CS570
Analysis of Algorithms
Fall 2014
Exam III

Name: _____

Student ID: _____

Email:_____

Wednesday Evening Section

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	16	
Problem 4	16	
Problem 5	16	
Problem 6	16	
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.

[/FALSE]
All the NP-hard problems are in NP.

[/FALSE]
Given a weighted graph and two nodes, it is possible to list all shortest paths between these two nodes in polynomial time.

[TRUE/]
In the memory efficient implementation of Bellman-Ford, the number of iterations it takes to converge can vary depending on the order of nodes updated within an iteration

[/FALSE]
There is a feasible circulation with demands $\{d_v\}$ if $\sum_v d_v = 0$.

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

[TRUE/]
If a problem can be reduced to linear programming in polynomial time then that problem is in P.

[/FALSE]
If we can prove that P ≠ NP, then a problem A ∈ P does not belong to NP.

[/FALSE]
If all capacities in a flow network are integers, then every maximum flow in the network is such that flow value on each edge is an integer.

[/FALSE]
In a dynamic programming formulation, the sub-problems must be mutually independent.

[TRUE/]
In the final residual graph constructed during the execution of the Ford–Fulkerson Algorithm, there's no path from sink to source.

2) 16 pts
In the Bipartite Directed Hamiltonian Cycle problem, we are given a bipartite directed graph $G = (V; E)$ and asked whether there is a simple cycle which visits every node exactly once. Note that this problem might potentially be easier than Directed Hamiltonian Cycle because it assumes a bipartite graph. Prove that Bipartite Directed Hamiltonian Cycle is in fact still NP-Complete.

Solution:

i) This problem is NP. Given a candidate path, we just check the nodes along the given path one by one to see if every node in the network is visited exactly once and if each consecutive node pair along the path are joined by an edge.

ii) To prove the problem is NP-complete, we reduce Directed Hamiltonian Cycle (DHC) problem to Bipartite Directed Hamiltonian Cycle (BDHC) problem.

Given an arbitrary directed graph G, we split each vertex v in G into two vertex v_in and v_out. Here v_in connects all the incoming edges to v in G; v_out connects all the outgoing edges from v in G. Moreover, we connect one directed edge from v_in to v_out. After doing these operations for each node in G, we form a new graph G'.

Here G' is bipartite graph, because we can color each v_in "blue" and each v_out "red" without any coloring conflict.

If there is DHC in G, then there is a BDHC in G'. We can replace each node v on the DHC in G into consecutive nodes v_in and v_out, and (v_in, v_out) is an edge in G'. Then the new path is a BDHC in G'.

On the other hand, if there is BDHC in G', then there is a DHC in G. Note that if v_in is on the BDHC, v_out must be the successive node on the BDHC. Then we can merge each node pair (v_in, v_out) on the BDHC in G' into node v and form a DHC in G.

In sum, G has DHC if and only if G' has a BDHC. This completes the reduction, and we confirm that the given problem is NP-complete

3) 16 pts

A tourism company is providing boat tours on a river with n consecutive segments. According to previous experience, the profit they can make by providing boat tours on segment i is known as a_i. Here a_i could be positive (they earn money), negative (they lose money), or zero. Because of the administration convenience, the local community of the river requires that the tourism company should do their boat tour business on a contiguous sequence of the river segments, i.e, if the company chooses segment i as the starting segment and segment j as the ending segment, all the segments in between should also be covered by the tour service, no matter whether the company will earn or lose money. The company's goal is to determine the starting segment and ending segment of boat tours along the river, such that their total profit can be maximized. Design an efficient algorithm to achieve this goal, and analyze its run time (Note that brute-force algorithm achieves $\Theta(n^2)$, so your algorithm must do better.)

Solution1:

Using dynamic programming:

Define OPT(i) as the maximum total profit the company can get when running the boat tours on a contiguous sequence of segments starting at segment i.

Base case: OPT(n) = a_n.

