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 $\mathrm{E}[t_2] \geqslant \mathrm{E}[t_1]$.

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 $\text{poly}(n)$ 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.

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. 😊

2.3 Global Min-Cut

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

Question: Global Min-Cut Problem

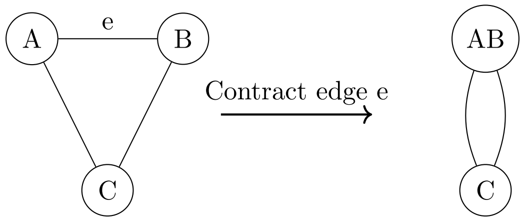
We are given an unweighted and undirected graph $G = (V, E)$. A "cut" of the graph is a partition of the vertices into two sets such that each set is non-empty. The "cost" of the cut (also called the cut "size") is the number of edges that have one endpoint in each set. The goal is to find the cut with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

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

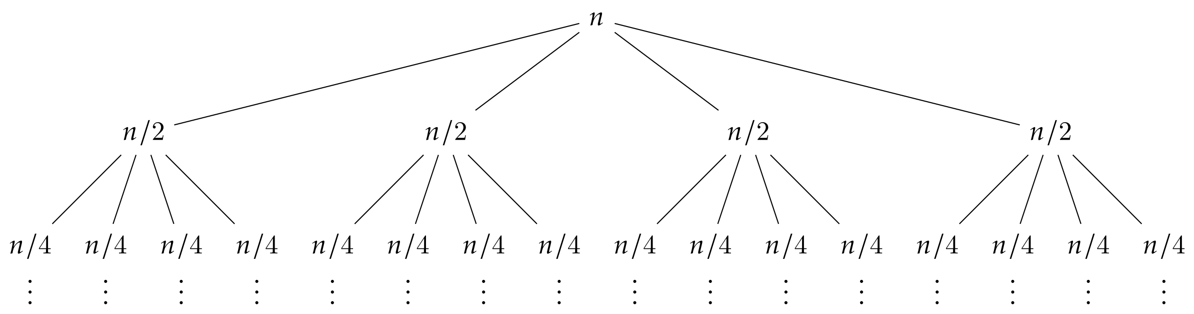
This is because after we telescope, the bottom two terms are both near n while the top two terms are near $n/2$. So now, if the first $n/2$ contractions succeed, what's the probability that the next $n/4$ succeed?

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

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.



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 $\leqslant (|\text{remaining vertices}|)^2 = \tilde{O}\left(\frac{n}{2^i}\right)^2 = \tilde{O}\left(\frac{n^2}{4^i}\right)$.

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

Lenma 3.0.1 Lovasz Local Lemma

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

- $P(A_i) < p$ for some p.
- Each A_i has a "dependency set" $D_i \subseteq \{A_1, A_2, \dots, A_n\}$ such that A_i is independent of $\{A_1, A_2, \dots, A_n\} \setminus D_i$ 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 $x_1, x_2, \dots$. 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. $x_1, x_2, \dots$ 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 $X_1, X_2, \dots, X_n$ 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 $X_1, X_2, \dots, X_m$.
- 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, $X_1, \dots, X_m$.
- n clauses $C_1, \dots, C_n$.
- 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 $R = 0010010100101\dots$. 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,

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

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:

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

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

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

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. 😊

4.2 Revisiting Theorem 4.0.1

We start with warm-ups to give an alternative proof to Theorem 4.0.1.

4.2.1 Poor Man's Chernoff Bound

Theorem 4.2.1

$$P(X \geqslant 2K\sqrt{n}) \leqslant \frac{1}{2^K}.$$

Proof. We make use of the core facts below:
Core fact 1: $P(X \geqslant 2\sqrt{n}) \leqslant \frac{1}{4}$, by Chebyshev's inequality.
Core fact 1': $P(\text{any prefix has sum} \geqslant 2\sqrt{n}) \leqslant \frac{1}{2}$. This is a direct result of **Core fact 1**, and the implication will be shown in the homework.

We can graph the running sum as a function of time and track the times that we hit the values $2\sqrt{n}$, $4\sqrt{n}$, and $6\sqrt{n}$ and call them t_1, t_2, t_3, et cetera. What the core fact is saying is that $P(t_1 \text{ exists}) \leqslant \frac{1}{2}$. Then, we can build the the following chain:

$$\begin{aligned} P(t_1 \text{ exists}) &\leqslant \frac{1}{2} \\ P(t_2 \text{ exists} \mid t_1 \text{ exists}) &\leqslant \frac{1}{2} \\ &\vdots \\ P(t_i \text{ exists} \mid t_{i-1} \text{ exists}) &\leqslant \frac{1}{2}. \end{aligned}$$

So,

$$P(X \geqslant 2K\sqrt{n}) \leqslant P(t_K \text{ exists}) \leqslant \frac{1}{2^K},$$

as desired. 😆

4.2.2 Chernoff Bound for Geometric Random Variables

Theorem 4.2.2
Let $X_1, \dots, X_n$ be non-negative independent random variables such that for all $j \in \mathbb{N}$,

$$P(X_i \geqslant j) \leqslant p^j.$$

Then $X = \sum X_i$ satisfies

$$P(X \geqslant 2n) \leqslant (4p)^n.$$

Proof. $X \geqslant 2n \implies \sum \lfloor X_i \rfloor \geqslant n \implies \exists \langle Y_1, \dots, Y_n \rangle =: \vec{Y}$ such that

- $\sum Y_i = n$.
- $X_i \geqslant Y_i \, \forall i$.

For a given option $\langle Y_1, \ldots, Y_n \rangle$ for $\vec{Y}$,

$$\begin{aligned} P(\vec{Y} \text{ occurs}) &= P(X_i \geqslant Y_i \forall i) \\ &\leqslant \prod_i P(X_i \geqslant Y_i) \\ &\leqslant \prod_i p^{Y_i} \\ &= p^{\sum_i Y_i} \\ &= p^n. \end{aligned}$$

Now we bound the number of options for $\vec{Y}$. We can express $\vec{Y}$ as a binary string, for example:

$$\underbrace{000}_{Y_1=3}\ 1\ \underbrace{0}_{Y_2=1}\ 1\ \underbrace{\quad}_{Y_3=0}\ 1\ldots$$

So the number of 0's is $\sum Y_i = n$ and the number of 1's is n. This means that the number o options satisfying $\sum Y_i = n$ is less than the number of binary strings of length $2n$ which is 4^n.

So,

$$\begin{aligned} P(X \geqslant 2n) \leqslant P(\text{some } \vec{Y} \text{ occurs}) \\ &\leqslant (\# \text{ of options for } \vec{Y}) \cdot P(\vec{Y} \text{ occurs}) \\ &\leqslant 4^n \cdot p^n \\ &= (4p)^n. \end{aligned}$$

😄

4.2.3 Anti-(Chernoff Bound)

Theorem 4.2.3

$$P\left(X \geqslant \frac{K}{4}\sqrt{n}\right) \geqslant \frac{1}{4^{K^2}}$$

Proof. Break the coins into K^2 groups. So each group has $m = \frac{n}{K^2}$ coins.
Core fact: $P\left(X \geqslant \frac{1}{4}\sqrt{n}\right) \geqslant \frac{1}{4}$.

Let C_i be the sum of the i-th group. So by the core fact,

$$P\left(C_i \geqslant \frac{1}{4}\sqrt{m}\right) \geqslant \frac{1}{4}.$$

So,

$$P\left(\text{every } C_i \geqslant \frac{1}{4}\sqrt{m}\right) \geqslant \frac{1}{4^{K^2}}$$

If this occurs, then

$$X = \sum C_i \geqslant K^2 \cdot \left(\frac{1}{4}\sqrt{m}\right) = \frac{K}{4}\sqrt{n}.$$

😄

4.2.4 Real Chernoff Bound

Theorem 4.2.4

$$P(X \geqslant 16k\sqrt{n}) \leqslant \frac{1}{2^{K^2}}.$$

Proof. We start by defining $Y_i = \max\left(0, \frac{C_i}{8\sqrt{m}}\right)$. We can apply the Poor Man's Chernoff bound to each C_i to get

$$\begin{aligned} P(C_i \geqslant 2\ell\sqrt{m}) &\leqslant \frac{1}{2^\ell} \\ \implies P\left(Y_i \geqslant \frac{2\ell\sqrt{m}}{8\sqrt{m}}\right) &\leqslant \frac{1}{2^\ell} \\ \implies P(Y_i \geqslant \ell) &\leqslant \frac{1}{2^{4\ell}} \leqslant \frac{1}{16^\ell}. \end{aligned}$$

This is saying that Y_i are geometric random variables. So by the geometric random variable bound,

$$P\left(\sum Y_i \geqslant 2K^2\right) \leqslant (4 \cdot 1/16)^{K^2} = \left(\frac{1}{4}\right)^{K^2}.$$

But also

$$\begin{aligned} X \geqslant 16K\sqrt{n} &\implies \sum C_i \geqslant 16K\sqrt{n} \\ &\implies \sum Y_i \geqslant \frac{16K\sqrt{n}}{8\sqrt{m}} = 2K^2. \end{aligned}$$

😄

Theorem 4.2.5
Let $X_1, X_2, \dots, X_n$ be independent $\{0,1\}$ coin flips with $P(X_i = 1) = p$ for some $p \leqslant \frac{1}{2}$. Define $X = \sum X_i$, $\mu = E[X] = np$.
Small-deviation regime: For $K \in \{1, \dots, \sqrt{\mu}\}$,

$$P(X \geqslant \mu + K\sqrt{\mu}) \leqslant e^{-\Theta(K^2)}.$$

Large-deviation regime: For $J \geqslant 2$ such that $J\mu \leqslant n$,

$$P(X \geqslant J\mu) \leqslant \frac{1}{J^{\Theta(J\mu)}}.$$

Proof. (large-deviation case)
<u>Warmup 1</u>

Theorem 1 (Poor Man's Chernoff Bound) If $\mu \leqslant 1$, then

$$P(X \geqslant K) \leqslant \mu^K.$$

Proof. **Core fact** $P(X \geqslant 1) \leqslant \mu$ by Markov's inequality.
Now repeat the ideas with $t_1, t_2, \dots$ to say that $P(X \text{ reaches } i) = \mu^i$. 😄

Warmup 2 This was already done prior as it is the exact same.

Warmup 3

Theorem 3

$$P(X \geqslant J) \geqslant \frac{1}{J^{O(J\mu)}}$$

Proof. We'll have $J\mu$ groups where each group has mean $\frac{1}{J}$.
Core fact: If $\mu \leqslant 1$, then $P(X \geqslant 1) \geqslant \Omega(\mu)$.

By the core fact, each C_i is $\geqslant 1$ with probability $\geqslant \Omega\left(\frac{1}{J}\right)$. So,

$$P(\text{every } C_i \geqslant 1) = P(X \geqslant J\mu) \geqslant \Omega\left(\frac{1}{J}\right)^{J\mu}.$$

"Squiggly square to signify almost proof done." ☺

So the Poor Man's Chernoff Bound implies $P(C_i \geqslant K) \leqslant \frac{1}{J^K}$. Then by Chernoff for geometric random variables,

$$P\left(\sum C_i \geqslant 2J\mu\right) \leqslant \left(\frac{4}{J}\right)^{J\mu}$$

☺

4.3 More Chernoff Bounds

Theorem 4.3.1 Chernoff Bound
Let $X_1, X_2, \ldots, X_n \in [0,1]$ be independent random variables. Let $X = \sum X_i$ and $\mu = E[X]$. Then, for $k \in O(\sqrt{\mu})$,

$$P(|X - \mu| \geqslant k\sqrt{\mu}) \leqslant e^{-\Omega(k^2)}.$$

And for $J \geqslant 2$,

$$P(X \geqslant J\mu) \leqslant \left(\frac{1}{J}\right)^{\Omega(J\mu)}.$$

Small-deviation regime:

- There are k^2 chunks.
- So each has expected value $\approx \frac{\mu}{k^2}$.
- If each exceeds expected value by $\sqrt{\frac{\mu}{k^2}}$, then the total sum is $\geqslant k^2 \cdot \sqrt{\frac{\mu}{k^2}} = k\sqrt{\mu}$.

Large-deviation regime:

- $J\mu$ chunks.
- Each chunk has expected value $\approx \frac{1}{J}$.
- If each chunk is at least 1, then the total sum is at least $J\mu$.

