CS570
Analysis of Algorithms
Fall 2017
Exam I

Name: ______________________

Student ID: __________________

Email Address:_________________

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	12	
Problem 3	25	
Problem 4	16	
Problem 5	12	
Problem 6	15	
Total	100	

Instructions:
1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.

1) 20 pts
   Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

1) 20 pts

[TRUE/[✓ FALSE]]
Every directed acyclic graph has only one topological ordering of its vertices

[TRUE/[✓ FALSE]]
Suppose we have a graph where each edge weight value appears at most twice. Then, there are at most two minimum spanning trees in this graph

[TRUE/[✓ FALSE]]
In every stable matching, there is a woman who is matched with her highest-ranked man.

[[✓ TRUE]/FALSE]
Every sorting algorithm requires $\Omega(n \lg n)$ comparisons in the worst case to sort n elements

[TRUE/[✓ FALSE]]
Amortized Analysis helps determine the average run time of an algorithm.

[[✓ TRUE]/FALSE]
$n^{\log_2 n} = \Theta(2^{(\log_2 n)^2})$.

[TRUE/[✓ FALSE]]
The total amortized cost of a sequence of n operations (i.e., the sum over all operations, of the amortized cost per operation) gives a lower bound on the total actual cost of the sequence.

[TRUE/[✓ FALSE]]
Consider a directed graph G in which every edge has a positive edge weight. Suppose you create a new graph G' by replacing the weight of each edge by the negation of its weight in G. For a given source vertex s, you compute all shortest path from s in G' using Dijkstra's algorithm. True or False: the resulting paths in G' are the longest (i.e., highest cost) simple paths from s in G.

[[✓ TRUE]/FALSE]
The Array [25, 19, 24, 13, 18, 22, 24] is a max_heap.

[[✓ TRUE]/FALSE]
A spanning tree of a given undirected, connected graph G(V,E) can be found in O(|E|) time.

2) 12 pts

Let G = (V, E) be a connected undirected graph and let v be a vertex in G. Let T by the depth-first search tree of G starting from v, and let U be the breadth-first search tree of G starting from v. Prove that the height of T is at least as great as the height of U.

Solution:

Let the depth of U be d and let w be a vertex on level d of U. We know that the BFS tree from v indicates the shortest-path distance from v to every node (counting each edge as distance 1). Thus there is no path in G of length less than d from v to w. If the depth of T were less than d, there would be a path in G of length less than d from v to w, given by the path in T. This is impossible, so T cannot have depth less than d.

Grading criteria:

12 points if the algorithm is correct.

- up to 8 points if the algorithm is incorrect however the explanation was closely related.
- 2 points for running complexity
- 2 points for running describing the algorithms
- up to 4 points for the mentioned algorithm
- 4 pints correctness of the proof

3) 25 pts
A new startup FastRoute wants to route information along a path in a communication network, represented as a graph. Each vertex represents a router and each edge a wire between routers. The wires are weighted by the maximum bandwidth they can support. FastRoute comes to you and asks you to develop an algorithm to find the path with maximum bandwidth from any source s to any destination t. As you would expect, the bandwidth of a path is the minimum of the bandwidths of the edges on that path; the minimum edge is the bottleneck. Explain how to modify Dijkstra's algorithm to do this.

A) Should there be any changes to the underlying data structure used in Dijkstra's? If so, how?

Data structure change: We'll use a max priority queue instead of a min priority queue used in Dijkstra's.

Rubric: (5 pts.)

1. Mentioning max heap/priority queue would receive 5 pts.
2. Vaguely mentions such as "put maximum entries first" might receive partial credits depending on follow-up description in B) and C)
3. Answer with "No" but subsequently modified the algorithm in B) and C), such as negating the weights, if correct, will receive full credits.

B) How would the data structure be initialized?

Initialization of the priority queue: Initially all nodes will have a distance (bandwidth) of zero from s, except the starting point s which will have a bandwidth of ∞ to itself.

Rubric: (6 pts.)

