Distributed and Online Algorithms

Thomas Nowak

These are the lecture notes for the M1 course Distributed and Online Algorithms at ENS Paris-Saclay in the academic year 2026–2027. Good references for the material covered here include Motwani and Raghavan [1] for randomized algorithms, and the lecture notes by Ghaffari [2] and the textbook by Suomela [3] for distributed algorithms. The website for the course is at https://algo.biodis.co/.

This document will inevitably contain some errors. If you find any, I will be grateful and happy to hear from you. You can reach me by email at thomas@thomasnowak.net.

Thomas Nowak
ENS Paris-Saclay
Academic Year 2026–2027
Last Update: September 2, 2026

Contents

January 18, 2027

Lecture 1: Algorithms and Probability I

We start our discussion of randomization in algorithms by going through a number of representative examples that can be studied either by using only elementary probability-theoretic facts or by looking solely at the expected value of certain quantities. The most important technical fact about the expected value that we need is that it is linear: $\mathbb{E}(X+Y) = \mathbb{E}X + \mathbb{E}Y$. This equality holds even if X and Y are not independent.

1.1 Expected Runtime of Quicksort

We recall the quicksort algorithm to sort the list S of distinct elements from a totally ordered universe:

1. If S has at most one element, return S.
2. Choose a pivot element $s \in S$.
3. Compare each element of S to s and construct lists S_1 and S_2 with the elements that are less than, respectively greater than, s.
4. Recursively sort S_1 and S_2.
5. Return S_1, x, S_2.

The deterministic version that chooses the first element of the list as the pivot element has the worst-case running time $\Omega(n^2)$ for lists of length n. If we make the choice in Step 2 an independent uniformly random one, this yields a randomized algorithm. To upper-bound the runtime of randomized quicksort, we first note that it is asymptotically dominated by the number of comparisons. We can then bound the expected number of comparisons:

Theorem 1.1. The expected number of comparisons of randomized quicksort on a list of length n is $2n \log n + O(n)$.

Proof. Let $x_1, \dots, x_n$ be the input values and $y_1, \dots, y_n$ the same values in increasing order. Let X be the number of comparisons, and let X_{ij} be the indicator variable for whether y_i and y_j are ever compared (for $i < j$).

Set $Y_{ij} = \{y_i, \dots, y_j\}$. We have $X_{ij} = 1$ if and only if y_i or y_j is the first pivot element selected from Y_{ij}. We thus have:

$$\mathbb{E}X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}X_{ij} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{k=2}^{n} \sum_{i=1}^{n+1-k} \frac{2}{k}$$
$$= \sum_{k=2}^{n} (n+1-k)\frac{2}{k} = (2n+2)\sum_{k=1}^{n} \frac{1}{k} - 4n = 2n \log n + O(n)$$

Here, we used Lemma 1.1 in the last step. □

The following lemma is useful in many situations. It corresponds to a special case of the Euler–Maclaurin summation formula. Its basic message is that oftentimes it is possible to exchange an integral for a sum.

Lemma 1.1. For all $n \in \mathbb{N}$, we have $\log(n+1) \leq H_n \leq 1 + \log n$.

Proof. For the upper bound, we calculate:

$$\begin{aligned} H_n &= \sum_{i=1}^{n} \frac{1}{i} = 1 + \sum_{i=2}^{n} \frac{1}{i} \int_{i-1}^{i} 1 \; dx = 1 + \sum_{i=2}^{n} \int_{i-1}^{i} \frac{1}{i} \; dx \leq 1 + \sum_{i=2}^{n} \int_{i-1}^{i} \frac{1}{x} \; dx \\ &= 1 + \int_{1}^{n} \frac{1}{x} \; dx = 1 + \log n \end{aligned}$$

For the lower bound, we note:

$$\begin{aligned} H_n &= \sum_{i=1}^{n} \frac{1}{i} = \sum_{i=1}^{n} \frac{1}{i} \int_{i}^{i+1} 1 \; dx = \sum_{i=1}^{n} \int_{i}^{i+1} \frac{1}{i} \; dx \geq \sum_{i=1}^{n} \int_{i}^{i+1} \frac{1}{x} \; dx \\ &= \int_{1}^{n+1} \frac{1}{x} \; dx = \log(n+1) \end{aligned}$$

This concludes the proof. □

1.2 The Secretary Problem

In the secretary problem, we seek to hire a new secretary. We must decide in an online fashion: we interview candidates sequentially, and once we pass on a candidate, we cannot return to that decision. We suppose each of the n candidates has a score, which we can perfectly assess during the interview. We do not assume any *a priori* knowledge of the distribution of the scores. Our goal is to pick the candidate with the highest score. We assume all candidate scores $x_1, x_2, \ldots, x_n \in \mathbb{R}$ are distinct, and that the order of the candidates is uniformly random, *i.e.*, all permutations are equally likely.

