Programació Avançada i Estructura de Dades

Pràctica 2

Grup 12

Roger Miranda Pérez (roger.miranda)

Guillem Serra Cazorla (guillem.serra)

Índex

1. Llenguatge de programació escollit

Probablement l'apartat més complicat. Al tractar-se d'una pràctica molt enfocada en l'optimització no ens volíem precipitar a l'hora d'escollir llenguatge.

L'algoritme de Backtracking (explicat més endavant) juntament amb la mida dels fitxers donats ens obliga a tenir molt en compte els costos.

Llenguatges interpretats, com Java o Python, son molt ràpids de programar però solen ser poc òptims. D'altre banda, llenguatges compilats com C o C++ tenen un major rendiment, però amb més complicacions per programar.

Al final (per la comoditat de programar mencionada) ens vam decantar per Java, ja que llegir JSON és extremadament senzill. De no haver sigut per aquest factor, segurament haguéssim optat per C.

A diferencia de la primera pràctica (que vam utilitzar una llibreria de Maven) aquest cop hem gastat GSON, ja que és la llibreria que ens van recomanar des de programació orientada a objectes.

2. Algoritmes

Per solucionar els problemes plantejats en la Practica hem utilitzat diferents algorismes com són Backtracking, Branch and Bound o Greedy. Hem escollit per solucionar el problema de combinacions d'equips hem utilitzat el Backtracking, per el de horaris el Greedy i finalment per el de curses de relleus el Branch and Bound.

El Backtracking el qual es molt semblant a la força bruta, tot i que poda una gran quantitat de possibilitats no valides o dolentes, ja que es busquen totes les possibilitats correctes i després es valora la millor, es a dir que té la diferencia de velocitat més baixa.

El següent ha estat Branch and Bound (B&B) que es basa en una cua de prioritats tot podant els casos invàlids.

Finalment hem treballat el Greedy que es basa en una cua amb la informació (horaris) ordenats per la hora de acabament i es comparen amb el primer de la cua sempre i quan no es solapi.

2.1. Backtraking (Combinació de equips)

El Backtracking és l'algoritme més balancejat, mantenint facilitat de programació amb optimització. Aquest és un algoritme basat en força bruta però afegint poda per evitar arribar al final si sabem que la solució és incorrecte.

Hi ha altres tècniques per optimitzar més, com marcatge.

Hem dividit aquest problema amb 2 Backtracking el primer per trobar totes les combinacions correctes de equips i el segon per trobar totes les combinacions possibles de equips.

En les dos parts hem utilitzat el símil de un binari per decidir qui agafàvem.

Es a dir si tenim 5 atletes el binari corresponent seria 01011 el que voldria dir que agafaríem el atleta 0, 1 i 3 i així obtindríem una combinació del equip. El algorisme anirà generant totes les combinacions vàlides de atletes que poden formar un equip i les agrega a un ArrayList.

En la segona part un cop ja tenim totes les combinacions dels equips apliquem el mateix concepte que anteriorment obtenint les diferents combinacions amb les combinacions dels equips i mirem quina és la més optima per minimitzar la diferencia de velocitat.

2.2. Branch and Bound (Curses Relleus)

Creem una cua de prioritat amb les solucions/estimacions (utilitzant la funció proporcionada per la API per obtenir les estimacions de cost) que anem trobant. Recorrem el arbre de possibilitats podant els invàlids i agregant a la cua les possibilitats.

En la nostra implementació per evitar que es transformi amb un Backtracking (recórrer totes les solucions), hem implantat un timeout per obtenir les millors solucions (serà gairebé la millor solució tot i no recorre-les totes).

Un apunt important és que si fiques un timeout suficientment gran el algorisme es comportarà com un Backtracking tradicional.

Un apunt de cara a l'execució: hem realitzat tantes crides que la JVM es pensa que el bucle és infinit. Per tal d'evitar-ho s'ha d'ampliar l'stack mitjançant l'argument -Xss (amb 1GB funciona, però segurament amb menys memòria també).

2.3. Greedy (Horaris)

El algoritme Greedy està basat en optar per la millor solució en cada una de les iteracions. Es molt óptim per alguns tipus de problemes com en aquest problema dels horaris.

