Greedy Algorithms

Lecture 3: Summer of Competitive Programming

Soumil Aggarwal

Algorithms and Coding Club
Indian Institute of Technology Delhi

3 July 2021

Greedy Algorithms - Introduction

- You make choices which are locally optimal (according to some heuristic), and try to prove that this produces a globally optimal solution
- Heuristic - A guiding principle which you think a solution to the problem should follow. Often involves choosing an element which is 'extreme' (biggest/smallest/best/worst) in some sense
- Heuristics are the key to solving any problem greedily. All magical solutions usually have some solid heuristics behind them.

.

Greedy Algorithms - Introduction

- Can also involve the idea of local changes - can you make small changes to the current configuration to move to a 'better' neighbouring configuration (in a way, looking at the 'derivative')
- Local change - Any change which is 'small' is some sense. Example - Swapping two elements in an array of n numbers
- Greedy algorithms are efficient because you only process small chunks of information at each step. They don't always work / it can be hard to prove their correctness, due to the same reason

Interval Scheduling

Let's make some of this stuff a little more concrete by considering the following classical example:

Problem

You are given a set of n tasks to be performed, along with the starting and ending time for each task. You are supposed to choose some k tasks so that the time intervals for no two tasks intersect, and k is the maximum possible.

Interval Scheduling

Example

Task	1	2	3	4	5	6
A						
B						
C						
D						

Here, note that choosing tasks A, C, and D gives us a valid selection with $k = 3$. It is easy to check that $k < 4$, so we're done.

Interval Scheduling

Some definitions

- Valid selection - Any selection of intervals that don't intersect with each other. We basically just need to find the largest valid selection.
- S - Set of intervals that have already been chosen (we assume it to be a valid selection)
- T - Set of intervals which aren't in S, and also don't intersect with any interval in S. Basically, the set of intervals from which you should pick the next interval.

Interval Scheduling

Heuristic 1

- Choose the smallest interval in T. Repeat until T becomes empty.
- Makes sense because smaller intervals should intersect with a lesser number of other intervals, so we should be able to select more intervals in this manner

It works for the original example we considered -

Task	1	2	3	4	5	6
A						
B						
C						
D						

Interval Scheduling

Heuristic 1

But it fails for the following set of tasks:

Task	1	2	3	4	5	6
A						
B						
C						

The heuristic says you should choose B and go home. But we could choose A and C and be happier. So we need to think of something else.

Interval Scheduling

Heuristic 2

- In the counterexample to the last heuristic, we saw that small intervals don't really have to have less number of intersections. So now we refine the strategy. We choose the element of T which intersects with the least number of other elements of T. Repeat until T is empty.
- Note that at each step, the size of T is reduced as little as possible locally.
- This seems like a pretty evolved strategy, and works for both the cases we've considered before.

