

CS 401: Computer Algorithm I

Greedy Algorithms: Interval Scheduling

Xiaorui Sun

Stuff

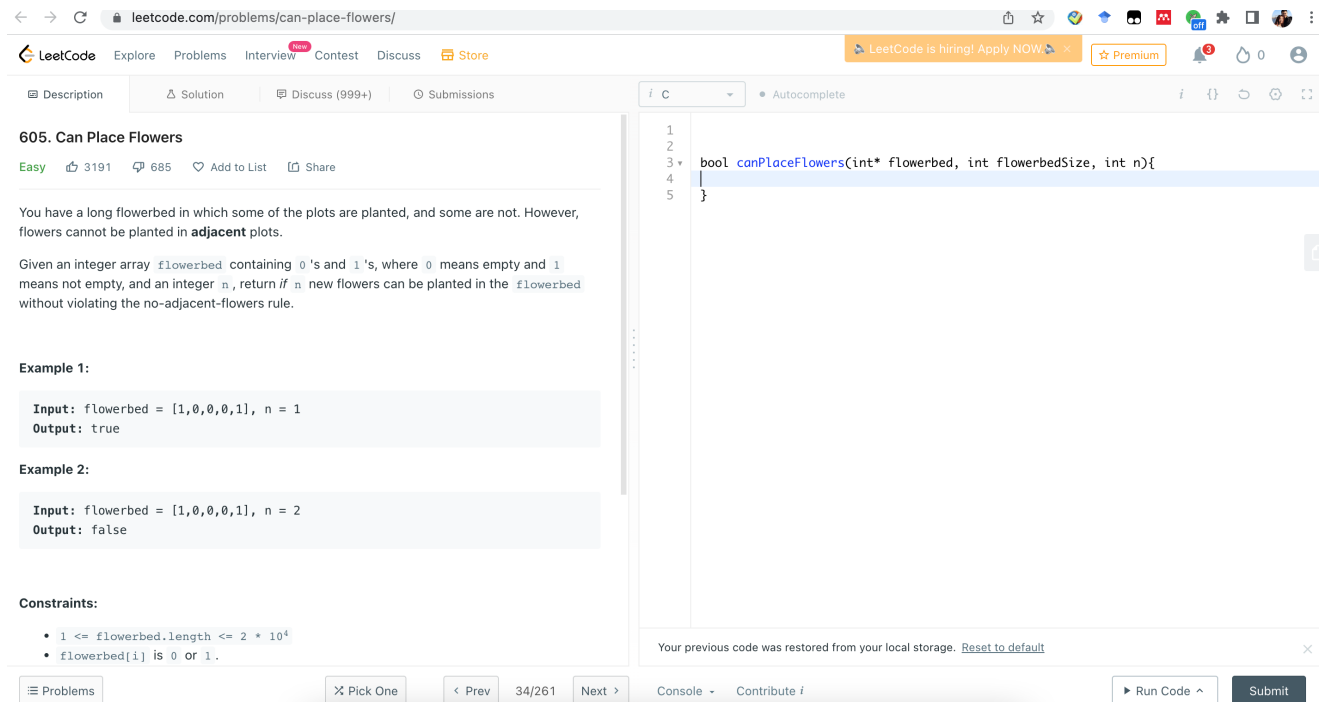
Homework 1 due tomorrow 11:59pm

Homework 2 will be released tomorrow, due Feb 23 11:59pm

Homework 2

Programming homework on Leetcode

- Register a Leetcode account (free), and programming on Leetcode
- You can use any programming language
- Submit your code to gradescope
- Score for each problem is proportional to the test cases on Leetcode you can pass (if you can pass all, you get full score on the problem)



The screenshot shows the LeetCode website interface for the problem "605. Can Place Flowers". The problem is categorized as "Easy" and has 3191 likes and 685 dislikes. The description states: "You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots. Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule." Two examples are provided: Example 1 with input [1,0,0,0,1] and n=1, output true; Example 2 with input [1,0,0,0,1] and n=2, output false. Constraints include 1 ≤ flowerbed.length ≤ 2 * 10⁴ and flowerbed[i] is 0 or 1. The code editor on the right shows a C++ function signature: `bool canPlaceFlowers(int* flowerbed, int flowerbedSize, int n){`. The bottom of the page shows navigation buttons like "Problems", "Pick One", "Prev", "Next", "Run Code", and "Submit".

