Greedy Algorithms

Refer Chapter 4 from Tardos

04-02-2022

Greedy Algorithms

- What is it?

Greedy Algorithms

- **What is it?**
- An algorithm is greedy if it builds the solution in small steps, choosing the best possible decision at each time step based on some underlying criteria.
- For the same problem, many greedy algorithms are possible each using a different criteria.
- Some problems cannot be solved by greedy approach at all.
- Choosing a current best solution without worrying about future. In other words the choice does not depend upon future sub-problems.
- Such algorithms are locally optimal, For some problems, as we will see shortly, this local optimal is global optimal also and we are happy.

**It is easy to invent a greedy algorithm for almost any problem.
Finding cases in which they work well, and proving that they work well is challenging.**

Interval Scheduling Problem

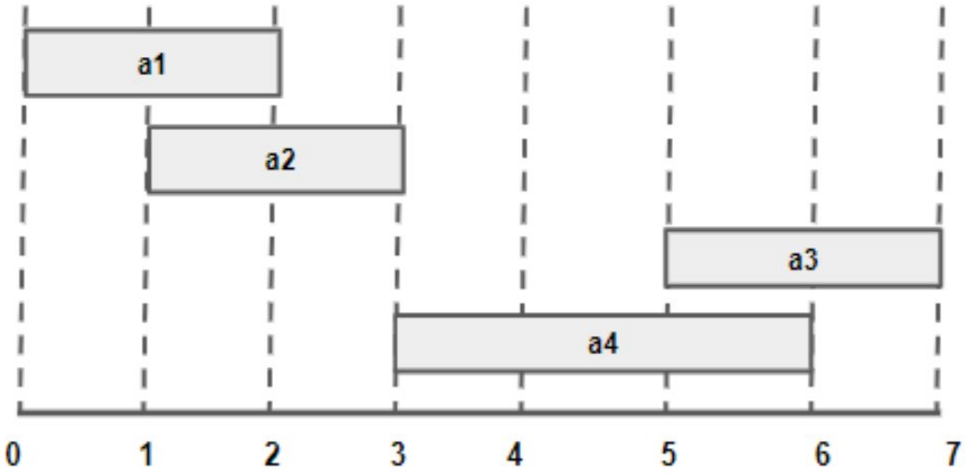
- We are given a set of job/interval requests, $S = a_1, a_2, \ldots, a_n$ that need to use some resource.
- Each request a_i has a start time s_i & finish time fi , such that $0 \leq s_i < f_i < \infty$.
- We need to allocate the resource in a compatible manner, such that the number of request scheduled is maximized.
- The resource can be used by one and only one request at any given time.

How is it different from Weighted Interval Scheduling (WIS)?

Compatible Jobs - Recall

Two requests a_i and a_j are said to be compatible, if the interval they span do not overlap i.e. $f_i \leq s_j$ or $f_j \leq s_i$

Example:



Some possible Greedy Solutions

Can you suggest some greedy solutions?

Some possible Greedy Solutions

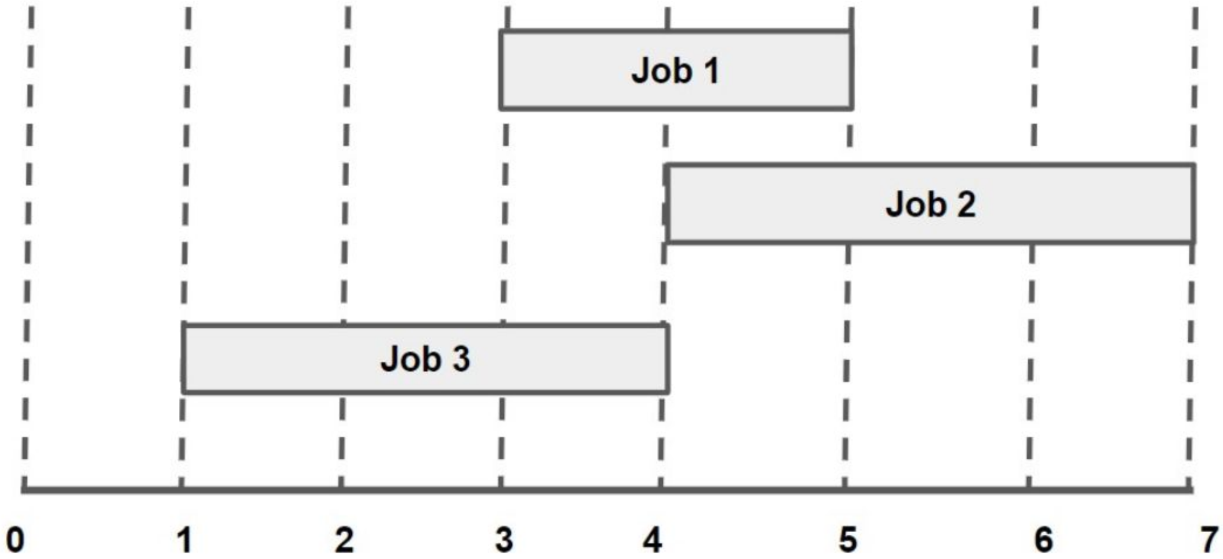
- Shortest job first
- In the order of increasing starting times
- In the order of increasing finishing times

