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 $c : V \longrightarrow \{1, 2, 3\}$ 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 $\text{poly}(n)$ 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 $\text{E}[t_2] \geqslant \text{E}[t_1]$.

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

Lemma 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(none of w'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. 😊

2.3 Global Min-Cut

We start by stating the question we are trying to solve.

Question: Global Min-Cut Problem

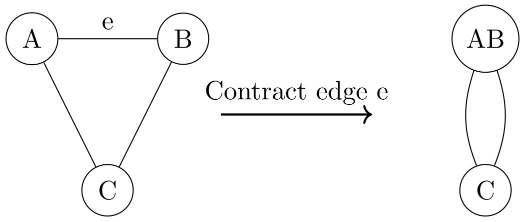
We are given an unweighted and undirected graph $G = (V, E)$. A "cut" of the graph is a partition of the vertices into two sets such that each set is non-empty. The "cost" of the cut (also called the cut "size") is the number of edges that have one endpoint in each set. The goal is to find the cut with the smallest cost.

Let $n := |V|$. We consider two randomized algorithms:

- Karger's Algorithm (1995) - poly(n)-time.
- Karger-Stein Algorithm (1996) - $\tilde{O}(n^2)$ time.

2.3.1 Edge Contraction

The key idea in both algorithms is edge contraction. Given an edge (u, v), we can contract it by removing the edge and merging the two vertices into a single vertex. The new vertex will have all the edges that u and v had. For example:



Each contraction reduces the number of vertices by 1, and we want to stop when we have two vertices left. This leaves us with two vertices and many edges between them.

Note:
We allow multi-edges.

Question: Should we keep self-edges?
Answer: No. Delete them.
The observation we make is that we win iff edges at the end are a min-cut.

2.3.2 Karger's Algorithm

The algorithm is as follows:

- Randomly pick an edge (u, v).
- Contract (u, v).
- Repeat until 2 vertices are left.

Lenma 2.3.1
For any min-cut C, the probability that Karger's algorithm finds C is $\geqslant \frac{1}{\text{poly}(n)}$.

Proof. Let $e_1, e_2, \ldots$ be the edges we contract. Let's start by considering $P(e_1 \in C)$.

We first have $P(e_1 \in C) = \frac{|C|}{|E|}$, but there's something better to notice. Let d be the minimum degree of any vertex in G. Then $|C| \leqslant d$ and $|E| \geqslant \frac{dn}{2}$. So $P(e_1 \in C) \leqslant \frac{2}{n}$ and $P(e_1 \notin C) \geqslant 1 - \frac{2}{n}$.

Now consider $P(e_2 \notin C \mid e_1 \notin C)$. By the same analysis, we have $P(e_2 \notin C \mid e_1 \notin C) \geqslant 1 - \frac{2}{n-1}$. This is because after we contract e_1, we have $n-1$ vertices left. So, C is still a min-cut with respect to the remaining vertices. Set d as the new minimum degree, and we have $|C| \leqslant d$ and $|\text{remaining edges}| \geqslant \frac{d(n-1)}{2}$. Then:

$$P(e_2 \in C \mid e_1 \notin C) = \frac{|C|}{|\text{remaining edges}|} \leqslant \frac{d}{\frac{d(n-1)}{2}} = \frac{2}{n-1}.$$

Now:

$$\begin{aligned} P(e_3 \notin C | e_1, e_2 \notin C) &\geqslant 1 - \frac{2}{n-2} \\ &\vdots \\ P(e_k \notin C | e_1, \dots, e_{k-1} \notin C) &\geqslant 1 - \frac{2}{n-k+1}. \end{aligned}$$

So,

$$\begin{aligned} P(\text{we find } C) = P(e_1, \dots, e_{n-2} \notin C) &\geqslant \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\dots\left(\frac{1}{3}\right) \\ &\approx \frac{2}{n(n-1)}. \end{aligned}$$

☺

Conclusion: The algorithm succeeds with probability $\geqslant \frac{2}{n(n-1)}$.
New Goal: Find min-cut with $\geqslant 1 - \frac{1}{n^2}$ probability.
Idea: Repeat t times and return the best cut we find. (We will determine the value for t below.)

