Randomized Algorithms

Huize Mao

Meeting 6., Nov 18, 2024

Competitive Programming Club @ UC San Diego

Table of Contents

- Randomized Algorithms
- Simulations
- Seeds, Random Generator
- Random Topics
    - Great Future of Competitive Programmers
    - (not so) Random People from the CP Community
    - Club Survey
- Practice Tips, Words of Encouragement, Next Quarter

Motivation

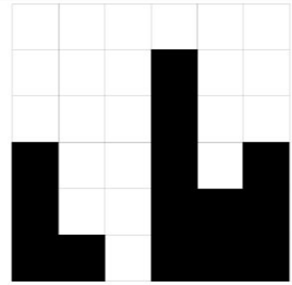
- Sometimes we could sacrifice 0.1–0.2% accuracy to reduce a problem to a much easier one
- That is, instead of coming up with a solution that will solve a very complex problem 100% of time, we come up with a solution that will solve the problem 99.9% of the time, but is much easier
- Now if you have 100 test cases, that is still 0.999^{100} ~= 0.904 = 90% chance of passing a problem
- if it doesn't pass…
  - submit again
  - and again
  - until it passes…
  - Or if you can’t get it passed truly due to chance, consider buying a lottery…

Example 1: Histograma

Given a rectangle of size n x n, there are black and white squares. The black squares are arranged so that, in any given column, they are consecutive from the bottom (positions (0, col), (1, col), ..., (max_row, col) are black), and the rest of the column is white.

You need to query the interactor to check if a specific position is black or white, and the goal is to determine the maximum "height" of black squares.

n <= 5000



Example 1: Repeated Binary Search

- Randomly shuffle the order of column ID that you will process
- For each column, binary search the tallest black height
- Record the highest of this height
- You don’t need to binary search a column x if that column has the pixel (max_height, x) as white
- Expected number of queries to the interactor: N + log(N)*ln(N)

why?

- You query at least once (the max current height) -> total of N operations
- If you have queried i columns, the probability of doing a binary search on the i-th iteration is 1/i, because if you randomly shuffled, the probability tha i-th column has the highest black height is 1/i
- The expected value for number of binary search is: logN * $(1+\frac{1}{2}+\frac{1}{3}+ \ldots + 1/n)$ = logN * ln(N) → combining two parts yields N+logN*ln(N)
- Better than the deterministic solution on average

Example 2: Line through N/4 points

Given $N \leq 10^5$ points, find a line that passes through the maximum number of points. It's guaranteed that the answer is at least N/4 .

Example 2: Choose random lines

- Choose random pairs of points and form a line, and see if it works
- It won't take many tries because it's guaranteed at least N/4 points are on a line.

How many times will it take then?

- Probability that a random pair of points are collinear is ¼
- -> not collinear is 15/16
- Now if we take 100 pairs, the probability that none of them are collinear is $(15/16)^{130}$ ~= 0.00022 = 0.022%
- So your probability of having the answer line is 99.978%
- Say there is 100 test cases… → 97.82% chance passing
- And checking how many points are on a chosen line is a much easier problem to solve

Example 3: DNA Prefix

Guess a hidden string S with characters A, C, T, G. You can choose some string and ask if it's a prefix of S. The length of S is at most 10000 and you can ask up to 25000 queries

Example 3: Randomly Guessing

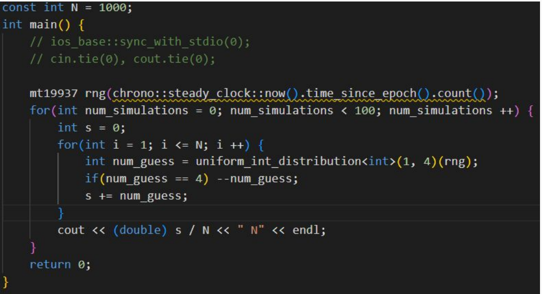
- Naive solution is to guess A, then C, then T, then G… -> 4N guesses, we want to reduce it to 2.5N guesses
- How about random guessing ACG or T?
- Expected number of guesses is (1+2+3+4)/4= 2.5 every time
- Then on average it will take 2.5N guesses, but it could easily go over 2.5N! If there are multiple test cases then we aren't so lucky this time
- Well, if you have guessed AC and G, why guess T?
- -> reduce it yet again to (1+2+3+3)/4 = 2.25 -> 2N times on average

Will 2N be good enough? That is an average, what's the probability that it will go over 2.5N?
That is, what's the variability in the distribution of number of guesses, what's the variance?

Simulations

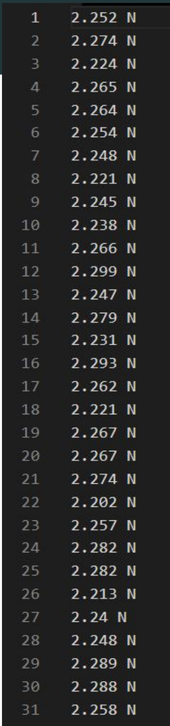
What's the variability in the number of guesses in our solution?