At the i^{th} step, the algorithm has seen the scores $x_1, \ldots, x_i$ of the first i candidates. It turns out that the following class of algorithms is optimal: Reject the first m candidates, but keep track of their maximum score. Then, for each of the candidates $m+1, \ldots, n$, hire the first candidate whose score exceeds all previously seen scores.

For this class of algorithms, we now want to determine the best parameter m, *i.e.*, the one that yields the highest probability of hiring the best candidate. Denote by E_i the event that the i^{th} candidate has the highest score and that we hire them. For $i \leq m$, we have $\mathbb{P}(E_i) = 0$, since we do not hire any of the first m candidates. For $i = m+1$, we have $\mathbb{P}(E_{m+1}) = 1/n$, since we hire the highest-scoring candidate in position $m+1$ only if they happen to be there. More generally, for positions $i > m$, we have

$$\mathbb{P}(E_i) = \mathbb{P}(H_i \mid B_i) \cdot \mathbb{P}(B_i)$$

by Bayes' theorem, where H_i is the event that we hire the i^{th} candidate and B_i is the event that the best candidate is in position i. Due to the uniformity assumption, we have $\mathbb{P}(B_i) = 1/n$. Now, to calculate $\mathbb{P}(H_i \mid B_i)$, we note that we hire candidate i if and only if we did not hire anyone before them. This happens precisely when the highest score among the first $i-1$ candidates occurred within the first block of m. We thus have $\mathbb{P}(H_i \mid B_i) = m/(i-1)$. In summary:

$$\mathbb{P}(E_i) = \begin{cases} 0 & \text{if } i \leq m \\ \frac{m}{i-1} \cdot \frac{1}{n} & \text{otherwise} \end{cases}$$

Hence, letting E denote the event that we hire the best candidate, we obtain:

$$\mathbb{P}(E) = \frac{m}{n} \sum_{i=m+1}^{n} \frac{1}{i-1} = \frac{m}{n} \left(H_{n-1} - H_{m-1}\right)$$

As in the proof of Lemma 1.1, we can show that $H_{n-1} - H_{m-1} \geq \log n - \log m$, so

$$\mathbb{P}(E) \geq \frac{m}{n} (\log n - \log m) \quad . \tag{1}$$

Differentiating this lower bound gives

$$\frac{d}{dm} \frac{m}{n} (\log n - \log m) = \frac{1}{n} (\log n - \log m) - \frac{1}{n} = \frac{\log n - \log m - 1}{n} \quad ,$$

which is zero if and only if $\log m = \log n - 1$, *i.e.*, $m = n/e$. Plugging this into (1) gives $\mathbb{P}(E) \geq 1/e \approx 37\%$.

We did make some approximations here: For one, we optimized a lower bound on $\mathbb{P}(E)$ rather than the probability itself. Also, we cannot choose $m = n/e$ exactly since this is not an integer, so we must round, *e.g.*, to $m = \lfloor n/e \rfloor$. In both cases, we introduce small errors, but asymptotically we remain optimal: The lower bound is tight since we also have the upper bound $\mathbb{P}(E) \leq \frac{m}{n}(\log(n-1) - \log(m-1)) \sim 1/e$ as $n \to \infty$. As for rounding, note:

$$\frac{\lfloor n/e \rfloor}{n} \log \frac{n}{\lfloor n/e \rfloor} \geq \frac{(n-e)/e}{n} \log \frac{n}{n/e} = \left(1 - \frac{e}{n}\right) \frac{1}{e} \sim \frac{1}{e}$$

as $n \to \infty$.

We also left open why this class of algorithms is optimal. Note that whenever an optimal algorithm chooses to hire candidate i, that candidate must be the best seen so far. Otherwise, we forgo a chance to hire the best candidate. Moreover, since we lack information about future scores, we always have $\mathbb{P}(B_i \mid H_i) = i/n$ for optimal algorithms. Since this increases with i, once we begin considering hiring, we must continue to do so. This implies that the choice of m depends only on n.

Thus, we conclude:

Theorem 1.2. An optimal algorithm for the secretary problem has an asymptotic success probability of $1/e$.

