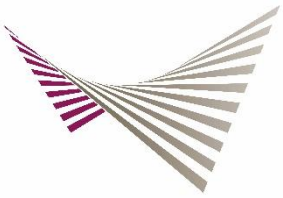


ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA

Documentació pràctica 2 Algorítmica I complexitat

Aarón Arenas Tomás – 78098697N

Marc Cervera Rosell – 47980320C

Jordi Planes Cid i Alba Lamas Varela

Curs 2020 – 2021

Índex

Observacions i aclariment previs:

Tota la pràctica ha estat desenvolupada en equip de dues persones, Aarón Arenas i Marc Cervera.

A continuació, s'adjunten els enllaços als perfils de GitHub dels autors:

Aarón Arenas	Marc Cervera
https://github.com/aaron-at97	https://github.com/marc7666

NOTA IMPORTANT PER ALS CORRECTORS: Els errors de pylint deshabilitats al llarg del codi son falsos positius de l'eina pylint.

Fitxers i mètodes:

Greedy iteratiu:

En primer lloc, s'inicialitzaran dues llistes anomenades "arbol" i "alture". La primera contindrà les coordenades 'x' i la segona les coordenades 'y'.

En segon lloc, s'inicialitzen quatre variables enteres. La primera anomenada "minimo", establirà el cost mínim, la segona anomenada "resultado" guardarà el resultat d'aplicar l'estratègia *greedy*, la tercera anomenada "pos" serveix controlar el valor del node anterior amb cost inferior per a un cop acabada la volta de bucle dels fills, buscar els fills de menor cost. L'última variable anomenada "i", és un posicionador per als vectors de coordenades 'x' i 'y' el qual permetrà les posició següent a cada volta del bucle.

Un cop inicialitzades les variables, "comença" l'estratègia en si. La condició del bucle es tradueix en que mentre no s'estigui en el nivell final de la branca amb menor cost, que es continuï donant voltes al *loop*.

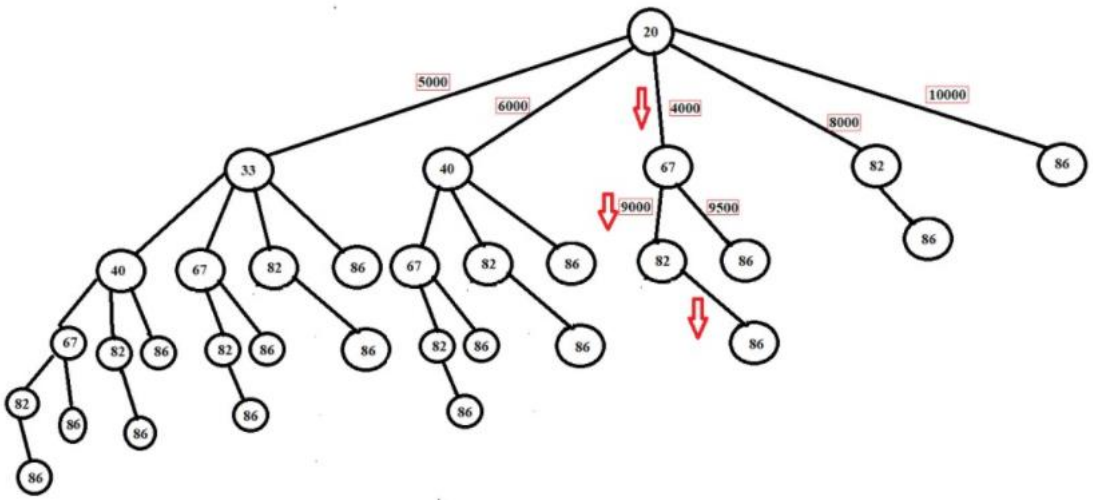
Un cop dins del bucle, el primer a fer és incrementar en 1 el posicionador per així ignorar l'arrel de l'arbre. Seguidament, si el posicionador no ha arribat al final de la llista de coordenades 'x', s'introdueixen, en les llistes pertinents, la coordenada 'x' i la coordenada 'y' que marqui el posicionador dins de les arrays que contenen les susdites coordenades. A continuació, mitjançant el mètode *obtain_distance*, és calculen les distàncies entre columnes i mitjançant el mètode *costs_aqueduct* es calcula el cost de construir l'aqüeducte i la possibilitat de fer-ho. Sense sortir d'aquest condicional, es comprova si el cost calculat és menor al mínim. Si ho és, s'actualitza el valor del mínim i la posició del susdit cost menor per a comparar els seus fills mes endavant. Un cop fet

