest segon pal ja tingués. Un disc més gran mai no pot quedar sobre d'un de més petit.

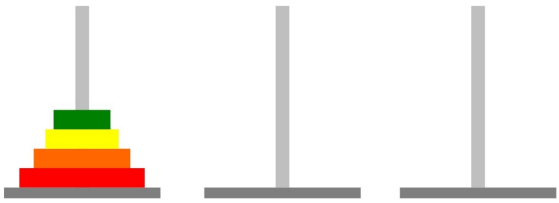


Figura 1.1: Posició inicial de les Torres de Hanoi amb 4 discs

Aquest problema té una solució algorítmica que es pot generalitzar per qualsevol

contacte se sumen (en el cas de dues caselles amb el 2, el número resultant serà el 4). El nou nombre no pot sumar-se amb cap altre en aquesta mateixa jugada. A més, després de cada moviment s'omple una de les caselles que romanien buides a la quadrícula, ja sigui amb el número 2 o 4, les dues xifres més inferiors, de manera que no es pot predir exactament quina serà la posició després de cada jugada. La seva pàgina web és https://2048game.com/.

[9] El sudoku és un passatemps que durant la dècada del 1980 es va escampar primer pel Japó i després ràpidament per la resta del món. En una graella de 9×9 les caselles estan agrupades en 9 regions de 3×3. Cal col·locar a cada fila, columna i regió els nombres de l'1 al 9 (un a cada casella) sense repetir-ne cap, partint d'una posició en què se saben només alguns nombres. Té múltiples variants (graelles més grans o més petites, condicions extra que cal respectar...). En aquest sentit és digne d'esment el treball de David Nacin *Math-Infused Sudoku*, publicat el 2019 per l'*American Mathematical Society* (vegeu https://bookstore.ams.org/mbk-123).

nombre de discs, i que us animem a trobar[10].

1.1.2 El joc del 15

No se sap amb certesa qui va ser el creador d'aquest trencaclosques, però en tot cas va assolir una enorme popularitat quan, el 1878, el matemàtic i escaquista estatunidenc Sam Loyd va prometre un premi de 1000 dòlars a qui aconseguís resoldre la posició inicial que es veu a sota a l'esquerra. Com es pot observar, aquest trencaclosques consisteix en un tauler quadrat amb 16 espais on hi ha 15 peces, numerades de l'1 al 15, i un espai buit. En cada moviment s'ha de moure al forat una de les peces que hi són adjacents horitzontalment o verticalment, fins a arribar a tenir els 15 nombres ordenats com es veu a sota a la dreta.

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

$\longrightarrow$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Figura 1.2: La posició inicial del joc del 15 de Sam Loyd i la posició que s'ha d'aconseguir

Us animem a jugar a aquest trencaclosques i a provar d'arribar a la posició de Sam Loyd (podeu utilitzar la nostra pàgina web: https://sites.google.com/insperecalders.cat/trencaclosques-permutacions/joc-del-15/jugar-hi), de la qual es tornarà a parlar més endavant (vegeu el punt 4.4).

1.1.3 El cub de Rubik

Aquest trencaclosques va ser inventat per l'arquitecte hongarès Erno Rubik durant la dècada de 1970. Es tracta d'un cub, cadascuna de les cares del qual està pintada d'un color diferent al de les altres cares i subdividida en 9 subcubs més petits. Així, hi ha subcubs que tenen 3 cares visibles (anomenats vèrtexs), subcubs que només en tenen 2 (anomenats arestes) i d'altres connectats als eixos que permeten realitzar els moviments, que tenen només una cara visible i s'anomenen centres. Partint d'una posició desordenada

[10] Podeu jugar a les Torres de Hanoi a la nostra pàgina web (https://sites.google.com/insperecalders.cat/trencaclosques-permutacions/torres-de-hanoi/jugar-hi) i consultar aquesta solució a l'annex.

dels subcubs cal aconseguir recompondre totes les cares. Els moviments permesos són les rotacions de $90°$ de cadascuna de les capes exteriors (les cares) i de les interiors.

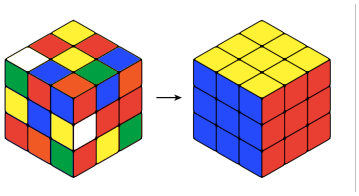


Figura 1.3: El cub de Rubik desordenat i resolt.

El cub de Rubik es va estendre ràpidament, assolint una enorme popularitat, i actualment n'existeixen nombroses variants: amb més o menys subcubs a cada cara (4, 16, 25 subcubs...) i amb altres formes (tetraèdrics, prismàtics no cúbics, dodecaèdrics...). Podeu jugar-hi a la nostra pàgina web (https://sites.google.com/insperecalders.cat/trencaclosques-permutacions/cub-de-rubik-jugar-hi).

Com ja s'ha dit, els moviments i les posicions d'aquests trencaclosques es poden expressar fàcilment com a permutacions. Per treballar amb comoditat amb les permutacions, però, ens anirà bé conèixer primer la teoria de grups, que veurem a continuació.

2 Una mica d'història...

La teoria de grups va començar a treure el cap quan Joseph-Louis Lagrange (1736-1813) estudiava la resolució d'equacions de grau n, un estudi que no va donar els resultats que ell esperava però on s'entreveuen els primers resultats sobre grups de permutacions. Lagrange va demostrar un teorema sobre la cardinalitat dels grups i els seus subgrups, teorema que ara rep el seu nom i que trobareu a l'annex. Tot i això, ell encara no utilitzava el llenguatge propi de la teoria de grups.

Poc més tard, Carl Friedrich Gauss (1777-1855), en la secció dedicada a l'estudi de les formes $ax^2 + 2bxy + cy^2$ del seu llibre *Disquisitiones Arithmeticae*, va definir la composició de formes i va estudiar les relacions i les classes d'equivalència, que com va observar tenen l'estructura d'un grup commutatiu, cosa que veurem al final del treball.

És, però, quan Évariste Galois (1811-1832) investiga la resolució d'equacions de grau major que 4, que la teoria de grups pren forma, introduint Galois, entre d'altres coses, les classes laterals, que també són presents en aquest treball.

A partir de llavors la teoria de grups es va anar desenvolupant de la mà de diversos autors: Cayley (1821-1895), von Dyck (1856-1934), etcètera, fins a arribar al punt que avui té nombroses aplicacions en cristal·lografia, criptografia, en l'estudi de les arrels d'un polinomi i, és clar, en la resolució de trencaclosques matemàtics.

3 La teoria de grups

Els grups matemàtics es defineixen com a conjunts (no buits) G amb una operació interna associada, que es denota amb $\cdot$ habitualment. Els grups han de complir també 3 axiomes[11]:

1. La propietat associativa, de manera que $g \cdot (g' \cdot g'') = (g \cdot g') \cdot g''$.
2. L'existència d'un sol element neutre que anomenarem e tal que $g \cdot e = g = e \cdot g$.
3. L'existència d'un element invers de cada g (denotat g^{-1}) de manera que $g \cdot g^{-1} = e = g^{-1} \cdot g$. El fet que $g \cdot g^{-1} = g^{-1} \cdot g$ implica que l'invers de l'invers de g és g.

Cal destacar que l'operació ha de ser interna al grup, de manera que $\forall g, g' \in G$, es compleix que $g \cdot g' \in G$. Per tant, quan s'opera amb qualsevol parell d'elements d'un grup, se n'obté un altre que també hi pertany.

Dins dels grups podem parlar dels anomenats grups abelians o commutatius, que són aquells que també compleixen la propietat commutativa tal que $g \cdot g' = g' \cdot g$.

Exemple 1. *Un possible exemple de grup abelià és el dels nombres enters $\mathbb{Z}$ (conjunt d'elements) amb la suma (operació). Podem exemplificar l'existència del seu element neutre $e = 0$ fent el càlcul de $1 + 0 = 1$; la de l'invers, que en aquest cas també és l'oposat, amb $2 + (-2) = 0$; veure la propietat associativa amb un possible exemple que seria $(1 + 2) + 3 = 1 + (2 + 3)$; i exemplificar també, com a grup abelià que constitueix, el compliment de la propietat commutativa amb el simple $3 + 2 = 2 + 3$.*

Exemple 2. *El conjunt de moviments del pla que deixen fix un triangle equilàter amb l'operació composició també forma un grup, que en aquest cas no és abelià. Aquest grup, anomenat diedral*[12]*, està format per les tres simetries respecte de les bisectrius i les rotacions de $0°$ (l'element neutre), $120°$ i $240°$ al voltant de l'incentre.*

[11] Es pot observar que es poden afeblir el segon i tercer axiomes; n'hi ha prou dient que $g \cdot e = g$ i que $g \cdot g^{-1} = e$. Sabent això, veiem que $e = g \cdot g^{-1} = g^{-1} \cdot (g \cdot g^{-1}) \cdot (g^{-1})^{-1} = g^{-1} \cdot g \cdot (g^{-1} \cdot (g^{-1-1})) = g^{-1} \cdot (g \cdot e) = g^{-1} \cdot g$, o sigui, que $g^{-1} \cdot g = e$. Ara podem demostrar que $e \cdot g = g$: si $g \cdot e = g$, aplicant l'invers de g a dreta i esquerra queda que $g^{-1} \cdot (g \cdot e) \cdot g^{-1} = g^{-1} \cdot (g \cdot g^{-1})$ i, per tant, $(g^{-1} \cdot g) \cdot g^{-1} = g^{-1} \cdot e$, d'on veiem que $e \cdot g^{-1} = g^{-1}$, és a dir, $e \cdot g = g$.

[12] Es parla a bastament dels grups diedrals a l'annex.

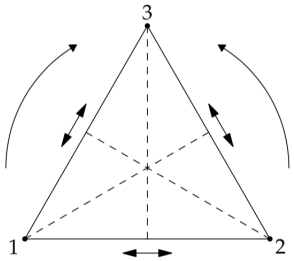


Figura 3.1: Els moviments del grup diedral d'un triangle equilàter.

Com s'acaba de veure, no tots els grups són abelians. De fet, en general, els grups de permutacions que anirem tractant al llarg del treball no ho són (només en cas de tenir menys de 3 elements). Això es deu al fet que l'operació d'aquest grup és a la composició de funcions (com en l'exemple anterior) i aquesta no compleix la propietat commutativa.

També és important saber que un conjunt no forma un grup associat a qualsevol operació, un exemple d'això pot ser l'incorrecte grup dels nombres naturals amb la suma. Per a demostrar-ho no cal més que fixar-se en les propietats universals dels grups destacades anteriorment i adonar-se que en aquest cas no existeix tal invers per a cada element de G.

3.1 Propietat de cancel·lació

Aquesta propietat consisteix en el simple fet que, si $a \cdot b = c \cdot b$, aleshores $a = c$ (cosa que es demostra aplicant l'invers de b a la dreta de la primera expressió).

A partir d'aquesta propietat es demostren la unicitat de l'element neutre de grup i la de l'invers de cada element del grup, com s'observa a continuació.

Demostració.

1. Suposem que un grup G pot tenir més d'un element neutre $e_1, e_2 \in G$, llavors han de verificar que $g \cdot e_1 = g = e_1 \cdot g$ i que $g \cdot e_2 = g = e_2 \cdot g$. Es pot veure que $e_1 \cdot g \cdot g^{-1} = e_2 \cdot g \cdot g^{-1}$, per tant $e_1 = e_2$. Hi ha, doncs, un sol element neutre.

2. Ara imaginem que un element $g \in G$ posseeix més d'un invers, als quals anomenarem $h, j \in G$. En aquest cas hauran de verificar que $(j \cdot g) \cdot h = j \cdot (g \cdot h)$ degut a la propietat associativa, de manera que $e \cdot h = j \cdot e \Rightarrow j = h$. Queda així demostrada la unicitat de l'invers.

□

3.2 Ordre o cardinalitat

L'ordre d'un grup, anomenat també com a cardinalitat[13], constitueix el nombre d'elements del mateix. La cardinalitat d'un grup sol expressar-se com a $|G|$. Si G és un grup finit el seu ordre es correspondrà llavors amb el seu nombre d'elements, i per als grups amb infinits elements direm que el seu ordre és infinit, $|G| = \infty$. Relacionat amb l'anterior, l'element g del grup G té ordre (denotat com a $ord(g)$) igual a l'enter positiu més petit p de manera que $g^p = e$ si existeix. Si no existeix direm que l'ordre de g és infinit.

Exemple 3. *Per a aclarir aquest darrer punt ens fixarem ara en els possibles moviments de rotació d'un cub de Rubik. Anomenant R el moviment de rotació de 90º de la cara que es disposa a la dreta, podrem identificar 4 rotacions diferents, és a dir 4 elements. Aquests serien el gir de $0°$ (la identitat), i les rotacions múltiples de $90°$, que anomenarem R, R^2 i R^3. Només existeixen aquests elements perquè una vegada es vol arribar a R^4 tornen a repetir-se els elements que ja han aparegut, de manera que $R^4 = e, R^{-1} = R^3$, etc. Com a conclusió, podrem afirmar que els moviments de rotació de 90º d'una cara del cub de Rubik tenen ordre 4.*

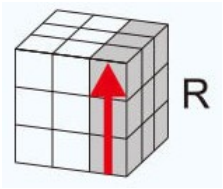


Figura 3.2: Moviment R del cub de Rubik.

Per definir un grup de rotacions i per saber l'ordre d'un grup amb SAGE, ho farem com es veu a sota:

```
sage: D=CyclicPermutationGroup(4)
sage: D.order()
    4
```

4 Permutacions

Per tal de definir amb precisió una permutació, convé recordar (o explicitar) què són les funcions i de quines menes n'hi ha.

[13] El concepte de cardinalitat també fa referència al nombre d'elements d'un conjunt.

Una funció és una correspondència que associa a cada element a d'un conjunt A (dit conjunt d'entrada) un sol element b d'un conjunt B (dit conjunt de sortida), denotada $f : A \to B$. Es diu que b és la imatge de a i a és l'antiimatge de b, i s'escriu $b = f(a)$.

Les funcions es poden classificar segons si són injectives o no i segons si són exhaustives o no. Una funció és injectiva si cada element del conjunt de sortida té com a màxim una antiimatge. D'altra banda, és exhaustiva si cada element del conjunt de sortida té almenys una antiimatge. Les funcions que són injectives i exhaustives alhora s'anomenen bijectives i són, doncs, aquelles en què el conjunt d'entrada té tants elements com el de sortida (és a dir, tenen la mateixa cardinalitat), relacionats un a un.

Les permutacions són funcions bijectives en què el conjunt d'entrada i de sortida són el mateix i és un conjunt finit amb n elements. S'acostumen a denotar els elements del conjunt com a 1, 2, fins a n, i les permutacions amb lletres gregues: σ, τ... Es poden entendre les permutacions, i així es fa per aplicar-les als trencaclosques de permutacions, com aplicacions que *envien* cada element a la seva imatge: sigui $1 \leq i \leq n$ i $\sigma(i) = j$, es pot dir que σ envia i a j (i on era j), j a $\sigma(j)$ (j on era $\sigma(j)$), etcètera.

Les permutacions són funcions que no es poden sumar o multiplicar, però sí que es poden composar. Habitualment, i també en aquest treball, es denota la composició de σ amb τ com $\tau \circ \sigma$, on s'aplica primer σ i, a continuació, τ (la composició s'efectua, doncs, de dreta a esquerra)[14]. Com que les permutacions no es poden multiplicar, sovint s'utilitza la notació multiplicativa per referir-se a la composició: $\tau \circ \sigma = \tau\sigma$ i $\sigma \circ \sigma \circ \sigma = \sigma^3$, etcètera.

La permutació en què cada element és la seva pròpia imatge s'anomena *identitat*, i es denota amb Id.

L'invers d'una permutació υ, és a dir aquella permutació que composada amb υ dona la identitat, s'acostuma a denotar amb υ^{-1} (igual que l'invers d'un element d'un grup), de manera que $\upsilon^{-1} \circ \upsilon = Id$. L'invers d'una permutació $\tau\sigma$ consistirà, per tant, en desfer, lògicament en l'ordre invers en què s'ha dut a terme, tot allò que facin σ i τ composades i, per tant, $(\tau\sigma)^{-1} = \sigma^{-1}\tau^{-1}$ i, de la mateixa manera, $(\tau^{-1}\sigma)^{-1} = \sigma^{-1}\tau$.

Exemple 4. *Siguin σ, τ dues permutacions. Aleshores, es defineix τ^σ com $\sigma^{-1}\tau\sigma$. Aquesta combinació de permutacions s'anomena **conjugat** de τ per σ[15], i ens serà útil més tard.*

A un conjunt de n elements s'hi poden aplicar $n!$ permutacions diferents, ja que la imatge del primer element pot ser qualsevol element; la del segon, qualsevol

[14] Cal recordar que la composició de funcions (i, per tant, de permutacions) no és commutativa.

[15] El conjugat de τ per σ s'expressa amb la notació exponencial perquè es compleixen les propietats de les potències: $(\sigma\tau)^\upsilon = \sigma^\upsilon\tau^\upsilon$ i $\sigma^{\tau\upsilon} = \sigma^\tau\sigma^\upsilon$.

excepte la del primer element, etcètera. Per això, el nombre de permutacions possibles és $n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 2 \cdot 1 = n!$. Aquest conjunt de permutacions associat a la composició constitueix un grup anomenat *grup simètric de permutacions*, i es denota amb S_n ($|S_n| = n!$ degut al que s'acaba de raonar). Acabem de veure que una de les permutacions que formen part del grup és la identitat, que serà, doncs, l'element neutre, i que les permutacions tenen invers, ja que són funcions bijectives. A més a més, la composició és una operació interna al grup, ja que en composar dues permutacions amb el mateix conjunt d'entrada (i, per tant, de sortida) se n'obtindrà una altra també amb aquest conjunt d'entrada, i és associativa, ja que $(\upsilon \circ \tau) \circ \sigma$ està definida en el mateix conjunt que $\upsilon \circ (\tau \circ \sigma)$ i la imatge de a serà igual en els dos casos:

$$((\upsilon \circ \tau)\sigma)(a) = (\upsilon \circ \tau)(\sigma(a)) = \upsilon(\tau(\sigma(a))), \text{ i}$$
$$(\upsilon \circ (\tau \circ \sigma))(a) = \upsilon((\tau \circ \sigma)(a)) = \upsilon(\tau(\sigma(a))).$$

S_n és, doncs, un grup en tota regla.

4.1 Expressió d'una permutació

A continuació, s'exposen les diverses maneres d'expressar una permutació i una mateixa permutació σ expressada mitjançant cadascuna de les maneres[16].

1. Amb una matriu de 2 files i tantes columnes com la cardinalitat del conjunt sobre el qual s'apliqui la permutació. A la fila de sobre s'hi escriuen els elements ordenats i, a la de sota, les seves imatges. És la més utilitzada.

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 5 & 1 & 3 \end{pmatrix}$$

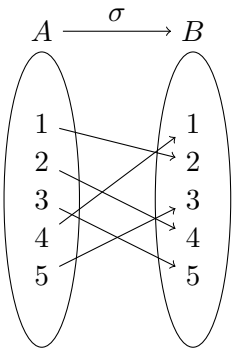
   A vegades, per tal d'abreujar, s'escriu només la fila de sota, entre claudàtors i amb els elements separats per comes. És una de les notacions que utilitza SAGE.

$$\sigma = [2, 4, 5, 1, 3]$$

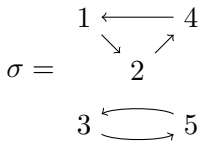
```
sage: s=Permutation([2,4,5,1,3])
```

[16] També hi ha altres maneres menys conegudes d'expressar una permutació. Una d'aquestes podria ser amb una matriu de 0 i 1 amb tantes files i columnes com elements, on, si i i j són una fila i una columna qualssevol respectivament, s'escriu 1 si i només si $\sigma(i) = j$.

2. Amb un diagrama que mostra el conjunt d'entrada i el de sortida (que són el mateix). Els elements del conjunt d'entrada s'uneixen a les seves imatges amb fletxes.



3. Amb un graf, representant els elements units a la seva imatge amb una fletxa.



En un joc de 5 peces en què els moviments que permetés consistissin en intercanviar diverses peces, la permutació anterior indicaria que la peça situada al lloc 1 es mou al 2; la del 2, al 4; la del 3, al 5; la del 4, a l'1, i la del 5, al 3.

Havent explicat la notació de les permutacions, podem posar un exemple de grup simètric de permutacions.