1. 1 pt. for mentioning the key-value for the heap.
2. 2 pts. for initializing other nodes. (Non-positive number will also receive full credits, assuming the graph is connected)
3. 3 pts for initializing the starting node s to +infinity/INT_MAX etc.
4. Depending on A) and C), the correct answer might be different, but the grading will be based on the same 1/2/3 pts. partition, since any modification to the Dijkstra's algorithm would not change the central role of the heap.

C) How would the rest of the algorithm be modified?

Change in relaxation step: Based on the definition of a path's bandwidth, instead of the distance from s to u through u's neighbor v being du = (dv + Cvu), the bandwidth of a path from s to u through u's neighbor v will be du = min (dv , Cvu), because the bandwidth of a path is equal to the bandwidth of its lowest bandwidth edge. Therefore, in the relaxation step, we will be replacing:

du = min (du, dv+Cvu)

with

du = **max** (du, min (dv , Cvu))

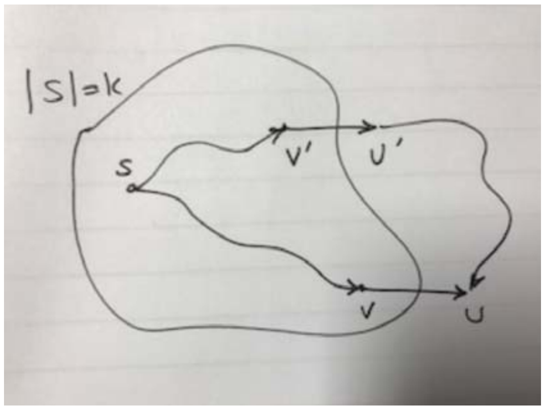
Rubric: (7 pts.)

1. Full credit for the correct relaxation modification
    a. If the student decides to describe the entire Dijkstra's algorithm, mistakes in description will result in points deduction.
    b. Common mistakes include:
        i. du = min (du, dv+Cvu) (no partial credit)
        ii. du = min (du, min (dv , Cvu)) (partial credit)
        iii. du = min/max (dv , Cvu) (partial credit)
    c. Although we require path as output and the simple Dijkstra's algorithm does not output path directly, many common implementations do trace the path, therefore, modification regarding path output is not required and mild mistake will be overlooked.
2. If the provided answer cannot be classified as modified Dijkstra's algorithm, for example, some answers are clearly modified Prim's algorithm, then the grading criteria will be changed.
    a. You answer will be considered as incorrect for A), B) and C), therefore ineligible for full credit. (If you are selecting "edges" instead of "vertices" in your algorithm, or you select the "vertex with maximum weighted edge", then your solution will not be considered as Dijkstra's algorithm.
    b. If your algorithm is correct, documented in detail, and asymptotically optimal in computational cost, you will receive 3 pts.
        i. Must output path
        ii. Correctly initialized
        iii. Key steps (that determines the correctness and the computational cost) must be written with unambiguity, such as the data structure/operation involved.
    c. Correct algorithm includes:
        i. Prim's algorithm for maximum spanning (or MST with negated edge weights).
        ii. Modified Prim's algorithm where at each step, instead of picking the maximally weighted edges on the cut, we aggregate the edges based on end node. That is, picking the vertex with maximally weighted outgoing edges.

D) Prove that your algorithm correctly finds maximum bandwidth paths from s to all other nodes in the network.

Proof is identical to proof in Dijkstra's. We prove that at each step our algorithm finds the maximum bandwidth path to a new node in the network. Proof is by induction:
Base case: Our algorithm picks s with a bandwidth of ∞ to itself

Inductive step:



Assume we have correctly found maximum bandwidth paths to the first k≥1 nodes found in the network. We will now prove that in the next step, the algorithm finds the maximum bandwidth path to the k+1st node in the network. Let's call the k+1st node "u", and the node in S through which we found the path to u (and has a direct edge to u) is called v. Then Min (Pv, Cvu) is the maximum bandwidth path to u.