Recursive relation: OPT(i) = max{OPT(i+1)+a_i, a_i}, for 1<= i < n

Algorithm:

For i = n, …., 1
    Compute OPT(i).
End

Maximum profit = max_{i} {OPT(i)}. Denote i* as the starting index which achieves the maximum profit, then i* is the optimal starting segment

For i = i*,……, n
    If (OPT(i) = = a_i)
        Set j* = i;
        Break;
    End
End
The value j* is the index of the optimal ending segment.

Complextiy: Computing all the OPT values takes time O(n), comparing all OPT values and find the maximum profit and the corresponding starting segment takes time O(n). Finding the optimal ending segment takes time O(n). In sum, the algorithm takes time O(n)

Remark: in this solution, we implicitly assume that the company must provide the boat tour, while providing no boat tour is not a choice. If you consider providing no boat tour also as a choice, it is also treated as a correct solution, however, the step of computing the maximum profit above solution should be modify as follows:

Maximum profit = max{0, max_{i} {OPT(i)}}.

Correspondingly, if 0 is the best result you can get, you need to claim that providing no boat tour is the best solution, and there is no need to confirm the starting segment or ending segment.

Solution 2 (outline):

Using divide and conquer.

i) Divide operation: Divide the profit sequences of the n consecutive segments, denoted as A[1,n], as two part:
a_1,…..a_{n/2} and a_{n/2+1},…..a_n, denoted as A[1,n/2] and A[n/2+1, n] respectively.

ii) Merge operation:
Suppose you successfully find the maximum profit in A[1,n/2] and A[n/2+1, n], denoted as P_left(1,n), P_right(1,n), and the corresponding contiguous sequence of segements, denoted as B_left(1,n) and B_right(1,n)
Then the optimal contiguous segment can be in one of the three cases:
a) B_left(1,n)
b) B_right(1,n)
c) An optimal contiguous segment sequence crossing A[1,n/2] and A[n/2+1, n], denoted as B_cross(1,n).

For B_cross(1,n), denote the corresponding profit as P_cross(1,n). The method to confirm B_cross(1,n) and P_cross(1,n) is as follows:

- Starting from sequence [a_{n/2},a_{n/2+1}], compute the summation S_left(1) = a_{n/2}+a_{n/2+1} as one candidate solution for P_cross(1,n).
- Next, including a_{n/2-1} into the above sequence as [a_{n/2-1}, a_{n/2}, a_{n/2+1}], compute the summation of the three elements in the sequence as the second candidate solution: S_left(2)=S_left(1)+a_{n/2-1}.

- Keep including the element one by one to the left until a_1 is included, during each step, compute and record the summation values.
- Find S_opt_left = max_{i} {S_left(i)}, record the corresponding index of the included element that achieves the maximum, say i_cross*. Then i_cross* is the optimal starting index for B_cross(1,n);
- Starting from [a_{i_cross*},……, a_{n/2+1}] includes the element to the right one by one until a_n is included, during each step, compute the summation values in the same way as in the left part, denote the value as S_right(i).
- Find P_cross (1,n) = max_{i} {S_right(i)}, record the corresponding index of the included element that achieves the maximum, say j_cross*. Then j_cross* is the optimal ending index for B_cross(1,n);

Then
Maximum Profit = max{P_left(1,n), P_right(1,n), P_cross(1,n)}, and the corresponding optimal starting index and ending index are the ones that achieve the maximum.

iii) Before doing step ii) for A[1,n], recursively run the above procedure in each array A[1,n/2] and A[n/2+1, n] and so on.

iv) Termination condition: for A[i,j], if i==j, return a_i as the maximum profit, return i as both the optimal starting and ending index in A[i,j].

Complexity:
The Divide operation takes time O(1).
The Merge operation takes time O(n).
Define T(n) as the running time,
T(n) = 2*T(n/2) + O(n)

The complexity is O(nlog(n)).

Remark: similar as in solution 1, considering no bout tour as a choice is also treated as a correct solution.

