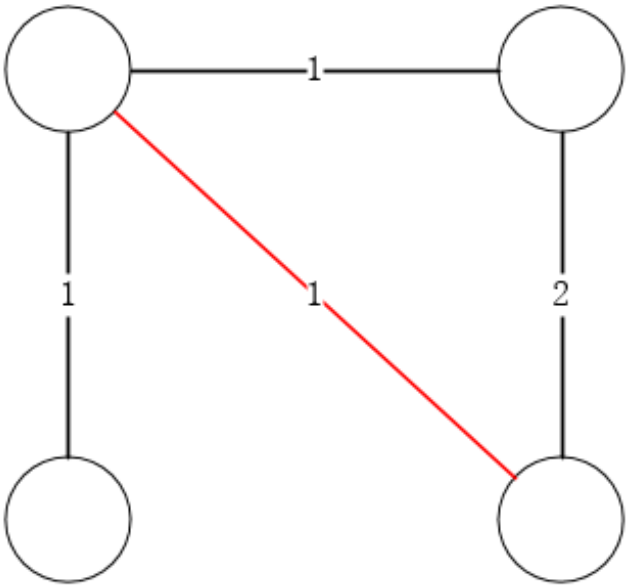
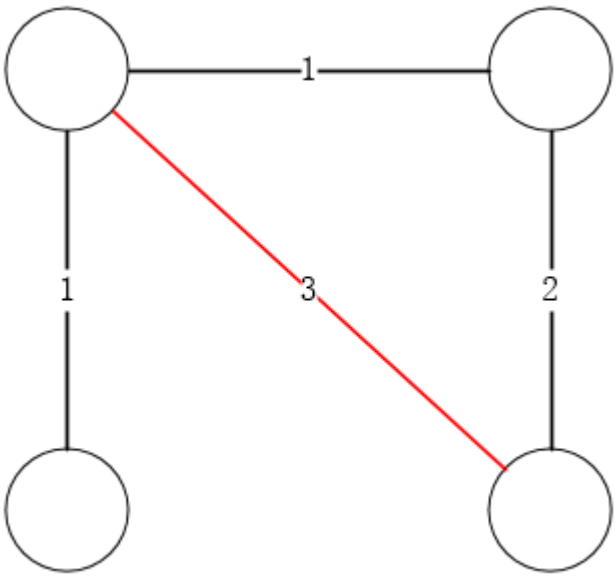
Homework 5 Solution

1. *Let T be a minimum spanning tree of G (where G is an edge-weighted graph). Suppose the cost of a certain edge e_0 in G has become changed. Give an example of G, edge weight $w(e)$ for each e in G, T, e_0, and the new weight $w'(e_0)$ such that T is no longer a minimum spanning tree of G after the weight of e_0 becomes $w'(e_0)$.*

Original G, red edge is e_0



Altered G, red edge is e_0



2. *The prim's algorithm works correctly when an arbitrary vertex is chosen as a start vertex. Assume that there is only one edge say* e_1 *whose weight is smaller than any other edges. Prove that* e_1 *is always included in a tree generated by the Prim's algorithm regardless which vertex is chosen as a start vertex.*

Let e_1 is not included in the MST T generated by Prim's algorithm. By adding e_1 into the MST will form a cycle. Remove the edge with largest weight in the cycle, then we have a new MST T'. Noted that e_1 has the smallest weight among all edges. Thus, the cost of T' is smaller than T. However, Prim's algorithm is proved to be correct and should be always giving a MST. Contradiction.
Thus e_1 must be included in the Prim's algorithm output.

3. *The input to this problem consists of an ordered list of* n *words. The length of the* i*th word is* w_i*, that is the* i*th word takes up* w_i *spaces. (For simplicity assume that there are no spaces between words.) The goal is to break this ordered list of words into lines, this is called a layout. Note that you cannot reorder the words. The length of a line is the sum of the lengths of the words on that line. The ideal line length is* L*. Assume that* $w_i \leq = L$ *for all* i*. No line may be longer than* L*, although it may be shorter. The penalty for having a line of length* K *is* $L - K$*. Consider the following greedy algorithm.*

For $i = 1$ *to* n
Place the i*th word on the current line if it fits*
Else place the i*th word on a new line*

(a) *The overall penalty is defined to be the sum of the line penalties. The problem is to find a layout that minimizes the overall penalty. Prove or disprove that the above greedy algorithm correctly solves this problem.*

Proof:
Let S denote the solution generated by greedy algorithm. Let S' be an optimal solution other than greedy algorithm. By trying to move the first word of S' each line into the previous line will not change the overall penalty. Thus S is as good as S'. The greedy algorithm is always generating an optimal solution.

