Pràctica 2 del primer semestre – PAED
Zapatería Zeballos 2

Alex Cano Gallego
Curs: 2023 - 2024
PAED

Índex

Introducció

Per aquesta segona pràctica del primer semestre, ens demanen solucionar dos problemes d'optimització combinatòria a la Sabateria Zaballos.

El primer problema se centra en empaquetar sabates de manera que el cost total de cada caixa no superi els 1000 euros i com a màxim podem tenir 6 sabates en cada caixa, evitant així aranzels addicionals. A més, cal considerar descomptes i recàrrecs basats en característiques específiques de les sabates, com ara la marca, la talla i la puntuació.

El segon problema implica dividir l'inventari de manera que el valor total dels productes a les dues botigues sigui tan equilibrat com sigui possible, evitant que una botiga superi l'altra en termes de popularitat i vendes.

Per solucionar aquests problemes, he implementat diferents algoritmes d'optimització combinatòria, permetent explorar exhaustivament les possibles solucions i garantir configuracions òptimes tant per a l'enviament de caixes com per a la divisió de l'inventari.

Aquesta pràctica no només soluciona els problemes pràctics de la sabateria, sinó que també permet comparar les diferents estratègies d'optimització aplicant els conceptes que em vist a classe.

Llenguatge de programació escollit

En aquesta segona pràctica, per a la implementació també he utilitzat el llenguatge de programació “Java” amb l’IntelliJ IDEA com a entorn de desenvolupament. El principal motiu d’aquesta dedició, igual que en la primera pràctica és per la experiència que tinc alhora de programar amb aquest llenguatge. També, ja que havia ja implementat la primera pràctica en “Java” i a l’haver de reutilitzar el codi anterior, facilitava així la meva feina feta anteriorment.

Tal i com havia comentat en la primera pràctica, la orientació en objectes que ofereix aquest llenguatge és molt beneficiós, ja que per exemple, he pogut crear la classe “Caixa” en aquesta pràctica per facilitar aquesta gestió del problema del enviament de caixes.

Finalment, com que la implementació anterior ja utilitzava la llibreria d’ArrayList, he cregut convenient seguir utilitzant aquesta estructura per gestionar els diferents algorismes, ja que “Java” ofereix una gran quantitat de llibreries molt útils i que poden ser molt beneficioses en un futur.

Il·lustració 1: Logo Java

Enviament de caixes

En aquest primer problema, se'ns demana solucionar l'enviament de caixes, aquestes no poden superar el 1000€ i no poden tenir més de 6 sabates. També, és gestiona els descomptes que aquestes sabates poden causar en cada caixa. Llavors, per poder fer aquesta gestió de la millor manera, he decidit crear la classe Caixa on es manipula tots aquest moviments per trobar la millor solució.

Backtracking

El primer algorisme per resoldre aquest problema es el backtracking, que és una tècnica que realitza la resolució de problemes d'optimització combinatòria, que es basa a explorar totes les possibles configuracions d'una solució per trobar la millor opció. Llavors, en el context de la pràctica, aquest algorisme s'ha implementat per resoldre el problema de l'enviament de caixes de sabates, amb l'objectiu de minimitzar els costos associats als aranzels.

La classe BacktrackingEnviament és la classe principal d'aquesta solució. El seu propòsit és gestionar el procés d'empaquetament de les sabates en caixes, assegurant que cada caixa compleixi les restriccions de preu i capacitat. L'algorisme inicia amb una configuració arrel, representada per la classe ConfiguracioEnviament, que conté una llista inicial de sabates i caixes buides. Des d'aquesta configuració, l'algorisme explora recursivament totes les maneres de distribuir les sabates a les caixes fins quedar-nos amb la millor.

Aquesta classe ConfiguracioEnviament exerceix un paper crucial en mantenir l'estat actual de la caixa. Aquesta classe no només guarda les caixes i les sabates restants, sinó que també implementa mètodes per generar configuracions successores. Aquesta generació de successors permet a l'algorisme de backtracking explorar totes les combinacions possibles d'empaquetatge.

Finalment, el procés del backtracking avança seleccionant successors de manera recursiva i avaluant si cada configuració és una solució vàlida. Si una configuració supera el preu màxim permès per a una caixa, l'algorisme retrocedeix i descarta aquesta branca de possibilitats. Aquesta tècnica de poda és important per reduir l'espai de cerca i millorar l'eficiència. Quan es troba una configuració vàlida que compleix totes les restriccions, es compara amb la millor solució trobada fins al moment, actualitzant-la si la nova configuració és millor.

