# Solutions to Problem Set 4

Rashmi Dwaraka

November 2, 2016

**Collaborator: Manthan Thakar, Nakul Camasundaram, Thirthraj Parmar**

## Contents

# 1 Project management

**Problem statement:**
Suppose you are a high-level manager in a software firm and you are managing n software projects.You are asked to assign m of the programmers in your firm among these n projects. Assume that all of the programmers are equally competent.
After some careful thought, you have figured out how much benefit i programmers will bring to project j. View this benefit as a number. Formally put, for each project j, you have computed an array $A_j$[0..m] where $A_j$[i] is the benefit obtained by assigning i programmers to project j. Assume that $A_j$[i] is nondecreasing with increasing i. Further make the economically sound assumption that the marginal benefit obtained by assigning an $i_{th}$ programmer to a project is nonincreasing as i increases. Thus, for all j and $i \geq 1, A_j[i+1] - A_j[i] \leq A_j[i] - A_j[i1]$.

Design a greedy algorithm to determine how many programmers you will assign to each project such that the total benefit obtained over all projects is maximized. Justify the correctness of your algorithm and analyze its running time.

**Solution:**
Given the $(n, m)$ benefit matrix $A_j$[0..m] where $A_j$[i] is the benefit obtained by assigning i programmers to project j. Let us consider a graph with vertices numbered 0..$n$. Let the edge between each pair of vertices (u,v) be the maximum value of the benefit assigning (u-v) programmers to $j_{th}$ project i,e, for assigning (n-0) programmers the edge formed would be max($A_j$[n]) for each project j 1..$m$. Algorithm to build the required graph is as below:

---

**Algorithm: Build-Graph**$(G, n, m)$

---

```
1: procedure: Build-Graph(G, n, m)
2:    Let there be max-benefit[],max-project[] of size m
3:    for i ← 1 to n
4:       max-wt = 0, project-id = 1
5:       for j ← 1 to m
6:          if max-wt < A_j[i]
7:             max-wt = A_j[i]
8:             project-id = j
9:       max-benefit[i] = max-wt
10:      max-project[i] = project-id
11:   for i ← n to 1
12:      for j ← 0 to (i − 1)
13:         Add edge e {(i, j)} with weight as max-benefit[i − j]
```

---

---

**Algorithm: MAX-BENEFIT**$(G(V, E, w)$,**max-project[1..m])**

---

```
1: procedure: MAX-BENEFIT(G(V, E, w),max-project[1..m])
2:    Initialize d[v] = -∞ for all v in G.V
3:    let d[n] = 0 # let n be the source
4:    for each vertex u in G.V
5:       for each vertex v in G.Adj[u]
```

6: if d[v] < d[u] + w(u,v)
7: d[v] = d[u] + w(u,v)
8: v.p = u
9: project[v] = max-project[u-v]

---

**Proof of correctness**
The DAG formed in Build-Graph is topologically sorted because we build the graph such that the edges leaving from i to j will have the relation ($i > j \geq 0$). Modification to DAG-SHORTEST-PATH(G,w,s) presented in page 655 in CLRS to return longest path will give us the max-benefit allocation of programmers to respective project id

**Analysis of worst-case running time**

(1) In the Build-Graph algorithm, it takes $O(n \cdot m)$ running time to find maximum benefit for each (project)-(number of programmers) relation represented in line 2-10

(2) In the Build-Graph algorithm, it requires $O(n \cdot n)$ running time to build the DAG represented in line 11-13

(3) The MAX-BENEFIT algorithm runtime will be $O(n + m)$.

Based on above analysis, overall run-time will be $O(n \cdot m)$+$O(n \cdot n)$ which is approx max($O(n \cdot m)$,$O(n \cdot n)$)

## 2 Optimal tank capacity

**Problem statement:**
You are given a transportation network as an undirected graph G = (V, E), where V is a set of cities and E is a set of roads. Each road e ∈ E connects two of the cities, and you know its length $l_e$ in kms. You want to get from city s to city t. Theres one problem: your car can only hold enough gas to cover L kms. There are gas stations in each city, but not between cities. Therefore,you can only take a route if every one of its edges has length $l_e \leq L$.

(a) Given the limitation on your cars fuel tank capacity, show how to determine in linear time whether there is a feasible route from s to t.

(b) You are now planning to buy a new car, and you want to know the minimum fuel tank capacity that is needed to travel from s to t. Give an efficient algorithm to determine this.

**Solution:**

(a) Given the limitation on your cars fuel tank capacity, show how to determine in linear time whether there is a feasible route from s to t.

