Homework1
Algorithm Design

Lombardo Andrea 1893440

Engineering in Computer Science La Sapienza

5/12/2019

Attached to the e-mail, in addition to this pdf file, there are also different files:

- named **ex1.py** that is respectively the Python implementation of the first exercise for the first part, the develop in O(kn^2)
- named **graph31.png** that is the graph of the third exercise on the first part
- named **graph32.png** that is the graph of the third exercise on the second part

1 Exercise 1- Andrea Lombardo-1893440

The goal of this problem is to find out an algorithm which returns the expected optimal reward, using two sets as inputs: the first,**Value** vector, with a range(0,n) which represents the possible prize hidden in the packs, the second set, **Cost** vector, identifies the costs of all packs from 0 to k-1 where in each entry I place the respective c_i for i with a range from 0 to k . Thus I've built a matrix with n+1 rows and k columns, and evaluating from last column to the first, I've computed the matrix. Main interested is the entry in the position [0][0] which represents our target. The expected revenue of a box depends on the expected revenues of the following boxes: considering the last column as the last box its expected revenue doesn't depend on any other box. In input we have the two sets: Value=[0,1,2,3..n], Cost=[c_1,c_2,c_3,...c_k], then I define n=Value[-1] and k=len(Cost).

```
for q in range(k,0,-1):
    for i in range(0,n+1):
     expected_value=0;
     for s in range(i+1,n+1):
       if q==k:
        expected_value+=Value[s]
       else:
        expected_value+=Matrix[s][q]
     if q==k:
       entry=max(i,((1/(n+1))*(expected_value+(i*(i+1)))-Cost[q-1]))
       Matrix[i][q-1]=entry
     else:
       entry=max(i,((1/(n+1))*(expected_value+(Matrix[i][q]*(i+1)))-Cost[q-1]))
       Matrix[i][q-1]=entry
```

In the first **for** with index **q** I've started from the last column then I've evaluated all the entries of that column with the second index **i**. Then with the third **for** I add all the values that start from **i+1** where **i** is the value that I've already won in the previous boxes. The **if-else** constructor helps me to understand how to behave in case I analyze the last box or the previous, because the previous as I've defined before depends by the following boxes. After the **for-cycle** with the index **s** I've decided if it is the case to open the new box or give up the game with **i**, the best prize found until now, so in the respective entry I've written the max between **i** and the expected value-cost of the opening box. In detail I've added to the variable **expected_value** a value: **i**×**i+1**, which represents how many times I'll prefer the value i, exactly i+1 and divided all to n+1, cardinality of the set of prizes. In the second part I should improve my algorithm in O(nk),substituting the last for-cycle for evaluating expected_value variable with the computation of series of Gauss which explains that the sum of the range(0,n) is ugual to n(n+1)/2 in case we don't start by 0 but from i we use the script on line 2:

```
if q==k:
   expected_value=(n*(n+1)/2)-(i*(i+1)/2)
   entry = max(i,((1/(n+1))*(expected_value+(i*(i+1)))-Cost[q-1]))
   Matrix[i][q - 1] = entry
   total1+=entry
   if i!=0:
     minus=(Matrix[i][q-1]-Matrix[i-1][q-1])*i + Matrix[i-1][q-2]
     Matrix[i][q-2]=minus
else:
   Matrix[i][q-1]=max(i,((1/(n+1))*(total1+Matrix[i][q-1]))-Cost[q-1])
   total2+=Matrix[i][q-1]
   if i != 0:
     minus=(Matrix[i][q-1]-Matrix[i-1][q-1])*i+Matrix[i-1][q-2]
     Matrix[i][q-2]=minus
```

After computing the value of entry in n+1 row on the previous of the last column(k-2), because the cycle start computing from the the k-1 to 0 column and from 0 to n+1 row, I should switch the role of the total variables and reset total1=0 instead total2 used to compute the previous column (k-3).

2 Exercise 2– Andrea Lombardo-1893440

The main goal of the first point of this problem is to develop an algorithm with a complexity polynomial to n which is able to return the complete graph of n nodes composed by $\frac{n(n-1)}{2}$ edges with the respective cost, using the Minimum Spanning Tree as input which should remain the unique path with the lowest total cost. MST is the subset of edges-weighted which manages to link n nodes with n-1 edges without any cycle with the minimum weighted edges respect the other possible. Using the cut property I've managed to split into two subset the nodes, the following rule say that in the cut-set the edge with the smaller weight belongs to the MST,so the remain edges of the cut-set have a weight which is at least the weight of the cut MST edge+1. In the first problem we use the following code for having a polynomial cost:

```
Sort all edges of MST by ascending weight with a cost: O(nlogn) with mergesort
for j in range(0,n-1) #identify the index of MST's edges sorted
    Select a cut with only an edge j of the MST consider the relative cut-set
    for k in range(0,lenght(cut-set which contain j))
        if (edge(k)==edge(j) ):
            continue
        else:
            if k do not exist:
                create the edge(k) with a cost(edge(k))=0
            cost(edge(k))=cost(edge(j))+1  #Update-phase
```