Theorem 4.3.2 Bennett's Inequality, Bernstein's Bound

Same bound, but let $v = \text{Var}(X)$.

$$P(X - E[X] \geqslant k\sqrt{v}) \leqslant e^{-\Theta(k)} \text{ for } k \in \{1, 2, \dots, \sqrt{v}\}$$
$$P(X - E[X] \geqslant Jv) \leqslant \left(\frac{1}{J}\right)^{\Omega(Jv)} \text{ for } J \geqslant 2.$$

4.3.1 Adaptive Version

Theorem 4.3.3 Azuma's Inequality

Let $X_1, X_2, \dots, X_n \in [-1, 1]$. Suppose $E[X_i | X_1, \dots, X_{i-1}] = 0$. Then, $X = \sum X_i$ satisfies

$$P(X \geqslant K\sqrt{n}) \leqslant e^{-\Omega(K^2)}.$$

Fancier versions:

Have random variables $X_1, X_2, \dots, X_n \in [0, 1]$ which are revealed one after the other. After $X_1, \dots, X_{i-1}$ are revealed, Alice gets to select the distribution P_i for X_i and then X_i is revealed from P_i.

Let $\mu_i = E[X_i \mid P_i]$, $v_i = \text{Var}(X_i \mid P_i)$, $X = \sum X_i$, $\mu = \sum \mu_i$ and $v = \sum v_i$. As a corollary to Azuma's inequality,

$$P(X \geqslant \mu + k\sqrt{n}) \leqslant e^{-\Omega(k^2)}.$$

Now if μ is deterministically at most $\bar{\mu}$, then

$$P(X \geqslant K\sqrt{\bar{\mu}} + \bar{\mu}) \leqslant e^{-\Omega(K^2)}$$
$$P(X \geqslant J\bar{\mu}) \leqslant \left(\frac{1}{J}\right)^{\Omega(J\bar{\mu})}.$$

Theorem 4.3.4 Freedman's Inequality

If v is deterministcally at most $\bar{v}$, then

$$P(X \geqslant K\sqrt{\bar{v}} + \mu) \leqslant e^{-\Omega(K^2)} \text{ for } K \in \{1, 2, \dots, \sqrt{v}\}$$
$$P(X \geqslant \mu + J\bar{v}) \leqslant \left(\frac{1}{J}\right)^{\Omega(J\bar{v})} \text{ for } J \geqslant 2.$$

Note:

Bill has used the following bound in ~ 90% of his papers.
"Sometimes when you don't need this bound you want to find a way to use it anyway."

Theorem 4.3.5 McDiarmid's Inequality

Let $X_1, X_2, \dots X_n$ be independent. Let $F : X_1, \dots, X_n \longrightarrow \mathbb{R}$. Suppose: If I change some X_i to a new value $\bar{X}_i$,

$$|F(X_1, \dots, X_i, \dots, X_n) - F(X_1, \dots, \bar{X}_i, \dots, X_n)| \leqslant 1.$$

Then,

$$P(|F - E[F]| \geqslant K\sqrt{n}) \leqslant e^{-\Omega(K^2)}.$$

Example 4.3.1

Consider a random graph:

- n vertices.
- each (i, j) is an edge with probability $\frac{1}{2}$.

Consider the chromatic number $\chi(G)$. There's a way to show that

$$P(|\chi(G) - E[\chi(G)]| \geqslant k\sqrt{n}) \leqslant e^{-\Omega(k^2)}.$$

We cannot define the random variables X_i on whether an edge is included or not. This is because we end up getting n^2 random variables, which makes the inequality flop. We proceed by defining $X_1, X_2, \ldots, X_n$ where X_i is the number of edges from vertex i to vertices $> i$. Then the result follows directly from McDiarmid's inequality.

Proof Idea: Let

$$\begin{aligned}
Y_0 &= E[F] \\
Y_1 &= E[F \mid X_1] \\
Y_2 &= E[F \mid X_1, X_2] \\
&\vdots \\
Y_n &= E[F \mid X_1, \ldots, X_n] = F.
\end{aligned}$$

Then let

$$\begin{aligned}
Z_1 &= Y_1 - Y_0 \\
Z_2 &= Y_2 - Y_1 \\
&\vdots \\
Z_n &= Y_n - Y_{n-1}.
\end{aligned}$$

Claim 1: $E[Z_i \mid Z_1, \ldots, Z_{i-1}] = 0$.
Claim 2: $|Z_i| \leqslant 1$.

So by Azuma, $P(\sum Z_i > K\sqrt{n}) \leqslant e^{-\Omega(K^2)}$. But $\sum Z_i = Y_n - Y_0 = F - E[F]$.

Chapter 5

Oblivious Routing

We start with an n-node hypercube:

- n vertices: $1, \dots n$.
- edges: For each pair i, j if i, j differ in exactly one bit, then we have edges (i, j) and (j, i).

This graph contains $n \log n$ edges. Each vertex i wants to send a message to another vertex $\pi(i)$, where π is a permutation. Each edge can only send 1 message per time step (may have to wait in queue before goign down edge).
Goal: Complete all the message sending within time $O(\log n)$.
Oblivious Routing: Each message decides its path at the beginning of time.

Theorem 5.0.1 Brebner, Valiant (Leslie) (1981)
With probability $\geqslant 1 - \frac{1}{n^{1000}}$, we can achieve $O(\log n)$ total time.

Proof. Algorithm:

- To send $i \longrightarrow \pi(i)$;
  - Pick random $r_i \in \{1, \dots, n\}$.
  - Phase 1: Send $i \longrightarrow r_i$.
  - Phase 2: Send $r_i \longrightarrow \pi(i)$.

We wait until a preplanned time to begin phase 2. To send $i \longrightarrow r_i$, use bit-fixing where we fix bits from left to right. Let $u_i :=$ message going from $i \longrightarrow r_i$, P_i = path that u_i takes. Focus on a fixed i.
Claim: With high probability, u_i completes p_i within $O(\log n)$ time.

Lemma 5.0.1 Lemma 1
If $i \neq j$, then edges in $P_i \cap P_j$ form a continguous subpath.

Proof. Consider an edge in P_i. For example if we have

$$\begin{gathered}010111011001\\010111111001\end{gathered}$$

we can say that the 011001 agrees with i and j, while the 0101111 agrees with r_i and r_j.

The values of k where paths agree,

$$\{k \mid r_i, r_j \text{ agree first } k \text{ bits}\} \cap \{k \mid i, j \text{ agree on final } n - k + 1 \text{ bits}\}.$$

Note that the first set is a prefix of $1, \dots n$ and the second set is a suffix of $1, \dots n$. Therefore, their intersection is a continguous interval. 😊

Lenma 5.0.2 Lemma 2

The number of time steps that u_i spends in queues is at most $|S_i|$, where $S_i = \{j \neq i \mid P_i \cap P_j \neq \emptyset\}$.

Proof. Say that a message w that is currently on P_i has "delay" d if

- w is on the t-th step of P_i for some t.
- We are in the $t + d$-th time step of the algorithm.

If u_i waits in a queue and its delay goes from d to $d+1$, give a note with the number d on it to whoever used the edge that u_i wanted to use. Later on, if a message has a note "d", and w waits in a queue:

- if edge $\notin P_i$, keep the note
- otherwise, pass the note to message that's using the edge we're waiting on.

If note d is currently held by a message w and if w is still on P_i, then w's delay is exactly d. This means that no two notes will be held by the same message. This implies that the number of notes is equal to the number of messages hold notes, which is then less or equal to $|S_i|$. 😊

Now let e be an edge. Define $m_e :=$ the number of messages tha tuse e.

Lenma 5.0.3 Lemma 3

$E[m_e] \leqslant 1$.

Proof. What does it mean to use edge e? The first k bits are free variables for j while the last $n-k+1$ agree with j. So the number of options for $j = 2^{k-1}$. FOr each such j,

$$P(r_j \text{ has correct first } k \text{ bits}) = \frac{1}{2^k}.$$

So,

$$E[\# \; j \text{ that use } e] \leqslant \sum_{\text{valid } j} P(j \text{ uses } e) \leqslant 2^{k-1}\frac{1}{2^k} = \frac{1}{2}.$$

😊

Lenma 5.0.4 Lemma 4

With high probability, $|S_i| \leqslant O(\log n)$.

Proof. First, fix P_i.

$$|S_i| = \sum_{j \neq i} \mathbb{I}[P_i \cap P_j \neq \emptyset],$$

where $\mathbb{I}$ is the indicator function. Note that these are all independent random variables determined by r_j. This is exactly the type of thing we want to apply Chernoff bounds to.

Define $\mu := E[|S_i|] \leqslant \sum_{e \in P_i} E[\# \text{ messages } j \neq i \text{ that use } e] \leqslant \frac{\log n}{2}$. 😊

So let's say $\mu = \frac{\log n}{2}$. So,

$$P(|S_i| \geqslant J\mu) \leqslant \left(\frac{1}{J}\right)^{\Omega(J\mu)} \leqslant \frac{1}{2}^{\Omega(J\mu)} \leqslant \frac{1}{\text{poly} n}.$$

😊

Suppose I flip a fair coin repeatedly:
Question 1: How many flips do I need to get $\geqslant 1$ heads with high probability.
Answer: $O(\log n) \implies P(\text{all tails}) \leqslant \frac{1}{2^{\Theta(\log n)}}$.
Question 2: How man flips do I need to get $\geqslant \log n$ heads with high probability?
Answer: $O(\log n)$. If I flip $1000 \log n$ coins, $P(< \log n \text{ heads}) = P(> 999 \log n \text{ tails})$. This means I exceeded the mean greatly as $E[\# \text{ of tails}] = 500 \log n$. If $\mu = E[T]$, then

$$\begin{aligned} T &> 1.9\mu \\ &> \mu + 0.9\sqrt{\mu}(\sqrt{\mu}). \end{aligned}$$

And the probability of this is $\leqslant e^{-\Omega(\log n)}$.

Chapter 6

Metric Embeddings

Definition 6.0.1: Metric Space

A metric space X is a set equipped with a function $d : X \times X \longrightarrow \mathbb{R}$ such that

1. $d(x, y) \geqslant 0$ for all $x, y \in X$, $d(x, y) = 0$ if and only if $x = y$.
2. $d(x, y) = d(y, x)$ for all $x, y \in X$
3. $d(x, z) \leqslant d(x, y) + d(y, z)$ for all $x, y, z \in X$

Example 6.0.1

- ℓ^p-distance. For $p \geqslant 1$, $X = \mathbb{R}^k$. Then

$$d_{\ell^p}(x, y) = \left(\sum_{i=1}^{k} |x_i - y_i|^p \right)^{1/p}.$$

- Graph-distance. Take any undirected graph with positive weights, then you can talk about the graph distance between two vertices where

$$d(v, u) = \text{graph distance}.$$

Definition 6.0.2: Metric Embeddings

Given two metric spaces, $(X_1, d_1), (X_2, d_2)$, a map $\phi : X_1 \longrightarrow X_2$ is a metric embedding with distortion at most α if $\exists \beta$ scaling parameter such that $\forall x, y \in X_1$, $d_1(x, y)$ is within a factor of α of $\beta d_2(\phi(x), \phi(y))$. That is,

$$\beta d_2(\phi(x), \phi(y)) \leqslant d_1(x, y) \leqslant \alpha \beta d_2(\phi(x), \phi(y)).$$

Theorem 6.0.1 Bourgain's Theorem

Let (X, d) be an n-point metric. Then there exists a function $\phi : (X; d) \to (\mathbb{R}^{O(\log^2 n)}, d_{\ell^1})$ such that the distortion is $O(\log n)$.

Proof. We review the algorithm. Let c be a large constant.

- For $i = 1, \ldots, \log n$,
  For $j = 1, \ldots, c \log n$,
  $S_{i,j}$ = sample each element $x \in X$ with probability 2^{-i}.

😊

Lenma 6.0.1 Key Lemma

For $i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, c \log n\}$, we have with probability $\Omega(1)$ that

$$|d(x, S_{i,j}) - d(yS_{i,j})| \geqslant r_{i+1} - r_i.$$

I LEFT LECTURE HERE, FILLIN LATER

6.1 Day 2

Theorem 6.1.1

Let (X, d) be an n-point metric space. Suppose that $\forall x \neq y \in X$, $d(x, y) \in [1, n^2]$. We will embed X into a tree metric (T, d_T) with the following properties:

- depth of tree is $O(\log n)$.
- For any root-leaf path, edge weights are decreasing powers of 2.
- Points in X get mapped to leaves of the tree.

There exists a randomized embedding $\phi : X \to (T, d_T)$, where T is also a random variable, such that

- $d_T(\phi(x), \phi(y)) \geqslant d(x, y)$.
- $E[d_T(\phi(x), \phi(y))] \leqslant O(\log n) d(x, y)$.

Proof. The algorithm:

1. Pick a random permutation $\pi = \pi_1, \ldots, \pi_n$ of the elements of X.
2. Pick uniformly random $\beta \in [1, 2]$. Define $\beta_i := 2^{i-1} \cdot \beta$.
   For each $x \in X$ and $i \in \{0, \ldots, 2\log n\}$, define
   $$\text{center}_i(x) = \text{first } \pi_j \text{ in permutation to satisfy } d(x, \pi_j) \leqslant \beta_i.$$
3. Construct ϕ as follows:
   Each node represents a set of elements. Root at level $2\log n + 1$ = all of X. For any node w at level $i + 1$, if node $>$ 1 element.
   - Group w's elements z by $\text{center}_i(z) \to$ child nodes.

