Greedy algorithms

Rossano Venturini[1]

1 **Computer Science Department, University of Pisa, Italy**
rossano.venturini@unipi.it

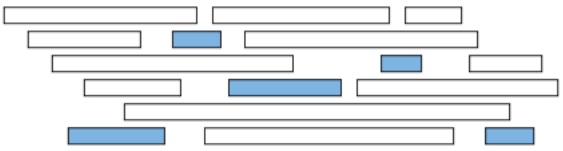
Notes for the course *"Competitive Programming and Contests"* at Department of Computer Science, University of Pisa.
Web page: `https://github.com/rossanoventurini/CompetitiveProgramming`

These notes sketch the content of the 12th lecture.

1 Activity selection

▶ **Problem 1** (Activity selection). *We are given two arrays $S[1 \dots n]$ and $F[1 \dots n]$ listing the start and finish times of each activity. We can assume that $0 \leq S[i] < F[i] \leq M$ for each i, for some value M. The task is to choose the largest possible subset $X \subseteq \{1, 2, \dots, n$ so that for any pair $i, j \in X$, either $S[i] > F[j]$ or $S[j] > F[i]$.*

We can illustrate the problem by drawing each activity as a rectangle whose left and right x-coordinates show the start and finish times. The goal is to find a largest subset of rectangles that do not overlap vertically.



A maximal conflict-free schedule for a set of classes.

Intuitively, we'd like the first activity to finish as early as possible, because that leaves us with the most remaining activities.

If this greedy strategy works, it suggests the following very simple algorithm. Scan through the activities in order of finish time; whenever you encounter a activity that doesn't conflict with your latest activity so far, take it!

The above algorithm runs in $\Theta(n \log n)$ time.

To prove that this algorithm actually gives us a maximal conflict-free schedule, we use an *exchange argument*.

We are not claiming that the greedy schedule is the only maximal schedule; there could be others. (See the figures on the.

All we can claim is that at least one of the maximal schedules is the one that the greedy algorithm produces.

▶ **Lemma 2.** *At least one maximal conflict-free schedule includes the activity that finishes first.*

Proof. Let f be the activity that finishes first. Suppose we have a maximal conflict-free schedule X that does not include f.

Let g be the first activity in X to finish.

Since f finishes before g does, f cannot conflict with any activity in the set $S \setminus \{g\}$.

Thus, the schedule $X' = X \cup \{f\} \setminus \{g\}$ is also conflict-free.

Since X' has the same size as X, it is also maximal. ◀

We use induction to complete the proof.

▶ **Lemma 3.** *The greedy schedule is an optimal schedule.*

Proof. Let f be the activity that finishes first, and let L be the subset of activities the start after f finishes.

The previous lemma implies that some optimal schedule contains f, so the best schedule that contains f is an optimal schedule.

The best schedule that includes f must contain an optimal schedule for the activities that do not conflict with f, that is, an optimal schedule for L.

The greedy algorithm chooses f and then, by the inductive hypothesis, computes an optimal schedule of activities from L. ◀

The basic structure of this correctness proof is an inductive exchange argument.

- Assume that there is an optimal solution that is different from the greedy solution.
- Find the "first" difference between the two solutions.
- Argue that we can exchange the optimal choice for the greedy choice without degrading the solution.

This argument implies by induction that some optimal solution that contains the entire greedy solution, and therefore equals the greedy solution.

2 Job sequencing

▶ **Problem 4** (Job sequencing). *Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes single unit of time, so the minimum possible deadline for any job is* 1. *How to maximize total profit if only one job can be scheduled at a time.*

As an example consider the following jobs.

JobID	Deadline	Profit
a	2	100
b	1	19
c	2	27
d	1	25
e	4	15

The optimal sequence is `c, a, e`.

A Simple Solution is to generate all subsets of given set of jobs and check individual subset for feasibility of jobs in that subset. Keep track of maximum profit among all feasible subsets. The time complexity of this solution is exponential.

An algorithm that uses the greedy approach is as follows.

- Use an array with a slot for each possible deadline
- Processes jobs in decreasing order of profit
- Let j be the current job and d be its deadline. Place job j the the rightmost empty slot to the left of position d, if any. If such a empty slot does not exit, discard job j.

1	2	3	4	5
c	a		e	

Trivial implementation has $\Theta(n^2)$ time, where n is the number of jobs. Can you implement it in $\Theta(n \log n)$ time?

3 Fractional knapsack

▶ **Problem 5** (Knapsack). *Given weights and values of n items. Each item i has a value v_i and a weight w_i. We need put a subset of these items in a knapsack of capacity W to get the maximum total value in the knapsack.*

In the 0-1 Knapsack problem, we are not allowed to break items. We either take the whole item or do not take it.

