1	2	3	4	5	Total

The University of Texas at Austin
Department of Electrical and Computer Engineering

EE w360C — Algorithms — Summer 2013

Exam III — Friday 2 August 2013

EID: _____

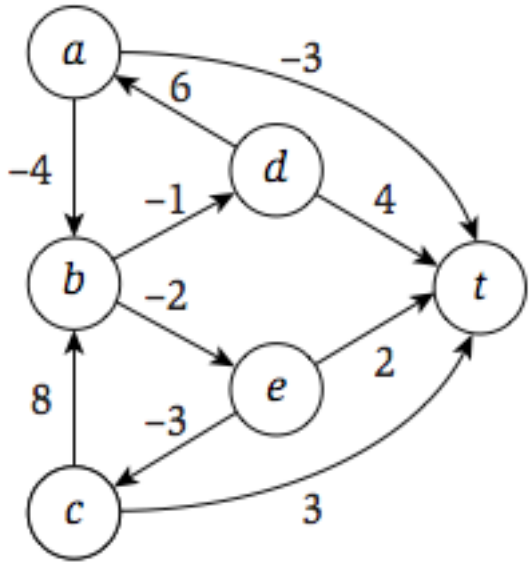
Name: _____

Instructions.

- You have 75 minutes to complete the exam. The maximum possible score is 75 including a 15 marks bonus question.
- The exam consists of 5 questions and 7 printed pages.
- It is a closed book / closed notes exam. No calculators, laptops, or other devices are allowed.
- Write your answers legibly on the test pages. Use back of test pages for scratch work. Show intermediate answers and process of solving questions.
- If there is any confusion, write down your assumptions and proceed to answer the question.
- In questions where you have to write an algorithm, you can describe it with reference to the algorithms we discussed in class. For example, you can say BFS with some additional operation at one step. Or use the output of topological sort directly etc.

1. Run the Bellman-Ford algorithm on the graph below. Show the matrix M and list the shortest paths from every vertex to t in the graph.

10 pts



Solution:

M	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3	-4	-6	-6
b	∞	∞	0	-2	-2	-2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

Shortest paths:

$a \to b \to e \to c \to t$

$b \to e \to c \to t$

$c \to t$

$d \to a \to b \to e \to c \to t$

$e \to c \to t$

2. Your friends have been studying the closing prices of tech stocks, looking for interesting patterns. They've defined something called a *rising trend*, as follows.

15 pts

They have the closing price for a given stock recorded for n days in succession; let these prices be denoted $P[1], P[2], \ldots, P[n]$. A *rising trend* in these prices is a subsequence of the prices $P[i_1], P[i_2], \ldots, P[i_k]$, for days $i_1 < i_2 < \ldots < i_k$, so that

- $i_1 = 1$, and
- $P[i_j] < P[i_{j+1}]$ for each $j = 1, 2, \ldots, k1$.

Thus a rising trend is a subsequence of the days—beginning on the first day and not necessarily contiguous—so that the price strictly increases over the days in this subsequence.

They are interested in finding the longest rising trend in a given sequence of prices.

Example. Suppose $n = 7$, and the sequence of prices is 10, 1, 2, 11, 3, 4, 12. Then the longest rising trend is given by the prices on days 1, 4, and 7. Note that days 2, 3, 5, and 6 consist of increasing prices; but because this subsequence does not begin on day 1, it does not fit the definition of a rising trend.

(a) (5 pts) Show that the following algorithm does not correctly return the *length* of the longest rising trend, by giving an instance on which it fails to return the correct answer.

```
Define i=1
  L=1
For j=2 to n
  If P[j]>P[i] then
    Set i=j.
    Add 1 to L
  Endif
Endfor
```

In your example, give the actual length of the longest rising trend, and say what the algorithm above returns.

Solution:
1 4 2 3
Actual length: 3
Algo returns: 2

(b) (10 pts) Give an efficient algorithm that takes a sequence of prices $P[1], P[2], \ldots, P[n]$ and returns the length of the longest rising trend.

