



ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA

Pràctica 1 Algorítmica i complexitat

Noms i cognoms: Aarón Arenas Tomás i Marc Cervera Rosell

DNI's: 78098697N i 47980320C

Curs: 2020 – 2021

Contingut

Especificació formal:	2
Precondició:	2
Postcondició:	2
Desenvolupament del codi:	2
read_file.py:	2
main.py:	3
mainRecursive.py:	3
calculs.py:	3
calculsRecursive.py:	4
Costos:	5
Iteratiu:	5
Recursiu:	6
Requeriments:	7
Link GitHub:	7

Especificació formal:

Precondició:

n (nombre de punts del terreny), h (alçada desitjada de l'aqüeducte), alfa (factor de cost), beta (factor de cost) => nombres enters (primera línia del fitxer d'entrada)

n línies amb dos nombres cadascuna que representen les coordenades de cada punt del sòl. => (x, y)
=> coordenades cartesianes => nombres enters

Postcondició:

Donades totes les coordenades del terreny, es calcula el cost de fabricar un pont i un aqüeducte (per separat) i finalment es fabrica el que sigui més eficient => En cas que es puguin construir totes les columnes sense superar la altura límit.

En cas que hi hagi una columna que no es pugui construir, el programa traurà per pantalla que no és possible construir un aqüeducte. Encara que l'aqüeducte no sigui possible, el pont encara podria ésser possible de construir.

Desenvolupament del codi:

Primerament, cal remarcar que per evitar tenir tot el codi de la pràctica mesclat en un sol fitxer, s'han realitzat diferents scripts per a les diferents parts del desenvolupament.

read_file.py:

Primera ment s'ha implementat un script anomenat 'read_file.py', el qual llegeix el fitxer línia a línia.

Primerament, es passa cada línia del fitxer per la funció *strip()* per eliminar possibles espais en blanc.

En segon lloc, s'introdueix la totalitat del fitxer a l'interior d'una llista, per així facilitar la feina posterior.

Seguidament, amb l'ajuda de la funció *map()* s'obtenen les variables pertanyents a les dades del problema (variables n, h, aplha, beta). Per obtenir cada valor per separat, s'empra la següent línia de codi:

```
n, h, alpha, beta = map(int, filtered_reader[0].split(data_separation))
```

La funció *map*, en aquest cas, converteix a enter la posició 0 del *filtered_reader*, posició en la qual es troben les variables en qüestió. Gràcies a la funció *split()*, s'obté cada valor per separat. *data_separation* és un paràmetre de la funció de lectura del fitxer que indica el criteri de separació a aplicar en el moment de la crida a la funció *split()*.

Abans de retornar res, amb la mateixa estratègia, s'introdueix en una llista anomenada *values* les tuples (x, y) les quals representen les coordenades del terreny.

Finalment, es retorna la llista *values*, n, h, aplha i beta.

main.py:

Aquest fitxer, contindrà el programa principal del mètode iteratiu.

En primer lloc, mitjançant un bucle `*for*` que va des d'1 fins a 21 s'aniran generant nombres. Si el nombre generat és menor a 10, aquest nombre es concatenarà darrere d'un 0 i posteriorment s'afegirà a un *String*, anomenat *fitx* que contindrà el nom complet de l'arxiu de test.

Seguidament, s'obtiniran les dades del problema mitjançant la funció que llegeix el fitxer per a posteriorment realitzar els càlculs oportuns.

A continuació, és traurà per pantalla el nom del fitxer i el resultat. Per deixar constància, s'obrirà el fitxer *output.ans* per escriure-hi el resultat i posteriorment es tancarà.

Finalment, guardem el contingut del fitxer de test i comprovem el resultat amb el nostre fitxer (el fitxer que ha generat el nostre programa). Si el contingut d'ambdós fitxers és igual, el test és correcte. Aquesta acció és realitzada per cadascun dels tests.

mainRecursive.py:

L'estratègia seguida és la mateixa que en l'script *main.py* de la versió iterativa, però fent la crida a la versió recursiva dels càlculs.

calculs.py:

En aquest script, és on es realitzen els càlculs en format iteratiu.

En obrir el fitxer de Python, el primer mètode que s'observa és un mètode anomenat *calc_impossiblepont*, el qual retorna veritat o fals segons si l'aqüeducte es possible o no.

Aquest mètode utilitza la següent línia de codi per calcular l'alçada del pont:

```
height = math.sqrt((r ** 2 - ((disX - posantX - r) ** 2))) + (h - r)
```

on:

- *r* = radi
- *disX* = distància entre pilars
- *posantX* = primer pilar
- *h* = alçada total del pont

Un cop calculada l'alçada del pont, es retorna cert o fals en funció de si aquesta altura, recentment calculada, és major a la coordenada 'y' del terreny.

