CSCI 323 Lecture 9 Note Monday, 03/07/2022

Announcements:

- The Lecture Recordings will be available on the following YouTube Playlists Link: https://youtube.com/playlist?list=PLZaTmV9UMKlgYpo2cAiMaEWxqyvbiXDFd

## Greedy Algorithm

References:
Algorithm Design - Chapter 4.1, 4.4

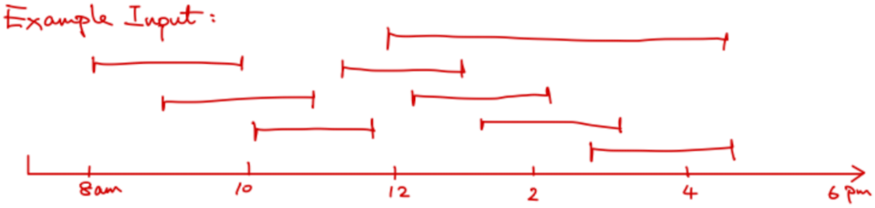
## Greedy Algorithm

- What are greedy algorithms?
  - You are given a problem and asked to find an optimal solution. Sometimes, there is a way to build a solution greedily.
  - So, you can come up with some greedy criteria and start building your solution. At the end, you're hoping that the final solution that you produce is optimal, the best solution.
  - There are very specific problems allow this to happen, which admit a greedy algorithm that will be optimal.
  - For most of problems, this is not going to be the case. However, this should be the first algorithm/attempt to try on a problem. Sometimes you can get very lucky, and your greedy algorithm turn out to be optimal.
  - Even if your (greedy) algorithm does not turn out to be optimal in many cases, it turns out to be approximately optimal.

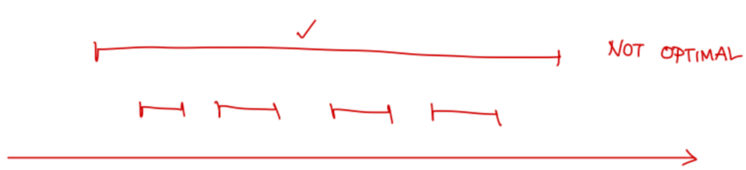
## Interval Scheduling [4.1]

- You have a resource – a lecture room or a supercomputer – and many people request to use the resource for periods of time. A request takes the form: Can I reserve the resource starting at time $s$, until time $f$? We will assume that the resource can be used by at most one person at a time. A scheduler wants to accept a subset of these requests, rejecting all others, so that the accepted requests do not overlap in time. The goal is to maximize the number of requests accepted.
- Input: Set of requests $\{1, 2, \dots, n\}$.
  - Each request is a time interval $[s(i), f(i)]$.
    - $s(i)$ is the starting time of the $i^{th}$ request.
    - $f(i)$ is the finishing time of the $i^{th}$ request.
    - $f(i) > s(i)$.
- A subset of $\{1, 2, \dots, n\}$ is *compatible* if no two requests in the subset overlap in time.
- Goal: Output a compatible subset of maximum size.

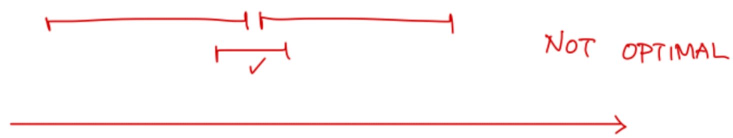
Example Input:



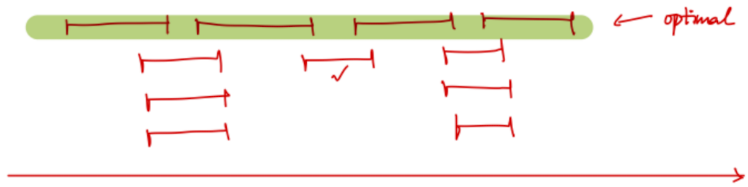

- Rule 1: Select the request that starts the earliest.



- Rule 2: Accept the shortest request first.

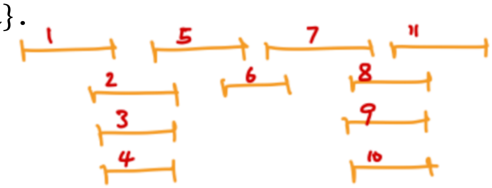



- Rule 3: Select the request with fewest conflicts.

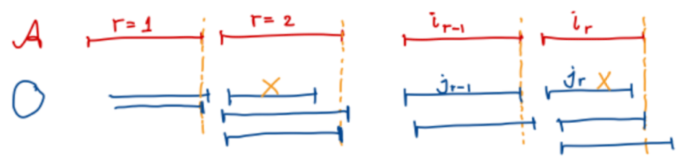



Best Rule 3 - 3 requests NOT OPTIMAL

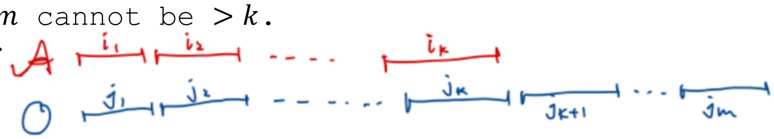
- Rule 4: Select the request that ends earliest.
    - Theorem: Rule 4 gives us the optimal solution.
    - Can be implemented in $O(n \log n)$ time, page 121.
    - Proof of optimality: "Staying ahead"
      By contradiction:
        - $A$ is the set of requests selected by Rule 4, which selects $k \le n$ out of the $n$ requests, $total = \{1, 2, \dots, n\}$.
        - $A = \{i_1, i_2, i_3, \dots, i_k\}$, e.x.: $A = \{1, 5, 7, 11\}$.