La nostra implementació del Greedy comença amb la col·locació en una cua de les diferents carreres/horaris de carreres ordenant-lo segons la seva hora de finalització, per tant el que es col·locaria al començament seria el que finalitza abans. Un cop ordenat agafem el primer sempre i quan no es solapi. Aquesta implementació permet un algoritme extremadament eficient i molt ràpid.

3. Comparativa dels algorismes

Per comparar els diferents algoritmes hem fet la mitjana de temps d'execució per cada un.

Per tal d'aprofitar codi ho hem generalitzat amb la funció *runAlgorithm*, que rep un algoritme[1] i el número de cops a executar, i retorna la mitjana de mil·lisegons.

3.1. Backtracking

Podem observar en la *Taula 3.1* que, mentre Backtracking ha resolt tant el dataset XS com el XS custom (explicat amb més detall al apartat *5. Mètode de probes*), li ha sigut incapaç de fer la S degut a la gran mida de les dades. Aquesta impossibilitat temporal es deu al cost que té i la quantitat de possibilitats que ha de mirar com a solucions.

Fitxer	Temps
XS	0ms
XS (custom)	27ms
S	>9h
M	-
L	-
XL	-
XXL	-

Taula 3.1: Comparativa de temps d'execució del Backtracking

[1] Hem definit un algoritme (amb una interfície) com una funció sense arguments que retorna el temps d'execució.

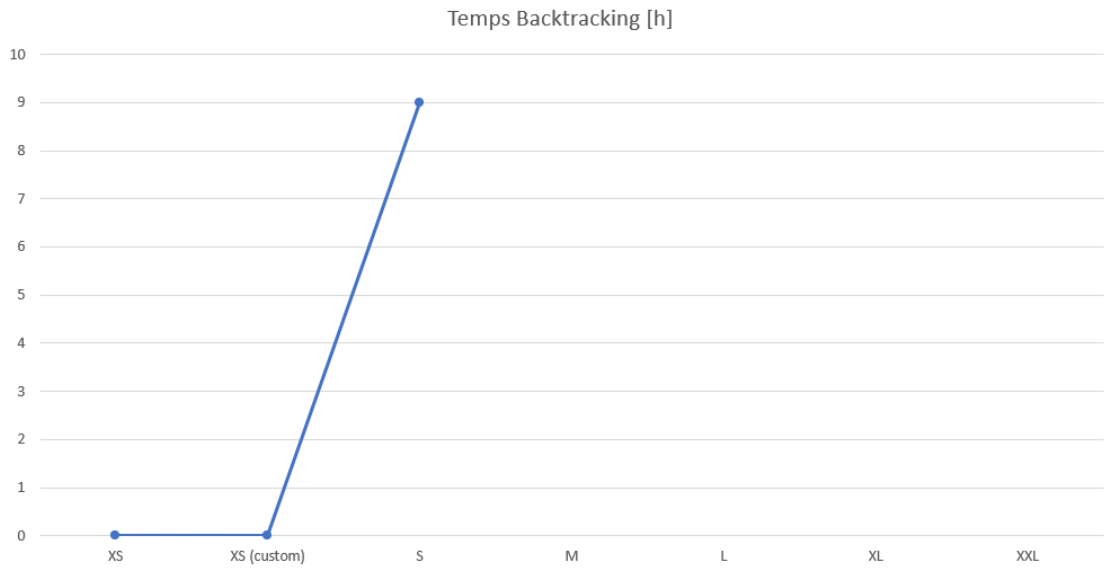

Figura 3.1: Comparativa de temps d'execució del Backtracking

3.2. Greedy

L'algoritme més òptim de tots i el que ha tardat menys temps a processar tota la execució. Ha estat molt eficient en tots els datasets; cal destacar que fins i tot XXXL aconsegueix un rendiment molt bo (sense passar de 50ms). Aquest resultat es deu a lo òptim que es aquest algorisme juntament amb que el problema a solucionar es més senzill que els altres.

