Algorithm Design - Homework 1

Sapienza University of Rome

Marco Costa, 1691388

December 5, 2019

Exercise 1

Problem

The problem consists in finding the expected optimal reward playing the game "*Open the Boxes and keep the Best!*". The input is composed by the number of boxes k, the number of different rewards n an the cost c_i of every box. Two algorithms are required. The first one's complexity must be $\boldsymbol{O}(n^2 \cdot k)$, while the second one's complexity must be $\boldsymbol{O}(n \cdot k)$

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 $\mathbf{M}[\mathbf{0}, \mathbf{0}]$ 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: $\sum_{i=j+1}^{n} i = \sum_{i=0}^{n} i - \sum_{i=0}^{j} i = \frac{n \cdot (n+1)}{2} - \frac{j \cdot (j+1)}{2} = \frac{n \cdot (n+1) - j \cdot (j+1)}{2}$;
- ***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 ($\boldsymbol{O}(n^2 \cdot k)$)

$M[k, n+1] \leftarrow 0$
for i in $(k-1, ..., 0)$ **do**
  for j in $(n+1, ..., 0)$ **do**
    $r \leftarrow 0$
    if $i = k-1$ **then**
      $r \leftarrow j \cdot \frac{j+1}{n+1} + \frac{1}{n+1} \sum_{h=j+1}^{n} h - c[i]$
    else
      $r \leftarrow M[i+1, j] \cdot \frac{j+1}{n+1} + \frac{1}{n+1} \cdot \sum_{h=j+1}^{n} M[i+1, h] - c[i]$
    $M[i, j] \leftarrow \max(j, r)$
return $M[0, 0]$

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

$M[k, n+1] \leftarrow 0$
$sums[n+1] \leftarrow 0$
for i in $(k-1, ..., 0)$ **do**
  for j in $(n+1, ..., 0)$ **do**
    $r \leftarrow 0$
    if $i = k-1$ **then**
      $r \leftarrow j \cdot \frac{j+1}{n+1} + \frac{1}{n+1} \cdot \frac{n \cdot (n+1) - j \cdot (j+1)}{2} - c[i]$
    else
      $r \leftarrow M[i+1, j] \cdot \frac{j+1}{n+1} + \frac{sums[j]}{n+1} - c[i]$
    $M[i, j] \leftarrow \max(j, r)$
    if $j = n$ **then**
      $sums[j] \leftarrow 0$
    else
      $sums[j] \leftarrow sums[j+1] + M[i, j+1]$
return $M[0, 0]$

Exercise 2

First problem

The problem is to 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 $\boldsymbol{G}$. Algorithm's run time must be polynomial in n.

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 ∈ V do                                                    (O(n))
2:      v.initializeUnionFindSingleton()
3:  T.sortEdgesByAscendingWeights()                                 (O(n log n))
4:  G ← ∅
5:  for e ← (v1, v2) ∈ E do                                         (O(n^3))
6:      G.addEdge(e)
7:      set1 ← v1.findComponents()
8:      set2 ← v2.findComponents()
9:      for u ∈ set1 do
10:         for v ∈ set2 do
11:             ê ← (u, v)
12:             if ê ∉ E then
13:                 ê.setWeight(e.getWeight() + 1)
14:                 G.addEdge(ê)
15:     union(set1, set2)
16: return G
```

Cost: $\boldsymbol{O}(|V|\log|V| + |E|\cdot|V|^2) = \boldsymbol{O}(n\log n + (n-1)\cdot n^2) = \boldsymbol{O}(n\log n + n^3 - n^2) = \boldsymbol{O}(n^3)$, given by the three nested cycles over n.

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 $\boldsymbol{G}$. Algorithm's complexity must be $\boldsymbol{O}(n\log n)$.

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 ∈ V do                                                    (O(n))
2:      v.initializeUnionFindSingleton()
3:  T.sortEdgesByAscendingWeights()                                 (O(n log n))
4:  w_total ← 0
5:  for e ← (v1, v2) ∈ E do                                         (O(n log n))
6:      w_total += e.getWeight()
7:      set1 ← v1.findComponents()
8:      set2 ← v2.findComponents()
9:      w_total += (set1.size() × set2.size() − 1) × (e.getWeight() + 1)
10:     union(set1, set2)
11: return w_total
```

Cost: $\boldsymbol{O}(|V|\log|V| + |E|\log|V|) = \boldsymbol{O}(n\log n + n\log n) = \boldsymbol{O}(n\log n)$ (*union* costs $\boldsymbol{O}(\log n)$).

