Universitat de Lleida

Pràctica 2: Aquæductus Optimus

Assignatura d'Algorítmica i complexitat del Grau d'Enginyeria Informàtica

Pablo Fraile Alonso

20 de juny de 2021

Índex

1 Greedy **4**
1.1 Cost algorisme 4
1.1.1 Iteratiu 4
1.1.2 Recursiu 5
1.1.3 Empíric 5

2 Backtracking **6**
2.1 Cost algorisme 6
2.1.1 Recursiu 6
2.1.2 Iteratiu 7
2.1.3 Empíric 8

3 Dynamic Programming **9**
3.1 Aplicació i funcionament en el nostre cas d'ús 9
3.2 Demostració per reducció al absurd 14
3.3 Especificació formal 15
3.4 Fòrmula algorisme 15
3.5 Cost algorisme 16
3.5.1 Iteratiu 16
3.5.2 Recursiu 16
3.5.3 Empíric 16
3.6 Pseudocodi algorisme iteratiu 17
3.7 Pseudocodi algorisme recursiu 18

4 Problemes al realitzar la pràctica **18**
4.1 Nombres en C++ 18

5 Consideracions **19**

6 Conclusions **19**

Apèndix A: Repositori github **21**

Índex de figures

1 Exemple algorisme Greedy 4

2	Cost empíric emprant Greedy en Python	5
3	Exemple crides recursives Backtracking	6
4	Nombre de nodes a les crides recursives de backtracking	7
5	Nombre d'iteracions de la funció is_valid a backtracking	8
6	Cost empíric emprant Backtracking en Python	8
7	Entrada exemple	9
8	Exemple representat eix de coordenades	10
9	Exemple representat en forma de dígraf	11
10	resultat de $f(E)$	11
11	resultat de $f(D)$	12
12	resultat de $f(C)$	12
13	resultat de $f(B)$	13
14	resultat del aqüeducte mínim ($f(A)$)	13
15	Aqüeducte de punt A a punt J	14
16	Aqüeducte de punt A a punt J passant per K	14
17	Graf que representa un possible $R'_{a...k}$	14
18	Cost empíric emprant Dynamic Programming en Python	16
19	Cost empíric emprant Dynamic Programming en C^{++}	17

1 Greedy

Un algorisme greedy (o algorisme voraç), segueix l'heurística de resolució de problemes de fer l'elecció local òptima en cada etapa. En aquest cas d'ús, veurem que aquesta no és una solució (real), ja que potser que les solucions locals òptimes, a la llarga no façin el aqueducte amb el cost mínim.

Un exemple per anar d'un punt a a un punt d seguint l'algorisme greedy, podría ser el de la figura 1.

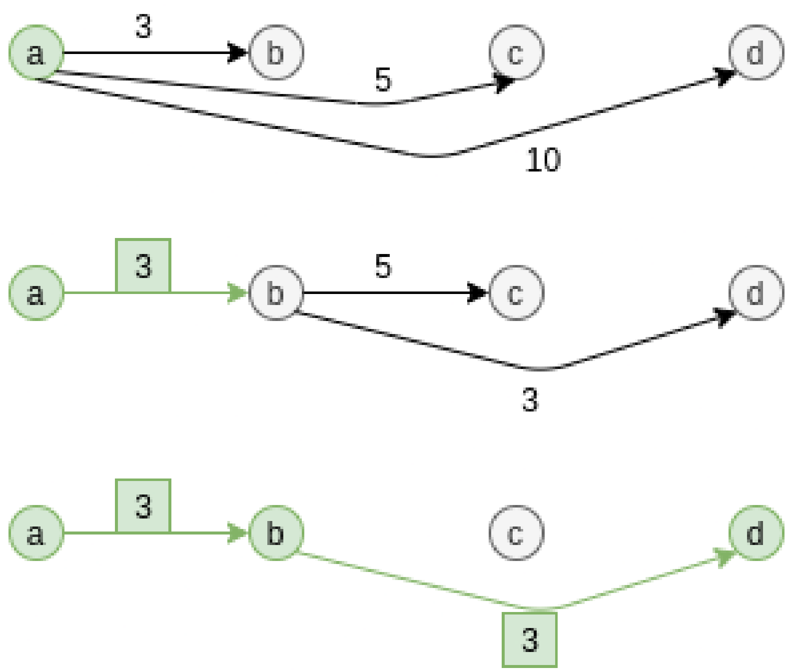

Figura 1: Exemple algorisme Greedy

1.1 Cost algorisme

1.1.1 Iteratiu

Veiem que el programa té 3 bucles anidats on cadascun té un pitjor cas amb cost de $O(n)$, i que en el pitjor dels casos haurem recorregut n^3 vegades. Per tant, podem dir que el cost de l'algorisme en forma iterativa és de $O(n^3)$.

1.1.2 Recursiu

Òbviament, si l'únic que hem fet ha sigut passar l'algorisme iteratiu a recursiu (o a l'inversa), el cost d'aquest continuarà sent el mateix. L'únic que canviarà serà que ara s'utilitzarà memòria de la pila d'execució i no memòria de heap, per tant el programa es més propens a sofrir un *stackoverflow*. Finalment podem dir que el cost és de $O(n^3)$.

