CS 170 **Second Midterm** **1 Nov 2011**

Name (1 pt): _____

SID (1 pt): _____

Section Number, e.g. 101 (1 pt): _____

Name of Neighbor to your left (1 pt): _____

Name of Neighbor to your right (1 pt): _____

Instructions: This is a closed book, closed calculator, closed computer, closed network, open brain exam, but you are permited a 1 page, double-sided set of notes, large enough to read without a magnifying glass.
You get one point each for filling in the 5 lines at the top of this page.
Write all your answers on this exam. If you need scratch paper, ask for it, write your name on each sheet, and attach it when you turn it in (we have a stapler).

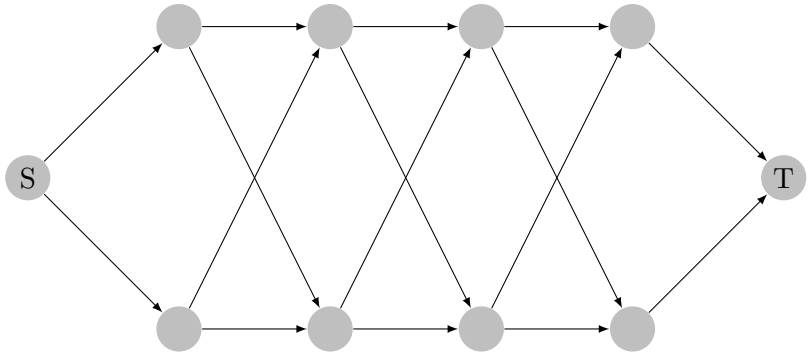
1	
2	
3	
4	
5	
Total	

1

Question 1: Counting Shortest Paths in a Weighted Directed Graph (20 points).

1. **(1 point)** What is the length of the shortest path from S to T in the graph below? Assume all edges have length 1.

2. **(2 points)** How many shortest paths are there from S to T in the graph below?



3. **(15 points)** Modify the algorithm shown below by filling in the boxes to compute the number of shortest paths numpaths[v] from the source vertex s to all other vertices v. Assume all edges have positive lengths.

```
for all u in V
    dist[u] = ∞
    numpaths[u] = [          ]
dist[s] = 0
numpaths[s] = [          ]
Q = makequeue(V,dist)
while Q not empty
    u = deletemin(Q)
    for all edges (u, v)
        if dist[v] > dist[u] + length(u, v) then
            dist[v] = dist[u] + length(u, v)
            decreasekey(Q,v)
            numpaths[v] = [          ]
        else if [          ]
            numpaths[v] = [          ]
        end if
```

Explain briefly why the code is correct:

In the first branch of the "if" statement in the inner loop, numpaths[v] is correctly updated because [fill in your answer below]:

In the second branch of the "if" statment in the inner loop, numpaths[v] is correctly updated because [fill in your answer below]:

4. **(2 points)** Assuming we use a binary heap for the priority queue, what is the complexity of your algorithm, in terms of $|E|$ and $|V|$ (in Big-Oh notation)?

Question 2: Huffman Codes (15 points).

1. **(3 points)** Over the alphabet $\{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5, \ell_6\}$, the i^{th} letter ℓ_i appears with frequency $\mathtt{a}[\ell_i]$, where $\mathtt{a}[\ell_1] = 3, \mathtt{a}[\ell_2] = 5, \mathtt{a}[\ell_3] = 7, \mathtt{a}[\ell_4] = 10, \mathtt{a}[\ell_5] = 13, \mathtt{a}[\ell_6] = 16$. If the Huffman algorithm is run on this input, let c_j be the j^{th} internal node created and let $\mathtt{a}[c_j]$ be its frequency computed by the algorithm for $1 \leq j \leq 5$, fill in the following table with the correct numbers.

node c_j	c_1	c_2	c_3	c_4	c_5
frequency $\mathtt{a}[c_j]$					

2. **(3 points)** Draw a tree representing the output of the Huffman algorithm, when executed on the input in Part 1. If an internal node c_j has left child v_L and right child v_R, always make sure $\mathtt{a}[v_L] \leq \mathtt{a}[v_R]$. Label each node in the tree with its frequency, i.e. the number $\mathtt{a}[\ell_i]$ or $\mathtt{a}[c_j]$.

3. **(9 points)** Assuming that the input frequencies to the Huffman algorithm are sorted: $\mathtt{a}[\ell_1] \leq \mathtt{a}[\ell_2] \leq \cdots \leq \mathtt{a}[\ell_n]$, with $\mathtt{a}[\ell_i] \geq 0$, design an implementation of the Huffman algorithm that runs in $O(n)$ time.

   State the key property *or an* invariant*, write a one-or-two-sentence* main idea *and short* pseudocode*. You do* not *need to analyze runtime or correctness.*

   Hint: In the usual Huffman algorithm, (the frequencies of) the internal nodes and the leaf nodes are all stored in a single data structure. For this problem, try not to mix the two kinds of nodes. Try to separately store leaf nodes in one data structure, and internal nodes in another (Part 1 may help here). What would be a good data structure to store each kind of nodes?

