CS570 Spring 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 residual graph of a maximum flow f can be strongly connected.
Note: A directed graph is strongly connected if there is a path from every node to every other node in the graph.

[TRUE/FALSE]
The Ford-Fulkerson algorithm can be used to **efficiently** compute a maximum matching of a given bipartite graph.

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

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

[TRUE/FALSE]
Any problem that can be solved with a greedy algorithm can also be solved with dynamic programming.

[TRUE/FALSE]
In a flow network G, if we increase the capacity of one edge by a positive amount x and we observe that the value of max flow also increases by x, then that edge must belong to **every** min-cut in G.

[TRUE/FALSE]
If we have a dynamic programming algorithm with n^2 subproblems, it is possible that the running time could be asymptotically strictly greater than $\Theta(n^2)$.

[TRUE/FALSE]
If we modify the Ford-Fulkerson algorithm by the following heuristic, then the time complexity of the resulting algorithm will be strongly polynomial:
At each step, choose the augmenting path with fewest edges.

[**TRUE/FALSE**]
The dynamic programming solution (presented in class) to the 0/1 knapsack problem has a polynomial run time.

[TRUE/FALSE]
Bellman-Ford algorithm runs in strongly polynomial time.

2) 20 pts.
Given a list of n integers $v_1, \ldots, v_n$ the "product-sum" of the list is the largest sum that can be formed by multiplying adjacent elements in the list, with the limitation that each element can only be multiplied with at most one of its neighbors. For example, given the list 1,2,3,1 the product sum is $8 = 1 + (2 \times 3) + 1$, and given the list 2,2,1,3,2,1,2,2,1,2 the product sum is
$19 = (2 \times 2) + 1 + (3 \times 2) + 1 + (2 \times 2) + 1 + 2$.

a) Verify that the product-sum of 1, 4, 3, 2, 3, 4, 2 is 29. (2pts)

$29 = 1 + (4 \times 3) + 2 + (3 \times 4) + 2$.

Give a dynamic programming algorithm for computing the value of the product-sum of a list of integers.

b) Define (in plain English) subproblems to be solved. (5 pts)

Let *OPT[j]* be the product-sum for elements 1..j in the list.

Rubric:
1. Mention OPT[j] is the product-sum: 3pts
2. The range should be "from v_1 to v_j": 2pts

Note: Some students incorrectly define the sub-problem as "OPT[i, j] is product-sum of v_i to v_j": 2pts deducted

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

$$OPT[j] = \begin{cases} \max\{OPT[j-1] + v_j, OPT[j-2] + v_j * v_{j-1}\} & if\ j \geq 2 \\ v_1 & if\ j = 1 \\ 0 & if\ j = 0 \end{cases}$$

Rubric:
1. Give a correct form of the recurrence, say, "OPT[j] = max (…, …)". In the max function there are two entries, "OPT[j-1] + v_j" and "OPT[j-2] + v_j*v_{j-1}": each for 3pts
2. Boundary case arranged correct as "OPT[0] = 0, OPT[1] = v_1 or OPT[1] = v_1, OPT[2] = max($v_1 + v_2$, v_1 * v_2)": 1pt

d) Write pseudo-code to compute the value of the product-sum (You do not need to determine which pairs are multiplied together – we just need the value of the optimal solution.) (4 pts)

```
Prod-Sum(int[ ]v, n)
if (n == 0)
return 0;
int [ ] OPT = new int [n + 1];
OPT[0] = 0; OPT[1] = v[1];
for int j = 2 to n
        OPT[j] = max(OPT[j − 1] + v[j], OPT[j − 2] + v[j] ∗ v[j − 1]);
endfor
return OPT[n];
```

Rubric:

1. If your recurrence in Part(b) is incorrect and the pseudo-code is formed on top of that: 2pts
2. If your Part(b) is correct but the pseudo-code is wrong, e.g., additional for-loop: 3pt
3. Boundary case is correct: 1pt

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

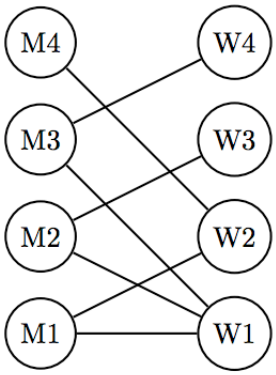
O(n)

Rubric:

1. If you leave Part(c) blank and simply write an answer here: no point
2. If your Part(c) is incorrect and you derive the complexity based on that: 1pt
3. If your Part(c) is correct but the complexity is wrong: 1pt

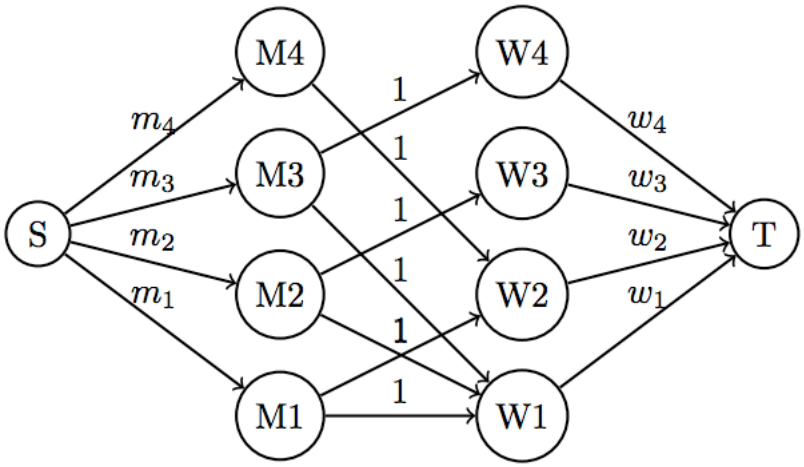
3) 20 pts

A dating website has used complicated algorithms to determine the compatibility between the profiles of men and women on their website. The following graph shows the set of compatible pairs. The website is trying to setup meetings between the men and the women. The i^{th} man has indicated a preference of meeting exactly m_i women, while the j^{th} woman prefers to meet at most w_j men. All meetings must be between compatible pairs. How would you use network flow to set up the meetings?



a) Construct a network flow solution, i.e. draw the network with which you can solve this problem, and explain your construction. (10 pts)

Solution:



We design a flow network, shown above, such that each unit of flow indicates a meeting between a man and a women. Observe that:

- Edges (S, Mi) have capacity mi , enabling man i to be matched with up to mi women.

- Since each man-woman pair should only be matched at most once, edges (Mi , Wj) have unit capacity.
- Edges (Wj , T) have capacity wj , preventing woman j from being matched with more than wj men.

Rubrics:

Add source and sink linking to all men and all women: 2pt
Edge capacity (S,Mi) should have m_i capacity 2 pts
Edge (Mi, Wj) should have unit capacity, directed from Mi to Wj: 4 pts
Edge (Wj,T) should have w_j capacity 2 pts

Or as with the lower bounds:
Add source and sink linking to all men and all women: 2pt
Edge capacity (S,Mi) should be m_i with m_i lower bound: 2pts
Edge capacity (Mi, Wj) should be 1 with 0 lowerbound:4 pts
Edge capacity (Wj,T) should be w_j with 0 lowerbound:2 pts

Small mistakes -1pt (can add up)

b) Explain how the solution to the network flow problem you constructed in part a will correspond to the solution to our dating problem, and prove the correctness of your solution. (10 pts)

We find max flow f. If the value of max flow v(f) equals Σ_i mi then we have a feasible solution. Unit flow on edge (Mi , Wj) indicates that meeting between Mi and Wj takes place in the schedule.

Proof of solution:

Prove that G has a max flow of Σ_i mi iff there is a feasible schedule of meetings for all men and women.

A – If we have a max flow of Σ_i mi in the network we will have a schedule of meetings between men and women that meets all constraints:

- Man i is scheduled to meet with exactly mi women since all edges out of S must be saturated, so there is a flow of exactly mi going into the node corresponding to man i and therefore there are mi edges carrying unit flows from i to mi women, i.e. mi meetings scheduled

- Woman j is scheduled to meet with at most wi men since flow over edge connecting woman j to sink t has a capacity of wj, so there cannot be more than wj unit flows coming into j, i.e. j can have up to wj meetings scheduled.

B - If we have a schedule of meetings between men and women that meets all constraints, we will have a max flow of Σ_i mi in the network:

- Send a flow of exactly mi from S to man i

- Send a unit flow from man i to woman j if there is a meeting scheduled between them

- Send a flow equal to the number of meetings scheduled for woman j, from j to T

  This will be a max flow since all edges out of S are saturated, and it will be a valid flow since conservation of slow is satisfied at all nodes.

