CSE 441/541 Midterm 1, 2015

Your Name: SOLUTIONS

Short Answer

1. [T or F] (2 points) In all cases, using a top-down approach when using dynamic programming to solve a problem is slower than using a bottom-up approach since the former would usually involve the overhead of recursively calls (setting up the stack and local variables).

2. In the 0-1 knapsack problem, the input is a list of items with integer weights \( w_1, w_2, \ldots, w_n \) and values \( v_1, v_2, \ldots, v_n \), and an integer capacity K. In class we gave an algorithm with running time \( O(nK) \), and we called this a pseudo-polynomial time algorithm.

(a) (2 points) In a few sentences, explain in general how pseudo-polynomial differs from a polynomial time algorithm.

Pseudo-polynomial algorithms have a run time that is polynomial in the size of the input + the value of one or more inputs. For example an \( O(nK) \) alg. for knapsack is exponential in the size of the input if \( K \) is \( 2^n \).

(b) (2 points) Suppose you consider the special case where the weight of every item is guaranteed to be less than 10n. Would the algorithm then complete in polynomial time? Why or why not?

Yes. If all items have weight 10n, then the max weight collectively of all items is \( n \cdot 10n \) or 10 \( n^2 \). So algorithm only needs to consider knapsacks of size \( \leq 10n^2 \) so overall alg. is polynomial.

3. (2 points) Recall that in the interval scheduling problem, you are given n requests, each having a start time \( s_i \) and finish time \( f_i \) and the objective is to schedule the maximum number of nonconflicting tasks. Which of the following greedy strategies is optimal? (List all that apply. No explanation needed.)

(a) Earliest start time first (select in increasing order of \( s_i \))
(b) Earliest finish time first (select in increasing order of \( f_i \))
(c) Latest start time first (select in decreasing order of \( s_i \))
(d) Latest finish time first (select in decreasing order of \( f_i \))
(e) Shortest activity first (select in increasing order of \( f_i s_i \))
(f) Lowest conflict first (select the activity that has the minimum number of conflicts with the remaining tasks)
Algorithms

4. Binary search is a common search algorithm that is used extensively in computer science which allows us to find a value in a sorted array in \( \Theta(\log n) \) time. However, if we no longer use an array to store the data, this algorithm may be useless. For this question, you are required to write an algorithm to find a value in an \( n \times n \) matrix (2D-array). You are guaranteed that each row and column of the matrix is sorted.

(a) (5 points) Write an \( O(n) \) time algorithm to find a given value in this matrix.

(b) (5 points, only required for CSE 541) Prove that your algorithm gives the correct answer.

Alg:

\[ \text{Call } \text{Find}_v( M[1..N, 1..N], v) \]

\[ \text{Find}_v( M[a..b], v) \]

\[ \text{Find}_v( M[am, .. amax, bm, .. bmax], v) \]

\[ a_{mid} = \left\lfloor \frac{a_{min} + a_{max}}{2} \right\rfloor \]

\[ b_{mid} = \left\lfloor \frac{b_{min} + b_{max}}{2} \right\rfloor \]

if \( v = M(a_{mid}, b_{mid}) \)

return "Found at \( a_{mid}, b_{mid} \)"

if \( v < M(a_{mid}, b_{mid}) \)

\[ \text{Find}_v( M[1..a_{mid}, 1..b_{mid}], v) \]

\[ \text{Find}_v( M[1..a_{mid}, bm+1..N], v) \]

\[ \text{Find}_v( M[a_{mid}+1..N, 1..b_{mid}], v) \]

else

\[ \text{Find}_v( M[a_{mid} .. amax], v) \]

\[ \text{Find}_v( M[a_{mid} .. amax, b_{mid} .. bmax], v) \]

\[ \text{Find}_v( M[1..amid, b_{mid} .. bmax], v) \]

\[ \text{Find}_v( M[a_{mid} .. amax, 1..b_{mid}] \)

end.

Time \( T(n) = 1 + 3T(n/4) \)

because I eliminate const. fraction \( (1/4) \) each time,

the run-time is linear.

\[ \text{IDEA:} \quad \begin{array}{c}
\text{if } v < M(a_{mid}, b_{mid}) \\
\text{then } v \text{ cannot be in lower right quadrant} \\
\text{so check others.}
\end{array} \]

\[ \begin{array}{c}
\text{if } v > M(a_{mid}, b_{mid}) \\
\text{v cannot be in upper left quadrant,} \\
\text{so check others.}
\end{array} \]

\[ \text{PROOF:} \quad \text{Suppose alg. does not find a value } v \text{ when it exists in } M. \quad \text{That is, alg. fails.} \]

Because we check all quadrants except 1, that value \( v \) must be in 1 quadrant.