1.3 The Online Paging Problem

In the paging problem, we are tasked with managing a cache of size k. We receive a request sequence $\rho_1, \rho_2, \ldots, \rho_n$ of pages in a much larger (virtually infinite) main memory. At each request, if the requested page is not currently in the cache, we can choose which of the pages to evict to make space for the newly requested one. The cache is initially populated with some set of pages. Popular deterministic online algorithms for this problem are LRU (last recently used), FIFO(first-in first-out), and LFU (least frequently used). In contrast to online algorithms, offline algorithms can look at the complete request sequence—in the past as well as into the future—for each decision.

We assume that each cache miss incurs a constant cost, which we normalize to being 1. Formally, for any paging algorithm A and each request sequence $\rho_1, \ldots, \rho_n$, we define $f_A(\rho_1, \ldots, \rho_n)$ to be the number of cache misses when executing A with the request sequence $\rho_1, \ldots, \rho_n$. For a randomized algorithm A, this is a random variable.

Since we can always choose a request sequence with $f_A(\rho_1, \ldots, \rho_n) = n$ by requesting n different pages, it does not make much sense to look at f_A in the absolute. Rather, we will compare to the theoretical optimum:

Definition 1.1

Let O be an optimal offline algorithm and let $c \in \mathbb{R}_+$. A (randomized) algorithm A is c-competitive if there is some $b \in \mathbb{R}_+$ such that

$$\mathbb{E} f_A(\rho_1, \ldots, \rho_n) \leq c f_O(\rho_1, \ldots, \rho_n) + b$$

for all n and all request sequences $\rho_1, \ldots, \rho_n$.

It turns out that randomization enables an exponential improvement in the competitivity ratio c. To show this, we first start with the lower bound for for deterministic algorithms:

Theorem 1.3. No deterministic online algorithm is c-competitive for $c < k$.

Proof. We construct an infinite request sequence of $k+1$ pages in the main memory, *i.e.*, $\rho_i \in \{1, \ldots, k+1\}$ for all $i \in \mathbb{N}$. We do this in an inductive fashion: Initially, we populate the cache with pages $1, \ldots, k$. Then, we request the unique page ρ_{i+1} that is not in the cache when executing A on the request sequence $\rho_1, \ldots, \rho_i$.

For analysis purposes, we introduce the notion of a round in the request sequence. A round is a maximal subsequence in which at most k different pages are requested. The first round is $\rho_1, \ldots, \rho_k$.

An optimal offline algorithm misses at most once in a round: Since at most k different pages are requested, there is one that is not requested. The optimal algorithm can thus choose to evict the non-requested page at the first cache miss. The online algorithm A, on the other hand, misses at least k times (since it misses at every request).

To conclude, we distinguish two cases. If there are infinitely many rounds, then

$$\frac{f_A(\rho_1, \ldots, \rho_{n_r})}{f_O(\rho_1, \ldots, \rho_{n_r})} \geq \frac{rk}{r} = k$$

where n_r is the last index of round number r. This is a contradiction to A being c-competitive with $c < k$. In the second case, there is only a finite number $R \in \mathbb{N}$ of rounds. Since $f_A(\rho_1, \ldots, \rho_n) = n$ by construction and $f_O(\rho_1, \ldots, \rho_n) \leq R$ for all $n \in \mathbb{N}$, we have

$$\lim_{n \to \infty} \frac{f_A(\rho_1, \ldots, \rho_n)}{f_O(\rho_1, \ldots, \rho_n)} \geq \lim_{n \to \infty} \frac{n}{R} = \infty \ ,$$

which is in contradiction to A being c-competitive for any c. □

When considering competitivity of randomized algorithms, multiple different definitions are possible. They differ in the amount of information that can be accessed to construct the request sequence. The procedure that constructs this sequence if often referred to as an *adversary*. For now, we restrict ourselves to *oblivious* adversaries, who have access to the source code of the algorithm, but not to any of the random choices made during execution. Adversaries that can do that are usually called adaptive.

In the Marker algorithm, each cache location has a marker bit associated with it. It proceeds in rounds. At the start of each round, all marker bits are set to zero. When a request results in a cache hit, the marker bit of that location is set to one. When a request results in a cache miss, a random unmarked location is evicted, the requested page is placed in that location, and the location's marker bit is set to one. The current round ends when all location's marker bits are set to one.

Theorem 1.4. There exists a randomized online algorithm that is $2(1+H_k)$-competitive against oblivious adversaries.

Proof. We use the Marker algorithm. Let $\rho_1, \ldots, \rho_n$ be a request sequence.