1.1.3 Empíric

Un cop executat varies vegades l'algorisme amb un conjunt de n diferents, ens crea la gràfica de la figura 2. Aquesta, concorda amb la justificació de la notació assimptòtica donada anteriorment.

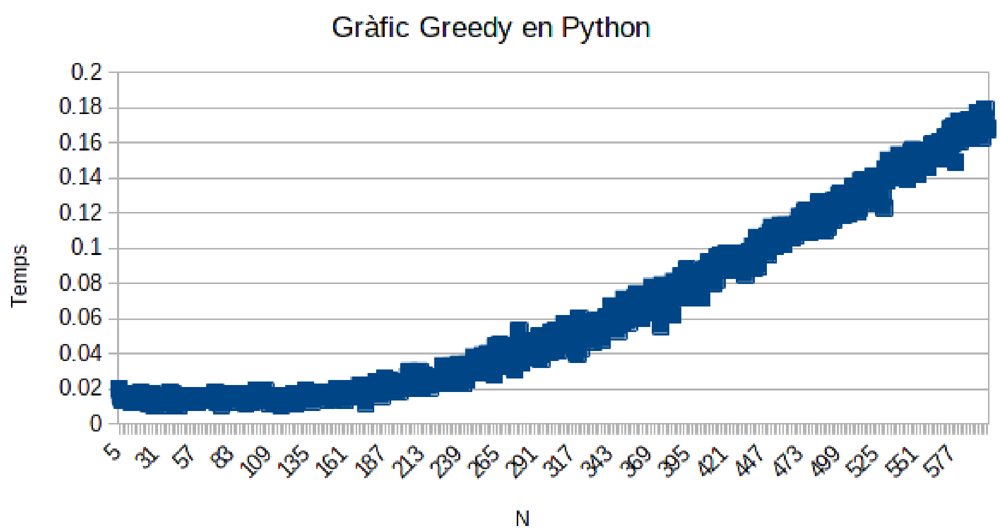

Figura 2: Cost empíric emprant Greedy en Python

2 Backtracking

La técnica de Bactracking consisteix en anar provant totes les diferents solucions i quedar-se amb la més òptima. Una resolució mitjançant backtracking per anar del punt A al punt D on es pot crear arcs entre tots els punts sería la figura 3.

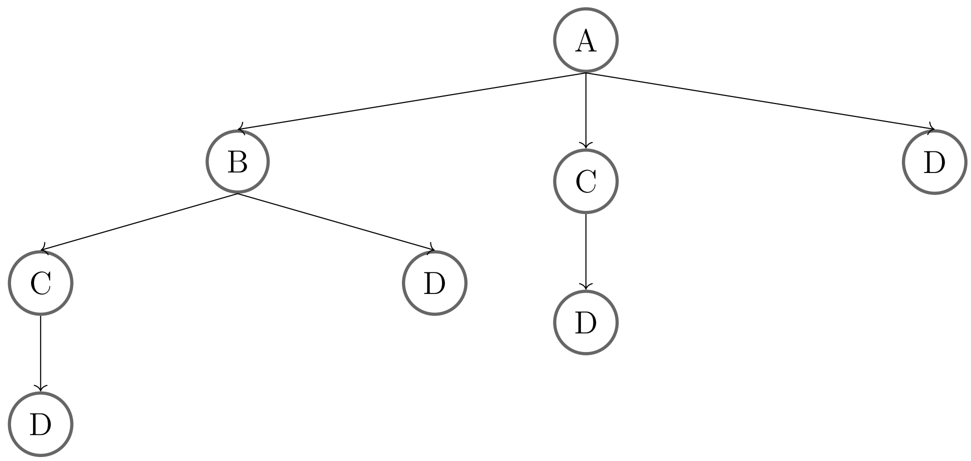

Figura 3: Exemple crides recursives Backtracking

2.1 Cost algorisme

2.1.1 Recursiu

Primerament es va tindre en compte quin sería el cost de les crides recursives que, tal i com s'aprecia a la figura 4, veiem que creen una forma d'arbre que haurem de recòrrer de forma postordre (finalitzem primer subarbre esquerra, després dret i finalment l'arrel).

Finalment, veiem que el nombre de nodes a recòrrer en funció de n (on n és el nombre de pilars) serà 2^{n-1}.