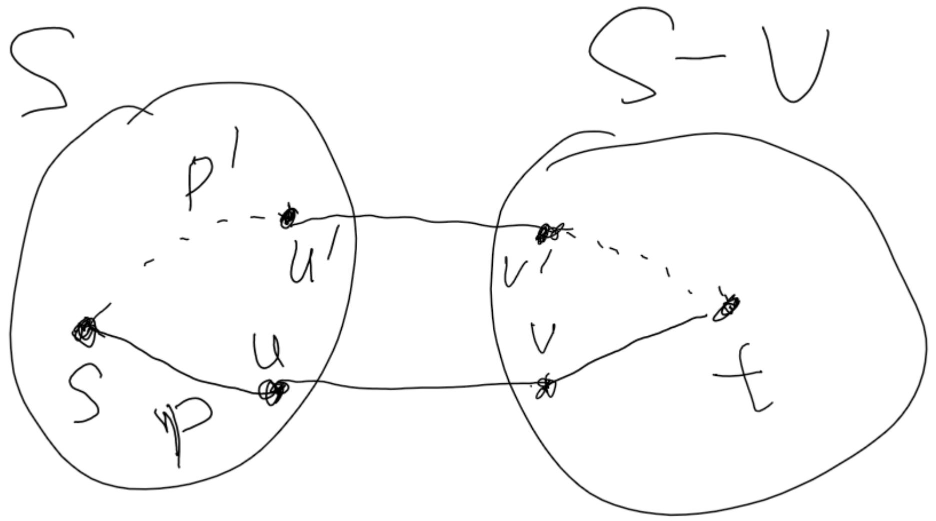
Proof by contradiction: Say there is another path to u with higher bandwidth. This path must cross the boundary of S to V-S at some point. Let's call the edge going thru the boundary v'u'. At the time that we chose u, we also could have chosen u', but we didn't because the maximum bandwidth from s to u' was lower. And since the bandwidth of the path cannot increase by going from u' to u, then this path cannot have a higher bandwidth than the one through v.

Rubric: (7 pts.)

1. If the answer to C) is incorrect, then you will not receive any credits for D).
2. Hand-waving argument will not be accepted as proof.
    a. If proof by induction, clearly provide the statement on which you would do induction. Do not miss the base case.
    b. If proof by contradiction, clearly states your assumption.
3. Partial credits will be awarded for close-to-complete proofs, such as a missing base case/clear typo in equations etc.
4. Common mistakes:
    a. Induction on the graph size instead of the size of visited nodes. i.e., |S|.

Remark: If your proof does not use the new definition of the cost of a path, then your proof is likely wrong. The argument provided by many students can be applied to the same problem that attempts to find the longest simple path, which is NP-complete. Therefore their proof of correctness for this polynomial algorithm is wrong assuming P! =NP.

For Prim's based algorithm: The path on maximum spanning tree (MST) has the maximum bandwidth.



Proof by contradiction: Assume that there exists a MST that violating our statement, there must exist a destination t, such that our path P from s to t has strictly less bandwidth than the optimal path P'. Let u-v be the bottleneck in path P, and set S be the visited node when we added u-v to our solution. Then the optimal path P' will also cross the cut (S, V-S). Denote the edge from P' on the cut as u'-v'. Since the bandwidth(P') <= w(u',v'), and w(u,v)=bandwidth(P), therefore, w(u',v') > w(u, v), which indicates that by replacing u-v with u'-v', we can get a better maximum spanning tree, which contradicts with our assumption.

4) 16 pts
You are a guidance counselor in charge of putting high school students into one of two study halls. It doesn't matter how many students are in each study hall; what does matter is that certain pairs of students do not get along well and would cause a major disruption if they were placed in the same study hall. There are n students and you have a list of b pairs of students who shouldn't be placed together. Give an algorithm that determines in time O(n + b) whether it is possible to allocate the students to the two study halls without violating the b constraints. (Note that some students may occur multiple times in the list of "bad" pairs, but no student would be paired with him/herself.)

Solution:

Algorithm (14 points):
There is an assignment of students to study halls if and only if the undirected graph G = (V, E), where V is the set of n students and E is the set of b bad pairs, is bipartite. A modification of BFS will do the job in time O(n + b).