Branch and bound

El segon algoritme és el de branch and bound, que també és una tècnica que realitza la resolució de problemes d'optimització combinatòria que millora l'eficiència del procés de cerca en prioritzar les configuracions més prometedores i descartar les que no poden millorar la solució actual.

Aquest algorisme s'implementa a la classe BranchAndBoundEnviament, que és el component central d'aquesta solució. La seva funció també és gestionar el procés d'empaquetament de les sabates en caixes, assegurant que cada caixa compleixi les restriccions de preu i capacitat. A diferència del backtracking, que explora totes les possibles configuracions de manera exhaustiva, el branch and bound utilitza una cua de prioritat per ordenar les configuracions segons el seu cost, explorant primer les que semblen més prometedores. La classe ConfiguracioEnviament també continua sent important en aquesta implementació, ja que representa l'estat actual la caixa.

L'algorisme de branch and bound comença amb una configuració arrel que inclou totes les caixes buides i la llista completa de sabates. Utilitza una cua de prioritat per gestionar les configuracions, prioritzant-les amb menys cost. A cada iteració, s'extreu la configuració amb el cost més baix de la cua i es generen les configuracions successores. Aquestes successores són avaluades i afegides a la cua de prioritat si tenen el potencial de millorar la solució actual. Si una configuració supera el preu màxim permès per a una caixa, es descarta, reduint l'espai de cerca i millorant l'eficiència de l'algorisme.

La poda de configuracions ineficients és una característica molt important del branch and bound. Cada cop que es troba una configuració vàlida, es compara amb la millor solució trobada fins ara, i si és millor, s'actualitza. Aquest enfocament permet que l'algorisme es concentri en les configuracions més prometedores, així la poda redueix el cost de l'algorisme.

Finalment, fer l'ús de l'algorisme de branch and bound a la classe BranchAndBoundEnviament, juntament amb la gestió detallada de les configuracions a ConfiguracioEnviament, ofereix una solució eficient i efectiva al problema de l'enviament de les caixes.

Divisió d’inventari

Per aquest segon problema de la pràctica, se’ns demana solucionar la divisió de l’inventari. Aquest problema s’ha de solucionar utilitzant dos algorismes, un d’ells és el backtracking que ja em vist anteriorment, i l’altre el greedy. Aquets dos algorismes han de crear dues botigues amb la diferencia mínima de preu entre les dues.

Backtracking

Com em vist anteriorment, l’algorisme de backtracking és una molt bona tècnica per resoldre aquets problemes i així podem implementar-lo novament per solucionar el problema amb l’objectiu d'equilibrar el valor total dels productes entre dues botigues, evitant que una eclipsa l'altra.

Per fer aquest problema, he creat la classe BacktrackingDivisio. La seva funció principal és gestionar el procés de partició de l'inventari de sabates en dos conjunts i assegurar que la diferència en el valor total dels productes entre les dues botigues sigui mínima. L'algorisme comença amb una configuració inicial, representada per la classe ConfiguracioDivisio, que conté la llista completa de sabates i dues llistes buides per a les botigues. A partir d'aquesta configuració, l'algorisme explora recursivament totes les maneres de distribuir les sabates entre les dues botigues.

La classe ConfiguracioDivisio juga un paper molt importat com amb la classe de configuració del problema anterior. Aquesta classe no només guarda les llistes de sabates assignades a cada botiga i la llista de sabates restants, sinó que també implementa mètodes per generar les configuracions successores. Llavors, el procés de backtracking avança seleccionant successors de manera recursiva i avaluant si cada configuració és una solució completa. Si una configuració és completa, l'algorisme calcula la diferència de valor entre les dues botigues. Si aquesta diferència és menor que la millor solució trobada fins ara, s'actualitza la millor solució. Aquest mètode garanteix que es considerin totes les maneres possibles de dividir l'inventari, buscant la distribució més equilibrada.

En aquest cas, també he implementat tècniques de poda que permeten descartar configuracions parcials que no poden millorar la solució actual. Si en algun punt del procés una configuració parcial mostra una diferència de valor que ja és més gran que la millor diferència trobada, l'algorisme retrocedeix i descarta aquesta branca de possibilitats. Aquest enfocament permet reduir significativament l’espai de cerca.

