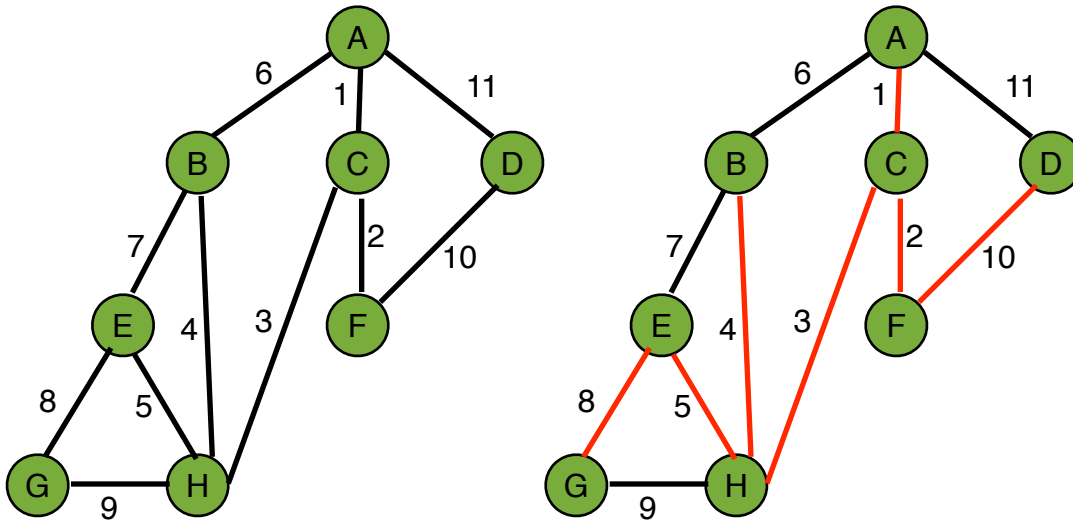


CS 312 - Algorithm Design

Final - Spring 2007

1. (10 pts.) Find a Minimum Spanning Tree for the following graph.



2. (10 pts.) As you leave for summer vacation, you may be wondering which of your computer science materials you should take with you. You have limited space in your backpack so you would like to maximize the value of what you take. Below is the value and weight of various items. The total weight your backpack can hold is 10. Use the Knapsack algorithm to decide what to take. Show the array that you create and then list the items you should take.

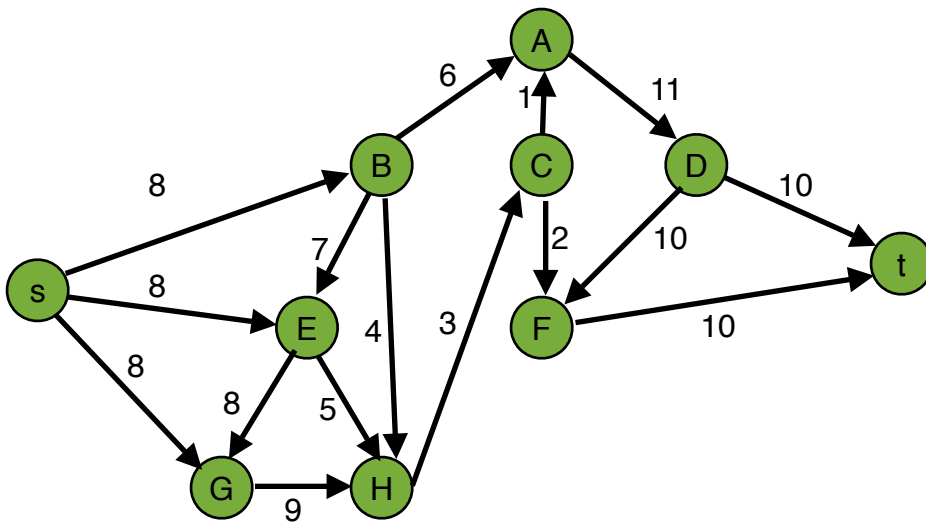
Item	Weight	Value
Algorithms textbook	4	10
C++ textbook	4	1
Laptop	5	7
Memory stick	1	3
Power cord	2	5