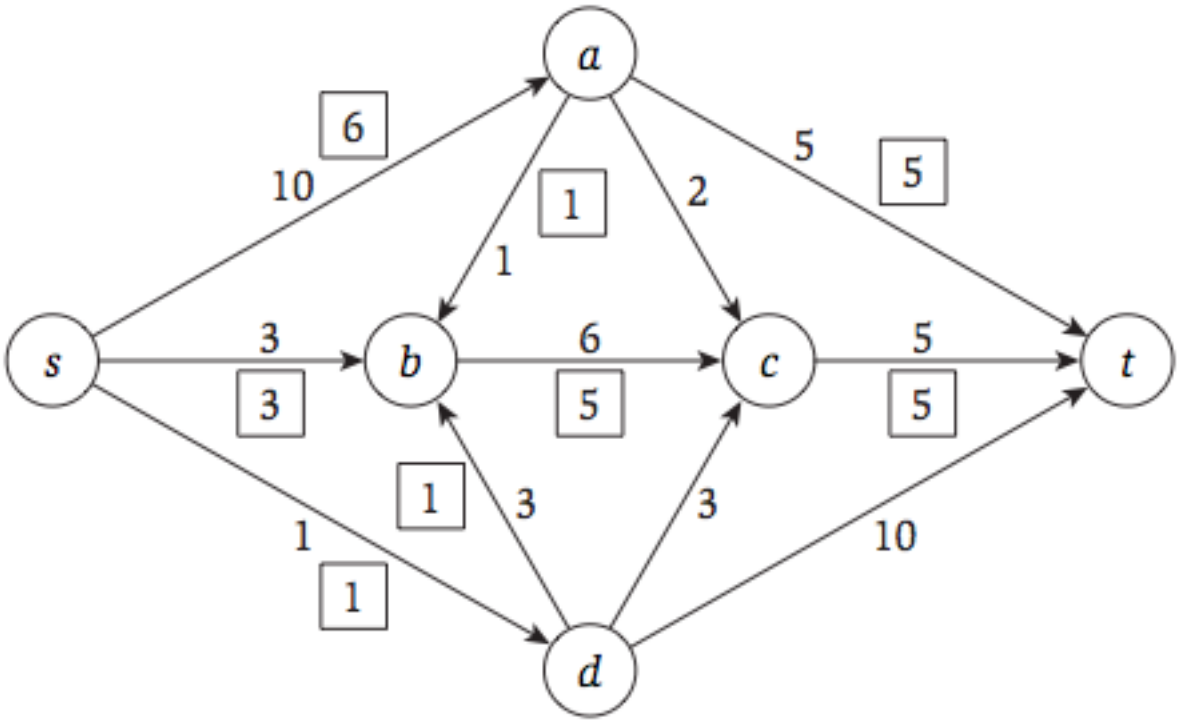
Solution:

```
For  i = 1, 2, ..., n
  M[i] = 1
  If  P[i] > P[1]
    For  j = 1, 2, ..., i − 1
      If  P[j] < P[i]  And  M[i] < M[j]
        M[i] = 1 + M[j]
```

Essentially we are using the formula:
$OPT(i) = 1 + max_{j|j<i, P[j]<P[i]} OPT(j)$

3. The following figure shows a flow networkon which an s-t flow has been computed. The capacity of each edge appears as a label next to the edge, and the numbers in boxes give the amount of flow sent on each edge. (Edges without boxed numbers have no flow being sent on them.)

20 pts



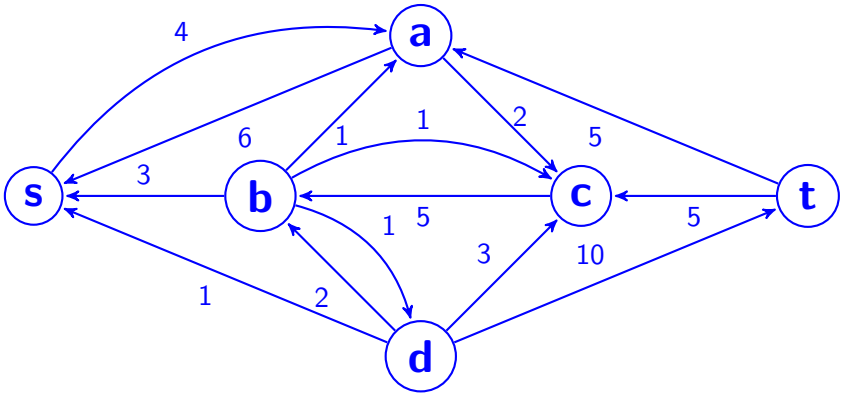
(a) (5 pts) What is the value of this flow?

Solution:

10

(b) (5 pts) Draw the residual graph.

Solution:



(c) (5 pts) Find the maximum (s,t) flow in this graph.

Solution:
11

(d) (5 pts) Find a minimum s-t cut and also say what its capacity is.

Solution:
$A = \{s, a, b, c\}, B = \{d, t\}, C(A, B) = 11$

4. We define the *Escape Problem* as follows. We are given a directed graph $G = (V, E)$ (picture a network of roads). A certain collection of nodes $X \subset V$ are designated as *populated nodes*, and a certain other collection $S \subset V$ are designated as *safe nodes*. (Assume that X and S are disjoint.) In case of an emergency, we want evacuation routes from the populated nodes to the safe nodes. A set of evacuation routes is defined as a set of paths in G so that (i) each node in X is the tail of one path, (ii) the last node on each path lies in S, and (iii) the paths do not share any edges. Such a set of paths gives a way for the occupants of the populated nodes to "escape" to S, without overly congesting any edge in G.

