Berke Can Rizai

Ex 6-1. ✓✓ Read (9/9)

6 8
100

Ex. 6-2.

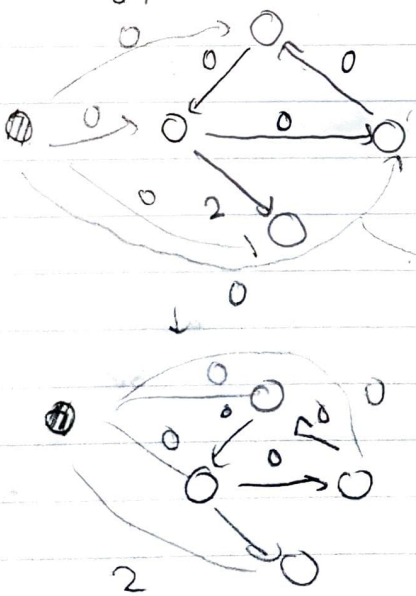
(8/8)

In the Johnson's algorithm we first aim to deal with negative weights and turn them positive in each cycle so that graph can be solved with dijktra or another algorithm. With function $\hat{w}(u,v)$ we ensure all edges are nonnegative.

Ex 6-3.

(3/8)

Say, we have this;



Since each time we go with edge + vertex at each iteration, we have 0,

Shortest dist from our vertex to each other vertex is 0 at the end.

Ex 6.4 Read ch 23 ✓✓ (9/9)

Ex 6-5.

Since in kruskal, we union the lowest price edges but in order to do that we first sort all the edges. Quicksort would take $O(n \log n)$ but counting sort would take $O(n+V)$ which can be faster. Since kruskal runs in $O(n \log V)$ and we consider counting sort $n+V+n\log V$ would take

$O(V+n\log V)$ since $cn \leq n\log V$ for c which means we can ignore c.

If we had $|W|$ instead of $|V|$ we could just change $|V|$ to $|W|$ so; we have $O(W+n\log W)$

7/8

Ex 6-6.

Here, what we do is, starting from a node, we always choose edge with least weight that is reachable from our constructed graph this means the number of edges we need to consider will be increasing but we can't surely guess correctly for example we might reach 2 edges or maybe n edges at 2nd iteration which means as worst case, we should always consider n many edges. This means if we have n many edges, we have $O(n)$ complexity.

6/8

P 6-1.

a) For updating distance matrix D, we can do;
if $r = w_{i,j}$: return // since we don't need to modify
elif $r < w_{ij}$:
    // since some shortest edge might have changed
    for each neighbour edges connected to edges s and t run the prims algorithm
else:
    for each neighbour vertex of s and t, run the prims algorithm which will take $O(n^2)$.

This would work since if edge weight decrease some shortest path needs to be changed perhaps.
If $r \geq w_{ij}$ it is $O(1)$

for the predecessor matrix;

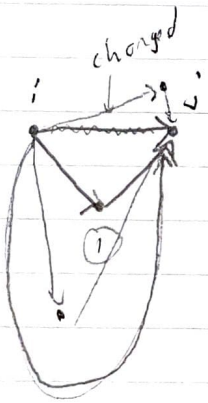
We can just run Floyd Worshall on dist. matrix

```
for k=1 to n:
    for i=1 to n:
        for j=1 to n:
            if D[i][k] + D[k][j] < D[i][j]
                D[i][j] = D[i][k] + D[k][j]
                Π[i][j] = Π[k][j]
return Π matrix.
```

3/6

This runs in $O(n^3)$ for any input since it recalculates whole matrix.

b)



1, 0/4

In the distance graph D there are n entries which means inorder to operation to take $O(n^2)$ time, at first weights of all edges must be really high and our i or j node should be connected to many of them since if they were outsider, no the most path would change but now, each shortest path is changed, effected.

c) Taking floyd worshall from the book, and modify

```
func(w)
    D^0 = w                                  and added h* for track.
    for k=1 to n
        let D^(k) = (d_ij^(k))
        for i=1 to n
            for j=1 to n  // modified here
            if(h*<h) d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
                h* ++;
return D^(n)
```

