CS570 Summer 2018: Analysis of Algorithms Exam II

	Points
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.
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]
The Ford-Fulkerson algorithm as described in class terminates when no augmenting paths can be found.

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

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

[TRUE/FALSE]
Let $(S, V - S)$ be a minimum (s,t)-cut in the network flow graph G. Let (u, v) be an edge that crosses the cut in the forward direction, i.e., $u \in S$ and $v \in V - S$. Then increasing the capacity of the edge (u, v) necessarily increases the maximum flow of G.

[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]
A dynamic programming algorithm always uses some type of recurrence relation.

[TRUE/FALSE]
The running time of a pseudo polynomial time algorithm depends polynomially on the number of integers in the input.

[TRUE/FALSE]
The running time of a pseudo polynomial time algorithm depends polynomially on the number of bits in the input.

[TRUE/FALSE]
In successive iterations of the max-flow algorithm, the total flow passing through a vertex in the graph never decreases.

[TRUE/FALSE]
If all capacities in a flow network are rational numbers (instead of integers), the Ford-Fulkerson algorithm will still correctly terminate and produce a maximum flow.

2) 20 pts.

You are given an exam with questions numbered 1, 2, 3, . . . , n. Each question i is worth pi points. You must answer the questions in order, but you may choose to skip some questions. The reason you might choose to do this is that even though you can solve any individual question i and obtain the pi points, some questions are so frustrating that after solving them you will be unable to solve any of the following fi questions. Suppose that you are given the pi and fi values for all the questions as input. Devise the most efficient algorithm you can for choosing set of questions to answer that maximizes your total points, and compute its asymptotic worst case running time as a function of n.

You may not receive more than 15 pts if your algorithm is not the most efficient one. (which is O(n))

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

Let S(i) be the maximum total score that can be obtained from questions i through n.
Common mistakes:
You must describe what OPT[i] or S[i] actually means, not describing the recurrence relation.
Partial credit is given if you wrote it in other sections.

b) Write the recurrence relation for subproblems. (7 pts)

Any such score is obtained from a set of questions that either includes i or not; in the first case, the best score is p_i +S(i+f_i + 1), and in the second case, the best score is S(i + 1).

2 points deducted if you wrote the recurrence relation in other sections.

c) Write pseudo-code to solve the above problem using the recurrence relation in part b. For full credit, your solution should run in linear time. (8 pts)

The following loop calculates the best possible total score, given a large array S with all entries initialized to 0:

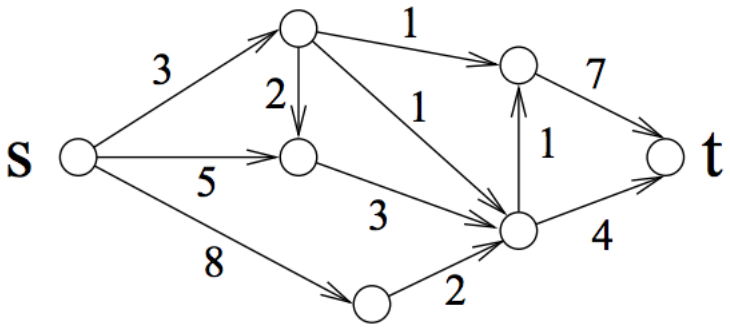
```
for i = n down to 1:
S[i] = max(p[i] + s[i+f[i]+1], s[i+1])
return S[1]
```

Base case: 2 points
Sudo Code: 6 points

d) Compute the runtime of the algorithm in terms of *n*. (2 pts)

The running time is easily seen to be O(n) (possibly with some additional tinkering to catch indices off of the end of the array).