Solution 2: Circulation

Build a network similar to that in solution 1, but instead place a demand value of Σ_i mi at T and - Σ_i mi at S

Proof will follow the same argument as in solution 1

Solution 3: Circulation with lower bounds

Build a network similar to that in solution 2, but also add lower bounds of mi to those edges between S and man i. These lower bounds are basically redundant in this particular problem and the solution works with or without them.

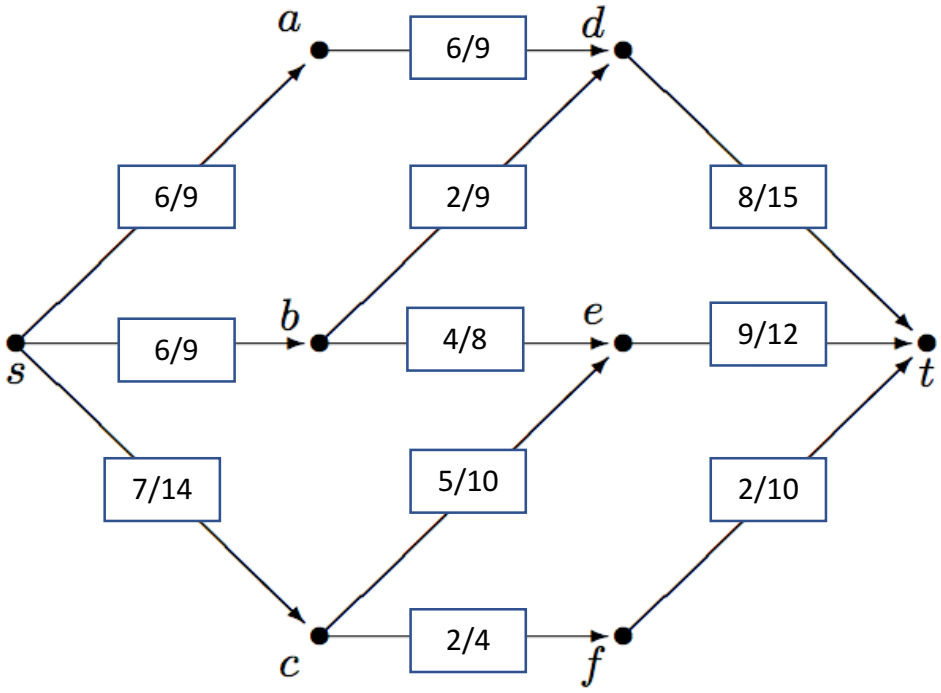
Proof will follow the same argument as in solution 1

Rubric:
Mention the matching is possible when we have a max flow 3 pts
(Will not be credited if graph in part a is not correct)

The value of max flow should be explicitly written as exactly F = sigma m_i to satisfy the condition: 3pts
(The value of max flow can be omitted only when the graph was constructed with lower bounds, and claimed a solution with valid flow/circulation)
Saying max flow F = sigma m_i = sigma w_j -2pts

Correct proof of correctness: 4pts
(In order to prove the correspondence, if and only if condition between max-flow ↔ dating solution should be proved. This was handled in the lecture.)
Case of weak proof: -2pts for one direction(e.g. stating only “if” statement) proof
Not enough explanation about proof: -2pts (e.g. writing statement only)

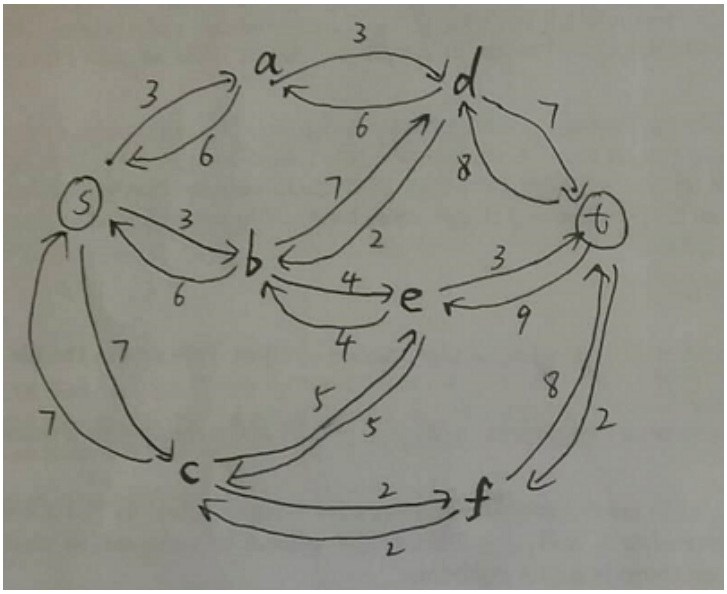
4) 20 pts.
Consider the following flow graph G, with an assigned flow f. The pair x/y indicates that the edge is carrying a flow x and has capacity y.