4) 16 pts

Consider the following matching problem. There are m students $s_1, s_2, \ldots, s_m$ and a set of n companies $C = \{c_1, c_2, \ldots, c_n\}$. Each student can work for only one company, whereas company c_j can hire up to b_j students. Student s_i has a preferred set of companies $\Lambda_i \subseteq C$ at which he/she is willing to work. Your task is to find an assignment of students to companies such that all of the above constraints are satisfied and each student is assigned. Formulate this as a network flow problem and describe any subsequent steps necessary to arrive at the solution. Prove correctness.

Construction of flow network: Represent each student and each company as a separate node. Add one source node s and a sink node t for a total of $m + n + 2$ nodes. Add the following directed edges with capacities:

i. $s \to s_i$ with capacity 1 unit, for each $1 \leq i \leq m$,
ii. For each $1 \leq i \leq m$, $s_i \to c$ with capacity 1 unit for all nodes $c \in \Lambda_i$.
iii. $c_j \to t$ with capacity b_j units, for each $1 \leq j \leq n$.

Constructing the solution: Run any max-flow algorithm on the above network to get the realization of edge flows under a max flow configuration (lets call it O for brevity). If an edge $s_i \to c_j$ shows a non-zero flow in this configuration, assign student s_i to company c_j. Repeat this process for each edge between the student nodes and the company nodes to get the final assignment.

Proof of correctness: We have to show that the solution so obtained satisfies all constraints of the problem.

i. Since all outgoing edges from s_i are to the node set Λ_i, it is impossible for s_i to have a flow to a node outside Λ_i in configuration O.
ii. Since the only outgoing edge from c_j is of capacity b_j, configuration O cannot have more than b_j incoming edges of non-zero flow to node c_j. Thus, not more than b_j students can get assigned to company c_j.
iii. As s_i has a single incoming edge and multiple outgoing edges of capacity 1, configuration O cannot have more than one outgoing edge from s_i with non-zero flow. Hence, s_i can get assigned to at most one company.

We have proved that our mapping from configuration O to an assignment does not violate any of the constraints except possibly that all students might not be assigned. Note that this may happen in practice if it is impossible to assign all students while satisfying all of the given constraints. Thus, what we need to show is that if there exists a feasible assignment then configuration O will have each $s \to s_i$ edge carry a non-zero flow. Given a feasible assignment $\{(s_i, c_{\sigma(i)}), 1 \leq i \leq m\}$, by construction of the flow network, it is possible to set each edge $s_i \to c_{\sigma(i)}$ to carry 1 unit of flow. By feasibility of the assignment, company c_j gets no more than b_j incoming edges with non-zero flow, so the outgoing edge from c_j has enough capacity to carry away all incident flow on node c_j. Finally, since each s_i has an outgoing flow of 1 unit in

this assignment, all $s \rightarrow s_i$ edges can be set to have 1 unit of flow, completing the proof.

5) 16 pts

Consider a directed, weighted graph G where all edge weights are positive. You have one Star, which lets you change the weight of any one edge to zero. In other words, you may change the weight of any one edge to zero. Give an efficient algorithm using Dijkstra's algorithm to find a lowest-cost path between two vertices s and t, given that you may set one edge weight to zero. Note: you will receive 10 pts if your algorithm is efficient. You will receive full points (16 pts) if your algorithm has the same run time complexity as Dijkstra's algorithm.

Solution:

Use Dijkstra's algorithm to find the shortest paths from s to all other vertices. Reverse all edges and use Dijkstra to find the shortest paths from all vertices to t. Denote the shortest path from u to v by $u \rightsquigarrow v$, and its length by $\delta(u, v)$.

Now, try setting each edge to zero. For each edge $(u, v) \in E$, consider the path $s \rightsquigarrow u \rightarrow v \rightsquigarrow t$. If we set $w(u, v)$ to zero, the path length is $\delta(s, u) + \delta(v, t)$. Find the edge for which this length is minimized and set it to zero; the corresponding path $s \rightsquigarrow u \rightarrow v \rightsquigarrow t$ is the desired path. The algorithm requires two invocations of Dijkstra, and an additional $\Theta(E)$ time to iterate through the edges and find the optimal edge to take for free. Thus the total running time is the same as that of Dijkstra:

$O(E + V \lg V)$: based on using Fibonacci heap

$O((|V|+|E|) \log (|V|))$: based on using binary heap

6) 16 pts

You are given n rods; they are of length $l_1, l_2, \ldots, l_n$, respectively. Our goal is to connect all the rods and form a single rod. The length after connecting two rods and the cost of connecting them are both equal to the sum of their lengths. Give an algorithm to minimize the cost of connecting them to form a single rod. State the complexity of your algorithm and prove that your algorithm is optimal.

