Justin Cabral

CS 5084

Assignment 4

Greedy Algorithms

Text

Description automatically generated with medium confidence

This statement is true. My argument is that e\* would be the first edge that Kruskal’s algorithm considers, so it will be included in the minimum spanning tree.

Text

Description automatically generated

This is true, because even if we were to square the cost of all edges and pass them to Kruskal’s algorithm, it would sort them in the same order, which in turn would put the same subset of edges into the minimum spanning tree. Therefore, T must still be a minimum spanning tree for this instance.

Text, letter

Description automatically generated

This is false. Consider an example where there is a minimum-cost path s-t that has a cost of 6 and an alternate path that has a cost of 3 + 4, consisting of two edges. If we square each edge, there will be a new minimum cost path s-t where 32 + 42 = 25 < 62.

A picture containing text, newspaper, screenshot

Description automatically generated

I plan to show that the greedy algorithm, S, based on maximizing the weight of each truck and placing each package based on arrival time, is optimal. I can represent the packages as each having and index p1,p2,p3,…pi. Along with that, a corresponding weight wi assigned to truck tk. If the greedy algorithm in use is not optical, then there exists another solution π. All solutions need to satisfy both constraints, being that no truck can have a weight greater than W, and all packages must be shipped in the order they arrived.

Base: When i = 1

* S will put P1 into T1
* Π needs at least 1 truck, so k ≥ 1.

Let us suppose for i = n that Sk ≤ Πk where k is the number of trucks in total needed

Inductive step:

* Let i = n + 1, so for Pn+1
* **Case 1**: Pn+1 doesn’t fit on previous the previous truck k for both S and Π
* The weight of truck k + wn+1 > W, so both S and Π must use another truck k+1
* **Case 1B:** Pn+1 does not fit on the previous truck k for S but it fits for Π
* Then the last truck for Π had fewer packages than S because no packages can be delivered out of order
* Thus, it means that Π used more trucks than needed by sending out a truck that was not full
* Therefore, S remains optimal and minimized the number of trucks it needed.
* **Case 2:**  Pn+1 fits on the previous truck for both algorithms.
* The weight for truck k + the weight of Pn+1 is <= W which is our weight limit
* Therefore, both algorithms still only use just k trucks
* Thus, in all cases S is ahead of all other algorithms

Text

Description automatically generated

Let us assume we have two distinct minimum spanning trees X and Y. Given that both X and Y are distinct, there exists an edge in one but not the other. Let e be the smallest edge that is exclusively in X or Y, but not in the other. Without loss of generality, let e be in tree Y but not in tree X. Let L be this current loop, then L contains an edge that is not in Y, which isn’t e. We can call this edge d for our use case. Given that e is the smallest edge that is exclusively in one but not he other, e < d. If we were to replace d in X with e, this would still be a spanning tree but now the overall weight of the tree is less than the original tree. This directly contradicts the assumption that T is a minimal spanning tree. Thus, G has a unique minimum spanning tree.

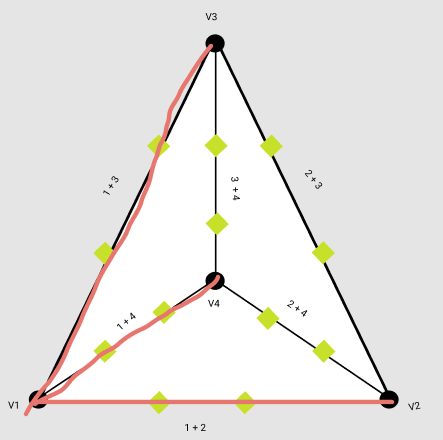
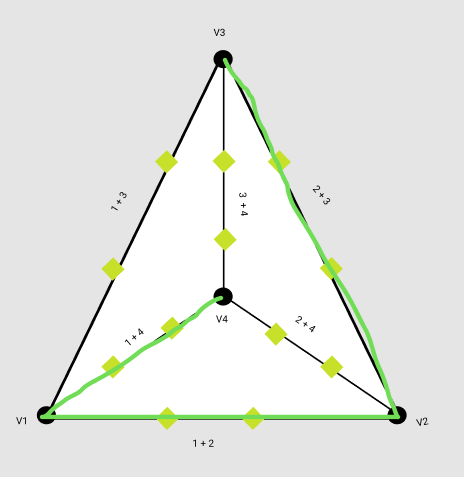


Let us suppose that by way of contradiction that tree X and X’ are both two distinct MSTs (minimum spanning trees) of G. Given that X and X’ both have the same number of edges but are not equal, then there exists some edge e’ found in X’ but not in X. If we were to add e’ to X, it would cause a cycle. If we were to let e be the most expensive edge in this cycle, then according to the cycle property, e wouldn’t belong to any MST, contradicting the fact that it should be in at least one of our trees, X or X’. Thus, G has a unique minimum spanning tree.

Text, letter

Description automatically generated

This statement is false. See the diagrams below for a counter example:



W = 13 W = 12

Since the total weight is greater than that of the tree with edges from v1 to every other vertex, its not a minimum spanning tree.



This statement is true. Assume that we have a minimum spanning tree T of the graph G, which is not a minimum-bottleneck tree of G. Then, there must exist a tree T’ that is a spanning tree with a bottleneck edge which is cheaper than the most expensive edge found in T. Therefore, the bottleneck edge of T is more expensive than other edge T’. If we were to add the bottleneck of edge T to T’, this would create a cycle and the edge would become the most expensive edge in the cycle. So, according to the cycle property, this edge does not belong to a minimum spanning tree. This contradicts that T is a minimum spanning tree and that it contains an edge. Thus, every minimum spanning tree of G is a minimum bottleneck tree of G.

Text

Description automatically generated

Function(G):

For i <- 1 to m – n + 1

tree <- breadth first search tree running BFS on graph G

e <- any edge (u,v) in G, but not in T

path <- The path between (u,v) but not in T

e <- Highest costing edge that is in the cycle when you introduce e into T

G <- G – e, removing e form G in accordance with the cycle property.

End For Loop

Return G

The outer loop will run for a constant number of times that will range form 1 to 9 depending on the number of edges the near tree has. Since the number of edges is at most n + 8 our breadth first search will run in O(n + 8 + n) which = O(2n +8) which reduces to O(n). Finding an edge in G, that is not in T can be done in O(n), and finding the path between (u,v) in T can be done in O(n). Then finding the highest costing edge in the cycle can take O(n) while removing the edge from G can be done in sublinear time. Thus, the algorithm provided takes O(n) running time.