2/5

d) First we take square matrix mult. from the book

```
mult(A, B):
n = A.rows
C = nxn matrix
for i = 1 to n
    for j = 1 to n
        c(i)(j) = 0
        for k = 1 to n
            C[i][j] = C[i][j] + A[i][k] • B[k][j]
```

1,5/5

and Extend-shortest_path(L, w):

```
n = L.rows
L' = new nxn matrix
for i = 1 to n
    for j = 1 to n
        L'ij = ∞
        for k = 1 to h        // instead of n since we are limited to k hops, vertices
            L'[i][j] = min(L'[i][j], L[i][k] + W[k][j])
return L';
```

This should work since if h was n in last for-loop it would find the shortest path in absolute but here at most k steps are taken. Complexity. Since there are 3 for loops within each other we have $O(n^3)$ but, each matrix mult costs $\log n$ which means we have $O(n^3 \log n)$

e) if newweight == r_{ij}
    return 0 // without any change.
elif r_{ij} != newweight then;
    recalculate with Extend-shortest_path

2,5/5

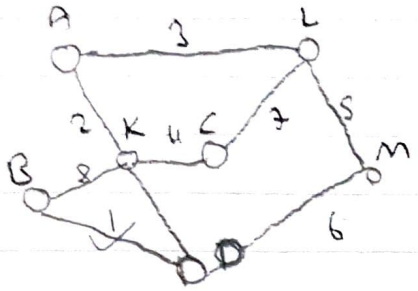
// this would take $O(n^3 \log n)$ since extend_shortest_path is that complexity if there is change in edge weights.
Else O(1)

=>

P 6-2.

a) We can prove this with Kruskal's algorithm. Say we have graph G s.t.

Here all weights are unique.



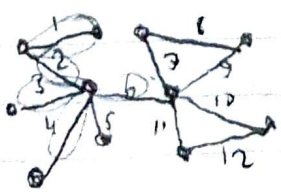
Assume that BD edge is added to tree since in Kruskal we choose smallest. Second edge would be (BK) with 2 weight and we move with choosing smallest.

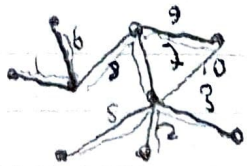
4,5/5

Say we have set S for all edges and since in each iteration of Kruskal we choose min(S) if there is no cycle we can see that this tree is unique. If in any iteration we choosed an edge that create a cycle this wouldn't be tree, if we didn't choose min, this wouldn't be minimum tree and if we do these then this tree is surely unique.

b) This kind of looks like prims algorithm, in each iteration we choose lighest edge that is connected to us which means that since we connect to lighest edge it is in our final MST because that is how we constructed it in the first place. With this, in each iteration, we don't add anything other than what we would have in MST. So, at the end we would have mst.

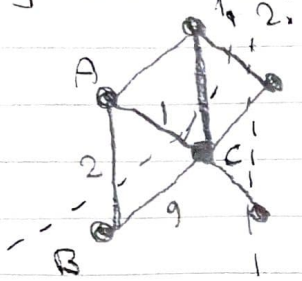
4/7





c) This wouldn't work since we might be disregarding really light edges which is not optimal,

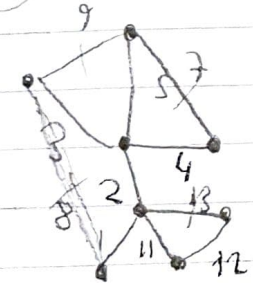
Say we have,



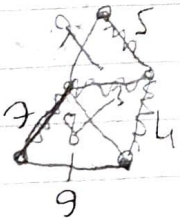
Say we divide from line shown, with this we disregard (A-C) edge since we divided in this way and in the final MST we would include (A-C) edge but now we have to include (B-C) which is 9 because our random division left us with these nodes.

Maybe connecting C with upper node in a edge that is divided with 2nd line is also better but we disregard them now.

d)



Trying this algorithm in the graph to the left, this yields the MST which suggests, algorithm is working properly.



We can think as, if there are 3 edges in a cycle that is triangle, if we delete 2 or 3 of them we don't cover graph.
If we don't delete most costly edge we don't have mst since it is more costly to delete light edge.

Applying it to ↑ this we can see it works when there are few cycles next to each other.