   Two elements $a, b \in w$ are in the same child iff $\text{center}_i(a) = \text{center}_i(b)$.
4. Finally, edge from level $i + 1$ to level i has weight 2^i.

Analysis:
We first claim that for $x, y \in X$, $d_T(\phi(x), \phi(y)) \geqslant d(x, y)$.

Proof. Observe that at level i, if x, y in the same node, then $d(x, y) \leqslant 2\beta_i \leqslant 2^{i+1}$. So let i be the highest level the path goes through, this implies that $d(x, y) \leqslant 2^{i+1}$. But also $d_T(\phi(x), \phi(y)) \geqslant 2 \cdot 2^i \geqslant d(x, y)$ as desired. 😊

Proposition 6.1.1

$$E[d_T(\phi(x), \phi(y))] \leqslant O(\log n) d(x, y).$$

Proof.
$$\begin{aligned} d_T(\phi(x), \phi(y)) &\leqslant O(\text{largest edge weight along the path}) \\ &\leqslant \sum_{i=0}^{2\log n} \mathbb{I}(\text{center}_i(x) \neq \text{center}_i(y)) \cdot 2^{i+1}. \end{aligned}$$

So we have

$$
\begin{aligned}
E[d_T(\phi(x),\phi(y))] &\leqslant \sum_{i=0}^{2\log n} \int_{r=2^{i-1}}^{2^i} P(B_i = r)\cdot P(\mathrm{center}_i(x) \neq \mathrm{center}_i(y) \mid \beta_i = r) 2^{i+1} \mathrm{d}r \\
&= \sum_{i=0}^{2\log n} \int_{r=2^{i-1}}^{2^i} \frac{1}{2^{i-1}} \cdot P(\mathrm{center}_i(x) \neq \mathrm{center}_i(y) \mid \beta_i = r) 2^{i+1} \mathrm{d}r \\
&= \sum_{i=0}^{2\log n} \int_{r=2^{i-1}}^{2^i} P(\mathrm{center}_i(x) \neq \mathrm{center}_i(y) \mid \beta_i = r) \mathrm{d}r.
\end{aligned}
$$

If we look through $\pi_1, \pi_2, \ldots, \pi_n$, the first of $\{\mathrm{center}_i(x), \mathrm{center}_i(y)\}$ to appear is w implies that w "cuts" (x, y) (and the centers are non-equal).

Let $w_1, w_2, \ldots, w_n$ be elements of x sorted by distance from w_i to $\{x, y\}$.

$$
\sum_{i=0}^{2\log n} \int_{r=2^{i-1}}^{2^i} P(\mathrm{center}_i(x) \neq \mathrm{center}_i(y) \mid \beta_i = r) \mathrm{d}r \leqslant \sum_{j=1}^{n} \sum_{i=0}^{2\log n} \int_{r=2^{i-1}}^{2_i} P(w \text{ cuts } (x, y) \mid \beta_i = r) \mathrm{d}r.
$$

Call the integrand Ξ. We consider what must occur for w_j to cut (x, y). Suppose WLOG $d(w_j, x) \leqslant d(w_j, y)$. We need

- $d(w_j, x) \leqslant \beta_i \leqslant d(w_j, y)$.
- w_j must appear before $w_1, \ldots, w_{j-1}$ in π, which has probability $\leqslant \frac{1}{j}$.

So we get

$$
\begin{aligned}
\sum_{j=1}^{n} \sum_{i=0}^{2\log n} \int_{r=2^{i-1}}^{2_i} P(w \text{ cuts } (x, y) \mid \beta_i = r) \mathrm{d}r &\leqslant \sum_{j=1}^{n} \int_{z=d(w_j,x)}^{d(w_j,y)} \frac{1}{j} \mathrm{d}z \\
&= \sum_{j=1}^{n} \frac{1}{j} |d(w_j, y) - d_w(w_j, x)| \\
&\leqslant d(x, y) \sum_{j=1}^{n} \frac{1}{j} = d(x, y) O(\log n).
\end{aligned}
$$

😄

😄

6.2 Day 3

Theorem 6.2.1 Johnson-Lindenstrauss Lemma (1994)

Let $\epsilon \in (0,1)$. Let $S \subseteq \mathbb{R}^d$ be a set of n points. There exists a map $\phi : S \longrightarrow \mathbb{R}^{O(\log n)\cdot\epsilon^{-2}}$ which satisfies:

$$\forall a, b \in S, (1 \pm \epsilon)\|a - b\|_2 = \|\phi(a) - \phi(b)\|_2.$$

Our goal is to construct a linear ϕ such that for any $a \in \mathbb{R}^d$, with high probability in n,

$$\|\phi(a)\|_2 = (1 \pm \epsilon)\|a\|_2.$$

If we can do this, then $\forall a, b \in S$, we have with high probability that

$$\begin{aligned}\|\phi(a) - \phi(b)\|_2 &= \|\phi(a - b)\|_2 \\ &= (1 \pm \epsilon)\|a - b\|_2.\end{aligned}$$

Then by applying the union bound over all $\binom{n}{2}$ pairs, can complete the proof.
We start with a warmup task. We want to construct a linear function $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ such that $E[f(a)^2] = \|a\|_2^2$. If we can find a function like this that also suffices:

$$P(f(a)^2 > K\|a\|_2^2) \leqslant e^{-\Omega(K)},$$

then we can finish the theorem off.
Review of normal random variables: Let v = variance, $\sigma = \sqrt{v}$ (standard-deviation). Then $N(0, v)$ represents a normal random variable with mean 0 and standard deviation σ.
Fact 1: Let $X \sim \mathrm{N}(0, \sigma^2)$. Then

$$P(|X| \geqslant K\sigma) < 2e^{-K^2/3}.$$

Fact 2: If $X \sim (0, v_1)$ and $Y \sim (0, v_2)$, and $X \perp Y$, then $X + Y \sim (0, v_1 + v_2)$.
Fact 3: Let $X \sim \mathrm{N}(0, v)$, $c \in \mathbb{R}$. Then

$$cX \sim \mathrm{N}(0, c^2 v).$$

For vector $\bar{a} = (a_1, \dots, a_d)$, define

$$f(a) = \sum_{i=1}^{d} X_i \cdot a_i$$

where X_i is an independent random variable such that $X_i \sim \mathrm{N}(0, 1)$. That means $f(a) \sim \mathrm{N}(0, \|a\|_2^2)$.

Corollary 6.2.1 1

$E[f(a)^2] = \|a\|_2^2$.

Corollary 6.2.2 2

$P(f(a)^2 > K\|a\|_2^2) \leqslant e^{-\Omega(K)}$.

Proof. By the previous Chernoff bound,

$$P(|f(a)| > J\|a\|_2) \leqslant e^{-J^2/3}$$
$$P(f(a)^2 > J^2\|a\|_2^2) \leqslant 2e^{-J^2/3}.$$

By setting $K = J^2$, we get the corollary. 😄

We now turn to the proof of the Johnson-Lindenstrauss lemma.

Proof. Define $\phi(\vec{a})$ as

$$\phi(\vec{a}) = \langle \phi_1(\vec{a}), \ldots, \phi_r(\vec{a}) \rangle$$

where $r = c\epsilon^{-2} \log n$ where c is a large constant. Now we define ϕ_i. We take the same argument as in the previous proof:

$$\phi_i(a) = \frac{1}{\sqrt{r}} \sum_{j=1}^{d} X_{i,j} \cdot a_j$$

where all the $X_{i,j} \sim \mathrm{N}(0,1)$ and are independent. We want to show:

$$\begin{aligned} \|\phi(\vec{a})\|_2^2 &= (1 \pm \epsilon)\|a\|_2^2 \\ \iff \sum \phi_i(\vec{a})^2 &= (1 \pm \epsilon)\|a\|_2^2. \end{aligned}$$

We now go over a Chernoff bound for geometric random variables. Let $X_1, \ldots, X_n$ be independent. Let $\mu = E[\sum X_i]$. Suppose each X_i satisfies

$$P(|X_i| \geqslant K) \leqslant e^{-\Omega(k)}.$$

Let $X = \sum X_i$. In the small-deviation regime ($K \in [1, \sqrt{\mu}]$), we have

$$P(|X - E[X]| \geqslant K\sqrt{\mu}) \leqslant e^{-\Omega(k^2)}.$$

Then for $J \geqslant 2$, we have

$$P(X \geqslant J\mu) \leqslant \left(\frac{1}{2}\right)^{\Omega(J\mu)}.$$

Note that the large-deviation regime bound we're used to does not apply because it is not true in the $n = 1$ case. Now for each ϕ_i, we have

$$r \cdot \phi_i(\vec{a}) \sim \mathrm{N}(0, \|a\|_2^2).$$

If we wanted to normalize this, we would get

$$Y_i := \left(\frac{\sqrt{r} \cdot \phi_i(\vec{a})}{\|a\|_2}\right)^2 \sim \mathrm{N}(0,1)^2.$$

Y_i has mean 1, and $|Y_i|$ is a geometric random variable. This means

$$P\left[\left|\sum_{i=1}^{r} Y_i - r\right| \geqslant K\sqrt{r}\right] \leqslant e^{-\Omega(k^2)}.$$

Let's set $K\sqrt{r} = \epsilon r$, or that $K = \epsilon\sqrt{r}$. So,

$$P\left[\sum_{i=1}^{r} Y_i \neq (1 \pm \epsilon) r\right] = e^{-\Omega(\epsilon^2 r)}.$$

But $-\Omega(\epsilon^2 r) = \frac{1}{\mathrm{poly}(n)}$. What follows is:

$$\begin{aligned} \sum \frac{r\phi_i(\vec{a})^2}{\|a\|_2^2} &= (1 \pm \epsilon) r \\ \implies \sum \phi_i(\vec{a})^2 &= (1 \pm \epsilon)\|a\|_2^2. \end{aligned}$$

This is exactly what we wanted to prove. 😃

We now prove the facts about normal random variables.

Corollary 6.2.3 from Central Limit Theorem

Suppoose you define $X^{(n)} := \sum_{i=1}^{n} X_i$ where X_i takes on 1 or -1 with probability $1/2$. Then as $n \to \infty$, $\frac{\sqrt{v}}{\sqrt{n}} X^{(n)}$ converges pointwise to a single distribution. This is the definition of $\mathrm{N}(0, v)$.

As a shorthand,

$$\lim_{n\to\infty} \frac{\sqrt{v}}{n} X^{(n)} \sim N(0, v).$$

We skip the first fact because it's an easy Chernoff bound. Fact 3 is also straight-forward from the definition of a random variable. So we will just prove that if we have $A, B \sim \mathrm{N}(0, v_1), \mathrm{N}(0, v_2)$, then $A + B \sim \mathrm{N}(0, v_1 + v_2)$. As an easy example, take $v_1 = v_2 = 1$. So then the distribution of $A + B$ would be

$$\begin{aligned}
\lim_{n\to\infty} \frac{1}{\sqrt{n}} (X^{(n)} + Y^{(n)}) &= \lim_{n\to\infty} \frac{1}{\sqrt{n}} (X_1 + \cdots + X_n + Y_1 + \cdots + Y_n) \\
&= \lim_{n\to\infty} \frac{1}{\sqrt{n}} X^{(2n)} \\
&= \lim_{n\to\infty} \frac{\sqrt{2}}{\sqrt{m}} X^{m} && \text{(where } m = 2n\text{)} \\
&= \lim_{m\to\infty} \frac{\sqrt{2}}{\sqrt{m}} \cdot X^{(m)} \sim \mathrm{N}(0, 2).
\end{aligned}$$

Chapter 7

The Train Track Problem

While very sleep deprived and on a train ride, Bill crossed a bridge and got very confused because he thought there were no tracks below him. So he thought of a problem regarding the number of wheels on the left and right quarter of the train, and how much track is actually necessary for a train to go forever.

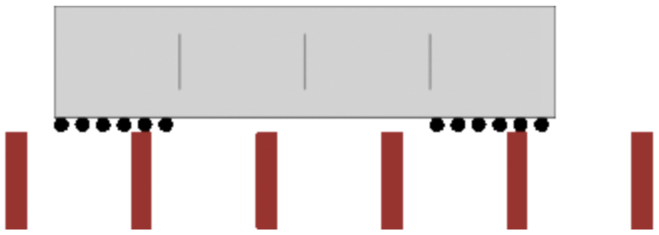
Question

Consider a set W of wheels where $W \subseteq \{0, \ldots, \ell - 1\}$ and $|W| = n$. Then consider the track $T \subseteq \{1, \ldots, 2\ell - 1\}$. A track T is valid if

$$\forall i \in \{1, \ldots, \ell\}, (i + W) \cap T \neq \emptyset.$$

Our goal is to use as little track as possible.

