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).

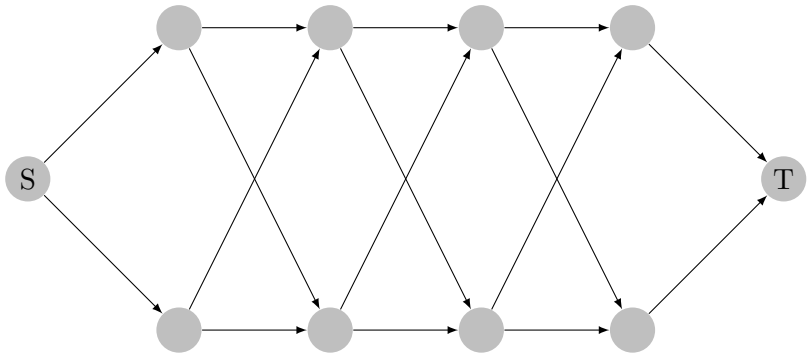
1	
2	
3	
4	
5	
Total	

1 , solutions

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. 5

2. **(2 points)** How many shortest paths are there from S to T in the graph below? 16



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] = 0
dist[s] = 0
numpaths[s] = 1
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] = numpaths[u]
        else if dist[v] = dist[u] + length(u, v)
            numpaths[v] = numpaths[v] + numpaths[u]
        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]:

The number of shortest paths to v found so far all lead through u, so numpaths[v] = numpaths[u].

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

The new shortest paths to v via u have been found, so we increase numpaths[v] by numpaths[u].

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)?

   $O\left((|E| + |V|)log(|V|)\right)$, the same as Dijkstra's algorithm.

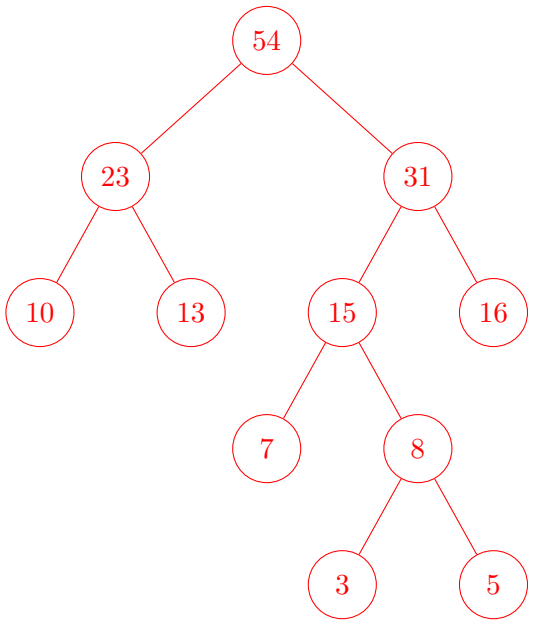
Question 2: Huffman Codes (15 points).

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 $\mathtt{a}[c_j]$	8	15	23	31	54

Grading: $0.5 \times$ # correct cells + [0.5 if at least one cell is correct].

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 $\mathtt{a}[c_j]$.



Grading: -0.5 if exactly one inversion ($a[v_L] > a[v_R]$), -1 if more than one inversion.

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: The internal nodes created later have higher frequencies, i.e. $\mathtt{a}[c_j] \leq \mathtt{a}[c_{j+1}]$ for $1 \leq j < n-1$. OR Maintain the invariant that the frequencies in each queue are in increasing order from head to tail.

Main Idea: Employ two queues, one for storing frequencies of letters, and one for storing frequencies of internal nodes created. Get the smallest element among the two queues using a merge-sort-like approach, i.e. compare and pop the smaller of the two.

Pseudocode:

Algorithm 1: Linear Time Huffman

Data: Frequencies $\mathtt{a}[\ell_1] \leq \mathtt{a}[\ell_2] \leq \cdots \leq \mathtt{a}[\ell_n]$ with $\mathtt{a}[\ell_i] \geq 0$.

Initialize an empty queue Q_l for leaf nodes, and an empty queue Q_v for internal nodes;
for $i := 1$ **to** n **do**
    Create a leaf node labeled ℓ_i with frequency $\mathtt{a}[\ell_i]$;
    Q_l.push(ℓ_i);
for $j := 1$ **to** $n-1$ **do**
    if $a[Q_l.head()] \leq a[Q_v.head()]$ **then** ($\triangleright$ treating $a[\cdot]$ for an empty queue as ∞)
        $v_L := Q_l$.pop() **else** $v_L := Q_v$.pop();
    if $a[Q_l.head()] \leq a[Q_v.head()]$ **then** ($\triangleright$ treating $a[\cdot]$ for an empty queue as ∞)
        $v_R := Q_l$.pop() **else** $v_R := Q_v$.pop();
    Create an internal node c_j with children v_L and v_R, and frequency $\mathtt{a}[c_j] := \mathtt{a}[v_L] + \mathtt{a}[v_R]$;
    Q_v.push(c_j);
Output the tree rooted at c_{n-1};

Grading:

- 2 points for explicitly stating the **s**ortedness as a key property/invariant;
- 2 points for using the correct data structure (FIFO/**Q**ueue) in code;
- 2 points for correctly **p**opping to merge smallest numbers from the two data structures; and
- 3 points for correct **i**mplementation: -1 if the number of iterations/termination condition of the final loop is incorrect, -1 if popping from an empty queue is not properly handled.

