CS570
Analysis of Algorithms
Spring 2015
Exam II

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	20	
Problem 3	20	
Problem 4	20	
Problem 5	20	
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]
A flow network with unique edge capacities has a unique min cut.

[TRUE/]
If a problem can be solved by dynamic programming, then it can always be solved by exhaustive search (Brute Force).

[TRUE/]
A divide and conquer algorithm acting on an input size of n can have a lower bound less than $\Theta(n \log n)$.

[/FALSE]
If a flow in a network has a cycle, this flow is not a valid flow.

[/FALSE]
In the divide and conquer algorithm to compute the closest pair among a given set of points on the plane, if the sorted order of the points on both X and Y axis are given as an added input, then the running time of the algorithm improves to O(n).

[TRUE/]
In a flow network, an edge that goes straight from *s* to *t* is always saturated when maximum *s* - *t* flow is reached.

[/FALSE]
The Bellman-Ford algorithm always fails to find the shortest path between two nodes in a graph if there is a negative cycle present in the graph.

[TRUE/]
If f is a max s-t flow of a flow network G with source s and sink t, then the capacity of the min s-t cut in the residual graph G_f is 0.

[/FALSE]
In a dynamic programming solution, the space requirement is always at least as big as the number of unique sub problems.

[/FALSE]
Decreasing the capacity of an edge that belongs to a min cut in a flow network may not result in decreasing the maximum flow.

2) 20 pts

A city is located at one node x in an undirected network $G = (V, E)$ of channels. There is a big river beside the network. In the rainy season, the flood from the river flows into the network through a set of nodes Y. Assume that the flood can only flow along the edges of the network. Let c_{uv} (integer value) represent the minimum effort (counted in certain effort unit) of building a dam to stop the flood flowing through edge (u, v). The goal is to determine the minimum total effort of building dams to prevent the flood from reaching the city. Give a pseudo-polynomial time algorithm to solve this problem. Justify your algorithm.

Algorithm:

1. Add node s, and add edges from s to each node in Y.
2. Set the capacity of each of these new edges as infinity.
3. Set city node x as the sink node. Then the flood network G becomes a larger network G' with source s and sink x.
4. Find the min s-x cut on G' by running a polynomial time max-flow algorithm, where you can treat the cost of building a dam on each edge to be the capacity of this edge.
5. Build a dam on each edge in the min-cut's cut-set.

Complexity:

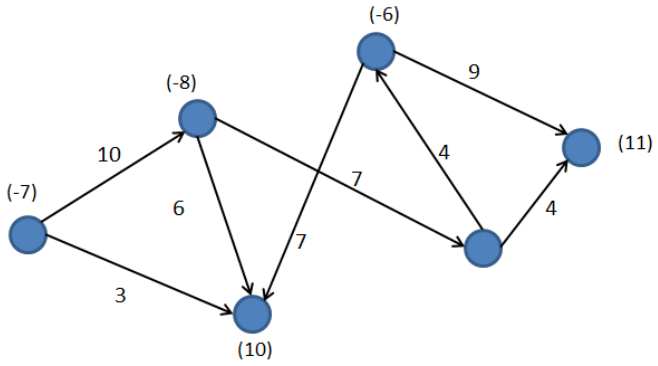
Since the construction of the new edges takes O(|Y|) times, together with a polynomial time max-flow algorithm to find the min-cut, the entire algorithm takes polynomial time.

Justification:

We're simply trying to separate x from Y by choosing edges with the minimum cost, which is a min-cut problem. But min-cut only works when Y is a single node. We fix this by creating a super source s directly connected to Y. But we want to make sure the min-cut's cut-set doesn't include any edge connecting s, because these edges do not belong to G. This is why we put the capacities arbitrarily large on these edges. Min-cut would then find the min-cost way to separate x from Y.

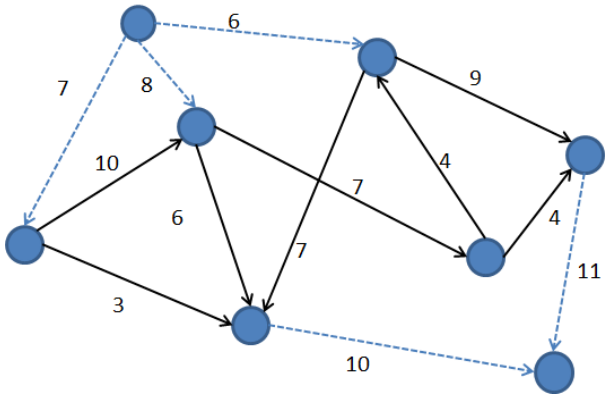
3) 20 pts

The following graph G is an instance of a circulation problem with demands. The edge weights represent capacities and the node weights (in parantheses) represent demands. A negative demand implies source.