Item	0	1	2	3	4	5	6	7	8	9	10
Nil	0	0	0	0	0	0	0	0	0	0	0
Alg	0	0	0	0	10	10	10	10	10	10	10

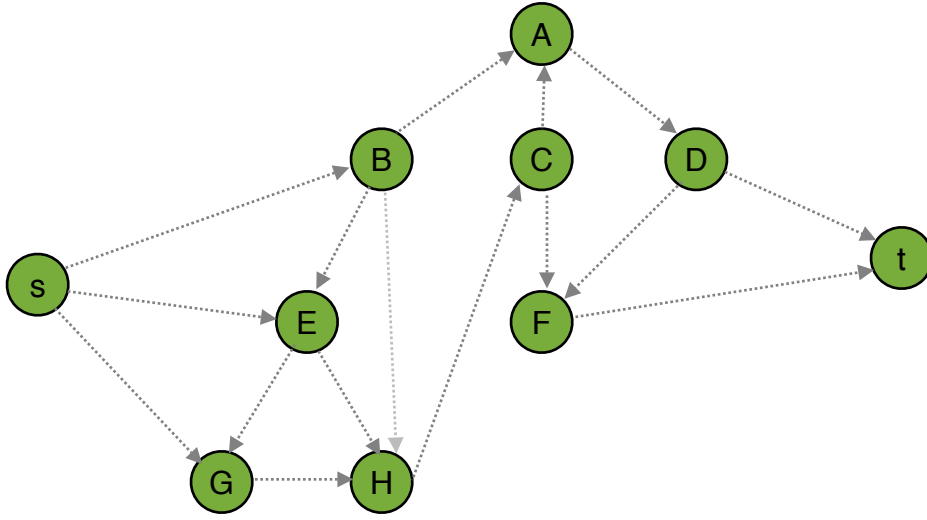
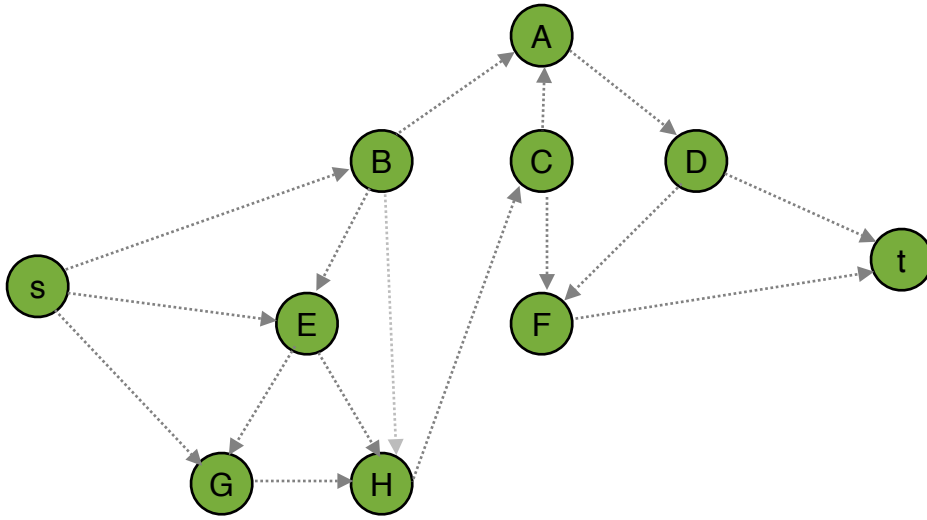
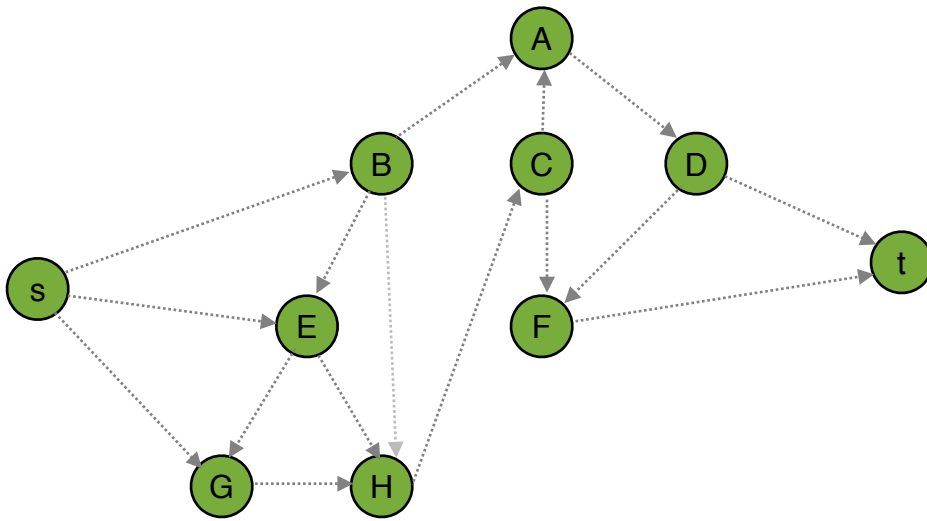
Item	0	1	2	3	4	5	6	7	8	9	10
C++	0	0	0	0	10	10	10	10	11	11	11
Laptop	0	0	0	0	10	10	10	10	11	17	17
Mem stick	0	3	3	3	10	13	13	13	13	17	20
Power	0	3	5	8	10	13	15	18	18	18	20