El següent és el mètode *calc_impossible*, el qual calcula la possibilitat de crear o no l'aqüeducte. Per calcular aquesta possibilitat, en primer lloc es calcula el radi de la semicircunferència de sota de l'aqüeducte com "*d* / 2" on '*d*' és la distància entre pilars. Seguidament es calcula el punt màxim on pot arribar la coordenada y del terreny. Aquest càlcul es realitza com "*h* - *r*", on '*h*' és l'alçada màxima del pont i '*r*' és el radi. Es retornarà cert o fals en funció de si aquest últim càlcul és major a la coordenada 'y' del terreny o no.

A continuació, s'observa el mètode *obtainValues*, el qual introdueix en llistes les coordenades 'x' del terreny, les coordenades 'y' del terreny i les distàncies entre pilars.

Les distàncies entre pilars es calculen com la coordenada 'x' del pilar de la dreta menys la coordenada 'x' del pilar de l'esquerra. Finalment, es retornen les tres llistes de valors.

El següent, és un mètode que senzillament retorna les coordenades del terreny donada la seva tupla.

A continuació, s'ha implementat un mètode de càlcul del cost del aqüeducte. Aquest mètode calcula la funció:

$$\alpha \sum_{i=1}^k h_i + \beta \sum_{i=1}^{k-1} d_i^2$$

Durant la realització del càlcul, es van fent crides al mètode `calc_impossible` (descriu anteriorment) i si en algun moment aquest mètode retorna fals, es talla l'execució del bucle. Finalment es retorna el cost i el booleà que indica la possibilitat de la construcció o no de l'aqüeducte.

Com a última funció de càlculs hi ha `costPont`, la qual calcula el cost de la construcció d'un pont. Per realitzar el càlcul, s'empra la fórmula vista en la descripció del mètode anterior però ara amb el pont.

`cost = ((alpha * costsAltPont) + (beta * (dPont ** 2)))`

on:

- `costsAltPont = (h - alt[0]) + (h - alt[n - 1])`
- `dPont = disX[n - 1] - dis[0]`
- la llista 'alt' son les coordenades 'y' del terreny.

Si en algún moment la crida a `calc_impossiblepont`, descriu anteriorment, retorna fals, es talla l'execució del bucle.

Finalment, es retorna el cost i la variable booleana que indica la possibilitat, o no, de construir el pont.

Finalment, en aquest script es troba la funció *calculate*, la qual, primerament, obté els valors de les coordenades 'x' i 'y' (per separat) i després les distàncies entre pilars.

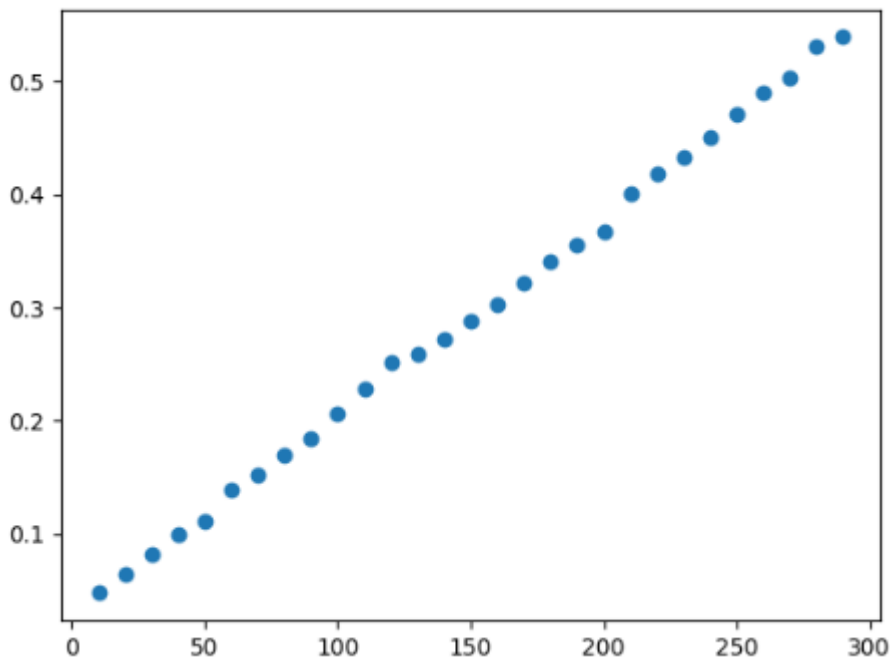
Seguidament, es comprova per `n == 2` si es possible la construcció de l'aqüeducte i en cas que `n != 2` s'estableixen els casos del *else*. En primer lloc es calcula tant la possibilitat de construcció com el cost de construcció tant del pont com de l'aqüeducte. Finalment, es comprova segons les impossibilitats i si un cost es major a l'altre quin és el cost que s'ha de retornar.

calculsRecursive.py:

En aquest script, s'han transformat a recursiu únicament les funcions de càlcul de costs. És a dir, s'han transformat les funcions `costPont()` i `costAque()`.