The probability that all t trials fail is $\leqslant \left(1 - \frac{2}{n(n-1)}\right)^t$. We want this to be $\leqslant \frac{1}{n^2}$. We utilize the inequality below:

$$\left(1 - \frac{1}{k}\right)^k \leqslant \frac{1}{e} \leqslant \left(1 - \frac{1}{k}\right)^{k-1}.$$

So,

$$\begin{aligned} \left(1 - \frac{2}{n(n-1)}\right)^t &\leqslant \frac{1}{e^{t \cdot \frac{2}{n(n-1)}}} \implies t = 2\binom{n}{2}\log n \\ &\leqslant \frac{1}{e^{2\log n}} = \frac{1}{n^2}, \end{aligned}$$

as desired.

2.3.3 Karger-Stein Algorithm

We start with the following motivation. We have:

$$P(\text{first } n/2 \text{ contractions succeed}) \geqslant \frac{n-2}{n}\frac{n-3}{n-1}\dots\frac{n/2-1}{n/2+1} \approx \frac{1}{4}.$$

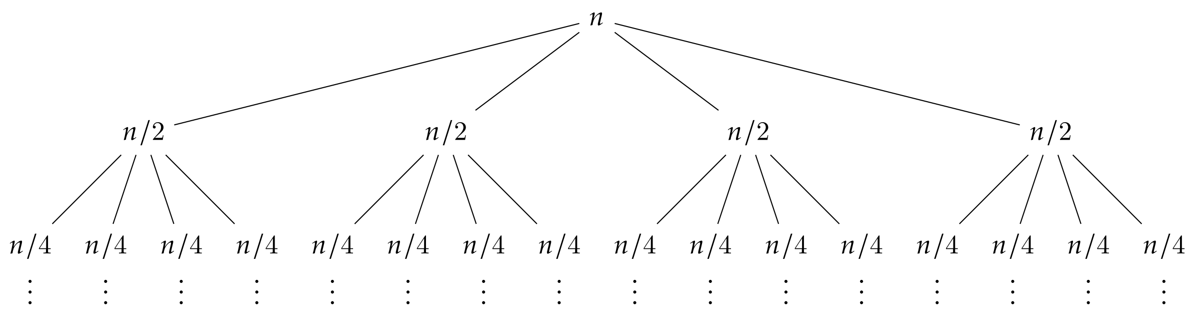
This is because after we telescope, the bottom two terms are both near n while the top two terms are near $n/2$. So now, if the first $n/2$ contractions succeed, what's the probability that the next $n/4$ succeed?

$$P(\text{next } n/4 \text{ contractions succeed}) \geqslant \frac{n/2-2}{n/2}\frac{n/2-3}{n/2-1}\dots\frac{n/4-1}{n/4+1} \approx \frac{1}{4}.$$

And this pattern continues.

We now go over the Karger-Stein algorithm:

- Represent the problem as a tree.
- For the first $n/2$ contractions, run Karger's algorithm 4 times.
- For the next $n/4$ contractions, run Karger's algorithm 16 times.
- So on, until the base case of the algorithm which is when there are two vertices left.



n
$n/2$
$n/2$
$n/2$
$n/2$
$n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$ $n/4$

Time Analysis: The first level takes $\tilde{O}(n^2)$ time. The second level takes $\tilde{O}\left(\frac{n^2}{2}\right)$ time. The third level takes $\tilde{O}\left(\frac{n^2}{4}\right)$ time. So the total time is $\tilde{O}(n^2)$.
So, at level i:

- There are 4^i edges.
- The work per edge $\leqslant$ (|remaining vertices|)$^2 = \tilde{O}\left(\frac{n}{2^i}\right)^2 = \tilde{O}\left(\frac{n^2}{4^i}\right)$.

This implies that the cost of the ith level is less than $\tilde{O}(n^2)$. Summing, we get a total cost $\tilde{O}(n^2)$.

