SOLUTION

CSE101 Final (Version B) March 18, 2014

Name: _____ Student ID: _____

Short Answer Questions (Total time: 75 minutes)

1. *(3 points)* The following procedure is invoked on an array A of size n:

```
Procedure Proc(A):
n ← |A|
Proc(A[1 ... n/3])
Proc(A[2n/3 ... n])
sum ← 0
for i = 1 to n:
    sum = sum + A(i)
print(sum)
```

Write down a recurrence relation for the running time of this function call. Solve the recurrence and state the big-O running time in terms of n.

$T(n) = 2T(n/3) + n$ $a=2, b=3, d=1$

By Master Theorem

$T(n) = O(n)$

2. *(1 point)* **True or False:** An $O(n^2)$ algorithm is faster than an $O(n^3)$ algorithm on any input.

FALSE

3. *(2 points)* If a DQ algorithm has $O(n^2 \log n)$ time complexity, and we can determine this using the Master Theorem, what combinations of values of a, b, d are possible for this algorithm?

$\log_b a = d$ (since there is log term) Also $d = 2$

$\Rightarrow a = b^2$ Any tuple of form $(b^2, b, 2)$ works

4. *(1 point)* A universal sink is a vertex that has in-degree $|V| - 1$. **True or False:** Finding a universal sink of G, if one exists, requires $\Omega(|V|^2)$ time because all of the input must be examined to give a correct answer, and there are $|V|^2$ entries in the adjacency matrix.

FALSE

5. *(1 point)* DFS is executed on a directed graph $G = (V, E)$ to find *pre* and *post* numbers for all vertices. For an edge $(u, v) \in E$, $pre(v) = 5$, $post(v) = 8$, $pre(u) = 10$ and $post(u) = 15$. The edge (u, v) is a

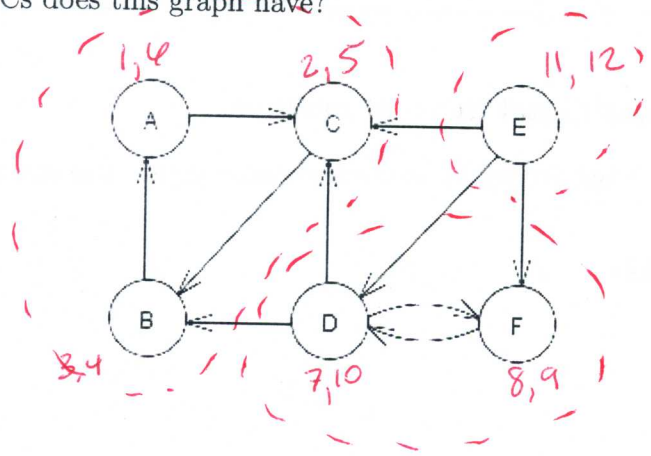
   (a) Forward Edge

   (b) Tree Edge

   (c) Cross Edge ✓

   (d) Back Edge

6. Perform DFS on this graph, starting from node A, breaking all ties lexicographically.
(2 points) Write down `pre` and `post` numbers for all nodes of the graph.
(1 point) How many SCCs does this graph have?



3 SCCs

7. *(4 points)* Match the items in the left column to appropriate choices in the right column.

4 1. Placing N Non-Attacking Queens on a $N \times N$ chessboard
6 2. Simulated Annealing
2 3. Matrix Chain Multiplication
1 4. Fractional Knapsack Problem

1. Greedy
2. $O(n^3)$ Dynamic Programming
3. Branch and Bound
4. Backtrack
5. $O(n^2)$ Dynamic Programming
6. Metaheuristic suitable for single-processor implementation
7. Metaheuristic suitable for multi-processor implementation

8. *(1 point)* **True or False: True or False:** For any flow network $G = (V, E)$ with source $s \in V$ and sink $t \in V$, there is always a **unique** minimum-capacity cut (L, R) (where $L \cap R = \emptyset$ and $L \cup R = V$) with $s \in L$ and $t \in R$.

F

9. *(2 points)* Explain why the Ford-Fulkerson algorithm always terminates when executed on any flow network with integer-valued edge capacities.