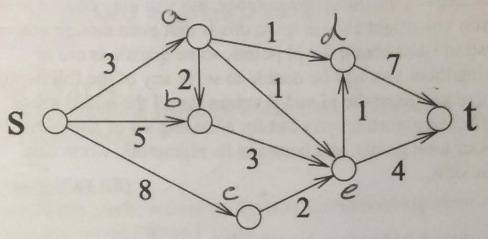
3) 20 pts.
Consider the following graph:



a) Use the Ford-Fulkerson algorithm to find the Maximum flow in this graph. Show your residual graph after each augmentation step. (15 pts)

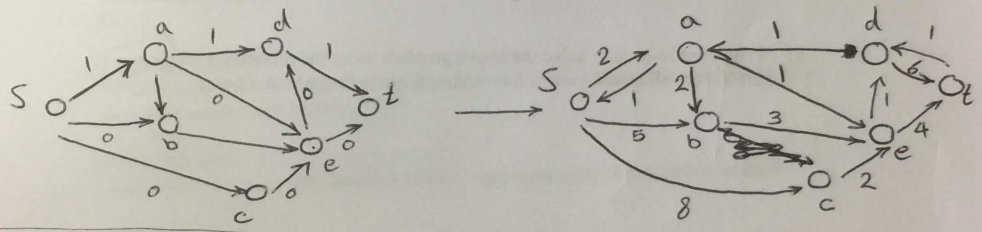
Final residual graph: 6
Max flow: 2
Steps: 7pts

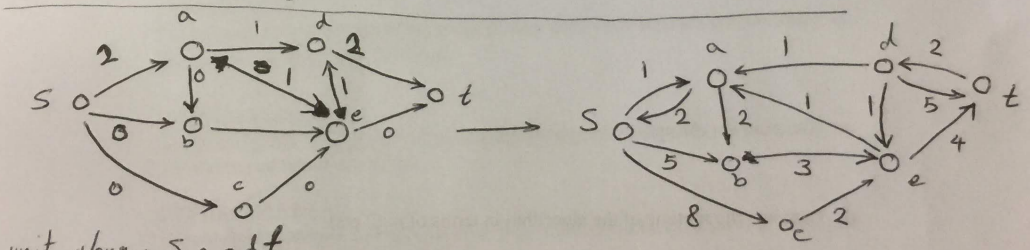
5) 20 pts.
Consider the following graph:



a) Use the Ford-Fulkerson algorithm to find the Maximum flow in this graph. Show your residual graph after each augmentation step. (15 pts)

