CS 270: Combinatorial Algorithms and Data Structures	Spring 2023
Lecture 11 — February 21, 2023	
Prof. Jelani Nelson	*Scribe: Amar Shah*

1 Chernoff Bound

To prove the Chernoff inequality, we will use the Markov inequality which we state here without proof:

Theorem 1.1 (Markov's inequality)**.** *For an non-negative random variable Z. We have that $\forall \lambda > 0$*

$$\mathbb{P}[Z > \lambda] < \frac{\mathbb{E}[Z]}{\lambda}$$

Theorem 1.2 (Chernoff's inequality)**.** *Take $X_1, ..., X_n \in \{0, 1\}$ independently, then $\mathbb{P}[X_i = 1] = p_i$, $X = \sum_{i=1}^{n} X_i$, and $\mu = \mathbb{E}[X]$. Then, $\forall \epsilon > 0$, we have that:*

$$\mathbb{P}[X > (1+\epsilon)\mu] < [\frac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}]^{\mu}$$

Proof. We first note that $\mathbb{P}[X > (1+\epsilon)\mu] = \mathbb{P}[e^{tX} > e^{t(1+\epsilon)\mu}]$. This is true for any $t > 0$. Then note that $\mathbb{P}[e^{tX} > e^{t(1+\epsilon)\mu}] < e^{-t(1+\epsilon)\mu}\,\mathbb{E}[e^{tX}]$ using Markov's inequality. We will now try and find an upperbound on the moment generating function $\mathbb{E}[e^{tX}]$:

$$\begin{aligned}
\mathbb{E}[e^{tX}] &= \mathbb{E}[e^{\sum_{i=1}^{n} X_i}] \\
&= \mathbb{E}[\prod_{i=1}^{n} e^{tX_i}] \\
&= \prod_{i=1}^{n} \mathbb{E}[e^{tX_i}] \\
&= \prod_{i=1}^{n} (1 - p_i + p_i e^t) \text{ by considering cases} \\
&= \prod_{i=1}^{n} (1 + p_i(e^t - 1)) \\
&\leq \prod_{i=1}^{n} e^{p_i(e^t-1)} \\
&= e^{\sum_{i=1}^{n} p_i(e^t-1)} \\
&= e^{\mu(e^t-1)}
\end{aligned}$$

Thus, we get that:

$$\begin{aligned}\mathbb{P}[e^{tX} > e^{t(1+\epsilon)\mu}] &< e^{-t(1+\epsilon)\mu}\,\mathbb{E}[e^{tX}] \\ &\leq e^{-t(1+\epsilon)\mu} e^{\mu(e^t-1)} \\ &= e^{\mu(e^t-1-t(1+\epsilon))}\end{aligned}$$

By taking the first and second derivative, we get that $e^{\mu(e^t-1-t(1+\epsilon))}$ is minimized when $t = ln(1+\epsilon)$. Plugging this in we get:

$$\begin{aligned}e^{\mu(e^t-1-t(1+\epsilon))} &= e^{\mu(1+-1-ln(1+\epsilon)\cdot[1+\epsilon])} \\ &= e^{\mu(\epsilon-ln(1+\epsilon)\cdot[1+\epsilon])} \\ &= \frac{e^{\mu\epsilon}}{(1+\epsilon)^{(1+\epsilon)\mu}}\end{aligned}$$

□

2 Load Balancing Review

Suppose that we have $n = m$ servers and tasks. Recall how we upperbounded the probability that one server would have more than λ tasks last time:

$$\begin{aligned}\mathbb{P}[\exists \text{ server w/ load } \geq \lambda] &= \mathbb{P}[\bigwedge_{i=1}^{m} \text{ server i has load } \geq \lambda] \\ &\leq \sum_{i=1}^{n} \mathbb{P}[\text{ server i has load } \geq \lambda] \text{ by Union Bound} \\ &= n \cdot \mathbb{P}[\text{ server 1 has load } \geq \lambda] \\ &= n \cdot \mathbb{P}[\exists \text{set } T \text{ of } \lambda \text{ jobs mapping to server 1}] \\ &\leq n \cdot \sum_{T\subseteq[n];|T|=\lambda} \mathbb{P}[\text{ all jobs } \in T \text{ map to 1}] \\ &= n \cdot \binom{n}{\lambda} \cdot (\frac{1}{n})^{\lambda} \text{ using independence}\end{aligned}$$

Then we can show that when $\lambda = O(\frac{log(n)}{log(log(n))})$, we can show that this quantity is much smaller than 1 using Stirling's approximation. The important thing to note here is that we did not need to use full independence for this proof. We just needed "λ-wise indpendence" for the last step. This realization motivates the following definitions in the next section.

