CS570 Spring 2019: Analysis of Algorithms Exam II

	Points		Points
Problem 1	20	Problem 5	20
Problem 2	6	Problem 6	14
Problem 3	20		
Problem 4	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.

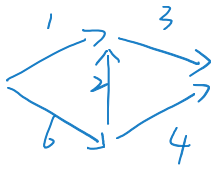
1) 20 pts

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

[TRUE/FALSE]

as long as nonnegative and integeral capacity

The Ford-Fulkerson algorithm can be used to find the maximum flow through a graph with cycles.



[TRUE/FALSE]

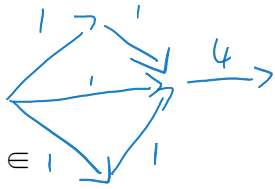
In a flow network where all edge capacities are unique, the min cut will be unique.

[TRUE/FALSE]

A Dynamic Programming algorithm with n^3 unique sub-problems will run in $O(n^4)$ time.

[TRUE/FALSE]

It is possible for a dynamic programming algorithm to have exponential running time.



[TRUE/FALSE]

Let G be a weighted directed graph with exactly one source s and exactly one sink t. Let (A,B) be a maximum cut in G, that is, A and B are disjoint sets whose union is V, $s \in A$, $t \in B$, and the sum of the weights of all edges from A to B is the maximum for any two such sets. Now let H be the weighted directed graph obtained by adding 1 to the weight of each edge in G. Then (A,B) must still be a maximum cut in H.

[TRUE/FALSE]

The recurrence $OPT(j) = \max_{1 \le i \le j} \{A[i] - B[j-i] + OPT(j-i)\}$ where A and B are fixed Input arrays, will lead to an $O(n)$ dynamic programming algorithm. n^2

[TRUE/FALSE]

In a graph with more than one minimum cut, if we take any edge e that is part of the (A,B) minimum cut and increase the capacity of that edge by any integer $k \ge 1$, then (A,B) will no longer be a minimum cut.

[TRUE/FALSE]

The Ford-Fulkerson algorithm (without scaling) can be used to efficiently solve the maximum matching problem in a bipartite graph.

[TRUE/FALSE]

The space efficient version of the sequence alignment algorithm (which uses a combination of divide and conquer and dynamic programming) has the same running time complexity as the basic dynamic programming algorithm for this problem.

[TRUE/FALSE]

Suppose, increasing the capacity of edge e by 1, increases value of max flow f by 1. Then it must be that $f(e)$ = capacity (e).

2) 6 pts
Let G = (V, E) be a flow network with source *s*, sink *t*, and integer capacities. Suppose that we are given a maximum flow *f* in G. Now, we increase the capacity of a single edge *e* by 1.

a) If *f (e)* < Capacity (*e*), can we say that *f* will still be a maximum flow in G? (2pts)

Yes.

Remarks: this part is not graded since there was confusion about whether the capacity was before the flow or after. So "No" is also accepted as correct answer.

b) If *f (e)* = Capacity (*e*), can we say that *f* will no longer be a maximum flow in G? (2 pts)

No.

c) Give an O(V + E)-time algorithm to find a new maximum flow in G. (2 pts)

Find new Gf and see if there is a path from s to t. If there is, do one more step of augmentation.

Rubric: All-or-nothing. No partial credit.

3) 20 pts

We are given a string S = $s_1s_2s_3\ldots s_n$, and we want to insert some characters such that the resulting string is a palindrome. Give an efficient dynamic programming algorithm to determine the minimum number of insertions.

Definition: A palindrome is a string that reads the same forward and backward such as ACCBCCA.

Example: Input: ACB Output: 2 (ACBCA)

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

opt(i,j) = minimum number of insertions to make substring s[si...sj] a palindrome.

grading rubric:
3pts if using opt(i) to represent the number of insertions to make substring s[0…si] a palindrome

reverse the string is ok and is graded in a similar scheme.
0pts everything else.