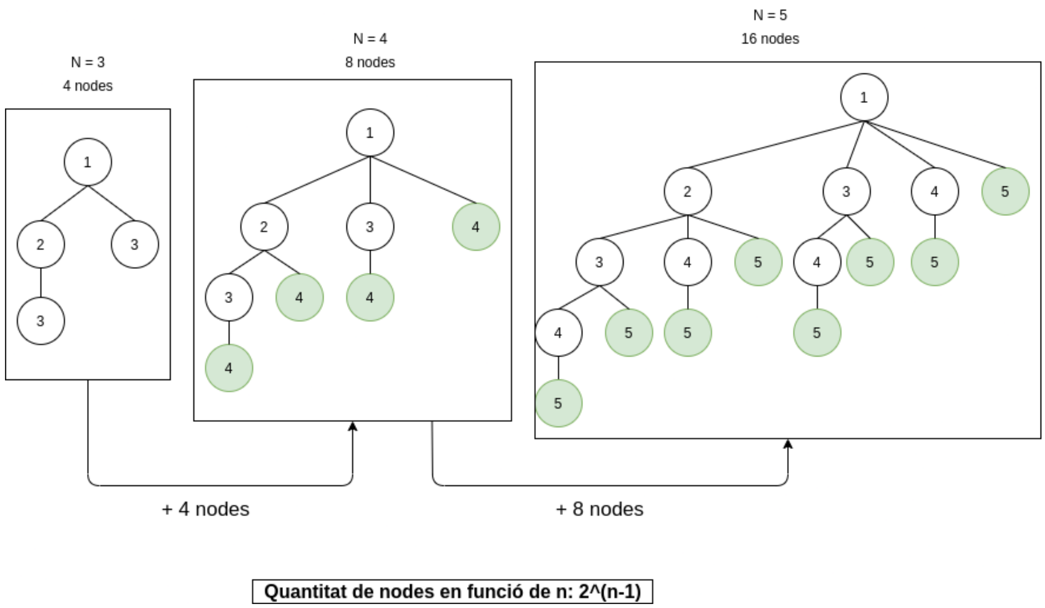

Figura 4: Nombre de nodes a les crides recursives de backtracking

A més, falta calcular el cost de la funció *is_valid* en l'arbre recursiu (figura 5). Aquest s'ha trobat calculant quantes vegades iterava el bucle en funció de n, i s'ha vist que es creava la seqüència 4, 11, 26, 57, 120, etc. S'ha deduït i comprovat que l'equació de recurrència en funció de n és $2^{n-1} - n$.

Finalment, un cop ja calculades les dos equacions de recurrència, podem sumar-les per obtenir el cost del algorisme, el que ens donaría:

$$2^{n-1} + 2^{n-1} - n \Longrightarrow 2 * 2^{n-1} - n$$

Que en notació assimptòtica, podem representar com: $O(2^n)$

2.1.2 Iteratiu

La forma més senzilla de pasar un algorisme que utilitza backtracking a iteratiu és mitjançant l'ús d'una pila[1]. El cost algorímic serà igualment $O(2^n)$, però ara s'utilitzarà memòria de heap i no de pila d'execució, per tant **NO** es probable que sofrim un *stackoverflow*.

[1] "Imitant" en certa forma el que fa la pila d'execució

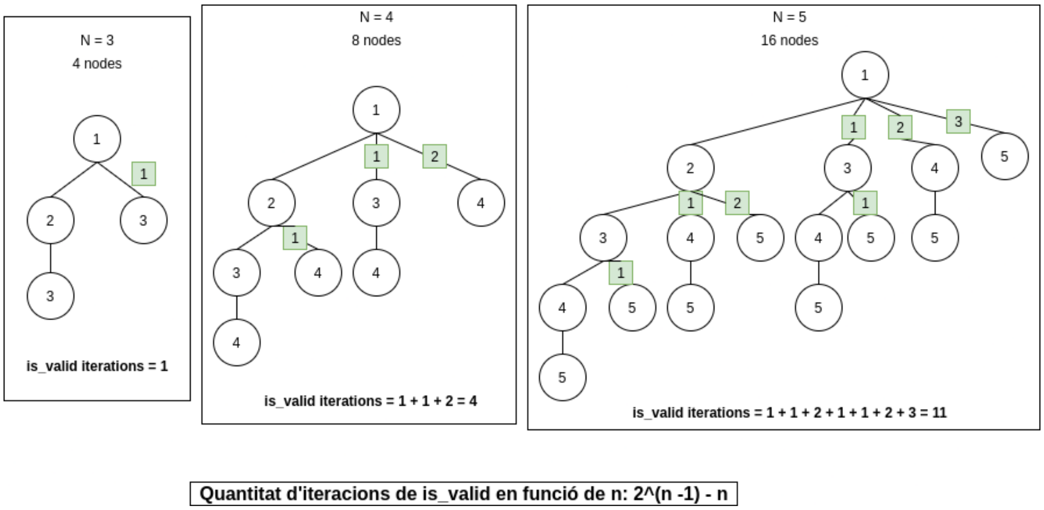

Figura 5: Nombre d'iteracions de la funció is_valid a backtracking

2.1.3 Empíric

Finalment, s'ha executat múltiples vegades el algorisme i s'ha aconseguit formar la gràfica de la figura 6, que concorda amb la nostra afirmació de que el cost és de $O(2^n)$.