a) Draw the residual Graph G_f for the flow f. (10 pts)

Solution:

We apply the standard construction for the residual graph. If there is a flow of f and capacity c on the edge (u, v), we create an edge of capacity c − f from u to v, and an edge of capacity f from v to u.
As shown in the graph below.



Rubric:

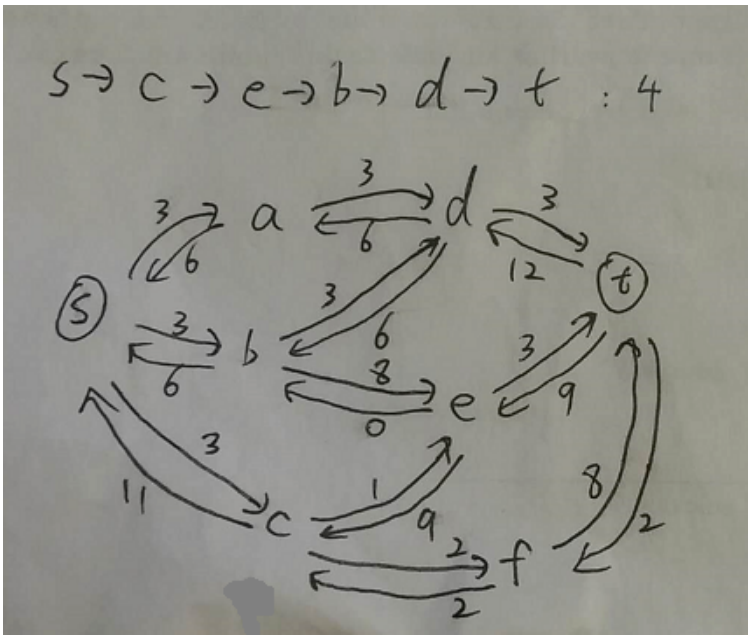
Wrong directions of the edges (-2 ~ -4)
Only draw one direction of the connected edges (-4 ~ -5)
Miscalculated the value of edges (-1 ~ -5, depends on how many mistakes)

b) Perform one additional step of the Ford-Fulkerson algorithm by choosing an augmenting path with the maximum bottleneck value. Show the residual graph at the end of this step of augmentation. (10 pts)

Solution:

Choose path S→c→e→b→d→t with bottleneck value of 4. Any other augmenting path cannot reach this value.

As shown in the graph below.



Rubric:

Correct residual graph, but doesn't tell us which is the path (-0.5 ~ -1)
Miscalculate the value of edges or the directions (-1 ~ -3, depends on how many mistakes. Notice the direction of the edge between *b* and *e*)

Wrote a wrong augmentation path (value is not 4), and drew the residual graph (-5 ~ -6)
Only drew the wrong residual graph (-6 ~ -8)

5) 20 pts

We want to solve the problem of determining the set of states with the smallest total population that can provide the votes to win the election (electoral college). Formally, the problem can be described as:

Let p_i be the population of state i, and v_i the number of electoral votes for state i. If a candidate wins state i, he/she receives v_i electoral votes, and the winning candidate is the one who receives at least V electoral votes, where $V = \lfloor (\sum_i v_i)/2 \rfloor + 1$. Our goal is to find a set of states S that minimizes the value of $\sum_{i \in S} p_i$ subject to the constraint that $\sum_{i \in S} v_i \geq V$.

Give a dynamic programming algorithm for finding the set S.

Hint: One way to solve this problem is to turn it into a 0/1 knapsack problem. If you choose this route, you still need to fill out all sections **a** through **e** as they relate to solving this particular problem. In other words, you still need to define subproblems, your recurrence formula, etc. in the context of this problem.

a) Define (in plain English) subproblems to be solved. (4 pts)
Missing some information (-1 pt, each)

Solution 1: use 0/1 knapsack to solve the problem

OPT(i,w) = **max** population of states 1..i such that their total vi **does not exceed** w.

