Pràctica 1: Limes transire

Preparat per: Blanco Lopez, Jordi – 20998
Fernàndez Cid de Rivera, Jesús – 48053542X

Data d'entrega: diumenge, 26 de Març del 2017

Introducció

Índex de CONTINGUTS

OBJECTIUS

L'objectiu principal d'aquesta pràctica és fer un programa que ens digui el nombre màxim de caixes que ens quedaran al camió per a que es puguin veure les etiquetes de totes les caixes. El programa ha d'acceptar tan l'entrada estàndard (teclat) com l'entrada des de fitxers. Hem de fer 2 algoritmes, un iteratiu i un altre recursiu, i avaluar els seus costos teòrics i pràctics. Finalment, hem de fer la documentació de la pràctica.

EXPLICACIÓ DEL PROGRAMA

Després d'obrir el terminal i col·locar-nos en el directori corresponent haurem d'escriure el següent per executar el programa i començar amb la fase d'obtenció de dades:

$ python ilerda.py [-r / -i] [fitxer-caixes.txt]

on –r ens indica recursiu i –i ens indica iteratiu. Per defecte el programa l'executa en iteratiu. I per últim, el fitxer que si no es posa, el programa et demanarà que l'introdueixis:

Numero de la Caixa + Alçada de la Caixa *ex. 1 9*

Separat per espai i que per deixar d'introduir més naus haurà de prémer "enter" sense escriure res. Per qualsevol altra cosa, el programa donarà error i el sistema acabarà amb l'execució del programa.

Després de la fase d'obtenció de dades, utilitzarem una funció de cerca lineal per trobar el màxim de caixes que poden anar sobre el camió. Hem triat aquest algoritme de cerca per poder observar entre totes les possibles solucions quina era la mes idònia per al nostre cas.

Quan ja tenim el màxim de caixes , segons s'hagi especificat a l'inici del programa, cridarem la funció recursiva o iterativa.

Quan l'algoritme trobi la cadena que compleix el màxim de caixes, retornarà a la llista amb el numero de les caixes i les seves respectives alçades i el programa ho mostrarà per sortida estàndard, és a dir, per pantalla.

EXPLICACIÓ DEL ALGORITME D'ORDENACIÓ

Per ordenar les dades, hem decidit emprar dos algorismes diferents, en funció de quina funció emprem en cada cas, per tal de dividir-ho en la millor forma possible. En el cas de la funció iterativa, la funció, anomenada searchBestLen, la qual li passem el paràmetre de l'alçada, i hi tractem, per mitjà d'un doble bucle tractem l'ordenació de les dades, en el primer del bucle, inicialitzem les variables que ens tracten la llargada màxima de la seqüència i el valor inicial de la seqüència. Un cop hem establert el valor des del qual partim, dins del segon bucle, el que fem és anar comparant el primer paràmetre amb les alçades posteriors, per tal d'establir la màxima seqüència possible, de tal forma que l'algoritme queda així:

```
def searchbestlen(weight):
    bestlen = 0
    for i in range(0, len(weight)):
        large = 1
        littleWeight = weight[i]

        for x in range(0, len(weight)):
            if littleWeight < weight[x] and i < x:
                littleWeight = weight[x]
                large = large + 1

        if bestlen < large:
            bestlen = large
            posbest = i

    return posbest
```

Figura 2: Codi emprat per l'algoritme d'ordenació iteratiu

En el cas de la funció recursiva usem un algorisme una mica més complex, en el qual hi contenim dues funcions, les quals una crida a l'altra de forma recursiva. En la primera, inicialitzem les variables pertinents, i per mitjà de les condicions, primer tractem els casos simples, i en el segon és quan iniciem l'ordenació, atorgant el valor d'inici de la seqüència i a partir d'aquí, dins de la segona funció és quan realitzem la recursivitat de l'algorisme, ja que comprovem els valors que poden seguir la seqüència i els anem emmagatzemant, per tal d'obtenir la seqüència final de forma correcta.

```
def searchbestlenR(weight):
    bestlen = 1
    posbest = 0
    index = 0
    if len(weight) == 1:
        return posbest

    else:

        for firstindex in range(0, len(weight)):
            littleWeight = weight[firstindex]
            large = 1
            large = searchbestlenR2(weight, large, littleWeight, firstindex, index)

            if bestlen < large:
                bestlen = large
                posbest = firstindex

        return posbest
```

```
def searchbestlenR2(weight, bestlen, littleWeight, firstindex, index):
    if index < len(weight):
        if weight[index] > littleWeight and firstindex < index:
            littleWeight = weight[index]
            bestlen += 1

        index += 1
        return searchbestlenR2(weight, bestlen, littleWeight, firstindex, index)
    else:
        return bestlen
```

