Assignment 2

Jason Cluck
Dr. Choi
Csci 6212

October 15, 2012

1 Problem 1

1.1 Description

Consider the following problem for n jobs, each one which takes exactly one minute to complete. At any time $T = 1, 2, 3, ...$, we can execute exactly one job. Each job i earns a profit of p_i dollars if and only if it is executed no later than time d_i, where d_i is given as an input. Assume that d_i is an integer value. The problem is to schedule the jobs to maximize the profit.

Consider the following greedy strategy: for all the jobs with deadline at 1 minute, schedule the job with the maximum profit. Next, for all the jobs with deadline at 2 minutes or less, pick the job with the maximum profit from the remaining unscheduled jobs etc. For example, consider $n = 4$, profits $P = (50, 10, 15, 30)$ and deadline $D = (2, 1, 2, 1)$. The greedy strategy will yield the following solution: job 4 and job 1 to be scheduled for a total of profit of 80 dollars.

Give a counter example to establish that this greedy strategy does not always work

1.2 Solution

Let $n = 4$, $P = (20, 40, 60, 80)$ and $D = (1, 2, 3, 3)$. The greedy strategy provided would perform job 1, job 2, then job 4. Job 3 is clearly has a larger profit than jobs 1 and 2 but is never executed. The greedy algorithm presented generates a profit of $80 + 40 + 20 = 140$ but it should generate $80 + 60 + 40 = 180$. Therefore this greedy strategy does not always work.

2 Problem 2

2.1 Description

A k-coloring of a graph $G = (V, E)$ is a mapping $f : V \rightarrow 1, 2, ..., k$ such that adjacent vertices are mapped out in different colors, i.e., no two neighbors in G receive the same color

(i.e., same integer). Prove that the following greedy algorithm colors the given graph G with at most $\Delta(G) + 1$ colors where $\Delta(G)$ denotes the maximum degree (number of adjacent vertices) of any node $v \in V$.

for $i = 1, ..., n$ do
Color vertex v_i using the smallest available color in $1, 2, ..., \Delta(G) + 1\}$.

2.2 Solution

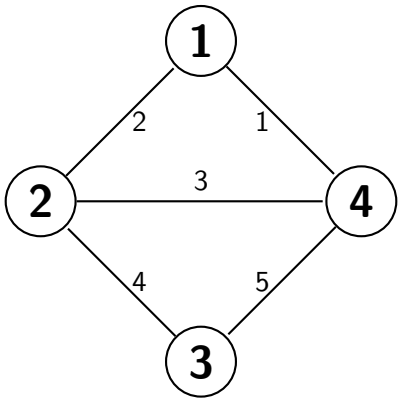
Proof. Let G be a graph with each node $v \in V$ having at most a degree of $\Delta(G)$. If node v has the max degree, a graph coloring algorithm will require at most $\Delta(G) + 1$ colors such that two adjacent nodes will not be the same color from that vertex. The worst case is if every node is connected. The algorithm still works because no connected node will have the same color. The root node and all of the adjacent nodes can have different colors with $\Delta(G) + 1$ colors being used. Repeating this for other vertices follow the same process and $\Delta(G) + 1$ colors will be required for the $\Delta(G)$ case. □

3 Problem 3

3.1 Description

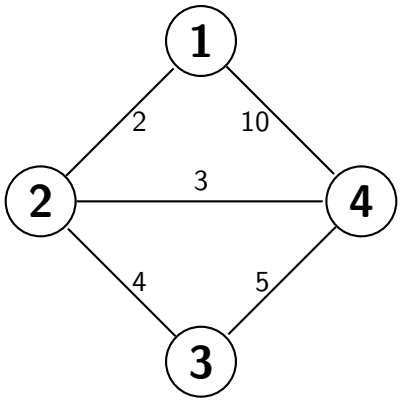
Let T be a minimum spanning tree of G (where G is an edge-weighted graph). Suppose the cost of edge e_0 in G is changed. Give an example of G, edge weight $w(e)$ for each e in G, T, e_0 and the new weight $w'(e_0)$ such that T is no longer a minimum spanning tree of G after the weight of e_0 becomes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

3.2 Solution



For this figure, using the edges $e = (1, 2, 4)$ will produced the MST T. If e_0 is the edge with weight 1 and the this weight is changed to 10, the graph on the next page will be produced.

The MST changes then and consists of edges $e = (2, 3, 4)$ and $T! = T'$



4 Problem 4

4.1 Description

Suppose that in 0-1 knapsack problem, the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing profit. Give an algorithm to find an optimal solution to this variant of the knapsack problem, and prove that your algorithm is correct.

