Algorithm Design - Homework 1

Sapienza University of Rome

Marco Costa, 1691388

December 5, 2019

Exercise 1

Problem

Given as input the number of boxes k, the number of different rewards n as well as the cost c_i (which is guaranteed to be integer) of every box, design an algorithm to find the expected optimal reward.

Solution

In order to solve the problem, we can define a matrix M composed of k rows and n columns, where the element $M[i,j]$ represents the expected value by opening the box i and having already as maximum prize j.
To populate the matrix we can use **backward induction**, proceeding first considering the last box and choosing what to do for all the possible prizes. Using this information, we can then determine what to do at the penultimate box. This process continues backwards until we have determined the best action for every possible prize for each box. So the element $\mathbf{M[0,0]}$ is the expected optimal reward.
To improve the complexity to $\boldsymbol{O}(n \cdot k)$ we can eliminate the summations, since in the worst case we have a sum over n ($\boldsymbol{O}(n)$) in nested cycles ($\boldsymbol{O}(n \cdot k)$) having a total cost of $\boldsymbol{O}(n^2 \cdot k)$. To do this we must distinguish the two cases:

- ***Last box***: we can remove this summation using a mathematical trick:
  $\sum\limits_{i=j+1}^{n} i = \sum\limits_{i=0}^{n} i - \sum\limits_{i=0}^{j} i = \frac{n\cdot(n+1)}{2} - \frac{j\cdot(j+1)}{2} = \frac{n\cdot(n+1)-j\cdot(j+1)}{2}$;
- ***Other boxes***: we can use an auxiliary vector in which we keep track of the sum of the elements of the matrix related to the box next to the current one.

Here there are the two ***Algorithms***:

Algorithm 1 Get optimal expected value ($\boldsymbol{O}(n^2 \cdot k)$)

```
1: $M[k, n+1] \leftarrow 0$
2: for $i$ in $(k-1, ..., 0)$ do
3:     for $j$ in $(n+1, ..., 0)$ do
4:         $r \leftarrow 0$
5:         if $i = k-1$ then
6:             $r \leftarrow j \cdot \frac{j+1}{n+1} + \frac{1}{n+1} \sum_{h=j+1}^{n} h - c[i]$
7:         else
8:             $r \leftarrow M[i+1, j] \cdot \frac{j+1}{n+1} + \frac{1}{n+1} \cdot \sum_{h=j+1}^{n} M[i+1, h] - c[i]$
9:         $M[i,j] \leftarrow \max(j, r)$
10: return $M[0,0]$
```

Algorithm 2 Get optimal expected value ($\boldsymbol{O}(n \cdot k)$)

```
1: $M[k, n+1] \leftarrow 0$
2: $sums[n+1] \leftarrow 0$
3: for $i$ in $(k-1, ..., 0)$ do
4:     for $j$ in $(n+1, ..., 0)$ do
5:         $r \leftarrow 0$
6:         if $i = k-1$ then
7:             $r \leftarrow j \cdot \frac{j+1}{n+1} + \frac{1}{n+1} \cdot \frac{n\cdot(n+1)-j\cdot(j+1)}{2} - c[i]$
8:         else
9:             $r \leftarrow M[i+1, j] \cdot \frac{j+1}{n+1} + \frac{sums[j]}{n+1} - c[i]$
10:        $M[i,j] \leftarrow \max(j, r)$
11:        if $j = n$ then
12:            $sums[j] \leftarrow 0$
13:        else
14:            $sums[j] \leftarrow sums[j+1] + M[i, j+1]$
15: return $M[0,0]$
```

Exercise 2

First problem

Find the complete graph $\boldsymbol{G}$ of minimum weight given a weighted tree $\boldsymbol{T}$, such that $\boldsymbol{T}$ is the unique minimum spanning tree of $\boldsymbol{G}$.

