UF Herbert Wertheim College of Engineering
Department of Computer & Information Science & Engineering
UNIVERSITY of FLORIDA

Greedy – Part I

Optimization Problems

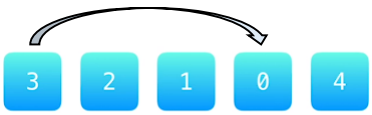
- Problems of finding the best solution from all feasible solutions
  - Feasible solution: any solution that satisfies some constraints
  - Optimal solution: a feasible solution that maximizes or minimizes the objective function
- A “greedy algorithm” sometimes works well for optimization problems
  - Always make the choice that looks best at the moment, regardless future consequences
  - The hope: by choosing a local optimum at each step, you will end up at a global optimum
  - Greedy algorithm will get a solution, but may not be the optimum solution
  - E.g.
    - Get on the shortest road first
    - Assign work to the fastest worker
  - Commonly used to find approximations for NP-Complete problems
- Greedy algorithms tend to be easier to code

Example: Counting Money

- Suppose you want to count out a certain amount of money, using the fewest possible bills and coins
- A greedy algorithm to do this would be:

  At each step, take the largest possible bill or coin that does not overshoot
  - Example: To make $7.26, you can choose:
    - a $5 bill
    - two $1 bills, to make $7
    - a 25¢ coin, to make $7.25
    - a 1¢ coin, to make $7.26
  - For US money, the greedy algorithm always gives the optimum solution
- What if you have coins value 1¢, 7¢, and 10¢?
  - The greedy algorithm results in a solution, but not an optimal solution

Leetcode 55. Jump Game




```
for i = 0 to n-1
   if (can not reach i)
      return false
   else
      update the farthest position can reach
return true
```


```
set target as n-1
for i = n-2 to 0
   if (i can reach target)
      update target as i
return true if target is 0
```

55. Jump Game

Medium 17.8K 996

Companies

You are given an integer array `nums`. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return `true` *if you can reach the last index, or* `false` *otherwise.*

Example 1:

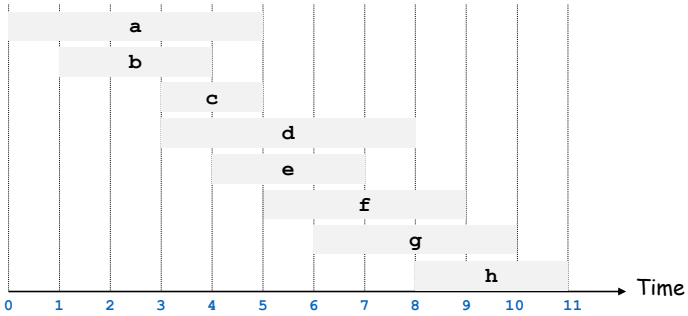
```
Input: nums = [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.
```

Example 2:

```
Input: nums = [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.
```

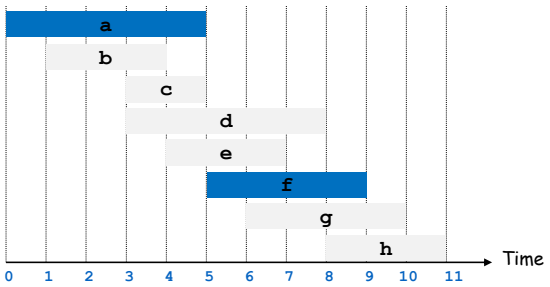
Meeting Scheduling

- Meeting scheduling with only one meeting room
  - Meeting *i* starts at s_i and finishes at f_i
  - Two meetings are compatible if they don't overlap
  - Goal: find maximum number of mutually compatible meetings

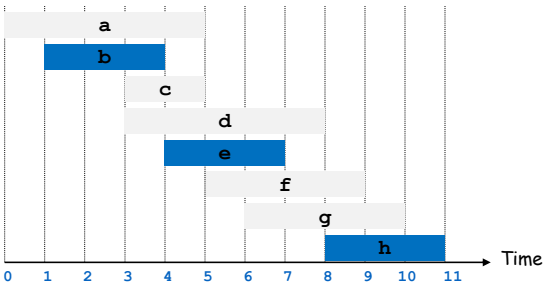


Meeting Scheduling – greedy

- Pick meetings with earliest start time

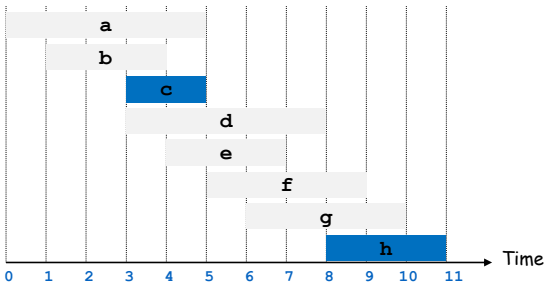


- Pick meetings with earliest finishing time

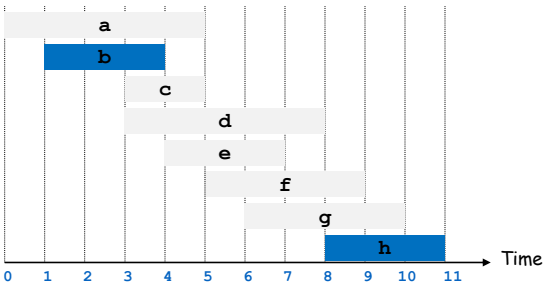


Meeting Scheduling – greedy

- Pick the shortest meetings



- Pick meetings with fewest conflicts



Meeting Scheduling – Algorithm

- Consider meetings in increasing order of finish time. Take each meeting provided it's compatible with the ones already taken