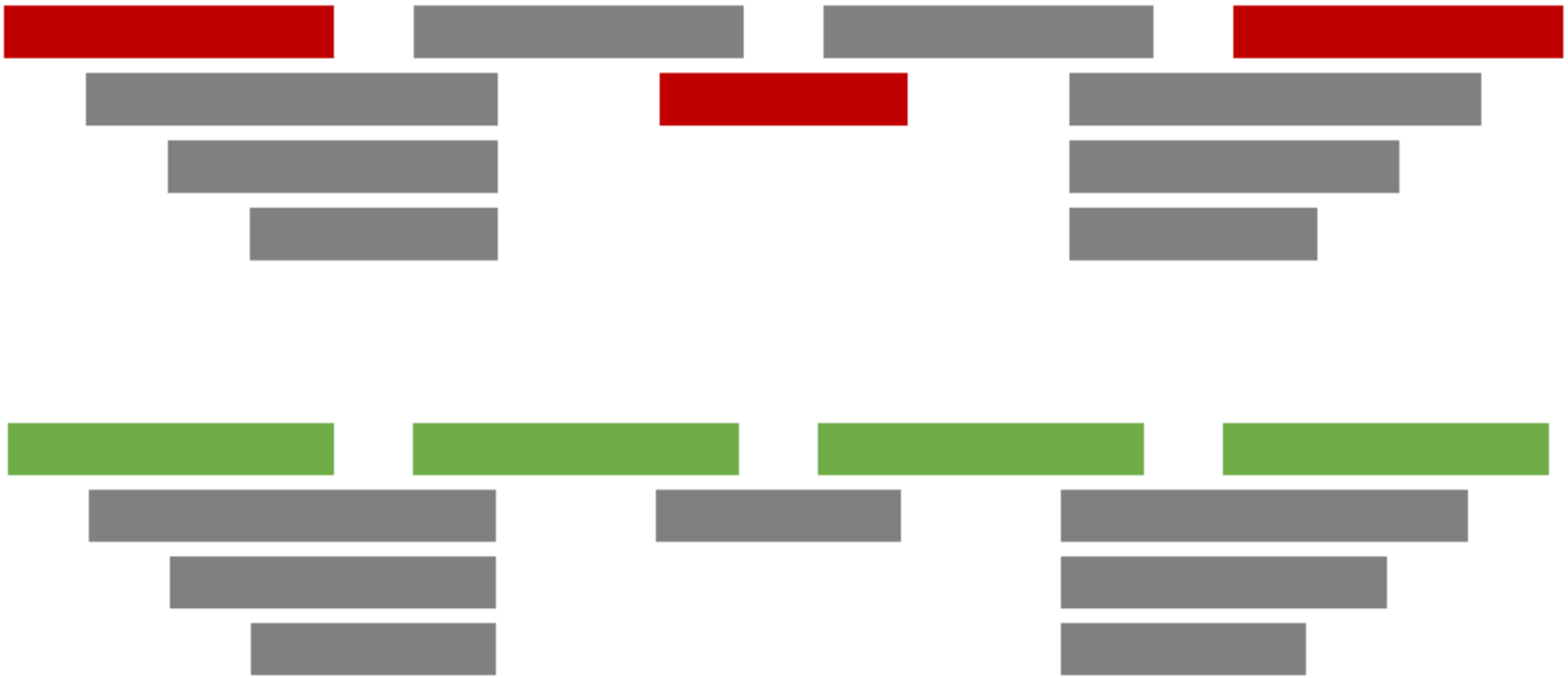
Interval Scheduling

Heuristic 2

- However, the following example kills all hopes and dreams. Adios.

Interval Scheduling

Heuristic 2



Interval Scheduling

Heuristic 3

- By now, we've realised that the number of intersections might not be the right thing to consider. But 'reduce the size of T as little as possible' still seems like a useful idea. So we look for some other way to do it.
- We notice that we can order the elements of S according to their position on the number line. More specifically, for an interval $I = (a, b)$, we define $L(I) = a$ and $R(I) = b$. Then we can order the elements of S as $I_1 < I_2 < \cdots < I_n$, where $R(I_j) \leq L(I_{j+1})$ for $1 \leq j < n$.
- What happens if you try to build up S so that you add I_1 first, I_2 next, and so on?

Interval Scheduling

Heuristic 3

- One thing to note is: If you've built S upto I_m, then T can be taken to be just the set of all intervals J which satisfy $L(J) \geq R(I_m)$.
- So, how do we choose I_{m+1} so that the size of T is reduced as little as possible?
- We have the following heuristic - Choose I_{m+1} to be the interval X in T such that $R(X)$ is smallest.
- We can check that this works for all the examples we've considered so far.

Interval Scheduling

Proof

We now try to prove that this strategy actually works, by trying to reduce any optimal choice of intervals to our greedy choice.

