CS570

Analysis of Algorithms

Fall 2013

Exam III

Name: ______________________

Student ID: __________________

____Tuesday/Thursday Session ____Wednesday Session ____DEN

	Maximum	Received
Problem 1	20	
Problem 2	20	
Problem 3	20	
Problem 4	20	
Problem 5	20	
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.

[TRUE/FALSE]
Let A and B be decision problems. If A is polynomial time reducible to B and B is in NP-Complete, then A is in NP.

[TRUE/FALSE]
In a network with source s and sink t where each edge capacity is a positive integer, there is always a max s-t flow where the flow assigned to each edge is an integer.

[TRUE/FALSE]
Let ODD denote the problem of deciding if a given integer is odd. Then ODD is polynomial time reducible to 3-SAT.

NP-C NP $\therefore NP \le_p NP\text{-}C$

[TRUE/FALSE]
Not every decision problem in P has a polynomial time certifier.

[TRUE/FALSE]
The set of all vertices in a graph is a vertex cover.

[TRUE/FALSE]
A minimum spanning tree of a connected undirected graph remains being a minimum spanning tree even if each edge weight is doubled.

[TRUE/FALSE]
A minimum spanning tree of a bipartite graph is not necessarily a bipartite graph.

[TRUE/FALSE]
Dijkstra's algorithm can always find the shortest path between two nodes in a graph as long as there is no negative cost cycle in the graph.

[TRUE/FALSE]
Given a binary max heap of size *n*, the complexity of finding the smallest number in the heap is O(log *n*).

[TRUE/FALSE]
Given a graph G=(V,E) and an approximation algorithm that solves the vertex cover problem in G with an approximation ratio *r*, then this algorithm can also provide a solution to the independent set of G with the same approximation ratio *r*.

2) 20 pts

At a dinner party, there are n families $\{a_1,a_2,\ldots,a_n\}$ and m tables $\{b_1,b_2,\ldots,b_m\}$. The i^{th} family a_i has g_i members and the j^{th} table b_j has h_j seats. Everyone is interested in making new friends and the dinner party planner wants to seat people such that no two members of the same family are seated in the same table. Design an algorithm that decides if there exists a seating assignment such that everyone is seated and no two members of the same family are seated at the same table.

Construct the following network G=(V,E). For every family introduce a vertex and for every table introduce a vertex. Let a_i denote the vertex corresponding to the ith family and let b_j denote the vertex corresponding to the jth table. From every family vertex a_i to every table vertex b_j, add an edge (a_i,b_j) of capacity 1. Add two more vertices s and t. To every family vertex a_i add an edge (s,a_i) of capacity g_i. From every table vertex b_j add an edge (b_j,t) of capacity h_j.

Claim: There exists a valid seating if and only if the value of max flow from s to t in the above network equals g_1+g_2+...+g_n.

Proof of Claim: Assume there exists a valid seating, that is a seating where every one is seated and no two members in a family are seated at a table. We construct a flow f to the network as follows. If a member of the ith family is seated at the jth table in the seating assignment, then assign a flow of 1 to the edge (a_i,b_j). Else assign a flow of 0 to the edge (a_i,b_j). The edge (s,a_i) is assigned a flow equaling the number of members in the ith family that are seated (which since the seating is valid equals g_i). Likewise the edge (b_j,t) is assigned a flow equaling the number of seats taken in the table b_j (which since the seating is valid is at most h_j). Clearly the assignment is valid since by construction the capacity and conservation constrains are satisfied. Further, the value of the flow equals g_1+g_2+...+g_n.

Conversely, assume that the value of the max s-t flow equals g_1+g_2+...+g_n. Since the capacities are integers, by the correctness of the Ford-Fulkerson algorithm, there exists a maxflow (call f) such that the flow assigned to every edge is an integer. In particular, every edge between the family vertices and table vertices has a flow of either 0 or 1 (since these edges are of capacity 1). Construct a seating assignment as follows, seat a person of the ith family at the jth table if and only if f(a_i,b_j) is 1. By construction at most one member of a family is seated at a table. Since the value of f equals the capacity of the cut ({s},V-{s}), every edge out of s is saturated. Thus by flow conservation at a_i, for every a_i the number of edges out of a_i with a flow of 1 is g_i. Thus in the seating assignment, every one is seated. Further, since the flow f(b_j,t) out of b_j is at most h_j, at most h_j persons are seated at table b_j. Thus we have a valid seating.

Remark. You can solve the problem using "circulation" as well.

