1. Consider the recurrence relation $T(0) = T(1) = 2$ and for $n > 1$

$$T(n) = \sum_{i=1}^{n-1} T(i)T(i-1)$$

We consider the problem of computing $T(n)$ from n.

   (a) Show that if you implement this recursion directly in say the C programming language, that the program would use exponentially, in n, many arithmetic operations.

   (b) Explain how, by not recomputing the same $T(i)$ value twice, one can obtain an algorithm for this problem that only uses $O(n^2)$ arithmetic operations.

   (c) Give an algorithm for this problem that only uses $O(n)$ arithmetic operations.

†

2. Give a polynomial time algorithm that takes three strings, A, B and C, as input, and returns the longest sequence S that is a subsequence of A, B, and C.

†

3. Give an efficient algorithm for finding the shortest common super-sequence of two strings A and B. C is a super-sequence of A if and only if A is a subsequence of C.

HINT: Obviously this problem is very similar to the problem of finding the longest common subsequence. You should try to first figure out how to compute the length of the shortest common super-sequence.

††

4. Consider the algorithm that you developed for the previous problem.

   (a) Show the table that your algorithm constructs for the inputs $A = zxyyzz$, and $B = zzyxzy$

   (b) Explain how to find the length of the shortest common super-sequence in your table.

   (c) Explain how to compute the actual shortest common super-sequence from your table by tracing back from the table entry that gives the length of the shortest common super-sequence.

†

5. The input to this problem is a pair of strings $A = a_1 \dots a_m$ and $B = b_1 \dots b_n$. The goal is to convert A into B as cheaply as possible. The rules are as follows. For a cost of 3 you can delete any letter. For a cost of 4 you can insert a letter in any position. For a cost of 5 you can replace any letter by any other letter. For example, you can convert $A = abcabc$ to $B = abacab$ via the following sequence: *abcabc* at a cost of 5 can be converted to *abaabc*, which at cost of 3 can be converted to *ababc*, which at cost of 3 can be converted to *abac*, which at cost of 4 can be converted to *abacb*, which at cost of 4 can be converted to *abacab*. Thus the total cost for this conversion would be 19. This is almost surely not the cheapest possible conversion.

††

NOTE: The Unix diff command essentially solves this problem.

6. Find the optimal binary search tree for keys $K_1 < K_2 < K_3 < K_4 < K_5$ where the access probabilities/weights are $.5, .05, .1, .2, .25$ respectively using the algorithm discussed in class and in the notes. Construct one table showing the optimal expected access time for all subtrees considered in the algorithm, and another showing the roots of the optimal subtrees computed in the other table. Show how to use the table of roots to recompute the tree.

†

7. Assume that you have an n by n checkerboard. You must move a checker from the bottom left corner square the board to the top right corner square. In each step you may either 1) move the checker up one square, or 2) move the checker diagonally one square up and to the right, or 3) move the checker right one square. If you move a checker from square x to square y you get $p(x, y)$ dollars. You are told all of the $p(x, y)$ a priori. The $p(x, y)$ may be negative, zero or positive. You want to get as much money as possible. Give an efficient algorithm, say the number of steps should be polynomial in n, for this problem.

   NOTE: I know that this can be solved without really using dynamic programming, but please as an exercise try to use dynamic programming.

   †

8. Find the optimal order, and its cost, for evaluating the matrix product

   $$A_1 \times A_2 \times A_3 \times A_4 \times A_5$$

   where A_1 is 10×4, A_2 is 4×5, A_3 is 5×20, A_4 is 20×2, and A_5 is 2×50.

   †

9. Give an efficient algorithm for the following problem. The input is an n sided convex polygon. Assume that the polygon is specified by the Cartesian coordinates of its vertices. The output should be the triangulation of the polygon into $n - 2$ triangles that minimizes the sums of the perimeters of the into triangles. Note that this is equivalent to minimizing the length of the cuts required to create the triangle.

   HINT: This is very similar to the matrix multiplication problem in the text.

   HINT HINT: Fix a side of the polygon and call it e. How many possible triangles are there that contain e? For each such triangle compute the number of triangulations that contain that triangle.

