CS570
Analysis of Algorithms
Spring 2014
Exam I

Name: ______________________

Student ID: __________________

____On Campus ____DEN

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

2 hr exam
Close book and notes

If a description to an algorithm is required please limit your description to within 150 words, anything beyond 150 words will not be considered.

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

F **[TRUE/FALSE]**
The number of cycles in a bipartite graph is never odd.

T **[TRUE/FALSE]**
Every tree is a bipartite graph.

T **[TRUE/FALSE]**
If a weighted undirected graph has two MSTs, then it has a cycle C such that the maximum weight edge in C is not unique.

T **[TRUE/FALSE]**
If T is an MST for a weighted undirected graph, then T remains being an MST even if the edge weights are doubled.

F **[TRUE/FALSE]**
Given a binary max-heap with *n* elements**,** the time complexity of extracting the smallest element from the heap is O(log n)

F **[TRUE/FALSE]**
If *f*, *g*, and *h* are positive increasing functions with *f* in *O(h)* and *g* in *Ω(h)*, then the function *f+g* must be in *Θ(h)*.

T **[TRUE/FALSE]**
The array [20 15 18 7 9 5 12 3 6 2] forms a binary max-heap.

T **[TRUE/FALSE]**
The number of spanning trees in a fully connected graph with *n* vertices goes up exponentially with respect to *n*.

T **[TRUE/FALSE]**
If the edges in a connected graph all have unit costs, then the shortest path between two nodes in found faster using BFS than it is using Dijkstra's algorithm.

F **[TRUE/FALSE]**
Given a problem with input of size *n*, a solution with O(n) time complexity always costs less in computing time than a solution with O(n^2) time complexity.

2) 16 pts

You are given a weighted directed graph $G = (V, E)$ and the shortest path **distances** $\delta(s, u)$ from a source vertex s to every other vertex in G. However, you are not given $\pi(u)$ (the predecessor pointers). With this information, give an algorithm to find a shortest path from s to a given vertex t in $\boxed{O(|V| + |E|) \text{ time.}}$

3) 16 pts

You are given n jobs each with a known start and end time. There are n identical processors. Two jobs with overlapping running times cannot be assigned to the same processor. Describe an algorithm to assign jobs to processors such that the number of processors utilized is minimized.

4) 16 pts

The police department in a city has made all streets one-way. The mayor contends that there is still a way to drive legally from any intersection in the city to any other intersection, but the opposition is not convinced. A computer program is needed to determine whether the mayor is right. However, the city elections are coming up soon, and there is just enough time to run a *linear-time* algorithm.

i) Formulate this as a graph problem and design a linear-time algorithm. Explain why it can be solved in linear time.

ii) Suppose it now turns out that the mayor's original claim is false. She next makes the following claim to supporters gathered in the Town Hall: “If you start driving from the Town Hall (located at an intersection), navigating one-way streets, then no matter where you reach, there is always a way to drive legally back to the Town Hall.” Formulate this claim as a graph problem, and show how it can also be verified in linear time.

5) 16 pts

Consider the following modification to Dijkstra's algorithm for single source shortest paths to make it applicable to directed graphs with negative edge lengths.
If the minimum edge length in the graph is $-w < 0$, then add $w+1$ to each edge length thereby making all the edge lengths positive. Now apply Dijkstra's algorithm starting from the source s and output the shortest paths to every other vertex.
Does this modification work ? Either prove that it correctly finds the shortest path starting from s to every vertex or give a counter example where it fails.

6) 16 pts

Design a data structure that has the following properties:

- Find median takes $O(1)$.
- Extract-Median takes $O(\log n)$.
- Insert takes $O(\log n)$.
- Delete takes $O(\log n)$.

(**Hint:** Your Data Structure should use a min-heap and a max-heap simultaneously where half of the elements are in the max-heap and the other half are in the min-heap).

a) Describe how your data structure will work.

b) Give an $O(\log n)$ algorithm for inserting a new element in the data structure.

c) Give an $O(\log n)$ algorithm for Extract-Median a new element in the data structure.

.

Question 1:

1. False. A bipartite graph cannot contain cycle of odd length, however the number of cycles could be odd. For instance, consider the graph with the edge set {(a,b),(b,c),(c,d),(d,a)}. The number of edges is 1, which is odd.

2. True. A tree by definition does not contain cycles, and hence does not contain cycles of odd length and is bipartite.

3. True. Let T_1 and T_2 be two distinct MSTs. Hence there exists an edge e that is in T_1 but not T_2. Add e to T_2 thereby inducing a (unique) cycle C. There are at least two edges of maximum weight in C (for if there were a unique maximum weight edge in C, then it cannot be in an MST thereby contradicting the fact that it is either in T_1 or T_2).

4. True. The claim in question follows from the following two facts: (i) the weight of a tree is linear in its edge weights and (ii) every spanning tree is has exactly |V|-1 edges. Hence the weight of a spanning tree under the modification goes from w(T) to 2w(T). Thus an MST of the original graph remains an MST even after the modification.

5. False. The max element in a min heap with n elements is one of the leafs. Since there are Theta(n) leafs and the structure of min heap does not carry any information on how the leaf values compare, to find a max element takes at least Omega(n) time.

6. False. Consider for instance f(n) = n and g(n)=n^2. Clearly n+n^2 is not in Theta(n).

7. True.

8. True. The number of spanning trees of a completely connected graph with n vertices is n^(n-2) (Cayley's formula) which grows at least exponentially in n. Even without the knowledge of Cayley's formula, it is easy to derive Omega(2^n) lower bound.

