ESO207: Data Structures and Algorithms *Time : 90min.*

Quiz: 3 *Max. Marks: 100*

Instructions.

a. Please answer all parts of a question together. You may leave blank spaces if you wish to return later to complete parts of the question.

b. The exam is closed-book and closed-notes. Collaboration of any kind is ***not*** permitted. You may not have your cell phone on your person. Please use the mounted clock in the rear of the examination hall for time reference.

c. You may cite and use results done in the class.

d. Grading will be based not only on the correctness of the answer but also on the justification given and on the clarity of your arguments. Always argue correctness of your algorithms or conclusions.

Problem 1. (50)

(a) Give an example of an undirected graph $G = (V, E)$ with non-negative edge weights such that the shortest path tree (from some source vertex) is not a minimum spanning tree. (8)

See Figure 1(a).

(b) Give an example of an undirected graph $G = (V, E)$ with non-negative edge weights such that the shortest path tree (from some source vertex) is a minimum spanning tree. (8)

See Figure 1(b).

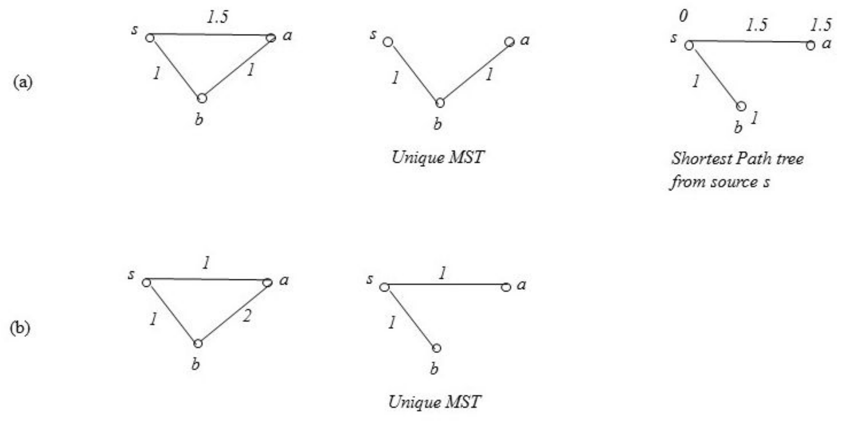


Figure 1: (a) MST is not SPT and (b) MST is SPT

(c) Let G be an undirected weighted graph with a cycle that has a unique heaviest edge e. Then, e is not part of any minimum spanning tree. Prove this statement or give a counterexample. (8)

Soln. . Suppose $e = \{a, b\}$ is the unique heaviest edge of some cycle C and e is part of an MST T.

Take T and temporarily delete e. This disconnects the graph into two connected subsets, S and $V - S$, say $a \in S$ and $b \in V - S$. Call this cut $(S, V - S)$. We will refer to only this cut below.

Since C is a cycle with edge e crossing the cut, there is at least one edge $e' \in C$ and different from C that also crosses the cut. Also, since e is the unique heaviest edge crossing the cut, $w(e') < w(e)$.

Claim 1: In $T \cup \{e'\}$, there is a cycle containing both e and e'.

Proof of Claim: Now consider $T \cup \{e'\}$. By adding e' to T, we get this cycle C'. So C' contains e'. Since e' crosses the cut, C' must have another edge crossing the cut. In T, e was the only edge crossing the cut(why? by construction of the cut $(S, V - S)$) so, C' must include e as well.

Now, delete e' from C', that is, consider

$$T' = T \cup \{e'\} - \{e\}$$

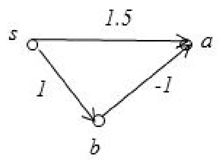
We have deleted an edge e from a cycle C', so the graph T' is still connected. T' has $n - 1$ edges, so it is a spanning tree. Moreover,

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

since, $w(e) > w(e')$.

Thus the cost of T' is less than that of T. Thus, we have a spanning tree that has smaller cost than that of the assumed minimum spanning tree T. This is a contradiction. Hence, e cannot be part of the any minimum spanning tree.