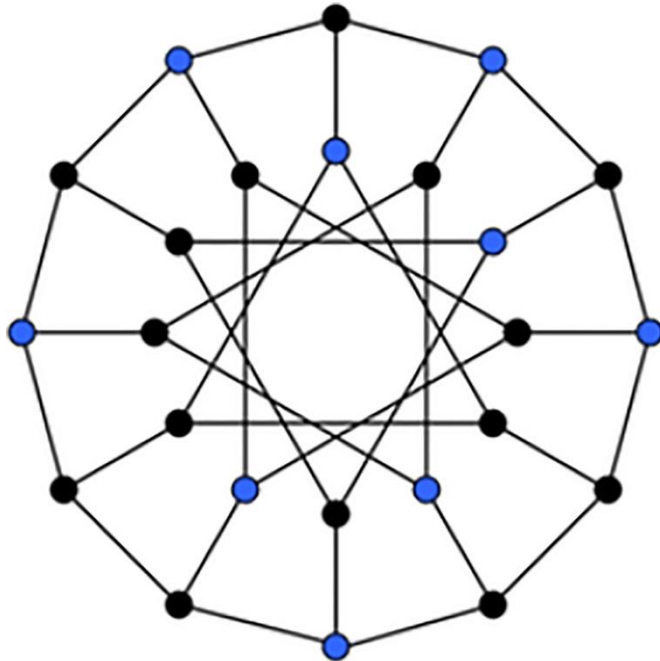
Greedy Algorithms

- High level idea
 - Solution is built in small steps
 - Decisions on how to build the solution are made to maximize some criterion without looking to the future
 - Want the 'best' current partial solution as if the current step were the last step