Let R be the number of rounds of the Marker algorithm with this request sequence. Call a page *fresh* in round r if it was newly added to the cache by the Marker algorithm in round r. Let f_r be the number of fresh cache locations in round r.

If $r \geq 2$, during rounds $r-1$ and r, there are at least $k + f_r$ different page requests. Of these, at least f_r are cache misses in any algorithm, in particular the optimal one. Setting $\Delta_O(0) = 0$ and $\Delta_O(r) = f_O(\rho_1, \ldots, \rho_{n_r}) - f_O(\rho_1, \ldots, \rho_{n_{r-1}})$ where n_r denotes the last index of round $r \geq 1$, we have $\Delta_O(r-1) + \Delta_O(r) \geq f_r$ for all $r \geq 1$. We showed this for $r \geq 2$, but it is also true for $r = 1$ since both O and A start with the same pages in the cache. Hence:

$$\begin{aligned} 2f_O(\rho_1, \ldots, \rho_n) &\geq 2\sum_{r=1}^{R} \Delta_O(r) \geq \sum_{r=0}^{R-1} \Delta_O(r) + \sum_{r=1}^{R} \Delta_O(r) \\ &= \sum_{r=1}^{R} (\Delta_O(r-1) + \Delta_O(r)) \geq \sum_{r=1}^{R} f_r \end{aligned} \tag{2}$$

We now turn to upper-bounding the expected number of cache misses of algorithm A. Let us focus on a given round r and write $\Delta_A(r) = f_A(\rho_1, \ldots, \rho_{n_r}) - f_A(\rho_1, \ldots, \rho_{n_{r-1}})$ for the number of cache misses in that round. There are exactly f_r cache misses due to requests for fresh pages. Let $i_1, \ldots, i_{k-f_r}$ be the requested non-fresh pages, in the order of the first request to them. Let X_ℓ be the indicator variable that is 1 if and only if the first access to i_ℓ is a cache miss. The expected number of cache misses is:

$$\mathbb{E}\Delta_A(r) = f_r + \sum_{\ell=1}^{k-f_r} \mathbb{E}X_\ell = f_r + \sum_{\ell=1}^{k-f_r} \mathbb{P}(X_\ell = 1) \tag{3}$$

To bound $\mathbb{P}(X_\ell = 1)$, denote by γ the number of fresh pages in the cache before the first access to i_ℓ. The page i_ℓ might have been replaced by a previously accessed page. Compared to the start of the round, exactly γ pages are different. We know that $\ell - 1$ of the pages have to be the same, namely $i_1, \ldots, i_{\ell-1}$, but we don't know the fate of i_ℓ. We thus have:

$$\mathbb{P}(X_\ell = 1) = \frac{\gamma}{k - \ell + 1} \leq \frac{f_r}{k - \ell + 1} \tag{4}$$

Combining (3) and (4), the expected number of cache misses in round r is at most

$$\mathbb{E}\Delta_A(r) = f_r \left(1 + \sum_{\ell=1}^{k-f_r} \frac{1}{k - \ell + 1}\right) \leq f_r(1 + H_k) \ . \tag{5}$$

Now, combining (2) and (5) gives

$$\mathbb{E}f_A(\rho_1, \ldots, \rho_n) \leq (1 + H_k) \sum_{r=1}^{R} f_r \leq 2(1 + H_k) f_O(\rho_1, \ldots, \rho_n)$$

and concludes the proof. □

The Marker algorithm is almost optimal. It can be proved that no randomized online algorithm is c-competitive against oblivious adversaries if $c < H_k$.

January 25, 2027

Lecture 2: Algorithms and Probability II

We now turn to more involved analysis techniques than elementary probability and expected values. In terms of systems under consideration, we focus on randomized data structures.

2.1 Skip Lists

A skip list is a collection of ordered doubly linked lists $L_1, L_2, \ldots, L_r$ such that $\text{Vals}(L_r) = \emptyset$ and $\text{Vals}(L_i) \subseteq \text{Vals}(L_{i-1})$ for all $1 < i \leq r$ where $\text{Vals}(L)$ denotes the set of values in the linked list L. In addition to the horizontal links inside of its linked list L_i, each node also has a vertical link to node in L_{i-1} that contains the same value. Each linked list L_i additionally contains a start node with value $-\infty$ and an end node with value $+\infty$. The skip list encodes the set $\text{Vals}(L_1)$ of values. The linked list L_i is called the i^{th} level of the skip list.

