CS570
Analysis of Algorithms
Spring 2016
Exam II

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	20	
Problem 4	20	
Problem 5	25	
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.

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

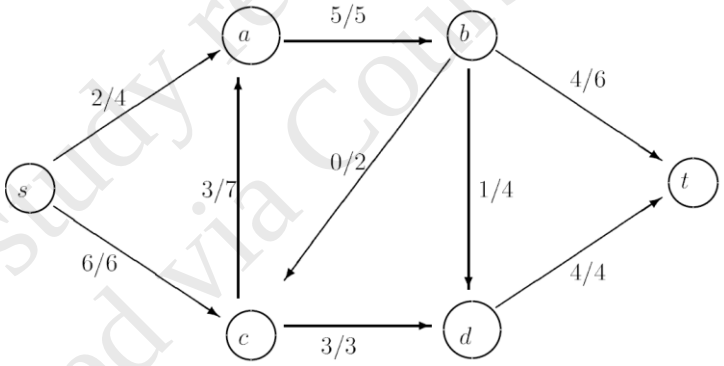
[TRUE]
The number of iterations it takes Bellman-Ford to converge can vary depending on the order of nodes updated within an iteration.

[FALSE]
In a flow network, if the capacity of every edge is odd, then there is a maximum flow in which the flow on each edge is odd.

[TRUE]
Maximum value of an s-t flow could be less than the capacity of a given s-t cut in a flow network.

[FALSE]
Any Dynamic Programming algorithm with n^2 unique sub-problems will run in $O(n^2)$ time.

[TRUE]
The following flow is a maximal flow.
Note: The notation *a/b* describes *a* units of flow on an edge of capacity *b*.



[FALSE]

In a circulation network, there is a feasible circulation with demands $\{d_v\}$, if $\sum_v d_v = 0$

[FALSE]

An optimal solution to a 0/1 knapsack problem will always contain the object i with the greatest value-to-cost ratio V_i/C_i.

[TRUE]

The dynamic programming solution presented in class for the 0/1 knapsack problem is not an efficient solution.

[FALSE]

For any edge e that is part of the minimum cut in G, if we increase the capacity of that edge by any integer k>1, then that edge will no longer be part of the minimum cut.

[TRUE]

Ford-Fulkerson Algorithm cannot solve the max-flow problem in a flow network in polynomial time, however, there are other algorithms that can solve this problem in polynomial time.

2) 15 pts

Suppose you have a DAG with costs $c_e > 0$ on each edge and a distinguished vertex s. Give a dynamic programming algorithm to find the most expensive path in the graph that begins at s. Your solution should include the recurrence formula for the cost of the path, pseudo code to show your implementation, and complexity analysis. Your algorithm should return the most expensive path. For full credit, your algorithm's runtime should be linear.

Let's consider the vertices in topological order. Define OPT(v) as the longest distance from s to v.

Our recurrence is then:
OPT(v) = max_{u ∈ adj(v) } {OPT(u) + w(u,v)},
where adj(v) is the set of vetices, each of which has an edge to node v.

Base case OPT(s) = 0; OPT(v) = -infinity, if v != s and adj(v) is empty

Algorithm:

Do topological ordering and get ordering indices as 1,….,V

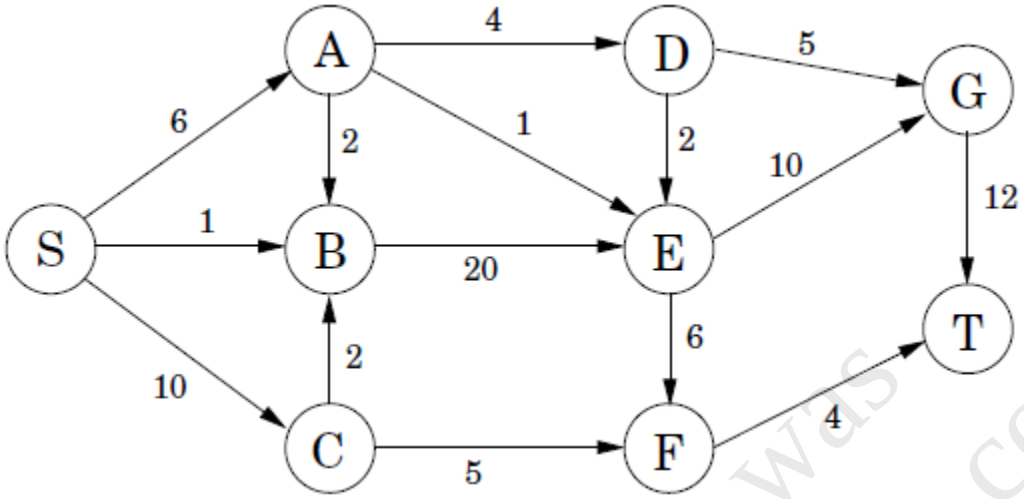
```
For v = 1 to V
        If v == s
                OPT(s) = 0
        else if  adj(v) is empty
                OPT(v) = -infinity
        else
                OPT(v) = max_{u ∈ adj(v) } {OPT(u) + w(u,v)},

        End
        Record the node u ∈ adj(v)  that achieves OPT(v), denote it as p(v);
End

/* the above running time is O(|V|+|E|)

longest distance = max_{v} {OPT(v)};
v* = argmax_{v} {OPT(V)};

/*the above running time is O(|V|)
```

```
/*Return the longest path as follows******************

Path = [p(v*), v*];  /* the longest path
u = v*;
While (p(u)!= s)
        u = p(u);
        Path = [p(u), Path];
End

Return Path;

/*the above running time is O(|V|)
```