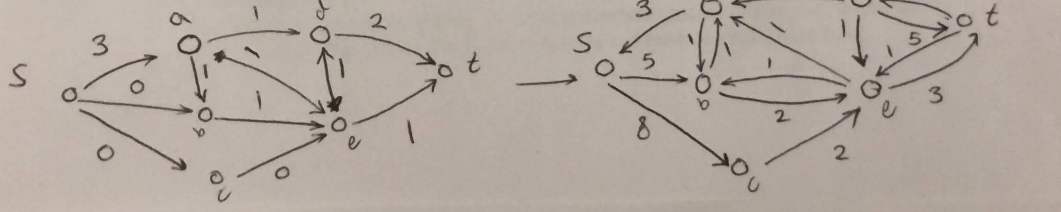
First path: s a d t augment 1 unit



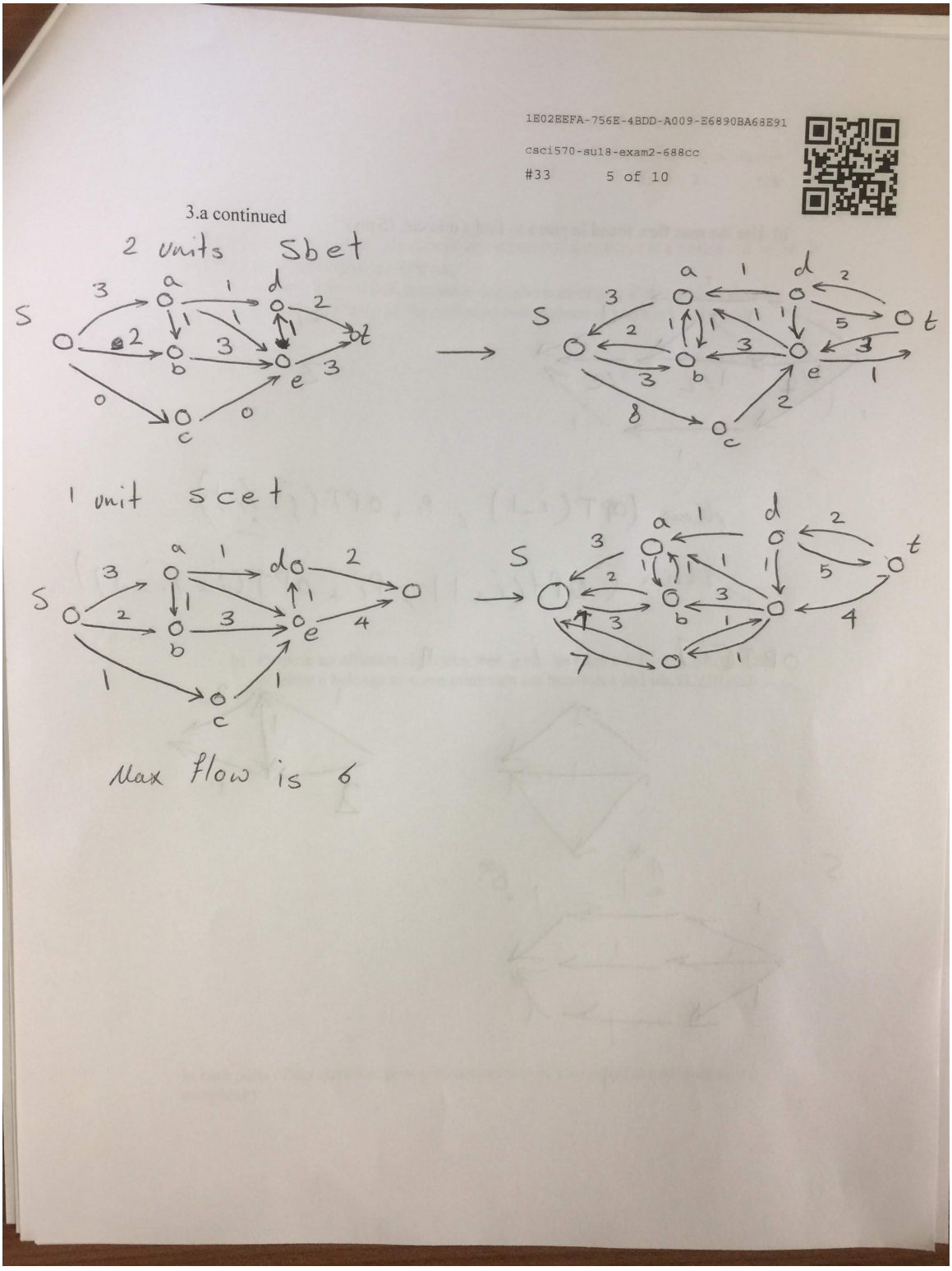


1 unit along: s a e d t

1 unit: s a b e d t



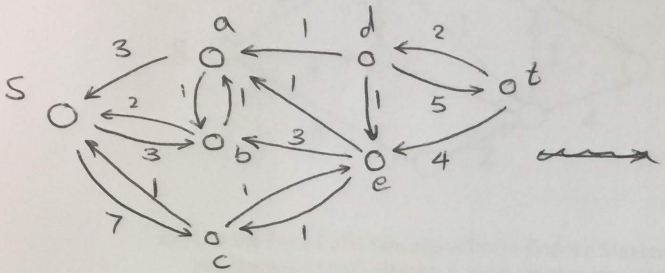
3.a continued



b) Use the max flow found in part a to find a min cut. (5 pts)

b) Use the max flow found in part a to find a min cut. (5 pts)

Final residual graph:



check which nodes are reachable from s.

$A = \{s, a, b, c, e\}$, $B = \{d, t\}$

edges in Min cut (A, B) are $\{(a,d), (e,d), (e,t)\}$

20 pts

4) Let $N\ (G(V,E),s,t,c)$ be a flow network, where s is a source, t is a sink, $c : E \rightarrow \mathbb{N}^+$ is a positive integer capacity function.

a) Propose an efficient algorithm that, given an edge e ∈ E, c(e) > 0, decides whether e belongs to all minimum cuts between s and t in G. (10 pts)

