# Design and Analysis of Algorithms

## Dynamic Programming and Greedy Method

April 18, 2000

Homework II: Turn in the solutions to problems 1 and 9 on May 2nd, 2000 in class. No late or early submissions.

1. Problem 16-4, Page 326; (CLR)
2. Problem 16-3, Page 325; (CLR)
3. Problem 16-2, Page 325; (CLR)
4. Problem 16-1, Page 324; (CLR)
5. Problem 17-3, Page 354; (CLR)
6. Problem 17.2-1, Page 336; (CLR)
7. Problem 17.2-2, Page 336; (CLR)
8. Problem 17.2-3, Page 336; (CLR)
9. Problem 17.2-4, Page 337; (CLR)
10. Problem 17.2-5, Page 337; (CLR)
11. Problem 17.2-6, Page 337; (CLR)
12. **Reliability** We are given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $r(u, v)$, which is a real number in the range $0 \leq r(u, v) \leq 1$ that represents the reliability of a communication channel from vertex $u$ to vertex $v$. We interpret $r(u, v)$ as the probability that the channel from $u$ to $v$ will not fail, and we assume that these probabilities are independent. Give an **efficient** algorithm to find the most reliable path between two given vertices. Write the complete algorithm. Analyze its time complexity.
13. **Speech Recognition** We can use dynamic programming on a directed graph $G = (V, E)$ for speech recognition. Each edge $(u, v) \in E$ is labeled with a sound $\sigma(u, v)$ from a finite set $\Sigma$ of sounds. The labeled graph is a formal model of a person speaking a restricted language. Each path in the graph starting from a distinguished vertex $v_0 \in V$ corresponds to a possible sequence of sounds produced by the model. The label of a directed path is defined to be the concatenation of the labels of the edges on that path.

    Describe an efficient algorithm that, given an edge-labeled graph $G$ with distinguished vertex $v_0$ and a sequence $s = (\sigma_1, \cdots, \sigma_k)$ of characters from $\Sigma$, returns a path in $G$ that begins at $v_0$ and has $s$ as its label, if any such path exists. Otherwise, the algorithm should return NO-SUCH-PATH. Analyze the running time of the algorithm. Clearly write any dynamic programming formulation you may use to solve this problem.

14. **Coin Change**

    In the United States, coins are minted with denominations of 1, 5, 10, 25, and 50 cents. Now consider a country whose coins are minted with denominations $\{d_1, d_2, \ldots, d_k\}$ of units. They seek an algorithm that will enable them to make change of $n$ units using the minimum number of coins.

    - The greedy algorithm for making change repeatedly uses the biggest coin smaller than the amount to be changed until it is zero. Provide a greedy algorithm for making change of $n$ units using US denominations. Prove its correctness and analyze its time complexity.
    - Show that the greedy algorithm does not always give the minimum number of coins in a country whose denominations are $\{1, 6, 10\}$.
    - Give an efficient algorithm that correctly determines the minimum number of coins needed to make change of $n$ units using denominations $\{d_1, d_2, \ldots, d_k\}$. Analyze its running time.

15. **Difference of Sums** Give an algorithm for the following problem. Given a list of $n$ distinct positive integers, parition the list into two sublists, each of size $n/2$, such that the difference between the sums of the integers in the two sublists is minimized. Formulate a dynamic programming definition for the problem. Develop an algorithm based on your definition. What is the time complexity of your algorithm? You can assume that $n$ is a power of 2.

16. **Name** Scheduling tasks for two workers

    **Problem** Suppose we are given N tasks where task $i$ requires exactly $t_i$ hours and $t_i$ is a positive integer between 1 and $K$. Each task is to be assigned to one of two workers. Our goal is to find an assignment that minimizes the difference in total time assigned to the two workers. (Ideally, we would assign the tasks so both workers have the same amount of work; if that isn't possible, we want to make the loads as equal as possible.)

