Greedy Algorithms

Alex Li

February 2019

1 Introduction

Today we will discuss greedy algorithms. Greedy algorithms are a broad topic so it's difficult to have a systematic approach to solving them. Many greedy algorithms you would be able to think of yourself without even knowing what the word 'greedy' means. However, knowing what the word means will be useful in helping you describe your approach to others and maybe even to your own brain.
Q1: What is the definition of a greedy algorithm?
A: A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum.
Example: Find path from top-left to bottom-right with maximum sum

$$\begin{bmatrix} St & 10 & 6 \\ 2 & 3 & 4 \\ 100 & 5 & End \end{bmatrix}$$

Today we will go over 2 problems with greedy solutions so you hopefully start to get a feel for when a problem has a greedy solution. They will also have some other random stuff in them. We won't go over classical algorithms like Kruskal's since you'll learn them eventually.

2 Problem 1: Missing Numbers

https://codeforces.com/problemset/problem/1081/E
Given the even-indexed elements of an array of length $< 10^5$, fill in the odd elements such that

$$\forall i \ \exists k \ s.t. \ \sum_{1}^{i} a_i = k^2$$

All array elements are less than 10^5. Suppose that it is possible on any input you are given. (though it isn't in general) Examples:
? 5 ? 11 ? 44

? 15 ? 19
Solutions:
4 5 16 11 64 44
1 15 65 19 or 49 15 17 19
Seems hard, htf do you even start!!! Hint: the lecture is about being greedy. What do you wish were true?
Greedy solutions work by choosing the best at each step, so what is a first step we can do that will be ok later for sure?
Key observation: We can go from left to right, always choosing the smallest square that works at each step.
Proof: When try to decide the value of a new element a_i. Call the sum of all elements previously in the array S. To satisfy the next equality, we want $S+a_i$ to be a perfect square, and $S + a_i + a_{i+1}$ to be a perfect square. If these equations can be satisfied for a particular value of S, they can be satisfied by any value of S less than that value, since we have control over a_i. Hence, if we keep S as small as possible at each step, the amount of possibilities at the next step will be maximal. (2 example is helpful)
When implementing, only check squares greater than the last square so it's not too slow.

3 Problem 2: Present

https://codeforces.com/problemset/problem/460/C
Given $n < 10^5$ flowers with known heights, you can water a consecutive row of $k <= n$ flowers to increase their heights by 1. What is the maximum height of the shortest flower you can achieve by performing this watering operation $m < 10^5$ times?
Example: $k = 3, m = 4$
5 4 2 8 4 4 6, answer = 5
What is the answer if $k = 1$ (4)? $k = 4$ (6)?
What is the strategy for $k = 1$? (This is greedy)
The strategy for $k > 1$ is harder. We know we can water the lowest one at each step, but it's unclear where to place the interval. ... Instead of asking what the maximum is, ask "can we achieve a maximum of M?"
Suppose we can answer that question. How is it helpful?
Now we sweep through the interval and choose greedily. Show on example.
Proof of accuracy: At each step, we water the leftmost plant with height less than M. This move is necessary to do eventually. Since we are only trying to get every plant to be above height M, there is no benefit to watering the plants before the given plant. So, putting all the water after the plant is surely an optimal move.
How to come up with this?