   † † †

10. Design a recursive linear time algorithm that given a rooted binary tree will compute the balance factor of each node in the tree. The balance factor of a node is the difference in the heights of its children. The height of a node x is the number of nodes in the longest path from x to some descendent of x.

    HINT: You will need to strengthen the induction hypothesis slightly to compute move information than the balance factor of each node.

    †

11. The input to this problem is a sequence S of integers (not necessarily positive). The problem is to find the consecutive subsequence of S with maximum sum. "Consecutive" means that you are not allowed to skip numbers. For example if the input was

    $$12, -14, 1, 23, -6, 22, -34, 13$$

    the output would be $1, 23, -6, 22$. Give a linear time algorithm for this problem.

    HINT: As a first step you might determine why a naive recursive solution is not possible. That is, figure out why knowing the nth number, and the maximum consecutive sum of the first n-1 numbers, is not sufficient information to compute the maximum consecutive sum of the first n numbers. There examples, which show you need to strengthen the inductive hypothesis, should give you a big hint how to strengthen the inductive hypothesis.

    HINT HINT: Strengthen the inductive hypothesis to compute two different different consecutive subsequences, the maximum consecutive sum subsequence, and one other one.

    ††

12. The input to this problem is a tree T with integer weights on the edges. The weights may be negative, zero, or positive. Give a linear time algorithm to find the shortest simple path in T. The length of a path is the sum of the weights of the edges in the path. A path is simple if no vertex is repeated. Note that the endpoints of the path are unconstrained.

    HINT: This is very similar to the problem of finding the largest independent set in a tree.

    † † †

13. Consider the following variant to the Towers of Hanoi puzzle. Assume that the pegs are placed at the vertices A, B, and C of an equilateral triangle. Further assume that we add the following restrictions to the rules: Discs my only be moved in clockwise order. So, any disc moved from peg A must go to peg B, any disc moved from peg B must go to peg C, and any disc moved from peg C must go to peg A. Assume all discs start on peg A. It is still the case that you only can move one disc at a time, and that no larger disc can go on top of a smaller disc. The goal is to get all the discs to peg B using the fewest possible number of disc moves. Give an algorithm for this problem.

    HINT: Use recursion. You will have to strengthen the inductive hypothesis.

    ††

14. The input for this problem consists of n keys $K_1, \ldots, K_n$, with $K_1 < K_2 < \ldots, K_n$, and associated probabilities $p_1, \ldots, p_n$. The problem is to find the AVL tree for these keys that minimizes the expected depth of a key. An AVL tree is a binary search tree with the property that every node has balance factor -1, 0, or 1. Give a polynomial time algorithm for this problem.

    HINT: You will have to strengthen the inductive hypothesis. Obviously the height of a subtree is relevant.

    ††

15. The input consists of n intervals over the real line. The output should be a collection C of non-overlapping intervals such the sum of the lengths of the intervals in C is maximized. Give a polynomial time algorithm for this problem.

    HINT: Strengthen the induction hypothesis. Consider the intervals by increasing order of their left endpoints. Just like in the Longest Increasing Subsequence (LIS) problem, there are two pieces of information that are relevant about a partial solution. One is obviously the length of the intervals, what is the other? Think about what the answer was in the LIS problem.

    ††

16. Consider the code for the Knapsack program given in the class notes. Explain how one can actually find the highest valued subset of objects, subject to the weight constraint, from the Value table computed by this code.

    †

17. Explain how to solve the Knapsack problem using only $O(L)$ memory/space and $O(nL)$ time. You need only find the value and weight of the optimal solution, not the actual collection of objects.

    †

18. Give an algorithm for the following problem whose running time is polynomial in $n + L$.

    Input: positive integers $v_1, \ldots, v_n$, with $L = \sum_{i=1}^{n} v_i$.

    Output: A solution (if one exists) to $\sum_{i=1}^{n} (-1)^{x_i} v_i = 0$ where each x_i is either 0 or 1.

    HINT: Very similar to subset sum algorithm.

    ††

19. Give an algorithm for the following problem whose running time is polynomial in $n + \log L$:

    Input: positive integers $v_1, \ldots, v_n$ and L.

