Algorithm Design - Homework 1

Sapienza University of Rome

Marco Costa, 1691388

December 4, 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

Algorithm 1 Get optimal expected value

$M[k, n+1] \leftarrow 0$
$c[k] \leftarrow 0$
$a[n+1] \leftarrow 0$
for i in $(k-1, ..., 0)$ **do**
    for j in $(n+1, ..., 0)$ **do**
        $nv \leftarrow 0$
        if $i = k-1$ **then**
            $nv \leftarrow j \times \frac{j+1}{n+1} + \frac{1}{n+1} \times \frac{n \times (n+1) - j \times (j+1)}{2}) - c[i]$
        else
            $nv \leftarrow M[i+1, j] \times \frac{j+1}{n+1} + \frac{1}{n+1} \times a[j] - c[i]$
        $M[i, j] \leftarrow \max(j, nv)$
        if $j = n$ **then**
            $a[j] \leftarrow 0$
        else
            $a[j] \leftarrow a[j+1] + matrix[i, j+1]$
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 2 Find complete graph

for $v \in V$ **do**
    $v.initializeUnionFindSingleton()$
$T.sortEdgesByAscendingWeights()$ $\quad (\boldsymbol{O}(n \log n))$
$G \leftarrow \emptyset$
for $e \leftarrow (v_1, v_2) \in E$ **do** $\quad (\boldsymbol{O}(n^2))$
    $G.addEdge(e)$
    $set_1 \leftarrow v_1.findComponents()$
    $set_2 \leftarrow v_2.findComponents()$
    for $u \in set_1$ **do**
        for $v \in set_2$ **do**
            $\hat{e} \leftarrow (u, v)$
            if $\hat{e} \notin E$ **then**
                $\hat{e}.setWeight(e.getWeight() + 1)$
                $G.addEdge(\hat{e})$
    $union(set_1, set_2)$
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 $|E'| = \frac{|V|(|V|-1)}{2}$ 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 3 Find weight of the complete graph

for $v \in V$ **do**
    $v.initializeUnionFindSingleton()$
$T.sortEdgesByAscendingWeights()$ $\quad (\boldsymbol{O}(n \log n))$
$w_{total} \leftarrow 0$
for $e \leftarrow (v_1, v_2) \in E$ **do** $\quad (\boldsymbol{O}(n))$
    $w_{total} \mathrel{+}= e.getWeight()$
    $set_1 \leftarrow v_1.findComponents()$
    $set_2 \leftarrow v_2.findComponents()$
    $w_{total} \mathrel{+}= (set_1.size() \times set_2.size() - 1) \times (e.getWeight() + 1)$
    $union(set_1, set_2)$
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.

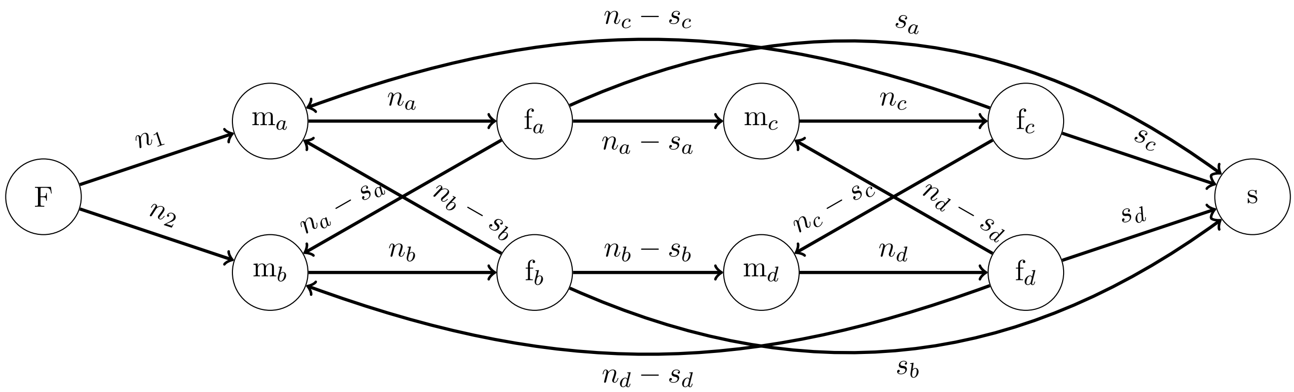
Exercise 3

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 $|F| = 4$.

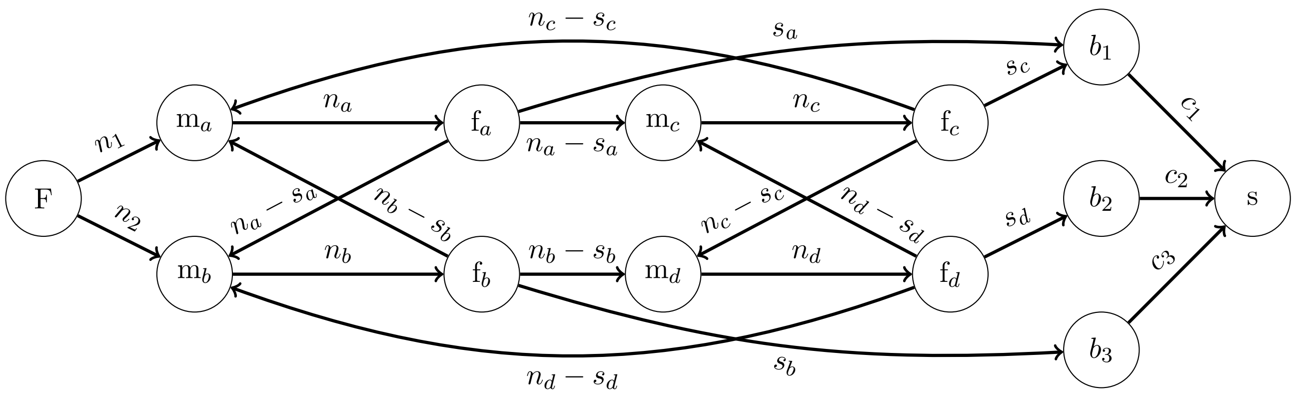
Solution

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. $\sum_{x_i \in S'} x_i = K$ 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