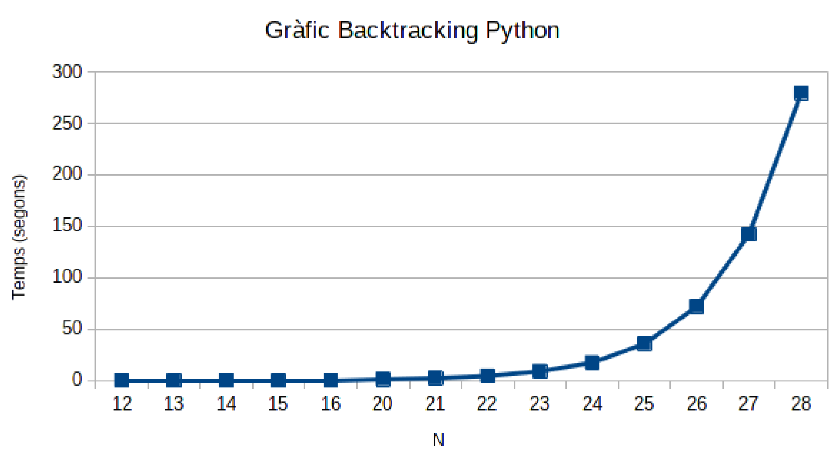

Figura 6: Cost empíric emprant Backtracking en Python

3 Dynamic Programming

Abans de poder comentar la solució, hem d'entendre que és el principi d'optimitat:

> Principi d'optimitat: Una política òptima té la propietat que sigui quin sigui l'estat inicial i la decisió inicial, les decisions restants han de construir una política òptima respecte a l'estat resultat de la primera decisió.
>
> (Richard E.Bellman)

Per tant, seguint aquesta definició podem dir que un problema podrà ser resolt seguint el principi d'optimitat si la seva solució òptima pot ser construïda eficientment a partir de les solucions òptimes dels seus subproblemes. En altres paraules, que podem resoldre un problema gran donades les solucions dels seus problemes petits.

3.1 Aplicació i funcionament en el nostre cas d'ús

En el nostre problema dels aqüeductes, veiem que podem aplicar el principi d'optimitat per a trobar una solució òptima, ja que la solució és construïda eficientment a partir de les solucions òptimes dels seus subproblemes. Per a explicar-ho millor he decidit resoldre un petit exemple.

Donada la següent entrada:

```
5 6 180 20
0 0
2 2
3 1
5 3
7 2
```

Figura 7: Entrada exemple

On podem veure que tenim 5 punts, una altura d'aqüeducte de 6, $\alpha = 180$ i $\beta = 20$. Si ho representem en un eix de coordenades, el perfil del sòl ens queda com la figura 8, en canvi, si ho volem examinar en forma de dígraf (ja descartant opcions que no són vàlides) ens queda com a resultat la figura 9

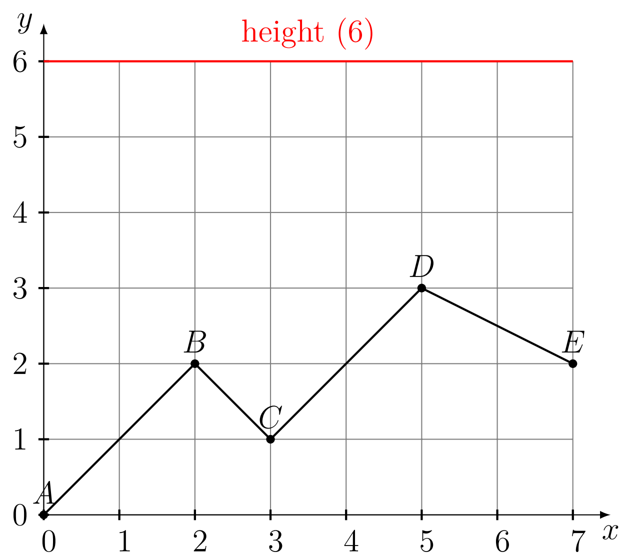

Figura 8: Exemple representat eix de coordenades

A continuació, anomenarem la funció $f(x)$ com el mínim cost per anar al node E. En el cas d'estar al propi node E, aquesta funció retornarà 0 (figura 10).

En el cas de $f(D)$, únicament té una opció possible, anar del node D a F, per tant el cost mínim serà el recorregut mostrat a la figura 11 i la funció retornarà el valor del cost de crear un pilar a D, més el cost de crear un pilar a F i el cost de crear el arc de D a F.

En el cas de $f(C)$, té l'opció d'anar a D o d'anar a E. En aquest cas calcularem el cost de C a E i el cost de C a D + $f(D)$ i agafarem el mínim. Calculem cost de C a E i ens dona 1940, en canvi, el cost de C a D + $f(D)$ ens dona 2320. Per tant, el cost mínim des de C serà anant de C a E (figura 12).