Solution

Insert edges that are not in the tree so as to obtain the complete graph. These edges must have a greater weight than those of the tree, so that $\boldsymbol{T}$ is the only $\boldsymbol{MST}$ of $\boldsymbol{G}$. Since *Kruskal*'s algorithm uses the *Union-Find* structures for representing the cuts, we use this structures to solve the problem. Let's define with $\boldsymbol{V}$ the set of nodes of $\boldsymbol{T}$ and with $\boldsymbol{E}$ the set of edges of $\boldsymbol{T}$.

Algorithm 3 Find complete graph

```
1: for $v \in V$ do
2:     $v.initializeUnionFindSingleton()$
3: $T.sortEdgesByAscendingWeights()$                    $(\boldsymbol{O}(n \log n))$
4: $G \leftarrow \emptyset$
5: for $e \leftarrow (v_1, v_2) \in E$ do                $(\boldsymbol{O}(n^2))$
6:     $G.addEdge(e)$
7:     $set_1 \leftarrow v_1.findComponents()$
8:     $set_2 \leftarrow v_2.findComponents()$
9:     for $u \in set_1$ do
10:        for $v \in set_2$ do
11:            $\hat{e} \leftarrow (u, v)$
12:            if $\hat{e} \notin E$ then
13:                $\hat{e}.setWeight(e.getWeight() + 1)$
14:                $G.addEdge(\hat{e})$
15:    $union(set_1, set_2)$
16: return $G$
```

Cost: Let's define $\boldsymbol{E'}$ = set of edges of the graph $\boldsymbol{G}$, the cost is $\boldsymbol{O}(|E'| + |V| \log |V|)$, but $|E'| = \frac{|V| \cdot (|V|-1)}{2}$ and $|V| = n$, so $|E'| + |V| \log |V| = \frac{n^2-n}{2} + n \log n$ and the cost is $\boldsymbol{O}(n^2)$

Second problem

Find the total weight of the complete graph $\boldsymbol{G}$ of minimum weight given a weighted tree $\boldsymbol{T}$, such that $\boldsymbol{T}$ is the unique minimum spanning tree of $\boldsymbol{G}$.

Solution

This problem is quite similar to the previous one, but in this case there is no need to create all the edges of the graph.

Algorithm 4 Find weight of the complete graph

```
1: for $v \in V$ do
2:     $v.initializeUnionFindSingleton()$
3: $T.sortEdgesByAscendingWeights()$                    $(\boldsymbol{O}(n \log n))$
4: $w_{total} \leftarrow 0$
5: for $e \leftarrow (v_1, v_2) \in E$ do                $(\boldsymbol{O}(n))$
6:     $w_{total}$ += $e.getWeight()$
7:     $set_1 \leftarrow v_1.findComponents()$
8:     $set_2 \leftarrow v_2.findComponents()$
9:     $w_{total}$ += $(set_1.size() \times set_2.size() - 1) \times (e.getWeight() + 1)$
10:    $union(set_1, set_2)$
11: return $w_{total}$
```

Cost: $\boldsymbol{O}(|V| \log |V| + |E|) = \boldsymbol{O}(n \log n + (n-1)) = \boldsymbol{O}(n \log n)$, given by ordering the edges.