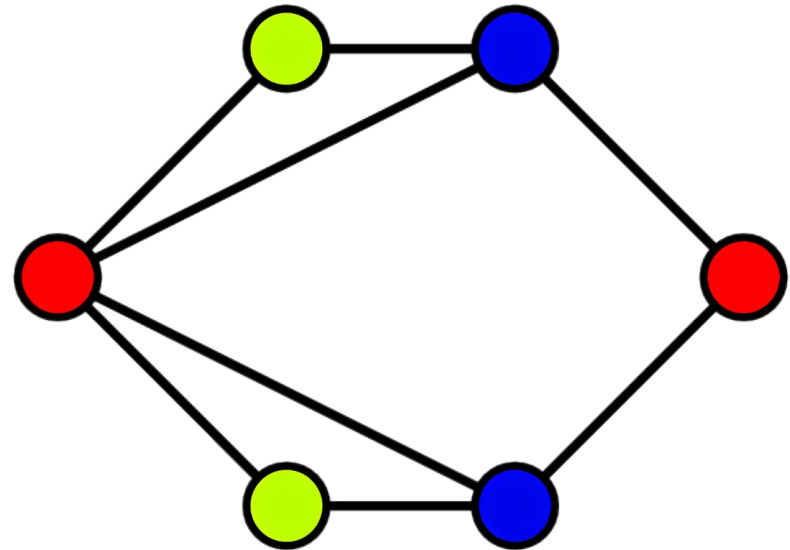
Greedy Algorithms

- High level idea
 - Solution is built in small steps

Independent set



Vertex coloring

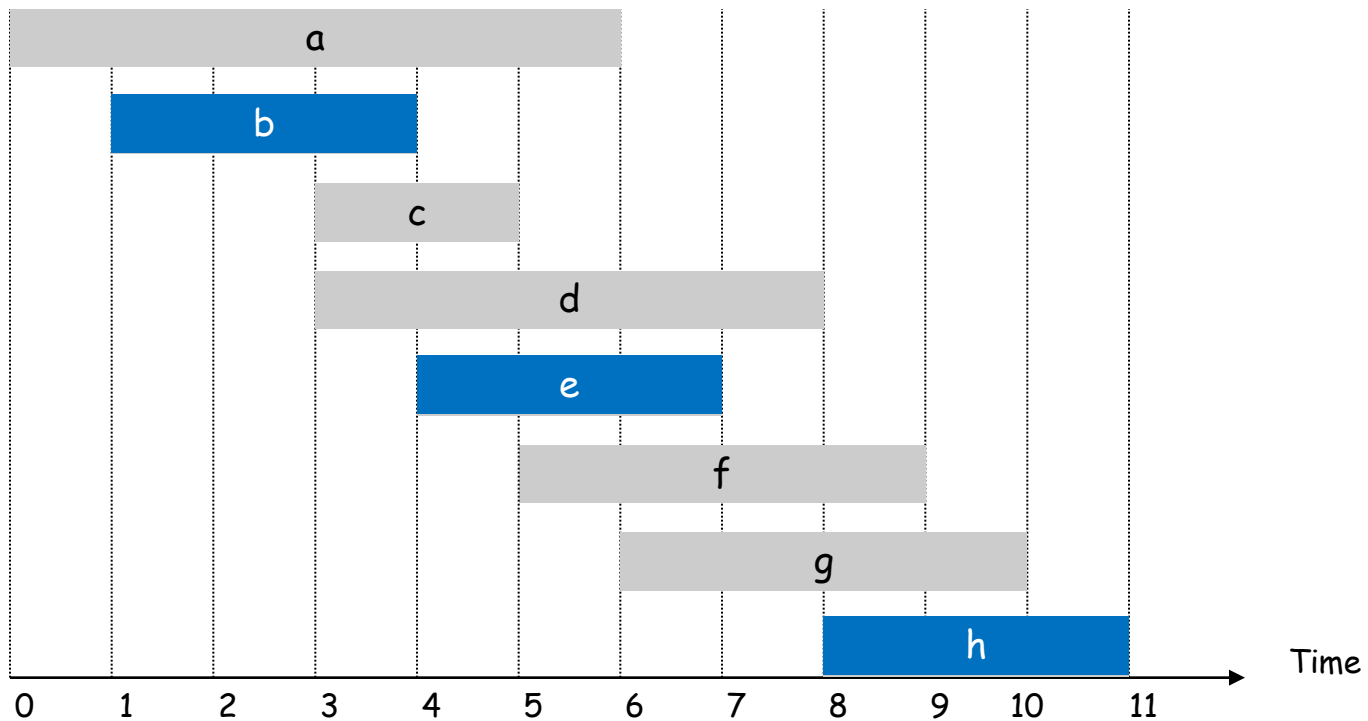


Greedy Algorithms

- High level idea
 - Solution is built in small steps
 - Decisions on how to build the solution are made to maximize some criterion without looking to the future
 - Want the ‘best’ current partial solution as if the current step were the last step
- General Recipe:
 - Order the input in a good way
 - Go over the input one by one and make decision on each input with a good strategy

Interval Scheduling

- Job j starts at $s(j)$ and finishes at $f(j)$.
- Two jobs **compatible** if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Greedy Strategy

Sort the jobs in **some** order. Go over the jobs and take jobs that are compatible with the previous jobs already taken.

Main question:

- What order?
- Does it give the optimum answer?
- Why?

Possible Approaches for Inter Sched

Sort the jobs in **some** order . Go over the jobs and take jobs that are compatible with the previous jobs already taken.

