**Greedy Problems:**

1. **Does Prim’s algorithm always work on a connected graph with negative edge weights? Yes/No? If not, give a counterexample.**

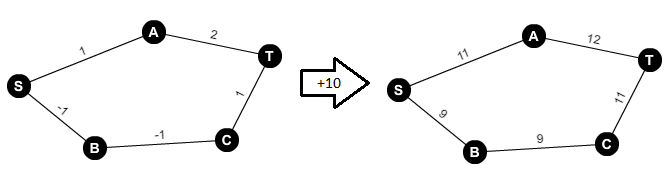
Yes. Prim’s algorithm works with negative edges because the tree is constructed using the smallest edge with accordance to the tree. So in the perspective of Prim’s algorithm, a negative edge – in comparison to a positive one – is just a very short edge.

1. **Dr. Genius suggests the following algorithm for finding the shortest path from node *s* to node *t* in a graph with some negative edge weights. Add a large constant to each edge weight (same constant for every edge) so that all the weights in the graph become positive. Then on this new graph run Dijkstra’s algorithm starting at node *s* until you find the shortest path to *t*. Does this algorithm always give the shortest path tree?**

**Yes/No? If not, give a counterexample.**

Nope. Paths with more edges will be more affected by the addition of a constant than paths with much fewer edges.

Consider the following graphs:



*Created using “Graph Creator” by National Council of Teachers of Mathematics*

In the second graph, using Dijkstra, we will get the following sequence of additions from S to T: S, B, A, T – resulting in a total path weight of 32. Dijkstra fails to pick up the true shortest path: S, A, T – which has a weight of 23.

1. **Suppose you are given *n* ropes of different lengths, you need to connect these ropes into one rope. The cost to connect two ropes is equal to sum of their lengths. You want to find the minimum cost way to do this.**

**For example: Suppose you have three ropes with lengths 2, 5, and 8. If you chose first to connect the length 5 and 8 ropes, then connect the length 2 and 13 ropes, the total cost would be (5 + 8) + (13 + 2) = 28. However, if you first chose to connect the length 2 and 5 ropes, then the length 7 and 8 ropes, the total cost would be (2 + 5) + (7 + 8) = 22 (which happens to be optimal).**

**Specify with pseudo code a greedy algorithm to connect ropes with minimum cost**

// A[0…n] is an array sorted in ascending order

// Approach: Connect rope lengths in ascending order

cost = 0

for i 0 to n

cost += A[i]

return cost

**Prove your algorithm always finds the least cost solution for distinct rope lengths.**

When continually calculating the cost of attaching two strings together, we see that the calculation involves the previous cost of the rope. For instance, if we add two ropes of length 1 and 2 together we get a total cost of 3; Then if we add a length of 3 to that, our cost goes to 3 + 3 = (1 + 2) + 3 = 6; If we then add 4 to that, we get 6 + 4 = (1 + 2 + 3) + 4 = 10.

Since the lengths of the previous ropes are included in every calculation upon concatenation, in order to minimize the total cost, we want to minimize the number that is constantly involved in every calculation. Hence, we add the smallest lengths to the rope chain first.

**Analyze your algorithm’s complexity**

The operation is addition, and we add for every item in the array. With up to ***n*** items, the complexity is Θ(n).

1. **There is a long straight country road with expensive houses scattered along it. The houses are owned by affluent stock traders who require cell phone service. You consult for the company that needs to provide the cell service to every house without exception. The towers only have a range of four miles.**

**So, you want to place the cell phone towers at locations along the road so that no house is more than four miles from the nearest cell phone tower. You know exact mileage along the road where each house is located. Design a greedy algorithm that will determine the set of locations for the cell towers that requires the fewest cell tower. Denote the location of the ith house as *Mi.***

**Give an efficient algorithm**

**Specify (pseudo code) an efficient greedy algorithm to achieve this goal with the fewest cell towers.**

**Prove your algorithm always finds the optimal solution.**

**Analyze your algorithm’s complexity.**