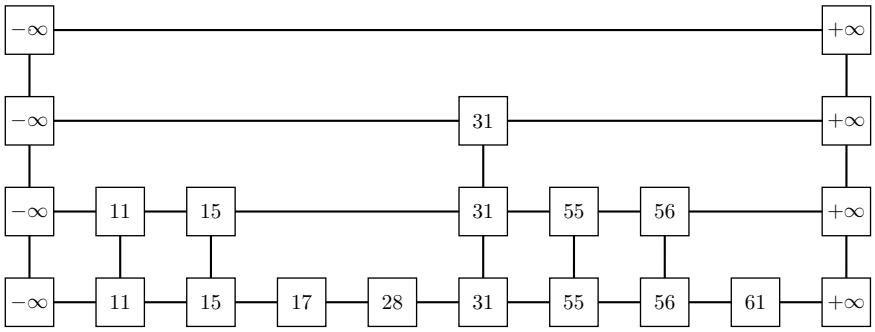

Figure 2.1: A skip list with $r = 4$ levels containing the values $\text{Vals}(L_1) = \{11, 15, 17, 28, 31, 55, 56, 61\}$.

For a value x in the skip list, we denote by $h(x) = \max\{i \mid x \in \text{Vals}(L_i)\}$ its height, *i.e.*, highest level that contains x. Since $\text{Vals}(L_r) = \emptyset$, we have $h(x) \leq r - 1$ for all values x of the skip list. On the other hand, we have the expression $r = 1 + \max\{h(x) \mid x \in \text{Vals}(L_1)\}$ for the number of levels.

An empty skip list has a single level. Starting from an initially empty skip list, this data structure supports three operations:

- `search`(x): returns true if x is a value in the skip list and false otherwise. The search starts at the $-\infty$ node of L_r. In each iteration, we look at the next node in the current level. If its value is equal to x, we return true. If its value is strictly smaller than x, then we advance to the next node in current level. If its value is strictly larger than x, then we drop down one level if possible. If it's not possible to drop down a level, then we return false.
- `insert`(x): inserts value x into the skip list if it's not already in the list. We first use `search`(x) to find the greatest lower bound and the least upper bound on x in each level. If x is already in the list, we do nothing more and return. Otherwise, we choose a random height $h(x)$ and insert x into the lists $L_1, \ldots, L_{h(x)}$ at the correct location and add the vertical links. If necessary, *i.e.*, if $h(x) \geq r$, we create new empty levels before inserting x into them.
- `delete`(x): deletes value x if it's in the list. We first use `search`(x) to find look for x in each level. If x is not in any level, we do nothing more and return. Otherwise, we delete x from each level in which we found it. We then delete all empty levels except the top level.

In the description of the operations, we left one thing unspecified: how to choose the random height $h(x)$ in the insertion operation? The most often employed method is to have $h(x)$ be a geometric random variable

with parameter $p = 1/2$. That is, the number of fair coin flips until we get one tails. During the analysis of the runtime of the operations, we will see why this is a good choice.

As a warm-up to analyzing the asymptotic performance of the operations, we show that the number of levels of a skip list of n values is $O(\log n)$ with high probability.

The notion "with high probability" is very common, but doesn't always have the same meaning. In its most basic meaning it says that the error probability decreases to zero with $1/n^\alpha$ for some $\alpha > 0$ as $n \to \infty$. Of course, this presupposes the parameter n to measure the size of the system under consideration in some meaningful way. The most useful version of "$X_n = O(f(n))$ with high probability" is "for all $\alpha > 0$ there exist $c > 0$ and $d > 0$ such that $\mathbb{P}\big(X_n > cf(n)\big) \leq d/n^\alpha$ for all (large enough) n". Definitions of "with high probability" that allow a free choice of the exponent α have the advantage of being composable much more easily.

Lemma 2.1. Let $\alpha > 1$. In a skip list of n values, we have $\mathbb{P}(r > \alpha \log_2 n) \leq 4/n^{\alpha-1}$ for the number r of levels.

Proof. The random variables $h(x)$ for $x \in \mathrm{Vals}(L_1)$ are i.i.d. geometric with parameter $p = 1/2$. This is true no matter the order of the preceding operations: the height chosen during the insertion of each of the currently present values is independent of all previous and later operations.