1. Run Ford-Fulkerson algorithm to find the maximum flow between s and t. Denote the maximum flow by f_1.
2. Increase the capacity of the edge e by 1.
3. Run Ford-Fulkerson algorithm to find the maximum flow between s and t. Denote the maximum flow by f_2.
4. The edge e belongs to all minimum cuts between s and t if and only if $f_2 = f_1 + 1$. Quick explanation (this is not a proper proof): if e belongs to all minimum cuts, then by increasing its capacity by one we increased the capacity of all minimum cuts by one (and all non-minimum cuts had capacity ≥ f_1 + 1), and so $f_2 = f_1 + 1$. On the other hand, if there was a minimum cut that e does not belong to it, its capacity is still f_1.

b) Propose an efficient algorithm that, given an edge e ∈ E, c(e) > 0, decides whether e belongs to some minimum cut between s and t in G. (10 pts)

1. Run Ford-Fulkerson algorithm to find the maximum flow between s and t. Denote the maximum flow by f1.
2. Decrease the capacity of the edge e by 1.
3. Run Ford-Fulkerson algorithm to find the maximum flow between s and t. Denote the maximum flow by f_2. 4. The edge e belongs to some minimum cut between s and t if and only if $f_2 = f_1 - 1$.
Quick explanation: if e belongs to some minimum cut, then by decreasing its capacity by one we decreased the capacity of this minimum cut by one, and so it is a minimum cut also in a new graph with $f_2 = f_1 - 1$. On the other hand, if e does not belong to any minimum cut, then the capacity of any minimum cut, f_1, is not affected by the change in the capacity of e. And, the capacity of any cut that e belongs to was at least $f_1 + 1$ before the decrease.

Rubric: Each part: 8 for the algorithm steps, 1 for Proof 1 for Time complexity

In both parts of the question, prove the correctness of your solution and analyze its complexity. Same as F-F

20 pts

5) You are given a collection of n points $U=\{u_1,\ldots,u_n\}$ in the plane, each of which is the location of a cell-phone user. You are also given the locations of m cell-phone towers, $C=\{c_1,\ldots,c_m\}$. A cell-phone user can connect to a tower if it is within distance Δ of the tower. For the sake of fault-tolerance each cell-phone user must be connected to at least three different towers. For each tower c_i you are given the maximum number of users, m_i that can connect to this tower.
Give a polynomial time algorithm, which determines whether it is possible to assign all the cell-phone users to towers, subject to these constraints.
(You may assume you have a function that returns the distance between any two points in O(1) time.)

We will do this by reduction to network flow. (It is also possible to reduce to the circulation problem.) Create an s-t network G=(V,E) where $V = U \cup C \cup \{s\} \cup \{t\}$. Create a unit capacity edge (u_i,c_j) if cell-phone user $u_i \in U$ is within distance Δ of cell phone tower $c_j \in C$. Create an edge from s to each $u_i \in U$ with capacity 3, and create an edge from $c_j \in C$ to t with capacity m_j.

Compute the maximum (integer) flow in this network. We claim that a feasible schedule exists if and only if all the edges exiting s are saturated in the maximum flow. To prove the "only if" part, we consider the following mapping between any valid assignment and a flow in G. We first observe that if there is a valid assignment, there is a valid assignment in which each user is assigned to exactly three towers (since if it was assigned to more than three, removing the additional assignments cannot violated any of the constraints of the problem). If user u_i is assigned to tower c_j , then $dist(u_i, c_j) \leq \Delta$ and so an edge exists between them. We apply one unit of flow along the edge (u_i, c_j). Since u_i is assigned to three towers, it has three units of flow entering it, and so the incoming edge (s, u_i) is saturated. Finally, since this is a valid assignment, the number of users assigned to any tower c_j is at most m_j , and so the flow leaving each c_j is at most m_j .

To prove the "if" part, suppose that G has a flow that saturates all the edges coming out of s. We show how to map this to a valid schedule. Because of the edge (s, u_i) is saturated, and the outgoing edges are of unit capacity, each user has three outgoing edges carrying flow. We assign u_i to these three towers. Since edges are created only when users and towers are within distance Δ, these are valid assignments. Since the flow coming out of tower j is at most m_j , tower j is assigned to more than m_j users. Thus, this is a valid assignment. The running time of the algorithm is dominated by the time for the network flow, which is $O(V^3)$, where $|V| = m + n$.

Rubric: Design as N/W flow: 15 Explain constraints: 3 Answer to when possible: 2