CS 180 Algorithms & Complexity ID :

Midterm Total Time: 1.5 hours October 27, 2016

Each problem has 20 points.

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.
- Let there be a stack S used to store vertices of graph G.
- Start with an arbitrary vertex v. Store it in S.
- Visit an unexplored neighbor of v and store it in S.
- Continue visiting the nodes in this manner and storing them in S until you reach a node for which there are no unexplored neighbors.

☆

+15

- We will need ~~to backtrack~~ until we find a node with an unexplored neighbor. Pop off the top node from the stack. If it has an unexplored neighbor, resume DFS. If it does not have an unexplored neighbor, continue popping off nodes until you find a node with an unexplored neighbor. If there are no such nodes, then we need to check for nodes in another connected component of G.
- Find a node not visited in G. Perform DFS from this node. ☆
- Continue this algorithm until all nodes have been visited.

b.
- For each vertex v, DFS will have visited all of its neighbors, which is the degree of v, d_v.

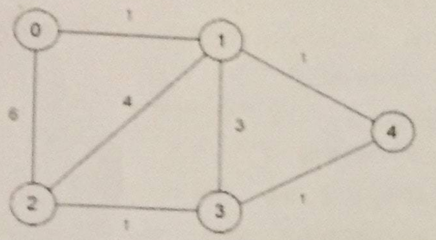
+5

- The sum of all degrees in a graph is 2e (twice the number of edges). If we remove all edges and add them back to the graph one by one, each edge contributes 2 to the total degree because each adjacent vertex connected by the edge has its degree incremented by 1.
- So, the time complexity for a connected graph will be $O(2e) = O(e)$.
- If the graph is unconnected, we visit each vertex once to discover it has no neighbors (worst case). So, the time complexity for an unconnected graph is $O(n)$.
- Therefore, the time complexity in general is $O(e) + O(n) = \boxed{O(e+n)}$

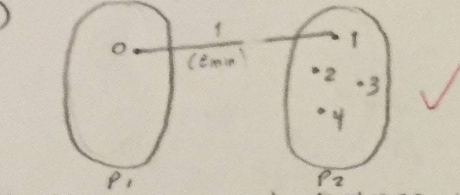
20

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

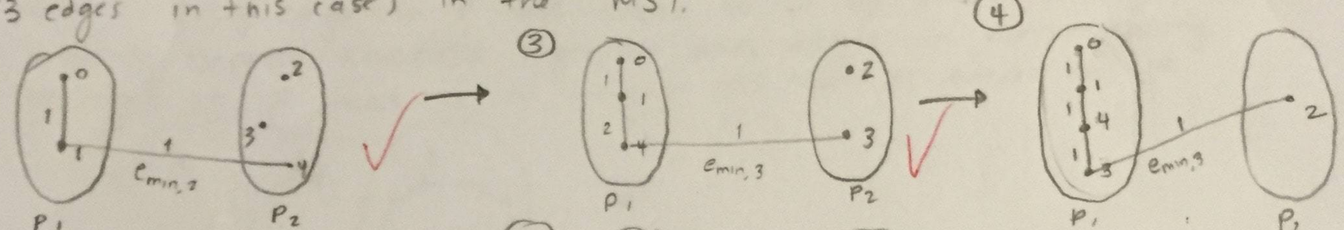
b. If some edges were negative would the algorithm still find an MST. Why?

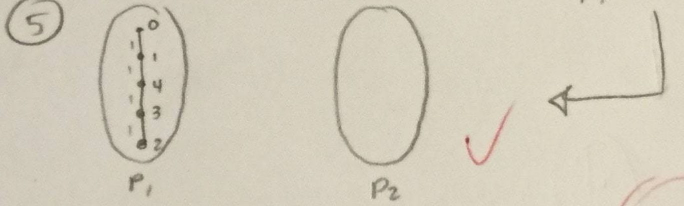


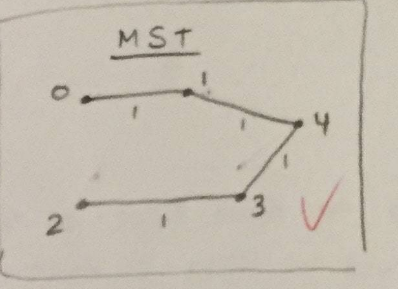
a. • Choose an arbitrary vertex v and place it in a partition P₁. Place the remaining vertices in another partition P₂. Determine the minimum edge from P₁ to P₂ ($e_{min,v}$). By MST Theorem, this edge must be in the MST



• Move the connected vertices to P₁. Find the minimum edge from P₁ to P₂ ($e_{min,2}$) and place it in the MST. Continue until there are n-1 edges (3 edges in this case) in the MST.







done

15

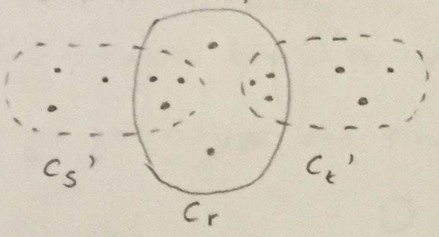
b. If some edges were negative, the algorithm would still find an MST. When we look at edges from P₁ to P₂, we choose the minimum edge adjacent from any vertex in P₁ to any vertex in P₂. So, if there was a negative edge, this edge would be chosen by our algorithm, which is the correct result. Including the negative edge would minimize the overall weight, and our algorithm would indeed choose the negative edges because it choose the minimum edges from P₁ to P₂.