Finalment, la implementació del backtracking en aquest problema de divisió de l'inventari assegura que es trobi la solució òptima en termes d'equilibri de valor entre les dues botigues.

Greedy

Per a resoldre aquest problema, he implementat també l'algorisme greedy, que és una tècnica d'optimització que es basa en la presa de decisions locals a cada pas, amb l'esperança de trobar una solució òptima.

Aquest algorisme esta implementat a la classe GreedyDivisio. La seva funció principal, com en tots els algorismes que hem vist, és gestionar el procés de partició de l'inventari de sabates en dos conjunts i mirar de minimitzar la diferència en el valor total dels productes entre les dues botigues. A diferència del backtracking, que explora exhaustivament totes les configuracions possibles, l'enfocament greedy pren decisions ràpides i directes sobre on assignar cada sabata basada en la situació actual dels inventaris de les dues botigues.

El procés de l'algorisme greedy a GreedyDivisio comença avaluant la llista completa de sabates. Després, de manera iterativa, cada sabata s'assigna a la botiga que, en aquell moment, té el menor valor total d'inventari. Aquesta assignació es fa amb l'esperança d'equilibrar els valors de les dues botigues a mesura que es processen totes les sabates. Cada decisió és localment òptima, ja que es basa únicament en la situació actual dels inventaris.

És per això, que l'eficiència de l'algorisme greedy cau en la seva capacitat per prendre decisions ràpides sense necessitat d'explorar totes les possibles configuracions. Tot i que aquest enfocament no garanteix trobar la solució òptima en tots els casos, normalment produeix resultats raonablement bons en un temps ràpid. Aquesta característica és especialment útil quan es treballa amb grans volums de dades i és necessita un bon resultat de manera ràpida.

Finalment, la implementació de l'algorisme greedy per a la divisió de l'inventari, aconsegueix una solució ràpida i pràctica al problema. La simplicitat d'aquest algorisme i la capacitat per produir solucions ràpides el converteixen en una bona opció.

Anàlisis dels resultats

Per a l'anàlisi de resultats, he realitzat diferents probes amb els datasets proporcionats. El problema obtingut amb el dataset XXXL, que al ser de 131072 elements, no he pogut comprovar-lo. Ja que la meva implementació amb ArrayList, s'ha vist afectat al intentar manejar aquestes grans quantitats de dades. Tot i així, he pogut realitzar les comprovacions amb els altres datasets. Però per al cas del algorisme Greedy, aquest si que ha sigut capaç d'executar-lo correctament i he pogut provar el funcionament amb aquest algorisme.

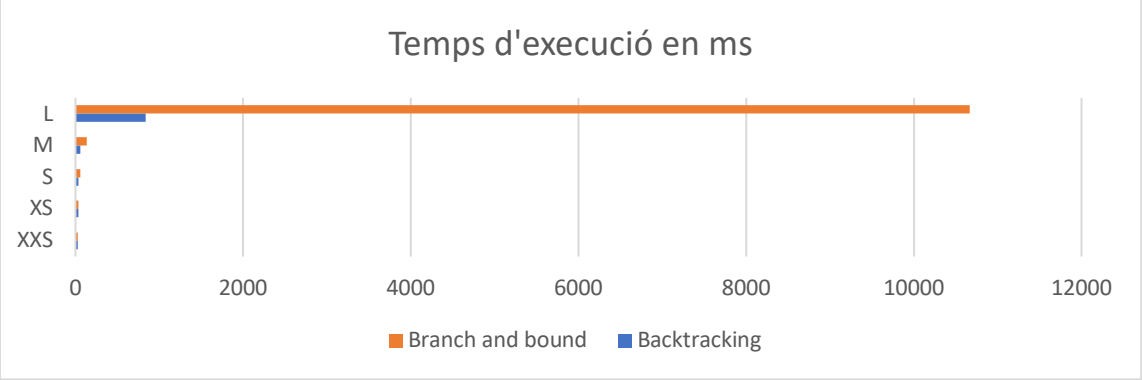
Enviament de caixes

Backtracking:

	XXS	XS	S	M	L
Quantitat	6	10	15	20	25
1	28 ms	34 ms	38 ms	55 ms	847 ms
2	28 ms	32 ms	38 ms	58 ms	810 ms
3	27 ms	32 ms	36 ms	56 ms	850 ms
4	28 ms	31 ms	39 ms	58 ms	817 ms
5	28 ms	32 ms	38 ms	56 ms	844 ms
Mitja	28 ms	32 ms	38 ms	57 ms	834 ms