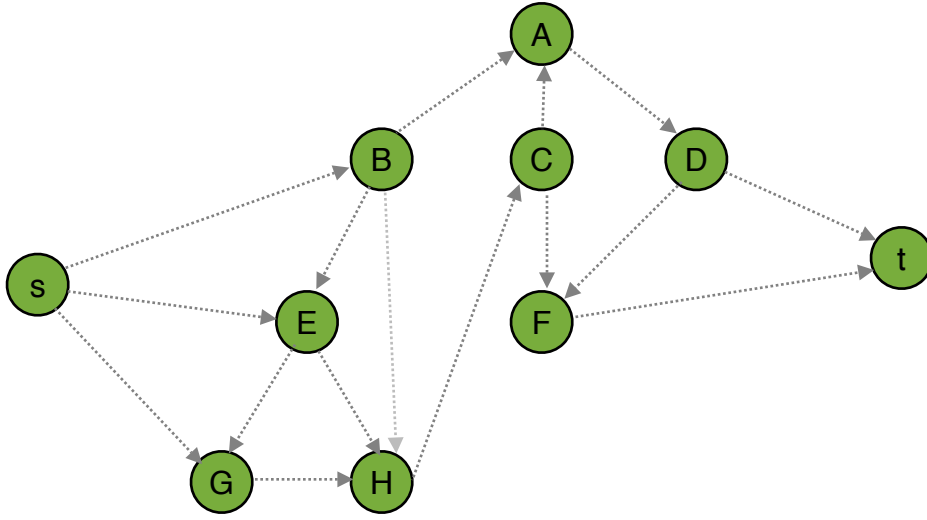
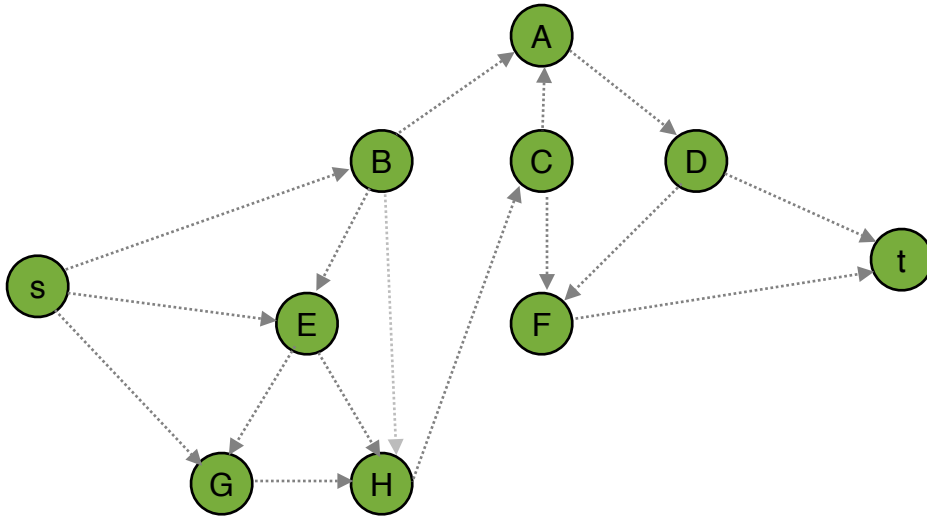
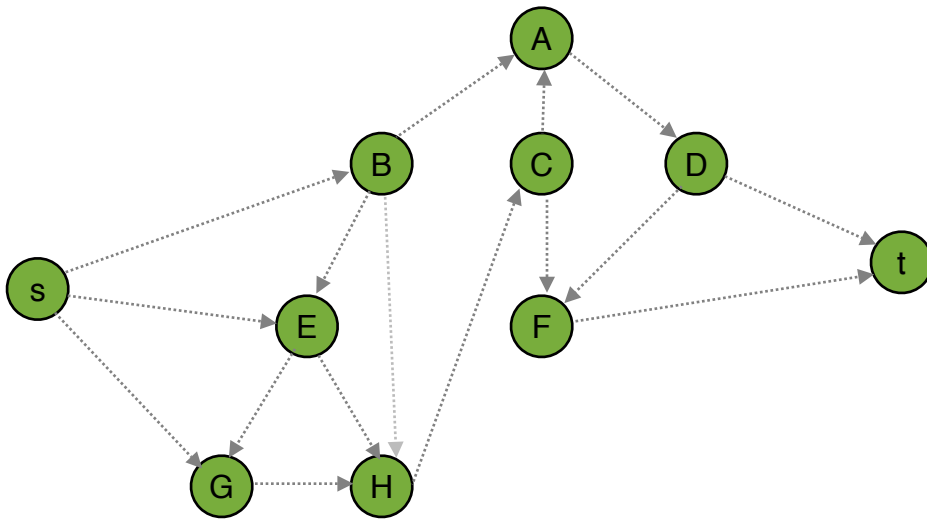
I would take the Memory Stick, Laptop, Algorithms text. This gives a value of 20 and a weight of 10.

