

**CS570**

**Analysis of Algorithms**

**Summer 2017**

**Exam III**

Name: \_\_\_\_\_

Student ID: \_\_\_\_\_

Email Address: \_\_\_\_\_

\_\_\_\_\_ **Check if DEN Student**

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	15	
Problem 4	15	
Problem 5	10	
Problem 6	15	
Problem 7	10	
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.

Question 3: Given a directed graph where every edge has weight as either 1 or 2, find the shortest path from a given source vertex 's' to a given destination vertex 't'. Expected time complexity is  $O(V+E)$ .

If we apply Dijkstra's shortest path algorithm, we can get a shortest path in  $O(E + V \log V)$  time. How to do it in  $O(V+E)$  time? The idea is to use BFS. One important observation about BFS is, the path used in BFS always has least number of edges between any two vertices. So if all edges are of same weight, we can use BFS to find the shortest path. For this problem, we can modify the graph and split all edges of weight 2 into two edges of weight 1 each. In the modified graph, we can use BFS to find the shortest path. How is this approach  $O(V+E)$ ? In worst case, all edges are of weight 2 and we need to do  $O(E)$  operations to split all edges, so the time complexity becomes  $O(E) + O(V+E)$  which is  $O(V+E)$ .

1) 20 pts

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

[ **TRUE**/FALSE ]

T

Every problem in P can be reduced to 3-SAT in polynomial time.

[ **TRUE**/FALSE ]

F


~~T~~

If there is a polynomial-time algorithm for 2-SAT, then every problem in NP has a polynomial-time algorithm. *2-SAT is in P*

[ **TRUE**/FALSE ]

T

~~F~~

If all edge weights are 1, 2, or 3, the shortest path problem can be solved in linear time. 

[ **TRUE**/FALSE ]

T

Suppose G is a graph with  $n$  vertices and  $n^{1.5}$  edges, represented in adjacency list representation. Then depth-first search in G runs in  $O(n^{1.5})$  time.

[ **TRUE**/FALSE ]

F

The weight of a minimum spanning tree in a positively weighted undirected graph is always less than the total weight of a minimum spanning path (Hamiltonian Path with lowest weight) of the graph.

[ **TRUE**/FALSE ]

F

If A is in NP, and B is NP-complete, and  $A \leq_p B$  then A is NP-complete.

[ **TRUE**/FALSE ]

F

~~T~~

Given a problem B, if there exists an NP-complete problem that can be reduced to B in polynomial time, then B is NP-complete. *NP-hard*

[ **TRUE**/FALSE ]

T

~~F~~

If an undirected connected graph has the property that between any two nodes  $u$  and  $v$ , there is exactly one path between  $u$  and  $v$ , then that graph is a tree. *There are two paths in a cycle.*

[ **TRUE**/FALSE ]

T

~~F~~

Suppose that a divide and conquer algorithm reduces an instance of size  $n$  to four instances of size  $n/5$  and spends  $\Theta(n)$  time in the divide and combine steps. The algorithm runs in  $\Theta(n)$  time.

$$T(n) = 4T(n/5) + \Theta(n)$$

[ **TRUE**/FALSE ]

F

~~T~~

An integer  $N$  is given in binary. An algorithm that runs in time  $O(\sqrt{N})$  to find the largest prime factor of  $N$  is considered to be a polynomial-time algorithm.

Ex: Prime factorization of 84:  $2 \times 2 \times 3 \times 7$ , so the largest prime factor of 84 is 7

*Running time is  $O(\sqrt{n})$ , which is exponential in the size of the binary representation of  $n$*

2) 15 pts

We are given a set of non-equality constraints of the form  $x_i \neq x_j$  over a set of Boolean variables  $x_1, x_2, \dots, x_n$ . We wish to determine if there is an assignment of Boolean values 0,1 to the variables that satisfies all the constraints, and compute such a satisfying assignment if there is one. Give an algorithm that solves this problem in time  $O(n+m)$ , where  $n$  is the number of variables and  $m$  the number of constraints. Justify the correctness and time complexity of your algorithm.

This can be reduced to the problem of testing whether a graph is bipartite.

Given a set  $C$  of non-equalities  $x_i \neq x_j$  on a set of Boolean variables  $x_1, x_2, \dots, x_n$  we can construct a graph  $G$  which has a node for each variable and has an edge  $(x_i, x_j)$  for each constraint  $x_i \neq x_j$ . We claim that the graph  $G$  is bipartite if and only if the set  $C$  of constraints has a satisfying assignment.