How increases by atleast one unit at each iteration (when augmenting path is found)

10. *(2 points)* While solving a linear program with a minimization objective, you find a feasible solution that is a local minimum. Is this solution guaranteed to be an optimal solution (yes or no)? Explain your answer.

Yes. Linear objective is convex ⇒ local optimum (over a convex domain) is a global optimum.

a) Algorithm:

1. Sort the given set of technologies by low frequency, l_i.
2. While [L, H] is not covered, repeat:
   Proceed along the list of intervals, and find the technology T_i with interval $[l_i, h_i]$ such that $l_i \leq L$ and h_i is the maximum of all the intervals with $l_i \leq L$.
   Select technology T_i, and discard all other technologies T_j with $l_j \leq l_i$.

So in words, the technology is selected based on two conditions: that it is able to cover a frequency sub-interval, which starts at or before the end of the frequency interval of the previous chosen technology, and whose frequency sub-interval end is the farthest along the spectrum.

b) Proof:

Consider the set S of all the intervals of the technologies chosen by the greedy solution, and let OPT be an optimal solution, with all intervals also ordered in increasing order of l_i. Consider the first interval in both solutions. Replacing the first interval in OPT, (l_{OPT}, h_{OPT}) with the first interval from S, (l_S, h_S), which has the farthest right endpoint, h_S and whose l_S is <= L, we get a possibly uncovered subinterval from h_S of length $h_S - h_{OPT}$, which is 0 if $h_S = h_{OPT}$, as we know that the greedy solution chose the interval with the largest h_i value. Thus, replacing the interval in the optimal solution with the interval from S either does not change the optimal solution, in which case S performs as well as OPT, or is a better solution, which is contradictory to the definition of OPT. Similarly extending this reasoning to any interval [a, H] from any point a, to the end of the spectrum, we conclude that the greedy algorithm is an optimal solution.

c) Time complexity:

Sorting the list takes O(nlog(n)) time.

While searching along the list, each technology is considered only once before it is either chosen or discarded. Hence, this step takes

O(n).

Therefore, the overall time complexity is O(nlog(n)), dominated by sorting.

2. **Dynamic Programming: Longest Increasing Subsequence**

Given a sequence of numbers $a_1, a_2, \ldots, a_n$, the Longest Increasing Subsequence (LIS) problem seeks the length k of the longest subsequence whose elements have monotonically increasing values. I.e., $a_{i_1} \leq a_{i_2} \leq \ldots \leq a_{i_k}$ with $1 \leq i_1 < i_2 < \ldots < i_k \leq n$. The LIS is not necessarily consecutive in the given sequence.

For example, given the sequence

$$\{ 10, 22, -9, 33, -71, 50, 41, 60, 80 \},$$

the length of the LIS is 6 (the LIS is { 10, 22, 33, 50, 60, 80 }).

Give a dynamic programming algorithm to find the longest increasing subsequence for a sequence of numbers $a_1, a_2, \ldots, a_n$.

(a) *(2 points)* Define your subproblems and state the solution to which subproblem will provide the desired answer.

Subproblem : LIS(i) = length of longest subsequence that ends at index i (chosen element a_i)

At the end, We scan through solutions of subproblems to find which has longest length which is the required answer

(b) *(3 points)* Give base case(s) and the recurrence for your DP algorithm.

Base Case : $LIS(1) = 1$

Recurrence : $LIS(i+1) = 1 + \max_{\substack{1 \leq j \leq i \\ a_j \leq a_i}} \{ LIS(j) \}$

(c) *(1 point)* Analyze the time complexity of your algorithm and give the time complexity in big-O notation.

There are n subproblems. Each subproblem requires $O(n)$ time. Final scan to find max length is $O(n)$ time. Total time is $O(n^2)$

(d) *(2 points)* For the same sequence of numbers $a_1, a_2, \ldots, a_n$, now find the length of the LIS $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ with the **additional constraint** that consecutive numbers in the LIS cannot be consecutive numbers in the original sequence. For example, given the sequence

{ 10, 22, -9, 33, -71, 50, 41, 60, 80 },

the length of the LIS without consecutive numbers is 4 (one LIS without consecutive numbers is { 10, 33, 50, 80 }).