Output: A solution (if one exists) to $(\sum_{i=1}^{n} x_i v_i) \bmod n = L \bmod n$ where each x_i is either 0 or 1. Here $x \bmod y$ means the remainder when x is divided by y.

HINT: This is like the subset sum problem but you will have to be more severe in your pruning since you can't afford to have L partial solutions anymore.

††

20. Give an algorithm for the following problem whose running time is polynomial in $n + W$:

    Input: positive integers $w_1, \ldots, w_n$, $v_1, \ldots, v_n$ and W.

    Output: The maximum possible value of $\sum_{i=1}^{n} x_i v_i$ subject to $\sum_{i=1}^{n} x_i w_i \leq W$ and each x_i is a nonnegative integer.

    HINT: The tree of feasible solutions will be different than the one in the knapsack/subset sum problem since you may include an item in the final sum more than once. Each node in the tree of feasible solutions may now have up to W children.

    ††

21. Give an algorithm for the following problem whose running time is polynomial in $n + L$, where $L = \max(\sum_{i=1}^{n} v_i^3, \prod_{i=1}^{n} v_i)$.

    Input: positive integers $v_1, \ldots, v_n$

    Output: A subset S of the integers such that $\sum_{v_i \in S} v_i^3 = \prod_{v_i \in S} v_i$.

    ††

22. The input to this problem is a set of n gems. Each gem has a value in dollars and is either a ruby or an emerald. Let the sum of the values of the gems be L. The problem is to determine if it is possible to partition of the gems into two parts P and Q, such that each part has the same value, the number of rubies in P is equal to the number of rubies in Q, and the number of emeralds in P is equal to the number of emeralds in Q. Note that a partition means that every gem must be in exactly one of P or Q. You algorithm should run in time polynomial in $n + L$.

    HINT: Start as in the subset sum example. Your pruning rule will have to be less severe. That is, first ask yourself why you may not be able to prune two potential solutions that have the same aggregate value.

    ††

23. The input to this problem consists of an ordered list of n words. The length of the ith word is w_i, that is the ith word takes up w_i spaces. (For simplicity assume that there are no spaces between words.) The goal is to break this ordered list of words into lines, this is called a layout. Note that you can not reorder the words. The length of a line is the sum of the lengths of the words on that line. The ideal line length is L. No line may be longer than L, although it may be shorter. The penalty for having a line of length K is $L - K$. *The total penalty is the* **maximum** *of the line penalties.* The problem is to find a layout that minimizes the total penalty. Give a polynomial time algorithm for this problem.

    HINT: Consider whether how many layouts of the first m words, which have k letters on the last line, you need to remember.

    ††

24. The input to this problem is two sequences $T = t_1, \ldots, t_n$ and $P = p_1, \ldots, p_k$ such that $k \leq n$, and a positive integer cost c_i associated with each t_i. The problem is to find a subsequence of of T that matches P with maximum aggregate cost. That is, find the sequence $i_1 < \ldots < i_k$ such that for all j, $1 \leq j \leq k$, we have $t_{i_j} = p_j$ and $\sum_{j=1}^{k} c_{i_j}$ is maximized.

    ††

25. The input to this problem is a sequence of n points $p_1, \ldots p_n$ in the Euclidean plane. You are to find the shortest routes for two taxis to service these requests in order. Let us be more specific. The two taxis start at the origin. If a taxi visits a point p_i before p_j then it must be the case that $i < j$. (Stop

and think about what this last sentence means.) Each point must be visited by at least one of the two taxis. The cost of a routing is just the total distance traveled by the first taxi plus the total distance traveled by the second taxi. Design an efficient algorithm to find the minimum cost routing.

HINT: Use dynamic programming. Consider exhaustively enumerating the possible tours one point at a time. So after the ith stage you would consider all ways to visit the points $p_1, \ldots, p_i$. Then find a pruning rule that will reduce the number of tours we need to remember down to a polynomial number.