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

if si== sj: opt(i,j) = opt(i+1, j-1)
else: opt(i,j) = min(opt(i+1,j), opt(i, j-1)) + 1

grading rubric:
si==sj: 2pts
opt(i+1, j): 2pts
opt(i, j-1): 2pts
+1: 7pts

reverse the string and compare are ok and is graded in a similar scheme. use sigma (or other constants) to represent mismatch penalty is ok.

c) Using the recurrence formula in part b, write pseudocode to compute the minimum number of insertions. (5 pts)
Make sure you have initial values properly assigned. (3 pts)

```
for i=1 to n:
        for j=1 to n:
                opt(i,j) = 0
```

```
for j=1 to n:
        for i = j-1 to 1:
                if si == sj:
                        if i < j-1:
                                opt(i,j) =opt(i+1, j-1)
                        else:
                                opt(i,j) = 0
                else:
                        opt(i,j) = min(opt(i+1,j), opt(i,j-1))+1
return opt(1,n)
```

grading rubrics:
initialization: 3pts (need to initialize to 0).
recursion: if si == sj: opt(i,j) =opt(i+1, j-1) 2pts; opt(i,j) = min(opt(i+1,j), opt(i,j-1))+1 3pts.

4) 20 pts

The Laver cup is a tennis tournament that takes place every year between Team Europe and Team US, each consisting of k players. Each player participating has a world ranking. A match between players of ranking r_i and r_j is fair if $| r_i - r_j | <= d$. The tournament must consist of n matches, each between a player from Team EU and a player from Team US. Describe an efficient algorithm that, given the rankings of all the players, determines if it is possible to schedule n fair matches between the two teams such that each player plays at least one match but no more than m matches, and no two players play each other more than once.

A4.

a) bipartite graph with k players of team US on left and k players of team EU on right.
b) connect with an edge each US player (us_i) to the corresponding EU player (eu_j) whose skill level is within d. Each such edge has weight one to ensure at most one match between any two players in opposing teams.
c) two nodes source and sink. Where edges (s, us_i) have a lower bound of 1 and capacity m. Likewise edges (eu_j, t) have lower bound 1 and capacity m.
d) Add a demand of n at t and supply of n at s.

To solve: Convert to the problem of circulation with demand.

Claim: a matching exists if there exists a feasible circulation in the network.

Grading Rubric:

1. S->US->EU->T (or S->EU->US->T) structure

2. Demand of -n for S and n for T
3. Connecting players with the restriction of fair matching
4. For S->US and EU->T edges, capacity=m and lower bound=1; for US->EU edges, capacity=1 and no lower bound
5. Final claim indicating a maxflow (available flow) of size n. Note that maxflow>=n is incorrect, because when maxflow>n, a flow of n is not necessarily possible in the network (e.g. when k>n, we get maxflow>=k>n, but a flow of size n is impossible).

4 points each for each correct use of above. No partial credit!

Algorithms other than network flow are considered incorrect/inefficient, with 0 point given.

20 pts

5) Your company, Stuckbars Coffee and Toffee is planning to open several stores on Main Street. Main Street has n blocks 1, 2,…, *n* from East to West, and for each block *i* you know the revenue $r_i > 0$ you expect from a store on this block. But if you open a store on block *i*, you cannot open another store in block *i*, or the two blocks to its East or the two blocks to its West. You have designed a dynamic programming algorithm for finding the store locations that maximizes total revenue, but you spilled coffee on it and now you have to reconstruct it. Here are the steps you need to follow.

All solution more complex than the linear solution are considered incorrect. The incorrect part will lead to score reduction in every section. If the solution uses other methods, a student must show it has a complexity of O(n). Otherwise, it is very likely to get a lower score if request to be regraded.

d) Define (in plain English) subproblems to be solved. (4 pts)

Choice of subproblem: R [i] = the maximum total revenue if stores can only be opened on blocks 1, …. ,i

Rubric:

If state opt(i) means max revenue for store 1 to i, or i to n. (full credits)
If state placement on a store on ith block but not mention 1 to i or i to n. (-1 or -2)
Other complicated sub problem or ambiguous statement (-2 or -3)
If not a subproblem: -4

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

Recurrence relation: R[i] = $max\{R[i-1], r_i + R[i-3]\}$

Rubric:

If max(ri+ opt(i-3), opt(i-1)) (full credits)
Write max(opt(i-1),opt(i-2),opt(i-3)+ri) is ok, since it doesn't increase complexity
If writes i-3 as i-2: (-1 or -2)
Other incorrect answer (-3 to -5)
Empty (-6)

f) Using the recurrence formula in part b, write pseudocode to compute the maximum revenue possible. (4 pts)
Make sure you have initial values properly assigned. (2 pts)

R[1] = r_1 R[2] = max(r_1,r_2) R[3] = max(r_1, r_2, r_3)
For i = 4, …. , n;
R[i] = $max\{R[i-1], r_i + R[i-3]\}$

