RCS570 Fall 2017: Analysis of Algorithms Exam II

	Points
Problem 1	20
Problem 2	20
Problem 3	15
Problem 4	15
Problem 5	15
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.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.

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

[TRUE/FALSE]
It is possible for a dynamic programming algorithm to have an exponential running time.

[TRUE/FALSE]
In a connected, directed graph with positive edge weights, the Bellman-Ford algorithm runs asymptotically faster than the Dijkstra algorithm.

[TRUE /FALSE]
There exist some problems that can be solved by dynamic programming, but cannot be solved by greedy algorithm.

[TRUE /FALSE]
The Floyd-Warshall algorithm is asymptotically faster than running the Bellman-Ford algorithm from each vertex.

[TRUE /FALSE]
If we have a dynamic programming algorithm with n^2 subproblems, it is possible that the space usage could be $O(n)$.

[TRUE /FALSE]
The Ford-Fulkerson algorithm solves the maximum bipartite matching problem in polynomial time.

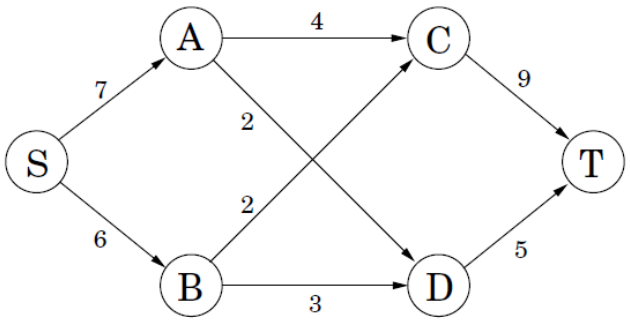
[TRUE /FALSE]
Given a solution to a max-flow problem, that includes the final residual graph G_f. We can verify in a *linear* time that the solution does indeed give a maximum flow.

[TRUE /FALSE]
In a flow network, a flow value is upper-bounded by a cut capacity.

[TRUE/FALSE]
In a flow network, a min-cut is always unique.

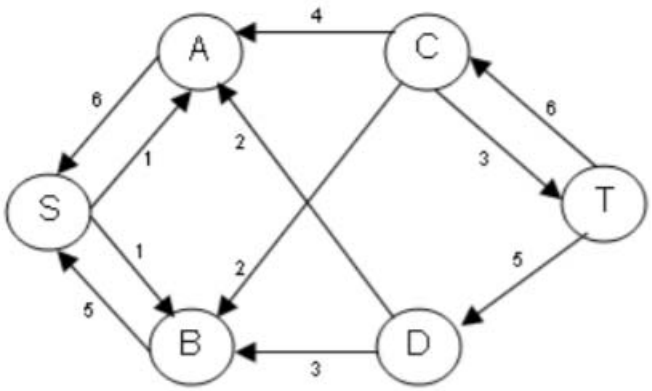
[TRUE/FALSE]
A maximum flow in an integer capacity graph must have an integer flow on each edge.

2) 20 pts.
You are given the following graph G. Each edge is labeled with the capacity of that edge.



a) Find a max-flow in G using the Ford-Fulkerson algorithm. Draw the residual graph G_f corresponding to the max flow. You do not need to show all intermediate steps. (10 pts)

solution



Grading Rubics:

Residual graph: 10 points in total

For each edge in the residual graph, any wrong number marked, wrong direction, or missing will result in losing 1 point.

The total points you lose is equal to the number of edges on which you make any mistake shown above.

b) Find the max-flow value and a min-cut. (6 pts)

Solution: f = 11, cut: ({S, A, B}, {C, D, T })

Grading Rubics:

Max-flow value and min-cut: 6 points in total

Max-flow: 2 points.

- If you give the wrong value, you lose the 2 points.

Min-cut: 4 points

- If your solution forms a cut but not a min-cut for the graph, you lose 3 points
- If your solution does not even form a cut, you lose all the 4 points

c) Prove or disprove that increasing the capacity of an edge that belongs to a min cut will always result in increasing the maximum flow. (4 pts)

Solution: increasing (B,D) by one won't increase the max-flow

Grading Rubics:

Prove or Disprove: 4 points in total

- If you judge it "True", but give a structural complete "proof". You get at most 1 point
- If you judge it "False", you get 2 points.
- If your counter example is correct, you get the rest 2 points.