9. True. If the edge lengths are identical, then BFS does find shortest paths from the source. BFS takes O(|V|+|E|) where as Dijkstra even with a Fiboannci heap implementation takes O(|E| + |V| log |V|)

10. False. The O() notation merely gives an upper bound on the running time. To claim one is faster than other, you need an upper bound for the former and a lower bound (Omega) for the latter.

Question 2:

Algorithm1:

1. Construct the adjacent list of G^{rev} in order to facilitate the access to the incoming edges to each node.

2. Starting from node t, go through the incoming edges to t one by one to check if the following condition is satisfied:

$$\delta(s,t) == e(u,t) + \delta(s,u), \ \ (u,t) \in E$$

3. There must be at least one node u satisfying the above condition, and this u must be a predecessor of node t.
4. Keep doing the same checking operation recursively on the predecessor node to get the further predecessor node until node s is reached.

Algorithm2:

Run a modified version of Dijkstra algorithm without using the priority queue. Specifically,

```
1. Set S={s}.
2. While (S does not include t)
        Check each node u satisfying $u \notin S,\ (v,u) \in E,\ v \in S$, if the following
        condition is satisfied:
                $\delta(s,u) == e(v,u) + \delta(s,v)$
        If yes, add u into S, and v is the predecessor of u.
   endWhile
```

Complexity Analysis:

In either of the above algorithms, each directed edge is checked at most once, the complexity is O(|V|+|E|).

Question 3:

Let S_i's and E_i's are start times and end times of all jobs. Let's A is an array of size (2×2n):

```
For i = 1 to n
    A(1,2i-1) = $S_i$
    A(2,2i-1) =1
    A(1,2i) = $E_i$
    A(2,2i) =0
End
Sort A based on the first row
m = 1
P = 0
```

```
Max = 0
While m < =2n
  If A(2,m) then
     P + +
     B(m) = P
     If Max < P
        Max = P
     endif
  Else
    P - -
  endif
endwhile
Return Max , B
```

Complexity = O(nlogn)

If you come up with O(n^2) you get half of the credit.

Question 4:

i) The mayor is merely contending that the graph of the city is a strongly-connected graph, denoted as G. Form a graph of the city (intersections become nodes, one-way streets become directed edges). Suppose there are n nodes and m edges. The algorithm is:

(1) Choose an arbitrary node s in G, and run BFS from s. If all the nodes are reached the BFS tree rooted at s, then go to step (2), otherwise, the mayor's statement must be wrong. This operation takes time $O(m+n)$

(2) Reverse the direction of all the edges to get the graph G^{inv}, this step takes time $O(m+n)$

(3) Do BFS in G^{inv} starting from s. If all the nodes are reached, then the mayor's statement is correct; otherwise, the mayor's statement is wrong. This step takes time $O(m+n)$.

Explanation: BFS on G tests if s can reach all other nodes; BFS on G^{inv} tests if all other nodes reach s. If these two conditions are not satisfied, the mayor's statement is wrong obviously; if these two conditions are satisfied, any node v can reach any node u by going through s.

ii) Now the mayor is contending that the intersections which are reachable from the city form a strongly-connected component. Run the first step of the previous algorithm (test to see which nodes are reachable from town hall).

Remove any nodes which are unreachable from the town hall, and this is the component which the mayor is claiming is strongly connected. Run the previous algorithm on this component to verify it is strongly connected.

Question 5:

No, the modification does not work. Consider the following directed graph with edge set {(a,b),(b,c),(a,c)} all of whose edge weights are -1. The shortest path from a to c is ((a,b),(b,c)) with path cost -2. In this case, w=1 and if 2 is added to every edge length before running Dijkstra starting from a, then Dijkstra outputs (a,c) as the shortest path from a to c which is incorrect.

Another way to reason why the modification does not work is that if there were a directed cycle in the graph (reachable from the chosen source) whose total weight is negative, then the shortest paths are not well defined since you could traverse the negative cycle multiple times and make the length arbitrarily small. Under the modification, all edge lengths are positive and hence clearly the shortest paths in the original graph are not preserved.

Remark: Many students who claimed that the modification works made this common mistake. They claimed that since the transformation preserved the relative ordering of the edges by their weights, the algorithm should work. This is incorrect. Even though the edge lengths are increased by the same amount, the path lengths change as a function of the number of edges in them.

Question 6:

a) We use the $\left\lceil \frac{n}{2} \right\rceil$ smaller elements to build a max heap. We use the remaining $\left\lfloor \frac{n}{2} \right\rfloor$ elements to build a min heap. The median will be at the root of max heap (We assume the case of even *n*, median is $\frac{n}{2}th$ element when sorted in increasing order (5 pts).
Part (a) grading break down:
- Putting the smaller half elements in max heap and the other half in min heap (4 pts).
- Mention where the median will be at (1 pts).

b) For a new element *x*, we compare *x* with current median (root of max heap). If $x < median$ we insert *x* into max heap. Otherwise we insert *x* into min heap. If $size(maxHeap) > size(minHeap) + 1$, then we ExtractMax() on max heap and insert the extracted value into the min heap. Also if $size(minHeap) > size(maxHeap)$ we ExtractMin() on min heap and insert the extracted value in max heap. (6 pts).
Part (b) grading break down:
- Correctly identifying which heap to insert the value into (2 pts).

- Maintaining the size of the heaps correctly (4 pts).
- If you only got the idea that you have to maintain the size of the heaps equal but don't do it correctly (1 pt).

c) ExtractMax() on max heap. If after extraction $size(maxHeap) < size(minHeap)$ then we ExtractMin() on min heap and insert the extracted value in max heap (5 pts).
Part (c) grading break down:
- Correctly identifying which root to extract and retaining heap property (1 pt).
- Maintaining the size of heaps correctly (4 pts).
- If you only got the idea that you have to maintain the size of the heaps equal but don't do it correctly (1 pt).