    Give a dynamic programming algorithm for finding the best assignment, and give a time analysis. The running time should be a polynomial in $N$ and $K$.

17. **Name** Single Machine Maximal Weighted Schedule

    **Instance** A set of $n$ jobs $J_i =< s_i, f_i, w_i >, 1 \leq i \leq n$, where $s_i < f_i$ and $0 < w_i$. $s_i$ is called the start time of the job, $f_i$, the finish time, and $w_i$ the weight.

    **Solution** A subset $S$ of the jobs.

    **Constraints** $S$ cannot contain two jobs $J_i$ and $J_k$, $i \neq k$, that overlap, i.e., where $s_i \leq s_k \leq f_i$.

    **Optimality** A solution $S$ has total weight $W(S) = \sum J_i w_i$.

    **Problem** Give the best algorithm you can for this problem. (Hint: It is possible to do in not too much more than linear time.)

18. **Name** Job Assignment

    **Problem** You are given a set of $n$ jobs and a set of $m$ machines. For each job you are given a list of machines capable of performing the job. An assignment specifies for each job one of the machines capable of performing it. The overhead for an assignment is the maximum number of jobs performed by the same machine. The job assignment problem is: given such lists and an integer $1 \leq k \leq n$, is there an assignment with overhead at most $k$?

19. **Name** Scheduling with Availability Constraints

**Problem** The input is a set of $n$ jobs, each with a positive real availability time, $a_i$, and a positive real duration, $d_i$. You want to schedule the jobs on a single processor without interrupts, under the constraint that no job can start before its availability time. in other words, each job gets assigned an interval of time $(s_i, f_i)$ where $s_i$ is called the start time and $f_i = s_i + d_i$ is the finish time. Since the processor can only do one job at a time, we have the constraint that all the intervals are disjoint, i.e., for each pair of jobs $i \neq j$ either $s_i \geq f_j$ or $s_j \geq f_i$. Also, since we cannot start jobs before they arrive, $a_i \leq s_i$ for each $i$. you wish to minimize the last finish time, i.e., the time when all jobs are completed. Describe a fast algorithm for this problem.

20. **Name** Maximum Length Chain of Subwords

    **Instance** A set of $n$ strings of length at most $k$ over a finite alphabet $\sigma$.

    **Possible Solution** A sequence of strings that form a chain under the (consecutive) subword relation; i.e., if the output is $w_1, w_2, \ldots, w_t$ then we can write $w_{i+1} = uw_iv$ for some strings $u, v$.

    **Problem** Find a chain of maximum size.

21. **Name** Longest Pattern Meeting Regular Description

    **Instance** An $n$ bit string $w$ over $0, 1$, and an $m$-state deterministic finite state automaton $D$ over the alphabet $0, 1$ (represented by its state transition table, start state, and an array saying for each state whether it is accepting.)

    **Solution Space** Substrings of $w$, i.e. consecutive subsequences $w_i, w_{i+1}, \ldots, w_{i+k}$

    **Problem** Find the longest substring accepted by $D$

    Give the fastest algorithm you can to solve this problem assuming $n \geq m$. (The time should be a function of both $n$ and $m$.) (Hint: An $O(n^2)$ algorithm is trivial)

22. **Name** Single Word Reusable Boggle (SWoRB)

    **Problem** You are given an $n \times n$ matrix of letters from a finite alphabet $\Sigma$, and a target word $w$ in $\Sigma$ of length $m$. You want to determine whether there is a (not necessarily simple, i.e., letters can be reused) path in the grid so that the letters along that path are $w$. Each step in the path can go from a point in the grid to any of its 8 neighboring points, including diagonal moves. Give an efficient algorithm to play SWoRB.

23. **Probabilities for multi-faceted dice:** There are $N$ dice, and the $i$-th die has its faces numbered from 1 to $k_i$ where each $k_i$ is a positive integer less than $K$. Assuming that on each die, each number is equally likely, we want to find the probability that when we roll all $N$ dice, the sum of the numbers is exactly $T$. To do so, we need to count exactly how many ways we can choose integers $d_1, d_2, ..., d_n$ with $d_i \leq k_i$, such that $\sum d_i = T$. (The probability is this count divided by the product of the $k_i$'s.)

