UCF ICPC Training Camp

Lecture I: Interactive/Constructive

Christian Yongwhan Lim

Friday, March 29, 2024

Christian Yongwhan Lim

Education

Stanford University

Part-time Jobs

Full-time Job

Google Research

TWO SIGMA

Workshops

Stanford ENGINEERING | Stanford Computer Forum

Coach/Judge

icpc International Collegiate Programming Contest

icpc.foundation

advancing the art and sport of competitive programming

https://www.yongwhan.io

Christian Yongwhan Lim

- Currently:
  - **CEO** (Co-Founder) in a Stealth Mode Startup;
  - **Co-Founder** in Christian and Grace Consulting;
  - ICPC **Internship Manager**;
  - ICPC **North America Leadership** Team;
  - Columbia ICPC **Head Coach**;
  - ICPC **Judge** for NAQ and Regionals;
  - ICPC NAPC **Trainer**;
  - **Adjunct** (Associate in CS) at Columbia;

https://www.yongwhan.io

Today's Format

9:30am ET - 11:30am ET	**Lecture: Interactive/Constructive Problems**
11:30am ET - 12:30pm ET	**Lunch**
12:45pm ET - 3:45pm ET	**Contest #2**
4pm ET - 6pm ET	**Contest #2 Review**

Discord Server: ICPC CodeForces Zealots

- Join the following discord servers, if you have not already!!!

  [Programming Zealots] https://discord.gg/7bvMnMyF6G

Success Pathways

- CodeForces group: **Programming Zealots**
- **800 - 2100 (A - N)**
  - **For those who are just starting**
  - To gain some experiences with an explicit goal to enjoy the process of solving new problems;
  - To make it to the ICPC North America Championship (NAC)!
- **2200 - 3500 (O - ZB)**
  - **For those who are more serious**
  - To make it to the ICPC World Finals (and potentially winning a medal)!

Practice Strategy

- If your goal is to get to a rating of **X**, you should practice on problems that are **X + 300** typically, with a spread of 100. So, picking problems within the range of:

  {X + 200, X + 300, X + 400}

  would be sensible!

- So, if you want to target becoming a **red (grandmaster)**, which has a lower-bound of 2400, you should aim to solving {2600, 2700, 2800}.
- **(Eventual) Target**: You should focus on solving it for 30 minutes or less!

Practice Strategy (con't)

- You should focus on solving each problem for **30 minutes or less**; if you cannot, you should consider solving a problem with a lower rating.
- You should aim to solve **~5 problems** each day within this range to expect a rank up within six months.
- If you cannot solve a problem, here is a sample recipe you can follow:
  - Look at editorial for **hints**, and try to solve the problem.
  - Look at editorial for **full solutions**, and try to solve the problem.
  - Look at **accepted code**, and try to solve the problem.
  - Make sure you **revisit after two weeks** and see if you can solve it.

Programming Contests

- **CodeForces**
- **AtCoder**

- **Universal Cup**: https://ucup.ac/register

- **Quarterly Contests** from ICPC Curriculum Committee, starting **June 2024**

Training Resources

- **U ICPC**: https://u.icpc.global/training/
- **CP Algorithms**: https://cp-algorithms.com/
- **USACO Guide**: https://usaco.guide/

- **Kattis**: https://open.kattis.com/
- **Methods to Solve**: https://cpbook.net/methodstosolve?oj=kattis&topic=all&quality=all
- **CSES**: https://cses.fi/problemset/

Interactive Problems

- The input data given to your program may not be predetermined but is built specifically for your solution.

- Jury writes a special program: **interactor**, such that its output is transferred to the input of your solution and the output of your program is sent to interactor's input.
  - Your solution and the interactor exchange the data and may decide on what to print based on the "history of communication".

Make sure to flush!

- When you write the solution for the interactive problem, it is important to keep in mind that if you output some data it is possible that this data is first placed to some internal buffer and may be not directly transferred to the interactor.
  - **You have to use flush operation each time you output some data.**
- Typically:
  - If you use “cout”, make sure to use **“endl”**; do not disable cin.tie/cout.tie.
  - If you use “printf”, make sure to use **“fflush(stdout)”**.

Warm-Up Problem: Guess the number

- In this problem there is some hidden number and you have to interactively guess it.
- You can make queries to the testing system. Each query is one integer from 1 to 1000000. There are two different responses:
  - “<” (without quotes), if (hidden number) < (your query);
  - “>=” (without quotes), if (hidden number) >= (your query);
- Print “! x” to provide x as the answer.
- Your program is allowed to make **no more than 25 queries** (not including printing the answer).

Warm-Up Problem: Guess the number (con't)

- **Input**: Use standard input to read the responses to the queries. The input will contain responses to your queries: "<" and ">=". When your program will guess the number print "! x", where x is the answer and terminate your program. The testing system will allow you to read the response on the query only after your program print the query for the system and perform flush operation.

Warm-Up Problem: Guess the number (con't)

- **Output**: To make the queries your program must use standard output. Your program must print the queries, one query per line. After printing each line your program must perform operation flush. The response to the query will be given in the input file after you flush output. In case your program guessed the number x, print "! x", where x is the answer and terminate your program.

Warm-Up Problem: Guess the number (con't)

- Any idea?