But that quadrant if \( v < M(a_{mid}, b_{mid}) \) we ignore. Ignore only bottom right quadrant which has all values > \( v \). [case for \( v > M(a_{mid}, b_{mid}) \) is similar.]

... deduces that \( v \) is in that quadrant. \( Q.E.D. \)
5. You are given a sequence $x_1, \ldots, x_n$ of integers, and asked to find the length of the longest increasing subsequence. For example, if your input was

$$(1, 20, 2, 19, 3, 18, 4, 17)$$

then increasing subsequences include $(1, 19)$, $(2, 3, 4)$, but you should find and return "5", the length of longest increasing subsequence: $(1, 2, 3, 4, 17)$

(10 points) Give a dynamic programming approach to this problem, analyze its run time, and prove that it is correct.

**ALG**

**IDEA:** MAINTAIN ARRAY $LIS[i]$ keeping longest sequence ending at element $i$

$\text{LIS}[i] = 1$

For $i = 2 \ldots n$

$\text{LIS}[i] = \max \left\{ 1, \text{LIS}[j] + 1 : \forall j < i, x_j < x_i, \text{ and } j < i \right\}$

End.

Find max value of $\text{LIS}[i]$ for all $i = 1 \ldots n$.

**RUNTIME:** $n^2$ — loop over all items, do a max of $\leq n$ choices. One more $O(n)$ loop to find max doesn't affect $n^2$ runtime.

**PROOF:** **COMPLETE CHOICE PROPERTY**:

1. **OPTIMAL SOLUTION MUST END AT SOME ELEMENT**— we evaluate (in last loop) all possible last elements.

2. IF $x_i$ is in solution, then we want to find longest subsequence ending from elements 1...i-1 with max value ≤ $x_i$.

**OPTIMAL SUBSTRUCTURE PROPERTY.**

For best solution that uses $x_i$, the starting part of that solution is the longest increasing subsequence that ends with value ≤ $x_i$, and adding $x_i$ to the end of that subsequence does not create any conflicts.
6. (3 points) The following graph has labels on each edge that show the capacity of the edge. Find the minimum st-cut in this graph, and clearly indicate which nodes are in which part of the cut.

- (2 points) What is the cost of the cut?
- (3 points) Prove that the cut you have shown is the minimum cut.

A valid flow in graph is shown here —

the min cut has a cost ≥ any valid flow.
Since this flow has value 11, then 11 is a lower bound on possible values for a cut. Therefore all other cuts have size 11 or larger, so the cut shown is minimum.
7. (BONUS PROBLEM FOR EVERYONE. THE TAs THINK THIS PROBLEM IS MUCH HARDER THAN THE REST, SO MAKE SURE YOU DO EVERYTHING ELSE FIRST. 2 bonus points)

The input to this problem is a sequence of \( n \) points \( p_1, \ldots, p_n \) in the Euclidean plane \((x, y)\). These points are scattered and have no specific ordering. Two taxis start at the origin. Each point must be visited by at least one of the two taxis. If a taxi visits a point \( p_i \) before \( p_j \) then it must be the case that \( i < j \). The cost of a route that includes all \( n \) points is just the total distance traveled by the first taxi plus the total distance traveled by the second taxi. Your goal is to find the cost of the least expensive route for the three taxis to service these \( n \) requests. Give an \( O(n^2) \) time algorithm to solve this problem.

```plaintext
define \( p_0 = \text{origin} \)

for \( i \) from 0 to \( n \)
    for \( j \) from \( i+1 \) to \( n \)
        if \( i, j = O, 1 \rightarrow \text{table}[i][j] = \text{dist}(\text{origin}, p_i) \)
        if \( j = i+1 \)
            \( \text{table}[i][j] = \min( \text{table}[k][i] + \text{dist}(p_k, p_j) \) for all \( k < i \)
        else
            \( \text{table}[i][j] = \text{table}[i][j-1] + \text{dist}(p_{j-1}, p_j) \)

return \( \min \) of column \( 'n' \).

\text{general idea:}
- you have to keep track of where the taxis are
- make a \( n^2 \) chart where \( 'i' \) is the point most recently visited by Taxi 1, and \( 'j' \) is the point most recently visited by Taxi 2.
- WLOG, let \( j \geq i \) always.
- fill in chart from \((0,0)\) towards \((n-1, n)\)

\text{Case 1:} \( j \neq i+1 \) ex \((5,8)\)
- 8, 5 must come from a square \( w/ 7 \), because \( 7 \) was visited just before 8. The 5 taxi can't have gone to 7, so \((5,8)\) must come from extending \((5,7)\)

\text{Case 2:} \( j = i+1 \) ex \((4,5)\)
- \((4,5)\) comes from a square with a 4. Possibilities: \((3,4), (2,4), (1,4), (0,4)\)

\text{Runtime:} \( \Theta(n^2) + \Theta(n \cdot \log(n)) = \Theta(n^2) \)
```