En el cas de $f(B)$, farem el mateix que amb $f(C)$. Calcularem quant val

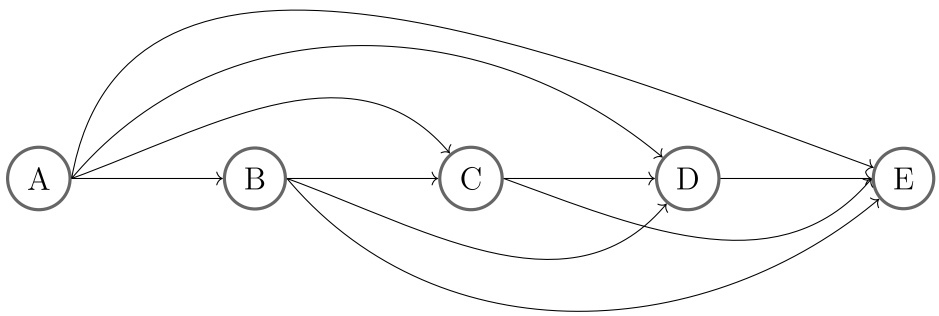

Figura 9: Exemple representat en forma de dígraf

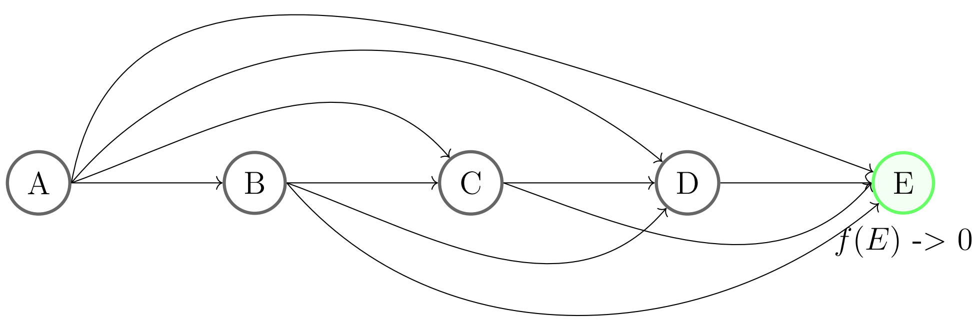

Figura 10: resultat de $f(E)$

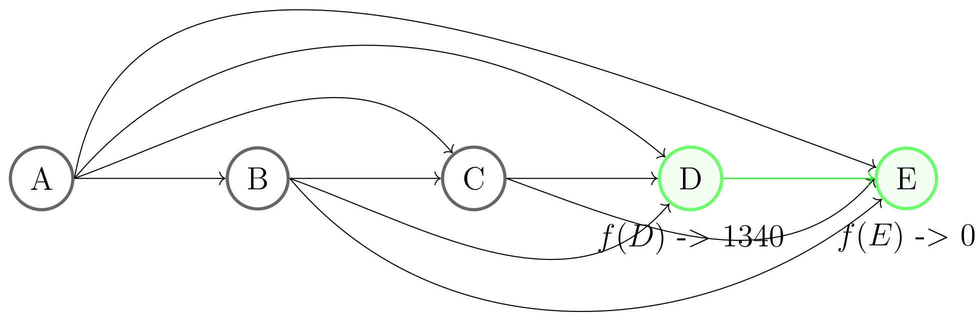

Figura 11: resultat de $f(D)$

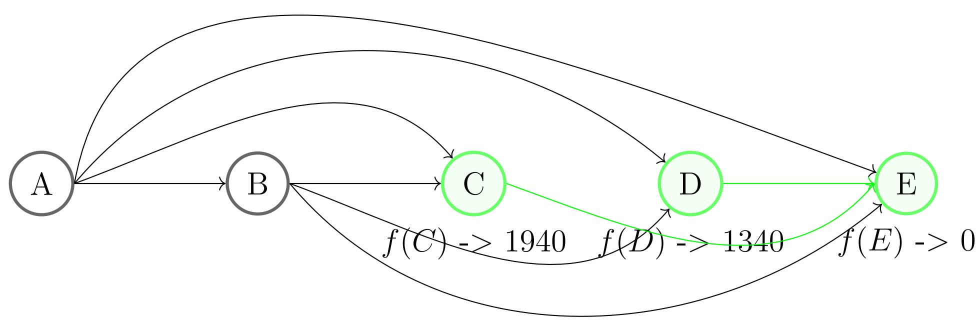

Figura 12: resultat de $f(C)$

el cost de B a E, de B a C + $f(C)$ i de B a D + $f(D)$ i agafarem el mínim. En aquest cas el mínim es de B a E (1940) (figura 13).