(b) *The overall penalty is now defined to be the maximum of the line penalties. The problem is to find a layout that minimizes the overall penalty. Prove or disprove that the above greedy algorithm correctly solves this problem.*

Consider the following situation:
When there are 5 word and each word length 5 and ideal line length is 24. The penalty of greedy algorithm is 19. While optimal penalty is 14. Thus the greedy algorithm is not always giving the correct answer.

4. *Prove or disprove that if a path connecting two nodes (say s and t) is a shortest path using weight function w, it is also a shortest path connecting s and t when weight function w' is used.*

It can be proved that after linear transformation of the edge weights, the shortest path stays the same. This is because after linear transformation, the order of the edge weights remains the same.
However, if the transformation of the edge weights is not linear. The shortest path may change.
Considering the following scenario:
There are three vertices a, b, c. $w(a,b) = 1, w(b,c) = 1, w(a,c) = 10,$
$w'(a,b) = 10, w'(b,c) = 1, w'(a,c) = 1.$
The shortest path using w from a to c is a, b, c but the shortest path using w' from a to c is a, c.

5. *Assume that edge weights are distinct in a given graph. Let T_k and T_p denote minimum spanning trees generated by the Kruskal's and the Prim's algorithm. Let $w(k_1) < w(k_2) < \cdots < w(k_{n-1})$ and $w(p_1) < w(p_2) < \cdots < w(p_{n-1})$, respectively, denote the weights in T_k and T_p. Prove that $k_i = p_i$ for each $1 \le i \le n-1$.*

We already have the Kruskal's and Prim's algorithm are correct giving solution of MST.
Let j denote the edge such that $k_j \neq p_j$
Without loss of generality we assume $w(k_j) \ge w(p_j)$.
Case 1: When T_k include p_j, $\exists l > j$ such that $w(p_j) = w(k_l)$. And we are having $w(p_j) = w(k_l) \ge w(k_j) \ge w(p_j)$. Thus $w(p_j) = w(k_j) = w(k_l)$, switching k_j with k_l in T_k will not change the order of edge weights in T_k. And we are having $w(k_j) = w(p_j)$
Case 2: When T_k does not include p_j, then by adding p_j into T_k will form a cycle. As T_k is a MST, any edges in this cycle other than p_j must be less than or equal to p_j.
$\exists k_l \notin T_p$ in this cycle. Thus, we have $w(k_l) \le w(p_j)$.
$\because l > j$
$\therefore w(p_j) \le w(k_j) \le w(k_l) \le w(p_j)$
$\therefore w(k_j) = w(k_l) = w(p_j)$
For $k_i = p_i$, obviously, we have $w(k_i) = w(p_i)$.
Thus, the statement is proved.

6. *Let G be a connected edge-weighted graph. Let e be an edge in G. Denote by $T(e)$ a spanning tree of G that has minimum cost among all spanning trees of G that contain e. Design an algorithm to find $T(e)$ for a given $e \in E(G)$. Your algorithm should run in $O(n^2)$ time.*

This can be done by altering Prim's algorithm for MST using adjacent matrix searching. By altering the very first step from choosing a single starting vertex to choosing a starting edge e (two vertices and the edge between them). This algorithm has the same time

complexity as the Prim's algorithm. Thus, it should run in $O(n^2)$ time.

7. *Suppose we assign n persons to n joibs. Let c_{ij} be the cost of assigning the ith person to the jth job. Use a greedy approach to write an algorithm that finds an assignment that minimizes the total cost of assigning all n persons to all n jobs. Is your algorithm optimal?*

Using a more global scale greedy approach and borrowing the idea of Kruskal's algorithm.
Step 1: sort all the c_{ij} in ascending order.
Step 2: selecting c_{ij} from small to large. If the regarding people and job is not occupied, then mark the regarding people and job as occupied. And add c_{ij} to solution. Repeat until all people and job are marked as occupied.
This algorithm is not an optimal solution. Consider the following scenario:
Person $\{A, B, C\}$
Job $\{1,2,3\}$
$c_{A1} = 1, c_{A2} = 2, c_{A3} = 10$
$c_{B1} = 2, c_{B2} = 2, c_{B3} = 5$
$c_{C1} = 3, c_{C2} = 3, c_{C3} = 10$
Applying the previous greedy solution will product c_{A1}, c_{B2}, c_{C3} with a total cost of 13. While the optimal solution should be c_{A1}, c_{B3}, c_{C2} with a total cost of 8.