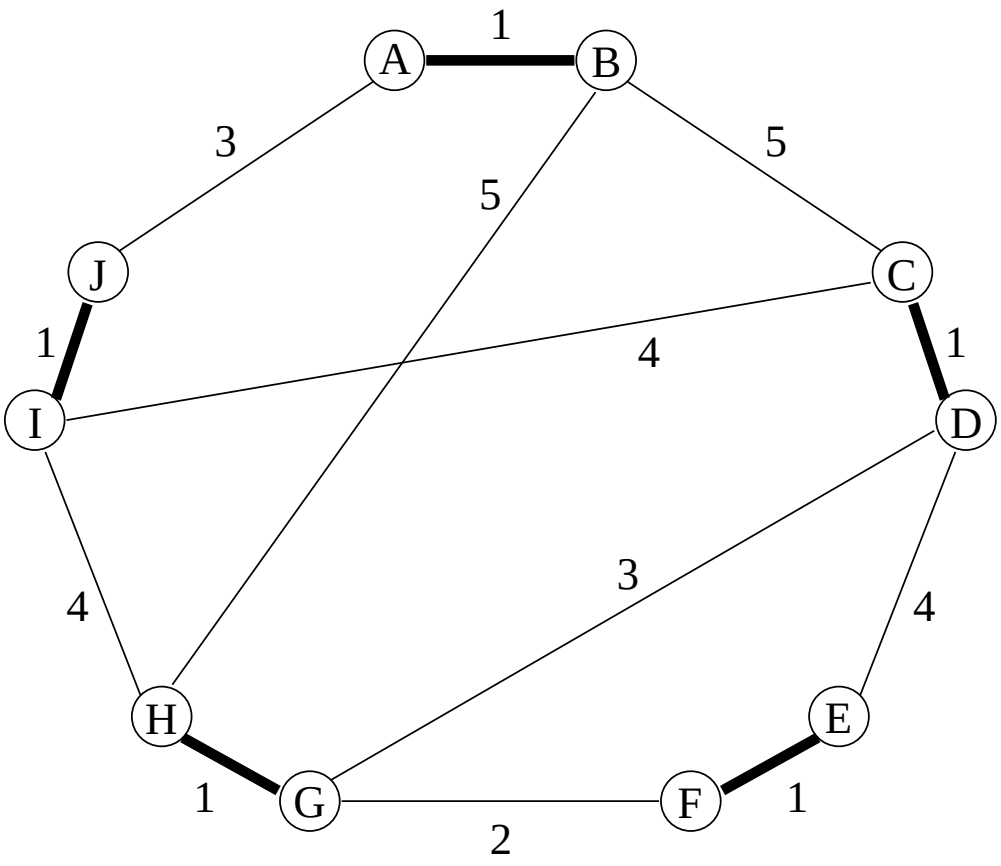
Question 3: Minimum Spanning Trees (16 points).

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:

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

Answer: $\{F, G\}$, $\{H, I\}$, $\{J, A\}$, $\{C, I\}$, $\{D, G\}$ are safe to include, and $\{B, C\}$, $\{D, E\}$, $\{B, H\}$ are not safe to include.



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.

   Job 1 takes 18 hours and pays \$100. Jobs two, three and four, take 8 hours each and pay \$40. Taking the three smaller jobs would earn you \$120 instead of \$100. *Grading: 1 point for general idea, 2 for concrete example.*

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?

   The entry $A[i, j]$ represents the maximum amount of money that can be earned in i minutes using jobs 1 through j. *Grading: one point for minutes in one dimension, one point for jobs in the other and one point for only jobs 1 thorugh j.*

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

   There will be 1441 rows and $n + 1$ columns. *Grading: All or nothing, off-by-one errors ignored.*

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

   Set all $A[\cdot, 0] = 0$. *Grading: Full credit for this or if both first row and column are initialized. One point if the whole table is initialized.*

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

   The recurrence is

$$A[i,j] = \begin{cases} A[i, j-1] & \text{if } t_j > i \\ \max\{A[i, j-1], A[i - t_j, j-1] + m_j\} & \text{otherwise} \end{cases}$$

   Grading: -2 points for not handling case where $i < m_j$, -2 for allowing repeats.

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

   In increasing order of i and j, either one first. *Grading: All-or-nothing.*

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

   $O(1440n) = O(n)$. *Grading: All or nothing, but credit given if consistent with parts 2, 3, and 5.*

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

   $A[1440, n]$. *Grading: All or nothing.*

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'. For any cut through the graph G, T contains a minimal edge. After modifying the edge weights using the affine transformation described above, this edge has still minimum weight on the cut. Hence it can also be selected as an edge for the minimum spanning tree of G'	X	
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. It is possible to apply `union()` arbitrarily often without ever using path compression. Then the depths of the trees are growing and a call to `find()` takes time $O(\log n)$. Note that $O(\log^*(n))$ is the amortized complexity. This, however, does not imply that every single call takes time $O(\log^*(n))$.		X
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. The updates must be executed in topological order from the source. Then, for each update, it is assured that the distance at the source of the corresponding edge has already been assigned correctly.	X	
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. For each node but the root, exactly one edge is added. The graph is connected. However, a connected graph with $\|V\|$ vertices and $\|V\| - 1$ edges is a tree.	X	
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. As a counterexample, consider an instance with n symbols with frequencies $f_n = f_{n-1} = 2^{-n+1}$ and $f_i = 2f_{i+1} \ \ \forall i \in [1, \cdots, n-2]$. Then the longest code has length $n - 1$.		X

The edit distance between the two strings `TUESDAY` and `THURSDAY` is 3.
The minimum edit distance is 2. To see this, consider the following alignment:
`T-UESDAY`
`THURSDAY`
		X