això, s'elimina l'últim valor de les llistes "arbol" i "alture" per a la següent volta del bucle afegir el següent valor a comprovar.

Un cop acabada la volta d'un nivell, es comença amb els següent nivell, per a comprovar els fills amb menor cost. Un cop seleccionat el fill amb menor cost, s'introdueix en les llistes corresponents les seves coordenades i es reinicia el valor del mínim.

Un cop es trobi una solució al problema, es retorna el resultat (el cost) i la possibilitat de construir l'aqüeducte.

Exemple d'execució:



Greedy recursiu:

En aquest fitxer s'ha passat a recursiu el mètode del fitxer *greedy.py*.

Backtracking recursiu:

En aquest fitxer s'ha passat a recursiu el mètode del fitxer *backtracking.py*.

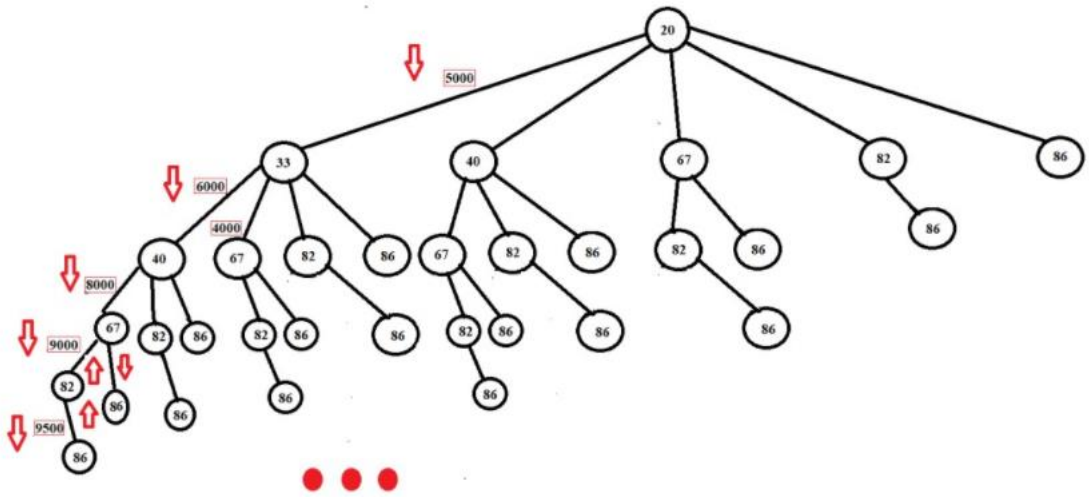
Backtracking iteratiu:

En primer lloc, s'introdueix a les llistes pertinents, les coordenades de l'arrel.

En segon lloc, és realitza un bucle *for* (es segueix l'esquema donat a teoria), que s'inicia a 1 (per evitar l'arrel). El que es farà a cada volta del bucle, és introduir en les llistes pertinents les coordenades del node que s'està tractant, es calcularà l'índex de la coordenada 'x' (l'índex que ocupa en la llista que conté totes les coordenades 'x') i s'utilitzarà aquest índex per realitzar una crida al mètode "hijos" definit en l'arxiu *arbol.py*. Finalment, s'eliminen de les llistes "arbol" i "alture" l'últim valor,

corresponents a les coordenades del node que s'està tractant en la volta actual del bucle, per evitar conflictes amb altres nodes en les següents voltes.