In Fractional Knapsack, we can break items for maximizing the total value of knapsack.

As an example, consider three items: $v = \{60, 100, 120\}$ and $w = \{10, 20, 30\}$ and a knapsack of capacity $W = 50$.

The maximum possible value is 240 obtained by taking full items of 10 and 20 and 2/3rd of last item of 30.

An efficient solution is to use the greedy approach.

The basic idea of greedy approach is to calculate the ratio value/weight for each item and sort the item on basis of this ratio. Then, we take the item with highest ratio and add them until we can't add the next item as whole and at the end add the next item as much as we can.

This strategy always obtains an optimal solution of this problem.

To see why associate a rectangle to each item. The rectangle of item i has a base of size w_i and a height of size v_i. The diagonal of this rectangle is a segment of slope v_i/w_i.

Consider now any selection of items whose total weight equals W.

We can sort the selected items in order of their ratio and draw the diagonals of their rectangles, one after the other.

There cannot exist any assignement whose drawn is above the one of the greedy selection.

4 Problems

▶ **Problem 6** (Chat room). *Vasya has recently learned to type and log on to the Internet. He immediately entered a chat room and decided to say hello to everybody. Vasya typed the word s. It is considered that Vasya managed to say hello if several letters can be deleted from the typed word so that it resulted in the word* `hello`*. For example, if Vasya types the word* `ahhellllloou`*, it will be considered that he said hello, and if he types* `hlelo`*, it will be considered that Vasya got misunderstood and he didn't manage to say hello. Determine whether Vasya managed to say hello by the given word s.*

It is a standard problem of establishing if a certain string w (in this case $w =$ `hello`) is subsequence of another string s. It can be solved with a simple greedy algorithm.

▶ **Problem 7** (Magic numbers). *A magic number is a number formed by concatenation of numbers* 1*,* 14 *and* 144*. We can use each of these numbers any number of times. Therefore* 14144*,* 141414 *and* 1411 *are magic numbers but* 1444*,* 514 *and* 414 *are not.*

You're given a number. Determine if it is a magic number or not.

This is a standard problem. We are given a dictionary D of strings (in this case, $D = \{1, 14, 144\}$) and our goal is to partition a given string S into pieces which are elements of D, if any. If D is suffix-close (i.e., there does not exist a string in D which is a proper suffix of another string in D), the partition can be found with a simple greedy approach.

▶ **Problem 8** (Wilbur and Array). *Wilbur the pig is tinkering with arrays again. He has the array $a_1, a_2, \ldots, a_n$ initially consisting of n zeros. At one step, he can choose any index i and either add* 1 *to all elements $a_i, a_{i+1}, \ldots, a_n$ or subtract* 1 *from all elements $a_i, a_{i+1}, \ldots, a_n$. His goal is to end up with the array $b_1, b_2, \ldots, b_n$.*

Of course, Wilbur wants to achieve this goal in the minimum number of steps and asks you to compute this value.

The minimum number of operations equals the sum of the absolute values of differences between consecutive elements of b, i.e., $b_1 + \sum_{i=2}^{n} |b_i - b_{i-1}|$.

▶ **Problem 9** (Alternative thinking). *Kevin has just recevied his disappointing results on the USA Identification of Cows Olympiad (USAICO) in the form of a binary string of length n. Each character of Kevin's string represents Kevin's score on one of the n questions of the olympiad—* 1 *for a correctly identified cow and* 0 *otherwise.*

However, all is not lost. Kevin is a big proponent of alternative thinking and believes that his score, instead of being the sum of his points, should be the length of the longest alternating subsequence of his string. Here, we define an alternating subsequence of a string as a not-necessarily contiguous subsequence where no two consecutive elements are equal. For example, $\{0, 1, 0, 1\}$, $\{1, 0, 1\}$, and $\{1, 0, 1, 0\}$ are alternating sequences, while $\{1, 0, 0\}$ and $\{0, 1, 0, 1, 1\}$ are not.

Kevin, being the sneaky little puffball that he is, is willing to hack into the USAICO databases to improve his score. In order to be subtle, he decides that he will flip exactly one substring—that is, take a contiguous non-empty substring of his score and change all 0*s in that substring to* 1*s and vice versa. After such an operation, Kevin wants to know the length of the longest possible alternating subsequence that his string could have.*

The longest alternating subsequence of a binary string s can be found with a greedy approach which starts for the beginning of s and moves to the first occurrence of (alternating) 0 or 1. We observe that a flip operation increases the length of a subsequence by exactly

2 and that this happen if and only there are two consecutive occurrences of either 0 or 1. The latter condition always happens if the length of the longest alternating subsequence is smaller than n. Thus, the answer is the minimum between n and the length of the longest alternating subsequence plus 2.