Exemple 5. *El grup de permutacions (amb la composició) que actua en un conjunt de* 3 *elements,* S_3*, conté les permutacions següents:*

$$\begin{pmatrix}1&2&3\\1&2&3\end{pmatrix}, \begin{pmatrix}1&2&3\\1&3&2\end{pmatrix}, \begin{pmatrix}1&2&3\\2&1&3\end{pmatrix}, \begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}, \begin{pmatrix}1&2&3\\3&1&2\end{pmatrix}, \begin{pmatrix}1&2&3\\3&2&1\end{pmatrix}$$

Amb SAGE, aquest mateix grup es defineix de la següent manera:

```
sage: S=SymmetricGroup(3)
```

Si es vol que SAGE mostri els elements del grup, es fa com es veu a sota[17]:

```
sage: S.list
    [(), (1,3,2), (1,2,3), (2,3), (1,3), (1,2)]
```

[17] SAGE denota les permutacions mitjançant la descomposició per cicles. Vegeu la descomposició per cicles a 4.3.

4.2 Òrbites i ordre d'una permutació

Donada una permutació σ, un element i d'aquesta permutació serà fix si $\sigma(i) = i$. D'altra banda, sigui j un element no fix, si apliquem una permutació diverses vegades successives (és a dir, la composem amb si mateixa varis cops) arribarà un moment en què, donat que el conjunt en la qual l'apliquem és finit, la imatge de j ja haurà aparegut prèviament: $\sigma^k(j) = \sigma^l(j)$, on definim que $l > k$. Aquest element que es repeteix per primera vegada serà j, ja que, composant $\sigma^{-k}(j)$ a banda i banda s'obté

$$\sigma^{-k}(j)\sigma^k(j) = \sigma^{-k}(j)\sigma^l(j) \Rightarrow j = \sigma^{l-k}(j)$$

L'òrbita de l'element j sota la permutació σ serà el conjunt d'elements que hem anat obtenint fins a arribar a j de nou. Els elements pertanyents a l'òrbita de j tindran tots la mateixa òrbita, ja que cada element forma part de la seva òrbita i la permutació per a tots ells és la mateixa. Es pot observar que calculant el mínim comú múltiple de les cardinalitats de les òrbites, sabrem el nombre de vegades r (m.c.m.) que cal repetir la permutació per tal d'obtenir la identitat: $\sigma^r(j) = j$ per a tot $j \in A$. Aquest nombre és l'ordre de la permutació (com que treballarem amb grups de permutacions, en denotarem l'ordre com $ord(\sigma)$).

Exemple 6. *En la permutació d'exemple del punt 4.1, hi ha dues òrbites:* $O_1 = \{1, 2, 4\}$ *(l'òrbita de l'*1*, el* 2 *i el* 4*) i* $O_2 = \{3, 5\}$ *(l'òrbita del* 3 *i el* 5*). Per això és d'ordre* 6*: composant-la amb si mateixa* 6 *cops s'obté la identitat. Això s'observa especialment bé en la representació mitjançant un graf, com apareix a sota:*

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 5 & 1 & 3 \end{pmatrix} \quad = \quad \begin{array}{ccccc} 1 & \longleftarrow & & & 4 \\ & \searrow & & \nearrow & \\ & & 2 & & \\ 3 & \curvearrowleft & & \curvearrowright & 5 \end{array}$$

4.3 Permutacions cícliques

S'anomenen *permutacions cícliques* o *cicles* les permutacions en què tots els elements no fixos tenen la mateixa òrbita. El seu ordre serà el nombre d'elements no fixos.

Les permutacions cícliques es poden denotar com es mostra a continuació: $\sigma = (j, \sigma(j), \sigma^2(j), ..., \sigma^{r-1}(j))$.

Exemple 7.

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 5 & 3 & 1 \end{pmatrix} = (1,2,4,3,5) = (4,3,5,1,2)$$

Si busquem les òrbites que provoca una certa permutació σ (cíclica o no) en cadascun dels elements sobre els quals s'aplica i ens quedem només amb les que són diferents (dues òrbites són iguals si tenen els mateixos elements), veurem que la intersecció entre elles és sempre el conjunt buit (ja que, com s'ha dit anteriorment, tots els elements pertanyents a una certa òrbita tenen aquella òrbita). Podem dir, doncs, que els elements d'una òrbita no afecten els de cap altra i, per això, és possible composar σ a partir de diversos cicles disjunts, un per a cada òrbita. De nou, això s'observa molt bé en la representació de permutacions mitjançant un graf, on s'observa també que, com que aquests cicles són disjunts, en aquest cas sí que es compleix la propietat commutativa.

Sovint, per tal d'abreujar, s'escriuen les permutacions directament com el producte dels diversos cicles disjunts que, composats, les formen.

S'anomenen *transposicions* els cicles d'ordre 2, i són importants perquè tot cicle d'ordre r es pot expressar com a composició de $r-1$ transposicions: $(a_1, a_2, ..., a_m) = (a_1, a_2)(a_2, a_3)...(a_{m-1}, a_m)$. Com que les permutacions es poden expressar com a composició de cicles disjunts i els cicles, de transposicions, tota permutació (cíclica o no) es pot expressar com a composició de transposicions.

Exemple 8.

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 5 & 1 & 3 \end{pmatrix} = (1,2,4)(3,5) = (3,5)(2,4,1) = (3,5)(2,4)(4,1) = (1,2)(2,4)(3,5)$$

4.4 Paritat d'una permutació

Hem dit que qualsevol cicle es pot expressar com a composició de transposicions, i que per tant qualsevol permutació es pot expressar com a composició de transposicions. La descomposició en transposicions, però, no és única. Ens podem plantejar, aleshores, si es pot expressar amb qualsevol nombre de transposicions. La resposta és que només es pot expressar o bé amb un nombre parell de transposicions o bé amb un nombre senar de transposicions.

Demostració. Donada una permutació σ, siguin i i j dos elements del conjunt A d'entrada tals que $i < j$, i siguin k i l dos elements de A tals que $k < l$.

Per tal de facilitar la comprensió, prendrem d'exemple la permutació

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 5 & 1 & 3 \end{pmatrix}$$

amb $k = 2$ i $l = 4$.

Definim P com el producte de totes les diferències possibles $\sigma(j) - \sigma(i)$. En composar la transposició (k, l) (en l'exemple $(2, 4)$) amb σ, les diferències $\sigma(j) - \sigma(i)$ varien, fent canviar P de diferents maneres[18]:

1. Les diferències en què no intervenen ni $\sigma(k)$ ni $\sigma(l)$ romanen iguals. En l'exemple, $(5-2), (3-2)$ i $(3-5)$.

2. En les diferències en què sí que intervenen:

   (a) En les diferències en què $j > l > k$ només hi ha un canvi de posició (entre $\sigma(j) - \sigma(k)$ i $\sigma(j) - \sigma(l)$), però el seu valor no canvia. En l'exemple, $(3-4)$ i $(3-1)$.

   (b) En les diferències en què $l > k > i$ només hi ha un canvi de posició (entre $\sigma(k) - \sigma(i)$ i $\sigma(l) - \sigma(i)$), però el seu valor no canvia. En l'exemple, $(4-2)$ i $(1-2)$.

   (c) Les diferències en què $l > j > k$ o $l > i > k$ canvien de signe, però el seu valor absolut no canvia. Com que n'hi ha un nombre parell (tantes passen a ser positives com passen a ser negatives), no fan variar el valor de P. En l'exemple, $(5-4)$ passa a ser $(5-1)$ i $(1-5)$ passa a ser $(4-5)$.

   (d) La diferència $\sigma(l) - \sigma(k)$ canvia només de signe (el seu valor absolut no canvia), fent canviar el signe de P. En l'exemple, $(1-4)$ passa a ser $(4-1)$.

D'aquí en podem extreure diverses conclusions:

1. En composar una transposició amb una permutació qualsevol fem canviar el signe de P. Per això, una permutació amb $P > 0$ s'ha de descomposar sempre en un nombre parell de transposicions i una permutació amb $P < 0$ s'ha de descomposar sempre en un nombre senar de transposicions.

[18] P no pot ser mai 0 perquè, per ser les permutacions bijectives, $\sigma(j) - \sigma(i) \neq 0$.

2. $|P|$ és igual en totes les permutacions de S_n, ja que, partint d'una permutació qualsevol, si anem composant-la amb transposicions (o permutacions composades per transposicions, tant és) només canviarà el signe de P i no el seu valor absolut.

□

D'aquesta manera, es defineix la paritat (també anomenada índex o signe) d'una permutació com el signe de P. Les permutacions amb $P < 0$ s'anomenen senars (perquè es descomposen en un nombre senar de transposicions) i les permutacions amb $P > 0$, parelles (pel mateix motiu)[19].

La permutació identitat sempre és parella, ja que el producte de totes les diferències $\sigma(j) - \sigma(i)$ amb $j > i$ és positiu.

Com que la composició de permutacions segueix la regla dels signes (la composició de dues permutacions de signe igual dona una permutació parella i la de dues de signe diferent, una de senar), es defineix la funció signe ε tal que

$$\varepsilon(\sigma) = +1 \text{ si } \sigma \text{ és parell,}$$
$$\varepsilon(\sigma) = -1 \text{ si } \sigma \text{ és senar.}$$

Així, es compleix que $\varepsilon(\tau\sigma) = \varepsilon(\tau) \cdot \varepsilon(\sigma)$[20].

Donada una certa permutació, SAGE ens en dirà el signe de la següent manera:

```
sage: s=Permutation([2, 4, 5, 1, 3])
sage: s.sign()
    -1
```

Havent definit la paritat d'una permutació, toca recuperar el Joc del 15 de Sam Loyd. Gràcies a la paritat de les permutacions podem saber si es pot arribar a la posició desitjada des d'una certa posició inicial. En la posició inicial de Loyd cal una transposició per tal d'aconseguir ordenar els nombres, cosa que implica un canvi de paritat. Ara bé, com que només estan permeses certes transposicions (intercanvis del forat i qualsevol dels nombres que hi toquen) i el forat ha d'acabar al mateix lloc on comença, s'ha de fer un nombre parell de moviments (transposicions) i, per això, és impossible canviar la

[19] Una manera senzilla de saber la paritat d'una permutació és comptar el nombre d'inversions que conté. Una inversió és una parella d'imatges $\sigma(i) > \sigma(j)$ tals que $j > i$. Són aquelles que tenen una diferència negativa i, per tant, les que fan canviar la paritat d'una permutació. Una permutació serà parella si i només si conté un nombre parell d'inversions. Si representem una permutació amb un diagrama, els encreuaments entre les fletxes que uneixen els elements amb les seves imatges representen les inversions (ja que hi ha un encreuament quan la imatge d'un element més petit que un altre sigui més gran que la d'aquest element).

[20] En altres paraules, és un homomorfisme. Es parla dels homomorfismes al punt 5.6.2.

paritat de la permutació inicial i la final. Per això el Joc del 15 de Loyd no es pot resoldre i, sortosament (per a Loyd), ningú no es va emportar el premi de 1000 dòlars.

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

Figura 4.1: La posició inicial del Joc del 15 de Sam Loyd

Considerant com a possibles posicions inicials aquelles en què el forat està en una altra casella, la paritat d'una permutació és rellevant perquè, a l'hora de buscar una solució, si la permutació inicial és parella (com la posició que es vol aconseguir), qualsevol solució haurà de tenir un nombre parell de moviments; i si la permutació inicial és senar[21], qualsevol solució haurà de tenir un nombre senar de moviments. Òbviament, això també és vàlid a l'hora de trobar la solució òptima.

4.4.1 Grup alternat A_n

Els elements del grup alternat de permutacions A_n són les permutacions parelles en n elements. Efectivament formen un grup, ja que:

1. Té element neutre, ja que $\varepsilon(Id) = +1$.
2. Cada element té el seu invers, ja que l'invers d'una permutació parella és una altra composició parella (perquè composades han de donar la identitat, que és parella).
3. La composició de permutacions és associativa.
4. Dues permutacions parelles composades donaran una altra permutació parella: com s'ha explicat abans, segueixen la regla dels signes.

L'ordre d'A_n serà $n!/2$, ja que, $\sigma(1)$ pot ser qualsevol dels n elements, $\sigma(2)$ qualsevol dels $n-1$ restants, etcètera. El penúltim element podrà prendre dos valors, en un cas crearà una inversió i, en l'altre, no, cosa que vol dir que aquestes dues permutacions

[21] Pot semblar que qualsevol permutació senar no es podrà resoldre. Això, però, no és així: s'explica en detall a l'annex, així com per què si una permutació no es pot resoldre, el motiu n'és la paritat.

tenen paritats diferents i, per tant una serà parella i l'altra senar, cosa que implica que la meitat de permutacions de S_n són parelles i, per tant, $|A_n| = |S_n|/2 = n!/2$.

5 Més grups

5.1 Subgrups

Podem definir un subconjunt S no buit del grup G com a subgrup de l'anterior si el subconjunt S també forma un grup amb l'operació pròpia del grup G, que anomenarem operació induïda. Es tracta de formar un grup aplicant l'operació de G però restringint-la als elements de S. Els subgrups, pel fet de ser grups, tenen també diverses propietats:

1. L'element neutre e' del subgrup $S \subseteq G$ ha de ser el mateix que l'element neutre e del grup G. Si $g \cdot e' = g$ i $g \cdot e = g$, sabrem que $g \cdot e' = g = g \cdot e$ del que obtenim, aplicant l'invers de g per la dreta, que $e' = e$.

2. Si un element $g \in S$ té un invers g^{-1}, aquest haurà de ser el mateix que l'invers que g té a G. D'aquesta manera $g \in S \Longrightarrow g^{-1} \in S$ degut a la coincidència entre l'operació induïda a S i l'operació pròpia del grup G.

Cal destacar que el comportament d'un subgrup qualsevol és com el d'un grup, de manera que també haurà de respectar l'existència i unicitat de l'element neutre, de l'invers, l'acompliment de la propietat associativa i l'operació induïda serà interna també a S, així que per a cada parell $g, g' \in S$, $g \cdot g' \in S$.

Exemple 9. *Tornarem al grup dels enters amb la suma, al qual anomenarem G. Dins d'aquest, seleccionarem el conjunt dels nombres enters parells (entenent que hi ha parells negatius: $-2, -4, ...$) que, amb la operació induïda de la suma, formaran el subgrup S. Recordem que l'element neutre del grup G és $e = 0$ que coincideix amb el del subgrup S. Es pot fer el càlcul de $2 + 0 = 2$ i es veu que efectivament $e = e'$. Com ja hem vist, l'invers que té 2 al grup dels enters amb la suma és -2, llavors aquest ha de ser el mateix invers que tingui dins del subgrup dels parells amb la suma de manera que $2 + (-2) = 0 = e'$.*

Un altre aspecte important és destacar que la intersecció de subgrups d'un cert grup sempre forma un altre subgrup però que la unió dels mateixos no té perquè fer-ho (l'únic cas en què pot passar és el cas en què un dels subgrups està inclòs dins de l'altre[22]).

[22] Ja que, per tal que la unió formi un grup, calen els productes de les parelles d'elements dels dos subgrups, que només hi són si un dels subgrups es troba inclòs en l'altre.

Demostració. Siguin S, T dos subgrups de G, la seva intersecccció serà $I = S \cap T = \{x \mid x \in S \wedge x \in T\}$. Cal demostrar que es compleixen les propietats dels subgrups (no cal demostrar l'associativitat perquè l'operació és induïda):

1. I conté l'element neutre: $e \in S, e \in T \Rightarrow e \in I$.
2. Cada element té un invers: $g \in I \Rightarrow g \in S \Rightarrow g^{-1} \in S$ i, alhora, $g \in I \Rightarrow g \in T \Rightarrow g^{-1} \in T$ i, per tant, $g^{-1} \in I$.
3. El producte de dos elements $g, g' \in I$ és un altre element de I. $g, g' \in I \Rightarrow g, g' \in S \Rightarrow g \cdot g' \in S$ i, alhora, $g, g' \in I \Rightarrow g, g' \in T \Rightarrow g \cdot g' \in T$ i, per tant, $g \cdot g' \in I$.

□

Exemple 10. *Sigui G el grup dels enters amb la suma. Dos subgrups en podrien ser el subgrup dels enters parells (inclosos els negatius), que anomenarem P; i el subgrup dels múltiples de 3 (incloent també els negatius), que escriurem com a T (per tant, $T, P \subset G$). La seva intersecció I estarà formada pels mútiples de 6, que formen un subgrup, però la unió $T \cup P$ no és un grup, ja que conté, per exemple, el 2 i el 3 però no el 5.*

5.2 Subgrups de G generats per un subconjunt S

Es diu que un subconjunt $S \subseteq G$ pot generar un subgrup, que contindrà els elements propis de S, els seus inversos i els productes d'uns i d'altres, i és el subgrup més petit que conté S (es trobarà inclòs dins de qualsevol altre subgrup que contingui S o, dit d'una altra manera, serà la intersecció de tots els subgrups que contenen S, i per això és únic). El subgrup generat per S es denota $\langle S \rangle$.

Cal dir, però, que sovint G no es defineix, ja que $\langle S \rangle$ serà un subgrup de qualsevol altre grup que contingui S (ja que és el grup més petit que conté S). Per això, és freqüent dir que S genera un grup (i no un subgrup).

Exemple 11. *Com ja és habitual, tornarem a un exemple donat anteriorment i el desenvoluparem una mica més. En aquest cas parlem del grup de permutacions del cub de Rubik G i del subgrup generat per un d'aquests moviments. Centrant-nos una altra vegada en el moviment $R \in G$, és a dir la rotació de $90°$ de la cara que es disposa a la dreta, veurem que les posicions (o els elements) del subgrup generat per R, és a dir $\langle R \rangle$, estan incloses dins del grup format per tots els moviments del cub de Rubik, així que $\langle R \rangle \in G$.*

El subgrup $\langle R \rangle$ compleix que:

1. *Conté els elements Id, R^1, R^{-1}, R^2... i els seus inversos, mentre que aquests també pertanyen al grup G.*

2. *El grup $\langle R \rangle$ està efectivament generat per R ja que els productes dels elements de R i els seus inversos hi pertanyen. Per exemple $R^2 \cdot R^2 = Id \in \langle R \rangle$ o $R \cdot R^{-2} = R^{-1} = R^3 \in \langle R \rangle$*

3. *És el subgrup més petit que conté R, ja que està inclòs dins de tots els subgrups que contenen R.*

5.2.1 Generadors del grup alternat

Si H és el subgrup generat per tots els cicles de 3 elements de S_n, llavors $H = A_n$.

Demostració. D'entrada, veiem que els cicles de 3 elements es poden descomposar en 2 transposicions i, per tant, tant els cicles de 3 elements com els seus productes seran transposicions parelles. És a dir, $H \subseteq A_n$.

D'altra banda, tots els elements de A_n es poden descomposar en un nombre parell de transposicions. Com que $(i,j) \circ (k,l) = (i,j,k) \circ (j,k,l)$ i $(i,j) \circ (j,k) = (i,j,k)$, si una permutació és parella llavors es pot expressar com el producte de diversos cicles de 3 elements. És a dir, $A_n \subseteq H$.

Combinant els dos resultats, veiem que $A_n = H$. □

5.2.2 Grup cíclic d'ordre r

És un tipus de grup G generat per un conjunt format per un sol element h, $G = \langle h \rangle$ per algun $h \in G$. Per això l'ordre r d'aquest element és el mateix que el del grup i, en aplicar un element amb ell mateix r vegades, s'obté l'element neutre després de passar per tots els altres del grup[23][24].

Un exemple d'aquest tipus de grup és un rellotge on podem definir els elements $\{0, 1, 2, ..., 11\}$, l'operació del grup és l'addició de mòdul 12[25] i el seu element neutre és el 0. En aquest cas l'invers de l'element 1, per exemple, és l'11 ja que ens permet arribar a la posició $12 = 0$ que és, com ja s'ha mencionat, l'element neutre.

[23] Abans del neutre s'obté h^{-1} i després h.

[24] Excepte en el cas que l'ordre de h sigui infinit, llavors mai no s'arriba a l'element neutre.

[25] El resultat de l'addició mòdul m és el residu de, donats dos nombres n, o, $n \cdot o/m$. En l'addició mòdul 12, per exemple, en sumar 5 a 9 s'obté 2.

De manera general un grup cíclic finit d'ordre r es pot identificar[26] amb un altre grup cíclic que tingui r elements, denotat C_r, i associat a l'addició de mòdul r.

Geomètricament parlant es pot identificar amb el grup format per totes les possibles rotacions del pla que deixen fix un r-àgon.

Figura 5.1: Els grups cíclics d'ordre 12 es poden identificar amb un rellotge, que és un exemple d'addició mòdul 12; o amb totes les rotacions del pla múltiples de $30°$, que deixen fix un dodecàgon regular.

Amb SAGE, un grup cíclic amb 12 elements es defineix així:

```
sage: C = CyclicPermutationGroup(12)
```

Un grup relacionat amb els grups cíclics és el grup diedral, que trobareu explicat a l'annex.