- You could do some math and figure it out
- Or…
- You could take advantage of your computer and do some simulations
- It turns out it rarely vary more than 0.05N (always between [2.2N, 2.3N])-> 2.5N is safe



```cpp
const int N = 1000;
int main() {
    // ios_base::sync_with_stdio(0);
    // cin.tie(0), cout.tie(0);

    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
    for(int num_simulations = 0; num_simulations < 100; num_simulations ++) {
        int s = 0;
        for(int i = 1; i <= N; i ++) {
            int num_guess = uniform_int_distribution<int>(1, 4)(rng);
            if(num_guess == 4) --num_guess;
            s += num_guess;
        }
        cout << (double) s / N << " N" << endl;
    }
    return 0;
}
```



```
2.252 N
2.274 N
2.224 N
2.265 N
2.264 N
2.254 N
2.248 N
2.221 N
2.245 N
2.238 N
2.266 N
2.299 N
2.247 N
2.279 N
2.231 N
2.293 N
2.262 N
2.221 N
2.267 N
2.267 N
2.274 N
2.202 N
2.257 N
2.282 N
2.282 N
2.213 N
2.24 N
2.248 N
2.289 N
2.288 N
2.258 N
```

Extra Note On Simulation

- On previous slide, I set N as 1000, but the N from the problem is 10000: from statistics, if sample size increase, then the variability goes down, so it should work even better.
- You could use it to see variability from the average like we just did for a random algorithm
- You could also use it to find patterns in other problems, not necessarily random algorithms: simulate a process the problem gives you, see if there are any patterns, especially math problems
- Also use it to find parameters that would give you worst case scenarios for your algorithm. Kind of like what we just see, but sometimes your algorithm f(x) might depend on the x, and you could simulate your f() on all possible x’s to find the worst case.

Seeds, Random Generator

Mersenne Twister (mt19937) RNG

`#include <random>`

- **mt19937** is a Mersenne Twister based on the prime **2^{19937} - 1** (its period).
- Higher-quality RNG than `rand()`, plus **faster**:
  - `mt19937`: **389 ms** to generate and add 10^8 numbers
  - `rand()`: **1170 ms** for the same task
- Outputs **full 32-bit unsigned integers** (0 to 2^32 - 1 = 4,294,967,295), whereas `rand()` maxes out at **32767**.

Seeds, Random Generator

Implementation:

cpp

```cpp
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
```

Seed:

```cpp
chrono::steady_clock::now().time_since_epoch().count()
```

Avoid fixed seeds (e.g., seed(47))- they are easily predictable.

Example Code

```
int main() {
    mt19937
rng(chrono::steady_clock::now().time_since_epoch().count());
    vector<int> permutation(N);

    for (int i = 0; i < N; i++)
        permutation[i] = i;

    shuffle(permutation.begin(), permutation.end(), rng);
    cout << average_distance(permutation) << '\n';

    for (int i = 0; i < N; i++)
        permutation[i] = i;

    for (int i = 1; i < N; i++)
        swap(permutation[i],
permutation[uniform_int_distribution<int>(0, i)(rng)]);
    cout << average_distance(permutation) << '\n';


}
```

Related Topics

- Random Variables
- Expected Values
- Probability

Useful in real-life but less so in Competitive Programming

- Statistics
- Information Theory
  - Btw the bonus problem of this week uses this. [Hint: same technique is used in decision trees in machine learning]

Practice Problems

https://vjudge.net/contest/672670#overview

- There are only a few this week but they are not easy
- After solving Problem 1, Read through the rest and work on the one you like the most
- Ask for hints

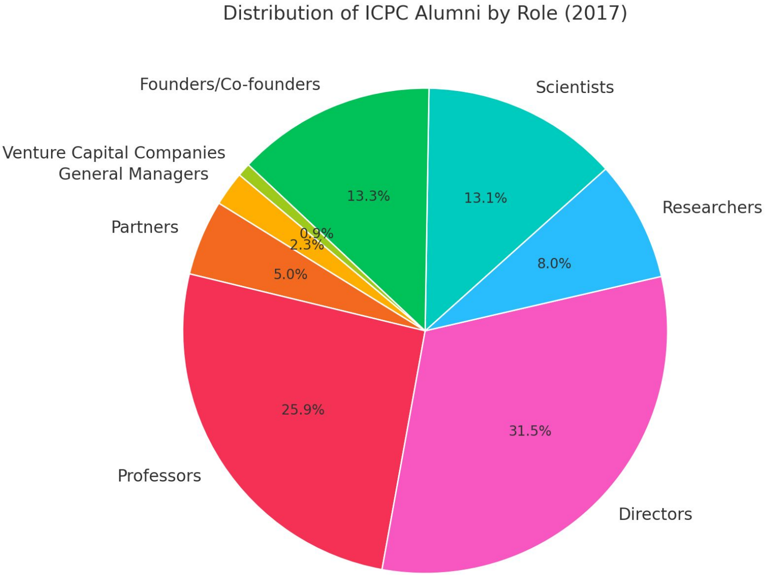
Random Topics!