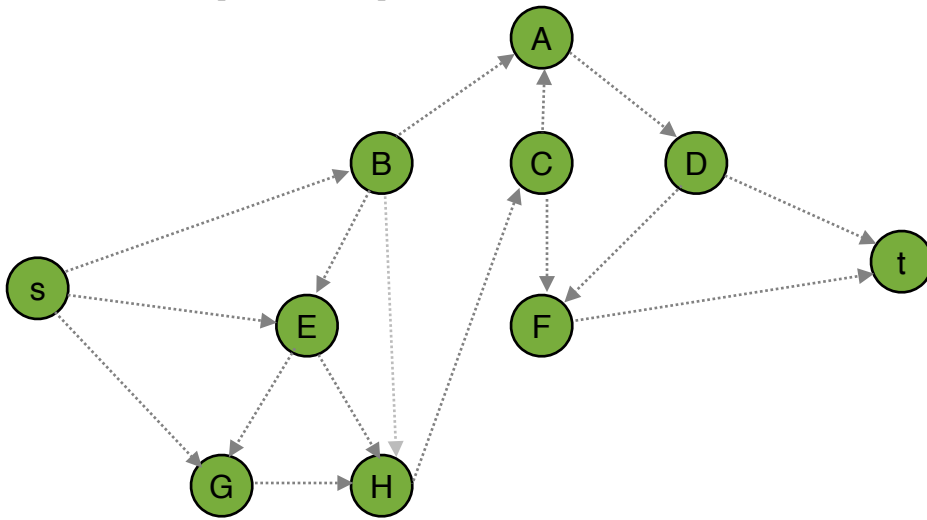
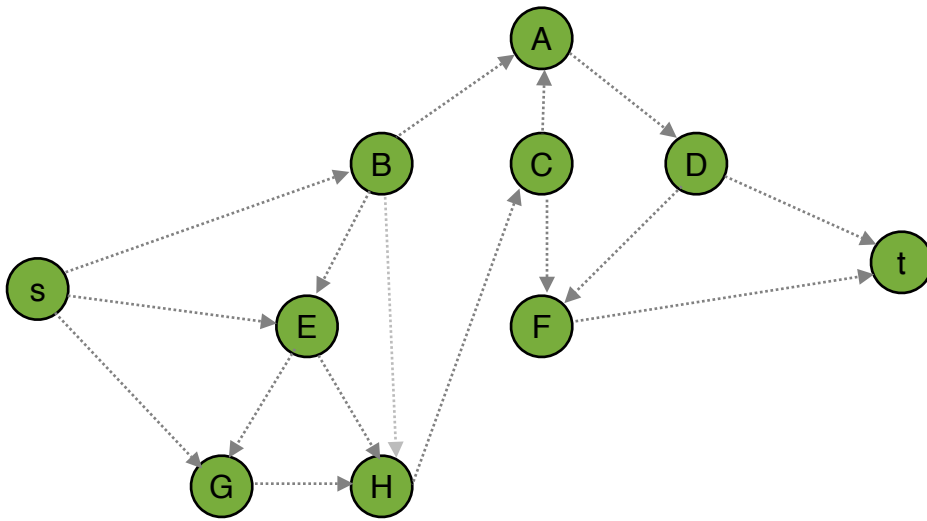
3. (10 pts.) Determine the maximum flow of the following network. Use the copies of the graph on the following pages to help you build the residual graphs. Please turn in these pages with your exam.