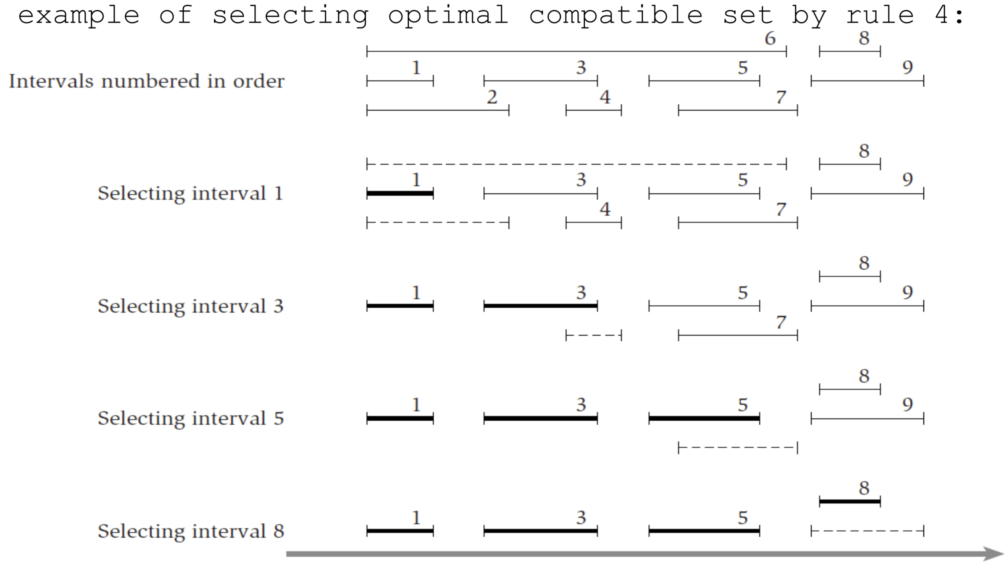

        - Also assume (for the sake of contradiction) that Rule 4 is not optimal. Let $O$ be an optimal set of requests. $O = \{j_1, j_2, \dots, j_m\}$, where $m > k$.



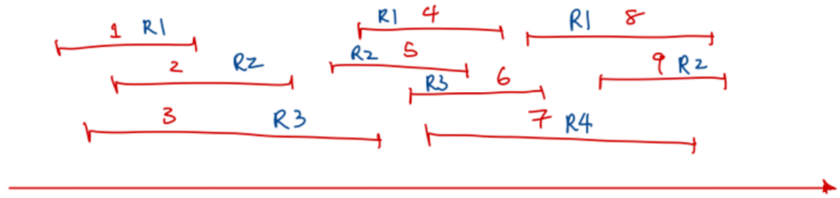

        - Observations:
          1) $f(i_1) \le f(j_1)$
          2) For all $r \le k$, $f(i_r) \le f(j_r)$
        - We want to prove that $m$ cannot be $> k$.
        - Look at our last job →




        - Then $A$ could also choose from $\{j_{k+1}, \dots, j_m\}$ and not stop at $i_k$. ← Contradiction!
    - An example of selecting optimal compatible set by rule 4:



- A Related Problem: Scheduling All Intervals
    - How many rooms are needed to schedule all intervals?
    - # rooms ≥ max depth
    - Optimal solution: # rooms = max depth is enough.

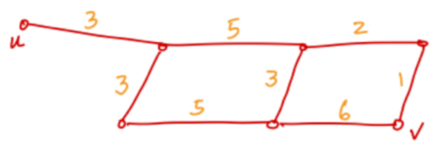



    - Algorithm:
        - Let $R = \{1, 2, \ldots, d\}$, where $d$ is the max depth/overlaps.
        - For each interval $I_i$ assign a number from $R$, that's distinct from all intervals $I_j$ that overlaps with (*I_i*).
    - Read 4.1

**Shortest Path – Dijkstra's Algorithm [4.4]**

- Given a weighted (directed) graph $G = (V, E)$, where each edge $(u, v) \in E$ has a length/weight $l_{(u,v)}$.


- Goal to find the shortest path between two vertices.
    - If all weights on the edges = 1, shortest = least number of edges.
    - $BFS(u)$ solve the problem.
    - What if weights are not all 1?
      (All weights are positive. )




    - If the weights are negative,
      Dijkstra's algorithm cannot solve the problem.
- Dijkstra's algorithm from a source $u$, returns
    1) the length of the shortest path from $u$ to any vertex $v$.
    2) the shortest path from $u$ to $v$, for any $v$.

**What to expect or prepare for the next class:**

- Shortest Path – Dijkstra's Algorithm
- Minimum Spanning Tree

**Reading Assignment**

Algorithm Design: 4.1, 4.4