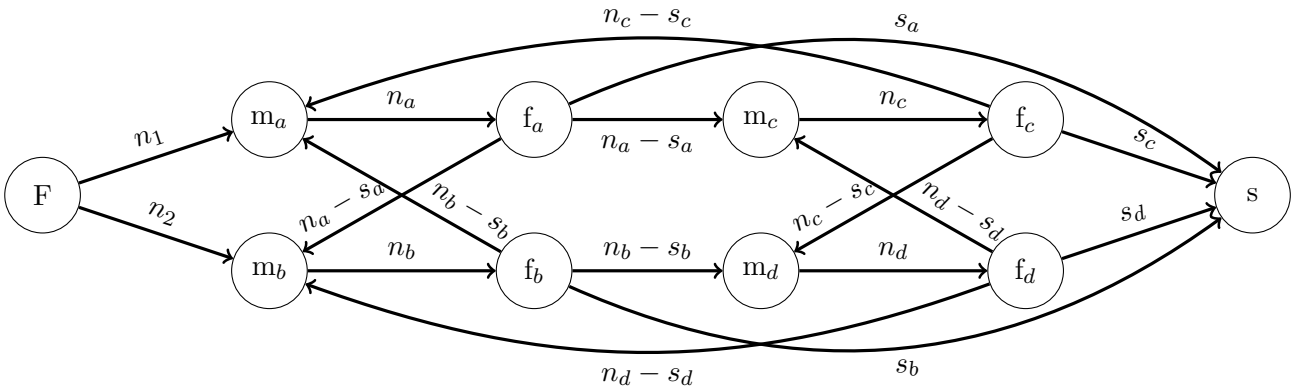
Exercise 3

First Problem

The problem consists in modeling Federico's business as a flow problem in a graph $\boldsymbol{G}$, only consisting of regular vertices, one source, one sink, and capacitated edges to find out how many chocolates he should make every week. An example with $|F| = 4$ is required.

Solution

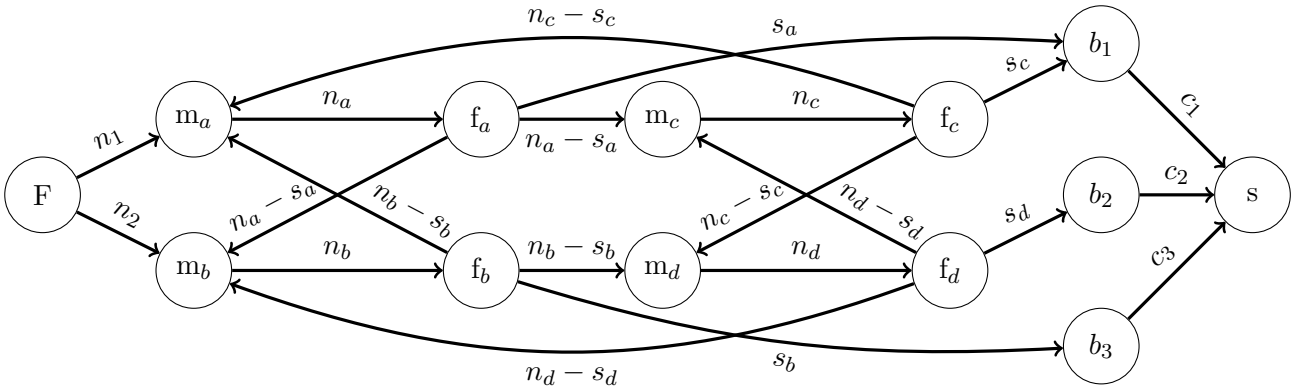
Example:



Second Problem

Adjust the network knowing that each friend will have an associated building $b_f \in B$ and find out if and how this impacts Federicos business.

Solution



Exercise 4

Problem

The problem consists in showing that it is not possible to solve the problem of scheduling n tasks with quick algorithm. Each job j of the n jobs has an earliest time s_j to start task, a deadline d_j when it has to be finished, and the length $l_j \in \mathbb{N}$ that specifies the time needed to complete it. There is also an upper bound time k and $\sum_{j \in J} l_j \leq k$.

Solution

In order to prove that this problem is not solvable with quick algorithm we need to prove that this problem is ***NP-Hard***. We can do it reducing another *NP-Hard* problem to the current problem. Let's define our problem as Θ, we need to find a well known *NP-Hard* problem Ω such that:

- $\omega \in \Omega \implies \theta \in \Theta$
- $\theta \in \Theta \implies \omega \in \Omega$ (or equivalently $\neg\omega \in \Omega \implies \neg\theta \in \Theta$)