1 Overview

In the previous lecture we finished covering data structures for the predecessor problem by studying Fusion Trees.

In this lecture, we cover several notions related to hashing: using hashing for load balancing, \( k \)-wise independent families of hashing functions and the dictionary problem for which we show two solutions: chaining with linked lists and linear probing.

2 Hashing

In hashing, we have a family \( H \) of hashing functions mapping \([u]\) to \([m]\) (where \([u]\) denotes the set \{0, \ldots, u - 1\}). For a set \( S \subseteq [u] \) with \(|S| = n\), we want that \( h \) picked at random in \( H \) behaves “nicely” on \( S \).

**Example** (load balancing). We have \( n \) jobs that need to be assigned to \( m \) machines and each job has a job ID in \([u]\). We want to evenly spread the job over the machines. We pick \( h \) at random in a family of hashing functions \( H \). When a new job \( j \in [u] \) comes in, we assign it to machine \( h(j) \in [m] \). This is usually referred to as the “balls and bins” random process.

We would like to say that \( \Pr(\exists \text{ machine receiving more than } \frac{m}{n} \text{ jobs}) \) is small. First, let us assume that the random variables \( h(i) \) for \( i \in [u] \) are independent. We will need the following concentration inequality.

**Theorem 1** (Chernoff bound). Let \( X_1, \ldots, X_n \) be \( n \) independent random variables with \( X_i \in \{0, 1\} \). Let us write \( X = \sum_{i=1}^{n} X_i \), then:

\[
\Pr\left(X \geq (1 + \delta) \mathbb{E}[X]\right) \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}}\right)^{\mathbb{E}[X]}.
\]

The Chernoff bound can be proved using Markov’s inequality.

**Lemma 2** (Markov’s inequality). Let \( X \) be a nonnegative, random variable, then:

\[
\forall \lambda > 0, \quad \Pr(X \geq \lambda) \leq \frac{\mathbb{E}[X]}{\lambda}.
\]

**Proof** (Chernoff bound). By Markov’s inequality, we have:

\[
\Pr(X \geq \lambda) = \Pr(e^{tX} \geq e^{t\lambda}) \leq \frac{\mathbb{E}[e^{tX}]}{e^{t\lambda}}.
\]
By independence:
\[
E[e^{tX}] = \prod_{i=1}^{n} E[e^{tX_i}] = \prod_{i=1}^{n} (p_i e^t + (1 - p_i)) \leq \prod_{i=1}^{n} e^{p_i(e^t-1)} = e^{(e^t-1)E[X]},
\]
where the inequality uses \(1 + x \leq e^x\). We then conclude the proof by choosing \(\lambda = (1 + \delta)E[X]\) and \(t = \log(1 + \delta)\).

Using the Chernoff bound, we can prove the following proposition.

**Proposition 3.** For the balls and bins random process, with \(m = n\), then for any \(c > 1\):
\[
P\left(\exists \text{ machine with more than } \frac{c \log n}{\log \log n} \text{ jobs}\right) \leq \frac{1}{\text{poly}(n)}.
\]

**Proof.** Focus on some machine \(t\) and define:

\[
X_i = \begin{cases} 
1 & \text{if } h(i) = t \\
0 & \text{otherwise}
\end{cases}
\]

Then, \(X = \sum_{i=1}^{n} X_i\) is the number of jobs assigned to machine \(t\) and \(E[X] = 1\). Applying the Chernoff bound:
\[
P\left(X \geq \frac{c \log n}{\log \log n}\right) \leq \left(\frac{e}{c \log n / \log \log n}\right)^{c \log n / \log \log n} = \left(\frac{e}{k}\right)^k
\]
where \(k = \frac{c \log n}{\log \log n}\). It is easy to see that for any \(\varepsilon > 0\) and for \(n\) large enough, \(\left(\frac{e}{k}\right)^k \leq \frac{1}{n^{c-\varepsilon}}\). This implies, by using a union bound:
\[
P(\exists \text{ overloaded machine}) \leq \sum_{t=1}^{n} P(t \text{ is overloaded}) \leq \frac{1}{n^{c-\varepsilon-1}}
\]
Choosing \(\varepsilon < c - 1\) is enough to conclude the proof.

We propose an alternative proof which does not rely on the Chernoff bound.