Warm-Up Problem: Guess the number (con't)

```
int main() {
  int l=1, r=1e6;
  while(l!=r) {
    int m=(l+r+1)/2;
    cout<<m<<endl;
    string s; cin>>s;
    if(s=="<") r=m-1;
    else l=m;
  }
  cout<<"! "<<l<<endl;
  return 0;
}
```

Exercise #1 [1300] PolandBall and Forest

- https://codeforces.com/problemset/problem/755/C

Exercise #1: Solution Idea

- **Disjoint Set Union** (DSU)!

Exercise #2: [1700] Tree Diameter

- https://codeforces.com/problemset/problem/1146/C

Exercise #2: Solution Idea

- **BFS/DFS**

- The standard algorithm for finding the diameter of a tree is:
  - find the farthest distance from node 1;
  - find the farthest distance from that node.

- We can apply this idea in this problem!

Exercise #3: [2100] Guess The Maximums

- https://codeforces.com/problemset/problem/1363/D

Exercise #3: Solution Idea

- **Binary Search**

- maximum of the array is the password integer for all but at most 1 position.
- find the subset (if the maximum is in a subset) in which the maximum exists using **binary search**.
- query the answer for this subset separately.
- for all the subsets, the answer is the maximum for the whole array.

Exercise #4: [2200] In Search of Truth (Easy Version)

- https://codeforces.com/problemset/problem/1840/G1

Exercise #4: Solution Idea

- **Ad Hoc**

- Do “+ 1” query 999 times.
- Do “+ 1000” query 1000 times.

Exercise #5: [2800] Matching Reduction

- https://codeforces.com/problemset/problem/1721/F

Exercise #5: Solution Idea

- **Dinic**, to find a maximum matching!
- Recall: |maximum matching| = |minimum vertex cover| (this only works in bipartite graphs).
- We can find the minimum vertex cover in the graph using the standard algorithm to convert the maximum matching into minimum vertex cover.
  - You can represent the minimum vertex cover as the **minimum cut**.
- For each query of type 1, take a vertex from the vertex cover we found.
- When we remove a vertex, we can simply remove the corresponding edge from the maximum matching.

Constructive Problems

- Constructive problems are **ad-hoc**, so they are difficult to solve even if you have done lots of constructive exercises before.
- These problems are remarkable in that solutions rarely use some template algorithm (e.g., segment tree, dynamic programming, etc).
- The best predictor of your ability to solve constructive problems is going to be having a strong mathematical intuition. **It is much harder to train**.
- A common strategy for these problems is to **manually solve a few inputs** first. Then, build insight from that to construct a deterministic algorithm that provably works for all inputs.

Warm-Up Problem

- Let $1 \le N \le 100{,}000$ and $1 \le K \le \log_2(N)$ be fixed.

- Imagine performing a binary search on the values 1, ..., N.

- Give a value X, $1 \le X \le N$, which would be found after exactly K steps of the binary search.

Warm-Up Problem (con’t)

- Any idea?

Warm-Up Problem (con’t)

- Just do the usual/vanilla implementation of the binary search!

Exercise #1: [1600] Fixed Point Guessing

- https://codeforces.com/contest/1698/problem/D

Exercise #1: Solution Idea

- **binary search**!

Exercise #2: [2000] Matching vs Independent Set

- https://codeforces.com/problemset/problem/1198/C

Exercise #2: Solution Idea

- **Greedy**

- Let's try to take edges to matching greedily in some order.
- If we can add an edge to the matching (both endpoints are not covered), then we take it.
- **~~It is easy to see~~ that all vertices not covered by the matching form an independent set; otherwise, we would add an edge to the matching.**
- Either matching or independent set has size at least n.

Exercise #3: [2000] Game with modulo

- https://codeforces.com/problemset/problem/1103/B

Exercise #3: Solution Idea

- **powers of two and binary search**
- Let's ask (0, 1), (1, 2), (2, 4), (4, 8), ..., (2^{29}, 2^{30}). Let's find the first pair in this list with the answer “x”.
  - This pair exists and it will happen for the first pair (l_0,r_0) that satisfy the inequality $l_0 < a \le r_0$. We can find this pair using at most **31** questions.
- Now, if we ask (l_0, x) for some $l_0 < x \le r_0$, we will get the answer “y” if $x < a$ and the answer “x” otherwise. Binary search! Here we will use at most **29** questions.
- We need to ask at most 31+29 = 60 questions!

Exercise #4: [2400] Neko and Flashback

- https://codeforces.com/problemset/problem/1152/E

Exercise #4: Solution Idea

- **Hierholzer**, to find Eulerian path

- For all elements of b' and c', we need to replace them with corresponding value from 1 to k, where k is the number of distinct value in b' and c'. Use map or sort.
- Build the graph G by adding an undirected edge between b' and c'.
- Find Eulerian path in G using Hierholzer's algorithm in $O(n)$ or detect that there is no such path.

Exercise #5: [2900] Red-Blue Graph

- https://codeforces.com/problemset/problem/1288/F

Exercise #5: Solution Idea

- **flow with demands** (https://cp-algorithms.com/graph/flow_with_demands.html)

References

- **Interactive**
  - https://codeforces.com/blog/entry/45307
  - https://codeforces.com/problemset?order=BY_RATING_ASC&tags=interactive

- **Constructive**
  - https://codeforces.com/blog/entry/80317
  - https://codeforces.com/problemset/page/1?order=BY_RATING_ASC&tags=constructive+algorithms

Contest #2

- https://codeforces.com/gym/513942


THANK YOU