1. Show how to find the maximum spanning tree of a graph, which is the spanning tree of the largest total weight.

We can simply modify the MST algorithm to solve this problem too. For Prim algorithm for example, the only thing we need to change is we add maximum instead of minimum weight edge between V and V - T in each iteration. The following is the pesudo code.

```
for u <- V
  c(u) <- 0
  p(u) <- NIL

c(u0) <- INF

H <- makeMaxPriorityQueue(V) by using c(u) as key
while H is not empty
  v <- extract_max(H)
  for each u, (v, u)  E
    if c(u) < w(u, v)
    c(u) <- w(u, v)
    p(u) <- v
```

2. Consider an undirected graph G = (V, E) with nonnegative edge weights We. Suppose that you have computed a minimum spanning tree of G, and that you have also computed shortest paths to all nodes from a particular node s. Now suppose each edge weight is increased by 1; the new weights are W'e = We + 1. (a) Does the minimum spanning tree change? Give an example where it changes or prove it cannot change. (b) Do the shortest paths change? Give an example where they change or prove they cannot change.

(a) No, The minial spanning tree will not change.

If we add 1 to all edges, and run prim algorithm again. Then all the vertext and edge we pick in each step still remain the same. The edge with smallest weight is still the min weight edge. Therefore, the tree we build will be the same tree as original one.

(b) Yes, the shortest path will change.

For this graph

```
      1      1
    /---- b ---- c
   a             /
    \           /  1
     ---- d ---
   4
```

The shortest path from a to d is a - b - c - d. However, if we add 1 to all edge

```
      2      2
    /---- b ---- c
```

```
    a             /
     \           /  2
      ---- d ---
    5
```

The shortest path from a to d become a - d.

3. Consider the following algorithm.

```
Input: a weighted graph G=(V, E) and w.

Maybe-MST(G, w)
Sort the edges into non-increasing order of edge weights w;
T <- E
for each edge e □ E, taken in non-increasing order by weight
 if T-{e} is a connected graph
 T <- T-{e}
Output T
```

Either prove that the output of the algorithm is a minimum-spanning tree or give a counterexample when the algorithm does not output a minimum-spanning tree.

Yes, it will generate the MST.

Let S be the MST of graph G. When we rmeove the heaviest edge e from graph, that edge may be in S or not in S. If e is not in the S, then we are good to remove it. If it is in the S, removed it will disconneced the graph, so we will need to keep it. If we remove e, while whole graph is still connected, that means there is another path e' in cycle, and e' is not in S. Otherwise, we will have a cycle in MST. Notice, removing e instead of e' will get the MST, since the weight of e is larger than e' by our algorithm, we must remove e first. Replacing e with e' will increase the total weight of tree and we cannot yield the MST.

4. The diameter of a tree T=(V, E) is defined as the largest length of all shortest path (i.e., the path with the smallest number of edges) between pairs of vertices. Devise an O(|V| + |E|) algorithm to compute a diameter for a given tree.

We run BFS at first, then we can get one longest shortest path, then running BFS again from the end of longest shortest path we get from the first time.

The first BFS is to find one end of the longest shortest path. The second time is to find another end of the longest shortest path.

The time complexity is O(|V| + |E|) since BFS is O(|V| + |E|).

5. Dr. Fuzzy suggests the following algorithm for finding the shortest path from node s to node t in a directed graph with some negative edges: add a large constant to each edge weight so that all the weights become positive, then run Dijkstra's algorithm starting at node s, and return the shortest path found to node t. Is this a valid method? Either prove that it works correctly, or give

a counterexample.

No.

The original shortest path for the following graph is a -> b -> c

If we add 2 to all edges, then shortest path become a -> c

6. Decide whether you think each of the following statements are true or false. Justify your answers. a. Let G be an arbitrary flow network with a source s, a sink t and a positive integer capacity c(e) on each edge. If f is a maximum s-t flow in G, then f saturates every edge out of s with flow, i.e., for all edges e out of s, we have f(e)=c(e). b. Let G be an arbitrary flow network with a source s, a sink t and a positive integer capacity c(e) on each edge. Let (A, B) be a minimum s-t cut w.r.t. these capacities. Now suppose we add 1 to every capacity, then (A,B) is still a minimum s-t cut w.r.t. these new capacities, i.e., c'(e)=c(e)+1 for every edge e.

a. No.

Consider the following flow.

The max flow is 2. However, for all edges e out of s, we have f(e) < c(e).

b. No.

Consider the following flow.

s to a b c d e all have capacity 1. a b c d e to x also capacity 1.

x to t has capacity 6.

Initially, the min cut is on the left of x (s to x).

When we increase all capacity by 1. Now the min cut is on x to t.

7. Consider the following scenario. Due to large-scale flooding in a region, parametics have identified a set of n injured people distributed across the region who need to be rushed to hospitals. There are k hospitals in the region and each of the n people needs to be brought to a hospital that is within a half-hour's driving time of their current location (so different people will have different options for hospitals, depending on where they are right now). At the same time, one does not want to overload any one of the hospitals by sending too many patients its way. The parametics are in touch by cell phone, and they want to collectively work out whether they can choose a hospital for each of the injured people in such a way that the load on the hospital is balanced: each hospital receives at most n/k people. Give a polynomial-time algorithm that takes the given information about the people's and the hospitals' location, and determines whether this is possible

We can solve this by modeling the problem as a directed graph flow problem.

Our graph will have two layers between source and sink.

The vertices in the first layer represent patients, and the vertices in the second layer represent hospitals.

The source will only connect to the vertices in the first layer; Vertices in the first layer will only connect to the vertices in the second layer. Furthermore, the sink will only connected to the vertices in the second layer.

All edges capacity between source and the first layer are 1. All edges capacity between the first layer and the second layer are also 1. All edges capacity between the second layer and sink are also n/k.

The edges between vertices in the first layer and vertices in the second layer represent the hospitals option the people will have.

Now we can just use the maximum flow algorithm to solve this problem.

If we the max flow is n, then we know all patients can be sent to hospitals in time. Otherwise, it is not.

Building graph will take $n*k$ time.

If we use Ford-Fulkerson maximum flow algorithm to solve this problem.

The time complexity for Ford-Fulkerson is `O(Ef)`. Here f is n, and E is O(nk). Therefore, the algorithm time complexity is `O(nk^2)`.