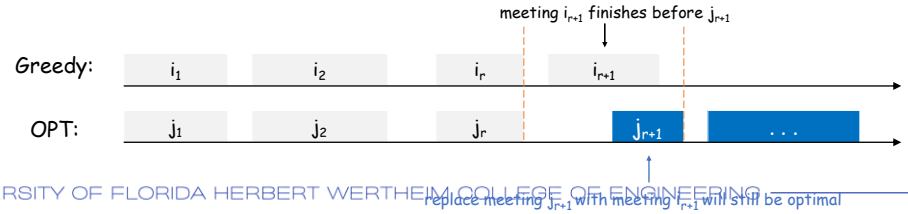
```
Sort meetings by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

A ← ϕ // set of meetings selected
for i = 1:n
   if (meeting i compatible with meetings in A)
      A ← A ∪ {i}
return A
```

- Time complexity
  - Sorting step: O(n log n)
  - Compatible step: O(n)
    - Assume meeting *j* be the one which was added last to A
    - Meeting i is compatible with A if $s_i \geq f_j$

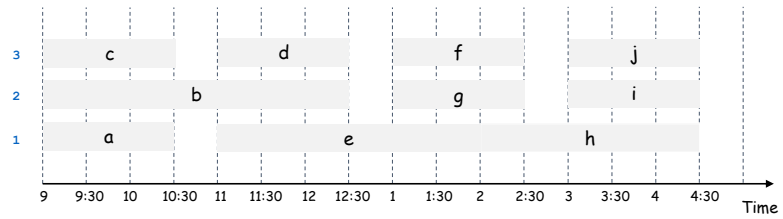
Meeting Scheduling - More

- Why this is greedy?
  - Maximizes the amount of unscheduled time remaining
  - Greedy: leaves as much opportunity as possible for the remaining activities to be scheduled
- Assume greedy is not optimal
  - Let $i_1, i_2, \ldots i_k$ denote set of meetings scheduled by greedy
  - Let $j_1, j_2, \ldots j_m$ denote set of meetings in the optimal solution with $i_1 = j_1$, $i_2 = j_2$, ..., $i_r = j_r$ for the largest possible value of r.



Multiple Meeting Rooms

- What if we have multiple meeting rooms?
- Assumption: all meeting rooms are equivalent
- Goal: find minimum number of meeting rooms to hold all meetings so that no two occur at the same time in the same room



Multiple Meeting Rooms – Algorithm

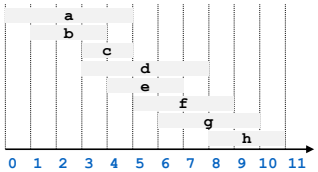
- Sort meetings in increasing order of start time
- From the first meeting to the last
  - Assign the meeting to any meeting room which is vacant
  - If no meeting room is available, get a new meeting room

```
Sort intervals by starting time so that s1 ≤ s2 ≤ ... ≤ sn

d ← 0
for i = 1 to n
   if (room j is free now)            Use priority queue
      schedule meeting i in room j
   else
      allocate a new room d + 1
      schedule meeting i in room d + 1
      d ← d + 1
```


Multiple Meeting Rooms – Proof

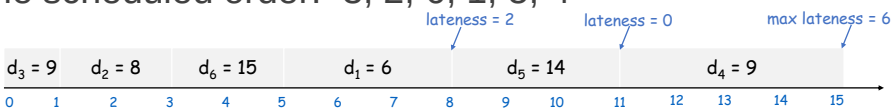
- Key observation
  - Number of meeting rooms needed ≥ number of meetings hold at the same time
- Let d = number of meeting rooms that the greedy algorithm allocates
- Meeting room d is opened because we needed to schedule meeting j when all other d-1 are not vacant
- These d-1 meetings each end after s_j (because of incompatible)
- Since we sorted by start time, all these incompatibilities are caused by meetings that start no later than s_j
- Thus, we have d meetings overlapping at time s_j



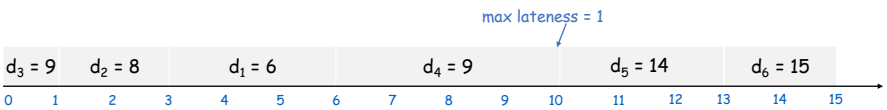
Minimizing Lateness

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

- Details
  - Single meeting room: one meeting at a time
  - Meeting j requires t_j units of time and is required to be finished by time d_j.
  - If j starts at time s_j, it finishes at time $f_j = s_j + t_j$.
  - Lateness definition: $\ell_j = \max \{0, f_j - d_j\}$.
  - Goal: schedule all meetings to minimize the maximum lateness $L = \max \ell_j$.
- E.g., if the scheduled order: 3, 2, 6, 1, 5, 4

