University of Washington
Department of Computer Science and Engineering
CSE 421, Fall 2006

November 26, 2006

Midterm Solutions, November 2006

NAME: ____________________________

Instructions:

- Closed book, closed notes, no calculators
- Time limit: 50 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.
- You may write in either English or Chinese.

1	/10
2	/10
3	/10
4	/10
5	/10
6	/10
7	/10
Total	/70

Problem 1 (10 points):

Consider the stable matching problem.

a) Show that it is possible to have a *last-choice* match: There exists an instance of the problem with a stable matching M that has m matched with w, where w is m's last choice, and m is w's last choice.

Answer: An example of a problem instance is a 2×2 example with:

$$\begin{array}{ll} m_1 : w_1, w_2 & \quad w_1 : m_1, m_2 \\ m_2 : w_1, w_2 & \quad w_2 : m_1, m_2 \end{array}$$

Since m_1 and w_1 are each other's first choice, they are matched, leaving m_2 and w_2 to be matched.

Another example is the trivial example, with just m and w. In this case, m and w are matched, and are their last choices (as well as their first choices).

b) Is it possible for a stable matching to have two *last-choice* matches: could a stable matching M have m_1 matched with w_1 where m_1 is w_1's last choice and w_1 is m_1's last choice, and m_2 matched with w_2 where m_2 is w_2's last choice and w_2 is m_2's last choice? Justify your answer.

Answer: No. If there are two last choice matches (m_1, w_1) and (m_2, w_2), then (m_1, w_2) is an instability, since m_1 preferes w_2 to w_1 and w_2 prefers m_1 to m_2.

Problem 2 (10 points):

Show that

$$\sum_{k=0}^{\log n} 4^k$$

is $O(n^2)$.

Answer:

$$\sum_{k=0}^{j} x^k = \frac{x^{j+1} - 1}{x - 1},$$

so

$$\sum_{k=0}^{\log n} 4^k = \frac{4^{\log n + 1} - 1}{4 - 1} = \frac{4n^2 - 1}{3}$$

which is $O(n^2)$.

Problem 3 (10 points):

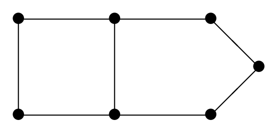
Let $G = (V, E)$ be an undirected graph.

a) True or false: If G is a tree, then G is bipartite. Justify your answer.

   True. If we label the vertices based upon their distance from the root, we observe that all edges go between even vertices and odd vertices.

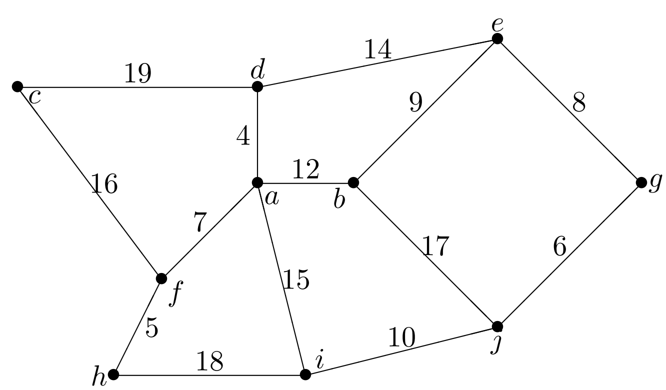
b) True or false: If G is not bipartite, then the shortest cycle in G has odd length. Justify your answer.

   False. A counter example is a graph made up of a cycle of length 4 connected to a cycle of length 5.



Problem 4 (10 points):

Consider the following undirected graph G.



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

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

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

   Answer: (a, i) is the most expensive edge on the cycle $\{a, i, j, g, e, b\}$.

Problem 5 (10 points):

The knapsack problem is: Given a collection of items $I = \{i_1, \ldots, i_n\}$ and an integer K where each item i_j has a weight w_j and a value v_j find a subset of the items which has weight at most K and maximizes the total value in the set. More formally, we want to find a subset $S \subseteq I$ such that $\sum_{i_k \in S} w_k \leq K$ and $\sum_{i_k \in S} v_k$ is as large as possible.
Suppose that the items are sorted in decreasing order of value, so that $v_i \geq v_{i+1}$. A simple greedy algorithm for the problem is:

$CurrWeight := 0;$
$Sack := \emptyset;$
for $j := 1$ **to** n
  if $CurrWeight + w_j \leq K$ **then**
    $Sack := Sack \cup \{i_j\}$
    $CurrWeight := CurrWeight + w_j$

a) Show that the greedy algorithm does not necessarily find the maximum value collection of items that can be placed in the knapsack.