Complexity: O(|V|+|E|)

3) 20 pts

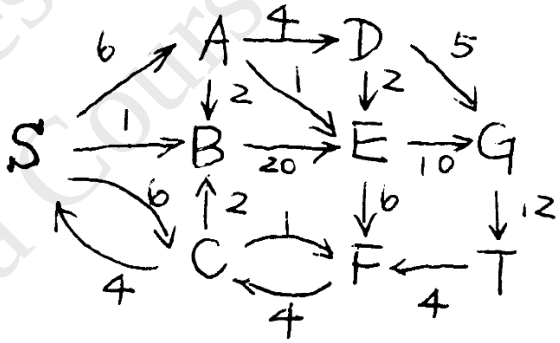
For the following network, with edge capacities as shown,



a) Find the maximum flow from *S* to *T* using the Ford Fulkerson algorithm. You need to show the work involved in each iteration including the residual graph and augmenting path.

Solution:

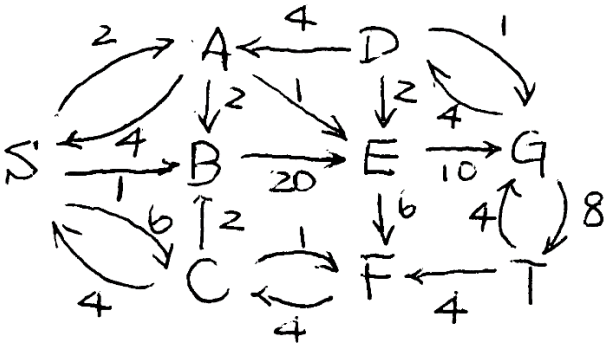
Iteration 1:



Residual graph:

Augmenting path: S-C-F-T (flow value = 4)

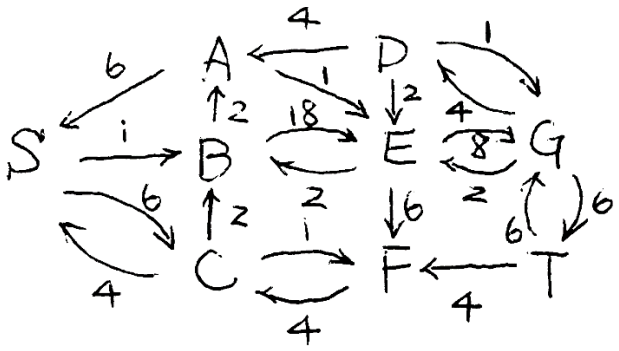
Iteration 2:



Residual graph:

Augmenting path: S-A-D-G-T (flow value = 4)

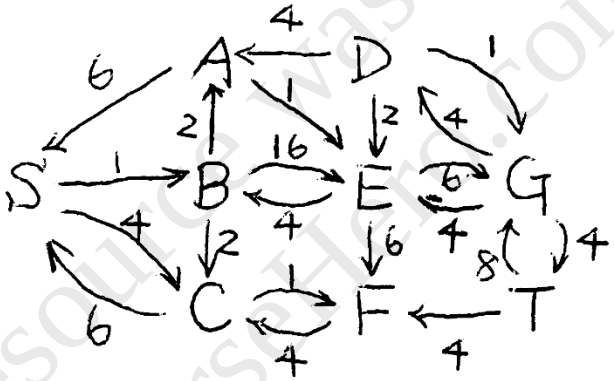
Iteration 3:



Residual graph:

Augmenting path: S-A-B-E-G-T (flow value = 2)

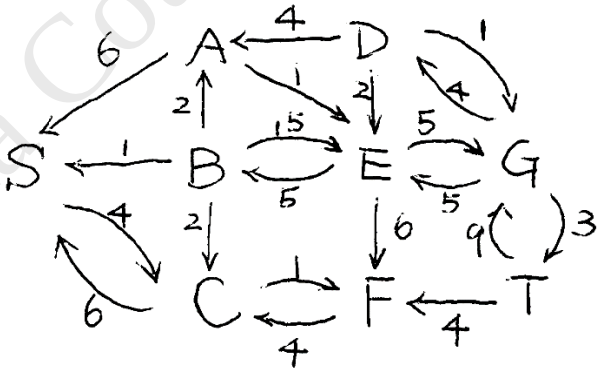
Iteration 4:



Residual graph:

Augmenting path: S-C-B-E-G-T (flow value = 2)

Iteration 5:



Residual graph:

Augmenting path: S-B-E-G-T (flow value = 1)

Since there is no more augmenting path in the last residual graph, the maximum flow value = 4 + 4 + 2 + 2 + 1 = 13

b) Using the solution to part a, find an s-t min cut. You need to show the steps to find the min cut.