Fitxer	Temps [ms]
XXS	0
XS	0
S	0
M	0
L	1
XL	4
XXL	19
XXXL	40

Taula 3.2: Comparativa de temps d'execució del Greedy

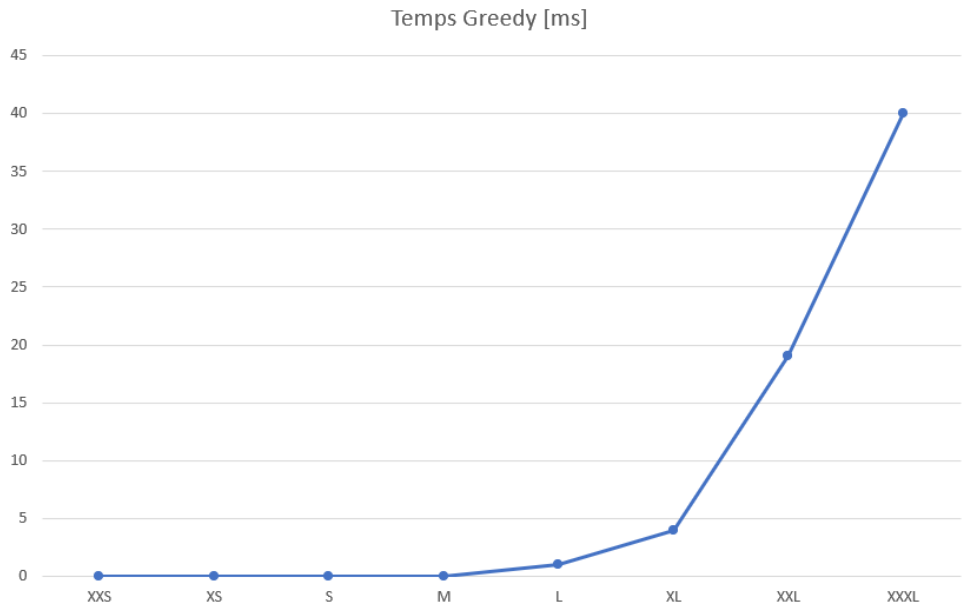

Figura 3.2: Comparativa de temps d'execució del Greedy

3.3. Branch and Bound

El Branch & Bounds aparentment té un rendiment superior al Backtracking. No obstant això, en la il·lustració no s'ha empleat el timeout (per tant s'han calculat totes les opcions no podades, com faria el Backtracking). Si utilitzéssim el timeout el B&B faria els 10 trams en 5 segons sense cap problema.

Trams	Temps [ms]
2	1
3	0
4	1
5	2
6	10
7	67
8	553
9	5574
10	64552

Taula 3.3: Comparativa de temps d'execució del B&B

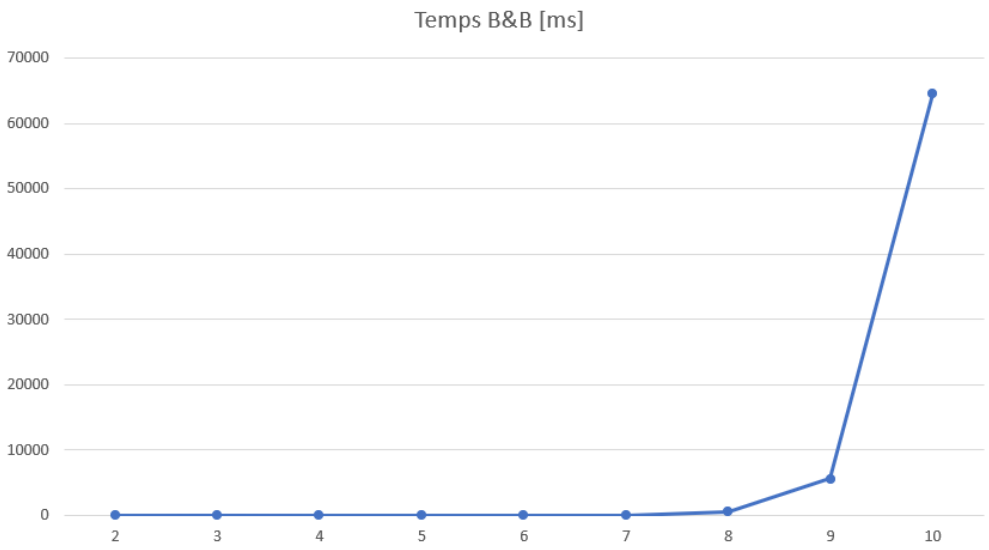

Figura 3.3: Comparativa de temps d'execució del B&B

4. Resultats

Com podem veure a les gràfiques anteriors trobem que el algorisme que ha tingut més dificultats (de fet, li ha sigut impossible) alhora de trobar una solució per els diferents datasets ha estat el Backtracking. Ja que com el seu cost és exponencial tal com augmentava el nombre de atletes es tornava molt complicat de córrer amb tantes possibilitats. Amb aquesta limitació tot i deixar-lo moltes hores i veure que no trobàvem resultat hem optat per parar el procés.

