University of Washington
Department of Computer Science and Engineering
CSEP 521, Spring 2011

April 30, 2011

Midterm Exam, Wednesday, April 27, 2011

NAME: _____

Instructions:

- Closed book, closed notes, no calculators
- Time limit: 60 minutes
- Answer the problems on the exam paper.
- If you need extra space use the back of a page
- Problems are not of equal difficulty, if you get stuck on a problem, move on.

1	/10
3	/10
4	/10
5	/10
6	/10
Total	/50

Problem 1 (10 points):

Consider the stable matching algorithm. Show how a w can be matched with all of the m's during the course of the algorithm. Hint: give the preference lists and describe an execution of the algorithm where some w is matched with each m in turn.

Solution:

Let $M = \{m_1, \ldots, m_n\}$, and $W = \{w_1, \ldots, w_n\}$.
For each $m_i \in M$, suppose that w_1 is listed first in the preference lists. The remaining order of the m_i lists can be arbitrary.
Suppose that w_1 has the preference order $m_n, m_{n-1}, \ldots, m_1$.
If the m_i's propose in the order $m_1, \ldots, m_n$, they will each propose to w_1, be accepted, and then dumped in the next round.

Problem 2 (10 points):

Let $G = (V, E)$ be an undirected graph, and v a vertex.

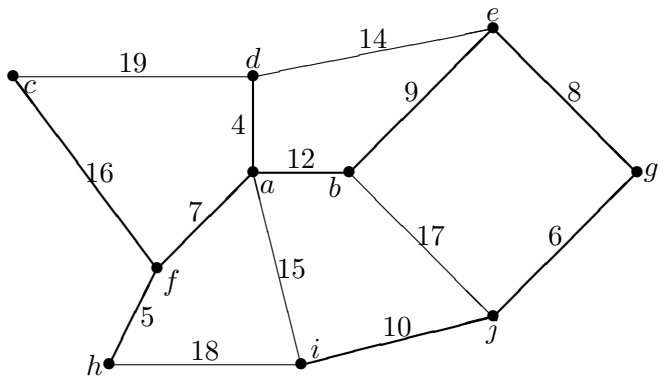
Give an $O(n + m)$ time algorithm that finds the shortest cycle in G that contains the vertext v.

Solution:

The initial idea is to construct a BFS tree from v and take use the first non-tree edge to complete a cycle. The problem is that this cycle does not necessarily include v.
To fix this problem, we label each vertex in the graph by the neighbor of v that it is descended from in the BFS tree. The first non-tree edge $e = (x, y)$ with distinct labels can then be used to complete a cycle. This cycle includes v, since the paths of tree edges from x and y go back to distinct neighbors of v.

Problem 3 (10 points):

Consider the following undirected graph G.



a) Highlight the edges of G that are in a minimum spanning tree.

Solution:

Edges: $(c, f), (f, a), (a, d), (a, b), (f, h), (b, e), (e, g), (g, j), (i, j)$.

b) Use the Edge Inclusion Lemma to argue that the edge (c, f) is in every Minimum Spanning Tree of G.

Solution:

Edge Inclusion Lemma: Suppose the edge costs are distinct. Let $e = (u, v)$ be the minimum cost edge between S and $V - S$. e is guaranteed to be in every minimum spanning tree.

(c, f) is the minimum cost edge between $\{c\}$ and $\{a, b, d, e, f, g, h, i, j\}$.

c) Use the Edge Exclusion Lemma to argue that the edge (b, j) is never in a Minimum Spanning Tree of G.

Solution:

Edge Exclusion Lemma: Suppose the edge costs are distinct. Let $e = (u, v)$ be the maximum cost edge on a cycle C. e is never in a minimun spanning tree.

(b, j) is the maximum cost edge on the cycle (b, e, g, j)

Problem 4 (10 points):

The binpacking problem is: Given a collection of items $I = \{i_1, \ldots, i_n\}$ where each item i_j has a s_j, an integer K, and a collection of bins $B = \{b_1, \ldots, b_m\}$ assign the items to bins such that the sum of the sizes of items assigned to each bin b_i is at most K. The goal is to minimize the number of bins that recieve items, i.e., to pack the items into as few bins as possible.

A greedy algorithm for the problem considers the items in order, and places each item in the first bin that has enough remaining space to hold the item.

