Algorítmica i
Complexitat
Practica 1: Limes transire

Lucian Marian Nedelcu

Albert Mari Riu

Index

1. Introducció.

L'objectiu d'aquesta pràctica es donat un problema, aplicar el coneixements que s'han vist a teoria, donar la especificació formal del algorisme, donar un algorisme que pugui resoldre el problema tan el forma iterativa com recursiva i finalment calcular els costos tan teòrics com pràctics del algorisme.

2. Especificació formal.

P: { L[0.. n-1] }

I: { find l[i..m], i<n-1<=m }

Q: $\{ l[i..j] \subseteq L[0..n-1], (0 \leq i \leq n-1, i \leq j \leq n-1) \wedge \nexists l\,[i'..j']/j'-i' > j-i\}$

3. Algorismes

Input: L[..n] = Llista de elements:(caixa, alçada) de longitud n, n>0.

Ouput: l[..m] = la subllista d'elements de L on cada caixa $c_i teunaalçada h_i < h_j, i < j$ **de longitud maxima.**

3.1. Versió iterativa

Iteratiu:

```
function find_longest_sequence ( L ):
    n = size( L )
    sequence_length_list = [ (0, -1 ) ] * n
    for ( int i = n - 1; i > -1; i--) do
        longest_in_i = 0
        next_increase = -1
        for ( int j = i+1; j < n; j++ ) do
            if ( *L[ i ][ 1 ] < L [ j ][ 1 ] and sequence_length_list [ j ][ 0 ] +1 > best ) do
                longest_in_i =  sequence_length_list [ j ][ 0 ] +1
                next_increase = j
        sequence_length_list [ i ] = ( longest_in_i, next_increase )

    increase = max ( x for x : L[ x ][ 0 ] )
    while increase != -1 do
        result.put(L[ increase ])
        increase = L [ increase ][ 0 ]

    return result
```

*note: L[i][1] $L[i][1] \equiv h_i$

3.2. Versió recursiva

```
function find_longest_sequence( L , m):
        longest_sequence = [ ]
        //base case
        if L.size == 0:
                return L[ 0 ]
        //recursive case 1
        longest_sequence = find_longest_sequence ( L[ 1: ], m )
        //recursive case 2
        test = L[ 0 ]
        if ( m.height < test.height ) do
                current_sequence = [ test ] + find_longest_sequence (  L[ 1: ], test )
                longest_sequence = max_size( current_sequence, longest_sequence  )

                return longest_sequence
```

4. Costos

4.1. Costos teòrics

4.1.1. Versió iterativa

cost:		algoritme:
		function find_longest_sequence (L):
		_n = size(L)
		_sequence_length_list = [(0, -1)] * n
c1*n		_for (int i = n - 1; 0; i--) do
		__longest_in_i = 0
		__next_increase = -1
c2*$\sum_{i+1}^{n}$	t_i	__for (int j = i+1; n-1; j++) do
c3*$\sum_{i+1}^{n}$	$t_i - 1$	___if (*L[i][1] < L [j][1] and
		sequence_length_list[j][0] +1 > best) do
c4*$\sum_{i+1}^{n}$	$t_i - 1$	____longest_in_i = sequence_length_list [j][0] +1
		____next_increase = j
c5* $\sum_{i+1}^{n}$	$t_i - 1$	__sequence_length_list[i]=(longest_in_i, next_increase)
		_increase = max (x for x : L[x][0])
c6*		_while increase != -1 do
		__result.put(L[increase])

__increase = L [increase][0]
_return result

c6* no cal calcular el cost ja que es una funcio que construeix el resultat un cop trobat.

$$T(n) = c1*n + c2*\sum_{i+1}^{n} t_i + c3*\sum_{i+1}^{n} t_i - 1 + c4*\sum_{i+1}^{n} t_i - 1 + c5*\sum_{i+1}^{n} t_i - 1$$

$$T(n) \in O(n) + O(n^2) + O(n^2) + O(n^2) + O(n^2) = O(n^2)$$

4.1.2. Versió Recursiva

T(n) = T(n-1)+T(n-2) = T (n-2) + T(n-3) + T(n-3) + T(n-4) = T (n-3) + T(n-4) + T(n-4) + T(n-5) + T(n-5) + T(n-6) = …