1st interpretation: At each step we connect a single rod to the set of rods that are already connected together.

Of course, the solution is to sort rods by length and connect them in the order of increasing length. To prove this is optimal, you can assume that there is an optimal solution and compare our solution to the optimal solution: At each step our solution stays ahead of (or at least does no worse than) the optimal solution.

Sort Takes O(n log n)

We can use mathematical induction to prove that we always stay ahead of the optimal solution.
Claim: at each step the cost of connecting rods in our solution is less than or equal to that of the other solution and the length of the rod after this connection is smaller than or equal in size to that of the other solution.
Base case: first two shortest rods – obviously this is the opt solution
Assuming that the cost of connecting k rods in our solution is less than or equal to that of another (optimal) solution, we can show that the cost of connecting the next (k+1st) rod will be less than or equal to the cost of connecting the k+1st rod in the other solution:
Cost of the last connection in our solution = total length of all rods 1 to k+1 (ordered by length)
Cost of the last connection in the other solution = total length of all rods 1 to k+1 (not necessarily ordered by length)
It is obvious that the cost of our last connection is smaller or equal to the cost of the other connection because the only way to get the minimum total length of k+1 rods is to pick the shortest k+1 rods.

2nd interpretation: At each step we can connect any rod or set of already connected rods to another rod or set of already connected rods.

The solution is to keep connecting the smallest two rods or sets of already connected rods.
Implementation: Place all rods in a min heap with their key values representing their length. At each step extract two elements from the set and insert the combined rod back into the min heap.
This takes a total of O(n log n) time

Fact 1: the cost contribution of a rod i to the total assembly cost is Length(i) * Level(i), where Level(i) is the level at which the rod is first assembled with other rods (level 1=root level, level 2= level below root, etc.).
Proof: By observation of cost of assembly tree

Fact2: There is an optimal assembly tree of rods in which the two smallest rods are leaf nodes at the lowest level of the tree and the children of the same parent.
Proof: let's say the two smallest rods are not leaf nodes at the lowest level of the tree, using fact 1, we can swap these rods with two rods at the lowest level of the tree and thereby reducing the total cost of the assembly tree. If the two rods are leaf nodes at the lowest level but not children of the same parent we can swap two rods to make these rods children of the same parent without changing the total cost of the tree (again based on Fact 1).

Assume that there is an optimal assembly tree T* and our solution produces tree T. We will show that our tree T is also optimal.
To do this, we apply fact 2 to T* and move the smallest rods to the lowest level of the tree as children of the same parent—without increasing the cost of T*. We then eliminate the two smallest rods at the bottom of the two trees(since these are the first two rods that are assembled in our algorithm) and assume that there is a new combined rod in place of their parent node. We will get two new trees T*' and T', where
Total assembly cost of T* = Total assembly cost of T*' + total length of the two smallest rods
Total assembly cost of T = Total assembly cost of T' + total length of the two smallest rods

Since the costs of the two new trees are reduced by the same amount we can now compare the cost of our new tree T' with the cost of the new optimal tree T*'. And show that assembly tree T' is also optimal (as compared to the optimal tree T*').

To do this, we apply fact 2 recursively and place the two smallest rods in T*' at the lowest level of the tree and under the same parent node without increasing the cost of T*'. These are the next two rods that are combined in our algorithm. We then eliminate these two rods in both trees T' and T*', etc.

By repeating the same steps (applying fact 2 and eliminating the two smallest rods from the optimal assembly tree and our tree) recursively, we will find an optimal solution that follows the same exact assembly sequence that is found in our algorithm. Therefore we can state that our algorithm also produces an optimal assembly tree.

Additional Space