Approach 1: Shortest Job First

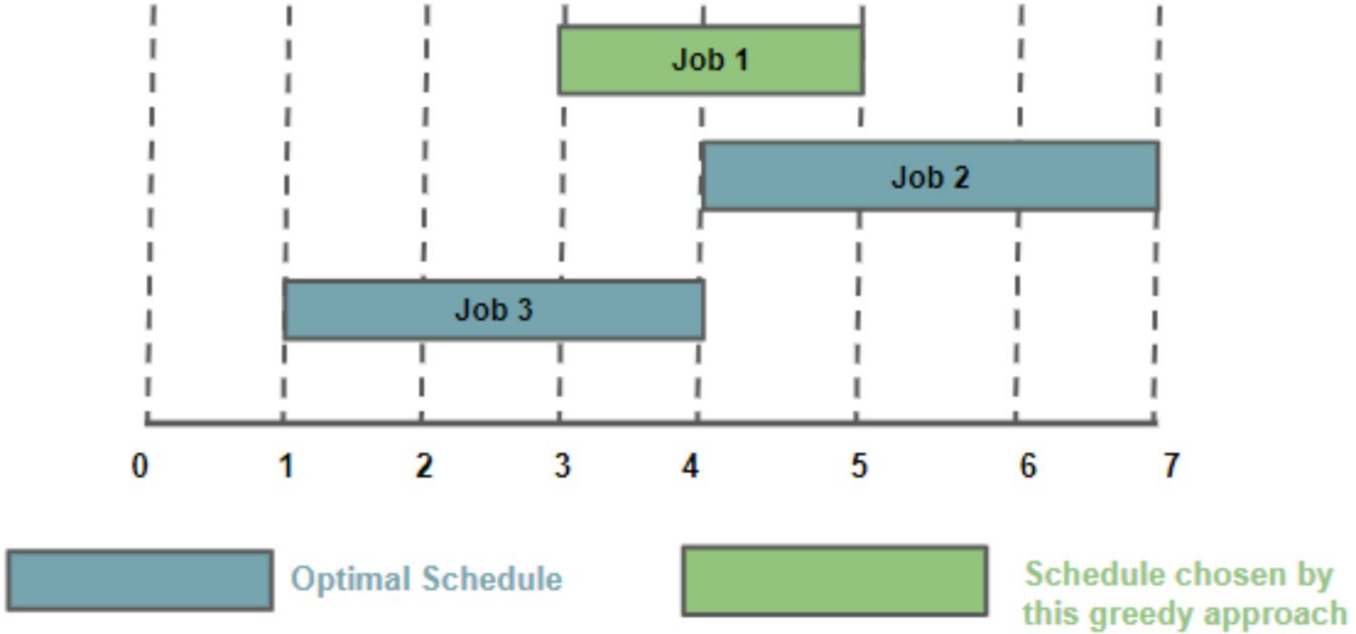
- Will it work? Give example
- Will it not work? Give example.

Write examples and then proceed.