(d) Give an example of a directed graph $G = (V, E)$ with possibly negative edge weights but no negative weight cycle, and a source vertex $s \in V$ such that Dijkstra's algorithm when run on G from s does not give the shortest path from s to all reachable vertices. Argue your answer. (8)



(e) Give an example of a directed weighted graph $G = (V, E)$ such that if Bellman-Ford is run on this graph, then upon termination, there is at least one edge (u, v) that is *not* on a negative cycle but reachable from the source via a negative cycle, that violates the triangle inequality, that is, $v.d > u.d + w(u, v)$. You may choose the source vertex and the ordering of the edges appropriately. (8)

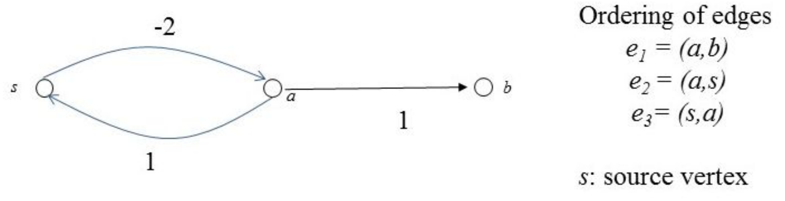


Figure 2: Graph for running the Bellman Ford algorithm

Consider the graph given in Figure 2.

The table of relaxations in the Bellman Ford algorithm is given below.

	$s.d$	$a.d$	$b.d$
Initially	0	∞	∞
Relax(a, b)	0	∞	∞
Relax(a, s)	0	∞	∞
Relax(s, a)	0	1	∞
Relax(a, b)	0	1	2
Relax(a, s)	-1	1	2
Relax(s, a)	-1	0	2

Now check for triangle inequality for each edge

Edge	Condition that holds	Conclusion
(a, b)	$b.d > a.d + w(a, b)$	$\triangle$ inequality fails
(a, s)	$s.d > a.d + w(a, s)$	$\triangle$ inequality fails
(s, a)	$a.d \leq s.d + w(s, a)$	$\triangle$ inequality holds

In particular, (a, b) is not on a negative cycle, but is reachable from s via a negative cycle, and fails to satisfy the triangle inequality.

(f) Suppose a government runs currency coins of k denominations $1, 2, 4, 8, \ldots, 2^{k-1}$. Does the greedy coin change strategy work optimally for this set of denominations? Prove your answer. The greedy strategy for obtaining coin change for n is the following. (Choose the largest denomination coin less than or equal to n, and use as many coins of this denomination possible to get as close to but not greater than n, recurse similarly on the remaining amount.) (10)

Soln. Let $n = b_l b_{l-1} \ldots, b_k b_{k-1} \ldots, b_1$ be the binary representation of n, which is unique.

Let $x_{k-1} = \sum_{j=k-1}^{l} 2^{j-(k-1)} b_j = \lfloor n/2^{k-1} \rfloor$. In any optimal scheme, there can be at most 1 coin of denomination 2^j, for $0 \leq j < k-1$, because if there are 2 coins of denomination 2^j, the two of them can be replaced by a single coin of denomination 2^{j+1}. Now, even if there is 1 coin of each denomination 2^j used by an optimal solution, for $j = 0, 1 \ldots, k-2$, the sum of its value will be $\sum_{j=0}^{k-2} 2^j = 2^{k-1} - 1$. Hence, the number of coins of denomination 2^{k-1}

used by any optimal solution matches that of the greedy solution. So let $n' = n - n_0$, where, $n_0 = 2^{k-1}x_{k-1}$ is the part of n covered by the 2^{k-1} value denominations.

The remainder n' is represented in binary by $b_{k-2}, \ldots, b_1$, which is unique, it cannot have multiple representations.

(g) Knapsack. (25)

A robber enters a store that has n items with values Rs. $v_1, v_2, \ldots, v_n$ and weights $w_1, w_2, \ldots, w_n$ (in kg.). The robber has a knapsack that can carry a weight of at most W kg. The n items are all distinct and there is exactly one copy of each item in the store (no repetitions). What is the set of items that the robber should put in his knapsack so as to maximize the value of the items picked up. That is,

$$\begin{aligned} \textit{Maximize} \ & \sum_{i=1}^{n} x_i v_i \\ \text{subject to} \ & \sum_{i=1}^{n} x_i w_i \leq W \\ & \text{and } x_i \in \{0,1\}, \text{ for } i = 1, \ldots, n \end{aligned}$$

1. Let $K[w, j]$ denote the maximum value that a knapsack with capacity of weight w can contain using the first j items. Design a recurrence equation for $K[w, j]$. (12)

   Soln. The optimal solution for knapsack capacity w using the items $1, \ldots, j$ either picks up item j or it doesn't. If it picks up item j (and this can happen only if $w \geq w_j$) the knapsack has remaining capacity $w - w_j$ and must be filled optimally using items $1, \ldots, j-1$. This option has optimal cost $K(w - w_j, j-1) + v_j$. The other option is that item j is not picked. Then, the knapsack with capacity w must be optimally filled using items $1, \ldots, j-1$. The better of the two options is the optimum, that is, we get,

   $$K[w, j] = \max(K[w - w_j, j-1] + v_j, K[w, j-1])$$

   with the boundary conditions: $K[x, 0] = 0$, for all x negative and positive and $K[0, j] = 0$, for each $j \geq 0$.