A clear observation we can make is that if W is an arithmetic progression (or rather, evenly spread out), we can say that there exists a valid track T of size $O\left(\frac{\ell}{n}\right)$ (just put a track at every length of the quarter train).



There are two theorems we want to prove:

Theorem 7.0.1
For any W, there exists a valid T such that $|T| \leqslant O\left(\frac{\ell}{n} \cdot \log n\right)$.

Theorem 7.0.2
$\exists W$ such that all valid T satisfy $|T| \geqslant \Omega\left(\frac{\ell}{n} \cdot \log n\right)$.

Proof of Theorem 7.0.1. Given W, WLOG, $0 \in W$.
Step 1: Let

$$T_1 = \text{ each } i \in \{1, \ldots, 2\ell - 1\} \text{ with probability } \frac{\ln n}{n}.$$

Step 2: Let

$$F = \{i \in \{1, \ldots, \ell\} \mid (i + W) \cap T_1 = \emptyset\}.$$

That is, "W falls through T, at point i".
Step 3: Define $T = T_1 \cup F$.

Note:
Because $0 \in W$, T is valid.

We move to analyzing T. First, we have

$$\begin{aligned} E[|T|] &= E[|T_1|] + E[|F|] \\ &\leqslant \frac{2\ell \ln n}{n} + \ ?. \end{aligned}$$

So,

$$\begin{aligned} E[|F|] &= \sum_{i=1}^{\ell} P[(i + W) \cap T = \emptyset] \\ &= \sum_{i=1}^{\ell} \prod_{j \in (i+W)} P(j \notin T_1) \\ &= \sum_{i=1}^{\ell} \left(1 - \frac{\ln n}{n}\right)^n \\ &\approx \sum_{i=1}^{\ell} \frac{1}{n} \\ &= \frac{\ell}{n}. \end{aligned}$$

So

$$\begin{aligned} E[|T|] &\leqslant \frac{2\ell \ln n}{n} + \frac{\ell}{n} \\ &= \ell \left(\frac{2 \ln n + 1}{n}\right) \\ &= O\left(\frac{\ell \ln n}{n}\right). \end{aligned}$$

From this, it follows that there exists a valid T such that $|T| \leqslant O\left(\frac{\ell}{n} \cdot \ln n\right)$ that is an outcome of this randomized algorithm.

☺

Proof of Theorem 7.0.2. Without loss of generality, we can set $\ell = 2n$ for the rest of this section.

Proposition 7.0.1

Let W = include each $i \in \{0, \dots, 2n-1\}$ independently with probability 1/2. Let $T \subseteq \{1, \dots, 4n\}$ satisfy

$$|T| \leqslant \frac{\ln n}{100}.$$

Then, $P(T \text{ valid}) < \dfrac{1}{n^{\Omega(\sqrt{n})}}$.

We'll defer the proof of the proposition to later. But assuming we have proved it, then

$$\begin{aligned} P\left(\exists \text{ any } T \text{ of size } \frac{\ln n}{100} \text{ that is valid}\right) &\leqslant \sum_{T \subseteq \{1,\dots,4n-1\}, |T| \leqslant \frac{\ln n}{100}} P(T \text{ valid}) \\ &\leqslant (4n)^{\frac{\ln n}{100}} \cdot \frac{1}{n^{\Omega(\sqrt{n})}} = \frac{1}{n^{\Omega(\sqrt{n})}}. \end{aligned}$$

Notice that with probabilty at least 1/2, $|W| \geqslant n$. So, there exists W such that $|W| \geqslant n$ such that there is no valid T of size $\leqslant \frac{\ln n}{100}$. It follows that there is W of exactly size n such that this is true.

So given Proposition 7.0.1, we will have proved Theorem 7.0.2.

☺

Proof of Proposition 7.0.1. Let

$$F = \{i \in \{1, \dots, 2n\} \mid (i + W) \cap T = \emptyset\}.$$

We have

$$\begin{aligned} E[|F|] &= \sum_{i=1}^{2n} P[(i+W) \cap T = \emptyset] \\ &= \sum_{i=1}^{2n} \prod_{j \in T} \underbrace{P(j \notin W + i)}_{\geqslant 1/2} \\ &\geqslant 2n \cdot \left(\frac{1}{2}\right)^T \\ &\gg n^{0.9}. \end{aligned}$$

If $f := |F|$, then f is a function $f(X_1, \dots, X_n)$ where $X_i = \mathbb{I}[i \in W]$. And if you toggle X_i, f changes by at most $|T| \leqslant \frac{\ln n}{100}$. Now we are in a perfect spot to apply McDiarmid's inequality:

$$\begin{aligned} \underbrace{P(f = 0)}_{T \text{ is valid}} &\leqslant P(f < E[f] - n^{0.9}) \\ &\leqslant n^{-\Omega(\sqrt{n})}. && \left(k = \frac{n^{0.4}}{\Theta(\lg n)}\right) \end{aligned}$$

☺

Chapter 8

Poissonization

Definition 8.0.1

A random variable X is Poisson distributed, $X \sim \text{Poisson}(\lambda)$ if

$$P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!}$$

for $\lambda, k \geqslant 0$.

Note:

Another definition is Binomial $\left(n, \frac{\lambda}{n}\right)$ as $n \to \infty$.

The above note is true because we can write

$$\begin{aligned}
\binom{n}{k}\left(\frac{\lambda}{n}\right)^k\left(1-\frac{\lambda}{n}\right)^{n-k} &\approx \left(\frac{en}{k}\right)^k\left(\frac{\lambda}{n}\right)^k\left(1-\frac{\lambda}{n}\right)^{n-k} \\
&\approx \left(\frac{e}{k}\right)^k \lambda^k e^{-\lambda(1-k/n)} \\
&\approx \left(\frac{1}{k!}\right)^k \lambda^k e^{-\lambda} \\
&= \frac{e^{-\lambda}\lambda^k}{k!}
\end{aligned}$$

as desired.
Now we review some properties of the Poisson distribution:

- If $X \sim \text{Poisson}(\lambda)$, then $E[X] = \text{Var}(X) = \lambda$.
- If $X \sim \text{Poisson}(\lambda)$ and $Y \sim \text{Poisson}(\mu)$, then $X + Y \sim \text{Poisson}(\lambda + \mu)$.
- If $X \sim \text{Poisson}(\lambda)$, then for any $c > 0$,

$$P(|X - \lambda| \geqslant c) \leqslant 2e^{\frac{-c^2}{2(c+\lambda)}}$$

In the old balls and bins problems, we would drop n balls into m bins. This was awkward because the number of balls in different bins aren't independent random variables. So consider a different variant of the problem:

Question: New balls and bins

Suppose $k \sim \text{Poisson}(n)$. We toss k balls uniformly at random into m bins.

Claim 8.0.1

The number of balls in bin i is distributed Poisson $\left(\frac{n}{m}\right)$ for all i. The number of balls in bin 1, bin 2,. . ., bin m are independent random variables.

Proof. We take $m = 2$ as this is easily generalizable. Let X_1, X_2 be the number of balls in bins 1 and 2 respectively. We claim that

$$P(X_1 = i, X_2 = j) = P(\text{Poisson}(n/2) = i) \cdot P(\text{Poisson}(n/2) = j).$$

We have

$$\begin{aligned} P(X_1 = i, X_2 = j) &= P(k = i + j) \cdot P(\text{Binomial}(i + j, 1/2) = i) \\ &= \frac{e^{-n} n^{i+j}}{(i+j)!} \cdot \binom{i+j}{i} \left(\frac{1}{2}\right)^{i+j} \\ &= e^{-n/2} \cdot e^{-n/2} \cdot n^i \cdot n^j \cdot \left(\frac{1}{i!}\right)\left(\frac{1}{j!}\right) \cdot \left(\frac{1}{2}\right)^i \cdot \left(\frac{1}{2}\right)^j \\ &= P(\text{Poisson}(n/2) = i) \cdot P(\text{Poisson}(n/2) = j) \end{aligned}$$

as desired. This proves the first claim. Now,

$$\begin{aligned} P(X_1 = i) &= \sum_j P(X_1 = i, X_2 = j) \\ &= \sum_j P(\text{Poisson}(n/2) = i) \cdot P(\text{Poisson}(n/2) = j) \\ &= P(\text{Poisson}(n/2) = i). \end{aligned}$$

Together, this gives us the second claim that $X_1 \perp X_2$. 😊

To learn Poissonization, we'll review the coupon collector. So let X be the number of steps until we get all n coupons.

Claim 8.0.2

For any constant c,

$$\lim_{n\to\infty} P(X \geqslant n \log n + cn) = 1 - e^{-e^{-c}}.$$

Proof. To prove this claim, we use Poissonization and view it as a balls and bins problems. So instead of n balls and m bins, we have $n \log n + cn$ steps and n coupons. Similarly, we'll have $k \sim \text{Poisson}(n \log n + cn)$ steps and n coupons. Here are the steps we take:

1. We want to show

$$\lim_{n\to\infty} P(\text{see every coupon after } k) = 1 - e^{-e^{-c}}.$$

2. Then we show

$$P(\text{see every type of coupon after } n \log n + cn) = P(\text{see every type of coupon after } k) \pm o(1).$$

Together, this will show the desired result. So suppose we do $k \sim \text{Poisson}(n \log n + cn)$ steps. Then the number of coupons of each type is independently distributed as Poisson$(\log n + c)$. So,

$$\begin{aligned} P(\text{we see all the coupons}) &= (1 - P(\text{Poisson}(\log n + c) = 0))^n \\ &= (1 - e^{-\log n + c})^n \\ &= (1 - e^{-c}/n)^n \\ &\approx 1 - e^{-e^{-c}} \end{aligned}$$

as desired.

So now we de-Poissonify.

(a) We want to show

$$P(\text{see all coupons after } b \text{ steps}) = P(\text{see all coupons after } b \pm n^{0.9} \text{ steps}) \pm o(1).$$

So let X be the number of boxes needed to get all the coupons. The two events we consider are $X \geqslant b - n^{0.9}$ and $X \geqslant b + n^{0.9}$. So,

$$\begin{aligned} P(X \geqslant b + n^{0.9}) &= P(X \geqslant b - n^{0.9}) + P(\text{find the last coupon in boxes } b - n^{0.9} + 1, \dots, b + n^{0.9}) \\ &\leqslant P(X \geqslant b - n^{0.9}) + \frac{2n^{0.9}}{n} \\ &= P(X \geqslant b - n^{0.9}) + o(1). \end{aligned}$$

This sort of proves the claim.

(b) So now we can have that

$$P(|k - (n \log n + cn)| \geqslant n^{0.9}) = o(1).$$

This follows directly from the tail-bound that we mentioned earlier.

This completes the overall result. ☺

Chapter 9

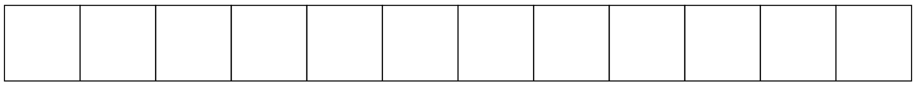
Hash Tables

9.1 Random-Probing Hash table

For this section, a hash table has two important operations:

1. `insert(key)` (adding)
2. `query(key)` (exists)

Also for this section, we'll support up to $(1-\epsilon)n$ total insertions. The hash table will be an array with n slots.



So to insert a key k, we consider a sequence of random hashes $h_1(k), h_2(k), \ldots$ and each $h_i(k)$ is uniformly random in $1, \ldots, n$ and h_i's are independent. We place the key in the first available position out of the sequence $h_1, h_2, h_3, \ldots$.
To query k, we try $h_1(k), h_2(k), \ldots$ until

- we find k, in which case we return true.
- or we find a free slot, in which case we return false.

Observations:

1. If the hash table is $1-\delta$ full, then
$$E[\text{time for next insertion}] = \delta^{-1}.$$
2. Suppose I fill $1-\epsilon$ full. Then,
$$\begin{aligned} E[\text{time to query random elements of those present}] &\approx \frac{n}{2} \cdot O(2) + \frac{n}{4} \cdot O(4) + \frac{n}{8} \cdot O(8) + \cdots + \epsilon n \cdot O(\epsilon^{-1}) \\ &\approx \underbrace{1 + 1 + \cdots + 1}_{\lg \epsilon^{-1} \text{ terms}} \\ &= O(\lg \epsilon^{-1}). \end{aligned}$$

"Open-addressed hash tables": Any hash table that is "like random-probing" but that possibly uses a different distribution for its probe sequence
$$h_1(k), h_2(k), \ldots$$
Some examples include linear probing and double hashing.