First we sort the edges of MST according to ascending order then I make the cut which include only an edge of MST and the rest should be edges between vertex of the two different sets created by the cut. Now according to the developed algorithm for each element of the cut-set if the edge is the same of the MST included in the cut I skip to the next element otherwise if the edge doesn't exist I built it linking the two nodes and I initialize edge(k)'s weight to 0, then I update the k element with a value of edge weight of MST in the cut-set +1. In this way I'm able to obtain a complete graph with the minimum possible weight starting from MST and with all the other edges of the complete graph which is not included in MST with a weight higher than MST edges such that MST will remain the same of the input. Thus I manage to verify this factor indeed MST will be unique also that the same as that giving in input. The demonstration of the authenticity of the algorithm is giving by the absurd so if we're considering the following idea **cost(k)=cost(j)** I could have different unique MST but this is against to the fact that exists only a unique MST instead if I am trying to substitute an edge of the complete graph with an edge of MST we can see how I demonstrate the contradiction due to the fact the new MST it's not the minimum possible MST. The main goal of the second point of the second problem is to develop an algorithm with a complexity O(nlogn) which is able to calculate the total weight of the minimum-weight complete graph .

```
1)Apply sorted algorithm: mergesort in ascending order in O(nlogn) of MST edges
2)Initialize a set of |n| subsets where each of them contain exactly 1 element which is
     a vertex: for example if |n|=4 we can have S=[[v1}],[v2],[v3],[v4] ]
3)Iterate for each edge=(vi,vl) of MST according the sorting made up in first step next
     instructions:
4)weight_total+=weight_edge_of_MST
5)Using the Find operation of Union-Find I am able to identify the numbers of
    components in set_{i} and set_{l} where set_{i} include vi instead set_{l} include
    vl.
6)weight_total+= (size_{i}*size_{l}-1)*(weight_edge_of_MST + 1)
7)before going to the 3) step make an union between [vi][vl] subsets so, in case of i=1
     and l=3 we have S=[[v2],[v1,v3],[v4] ]
```

In practise the cost of the algorithm is giving by the fact that the first line cost O(nlogn), then in the 3 line I 've a for-cycle that is going from 0 to n-1(edges of MST) according an ascending order. The next instructions on 4,5,6,7 lines are included in the for-cycle so the cost of the algorithm is O(n+nlogn) which we could see also as O(nlogn). I want to make in evidence the fact that in the 2 line I prepare the field to using routines of Union-Find, then in the 5, 6 lines I use a find and then a Union operations.
In detail in the lines of the for-cycles I identify the number of all edges which should be a subset of edges($size_i \times size_l$) minus 1 for the fact I should remove the edge of MST and the cost of them must be weight_edge_of_MST + 1 otherwise we don't prove that minimum complete graph. I want to make in evidence that in the first iteration in the line 6 for example we will add in the weight_total a value of 0 due to the fact we will start from the the less heaviest.

3 Exercise 3- Andrea Lombardo-1893440

The main goal of this exercise is to model Federico's chocolates business. In the following problem Federico wants to know how many chocolates should prepare every week according network Federico' friends. It's a classic flow problem which should be resolved with Ford Fulkerson algorithm.Now we're interest to create the right direct graph that respects different constraints:

- Source must have only outgoing links and I identify it as **Federico**
- Sink must have only incoming links which in our case is like an absorbent node that identifies the **collection of Costumers**
- **capacity conditions** for each edges we have to a flow f is minus than the capacity of that edge so 0≤f(e)≤c(e)
- **Conservation conditions**

$$\sum_{e\ into\ v} f(e) = \sum_{e\ out\ of\ v} f(e)$$

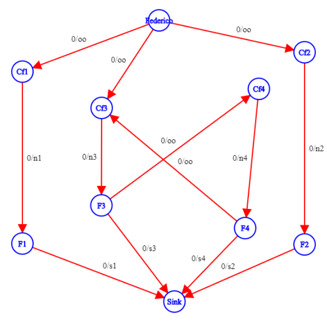
We know from the text that each Federico's friend has a capacity of n_i but each of them is able to sell no more than s_i to costumers. In particular I decide to split the job of each Federico's friends into two nodes: Cf_i represents the capacity of Federico's friend in detail it manages the incoming flow which come from Federico or Federico's friend instead F_i represents the activity of selling to costumers or Federico's friend so it manages the outgoing flow. In my example I decide to implement all the possibility related on incoming flows, so related on Cf_i:

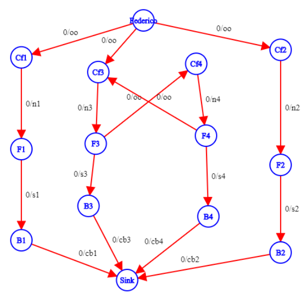
- Federico's friend receives only from Federico [example in Cf_2 and Cf_1]
- Federico's friend receives only from third part [example in Cf_4]
- Federico's friend receives from other and Federico [example in Cf_3]

In each graph how describe by the content of the text could happen that a friend shouldn't manage to sell to costumer his chocolates, in my case it's F_1 . The node identifies as Cf_i, has got only an outgoing link to the relative friend F_i with a capacity of n_i and the incoming edges with a fix capacity as ∞ depends in numbers according the configurations. Outgoing edges of Cf_i could start from the source or when happened that Federico could not meet with one of his friend they start from F_x to Cf_i where x must be different to i. We should analyze that when Cf_i receives chocolates by a F_x there should be another edge from Cf_x to F_i for proving the symmetric relationship between two Federico' friends. For modelling the second part I modify the previous graph:

- for each b ∈ B create a new node B_i that identifies a particular building
- link each f ∈ F with the respective building where he should sell chocolates with an edge with a capacity of s_i
- link the building with the costumers node(sink) with the edge of capacity cb_i

Behind there are the examples required with | F |=4. For modelling the fact that F1 doesn't manage to sell in that week analyzed I should consider the following restriction s_1=**0**. I want to mention that in the graph on the right that adding the node building the impact could be dangerous for Federico's business in case for example when cb_i is higher than s_i.





4 Exercise 4- Andrea Lombardo-1893440

The main goal of this exercise is to understand if Giovanni might be able to run n task in k available hours. Tasks are included in the task set J where each element of the set should have a l_j,d_j and s_j values which represent respectful duration, deadline and starting times of the task number j. Assuming that this parameters are natural number we should be able to verify that for each task of the task set the following constraints are valid:

- s_j is the starting time of the task number j this means that in the applied scheduling the task number j must start into a time value $\geq s_j$
- d_j is the deadline time of the task number j this means that in the applied scheduling the task number j must finish into a time value $\leq d_j$
- Following the two constraints above we can say that the time value when I start a task , as 1 constraint said $\geq s_j$, so we can resume this other constraint $s_j \leq s_{j_j} + l_j \leq d_j$ where s_{j_j} is the real time value when I start the task.

Now we try to demonstrate that this is a NP problem. This problem is a NP problem and we've proved it due to the fact I've got a sequence of tasks and through them I can check if that sequence is correct or not by checking if each task satisfy the constraints above. In the description of the problem I've got the following information

$$\sum_{j=1}^{n} l_j \leq k$$

Thus we would demonstrate that our algorithm is an NP-Hard problem. How? For proving it we should take another problem that we know that is NP-Hard and see if our problem is able to resolve all the possible instance of the Np-Hard problem considered. Taking the **Subset Sum Problem** according to which given n non-negative integers $w_1, \ldots, w_n$ and a target sum W, the question is to decide if there is a subset **I** which contains 1, . . . , n such that

$$\sum_{i \in I} w_i = W$$

so we can see it as an instance of our problem and we know it's a NP-Hard problem. For evaluating this factor we should consider the first n-1 task with best constraints possible for the single task we 'll have s_j=0 and $d_j = l_n + \sum_{j=1}^{n-1} l_j \leq k$.

The first constraint means that a job of the task set considered in the range (0,n-1) could start to a time value grater than 0. Instead the second constraint means that the n-1 tasks must end before the k, available hours before the party starts, given by the sum of the duration of each task. Now the task number n must have a $s_n = X$ and the deadline $d_n = X + l_n$ .After defining it we should find X through the subset sum problem of a set of value of the task set J and so the solution of our problem contains the solution of the subset sum problem. Now we've shown our problem is NP-hard.

I've proved that it's NP and NP-hard, so we can say that it is NP-Complete.

References

[1] Algorithm Design John- Kleinberg,Eva Tardos

[2] Algoritmi e strutture dati- C.Demetrescu, I.Finocchi ,G.P.Italiano