Give a DP recurrence to solve this **LISWC** (LIS-without-consecutive) problem.

Only the Sub problem is same as before (with LISWC instead of LIS)

$LISWC(1) = 1$, $LISWC(2) = 1$

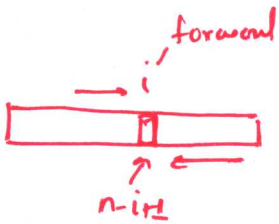
$LISWC(i+1) = \max_{1 \le j < i,\ a_j \le a_{i+1}} \{ 1 + LISWC(j) \}$, ← Strictly less than i

(e) *(2 points)* A sequence $b_1, \ldots, b_k$ is *bitonic* if there is an i, $1 \le i \le k$, such that the sequence is monotonically increasing up to b_i and then it is monotonically decreasing, that is, $b_1 \le b_2 \le \ldots \le b_i$ and $b_i \ge b_{i+1} \ge \ldots b_k$. If $i = 1$, the sequence degenerates into a monotonically decreasing sequence. If $i = k$, the sequence is a monotonically increasing one. In other words, monotonically increasing or decreasing sequences are also bitonic. For example, given the sequence

{ 10, 22, -9, 33, -71, 50, 41, 30, 80 },

the length of the LIS without consecutive numbers is 5 (one longest bitonic sequence is { 10, 22, 33, 50, 41, 30 }).

Assuming that you already have an algorithm to find the length of the LIS, explain in words how to find the length of the longest bitonic subsequence of a given sequence.



The bitonic sequence is a concatenation of an increasing sequence and a decreasing sequence. We can reverse given array and find LIS from that position and return sum of the lengths as the length of longest bitonic sequence with peak at index i

Then $BIT(i) = LIS(i) + LIS_R(n-i+1) - 1$

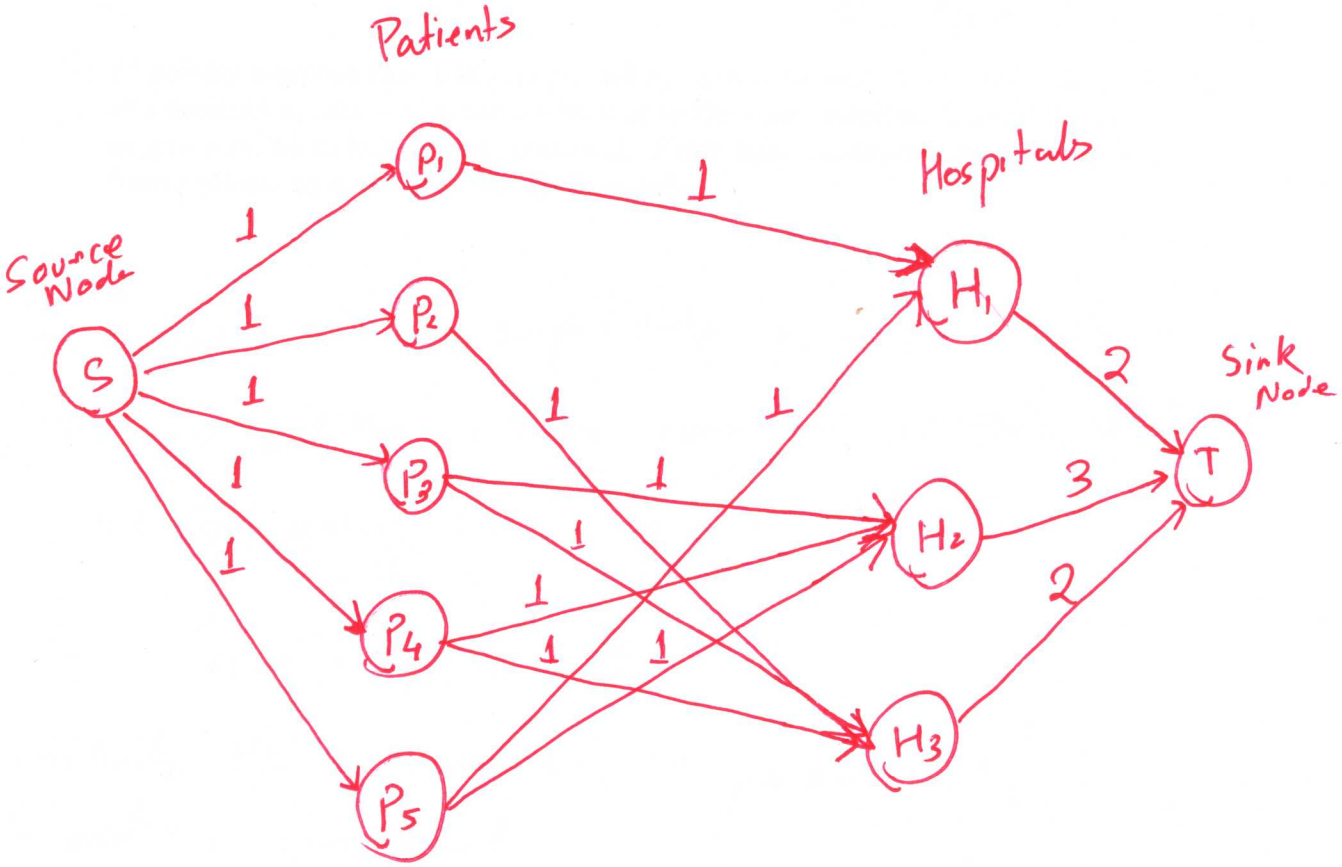
Scan through BIT(i) to find longest subsequence