Claim 9.1.1 Ullman '72

For any open address hash table,

$$E[\text{time to query rnadom elements of those present}] \geqslant \Omega(\lg \epsilon^{-1}).$$

This was proven by Andrew Yao in 1985. The title of his paper was "Uniform Hashing is Optimal." He ended the paper with the following conjecture:

Claim 9.1.2 Yao '85

For any open address hash table, if you fill it to $(1-\epsilon)$ full, the expected time of the next insertion is at least $\Omega(\epsilon^{-1})$.

Theorem 9.1.1

Possible to achieve

$$E[\text{insertion time}] = O(\lg^2 \epsilon^{-1}).$$

We now turn attention towards the coupon collector problem:

Question: Coupon Collector

We have a hat with n coupons in it. We repeatedly draw random coupons from it, with replacement. The classical question is how many draws in expectation do we need to collect all n coupons? The answer to this is $O(n \log n)$.

Theorem 9.1.2

Let $\delta \in (0,1)$ be such taht $\delta \geqslant \frac{1}{n^{1/10}}$. Let X be the number of draws until we have collected a $(1-\delta)$ fraction of all the coupons. Then, with high probability, $X \leqslant 2n \ln \delta^{-1}$.

Proof. Do $2n \ln \delta^{-1}$ draws. Let Y be the number of distinct coupons not sampled. Then,

$$\begin{aligned} E[Y] &= \sum_{i=1}^{n} P(\text{coupon } i \text{ is not sampled}) \\ &= n \cdot \left(1 - \frac{1}{n}\right)^{2n \ln \delta^{-1}} \\ &\approx n\delta^2. \end{aligned}$$

Applying McDiarmid's to Y,

- Y is a function of $m = 2n \ln \delta^{-1}$ random variables.
- Each of these random variables can only affect Y by ± 1.

So by McDiarmid's, we have with high probability that

$$\begin{aligned} & Y \leqslant E[Y] + \tilde{O}(\sqrt{m}) \\ & \implies Y \leqslant \delta^2 n + \tilde{O}(\sqrt{m}) \\ & \implies \text{number of sampled coupons} \geqslant n - \underbrace{(\delta^2 n + \tilde{O}(\sqrt{m}))}_{\lll \delta n} \\ & \geqslant (1-\delta)n. \end{aligned}$$

😊

We now learn about a new hash table design. Break up the table as follows:

- p_1 is the first $\frac{n}{2}$ slots.
- p_2 is the next $\frac{n}{4}$ slots.
- $\vdots$
- Do this until there is a $\epsilon n/100$ size part.

Probe sequence:

- Try $100 \lg \epsilon^{-1}$ probes in p_1.
- Try $100 \lg \epsilon^{-1}$ probes in p_2.
- $\vdots$

If this fails for all parts,

- Do $100 \log n$ probes in the final part.
- Loop through the array until there is a free spot in the worst case.

The most recent step happens with very very low probability, but is required for correctness of the data structure.

Now suppose p_i gets $\geqslant 50\%$ full for some i. This means that p_{i-1} must have gotten at least $\frac{|p_i|}{2} \cdot 100 \lg \epsilon^{-1}$ probes. So, with high probability, if p_i gets 50% full, we have that

$$\frac{|p_i|}{2} \cdot 100 \lg \epsilon^{-1} = |p_{i-1}| \cdot 25 \lg \epsilon^{-1},$$

which means that p_{i-1} is $1 - \epsilon^{12.5}$ full. Then p_{i-2} is also at least $1 - \epsilon^{12.5}$ full, and all $p_{i-1}, p_{i-2}, \dots, p_1$ are all at least $1 - \epsilon^{12.5}$ full. We claim that the final part is $\leqslant \frac{1}{2}$ full. This is because if the final part were $\geqslant \frac{1}{2}$ full, then all the earlier parts would be $\geqslant 1 - \epsilon^{12.5}$ full, meaning that the overall number of elements is

$$\begin{aligned} &\geqslant (1 - \epsilon/10)(|p_1| + \cdots + |p_k|) \\ &= n \cdot \left(1 - \frac{\epsilon}{50}\right) \\ &> (1 - \epsilon)n. \end{aligned}$$

But this yields a contradiction, meaning the final part never reaches 50% full. As such, we have disproved Yao's claim. So,

- Once we get to the final part, the expected additional time is $O(1)$.
- So, the total expected time is

$$\begin{aligned} &O(1) + \text{number of probes before final part} \\ &= O(1) + 100 \lg \epsilon^{-1} \cdot O(\lg \epsilon^{-1}) = O(\lg^2 \epsilon^{-1}). \end{aligned}$$

9.2 Linear Probing

To insert an element v into a hash table that incorporates linear probing, we just

- place v in the first free slot out of $h(v), h(v)+1, \ldots$ (modulo the table size).

Searching for an element v uses the same process. At relatively low capacity, linear probing is the most efficient hash table known to man. But now suppose we perform $(1-1/x)\cdot n$ insertions and perform one more insertion.

Claim 9.2.1 Peterson 1957

The expected insertion time is $O(x)$.

Theorem 9.2.1 Knuth 1962

The expected insertion time is $\Theta(x^2)$.

Intuition for clustering:

- The longer your run gets, the more likely you are to hash into the run. "Winner keeps winning."
- "Globbing effect." This is when an insertion connects two clusters.

Intuition #2:

- Consider an interval I with size x^2. Let R be the number of items whose hashes are in I. We consider properties of R. So first,

$$E[R] = (1-1/x)\cdot x^2 = x^2 - x$$
$$\text{std}(R) \approx \sqrt{E[R]} \approx x.$$

  Moral: intervals of size $\leqslant x$ often have more elements hashing to them than can fit.

Lenma 9.2.1

Throw $(1-1/x)\cdot n$ balls into n bins. Let $k>0$. Define P_i to be the number of balls in the first i bins. Then

$$P(\exists \text{ any } i \geqslant kx^2 \text{ such that } P_i \geqslant i) \leqslant e^{-\Omega(k)}.$$

Call the event above $*$.

Warmup: What is $P(P_{kx^2} \geqslant kx^2)$? (call this event $\dagger$)

$$\mu = E[P_{kx^2}] = (1-1/x)\cdot kx^2 = kx^2 - kx.$$

So $\sqrt{\mu} \leqslant \sqrt{kx^2} = \sqrt{k}x$.So,

$$P(P_{kx^2} \geqslant kx^2) \approx P(P_{kx^2} \geqslant \mu + \sqrt{k}\sqrt{\mu}) = e^{-\Omega(k)}.$$

Now we make a critical claim. If we condition on any prefix being problematic, then

$$P(\dagger \mid *) = \Omega(1).$$

This would mean that $P(*) = \Theta(P(\dagger))$, so by the warmup, we'd be done.

Proof. Assume event $*$. Let i be the largest i such that $P_i \geqslant i$. Note that $kx^2 \leqslant i$ because we assumed $*$. If you tell me I, P_i, what is the conditional distribution for P_{kx^2}. Then,

$$P_{kx^2} \sim \text{binomial random variable given by sum of } P_i \text{ balls,}$$
$$\text{each of which has probability } kx^2/i \text{ of landing in the first } kx^2 \text{ bins.}$$

Therefore, P_{kx} is now a binomial random variable with mean $P_i \cdot \frac{kx^2}{i} \geqslant kx^2$. Any binomial random variable with mean $\mu \geqslant 1$ has $\Omega(1)$ probability of meeting or exceeding its mean. So it follows that $P(\dagger \mid *) \geqslant \Omega(1)$. This completes the proof of the lemma.

Returning to Knuth's theorem, consider insertion of an element u at $(1-1/x)$ full. Then we want to show that

$$P(\text{insertion time} \geqslant kx^2) \leqslant e^{-\Omega(k)}.$$

Now consider a table where the range $[h(v)-r, h(v)+kx^2)$ is saturated. Then, there exists a suffix of array slots ending at slot $h(v)+kx^2-1$ such that

- $|\text{suffix}| \geqslant kx^2$
- the number of elements that hash to suffix $\geqslant |\text{suffix}|$.

This is where we apply the lemma. This directly gives us that

$$P(\text{insertion time} \geqslant kx^2) \leqslant e^{-\Omega(k)}$$

which completes the proof. ☺

9.3 Ordered Linear Probing

Theorem 9.3.1 Knuth 1963
Suppose you fill a linear-probing hash table to $(1-1/x)$ full. Then, the expected insertion time is $\Theta(x^2)$.

Amble and Knuth had the following idea: modify the hash table such that in each "run", maintain an invariant that the elements in the run are in sorted order by hash.
Aside: if elements have the same hash, imagine we break ties with another hash function.

Note:
Also note that this is a history-independent data structure, meaning that the final state of the hash table is independent of the order of the elements added.

Theorem 9.3.2 Amble and Knuth 1973
$E[\text{time to query an element}] = O(x)$.

Proof. Just to simplify things, we will assume positive queries, meaning we only query elements that exist in the hash table.

Let $S = \{\text{elements in the hash table}\}$. For $s \in S, Q_s$ is the query time for s. Also define I_s as the insertion time for s when it was inserted.
Observation 1: $E[Q_s]$ is the same $\forall s \in S$.
Proof: By symmetry and because the hash table is history-independent. ☺
Observation 2:

$$\sum_{s\in S} Q_s = \sum_{s\in S} I_s.$$

Proof: Just think about it lowkey. Basically if the insertion time is k, then we increase the sum of the query times by k. ☺
Observation 3:

$$E\left[\sum_{s\in S} I_s\right] = \Theta(n\cdot x).$$

Proof: Well we have

$$\begin{aligned} E\left[\sum_{s\in S} I_s\right] &= \Theta\left(\frac{n}{2}\cdot 2^2 + \frac{n}{4}\cdot 4^2 + \frac{n}{8}\cdot 8^2 + \cdots + \frac{n}{x}\cdot x^2\right) \\ &= \Theta\left(n\cdot x\right). && \text{(dominated by last summand)} \end{aligned}$$

This completes the proof of the observation. 😃
Moving to the proof of the theorem, we have:

$$\begin{aligned} E[Q_s] &= \frac{1}{|S|} \cdot \sum_{s \in S} E[Q_s] \\ &= \frac{1}{|S|} \cdot \sum_{s \in S} E[I_s] \\ &= \frac{1}{\Theta(n)} \cdot \Theta(n \cdot x) \\ &= \Theta(x). \end{aligned}$$

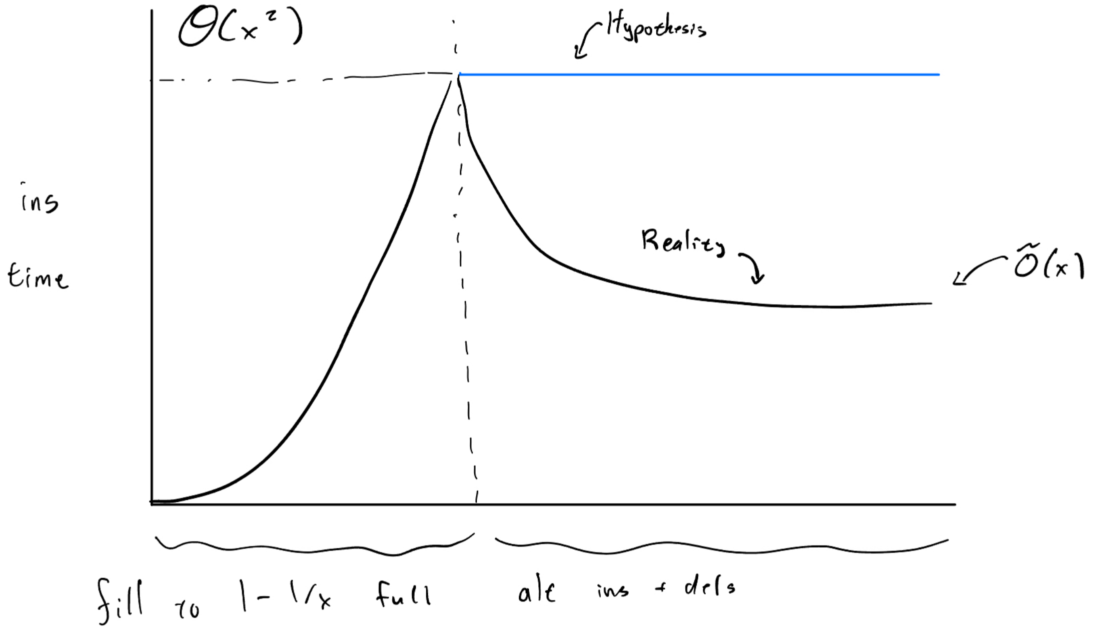
😃

9.3.1 Deletions

Idea: use "tombstones." That is, rather than entirely deleting the element, we just mark the index as "deleted." What that means is that when we query that index, all we know is that there used to be an element at that spot. Also when trying to insert an element, the tombstone spot is just as good as any other open spot to insert the element.