Max flow should be 9. There are numerous ways to get that. The minimum cut has $A = \{s, B, E, G, H\}$.



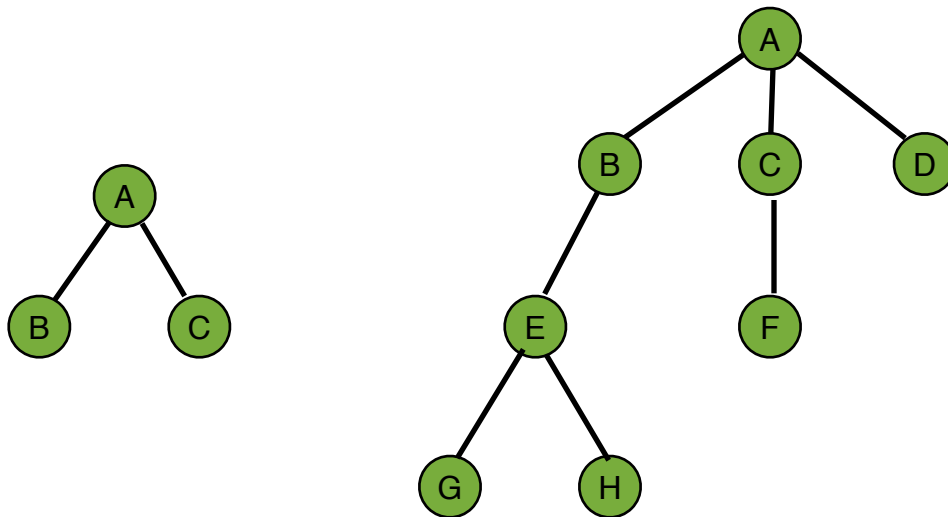




4. (15 pts.) In class, we discussed the Vertex Cover problem. A vertex cover for G is a subset of vertices $U \subseteq V$, such that every edge is adjacent to at least one vertex in U . We discovered that, for general graphs, finding a vertex cover is in NP. In general there are several possible vertex covers. We are interested in those for which the size of U is minimal. We call such a cover a minimal vertex cover.

For this question, I would like you to consider vertex covers for trees, rather than general graphs.

- a. For each of the following trees, find a vertex cover of minimum size that does not include a leaf vertex.



The minimal vertex cover for the first tree is $\{A\}$. For the second tree it is $\{A, C, E\}$.

- b. Prove the following statement: For any tree, there is always a minimal vertex cover that does not contain any leaves of the tree.

A leaf node can only have one edge incident on it. Thus, anytime that we include a leaf node in a vertex cover, it can cover at most one edge. An interior node, other than the root, always has at least 2 edges incident on it. Thus, a non-leaf node, other than the root, can always cover at least 2 edges. So, we always do at least as well picking the parent of a leaf rather than a leaf.

- c. Provide an $O(m+n)$ algorithm to find a minimum vertex cover in a tree, where n is the number of nodes in the tree and m is the number of edges in the tree. **It is more important that you provide a careful English explanation of your algorithm than a pseudocode version of your algorithm.**

Let S be the vertex cover. Initially, S is empty.