$$T(n) \leq 2T(n-1) + n \text{ /k}$$

$$T(k) - 2T(k-1) = k$$

$$\frac{T(k)}{2^k} - \frac{T(k-1)}{2^{k-1}} = \frac{k}{2^k}$$

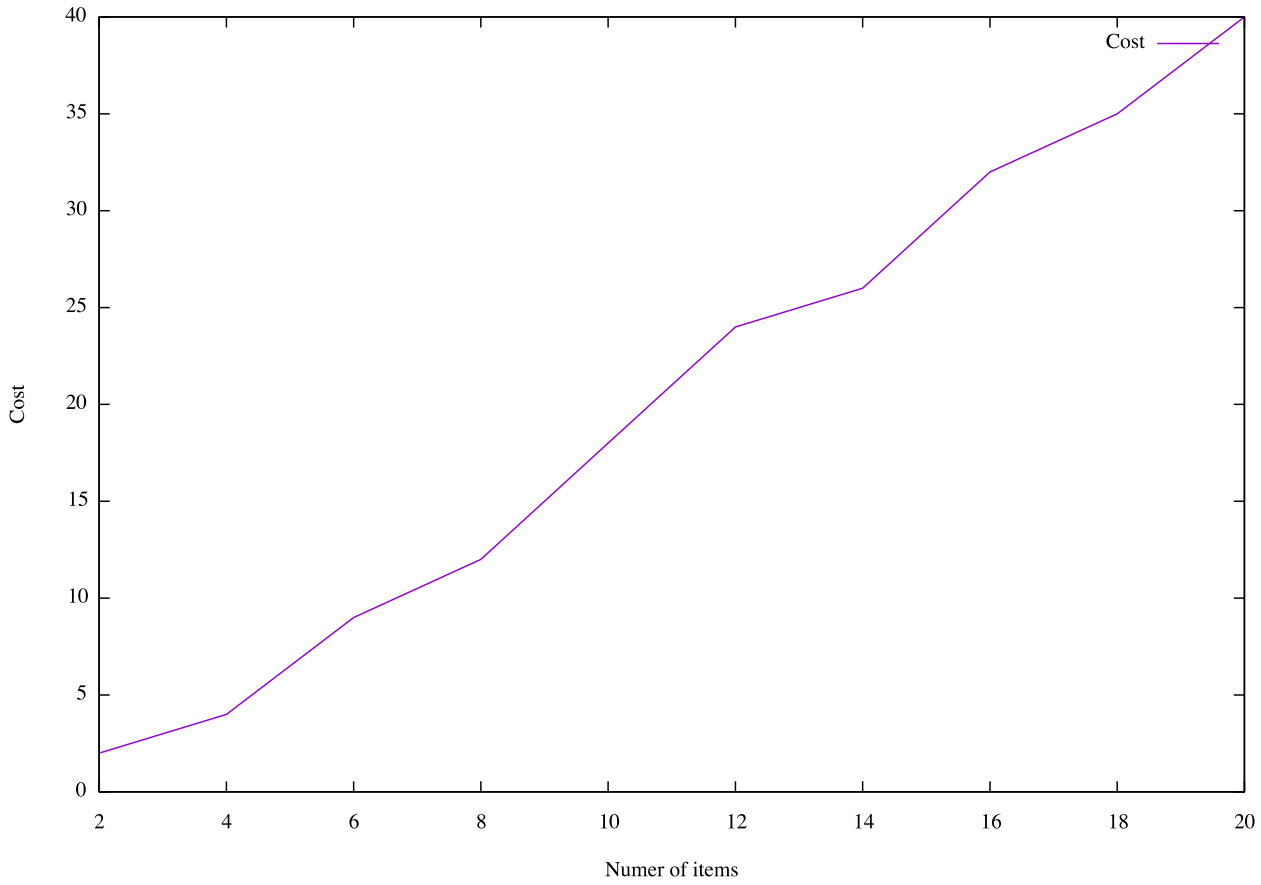
$$\frac{T(n)}{2^n} - \frac{T(0)}{1} = \frac{T(n)}{2^n} => T(n) = \sum_{k=1}^{n} k * 2^{n-1} \ldots (1) => \ldots$$