However, queries cannot treat tombstones are empty. This is because the element we are searching for could still be to the right of a tombstone. So queries skip over tombstones.

Every so often, we should pause and rebuild the hash table, but with the tombstones removed. How often? The classical measure is every $R = \frac{n}{2x}$ operations.
Question: What if we have a table with insertions and deletions?
Experiment: initially fill our hash table to $1 - 1/x$ full. Then, we alternate between insertions and deletions. We get this graph:



Theorem 9.3.3 Kuszmaul (2022)
Consider any workload such that the hash table is never $> 1 - 1/x$ full. Then, the amortized expected time per operation is $\widetilde{O}(x)$. The actual optimal bound is $\Theta(x \lg^{1.5} x)$.

Big Idea 1: Consider all the opeartions in a given rebuild window. Then we draw a random line in the hash table and ask how many insertions crossed the line. If we can answer this for every dotted line, we can sum to get the sum of the insertion times.
Big Idea 2: Before rebuilding, consider the state of the hash table. Draw an X in every empty slot before the random line. Then for every deletion that occurs in the rebuild window, we draw an X at the time of deletion and at the position of the deleted element. Similarly for insertions, but use ✓. Now we draw a path from the bottom left to the top right of the graph (nondecreasing). It turns out that

$$\text{the number of insertions that cross the line} = \max_{\text{all possible paths } p} (\text{number of ✓ under path} - \text{number of X under path}).$$

Big Idea 3: With enough work, we get a simpler problem which we call the path surplus problem. That is, we have an $x \times x$ grid where each entry is randomly ± 1. Consider every monotonic path to the right. For each path p, we have

$$\text{surplus}(p) = \sum \text{entries under } p.$$

The question is what is the expected max surplus over all path?

Theorem 9.3.4
The expected maximum surplus is $\Theta(x \lg^{0.75} x)$.

9.4 Quadratic Probing

To insert an element v into a hash table that incorporates quadratic probing, we just

- place v into the first free slot out of $h(v), h(v)+1, h(v)+4, \ldots$ (modulo the table size).

Claim 9.4.1 Maurer 1968
If you fill to $1 - 1/x$ fraction full, then the expected time for the next insertion is $O(x)$.

Note:
The above conjecture is an open problem, as are the following conjectures.

Claim 9.4.2
At least $O(x^2)$?

Claim 9.4.3
$f(x)$ for some f that doesn't depend on the hash table size n?

Claim 9.4.4
If the hash table is $\leqslant 1/2$ full, then expected insertion time is $O(1)$.

Claim 9.4.5 ICALP '24 (Kuszmaul, Zoe Xi)
If the hash table is $\leqslant 1/20$ full, then expected insertion time is $O(1)$.

The one above is actually solved.

Suppose the hash table is filled to 0.0001 full and consider the next insertion.

Theorem 9.4.1

$$P(\text{insertion time} \geqslant t) \leqslant e^{-\Omega(t)}.$$

Suppose the insertion time takes $\geqslant t$ time. Then construct a "witness set" $S \subset \{1, \dots, n\}$ fo teh positions that collectively did somethign bad.

Definition 9.4.1

Given a set S of positions, $q \in S$, u in the hash table, and $i \geqslant 1$, we say that "u enters S at q after its i hashes" if

- u is in some position in the S,
- $h(u)$, $h(u) + 1^2$, $\dots$, $h(u) + (i-1)^2$ (first i probes for u) are not in S,
- $h(u) + i^2 = q$.

Constructing S: suppose we're inserting an element v which takes time $\geqslant t$.

- $S \leftarrow \{h(v), h(v)+1, \dots, h(v)+(t-1)^2\}$.
- (a queue) $Q = S$.
  while Q is non empty:
  - $q \leftarrow pop(Q)$ (pick the largest $q \in Q$).
  - while there exists u and $i \geqslant 1$ such that "u enters S at q after i hashes":
    - pick such u and i arbitrarily and add $h(u), h(u)+1, \dots, h(u)+(i-1)^2$ to S and Q.

Properties of S:

1. Every position in S is occupied by an element that hashes into S.

   Proof. Consider the first time the probe sequence intersects with a value of S. What has to happen is that when we process it, $h(v)$ will get added to S. 😊

2. $|S| \geqslant t$.
3. For any given option for S such that $|S| = k$, then

   $$P(\text{that } S \text{ occuring}) \leqslant \frac{1}{18^k}.$$

   Proof. Let X be the number of elements in the hash table such that $h(v) \in S$. If S occurs, that means $X \geqslant k$. X is the sum of $0, 1$ random variables, so

   $$E[X] = 0.0001 \cdot k.$$

   By a Chernoff Bound,

   $$P(X \geqslant k) \leqslant \frac{1}{18^k}.$$

   😊

Lenma 9.4.1
Number of options for S of size k is $\leqslant 9^k$.

To finish the proof of the theorem, we'd get

$$\begin{aligned} P(\text{insertion time} \geqslant t) &\leqslant \sum_{k\geqslant t} \sum_{\text{options of } S \text{ of size } k} P(S \text{ occurs}) \\ &\leqslant \sum_{k\geqslant t} \sum_{\text{options of } S \text{ of size } k} \frac{1}{18^k} \\ &\leqslant \sum_{k\geqslant t} \frac{9^k}{18^k} \\ &\leqslant O\left(\frac{1}{2^t}\right). \end{aligned}$$

So now we go ahead and prove the lemma.

Proof. Given the process of constructing S, we record a transcript T, a trinary string.

- $S \leftarrow \{h(v), h(v)+1, \ldots, h(v)+(t-1)^2\}$.
- (a queue) $Q = S$.
- $T =$ "$000\ldots1$" such that $|T| = t$.

  while Q is non empty:

  - $q \leftarrow pop(Q)$ (pick the largest $q \in Q$).
  - while there exists u and $i \geqslant 1$ such that "u enters S at q after i hashes":
    - pick such u and i arbitrarily and add $h(u), h(u)+1, \ldots, h(u)+(i-1)^2$ to S and Q.
    - $T \leftarrow T + \underbrace{000\ldots1}_{i}$.
  - $T \leftarrow T + 2$.

If I knew T an $h(v)$, I can recover the entire construction of S, specifically the final value of S. Also, $|T| \leqslant \#0s + \#1s + \#2s \leqslant |S| + |S| = 2k$. Since T fully determines S, this means the number of options of S is bounded by the number of options for S. This is less than $(3)^{2k} = 9^k$. 😃

9.5 Cuckoo Hashing

Invented in 2001 by Pagh and Rodler. Note that for a hashtable with n slots,

$$P(\text{no collision}) = \prod_{i=1}^{n}\left(1 - \frac{i-1}{n}\right) \approx \prod_{i=1}^{n} e^{-(i-1)/n} \approx e^{-m^2/(2n)}.$$

What Cuckoo hashing does is that it creates a second hash table, also with n slots, and has two different hash functions h_1 and h_2. So if a slot is occupied in the first table, we kick that element out and put it in its proper spot in the second table. We now have two questions:

1. Does there exist an assignment of elements $x \mapsto h_1(x)$ or $h_2(x)$ without collisions.
2. Do we have efficient insertion?

Theorem 9.5.1 Erdos-Renyi

If the hash table has $\leqslant (1-\Omega(1))\cdot n$ elements, with probability $1-O(1/n)$, there exists an assignment.

Proof. If there does not exist such an assignment, by Hall's Theorem, there exists a set $S \subseteq$ elements such that $T = \bigcup_{s\in S}\{h_1(s)\cup h_2(s)\}$ such that $|T| < |S|$. Take a minimal such S. That is $|T| < |S|$ but $\forall R \subseteq S$, $|N(R)| \geqslant |R|$.

We can create a graph of the hash table where the slots are the vertices and the edges are drawn between $h_1(x)$ and $h_2(x)$ for all x. So in this case, our graph has $|S|$ edges and $< |S|$ vertices ($|S|-1$ actually) that has no degree 1 vertices. There are only so many structures this graph can take on. There are $\leqslant (|S|-1)!(|S|-1)^2$ such graphs.

Let m be the total number of elements where $m \leqslant (1-1/x)n$.

So,

$$\begin{aligned} P(\exists \text{ a bicyclic subgraph}) &\leqslant E[\text{bicyclic subgraphs}] \\ &\leqslant \sum_{s=1}^{m}\binom{2n}{s-1}(s-1)!(s-1)^2 m^s (2n)^{-2s}2^s \\ &\leqslant \sum_{s=1}^{m}(2n)^{s-1}(s-1)^2 m^s (2n)^{-2s}2^s \\ &\leqslant \sum_{s=1}^{m}(2n)^{-1}(s-1)^2\left(\frac{m}{n}\right)^s \\ &\leqslant (2n)^{-1}\sum_{s=1}^{m}(s-1)^2(1-1/x)^s \\ &\leqslant O\left(\frac{1}{n}\right). \end{aligned}$$

😆

This could be stated as $G(2n, m)$ has no bicyclic components for $m = cn$, $c < 1$.

Theorem 9.5.2 Pagh-Rodler 2001

$$E[\text{insertion time}] = O(x).$$

Proof. What we're really going to prove is that $P(\text{insertion time} \geqslant t) = e^{-\Omega(xt)}$. Assume the insertion time $\geqslant t$. Then there are t keys, $x_1, \ldots, x_t$ such that

$$\begin{aligned} h_\circ(x_1) &= h_{i_2}(x_2) \\ h_{3-i_2}(x_2) &= h_{i_3}(x_3) \\ &\vdots \end{aligned}$$

So,

$$m^{t-1}(2n)^{-(t-1)}2^t \leqslant \left(\frac{m}{n}\right)^{t-1} \leqslant \left(1-\frac{1}{x}\right)^t.$$

😆

Theorem 9.5.3
If $m > (1 + 1/x)n$, with probability $1 - o(1)$, there does not exist an assignment.

9.5.1 d-ary Cuckoo hashing

Imagine we have three hashes. Query and deletion will take $\leqslant 3$ slots. But how does this affect our load factor and insertion time?

Theorem 9.5.4 FPSS 2005
d hashes $\implies$ can't get $\geqslant 1 - \frac{1}{e^d}$ full.

Theorem 9.5.5 2009
For d hashes, there exists c_d such that load factor $> c_d$ implies there does not exist an assignment. And load factor $< c_d$ implies that there does.

We already showed that $c_2 = 0.5$. Turns out $c_3 = 0.918$, $c_4 = 0.977$. As $d \longrightarrow \infty$, $c_d = 1 - e^{-d} - o(e^{-d})$. So if we are $1 - 1/x$ full, we need $\ln(x)$ hashes. So query and deletion are both $O(\ln(x))$ in this case.

Theorem 9.5.6 FPS 2013
The expected insertion time is $O(\text{polylog} n)$ up to c_d.

Theorem 9.5.7 Walzer 2022
For 3 hashes, load factor $\leqslant 0.818$, the expected insertion is $O(1)$.

Theorem 9.5.8 Tolson, Frieze
For at least 4 hashes, the expected insertion is $O_x(1)$ up to c_d.

Lenma 9.5.1
With probability $1 - o(1)$, $\forall T \subseteq$ slots, the number of hashes to T is $\leqslant O(d \log(n/|T|))|T|$.

Lenma 9.5.2
With probability $1 - o(1)$, $\forall S \subseteq$ keys, the number of slots hashed to S is

$$\geqslant \left(d - 1 - \frac{\log(e^d(d-1))}{\log(dn/|S|)}\right)|S|$$

9.6 woo hash table theory idk

Theorem 9.6.1

Suppose we want to store up to n elements where each element is w bits. There exists a hash table such that

1. space: $nw \cdot (1 + o(1))$ bits.
2. expected time per operation $O(1)$.

Warmup 1: Chained hash table. Recall that this is where each bin j stores a doubly linked list of elements v such that $h(v) = j$.
Observation 1: Expected time per operation is $O(1)$. However, the psace is $nw + O(n \log n)$ where the latter summand comes from pointers.

Let's talk about what a hash table looks like in memory. We have a list of n pointers and an array of size n. Each internal node takes $O(\log n)$-bit pointers to a w-bit item.

When doing operations, we want to maintain an invariant that the in-use slots in the intermediate array are a prefix of the array.
Warmup 2: Balls and bins! Suppose we throw n balls into $\frac{n}{b}$ bins. Suppose $b = (\log n)^4$.

Claim 9.6.1

With high probability, every bin has $\leqslant b \cdot (1 + 1/\log n)$ balls.