    Find an efficient algorithm, and give its time analysis in terms of $N$ and $K$.

24. Give the best algorithm you can for the following problem:

    **Name** Blackjack Hand Card Counting

    **Instance** An array $A$ of $n$ positive integers (cards with face values) with values from 1 to $k$, and positive integers $l < n, v < kn$.

    **Problem** Count the number of sets of $l$ array positions (hands of $l$ cards) whose total value is equal to $v$.

Analyze your algorithm in terms of $n, k$ and $l$. Your algorithm should take time polynomial in all 3 parameters.

25. **Multiplication** Consider the problem of examining a string $x = x_1x_2 \ldots x_n$ of characters from an alphabet of $k$ symbols, and a multiplication table over this alphabet, and deciding whether or not it is possible to parenthesize $x$ in such a way that the value of the resulting expression is $a$, where $a$ belongs to the alphabet. The multiplication tables is neither commutative nor associative, so the order of multiplication matters.

    Give an algorithm, with time in polynomial in $n$ and $k$, to decide whether such a parenthesization exists for a given string, multiplication table, and goal element.

26. **Data Compression** Consider the following data compression technique. We have a table of $m$ text strings, each of length at most $k$. We want to encode a data string $D$ of length $n$ using as few text strings as possible. For example, if our table contains $(a, ba, abab, b)$ and the data string is $bababbaababa$, the best way to encode it is $(b, abab, ba, abab, a)$ — a total of five code words. Give an $O(nmk)$ algorithm to find the length of the best encoding. You may assume that the string has an encoding in terms of the table.

27. **Fix Point** Suppose you are given a sorted sequence of *distinct* integers $\{a_1, a_2, \ldots, a_n\}$. Give an $O(\log_2 n)$ algorithm to determine whether there exists an index $i$ such that $a_i = i$.

28. **Bad Intersections** Consider a city whose streets are defined by an $X \times Y$ grid. We are interested in walking from the upper left-hand corner of the grid to the lower right-hand corner.

    Unfortunately, the city has bad neighbourhoods, which are defined as intersections we do not want to walk in. We are given an $X \times Y$ matrix $BAD$, where $BAD[i, j] = "Yes"$ if and only if the intersection between streets $i$ and $j$ is somewhere we want to avoid.

    (a) Ginve an example of the contents of BAD such that there is no path across the grid avoiding bad neighbourhoods.

    (b) Give an $O(XY)$ algorithm to find a path that avoids bad neighbourhoods.

    (c) Give an $O(XY)$ algorithm to find the shortest path across the grid that avoids bad neighbourhoods. You may assume that blocks are of equal length.

29. **Book Packing** Consider the problem of storing $n$ books on shelves in a library. The order of the books is fixed by the cataloging system and so cannot be rearranged. Therefore, we can speak of a book $b_i$, where $1 \leq i \leq n$, that has a thickness $t_i$ and height $h_i$. The length of each bookshelf at this library is $L$.

    (a) Suppose all the books have the same height $h$ and the shelves are all separated by a distance of greater than $h$, so any book fits on any shelf. The greedy algorithm would fill the first shelf with as many books as we can until we get the smallest $i$ such that $b_i$ does not fit, and then repeat with the subsequent shelves. Show that the greedy algorithm always finds the optimal shelf placement, and analyze its time complexity.

    (b) Now consider the case where the height of the books is not constant, but we have the freedom to adjust the height of each shelf to that of the tallest book on the shelf. Thus the cost of a particular layout is the sum of the heights of the largest book on each shelf. Give an example to show that the greedy algorithm of stuffing each shelf as full as possible does not always give the minimum overall height.

    Give an algorithm for this problem, and analyze its time complexity.