Branch and bound:

	XXS	XS	S	M	L
Quantitat	6	10	15	20	25
1	31 ms	38 ms	58 ms	120 ms	10665 ms
2	30 ms	37 ms	58 ms	134 ms	10662 ms
3	31 ms	38 ms	58 ms	139 ms	10537 ms
4	31 ms	38 ms	57 ms	132 ms	10609 ms
5	30 ms	37 ms	58 ms	138 ms	10847 ms
Mitja	31 ms	38 ms	58 ms	133 ms	10664 ms



L'anàlisi dels resultats obtinguts en la implementació dels algoritmes de backtracking i branch and bound per resoldre el problema de l'enviament de caixes revela diferències significatives en termes d'eficiència i temps d'execució. Aquests resultats són molt importants per comprendre cada algorisme en diferents mides.

Per a l'algorisme de backtracking, els temps d'execució es mantenen relativament baixos i constants per als conjunts de dades més petits (XXS, XS i S), amb temps mitjos de 28 ms, 32 ms i 38 ms respectivament. A mesura que la mida del dataset augmenta, el temps d'execució creix, aconseguint una mitjana de 57 ms per a la mida M i 834 ms per a la mida L. Aquest comportament és característic del backtracking, que explora exhaustivament totes les configuracions possibles i la complexitat temporal augmenta exponencialment amb la mida del problema. Tot i així, per als petits, l'algorisme ofereix temps de resposta acceptables i consistents.

D'altra banda, l'algorisme de branch and bound presenta un increment més notable en els temps d'execució a mesura que creix el dataset. Per a les mides més petites (XXS i XS), els temps mitjans són lleugerament més grans que els del backtracking, amb 31 ms i 38 ms respectivament. Tot i així, per a inventaris mitjans i grans (M i L), el temps d'execució de branch and bound augmenta de manera dràstica, assolint una mitjana de 133 ms per a la mida M i 10664 ms per a la mida L. Aquest augment es deu a la naturalesa de l'algorisme, que encara que prioritza les configuracions prometedores i poda aquelles que no milloren la solució actual.

Finalment, gràcies a la anàlisis podem veure que l'algorisme de backtracking és més eficient, oferint temps d'execució més ràpids i consistents quan utilitzem una molt bona poda. L'elecció entre backtracking i branch and bound s'ha de basar en la mida de l'inventari i les restriccions de temps d'execució.

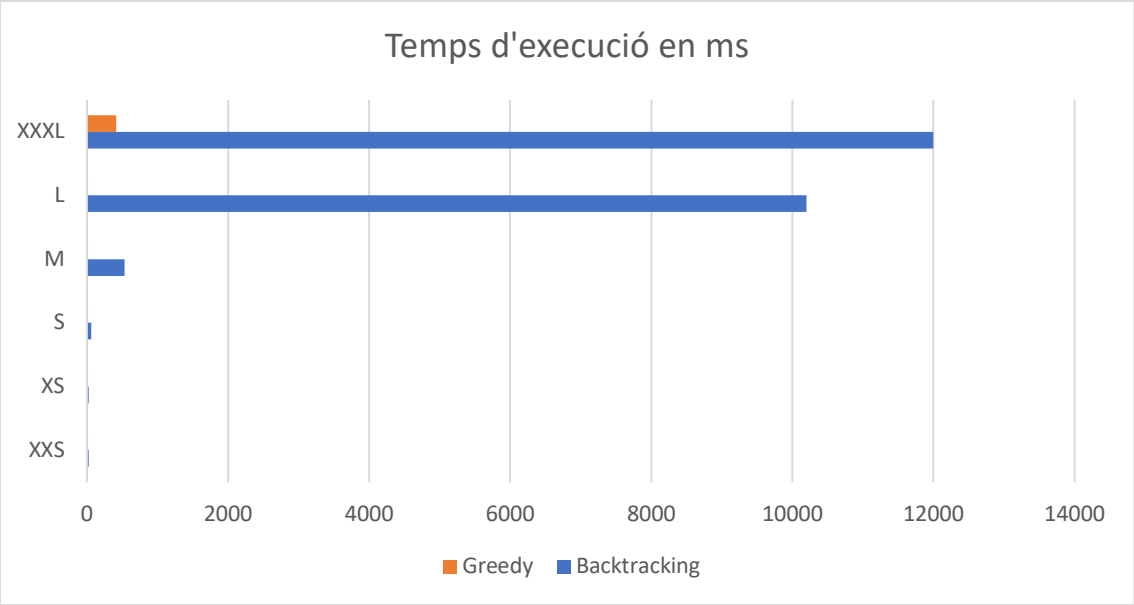
Divisió d'inventari

Backtracking:

	XXS	XS	S	M	L	XXXL
Quantitat	6	10	15	20	25	131072
1	20 ms	23 ms	62 ms	529 ms	10090 ms	No acaba
2	19 ms	23 ms	59 ms	535 ms	10219 ms	No acaba
3	20 ms	22 ms	60 ms	539 ms	10239 ms	No acaba
4	20 ms	23 ms	60 ms	538 ms	10231 ms	No acaba
5	20 ms	23 ms	61 ms	534 ms	10228 ms	No acaba
Mitja	20 ms	23 ms	61 ms	535 ms	10201 ms	No acaba

Greedy:

	XXS	XS	S	M	L	XXXL
Quantitat	6	10	15	20	25	131072
1	17 ms	18 ms	18 ms	19 ms	19 ms	417 ms
2	18 ms	18 ms	18 ms	19 ms	19 ms	418 ms
3	18 ms	18 ms	18 ms	18 ms	19 ms	419 ms
4	18 ms	18 ms	18 ms	19 ms	19 ms	418 ms
5	18 ms	18 ms	18 ms	19 ms	19 ms	418 ms
Mitja	18 ms	18 ms	18 ms	19 ms	19 ms	418 ms



L'anàlisi dels resultats obtinguts en la implementació dels algoritmes de backtracking i greedy per resoldre el problema de la divisió de l'inventari, podem veure una clara diferencia en el temps d'execució dels dos algorismes.

Per al algorisme de backtracking, els temps d'execució són raonables per als conjunts de dades més petits. Els temps mitjans són de 20 ms per a la mida XXS i 23 ms per a la mida XS, cosa que demostra l'eficiència de l'algorisme en aquests casos. Llavors, a mesura que la mida del dataset augmenta, els temps d'execució també ho fan de manera significativa. Per a mides mitjanes i grans, com S i M, els temps mitjans pugen a 61 ms i 535 ms respectivament. L'increment es torna més pronunciat amb la mida L, aconseguint una mitjana de 10201 ms. És normal que per a la mida XXXL, l'algorisme de backtracking no aconsegueix completar l'execució, però principalment pel problema d'utilitzar l'ArrayList.

En canvi, l'algorisme greedy mostra una consistència en els temps d'execució, mantenint-se extremadament baixos fins i tot per als datasets grans. Per a les mides XXS i XS, els temps mitjans són de 18 ms, i aquesta eficiència es manté constant amb les mides més grans. Per a les mides S, M i L, els temps es mantenen en un rang de 18 ms a 19 ms, cosa que destaca la rapidesa de l'algorisme. Fins i tot per a la mida XXXL, l'algorisme greedy completa la tasca en una mitjana de 418 ms, la qual cosa és significativament més ràpid que el backtracking.

Finalment, tot i que l'algorisme de greedy es significadament molt més ràpid que el backtracking, si es vol aconseguir un resultat exactament perfecte, haurem d'utilitzar l'algorisme de backtracking i amb una boa poda per reduir el temps. Ja que l'algorisme de greedy és un molt bon resultat, però no és exactament perfecte. Amb aquest anàlisis, he pogut observar perfectament la funcionalitat i observar les diferencies de cadascun i així poder triar el millor en futures implementacions.

Problemes observats

Durant el desenvolupament d'aquesta segona pràctica del semestre, d'optimització combinatòria, em van sorgir diversos problemes que van requerir solucions específiques per aconseguir implementar els algorismes de manera eficient i efectiva.

Un dels principals problemes observats va ser la necessitat de crear la classe Caixa per gestionar millor el problema de l'enviament de caixes. Aquesta classe va ser important per mantenir l'organització i el càlcul dels preus de les caixes, incloent-hi els descomptes i els recàrrecs basats en les característiques de les sabates. Aquesta classe permet afegir i eliminar sabates, calcular el preu total amb els descomptes aplicables i verificar la validesa de la configuració de la caixa. Sense aquesta estructura, la gestió de les caixes hauria estat molt més difícil gestionar aquest problema del enviament de caixes.