Exemple d'execució:



Dynamic programming iteratiu:

```
arbol.Arbol.arbol.append(distance_x[0])
arbol.Arbol.alture.append(alt[0])
arbol.Arbol.arbol.append(distance_x[1])
arbol.Arbol.alture.append(alt[1])
arbol.Arbol.hijos(arbol.Arbol, 2, distance_x, alt, height_aqueduct, alpha, beta)
```

La imatge anterior realitza el càlcul de tots el fills de la primera branca del node arrel.

Abans d'entrar al bucle que realitza els càlculs, es defineix una variable entera anomenada "pos" que servirà per avançar dins de cada rama (de posició en posició) i per saltar a la rama següent (cada dos posicions). El fet que sigui cada dos posicions és per ignorar l'arrel de l'arbre i la primera branca de l'arrel.

Seguidament, s'entra en el bucle *for* que s'inicialitza a 2 i donarà voltes mentre no arribi al final de la llista que conté les coordenades 'x' de la branca calculada per la funció *hijos* (la primera branca de l'arrel).

El bucle està estructurat en una sentència bàsica de condicionals *if*, *else*. La condició d'entrada a l'*if* que no s'estigui en una fulla. En cas d'entrar, s'introdueixen a les llistes pertinents (res_distance i res_alture), les coordenades 'x' i 'y' del node que s'està tractant. En cas de estar en una fulla, es reescriuen els dos vectors de resultat, posant l'arrel en la posició 0 de cada vector (en cada vector la coordenada de l'arrel que correspongui). A continuació, es calcula les distàncies entre columnes mitjançant una crida al mètode *obtain_distance* i es calcula tant el cost com la possibilitat de construir l'aqüeducte.

Seguidament, es comprova si cost calculat és menor al mínim i si és així s'actualitza el valor.

Per continuar, es comprova si "res_distance", vector que conté les coordenades 'x' del resultat solsament té 2 valors, cosa que significa que només hi ha l'arrel de l'arbre i una fulla i això també implica que ja s'ha recorregut tot l'arbre. Per finalitzar la part de l'*else*, s'incrementa en 2 la variable "pos" i es reinicien els vectors de resultat.

Finalment, al final de cada volta del bucle s'incrementarà en 1 la variable "pos" i es retornarà la possibilitat de construir o no l'aqüeducte.

Dynamic programming recursiu:

En aquest fitxer s'ha passat a recursiu el mètode del fitxer *dynamic_programming.py*.

Read file:

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

Makefile:

Amb la comanda *make lint*, s'executarà l'eina pylint per a tots els fitxers de codi.

Amb la comanda *make test*, s'executaran tots els arxius del directori de testos.

Amb la comanda *make test2*, s'executaran els testos més ràpids. S'ha triat aquesta via per facilitar la correcció a l'equip docent.

Amb la comanda *make all* s'executaran a més de tots els testos, l'eina pylint.

Arbol.py:

El primer que s'observa en el fitxer és una classe anomenada "Arbol", la qual conté 6 variables globals i un dos mètodes: el constructor i un que genera tots els possibles fills i en calcula el cost.

Les variables globals són:

arbol => Llista que conté totes les coordenades 'x' de l'arbre.

alture => Llista que conté totes les coordenades 'y' de l'arbre.

impossible => Booleà que controla la possibilitat de crear l'aqüeducte.

minimo => Enter que marca el cost mínim.

dynamic => Llista que conté les coordenades 'x' de la branca que s'està analitzant.

dynamic2 => Llista de coordenades 'y' de la branca que s'està analitzant.

L'explicació del mètode "hijos", es dividirà en dues parts per facilitar-ne l'explicació.

La primera part es la de l'*if*, on la condició d'entrada passa per comprovar si el node pare no es una fulla de l'arbre. Un cop dins del condicional, el primer que es fa és introduir en les llistes corresponents, la 'x' i la 'y' del node que s'està analitzant. Seguidament, es realitza una crida recursiva amb el següent node.