5.3 Taula de multiplicació d'un grup (o taula de Cayley)

Si un grup té ordre finit, llavors es pot fer la taula de multiplicació del grup, també dita taula de Cayley en honor a Arthur Cayley, matemàtic britànic que les va introduir. Aquesta taula té tantes files i tantes columnes com elements el grup, i a una certa casella d'una fila i i una columna j hi anirà el producte $g_i \cdot g_j$. Cal observar que es pot fer la taula de multiplicació de qualsevol grup finit, però que una taula no sempre es correspon a un grup, ja que en una taula de multiplicació no queda palesa la propietat associativa.

Exemple 12. *En el grup $G = \{1, i, -1, -i\}$ (les arrels quartes d'1) amb la multiplicació la taula de multiplicació de G seria:*

[26] Matemàticament, es diu que dos grups cíclics d'ordre r són isomorfs. Siguin G i H dos grups i $g, g'...$ i $h, h', ...$ els seus respectius elements: si G i H són isomorfs, aleshores existeix una aplicació bijectiva ϕ tal que per a tot g i g', si $\phi(g) = h$ i $\phi(g') = h'$, llavors $\phi(g \cdot g') = h \cdot h'$. S'explica amb detall a l'apartat 5.6.2.

$*$	1	i	-1	$-i$
1	1	i	-1	$-i$
i	i	-1	$-i$	1
-1	-1	$-i$	1	i
$-i$	$-i$	1	i	-1

Com que la taula és simètrica respecte de la diagonal, és un grup abelià.

Exemple 13. *Un cas en què la taula no és simètrica respecte de la diagonal i que, per tant, els seus elements no formen un grup abelià, pot ser el del grup de permutacions S_3. La seva taula de Cayley, feta amb SAGE, seria així:*

```
sage: S=SymmetricGroup(3)
sage: S.cayley_table(names='elements')
      *       ()   (2,3)   (1,2) (1,2,3) (1,3,2)   (1,3)
       +-----------------------------------------------
     ()|       ()   (2,3)   (1,2) (1,2,3) (1,3,2)   (1,3)
  (2,3)|   (2,3)       () (1,2,3)   (1,2)   (1,3) (1,3,2)
  (1,2)|   (1,2) (1,3,2)       ()   (1,3)   (2,3) (1,2,3)
(1,2,3)| (1,2,3)   (1,3)   (2,3) (1,3,2)       ()   (1,2)
(1,3,2)| (1,3,2)   (1,2)   (1,3)       () (1,2,3)   (2,3)
  (1,3)|   (1,3) (1,2,3) (1,3,2)   (2,3)   (1,2)       ()
```

5.4 Graf de Cayley d'un grup

Un graf és un parell de conjunts (V, E) en què V és un conjunt d'elements anomenats *vèrtexs* i E un subconjunt del conjunt de totes les parelles **no ordenades** $\{v_1, v_2\}$ d'elements de V (on $v_1 \neq v_2$), anomenades *arestes*. Un graf finit és un graf (V, E) en què V és finit, i un digraf (o graf dirigit) és un parell de conjunts (V, E) semblant a un graf, amb la diferència que E està format per un subconjunt de les parelles **ordenades** (v_1, v_2) d'elements de V, anomenades igualment *arestes*.

Si hi ha una aresta entre dos elements v_1, v_2 de V ($\{v_1, v_2\} \in E$), es diu que són adjacents, i siguin $w_1, w_2 \in V$, un camí de w_1 a w_2 és una seqüència finita d'arestes que comencen a w_1 i acaben a w_2. De la mateixa manera es diu que dos elements de V estan connectats si hi ha un camí que els uneix, i que un cert graf és connex si qualsevol parell d'elements de V estan connectats.

Donat un grup G generat per un conjunt X[27], el graf de Cayley de G generat per X (denotat $Cay(G, X)$) és el graf (V, E) que compleix que:

1. El conjunt d'elements de G és V (és a dir, $G = V$); i que
2. Siguin $g, g' \in G$, hi ha una aresta (no dirigida) que uneix g i g' si i només si $g = g' \cdot x$ o $g' = g \cdot x$ per a algun $x \in X$.

Semblantment es poden definir digrafs de Cayley, que es diferencien dels grafs de Cayley pel fet que les arestes són dirigides i, donats dos vèrtexs $g, g' \in G$, hi ha una aresta que els uneix si i només si $g = g' \cdot x$ per a algun $x \in X$.

Exemple 14. *Veurem diversos grafs i digrafs de Cayley del grup S_3, tots connexos perquè el subconjunt és, en tots els casos, generador de S_3*[28]*.*

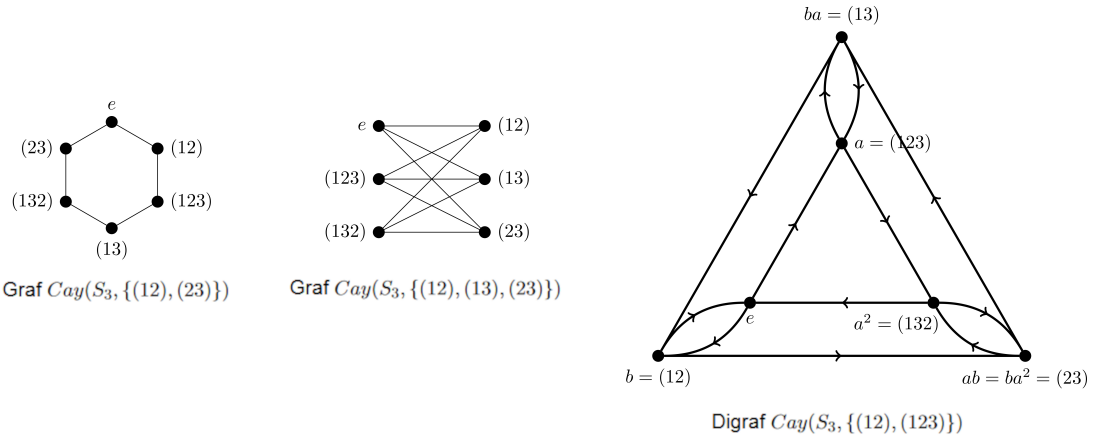


Figura 5.2: Alguns grafs de Cayley i grafs dirigits de Cayley de l'S_3, extrets de [2]

Veurem un exemple de graf de Cayley inconnex, fet amb SAGE.

```
sage: S=AlternatingGroup(4)
sage: C=S.cayley_graph(generators=[PermutationGroupElement([2,1,4,3]),\
    PermutationGroupElement([4,3,2,1])])
sage: CC=C to_undirected()
sage: show(CC)
```

[27] A vegades no s'imposa que X sigui un generador de G, però aleshores el graf que s'obté és inconnex. Degut a la condició que han de complir dos vèrtexs per tal de ser adjacents, els diferents conjunts de vèrtexs connectats entre si seran, llavors, les classes laterals mòdul el subgrup generat per X. Per saber què són les classes laterals, vegeu l'apartat de l'annex dedicat al teorema de Lagrange.

[28] Les permutacions s'escriuen en notació cíclica i sense comes per abreujar.

Figura 5.3: Graf de Cayley que resulta del programa anterior

5.5 Accions

Al llarg del treball hem anat dient que el grup simètric de permutacions S_n, o també, per exemple, el grup alternat de permutacions A_n, actuen sobre un cert conjunt de n elements. Però no hem definit què vol dir que un grup *actua* en un conjunt, i de fet les accions no es restringeixen a grups de permutacions sinó que són importants per a qualsevol mena de grup. Es diu que un grup G actua sobre un conjunt X per l'esquerra si:

1. Tot element $g \in G$ dona lloc a una aplicació

$$\phi_g : X \to X$$

2. L'element neutre de G defineix l'aplicació identitat en X.

3. $\phi_{h \cdot g}(x) = \phi_h(\phi_g(x)), \forall x \in X, \forall g, h \in G$.

Per tal d'abreujar la notació, es defineix la multiplicació per l'esquerra de G en X de la següent manera: $\phi_g(x) = g \cdot x$.

Cal, però, definir de quina manera els elements de G donen lloc a les funcions $\phi : X \to X$, és a dir, de quina manera G actua en X.

Exemple 15. *El grup S_5 actua en el conjunt $X = \{1, 2, 3, 4, 5\}$. Quan es tracta d'un grup de permutacions, sempre que no es digui el contrari es dona per fet que $\phi_\sigma(x) = \sigma(x)$, tot i que no té per què ser així*[29] *(en aquest exemple, però, sí que ho definim així). El mateix grup S_5, però, també pot actuar en el conjunt $Y = \{1, 2, 3, 4, 5, 6\}$, per exemple, i de la mateixa manera que en X. Simplement la imatge de 6 és sempre 6.*

Exemple 16. *Imaginem-nos un pla cartesià, que serà el nostre conjunt X, i un vector $\vec{v}$ d'aquest pla. Definim l'acció del grup dels enters amb la suma, $G = (\mathbb{Z}, +)$, en aquest pla*

[29] Per posar un exemple senzill (però poc útil a la pràctica) d'aquest cas podem definir que A_3 actua en $X = \{1, 2, 3\}$ de manera que una permutació σ dona lloc a una aplicació $\phi_\sigma(x) = \sigma^{-1}(x)$.

com l'aplicació del vector $g \cdot \vec{v}$, on g és un element de G. Efectivament, G actua en X (es verifiquen les tres propietats explicades anteriorment): tot element de G dona lloc a una aplicació en X; l'element neutre de G, el número 0, dona lloc a l'aplicació identitat; i, de la mateixa manera, l'aplicació que resulta de sumar dos enters coincideix amb la composició de les aplicacions a les quals donen lloc.

5.5.1 Òrbita d'un element sota l'acció d'un grup

De la mateixa manera que es pot definir l'òrbita d'un element sota una permutació, es pot definir l'òrbita $G \cdot x$ d'un element x d'un conjunt X sota l'acció d'un grup G. Concretament, serà el conjunt d'imatges que pot tenir x sota l'acció de G:

$$G \cdot x = \{g \cdot x \mid g \in G\}$$

Com en el cas de l'òrbita d'un element sota una permutació, si $x, y \in X$, llavors $x \in G \cdot x$, ja que $e \cdot x = x$, i si $y \in G \cdot x$, llavors $G \cdot x = G \cdot y$, degut a que G és un grup. Això implica que si dos elements tenen òrbites diferents O_1, O_2 la seva intersecció és el conjunt buit i que la unió de totes les òrbites és X (en altres paraules, les òrbites formen una partició de X).

Exemple 17. *Tornem a l'acció del grup dels enters amb la suma G en el pla cartesià on $\phi_g = g \cdot \vec{v}$. Si $\vec{v} = (1, 1)$ i $P(-2, -2)$ és un punt del pla, llavors l'òrbita de P sota G estarà formada per tots els punts $Q(z, z)$, on $z \in \mathbb{Z}$:*

$$G \cdot P = \{Q(z, z) \mid z \in \mathbb{Z}\}$$

Exemple 18. *El grup de moviments del cub de Rubik G actua en el conjunt dels petits subcubs del cub. L'òrbita d'una aresta serà el conjunt de totes les arestes (E); l'òrbita d'un vèrtex, el conjunt dels vèrtexs (V), i l'òrbita d'un centre, el conjunt dels centres (C)*[30].

5.6 Comparació de grups

L'ànalisi i estudi dels grups no només consisteix en veure o entendre les seves propietats i utilitats, sinó que també es poden fer un munt de coses més. En aquest cas indagarem en les comparacions o similituds, en el fet de trobar certs elements o patrons que es

[30] Vegeu 1.1.3 per recordar què són arestes, els vèrtexs i els centres quan es parla del cub de Rubik.

repeteixen en diversos grups i que poden ser de molta utilitat per a entendre els seus caràcters. Ara que sabem classificar-los més enllà de per la seva operació pròpia i conjunt d'elements, com per exemple segons si són simètrics o abelians, també podem comparar aquests altres aspectes.

5.6.1 Funcions entre dos grups

Una manera de comparar grups és analitzant les funcions entre ells, de manera que es pugui obtenir informació sobre els mateixos. Per a realitzar questa tasca també pot ser molt útil saber identificar quin tipus de funció els relaciona.

5.6.2 Homomorfisme de grups

Si els grups G_1 i G_2 amb les seves respectives operacions $*_1$ i $*_2$ presenten un homomorfisme entre ells donat per la funció $f : G_1 \to G_2$, llavors hauran de respectar que $\forall a, b \in G_1$ es compleix que: $f(a *_1 b) = f(a) *_2 f(b)$. En aquest punt ens centrarem en aquells homomorfismes que presentin una funció bijectiva entre els grups involucrats, és a dir, que hi hagi tants elements d'entrada com de sortida, de manera que tinguin el mateix ordre (una sola imatge per a cada element). Tornant a l'explicat sobre les funcions bijectives, recordem que són exhaustives i injectives alhora de manera que tenen invers (ja que sabent la imatge i la funció es pot saber l'antiimatge i viceversa) i relacionen els elements dels grups involucrats un a un. Això és molt important ja que ens ajuda a arribar a la conclusió que els grups relacionats per una funció bijectiva tenen la mateixa estructura però poden presentar elements i operacions diferents, en altres paraules són isomorfs.

Es defineix un automorfisme com un isomorfisme d'un grup en si mateix.

Seguint, doncs, les condicions mencionades posem alguns exemples.

Exemple 19. *L'homomorfisme identitat és aquell que donada una funció $f : G \to G$ confirma que $\forall x \in G$ romandrà igual (idèntic) una vegada se li apliqui la funció mencionada. Així doncs $x \longmapsto f(x) = x$.*

Per últim, comprovem que es compleix la propietat dels homomorfismes, per tant, $\forall a, b \in G$, $f(a \cdot b) = f(a) \cdot f(b)$ ja que, al tractar-se del mateix grup d'entrada i de sortida, manté la mateixa operació interna. Observem que també és un isomorfisme.

Exemple 20. *En aquest cas presentarem els grups formats pels nombres reals amb la suma $(\mathbb{R}, +)$ i pels nombres reals sense incloure el 0 amb la multiplicació $(\mathbb{R}^*, \cdot)$. També*

haurem de considerar, es clar, la funció $f : \mathbb{R} \to \mathbb{R}^*$ *donada per* $f(x) = 3^x$ *.*

Ara només queda fer la demostració de la propietat dels homomorfismes seguint que $\forall a, b \in \mathbb{R}$ *es compleix que* $f(a + b) = 3^{a+b} = 3^a \cdot 3^b = f(a) \cdot f(b)$

Exemple 21. *Com ja hem dit anteriorment, recordem que la funció* ε*, que retorna el signe d'una permutació, és un homomorfisme amb un grup de permutacions com a conjunt d'entrada i el grup de les arrels quadrades d'*1 *(amb el producte) com a conjunt de sortida. Efectivament,* $\varepsilon(\sigma)\varepsilon(\tau) = \varepsilon(\sigma\tau)$

5.7 Productes de conjunts i de grups

5.7.1 Producte cartesià de dos conjunts

Siguin S i T dos conjunts. El producte cartesià d'S i T, denotat $S \times T$, és el conjunt format per totes les parelles ordenades d'elements, un de S i un de T:

$$S \times T = \{(s, t) \mid s \in S, t \in T\}$$

5.7.2 Producte directe de dos grups

Siguin G i H dos grups. El producte directe de G i H, denotat $G \times H$, és el grup format pel producte cartesià de G i H i una operació $*$. Si $g_i \in G, h_i \in H$, llavors $(g_1, h_1) * (g_2, h_2) = (g_1 *_G g_2, h_1 *_H h_2)$, on $*_G$ és l'operació de G i $*_H$ és l'operació de H.

A vegades s'escriu el producte directe d'un grup G amb ell mateix n vegades com G^n.

Exemple 22. *El producte directe* P *de* C_2 *i les arrels quartes d'*1 *(amb el producte), és* $P = \{(0, 1), (0, i), (0, -1), (0, -i), (1, 1), (1, i), (1, -1), (1, -i)\}$*. El producte de* $(0, 1)$ *i* $(1, -i)$*, per exemple, és* $(0 + 1, 1 \cdot (-i)) = (1, -i)$*.*

5.7.3 Producte semidirecte de dos grups

Per poder fer el producte semidirecte de dos grups G, H cal que G actuï bijectivament en H, i s'ha de complir que les bijeccions donades per l'acció siguin automorfismes. Aleshores, es complirà que $g(h_1 \cdot h_2) = g(h_1) \cdot g(h_2)$.

Així, el producte semidirecte de G i H, denotat $H \rtimes G$, és el grup format pel producte cartesià de G i H i l'operació $*$, de manera que, si $g_i \in G, h_i \in H$, llavors $(g_1, h_1) * (g_2, h_2) = (g_1 *_G h_1(g_2), h_1 *_H h_2)$, on $*_G$ és l'operació de G i $*_H$ és la de H.

Exemple 23. *El producte semidirecte* P *de* C_2^2 *i* S_2 *és*

$$P = \{((0,0), Id), ((0,1), Id), ((1,0), Id), ((1,1), Id),$$
$$((0,0), (1,2), ((0,1), (1,2)), ((1,0), (1,2)), ((1,1), (1,2)).$$

El producte de $((0,1),(1,2))$ *amb* $((0,1), Id)$*, per exemple, és* $((0,1) + (1,2)((0,1)), Id \circ (1,2)) = ((0,1) + (1,0), (1,2)) = ((1,1), (1,2))$.

6 Una primera aplicació: el cub de Rubik

En aquesta part del treball ens endinsarem en l'estudi del cub de Rubik a partir de la teoria de grups. No es tracta d'una part merament teòrica perquè el seu objectiu no és introduir nous conceptes de la teoria de grups sinó aplicar els que s'han explicat anteriorment; però tampoc no és una part purament pràctica perquè allò que s'explica aquí no són conclusions dels autors del treball sinó informació extreta d'altres fonts.

Per començar, explicarem breument la notació que s'utilitza per descriure els moviments del cub de Rubik. A continuació, ens fixarem en les característiques del grup de permutacions del cub de Rubik i, posteriorment, utilitzarem aquestes característiques per descobrir altres aspectes del cub de Rubik.

6.1 Nomenclatura del cub de Rubik

Els jocs acostumen a tenir certa nomenclatura per a anomenar els seus elements o moviments. Això passa també en el cas del Cub de Rubik, que té una notació que permet descriure els seus moviments segons la orientació del cub que s'estigui fent servir per a resoldre'l, és a dir, el nom dels moviments sempre depèn de la perspectiva. Seguint amb aquest punt, cal destacar que els moviments es realitzen en sentit horari (antihorari en el cas dels inversos).

La notació dels moviments del cub de Rubik es pot dividir segons els moviments elementals i els no elementals, tot i que els segons són conseqüència de la composició dels primers. Els moviments elementals, doncs, són aquells moviments individuals de rotació d'una única cara i es descriuen amb una sola lletra. Els no elementals, en canvi, en requereixen totes les lletres que descriuen els moviments elementals que els formen. Ja s'ha explicat amb anterioritat el moviment R, però ara caldrà mostrar tots els altres per

tal de poder entendre els punts que van a continuació.

Moviments elementals de les capes bàsiques donada una posició qualsevol:

- F representa el moviment de la cara de davant (és a dir la frontal).
- B es refereix al moviment de la cara que queda darrere.
- U és el moviment de la cara que es localitza a dalt.
- D correspon al moviment de la cara de sota.
- L moviment de la capa de l'esquerra.
- R representa el moviment de la cara que queda a la dreta.

Tots els noms de la notació de les capes bàsiques tenen l'origen a l'anglès essent així *front, back, up, down, left i right* respecte del llistat anterior.

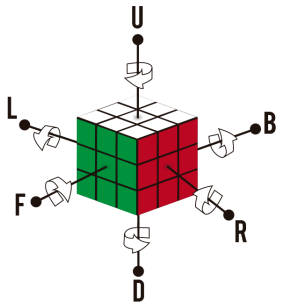


Figura 6.1: Notació del cub de Rubik

Moviments elementals de les capes centrals donada una posició qualsevol:

- S: moviment de la capa central paral·lela a F i B.

Figura 6.2: Moviment S

- E: correspon al moviment de la capa central paral·lela a U i D.

E

Figura 6.3: Moviment E

- M: es refereix al moviment de la capa central paral·lela a L i R.

M

Figura 6.4: Moviment M

Per aplicar diversos moviments en cert ordre (composar-los) s'escriu la seqüència dels mateixos, essent el de l'esquerra el primer a realitzar i el de la dreta, l'últim. D'aquesta manera en la composició REL s'aplica primer R, després E i finalment L.

En aquest treball es fan servir els exponents per indicar els inversos i les repeticions d'un mateix moviment. Sovint, però, es pot trobar que per a esmentar els inversos es fa servir una notació que involucra els apòstrofs, d'aquesta manera L' seria l'invers de L.

6.2 El grup de moviments del cub de Rubik

Per estudiar el grup de moviments possibles del cub de Rubik, al qual anomenarem G, utilitzarem homomorfismes amb altres grups dels quals coneixem més propietats. Definim també H com el grup els elements del qual són totes les ordenacions possibles dels cubets del cub de Rubik, si està permès desmuntar el cub i tornar-lo a muntar. G serà, doncs, un subgrup de H.