Popular mistake: a number of students try to disprove it by showing that if the min-cut in the original graph is non-unique, then it is possible to find an edge in one min-cut set, such that increasing the capacity of this does not result in max-flow increase.

But they did not do the following thing:

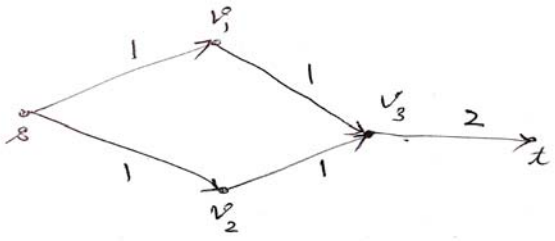
The existence of the network with multiple min-cuts needs to be proved, though it seems to be obvious. The most straightforward way to prove the existence is to give an example-network that has multiple min-cuts. Then it turns out to be giving a counter example for the original question statement.

3) 15 pts.
Given a flow network with the source *s* and the sink *t*, and positive integer edge capacities *c*. Let (S,T) be a minimum cut. Prove or disprove the following statement:
If we increase the capacity of every edge by 1, then (S,T) still be a minimum cut.

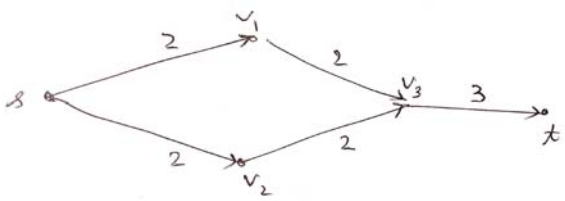
False. Create a counterexample.

An instance of a counter-example:

Initially, a (S,T) min cut is `S:{s,v_1,v_2}` and `T:{v_3,t}`



After increasing capacity of every edge by 1, (S,T) is no longer a min-cut. We now have `S':{s,v_1,v_2,v_3}` and `T':{t}`



Grading rubric:
If you try to prove the statement: -15
For an incorrect counter-example: -9
No credit for simply stating true or false.

4) 15 pts.
Given an unlimited supply of coins of denominations $d_1, d_2, \ldots, d_n$, we wish to make change for an amount C. This might not be always possible. Your goal is to verify if it is possible to make such change. Design an algorithm by *reduction* to the knapsack problem.

a) Describe reduction. What is the knapsack capacity, values and weights of the items? (10 pts.)

Capacity is C. (2 points)
values = weights = d_k. (4points)

- You need to recognize that the problem should be modeled based on the unbounded knapsack problem with description of the reduction: (5 points)
- Explanation of the verification criteria (4 points)
  Opt(j): the maximum change equal or less than j that can be achieved with d1, d2, …, dn
  Opt(0)=0
  Opt(j) = max[opt(j-di)+di] for di<=j
  If we obtain Opt(C) = C, it means the change is possible.

b) Compute the runtime of your verification algorithm. (5 pts)

O(nC) (3 points), Explanation (2 points).

5) 15 pts.
You are considering opening a series of electrical vehicle charging stations along Pacific Coast Highway (PCH). There are n possible locations along the highway, and the distance from the start to location k is $d_k \geq 0$, where $k = 1, 2, \ldots, n$. You may assume that d i < d k for i < k. There are two important constraints:

1) at each location k you can open only one charging station with the expected profit p_k, $k = 1, 2, \ldots, n$.
2) you must open at least one charging station along the whole highway.
3) any two stations should be at least M miles apart.

Give a DP algorithm to find the maximum expected total profit subject to the given constraints.

a) Define (in plain English) subproblems to be solved. (3 pts)

Let OPT(k) be the maximum expected profit which you can obtain from locations 1, 2, …, k.