(You could choose to listen to this or work on problems)

2017 Statistics About ICPC Alumni

- **400,000 total alumni** globally.
- **7% are Software Engineers**, many holding senior and leadership roles.
- **1,500 alumni** are founders or co-founders (**0.3%** of total alumni).
- **Several thousand** alumni hold executive titles, including President, CTO, CEO, COO, or Owner. (Specify the number if possible.)
- **Notable professional roles among alumni:**
  - **260 General Managers**
  - **570 Partners**
  - **2,933 Professors**
  - **3,558 Directors**
  - **908 Researchers**
  - **1,484 Scientists**
- **Over 100 venture capital companies** are represented by ICPC alumni.

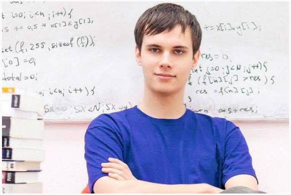
Alumni Pie Career Roles Chart



Figures from the Competitive Programming Community

Gennady Korotkevich

- Handle: **tourist** on Codeforces.
- Often regarded as the best competitive programmer in history.
- World Finals ICPC champion.
  - Our amazing Jingbo, professor at UCSD, faculty advisor of this club, was technically the world champion in the sense that tourist should not count lol. The man is immortal.
- Multiple-time winner of:
  - Google Code Jam
  - Facebook Hacker Cup
  - Topcoder Open
  - AtCoder Grand Contest



- Codeforces rating leader.
  - The platform created a rating title after him **tourist**

Figures Continued

Mike Mirzayanov

- **Founder of Codeforces**: One of the largest competitive programming platforms globally, established in 2010.
- **Creator of Polygon**: A widely-used system for creating and testing programming contest problems.
- **Global Influence**: Continues to shape the competitive programming community and foster talent worldwide.

Atsushi Takahashi

- **Founder of AtCoder**: Established in 2012, AtCoder is a prominent competitive programming platform known for its high-quality contests and problems.
- **Advocate for Programming Education**: Actively promotes programming education and regularly conducts workshops and seminars for students and professionals.

Figures Continued

Neal Wu

- Three-time International Olympiad in Informatics (IOI) gold medalist.
- Maintains a YouTube, sharing insights and tutorials with the community. YouTube
- Co-founder of Cognition, an AI startup that launched 'Devin AI,' an autonomous AI software engineer

Kamil Dębowski, Errichto

- Operates a YouTube channel, "Errichto," providing tutorials and live streams on algorithms and competitive programming.
- Achieved LGM
- Works with Huawei and Harbour Space University

Practice Tips

Favorite blogs from Codeforces:

- Radewoosh: https://codeforces.com/blog/entry/91114

“you really have to have CP in your mind. After solving a problem, it doesn't mean that it's gone and you have to forget about it. Maybe you'll find yourself thinking about some interesting aspects of some task and you'll invent a harder one?”

“So yeah, that's my opinion. Let CP get into your mind and find a true desire to practice. Don't try to force yourself to practice in an organized way.”

- Um_nik: https://codeforces.com/blog/entry/98806

“In archives you learn how to solve problems, in contests you learn how to do it fast. I’m not saying that contests are not needed: speed is also very important, and it is good to keep you in competitive shape. Also participating in contests is just fun, that’s also very important.”

- Solve many problems. There is no shortcut.
- Start with problems around your rating, when it gets easy go up by 100-200 rating
- Participate in contests and discussions

Words of Encouragement

- We did amazing in SoCal, all of our six teams scored top 25 percentile, one 2nd place and one 7th place. We had more than 70% new members among the teams. This is truly amazing and feel proud about yourselves!!
- I have no doubt that if you are here, in this meeting room, and dedicating your time in such a sport that is very challenging intellectually, that you have a immense potential for a bright future in terms of professional careers.
- Also, there are things that are as important as how much money one makes. Knowledge is power, coding is power, sometimes think about how to use your talent.

Survey

https://forms.gle/JHCemSdSdUR1VUDq8

New Beginning

- Keep practicing, we might post weekly practice problem sets on vjudge and be done asynchronously.
- Next quarter we likely will have weekly meeting from week 2 - 8 with similar format.
- we might have a big annual contest, at least hosted by UCSD, maybe along UCLA and UCI.

Thank you for your time