Primer, ens fixarem en els 8 vèrtexs. Per cada parella de vèrtexs, sempre hi ha un moviment (no elemental) que els intercanvia, sense moure cap altre vèrtex (però sí una parella d'arestes)[31]. Aquest moviment, si tenim en compte només els vèrtexs, és una transposició. Sabent això, i com que tota permutació és producte de transposicions, les ordenacions possibles dels vèrtexs es corresponen als elements de S_8. En altres paraules, existeix un homomorfisme $\rho : G \to S_8$ i és exhaustiu.

[31] En aquest punt es farà sovint referència a alguns moviments. Trobareu una llista de tots aquests moviments emprats a l'annex.

Fixem-nos ara en les 12 arestes. Per cada parella d'arestes, sempre hi ha un moviment (no elemental) que les intercanvia, sense moure cap altra aresta (però sí una parella de vèrtexs). Aquest moviment, si tenim en compte només les arestes, és una transposició. Sabent això, i com que tota permutació és producte de transposicions; les ordenacions possibles de les arestes es corresponen als elements de S_{12}. En altres paraules, existeix un homomorfisme $\sigma : G \to S_{12}$ i és exhaustiu.

Els dos resultats anteriors permeten assegurar que sempre es pot representar la posició dels subcubs del cub de Rubik amb un element de S_8 i un de S_{12}. En altres paraules, el grup de moviments dels subcubs G (sense tenir en compte les rotacions dels subcubs) és un subgrup del producte directe $I = S_8 \times S_{12}$ (no podem assegurar que sigui el grup $S_8 \times S_{12}$ perquè no sabem si totes les combinacions d'elements de S_8 i de S_{12} estan permeses).

En el paràgraf anterior recordàvem que no estem tenint en compte l'orientació dels cubets. Per fer-ho, cal marcar amb una creueta (imaginàriament) una cara de cada subcub. La de sota és una manera de fer-ho, que prendrem de referència. Aprofitem també per numerar els vèrtexs i les arestes, ja que caldrà per a posteriors exemples.

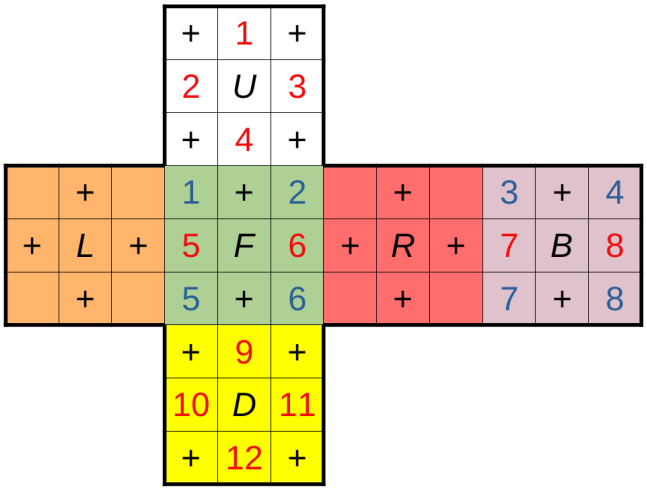


Figura 6.5: Una possible manera de marcar una cara de cada subcub i numerar-los

Havent fet això, podem expressar amb precisió les rotacions dels subcubs. En el cas dels vèrtexs, utilitzarem un vector $v(g) = \vec{v}$ (g és un element de G) amb 8 components, un per a cada vèrtex. Les rotacions d'un vèrtex poden ser de $0°$, $120°$ o $240°$ i es poden composar, de manera que formen un grup cíclic d'ordre 3 isomorf al grup format per 0, 1 i 2 amb l'addició mòdul 3. Un cert component del vector que correspongui a un determinat

vèrtex a serà 0 si la seva creueta és on hi havia la creueta del vèrtex que en la posició identitat ocupava el lloc on ara hi ha el vèrtex a; serà 1 si, respecte aquesta última creueta, està rotada $120°$ en sentit horari; i serà 2 si, respecte a aquesta mateixa creueta, està rotada $240°$ en sentit horari. Es pot entendre que $\vec{v}$ és un element del producte directe de C_3 amb ell mateix 8 vegades ($\vec{v} \in C_3^8$).

Per a les arestes emprarem un vector $w(g) = \vec{w}$ amb 12 components, un per a cada aresta. Les rotacions d'una aresta formen un grup isomorf al cíclic d'ordre 2 amb l'addició mòdul 2. Un cert component del vector que correspongui a una certa aresta serà 0 si l'aresta no està rotada i 1 si sí que ho està, prenent la mateixa referència que en el cas anterior (la creueta de l'aresta que en la posició identitat ocupava l'espai en qüestió). Es pot entendre que $\vec{w}$ és un element del producte directe de C_2 amb ell mateix 12 vegades ($\vec{w} \in C_2^{12}$).

Ara podem expressar amb precisió qualsevol posició o moviment del cub amb la forma $g = (r, \vec{v}, s, \vec{w})$, on $\vec{v}$ i $\vec{w}$ són els vectors descrits anteriorment; r, un element de S_8, concretament, $r = \rho(g)$; i s, un element de S_{12}, concretament $\sigma(g)$[32].

Exemple 24. *Posarem d'exemple els moviments elementals partint de les creuetes i la numeració de la figura 6.5, ja que ens serà útil posteriorment.*

- $R = ((2,3,7,6),(0,1,2,0,0,2,1,0),(6,3,7,11),(0,0,0,0,0,0,0,0,0,0,0,0))$
- $L = ((1,5,8,4),(2,0,0,1,1,0,0,2),(5,10,8,2),(0,0,0,0,0,0,0,0,0,0,0,0))$
- $F = ((1,2,6,5),(1,2,0,0,2,1,0,0),(5,4,6,9),(0,0,0,1,1,1,0,0,1,0,0,0))$
- $B = ((3,4,8,7),(0,0,1,2,0,0,1,2),(7,1,8,12),(1,0,0,0,0,0,1,1,0,0,0,1))$
- $U = ((1,4,3,2),(0,0,0,0,0,0,0,0),(1,3,4,2),(0,0,0,0,0,0,0,0,0,0,0,0))$
- $D = ((5,6,7,8),(0,0,0,0,0,0,0,0),(9,11,12,10),(0,0,0,0,0,0,0,0,0,0,0,0))$

Ara, però, sorgeix una pregunta important: com composarem dues d'aquestes posicions o moviments $g = (r, \vec{v}, s, \vec{w})$ i $g' = (r', \vec{v'}, s', \vec{w'})$ ($g, g' \in G$)?

El problema sorgeix amb les rotacions: després del primer moviment els subcubs han canviat de lloc, i per això cal reordenar els elements dels vectors del segon moviment per tal de, quan sumem component a component (l'operació de C_n^k és l'addició mòdul n), estiguem sumant de veritat les rotacions d'un mateix vèrtex. En concret, en el cas

[32] Cal recordar els homomorfismes ρ i σ establerts anteriorment.

dels vèrtexs (es fa anàlogament amb les arestes; de moment, oblidem-nos-en), definim una acció de S_8 en C_3^8 que el que fa és reordenar els components de $\vec{v'}$. Llavors, el vector de la composició de dos moviments $g, g' \in G$ serà $v(g' \circ g) = v(g) + \rho(g)^{-1}(v(g'))$ i no simplement $v(g' \circ g) = v(g) + v(g')$, ja que $\rho(g)^{-1}$ reordena $v(g')$ com s'explica a continuació. Si $g(i) = j$ (i i j són vèrtexs), l'acció de l'invers de $\rho(g)$ sobre el vector $\vec{v'}$ envia el component j del vector $\vec{v'}$ al component i del mateix vector, de manera que al final totes les rotacions del vèrtex i es troben en el component i del vector i es poden sumar correctament.

Exemple 25. *Seguirem sense tenir en compte, de moment, les arestes i agafarem* $g = U = (r, \vec{v}) = ((4,3,2,1),(0,0,0,0,0,0,0,0))$ *i* $g' = R = (r', \vec{v'}) = ((2,3,7,6),(0,1,2,0,0,2,1,0))$. *Si composéssim els moviments sense tenir en compte l'acció de* S_8 *en* C_3^8 *obtindríem, per exemple, que el subcub* 2 *està rotat quan, com que no s'ha vist "afectat" pel moviment* R*, no ho està. L'acció "fa" que la rotació que si apliquem el moviment* R *(sol) afecta al subcub* 2 *"es computi" per al subcub* 3*, que és el que efectivament "rep" la rotació, i ho "fa" com s'ha explicat anteriorment.*

Com s'ha dit, es fa anàlogament amb les arestes, de manera que $w(g'g) = w(g) + \sigma(g)^{-1}(w(g'))$. Ara ja podem composar $g = (r, \vec{v}, s, \vec{w})$ i $g' = (r', \vec{v'}, s', \vec{w'})$:

$$(r', \vec{v'}, s', \vec{w'}) \circ (r, \vec{v}, s, \vec{w}) = (r' \circ r, \vec{v} + \rho(g)^{-1}(\vec{v'}), s' \circ s, \vec{w} + \sigma(g)^{-1}(\vec{w'}))$$

Havent definit una operació, mitjançant una acció, en el conjunt d'elements $(r, \vec{v}, s, \vec{w})$, podem definir H mitjançant dos productes semidirectes i un producte directe:

$$H = (C_3^8 \rtimes S_8) \times (C_2^{12} \rtimes S_{12})$$

6.2.1 Teorema fonamental del cub de Rubik

El teorema fonamental del cub de Rubik serveix per discernir quins elements de H pertanyen també a G, i diu que una posició del cub de Rubik es pot resoldre si i només si es compleixen les següents 3 propietats:

1. El signe de r i de s és igual: $\varepsilon(r) = \varepsilon(s)$,
2. Els components de $\vec{v}$ sumen 0 mòdul 3, i
3. Els components de $\vec{w}$ sumen 0 mòdul 2.

Demostració. Demostrarem primer que si es pot resoldre es compleixen aquestes propietats o, en altres paraules, que si s'hi pot arribar mitjançant una seqüència de moviments, aleshores es compleixen aquestes propietats (el *només si*). Qualsevol moviment del cub és la composició d'un cert nombre de moviments elementals i, per tant, si els moviments elementals compleixen les tres propietats i la composició de dos moviments qualssevol que les compleixin (per exemple, elementals) també, aleshores haurem demostrat aquesta part del teorema, ja que qualsevol posició resoluble és la composició de diversos moviments elementals.

1. Efectivament, en els moviments elementals $\varepsilon(r) = \varepsilon(s) = -1$. Com que el signe és un homomorfisme, $\varepsilon(r' \circ r) = \varepsilon(r) \cdot \varepsilon(r') = \varepsilon(s) \cdot \varepsilon(s') = \varepsilon(s' \circ s)$ i, per tant, si un moviment és legal (no necessàriament elemental), llavors $\varepsilon(r) = \varepsilon(s)$.

2. Efectivament, en els moviments elementals els components de $\vec{v}$ sumen 0 mòdul 3. Donat que l'addició mòdul n és distributiva, si els components de $\vec{v}$ i també els de $\vec{v'}$ sumen 0 mòdul 3, llavors els components de $\vec{v} + \vec{v'}$ també sumaran 0 mòdul 3.

3. Efectivament, en els moviments elementals els components de $\vec{w}$ sumen 0 mòdul 2. Donat que l'addició mòdul n és distributiva, si els components de $\vec{w}$ i també els de $\vec{w'}$ sumen 0 mòdul 2, llavors els components de $\vec{w} + \vec{w'}$ també sumaran 0 mòdul 2.

Queda, doncs, demostrar que si es compleixen les tres propietats, aleshores es tracta d'un moviment legal. És a dir, volem demostrar que, partint d'un element $(r, \vec{v}, s, \vec{w})$ que compleixi les 3 propietats, sempre podrem arribar a la identitat, o sigui, a $(Id, \vec{0}, Id, \vec{0})$. Ho farem per passos (trobareu els moviments emprats a l'annex).

1. Com que existeixen moviments que són transposicions de vèrtexs, existeix un moviment M_1 que permet passar de la posició $(r, \vec{v}, s, \vec{w})$ a $(Id, \vec{v}', s', \vec{w}')$ (degut al que s'ha vist al 4.3).

2. Per cada parella de vèrtexs, existeix un moviment que en canvia l'orientació i no la posició i, degut a la part del teorema provada anteriorment, els components dels seus vectors $\vec{v}$ sumen 0 mòdul 3. Per tant, és possible passar d'una posició $(Id, \vec{v}', s', \vec{w}')$ a la posició $(Id, \vec{0}, s'', \vec{w}'')$ mitjançant un moviment M_2, que serà la composició de diverses rotacions de vèrtexs.

3. Ara, com que $r = Id$, per això $\varepsilon(r) = 1$, i per tant s'' és una permutació parella. En altres paraules, pertany a A_{12}. A més, per cada tríada d'arestes existeix un moviment

que la permuta, i per tant, com que A_n és el grup generat per tots els cicles de S_n de 3 elements (vegeu 5.2.1), és possible passar de la posició $(Id, \vec{0}, s'', \vec{w}''')$ a la posició $(Id, \vec{0}, Id, \vec{w}'')$ mitjançant un moviment M_3, que serà la composició de diversos cicles de 3 arestes.

4. Per cada dues arestes qualssevol existeix un moviment que en canvia l'orientació sense canviar res més en tot el cub, i degut a l'enunciat teorema, el nombre d'arestes invertides ha de ser parell. Per tant, existeix un moviment M_4 que permet passar de $(Id, \vec{0}, Id, \vec{w}''')$ a $(Id, \vec{0}, Id, \vec{0})$, que serà la composició d'alguns dels moviments mencionats.

Amb això, queda demostrat el teorema fonamental del cub de Rubik. □

Part III

Estudi i programa d'un trencaclosques propi

7 El nostre trencaclosques

La part pràctica del nostre treball consta bàsicament de dos blocs: un primer bloc en el qual expliquem el procés de creació i d'estudi del nostre trencaclosques de permutacions, i un segon bloc dedicat a la programació. Al final, aquests dos blocs conflueixen i s'explica el procés que hem seguit per tal de crear el programa capaç de resoldre el nostre trencaclosques.

Un aspecte que teníem clar sobre com havia de ser en el nostre trencaclosques de permutacions és la paritat. Volíem que la paritat de tot moviment possible fos senar; d'aquesta manera, donada una posició qualsevol, es podria saber ja d'entrada si el nombre de moviments necessaris per arribar a la configuració final era parell o senar, aspecte que pensàvem que ens podia ser útil a l'hora de trobar solucions òptimes. A partir d'aquí, es tractava de crear un trencaclosques que, a més, tingués les característiques dels trencaclosques de permutacions (vegeu el punt 1).

La idea que vam tenir, a partir de la qual es desenvolupen totes les variants del nostre trencaclosques, és la que s'explica a continuació. El joc es desenvolupa en una graella amb n caselles (que pot prendre mides i formes diverses segons la variant) en què en cada casella hi ha un número natural comprès en l'interval $[1, n]$. Partint d'una posició desordenada, mitjançant moviments successius, s'ha d'arribar a la posició en què tots els nombres estan ordenats. Els moviments elementals permesos són aquells que consisteixen en, donada una certa fila de la graella, moure cada nombre en una casella adjacent de la mateixa fila, tots en el mateix sentit. Sempre, però, hi haurà un nombre d'un extrem de la fila que hauria de quedar "fora" de la graella, en una casella inexistent. Aquest nombre passa a ocupar la casella que ha quedat buida, a l'extrem oposat de la fila[33]. Aquests moviments elementals es podran visualitzar de manera més clara quan es presentin algunes variants del trencaclosques.

[33] Cal dir que, justament per aquest motiu, no es tracta de trencaclosques que es puguin construir físicament, sinó, en tot cas, de manera digital. És una opció que, com es veurà posteriorment, també ha estat valorada i, fins i tot, ha ofert els seus fruits.

S'ha dit que el nostre trencaclosques té diverses variants. En vam crear més d'una perquè trobàvem interessant estudiar les relacions entre trencaclosques i els possibles homomorfismes que presentessin. És quelcom que finalment no ha sigut possible, però que es pot plantejar en el futur. D'altra banda, com s'ha explicat a la introducció, no volíem cenyir-nos a res concretíssim, sinó que volíem que el nostre programa pogués resoldre un trencaclosques de mides (i, per què no, de formes) diverses.

Per poder-ho fer hem hagut d'idear maneres de referir-nos als diversos moviments elementals, que també segueixen la mateixa lògica en tots els trencaclosques però amb les adaptacions que requeria cada cas.

A continuació, presentem les variants que ens han semblat més interessants del nostre trencaclosques amb una explicació dels grups que generen els diversos moviments elementals en cada cas, que hem estudiat de la mateixa manera que hem vist que s'estudia el grup del cub de Rubik. En altres paraules, hem pres com a referència l'estudi del cub de Rubik.

7.1 Versió rectangular (o quadrada)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

Figura 7.1: Trencaclosques rectangular de $4 \cdot 6$ resolt

La graella anterior és un exemple de trencaclosques rectangular en la posició a la qual cal arribar, en aquest cas de $4 \cdot 6$. Els moviments de columnes (de dalt cap a baix, excepte el nombre de baix, que passa a dalt de tot) es denoten amb els números de l'1 al 6 (1 és el moviment de la columna de l'esquerra) i els de files (d'esquerra a dreta, excepte el nombre de la dreta, que passa a l'esquerra), amb lletres de la A a la D (A és el moviment de la fila de dalt). Segons la mida de la graella s'afegeixen tantes lletres o números com convinguin per tal de denotar tots els moviments.

Per denotar l'invers de cada moviment escriurem un - al seu davant, ja que a l'hora de programar és més senzill que utilitzar exponents. En composar dos o més

moviments, s'escriuen tots seguits i es llegeix d'esquerra a dreta. També es poden utilitzar parèntesis. Els exponents, que indiquen quantes vegades es repeteix un moviment (o seqüència de moviments), s'empren només quan és estrictament necessari, i escrivint tots els parèntesis necessaris per evitar dubtes en la prioritat d'operacions. Les explicacions d'aquest paràgraf també són vàlides per a les altres variants del joc.

En el cas de la graella rectangular, és necessari que cada costat tingui un nombre parell de caselles, ja que si no, no tots els moviments tindrien paritat senar.

Exemple 26. *En la graella d'exemple anterior,* 3 *denota el moviment de dalt cap a baix de la tercera columna, és a dir, en notació cíclica,* $(3, 9, 15, 21)$*; i* $-B$*, el moviment de dreta a esquerra de la segona fila, és a dir,* $(12, 11, 10, 9, 8, 7)$*. Així doncs,* 3-B-B *equival a* $(12, 11, 10, 9, 8, 7)^2 \circ (3, 9, 15, 21)$.

Quan ja teníem bastant avançat el treball vam saber, gràcies a Herbert Kociemba, que aquesta versió del nostre trencaclosques ja existia, amb el nom de *Loopover* (en canvi, les altres versions no existien). En trobareu la versió original a https://loopover.xyz/. Encara que no era la nostra idea inicial, el fet que ja existís ens ha facilitat la feina en alguns casos (ho especificarem quan així sigui). En aquest cas la recerca no està tan avançada com en el cas del Cub de Rubik, i encara que se saben certes coses, no s'ha investigat a fons.

7.1.1 Estudi del grup que generen els moviments de la versió rectangular

A continuació procedirem a analitzar el grup generen els moviments de la versió recatngular del Loopover. Si n indica el nombre de caselles i G, el grup generat, llavors, com es pot observar, existeix un homomorfisme $h : G \to S_n$ i és injectiu. Si aconseguim veure que, a més, és exhaustiu, aleshores sabrem que G i S_n són isomorfs (ja que h serà bijectiu). Com ja sabem, això ho podem aconseguir demostrant que totes les transposicions pertanyents a S_n pertanyen també a G.

De fet, però, només trobant una transposició en tenim prou, perquè a partir d'una certa transposició podem trobar totes les altres. Si trobem una transposició (a, b), la transposició (c, d) serà la composició $\mu^{-1} \circ (a, b) \circ \mu$ (és a dir, el conjugat de (a, b) per μ), on μ és una seqüència de moviments tals que $\mu(c) = a$ i $\mu(d) = b$. Es tracta, doncs, de demostrar que existeixen tant (a, b) com μ.

Són demostracions constructives. En el cas de la primera, només cal trobar una

permutació qualsevol (si algun dels costats del rectangle té només 2 caselles, (a, b) és un moviment elemental paral·lel a aquest costat). En cas contrari, sigui c el nombre de columnes i f el nombre de files (cal recordar que c i f són parells), el moviment $(2AA\text{-}2\text{-}A)^{c-1}$ és una transposició de l'1 i el 2, ja que $(2AA\text{-}2\text{-}A)$ és el moviment $(1, 2)(3, 4, ..., c, c\cdot(f-1)+2)$.