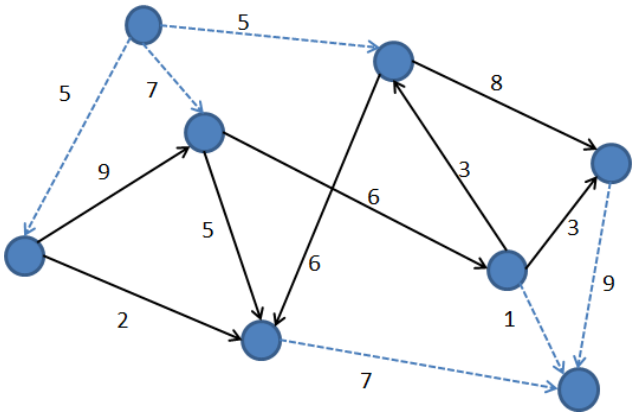
(i) Transform this graph into an instance of max-flow problem.

Solution:



(ii) Now, assume that each edge of G has a constraint of lower bound of 1 unit, i.e., one unit must flow along all edges. Find the new instance of max-flow problem that includes the lower bound constarint.

Solution:



4) 20 pts

There is a series of activities lined up one after the other, $J_1, J_2, \ldots, J_n$. The i^{th} activity takes T_i units of time, and you are given M_i amount of money for it. Also for the i^{th} activity, you are given N_i, which is the number of immediately following activities that you cannot take if you perform that i^{th} activity. Give a dynamic programming solution to maximize the amount of money one can make in T units of time. Note that an activity has to be completed in order to make any money on it. State the runtime of your algorithm.

Solution:
Recurrence formula:
If $T_i > t$ then opt(i, t)= opt(i+1, t),
Otherwise, opt(i, t)= max(opt(i+1, t), opt(i + Ni + 1, t - T_i) + Mi)

Boundary conditions: opt(i , t) = 0 for i > n and all t,
opt(i , t) = 0 for t < 1 and all i

To find the value of the optimal solution:
for i=n to 1 by -1
for t= 1 to T
If $T_i > t$ then opt(i, t)= opt(i+1, t),
Otherwise, opt(i, t)= max(opt(i+1, t), opt(i + Ni + 1, t - T_i) + Mi)
end for
end for

opt(1, T) will hold the value of the optimal solution (maximum money that can be made). Complexity is O(nT)

Given all the values in the two dimensional opt( ) array, the optimal set of activities can be found in O(n)

Overall complexity of the algorithm is O(nT) which is pseudopolynomial.

5) 20 pts

A polygon is called convex if all of its internal angles are less than 180°and none of the edges cross each other. We represent a convex polygon as an array V with n elements, where each element represents a vertex of the polygon in the form of a coordinate pair (x, y). We are told that $V[1]$ is the vertex with the least x coordinate and that the vertices $V[1], V[2], \ldots, V[n]$ are ordered counter-clockwise. Assuming that the x coordinates (and the y coordinates) of the vertices are all distinct, do the following.

Give a divide and conquer algorithm to find the vertex with the largest x coordinate in $O(\log n)$ time.

Solution:

Since $V[1]$ is known to be the vertex with the minimum x-coordinate (leftmost point), moving counter-clockwise to complete a cycle must first increase the x-coordinates and then after reaching a maximum (rightmost point), should decrease the x-coordinate back to that of $V[1]$. To see this more formally, we claim that there cannot be two distinct local maxima in the x-axis projection of the counter-clockwise tour. For the sake of contradiction, assume otherwise. Then there exists a vertical line that would intersect the boundary of the polygon at more than two points. This is impossible by convexity of the polygon since the line segments between the intersection points must lie completely in the interior of the polygon, thus contradicting our assumption. Thus, $V_x[1:n]$ is a unimodal array, and the first part of this question is synonymous with detecting the location of the maximum element in this array.

Consider the following algorithm:

(a) If $n = 1$, return $V_x[1]$.

(b) If $n = 2$, return $\max\{V_x[1], V_x[2]\}$.

(c) $k = \lceil \frac{n}{2} \rceil$.

(d) If $V_x[k] > V_x[k-1]$ and $V_x[k] > V_x[k+1]$, then return $V_x[k]$.

(e) If $V_x[k] < V_x[k-1]$ then call the algorithm recursively on $V_x[1 : k-1]$, else call the algorithm recursively on $V_x[k+1 : n]$.

Complexity: If $T(n)$ is the running time on an input of size n, then beside a constant number of comparisons, the algorithm is called recursively on at most one of $V_x[1 : k-1]$ (size $= k - 1 = \lceil \frac{n}{2} \rceil - 1 \leq \lfloor \frac{n}{2} \rfloor$) or $V_x[k+1 : n]$ (size $= n - k = n - \lceil \frac{n}{2} \rceil = \lfloor \frac{n}{2} \rfloor$). Therefore, $T(n) \leq T(\lfloor \frac{n}{2} \rfloor) + \theta(1)$. Assuming n to be a power of 2, this recurrence simplifies to $T(n) \leq T(n/2) + \theta(1)$ and invoking Master's Theorem gives $T(n) = O(\log n)$.

Additional Space