15 pts

Given G, X, and S, show how to decide in polynomial time whether such a set of evacuation routes exists.

Solution:
Create a network flow problem like this. Assign unit capacity to all existing edges. Introduce source node s and connect it to each node in X with a unit capacity directed edge. Introduce sink node t and connect each node in S with a directed edge to t with capacity $|X|$. Caclulate max-flow. We argue that max-flow=$|X|$ if and only if such routes exist. [Almost full credit till this point]

Part I: If required routes exist, max-flow=$|X|$.
Use $|X|$ unit capacity edges from source to each of the nodes in X. From there, we have a path from each node in X to some node in S using unique edges. Thus the $|X|$ units of flow can reach nodes in S from where they have $|X|$ capacity paths to the sink.

Part II: If max-flow=$|X|$, required routes exist.
If max-flow=$|X|$, each node in X is receiving unit capacity flow from s, and since all this flow is reaching t and each edge between X and S is unit capacity, all of this flow must be using unique edges. Thus we have a unique path from every node in X to some node in S.

Note that you may have edges between X and they might even be used. It is also not required to have $|X| = |S|$. Consider a small example of a graph with nodes X_1, X_2, X_3, S_1, S_2 and edges $(X_1, X_2), (X_2, S_1), (X_2, S_2), (X_3, S_2)$. The required routes exist in this graph. Also, there can be any number of interconnecting nodes between X and S.

15 pts

5. Bonus Problem: Recall that in the Knapsack Problem, we have n items, each with a weight w_i and a value v_i. We also have a weight bound W, and the problem is to select a set of items S of highest possible value subject to the condition that the total weight does not exceed W, that is, $\Sigma_{i\in S} w_i \leq W$. Here's one way to look at the approximation algorithm that we designed in this chapter. If we are told there exists a subset ϑ whose total weight is $\Sigma_{i\in\vartheta} w_i \leq W$ and whose total value is $\Sigma_{i\in\vartheta} v_i = V$ for some V, then our approximation algorithm can find a set A with total weight $\Sigma_{i\in A} w_i \leq W$ and total value at least $\Sigma_{i\in A} v_i \geq V/(1+\epsilon)$. Thus the algorithm approximates the best value, while keeping the weights strictly under W.

Now, as is well known, you can always pack a little bit more for a trip just by "sitting on your suitcase", in other words, by slightly overflowing the allowed weight limit. This too suggests a way of formalizing the approximation question for the Knapsack Problem, but it's the following, different, formalization.

Suppose that you are given n items with weights and values, as well as parameters W and V; and you are told that there is a subset ϑ whose total weight is $\Sigma_{i\in\vartheta} w_i \leq W$ and whose total value is $\Sigma_{i\in\vartheta} v_i = V$ for some V. For a given fixed $\epsilon > 0$, design a polynomial-time algorithm that finds a subset of items A such that $\Sigma_i w_i \leq (1+\epsilon)W$ and $\Sigma_{i\in A} v_i \geq V$. In other words, you want A to achieve at least as high a total value as the given bound V, but you are allowed to exceed the weight limit W by a factor of $1+\epsilon$.

Solution:
Take $\theta = \epsilon W/n$ and scale all individual item weights by θ while rounding up i.e. $w_i' = \lceil w_i/\theta \rceil$. Run the standard knapsack algorithm with a weight limit of $W' = W/\theta + n$. [Almost full credit till this point]

Part I: Algorithm is efficient
It takes time $O(nW') = O(nW/\theta + n^2) = O(n^2/\epsilon + n^2)$ so the algorithm is efficient.

Part II: $\Sigma_i w_i \leq (1+\epsilon)W$
If A is a solution of the scaled problem:
$\Sigma_{i\in A} w_i \leq \theta \Sigma_{i\in A} w_i' \leq \theta(W/\theta + n) = W + n\theta = W + \epsilon W = (1+\epsilon)W$
This means the weight limit is satisfied.

Part III: $\Sigma_{i\in A} v_i \geq V$
If B is *any* solution satisfying weight limit for original problem, then $\Sigma_{i\in B} w_i' \leq \Sigma_{i\in B} w_i/\theta + \theta \leq W/\theta + n$ and thus it is *one* solution for the scaled problem as well and therefore *any* solution of the scaled problem achieves a value *at least* V.