Approach 1: Shortest Job First (2)



Approach 1: Shortest Job First (3)

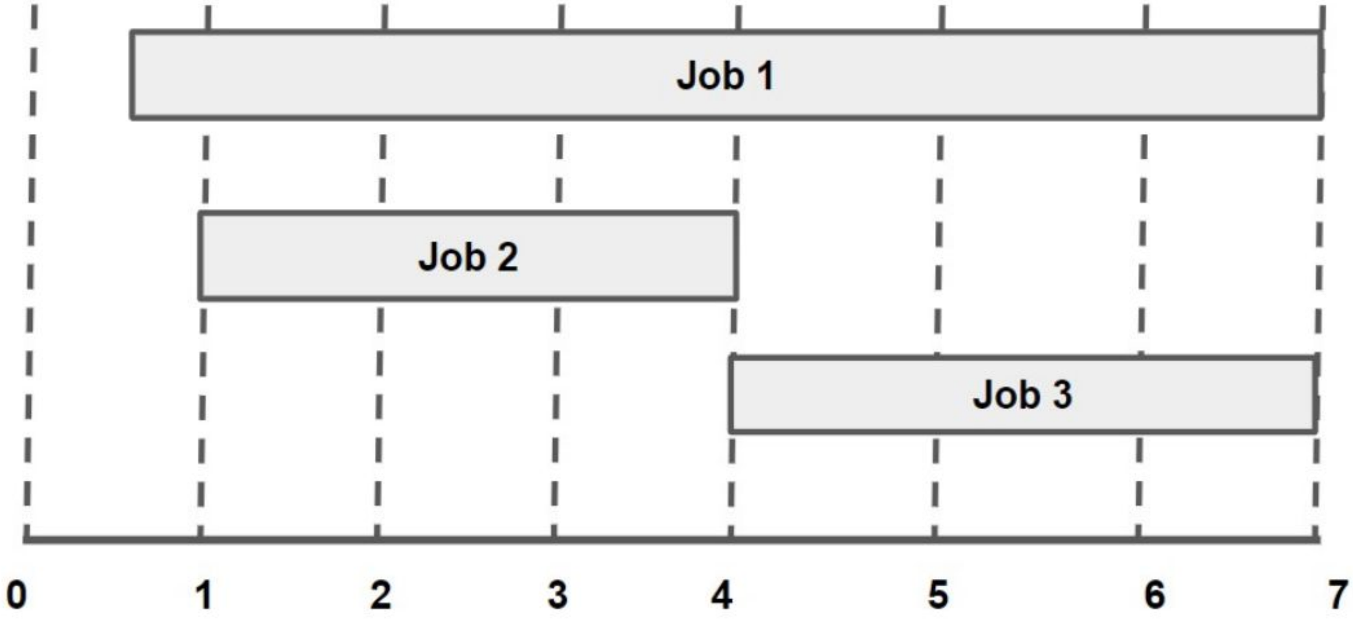


Approach 2: Increasing order of start times

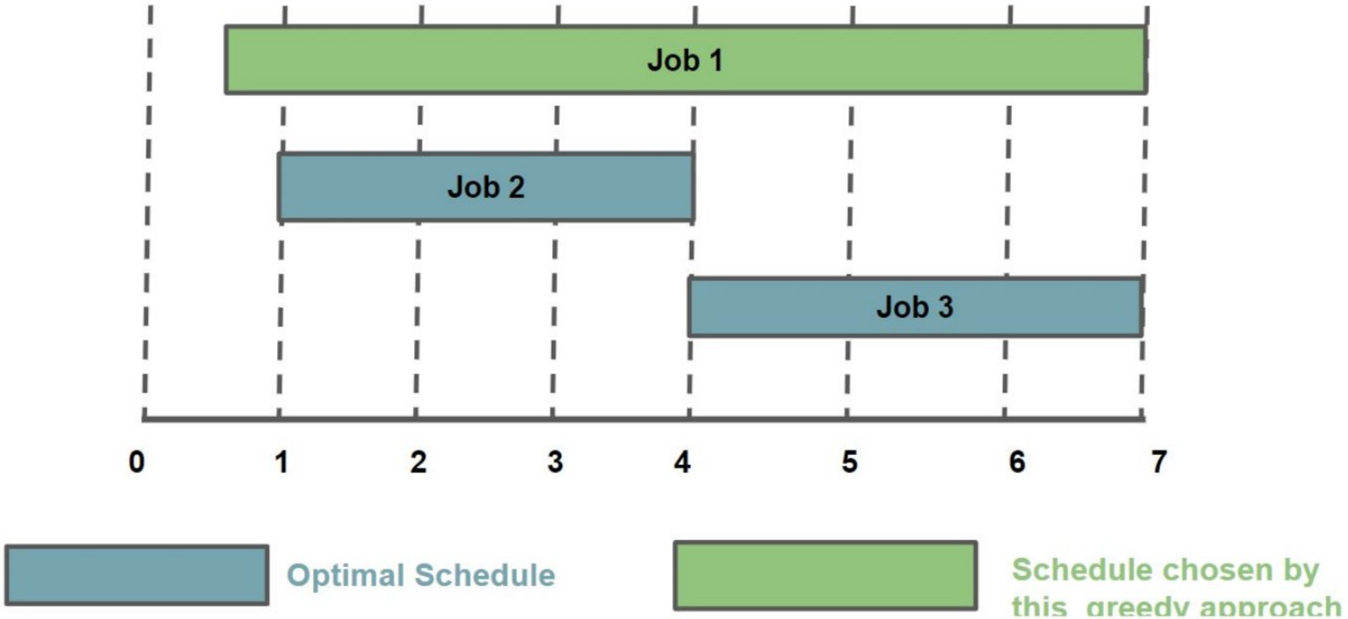
- Will it work? Give example
- Will it not work? Give example

Write examples and then proceed.

Approach 2: Increasing order of start times(2)



Approach 2: Increasing order of start times(3)