En els algorismes de Branch & Bound i Greedy els temps de execució han estat molt més raonables, sent el Greedy el que té un rendiment molt bo (cal considerar però, que és el problema més senzill dels tres). Trobem doncs que la solució del Branch and Bound es la millor solució per problemes més difícils i grans.

5. Mètode de proves

Per provar els algoritmes hem utilitzat els diferents datasets que ens han donat. Un cop posats, miràvem com evolucionaven amb el debugger i també ens fixàvem en el resultat que retornaven.

Amb el Backtracking però no podíem utilitzar el dataset XS, ja que només podia formar un equip (i, per tant, no es podia realitzar cap tipus de combinació). Com el fitxer S era massa gran per executar-lo, n'hem creat un més. Aquest és el mateix que el XS, però canviant alguns tipus de corredors per tal de que es puguin formar 4 equips ja que sinó el resultat hagués estat erroni per els nostres tests de proves.

6. Problemes Observats

En el nostre cas hem trobat principalment els problemes amb el Backtracking ja que no aconseguíem fer un algorisme recursiu que ens fes les combinacions que esperàvem. Vam provar diferents mètodes per intentar arreglar-ho sense èxit. Al final vam optar per fer la representació binaria descrita als Algorismes per aconseguir les combinacions. Una vegada vam seguir aquesta idea ens va facilitar molt la implementació.

En quant al algorisme Branch and Bound una de les coses que ens ha suposat dubtes ha estat quin valor serà el valor més òptim per obtenir bona solució contra el temps de execució. Buscant una mica per internet hem trobat mètodes per valorar-ho però eren extremadament complicats alguns d'ells i requerien un anàlisis molt profund del dataset.

7. Conclusions

Degut a la complexitat del problema implementat amb Backtracking ens ha resultat complicada aquella part. Al enfocar-ho en binari hem provat una forma diferent a la que estem acostumats.

Tot i que ens hagués agradat que el Backtracking acabes amb el dataset S, ni amb marcatge ha baixat de les 9 hores. Aquests fets ens han ensenyat que mentre que pensàvem que tot i que portaria temps seria possible trobar els resultats per a datasets grans ens hem donat compte que ens costa molt imaginar-nos a simple vista la complexitat dels algorismes i el temps que costen executar-se.

Respecte el Branch & Bounds hem estat testejant diferents timeouts i com afectaven tot i buscant com varia el cost temporal de la execució depenent del timeout. Al posar un timeout molt gran el resultat temporal és pràcticament igual que el Backtracking, ja que el B&B recorre tot l'espai igualment. Al afegir el timeout s'aprofita la propietat de l'algoritme de trobar una solució molt òptima ràpidament, i es dona aquesta com a vàlida. Si el timeout és molt petit ens arrisquem a que la solució no sigui gaire optima.

El Greedy ha estat el que ens ha semblat mes senzill ja que ja l'havíem treballat amb anterioritat en problemes molt semblat en competicions de programació com va ser la de Algorísmia de La Salle.

El cost ha estat extremadament petit, inclús en els datasets més grans.

8. Bibliografia

Wikipedia contributors. (2021, January 23). Branch and bound. In *Wikipedia, The Free Encyclopedia*. Retrieved 14:29, March 12, 2021, from https://en.wikipedia.org/w/index.php?title=Branch_and_bound&oldid=1002333338

Wikipedia contributors. (2021, February 2). Backtracking. In *Wikipedia, The Free Encyclopedia*. Retrieved 14:29, March 12, 2021, from https://en.wikipedia.org/w/index.php?title=Backtracking&oldid=1004367433

Wikipedia contributors. (2021, February 8). Greedy algorithm. In *Wikipedia, The Free Encyclopedia*. Retrieved 14:29, March 12, 2021, from https://en.wikipedia.org/w/index.php?title=Greedy_algorithm&oldid=1005603733