- Suppose the set U containing $I_1 < I_2 < \cdots < I_k$ is an optimal choice of intervals.
- We will change the above set of intervals to the set U containing $G_1 < G_2 < \cdots < G_k$, which will be chosen according to our greedy strategy. We will do this by replacing I_j by G_j one-by-one, going from $j = 1$ to $j = k$, and claiming that U remains valid at each step. We prove all this works using induction on j.
- If $S = \phi$, then T includes all the intervals. So, it is trivial to see that G_1 can replace I_1, as $R(G_1) \leq R(I_1)$ by the definition of G_1.

Interval Scheduling

Proof

- Now, suppose U has become $G_1 < \cdots < G_j < I_{j+1} < \cdots < I_k$. Then for $S = \{G_1, \ldots, G_j\}$, T contains I_{j+1}, so T is non-empty. So a valid choice of G_{j+1} can be made, and $R(G_{j+1}) \leq R(I_{j+1})$ by definition, due to which U remains valid on replacing I_{j+1} by G_{j+1}. Hence our proof by induction is complete.

So we saw that:

- The underlying mega heuristic was - reduce the size of T as little as possible. But since we only optimised this locally, how exactly we did it made the difference between failure and success (heuristics 2 and 3). (On thinking deeply about it, we see that the main difference that matters is - In heuristic 3, suppose that on choosing interval A, T will become T_A, and on choosing interval B, T will become T_B. Then either $T_A \subseteq T_B$ or $T_B \subseteq T_A$. Such a structure isn't there in heuristic 2, where we only consider the sizes of T_A and T_B.)
- Proving that your greedy algorithm works is really important, as a really solid-seeming heuristic can end up crashing and burning for some case, and that will just lead to frustration and loss of points.
- It is important to come up with counterexamples. One way to do it is challenging the underlying assumption of the heuristic, and constructing test sets accordingly.

Proof Strategies

- Considering the optimal solution and trying to 'reduce' it to the greedy solution is a nice strategy, especially when non-greedy optimal solutions also exist (just what we did in the previous example).
- Comparison with the optimal solution - This can also be helpful when the only optimal solution is the greedy solution. You compare an optimal solution A with the greedy solution G, and prove that if A is different from G, you can make a local change to A to strictly optimise it further, which would be a contradiction.
- Inequalities and monovariants related to the quantity we are trying to optimise are often helpful. Induction and contradiction are of course, ubiquitous.

Another Example

We consider another, slightly different example:

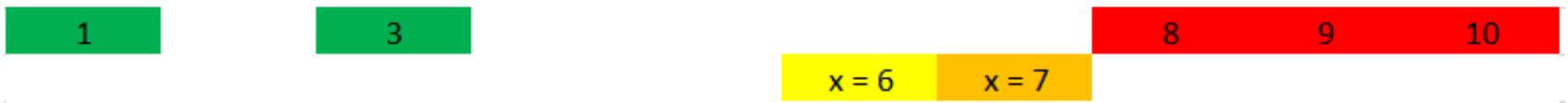
Problem

You are given n real numbers $a_1 \leq \cdots \leq a_n$. Find x so that $f(x) = \sum_{i=1}^{n} |x - a_i|$ is minimised.

Another Example

For this problem, the idea of making small local changes is very helpful. Specifically, we consider what happens to $f(x)$ when we change x to $x + \epsilon$ for a small $\epsilon > 0$.

- Let $s(x)$ be the number of a_i satisfying $a_i \leq x$, and $l(x)$ be the number of a_i satisfying $a_i > x$. Then note that $f(x)$ will change by $(s(x) - l(x))\epsilon$.
- Thus, you should increase x iff $(s - l)(x) < 0$, i.e. there are more numbers in front of it than behind it.



Another Example

- $(s - l)(-\infty) = -n$, $(s - l)(\infty) = n$, $(s - l)$ is a non-decreasing integer function.
- So, x must lie just beyond the region where $(s - l)(x) < 0$. x will be the median of the sequence, i.e. $x = a_{\frac{n+1}{2}}$ if n is odd, and $x \in [a_{\frac{n}{2}}, a_{\frac{n}{2}+1}]$ if n is even (x can take any value in the interval). Basically, x must have as many values in front of it as there are behind it.