CS 570
Analysis of Algorithms
Spring 2008
Final Exam Solutions

Kenny Daniel (kfdaniel@usc.edu)

Question 1

[FALSE] In a flow network whose edges have capacity 1, the maximum flow always corresponds to the maximum degree of a vertex in the network.

[FALSE] If all edge capacities of a flow network are unique, then the min cut is also unique.

[TRUE] A minimum weight edge in a graph G must be in one minimum spanning tree of G.

[TRUE] When the size of the input grows, any polynomial algorithm will eventually become more efficient than any exponential one.

[FALSE] NP is the class of problems that are not solvable in polynomial time.

[FALSE] If a problem is not solvable in polynomial time, it is in the NP-Complete class.

[TRUE] Linear programming can be solved in polynomial time.

[FALSE] $10^{2\log 4n+3} + 9^{2\log 3n+21}$ is $O(n)$.

[FALSE] $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.

[FALSE] If X can be reduced in polynomial time to Y and Z can be reduced in polynomial time to Y, then X can be reduced in polynomial time to Z.

Question 2

Suppose you are given a number x and an array A with n entries, each being a distinct number. Also it is known that the sequence of values $A[1], A[2], \ldots, A[n]$ is unimodal. In other words for some unknown index p between 1 and n, we have $A[1] < \ldots < A[p-1] < A[p] > A[p+1] > \ldots > A[n]$. (Note that $A[p]$ holds the peak value in the array).

Give a algorithm with running time $O(\log n)$ to determine if x belongs to A, if yes the algorithm should return the index j such that $A[j] = x$. You should justify both the correctness of your algorithm and the running time.

Solution: Since the array is unimodal, if we know where the maximum value is in the array, we can split the array in two and perform a binary search on each half (since each half is sorted in increasing and decreasing order respectively). So the real problem is finding the maximum value in time $O(\log n)$.

To find the maximum value, we use a divide and conquer algorithm which cuts the size of the search in 2/3 each time. We compare the 1/3 median and 2/3 median, and discard the bottom 1/3 or top 1/3 depending which is larger (convince yourself why this works!).

```
int find_max(start, end) {
    length := end - start;
    m1 := start + length / 3;
    m2 := start + length * 2 / 3;
    if(length = 0)
       return start;
    if(A[m1] < A[m2])
       return find_max(m1, end);
    else
       return find_max(start, m2);
}
```

Question 3

You are given two sequences $a[1], ..., a[m]$ and $b[1], ..., b[n]$. You need to find their longest common subsequence; that is, find a subsequence $a[i_1], ..., a[i_k]$ and $b[j_1], ..., b[j_k]$, such that $a[i_1] = b[j_1], \ldots, a[i_k] = b[j_k]$ with k as large as possible. You need to show the running time of your algorithm.

Solution: First note that this problem is actually to find the longest common substring (not subsequence, since they must be contiguous). That being said, we solving this problem using dynamic programming. We recursively build up the solution from the following subproblem: Let $LCSuffix(i, j)$ be the length of the longest common suffix of the strings $a[1], ..., a[i]$ and $b[1], ..., b[j]$.

$$LCSuffix(i,j) = \begin{cases} LCSuffix(i-1, j-1) + 1 & \text{if } a[i] = b[j] \\ 0 & \text{otherwise} \end{cases}$$

Constructing the above table for $1 \leq i \leq m$ and $1 \leq j \leq n$ takes time $O(mn)$. Then we find the longest common substring by finding the maximal value in the table:

$LCSubstring = \max_{1 \leq i \leq m, 1 \leq j \leq n} LCSuffix(i, j)$

Question 4

In Linear Programming, variables are allowed to be real numbers. Consider that you are restricting variables to be only integers, keeping everything else the same. This is called Integer Programming. Integer Programming is nothing but a Linear Programming with the added constraint that variables be integers. Prove that integer programming is NP-Hard

Solution: Integer Programming is in NP (since it is easy to check solutions), and is in NP-hard by reduction from Independent Set. Suppose we are given an instance of the Independent Set problem, consisting of a graph G such that we want to find the largest set S of vertices such that no two vertices share an edge.

We can express this as an Integer Program as follows: Each vertex v is represented by a variable x_v which is either 0 or 1. For each edge (u, v), we add one constraint such that $x_u + x_v \leq 1$. Then we maximize the sum: $\max \sum_{v \in S} x_v$. It can then be shown that a solution to this Integer Program gives a solution to the Independent Set problem, and so Integer Programming must be NP-hard.

Question 5

A carpenter makes tables and bookshelves, and he wants to determine how many tables and bookshelves he should make each week for a maximum profit. The carpenter knows that a net profit for each table is $25 and a net profit for each bookshelf is $30. The wooden material available each week is 690 units, its working hours are 120 hours per week. The estimated wood and working hours for making a table are 20 units and 5 hours respectively, while for making a bookshelf are 30 units and 4 hours. Formulate the problem using any technique we have covered in class. You do not need to solve it numerically.

Solution: This problem can be solved using Integer Programming. We define two variables t for the number of tables made, and b for the number of bookshelves. The maximum wood available is 690, so we have the constraint:

$$20 \cdot t + 30 \cdot b \leq 690$$

The maximum working hours is 120 hours, so we have the constraint:

$$5 \cdot t + 4 \cdot b \leq 120$$

Then the carpenter wishes to maximize profit:

$$\max 25 \cdot t + 30 \cdot b$$

Question 6

A variation of the satisfiability problem is the MIN 2-SAT problem. The goal in the MIN 2-SAT problem is to find a truth assignment that minimizes the number of satisfied clauses. Give the best approximation algorithm that you can find for the problem.