**Proof.** Using union bounds:
\[
P(\exists \text{ overloaded machine}) \leq n P(\text{machine 1 is overloaded})
\]
\[
= P(\exists k \text{ items mapping to machine 1})
\]
\[
\leq \sum_{T \subseteq [u]} P(\text{items in } T \text{ all map to 1})
\]
\[
\leq \sum_{T \subseteq [u]} \frac{1}{n^k} = \frac{1}{k!} \leq \left(\frac{1}{(k/2)^{k/2}}\right)^{k/2}
\]
and we conclude as in the previous proof.
Note that the second proof does not require full independence, but only that sets of \( k \) elements are mapped to independent random variables. This motivates the following definitions.

**Definition 4.** The random variables \( X_1, \ldots, X_n \) are \( k \)-wise independent iff for any set of indices \( 1 \leq i_1 \leq \ldots \leq i_k \leq n \), the random variables \( X_{i_1}, \ldots, X_{i_k} \) are independent.

**Definition 5.** A set of hash functions \( \mathcal{H} \) is a \( k \)-wise independent family iff the random variables

\[
\begin{align*}
&h(0), h(1), \ldots, h(u-1) \\
&\text{are } k\text{-wise independent when } h \in \mathcal{H} \text{ is drawn uniformly at random.}
\end{align*}
\]

**Example.** The set \( \mathcal{H} \) of all functions from \([u]\) to \([m]\) is \( k \)-wise independent for all \( k \). Note that this family of functions is impractical for many applications since you need \( \Theta(u \log m) \) bits to store a totally random function. For a hash table, this is already larger than the space of the data structure itself, \( O(n) \).

**Example** \((u = m = p, \ p \text{ prime})\). We define \( \mathcal{H} = \{ h : h(x) = \sum_{i=0}^{k-1} a_i x^i \mod p \} \) where \( a_i \in \{0, \ldots, p-1\} \). By using polynomial interpolation, we know that for any \((x_1, \ldots, x_k, y_1, \ldots, y_k) \in [p]^{2k}\) there exists a unique polynomial \( h \in \mathcal{H} \) such that \( h(x_1) = y_1, \ldots, h(x_k) = y_k \). Since there are \( p^k \) elements in \( \mathcal{H} \), if \( h \in \mathcal{H} \) is drawn uniformly at random, \( \mathbb{P}(h(x_1) = y_1, \ldots, h(x_k) = y_k) = 1/p^k \), which shows that \( \mathcal{H} \) is \( k \)-wise independent. It is also easy to see that this family is not \( k+1 \)-wise independent: again using polynomial interpolation, once \( h(x_1), \ldots, h(x_k) \) are fixed, \( h(x_{k+1}) \) is entirely determined.

**Example** \((u = p, \ p \text{ prime})\). We define \( \mathcal{H} = \{ h : h(x) = (\sum_{i=0}^{k-1} a_i x^i \mod p) \mod m \} \). We see that \( \mathbb{P}(h(x_1) = y_1, \ldots, h(x_k) = y_k) \leq \left(\frac{m}{p}\right)^k \). Note that in this example, \( h(x_1), \ldots, h(x_k) \) does not behave exactly like \( k \) independent uniformly distributed variables, but this bound is sufficient in most applications.

### 3 Dictionary problem

This is a data structural problem with two operations:

- **update** \((k, v)\): associates key \( k \) with value \( v \).
- **query** \((k)\): returns value \( v \) associated with key \( k \) or null if \( k \) is not in the dictionary.

The keys and the values are assumed to live in the same universe \([u]\).

#### 3.1 Chaining with linked lists

The dynamic dictionary problem can be solved by hash tables with chaining. In this solution the hash table is an array \( A \) of size \( m \). Each element in the array is a pointer to a linked list containing (key, value) pairs.

We pick \( h \in \mathcal{H} \) from a 2-wise independent family. The operations on key \( k \) are made in the linked list pointed to by \( A[h(k)] \).

\(^1\)In practice, we do not lose by requiring \( u \) to be prime. Indeed, by Bertrand’s postulate, there always exists a prime between \( u \) and \( 2u \).
The running time of an operation on key $k$ is linear in the length of the linked list at $h(k)$. It is possible to show that $E_n[\text{length of list at } h(k)] = O(1)$ if $m \geq n$.

