COMP 271 Design and Analysis of Algorithms
2003 Spring Semester
Question Bank 3

There is some overlap between these question banks and the tutorials. Solving these questions (and those in the tutorials) will give you good practice for the midterm. Some of these questions (or similar ones) will definitely appear on your exam! Note that you do not have to submit answers to these questions for grading. Your TAs will discuss answers to selected questions in the tutorials.

1. Recall that in the matrix-chain multiplication problem, we are given a sequence $A_1, A_2, \ldots, A_n$ of n matrices, where A_i has size $p_{i-1} \times p_i$. The goal is to determine the least number of scalar multiplications needed to compute the matrix chain product, $A_1 A_2 \ldots A_n$.

   (a) Let $m(i, j)$ denote the least cost of multiplying matrices $A_i \ldots A_j$, where $j \geq i$. In class we wrote a recurrence relation for $m(i, j)$. This recurrence relation can be directly translated into a recursive function that takes two arguments, i and j. Draw a recursion tree to show what recursive calls result when this function is called with arguments $i = 3$ and $j = 6$, respectively.

   (b) By looking at the recursion tree of part (b), give two examples of subproblems which are being solved more than once.

   (c) Give an intuitive explanation as to why the purely recursive approach is inefficient for solving this problem. (Answer in just two lines.)

   (d) Dynamic programming can be used to solve this problem much faster than the purely recursive approach. Give the basic idea of this method, clearly explaining how it achieves this speed-up. (Answer in just two or three lines.)

2. A student in class suggested the following algorithm for the chain matrix multiplication problem:

   > Suppose $n > 2$ and p_i is the smallest of $p_0, \cdots, p_n$. Break the product after A_i, and recursively apply this procedure to the product $A_1..A_i$ and $A_{i+1}..A_n$.

   Does this algorithm work (i.e, does it minimize the number of multiplications needed)? If yes, prove your conclusion. If not, give a counter-example.

3. Consider the following purely recursive approach to solving the 0-1 Knapsack problem studied in class.

```
KnapSack(val, wt, n, W) {
    return RecKnapSack(val, wt, n, W);
}

RecKnapSack(val, wt, i, w) {
    if (w < 0) return -INFINITY;                          // neg. capacity--illegal
    if (i == 0) return 0;                                 // no items--no value
    leave_val = RecKnapSack(val, wt, i-1, w);               // if we leave item i
    take_val = val[i] + RecKnapSack(val, wt, i-1, w-wt[i]); // if we take item i
    return max(leave_val, take_val);
}
```

   (a) Show that this approach is inefficient by proving that the worst-case running time of `KnapSack(`*val*`,`*wt*`,n,W)` is at least $\Omega(2^{\min(n,W)})$.

   (b) Give an intuitive explanation as to why this algorithm is so much slower than the one based on dynamic programming given in class?

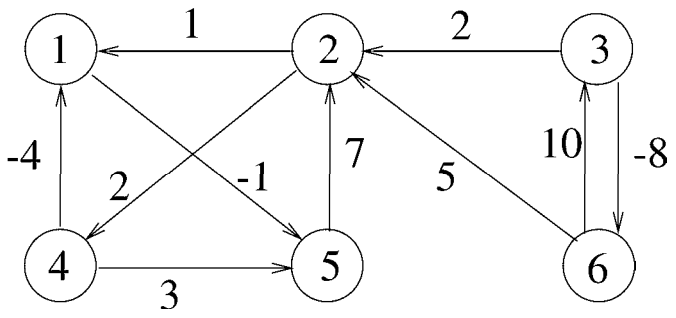
4. In this question, you are required to solve the 0-1 Knapsack problem for *two* knapsacks. You are given a set of n objects. The weights of the objects are $w_1, w_2, \ldots, w_n$, and the values of the objects are $v_1, v_2, \ldots, v_n$. You are given two knapsacks each of weight capacity C. If an object is taken, it may placed in one knapsack or the other, but not both. All weights and values are positive integers. Design an $O(nC^2)$ dynamic programming algorithm that determines the maximum value of objects that can be placed into the two knapsacks. Your algorithm should also determine the contents of each knapsack. Justify the correctness and running time of your algorithm.

5. Give an $O(n^2)$ time dynamic programming algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers (i.e, each successive number in the subsequence is greater than or equal to its predecessor). For example, if the input sequence is $\langle 5, 24, 8, 17, 12, 45\rangle$, the output should be either $\langle 5, 8, 12, 45\rangle$ or $\langle 5, 8, 17, 45\rangle$.