**Solution:**

– Given G = (V,E), where V is a set of cities and E is a set of roads. The algorithm takes in the Graph (G), Distance constraint (L), source city (s) and destination city (t).
By removing all the edges which have distance greater than L we ensure that if there is a path between s to t, then it is feasible path given the car's limitation. By Depth first search on the remaining graph from the source city we are able to reach t, then there is

a feasible path.

**Algorithm: Feasibility-Check**$(G, s, t, L)$

```
1: procedure: Feasibility-Check(G, s, t, L)
2:    For each e in G.E
3:       remove edges with w(e) < L
4:    Do DFS on G with s as source
5:    If t is visited, then return FEASIBLE
6:    else return NOT-FEASIBLE
```

- **Analysis of worst-case running time**

  (1) Removing the edges with weight less than L requires run-time of $O(|E|)$.

  (2) DFS on remaining graph has worst case run-time of $O(|V| + |E|)$.

  Overall, the worst case run-time of the algorithm is $O(|V| + |E|)$.

(b) You are now planning to buy a new car, and you want to know the minimum fuel tank capacity that is needed to travel from s to t. Give an efficient algorithm to determine this.

**Solution:**
Given G = (V,E), where V is a set of cities and E is a set of roads. The algorithm takes in the Graph (G), source city (s) and destination city (t).
We can modify the Dijkstra's RELAX algorithm (mentioned in pg 649 of CLRS) as below. The attribute d of t vertex will represent the minimum fuel capacity requirement of the new car.

**Algorithm: RELAX**$(u, v, w)$

```
1: procedure: RELAX(u, v, w)
2:        if v.d > max(u.d, w(u, v))
3:           v.d = max(u.d, w(u, v))
4:           v.p = u
```

- **Correctness of the Algorithm**
  Instead of relaxing the edges with the updated distance as in original Dijkstra's Algorithm, update the maximum distance edge found in that particular path. Eventually, by Dijkstra's shortest path logic, minimum of the maximum distance along each path will provide us the minimum fuel capacity required for the new car considering the map.

- **Analysis of worst-case running time** The worst case run-time of the algorithm is $O(|V|log|V| + |E|)$ by using Fibonacci heap data structure.

# 3 Consistency of constraints

**Problem statement:**
The following problem occurs in program analysis. For a set of n variables $x_1, ..., x_n$, you are given

a set of m constraints, of one of two forms: equality, of the form $x_i = x_j$ ; strict inequality, of the form $x_i > x_j$ . The goal of this problem is to determine whether a given set of constraints can be satisfied. For example, the following set of constraints can be satisfied

$$x_1 = x_2; x_1 = x_3; x_2 > x_4; x_3 > x_5$$

One can set x1 = x2 = x3 = 1, x4 = 0, and x5 = 0. The following set of constraints, however, cannot be satisfied.

$$x1 = x2; x1 = x3; x2 > x4; x4 > x5; x5 > x3$$

This is because the constraints yield the contradiction $x2 > x4 > x5 > x3 = x2$.
Give an algorithm that takes as input m constraints over n variables and determines whether the constraints can be satisfied. Your algorithm only needs to return a yes/no answer (yes if the constraints can be satisfied; and no otherwise).
Briefly justify the correctness of your algorithm. State and analyze its running time. The more efficient your algorithm is, in terms of its worst case running time as a function of n and m, the more credit you will get.

**Solution:**

- **Algorithm**
  **Step1**: Build the Maintained Connected Components (MCC) using the algorithm mentioned in CLRS section 21.3 Disjoint-set forests using the equality constraints.
  **Step2**: Build a graph using MCCs and inequality constraint.
  **Step3**: Do DFS: If there is back edge, return NO
  **Step4**: return YES

- **Proof of correctness**
  If there is a back edge while Depth first search indicates, that the graph includes a cycle. Cycle in the graph built indicates that the constraints contradict each constraints in the cycle and hence returns NO. If there are no cycles then it indicates that constraints are not contradicting each other and hence returns YES.

- **Analysis of worst-case running time**

  (1) Building the MCCs will require $O(m \log n)$

  (2) DFS requires $O(m + n)$

  The overall run-time is $O(m \log n)$

# 4 Network forensics

**Problem statement:**
You are the chief network administrator of a large corporate network, and have just been informed of a major virus infection in the network. The entire network has been shut down to prevent further damage. You and your forensics team get down to business and immediately start poring over the network logs to figure out what happened and when. Your network consists of n computers which we label $C_1, C_2$, through $C_n$. You are able to collect all communication information in the form of m triples of the form ($C_i$, $C_j$ , t), which means that $C_i$ and $C_j$ communicated with each other at time t.