For details of the analysis, see Lecture Notes 12 from the Spring 2014 offering of CS 124, or see [CLRS, Section 11.2].

### 3.2 Perfect hashing

In this problem there is some set $S \subset [u]$ where $|S| = n$, and we would like to find a hash function $h : [u] \to [m]$ (hashing into the $m$ tables of some array) which acts injectively on $S$. This was mentioned in class, but no details were given – for the interested reader, see the lecture notes on perfect hashing from the Spring 2014 offering of CS 124, or read [CLRS, Section 11.5], or see the original paper [FKS84].

### 3.3 Linear probing

In real architectures, sequential memory accesses are much faster. A solution to the dictionary problem with better spacial locality is linear probing. In this solution, the hash table is an array $A$, but unlike the solution with chaining and linked lists, the values are stored directly in $A$. More specifically, the value associated with key $k$ is stored at $h(k)$. If this cell is already taken, we start walking to the right, one cell at a time ², until we find an empty cell and insert the value in it.

**History.** Linear probing dates back at least to 1954 where it was used in an assembly program written by Samuel, Amdahl and Boehme. It was first analyzed in a note published by Knuth [Knu63] in the case where $h$ is totally random ($\mathcal{H}$ is the set of all functions). In this paper, he showed that when $m = (1 + \varepsilon)n$, then $\mathbb{E}[\text{time per operation}] = O\left(\frac{1}{\varepsilon^2}\right)$. There was a breakthrough in a paper by Pagh, Pagh and Ruzic [PPR09] where it was proved that for any 5-wise independent family, $\mathbb{E}[\text{time per operation}] = O(1)$. Finally Patrascu and Thorup [PT10] showed that 4-wise independent families do not guarantee a constant running time per operation (that is, they provided an example of a 4-wise independent family for which the expected running time is not constant).

Here, we will prove the following proposition.

**Proposition 6.** If $m \geq cn$ and $\mathcal{H}$ is 7-wise independent, then $\mathbb{E}[\text{query time}] = O(1)$.

To prove this proposition, we need to able to compute the expected number of cells we have to probe before finding an empty cell. We are going to show that this expected number is constant.

**Definition 7.** Given an interval $I$ of cells in $A$, we define $L(I) = |\{i : h(i) \in I\}|^3$.

**Definition 8.** We say that $I$ is full iff $L(I) \geq |I|$.

**Lemma 9.** If $\text{query}(x)$ for some $x$ makes $k$ probes, then there exist at least $k$ full intervals containing $h(x)$.

²We wrap back to the beginning of the array when we reach its right edge.
³We allow intervals to wrap around at the end of the table.
Proof. Let us define \( z_1 \) the position of the first empty cell on the left of \( h(x) \) and \( z_2 \) the position of the first empty cell on the right of \( h(x) \). It is easy to see that each of the intervals \([z_1, h(x)], [z_1, h(x) + 1], \ldots, [z_1, z_2 - 1]\) are full. There are \( z_2 - h(x) \) such intervals. But by the assumption of the lemma, \( z_2 = h(x) + k \) which concludes the proof of the lemma.

Proof of Proposition 6. By applying the previous lemma:

\[
E[\text{time of } \text{query}(x)] \leq E[\text{number of full intervals containing } h(x)] \\
\leq \sum_{k \geq 1} \sum_{I, |I| = k} \mathbb{P}(I \text{ full}) \leq \sum_{k \geq 1} k \max_{I, |I| = k} \mathbb{P}(I \text{ full})
\]

We need to show that the max in the previous sum is \( O\left(\frac{1}{k^3}\right) \) so that the sum converges. We assume that \( m = 2n \) where \( n \) is the number of different keys active.

\[
\mathbb{P}(I \text{ full}) = \mathbb{P}(L(I) \geq |I|) \leq \mathbb{P}\left(|L(I) - \mathbb{E} L(I)| \geq \mathbb{E} L(I)\right)
\]

where the inequality follows from \( E[L(I)] = \frac{|I|}{2} \). Now we would like to apply Chernoff, but we don’t have independence. Instead we can raise both sides of the inequality to some power and apply Markov’s inequality. This is called the methods of moments. We will do this in the next lecture.

References