Només queda, doncs, demostrar que μ existeix. Donat que l'acció de G en $\{1, 2, ..., n\}$ genera una sola òrbita, existeix un moviment μ_1 tal que $\mu_1(c) = a$. El moviment μ_2 tal que $\mu_2(\mu_1(d)) = b$, i $\mu_2(a) = a$ es construirà movent d a una fila diferent a la de a (si hi és) amb un sol moviment elemental (vertical), movent d a continuació fins a la columna de b mitjançant només moviments horitzontals, i finalment movent d fins a b mitjançant només moviments verticals. $\mu_2 \circ \mu_1 = \mu$ i, per tant, l'homomorfisme h és exhaustiu i, en definitiva, és bijectiu i G i S_n són isomorfs[34].

Per acabar de verificar aquests resultats, podem utilitzar el SAGE per mirar si els dos grups són isomorfs:

```
sage: G=PermutationGroup(['(1,2,3,4)','(5,6,7,8)','(9,10,11,12)',\
    '(13,14,15,16)','(1,5,9,13)','(2,6,10,14)','(3,7,11,15)','(4,8,12,16)'])
sage: G.is_isomorphic(SymmetricGroup(16))
    True
```

El graf de Cayley de la portada és el del grup de moviments generats en el trencaclosques $2 \cdot 2$. Com s'hi pot obervar, el grup generat és l'S_4.

7.2 Versió triangular

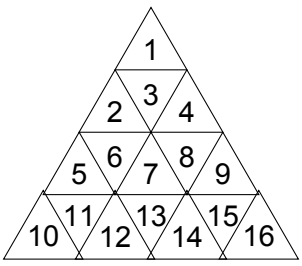


Figura 7.2: Trencaclosques de graella triangular de 4 files

La graella anterior és un exemple de trencaclosques triangular, en aquest cas de 4 files, en la posició a la qual cal arribar. En el cas del triangle, els moviments no són tan simples, ja que a cada fila hi ha un nombre senar de caselles i, per això, els moviments esperats

[34] Com es pot observar, la demostració també funciona per a un cas amb un nombre parell de columnes i un nombre senar de files. Ara bé, en tal cas no tots els moviments canvien la paritat de la posició.

tindrien paritat parella. Calia fer, doncs, un petit canvi. Concretament, els triangles orientats ”cap amunt” conformaran una òrbita, i la resta, una altra. D'aquesta manera, en fer un moviment, els nombres que ocupen una casella orientada ”cap amunt” passen a una casella orientada ”cap amunt” adjacent, i els que ocupen una casella orientada ”cap avall” passen a una casella orientada ”cap avall” adjacent (tots els nombres en el mateix sentit).

Pel que fa a la nomenclatura dels moviments, escriurem una D (de *down*, en anglès) per referir-nos als moviments paral·lels al costat inferior (que van de dreta a esquerra), una L (de *left*) pels que siguin paral·lels al costat esquerre (que van de baix a dalt), i una R (de *right*) pels paral·lels al costat dret (que van de dalt a baix). Després de cada lletra s'escriurà un número en funció de la proximitat amb el costat referit, ordenant-los de més proper a més llunyà.

Exemple 27. *En la graella d'exemple anterior,* $D2$ *equival a* $(9,7,5)(8,6)$*,* $L3$ *equival a* $(14,9)$*, i -*$R1$ *equival a* $(16,9,4,1)(15,8,3)$*. Així, doncs,* $D2L3$-$R1$-$R1$ *es correspon a* $((16,9,4,1)(15,8,3))^2 \circ (14,9) \circ (9,7,5)(8,6)$.

7.2.1 Estudi del grup que generen els moviments de la versió triangular

En aquest cas, pel fet d'haver-hi dues òrbites sabem d'entrada que, si n és el nombre de caselles i G el grup generat pels moviments elementals, l'homomorfisme $h : G \to S_n$ és injectiu però no pas exhaustiu i, per això, serà més útil estudiar les dues òrbites per separat. Es pot observar que n és un quadrat perfecte (ja que és la suma de dos nombres triangulars consecutius, un per a cada òrbita). El nombre de caselles de cada òrbita és $n \pm \sqrt{n}$[35].

Ara bé, com es pot observar, les dues òrbites són iguals, amb l'única diferència que una té una fila més que l'altra, cosa irrellevant perquè volem trobar el grup que es genera en cada òrbita per separat. En el paràgraf següent, per denotar els moviments entendrem qualsevol òrbita com una òrbita gran, ja que com hem dit, són iguals i, de fet, l'òrbita petita d'un triangle d'f files és l'òrbita gran d'un triangle d'$f-1$ files. Prendrem a més el triangle de 4 files d'exemple.

[35] L'enèsim nombre triangular és $\frac{n(n+1)}{2}$. Com es pot comprovar, la suma de dos nombres triangulars consecutius és un quadrat: $\frac{(n^2-n)+(n^2+n)}{2} = n^2$. Si f és el nombre de files del tauler, les òrbites tenen $\frac{f(f+1)}{2}$ i $\frac{f(f-1)}{2}$ elements, i el nombre de caselles del tauler n és $n = f^2$, de manera que el nombre de caselles de les òrbites és $n \pm \sqrt{n}$. Trobareu més informació sobre els nombres triangulars a https://ca.wikipedia.org/wiki/Nombre_triangular

Com que en cada òrbita hi ha tres files formades per només dos elements (una en cada direcció), en cada òrbita tenim almenys 3 transposicions (els moviments elementals $D(\sqrt{n}-1)$, $L(\sqrt{n}-1)$ i $R(\sqrt{n}-1)$; en l'exemple $(2,4)$, $(14,9)$ i $(5,12)$ respectivament). Com hem fet abans, en aquest cas també podem veure que qualsevol transposició es pot aconseguir a partir d'alguna d'aquestes (a la qual anomenarem $\tau=(a,b)$) mitjançant el conjugat de τ per σ. Per aconseguir la transposició (c,d) (c i d pertanyen a la mateixa òrbita), necessitem que $\sigma(c)=a, \sigma(d)=b$. Construirem σ de la següent manera: portarem primer c fins a a, i després portarem d fins a una casella que toqui el costat del triangle no paral·lel a la transposició i diferent del que toca ara c mitjançant moviments paral·lels a la transposició. Finalment, durem d fins a l'altra casella de τ (b) mitjançant moviments paral·lels al costat que toca. El moviment resultant és σ, com es pot observar. Per tant, existeix un homomorfisme bijectiu entre el grup generat en cada òrbita i el grup simètric de transposicions corresponent.

Si tornem al triangle complet veurem que, a més a més, cap transposició de l'òrbita gran no afecta l'òrbita petita i, per tant, donada una posició de l'òrbita petita es pot aconseguir qualsevol posició de l'òrbita gran[36]. Això ens indica que G serà el producte directe dels dos grups simètrics corresponents: sigui n el nombre de caselles, $G=S_{(n+\sqrt{n})/2}\times S_{(n-\sqrt{n})/2}$.

Per acabar de verificar aquests resultats, podem utilitzar el SAGE per mirar si l'ordre del grup generat pels moviments del trencaclosques (en el cas concret del triangle de 4 files) és isomorf al producte directe de l'S_6 i l'S_{10}:

```
sage: G=PermutationGroup(['(2,4)','(5,7,9)(6,8)','(10,12,14,16)(11,13,15)',\
    '(1,2,5,10)(3,6,11)','(4,7,12)(8,13)','(9,14)','(1,4,9,16)(3,8,15)',\
     '(2,7,14)(6,13)','(5,12)'])
sage: G.is_isomorphic(direct_product_permgroups([SymmetricGroup(6),\
    SymmetricGroup(10)]))
    True
```

8 Programació amb Python

En aquest apartat s'explica el procediment seguit a l'hora de fer servir el llenguatge de programació Python. És un llenguatge que nosaltres no coneixíem i que, per tant, vam

[36] De fet, el moviment que és transposició de l'òrbita petita repetit tres vegades és una transposició de dos elements de l'òrbita petita que no afecta l'òrbita gran i, per tant, es pot fer el mateix raonament en aquesta altra direcció.

haver d'aprendre des de zero. Per això, vam anar-nos proposant diferents objectius relacionats amb el programa que nosaltres volíem crear per tal de dissenyar primer programes que se li assemblessin i que ens servissin de base per al nostre programa definitiu.

Ens vam introduir al Python amb un llibre que trobareu a la bibliografia (vegeu [3]) i que ens va proporcionar les nocions bàsiques d'aquest llenguatge de programació (definició i ús de llistes, variables, operacions, funcions...). Havíem utilitzat Scratch[37] prèviament, i això ens va permetre començar a utilitzar Python amb una certa experiència en l'àmbit de la programació. D'aquesta manera, vam estalviar una mica de temps i feina, que d'altra banda hauríem hagut de destinar a aprendre a programar.

8.1 Primer programa: les Torres de Hanoi

L'objectiu d'aquest programa era que trobés una solució òptima d'una certa versió (amb més o menys discos) del problema de les Torres de Hanoi. Per tant, un dels objectius d'aquest programa era que no es restringís a un número concret de discos sinó que fos tan ampli com fos possible.

Cal tenir en compte que hi havia bàsicament dues maneres d'aconseguir que el programa trobés la solució. D'una banda, podia fer una cerca exhaustiva de solucions, i d'altra banda, podia aplicar l'algorisme explicat a l'annex que resol el problema de manera òptima. Nosaltres, però, ens vam decantar per la primera opció perquè s'assemblava més als programes resolutoris que hauríem de desenvolupar més tard, ja que d'aquests no disposàvem d'algorismes resolutoris (òptims) coneguts.

D'aquesta manera, vam dissenyar l'algorisme següent:

1. Crear la posició inicial en funció del nombre de discos i afegir-la en una llista de posicions.

2. Mirar a quines posicions donen lloc els moviments possibles en cadascuna de les posicions que cal analitzar (la posició inicial la primera vegada; després, les obtingudes en el pas 3 anterior).

3. Per cadascuna de les posicions obtingudes, si no havia aparegut anteriorment

[37] Scratch és un interessant llenguatge de programació amb finalitats educatives. Vegeu-ne, per exemple, la seva pàgina web (https://scratch.mit.edu/), la Viquipèdia (https://ca.wikipedia.org/wiki/Scratch_(llenguatge_de_programaci%C3%B3)), o l'extensa bibliografia sobre Scratch.

s'emmagatzema, així com també els moviments que hi porten (els moviments que duien a la posició anterior i el que cal afegir per arribar a aquesta).

4. Si alguna de les posicions emmagatzemades és la posició final, el programa mostra la seqüència de moviments òptima que hi duu i la seva llargada, així com també les posicions per les quals es passa durant el procés. Si no, es repeteix el procés des del pas 2 amb les posicions emmagatzemades en el pas 3.

Podem entendre que el que fa el programa és anar seguint un arbre. Un arbre (en aquest cas d'una certa cerca exhaustiva) és una estructura de nodes interrelacionats, cadascun d'ells associat a un altre node que s'anomena pare. Hi ha un node, però, que no té pare i que s'anomena arrel. En aquest algorisme, cada node és una posició, l'arrel és la posició de la qual es vol trobar la solució i dos nodes estan relacionats si es pot passar d'un a l'altre mitjançant un moviment. El programa va explorant l'arbre en totes les direccions possibles, i si troba una posició que ja havia trobat prèviament per un camí més curt, deixa d'explorar per aquell camí. Es pot observar que un arbre és un graf de Cayley vist des d'una perspectiva diferent. La profunditat o el nivell d'una posició és el nombre d'arestes que cal recórrer des de l'arrel per arribar a aquell node.

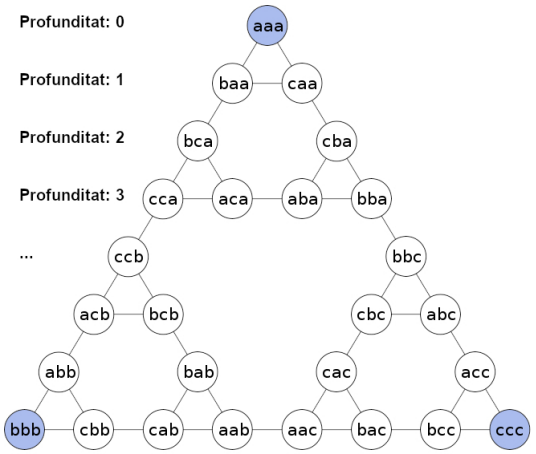


Figura 8.1: Arbre de les Torres de Hanoi amb 3 discs. Les posicions s'expressen amb la imatge de cada disc (*a* és la primera columna; *b*, la segona; i *c*, la tercera).

A l'hora de traslladar aquest algorisme al Python, les posicions s'expressen mitjançant llistes amb 3 elements, cadascun dels quals és una llista amb tots els discos (ordenats) que hi ha en la columna en qüestió (els discos de la primera columna en la primera llista, etcètera). Els moviments s'expressen amb dues lletres, la columna des d'on es desplaça el disc mogut primer i, després, la columna on arriba (A és la primera

columna, etcètera).

Si hi ha d discos, el disc més petit es numera amb el número 1, el més gran amb el número d, i la resta de tal manera que es compleixi sempre que un disc més petit té un número més petit que un disc més gran. Així, per imposar que un disc més gran mai no pot estar sobre un de més petit, es distingeix una posició com a vàlida si en totes les columnes (o, de fet, en les llistes homòlogues) els nombres estan ordenats de petit a gran.

Pel que fa a la manera de ”recordar” quina seqüència de moviments o posicions porta a una certa posició, en les llistes cada posició és precedida per la seqüència de posicions per les quals cal passar per arribar-hi, i les seqüències de moviments es desen en una altra llista en el mateix ordre.

Trobareu l'script del programa a l'annex i a la nostra pàgina web.

Exemple 28. *A continuació es mostra un exemple dels resultats del programa.*

```
torres_hanoi(2)
```

```
Els moviments són ['AB', 'AC', 'BC']
i les posicions [[[1, 2], [], []], [[2], [1], []], [[], [1], [2]], [[], [], [1,
2]]]
En total són 3 moviments
```

```
torres_hanoi(4)
```

```
Els moviments són ['AB', 'AC', 'BC', 'AB', 'CA', 'CB', 'AB', 'AC', 'BC', 'BA',
'CA', 'BC', 'AB', 'AC', 'BC']
i les posicions [[[1, 2, 3, 4], [], []], [[2, 3, 4], [1], []], [[3, 4], [1],
[2]], [[3, 4], [], [1, 2]], [[4], [3], [1, 2]], [[1, 4], [3], [2]], [[1, 4],
[2, 3], []], [[4], [1, 2, 3], []], [[], [1, 2, 3], [4]], [[], [2, 3], [1, 4]],
[[2], [3], [1, 4]], [[1, 2], [3], [4]], [[1, 2], [], [3, 4]], [[2], [1], [3,
4]], [[], [1], [2, 3, 4]], [[], [], [1, 2, 3, 4]]]
En total són 15 moviments
```

Figura 8.2: Execució del programa de les Torres de Hanoi

8.2 Segon programa: un programa per jugar al nostre trencaclosques

Com s'ha comentat anteriorment, el nostre trencaclosques no es pot construir físicament, i aquest va ser un dels motius que ens va empènyer a fer un programa que permetés jugar-hi. A més a més, tot i les diferències que hi ha entre un programa que permeti jugar a un trencaclosques i un que el resolgui, també presenten semblances, ja que la manera de definir quins moviments són possibles pot ser la mateixa en tots dos casos. Així doncs, el segon objectiu per programar va consistir en desenvolupar un programa que permetés jugar a una de les variants del nostre trencaclosques, perquè ens facilitaria el posterior

desenvolupament del programa que resolgués el nostre trencaclosques.

Com en el cas anterior, no volíem cenyir-nos a un cas particular i per això vam pensar que el programa havia de permetre jugar a qualsevol mida de la variant rectangular del nostre trencaclosques. Vam triar la variant rectangular perquè ens va semblar la més senzilla.

D'aquesta manera, vam dissenyar el següent algorisme, que és el que segueix el nostre programa:

1. Es pregunta de quina mida es vol el trencaclosques i si amb aquella mida el grup generat és l'S_n, es crea una posició aleatòria amb aquella mida i es mostra. Si el grup generat no és l'S_n (sinó l'A_n), se n'adverteix i es repeteix el pas 1.

2. El programa pregunta quin moviment es vol fer, i tot seguit el duu a terme. Finalment, mostra la posició a la qual porta aquell moviment.

3. Si s'ha arribat a la posició resolta, el programa indica el nombre de moviments emprats. Si no, es repeteix des del pas 2.

A l'hora de traslladar aquest algorisme al Python, les posicions es denoten de dues maneres diferents. En els passos 3 i 1, s'expressa mitjançant una llista que simplement conté totes les peces tal com estan ordenades, mentre que en el pas dos (i per mostrar la posició, també en el pas 1) s'expressa mitjançant una llista en la qual hi ha una subllista per a cada fila.

D'altra banda, en el pas 3, per saber si la posició a la qual s'ha arribat és la posició final, mira si els elements de la llista (denotem-la $L = [l_1, l_2, ..., l_i, l_{i+1}, ..., l_n]$) estan ordenats efectuant totes les diferències $l_{i+1} - l_i$ (estan ordenats si i només si totes aquestes diferències són positives). Pel que fa a "l'execució" dels moviments, si són moviments horitzontals cap a la dreta el programa afegeix l'últim element de la subllista (fila) en qüestió al primer lloc d'aquesta subllista i després n'elimina l'últim element, i si són cap a l'esquerra fa el mateix però amb el primer element, que posa al final de la subllista. Si és un moviment vertical de la columna enèsima, en cada subllista afegeix en l'enèsima posició l'enèsim element de la subllista anterior o posterior (segons si es mou cap amunt o cap avall), i a continuació en suprimeix l'element que ha passat a formar part de la subllista posterior o anterior (també segons si es mou cap amunt o cap avall).

Trobareu l'script del programa a l'annex.

Exemple 29. *A continuació es mostra un exemple del programa en funcionament.*

```
trencaclosques()
De quina mida vols que sigui el costat horitzontal del trencaclosques?
3
De quina mida vols que sigui el costat vertical del trencaclosques?
3
Almenys una de les teves respostes ha de ser un nombre parell.
De quina mida vols que sigui el costat horitzontal del trencaclosques?
4
De quina mida vols que sigui el costat vertical del trencaclosques?
3
[11, 3, 6, 1]
[5, 8, 9, 7]
[4, 10, 2, 12]
Quin moviment vols fer? A
[1, 11, 3, 6]
[5, 8, 9, 7]
[4, 10, 2, 12]
Quin moviment vols fer? -C
[1, 11, 3, 6]
[5, 8, 9, 7]
[10, 2, 12, 4]
Quin moviment vols fer? 2
[1, 2, 3, 6]
[5, 11, 9, 7]
[10, 8, 12, 4]
Quin moviment vols fer?
```

Figura 8.3: Execució del programa per jugar al Loopover

8.3 Tercer programa: el joc del 15

L'últim dels programes que volíem desenvolupar abans de crear el programa definitiu era un programa que resolgués òptimament posicions qualssevol del joc del 15. Aquestes posicions havien de poder ser de dues menes: triades per qui executés el programa o creades aleatòriament. D'aquesta manera, podem separar el programa en dues parts: una part que permet crear posicions aleatòries i una altra part que les resol òptimament.

Dels programes previs al programa final, segurament aquest és el més útil, ja que l'única diferència que hi ha entre aquest i el final és que aquest fa referència al joc del 15 i el programa final, al Loopover, un parell de trencaclosques que, almenys si ens fixem en la versió rectangular del Loopover, són bastant semblants.

8.3.1 Crear posicions aleatòries del joc del 15

Començarem centrant-nos, doncs, en el programa que crea posicions aleatòries del joc del 15. Com ja s'ha vist, l'única condició que cal que es compleixi per tal que una posició del joc del 15 es pugui resoldre té a veure amb la seva paritat. Si el nombre de vegades que s'ha de moure el forat fins a portar-lo a la seva casella final és senar, caldrà que la paritat de la permutació sigui senar, i si no, caldrà que sigui parella (vegeu 4.4 i l'annex).