For any node in G without any edge, we can assign that node to any hall as the corresponding student has no conflict with anyone else.
Moreover, the graph G can have multiple connected component so we run the modified BFS on each connected component.
We start BFS for any arbitrary node for each connected component. In the resulting tree each node in an odd layer is assigned to hall 1 and nodes in even layers are assigned to hall 2.
Once we've made the initial assignment, we iterate over the bad pairs and check to see if the two students in each pair have been assigned to different rooms. If not, it means there is no possible assignment and the algorithm returns FAILURE.

Proof of correctness (0 points):
Based on our algorithm, if two ends of an edge (i.e., two students in a bad pair) end up in the same hall, it indicates the existence of a cycles in the graph with odd number of edges/nodes. According to class notes, a graph is bipartite iff there is no cycles in the graph with odd number of edges/nodes. Hence, if our algorithm doesn't fail, it means the graph is bipartite and consequently, the students/nodes can be separated into two disjoint sets.

Complexity (2 points):
Building the graph with n nodes and b edges takes O(n+b)
Finding isolated nodes takes O(n)
Running one (or even multiple) BFS(s) will take at most O(n+b) as there are at most n nodes and b edges that will be traversed
Final iteration over the b pairs to check the two ends takes O(b)
Total complexity: O(n+b) + O(n) + O(n+b) + O(b) = O(n+b)

Common Mistakes:

- Failing on detecting any cycles: -5 points
- Iterating bad pairs in random order and checking for conflicts: -7 points
- Not accounting for conflicts: -9 points
- Correct algorithm with O(n^2) or O(b^2) complexity: -8 points

5) 12 pts

Solve the following recurrence equation using the Master Theorem

$$T(n) = 4\left(T\left(\frac{n}{2}\right) + n\right) + \Theta(n^2)$$

Solution: T(n) = $\Theta(n^2 \lg n)$

Grading Rubrics

Without or Big O notation: -1pts

Wrong answer with claiming correct case 2: no credit

6) 15 pts
We know that binary search on a sorted array of size *n* take *O(lg n)* time. Design a similar divide-and-conquer algorithm for searching in a sorted singly linked list of size n>1.

a) Describe the steps of your algorithm in plain English (6)

Assuming we are given the size of the linked list as N and asked to search for value in the list. I make the distinction between traversal complexity (right = right->next) and comparison complexity (value > right->value), but it is not necessary.

```
    int size = N; right = left = headNode
  while (size > 1)       // iterative condition {
    int newSize = (size / 2); //middle element
        size -= newSize;//next part is half the size
    right = left; // right and left are end points

        //Traverse to middle element of the partial linked list
   for (int i = 0; (i < newSize) && (right->next!=nullptr); i++)
      right = right->next;

        //Make comparisons to the element and set iterators to iterate on the
        corresponding half of the list
    if (v == right->value) return right;
    else if (v > right->value) left = right;
   }
  return nullptr; //element not found
```

It is clear that the outer loop iterates O(log N) times where N = size of the list. For each iteration, we make 2 comparisons, so the total # of comparisons is O(log N).
The number of traversal steps is the number of times right = right->next; gets executed, which is O(N). This is because the # of iterations in the inner loop decreases by half at each iteration of the outer loop, so N/2 + N/4 + ... + 1 = N

If linear search, or divide and conquer (1, n-1) linear search, or maxheap-minheap construction, or BST: -6
If no use of linked list traversal or pointer manipulation: -3
If binary search on array or list: only 3 marks total.

b) Write a recurrence equation for the runtime complexity (6)

$T(n) = T(n/2) + O(n)$
$T(1) = 1$ (not in grading criteria)
If either terms are incorrect. : -4

c) Solve the equation using a recursion tree method (3)

N/2 + N/4 + ... + 1 = N

O(log N) comparisons, O(1) memory, O(N) traversal steps

If no distinction is made between traversal and comparison and traversal, then O(N)is accepted solution.

If Master theorem is used: -1

If tree is incorrect: -2

If final solution is incorrect: -2