Greedy

(We know)

Greedy Method
• a strategy to solve optimization problems
• applicable where the global optima may be found by successive selections of local optima

The Greedy principle (strategy) is
• to successively incorporate elements that realize the local optimum

General abstraction for a Greedy-like problem
Let us consider the given set C of candidates to the solution of a given problem P. We are required to provide a subset B ($B \subseteq C$) to fulfill certain conditions (called internal conditions) and to maximize (minimize) a certain objective function.

Greedy algorithm– sample

```
Greedy algorithm
        Input:  C - a collection of candidates
        Output: Greedy =      true    if a solution exists;
                                      and B - the solution found
                              false   (otherwise)

Function Greedy (C, B)
        B = empty
        while not solution(B) and (C is not empty) do
                candidate := selectLocalOptimum (C) // selectMostPromissing
                remove(C , candidate)
                if acceptable(B, candidate) then
                        append(B, candidate)
                endif
        endwhile
        if solution(sol) then
                GreedySub:=true
        else
                GreedySub:=false
        endif
end_Greedy
```

It is necessary that before applying a Greedy algorithm to prove that it will provide the optimal solution.

<u>Example</u>:
The ***activity selection problem*** :
the goal is to pick the maximum number of activities that do not clash with each other.
The activity selection problem is notable in that using a greedy algorithm to find a solution will always result in an optimal solution.

Interval Scheduling

Given: n jobs, each with a start and finish time $[s_i, f_i)$.

Goal: Schedule the maximum number of (non-overlapping) jobs on a single machine.

<u>Remark</u>: project **problems in set 1** can be solved in a similar way with this kind of problem

To apply the greedy approach to this problem, we will schedule jobs successively, while ensuring that no picked job overlaps with those previously scheduled. The key design element is to decide the order in which we consider jobs.

A pseudocode sketch of the algorithm:

```
Subalg. GreedyInterval (C, B)
    sort(C)
    B = empty
    while not solution(B) and (C is not empty) do
        candidate := getNextFromSorted (C) // or extract it
        if acceptable(B, candidate) then
            append(B, candidate)
        endif
    endwhile
end_Greedy
```

There are several ways of sorting jobs.

- Shortest job first
- Earliest start first
- Fewest conflict first   BAD!

Picking jobs in increasing order of finish times gives the optimal solution.
(It is necessary to prove it!)

Subalgorithm: ***sort***(C) sort the activities in descending order by finishing time

(What and) How to prove:

The "earliest finish time first" algorithm described above generates an optimal schedule for the interval scheduling problem.

Consider an optimal solution S with at n jobs:

s1, s2, …, sn – being the jobs from S, ordered by their finish time.
(and they do not overlap)

Consider a solution G given by Greedy:

g1, g2, …– being the jobs from G, ordered by their finish time

(we didn't specify the number of jobs in G)
(G <= picking jobs in increasing order of finish times, and not overlapping)

We prove by induction on k that **P(k)**:

(For any k=1,n :)

P(k) **P1(k):** it exists g_k ⇔ greedy algorithm schedules at least k jobs

P2(k): g_k finishes earlier or in the same time with job s_k

And because: last finishing job in G from 1,..,k is g_k
And last finishing job in S is s_k

⇨ the first k jobs in G (scheduled by greedy algorithm) finish no later than the first k jobs in the optimal solution S.

<u>Remark</u>:

for any k=1,..,n means also that k<=n.
there are at least k jobs in solution

We now prove the claim:

Base case:

P(1) **P1(1)**

Because there are at least k jobs in solution => exist g_1
(We can't pick a job only if there are no jobs)

P2(1)

Greedy choses the first ending job => g_1 finishes earlier or at the same with s_1

Inductive step:

P(k)=>P(k+1)
(only if k+1<=n)

What we know:

P(k) => job g_k finishes earlier or in the same time with job s_k

let s_{k+1} – the next job in S

- ⇨ s_{k+1} starts after s_k ends
- ⇨ s_k finishes after or at the same time with g_k <= P(k)
- ⇨ it exists an nonoverlapping job that starts after g_k
  => Greedy (can) choose a job g_{k+1}
  => greedy algorithm schedules at least k+1 jobs **P1(k)**

- ⇨ s_{k+1} finishes after s_k that finishes in the same time or after g_k
- ⇨ s_{k+1} was not among of $g_1 \ldots g_k$
  In the set of jobs from which Greedy chooses the first finishing job, there is also job s_{k+1}; Greedy choses the first finishing job from here
- ⇨ g_{k+1} finishes at the same time or earlier than s_{k+1} **P2(k)**
  (This completes the proof of the claim.)