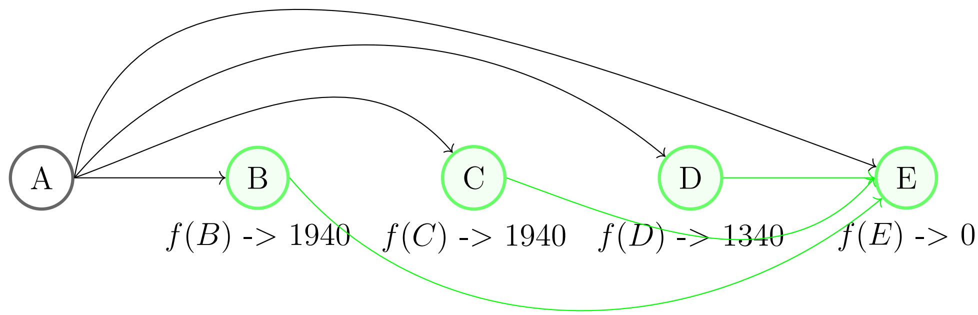

Figura 13: resultat de $f(B)$

Finalment, en el cas de $f(A)$, haurem de calcular el cost de A a E, de A a B + $f(B)$, de A a C + $f(C)$ i de A a D + $f(D)$ i agafar el mínim cost. En aquest cas el mínim es de A a E (2780) (figura 14)

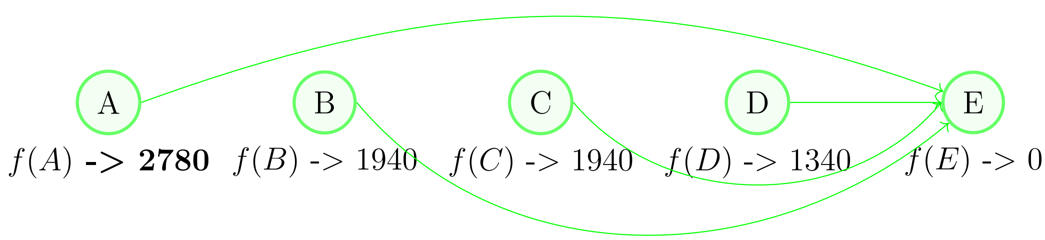

Figura 14: resultat del aqüeducte mínim ($f(A)$)

3.2 Demostració per reducció al absurd

Donat un aqüeducte que va d'un punt A a un punt J i del qual sabem que el recorregut $R_{a...j}$ és l'òptim (figura: 15).

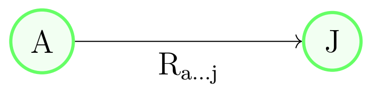

Figura 15: Aqüeducte de punt A a punt J

Assumirem també que aquest recorregut passa per el punt K, per tant ara podem separar el recorregut com $R_{a...k}$ & $R_{k...j}$ (figura 16)

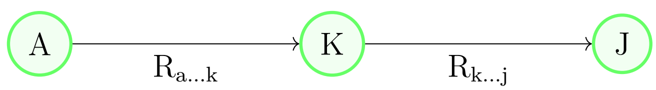

Figura 16: Aqüeducte de punt A a punt J passant per K

Ara donarem com a hipòtesis que del punt A al punt K pot haver-hi un recorregut mes òptim, que anomenarem $R'_{a...k}$ (figura 17).

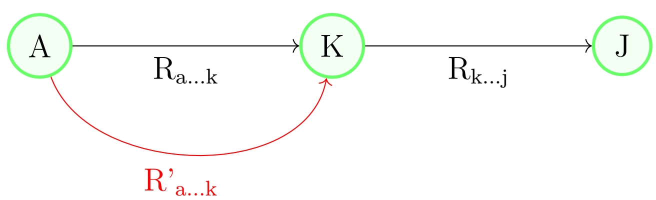

Figura 17: Graf que representa un possible $R'_{a...k}$

Si $R'_{a...k}$ és més òptim que $R_{a...k}$, llavors vol dir que:

$$R'_{a...k} < R_{a...k}.$$

Llavors:

$$\mathbf{R'_{a...k} + R_{k...j} < R_{a...k} + R_{k...j}}$$

Però aquesta afirmació **NO** pot ser certa! Ja que en un principi hem assegurat que $\mathrm{R_{a...k}} + \mathrm{R_{k...j}}$ era la solució òptima i per tant no hi pot haver-hi cap més petita que aquesta.

3.3 Especificació formal

Precondició :

- Una altura màxima del aqueducte (h), on $(1 \leq h \leq 10^5)$
- Els factors de cost α i β, on $(1 \leq \alpha, \beta \leq 10^4)$
- Un conjunt de punts (x, y):
  $P = \{p1, p2, ...p_{\mathrm{n}} \mid p_{\mathrm{n}} = (x, y)\} \wedge \forall_{\mathrm{x,\,y}} \in \mathbb{N} \wedge n \ni \mathbb{N} \wedge n >= 2 \wedge x_{\mathrm{i}}(0 \leq x_1 < x_2 < ... < x_{\mathrm{n}} \leq 10^5) \wedge y(0 \leq y_{\mathrm{i}} < h)$

Postcondició: cost mínim de crear l'aqueducte mitjançant els valors α i β o ∞ en cas de que no sigui possible [2]:

$resultat \in \mathbb{N} \wedge resultat \leq \min(R)$, on R és el conjunt de possibles resultats.