3. **Network Flow and Linear Programming: Patients and Hospitals**

Five patients $p_1, \ldots, p_5$ need to be rushed to hospitals. Three hospitals, H_1, H_2 and H_3, are available, but they have capacities of two, three and two patients respectively. Furthermore, each patient p_i must be taken to a hospital that accepts his/her insurance policy. The table below shows the hospitals that accept the insurance policy of each patient.

p_1	p_2	p_3	p_4	p_5
H_1	H_3	H_3, H_2	H_2, H_3	H_2, H_1

(a) *(4 points)* Your task is to send all patients to hospitals that accept their insurance policies by reducing this to the problem of finding a maximum s-t flow in a suitable flow network. Draw this flow network, including all edge capacities.



(b) *(3 points)* Formulate the above problem as a linear program. Write down the objective function and all constraints.

Objective function:

Maximize: $f_{s,p_1} + f_{s,p_2} + f_{s,p_3} + f_{s,p_4} + f_{s,p_5}$

Conservation Constraint:

p_1 : $f_{s,p_1} = f_{p_1,H_1}$

p_2 : $f_{s,p_2} = f_{p_2,H_3}$

p_3 : $f_{s,p_3} = f_{p_3,H_2} + f_{p_3,H_3}$

p_4 : $f_{s,p_4} = f_{p_4,H_2} + f_{p_4,H_3}$

p_5 : $f_{s,p_5} = f_{p_5,H_2} + f_{p_5,H_1}$

Capacity constraint

$0 \le f_{H_1,T} \le 2$

$0 \le f_{H_2,T} \le 3$

$0 \le f_{H_3,T} \le 2$

All else between 0 and 1

i.e. for all other edges e

$0 \le f_e \le 1$

(c) *(3 points)* Suppose that patients p_2 and p_3 cannot be sent to the same hospital, and patients p_4 and p_5 also cannot be sent to the same hospital. Extend the linear program in (b) to handle this constraint. *Hint:* Add constraints on outgoing flow from patients to common available hospitals.

The hospital common to patients p_2 & p_3 is H_3

Thus we put the following constraint on flow from p_2 & p_3 going into H_3

$$f_{p_2,H_3} + f_{p_3,H_3} \le 1$$

Similarly H_2 is common to p_4 and p_5

We add the constraint

$$f_{p_5,H_2} + f_{p_4,H_2} \le 1$$