Proof. By union bound, we can focus on bin 0. So let X = the number of balls in the bin. So

$$X = \sum_i^n \mathbb{I}_i$$

where $\mathbb{I}_i$ is an indicator of whether ball i went into the bin or not. So, $E[X] = b$ and we can apply a Chernoff bound. So

$$P(X > b(1 + 1/logn)) = P(X > b + \log n \cdot \sqrt{b}) \leqslant e^{-\Omega(\log^2 n)}.$$

This completes the proof. 😊

Now we move to the proof of the theorem.
Proof: Again let $b = \log^4 n$. Hash the elements into $\frac{n}{b}$ bins, each with capacity $(1 + 1/\log n) \cdot b$. In each bin, use a chained hash table. Each chained hash table uses $kw + O(k \log k)$ bits where k is the size of the hash table. So for each bin, $k = \Theta(b)$, meaning the space per pointer is order $\log b = \log(\log^4 n) = 4 \log \log n$ bits per pointer.

So for the total space, we have

$$O(n \log \log n) + (1 + 1/\log n) \cdot nw$$

bits. As $w > \log n$, we have

$$O(n \log \log n) + (1 + 1/\log n) \cdot nw \leqslant \left(1 + O\left(\frac{\log \log n}{\log n}\right)\right) \cdot nw.$$

How much space does a hash table really need? In total, we have

$$\begin{aligned} \log \binom{2^w}{n} &\approx \log \left(\frac{2^w(2^w - 1) \cdots (2^w - n + 1)}{n!}\right) \\ &\approx \log \left(\frac{2^{wn}}{n^n / e^n}\right) \\ &= wn - n \log n + O(n). \end{aligned}$$

Usually $w = \Theta(\log n)$, and we have room to save $n \log n$ bits, which in the usual case is a constant fraction of the total space we have.

Theorem 9.6.2

We can improve our space to

$$wn \cdot (1 + o(1)) - n \log n + O(n \log \log n)$$

bits.

Quotienting (Knuth).

Let's say we have $m = \frac{n}{b}$ bins. Imagine that each element was split into the first $\log m$ bits and $w - \log m$ bits. Suppose the first $\log m$ bits are random. So we're going to use the first $\log m$ bits to decide which bin it goes in.

By doing this, we don't actually need to store the first $\log m$ bits of each element because it is given by the bin number.

Now our idea is that instead of storing u, we store $\pi(u)$ where π is a random permutation from $[2^w] \to [2^w]$. This is called a permutation hash and is invertible.

The goal is to use random hash functions to build a pmerutation hash function that is "as good" as random.

Single-round Feistel Permutation:

Think of an element as $u = (x, y)$ where x is the first $\log m$ bits. Then let $h(y) \to$ the first $\log m$ bit number. Then

$$\pi(u) = (h(y) \oplus x, y).$$

Claim 9.6.2

π is invertible.

Proof. We can see that

$$\begin{aligned}\pi(\pi(u)) &= \pi(h(y) \oplus x, y) \\ &= (h(y) \oplus h(y) \oplus x, y) \\ &= (x, y) \\ &= u.\end{aligned}$$

😊

Now let $u_1, \ldots u_n$ be the elements and $u_i = (x_i, y_i)$. Let X be the number of elements in bin 0. An element u_i gets put into bin 0 if and only if

$$h(y_i) \oplus x_i = 0.$$

Observation 1: $E[X] = \frac{n}{m} = b$.

Proof. Each balls has probability $\frac{1}{m}$ of being in bin 0. 😊

Claim 9.6.3

X is still the sum of independent indicator random variables.

Proof. Let $G_y = \{u_i \mid y_i = y\}$. Let X_y be the number of elements in G_y that go to bin 0. We see that $X = \sum_y X_y$.

Also X_y is independent across all y because X_y is determined by $h(y)$ which is independent across all y.

The last thing to show is that each X_y is in $\{0, 1\}$. If $X_y \geqslant 2$, then there are distinct $u_1 = (x_1, y), u_2 = (x_2, y)$ such that

$$h(y) \oplus x_1 = h(y) \oplus x_2 = 0,$$

which would imply $x_1 = x_2 \implies u_1 = u_2$. This completes the proof. 😊

This completes the proof of the theorem.

Theorem 9.6.3 (STOC 2022)
$\log(\zeta) + O(n \underbrace{\log\log\log\cdots\log n}_{k \log s})$, has time k per operation.

Theorem 9.6.4 (FOCS 2023)
This is the optimal space/time tradeoff curve across all data structures.

Chapter 10

List Labeling Problem

Bill likes to call this the "dynamic sorting" problem.

Question: The list labeling problem

We have an array of size $m = 2n$.

- Elements are going to be inserted and deleted over time.
- $\leqslant n$ elements at once.
- Our job is to keep the elements in sorted order in the array.

The goal is to minimize the "cost" per operation:

- Really just the number of items whose array position changes.

Theorem 10.0.1 Itaei, Konheim, Rodeh 1981
There exists a solution with amortized cost $O(\log^2 n)$ per operation.

The algorithm: we're going to think of the list as a binary tree. That is, the root will be the array of size m, the children will be the array split into subarrays of size $m/2$, and the i-th level contains $m/2^i$ elements in each node. The leaves are going to be subarrays with $\leqslant 1000 \lg n$ elements.
The invariant: consider a subproblem A with children B and C. Define the density of a problem to be

$$\text{density}(A) = \frac{\text{number of elements in } A}{|A|}$$

The invariant is going to be that

$$\text{density}(B), \text{density}(A) \leqslant \left(1 + \frac{1}{10 \lg m}\right) \text{density}(A).$$

If the invariant ever breaks, we take the elements in A and spread them out evenly across A. For an insertion:

1. Fix any invariant that got broken.
2. Incur up to $O(\lg n)$ cost at leaf.

Lenma 10.0.1
Every subproblem has density $\leqslant 1$.

Proof. We have that

$$\begin{aligned} \text{density(subproblem)} &\leqslant \text{density(root)} \cdot \left(1 + \frac{1}{10 \lg m}\right)^{\lg m} \\ &\leqslant \frac{1}{2} \cdot e^{1/10} \leqslant 0.6. \end{aligned}$$

😊

Lenma 10.0.2
Amortized cost per operation is $O(\lg^2 n)$.

Proof. Every insertion/deletion pays $\lg n$ dollars to each subproblem it encounters. So this is $\lg^2 n$ total dollars paid. For a subproblem A, consider time between 2 rebuilds.

- After previous rebuild, let t = the number of elements in A.
- Between then and the next rebuild, at least $\Omega\left(\frac{t}{\lg n}\right)$ insertions/deletions must go through A. This implies that A collects $\Omega\left(\frac{t}{\lg n}\right) \cdot \lg n = \Omega(t)$ dollars. Actually, if A's new size of t', then A has $\Omega(t')$ dollars.

So rebuilding costs t' dollars and takes time t'. So we have

$$\text{total cost} \leqslant \text{total dollars spent} \leqslant O(\lg^2 n) \cdot \text{number of operations}.$$

😊

Other work on this problem:

- 1990: any "smooth" algorithm must incur $\Omega(\lg^2 n)$ per operation. In this context, "smooth" means that whenever the algorithm rebuilds a subarray, it spreads the elements evenly in the subarray.
- 2012: any deterministic algorithm must incur $\Omega(\lg^2 n)$ per operation.
- 2022: there exists a randomized algorithm with expected cost $O(\lg^{1.5} n)$ per operation.
- 2024: there exiss a randomized algorithm with expected amortized cost $O(\lg n(\lg\lg n)^3)$.

Theorem 10.0.2 2015
There is a history independent solution that gets $O(\lg^2 n)$ per operation.

Strong history independence: This says that the state of the data structure at any given moment is fully determined by

1. The current set of elements and
2. some hash functions.

Proof. The algorithm: we are given a set S of elements and a hash function $h : S \to [0, 1]$. In the middle of our array, we are going to designate a pivot, meaning the left side has size $(m-1)/2$ as well as the right side.
How to pick a pivot: consider the elements in sorted order:

$$s_1, s_2, \ldots s_k.$$

Then the candidates will be the middle $\frac{k}{10 \lg n}$ elements. We pick the pivot of as the candidate who has the min hash value.
Correctness: we observe that if we have a problem A with subproblems B and C, then

$$\text{density}(B), \text{density}(C) \leqslant \left(1 + \frac{1}{10 \lg m}\right) \text{density}(A)$$

because of the choice of our pivot. This yields correctness.
Cost bound: consider an insertion, which has to take a path down the tree from $A_1, \dots A_k$. The cost of the insertion can be bounded by

$$\text{cost} \leqslant \sum_i \sum_t P(A_i \text{ has size } t) \cdot E[\text{cost on } A_i \mid \text{size } t].$$

Lenma 10.0.3
If an insertion goes through a subproblem A and if A has t elements, then

$$P(\text{pivot changes for A}) = O\left(\frac{\lg n}{t}\right).$$

The point is that

$$E[\text{cost from } A_i \mid \text{size is } t] = O\left(\frac{\lg n}{t}\right) \cdot t = O(\lg n).$$

Proof of lemma: Let C_1 be the pivot candidtions before the insertion, and C_2 be the pivot conditions after the insertion. We observe that C_1 and C_2 by $O(1)$ elements. Let Δ be the elements where they differ. The only way the pivot can change is if the element with the smallest hash happens to be in Δ. The probability that this happens is

$$\frac{|\Delta|}{|C_1 \cup C_2|} = \frac{\Theta(1)}{\Theta\left(\frac{t}{\lg n}\right)} = O\left(\frac{\lg n}{t}\right).$$

😊

Theorem 10.0.3 Bender, Conway, Kuszmaul, etc. (2022)
There exists a history independent solution with expected cost $\widetilde{O}(\lg^{1.5} n)$ per operation.

Theorem 10.0.4
There exists a history independent solution such that if an insertion is to a random rank, then the expected cost is $\widetilde{O}(\lg^{1.5} n)$ per operation.

We construct the algorithm with the same recursive structure with a pivot in the middle. So last time for picking the pivot, if $s_1, \dots, s_t$ are the elements being stored, then the set of pivot "candidates" as the middle $\frac{t}{\Theta(\lg n)}$ elements. This time, we'll instead consider the middle

$$\frac{t}{100\sqrt{\lg n \lg \lg n}}$$

elements. As before, the pivot will be the candidate with the minimum hash value. So now,

$$P(\text{insertion into subproblem changes the pivot}) = O\left(\frac{1}{|\text{candidates}|}\right),$$

which is now roughly decreased by $\approx \sqrt{\lg n}$. This would lead to a $\widetilde{O}(\lg^{1.5} n)$ bound. This is great!

However the bad news is that this is not a correct algorithm! This is because the density could get as high as

$$\left(1+\widetilde{O}\left(\frac{1}{\sqrt{\lg n}}\right)\right)^{\lg n} \gg 1.$$

The fix is that if the subproblem's density ever reaches $\geqslant 3/4$, then swap back to the $O(\lg^2 n)$ algorithm for this subproblem and it's descendants. We will call these subproblems "corrupted."

Lenma 10.0.4 Main Lemma

Consider insertions into random rank. With probability $1-\frac{1}{\lg^{10} n}$, the insertion does not go through any corrupted subproblems.

How much does insertion cost:

- If we void corrupted subproblems, then $\widetilde{O}(\lg^{1.5} n)$ expected cost.
- If we hit a corrupted subproblem, then $\widetilde{O}(\lg^2 n)$ expected cost, but only occurs with probability $\frac{1}{\lg^{10} n}$.
- Also, might have to rebuild subproblem if the insertion makes it corrupt.

Lenma 10.0.5

Expected cost from rebuilding is $\widetilde{O}(\lg^{1.5} n)$.

We move to the proof of the main lemma. So let $d_1, d_2, \ldots$ be the densities of the subproblems that we encounter. Here's the intuition we have. Consider any subproblem with density d_i. Then it's children have densities of roughly

$$(1-1/\sqrt{\lg n})d_i \quad \text{and} \quad (1+1/\sqrt{\lg n})d_i.$$

So if we condition on an insertion reaching the subproblem with density d_i, then it will go to either of the two children with roughly 50% chance. So the d_i's are really doing a random walk where each step is $\pm\frac{1}{\widetilde{\Theta}(\sqrt{\lg n})}$ where there are $\lg n$ steps.