3.4 Fòrmula algorisme

Un cop ja sabem el funcionament del algorisme i hem demostrat que el seu comportament es correcte, podem especificar-lo amb una formula molt similar a la de l'equació de Bellman (ja que tal i com s'ha dit a la subsecció 3.1 aquest problema es resolt seguint el principi d'optimitat).

$$v(x_0) = min(f(x_0) + v(x_1)) \tag{1}$$

On $v(x)$ és la fórmula per a calcular el cost mínim del aqüeducte, x_0 es el primer pilar del aqüeducte i x_1 es el resultat d'aplicar $v(x)$ al pilar que va desprès de x_0

[2] Ja que tal i com es comentava a classe, havía de retornar un tipus, en aquest cas un enter o infinit. Realment en cas del programa s'ha de treure un enter si és possible o una cadena de carácters dient que és impossible

3.5 Cost algorisme

3.5.1 Iteratiu

Veiem que el programa té 3 bucles anidats, i que en el pitjor dels casos (quan estem analitzant el primer punt, p_0) haurem recorregut els tres n^3 cops. Per tant, podem dir que el cost de l'algorisme en forma iterativa és de $O(n^3)$.

3.5.2 Recursiu

Òbviament, si l'únic que hem fet ha sigut passar l'algorisme iteratiu a recursiu, el cost d'aquest continuarà sent el mateix. L'únic que canviarà serà que ara s'utilitzarà memòria de la pila d'execució i no memòria de heap, per tant el programa es més propens a sofrir un *stackoverflow* (o *recursion error* en python). Finalment podem dir que el cost és de $O(n^3)$.

3.5.3 Empíric

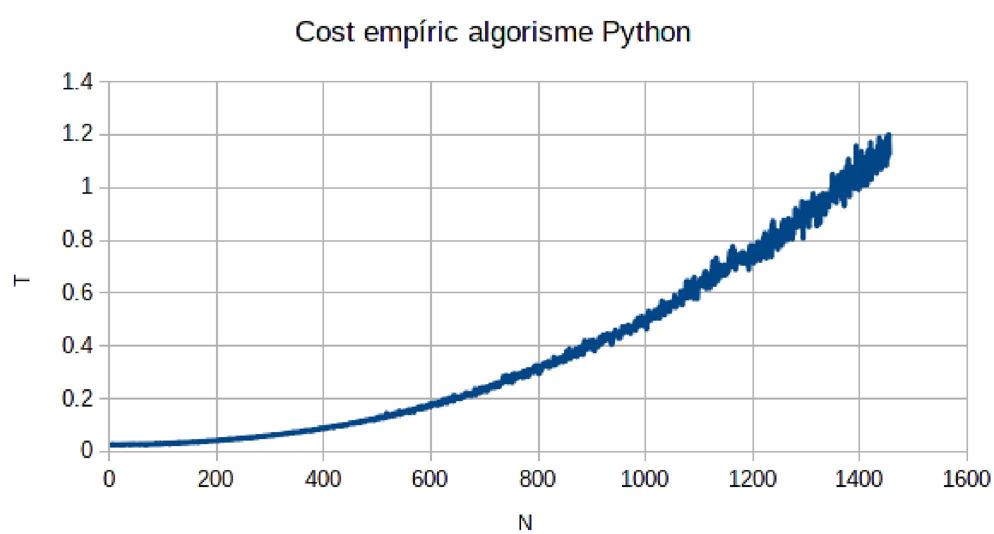

Figura 18: Cost empíric emprant Dynamic Programming en Python

Recalcar que en Python, es van fer els testos fins N=1450 (figura 18), ja que sino el temps d'execució ja era sumament lent, mentres que amb

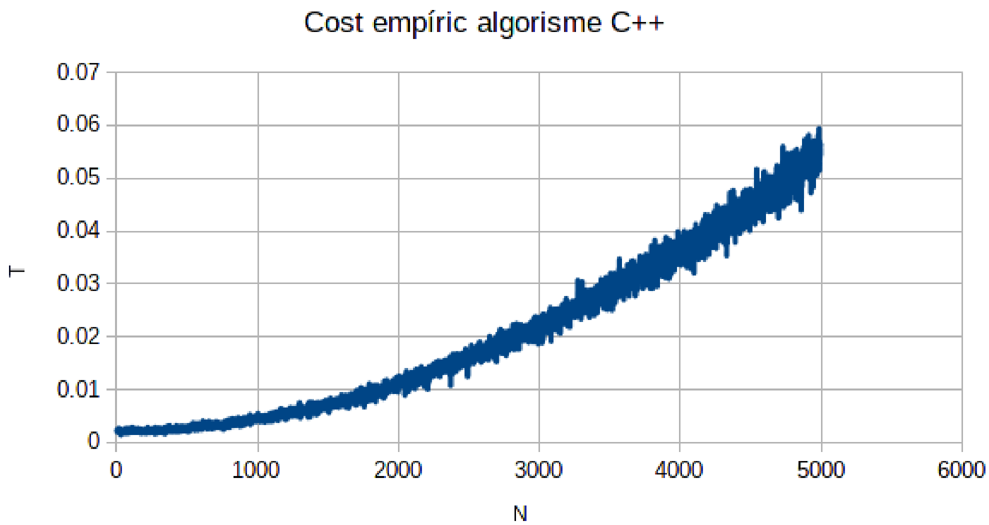

Figura 19: Cost empíric emprant Dynamic Programming en C++

C++(figura 19), si que es van poder fer execucions fins N=5000.