Now we consider the probability that our algorithm succeeds. Again, we have a 4-ary tree with a depth of $t = O(\log n)$. Each edge flips a coin where $P(\text{heads}) = \frac{1}{4}$. So what we want is a t length chain of all heads. We can show the following result:

$$P(\text{good path}) \geqslant \frac{1}{t} \geqslant \Omega\left(\frac{1}{\log n}\right).$$

The goal is an algorithm that succeeds with probability greater than $1 - \frac{1}{n^2}$.
Exercise: $O(\log^2 n)$ repetitions suffice.

2.4 Galton Walton Tree

Building on the previous problem, we introduce the concept of a Galton Walton tree. This is a 4-ary tree with t levels where each edge flips a biased coin. The probability of heads is $1/4$. A root-to-leaf path in this tree is "successful" if all edges are heads.
Claim: $P(\text{successful path}) \geqslant \frac{1}{t}$.

Proof. Pretend the paths are independent. Then we have that

$$P(\text{No successful path}) = \left(1 - \frac{1}{4^t}\right)^{4^t} \leqslant \frac{1}{e}.$$

This is not a very interesting result. It is worth noting that the paths are extremely dependent on each other. Consider a path of length $t-2$ and creating two separate paths by diverging at the last edge. This should illustrate the dependence of the paths.

Now let f_t the probability of tailing in a tree with t levels. We can define this as a recurrence on f_{t-1}. We have:

$$f_t = \left(\frac{1}{4}f_{t-1} + \frac{3}{4}\right)^4.$$

To proceed with induction on t, we first prove the following identity:

$$1 - 1/k < \frac{1}{e^{1/k}} < 1 - 1/(k+1).$$

This is a useful identity to have in our back pocket. We have:

$$\begin{aligned}
&(1-1/k)^k < 1/e < (1-1/k)^{k-1} \\
\implies &1 - 1/k < 1/e^{1/k} \\
&1/e^{1/(k-1)} < 1 - 1/k \implies 1/e^{1/k} < 1 - 1/(k+1).
\end{aligned}$$

Putting it all together, we get the desired result. We can now proceed with induction.

Our hypothesis: $f_{t-1} \leqslant 1 - \frac{1}{t-1}$. What we want to show is that $f_t \leqslant 1 - \frac{1}{t}$.

We have:

$$\begin{aligned}
f_t &\leqslant \left(\frac{3}{4} + \frac{1}{4}\left(1 - \frac{1}{t-1}\right)\right)^4 \\
&= \left(1 - \frac{1}{4(t-1)}\right)^4 \\
&\leqslant \left(\frac{1}{e^{\frac{1}{4(t-1)}}}\right)^4 \\
&= \frac{1}{e^{1/(t-1)}} && \text{(by the above identity)} \\
&< 1 - \frac{1}{t}. && \text{(by the above identity)}
\end{aligned}$$

This completes the induction, as well as the entire proof. 😄

Chapter 3

Lovasz Local Lemma

Lemma 3.0.1 Lovasz Local Lemma

Let $A_1, A_2, \ldots, A_n$ be "bad events." Suppose:

- $P(A_i) < p$ for some p.
- Each A_i has a "dependency set" $D_i \subseteq \{A_1, A_2, \ldots, A_n\}$ such that A_i is independent of $\{A_1, A_2, \ldots, A_n\} \setminus D_i$ and $|D_i| \leqslant d$.

If $p < \frac{1}{ed}$, then there exists an outcome where none of the A_i occur. That is, $P(\text{no bad events}) \geqslant \frac{1}{d^n}$.

Application: KSAT. In KSAT, we have a bunch of boolean variables $x_1, x_2, \ldots$. A K-SAT formula is a conjunction of n clauses, where each clause selects k unique variables that cannot take a specific form. For example, if we have $k = 3$, we might have a clause that says $(x_1, x_2, x_3) \neq (0, 1, 0)$. The question is whether we can find a satisfying assignment of these variables.

Theorem 3.0.1

Suppose each clause overlaps in variables $\leqslant d - 1$ other clauses. Also suppose $\frac{1}{2^k} < \frac{1}{ed}$. Then, there exists a satisfying assignment.