Note that if you have a random walk with step size s and goes for ℓ steps, then at the end we expect the random walk to be within roughly $s\sqrt{\ell}$ from its starting point. So we really have

$$\sqrt{\lg n}\cdot\frac{1}{100\sqrt{\lg n}\lg\lg n} = \frac{1}{100\sqrt{\lg\lg n}}.$$

Morally this means that the probability of hitting $3/4$ density is very very low. So now for an actual proof:

Proof. Define $\Delta_i = d_{i+1} - d_i$. Then

$$|\Delta_i| \leqslant \frac{1}{100\sqrt{\lg n}\lg\lg n}.$$

Then we have

$$E[\Delta_i \mid \Delta_1, \ldots, \Delta_{i-1}] = ?$$

So the probaiblity that an insertion goes left would be

$$\frac{1-\Delta}{2}$$

whereas the probabilty of going right would be

$$\frac{1+\Delta}{2}.$$

So the change in density, Δ_i, would be

$$\Delta_i = -\Delta d_i$$

in the left case and

$$\Delta_i = \Delta d_i$$

in the right case. So,

$$\begin{aligned} E[\Delta_i] &= \left(\frac{1-\Delta}{2}\right) \cdot (-\Delta d_i) + \left(\frac{1+\Delta}{2}\right) \cdot (\Delta d_i) \\ &= \Delta^2 d_i \\ &\leqslant \Delta^2 \\ &\leqslant \left(\frac{1}{100\sqrt{\lg n \lg\lg n}}\right)^2 \\ &= \frac{1}{10000 \lg n \lg\lg n}. \end{aligned}$$

So

$$E[\Delta_i \mid \Delta_1, \ldots, \Delta_{i-1}] \leqslant \frac{1}{10000 \lg n \lg\lg n}.$$

So, we have that

$$P(\text{get to a corrupt subproblem}) \leqslant P(\exists j \in [\lg n] \text{ such that } \Delta_1 + \cdots + \Delta_j \geqslant 1/4).$$

We'll focus on $j = \lg n$ and then union bound. So we analyze

$$P(\sum \Delta_i \geqslant 1/4).$$

Now define $\Delta_i' \coloneqq \Delta_i - \frac{1}{10000 \lg n}$. So,

$$\sum \Delta_i > \frac{1}{4} \implies \sum \Delta_i' > \frac{1}{8}.$$

So then define $\Delta_i'' \coloneqq \Delta_i' \cdot 100\sqrt{\lg n \lg\lg n}$. Then,

$$\sum \Delta_i'' > \frac{100\sqrt{\lg n \lg\lg n}}{8}.$$

This is exactly the type of random variable we can apply an adaptive Chernoff bound to as $\Delta_i'' \leqslant 1$ and $E[\Delta_i'' \mid \text{the others}] \leqslant 0$ and there are $r = \lg m$ terms. So,

$$P(\sum \Delta_i'' \geqslant k\sqrt{r}) \leqslant e^{-k^2/3}$$

In our case, $r = \lg n$ and $k \geqslant 10\sqrt{\lg n \lg\lg n}$. This gives us that

$$P(\sum \Delta_i'' \geqslant 1/4) \leqslant \frac{1}{\lg^{30} n}.$$

☺

Chapter 11

The Cup Game

We describe the game:

- Start with n cups.
- They are all empty initially.

There are two players. In each step of the game,

- The filler distributes up to $1 - \epsilon$ units of water into the cups.
- The emptier picks a cup and remove up to 1 unit of water from the cup.

The emptier's goal is the minize the backlog, which is the fill of the fullest cup. Note that $\epsilon \in [0, 1/2]$.

Theorem 11.0.1 Dietz, Sleator 1987

Consider the "greedy" emptying algorithm, meaning we just empty from the fullest cup. This algorithm achieves backlog $O(\lg n)$.

Note:

This result even holds for $\epsilon = 0$.

Theorem 11.0.2

If the emptier is deterministic, then filler can force backlog $\Omega(\lg n)$.

Note:

This result even holds for $\epsilon = \frac{1}{2}$.

Proof. The filler is gonna put $\frac{1-\epsilon}{n}$ water in all cups. Then the emptier empties from some cup. So the filler puts $\frac{1-\epsilon}{n-1}$ in each remaining cup, and we keep doing this. At the end of the construction, the backlog is $(1-\epsilon) \cdot H_n \leqslant \frac{1}{2} \ln n$. 😃

Theorem 11.0.3 Bender, Farach-Colton, Kuszmaul 2019

There exists a randomized emptying algorithm that gets backlog $O(\lg \lg n)$ with high probability if $\epsilon = \Theta(1)$.

After this theorem, they also proved it for $\epsilon = \frac{1}{\text{poly}(n)}$.

Smoothed Greedy Algorithm: At the start, put $r_i \in [0, 1]$ water into cup i for each $i \in [n]$ where each r_i is i.i.d. and uniform. Then on each step, empty from the fullest cup. BUT, if the fullest cup has less than 1 unit of water, we skip our turn.

This last caveat is extremely important because we want to preserve the randomness we put in each cup. From now on:

$$\begin{aligned} a_i(t) &= \text{amount of water filler adds to cup } i \text{ on step } t \\ b_i(t) &= \text{amount of water emptier removes from cup } i \text{ on step } t \\ c_i(t) &= \text{amount of water in cup } i \text{ at the end of step } t. \end{aligned}$$

Claim 11.0.1
We claim that $(c_i(t) \mod 1)$ is uniformly random on $[0, 1]$.

Proof.

$$c_i(t) = r_i + \sum_{j=1}^{t} a_i(t) - \sum_{j=1}^{t} b_i(t).$$

Note that r_i is random on $[0, 1]$, the second summand is a fixed number chosen by the oblivious adversary, and the third value is an integer. 😊

Now we create a potential function:

$$\phi(t) = \sum_{t=1}^{n} \lfloor c_i(t) \rfloor.$$

Lenma 11.0.1
Consider time t_1. Let $t_0 \leqslant t_1$ be most recent time when $\phi(t_0)$ was 0. With high probability, $t_1 - t_0 \leqslant O(\lg n)$.

During steps $t_0 + 1, \ldots, t_1$,

- The emptier decreases ϕ by 1 per step. So the total decrease by the emptier is $t_1 - t_0$.

But how much does the filler increase ϕ? We consider

$$P(\text{filler increases } \lfloor c_i \rfloor) = a_i(t).$$

This means that the total increase to ϕ is a sum of independent Bernoulli random variables with total mean $\sum_{i=1}^{n} a_i(t) \leqslant 1 - \epsilon$. Now we move to analyzing a sequence of k steps. Now we draw a picture:

- Draw water in cups initially.
- And, draw water added by fillter during the k steps.

Let $x_1, \ldots, x_n$ be the amount of water added into cup i during the k steps. Note that cup i crosses:

- $\lfloor x_i \rfloor$ thresholds for sure.
- One more threshold with probability $\{x_i\}$.

This is a sum of $\lfloor x_i \rfloor + 1$ independent Bernoulli random variables with total mean x_i. Note that these sums are also independent across cups. Specifically, the total increase to ϕ by the filler during the k steps is the sum of Bernoulli random variables with total mean $\sum x_i \leqslant k(1 - \epsilon)$.

Corollary 11.0.1

For any k steps,

$$\begin{aligned} &P(\text{filler increases } \phi \text{ by at least } k \text{ during steps}) \leqslant \\ &\quad \leqslant P(\text{sum of independent Bernoulli random variables with total mean } k(1-\epsilon) \text{ is at least } k) \\ &\quad \leqslant e^{-\Theta(k)}. \end{aligned}$$

If $t_1 - t_0 = k$, then the emptier decreases ϕ by k, but the filler must increase ϕ by k. But then the probability of this is $e^{-\Omega(k)}$. It follows that for all k,

$$P(t_1 - t_0 = k) \leqslant e^{-\Theta(k)}$$

So, $[t_1 - t_0]$ is a geometric random variable, so with high probability, we have that

$$[t_1 - t_0] \leqslant O(\lg n).$$

Moving on to the proof of the main theorem, consider the amount for which each cup exceeds 2. Consider the heights above 2 during time $t_0, \ldots, t_1 \leftarrow$ now. The number of cups that get to height 2 is at most $t_1 - t_0 \leqslant O(\lg n)$. This means that these cups are playing a cup game with $n' = O(\lg n)$ cups. From the Dietz/Sleator result, the backlog is $\leqslant 2 + O(\lg n') = O(\lg \lg n)$ with high probability. 😄

Chapter 12

Random Hash Functions

Definition 12.0.1

A hash function $h : U_1 \to U_2$ is pairwise independent if for all $x \neq y \in U_1$, $h(x)$ and $h(y)$ are independent random variables.

Constider a prime p. Construct $h : \mathbb{Z}_p \to \mathbb{Z}_p$. Let $a, b \in \mathbb{Z}_p$ be uniformly random. Define $h(x) = ax + b \pmod{p}$.

Theorem 12.0.1

This definition of h is pairwise independent and uniformly random.

Let $a_1, \ldots, a_k$ be random in $\mathbb{Z}_p$. Then

$$h(x) := a_1 + a_2 x + a_3 x^2 + \cdots + a_k x^{k-1} \pmod{p}$$

is k-wise independent. However, it takes $\Theta(k)$ time to evaluate.

Example 12.0.1 (Chained Hashing)

$k = 2$ works.

$$\begin{aligned} E[\text{time to query } u] &= \sum_{v \neq u \text{ in hash table}} P[h(v) = h(u)] \\ &= (n-1) \cdot \frac{1}{n} \\ &= O(1). \end{aligned}$$

Example 12.0.2 (Linear Probing)

- 5-way independence works.
- 4-way independence does not work.

Example 12.0.3 (Cuckhoo Hashing)

- 5-wise doesn't work.
- $O(\lg n)$-wise works.

Example 12.0.4 (Balls and bins)

n balls into n bins. What is the expected maximum load?

- $k = 2$ is not enough.
- $k = O(1)$ is also not enough.

It is an open problem to whether $[ax + b \pmod p] \pmod n$ achieves expected load $O(\log n / \log\log n)$.

Theorem 12.0.2

There exists a hash function $h : U \to U$, where $U = \{1, \dots, \mathrm{poly}(n)\}$, such that

- The evaluation time is $O(1)$.
- The space is $O(n^{0.9})$ machine works.
- For any set $S \subseteq U$ such that $|S| = n^{0.8}$, with high probabiilty in n, h is random on S.

Proof. Let c be a large constant. Let $A_1, A_2, \dots, A_c$ be arrays of size $n^{0.9}$, each filled with random numbers from U. Let $g_1, \dots, g_c$ be pairwise independent hash functions from $U \to \{1, \dots, n^{0.9}\}$. Define

$$h(x) = \bigoplus_{i=1}^{c} A_i[g_i(x)].$$

Call an element $x \in S$ "happy" if there exists A_i such that $g_i(x) \neq g_i(y)$ for all $y \in S \setminus \{x\}$. We observe that if all $x \in S$ are happy, then h is fully random on S.

So now we observe that with high probability, all $x \in S$ is happy. So fix an element $x \in S$ and see that

$$\begin{aligned} P(x \text{ is not happy}) &\leqslant \prod_{i=1}^{c} P(g_i(x) \in \{g_i(y) \mid y \in S \setminus \{x\}\}) \\ &\leqslant \prod_{i=1}^{c} \sum_{y \in S\setminus\{x\}} P(g_i(x) = g_i(y)) \\ &= \prod_{i=1}^{c} \sum_{y \in S\setminus\{x\}} \frac{1}{n^{0.9}} \\ &\leqslant \prod_{i=1}^{c} \frac{n^{0.8}}{n^{0.9}} \\ &= \left(\frac{1}{n^{0.1}}\right)^{c} \\ &= \frac{1}{n^{c/10}}. \end{aligned}$$

This is as desired. ☺

Example 12.0.5 (Balls and Bins)

Dietzfelbinger and Rink's Splitting Trick:

1. Use a hash function h_1 to map balls to $n^{0.2}$ collections $C_1, C_2, \dots, C_{n^{0.2}}$ such that each C_i has size $(1 \pm o(1)) \cdot n^{0.8}$.
2. Use a hash function h_2 to map each collection to $n^{0.8}$ bins.

So in total we see that there are $n^{0.2} n^{0.8} = n$ bins. Now we use the hash function from the previous theorem to conclude that with high probability, h_2 is fully random on C_1 and on C_2 and so on. So, the max load in each of these collections is $O(\lg n / \lg\lg n)$ with high probability, meaning overall the max load is $O(\lg n / \lg\lg n)$.

Note:

For step 1, we can also use the hash function from the previous theorem. For analysis, break into sets $S_1, \dots S_{n^{0.2}}$ of size $n^{0.8}$. So with high probability, h is random on each S_i. So we can use a Chernoff bound to argue that with high probability,

$$|S_i \cap C_j| = (1 \pm o(1)) \cdot n^{0.6}.$$

So it follows that

$$|C_j| = \sum_{i=1}^{n^{0.2}} |S_i \cap C_j| = (1 \pm o(1)) \cdot n^{0.8}.$$

Theorem 12.0.3 Siegel 1989

There exists a hash function h that

- $O(1)$ time.
- $n^{0.9}$ words.
- $n^{0.8}$ independent with high probability.

Theorem 12.0.4 Ostlin + Pagh 2001 $\rightarrow$ Pagh + Pagh 2004

Can do

- $O(1)$ time.
- $O(n)$ words.
- On any set S of size n, we are random with high probability.