L'algorisme dissenyat ordena aleatòriament els nombres de l'1 al 15 en una llista, i finalment, afegeix el 16 (que hi denota el forat) en una casella de tal manera que respecti la norma de la paritat. Ara bé, cal tenir en compte que, havent distribuït els nombres de l'1 al 15, el càlcul de la paritat s'efectua sense tenir en compte la posició del 16 (ja que encara no està col·locat). Per tant, es tracta de saber com canvien les "restriccions" si no tenim en compte el 16 quan calculem la paritat. Es pot observar que els moviments horitzontals no afecten la paritat (ja que l'ordre dels nombres sense tenir en compte el 16 és el mateix). En canvi, en fer un moviment vertical, canvia el signe de 3 diferències (recordeu el punt 4.4), cosa que implica un canvi de paritat (les inversions augmenten o disminueixen en 1 o en 3). Així doncs, si i només si el 16 comença en la segona o quarta files la permutació ha de ser parella (sense tenir en compte el 16), ja que caldrà un nombre parell de moviments verticals per portar el 16 al seu lloc. En la resta de casos, ha de ser senar i, si es compleixen totes aquestes condicions, la posició es pot resoldre. Havent raonat això, ja es disposa de prou informació com per desenvolupar l'algorisme assenyalat. El programa que s'hi basa el trobareu, com de costum, a l'annex.

Exemple 30. *A sota s'hi pot veure una posició resoluble creada aleatòriament.*

```
quinze()
[4, 5, 1, 13, 9, 6, 8, 14, 12, 15, 11, 16, 7, 3, 10, 2]
```

Figura 8.4: Posició resoluble creada a l'atzar

8.3.2 Càlcul de fites en el joc del 15

Hem vist en el programa de les Torres de Hanoi que hi havia dues maneres de fer que el programa trobés la solució: amb una cerca exhaustiva o mitjançant un algorisme senzill que se sap que proporciona la solució òptima. En aquest cas, però, no es pot seguir cap d'aquestes estratègies: amb una cerca exhaustiva caldria buscar entre una quantitat

ingent de successions de moviments (cal tenir en compte que existeixen $16!/2$ posicions resolubles), i d'altra banda no existeix un algorisme senzill com el de les Torres de Hanoi que proporcioni la solució òptima. Per tant, calia buscar recursos alternatius.

D'entre els llibres consultats, n'hi havia un, *Orden en el caos* [4], que parlava de programes que resolen òptimament el joc del 15. Bàsicament, s'assenyalaven algunes heurístiques per tal de reduir els arbres de cerca exhaustiva.

Una heurística és un mecanisme exploratori que avalua els progressos fets. En el cas del joc del 15, una bona heurística és el càlcul de fites. Una fita és un mínim (de moviments, en aquest cas) que se sap que serà necessari per a arribar a la solució a partir d'una certa situació i que, d'aquesta manera, es converteix en un "objectiu" (d'aquí el nom de fita).

Una de les heurístiques aplicables al joc del 15 és la suma Manhattan, que és la suma de les distàncies Manhattan entre cadascuna de les peces i la posició on han d'acabar. La distància Manhattan entre dos punts és la distància entre els dos punts mesurada amb segments paral·lels als eixos (la distància que cal recórrer per anar d'una cantonada a una altra del districte de Manhattan, per exemple). En el context del joc del 15, la posició inicial d'una certa peça i la seva posició final equivaldrien a aquests dos punts. Es pot observar que, en el joc del 15, cada peça ha de recórrer com a mínim la seva distància Manhattan per arribar a la seva posició final i, per tant, des d'una certa posició desordenada del joc del 15 caldran com a mínim tants moviments com la suma Manhattan d'aquestes distàncies.

Ara bé, com es pot observar, aquesta suma Manhattan no té en compte les interaccions entre les peces i, per això, habitualment calen més moviments que els indicats per la suma Manhattan per a resoldre el joc del 15. Com que el que volem és utilitzar-la com a heurística, és a dir, per poder descartar a priori certs camins en l'arbre de la cerca exhaustiva, com més s'aproximi el càlcul de les fites al nombre real de moviments, millor funcionarà. Per això, vam idear algunes millores per al nostre càlcul de fites:

- Si dues peces es troben a la fila o columna on han d'acabar però estan mal ordenades, almenys una haurà de sortir de la fila o columna per deixar passar l'altra i, després, tornar-hi. Per tant, calen almenys 2 moviments més dels previstos. Si en comptes de dues peces en són més, per cada inversió caldrà que "s'aparti" una peça més i, per tant, per cada inversió caldran dos moviments més.

- Si una peça és al seu lloc però impedeix el pas d'una altra peça, també n'haurà de sortir i, per tant, caldran dos moviments més. Concretament, una peça que encara no estigui col·locada necessita una casella adjacent per sortir de la casella on es troba i una altra casella adjacent que permeti arribar-hi a la peça que hi ha d'acabar, ja que en moure's una peça el forat passa a ocupar el seu lloc, que en el millor dels casos podrà ser ocupat just després per la peça que hi ha d'acabar (si n'hi ha). Per tant, cada peça no col·locada ha de ser adjacent a com a mínim dues peces no col·locades (o el forat) i, quan no sigui així, s'augmentarà la fita com s'ha indicat al principi d'aquest punt.

- Si amb qualsevol dels moviments possibles en la posició donada s'augmenta la distància Manhattan de la peça moguda, també es pot augmentar en 2.

- L'últim dels moviments consistirà o bé en moure el 12 o bé en moure el 15 des de la casella on ha d'acabar el 16, per tant, o bé el 12 o bé el 15 haurà de passar per aquella posició. Si això fa augmentar la distància Manhattan en els dos casos, com que almenys un dels dos succeirà, també es pot augmentar en 2.

- Cal tenir present que, si per una peça ja s'ha augmentat la fita prèviament, després no es pot tornar a augmentar a causa de la mateixa peça però per un altre motiu. És a dir, si ja s'ha raonat que per un cert motiu una peça s'ha "d'apartar", si posteriorment es raona que s'ha "d'apartar" també per un altre motiu, això no vol dir que la peça s'hagi d'apartar dues vegades. Amb una n'hi pot haver prou.

L'algorisme desenvolupat per al càlcul de fites, doncs, consisteix en calcular la suma Manhattan i, posteriorment, aplicar aquestes millores. A l'hora de traslladar aquest algorisme al Python, és interessant fer observacions per a cadascuna de les parts del procés del càlcul de la fita:

1. Càlcul de distàncies Manhattan. La posició s'expressa amb una llista que conté els números ordenats com en la posició en qüestió. Per calcular la distància Manhattan de cada peça, se'n calculen els components horitzontal (tenint en compte les columnes final i inicial de la peça) i vertical (tenint en compte les files final i inicial de la peça) per separat. La fila en la qual es troba una certa peça (o on ha d'acabar) s'obté amb la divisió euclidiana o entera de la posició del nombre (o la posició on ha d'acabar) entre quatre i la columna coincideix amb el residu d'aquesta divisió.

Es calcula la suma Manhattan com la suma de les distàncies Manhattan de cada nombre de la llista (inclòs el 16) i s'hi resta la distància Manhattan relativa al 16.

2. Peces invertides dins d'una mateixa fila/columna. Se separa cada fila/columna de la posició en qüestió en una llista diferent, a partir de les quals es creen llistes que només contenen les peces que ja es troben a la fila/columna on han d'acabar. Per cada inversió (diferència negativa) que hi hagi en cadascuna d'aquestes llistes, s'augmenta la fita en 2 (no es té en compte el 16). Les peces invertides s'afegeixen a una llista de peces per les quals ja no es pot tornar a augmentar la fita.

3. Peces no col·locades que no són adjacents a almenys dues peces no col·locades. La posició s'expressa amb una llista per a cada fila i per a cada columna. Als extrems de cada llista s'afegeixen dos 0 que indiquen els límits del tauler. Per comprovar que una peça no col·locada és adjacent a almenys dues peces no col·locades, es compten les seves caselles adjacents ocupades per peces col·locades no presents a la llista de peces "utilitzades" o 0. Si en són 3, s'augmenta la fita en 2, i si en són 4, en 4. Les peces col·locades que han contribuït a augmentar el mínim, s'afegeixen a la llista de caselles utilitzades.

4. Moviments possibles. S'utilitzen les mateixes llistes amb 0 que en el cas anterior per saber quines peces són adjacents al 16. Es mira si la fila o columna on han d'acabar està en el mateix "sentit" que la fila o columna del 16. Si no ho està en cap dels casos i cap d'aquestes peces es troba a la llista de peces utilitzades, s'hi afegeixen i s'augmenta en 2 la fita.

5. Últim moviment. Si ni el 15 ni el 12 es troben en la llista de peces utilitzades i ni el 15 es troba a l'última columna ni el 12 a l'última fila, s'augmenta la fita en en 2.

Com sempre, trobareu el programa a l'annex.

Exemple 31. *A continuació, calcularem la fita de la posició de sota a mode d'exemple.*

11	2	12	1
6	7	3	8
9	5		4
14	10	13	15

1. *Calculem la suma Manhattan calculant les distàncies Manhattan que separen cada nombre de la seva posició final i sumant-les. Per exemple, l'11 s'ha de moure 4 llocs*

(dos avall i dos cap a la dreta), el 2 està al seu lloc, el 12 s'ha de moure 3 posicions... Tot plegat suma 22.

2. *Observem que l'1 i el 2 ja es troben a la fila on han d'acabar però que estan mal ordenats. Per tant, per creuar-se almenys un n'haurà de sortir i tornar-hi a entrar, i per tant necessitarem 2 moviments més dels previstos. Veiem que passa el mateix amb el 13 i el 14 i amb el 8 i el 4, i per tant com a mínim caldran $22 + 2 \cdot 3 = 28$ moviments per resoldre la posició. Afegim els nombres 1, 2, 4, 8, 13 i 14 en una llista, ja que ja els hem emprat per a millorar la predicció inicial i, per això, no els podrem tornar a utilitzar.*

3. *Ens adonem que el 9 ja està al seu lloc mentre que el 13, que hauria d'anar sota seu, no hi és. Per poder col·locar el 13 haurem de moure el 9, ja que necessitarem un espai per on treure el 14 i un altre per on fer entrar el 13, i aquests espais han de ser per força les caselles ocupades pel 9 i el 10. Així doncs, sí que necessitarem moure el 9: almenys dues vegades. Caldran, doncs, almenys $28 + 2 = 30$ moviments per resoldre aquesta posició. Cal fer notar que, encara que podríem augmentar la fita amb el 2 i el 8 per aquest mateix motiu, no ho perquè ja els hem tingut en compte en el punt anterior. Afegim el 9 a la llista de nombres fets servir.*

4. *Observem que, a priori, cap dels moviments possibles en aquesta posició ens ajuda a avançar en la seva resolució, i per això potser es podria augmentar la fita en 2. No obstant, com que el 13 i el 4 ja els hem fets servir per augmentar la fita, no podem tornar a usar-los, ja que, de fet, moure el 4, per exemple, ens és ben útil (recordem que s'haurà de creuar amb el 8) a priori. Per això, en aquest punt no sumem res.*

5. *Sabem que l'últim moviment haurà de moure's o bé el 15 o bé el 12 des de la casella on ara hi ha el 15. Justament per aquest motiu, en aquest pas no augmentem la fita.*

Hem obtingut, doncs, la fita de 30 moviments. La solució òptima d'aquesta posició en té 46, i podem considerar que hem obtingut un bon càlcul de fites.

8.3.3 Resoldre òptimament el joc del 15

Havent dissenyat l'heurística de fites, ja es pot desenvolupar el programa pel joc del 15. Nosaltres partíem de la idea que, si fèiem que el programa busqués la solució des de la posició per resoldre cap a la posició final, i des de la posició final cap a la posició inicial,

caldria que el programa busqués menys, ja que arribaria un moment en què els dos arbres es trobarien amb la mateixa posició. Aleshores, només caldria ajuntar els camins que hi duguessin i ja tindríem la solució.

Per poder-ho aplicar, però, calia adaptar el càlcul de fites, ja que estava pensat per anar només cap a la posició resolta. El que vam fer va ser que el programa calculés la fita per una parella ordenada de posicions (I, F), on I és la posició inicial, i F, la final. D'aquesta manera, en comptes de donar per suposada la posició final de cadascuna de les peces, es mira quina és la seva posició en F.

A més a més, per poder desenvolupar el nostre algorisme vam definir alguns altres conceptes. Donat un conjunt de posicions obtingudes mitjançant la cerca des de la posició inicial, la *fita global endavant* serà la fita mínima del conjunt de les seves fites. Igualment, la *fita global enrere* serà la fita mínima de les fites d'un conjunt de posicions obtingudes mitjançant la cerca des de la posició final. Finalment, la *fita global* serà el valor màxim d'entre la fita global endavant i la fita global endarrere, que coincidirà amb la fita global endavant si només es busca a partir de la posició inicial i amb la fita global enrere si només es busca des de la posició final. Cal tenir present que cada fita s'actualitza després de cada nova cerca.

Amb aquest canvi i aquestes definicions, ja es pot escriure el programa. L'algorisme que vam fer servir és el següent (direm "fita d'una posició" referint-nos a la suma de la fita d'aquella posició i el nombre de moviments que hi duen):

1. Per cadascuna de les posicions obtingudes al pas 1, (la posició inicial al començament), si la seva fita és igual a la fita global endavant, es mira a quines posicions pot donar lloc que no s'hagin trobat prèviament.

2. Per cadascuna de les posicions obtingudes al pas 2, (la posició final al començament), si la seva fita és igual a la fita global enrere, es mira a quines posicions pot donar lloc que no s'hagin trobat prèviament.

3. Si en el pas 1 s'ha trobat una posició trobada prèviament en el pas 2, o en el pas 2 s'ha trobat una posició trobada prèviament en el pas 1, es retorna com a solució òptima la seqüència de moviments que hi porta des de la posició inicial unida a la seqüència invertida de moviments que hi duu des de la posició final. Si no, es repeteix des del pas 1.

Baixant-nos un programa que troba solucions òptimes del joc del 15 desenvolupat

per Herbert Kociemba (el trobareu a [5]), vam comprovar amb diverses posicions que el programa retornés una solució correcta. Desgraciadament, no era així.

Un dels problemes que vam detectar és el següent. Podria passar que la condició del punt 3 fos certa, però que la llargada de la solució que oferís fos major que la de qualsevol de les fites globals. En aquest cas, podria passar que la solució trobada no fos òptima. Per tal de corregir aquest error, vam fer alguns canvis en l'algorisme:

- En els passos 1 i 2, s'utilitza la fita global i no les fites globals endavant i endarrere.

- En el pas 3, es dona l'algorisme per acabat si i només si en el pas 1 s'ha trobat una posició trobada prèviament en el pas 2, o en el pas 2 s'ha trobat una posició trobada prèviament en el pas 1, i en qualsevol d'aquests dos casos la llargada de la solució trobada no és major que fita global. Si és superior a aquesta fita global, es retorna la solució indicant que no té per què ser òptima, i continua buscant.

L'altre problema que vam detectar és que, havent trobat una posició amb una seqüència d'm moviments, després el programa trobava una altra seqüència de n moviments que també hi duia, on $n < m$, però de la qual no quedava constància perquè aquella posició ja havia aparegut abans. Això contradiu el principi d'optimalitat de Bellman, que diu que, donada una seqüència òptima de decisions, tota subseqüència d'ella mateixa també és òptima. Per tal d'evitar aquest problema, vam fer una petita variació dels passos 1 i 2: en lloc de mirar a quines posicions pot donar lloc "que no s'hagin trobat prèviament", el que fa és mirar a quines posicions pot donar lloc "que no s'hagin trobat prèviament o, en tal cas, que el nombre de moviments que hi duen sigui menor que el nombre de moviments que calen arribant-hi d'una altra manera trobada anteriorment".

Havent fet aquests canvis, vam observar que el nombre de posicions estudiades pel programa augmentava fins a una certa profunditat i, aleshores, començava a disminuir. Vam plantejar-nos, aleshores, si era realment útil que el programa busqués alhora endavant i endarrere. Vam fer una segona versió del programa, simplificant-lo de tal manera que busqués només endavant, i vam comparar el nombre de posicions estudiades en ambdós casos mitjançant algunes gràfiques.

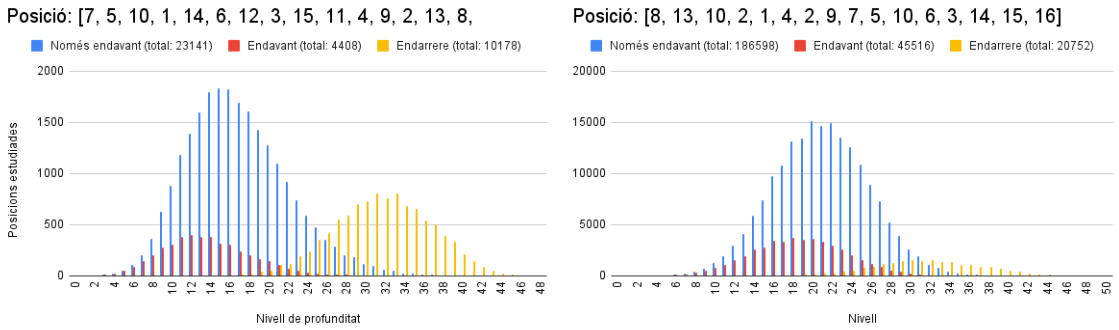


Figura 8.5: Nombre de posicions estudiades en funció de la profunditat, en el programa que busca només endavant i en el que busca endavant i endarrere. En blau, les posicions estudiades buscant només endavant; en vermell i en groc, les posicions estudiades endavant i endarrere alhora, respectivament. S'expressen les posicions amb una llista de les imatges dels nombres.

S'hi pot observar que, com s'ha dit anteriorment, el nombre de posicions estudiades augmenta inicialment, i abans d'arribar a la meitat de la corba, comença a disminuir. Tot i això, la quantitat de posicions estudiades quan es busca endavant i enrere alhora és significativament menor que quan es busca només endavant. Això és així perquè, anant endavant i endarrere, el programa troba la solució abans d'arribar a la conclusió que la solució òptima té un cert nombre de moviments. Llavors, quan arriba a aquesta conclusió, ja no cal que desenvolupi tots els camins candidats a ser òptims, perquè ja en té un; mentre que si busca només endavant, li cal desenvolupar un gran nombre de posicions fins a arribar a la posició final. Lògicament, a més profunditat més gran és aquesta diferència, mentre que quan la solució és molt curta pot arribar a ser contraproduent[38]. De la mateixa manera, com més s'aproxima la fita calculada inicialment al nombre de moviments de la solució, més subtil és la diferència, i en canvi, quan l'"equivocació" és major, més útil resulta aquesta cerca dual[39]. Per tant, la cerca en dos sentits presenta dos avantatges: sovint ofereix diverses solucions (entre les quals, trobada al final, l'òptima), i troba la solució òptima en un temps menor (quan la llargada de la solució no és molt petita). Tot i això, el temps que es necessita segueix sent bastant gran.

Com s'explicarà més tard, més endavant vam saber que mirar si una posició havia aparegut o no era bastant contraproduent: necessita molt més temps. Vam canviar-ho en el programa que busca endavant i endarrere, evitant, això sí, que després d'un moviment fes el seu invers. Fent alguns canvis més que no afectaven l'algorisme però que permetien

[38] En tot cas, no és rellevant: es tracta de casos dels quals troba la solució molt ràpidament.
[39] Podeu trobar un parell de gràfiques més a l'annex.

que s'executés més ràpid[40], vam aconseguir que el nostre programa trobés solucions òptimes en menys de 5 minuts per a la majoria de les posicions provades. Tot i això, per comprovar que aquestes solucions són òptimes necessita bastant temps més.

Trobareu aquest programa definitiu a l'annex, i a la pàgina web hi podeu veure, a més d'aquest mateix programa, els programes que busquen només endavant o endavant i endarrere alhora, i que comproven si una certa posició ha aparegut abans o no.

Exemple 32. *Es pot observar una posició escollida a l'atzar i resolta òptimament pel programa.*

```
resol(quinze())
[10, 9, 12, 13]
[5, 14, 1, 16]
[8, 4, 3, 15]
[2, 11, 7, 6]
La llargada mínima de la solució és 46
La llargada mínima de qualsevol solució és 48
La llargada mínima de qualsevol solució és 50
La llargada mínima de qualsevol solució és 52
[13, 12, 1, 14, 4, 3, 14, 4, 9, 1, 4, 13, 12, 4, 1, 10, 5, 9, 3, 14, 13,
3, 9, 8, 14, 13, 7, 6, 15, 7, 6, 11, 13, 14, 2, 13, 14, 9, 10, 1, 3, 10,
8, 2, 9, 6, 10, 8, 2, 5, 1, 2, 6, 10, 7, 12, 8, 7, 11, 15] és una solució
de 60 moviments. No sabem si és òptima.
La llargada mínima de qualsevol solució és 54
[1, 3, 4, 14, 3, 12, 13, 1, 12, 4, 7, 6, 15, 12, 4, 13, 1, 4, 12, 7, 13,
3, 9, 10, 5, 8, 14, 13, 6, 11, 13, 14, 2, 13, 14, 9, 10, 1, 3, 10, 8, 2,
9, 6, 10, 8, 2, 5, 1, 2, 6, 10, 7, 12, 8, 7, 11, 15] és una solució de 58
moviments. No sabem si és òptima.
[1, 3, 4, 8, 2, 11, 7, 6, 15, 4, 8, 14, 3, 12, 13, 1, 4, 8, 12, 13, 1, 4,
8, 12, 13, 3, 9, 10, 5, 2, 14, 13, 6, 7, 11, 14, 13, 9, 10, 1, 3, 6, 7,
11, 14, 13, 9, 10, 2, 5, 1, 2, 6, 7, 11, 15] és una solució de 56
moviments. No sabem si és òptima.
La llargada mínima de qualsevol solució és 56
És una solució anterior és òptima.
```

Figura 8.6: Posició resolta mitjançant IDA* endavant i endarrere

8.4 Quart programa: Loopover

Després d'haver dissenyat els programes explicats anteriorment sobre el Joc del 15, vam adonar-nos que calia que ens informéssim més sobre els diversos algorismes de cerca existents per així evitar temps tan llargs. Per això, vam decidir consultar algun llibre sobre

[40] D'entre aquests canvis, són destacables l'expressió de les posicions amb nombres (en lloc de llistes) a l'hora de comprovar si una posició havia aparegut abans, o l'ús de dues funcions per al càlcul de fites, una per a la cerca endavant i una altra, per a la cerca endarrere. Això permet estalviar temps perquè amb el càlcul de la fita endavant es pot donar per suposada la posició final de cada nombre, i cal realitzar menys cerques.

el tema (vegeu [6]) i adreçar-nos a experts en teoria de grups i programació. Vam obtenir la resposta d'Herbert Kociemba i de David Joyner.