Un altre problema significatiu va ser la necessitat de crear classes de configuracions específiques per als algorismes de backtracking i branch and bound. Aquestes classes, ConfiguracioEnviament i ConfiguracioDivisio, han sigut molt importants per manejar l'estat de les solucions parcials i generar configuracions successores de manera eficient. Els algorismes de backtracking i branch and bound requereixen explorar múltiples configuracions de manera recursiva i, sense aquestes classes, la implementació hauria estat molt més difícil.

Un altre problema particular va sortir amb l'execució del fitxer XXXL. Tot i que els altres datasets si que s'executaven perfectament, l'algoritme de backtracking no va poder completar la tasca per a aquesta mida, a causa de la seva gran quantitat d'elements que es necessitaven gestionar per a les ArrayList. En canvi, l'algorisme greedy si que va aconseguir processar el fitxer XXXL en un temps raonable i va demostrar la seva eficiència en escenaris amb grans volums de dades.

Finalment, més enllà d'aquests problemes específics, no es van presentar altres inconvenients significatius durant el desenvolupament de la pràctica. La creació de les classes Caixa i les configuracions per als algorismes van ser solucions adequades que van permetre superar els obstacles principals.

Temps de dedicació a la pràctica

Comprensió	Investigació	disseny	implementació	Anàlisi	documentació	total
2 h	4 h	6 h	24 h	4 h	4 h	44 h

El temps total dedicat a la pràctica va ser de 44 hores, distribuït en diverses etapes clau. Vaig necessitar 2 hores en comprendre el problema. La fase de investigació em va costar 4 hores. Per al disseny unes 6 hores, durant les quals es va crear la classe Caixa. La implementació va ser l'etapa més llarga, amb 24 hores dedicades. L'anàlisi de resultats em va prendre 4 hores, avaluant el rendiment dels algorismes amb diferents mides. Finalment, la documentació em va ocupar 4 hores. Aquesta distribució del temps em va permetre resoldre tots els aspectes de la pràctica.

Conclusions

En aquesta segona pràctica del primer semestre, m'ha proporcionat coneixements molt importants sobre els diferents algorismes d'optimització combinatòria. A través de la implementació de backtracking, branch and bound i greedy, vaig poder avaluar els avantatges i limitacions de cada enfocament en els diferents contextos.

Amb l'algorisme de backtracking, he pogut veure i trobar solucions òptimes en problemes de menor escala, però la seva complexitat exponencial el fa inviable per a grans volums de dades, com és el cas del fitxer XXXL. D'altra banda, el branch and bound, també he pogut observar les seves limitacions, sobre tot quan he de gestionar grans quantitats de dades, que creix exponencialment el temps d'execució.

En canvi, per l'algorisme greedy he pogut veure la seva rapidesa, però sense destacar el seu resultat, ja que no em trobava la millor opció possible, però si una solució molt bona i aproximada. Això al converteix en una molt bona opció que tinc en un futur quan hagi de gestionar grans volums de dades i hagi d'aconseguir un molt bon resultat.

Aquesta pràctica m'ha proporcionat uns coneixements pràctics molt més clars i entenedors, ja que al posar aquest algoritmes en pràctica, he pogut observar com la teoria de classe serveix per tenir clar els conceptes d'aquets algorismes. Llavors, quan els he posat en pràctica he pogut ressaltar-los i així acabar d'entendre'ls molt millor.

Finalment, al poder provar i implementar aquests algorismes de manera clara, en un futur podré tenir una millor decisió de quin és el millor algorisme en cada context. Així, podré ressaltar els meus coneixements d'aquesta pràctica.

Bibliografia

- Imatge de la portada: imatge_portada
- Il·lustració 1: logo_java
- colaboradores de Wikipedia. (2023, 26 abril). *Optimización combinatoria*. Wikipedia, la Enciclopedia Libre. https://es.wikipedia.org/wiki/Optimizaci%C3%B3n_combinatoria
- GeeksforGeeks. (2024a, junio 24). *Introduction to Backtracking*. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-backtracking-2/
- GeeksforGeeks. (2024a, febrero 22). *Branch and Bound Algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/branch-and-bound-algorithm/
- GeeksforGeeks. (2024b, mayo 2). *Greedy algorithms*. GeeksforGeeks. https://www.geeksforgeeks.org/greedy-algorithms/