Proof. $x_1, x_2, \ldots$ are random $0, 1$ variables. Let A_i be the event that clause i is not satisfied. Then $P(A_i) \leqslant \frac{1}{2^k}$. By the LLL, there exists a satisfying assignment. 😊

3.1 Algorithmic Lovasz Local Lemma

3.1.1 Algorithmic Setting

We have random variables $X_1, X_2, \ldots, X_n$ and events A_i that are determined by some subset of the X_is. We can think about this as a dependency graph where all the X_i and A_i are nodes, and edges are drawn between X_i and A_j if X_i determines A_j.

Lemma 3.1.1 Algorithmic Lovasz Local Lemma

Suppose each A_i overlaps with at most $d - 1$ other A_js on which X_is they use. Suppose $P(A_i) < \frac{1}{4d}$. Then there exists a polynomial time algorithm* that finds an outcome of X_is such that none of the A_is occur.

3.1.2 Moser's Fix-it Algorithm

Algorithm:

- Sample $X_1, X_2, \ldots, X_m$.
- While $\exists$ bad event A_i:
  - Pick A_i arbitrarily.
  - Resample the X_i's that determine A_i.

Theorem 3.1.1 Moser's Theorem (KSAT)
Consider KSAT:

- m vars, $X_1, \ldots, X_m$.
- n clauses $C_1, \ldots, C_n$.
- Each C_i depends on $\leqslant d$ other C_i's.

If $8d < 2^k$, then Moser's algorithm performs $t = O(n)$ (really $O\left(\frac{n}{d}\right)$) fix-its in expectation.

Proof Idea: Use random bits, for example $R = 0010010100101\ldots$. Construct a "transcript" T such that if you tell me T, I can recover all the random bits you used. The interesting property of T is that:

$$|T| \leqslant (\text{number of fix-its}) \cdot (2 \lg d) + O(n).$$

The number of random bits we use is less than the number of fix-its times k. We know that $k > (2 + \lg d) \cdot 100$.

If Moser's algorithm runs for a long time, say $> 100n$ fix-its, then

$$|T| \leqslant 100n \cdot (2 \lg d) + O(n).$$

Then,

$$\begin{aligned}\text{number of random bits} &= 100n \cdot k \\ &> 100n \cdot (2 + \lg d) \cdot 100.\end{aligned}$$

This means $|T| << \text{number of random bits}$. So if we can show that such a T exists, then we can show that the expected number of fix-its to be pretty small.

3.1.3 Proof of Moser's Theorem

Proof. We start by defining variables we are going to use:

- R = random bits used by the algorithm.
- M_1 = final values of X_i's.
- M_2 = sequence of which clauses we fix.

Lenma 3.1.2
M_1 and M_2 fully determine R.

Proof. Work backwards! Consider the last C_i that was fixed. We know its variables after the fix and the variables before the fix. So we can "undo" the fix. We can keep doing this until we get to the beginning. ☺

Lemma 3.1.3

M_2 can be compressed to $n + t(2 + \lg d)$ bits.

We'll proceed by assuming the lemma first. So now we have $|M_1| = m$ and $|R| = m + tk$. Also since $8d < 2^k$¡ we have $3 + \lg d \leqslant k$. So, $|R| \geqslant m + t(3 + \lg d)$.

Now,

$$\begin{aligned} |M_1| + |M_2| &= n + m + t(2 + \lg d) \\ &\leqslant |R| + n - t. \end{aligned}$$

If $E[t] > n$, then $E[|M_1 \circ M_2|] < E[|R|]$. This is a contradiction by the incompressibility of random bits. So, $E[t] \leqslant n$ as desired. 😊

Now we return to proving the second lemma.

Proof. We start by defining processes.
Define process(C_i):

- Resample C_i's variables.
- While $\exists$ another C_j that is broken and depends on C_i:
  - Select such a C_j.
  - process(C_j).

Define Algorithm:

- For $i = 1, \ldots, n$:
  - If C_i is broken, process(C_i).