L'esquema de funcionament del mètode seria: "El node M té fills? si en té, li preguntem al node M + 1 (el fill) si te fills. En cas que el node M no tingui fills anem al node M - 1 (el seu pare) i realitzem la mateixa operació. Un cop generada la rama es calcula el cost".

Finalment, en aquesta primera part, com ja s'ha arribat a una fulla de l'arbre, s'extreuen nodes (es puja a l'arbre) fins a arribar a un node que tingui més fills.

La segona part, la de l'*else*, és la part que calcula els costos de les rames. Per fer-ho s'ha utilitzat l'estratègia descrita a continuació.

En primer lloc, el condicional assegura que hem arribat a una fulla. Un cop s'ha comprovat que s'està en una fulla, s'obté la distància entre un parell de columnes gràcies

al mètode *obtain_distance* del fitxer *calculs.py*. Seguidament, i gràcies també al mètode *costs_aquedutct* del mateix fitxer, es calcula el cost de construir l'aqüeducte i la possibilitat.

Finalment, s'actualitza el valor del booleà "impossible", el "dynamic", el "dynamic2" i es comprova el cost mínim.

Càlculs.py:

En obrir el fitxer, el primer mètode que s'observa és *calc_impossible_pont* però d'aquest mètode no se'n donarà cap especificació atès que és de la implementació de la primera pràctica. L'equip de desenvolupament, ha optat per deixar el codi d'aquest mètode juntament amb el del mètode *cost_pont* que també és de la primera pràctica.

El primer mètode "útil" per al desenvolupament d'aquesta segona pràctica, és el mètode *calc_impossible* que retorna cert o fals en funció de si es possible construir, o no, un aqüeducte. Per determinar aquesta possibilitat, primerament és realitza el càlcul del radi com "distance / 2" on:

distance => distància entre dues columnes

En segon lloc es calcula l'altura de l'aqüeducte com la resta entre l'altura total i el radi.

Finalment, es retorna *True* si l'altura recentment calculada és major a una coordenada 'y'.

El següent mètode observable, és *obtain_values*, el qual retorna tres valors diferents; la distància entre dues columnes, les diferents altures (coordenades 'y') i les coordenades 'x'.

L'estratègia seguida és: en primer lloc, s'inicialitzen 4 variable:

distance => Llista que contindrà les distàncies entre columnes.

distance_x => Llista que contindrà les coordenades 'x'.

alt => Llista que contindrà les coordenades 'y'.

ant_dis => Distància anterior.

Aleshores per emplenar cada llista amb els valors, s'utilitza un bucle que primerament obté les coordenades 'x' i 'y' les quals introdueix en la llista corresponent i seguidament, comprova que la distància anterior sigui diferent al valor inicial (significa que no estem a la primera columna). En cas d'entrar en aquest condicional, s'introdueix a la llista de distàncies la resta entre la 'x' de la columna actual i la 'x' de la columna anterior.

Finalment, s'actualitza el valor de la distància anterior i es retornen les tres llistes.

A continuació, el mètode *obtain_distance* és presentat en el fitxer. Aquest mètode retorna una llista amb les distànciues entre columnes. El funcionament és paregut al de la funció anterior però amb algun petit canvi. En primer lloc s'observa un condicional que comprova que hi hagi exactament dues coordenades 'x', cosa que vol dir que només hi haurà dues columnes i per tant, un pont. En aquest cas, s'introdueix la distància entre aquestes columnes amb la resta descrita en l'anterior mètode i es retorna la llista.

En cas d'haver més de dues coordenades 'x' i que, per tant, hi hagi més d'un possible pont, es calcula per cada coordenada 'x' la distància entre columnes i finalment, es retorna la llista amb les distàncies.

El mètode següent, és l'anomenat *measures* que retorna la coordenada 'x' i la coordenada 'y' d'un punt del terreny donada la seva tupla.