[Shortest interval] Consider jobs in ascending order of interval length $f(j) - s(j)$.

[Earliest start time] Consider jobs in ascending order of start time $s(j)$.

[Earliest finish time] Consider jobs in ascending order of finish time $f(j)$.

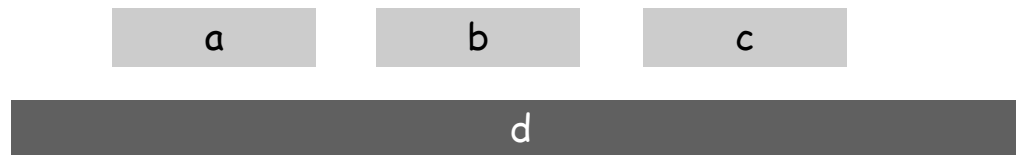
Possible Approaches for Inter Sched

Sort the jobs in **some** order . Go over the jobs and take jobs that are compatible with the previous jobs already taken.

[Shortest interval] Consider jobs in ascending order of interval length $f(j) - s(j)$.



[Earliest start time] Consider jobs in ascending order of start time $s(j)$.



[Earliest finish time] Consider jobs in ascending order of finish time $f(j)$.

Greedy Alg: Earliest Finish Time

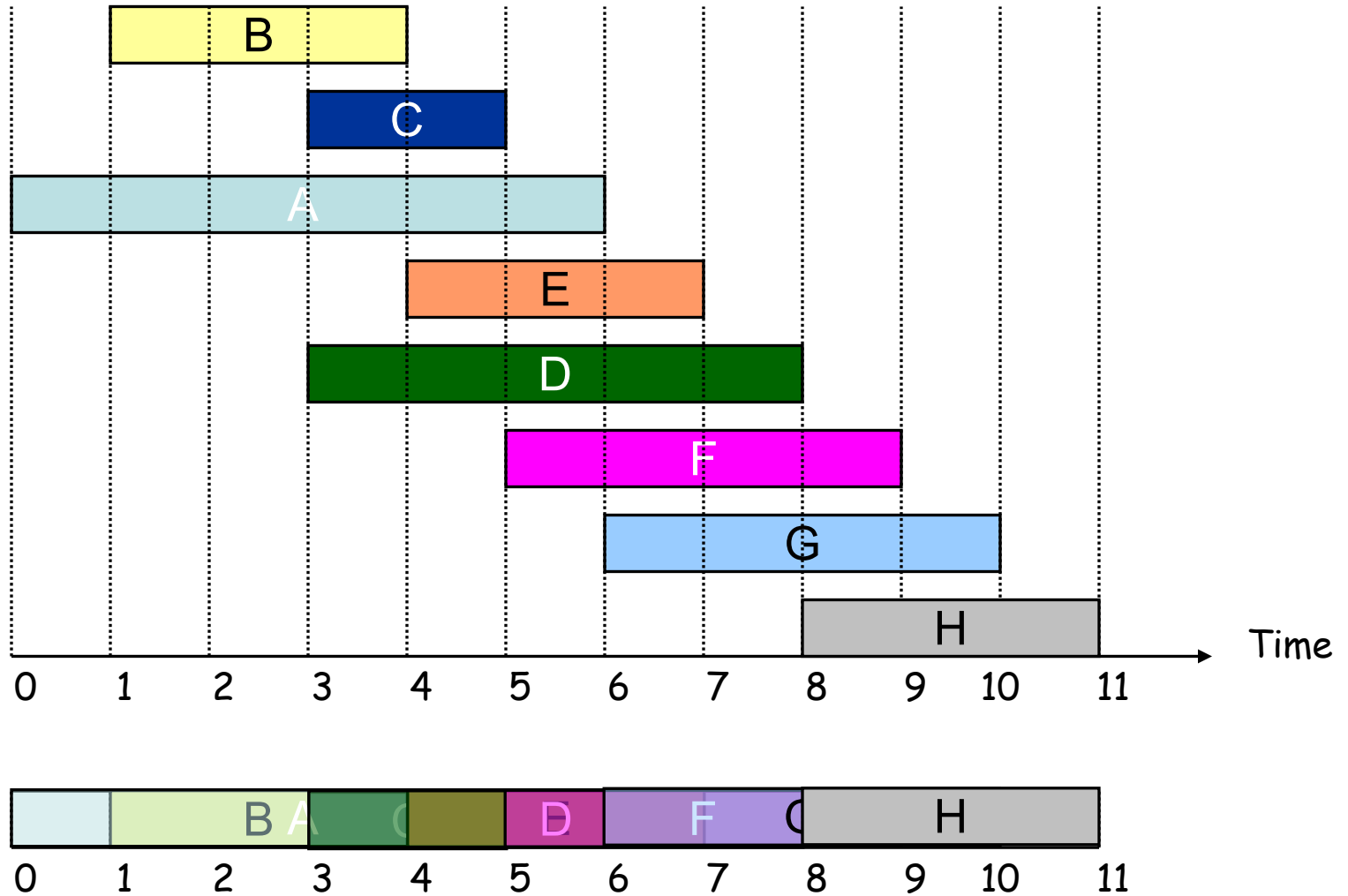
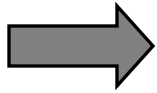
Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

```
Sort jobs by finish times so that  $f(1) \leq f(2) \leq \dots \leq f(n)$ .  
 $A \leftarrow \emptyset$   
for  $j = 1$  to  $n$  {  
    if (job  $j$  compatible with  $A$ )  
         $A \leftarrow A \cup \{j\}$   
}  
return  $A$ 
```