One way is to modify the above network by adding (for every a_i) a lower bound of g_i to the edge (s,a_i) and adding an edge (t,s) of infinite capacity. Then the modified network has a valid circulation if and only if there is a valid seating.

Another way is to remove s and its incident edge from the above network and add at t a demand of g_1+g_2+..+g_n and at each a_i a demand of -g_j. The modified network has a circulation if and only if there is a valid seating.

3) 20 pts

A decision version of the subset sum problem is as follows: Given a set of n integer numbers $A = \{a_1, a_2, \ldots, a_n\}$ and a target number *t*. Determine if there is a subset of numbers in A that add up precisely to *t*? That is, the output is *yes* or *no*. Describe an algorithm (and provide pseudo-code) to solve this problem, and analyze its complexity.

Solution: Note that this is not to show the subset-sum is an NP-complete problem. In this question, students are asked to provide a specific solution to the problem without restriction on the complexity.

Use dynamic programming:

Let S[i,j] shows if there exists a subset of $\{ a_1, a_2, \ldots, a_i \}$ that add up to j, where $0 \leq j \leq t$. S[i,j] is true or false.

Initialize: S[0,0] = true; S[0,j] = false if $j \neq 0$

Recurrence formula:

$S[i,j] = S[i-1,j]$ OR $S[i-1, j-a_i]$

Output is S[n,t]

Complexity is O(t.n) – pseudo-polynomial

4) 20 pts

Given a binary search tree T, its root node *r*, and two random nodes *x* and *y* in the tree. Find the lowest common ancestry of the two nodes *x* and *y*. Note that a parent node *p* has pointers to its children *p.leftChild()* and *p.rightChild()*, a child node does **not** have a pointer to its parent node. The complexity must be *O*(log *n*) or better, where *n* is the number of nodes in the tree.

Recall that in a binary search tree, for a given node *p*, its right child *r*, and its left child *l*, *r.value()* ≥ *p.value()* ≥ *l.value()*. Hint: use divide and conquer

Solution: Use divide and conquer:

```
Node  LCA(r, x, y){
        If (x.value() < r.value && y.value() > r.value())
                Return r;
        Else If(x.value() < r.value && y.value() < r.value())
                Return LCA(r.leftChild() x, y);
        Else
                Return LCA(r.rightChild() x, y);
}
```

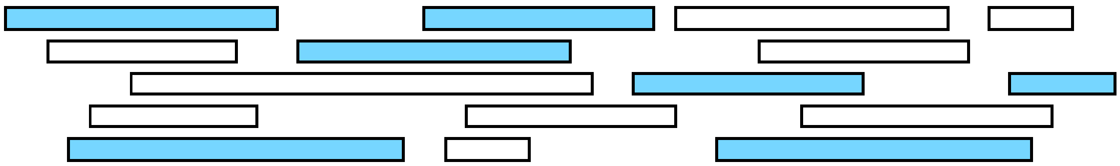
Complexity O(log n)

Note: Using the property of the binary search tree to decide if two nodes belong to the same branch or different branches of a parent node is crucial for maintaining a complexity of O(log n). Simply using the binary search to answer this question will make the overall complexity of the algorithm to be $O(\log^2 n)$.

Note that the property of a classic binary search tree is that no duplicates are allowed, so students need not care about the fact that the elements in the tree can be equal. Some few modern implementations allow duplicates in the binary search tree, but that is specifically implemented according to the requirements of each application and is not considered as a pure binary search tree.

5) 20 pts

Let X be a set of n intervals on the real line. A subset of intervals $Y \subset X$ is called a tiling path if the intervals in Y cover the intervals in X, that is, any real value that is contained in some interval in X is also contained in some interval in Y. The size of a tiling cover is just the number of intervals.



A set of intervals. The seven shaded intervals form a tiling path

The Tiling problem can be posed as follows: Given a set of intervals with their start times and end times, find the smallest tiling path of X.

Decide if

a) This problem can be solved in polynomial time. If so, provide a solution and analyze its complexity, or

b) State the decision version of the problem and prove that it is NP-complete

It can be solved in polynomial time using the following greedy approach:
First sort all intervals based on their start times in ascending order. If the start time are identical, sort them based on the length of the intervals in descending order.
At each step, pick up the first interval ϑ from X and add it to Y, deletes all intervals included completely in that interval, and change the start time of the interval whose start time is in ϑ but they finish after ϑ to the finish time of . Do the sorting based on the above explanation and then continue until we do not have any interval left in X.

Additional Space

Additional space