Greedy Algorithms

In many optimisation problems, an optimal solution is constructed by making a sequence of successive choices/decisions.

Ex: Martix chain order

$$\overset{k}{\downarrow}$$

$$A_1 \cdots A_k \cdots A_n \longleftarrow \text{Choice}$$

$$\underbrace{(A_1 \cdots A_k)}\underbrace{(A_{k+1} \cdots A_n)}$$

more choices

In dynamic programming solutions these choices were made after solving the subproblems.

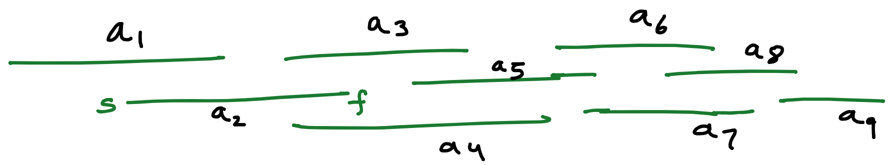
In contrast, in greedy algorithms a choice is made without solving the subproblems. The choice is made based on some heuristics which chooses the current best (locally optimal). That is why the name greedy.

Greedy choice does not yield optimal solution for all problems but for some problems it does.

In the cases where greedy choice succeeds, we get an algorithm which is more efficient than dynamic programming algorithm. This is because a greedy algorithm solves only those subproblems which are needed to be solved after making a greedy choice rather than solving all subproblems as in dynamic programming.

Greedy algorithm is top-down recursive algo.

Example: Choosing set of activities



Given a set of n activities. Each activity has a start and a finish time, given by arrays $s[1...n]$ and $f[1...n]$

We assume that activities are sorted by their finish time.

Goal is to find the maximum cardinality subset of activities s.t. no two activities in this set overlap.

Concrete application: suppose we have a lecture hall. The activities are classes or meetings of variable time lengths. What is the maximum no. of activities (out of these) that can be scheduled in a lecture hall?

$\Rightarrow$ non-overlapping set of activities of maximum cardinality.

Structure of optimal solution

In any solution the set of activities is totally ordered

(— — —)

Let S_i be the set of activities that start after a_i finishes.

Subproblem: Let U_i be the maximum size subset of S_i which has non-overlapping activities.

$$U_i = \max_{j \in S_i} \{ 1 + U_j \}$$

a_j { U_j }

(Solution to original problem may be assumed as so, we assume a fictitious activity of 0 duration which starts and finishes before any other activity starts)

Total no. of subproblems is $n+1$, To compute solution of a subproblem we need to consider atmost $O(n)$ subproblem

$\Longrightarrow$ DP solution $O(n^2)$.

Consider a 'greedy strategy' in solving U_i:

Choose $j \in S_i$ which finishes earliest.

Based on the rationale, choosing such an activity will leave the largest interval of time to select other activities.

Claim: U_i has an optimal solution in which greedy choice has been made.

Proof Let a_k be the greedy choice for U_i. Let a_ℓ be a choice corresponding to some optimal solution.

Let the optimal solution be $a_\ell, a_{r_1}, \ldots, a_{r_m}$

a_k finishes before (or at the same time) a_ℓ $[f(k) \leq f(\ell)]$.

$a_{r_1}, a_{r_2}, \ldots, a_{r_m}$ is an optimal solution for U_ℓ

greedy choice corresponds to $\{a_k\} \cup \underbrace{\{\ldots\ldots\}}_{U_k}$

since $f(k) \le f(\ell)$, $a_{r_1}, a_{r_2} \ldots a_{r_m} \in S_k$

so $a_k, a_{r_1}, a_{r_2} \ldots a_{r_m}$ is another optimal solution. $\square$

Pseudo-code for greedy algorithm.

```
max-activities (n, s, f)
    m = 1    // a_m is the activity being considered currently
    c = 0    // finish time of last activity in A
    A = ∅    // set of activities picked so far
    while m ≤ n do
        if (s[m] > c)
            A = A ∪ {a_m}
            c = f(m)

        m = m+1
    print A.
```

Implements greedy strategy.

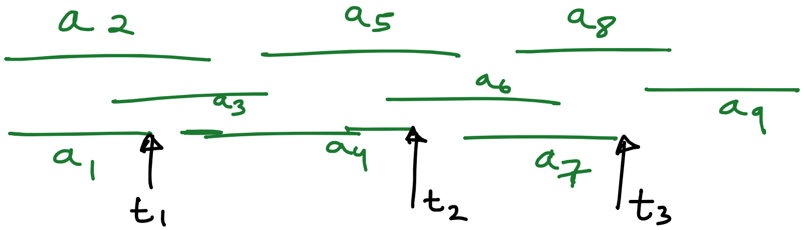
Time Complexity

$O(n)$

[$O(n \log n)$, if we need to sort the input list by finishing time of activities]

$\Longrightarrow$ Improvement over $O(n^2)$ DP algorithm.

Activities sorted by their finish times.



$m = 1$

$c = 0$

$A = \phi$

$A = \{a_1, a_4, a_7\ a_9\}$ ⟵ output

$c = t_1$

a_2 ✗

a_3 ✗

$c = t_2$

a_5 ✗

a_6 ✗

$c = t_3$

a_8 ✗