Implementation. $O(n \log n)$.

- Remember job j^* that was added last to A .
- Job j is compatible with A if $s(j) \geq f(j^*)$.

Greedy Alg: Example



Correctness

- The output is compatible. (This is by construction.)

How to show it gives maximum number of jobs?

Let i_1, i_2, i_3, \dots be jobs picked by greedy (ordered by finish time)

Let j_1, j_2, j_3, \dots be an optimal solution (ordered by finish time)

How about proving $i_k = j_k$ for all k ?

No, there can be multiple optimal solutions.

Idea: Prove that greedy outputs the “best” optimal solution.

Given two compatible orders, which is better?

The one finish earlier.

How to prove greedy gives the “best”?

Induction: it gives the “best” during every iteration.

Greedy stays ahead: At each step any other solution has a worse value for some criterion that eventually implies optimality

This example: criterion = finish time

Theorem: Greedy algorithm is optimal.

Proof: (technique: “Greedy stays ahead”)

Let $i_1, i_2, i_3, \dots, i_k$ be jobs picked by greedy, $j_1, j_2, j_3, \dots, j_m$ those in some optimal solution in order.

We show $f(i_r) \leq f(j_r)$ for all r , by induction on r .

Base Case: i_1 chosen to have min finish time, so $f(i_1) \leq f(j_1)$.

IH: $f(i_r) \leq f(j_r)$ for some r

IS: Since $f(i_r) \leq f(j_r) \leq s(j_{r+1})$, j_{r+1} is among the candidates considered by greedy when it picked i_{r+1} , & it picks min finish, so $f(i_{r+1}) \leq f(j_{r+1})$

Observe that we must have $k \geq m$, else j_{k+1} is among (nonempty) set of candidates for i_{k+1} .

Lesson

Order is important for greedy algorithms

- In general, the order gives priorities to different elements (the most important element is ordered first)
- This example: the job can be finished earliest is the most important job because finishing this job gives more freedom to finish other jobs
- If you want to solve a problem by greedy, first think about what is the “right” order of the elements

Greedy stays ahead

- A useful strategy to argue why the solution is the best