An *optimization problem* is a problem in which we want to find a "best" solution out of a set of *feasible* solutions.

- A solution is feasible if it satisfies a given set of constraints.

Generally an optimization problem will want to maximize some value or minimize some cost.

- Maximization ex: Toll Booth problem.
- Minimization ex: Matrix Chain.

We have considered some dynamic programming approaches to some optimization problems. For some problems, dynamic programming is overkill, and there may be a more efficient algorithm.

A **greedy algorithm** is an algorithm which solves a subproblem by making a choice which appears to be the best choice at the moment. After making this choice, we obtain a new subproblem which we again solve by making a choice which appears to be the best choice at that moment.

- Matrix Chain example: Find the largest p_i and multiply A_i with A_{i+1}. Repeat this procedure until there is only one matrix.
- Toll Booth example: Choose the toll booth with the largest value, and remove any toll booths which would be too close to this booth. Repeat this procedure until we cannot add any more toll booths.

Neither one of these examples guarantees an optimal solution. That is, there are *counterexamples* for which these algorithms will compute a solution which is not optimal.

That being said, there are many problems for which there exist greedy algorithms which guarantee to return an optimal solution, and generally a greedy algorithm will be more efficient (in time and space) than a dynamic programming algorithm.

Suppose we have a set $S = \{a_1, a_2, \ldots, a_n\}$ of n proposed *activities* that wish to use a resource (e.g. a lecture hall) which can serve only one activity at a time.

Each activity a_i has a *start time* s_i and a *finish time* f_i where

$$0 \leq s_i < f_i < \infty$$

If selected, activity a_i takes place during the half-open time interval $[s_i, f_i)$. Activities a_i and a_j are *compatible* if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

The **activity-selection problem** wants to select a maximum-size subset of mutually compatible activities.

We assume that the activities are listed in non-decreasing order according to finishing time:

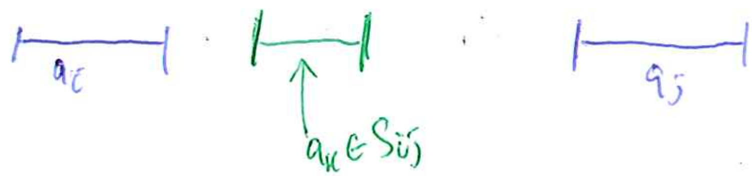
$$f_1 \leq f_2 \leq \ldots \leq f_n$$

Example:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Note we can view the activities as intervals where the left end is s_i and the right end is f_i.

Let S_{ij} denote the set of activities that start after a_i finishes and finish before a_j starts. Suppose we wish to find a maximum set of mutually compatible activities in S_{ij}.



Let A_{ij} denote the optimal solution, and suppose it contains some activity a_k. Note that a_k divides S_{ij} into two subproblems S_{ik} and S_{kj}.

The optimal solution for S_{ij} is therefore $A_{ik} \cup A_{kj} \cup \{a_k\}$ (using similar "cut-and-paste" argument). The size of the solution is $|A_{ik}| + |A_{kj}| + 1$.

Let $c[i, j]$ denote the size of an optimal solution for S_{ij}. We have the following recurrence relation (note this is similar to the matrix chain problem):

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max\limits_{a_k \in S_{ij}} \{ c[i,k] + c[k,j] + 1 \} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

We could solve this problem via dynamic programming, and the running time would be $O(n^3)$ (we have $O(n^2)$ subproblems, and each one takes $O(n)$ time to compute). Can we do better?

What might be a good greedy strategy when determining an activity to include in our optimal solution?

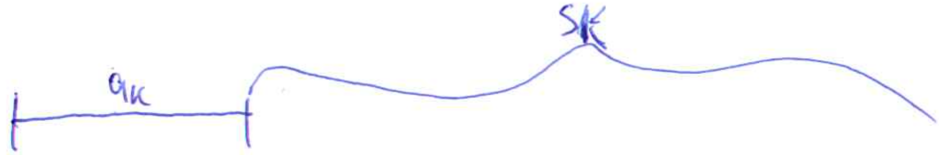
Intuition tells us that it may be a good idea to include an activity which ends as early as possible, as we increase the number of activities in our solution and we leave as much time left as possible for the remaining activities.

Could it be that repeating this procedure repeatedly until we cannot add any more activities results in an optimal solution? Can we either prove that the algorithm is correct or can we construct a counterexample which shows that it fails?

Toll Booth Counterexample:

$X_1 = 5$, $X_2 = 10$, $X_3 = 15$

$t_1 = 9$, $t_2 = 10$, $t_3 = 9$



Let $S_k = \{a_i \in S : s_i \geq f_k\}$ be the set of activities that start after a_k finishes. We will prove the following theorem:

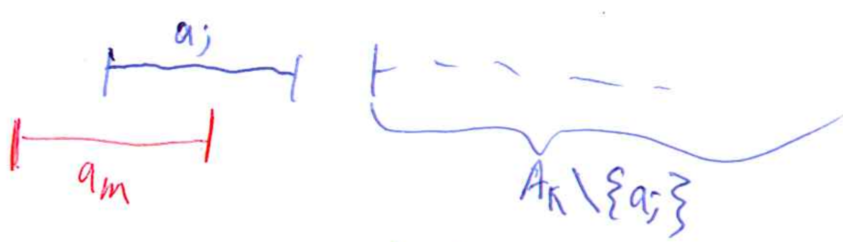
Theorem

Consider any nonempty subproblem S_k and let a_m denote the activity in S_k with the earliest finishing time. Then a_m is included in some maximum-sized subset of compatible activities of S_k.

Proof

Let A_k be an optimal solution for S_k, and let a_j be the activity in A_k with earliest finishing time. If $a_j = a_m$, we are done, so assume $a_j \neq a_m$. Consider the set

$$A_k' = A_k \setminus \{a_j\} \cup \{a_m\}.$$



A_k' is feasible and $|A_k'| = |A_k|$ so A_k' is optimal.

An iterative greedy algorithm:

```
A = {a_1}
k = 1
for m = 2 to n {
    if s[m] ≥ f[k] {
        A = A ∪ {a_m}
        k = m
    }
}
Return A
```