2. Write pseudo code to compute $K[W, n]$. (9)

```
1.  $K[0,\ldots,n][0\ldots w]$ is a two dimensional array
2.  $\pi[0,\ldots,n][0\ldots w]$ is used to reconstruct the optimal solution.
3.  **for** $x = 0$ to $w$   $K[x,0] = 0$
4.  **for** $j = 0$ to $n$   $K[0,j] = 0$
5.  **for** $x = 1$ to $w$
6.      **for** $j = 1$ to $n$
7.          $K[x,j] = K[x,j-1]$
8.          $\pi[x,j] = 0$
9.          **if** $x \geq w_j$
10.             **if** $K[x,j] < K[x-w_j,j-1] + v_j$
11.                 $K[x,j] = K[x-w_j,j-1] + v_j$
12.                 $\pi[x,j] = 1$
```

Students are note required to write the code corresponding to reconstructing the optimal solution using π. This is given for completeness here.

$\pi[x, j] = 0$ means that the optimal solution constructed for $K[x, j]$ does not use item j, and $\pi[x, j] = 1$ means the opposite. The optimal solution deduced is computed as follows. It uses the π array computed earlier.

```
// Reconstruct the computed optimal solution for $K[w, n]$.
1.  $j = n$
2.  $x = w$
3.  while $x > 0$ and $j > 0$
4.      if $\pi[x, j] == 1$
5.          print " $j$ " // include item $j$
6.          $x = x - w_j$
7.      $j = j - 1$
```

3. Analyze the complexity of your code. Is it polynomial time? (4)

   The complexity is $O(nw)$. It is not polynomial time, a polynomial time algorithm would have time complexity $O((n \log w)^c)$ for some constant c.

Problem 2. Concrete Roads MST

You are given a road network $G = (V, E)$ where the vertices are cities and the roads are bidirectional pairs of cities. Assume that G is connected and all roads are bi-directional. Associated with each road segment $\{a, b\}$, between two cities a and b, the distance of this road segment is given as $w(a, b)$. Further, each road segment $\{a, b\}$ is labeled to be exactly one of *gravel* or *concrete*. The problem is design an efficient algorithm to find a subset T consisting only of the *concrete* labeled road segments in E that form an acyclic connected graph over V and has the minimum cost $cost(T) = \sum_{e \in T} w(e)$ among all such subsets T' consisting of only *concrete* labeled road segments in E and forms an acyclic connected subgraphs T' over V. The cost of a subset $T' \subset E$ is defined as $cost(T') = \sum_{e \in T'} w(e)$. Design an $O(|E| \log |V|)$ solution. Your algorithm should be able to terminate and say *NO* (or give cost ∞) if there is no subset of edges $T \subset E$ that forms acyclic connected subgraph. You *must* argue the correctness of the algorithm. (25)

Soln. Let E_c = the set of concrete edges of G. Let $G_c = (V, E_c)$ be the subgraph of G consisting only of concrete edges. The problem is to find the *MST* in G_c. So one can use Kruskal's algorithm or Prim's algorithm to find the *MST* in G_c. If either of these algorithms fail to produce an *MST*, the algorithm returns *NO MST*.

The following sequence of questions is supposed to enable you to design an algorithm. You may or may not use it.

(a) Define a concrete minimum spanning tree T to be a set of edges that are labeled only concrete and T forms an acyclic connected subgraph over V (i.e., it is a spanning tree).

(b) Modify the cut-property to make a statement about the lightest concrete edge crossing a non-trivial cut. A cut $(S, V - S)$ is non-trivial if $S \neq \{\}$ and $V - S \neq \{\}$.

This is the same cut-property now applied to a set X of concrete edges.

This is for pedagogical purposes only. Claim: Let X be a set of concrete edges that is part of some concrete *MST*. Let $(S, V - S)$ be a non-trivial cut such that none of the edges of X

cross the cut. Let e be a lightest concrete edge crossing the cut. Then, $X \cup \{e\}$ is part of some concrete *MST*.