Amb les seves respostes i el que vam trobar per la nostra banda, vam veure que ens calia informar-nos millor sobre els algorismes de ramificació i poda i, concretament, sobre IDA*.

8.4.1 Conceptes previs: ramificació i poda i IDA*

Els algorismes de ramificació i poda segueixen el següent esquema:

1. S'obté una solució qualsevol del problema mitjançant un algorisme voraç (un algorisme dissenyat amb aquesta finalitat). En un problema com el nostre, si se sap que totes les seves variants es poden resoldre amb un cert nombre de moviments com a molt (el que es coneix com a "God's number"), també es pot utilitzar aquest nombre. Aquesta solució s'anomena incumbent, i qualsevol solució òptima serà millor (més curta, en el nostre cas) que aquesta. Això marca una cota superior.

2. S'utilitza el càlcul de fites per establir un mínim. Qualsevol branca de l'arbre de cerca que tingui una fita major o igual a la incumbent, es poda: no cal explorar-la.

La manera d'explorar l'arbre de cerca varia segons l'algorisme; en alguns casos pot ser més útil fer-ho d'una certa manera i, en uns altres, d'una altra. La rapidesa de l'algorisme dependrà, bàsicament (a més de la manera d'explorar), de com d'acurat sigui el càlcul de les fites.

L'algorisme IDA*[41] és igual al que nosaltres havíem fet servir pel Joc del 15, amb l'única diferència que no comprova si una posició havia aparegut anteriorment, que és l'aspecte al qual el nostre algorisme dedicava més temps. Es pot fer servir tant per buscar només endavant com per buscar només endarrere.

Podem representar l'algorisme IDA* (quan busca només endavant) amb el següent diagrama de flux:

[41] IDA* és acrònim de *Iterative Deepening A**. A* és un altre algorisme de cerca.

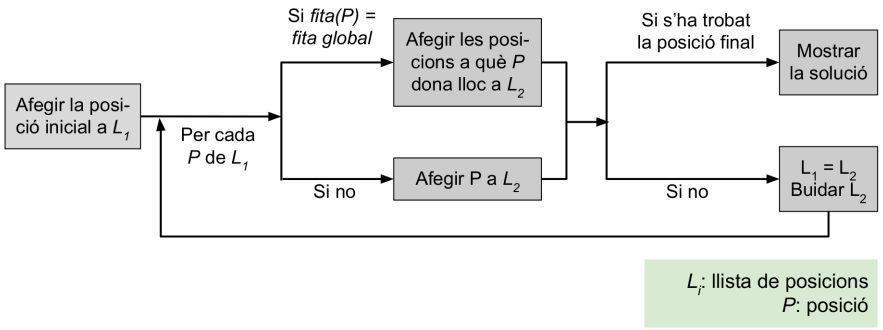


Figura 8.7: Diagrama de flux de l'algorisme IDA* aplicat a trencaclosques de permutacions.

Sabent això, vam veure clar que ens calia desenvolupar un càlcul de fites per al nostre programa. També fou en aquest moment que vam millorar definitivament el programa sobre el joc del 15, com s'ha explicat anteriorment.

8.4.2 Càlcul de fites

Per fer aquest càlcul, vam basar-nos en el càlcul de fites que havíem utilitzat en el Joc del 15, és a dir, en la suma Manhattan (la suma de les distàncies Manhattan de cadascun dels nombres) amb algunes millores. Cal dividir el nombre resultant entre 4 perquè cada moviment mou quatre nombres. Anomenarem "moviments senzills" els moviments d'una sola peça i "moviments complets", els moviments reals i possibles del joc. Les millores a la suma Manhattan eren, concretament:

1. Efectua la suma de les distàncies horitzontal i la vertical per separat, ja que els moviments són o bé horitzontals o bé verticals, i a l'hora de dividir entre 4 el resultat final això és important.

2. Per cadascuna de les files i columnes, mira quins nombres hi són que també hi han de ser en la posició final. Mira les distàncies respectives, i compta quants d'aquests nombres han de sortir de la fila/columna per tal d'aconseguir que les distàncies respectives siguin "correctes". Suma 2 a la suma Manhattan horitzontal o vertical segons convingui.

3. Si una fila està completada, però s'han de fer moviments verticals, suma 1 per cada moviment vertical complet que s'hagi de fer, ja que la fila completada s'haurà de desfer almenys en part (no es pot assegurar que es pugui sumar més de 1). Fa el mateix amb els moviments horitzontals.

4. Si, quan ja ha tingut en compte tots els aspectes anteriors, la paritat de la permutació no encaixa amb el nombre de moviments complets, augmenta aquest nombre en 1.

És interessant ressaltar alguns aspectes a l'hora de traslladar aquest algorisme al llenguatge de programació:

1. Distàncies horitzontals i verticals. Com amb el joc del 15, identifica les files amb la divisió euclidiana de la posició d'un nombre entre 4, i les columnes amb el residu d'aquesta divisió.

2. Posicions respectives dins una fila. Crea una llista de la fila amb els nombres que hi han de romandre, i també 0 per tal de mantenir les distàncies entre ells. Si hi ha alguns nombres mal col·locats respecte dels altres, ho compta de tal manera que els que estan mal col·locats siguin el mínim.

Exemple 33. *Calcularem la fita de la posició següent a mode d'exemple.*

12	2	7	15
5	6	13	4
3	10	1	8
11	14	9	16

1. *Calculem les distàncies horitzontal i vertical per separat. Per exemple, el 12 s'haurà de moure* 1 *posició cap a l'esquerra, el 4 no s'haurà de moure horitzontalment i el 3 s'haurà de moure* 2 *llocs (cap a la dreta o cap a l'esquerra); i alhora el 12 haurà de fer* 2 *moviments verticals, el 15,* 1 *(cap amunt) i el 3 també* 2*. Tot plegat suma* 12 *moviments horitzontals senzills i* 15 *moviments verticals senzills.*

2. *Observem la quarta columna: el 4, el 8 i el 16 hi han de romandre. Ara bé, tot i que el 4 està ben col·locat respecte del 8, no ho està respecte del 16. Per tant, com que el 4 i el 8 estan respectivament ben col·locats i el 16 no, caldrà que com a mínim el 16 s'aparti per després tornar a la seva columna i, per tant, els moviments horitzontals simples necessaris seran* $12 + 2 = 14$*. Cal fer notar que si estiguessin ordenats* $16, 8, 4$ *(sense cap altre nombre enmig), aleshores tots estarien malament respecte de qualsevol dels altres dos i, per això, podríem augmentar la distància horitzontal senzilla en* 4 *i no només en* 2*.*

3. *Calculem els moviments complets necessaris dividint cadascuna de les distàncies entre* 4*: caldran almenys* 4 *moviments horitzontals complets (*$14/4 = 3$ *i el residu és diferent de* 0*) i almenys* 4 *moviments verticals complets (*$15/4 = 3$ *i el residu és diferent de* 0*).*

4. *La segona columna està completada però, no obstant, com que haurem de fer almenys* 4 *moviments horitzontals complets, l'haurem de desfer i, en conseqüència, necessitarem almenys* 4 *moviments simples més, és a dir,* 1 *moviment complet més.*

5. *Calculem els moviments complets totals (*$4 + 4 + 1 = 9$*) i la paritat de la posició (senar). Com que la paritat de moviments complets i la de la posició coincideixen, la fita serà de* 9 *moviments.*

Amb aquestes fites, vam escriure un programa per al Loopover igual a l'última versió dels programes del joc del 15 (utilitzant IDA*), però les posicions que havia d'estudiar eren moltes. Com que el programa trigava molt, vam buscar maneres de fer que anés més ràpid, encara que potser impliqués trobar solucions que no eren òptimes. I aquí és on la teoria de grups ens va ajudar més: d'una banda, vam decidir fixar-nos en quines posicions presentaven relacions de simetria; i d'altra banda, vam donar importància a algun subgrup del joc.

8.4.3 Posicions relacionades simètricament

En el nostre joc, el tauler té uns límits arbitraris. Què diferencia una certa casella d'una altra? Res, ja que els moviments que s'hi poden aplicar són els mateixos (totes les files i columnes es mouen de la mateixa manera) i tota casella és adjacent a quatre altres caselles. Aleshores, si expresséssim una posició dient, de cada nombre, com està desplaçat respecte de la seva posició original, podríem inventar una funció que, donada una certa posició π, en donés una altra ρ on cada nombre estigués desplaçat tantes caselles (respecte del seu lloc) com en π un altre nombre, sense repetir-ne cap. Si la "transformació" donada per aquesta funció fos una simetria, és a dir, mantingués adjacents les caselles adjacents (i, per tant, també els moviments adjacents), les dues posicions tindrien solucions equivalents: sabent la solució òptima d'una d'elles podríem saber la de l'altra.

Concretament, si expressem aquestes transformacions simètriques com permutacions, si π és una posició i σ una simetria, aleshores, $\rho = \sigma^{-1}\pi\sigma$, el conjugat de π per σ,

és una posició relacionada amb π simètricament.

Exemple 34. *En el primer exemple de sota s'hi veu, en la posició de l'esquerra (la identitat), com els límits del tauler són arbitraris. En aquest cas, se l'ha representat amb la fila de sota a sobre. S'hi aprecia que els moviments possibles segueixen movent els mateixos conjunts d'elements. Si apliquem el moviment B (en la posició representada d'aquesta manera) obtenim la segona de les posicions. Si finalment tornem a representar la posició com estem habituats a fer, veiem que és com si haguéssim efectuat el moviment A, i no pas B. En altres paraules, B equival a A; C, a B; D, a C; A, a D; 1, 2, 3 i 4 són equivalents a si mateixos; i l'invers d'un moviment equival a l'invers del moviment corresponent.*

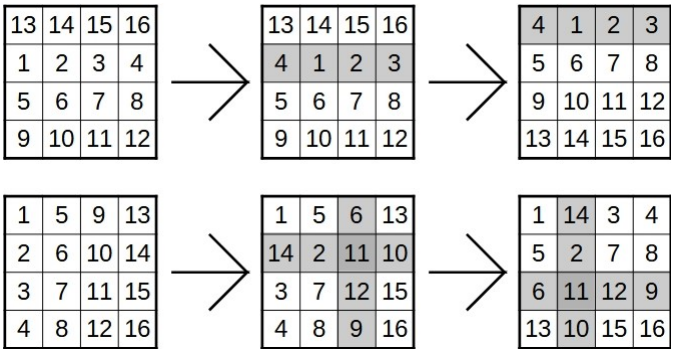


Figura 8.8: Dos exemples de posicions relacionades simètricament

Fixem-nos ara en el segon exemple. En aquest cas no es tracta d'un simple moviment de l'"enquadrament" del tauler, sinó que hem invertit la identitat respecte d'una diagonal. Tot i això, els nombres que en la identitat eren adjacents ho segueixen sent, i per tant els moviments segueixen movent els mateixos conjunts de nombres que en la identitat. Per això, si ara realitzem els moviments B i -3 (es veu en la segona posició) i a continuació tornem a invertir el tauler com hem fet a l'inici, veiem que obtenim una posició com si haguéssim fet un moviment 2, en lloc de B, i un moviment -C, en comptes de -3. És a dir, A equival a 1; B, a 2; C, a 3; D, a 4; i viceversa; i l'invers d'un moviment equival a l'invers del moviment corresponent.

Com s'ha dit anteriorment i com s'acaba d'exemplificar, el fet que una simetria (considerarem el primer exemple com una simetria) mantingui adjacents les caselles adjacents ens assegura que, en aplicar-ne una, després, una seqüència de moviments (o posició), i finalment, l'invers de la simetria (les simetries no són sempre d'ordre 2) obtindrem una altra posició a la qual podrem arribar amb una seqüència de moviments com la inicial però desplaçada, rotada, invertida... Per tant, sabent la solució d'una posició

i la simetria que la relaciona amb una altra, podem saber també la solució d'aquesta altra. Això és, com ja s'ha dit, perquè com que les caselles adjacents ho segueixen sent, els moviments segueixen afectant als mateixos conjunts de nombres i els moviments adjacents també segueixen essent-ho (i l'invers d'un moviment equival a l'invers del moviment corresponent).

Les simetries es poden composar, i qualsevol resultat de la composició de dues simetries serà una altra simetria. Per tant, com que a més les simetries tenen invers i podem considerar la identitat com una simetria, les simetries formen un grup G. De la mateixa manera que abans hem dit que $\rho_1 = \sigma^{-1}\pi\sigma$ està relacionada amb π simètricament, $\rho_2 = \sigma\pi\sigma^{-1}$ també ho estarà. Aleshores, podem dir que aquest grup actua en l'S_{16} amb l'acció $\phi_\sigma(\pi) = \sigma\pi\sigma^{-1}$. Efectivament, es compleix que:

1. Tot element $\sigma \in G$ dona lloc a una funció $\phi_\sigma : S_{16} \longrightarrow S_{16}$.

2. $\phi_{Id}(\pi) = \pi$.

3. $\phi_{\tau\circ\sigma}(\pi) = (\tau\sigma) \circ \pi \circ (\tau\sigma)^{-1} = \tau\sigma\pi\sigma^{-1}\tau^{-1} = \phi_\tau(\phi_\sigma(\pi))$.

Prenent com a generadors d'aquest grup les simetries respecte de l'eix horitzontal central, respecte de l'eix vertical central, una rotació de 90° amb centre al mig del tauler i els desplaçaments conjunts de totes les files o columnes (cal recordar que els límits del tauler són arbitraris), amb SAGE podem veure que aquest grup té 128 elements:

```
sage: G=PermutationGroup(['(1,13)(2,14)(3,15)(4,16)(5,9)(6,10)(7,11)(8,12)',\
    '(1,4)(2,3)(5,8)(6,7)(9,12)(10,11)(13,16)(14,15)',\
    '(1,4,16,13)(2,8,15,9)(3,12,14,5)(6,7,11,10)',\
    '(1,2,3,4)(5,6,7,8)(9,10,11,12)(13,14,15,16)',\
    '(1,5,9,13)(2,6,10,14)(3,7,11,15)(4,8,12,16)'])
sage: order(G)
 128
```

Encara que 128 simetries són moltes, no són prou com per què valgui la pena intentar valdre-se'n per millorar el programa (cal tenir present que $|S_{16}| = 16! \approx 2,09 \cdot 10^{13}$), i d'altra banda la implementació d'aquesta heurística no és gens fàcil. Per tant, vam haver de descartar aquesta hipòtesi i centrar-nos en altres aspectes.

8.4.4 Subgrups de l'S_{16}

Un dels mètodes utilitzats per resoldre trencaclosques complicats com el Cub de Rubik es basa en subgrups, i per això hi ha qui l'anomena "mètode del subgrup". Si G_0 és el grup que generen els moviments elementals del trencaclosques, es busquen una sèrie de subgrups tals que:

$$\{e\} = G_n \subset G_{n-1} \subset ... \subset G_1 \subset G_0$$

Aleshores, una posició g que es vulgui resoldre pertanyerà a G_0. El mètode consisteix en buscar una seqüència de moviments que permeti obtenir, composada amb la posició inicial, un element del següent subgrup, i així successivament[42]. D'aquesta manera, es divideix el problema en d'altres de més petits i més fàcils de resoldre. Per això, no és un mètode que ofereixi solucions òptimes, encara que les solucions trobades amb aquest mètode puguin servir per aquesta finalitat.

Basant-nos en un document (vegeu [7]) que vam trobar en el fòrum (vegeu [8]) al qual ens havia adreçat Herbert Kociemba, vam avaluar diferents possibilitats a l'hora d'escollir quins subgrups eren més adients per el nostre objectiu. Vam arribar a la conclusió que el millor era agafar només un subgrup entre la posició per resoldre i la identitat, perquè d'aquesta manera les solucions trobades s'aproparien més a les òptimes, i d'entre aquests subgrups, vam decantar-nos pel subgrup generat pel conjunt de moviments $\{C, D, 4\}$ perquè divideix el problema en dues parts bastant semblants. D'aquesta manera, cap de les dues parts hauria de ser gaire difícil de ser resolta.

L'algorisme dissenyat resol les dues parts amb IDA*. Tot i això, té una petita modificació: en cadascuna de les parts, quan troba una solució, no s'assegura que sigui òptima. En general, els resultats ja són prou bons i seria invertir-hi massa temps per les millores que es podrien aconseguir. És una cerca que no sol excedir gaire els 10 minuts, i sovint triga menys.

Com que SAGE és una interfície de Python que permet accedir a diversos paquets, aquest programa el vam desenvolupar amb SAGE. Així, vam poder utilitzar algunes funcions (la paritat, l'invers, la composició de funcions...) o objectes que eren permutacions pròpiament dites, coses que en els programes anteriors no eren tan necessàries.

D'altra banda, va caldre fer alguns petits canvis en l'algorisme, que s'expliquen a continuació:

[42] Per aquest motiu, els inversos d'aquestes seqüències de moviments pertanyeran a la classe lateral (per l'esquerra) de g. Vegeu l'annex per saber què són les classes laterals.

- Durant la primera part:

  - No es consideren els moviments que no mouen els nombres $1, 2, 3, 5, 6, 7$, que són els que romanen fixos durant la segona part.

  - En el càlcul de fites, només es tenen en compte els nombres $1, 2, 3, 5, 6, 7$. Això fa que perdin bastanta precisió.

  - Es considera que els arbres de cerca endavant i endarrere s'han trobat si dues posicions (una de cada arbre) tenen col·locats els nombres $1, 2, 3, 5, 6, 7$ als mateixos llocs.

  - Per buscar endarrere, es parteix de la identitat, que té els nombres $1, 2, 3, 5, 6, 7$ al seu lloc final (d'aquesta primera part). Un cop trobada la solució, partint de la posició per resoldre, s'apliquen els diversos moviments per obtenir la posició que realment s'obté duent a terme aquells moviments (que no és, normalment, la identitat). Finalment es retorna aquesta posició i la seqüència de moviments que hi porten.

- Durant la segona part:

  - Només són possibles els moviments $C, D, 4$ i els seus inversos. Encara que en el càlcul de fites això no es té en compte, la seva precisió es veu poc afectada.

- En ambdues parts:

  - Quan els dos arbres es troben, a més d'invertir la seqüència de moviments que duen a la posició comuna buscant enrere, s'han d'agafar els seus inversos (A per -A, -1 per 1, etc.).

Trobareu el programa a l'annex.

Exemple 35. *A continuació, es mostra una posició resolta pel programa definitiu.*

```
resoldre(aleatoria())
```

```
[16, 6, 11, 9]
[7, 12, 10, 4]
[2, 1, 14, 8]
[15, 13, 3, 5]
La primera part té com a mínim 4 moviments.
La primera part té com a mínim 5 moviments.
La primera part té com a mínim 6 moviments.
La primera part té com a mínim 7 moviments.
Ha acabat la primera part. Té 9 moviments.
[1, 2, 3, 16]
[5, 6, 7, 8]
[15, 11, 4, 14]
[12, 10, 13, 9]
La segona part té com a mínim 6 moviments.
La segona part té com a mínim 8 moviments.
La segona part té com a mínim 10 moviments.
Una solució és ['-2', '-B', '-4', '-A', 'C', '2', '3', '2', '-1', '4', '-D',
'4', '-C', '4', '-D', '-C', '-4', '-C', '4', '-C', '4'] . Té 21 moviments.
```

Figura 8.9: Posició resolta mitjançant el mètode del subgrup.

