CS570 Fall 2022: Analysis of Algorithms Exam II

	Points		Points
Problem 1	16	Problem 4	18
Problem 2	16	Problem 5	18
Problem 3	16	Problem 6	18
	Total	**100**	

Instructions:

1. This is a 2-hr exam. Open book and notes. No electronic devices or internet access.
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.
8. This exam is printed double sided. Check and use the back of each page.

1) 16 pts

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

[TRUE/FALSE]

In dynamic programming, solving for the value of the optimal solution to subproblems is performed in the bottom up pass.

[TRUE/FALSE]

Given a maximum flow f in a flow network G with m edges, we can determine if G has a unique min-cut in $O(m)$ time.

[TRUE/FALSE]

For a flow network G, if a flow network G' is constructed from G by increasing the edge capacity of each edge in G by 1, then the value of a maximum flow in G' is at most g units more than the value of a maximum flow in G, where g is the number of edges leaving the source in G.

[TRUE/FALSE]

The running time of a pseudo-polynomial time algorithm can be upper bounded by a polynomial function of the size of the problem input (or output).

[TRUE/FALSE]

Every iteration-based dynamic programming algorithm with n^2 unique subproblems has running time $\Omega(n^2)$.

[TRUE/FALSE]

Ford-Fulkerson can be used to find the maximum size matching between two strings in polynomial time.

[TRUE/FALSE]

The problem of checking whether a given flow f of a flow network G is a maximum flow can be solved in linear time.

[TRUE/FALSE]

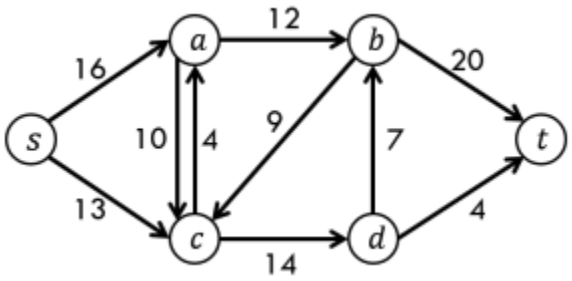
Let G be a flow network with source s and sink t. Let (A, B) be a maximum cut in G. Let G' be a flow network obtained by adding 1 to the capacity of each edge in G. Then (A, B) must also be a maximum cut in G'.

2) 16 pts (a, b, d, bc)

I. Consider a bipartite graph G with m edges and $2n$ nodes whose node set is partitioned into two sets X and Y with the property that every edge in G has one end in X and the other in Y.
To find the maximum size matching in G, Ford-Fulkerson is guaranteed to terminate after at most how many iterations? Select the smallest correct upper bound. (4 pts)

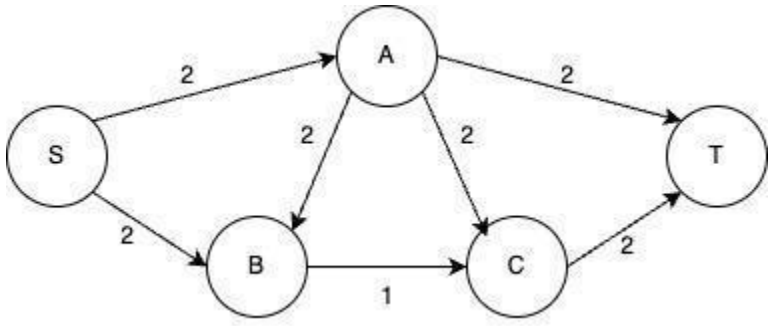
a) n
b) $2n$
c) $2n^2$
d) mn

II. Which of the following s-t cuts are min-cuts in the flow network below? Circle all correct answers. (4pts)



a) $A = \{s, c, d\}, B = \{a, b, t\}$
b) $A = \{s, a, c, d\}, B = \{b, t\}$
c) $A = \{s, a, c\}, B = \{b, d, t\}$
d) None of the above

III. Consider the flow network G below with source S and sink T. Suppose we are using the Ford-Fulkerson algorithm to find a maximum flow in G. Suppose that the first augmenting path used by the Ford-Fulkerson algorithm is S-A-B-C-T. How many distinct possible second augmenting paths pass through the vertex A? (4 pts)



a) 1
b) 2
c) 3
d) 4

Explanation: The paths are S-A-T, S-A-C-T, S-B-A-T and S-B-A-C-T

IV. Suppose a certain problem is described by the following inputs:

- A string of length m.
- An array of integers of size n, where the maximum integer in the array is C.

Which of the statements below are true? Circle all true statements. (4 pts)

a) An algorithm for solving the problem with running time $\Theta(nm)$ is called a linear time algorithm.
b) If for all input instances $C = 10$, then an algorithm with running time $\Theta(nC)$ is strongly polynomial.
c) If for all input instances the average of the integers is 1, then an algorithm with running time $\Theta(n*Sum(Array))$ is strongly polynomial.
d) An algorithm with running time $\Theta(n\log m)$ is called a pseudo-polynomial time algorithm.