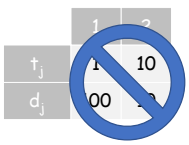


- Best:



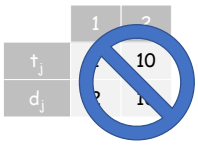
Minimizing Lateness: Greedy Approach

- Consider meetings in ascending order of meeting time t_j.
  - Shortest meeting time first

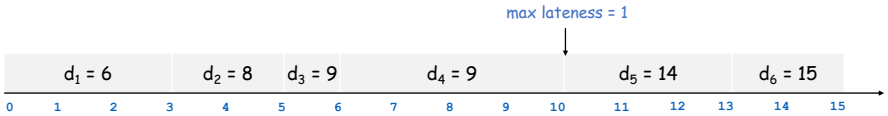


	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

- Consider meetings in ascending order of slack $d_j - t_j$
  - Smallest slack



- Earliest deadline first



Minimizing Lateness: Greedy Approach

- Earliest deadline first

```
Sort n meetings by deadline so that d_1 ≤ d_2 ≤ … ≤ d_n

t ← 0
for j = 1 to n
   Assign job j to interval [t, t + t_j]
   s_j ← t, f_j ← t + t_j
   t ← t + t_j
output intervals [s_j, f_j]
```

- Proof
  - There exists an optimal schedule with no idle time
  - All schedules with no inversion and no idle time have the same maximum lateness
    - Given a schedule S, an inversion is a pair of jobs i and j such that $d_i < d_j$ but j scheduled before i

Stall Reservations

- https://vjudge.net/problem/POJ-3190

Oh those picky N (1 <= N <= 50,000) cows! They are so picky that each one will only be milked over some precise time interval A..B (1 <= A <= B <= 1,000,000), which includes both times A and B. Obviously, FJ must create a reservation system to determine which stall each cow can be assigned for her milking time. Of course, no cow will share such a private moment with other cows.

Help FJ by determining:

- The minimum number of stalls required in the barn so that each cow can have her private milking period
- An assignment of cows to these stalls over time

Many answers are correct for each test dataset; a program will grade your answer.

UVa 11292 Dragon of Loowater

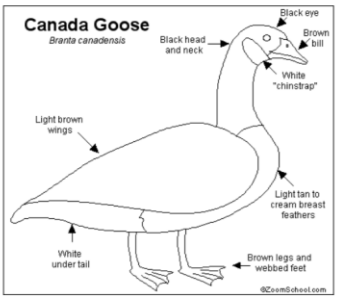
- Approach 1
  - For each head, find the shortest knight whose height is taller than the diameter of the head
    - Can use binary search
- Approach 2
  - Sort the diameter of the heads
  - Sort the height of the knight
  - Set knight index j as 0
  - For each diameter of the heads
    - If knight j can do the job, j++
    - If no more knight available, print “Loowater is doomed!”
  - Print the cost

11292 The Dragon of Loowater

Once upon a time, in the Kingdom of Loowater, a minor nuisance turned into a major problem.

The shores of Rellau Creek in central Loowater had always been a prime breeding ground for geese. Due to the lack of predators, the geese population was out of control. The people of Loowater mostly kept clear of the geese. Occasionally, a goose would attack one of the people, and perhaps bite off a finger or two, but in general, the people tolerated the geese as a minor nuisance.

One day, a freak mutation occurred, and one of the geese spawned a multi-headed fire-breathing dragon. When the dragon grew up, he threatened to burn the Kingdom of Loowater to a crisp. Loowater had a major problem. The king was alarmed, and called on his knights to slay the dragon and save the kingdom.



The knights explained: “To slay the dragon, we must chop off all its heads. Each knight can chop off one of the dragon’s heads. The heads of the dragon are of different sizes. In order to chop off a head, a knight must be at least as tall as the diameter of the head. The knights’ union demands that for chopping off a head, a knight must be paid a wage equal to one gold coin for each centimetre of the knight’s height.”

Would there be enough knights to defeat the dragon? The king called on his advisors to help him decide how many and which knights to hire. After having lost a lot of money building Mir Park, the king wanted to minimize the expense of slaying the dragon. As one of the advisors, your job was to help the king. You took it very seriously: if you failed, you and the whole kingdom would be burnt to a crisp!

Greedy Procedure

- Choose the one that looks best right now
  - Make a locally optimal choice in hope of getting a globally optimal solution
- Greedy algorithms don’t always yield an optimal solution
- Keys:
  - Select
    - A greedy procedure, based on a given objective function, which selects input from A, removes it and assigns its value to x
  - Feasible
    - A boolean function to decide if x can be included into solution vector (without violating any given constraint)

```
procedure Greedy (A, n)
  begin
    solution ← Ø;
    for i ← 1 to n do
      x ← Select (A);   // based on the objective
                        // function
      if Feasible (solution, x),
        then solution ← Union (solution, x);
  end;
```