15756 Randomized Algorithms

Rohan Jain

Contents

Chapter 1

100 Prisoners Problem

This is a problem based on a riddle:

Question

There is a jail and a warden. However, the warden is very lazy and says if they win a game, they're all set free, otherwise they all die. There is 100 boxes in a room and each box has a hidden number from 1 to 100. They are assorted in a random permutation. Each prisoner is also given a number from 1 to 100. Each prisoner is allowed to enter the room and will be allowed to open 50 boxes and if they find their number, they win. They will only live if everyone finders their number. The prisoners can't communicate with each other. What strategy should they use to maximize their chances of winning?

Naive strategy: Each prisoner opens 50 boxes at random. The probability of winning is $\frac{1}{2^{100}}$.

Theorem 1.0.1

There exists a strategy such that the probability of winning is $\geqslant \frac{30}{100}$.

Proof. The algorithm is as follows:

1. Each prisoner opens the box with their number.
2. If they find their number, they open the box with the number they found.
3. They continue this process until they find their number or they open 50 boxes.

Cycle notation: Consider the following arrangement, where the ordinal is the box number:

1. 7
2. 6
3. 4
4. 3
5. 8
6. 1
7. 2
8. 0
9. 5
10. 10

This can be represented as 4 individual cycles: $(1, 7, 2, 6)$, $(3, 4)$, $(5, 8, 9)$, (10).

A critical observation to make is that prisoner i will win if and only if the cycle containing i is of length $\leqslant 50$. That is, everyone wins iff there are no cycles of length > 50.

We warm up by considering opening 75 boxes instead of 50. The observation is that if anyone loses, at least 75 people will lose. Let X be a random variable representing the number of people who lose. Then, $\mathrm{E}[X] = 25$ as each player has a $\frac{1}{4}$ chance of losing. Recall Markov's inequality:

Theorem 1.0.2

Let X be a non-negative random variable. Then, for any $t > 0$, $\Pr[X \geqslant t] \leqslant \frac{\mathrm{E}[X]}{t}$.

Applying Markov's inequality, we have $\Pr[X \geqslant 75] \leqslant \frac{25}{75} = \frac{1}{3}$. Thus, the probability of winning is $\geqslant \frac{2}{3}$. Moving back to the original problem, we can apply the same logic.

$$\Pr(\text{anyone loses}) = \Pr(\text{at least one cycle of length} > 50).$$

Recall how we can count the number of cycles of length $\geqslant 50$. For a cycle to be exactly length $n > \frac{100}{n} = 50$, we need to choose n boxes and then permute them. Thus, the number of cycles of length n is $\frac{100!}{n \cdot (100-n)!}$. Now we have to worry about the number of arrangements of the other $100 - n$ boxes, which is $(100 - n)!$. Multiplying these two quantities, we get:

$$\frac{100!}{n \cdot (100-n)!} \cdot (100-n)! = \frac{100!}{n}.$$

As there are 100! total permutations, we realize that exactly $\frac{1}{n}$ of the permutations contain a cycle of length n. Thus, the probability of losing is:

$$\Pr(\text{anyone loses}) = \sum_{n=51}^{100} \frac{1}{n} = \sum_{n=1}^{100} \frac{1}{n} - \sum_{n=1}^{50} \frac{1}{n} = H_{100} - H_{50} \approx \ln 100 - \ln 50 = \ln 2 \approx 0.693.$$

Thus, the probability of winning is $\geqslant 1 - 0.693 \approx 0.307$ as desired.

😄

We move on to proving that this algorithm is actually optimal:

Theorem 1.0.3

The algorithm described above is optimal.

Proof. We consider a second version of the game, where any box that a previous prisoner has opened stays open. So if prisoner i walks in to see that i has been revealed, he just leaves. Otherwise, he opens boxes until he finds i or has opened 50 boxes. Boxes are never closed.

Lenma 1.0.1

Cycle algorithm is no better at version 2 than version 1.

Proof. Option 1: Someone from my cycle in the past lost.

Option 2: I am the first person in my cycle to enter.

In both versions, the boxes in my cycle are closed. This means that in both versions, I win iff the cycle is of length $\leqslant 50$.

Option 3: Someone from my cycle has already entered, and they won. This means the cycle is of length $\leqslant 50$, so I win in both versions. 😄

Lenma 1.0.2

All algorithms are equally good in version 2.

Proof. By symmetry: If I'm opening a box, I have to pick out of the remaining boxes, and they all have the same probability of containing my number. 😊

Together, these lemmas show that the cycle algorithm is optimal in version 1. 😊

Chapter 2

Graph algorithms

*Note: all graphs are going to be undirected and unweighted.

2.1 Graph coloring

Suppose we are given a graph $G = (V, E)$. Suppose G is 3-colorable. Recall that a 3-coloring is a function $c : V \to \{1, 2, 3\}$ such that $c(u) \neq c(v)$ for all $(u, v) \in E$. Simply put, you can color the vertices such that no two adjacent vertices have the same color.

Question

Can we efficiently find a 3-coloring of G?

Answer: No, this is NP-hard.
Observation: It's possible to partition the vertices in the graph into two parts that are each $\triangle$-free. This is easy because we can create the partition of one set being the odd vertices (1 and 3) and the other set being the even vertices (2).