Rubrics
Any other definition is okay as long as it will recursively solve subproblems

b) Write the recurrence relation for subproblems. (6 pts)

$OPT(k) = \max\{OPT(k-1), p_k + OPT(f(k))\}$
where f(k) finds the largest index j such that $d_j \leq d_k - M$, when such j doesn't exist, f(k)=0

Base cases:
$OPT(1) = p_1$
$OPT(0) = 0$

Rubrics:
Error in recursion -2pts, if multiple errors, deduction adds up
No base cases: -2pts
Missing base cases: -1pts
Using variables without definition or explanation: -2pts
Overloading variables (re-defining n, p, k, or M): -2pts

c) Compute the runtime of the above DP algorithm in terms of *n*. (3 pts)

algorithm solves n subproblems; each subproblem requires finding an index f(k) which can be done in time O(log n) by binary search.
Hence, the running time is O(n log n).

Rubric:
$O(n^2)$ is regarded as okay
O(d_n n) pseudo-polynomial is also okay if the recursion goes over all d_n values
O(n) is also okay

$O(n^3)$,O(nM), O(kn) : no credit

d) How can you optimize the algorithm to have O(*n)* runtime? (3 pts)

Preprocess distances $r_i = d_i - M$. Merge d-list with r-list.

Rubric
Claiming "optimal solution is already found in c" only gets credit when explanation about pre-process is described in either part b or c.
Without proper explanation (*e.g.* assume we have, or we can do): no credit
Keeping an array of max profits: no credit, finding the index that is closest to the installed station with M distance away is the bottleneck, which requires pre-processing.

6) 15 pts.
A group of traders are leaving Switzerland, and need to convert their Francs into various international currencies. There are n traders t 1 ,t 2 ,…,t n and m currencies c 1 ,c 2 ,…,c m . Trader t k has F k Francs to convert. For each currency c j , the bank can convert at most B j Franks to c j . Trader t k is willing to trade as much as S kj of his Francs for currency c j .(For example, a trader with 1000 Francs might be willing to convert up to 200 of his Francs for USD, up to 500 of his Francs for Japanese's Yen, and up to 200 of his Francs for Euros). Assuming that all traders give their requests to the bank at the same time, describe an algorithm that the bank can use to satisfy the requests (if it can).

a) Describe how to construct a flow network to solve this problem, including the description of nodes, edges, edge directions and their capacities. (8 pts)
Bipartite graph: one partition traders $t_1, t_2, \ldots, t_n$. Other, available currency, $c_1, c_2, \ldots, c_m$.
Connect t_k to c_j with the capacity S_{kj}
Connect source to traders with the capacity F_k.
Connect available currency c_j to the sink with the capacity B_j.
Rubrics:
- Didn't include supersource (-1 point)
- Didn't include traders nodes (-1 point)
- Didn't include currencies nodes (-1 point)
- Didn't include supersink (-1 point)
- Didn't include edge direction (-1 point)
- Assigned no/wrong capacity on edges between source & traders (-1 point)
- Assigned no/wrong capacity on edges between traders & currencies (-1 point)
- Assigned no/wrong capacity on edges between currencies & sink (-1 point)

b) Describe on what condition the bank can satisfy all requests. (4 pts)
If there is a flow f in the network with $|f| = F_1 + \ldots + F_n$. then all traders are able to convert their currencies.
Rubrics:
- Wrong condition (-4 points): Unless you explicitly included $|f| = F_1 + \ldots + F_n$, no partial points were given for this subproblem.
- In addition to correct answer, added additional condition which is wrong (-2 points)
- No partial points were given for conditions that satisfy only some of the requests, not all requests.
- Note that ΣS_{kj} and ΣB_j can be larger than ΣF_k in some cases where bank satisfy all requests.
- Note that ΣS_{kj} can be larger than ΣB_j in some cases where bank satisfy all requests.
- It is possible that ΣS_{kj} or ΣB_j is smaller than ΣF_k. In these cases, bank can never satisfy all requests because max flow will be smaller than ΣF_k.

c) Assume that you execute the Ford-Fulkerson algorithm on the flow network graph in part a), what would be the runtime of your algorithm? (3 pts)

O(n m $|f|$)

Rubrics:

- Flow($|f|$) was not included (-1 point)
- Wrong description (-1 point)
- Computation is incorrect (-1 point)
- Notation error (-1 point)
- Missing big O notation (-1 point)
- Used big Theta notation instead of big O notation (-1 point)
- Wrong runtime complexity (-3 points)
- No runtime complexity is given (-3 points)