Sect 1C

UCLA Computer Science Department

CS 180 **Algorithms & Complexity**

Midterm **Total Time: 1.5 hours** **October 27, 2016**

Each problem has 20 points.

A. Li 1C?

All algorithm should be described in English, bullet-by-bullet

1 a. Describe Depth First Search on an undirected and un-connected graph.
b. Analyze the time complexity of DFS when there are no cycles.

a. 1. • For ~~each~~ connected component G_1, choose an arbitrary node S_1 as its source node ?

2. • Find nodes that S_1 has an edge to and put them on a sta[ck]

3. • ~~P~~ Pop a node out of the stack and find nodes that it has an edge [to] and put them on the stack.

4. • Repeat step~~s~~ 3 until all nodes in this connected component has [been] visited

5. Repeat for each connected component, ~~and start at~~ starting from step 1.

+15

b. If there are no cycles, the DFS visits each node (n) once and traverses each edge once (e), so we get a n+e time complexit[y]. It is not just O(e) due to the fact that there maybe separate conne[cted] components.

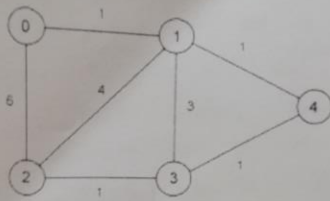
O(n+e)

+5

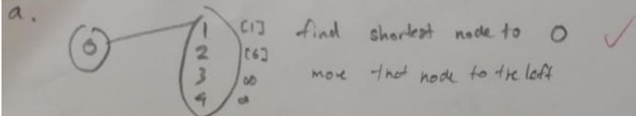
20

2 . **a**. Use Prim's MST algorithm to find an MST in this graph. Show each step on the graph shown below.

b. If some edges were negative would the algorithm still find an MST. Why?

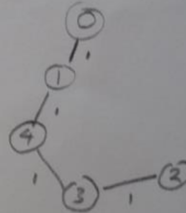


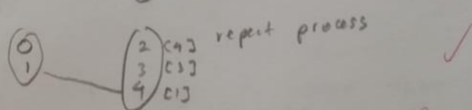
a.



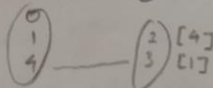
find shortest node to 0 ✓

move that node to the left



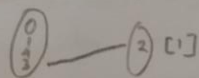


repeat process ✓



✓

+15



✓

. Yes, because we don't deal with directed edges like dijkstra. If edges have
weights, just add a positive constant to each edge and calculate MST then
and subtract the constant at nodes later, so negative edges w
pose a problem.

+5

20

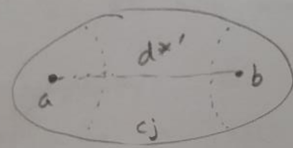
15

3. Prove that solving the k-clustering problem (described in class and in the book) using Kruskal's MST algorithm, produces an optimal clustering. That is, it will produce an optimal set of clusters C1, C2, …, Ck with Maximum cluster distances. (Use a figure to better describe your proof: as was done in class / book).

Suppose that our set is not optimal, and that this new set $C_1', C_2' \dots C_k'$ is optimal.

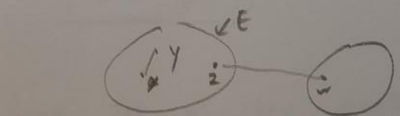
That means that we have a cluster in set of C_ks that doesn't exist in C_k's (let's call that cluster C_j).

Let C_j be the cluster that contains nodes (a,b) that would make $d*'$ in C_k' set.



We want to show that $d^{*\prime}$ is actually smaller than d^*.

Kruskal's algorithm works by taking the smallest edges and putting those nodes in a cluster.



In this example z-w is the d^*. If we break up cluster E, we find that zw is still d^* because xy less than zw due to Kruskal's alg.

Any two nodes in the same cluster have less distance apart than any two edges in diff clusters because Kruskal's groups ones together.

Therefore, if we break C_j to make a-b, we find that it would be smaller than d^* of C_k set, so $d^{*\prime}$ is bigger than d^*, so this is a contradiction. Our set $C_1 \dots C_k$ is an optimal set

longer
' cannot
obtained by
breaking up an
existing cluster.

4. Consider a sequence of n real numbers X = (x1, x2, …, xn). greedy

a. Design an algortihm to partition the numbers into n/2 pairs. We call the sum of each pair S1, S2, …, Sn/2. The algorithm should find the partitioning that minimizes the maximum sum.

b. Analyze the time comlpexity of your algorithm.

sum largest with least, then next largest with next least

a. • sort the set of numbers

- pair up the first term and last term
- keep pairing up numbers in this fashion

$((1, n), (2, n-1) \ldots (k, n-k+1))$ and until all #'s have been paired up.

10

b. Time complexity for my algorithm is

n logn for the sort

$\left(\frac{n}{2}\right)2$ for the pairing.

5

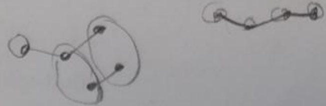
n logn + n

$O(n \log n)$

Proof: let us say another pair is smaller.
If smallest pair for our alg exists in smallest + largest,
it means for this it is some another pair.
smallest + 2nd largest. But if this happens, it means that
they have a pair that is 2nd smallest + largest
that cannot be!

5

5. Let G be a DAG and let K be the maximal number of edges in a path in G.
a. Design an algorithm that divides the vertices into at most k+1 groups such that for each two vertices v and w in the same group there is no path from v to w and there is no path from w to v.
b. Analyze the time complexity of your algorithm.



a. We can use a BFS for this.

1. • For each connected component G, , find an arbitrary node source.

2. • Put s_1 in group ①
3. • Find nodes that s_1 has edges to and put them in different (if there are j such edges, put them in group ②, ③ ... until (j+1).) And in a stack.

(7)

4. • Pop node from stack and repeat above step for this node (insta
Repeat step 4
5. • until all nodes in this connected component are in diff
• Repeat from step 1 for all other connected components. (Start to group 1).

Let us assume that in a group we have an edge between 2 node must mean that they are in the same connected component, but according it would have put nodes of connected component in different grou is not possible. (Contradiction).

(4)

Also, say there are > k+1 groups. Let's say $j > k+1$ and we ha
As my alg creates 1 more group than max # of edges in a path, we would h
So we would have $j-1 > k+1-1$ max edges in a path, but that contradiction to assumption above.

b. Time complexity (nodes & edges)
BFS takes $O(n+e)$ time
putting nodes in groups takes $O(n)$ time total
So we have $O(n+e+n) \rightarrow O(2n+e)$

$O(e+n)$

(5)

if you are checking for
Then it is more complex
how do you k
two nodes have
my above