Igualment, es pot apreciar que les dos gràfiques surten similar a una funció $O(n^3)$.

3.6 Pseudocodi algorisme iteratiu

El codi iteratiu segueix l'algorisme explicat a la secció 3. Recalcar que la llista que guarda els resultats és utilitzada dintre de la funció *get_minimum_cost_for_index* per a no haver de tornar a calcular valors mínims de punts que ja havíem resolt anteriorment.

```
def get_minimum_aqueduct(self):
    for i in range(self.num_points - 2, -1, -1):
        minimum_of_this_point = self.get_minimum_cost_for_index(
                                        i)
        self.point_values_buffer[i] = minimum_of_this_point
    return self.point_values_buffer[0]
```

3.7 Pseudocodi algorisme recursiu

El codi recursiu és molt similar al iteratiu. De fet, l'únic que s'ha fet és pensar el cas base (quan el índex és 0) i a partir d'allí anar movent-se per tots els punts augmentant el sufix de l'array mentres que disminueix el prefix fins arribar al cas base (índex 0, on el sufix és tota l'array mentres que el prefix és inexistent).

Igualment, per a mantindre la concordança de codi amb la versió iterativa, s'ha decidit fer una funció que serveix de *wrapper* per a amagar que realment estem cridant a una solució recursiva.

```
def get_minimum_aqueduct_recursive(self, index: int):
    if index == 0:
        minimum_of_this_point = self.get_minimum_cost_for_index(
                                    index)
        return minimum_of_this_point
    self.point_values_buffer[index] = self.
                                get_minimum_cost_for_index(
                                index)
    return self.get_minimum_aqueduct_recursive(index - 1)

# wrapper for recursive function
def get_minimum_aqueduct(self):
    return self.get_minimum_aqueduct_recursive(self.num_points -
                                2)
```

4 Problemes al realitzar la pràctica

4.1 Nombres en C++

Primerament es va pensar i elaborar l'algorisme en el llenguatge de programació python, i a continuació es va migrar el codi a C++. El problema va estar en no es va pensar que python al tindre tipus dinàmics el mateix intèrpret assigna un tipus a cada variable de forma intel·ligent, mentres que a C++el propi programador és el que assigna els tipus. Això va generar un problema ja que, com els tests eren summament grans i podien arribar a fer operacions com per exemple 10000^2, feia que no fos suficient amb els tipus integer, i s'hagués d'utilitzar tipus com per exemple "long long int" o "unsigned long long int".

5 Consideracions

1. Al arxiu Makefile del projecte de python, s'ha afegit una opció per a fer test del codi mitjançant l'eina pylint, però habilitant l'opció d'ignorar els "warnings" provocats per no tindre comentaris per cadascún dels métodes i clases.

2. S'han mogut tots els tests a un directori anomenat *test*, ja que com s'ha fet la pràctica en python y C++si no es separava s'havia de tindre els mateixos tests repetits en dos carpetes diferents.

3. S'ha decidit que el binari resultant del codi escrit en C++porti habilitades les opcions d'optimització O3, ja que sinò el rendiment baixava considerablement.

4. El codi no inclou anàlisi d'errors sobre el fitxer que se li proporciona. En cas de que el format/fitxer sigui erroni, es veurà l'excepció corresponent de python o C++.

5. Tot i que s'ha augmentat la pila d'execució en el codi recursiu de python, recordem que la pila no és infinita i per tant en cas d'una N molt gran el programa llançarà un error de recursivitat.

6. Al programari de python s'ha emprat el patró de disseny *template*, per tal de que el codi de les classes *Point*, *Circumference* i el *parser* del fitxer no estigués repetit en 6 fitxers diferents. El fet de que python s'hagi de situar a un altre directori (anomenat *common*) per poder importar els fitxers ha ocasionat afegir linees de codi que amb ulls de l'eïna pylint eren poc correctes. Aquestes s'han evitat afegint els comentaris:

   ```
   # pylint: disable=wrong-import-position
   # pylint: disable=import-error
   ```

   per tal de poder obtenir bona puntuació a l'evaluació.

6 Conclusions

Aquesta pràctica m'ha servit per adquirir els següents coneixements:

- Comprendre i implementar un algorisme mitjançant backtracking.
- Comprendre i implementar un algorisme mitjançant greedy.
- Aprendre les bases de la programació funcional i veure quina millora afegíen en aquest algorisme.
- Iniciarme en la programació en C++.
- Analitzar els diferents costs (empírics i teòrics) d'un algorisme.
- Convertir un algorisme iteratiu a recursiu.
- Crear un informe a partir de LaTeX per a poder afegir fórmules lògiques i matemàtiques.

Apèndix A: Repositori github

El repositori de github es pot trobar aquí. Recalcar que llegir el README adjuntat amb el repositori és indispensable per a saber com executar els tests de forma correcta i donarse una idea de com es troba estructurat el projecte.