Competitive Programming Club

Meeting 1

What is competitive programming?

Art of solving algorithmic problems in an efficient way

Solutions should not only always be correct, but also fast

Usually we are looking for deterministic algorithms finding the best solution

The algorithm is tested on prepared test-cases, meaning that not only the solution has to be correct and fast, but it also has to be implemented without flaws

What is ICPC?

ICPC – International Collegiate Programming Contest

Organization preparing world-wide competitions in competitive programming on collegiate level

Format:

- 8-13 problems
- Teams of 3 people, but only with one computer
- Each correct solution is 1 point, total time of submissions is the tie-breaker
- Every incorrect solution – 20 minutes penalty
- Teams qualifying starting with regional contests, ending with the World Finals

Competitions

- SoCal – happening every fall in Riverside
  - UCSD sends 3-6 teams, so no high skills needed to attend
  - Low difficulty fun contest, we encourage everyone to participate
- NAC/World Finals
  - Only the best UCSD team from SoCal participates
  - Challenging but rewarding competitions
  - Next year new team will be needed – this can be you!
- Other (less serious, more for practice/fun)
  - CALICO - team competition in a mixed style. Chill and fun, though not so serious as the other ones. Perfect for beginner and intermediate contestants. More information here
  - USACO - individual IOI-style competition, pretty difficult. More information here
  - Ask us for more options if you are interested

Platforms

- Codeforces
  - Great CP platform with varying difficulty of problems and contests
  - A lot of blogs and tutorials – good place to learn
- Leetcode
  - Focused on interview prep, but still pretty good quality problems
  - A lot of beginner materials
- Vjudge
  - We will use this one for our meetings – make sure to create an account!

Officers

- This year – Stan, Huize, Raymond, Shang and Qihao
- We are here to help you – please ask us any questions you have, both regarding logistics and competitive programming

Meeting plan

6:00-6:15 – Pizza

6:15-7:00 – Tutorial

7:00-8:00 – Practice problems

Room: TBA

Binary Search

Motivation

Sometimes we want to find a value without checking all the possibilities

If we can somehow organize (sort?) all the possible values, we can use o more sophisticated method of searching through them

Toy problem – guess the number

I think of a number between 1 and 100, you have to guess it.

You can ask questions “is the number x?” and I will answer “my number is greater/smaller/equal to x”.

Checking all 100 numbers will take quite some time and we can definitely do better.

What’s the best strategy?

Toy problem – guess the number

If we ask for number 50, we get rid of half of the numbers in the first guess

We can repeat the procedure, each time time decreasing the number of possibilities by half.

This way, we find the answer using log2(n) queries, where n was the number of initial possibilities

Pseudocode

Initialize interval [l, r] to [1, n]

while not guessed:

    mid = (l+r) / 2 # middle of the interval
    ask for mid

    if mid is correct:
        you guessed

    if mid is smaller:
        l = mid + 1

    if mid is larger:
        r = mid - 1

Real life problem – calculate square root

There is no formula to calculate the value of sqrt(x)

We can however binary search the approximation

We will never guess the perfect number - it’s a real number

Instead, we stop the algorithm once our approximation is good enough – the possible interval is small enough

Binary search over the answer

We have a big boat that can carry x tons of cargo.

We have n trucks in a line, each weighting a_i tons

We need to carry the trucks to the other side of a river, we can do at most k boat trips (and trucks have to be packed in correct order)

How big does x have to be to succeed?

Binary search over answer

If we guess some x, we can easily check if we can succeed – just greedily pack trucks while there is still space (since order is determined) and check how many trips we have to make

Then we can binary search the x. Out greater/less/equal answer will be given by our greedy program

Note that here we have only greater or less/equal. Since we never know that we guessed the correct answer, we have to keep running our binary search until out interval is small enough (in that case, of size 1)

When to do binary search over answer? If:

If success is possible for x, it is surely possible for x+1, but if it’s not possible for x, then it’s surely not possible for x-1

You can guess some x and easily verify is this x is a valid solution

You are looking for the smallest/biggest possible solution

You have to find the smallest possible maximum, or the biggest possible minimum

THEN it is very likely binary search over answer

Parallel binary search (advanced)

You are given a graph where we add new edges one at a time. We also get many queries to find the earliest moment that two vertices are connected.

While there are smarter solutions, we will focus on a binary search ones.

If we add edges one by one, we can easily check for any query if it’s fulfilled (DSU).

However, for any query, it’s hard to check if it’s fulfilled in the arbitrary moment in time

Parallel binary search (advanced)

Doing a binary search on one query is easy, we just add the edges and at a given moment in time, we check if the query is satisfied

…but we can check all the queries, since such a check is easy

We run all binary searches at once, so even if there are q queries in total to answer, we will simulate adding all edges only log(n) times