$$T(n) \in O(2^n)$$

4.2 Costos empírics

4.2.1. Versió iterativa

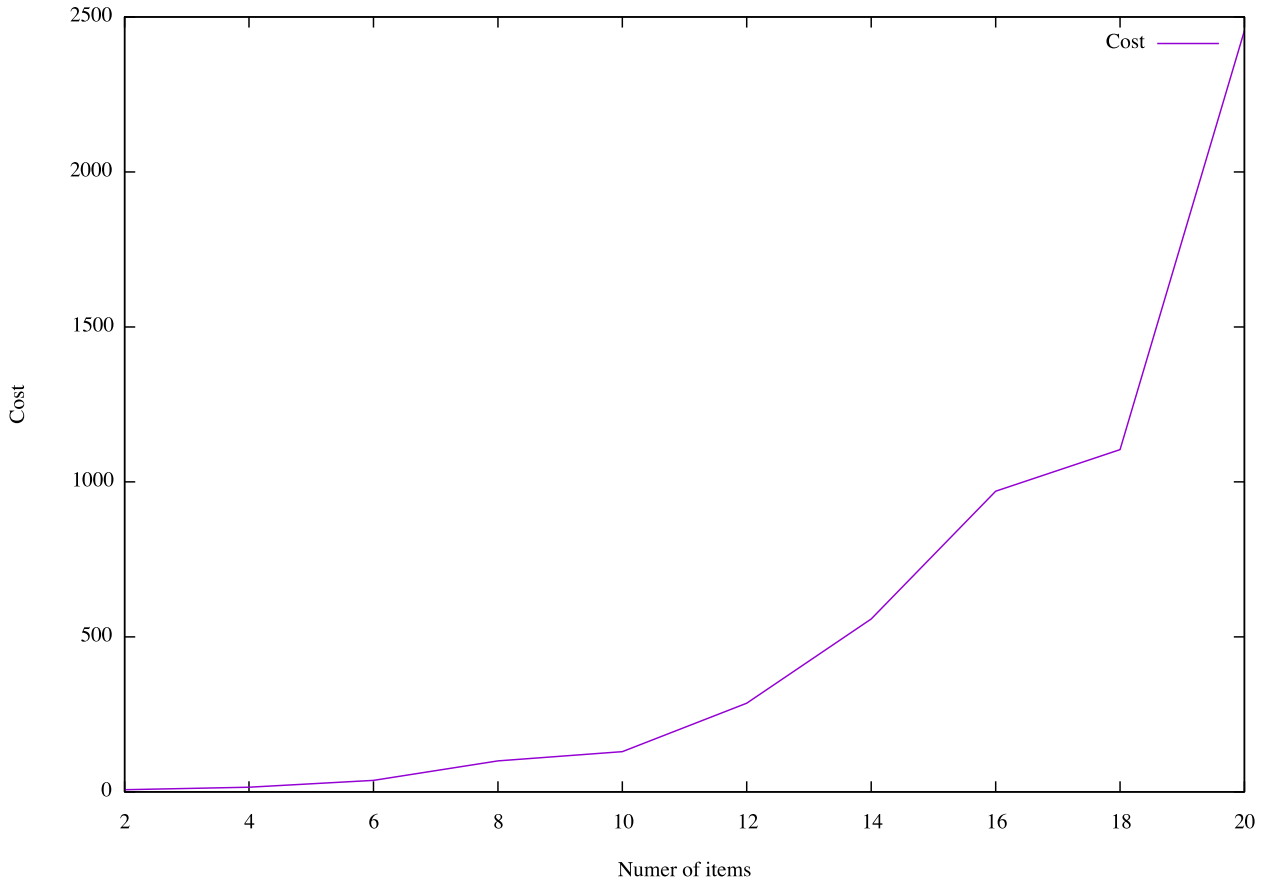
En la versió iterativa s'han mesurat els costos mitjançant un comptador que es va incrementant a mesura que es van recorrent els bucles. En la següent gràfica tenim els resultats obtinguts per a 20 elements:



Com es pot observar en la gràfica anterior el creixement del cost sembla ser lineal però creix més ràpid. Per això podríem dir que el cost es $O(n^2)$.

4.2.2. Versió recursiva

En la versió recursiva s'han mesurat els costos mitjançant un comptador que es va incrementant a mesura que es van fent crides recursives. En la següent gràfica tenim els resultats obtinguts per a 20 elements:



Com es pot observar en la gràfica anterior el creixement del cost tendeix a una corba ascendent que tindria forma de funció exponencial. Per això podríem dir que el cost de la versió recursiva es de $O(2^n)$.

4.3. Conclusions

Una de les conclusions que es pot treure sobre els dos tipus de costos es que son idèntics, ja que s'ha utilitzat, en el cas del costos empírics, les mateixes mesures, ja que en els teòrics, la versió iterativa es conta a través del nombre de voltes que es fa en un bucle i en la versió recursiva el nombre de crides que es fan, de la mateixa manera que s'han calculat de manera empírica.

Podrien haver variat una mica en el cas de que s'hagués agafat com a mesura per a calcular el cost el temps.

5. Referències

https://www.quora.com/How-do-I-solve-this-recurrence-relation-T-n-2T-n-1-+n-given-T-0-0

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.174.5400&rep=rep1&type=pdf