ESO207A: Data Structures and Algorithms		
Homework/Practice	Set	4b

Instructions.

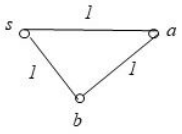
1. For each problem, write your name, Roll No., the problem number, the date and the names of any students with whom you collaborated.
2. For questions in which algorithms are asked for, give (a) a clear description of the algorithm in English and/or pseudo-code, (b) A proof/argument of the correctness of the algorithm, and, (c) an analysis of the running time of the algorithm.

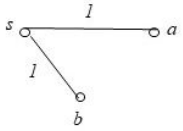
Full marks will be given only to correct solutions which are described clearly. Convoluted and unclear descriptions will receive *low marks.*

Problem 1. Let $G = (V, E)$ with an undirected graph with non-negative edge weights $w_e \geq 0$. Suppose T is a minimum spanning tree of G and G_π is a single source shortest path tree with root s (say, it is the result of running Dijkstra's algorithm from source s).

(a) Can T and G_π be the same? Can they be different? Give examples in each case.

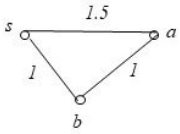
(a)

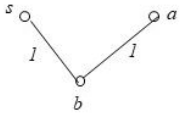




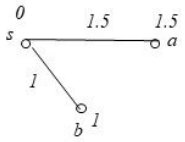
This tree is both MST and shortest path tree with source s.

(b)





Unique MST



Shortest Path tree from source s

Suppose each edge weight is incremented by 1: $w'_e = w_e + 1$.

(b) Does the minimum spanning tree change? Give an example where it changes or prove that it does not change.

A spanning tree has n vertices and $n-1$ edges. If the weight of each edge is increased by some value v, the weight of each spanning tree increases by $(n-1)v$. Hence, the minimum spanning tree prior to the increase continues to be minimum among all spanning trees after the increase.

(c) Does the shortest path tree change? Give an example where it changes or prove that it does not change.

The shortest path tree can change. The cost of a path with k edges increases by k, so the increase is not uniform across the paths. An example is given in Figure 1.

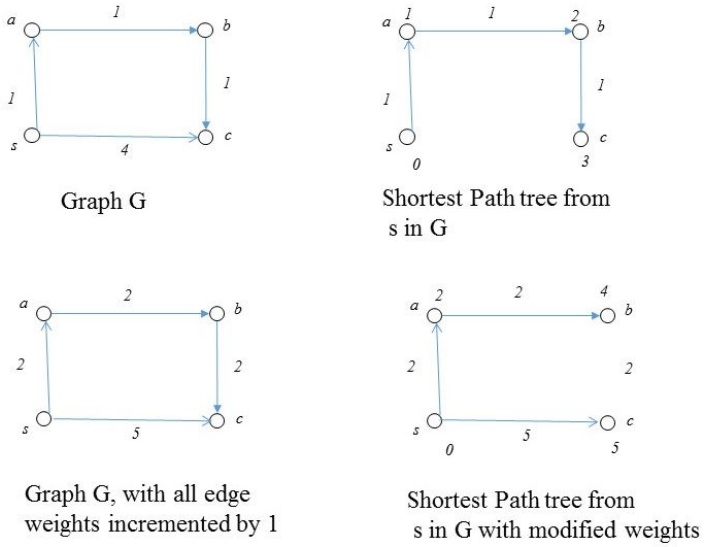


Figure 1: Shortest paths tree can change if edge weights are incremented by 1

Problem 2. Let $G = (V, E)$ be an undirected graph with all *distinct* edge weights. Prove that it has a unique minimum spanning tree.

Let $(S, V-S)$ be a non-trivial cut, that is, $S \neq \phi$ and $V - S \neq \phi$. Then we can claim that the lightest edge $e = \{a, b\}$ crossing the cut is included in every minimum spanning tree.