Costos:

Iteratiu:



Com es pot observar, el programa és completament lineal. Per fer càlculs solament s'utilitzen 3 bucles, un dels quals es solament per obtenir valors, no per fer càlculs. Un cop obtinguts aquests valors, es comprova el cost del pont i si es possible la seva construcció en el mateix bucle. Es realitza el mateix per a l'aqüeducte.

En cas que $n == 2$, és a dir, 2 columnes, el programa solament farà la comprovació de l'aqüeducte ja que no farà falta comprovar la del pont ja que se sap que no hi ha coordenades entre mig.

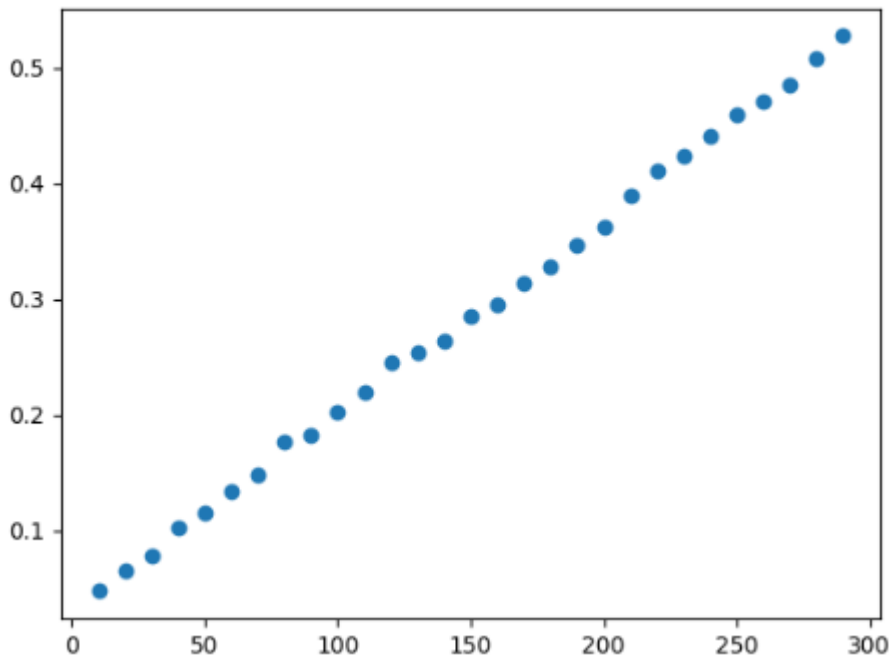
$$3 * (n) \Rightarrow O(n)$$

La notació dona lineal.

$O(\min)$ → Seria el cas en el qual la primera columna impossibiliti la construcció del pont/aqüeducte.

$O(\max)$ → Seria el cas en el qual el pont sigui impossible o l'última columna comprovada sigui la que impossibiliti la construcció del pont/aqüeducte.

Recursiu:



El cost experimental del programa és igual al teòric amb algun pic que es deu, sobre tot, a la càrrega de la CPU.

Com s'observa a la gràfica, el cost del programa és lineal atès que per realitzar els càlculs només s'utilitza 1 bucle per obtenir valors. Un cop obtinguts aquests valors, es comprova el cost del pont i la possibilitat de la seva construcció en el mateix bucle. Es realitza la mateixa operació per a l'aqüeducte. En el cas que hi hagi 2 columnes, el programa només farà la comprovació de l'aqüeducte. No farà falta comprovar el pont atès que sabem que no hi ha coordenades entre mig.

Al transformar a recursiu el mètode *costPont()* s'ha modificat la part per calcular el pont per tal de reduir el cost experimental ja que el cost és estàtic i no interessa que cada vegada que es faci la crida recursiva es torni a calcular el mateix cost.

$$1 * (n) \Rightarrow O(n)$$

La notació dona lineal.

$O(\min)$ → Seria el cas en el qual la primera columna impossibiliti la construcció del pont/aqüeducte.

$O(\max)$ → Seria el cas en el qual el pont sigui possible o l'última columna comprovada sigui la que impossibiliti la construcció del pont/aqüeducte.

NOTA: Per fer la comprovació dels costos, no s'ha contemplat la lectura del fitxer ni la obtenció de dades. Encara i així, el fet de no contemplar-ho no modifica la complexitat del programa atès que hi ha 3 bucles de cost $O(n)$.

Requeriments:

- `python main.py <File>`
- `python mainRecursive.py <File>`

Cal destacar que els arxius `.py` han d'estar en el mateix directori atès que tenen dependències entre ells.

Link GitHub:

- <https://github.com/marc7666/AiC-practica-1.git>