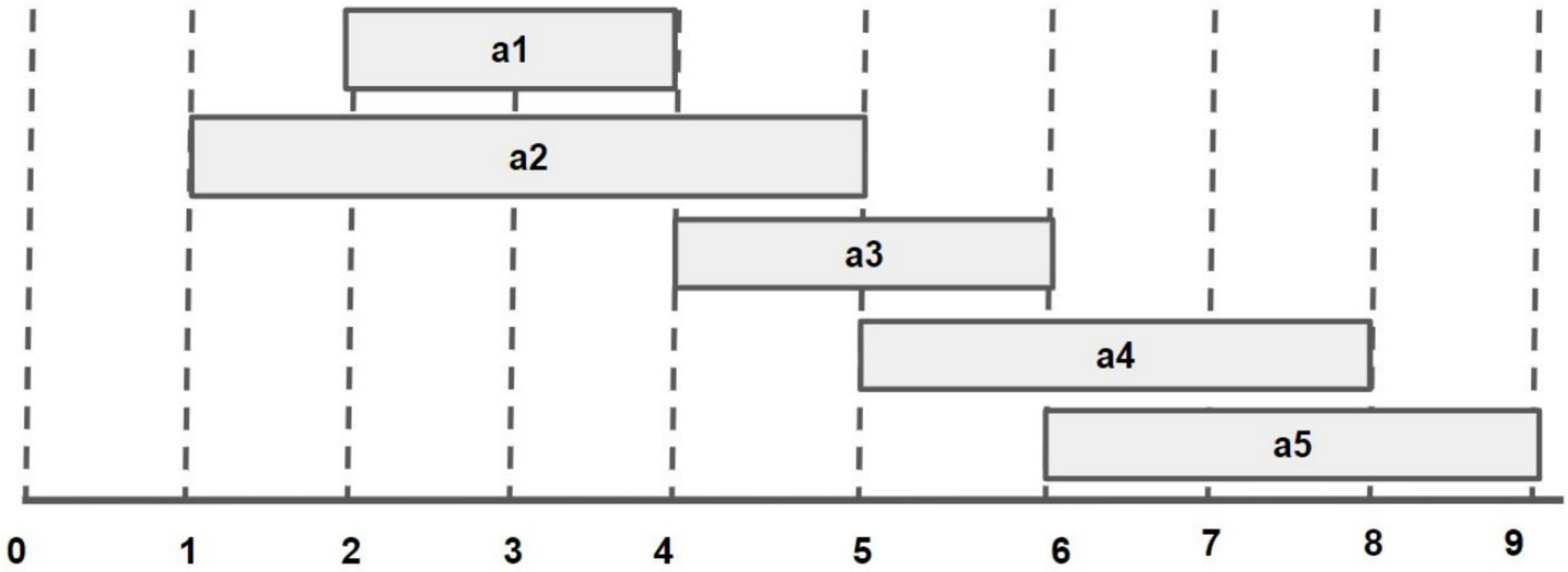


Approach 3: Increasing order of finish times

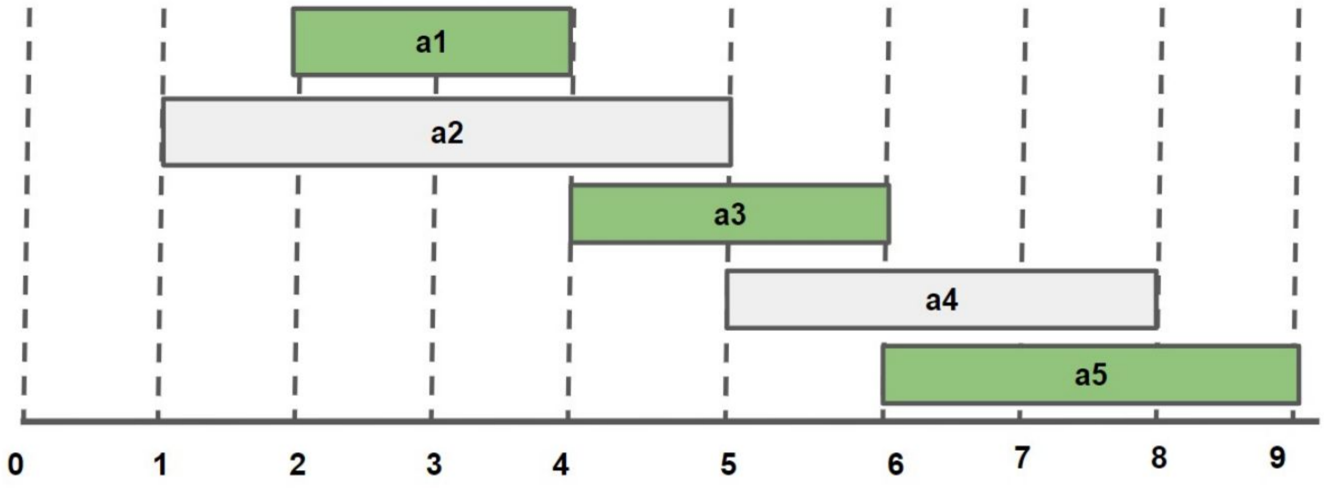
- **Will it work? Give example**
- **Will it not work? Give example**

Write examples and then proceed.

Approach 3: Increasing order of finish times(2)



Approach 3: Increasing order of finish times(2)



Algorithm

- Sort the requests in the increasing order of their finish times. Let P be the sequence so obtained.
- Consider the next request a_i in P.
  - Include a_i in the schedule S.
  - For each interval a_j that follows: Discard a_j and delete from P if it conflicts with a_i
  - Repeat until P is empty.

Proving it works!

- We will prove our claim by contradiction using "stay ahead of the optimal" strategy.
- S : My solution (in increasing order of finishing times) $i_1, i_2, \ldots, i_k$
- O : Optimal solution (in increasing order of finishing times) $j_1, j_2 \ldots, j_m$
- There might be more than one optimal solution but number of intervals is same. That is, I wish to prove that |O| = |S|

Or we would like to prove $m = k$

Claims

- **Claim 1:** (Stay ahead policy):

$f(i_r) \leq f(j_r)$ for all r (will prove later)

- **Claim 2:** $k = m$.