Figures 3 i 4: Funcions de l'algorisme recursiu

EXPLICACIÓ DE COSTOS DEL ALGORITME DE CERCA

Costos Teòrics de l'algoritme iteratiu

L'algoritme emprat per la busca iterativa que hem emprat, basant-nos en la figura 2, podem arribar a calcular, mitjançant la notació Big-Oh, que el cost de les operacions de la funció és de $O(N^2)$, ja que tenim el doble bucle per tractar totes les possibles combinacions de caixes possibles, en cas que ens toqués retrocedir.

Els costos sempre es calculen en el pitjor dels casos possibles, però si ens fixéssim en el millor dels casos que ens poguéssim trobar, que en aquest cas seria una llista de caixes que només contingués una caixa, el cost de la mateixa operació seria de O(1), que ja directament, com es veu en la figura ho comprovem dins d'una condició. Els possibles casos de llargada de les cadenes de caixes per mostrar el cost de la funció es mostraran en la següent taula:

Llargada de la cadena de caixes	Cost de l'operació (Worst case scenario, en unitats de segon)
5	25
10	100
20	400
50	2500
100	10000
200	40000
500	250000

Taula 1: Costos teòrics de l'algoritme de cerca iteratiu:

Costos Teòrics de l'algoritme recursiu

En l'algoritme recursiu, tal i com hem fet en l'anterior cas, per mitjà de la notació Big-Oh, basant-nos en les figures 3 i 4 podem comprovar que dins del bucle de la primera funció, el que fem és cridar a la funció searchBestLenR2, que és la que ens va realitzant les comprovacions pertinents. Un cop hem comprovat això, podem definir que el cost teòric de l'algorisme recursiu és de $O(N^2)$, definint la taula de forma similar a la Taula 1, ja que al ser el mateix cost, calcularíem els mateixos costos que en la primera funció.

Cal destacar que a l'hora de calcular el cost d'una crida recursiva, sempre serà de mida O(N), ja que el que estem realitzant és la mateixa crida a la funció, tants cops com paràmetres tinguem que analitzar.

EXPLICACIÓ DE L'ALGORITME ITERATIU

Per crear l'algoritme iteratiu, seguint l'informe de la pràctica, el primer que fem és crear dos arrays, que seran els que després printejarem per pantalla, la qual la seva funció principal és anar guardant els resultats que siguin els que segueixen la cadena, tant de número de caixa com de alçada. Una vegada tenim les caixes assignades, passem a tractar, per mitjà d'un bucle, el cas simple, que és el que hem comentat al algoritme d'ordenació iteratiu.

Un cop tractat aquest cas, passem a ordenar les caixes que disposem, mitjançant la crida a l'algoritme d'ordenació, que anem agafant els paràmetres que ens interessen per tal d'establir el ordre correcte de les caixes, i després els anem col·locant als arrays, de forma que acabarem tenint en cada un el nombre de les caixes i les alçades que són necessàries per a que estigui ordenada la seqüència.

```
def iteratiu(boxnum, weight):
    truckB = []
    truckW = []
    if len(boxnum) == 1:
        return boxnum[0], weight[0]
    else:
        posbest = searchbestlen(weight)

        for i in range(0, len(weight)):
            if i == posbest:
                truckB.append(int(boxnum[i]))
                truckW.append(int(weight[i]))
                lastbox = weight[i]

            elif i > posbest:
                if weight[i] > lastbox:
                    lastbox = weight[i]
                    truckB.append(int(boxnum[i]))
                    truckW.append(int(weight[i]))

        return truckB, truckW
```

Figura 5: codi de l'algorisme Iteratiu

Com podem observar, un cop tenim posbest, anem comparant-lo amb el bucle de l'algoritme, i anem afegint als arrays en funció del que tenim de l'algoritme d'ordenació.