Theorem 2.1.1 McDiarmid (1993)

Given a 3-colorable graph, you can construct a triangle-free partition of the vertices in poly(n) expected time.

Proof. Consider the following algorithm: in each round, find a triangle in some side, pick a random vertex from the triangle, and move it to the other side. Repeat this process until there are no more triangles.

Now we analyze the algorithm. Fix a 3-coloring of G. Let X be the number of vertices of color 1 on the left side. Let Y be the number of vertices of color 2 on the right side. Now we analyze what happens to $X + Y$ during a round:

- If we move a vertex from the left to the right, X can decrease by 1 with probability $\frac{1}{3}$, Y can increase by 1 with probability $\frac{1}{3}$, or both can stay the same with probability $\frac{1}{3}$.
- If we move a vertex from the right to the left, Y can decraese by 1, X can increase by 1, or both can stay the same all with probability $\frac{1}{3}$.

So overall, $X + Y$ increases by 1 with probability $\frac{1}{3}$, decreases by 1 with probability $\frac{1}{3}$, and stays the same with probability $\frac{1}{3}$. This is just a random walk.

The random walk starts at an unknown value, but if it ever leaves $[0, n]$, the algorithm terminates. It can look something like $+1, -1, 0, 0, -1, +1, +1, +1$, and at some point t_1 the algorithm finishes. For sake of discussion, assume this hypothetical random walk keeps going. At some point, the walk may leave $[0, n]$ at time t_2. Note that t_2 can exist only if t_1 exists, and $t_1 < t_2$. So $\mathrm{E}[t_2] \geqslant \mathrm{E}[t_1]$.

Goal: prove that $\mathrm{E}[t_2] \in O(n^2)$. This will imply that $\mathrm{E}[t_1] \in O(n^2)$, which will imply that the algorithm runs in poly(n) time.

We step away from the graph problem to analyze random walks further. Consider a random walk on $\mathbb{Z}$ that starts at 0. Let T be the first time the walk arrives at n or $-n$. We want to show that $\mathrm{E}[T] \in O(n^2)$.

Fact: after t steps, the middle 90% of possible positions would be $O(\sqrt{t})$ from the origin. Additionally, within this 90%, the walk is about uniformly distributed.

Using the fact above, we want $t = n^2$ to be the time it takes to reach n or $-n$.

Now we return to the original problem.

Lenma 2.1.1

For any power of 2, the expected time to get from the origin to n or $-n$ is n^2.

Proof. If $n = 1$, the expected time is 1. Define T_n as the expected time to get from the origin to n or $-n$. Pretend we split the number line into $-n, -n/2, 0, n/2, n$. Then we can write T_n as:

$$T(n) = T(n/2) + T(n/2) + \frac{1}{2}T(n).$$

Solving, we get $T(n) = 4T(n/2) \implies T(n) = n^2$. 😊

This means that our algorithm is $O(n^2)$ rounds, meaning the algorithm runs in poly(n) time. 😊

2.2 Shortest paths

Given $G = (V, E)$, for each $x, y \in V$, we want $d(x, y) \pm O(1)$ in time $O(n^{2.5} \log n)$.

Naive algorithm: $O(|V| \cdot |E|) = O(n^3)$.

Now consider a shortest path from x to y.

- Option 1: Every vertex on the path has degree $< \sqrt{n}$.

  Then for each vertex, we can BFS, but only on low degree vertices. This takes $O(n^{2.5})$ time. The result is that we get exact distances for all vertices on the path.

- Option 2: There is a vertex on the path with degree $\geqslant \sqrt{n}$.

  Now we sample $100\sqrt{n} \log n$ iid uniformly random vertices and call this set S. We claim that with high probability, every high degree vertex w will be incident to a vertex $s \in S$.

  If we take this claim for granted, observe that for x, y in option 2, $d(x, y) = d(x, s) + d(s, y) \pm 2$ for some $s \in S$. So for each $s \in S$, do a BFS which takes $O(n^{2.5} \log n)$ time. For each pair (x, y), compute the minimum $d(x, s) + d(s, y)$ over all $s \in S$. This takes $O(n^{2.5} \log n)$ time.

  Now we prove the claim made in option 2.

Proof. Let w be a high degree vertex. We want to find $P(\text{none of } w\text{'s neighbors are sampled})$.

$$\begin{aligned} P(\text{none of } w\text{'s neighbors are sampled}) &\leqslant P(\text{given sample misses})^{|S|} \\ &= \left(1 - \frac{1}{\sqrt{n}}\right)^{100\sqrt{n}\log n}. \end{aligned}$$

Now we use the fact that $\left(1 - \frac{1}{n}\right)^n \leqslant \frac{1}{e}$. So now,

$$\left(1 - \frac{1}{\sqrt{n}}\right)^{100\sqrt{n}\log n} \leqslant \frac{1}{e^{100\log n}} = \frac{1}{n^{100}}.$$

This is the probability of w screwing up. The probability of any high degree vertex screwing up is $\leqslant \frac{n}{n^{100}} = \frac{1}{n^{99}}$ by union bound. 😊