First, suppose that the graph  $G$  is bipartite, i.e., there is a partition of the set  $N$  of nodes into two sets subsets  $N_1, N_2$  ( $N_1 \cap N_2 = \emptyset, N_1 \cup N_2 = N$ ) so that every edge connects a node of  $N_1$  with a node of  $N_2$ . Then the assignment that gives value 1 to all variables corresponding to nodes in  $N_1$  and value 0 to all variables corresponding to nodes in  $N_2$ , satisfies all the constraints: Since every edge connects a node of  $N_1$  with a node of  $N_2$ , this assignment satisfies all the constraints.

Conversely, if there is an assignment that satisfies all the constraints, then we can form a bipartition of the node set by letting  $N_1$  be the set of all the nodes corresponding to variables that are assigned 1 and  $N_2$  the set of nodes corresponding to variables that are assigned 0. Since the assignment satisfies all the constraints, in every constraint  $x_i \neq x_j$  one variable is assigned 1 and the other 0, hence every edge connects a node of  $N_1$  with a node of  $N_2$ .

The graph  $G$  has  $n$  nodes and  $m$  edges. We know that we can test in  $O(n+m)$  time whether a graph is bipartite using DFS or BFS.

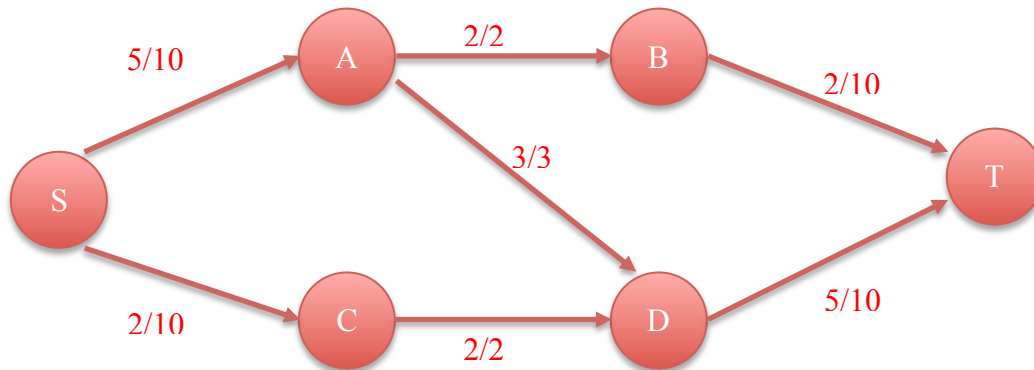
Grading Rubrics:

1. correctly define the graph and claim that problem can be reduced into checking whether  $G$  is bipartite. 5 pts
2. Justify the correctness of the algorithm. 7 pts
3. Justify the running time of the algorithm. 3 pts

Consider the following problem: Given a flow network  $G$  with source  $s$  and sink  $t$ , and a unique  $s - t$  min cut, find  $k$  edges that, when deleted, reduce the maximum  $s - t$  flow in the graph by as much as possible.

- Find the  $s - t$  min cut in  $G$ .
- Sort edges on the min cut in decreasing order of weight
- Delete the first  $k$  (highest capacity) edges from the ordered list

Counter example:



- Saying statement is false without a correct counterexample (5 points)
- Proving the statement is True because all edges on the min-cut are saturated and hence, removing each edge from the cut will have a reduction in max flow equal to the capacity of that edge (5 points)

4) 15 pts

Given a list of non-negative numbers and a target integer  $k$ . Design a dynamic programming algorithm to check if the array has a continuous subarray of size at least 2 that sums up to a multiple of  $k$ , that is, sums up to  $n*k$  where  $n$  is also an integer.

Example:

Input: [23, 2, 4, 6, 7],  $k=6$

Output: True

Explanation: Because [2, 4] is a continuous subarray of size 2 and sums up to 6.

Example:

Input: [23, 2, 6, 4, 7],  $k=6$

Output: True

Explanation: Because [23, 2, 6, 4, 7] is a continuous subarray of size 5 and sums up to 42.

First, if  $k==0$ , we check whether there are two continuous zeros in the list. If so, return True. Otherwise, return False.

Then, for  $k!=0$ , we use the following algorithm.

Assume  $dp[i][j]$  representing the sum of sub-array from  $i$  to  $j$  in array. If there is a pair of  $(i,j)$  satisfies  $dp[i][j] \% k == 0$ , the algorithm returns true. Otherwise, it returns false.

Initialize all elements in the  $dp$  array to zero.

We can calculate  $dp[i][j]$  in the follow way:

Assume  $n=length$  of list and we use  $nums$  to represent the input list

For  $i$ , we traverse from  $n - 1$  to 0,

Set  $dp[i][i] = nums[i]$

For  $j$ , we traverse from  $i+1$  to  $n-1$ ,

$dp[i][j] = dp[i+1][j] + nums[i]$

after setting the value  $dp[i][j]$ , we check whether  $dp[i][j] \% k == 0$ , if so, return True.