EXPLICACIÓ DE COSTOS DEL ALGORITME ITERATIU

Costos Teòrics de l'algoritme

L'algoritme, com hem dit al començament, disposa d'un bucle i d'una crida a la funció que ordena les caixes, emprant la notació Big-Oh, podem dir que l'algoritme iteratiu, de la mateixa forma que l'algoritme d'ordenació, té un cost de $O(N^2)$, quedant els resultats de manera similar a la Taula 1, ja que conté el mateix cost que aquest algoritme.

EXPLICACIÓ DE L'ALGORITME RECURSIU

Per a la realització de la funció recursiva, emprem un mètode similar a l'algorisme de cerca recursiu, ja que hi emprem dues funcions, de les quals la primera realitza una crida a la segona i així complir la recursivitat que se'ns demana. En la primera de les funcions, igual que en l'algorisme iteratiu, creem dos arrays que seran els que guardarem les seqüències ordenades. Primer de tot, igual que en el cas de l'algoritme iteratiu, tractem els casos simples, dins d'una condició, ja després mitjançant l'algorisme d'ordenació explicat a les figures 3 i 4, agafem els valors que necessitem per a ordenar la seqüència, i un cop ho tinguem, cridem a la segona funció, que és la que realitza la recursivitat, per tal d'anar col·locant les caixes als arrays pertinents, que serà el que retornarem al final. Cal destacar d'aquesta funció, que en la recursiva passem tots els paràmetres que necessitem per a la ordenació, i els anem col·locant per a cada volta dins la mateixa.

```
def recursiu(boxnum, weight):
    truckB = []
    truckW = []
    lastbox = []
    index = 0
    if len(boxnum) == 1:
        return boxnum[0], weight[0]
    else:
        posbest = searchbestlenR(weight)
        recursiu2(boxnum, weight, posbest, index, truckB, truckW, lastbox)
        return truckB, truckW


def recursiu2(boxnum, weight, posbest, index, truckB, truckW, lastbox):
    if index < len(weight):
        if posbest == index:
            truckB.append(int(boxnum[index]))
            truckW.append(int(weight[index]))
            lastbox = weight[index]

        elif posbest < index:
            if weight[index] > lastbox:
                lastbox = weight[index]
                truckB.append(int(boxnum[index]))
                truckW.append(int(weight[index]))

        index += 1
        recursiu2(boxnum, weight, posbest, index, truckB, truckW, lastbox)
```

Figura 6: Codi de l'algoritme recursiu

EXPLICACIÓ DE COSTOS DEL ALGORITME RECURSIU

Costos Teòrics de l'algoritme

Tal com hem dissenyat l'algoritme iteratiu, també hem dissenyat el recursiu. L'algorisme emprat , seguint la notació Big-Oh, igual que els casos anteriors, té un cost de $O(N^2)$, ja que la crida a la segona funció recursiva, està plena de condicions, així que el cost és de O(1). El sobrecost de la funció ve donat per la crida a l'algoritme de cerca, que és el que té el cost de $O(N^2)$.

CONCLUSIONS

Amb la realització d'aquesta pràctica hem pogut aprofundir amb el llenguatge de programació Python i aplicar els coneixements adquirits a les sessions de grup gran per aconseguir l'objectiu, que és la realització d'aquesta pràctica.

Durant la realització de la pràctica, vam estudiar diversos casos de com poder resoldre aquest problema, de tal forma que, encara i tot amb les nostres limitacions, l'hem pogut treure d'una forma mitjanament satisfactòria.

Cal destacar també, l'aprofundiment dels nostres coneixements que hem adquirit mitjançant la realització de la mateixa, podent posar a prova aspectes ensenyats a classe.