Claim: When the algorithm finishes, all clauses holds.
Claim: Algorithm terminates with probability 1.
We now look towards constructing M_2:

- A = Part 1: n bits, which C_i's get fixed in for loop.

Now recall the process algorithm. Notice here that for each C_j, there are only $d + 1$ options. This means that we can communicate C_j with $\lg(d + 1)$ bits, which we'll call q. We now rewrite process considering a string B.
Define process(C_i):

- Resample C_i's variables.
- While $\exists$ another C_j that is broken and depends on C_i:
  - Select such a C_j.
  - $B = B +$ "1" $+ q$.
  - process(C_j).
- $B = B +$ "0".

So given M_2, A lets us recover which clauses the for loop fixes. B lets us figure out which clauses are fixed within each process subroutine. This means we can recover the entire sequence of clause fixes that the algorithm performs.

So, $|A| = n$ and $|B| = t(2 + \lg d)$. This means $|A \circ B| = n + t(2 + \lg d)$, which completes the proof. 😊

Chapter 4

Chernoff Bounds

We start with an example. Suppose I flip 100 fair coins. What is the probability that the number of heads is greater than or equal to 75? Turns out,

$$P(\text{heads} \geqslant 75) \approx \frac{1}{3.5 \times 10^6}.$$

Theorem 4.0.1

Consider n fair coin flips, $X_1, X_2, \ldots, X_n$ iid with $X_i = \begin{cases} 0 & \text{w.p. } 1/2 \\ 1 & \text{w.p. } 1/2 \end{cases}$. Let $X = \sum X_i$. Then,

$$P\left(X > \frac{n}{2} + K\sqrt{n}\right) \leqslant e^{-K^2/2}.$$

Corollary 4.0.1

$$P\left(X \geqslant \frac{3}{4}n\right) \leqslant e^{-n/32}$$

You don't actually need $X_1, \ldots, X_n$ to be independent. It suffices to have that, for any i and any outcomes of $X_1, \ldots, X_{i-1}$,

$$P(X_i = 1 \mid X_1, \ldots, X_{i-1}) \leqslant 1/2.$$

Note:

"Azuma's Inequality" is a citable Chernoff bound. Alternatives include "multiplicative version of Azuma's inequality" and "Freedman's inequality."

We turn to the proof of the theorem.

Proof. Idea: suppose we have a function f such that:

- f is non-negative.
- f is increasing on $[n/2, n]$.

Then,

$$P\left(x \geqslant \frac{n}{2} + K\sqrt{n}\right) \leqslant P\left(f(x) \geqslant f\left(\frac{n}{2} + K\sqrt{n}\right)\right) \leqslant \frac{E[f(x)]}{f\left(\frac{n}{2} + K\sqrt{n}\right)}.$$

Example 4.0.1 (Trying f's)

$f(x) = \left(x - \frac{n}{2}\right)^2$. This clearly satisfies the requirements. This tells us that:

$$\begin{aligned} P\left(x > \frac{n}{2} + K\sqrt{n}\right) &\leqslant \frac{E[f(x)]}{K^2 n} \\ &= \frac{\mathrm{Var}(X)}{K^2 n}. \end{aligned}$$

We use the linearity of variance. We have $\mathrm{Var}(X_i) = \frac{1}{4}$. So, $\mathrm{Var}(X) = \frac{n}{4}$. Thus,

$$P\left(x > \frac{n}{2} + K\sqrt{n}\right) \leqslant \frac{1}{4k^2}.$$

This is exactly Chebyshev's inequality.