20

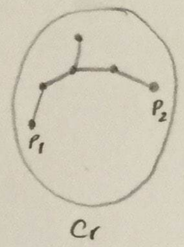
20

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).

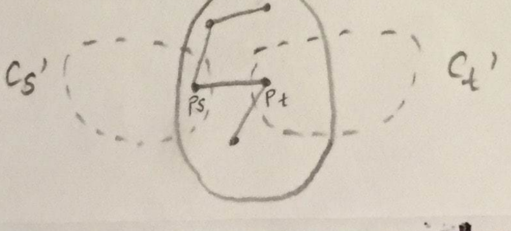
- Suppose Kruskal's algorithm produces a set of clusters $C_1, C_2, \dots, C_k$ and assume there is an optimal set of clusters $C_1', C_2', \dots, C_k'$.
- Let the cluster distance of the set of clusters produced by Kruskal's algorithm be d^* and let the cluster distance of the optimal set of clusters be $d^{*\prime}$.
- $C_1, C_2, \dots, C_k$ cannot all be subsets of $C_1', C_2', \dots, C_k'$ because we would be missing vertices in our clustering if this was the case. Since we assume that $C_1, C_2, \dots, C_k$ produces a different set of clusterings than $C_1', C_2', \dots, C_k'$ we must have some cluster C_r that has points from C_s' and C_t' (see figure).



- In C_r, assume we have two points p_1 and p_2 that are connected on some path P. Every edge in P was added to C_r by Kruskal's algorithm before stopping. Because Kruskal's algorithm adds edges in nondecreasing order, the edges on P must be of weight less than or equal to d^*.



- Let p_s be last node in C_s' that is still in C_r. Let p_s be the first node in C_t' that is still in C_r. By the statement above, we know the edge connecting p_s and p_t must be of weight less than or equal to d^* since it was added by Kruskal's algorithm before stopping. However, p_s and p_t are in distinct clusters C_s' and C_t' of the optimal solution. So, $d^{*\prime} \le d^*$ since the edge between p_s and p_t is less than or equal to d^*. This means the optimal solution can be at most d^*, or the cluster distance of the optimal solution is d^*. So, Kruskal's algorithm produces an optimal maximum cluster distance clustering.



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

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.

a. Algorithm

- Start at x_1 and find its sum with each remaining number $x_2, \dots, x_n$.
- Store these sums in an array for x_1.
- Continue to x_2 and find its sum with each remainng vertex $x_3, \dots x_n$.
- Store these sums in an array for x_2.
- For each x_i, find its sum with $x_{i+1}, \dots, x_n$ and store it in an array.
- Iterate through the arrays and find the max value.

0

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. Algorithm

- We will use topological sort.

1.

- Compute all in degrees and out degrees of the vertices in G.
- Determine the sources (vertices with in degree of 0).
- Place these sources in a group.
- Remove these sources from G and modify the in degrees of the remaining vertices in G.
- Find sources in G and place them in a different group.
- Continue until there are no remaining vertices in G.

Proof

5

- By way of contradiction, assume vertices v and w in the same group have an edge going from v to w. This means that when v and w were added to the same group, there was an edge going from v to w. So, the in degree of w is at least 1. This is a contradiction because when we add vertices to the same group, the vertices are sources, indicating their in degrees are 0.
- Now, we will show that our algorithm produces at most k+1 groups. Let k be the maximum number of edges in a path in G from vertex a to b. When a is added to a group, it is removed from the graph, along with the edges leading out from a. Then, the next set of sources are added to a group. with their edges releted as well.
- Because there are k edges between a and b, b will not be added to a group until the algorithm deletes all edges from a to b since b is added to a group only when it becomes a source.
- So, the algorithm will have created k groups when b becomes a source. So, b is added to the k+1 group.
- Since there can be no edges leading out from b since a to b is the maximum length path, we have at most k+1 groups.

b. Time Complexity

- computing in degrees and out degrees takes O(e). If we remove all edges and add them back into G, then each edge contributes 1 to the in degree of a vertex and one to the out degree of a vertex. So, we find the in degree and out degrees by adding all edges back in, which is O(e).

→

- Determining the first set of sources requires us to look at each node, which is $O(n)$.
- When we find a source we remove it from G and remove the edges leading out from the source. So, we only look at each edge once as we traverse through the graph, which is $O(e)$.
- So, the time complexity is $O(e) + O(n) + O(e) = \boxed{o(e+n).}$