Let us write $\{x_1, x_2, \ldots, x_n\}$ for the set of values in the skip list. For the number of levels, we already established the formula $r = 1 + \max\{h(x_j) \mid 1 \leq j \leq n\}$. We thus have

$$\begin{aligned}\mathbb{P}(r > \alpha \log n) &= \mathbb{P}\big(1 + \max\{h(x_j) \mid 1 \leq j \leq n\} > \alpha \log_2 n\big) \\ &= \mathbb{P}\big(\exists 1 \leq j \leq n\colon\ h(x_j) > \alpha \log_2 n - 1\big) \\ &\leq n \cdot \mathbb{P}\big(h(x_1) > \alpha \log_2 n - 1\big)\end{aligned} \tag{6}$$

where we used the union bound $\mathbb{P}(E_1 \cup \cdots \cup E_n) \leq \mathbb{P}(E_1) + \cdots + \mathbb{P}(E_n)$.

Now, using the specific distribution of the height $h(x_1)$, we have $\mathbb{P}\big(h(x_1) > t\big) = (1-p)^t = 1/2^t$ for every nonnegative integer t. From this, we can deduce an bound that is true for all real numbers $t \geq 1$:

$$\mathbb{P}\big(h(x_1) > t - 1\big) = \mathbb{P}\big(h(x_1) > \lfloor t \rfloor - 1\big) = 1/2^{\lfloor t \rfloor - 1} \leq 1/2^{t-2}$$

Here, we used that $n > x$ if and only if $n > \lfloor x \rfloor$ for all integers n and real numbers x, and the fact that $\lfloor t \rfloor - 1 \geq t - 2$. We also see that the bound trivially remains true for the remaining values of t, *i.e.*, for $t < 1$ since then $1/2^{t-2} > 1/2^{-1} = 2$, which is bigger than any probability.

Plugging in $t = \alpha \log_2 n$, we get:

$$\mathbb{P}\big(h(x_1) > \alpha \log_2 n - 1\big) \leq 1/2^{\alpha \log_2 n - 2} = 4/n^\alpha$$

Combining this with (6) now concludes the proof. □

For the complete analysis of the time complexity of the three operations of a skip list, we will need the arguably most important inequality for the analysis of randomized algorithms, the Chernoff bound. It is an upper bound on the probability of a sum of Bernoulli random variables being far from its expected value.

Theorem 2.1 (Chernoff bound). Let $X_1, X_2, \ldots, X_m$ be mutually independent 0/1 random variables

and let $X = \sum_{k=1}^{m} X_k$. Then

$$\mathbb{P}\big(X \leq (1-\delta)\mu\big) \leq e^{-\mu\delta^2/2}$$

for all $0 < \delta < 1$ where $\mu = \mathbb{E}\, X$.

We can now prove that the search operation takes only logarithmic time with high probability. Since the time complexity of the insertion and deletion operations are dominated by the search, we get the same result for all three operations of the skip list.

Theorem 2.2. The time complexity of `search`(x) in a skip list of n values is $O(\log n)$ with high probability.

Proof. We bound the number of iterations of the search operation, which asymptotically dominates the time complexity. Independently of whether the searched value x was found, we consider the node visited in the last iteration and work our way backwards to the initial $-\infty$ node of level L_r.

The claimed bound of $m = O(\log n)$ on the number m if iterations comes from the symmetry of following vertical or horizontal links due to the choice of the fair parameter $p = 1/2$ for the heights, and the already established height bound from Lemma 2.1. Once we reach the highest level L_r, we are back at the node of the first iteration.

We are thus done if we find a constant d such that a sequence of at least $m = d \log n$ fair coin flips (iterations) has at least $\alpha \log_2 n = c \log n$ heads (vertical moves). By Theorem 2.1, the probability of this event is upper-bounded by

$$\mathbb{P}\left(X \leq (1-\delta)\frac{m}{2}\right) \leq e^{-m\delta^2/4}$$

where $(1-\delta)\frac{m}{2} = c \log n$. Solving for δ, we get $\delta = 1 - 2c/d$. Choosing $d = 4c$, we get $\delta = 1/2$ and thus:

$$\mathbb{P}\big(\leq c \log n \text{ vertical moves}\big) \leq \mathbb{P}\left(X \leq c \log n\right) \leq e^{-d \log n} = e^{-4c \log n}$$

Now, putting things together, we have:

$$\begin{aligned}\mathbb{P}\left(\geq d \log n \text{ iterations}\right) &\leq \mathbb{P}\left(\leq c \log n \text{ vertical moves}\right) + \mathbb{P}\left(r > c \log n\right) \\ &\leq \frac{1}{n^{4c}} + \frac{4}{n^{\alpha-1}}\end{aligned}$$