8.4.5 Funció *word_problem* de SAGE

Per tal de tenir un programa amb un abast més ampli i una mostra de tot el que es pot aconseguir amb la teoria de grups, vam decidir utilitzar la funció *word_problem* de SAGE, com ens havia indicat David Joyner[43]. És una funció que, donada una permutació i un conjunt de generadors, troba una seqüència de moviments (no té per què ser òptima) que hi duen, utilitzant GAP.

Aquesta funció utilitza, entre d'altres, els grups quocient. Un grup quocient és un grup format per les classes laterals mòdul un subgrup[44]. Ara bé, cal tenir present que, per tal que l'operació d'aquest grup estigui ben definida, el resultat d'aquesta operació no ha de dependre del representant[45] de la classe. En l'algorisme que nosaltres havíem desenvolupat prèviament i del qual s'ha parlat abans, les classes laterals mòdul el subgrup $\langle\{C, D, 4\}\rangle$ no formen un grup perquè l'operació entre elles no està ben definida. Tot i això, aquesta funció *word_problem* ens ensenya tot el que es pot assolir aprofundint més en la teoria de grups, fixant-se, per exemple, en els grups quocient.

A part de l'ús d'aquesta funció, vam dissenyar-ne d'altres que creen el conjunt de generadors del grup en funció del nombre de files i columnes, així com també verifiquen

[43] Es coneix com a *word_problem* el problema consistent en trobar una seqüència de moviments (o elements generadors) que permetin obtenir una certa posició.

[44] Les classes laterals són conjunts d'elements d'un grup, tots amb la mateixa cardinalitat. Vegeu detalladament què són les classes laterals a l'annex.

[45] El representant de la classe és un membre d'aquest conjunt.

que una certa posició sigui resoluble (el programa permet la creació de graelles amb un nombre senar tant de files com de columnes, que com s'ha explicat anteriorment creen el grup alternat i no el grup simètric). Amb aquest conjunt de funcions, podem obtenir un programa que resolgui bastant ràpidament posicions aleatòries d'una graella rectangular d'una mida qualsevol. Es tracta d'unes funcions que en el futur es podrien modificar de manera senzilla per obtenir un programa que resolgui les altres graelles ideades.

Trobareu el programa a l'annex, així com també a la pàgina web.

Exemple 36. *A continuació, es mostra l'últim dels nostres programes en funcionament.*

```
P=Permutations(range(1,17)).random_element()
print(P)
resoldre(PermutationGroupElement(P),4,4)
```

```
[4, 5, 7, 8, 2, 11, 12, 1, 6, 10, 13, 15, 9, 3, 16, 14]

[['(9,10,11,12)', -2],
 ['(3,7,11,15)', 1],
 ['(9,10,11,12)', 1],
 ['(3,7,11,15)', -1],
 ['((3,7,11,15)', -1],
 ['(9,10,11,12)', 1],
 ['(9,10,11,12)', -1],
 ['(2,6,10,14)', 1],
 ['(9,10,11,12)', -1],
 ['(2,6,10,14)', -1],
 ['(5,6,7,8)', -1],
 ['(3,7,11,15)', -1],
 ['(5,6,7,8)', 1],
 ['(1,5,9,13)', -1],
 ['(1,2,3,4)', -1],
 ['(1,5,9,13)', 1],
 ['(1,2,3,4)', -1],
 ['(4,8,12,16)', 1],
 ['(1,2,3,4)', -1],
 ['(13,14,15,16)', 2],
 ['(3,7,11,15)', -1]]
```

Figura 8.10: Programa que utilitza la funció *word_problem* per resoldre posicions de Loopover.

Part IV

Conclusions del treball

Hem arribat a l'última part del treball, que ens permetrà fer una anàlisi final dels nostres resultats i avaluar tot el procés, així com també ens ajudarà a establir els avantatges i utilitats de tots els conceptes apresos de cara la futur.

Partint de l'objectiu de crear un joc propi, vam haver d'estudiar quin tipus de característiques (tipus de moviments o de tauler, posició final coneguda…) es relacionaven amb el que volíem estudiar, cosa que ens va portar a la classificació dels jocs. Vam arribar a l'apartat de jocs de permutacions de David Joyner i vam començar a veure'ls des d'una nova perspectiva, així com també vam investigar sobre els orígens de cadascun o les seves curiositats.

Una vegada vam establir les bases quant a les característiques dels jocs, vam voler investigar què els fa diferents entre si matemàticament, i així és com vam arribar a la teoria de grups, i, més concretament, als grups de permutacions. Aquesta part del treball ens ha aportat beneficis com saber reconèixer quines posicions són o no resolubles a través de conceptes com la paritat d'una permutació. També ha estat interessant entendre la naturalesa dels moviments dels jocs de permutacions aprofitant l'expressió d'una permutació en cicles disjunts, o els moviments generadors del grup d'un joc.

Dit això, l'aprofundiment en l'aplicació de la teoria al Cub de Rubik ha facilitat la posterior creació del nostre propi joc. Durant aquesta invenció hem sigut capaços de crear un joc aplicant el coneixement adquirit, en comptes d'intentar entendre'l i analitzar-lo després d'haver-lo ideat. Així doncs, hem pogut inventar un joc amb diverses variants basat en la composició d'una graella sense un nombre concret de caselles. Cada variant té alguna particularitat, per exemple n'hi ha algunes que generen l'S_n corresponent i d'altres que no. Es pot dir, per tant, que l'objectiu de fer un joc ampli i sense limitacions ha quedat ben assolit, ja que el fet de no tenir un nombre concret de caselles permet augmentar o disminuir la dificultat del repte que es proposa al gust del jugador. Això planteja un enorme nombre de possibilitats partint d'una estructura de moviments i objectius similars, respectant el que fa del Loopover un joc de permutacions.

Quant a la programació, considerem que hem aprofitat una eina molt potent i que, com plantejàvem a les hipòtesis, forma part del camí a seguir en el nínxol d'investigació

treballat. Començant per ordres senzilles i per la comprensió de les funcions més bàsiques, per a després, a poc a poc, ampliar el ventall de possibilitats, hem realitzat un enriquidor procés d'aprenentatge mentre tractàvem d'aprendre la manera de sintetitzar tots els passos per a arribar a una finalitat determinada. Així doncs, ens hem endinsat en el món de l'algorísmia i hem començat a idear els nostres propis algorismes, raonant els passos més adients i la manera de traduir-los al llenguatge del Python. Per a aquesta part hem fet servir conceptes com la suma Manhattan i l'ús d'heurístiques, que definitivament ens han permès establir algunes certeses (com per exemple un nombre mínim de moviments necessaris per a resoldre una posició) i relacionar-les constantment amb els nostres algorismes. També ha sigut de gran importància en aquest apartat el coneixement de la teoria de grups, ja que ens ha permès, entre d'altres coses, assegurar-nos que plantegem al programa posicions resolubles. Alhora, gràcies a aquesta teoria hem pogut identificar similituds entre els grafs de Cayley i els arbres de cerca o entre diverses posicions d'un mateix joc, quan presenten relacions simètriques.

D'altra banda, com sol passar en tot tipus de recerca científica, ens hem trobat amb moltes dificultats durant el treball i ens hem hagut d'equivocar incomptables vegades. Ordres incorrectes, errors de codificació, incoherències entre passos, o simplement oblidar-se d'escriure una majúscula… Ens hem trobat constantment amb inconvenients d'aquest estil que ens han portat a reflexionar sobre per què un programa funciona o no ho fa, entre d'altres coses. Una de les parts de més dificultat en aquest sentit ha estat la de l'optimització de l'algorisme i del programa, és a dir, una vegada ja s'ha estructurat una base funcional i sense errors, com es pot treure el màxim rendiment del programa i com es pot aconseguir que els passos siguin a la vegada el més eficients i eficaços possible? Respondre aquestes preguntes resultava encara més difícil quan vèiem que el programa funcionava i feia el que se li demanava, però trigava molt a dur a terme els nostres algorismes fins a trobar la solució òptima. Per tant, com és natural, a mesura que vam anar treballant en jocs amb un major nombre de moviments possibles que els anteriors, els períodes de temps fins a trobar la solució òptima es van anar accentuant. En aquest sentit sabíem que programes com el del cas del quadrat 4x4 del Loopover havien de treballar amb grups enormes com l'S_{16}, cosa que es feia notar encara més si no utilitzàvem heurístiques. Hem pogut comprovar la seva gran utilitat, que augmenta com més s'ajusten a la realitat. Per a realitzar aquesta labor d'optimització, hem hagut de formular hipòtesis, comprovar-les i extreure conclusions moltes vegades, de manera

que hem modificat el codi constantment per a posteriorment veure si els canvis introduïts eren fructífers i per què. El que més ens ha funcionat a nosaltres, tot i que en un principi endarreria els resultats i no semblava el més adient, ha estat realitzar la cerca endavant i endarrere simultàniament. Així, hem pogut aprofitar una de les característiques dels jocs de permutacions com és saber la posició de què es parteix i a quina es vol arribar. També, però, ens hem trobat que sovint plantejàvem podes imperfectes i que, per tant, el programa analitzava massa posicions, la qual cosa pot augmentar significativament el temps que triga el programa en acabar. Relacionat amb això, ens hem plantejat fins a quin punt és útil per a la nostra eina de resolució tenir en compte si ja s'ha passat per una posició o si això només allarga el procés. En un principi ens resultava molt lògic pensar que sí, però més endavant hem trobat amb algorismes com el que ens va ensenyar el matemàtic Herbert Kociemba, l'IDA*, que no ho contempla en absolut. Aquest algorisme ens ha servit per crear un programa que resol el joc del 15 i el Loopover. Per a aquest últim, la teoria de grups ens ha ajudat a implementar un mètode de resolució que troba solucions ràpidament, encara que no assegura que siguin òptimes.

Com s'ha vist al treball, hem acabat tractant de millorar el nostre programa al màxim i de buscar també una manera més ràpida, però a la vegada menys pròpia, de resoldre el joc òptimament. Gràcies a l'orientació del matemàtic David Joyner, hem conegut la funció anomenada *word_problem* i l'hem incorporada al SAGE per tal d'aconseguir solucions més ràpides. Encara que existeixin recursos com aquests, una part dels objectius del treball ha estat la d'aconseguir inventar un programa propi a partir del qual hem après tant de programació com de teoria de grups, fins i tot si al final cal reconèixer que es pot millorar i que per tant no és perfecte. Considerem que el que ens aporta el *word_problem* és la demostració que la nostra feina es pot optimitzar encara més i que encara ens queden moltes coses per millorar, mentre que també hem après a fer-la servir i veure els seus punts forts.

Amb tot això, podem concloure que la nostra hipòtesi envers la utilitat de la teoria de grups a l'hora de fer un programa que resol un joc de permutacions és correcta, però amb alguns matisos. Si bé és cert que la teoria de grups és una àrea de les matemàtiques amb moltes sortides i que ens ha ajudat força per a entendre millor els jocs, la qual cosa també era important de cara a idear els programes, no té una relació estrictament directa amb l'àmbit informàtic i sovint ens ha costat aplicar-la. Així doncs, pensem que cal aprofundir més en la teoria de grups per tal de poder treure'n grans fruits en la programació.

Ens sembla que potser hem dedicat massa temps a l'apartat matemàtic del treball i que l'hauríem pogut aprofitar per a aprofundir més en l'àmbit de la programació, ja sigui practicant-la més, dedicant-nos més a trobar i entendre altres algorismes, llegint altres fonts que ens poguessin aportar noves funcions o simplement familiaritzant-nos amb el llenguatge. A més a més, creiem que ens hauria anat bé treballar la programació des d'un punt de vista teòric abans de començar a programar.

També hem pogut extreure conclusions personals d'aquesta experiència, ja que hem hagut de treballar diferents àmbits als quals no estem acostumats, però poden ser de molta utilitat de cara al futur. Per començar sabíem que anàvem a treballar conceptes que surten fora del currículum de batxillerat, ja que ni la part teòrica ni la pràctica en formen part i representaven reptes molt nous per a nosaltres. Nou ha estat també l'àmbit de la recerca científica, concretament la vessant més matemàtica. En molts moments ens ha resultat difícil la comprensió d'algunes fonts com el llibre *Adventures in group thepry* (vegeu [1]); sobretot al començament, quan estàvem menys acostumats a l'ús de símbols matemàtics i cert vocabulari científic. Això va millorar a mesura que anàvem avançant, i, finalment, hem aconseguit llegir i entendre amb més fluïdesa els texts que ens resultaven més difícils inicialment, mentre que a la vegada hi hem sabut trobar amb més facilitat la relació amb la resta del treball i la possible utilitat per al futur del projecte. De la mateixa manera, la redacció d'un treball de caire científic amb certa pretensió de professionalitat ens ha aportat experiència, i ha estat una pràctica que ens pot servir més endavant, triem el camí que triem. També tenim clar que l'Overleaf, l'editor que hem utilitzat per a redactar el projecte, ha estat molt útil a l'hora de plantejar el treball escrit. Aquesta eina ens ha permès escriure expressions matemàtiques o representar algunes figures de la manera més adient i còmoda, és bastant diferent als documents convencionals als que estem acostumats però ens ha aportat molts avantatges.

Més enriquidor encara ha estat el nostre inici en el món de la programació, un món que ens interessava molt des d'un començament però que no sabíem fins a quin punt ens podia obrir portes i noves possibilitats. El Python és un llenguatge de programació molt ampli que ens ha servit per a assolir l'objectiu principal de resoldre el nostre joc de manera òptima. No ens ha resultat fàcil treballar-hi des d'un començament, però ens hem adonat que tenir coneixements previs d'un altre programa com és l'Scratch ens ha ajudat molt a fer la transferència al Python. Dit això, considerem que seria de molta utilitat i estaria molt adaptat als temps actuals introduir aquest llenguatge com a part de la

docència des de ben petits, adaptant-lo a la dificultat de cada curs, és clar. Hem vist de primera mà que és una eina realment beneficiosa i facilitadora que pot millorar molt la competència tecnològica dels estudiants, mentre que també pot ajudar a no evitar-la des d'un principi per pensar que és massa difícil de fer anar. Així doncs, podem concloure que els entesos en el tema no menteixen quan diuen que aprendre un llenguatge de programació no només obre les portes d'aquest mateix, sinó que també facilita molt la transferència cap a altres formes de programació; realment ens agradaria haver conegut aquesta eina molt abans i, a partir d'ara, no refusarem la idea de fer-la servir.

Una part fonamental del treball han estat els problemes que hem anat enfrontant durant el procés, en aquest cas no pel que fa a errors en l'algorisme d'un programa o quelcom semblant, sinó respecte a les dificultats generals i les errades de plantejament d'alguns aspectes. Considerem que des d'un inici ens vam voler endinsar en una investigació amb un objectiu massa optimista i del qual en sabíem més aviat poc. La falta de coneixements previs ha estat un inconvenient a l'hora de determinar quins passos calia seguir o fins a quin punt calia aprofundir en la teoria de grups. En alguns casos també hem pogut plantejar dubtes al nostre entorn, ja que tenim persones al voltant que, si bé no són expertes en el nostre projecte en particular, han tractat de donar-nos alguna indicació o idea que ens pogués servir amb bona voluntat. De cara al final del treball també vam decidir posar-nos en contacte amb els dos matemàtics mencionats anteriorment: Herbert Kociemba i David Joyner. Va ser un plaer rebre els seus consells i ambdós ens van aportar informació de gran utilitat. Amb la resposta de Herbert Kociemba vam descobrir que el "nostre joc" ja existia en la variant del quadrat i del rectangle i que teníem la possibilitat de jugar-hi o d'informar-nos en fòrums de gent més experimentada, alhora que ens va presentar l'algorisme IDA*. David Joyner ens va recordar que estàvem treballant amb nombres molt grans i que per tant l'alta duració dels nostres programes era normal, tot i que el més important de la seva col·laboració va ser la introducció de la funció *word_problem*.

El nostre treball forma part de camps que es troben en constant recerca i desenvolupament com són la programació orientada a jocs i l'àlgebra computacional, i és per això que creiem que amb una mica més de pràctica i dedicació es podria anar molt més enllà. Un dels primers passos seria la programació de totes les versions que hem proposat del Loopover; com ja hem dit, en un principi vam triar la del quadrat perquè ens va semblar la que té més similituds amb el joc del 15 i probablement ens anava a resultar més fàcil,

però tenint en compte els punts comuns entre totes les variants, segurament es podria fer un programa per a cada cadascuna de les diferents graelles. De la mateixa manera, hi ha la possibilitat d'aprofundir en l'estudi dels homomorfismes i les relacions entre els diversos trencaclosques. També creiem que pot ser interessant la implementació de noves heurístiques, sobretot en el càlcul de fites, i l'exportació de la informació i pràctica que hem realitzat a altres tipus de joc. Si ens centrem en el programa més important del treball, el programa basat en el mètode del subgrup, pensem que es pot millorar sobretot la primera part; concretament, es podria fer que el subgrup escollit s'adaptés mínimament a la posició que cal resoldre (variant, per exemple, quina zona 3×2 es resol primer), així com també es podria refinar el càlcul de fites.

Finalment, esperem que aquest projecte sigui d'utilitat no només per a nosaltres, sinó també per a aquells lectors que es puguin haver vist interessats pel món de la programació, o de les matemàtiques que es troben rere coses tan comunes com els jocs que hem vist des de petits.

Referències

(1) JOYNER, D., *Adventures in Group Theory: Rubik's Cube, Merlin's Machine, and Other Mathematical Toys*; Baltimore: Johns Hopkins University Press, 2008, ISBN: 9780801890130.

(2) GANESAN, A. Cayley graphs and symmetric interconnection networks, 2017.

(3) BONNAUD, S.; SAVINAS, C., *Numérique et sciences informatiques*; 3.0; Bordas, 2021, ISBN: 2047338336.

(4) SANTOS, J.; VALEIRAS, R., *Orden en el caos: el mundo de los rompecabezas matemáticos*; Córdoba: Almuzara, 2006, ISBN: 848858640X.

(5) KOCIEMBA, H. Fifteen Puzzle Optimal Solver http://kociemba.org/themen/fifteen/fifteensolver.html (cons. 10-10-2022).

(6) FRANQUESA I NIUBÒ, C., *Algorísmia comentada*; Textos docents; 383; Barcelona: Publicacions i Edicions de la Universitat de Barcelona, 2013, ISBN: 9788447537150.

(7) ZURAWSKY, Z. Loopover Research https://docs.google.com/spreadsheets/d/1gij270TVlA1sHmvvuyMXYKToGGpQKoBGbT2yYNwBe6I/edit#gid=0 (cons. 27-11-2022).

(8) Loopover god's number upper bounds: 4×4, asymptotics, etc. https://www.speedsolving.com/threads/loopover-gods-number-upper-bounds-4%5C%C3%5C%974-asymptotics-etc.75180/ (cons. 27-11-2022).

(9) ANTOINE, R.; CAMPS, R.; MONCASI, J., *Introducció a l'àlgebra abstracta: amb elements de matemàtica discreta*; Manuals. Matemàtiques; 46; Bellaterra: Universitat Autònoma de Barcelona, Servei de Publicacions, 2007, ISBN: 9788449025150.

(10) CASTELLET, M.; LLERENA, I., *Àlgebra lineal i geometria*; Manuals. Matemàtiques; 1; Universitat Autònoma de Publicacions, Servei de Publicacions, 1988, ISBN: 8474882844.

(11) DE GUZMÁN, M., *Aventuras matemáticas*; Barcelona: Labor, 1986, ISBN: 8433551132.

(12) FARRELL, P., *Math adventures with python: an illustrated guide to exploring math with code*; San Francisco: William Pollock, 2019, ISBN: 9781593278670.

(13) CERDÀ, X.; ALFORCEA, A., *El llibre dels enigmes*; Barcelona: Barcanova, 2011, ISBN: 844892505X.

(14) DEAN, R. A., *Classical Abstract Algebra*; Nova York: Harper & Row, 1990, ISBN: 006041601.

(15) BANDELOW, C., *Inside Rubik's Cube and Beyond*; Springer Science & Business Media, 2012, ISBN: 9781468477795.

(16) GARCÍA, B., *Manual del cubo de Rubik*; García, Berta, 2018, ISBN: 1790810833.

(17) ARCHER, A. F. *The American Mathematical Monthly* **1999**, *106*, 793 - 799, DOI: 10.1080/00029890.1999.12005124.

(18) KORF, R. E. *Artificial Intelligence* **1985**, *27*, 97 - 109, issn: 0004-3702, DOI: https://doi.org/10.1016/0004-3702(85)90084-0.

(19) SERESS, Á., *Permutation Group Algorithms*; Cambridge Tracts in Mathematics; Cambridge University Press, 2003, DOI: 10.1017/CBO9780511546549.

(20) LAURITZEN, N., *Concrete Abstract Algebra: From Numbers to Gröbner Bases*; Cambridge University Press, 2003, DOI: 10.1017/CBO9780511804229.