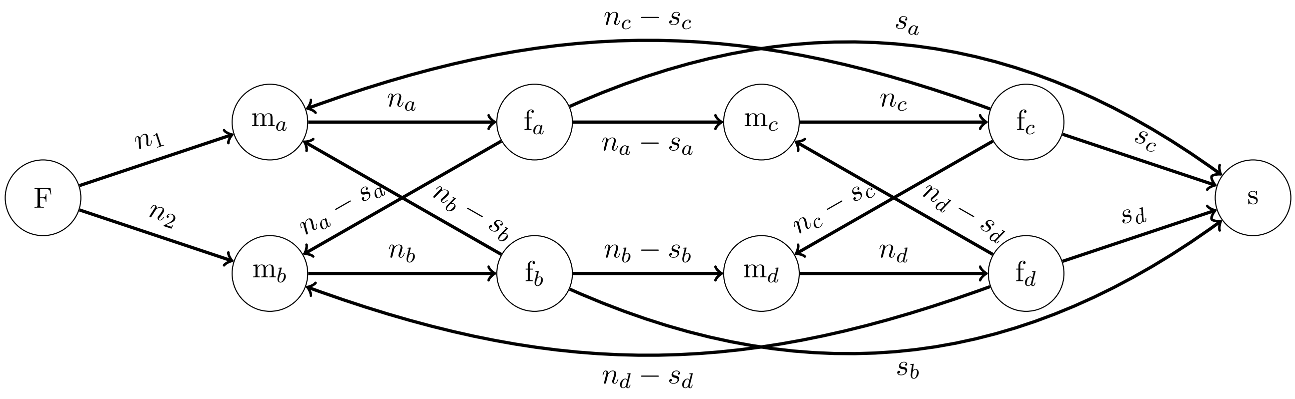
Exercise 3

First Problem

Model the problem as a flow problem in a graph G, only consisting of regular vertices, one source, one sink, and capacitated edges to find out how many chocolates Federico should actually make every week. Give a formal definition of your network, and also draw a small example, e.g. for $|F| = 4$.

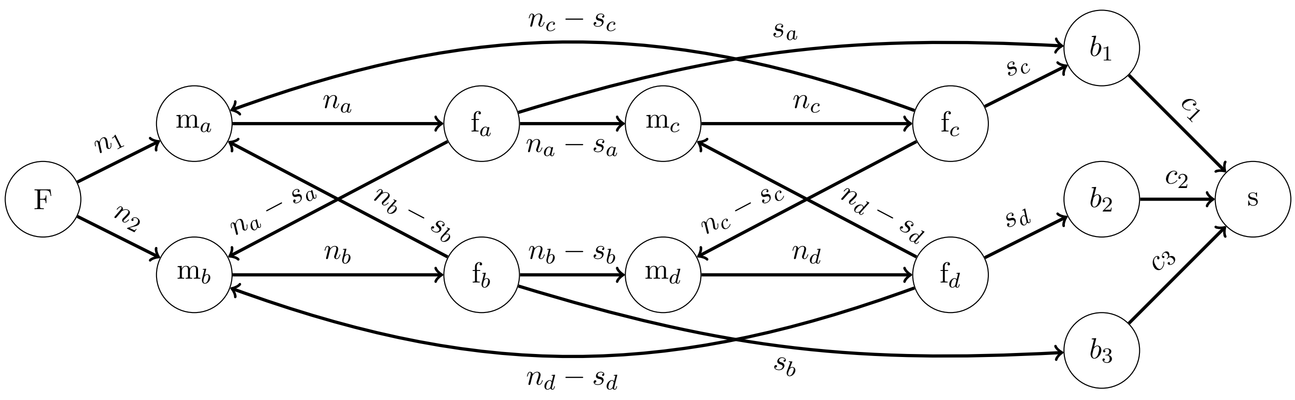
Solution

Example:



Second Problem

Solution



Exercise 4

Problem

Solution

In order to prove that this problem is **NP-HARD**, we need to find a well-know **NP-HARD** algorithm s.t. $\alpha \leq_p EX4$. We can do it with the **SUBSET SUM** problem. We can use a reduction from the problem above: given a set of integers $S = x_1, x_2, ..., x_n$ and a target integer γ, there exists a subset $S' \in S$ s.t. $\sum\limits_{x_i \in S'} x_i = K$ So, we construct an instance of the EX4 problem with n jobs, each having earliest start time 0 (so $s_j = 0$ for all $j \in S'$). For $j \in J$, job j has length $l_j = x_j$ and deadline $d_j = \gamma$. This instance solves the scheduling problem because we can arrange the jobs in any order and they will always meet their deadline. Now we need to prove