   Key Property/Invariant:

   Main Idea:

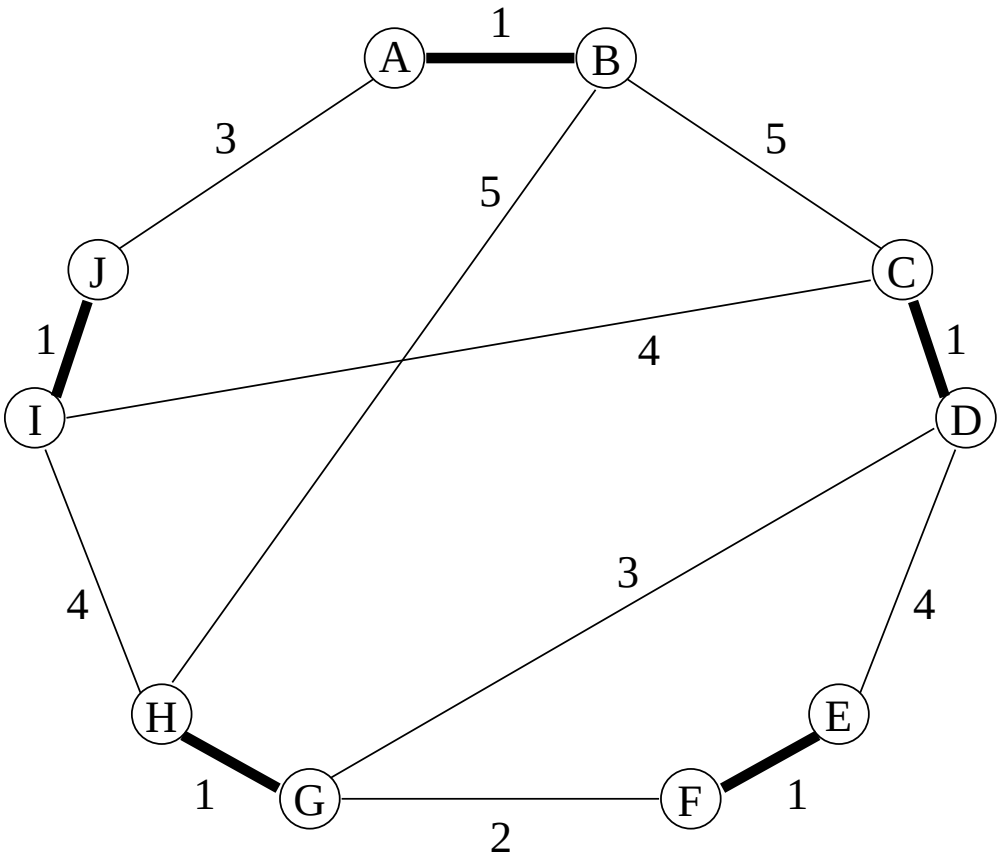
   Pseudocode:

Question 3: Minimum Spanning Trees (16 points).

Suppose we are constructing a minimum spanning tree, and we have already decided that the bold edges will go in the tree. Now, according to the Cut Property, which of the other edges could safely be added to the tree in the next step? (Note: We are not necessarily running Kruskal or Prim.)

In other words, for each of these edges, circle the name of the edge if the Cut Property guarantees that there exists a minimum spanning tree containing the edge and all the bold edges:

{B, C} {D, E} {F, G} {H, I} {J, A} {B, H} {C, I} {D, G}



Question 4: Dynamic Programming (17 points).

You own a supercomputer. On a given day, n people want to run jobs on your machine. However, each job takes a certain integer number of minutes, $t_1, \ldots, t_n$, for their job. Each person is willing to pay you a certain amount of money, $m_1, \ldots, m_n$. There are more requests per day than you can accomodate. You wish to maximize your total earnings. You do not get paid anything for a job until you finish it completely. We want to write an algorithm that outputs which jobs you should accept on a particular day. Assume no job takes longer than one day (1440 minutes) to complete.

1. **(2 points)** Louis Reasoner proposes to solve this problem as follows: sort jobs by the ratio of money to time, m_i/t_i. Greedily take as many jobs as you can with the highest ratio until you run out of time. Give a counterexample that shows that his algorithm is incorrect.

2. **(3 points)** We will now attempt to solve this problem with dynamic programming. We will use a two-dimensional array A. In plain English, what does $A[i, j]$ represent?

3. **(2 points)** How many rows and columns will A have?

4. **(2 points)** How should we initialize A?

5. **(4 points)** Give the recurrence for $A[i, j]$.

   $A[i, j] =$

6. **(1 point)** In what order can you calculate the entries of A?

7. **(2 points)** What is the asymptotic running time of this algorithm?

8. **(1 point)** Where in A will the final result be stored?

Question 5: True/False (18 points).

Statement	True	False
Given are two structurally identical graphs $G = (V, E, l)$ and $G' = (V, E, l')$ where l and l' are functions $E \to \mathbb{Z}$ that associate a length with each edge. For given integers $m > 0$ and b, we have that $l'(e) = ml(e) + b \ \ \forall e \in E$. Then each minimum spanning tree T of G is also a minimum spanning tree for G'.		
In Kruskal's algorithm using the data structure for disjoint sets with union by rank and path compression as discussed in lecture, every call of `find()` takes time $O(\log^*(n))$ where n is the number of vertices in the graph.		
Assume that the Bellman-Ford algorithm is applied to a DAG G. Then there exists an ordering for the updates such that the `dist` array contains the correct distances after one iteration of the main loop of the algorithm, i.e. after each edge has been relaxed once.		
Let `prev` be the array computed by the Bellman-Ford algorithm if applied to a strongly connected directed graph $G = (V, E)$ which has no negative cycles. Let $G' = (V', E')$ be the undirected graph where $V' = V$ and $E' = \{\{u, \texttt{prev}[u]\} \ \ \forall u \in V\}$. Then G' is a tree.		
If we use the Huffman encoding scheme to encode n symbols to binary codes, then none of the codes is longer than $O(\log n)$ bits.		
The edit distance between the two strings `TUESDAY` and `THURSDAY` is 3.		