Proof of Claim: Suppose there is a minimum spanning tree T that does not include e. Then, consider $T \cup \{e\}$. Then the edge e is now part of a cycle, since in T, there already was a path from a to b. Consider this path in T from a to b. It starts in S and ends in $V - S$, hence, there is an edge e' such that e' crosses the cut. (The path may cross the cut back and forth an odd number of times). If we delete $e' = \{u, v\}$ say from $T \cup \{e\}$ the vertices u and v still remain connected, since deleting an edge from a cycle still leaves it connected. The graph $T' = T \cup \{e\} - \{e'\}$ is therefore connected, has $n-1$ edges (since T had $n-1$ edges, and we added and deleted one edge each), and hence is a tree. Further, it is a spanning tree, since all vertices are part of its vertex set. But $cost(T') < cost(T)$ since,

$$cost(T') = cost(T) + w(e) - w(e')$$

and $w(e) < w(e')$ since we assumed e to be the lightest edge crossing the cut and all edge weights are

distinct, so $w(e) \neq w(e')$. But this contradicts the assumption that T is an *MST*. Hence, we conclude that e is included in T. This proves the claim.

Now let us prove that there is a unique minimum spanning tree. Assume to the contrary, and let T and T' be both *MST*s. Since, T and T' both have the same number of edges, each has at least one edge that the other does not have. So T has an edge $e = \{a, b\}$ that T' does not have. If we delete e from T (but keep the vertices), then T gets disconnected into two components, call the set of vertices in one component S and the other $V - S$. Say a is in S and $b \in V - S$. By the claim above, e is the lightest edge crossing this cut, because, if there was a lighter edge crossing the cut, it would be included in T by the claim, but then T would have a cycle. Since e is the lightest edge crossing this cut, it must be included in T', by the claim above. Hence, the set of edges of T are a subset of the set of edges of T'. But T and T' have the same set of edges. Thus, $T = T'$, that is the minimum spanning tree is unique.

Problem 3. Let $G = (V, E)$ be an undirected graph with non-negative (real valued) edge weights. Many of these edge-weights are integral. You have to design an efficient algorithm to find the *minimum integral spanning tree* assuming one exists. An integral spanning tree is a set T of edges that forms a spanning tree and the edge-weights of all edges in T are integral. A minimum integral spanning tree is an integral spanning tree whose cost (sum of edge-weights in T) is no larger than the cost of every integral spanning tree. The algorithm should find the minimum integral spanning tree if one exists and should return *NO* if there is no integral spanning tree.

The question assumes that there exists an integral spanning tree, and hence a minimum integral spanning tree. We start by modifying the cut-property.

Modified Cut Property: Let X be a set of edges with integral weight that is part of an integral *MST*. Let $(S, V - S)$ be any cut and suppose that (1) no edge of X crosses the cut, and (b) e is the minimum integral weight edge crossing the cut. Then, there is an integral *MST* including $X \cup \{e\}$.

Remark: Suppose there is a non-trivial cut $(S, V - S)$ with no integral weight edge crossing it. Then, there is no integral spanning tree. Why? Suppose there was an integral spanning tree T. Then, there would at least one edge in T crossing this cut, and all those edges would have integral weight.

Proof of the modified cut property. Let T be an integral *MST* containing the edges of X. Let $e = \{a, b\}$ be the lightest integral weight edge crossing the cut. Suppose T does not include e. Then, let $T' = T \cup \{e\}$. Since T is a tree, this creates a cycle involving edge e. Since, e crosses the cut, there is another edge e' on this cycle that crosses the cut, call it e'. (In fact there may be an odd number of edges in this cycle that cross this cut). Note that since e' is an edge of T, it has integral weight. Consider $T' = T \cup \{e\} - \{e'\}$. Then,

$$cost(T') = cost(T) + w(e) - w(e')$$

Since, e is the lightest integral weight edge crossing the cut and e' crosses the cut, $w(e') \geq w(e)$, and hence,

$$cost(T') \leq cost(T) \ .$$

Since we had assumed that T is the integral *MST*, it follows that $cost(T') = cost(T)$ and hence that T' is an integral *MST*. This proves the claim.

Problem 4. [CLRS 16-1] Consider the problem of making change for n Rupees (n positive integral) using the fewest number of coins with denominations 10, 5, 2 and 1.

(a) Describe a greedy algorithm for this specific problem. Prove that your algorithm yields an optimal

solution.

(b) Give a collection of coin denominations where the greedy algorithm does not yield an optimal solution. The collection must include the Re 1 coin so that there is a solution for every value of n.