Explanation: 'a' Is strongly polynomial. 'b' reduces to $\Theta(n)$ as C is trivial. 'c' is reduced to $\Theta(n^2)$ as sum of array becomes n when average is 1. 'd' is strongly polynomial.

3) 16 pts

You have found a hidden treasure that contains n diamonds placed in a row. You know each of these diamonds' values $[v_0, v_1, \ldots, v_{n-1}]$. Ideally, you would have taken all the diamonds, but this treasure is cursed, and the cave will collapse if you pick up 3 consecutive diamonds. Knowing this, find the maximum total value of the diamonds you can pick up such that no three consecutive diamonds are picked.

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

dp[i] = Max possible total value from v_0 to v_i, such that no three elements are consecutive.

b) Write a recurrence relation for the subproblems (4 pts)

dp[i] = Max of the following 3 cases:

- dp[i] = dp[i-1] (Exclude v_i)
- dp[i] = dp[i-2] + v_i (Exclude v_{i-1})
- dp[i] = dp[i-3] + v_i + v_{i-1} (Exclude v_{i-2})

Rubric: one point for each of the 3 cases, one for taking max of the 3

c) (6 pts for part c)
Using the recurrence formula in part b, write pseudocode using iteration to find the maximum total value of diamonds you can obtain without having the cave collapse. (4 pts)
Make sure you specify base cases and their values (2 pts)

Base case:

- $dp[0] = v_0$
- $dp[1] = v_0 + v_1$
- $dp[2] = \max(dp[1], v_1 + v_2, v_0 + v_2)$
  - Alternatively, $dp[-1] = 0$

Pseudocode:

```
*Compute Base cases above*
For i=3 to n-1, do:
        dp[i] = max of the 3 cases          \\ refer to the recurrence relation

return dp[n-1]
```

Rubric:

- 2 points for the base cases
- Pseudocode: 1 point for the order/structure of the computations, 2 points for the loop, 1 point for the final return

d) What is the time complexity of your solution? Is your algorithm efficient? (2 pts)

O(n). Thus, it is efficient.

4) 18 pts

Let $G = (V, E)$ be a directed graph, and let s and t be two distinct non-adjacent vertices in G.

(a) Design a polynomial-time algorithm for finding a set of nodes C of minimum size such that $s \notin C$ and $t \notin C$ and deleting C from G disconnects s from t so that there is no longer any directed path from s to t. (12 pts)

Given G we create a new graph G', where each vertex v of G, G' has two copies v_{in} and v_{out}, connected with an edge $v_{in} \rightarrow v_{out}$ that has capacity 1. Additionally, for each edge e = (u,v) ∈ E(G), we create a new edge in G' with infinite capacity that connects $u_{out} \rightarrow v_{in}$. Let s_{out} be the source node and t_{in} be the sink node of G'.

Run Ford-Fulkerson (or other algorithms) for finding a max-flow on G', and use the final residual graph to find a min-cut of G'. This min-cut value is the size of such smallest C, and we return C as the set of nodes u for which $u_{in} \rightarrow u_{out}$ is in this min-cut.

Rubrics:

- Construction of G' (8 points)
  - 2 nodes v_{in} and v_{out} for each v (1 pts) and a directed edge betn them (1 pt)
  - Capacity 1 for the edges above (2 pts)
  - Edge $u_{out} \rightarrow v_{in}$ for each (u,v) ∈ E(G) (1 pt) with infinite capacity (2 pts). Any capacity higher than |V| works.
  - s_{out} as the source node and t_{in} as the sink (1 pt)
- Find min-cut (2 points). Describe how, i.e. something like "run FF and use the residual graph" (1 point). Getting the final answer C from the min-cut (1 point)

(b) Prove the correctness of your algorithm. (6 pts)

Let Cut be a minimum s_{out}-t_{in} cut in G'. Note that the cut that simply removes all edges of type $u_{in} \rightarrow u_{out}$ would disconnect s_{out} and t_{in} successfully with cost |V| (since s_{out} and t_{in} are not directly connected). Therefore, Cut must have a cost at most |V|, and thus cannot contain the edges of infinite capacity (or edges of cost higher than |V|), i.e. it contains only edges of type $u_{in} \rightarrow u_{out}$.

Let $C = \{u \in V \mid (u_{in}, u_{out}) \in Cut$. Thus, |C| = capacity of Cut. Removing C from the original graph G will disconnect s from t, since removing the corresponding $u_{in} \rightarrow u_{out}$ edges disconnects s_{out} from t_{in}.

Also $s, t \notin C$, since $s_{in} \rightarrow s_{out}$ and $t_{in} \rightarrow t_{out}$ cannot be in the cut set with source s_{out} and sink t_{in}. Thus C is a valid set of nodes satisfying the requirement on G.

The opposite is also true: any set of nodes C' that disconnects s from t in G defines a s_{out}-t_{in} cut in G' of the same capacity as |C'|. Therefore the size of C defined above is the smallest possible (otherwise a smaller sized C' would define a smaller cut).