Initializing n, n-1 and n-2, or R[0], R[1], R[2], or R[<0],R[1] are all correct
If index is incorrect (-1 or -2)
If other incorrect methods which have higher complexity (-3 or -4)
Initialization as R[3] = r3 R[2]=r2 (-1 or -2)

g) Using the results in part c, write pseudocode to determine the store locations. (2 pts)

Recover solution: $prev[i] = \begin{cases} i-3, & if\ R[i] = r_i\ +\ R[i-3] \\ i-1, & otherwise \end{cases}$

Rubric:
Only mention Trace back (-2 or -1)
Wrong in bound (-1)
Not using result in c or empty (-2)
Include i-1 or i-3 in the location(-1 or -2) consider a case: r1=1, r2=1, r3=1000, r4=1,r5=1,r6=1,r7=1,r8=1000,r9=1,r10=1
we should only select r3 and r8.

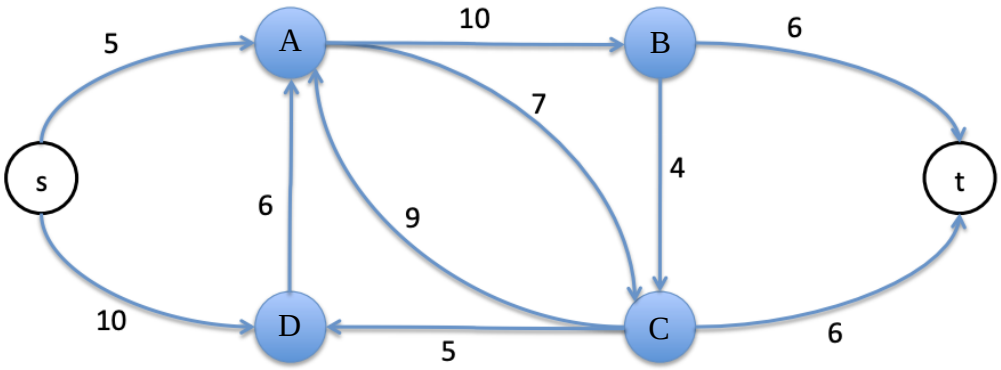
h) Compute the runtime of the algorithm described in part c and state whether your solution runs in polynomial time or not (2 pts)

Running time: O(n), since each subproblem takes constant time to compute based on smaller subproblems, and there are n subproblems to compute.

Rubric:
If not o(n) (-1)
If not strong polynomial (-1)