We are basically trying to find those states with the highest total population whose total vi does not exceed W = Σ v_i – V. This set is the complement of the set that the problem statement is asking for.

b) Write the recurrence relation for subproblems. (7 pts)
If one part of the recurrence is wrong (-2 pt, each)
We accept answer if only mention Max(OPT(i-1,w-vi) + pi, OPT(i-1,w)).

Same as in knapsack:

OPT(i,w) = OPT(i-1, w) if w < vi ,
Otherwise Max(OPT(i-1,w-vi) + pi, OPT(i-1,w))

c) Write pseudo-code to compute the optimal value of all unique subproblems. (4 pts)
If the base cases / initialization are wrong (-2 pt)
If you do not consider the case if W < vi (-2 pt)
If one part of the pseudo-code is wrong (-2 pt, each)

W = Σ v_i – V
Initialize OPT(0,w) = 0 for w=0 to W

```
For i=1 to n
    For w=0 to W
        If W< vi
            OPT(i,w)    = OPT(i-1, w)
        Otherwise
            OPT(i,w)    = Max( OPT(i-1,w-vi) + pi, OPT(i-1,w))
    Endfor
Endfor
Return OPT(n,W)
```

d) Write pseudo-code to find the set of states S. (3 pts)
If one part of the pseudo-code is wrong (-2 pt, each)

First find those states with the highest total population whose total vi does not exceed
Initialize set S to NULL
$W = \Sigma v_i - V$

```
For i=n to 1 by -1
    If ( OPT(i-1,W-vi) + pi > OPT(i-1,W) ) then
//or
    If ( OPT(i-1,W-vi) + pi = OPT(i,W) ) then

        Add i to the set of states S
        W = W-vi
    Endif
Endfor
```

Our Solution S' = complement of the set S

e) Is the solution you have provided an efficient solution? Give a brief explanation. (2 pt)
The answer is correct (1 pt). The explanation is correct (1 pt)

No the solution runs in O(nW) which is pseudo-polynomial with respect to the input size, since W represents the numerical value of the input and not the size of the input.

Solution 2: Solve directly

a) Define (in plain English) subproblems to be solved. (4 pts)
Missing some information (-1 pt, each)

OP T[i, v] = minimum populations of a set of states from 1, 2, . . . , i such that their votes sum to **exactly** v electoral votes.

b) Write the recurrence relation for subproblems. (7 pts)
If one part of the recurrence is wrong (-2 pt, each)
We accept answer if only mention $\min\{OP\ T[i-1, v], OP\ T[i-1, v-v_i] + p_i\}$

$$OP\ T[i, v] = \begin{cases} OP\ T[i-1, v] & \text{if } v < v_i, \\ \min\{OP\ T[i-1, v], OP\ T[i-1, v-v_i] + p_i\} & \text{Otherwise} \end{cases}$$

c) Write pseudo-code to compute the optimal value of all unique subproblems. (4 pts)
If the base cases / initialization are wrong (-2 pt)
If you do not consider the case if v < vi (-2 pt)
If one part of the pseudo-code is wrong (-2 pt, each)

$W = \Sigma\ v_i$
The important thing here is the initialization:
$OP\ T[0, 0] = 0$
$OP\ T[0, v] = \infty$ for $v \geq 1$

```
For i=1 to n
        For v=0 to W
                If v< vi
                        OPT[i, v] = OPT[i − 1, v]
                Otherwise
                        OPT[i, v] = min{OPT[i − 1, v], OP T[i − 1, v − vi ] + pi}
        Endfor
Endfor
```

Return smallest OPT[i, v] where v >=V and i=n.

d) Write pseudo-code to find the set of states S. (3 pts)
If one part of the pseudo-code is wrong (-2 pt, each)

Initialize set S to NULL
Let W = v, where OPT[n, v] was the optimal value of the solution found in part c

For i=n to 1 by -1

```
        If ( OPT(i-1,W-vi) + pi < OPT(i-1,W) ) then
//or
        If ( OPT(i-1, W-vi) + pi = OPT(i,W) ) then
                Add i to the set of states S
                W = W-vi
        Endif
Endfor
```

Our solution is the set S

e) Is the solution you have provided an efficient solution? Give a brief explanation. (2 pt)
The answer is correct (1 pt). The explanation is correct (1 pt)

No, similar to solution 1.

No the solution runs in O(nW) which is pseudo-polynomial with respect to the input size, since W represents the numerical value of the input and not the size of the input.