For any $\alpha > 1$, we have $c = \alpha / \log 2 > \alpha$ and thus:

$$\mathbb{P}\left(\geq d \log n \text{ iterations}\right) \leq \frac{1}{n^{4\alpha}} + \frac{4}{n^{\alpha}} \leq \frac{5}{n^{\alpha}}$$

This concludes the proof. □

2.2 Universal Hashing

Consider a universe U of keys of items that we want to store in a hash table. A hash function is a function $h\colon U \to V$ where $V = \{0, 1, \ldots, m-1\}$ and $|U| > m$. In the worst case, an n-element hash table can take $\Omega(n)$ time to search for an element; for example if all elements are hashed to the same value and end up in the same linked list. This is true no matter the hash function chosen. The idea of universal hashing is to chose a random hash function to circumvent the deterministic worst case. For that, we need to look at families of hash functions from which we will make the random choice.

Definition 2.1

A family $\mathcal{H}$ of hash functions $h\colon U \to V$ is *universal* if

$$\mathbb{P}\big(h(x) = h(y)\big) \leq \frac{1}{m}$$

for all $x, y \in U$ with $x \neq y$ where the hash function h is chosen uniformly at random in $\mathcal{H}$.

The use of universal families of hash functions allows to cut down on the length of the linked lists in the hash table. Here is a result on the expected number of collisions:

Theorem 2.3. In an n-element hash table S with the hash function h chosen uniformly at random from a universal family, for every element $x \in U$ of the universe, we have:

$$\mathbb{E}\,|\{y \in S \mid h(x) = h(y)\}| \leq \begin{cases} n/m & \text{if } x \notin S \\ 1 + (n-1)/m & \text{if } x \in S \end{cases}$$

Proof. Let $y_1, y_2, \ldots, y_n$ be the elements of S and let X_i be the indicator variable of the event $h(y_i) = h(x)$. Since $\mathcal{H}$ is universal, we have $\mathbb{E}\,X_i = \mathbb{P}(X_i = 1) \leq 1/m$ if $x \neq y_i$. Now, if $x \notin S$, then:

$$\mathbb{E}\sum_{i=1}^{n} X_i = \sum_{i=1}^{n} \mathbb{P}(X_i = 1) \leq \frac{n}{m}$$

On the other hand, if $x = y_j$ for some j, then

$$\mathbb{E}\sum_{i=1}^{n} X_i = \sum_{i=1}^{n} \mathbb{P}(X_i = 1) = 1 + \sum_{\substack{1 \leq i \leq n \\ i \neq j}} \mathbb{P}(X_i = 1) \leq 1 + \frac{n-1}{m}\ ,$$

which concludes the proof. □

Universal families of hash functions are clearly useful, but we have not yet answered the question whether any actually exists. This question is quite easily answered in the affirmative via the probabilistic method. A harder question is whether we can find a universal family that we can easily sample from and whose hash functions are simple to compute. Fortunately, the answer to this question is also yes. There are a few constructions known, here is one of them:

Let us assume that $U = \{0, 1, \ldots, u-1\}$ and choose a prime number $p \geq u$. Then, for integers $0 \leq a, b < p$, we define the hash function

$$h_{a,b}(x) = \big((ax + b) \bmod p\big) \bmod m$$

and define the family $\mathcal{H} = \{h_{a,b} \mid 0 \leq a, b < p\}$.

Theorem 2.4. The family $\mathcal{H}$ is universal.

Proof. Let $x, y \in U$ with $x \neq y$.

For every pair $(u, v) \in \{0, \ldots, p-1\}^2$ with $u \neq v$, there is a unique pair $(a, b) \in \{1, \ldots, p -$

$1\} \times \{0, \ldots, p-1\}$ with $ax + b \equiv u \mod p$ and $ay + b \equiv v \mod p$. In fact, we have the formulas $a \equiv (v-u) \cdot (y-x)^{-1} \mod p$ and $b \equiv u - ax \mod p$.