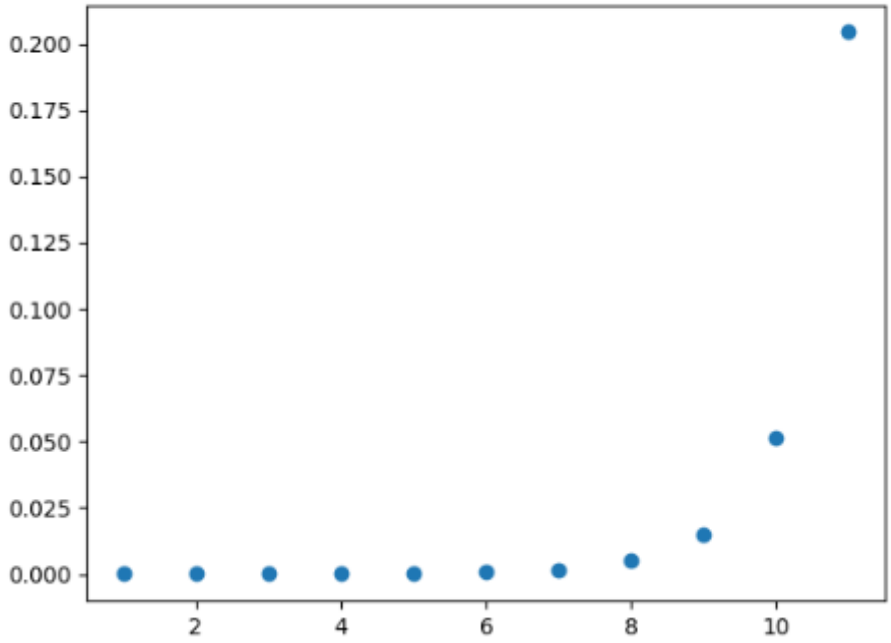
Com a últim "mètode" útil per a la resolució d'aquesta pràctica, hi ha el mètode *costs_aqueduct* el qual retorna el cost de construir un aqüeducte i la possibilitat de fer-ho.

Primerament, s'inicialitzen dues variables enteres que controlaran, una els costos d'altura i l'altra, els costos de distància. Com a última variable a inicialitzar, s'inicialitza un booleà que controlarà la possibilitat de crear l'aqüeducte.

Seguidament, mitjançant un bucle, que donarà voltes fins que la variable de control hagi arribat al valor de la variable "n" proporcionada per les dades del propi problema és calcula: primerament, costos d'altura establerts com la suma dels costos anteriors mes la resta l'altura total de l'aqüeducte i la coordenada 'y' corresponent. A continuació si la variable de control és major a 0 (hi ha mínim un valor més a la llista de distàncies) és calcula el valor del booleà mitjançant una crida a *calc_impossible*. Seguidament, si la variable del bucle es menor al nombre punts del terreny menys 1 (si no es fes la resta la llista sortiria d'índex), s'actualitzen els costos de distància establerts com la suma entre: els costos de distància anteriors mes la distància entre un parell de columnes al quadrat. Com a última comprovació del bucle, hi ha un condicional que tallarà l'execució del bucle quan el valor de la variable booleana canviï de valor.

Finalment, es calcula el cost total mitjançant la formula proporcionada en l'enunciat de la pràctica i es retorna aquest cost i el booleà que marca la possibilitat de construcció.