††

26. The input to this problem is n points $x_1, \ldots, x_n$ on a line. A good path P has the property that one endpoint of P is the origin and every x_i is covered by P. Note that P need not be simple, that is, it can backtrack over territory that it has already covered. Assume a vehicle moves along this path from the origin at unit speed. The response time r_i for each x_i is the time until the vehicle first reaches x_i. The problem is to find the good path that minimizes $\sum_{i=1}^{n} r_i/n$, the average response time. For example, if the points are $x_1 = 1$ $x_2 = 8$ and $x_3 = -2$ and the path visited the points in the order x_1, x_3, x_2, the average response time for this path would be $1/3 + (1+3)/3 + (1+3+10)/3$. Give a polynomial time algorithm for this problem.

27. Consider the problem where the input is a collection of n train trips within Germany. For the ith trip T_i you are given the date d_i of that trip, and the non-discounted fare f_i for that trip. The German railway system sells a Bahncard for 240 Marks that entitles you to a 50% fare reduction on all train travel within Germany within 1 year of purchase. The problem is to determine when to buy a Bahncard to minimize the total cost of your travel.

    For example if the input was $d_1 =$ January 11, 1997, $f_1 = 20$ Marks, $d_2 =$ February 11, 1998, $f_2 = 200$ Marks, $d_3 =$ January 11, 1999, $f_3 = 200$ Marks, $d_4 =$ March 13, 1999, $f_4 = 100$ Marks, $d_5 =$ February 11, 2002, $f_5 = 200$ Marks, and $d_6 =$ January 11, 2003, $f_6 = 600$ Marks, then you might buy a Bahncard on February 11, 1998, and February 11, 2002, resulting in a total cost of 1200 Marks.

    Give a polynomial time algorithm for this problem. The running time of you algorithm should be independent of the cost of a Bahncard.

    ††

28. Give a polynomial time algorithm for the following problem. The input consists of a sequence $R = R_0, \ldots, R_n$ of non-negative integers, and an integer k. The number R_i represents the number of users requesting some particular piece of information at time i (say from a www server. If the server broadcasts this information at some time t, the the requests of all the users who requested the information strictly before time t are satisfied. The server can broadcast this information at most k times. The goal is to pick the k times to broadcast in order to minimize the total time (over all requests) that requests/users have to wait in order to have their requests satisfied.

    As an example, assume that the input was $R = 3, 4, 0, 5, 2, 7$ (so $n = 6$) and $k = 3$. Then one possible solution (there is no claim that this is the optimal solution) would be to broadcast at times 2, 4, and 7 (note that it is obvious that in every optimal schedule that there is a broadcast at time $n + 1$ if $R_n \neq 0$). The 3 requests at time 1 would then have to wait 1 time unit. The 4 requests at time 2 would then have to wait 2 time units. The 5 requests at time 4 would then have to wait 3 time units. The 2 requests at time 5 would then have to wait 2 time units. The 7 requests at time 6 would then have to wait 1 time units. Thus the total waiting time for this solution would be

    $$3 * 1 + 4 * 2 + 5 * 3 + 2 * 2 + 7 * 1$$

    ††

29. Assume that you are given a collection $B_1, \ldots, B_n$ of boxes. You are told that the weight in kilograms of each box is an integer between 1 and some constant L, inclusive. However, you do not know the specific weight of any box, and you do not know the specific value of L. You are also given a pan balance. A pan balance functions in the following manner. You can give the pan balance any two

disjoint sub-collections, say S_1 and S_2, of the boxes. Let $|S_1|$ and $|S_2|$ be the cumulative weight of the boxes in S_1 and S_2, respectively. The pan balance then determines whether $|S_1| < |S_2|$, $|S_1| = |S_2|$, or $|S_1| > |S_2|$. You have nothing else at your disposal other than these n boxes and the pan balance. The problem is to determine if one can partition the boxes into two disjoint sub-collections of equal weight. Give an algorithm for this problem that makes at most $O(n^2\, L)$ uses of the pan balance.

† † †

30. Give a algorithm that takes a positive integer n as input, and computes the number of possible orderings of n objects under the relations $<$ and $=$. For example, if $n = 3$ the 13 possible orderings are as follows: $a = b = c$, $a = b < c$, $a < b = c$, $a < b < c$, $a < c < b$, $a = c < b$, $b < a = c$, $b < a < c$, $b < c < a$, $b = c < a$, $c < a = b$, $c < a < b$, and $c < b < a$. Your algorithm should run in time polynomial in n.

† † †