It thus suffices to upper bound the number of pairs (u, v) with $u \not\equiv v \mod p$ but $u \equiv v \mod m$. If we fix a $u \in \{0, \ldots, p-1\}$, then there are at most $\lceil p/m \rceil - 1 \leq (p-1)/m$ possible values for v. In total, we thus have at most $p(p-1)/m$ such pairs. Since each pair corresponds to a unique hash function $h_{a,b}$ in $\mathcal{H}$, we have

$$\mathbb{P}\big(h_{a,b}(x) = h_{a,b}(y)\big) \leq \frac{p(p-1)/m}{p(p-1)} = \frac{1}{m} \ ,$$

which shows that $\mathcal{H}$ is a universal family. □

We now turn to the question whether we can completely avoid collisions. In classical hash tables, it turns out that can only be guaranteed when $m \geq n^2$, which leads to a prohibitive space complexity of $\Omega(n^2)$. It *is*, however, possible to get rid of collisions if we introduce a second level of hashing for each slot. This is a technique called *perfect hashing* and is able to guarantee a worst-case time complexity of searches of $O(1)$ while keeping an $O(n)$ space complexity. We will analyze the static variant, which does not support insertions or deletions, but dynamic variants also exist.

For the analysis of perfect hashing, we use Markov's inequality, which can be quite loose, but is often enough to prove asymptotic probability bounds. It is also the basis for more advanced bounds, including Chebyshev's inequality and the Chernoff bound.

Theorem 2.5 (Markov's inequality)**.** Let X be a nonnegative random variable and let $a > 0$. Then:

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}\, X}{a}$$

Lemma 2.2. In an n-element hash table S with hash function h chosen uniformly at random from a universal family, if $m \geq n^2$, then the probability of having no collisions is at least $1/2$.

Proof. Let $y_1, y_2, \ldots, y_n$ be the elements of S and let $X_{i,j}$ be the indicator variable of the event $h(y_i) = h(h_j)$. Defining X to be the number of collisions, we have:

$$\mathbb{E}\, X = \sum_{1 \leq i < j \leq n} \mathbb{P}(X_{i,j} = 1) \leq \frac{n(n-1)}{2} \frac{1}{m} \leq \frac{n^2}{2m}$$

Now, applying Markov's inequality (Theorem 2.5) with $a = n^2/m$ concludes the proof since $n^2/m \leq 1$ by assumption. □

The construction of a 2-level perfect hash table is as follows:

1. Pick a hash function $h_1 \in \mathcal{H}_m$ uniformly at random where $m = n$. Denote by ℓ_j the number of elements hashed to value j. If $\sum_{j=1}^{m} \ell_j^2 > cn$, then pick another hash function and check again.
2. For every $j \in \{0, \ldots, m-1\}$, pick a hash function $h_{2,j}$ uniformly at random from $\mathcal{H}_{m_j}$ where $m_j = \ell_j^2$. If there is a conflict for one of the hash functions $h_{2,j}$, pick another and check for conflicts again.

If this algorithm terminates, the perfect hash table allows searches in $O(1)$ time since there are no conflicts in the second level. Moreover, its space complexity is $O(n) + \sum_{j=1}^{m} O(\ell_j^2) = O(n)$.

Theorem 2.6. A perfect hash table can be constructed in time $O(n \log^2 n)$ with high probability.

Proof. Picking and checking a hash function in $\mathcal{H}_m$ can be done in time $O(n)$. Denoting by $X_{i,j}$ the indicator function of the event $h_1(y_i) = h_i(y_j)$, we have

$$\mathbb{E} \sum_{j=0}^{m-1} \ell_j^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{E}\, X_{i,j} \leq n + 2\frac{n(n-1)}{2}\frac{1}{m} \leq \frac{c}{2}n$$

for some constant $c > 0$. Application of Markov's inequality (Theorem 2.5) now shows $\mathbb{P}\big(\sum_{j=0}^{m-1} \ell_j^2 > cn\big) \leq 1/2$. We can thus see the picks of h_1 as i.i.d. coin flips with success probability $p \geq 1/2$. The number of coin flips until the first success is $O(\log n)$ with high probability.

Picking and checking each $h_{2,j}$ can be done in time $O(\ell_j)$. An application of the Chernoff bound (Theorem 2.1) shows $\ell_j = O(\log n)$ with high probability. Picking one set of $h_{2,j}$ thus takes time $O(n \log n)$ with high probability. As above, Lemma 2.2 shows that we need to pick the $h_{2,j}$ at most $O(\log n)$ times with high probability. This means that constructing the second level is done in $O(n \log^2 n)$ time with high probability. □

References

[1] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[2] Mohsen Ghaffari. Distributed Algorithms. Lecture notes, ETH Zürich, 2023.

[3] Jukka Suomela. *Distributed Algorithms: An Intuitive Approach*. Online textbook, 2020.

[4] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. 2nd edition, Cambridge University Press, 2017.

[5] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.

[6] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[7] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[8] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 27–30, 1983.

[9] Bernadette Charron-Bost and André Schiper. The Heard-Of model: Computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.