You know that the virus first infected computer $C_1$ from an external source at time 0. From then on, there were no further infections from outside the network. But the virus is very infectious. So if $C_i$ communicated with $C_j$ at time t, and if one of them is infected prior to the communication, then the other will certainly be infected at time t.

Given the set of all communication triples, you would like to determine (a) which computers have been infected at the current time, (b) the precise times at which these computers were first infected, and (c) for each infected computer, the path by which the infection first reached the computer.

Design an efficient (time polynomial in n and m) algorithm to solve this problem, and analyze its worst-case running time. You may assume for convenience that all times listed in the communication triples are integers.

**Solution:**
Let the triples ($C_i$, $C_j$ , t) into a directed graph where the vertices represent the communicating computers and any edge $(u, v)$ indicates the computer $u$ with $v$ at time $w(u, v)$. Initialize all the vertices in the graph with color WHITE and set the $\pi$ as NIL and *time* attribute as 0. By performing DFS on the graph starting at computer $C_1$. At each iteration, each vertex is updated with parent attribute and time at which it was first infected.

---

**Algorithm: Forensic**$(G, u, t)$

---

1: for each vertex $u \in$ G.V
2: $u.color$ = WHITE
3: $u.\pi$ = NIL
4: $time = 0$

---

1: **procedure: Forensic**$(G, u, t)$
2: $time = time + t$
3: $u.time = time$
4: $u.color$ = GRAY
5: for each $v$ in $G.Adj[u]$
6: if $v.color ==$ WHITE
7: $v.\pi = u$
8: Forensic$(G, v, w(u, v))$
9: $u.color$ = BLACK

---

(a) All the vertices in the graph whose color $\neq$ WHITE represent the computers that are infected.

(b) The *time* attribute associated with each vertex of the graph represent the time when the computer was first infected.

(c) By tracing back $\pi$ attribute which indicates parent, we can find the path in which each computer is infected.

The runtime of the algorithm is $O(|V| + |E|)$

## 5 Uniqueness of MSTs when all weights are distinct

**Problem statement:**

(a) Suppose $T_1$ and $T_2$ are distinct minimum spanning trees for graph G. Let $(u, v)$ be the lightest edge (smallest weight edge) among all edges that are in $T_1$ and but not in $T_2$. Let $(x, y)$ be any edge that is in $T_2$ and not in $T_1$. Show that $w(x, y) \geq w(u, v)$.

(b) Prove that if the weights on the edges of a connected, undirected graph are distinct, then there is a unique minimum spanning tree.

**Solution:**

(a) Suppose $T_1$ and $T_2$ are distinct minimum spanning trees for graph G. Let $(u, v)$ be the lightest edge (smallest weight edge) among all edges that are in $T_1$ and but not in $T_2$. Let $(x, y)$ be any edge that is in $T_2$ and not in $T_1$. Show that $w(x, y) \geq w(u, v)$.

**Solution:**
Since, $(u, v)$ is the lightest edge (smallest weight edge) among all edges that are in $T_1$ and but not in $T_2$, we can assume without loss of generality that $(u, v)$ is not in $T_2$.
As $T_2$ is a minimum spanning tree, $T_2 \cup \{(u, v)\}$ will result in forming a cycle, say C. The cycle C has an edge $(x, y)$ whose weight is greater than the weight of $(u, v)$, since all the edges in $T_2$ with less weight are in $T_1$ by the choice of $(u, v)$. C must have at-least one edge that is not in $T_1$ by the definition of MST, otherwise $T_1$ would contain a cycle.
Replacing $(u, v)$ with $(x, y)$ in $T_1$ yields a spanning tree with a smaller total weight. This contradicts are statement that $T_1$ is a MST. Hence, $w(x, y) \geq w(u, v)$ must be true.

(b) Prove that if the weights on the edges of a connected, undirected graph are distinct, then there is a unique minimum spanning tree.

**Solution:**
Let us assume, that there are two distinct MSTs $T_1$ and $T_2$ for the undirected, connected graph G. Let there be an edge $(u, v)$ with the minimum weight in G. Let us consider, (u,v) as a part of $T_1$, but not in $T_2$. By adding $(u, v)$ to $T_2$ we would create a cycle C in $T_2$. Lets say an edge (x,y) is part of the cycle formed. Since, $(u, v)$ is the lightest and all the weights are distinct, w(x,y) > w(u,v). By replacing (x,y) with (u,v) in $T_2$ yields a new spanning tree T' with total weight less than T. This contradicts are assumption that T was a minimal spanning tree. Hence, there can be only one MST when the weights are distinct.