Rubric:

- 2 pts for showing the min-cut contains only the edges of type $u_{in} \rightarrow u_{out}$
- 1 pt for showing that the set C obtained from the min-cut disconnects s, t in G.
- 1 pt for showing C doesn't contain s or t.
- 2 pts for Showing C is indeed minimum

5) 16 pts

A researcher is deciding which programs to run on a supercomputer. The researcher is given a sequence of n programs. Each program i is associated with three positive integers: s_i, w_i, and t_i. If the researcher chooses to run program i, then the supercomputer will be occupied for t_i units of time, during which the supercomputer performs w_i units of work, and the researcher will be unable to run programs $i+1$ through $i+ s_i$. The total time the supercomputer is occupied cannot exceed a given positive integer T. Design a dynamic programming algorithm for selecting a valid subsequence of programs for the researcher to run such that the supercomputer performs a maximal amount of work.

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

opt(i, t) = maximum amount of work that can be accomplished in t unit of time using programs i to n.

b) Write a recurrence relation for the subproblems (4 pts)

If T_i > t then opt(i, t)= opt(i+1, t),
Otherwise, opt(i, t)= max(opt(i+1, t) , opt(i + S_i + 1, t - Ti) + W_i)

Rubric:

- 1 point for the case T_i > t.
- 3 pts for the other case: 1 pt for each of the terms, 1 pt for taking max of the two.

(6 pts for part c)
Using the recurrence formula in part b, write pseudocode using iteration to find the maximum amount of work that can be accomplished. (4 pts)
Make sure you specify base cases and their values (2 pts)

Boundary conditions (Base cases): 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
        for t= 1 to T
                Use the recurrence formula
        end for
end for
```

opt(1, T) will hold the value of the optimal solution (maximum work load that can be accomplished).

Rubric:

- 1 pt for each base case.
- Pseudocode: 1 point for the order/structure of the computations, 2 points for the two for loops, 1 point for the final return.

c) What is the time complexity of your solution? Is your algorithm efficient? (2pts)

Complexity is O(nT)

Overall complexity of the algorithm is O(nT) which is **pseudopolynomial**. Not efficient

NOTE : An alternate solution is acceptable which uses a *forward iteration* approach, i.e. the subproblem opt(i, t) depends on opt(j, t) with j < i in the recurrence. The overall complexity of there will be $O(n^2T)$ which is also **pseudopolynomial**. Not efficient

6) 18 pts

Let G be a flow network. An edge e in G is called *critical* if it crosses every min-cut in G, in other words it goes from set A into set B in every (A,B) min-cut. And it is called *pseudo-critical* if it crosses at least one min-cut in G.

a) Design an efficient algorithm for determining whether a given edge e is critical. (9 pts)

(a) Run Ford-Fulkerson algorithm to find the maximum flow between s and t. Denote the maximum flow by f1.

Increase the capacity of the edge.

Run Ford-Fulkerson algorithm again to find the maximum flow between s and t. Denote the maximum flow by f2.

The edge e belongs to all minimum cuts between s and t if and only if
f2 > f1.

Alternative Solution:

Run FF algorithm and get a residual graph Gf

Do a BFS/DFS to find all leaf nodes that are connected to S which have a saturated edge.

Reverse Gf, Do a BFS/DFS again to find all leaf nodes that are connected to T which have a saturated edge.

Check if the edge e(u,v) have u in one union and v in the other union.

If True, that means the

Rubric:

Breakdown for algorithms that are nearly correct (and efficient):

- 2 Points for using maxflow method
  - -1 if not Mention efficient maxflow algorithm used
- 4 Points for modifying weight of edge correctly
  - 2 Points for change weight of input edge
  - 2 Points for Increasing the capacity of input edge
- 3 Points for validation of new flow
  - 1 Points for running FF again
  - 2 Points for showing new flow Increased

OR

- 5 Points for Inefficient Algorithm that find the answer
  - -1 Points for every minor Error
  - -1 if not Mention efficient maxflow algorithm used
  - * Note that Finding all min-cut would cost O(n^2)

b) Design an efficient algorithm for determining whether a given edge *e* is pseudo-critical. (9 pts)

(b) Run Ford-Fulkerson algorithm to find the maximum flow between s and t. Denote the maximum flow by f1.

Decrease the capacity of the edge e by 1.

Run Ford-Fulkerson algorithm again to find the maximum flow between s and t. Denote the maximum flow by f2.

The edge e belongs to some minimum cuts between s and t if and only if f2 < f1.

Rubric:
Breakdown for algorithms that are nearly correct (and efficient):

- 2 Points for using maxflow method
  - -1 if not Mention efficient maxflow algorithm used
- 4 Points for modifying weight of edge correctly
  - 2 Points for change weight of input edge
  - 2 Points for Decrease the capacity of input edge
- 3 Points for validation of new flow
  - 1 Points for running FF again
  - 2 Points for showing new flow Decreased

OR

- 5 Points for Inefficient Algorithm that find the answer
  - -1 Points for every minor Error
  - -1 if not Mention efficient maxflow algorithm used

Additional Space

Additional Space