After checking all  $(i,j)$  pairs through above process, return False if there is no such  $(i,j)$  pair that meets the condition.

Grading Rubrics:

1. Correctly take care of the case when  $k=0$ . 3 pts
2. Correctly define the dp array and point out how dp array relates to the output of the algorithm. 3 pts
3. Correctly initialize the dp array. 3 pts
4. Correctly define how to fill in dp array. 6 pts

5) 10 pts

Analyze the run time complexity of the following function with respect to  $n$ .

```
function printalot (n: an integer power of 2)
    if n > 1 {
        printalot (n/2)
        printalot (n/2)
        printalot (n/2)
        for i=1 to n4
            printline ("Are we done yet?")
        endfor
    }
```

If  $T(n)$  is the number of lines printed for input  $n$ , then  $T(n) = 3T(n/2) + n^4$  with a base case of  $T(1) = 0$ .

For an asymptotic solution, we can apply the master theorem with  $n^{\log_b a} = n^{\log_2 3}$  and  $f(n) = n^4$ . Since  $f(n)$  polynomially dominates  $n^{\log_b a}$ ,  $T(n) = \Theta(f(n)) = \Theta(n^4)$  by the master theorem.

Grading:

- Used Big-O notation instead of Big-Theta notation (-1 point)
- Wrong recursion (-3 points)
- Wrong approach (-7 points)

6) 15 pts

Recall the following T/F question on exam 2:

*Bellman-Ford algorithm is not guaranteed to work on all undirected graphs.*

Which was a true statement.

Prove that if the Bellman-Ford algorithm were able to find a shortest simple path in an undirected graph with negative cost edges, then P must be equal to NP. In other words, prove that the problem of finding a shortest simple path in an undirected graph with negative cost edges is NP-complete. You can use ANY NP-complete problem that we have discussed in class for your reduction.

Step #1. Prove X (the given problem) is in NP

Certificate: ordered list of vertices on the shortest simple path

Certifier: Check if all vertices are visited. There has to be an edge between every pair of adjacent vertices in the order. Check if the given certificate is the shortest path.

Step #2. Choose a problem Y that is known to be NP-complete

Y can be HAM-PATH or HAM-Cycle.

Step #3. Prove that  $Y \leq_p X$

Reduction from HAM-PATH

- Given undirected graph  $G = (V, E)$
- Transform to graph  $G'$  with -1 weight for every edge.
- Claim:  $G$  has a HAM-PATH if and only if SP w/ negative edge has a path of length  $-|V| + 1$  between two arbitrary nodes. (Need to call the blackbox  $n$  times each time using a different starting point and finding the shortest path to all other nodes)

Reduction from HAM-Cycle

- Given undirected graph  $G = (V, E)$
- Transform to graph  $G'$  with -1 weight for every edge. Remove an arbitrary edge  $(uv)$  in  $G'$ , add two new nodes  $v'$  and  $u'$ , and add edges  $uu'$  and  $vv'$  with weight -1.
- Claim:  $G$  has a HAM-Cycle if and only if SP w/ negative edge has a path of length  $-|V| - 1$  between any two points  $v'u'$ . (Need to call the blackbox  $m$  times, each time removing a different edge)



Grading:

- Didn't mention X is in NP (-5 points)
- Mentioned X is in NP, but didn't provide certificate/certifier (-3 points)
- Didn't mention a problem Y is known to be NP-complete (-3 points)
- Missing step #2 and #3 (-10 points)
- Missing/wrong proof for step #3 (-5 points)
- Wrong notation (-1 point)
- Wrong approach without mentioning three steps (-15 points)

7) 10 pts

720 Students have pre-enrolled for the “Analysis of Algorithms” class in Fall. Each student must attend one of the 16 discussion sections, and each discussion section  $i$  has capacity for  $D_i$  students. The happiness level of a student assigned to a discussion section  $i$  is proportionate to  $\alpha_i (D_i - S_i)$ , where  $\alpha_i$  is a parameter reflecting how well the air-conditioning system works for the room used for section  $i$  (the higher the better), and  $S_i$  is the actual number of students assigned to that section. We want to find out how many students to assign to each section in order to maximize total student happiness.

Express the problem as a linear programming problem.

$$\sum_{i=1}^{16} \alpha_i (D_i - S_i) \quad \leftarrow \text{maximum}$$

$$\sum S_i = 720$$

Our variables will be the  $S_i$  's.

Objective function:

$$\text{maximize } \sum_{i=1}^{16} \alpha_i (D_i - S_i)$$

$$\text{subject to: } D_i - S_i \geq 0 \quad \text{for } 0 < i \leq 16$$

$$S_i \geq 0 \quad \text{for } 0 < i \leq 16$$

$$\sum_{i=1}^{16} S_i = 720$$

Grading:

- Objective function (4 points)
- Each constraint (2 points)