4.2 Solution

Assume weights $W = (1, 4, 8, 16, 30)$ and profits $P = (50, 20, 10, 4, 2)$. The premise of the knapsack problem is to maximize $\sum_{i=1}^{n} p_i x_i$ while $\sum_{i=1}^{n} w_i x_i \leq W$. Since this variation has the smallest items being of the greatest value, the following algorithm can maximize profit.

```
def knapsack_solution:
        i = 0 #index counter
        while sum(W) < max_weight || knapsack == None:
                add item i to knapsack
                i++
```

Proof. knapsack_solution works by relying on the arrays already being sorted. The first item, $i = 0$ will always have a better ratio than item $i + 1$. That is to say:

$$\frac{p_i}{w_i} \geq \frac{p_{i+1}}{w_{i+1}} \forall i$$

Since this holds for all values of i, an optimal solution is provided by starting at $i = 0$ incrementing i until the knapsack is full or there are no more items. □

5 Problem 5

5.1 Description

Prim's algorithm works correctly when an arbitrary vertex is chosen as a start vertex. Prove that the smallest weighted edge (assuming there is only one smallest edge) is always in the tree generated by Prim's algorithm.

5.2 Solution

Proof. Prim's algorithm creates a MST by storing vertices in a minimum priority queue based on the smallest weight edge connecting vertices currently not in the tree with a vertex that is currently in the tree. Let the MST of graph G be T where $T \in G$. Assume there is an edge $e_i \in E$ that connects to vertex v_j.

case 1 – $min(w(E))$ is the only edge connecting v_j:
By definition of MST, $min(w(E)) \in T$
case 2 – $min(w(E))$ and other edges E_j connect to vertex v_j:
$min(w(E)) \leq \forall w(E_j)$
In this case, $min(w(E)) \in T$ by definition of MST. The other edges that connect to v_j are more expensive and are not included. From this, it can extrapolated that $min(w(E)) \in T$. □

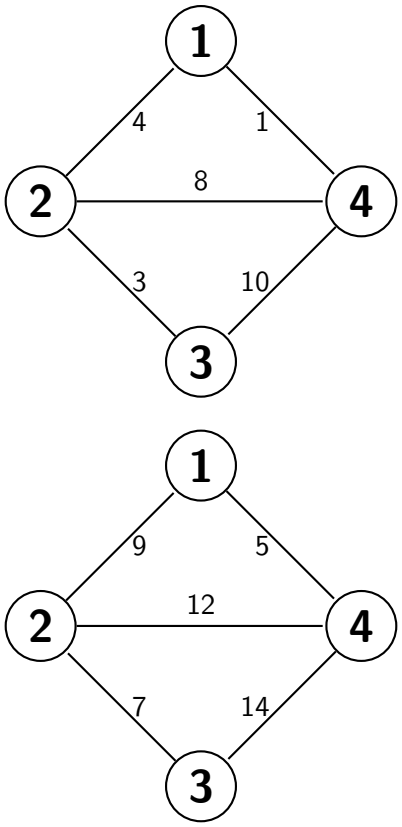
6 Problem 6

6.1 Description

Let G be an edge-weighted graph with $w(e)$ denoting the weight of edge e. Now, consider a new weight function $w'(e)$ such that $w'(e) = w(e) + c$ for each $e \in E(G)$, where c is a positive constant.
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.

6.2 Solution

Proof. Let $w(E(G)) = (1, 4, 8, 3, 10)$ and $w'(E(G)) = (5, 9, 12, 7, 14)$ with $c = 4$. Two graphs with these weights are shown below: The MST for the first graph uses edges with weights: $(1, 4, 3)$. The graph generated using $w'(e) \forall e \in E$ has an MST of $(5, 9, 7)$ which are the same edges as in the first graph. Since the constant c just scales every edge equally, there is no way the MST could ever be different by adding a constant factor to every edge. □



7 Problem 7

7.1 Description

Assume that edge weights are distinct in a given graph. Let T_k and T_P denote minimum spanning trees generated by Kruskal's and Prim's algorithm. Let $k1 < k2 < ... < k_{n-1}$ and $p_1 < p_2 < ... < p_{n-1}$, denote the weights in T_K and T_P respectively. Prove that $k_i = p_i$ for each $1 \leq i \leq n-1$.

7.2 Solution

Proof. To prove this, first examine how Prim works. Start by initializing the queue Q and set the root key to 0. While Q is not empty, dequeue the vertex with minimum weight edge, then add it to the tree by adding the edge to T. For each additional vertex v that are adjacent, check if any of the edge weights that are not in T are less than the current smallest edge weight of v. If so, update T with the lesser edge. Prim's algorithm starts with the edge with the smallest weight and then adds the next smallest and so on.

Kruskal's algorithm takes a similar approach. It first makes each vertex a separate tree and then sorts the edges in nondecreasing orders. It adds each edge, in order, and if the particular edge connects different trees it then merges those trees together. Kruskal's algorithm also starts with the edge of the lowest weight and then iterates through the list until all the

trees are connected.

These algorithms find the same edges for the MST in the same order by starting with the edge of lowest weight and iterating through a list. Therefore, $\forall\ i$ in T_K and T_P, $k_i = p_i$. $\square$