   Answer: The following counter example shows that the greedy algorithm does not find the optimal soultion. Let $K = 2$ and suppose there are three jobs $\{i_1, i_2, i_3\}$ with $v_1 = 3$, $w_1 = 2$, $v_2 = 2$, $w_2 = 1$, and $v_3 = 2$, $w_3 = 1$. The greedy algorithm selects i_1, while the optimal solution is i_2 and i_3.

b) Prove that if all weights are the same, then the greedy algorithm finds the maximum value set. (For convenience, you may assume that each item has weight 1).

   Proof: If there are fewer than K items, then the greedy algorithm selects all items, so assume there are at least K items. The greedy algorithm constructs the solution $\{i_1, \ldots, i_K\}$. Let $Opt = \{i_{j_1}, \ldots, i_{j_K}\}$ (where $j_r < j_{r+1}$). We must have $r \leq j_r$, so $v_r \leq v_{j_r}$ for all r, so the value of the set constructed by the greedy algorithm is no more the the optimal.

Problem 6 (10 points):

Give solutions to the following recurrences. Justify your answers.

a)

$$T(n) = \begin{cases} 2T(\frac{n}{3}) + n & \text{if } n > 1 \\ 1 & \text{if } n \leq 1 \end{cases}$$

Answer: Unrolling the recurrence, we get:

$$T(n) = 2T\left(\frac{n}{3}\right) + n = 4T\left(\frac{n}{9}\right) + \frac{2n}{3} + n = 8T\left(\frac{n}{27}\right) + \frac{4n}{9} + \frac{2n}{3} + n,$$

which gives us:

$$T(n) = \sum_{i=0}^{\log_3 n} \left(\frac{2}{3}\right)^i n \leq 3n,$$

so the solution is $O(n)$.

b)

$$T(n) = \begin{cases} 8T(\frac{n}{2}) + n^3 & \text{if } n > 1 \\ 0 & \text{if } n \leq 1 \end{cases}$$

Answer: Unrolling the recurrence, we get:

$$T(n) = 8T\left(\frac{n}{2}\right) + n^3 = 64T\left(\frac{n}{4}\right) + n^3 + n^3 = 512T\left(\frac{n}{8}\right) + n^3 + n^3 + n^3.$$

We observe that each level of the recurrence yields the n^3, and the depth of the recurrence is $\log 2n$, so the answer is $O(n^3 \log n)$.

Problem 7 (10 points):

A k-wise merge takes as input k sorted arrays, and constructs a single sorted array containing all of the elements of the input arrays.

a) Describe an efficient divide and conquer algorithm $MultiMerge(k, A_1, \ldots, A_k)$ which computes a k-wise merge of its input arrays.

Answer: We give a recursive algorithm, which makes use of a routine $Merge(A_1, A_2)$ which merges a pair of sorted arrays, and returns the result. We assume that k is a power of two, and that $k \geq 2$.

$MultiMerge(k, A_1, \ldots, A_k)$
  if $k = 2$
    return $Merge(A_1, A_2)$;
  else
    $B_1 := MultiMerge(\frac{k}{2}, A_1, \ldots, A_{\frac{k}{2}})$;
    $B_1 := MultiMerge(\frac{k}{2}, A_{\frac{k}{2}+1}, \ldots, A_k)$;
    return $Merge(B_1, B_2)$;

b) What is the run time of your algorithm with input of k arrays of length n. Justify your answer.

The run time of the algorithm is $O(kn \log k)$. One way to see this is to write the run time as a recurrence. Let cn be a bound on the cost of merging two arrays of length n. The recurrence for the run time is $T(k) = 2T\left(\frac{k}{2}\right) + ckn$, so the solution is $ckn \log k$.