Mark all edges as not covered.

Do a depth first search. When you backtrack, if the edge you backtrack over is not covered, add the node that you backtrack to to S . Mark all edges incident from this node as covered.

5. (10 pts.) Imagine a natural disaster, such as a hurricane, that affects a fairly large area of land. Suppose there are n injured people who need to be sent to hospitals and k hos-

pitals available. Design an algorithm that will tell you which patients to send to which hospitals under the following constraints:

- A patient should travel no more than 30 minutes to get to a hospital
- The load on the hospitals should be balanced. That is, no more than $\lceil n/k \rceil$ patients should be sent to any hospital.

If there is no way to satisfy these constraints, your algorithm should say that it is not possible. **It is more important that you provide a careful English explanation of your algorithm than a pseudocode version of your algorithm.**

This is a network flow problem. Create a bipartite graph where the two sets of nodes represent patients and hospitals. Create an edge from a patient to a hospital if the patient can reach the hospital in no more than 30 minutes. Place a capacity on the edges created between patients and hospitals. Add a source node, s , with an edge to each patient with a capacity of 1. Add a sink node with an edge from each hospital to the sink node. Place a capacity of $\lceil n/k \rceil$ on these edges.

Now, solve the network flow problem. If there is a solution, the edges from patients to hospitals with flow of 1 indicate where each patient should go. If there is no solution, it means the constraints cannot be satisfied.

6. (10 pts.) A permutation of the numbers 1 to n is a rearrangement of the numbers in arbitrary order, such that each number appears exactly once. For example, two different permutations of the numbers 1-5 are: 1, 2, 3, 5, 4 and 3, 5, 1, 2, 4.

The distance between a value and the next higher value is defined to be the difference between the position of the value and the position of the next higher value, where the higher value appears to the right of the lower value. If the lower value appears to the right, we define the distance to be 0. For example, in 3, 1, 2, 5, 4, the distance between 1 and 2 is 1, while the distance between 1 and 3 is 0. The maximum distance occurs between the values of 3 and 4, where 3 is in position 1 and 4 is in position 5, giving a distance of 4.

- a. Below is an algorithm to find the maximum distance in such a permutation of numbers. What is the $O()$ cost of this algorithm?

```
max = 0
for i = 1..n-1 {
  for j = i+1..n {
    if (a[i] + 1 = a[j]) {
      if (j - i > max) {
        max = j - i;
        break;
      }
    }
  }
}
return max;
```

$O(n^2)$

- b. Use dynamic programming to define another algorithm to solve this problem in $O(n)$ time. Explain your algorithm.

```
// Preprocess the input, determining what position each
// number is in.
int[] pos = new int[permutation.length];
for (int i = 0; i < pos.length; i++) {
  pos[permutation[i]] = i;
}

// Use a dynamic programming solution. For each number, determine
// the best distance up to and including that number.
int[] distance = new int[pos.length];
distance[0] = 0;
for (int i = 1; i < pos.length; i++) {
  // The best distance so far is the maximum of the distance from
  // this number to its predecessor OR the best distance found up
  // until the preceding value.
  distance[i] = Math.max(distance[i-1], pos[i] - pos[i-1]);
}
return distance[distance.length-1];
```

7. (10 pts.) An Independent Set of a graph is a set of vertices, such that there is no edge connecting two vertices in the Independent Set. Consider the following greedy algorithm to find a maximal independent set. Will this set always find a maximal independent set? If yes, provide a convincing explanation (less formal than a proof) of this. If not, either provide a convincing explanation that it won't or provide a counterexample.

```
sort the vertices based on number of incoming edges from low to high
mark all edges in the set as not covered
initialize S to be the empty set
```

```
for each vertex v {
  if none of its edges are covered {
    add v to S
    mark all of the edges incident on v as covered
  }
}

return S
```

The cost of this algorithm is $O(m)$ since each edge is considered twice, once for each endpoint. I conclude that this algorithm does not always find a maximal independent set. We know that the independent set problem is NP-complete. Therefore if this algorithm is in P, then $P = NP$, which is highly unlikely and certainly not proven this easily!