We now find an f that proves our result. Try $f(x) = e^{\lambda x}$ for some $\lambda \in (0,1)$. This is approximately $(1+\lambda)^x$. We first find the expectation:

$$\begin{aligned} E[f(x)] = E[e^{\lambda \sum_i X_i}] &= E\left[\prod_i e^{\lambda X_i}\right] \\ &= \prod_i E[e^{\lambda X_i}] && \text{(by independence)} \\ &= \left(\frac{1+e^\lambda}{2}\right)^n. \end{aligned}$$

Fact:

$$\begin{aligned} e^\lambda &= 1 + \lambda + \frac{\lambda^2}{2!} + \frac{\lambda^3}{3!} + \cdots && (4.1) \\ &\leqslant 1 + \lambda + \lambda^2 \cdot \left(\frac{1}{2!} + \frac{1}{3!} + \cdots\right) && (4.2) \\ &\leqslant 1 + \lambda + \lambda^2. && (4.3) \end{aligned}$$

It follows that

$$E[e^{\lambda X_i}] \leqslant 1 + \frac{1}{2}\lambda + \frac{1}{2}\lambda^2.$$

Now we make use of the other fact that $1 + x \leqslant e^x$. So

$$\begin{aligned} E[e^{\lambda X_i}] &\leqslant 1 + \frac{1}{2}\lambda + \frac{1}{2}\lambda^2 \\ &\leqslant e^{\lambda/2 + \lambda^2/2}. \end{aligned}$$

Thus,

$$E[f(x)] \leqslant e^{n(\lambda/2 + \lambda^2/2)}.$$

So now,

$$\begin{aligned} P\left(X > \frac{n}{2} + K\sqrt{n}\right) &\leqslant \frac{E[f(x)]}{f\left(\frac{n}{2} + K\sqrt{n}\right)} \\ &\leqslant \frac{e^{n(\lambda/2 + \lambda^2/2)}}{e^{n\lambda/2 + \lambda K\sqrt{n}}} \\ &= e^{n\lambda^2/2 - \lambda K\sqrt{n}}. \end{aligned}$$

Now we try to find an appropriate value for λ. We try:

$$\lambda K\sqrt{n} = n\lambda^2$$
$$\frac{K}{\sqrt{n}} = \lambda.$$

This yields:

$$e^{n\lambda^2/2-\lambda K\sqrt{n}} = e^{-\lambda K\sqrt{n}/2} = e^{-k^2/2},$$

as desired. 😊

4.1 Quicksort Algorithm (1961)

We review the algorithm for quicksort:

- Pick random pivot P.
- Partition elements into 2 pieces: elements smaller/larger than P.
- Recurse on the pieces.

Theorem 4.1.1
The running time of quicksort satisfies $T = O(n \lg n)$ with probably at least $1 - \frac{1}{n^2}$.

Proof. Note that the time is $\leqslant$ #levels $\cdot\, O(n)$. So what we have to prove is that with high probability, the depth of the algorithm doesn't exceed $\lg n$. So,

$$\begin{aligned} P(\text{max depth} \geqslant 1000\lg n) &\leqslant P(\exists\text{element with max depth} \geqslant 1000\lg n) \\ &\leqslant n \cdot P(\text{a given } v \text{ has depth} \geqslant 1000\lg n). \end{aligned}$$

We want to show that the probability on the last line is $\leqslant \frac{1}{n^3}$. We have that

$$P(\text{next subproblem} \leqslant 3/4\text{current size}) \geqslant \frac{1}{2}.$$

Let $P_1, P_2, \ldots$ be subproblems in v's path. Let

$$X_i = \begin{cases} 1 & P_{i+1}\text{exists and } |P_{i+1}| > \frac{3}{4}|P_i| \\ 0 & \text{otherwise} \end{cases}$$

We observe that $P(X_i = 1 \mid X_1, \ldots, X_{i-1}) \leqslant 1/2$. So by Chernoff bound, if we look at $X = X_1 + X_2 + \cdots + X_{1000\lg n =: m}$, we get:

$$P\left(X \geqslant \frac{3}{4}m\right) \leqslant e^{-m/32} <<< \frac{1}{n^3}.$$

If $X \leqslant \frac{3}{4}m$, either P_m doesn't exist or $|P_m| \leqslant n \cdot \left(\frac{3}{4}\right)^{m/4} = n \cdot \left(\frac{3}{4}\right)^{250\log n} << 1$, which is a contradiction. This means P_m in fact does not exist. As such, we are able to conclude that we have a running time of $O(n \lg n)$ with the desired probability. 😊