6. Give an algorithm which, given sequences X and Y, determines the longest sequence Z that is a subsequence of both X and Y. Your algorithm should run in $O(n^2)$ time, where $n = |X| + |Y|$.

7. The subset sum problem is: Given a set of n positive integers, $S = \{x_1, x_2, \ldots, x_n\}$ and an integer W determine whether there is a subset $S' \subseteq S$, such that the sum of the elements in S' is equal to W. For example, if $S = \{4, 2, 8, 9\}$ and $W = 11$, then the answer is "yes" because there is a subset $S' = \{2, 9\}$ whose elements sum to 11. Give a dynamic programming solution to the subset sum problem that runs in $O(nW)$ time. Justify the correctness and running time of your algorithm.

8. Run the Floyd-Warshall algorithm on the weighted, directed graph shown in the figure. Show the matrix $D^{(k)}$ that results for each iteration of the outer loop.



9. You are given a directed graph on n vertices in which each edge has a weight $c(u, v)$ which is equal to the *capacity* of the edge. (For example, $c(u, v)$ might represent the communication rate along a link in a telecommunications network.) The *capacity of a path* $\langle u_1, u_2, \ldots, u_k \rangle$ is defined to be the minimum capacity of any edge on the path, that is $\min(c(u_1, u_2), c(u_2, u_3), \ldots, c(u_{k-1}, u_k))$. By convention, $c(u, u) = \infty$. For every $u, v \in V$, define $C(u, v)$ to be the maximum capacity over all paths from u to v. Give a dynamic programming algorithm that computes $C(u, v)$ for all $u, v \in V$. Your algorithm should run in $O(n^3)$ time. You should briefly describe your algorithm and derive its running time.

10. You are to play a chessboard game to move a toy car from a starting point to a destination in at most M moves, where M is a fixed number specified in the input. The game is to be played on a chessboard $A[1..n, 1..n]$ of n^2 squares. The starting point is the square $A[1, 1]$ corresponding to the lower left corner. The destination is $A[n, n]$ corresponding to the upper right corner. In one move, you can move your car from one square to either its upper neighbor or its right neighbor or its left neighbor or its lower neighbor. Diagonal movement is disallowed and you cannot move your toy car outside the chessboard. Each square $A[i, j]$ is assigned a number $C[i, j]$ that denotes its cost. The goal is to go from $A[1, 1]$ to $A[n, n]$ in at most M moves while minimizing the maximum cost of any square visited. Design a dynamic programming algorithm to solve this problem. The following figure provides an example.

11. Consider the problem of neatly printing a paragraph on a printer. The input text is a sequence of n words of lengths $\ell_1, \ell_2, \ldots, \ell_n$, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j and we leave exactly one space between words, the number of extra space characters at the end of the line is $M - j + i - \sum_{k=i}^{j} \ell_k$. We wish to minimize the sum, over all lines except the last, of the cubes of the numbers of extra space characters at the end of lines. Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer. Analyze the running time and space requirements of your algorithm.

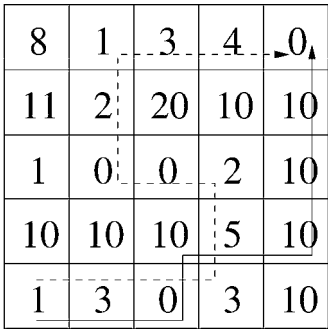


Figure 1: Suppose that $M = 20$. The solid path makes 8 moves and visits squares A[1,1], A[2,1], A[3,1], A[3,2], A[4,2], A[5,2], A[5,3], A[5,4], A[5,5] and the maximum cost of any square visited is 10. The dashed path makes 12 moves and visits squares A[1,1], A[2,1], A[3,1], A[4,1], A[4,2], A[4,3], A[3,3], A[2,3], A[2,4], A[2,5], A[3,5], A[4,5], A[5,5] and the maximum cost of any square visited is 5. Thus, one should prefer the dashed path to the solid path.

12. The president of A.-B. Corporation is planning a company party. The company has a hierarchical structure; that is, the supervisor relation forms a tree rooted at the president. The personnel office has ranked each employee with a *conviviality rating*, which is a real number. In order to make the party fun for all attendees, the president does not want both an employee and his or her immediate supervisor to attend.

    For each person u in the company, let $c[u]$ denote the conviviality rating for u, and let $S[u]$ denote the list of persons for which u is the immediate supervisor. If u does not supervise anyone, then $S[u]$ is empty.

    (a) Describe an algorithm to make up the guest list. The goal should be to maximize the sum of the conviviality ratings of the guests. Analyze the running time of your algorithm, as a function of the number of employees.

    (b) Suppose the president insists on being invited to his own party. How will you modify your algorithm to satisfy him?