We can do (b) first. Let the denominations be $\{1, 3, 4\}$. For changing 6, the greedy method yields $6 = 4 + 1 + 1$ requiring 3 coins. The optimal is $6 = 3 \times 2$ i.e., 2 coins. So greedy is not optimal.

(**a**). First consider the set of denominations, $\{1, 2, 5, 10\}$, denoted, $\{d_1, d_2, d_3, d_4\}$, $d_1 = 1, d_2 = 2, d_3 = 5$ and $d_4 = 10$. The problem is to make change for a positive integer n, using x_i coins of denomination d_i so that $\sum_{i=1}^{4} x_i d_i = n$ and $\sum_{i=1}^{4} x_i$ is minimized. The problem can be written as

$$\text{minimize} \sum_{i=1}^{4} x_i$$

$$\text{subject to} \sum_{i=1}^{4} x_i d_i = n$$

$$\text{and } x_i \text{ are non-negative integers, for } i = 1, \ldots, 4.$$

To change an amount n, in the optimal solution, there can be at most 1 coin of Re 1, since, if there are two coins of Re 1, they can be replaced by 1 coin of Rs. 2. Similarly, there can be at most 2 coins of Rs. 2 in the optimal solution, otherwise, if there are 3 coins of Re 2, then it can be replaced by $5 + 1$, that is, 2 coins. Similarly, the optimal solution has at most 1 coin of Rs 5. So, we have shown that $x_1 \leq 1$, $x_2 \leq 2$ and $x_3 \leq 1$. However, all three conditions cannot be strictly met, since, if $x_1 = 1, x_2 = 2$ and $x_3 = 1$, then, $\sum_i d_i x_i = 10$ and the optimal solution will use a Rs. 10 coin. Hence, in the optimal solution, we have,

$$x_1 \leq 1$$
$$x_2 \leq 2$$
$$x_3 \leq 1$$

and at least one of these inequalities is strict. Hence, the amount covered by denominations, 1,2, and 5 is at most 9. Thus, the number of Rs. 10 notes in the greedy solution matches that in the optimal solution. (The number of Rs. 10 notes in the greedy solution is $\lfloor n/10 \rfloor$.

Without loss of generality, now we can assume that $1 \leq n \leq 9$. We can now show that the number of Rs. 5 notes in the optimal solution matches the greedy solution. By following a similar argument, we have,

$$x_1 \leq 1$$
$$x_2 \leq 2$$

and at least one of these inequalities is strict. Thus, the total amount covered by denominations 1 and 2 is at most 4. Hence, the number of Rs. 5 denominations in the greedy solution $\lfloor n/5 \rfloor$ matches the optimal solution (i.e., optimal solution has exactly 1 Rs. 5 coin if $n \geq 5$ and has 0 Rs. 5 coins otherwise.)

The same argument generalizes for the Rs. 2 coin. Hence, the greedy solution for the coin changing works for these denominations.

A general dynamic programming solution.

Input is an array $d[1 \ldots, k]$ of denominations and an integer W that has to be changed using the minimum number of coins of the given denominations.

Let $C[n]$ denote the minimum number of coins under the given denominations required to change n. Then, $C[0] = 0$. We obtain the recurrence equation

$$C[n] = 1 + min\{C[n - d_j] \mid d_j \leq n, j = 1, 2, \ldots, k\}$$

This can be computed bottom up as follows. The precise sequence of coins in remembered using the L array

1. $C[0] = 0$
2. **for** $n = 1$ to W
3. $C[n] = \infty$
4. **for** $j = 1$ to k
5. **if** $n \geq d[j]$ **and** $C[n] > 1 + C[n - d_j]$
6. $C[n] = 1 + C[n - d_j]$
7. $L[n] = j$

The algorithm takes time $O(kW)$ and is a pseudo-polynomial algorithm, not a polynomial algorithm. The optimal change found can be printed out as follows.

1. **for** $j = 1$ **to** k
2. $X[j] = 0$
3. $n = W$
4. **while** $n > 0$
5. $X[L[n]] = X[L[n]] + 1$
6. $n = n - d[L[n]]$
7. **for** $j = 1$ **to** k
8. print $X[j]$ coins used of denomination $d[j]$