Assume that the bin size is $K = 3$, and the items have sizes 1, 2, or 3.

a) Give an example that shows that the greedy algorithm does not necessary find the optimal solution (minimizes the number of bins).

Solution:

Consider items with size 1, 1, 2, and 2 respectively. If they are packed by the greedy algorithm in that order, the first bin gets two items of size 1, and the next two bins each get an item of size 2, so three bins are used all together. A better packing is to put an item of size 1 with an item of size 2 in each of the bins.

b) Describe an algorithm which find an optimal solution. Explain why your algorithm minimizes the number of bins used. (You do not need to give a formal proof - just identify the key ideas.)

Solution:

First sort the items in non-increasing order, so that items of size 3 come first, followed by items of size 2, and finally, items of size 1. Then place the items with the greedy algorithm, so there are bins with items of size 3, followed by bins of size 2. The items of size 1 are then paced on top of the items of size 2 to fill up those bins.

The key to getting an optimal packing in this case is to match up items of size 1 and items of size 2, so an alternate approach would be to pair up the items first to fill as many bins as possible.

Problem 5 (10 points):

Give solutions to the following recurrences. Justify your answers.

a)

$$T(n) = \begin{cases} T(\frac{9n}{11}) + n & \text{if } n > 1 \\ 1 & \text{if } n \leq 1 \end{cases}$$

Solution:

Unrolling the recurrence:

$$T(n) = T(\frac{9}{11}n) + n = T((\frac{9}{11})^2 n) + \frac{9}{11}n + n = T((\frac{9}{11})^3 n) + (\frac{9}{11})^2 n + \frac{9}{11}n + n.$$

Which becomes

$$T(n) = n \sum_{j=0}^{\log_{11/9} n} (\frac{9}{11})^j \leq n \sum_{j=0}^{\infty} (\frac{9}{11})^j = n \frac{1}{1 - 9/11} = n \frac{11}{2}.$$

Hence, $T(n) = O(n)$.

b)

$$T(n) = \begin{cases} 16T(\frac{n}{4}) + n^2 & \text{if } n > 1 \\ 1 & \text{if } n \leq 1 \end{cases}$$

Solution:

To solve this, we expand the recursion tree, and look at the work per level. Since we are dividing the problem size by 4 at each recursion, the depth is $\log_4 n$. The work on the first level in n^2. The work on the second level is $16(n/4)^2 = n^2$, and the work on the third level is $16^2(n/16)^2 = n^2$, and so on. Every level has n^2 work, so the total is $n^2 \log_4 n = O(n \log n)$.

Problem 6 (10 points):

Given an array of n real numbers, consider the problem of finding the maximum sum in any contiguous subvector of the input, for example, in the array

$$\{31, -41, 59, 26, -53, 58, 97, -93, -23, 84\}$$

the maximum is achieved by summing the third through seventh elements, where $59 + 26 + (-53) + 58 + 97 = 187$. When all numbers are positive, the entire array is the answer, while when all numbers are negative, the empty array maximizes the total at 0.

Give an $O(n \log n)$ divide and conquer algorithm for solving this problem.

Solution:

The basic idea of the divide an conquer algorithm is to split the array at the midpoint, and find the best solution in the left or right halves of the array. The recursion hits bottom when it gets to a single element, where it return the maximum of the element and 0. The key idea is to test whether there is a better solution than the left or right solutions by looking for a MSCS that spans the boundary between the left and right subproblem. We find the maximum solution by a linear scan in each direction.

Suppose the elements are $S_1, S_2, ..., S_n$.

The key subroutines for this algorithm are *MaxSumStarting*(i, j) which finds the maximum sum $S_i + S_{i+1} + \cdots + S_k$ for $i \leq k \leq j$ and *MaxSumEnding*(i, j), which finds the maximum sum $S_k + S_{k+1} + \cdots + S_j$ for $i \leq k \leq j$. These two routines can be implemented by iterative algorithms which run in $O(j - i)$ time.

The overall recursive algorithm is:

```
MSCS(i, j)
  if i = j;
    return Max(S_i, 0);
  m := (i + j)/2;
  left := MSCS(i, m);
  right := MSCS(m + 1, j);
  leftSpan := Max(MaxSumEnding(i, m), 0);
  rightSpan := Max(MaxSumStarting(m + 1, j), 0;
  return Max(left, right, leftSpan + rightSpan);
```