1A Lent Algorithms

Supervision 2: Algorithm Design

Hayk Saribekyan

January 31, 2019

Where you are asked to describe an algorithm, also analyse its time and space complexity. Mention how you would implement the solution and give recurrence relations for dynamic programming algorithms. You should also always prove that your algorithms are correct. This is, in particular, important for greedy algorithms.

In this supervision we will concentrate on few important approaches for algorithm design: dynamic programming and its implementation techniques, greedy algorithms and their proofs, divide and conquer.

1 Refresher and Warm-up

1. What does the statement "The running time of an algorithm is at least $O(n)$" mean, if anything?

2. Show that for any two real a and $b > 0$
$$(n+a)^b = \Theta(n^b)$$

3. Write a pseudocode of a binary search algorithm that finds element x in a sorted array a_i in $O(\log n)$ time. Pay attention on how l and r are modified. In binary search always think of the base cases when $r - l = 0$ or 1.

4. When would you prefer dynamic programming using tabulation vs memoisation and vice versa? Give an example of a problem, which can be decomposed into smaller instances of itself but no DP is necessary.

5. You are playing a game on a map represented by an $n \times n$ matrix. You start in the upper left corner, and your objective is to reach the bottom right corner. At each step, you are only allowed to go right or down. Each cell contains some amount of coins. Explain the algorithm you would use to determine the path that will maximise your number of coins.

   Below is an example of an optimal path:

1	**3**	**5**	3	1
4	2	**5**	**3**	4
2	2	2	**4**	5
4	4	1	**5**	**3**
5	1	2	3	**1**

Credit: this problem is from Petar Velickovic's notes.

2 Examples

1. The Knapsack problem. The hiker has to choose some of the n items to take to a trip in his knapsack of capacity W pounds. Item i weighs w_i pounds and has a value v_i.

   (a) Describe an algorithm that will maximize the value the hiker can carry.

   (b) Modify your solution for the case when the hiker has infinite supply of each item.

   (c) Give an example, where a simple greedy algorithm that picks items with largest v/w ratio does not work.

   (d) Can you solve the problem using just $O(W)$ additional memory?

2. The shoemaker has to complete n orders. Order i takes t_i days. For every day of delay of order i, the shoemaker gets fined s_i pence. Unfortunately, the shoemaker cannot work on more than one order a day. Write an algorithm that will help the shoemaker decide the order in which he should do the jobs so that the total amount of fine is minimised.

   If you come up with a simple solution make sure to prove its correctness.

3. Given an array of integers. Implement a divide-and-conquer algorithm that returns the maximum sum of a contiguous subsequence of the integers. Your algorithm should run in $O(n \log n)$ time or better.

4. I have n supervisions to schedule. Supervision i starts at time a_i and ends at time b_i. Help me find the minimum number of supervisions that I will have to reschedule so that I do not have any overlapping ones. It would be wonderful if your algorithm could also give me the list of the supervisions that should be rescheduled!

5. Given n line segments on an axis described by their endpoints a_i and b_i. Describe an algorithm that picks the smallest number of them so that their union covers the interval $[0, t]$.

6. A string is called palindrome if it reads the same from both ends e.g. `madam`. Given a string of length n. Describe an algorithm that determines the size of its longest palindrome subsequence? The subsequence does not have to be contiguous. Example: if the string is `baobab`, then we can remove `o` and get `babab`, which is palindrome.

   Explain how can we find this subsequence.

   Hint: what can you say if the first and last letters of the string are the same. What if they are different?

   The DP implementation can either use memoisation or fill in a matrix in a diagonal order.

   There is another DP solution (found by Scarlet Cox). The longest subpalindrome's length is equal to the longest common subsequence of the given string and its reverse. The longest common subsequence of two strings can be found using DP in $O(n^2)$ time. Of course, the above statement needs proving.