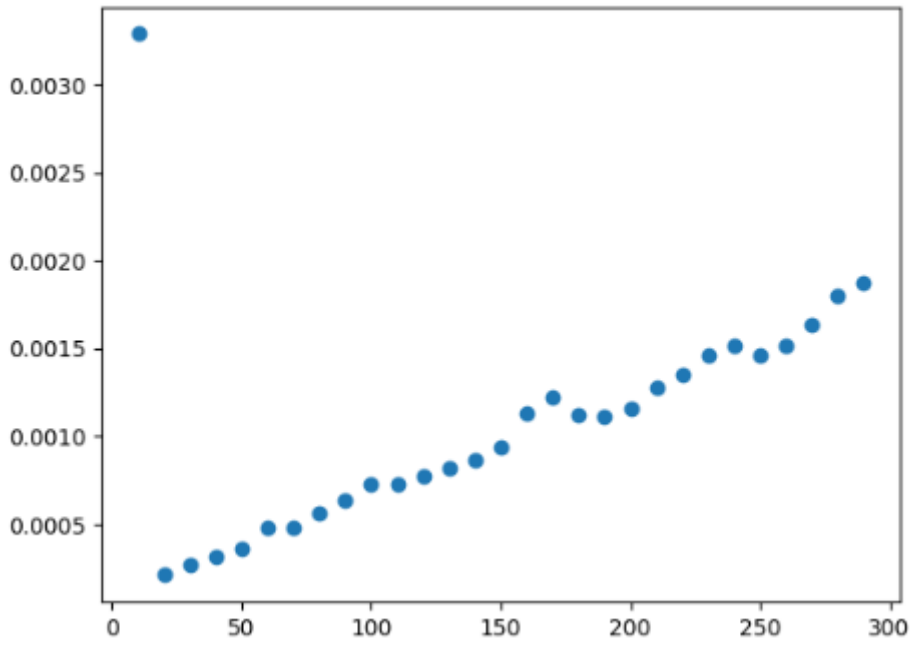
Costos:



Il·lustració 1: Cost backtracking

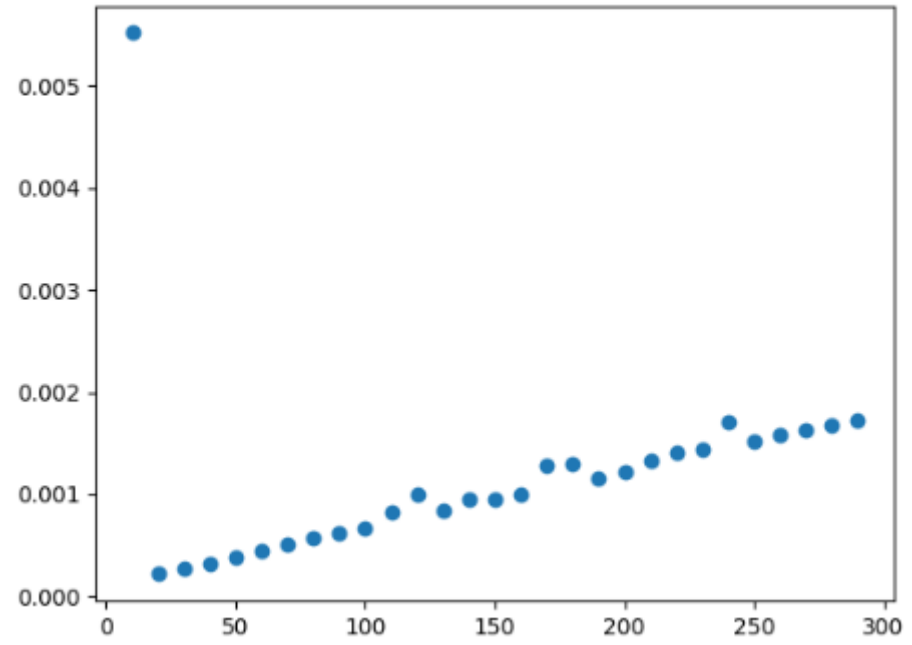
Cost = O(n^2).