Proof: Since X is part of some concrete *MST*, let this concrete *MST* be denoted by T. Then, if e is part of T we are done. Otherwise, suppose e is not part off T. Then, by adding e to T, we get a cycle C that contains e. This cycle has another edge e' crossing the cut (since e is part of the cycle and it crosses the cut). This edge being part of T is concrete. Consider the graph obtained by deleting e' from $T \cup \{e\}$.

$$T' = T \cup \{e\} - \{e'\}$$

BY deleting e' from a cycle does not disconnect the cycle, and hence does not disconnect the graph $T \cup \{e\}$. So T' is connected and has $n-1$ edges. It is therefore a tree. The cost of T' is

$$\mathit{cost}(T') = \mathit{cost}(T') + w(e) - w(e')$$

and since $w(e) \leq w(e')$, $\mathit{cost}(T') \leq \mathit{cost}(T)$. But T was assumed to be an *MST*. So T' is also a concerte *MST* and it contains e.

This proof is identical to the cut-property for *MST*, and applied only to the graph $G_c = (V, E_c)$.

(c) Modify Prim's or Kruskal's algorithm according to the modified cut-property. Argue the correctness. *This is for pedagogical purposes only.*

Based on the cut property, Kruskal's algorithm can work by considering only the concrete edges in non-decreasing order of weight. Similarly, Prim's algorithm can proceed by extending the current minimum subtree over vertices S to include the lightest concrete edge crossing the cut $(S, V-S)$. These run in time $O(|E_c| \log |V_c|)$.

Correctness of Kruskal's algorithm: It is clear that Kruskal's algorithm produces an acyclic graph at each iteration: a collection of trees, i.e., a forest. Upon termination, one of two possibilities may happen. The forest is a single tree, or, the forest has multiple trees. If the forest is a single tree, Kruskal has found a spanning tree.

If not then there are at least two trees T_1 and T_2 that are disconnected. Let S be the set of vertices of T_1. Then, we claim that there is no edge crossing the cut $(S, V-S)$ in G. For sake of contradiction, suppose there was an edge crossing this cut. Let e be the lightest edge crossing the cut, and the earliest among those crossing the cut in the sorted ordering of Kruskal. Then, in the ordering of edges, e would have been considered by the algorithm in due course. Now e is not included in the final set of edges. This means that by including e, a cycle was created. This cycle would involve another edge crossing the cut, call it e'. Then, e' is already included by Kruskal, so e' is at least as light as e and appears before e in the ordering of edges. But we assumed that e occurs earliest in the ordering among all the lightest edges crossing the cut. Hence, we have a contradiction. That is, there is no such lightest edge e crossing the cut, that is, there is no edge e crossing the cut. Since the cut is non-trivial, there is no spanning tree, as every spanning tree has at least one edge crossing every non-trivial cut.

Now suppose Kruskal's algorithm produces a single spanning tree T. We have to show that it is an *MST*. The proof is by induction on the number of edges included. The base case is when the first edge (with least weight) is included. Let $e = \{a, b\}$ be this edge. Consider the cut $(\{a\}, V - \{a\})$. Being the globally least cost edge, e is a lightest edge crossing this cut.

Now suppose e_{i+1} is the $i+1$th edge included. Let X_i be the set of the first i edges included. to which e_{i+1} is added. At the time e_{i+1} was included, it merged two subtrees t_1 and t_2. Let S be the set of vertices in t_1. Then, we claim that e_{i+1} is a lightest edge that crosses the cut $(S, V-S)$ and X_i respects the cut, that is, no edges of X_i cross the cut (the latter is clear by construction). If e_{i+1} is indeed a lightest edge crossing this cut, then by the cut property there is an MST containing $X_i \cup \{e_{i+1}\}$, and we are done.

Suppose it is e' that is the lightest edge crossing this cut and is the earliest to occur in the Kruskal's edge ordering among all the equally lightest edges crossing this cut. So $w(e') < w(e_{i+1})$. So in Kruskal's algorithm e' was encountered before e_{i+1}. But e' was not included by the algorithm. That is because e' caused a cycle, which is possible only if both end points of e' are in S or both in $V-S$, that is, it is not a cut edge. This is a contradiction. So, e_{i+1} is indeed a lightest edge crossing the cut. By the cut property there is an MST containing $X_i \cup \{e_{i+1}\}$, and we are done.