Solution:

In the last residual graph in a), starting from *S*, the nodes can be reached are *S*, *C*, and *F*. Thus, the minimum cut from a): {S, C, F} {A, B, D, E, G, T}.

4) 20 pts

You are given a sequence of *n* numbers (positive or negative): $x_1, x_2, \ldots, x_n$. Your job is to select a subset of these numbers of maximum total sum, subject to the constraint that you can't select two elements that are adjacent (that is, if you pick x_i then you cannot pick either x_{i-1} or x_{i+1}). On the boundaries, if x_1 is chosen, x_2 cannot be chosen; if x_n is chosen, then x_{n-1} cannot be chosen. Give a dynamic programming solution to find, in time polynomial in *n*, the subset of maximum total sum. Please give the optimal cost equation, and pseudo code for your solution. Also state the complexity of your solution.

Solution: Let sum_i be the maximum sum of the numbers $x_1, x_2, \ldots, x_i$ given the adjacency constraint.

$$sum_0 = 0$$
$$sum_1 = max(0, x_1)$$
$$sum_i = max(sum_{i-2} + x_i, sum_{i-1})$$

This last step works because either we include x_i, in which case we also want to include the bestsolution on up to i – 2, or we don't include x_i, in which case we can just use the best solution on i – 1.

Our final answer is then just sum_n.To calculate the set that gives the max sum, we could simply keep pointers back from i to either i – 1 or i – 2 depending on which one was bigger (or we could go back and check which was bigger).We follow those pointers, including appropriate numbers. Because there are n subproblems, and each subproblem takes O(1) time to solve, this runs in O(n) time.

```
01: OPT[-1, 0, 1, …, n] := [0, …, 0]
02: for i = 1, …, n:
03:     OPT[i] := max(OPT[i-2]+x_i, OPT[i-1])
04: S := [], j := n
05: while j > 0:
06:     if OPT[j] = OPT[j-2] + x_j:
07:         S.append(x_j), j := j-2
08:     else:
09:         j := j-1
10: return S
```

5) 25 pts

We define a most vital edge of a network as an edge whose deletion causes the largest decrease in the maximum s-t-flow value. Let f be an arbitrary maximum s-t-flow. Either prove the following claims or show through counterexamples that they are false:

(a) A most vital edge is an edge e with the maximum value of c(e).

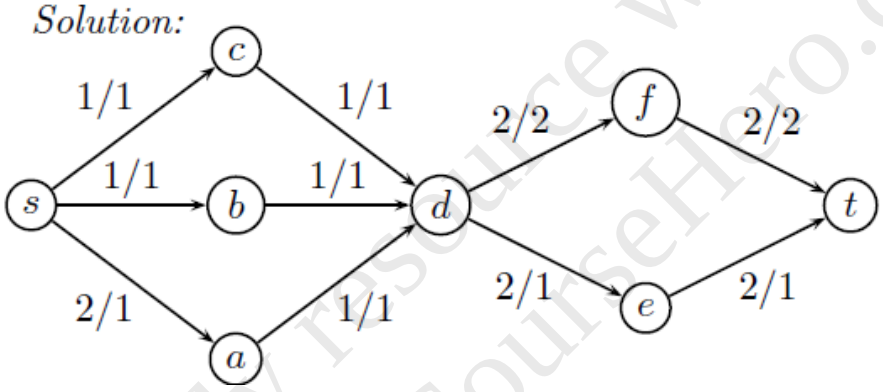
(b) A most vital edge is an edge e with the maximum value of f (e).

(c) A most vital edge is an edge e with the maximum value of f (e) among edges belonging to some minimum cut.

(d) An edge that does not belong to any minimum cut cannot be a most vital edge.

(e) A network can contain only one most vital edge.

Solution:



This is a counterexample for (a), (b) (d) and (e). The specified row f is a maximum flow with the value three. The deletion of any edge decreases the flow value by one. Hence, each edge is a most vital arc ((e) is false). The edge e = (s, b) has neither the maximum value of c(e) nor the maximum value of f(e); still, it is a most vital arc ((a) and (b) are false). The edge (s, a) does not belong to any minimum cut; still, it is a most vital arc ((d) is false).

Part (c) is false and here is the counter example: all the edges except the one connecting s belong to at least one min-cut, and the flow on each of those edges is 1, and therefore also the maximum . However, deleting any of the edges belonging to min-cut can only reduce the max-flow value by 1, in contrast, deleting the edge connecting s can decrease the flow by 2.