El cost per a la versió recursiva és de O(n)



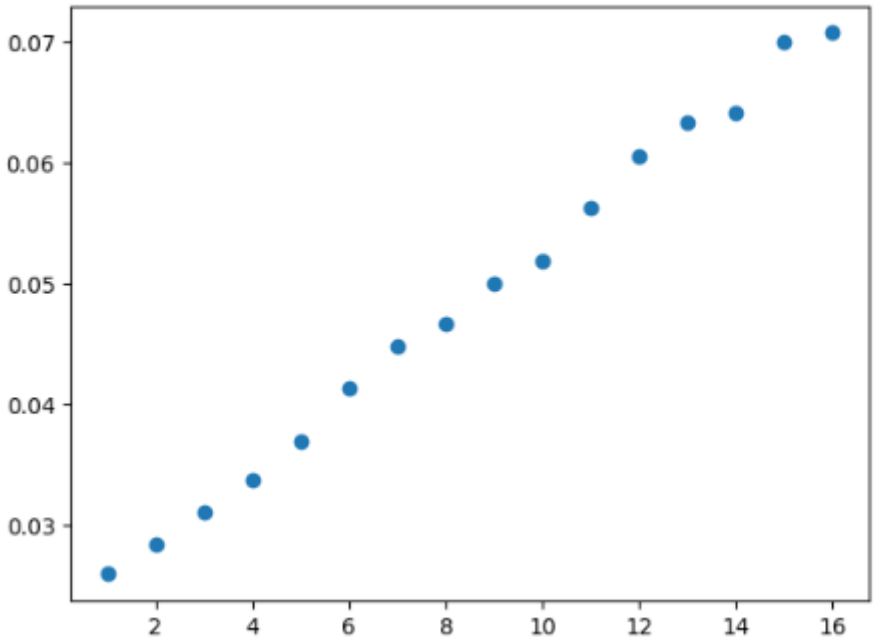
Il·lustració 2: Cost dynamic programming iteratiu.

Cost = O(n)



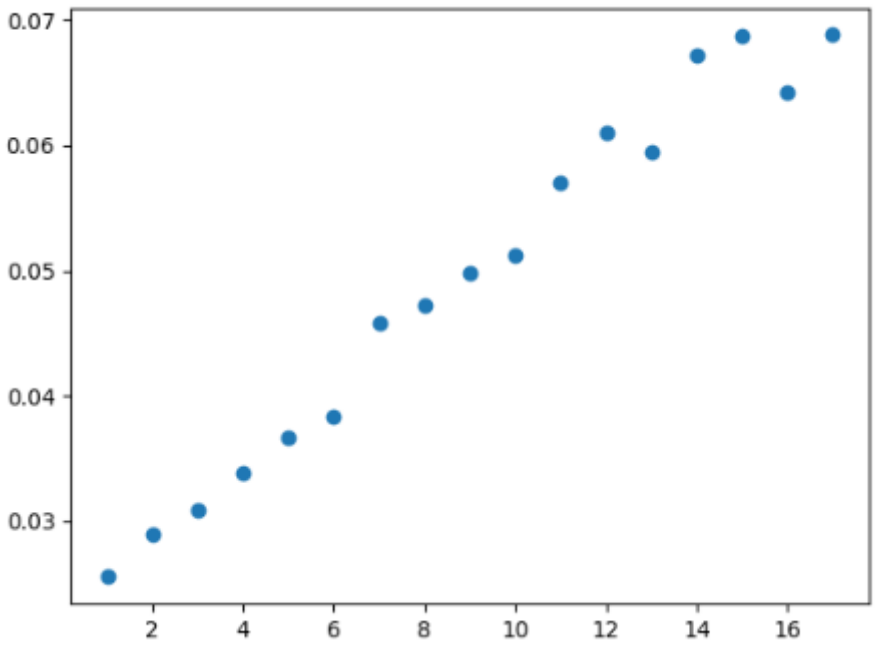
Il·lustració 3: Cost dynamic programming recursiu.

Cost = O(n)



Il·lustració 4: Cost greedy recursiu.

Cost = O(n)



Il·lustració 5: Cost greedy iteratiu.

Cost = O(n)

En la imatge veiem dispersió en els valors. Això és degut a la memòria del PC i al canvi de posició del vector de l'arbre.