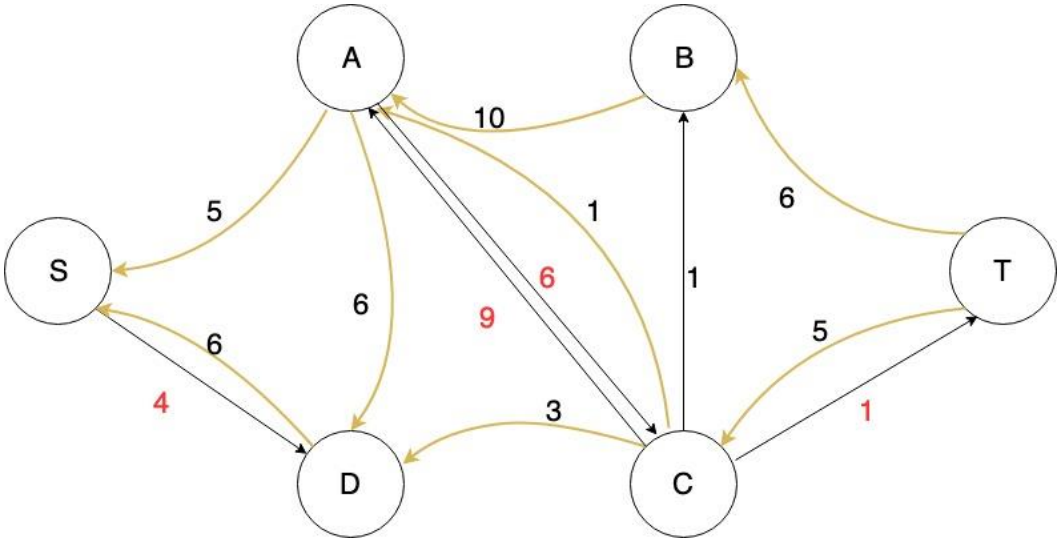
10 pts

6) For the following flow network, with edge capacities as shown,

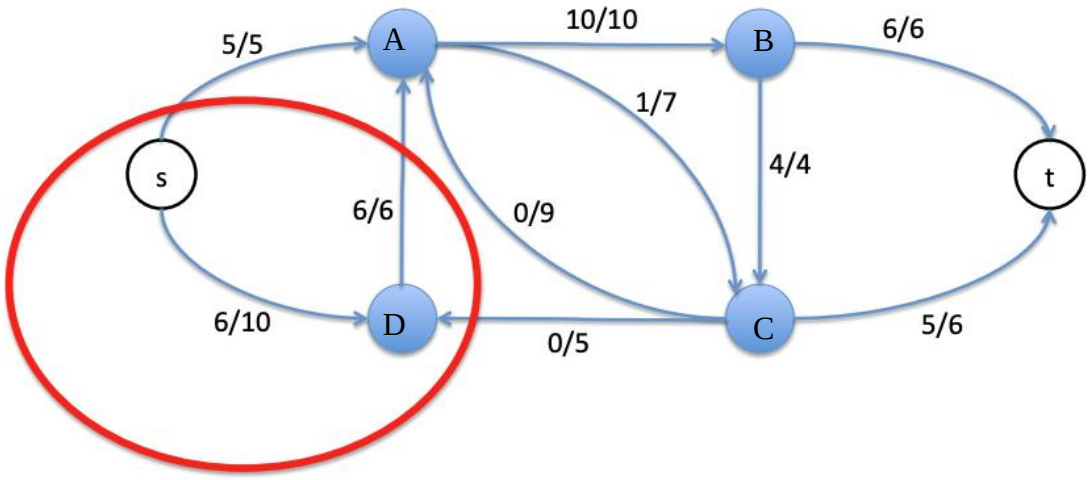


a) Find a minimum s-t cut in the network and indicate what its capacity is. (Clearly show which nodes are in which set)

Solution



This is the final residual graph, yellow ones are back edge, red values are residual capacity.



The cut has cost 11, and we can show that it is the minimum cut by demonstrating a flow with value 11—see above for the flow assignment.
Two sets: set1{D},set2{A, C, B}
Since the maximum-flow is equal to the minimum cut, this proves that 11 is the minimum cut.

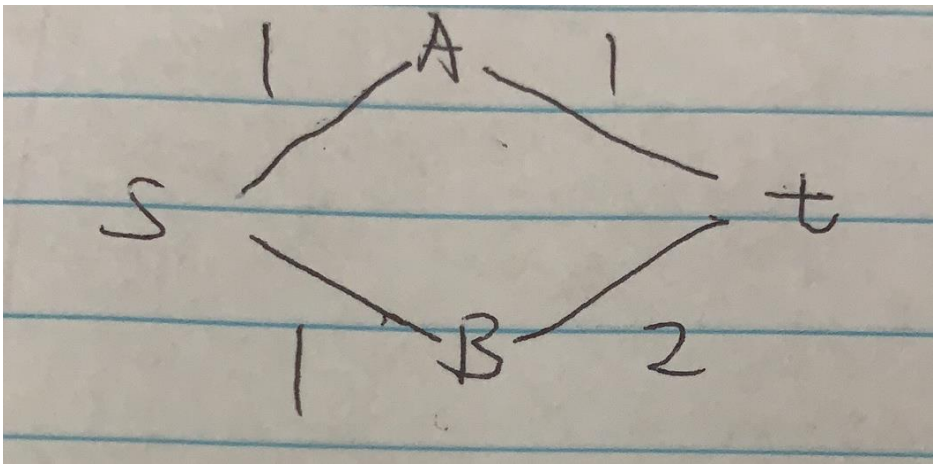
b) Describe how you would determine if this is the only minimum cut in G.

Solution1. BFS from source to output the min-cut. Then BFS from sink to output another min-cut. If same then this is the only mincut, else multiple min-cuts exist.

Solution2. For every edge e in (set1, set2), increase the capacity of that edge and compute the value of the max-flow again. If the value does not increase, then it means that there is another min-cut that there was another min-cut in the original graph that did not include this edge, and hence the min-cut was not unique. If for all the edges in (set1, set2) the value of the max-flow increases, then the min-cut is unique.

Common mistakes:
1. Increase value on only one of the edges in min-cut. Or increase all of them at the same time instead of one by one.

Counterexample:



2. Try to find augment path in other orders to find different min-cut.
   Counterexample: s-1-A-2-B-1-t

Rubric:
a.
Steps for 5 points.
Min-cut for 3 points.
Capacity for 2 points.

b.
Partial score is given only on common mistake 1.