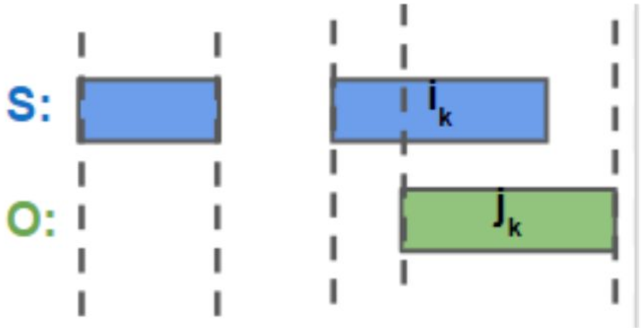
Proof: Clearly $k \leq m$. We will prove that k cannot be $< m$

(by contradiction) and hence k must be $= m$.

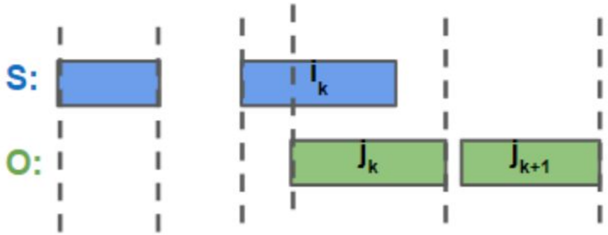
Stay ahead policy: we will compare partial solutions that the greedy algorithm constructs to initial segments of optimal solution O, and show that greedy algorithm is doing better in a step-by-step fashion.

Proof of claim 2

- Suppose $k<m$
- We have, $f(i_k) \leq f(j_k)$ . . . (by claim 1)



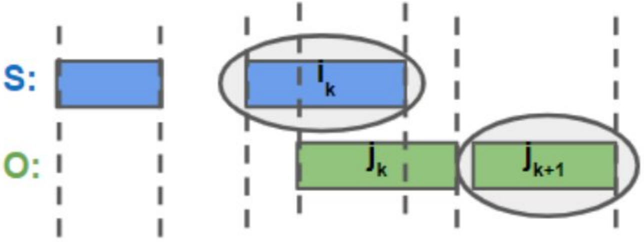
Proof of claim 2



- Suppose $k<m$
- We have, $f(i_k) \le f(j_k)$ . . . (by claim 1)
- As $k < m$, $\exists\, j_{k+1} \in O$
- Since j_{k+1} is compatible with j_k we have, $f(j_k) < s(j_{k+1}) < f(j_{k+1})$)
- Since $f(i_k) \le f(j_k)$, we have $f(i_k) < s(j_{k+1})$ and hence j_{k+1} is also compatible with i_k also.

Our algorithm would have picked j_{k+1} after i_k in our solution.

- This contradicts our assumption that i_k is the last interval in our solution.
- Thus $k = m$.



Proof of claim 1

Claim 1 (Stay ahead policy):

$H(r) : f(i_r) \leq f(j_r)$ for all r

- **Base case : $r = 1$, $f(i_1) \leq f(j_1)$ by choice of our algorithm**

Proof of claim 1

Claim 1 (Stay ahead policy):

$H(r)$: $f(i_r) \le f(j_r)$ for all r

- Base case : $r = 1$, $f(i_1) \le f(j_1)$ by choice of our algorithm
- Induction Hypothesis : Assume that $H(r)$ is true. We will prove $H(r + 1)$ is true

Since j_{r+1} is compatible with j_r and $f(i_r) \le (j_r))$ therefore j_{r+1} is also compatible with i_r also.

- Thus our algorithm had the option of picking j_{r+1} . Thus, $f(i_{r+1}) \le f(j_{r+1})$
- Hence proved.

Time Complexity

Algorithm

- Sort the requests in the increasing order of their finish times. Let P be the sequence so obtained
- Consider the next request a_i in P.
  - Include a_i in the schedule S.
  - For each interval a_j that follows: Discard a_j and delete from P if it conflicts with a_i.
  - Repeat until P is empty.

What is the associated complexity with each step?

Time Complexity

Algorithm

- Sort the requests in the increasing order of their finish times. Let P be the sequence so obtained.....O(n log n)
- Consider the next request a_i in P.
    - Include a_i in the schedule S.
    - For each interval a_j that follows: Discard a_j and delete from P if it conflicts with a_i.
    - Repeat until P is empty.

In step 2, it takes O(1) time to take decision for each request whether to keep it or discard it making a total of O(n).

Thus, total time is O(n log n)

Next Lecture

- Interval Partitioning