3 k-wise Independence

3.1 k-wise Independent variables

Definition 3.1 (k-wise Independent Random Variables). $Y_1, Y_2, ..., Y_n$ are $k - wise$ independent if for all subsets of size k $Y_{i_1}, ..., Y_{i_k}$ and for all values $y_1, ..., y_k$, we have that $\mathbb{P}[\bigvee_{j=1}^{k} Y_{i_j} = y_j] = \prod_{j=1}^{k} \mathbb{P}[Y_{i_j} = y_j]$, i.e. any subset of size k are independent

Fact 3.2. *k-wise independence of a set of variables $Y_1, ..., Y_n$ for $k > 1$ implies $(k-1)$-wise independence. And thus it implies l-wise independence for all $1 \leq l < k$*

Proof. Say we have that $Y_1, ..., Y_n$ that is k-wise independent and we have some subset $Y_{i_1}, ..., Y_{i_{k-1}}$. We pick some Y_t that is not in this subset(we know that this can be done since $n \geq k$, otherwise k-wise independence would not make any sense). Then we have that:

$$
\begin{aligned}
\mathbb{P}[\bigwedge_{j=1}^{k-1} Y_{i_j} = y_j] &= \sum_z \mathbb{P}[Y_t = z \wedge \bigwedge_{j=1}^{k-1} Y_{i_j} = y_j] \\
&= \sum_z [\mathbb{P}[Y_t = z] \prod_{j=1}^{k-1} \mathbb{P}[Y_{i_j} = y_j]] \text{ by k-wise independence} \\
&= (\prod_{j=1}^{k-1} \mathbb{P}[Y_{i_j} = y_j]) \cdot \sum_z \mathbb{P}[Y_t = z] \\
&= (\sum_z \mathbb{P}[Y_t = z]) \cdot 1 \\
&= \sum_z \mathbb{P}[Y_t = z]
\end{aligned}
$$

□

3.2 k-wise Independent Hash Functions

Definition 3.3 (k-wise Independent Hash Family). A hash family $\mathcal{H}$ is just a set of functions mapping $[U]$ into $[m]$. A family is k-wise independent if $h(0), h(1), ..., h(U-1)$ are k-wise independent for some h drawn uniformly at random from the family

The idea behind these hash functions is that we pick some $h \in \mathcal{H}$ u.a.r, but if we think about $h(0), ..., h(U-1)$ as random variables based distributed over the possible values they take for each function $h \in \mathcal{H}$, then these are k-wise independent.

Fact 3.4. *Specifying some $h \in \mathcal{H}$ takes $log_2(|\mathcal{H}|)$ bits.*

Our goal will be to make $|\mathcal{H}|$ as small as possible.

3.3 Some Examples

Attempt 1: Set $\mathcal{H}$ as the set of all functions mapping $[U]$ into $[m]$. Clearly, this is k-wise independent. To see this we take $m = 2$ for simplicity, i.e. we will match each x to either 0 or 1. Then the probability that some $x \in [U]$ maps to 0 is $\frac{2^{U-1}}{2^U} = \frac{1}{2}$ since there are 2^U total hash functions in $\mathcal{H}$ but if want that x maps to 0, there are 2^{U-1} possible hash functions that this could be since there are $U - 1$ possible inputs that can map to 0 or 1.

Now once we have have that x maps to 1, what is then the probability that some $y \in [U]$ maps to 1. By a similar argument it must be $\frac{2^{U-2}}{2^{U-1}} = \frac{1}{2}$.

Thus, it is not hard to see in fact that this is in fact an independent hash family(not just k-wise), since setting any number of inputs to something, will not effect the probability of what the other inputs can map to.

However, since $|\mathcal{H}| = m^U$, we know that $log|\mathcal{H} \ = Ulog(m)$. We want to do better.

Attempt 2: We start in the case where $U = m = p$ which is some prime. Set $\mathcal{H}_{poly(k)} = \{h(x) : h(x) = (\sum_{i=0}^{k-1} a_i x^i)(mod\ p)\}$. Then we know that $|\mathcal{H}_{poly(k)}| = p^k = m^k$ and thus $log|\mathcal{H}_{poly(k)}| = klog(m)$ which is much better.

To show that this is k-wise independent, take $i_1, ..., i_k \in [U]$ and $y_1, ..., y_k \in [m]$. Then:

$$\begin{aligned}\mathop{\mathbb{P}}_{h \in \mathcal{H}_{(k)}}[\bigwedge_{j=0}^{k-1} h(i_j) = y_j] &= \frac{\#\text{of h's s.t. } \forall\ jh(i_j) = y_j}{|\mathcal{H}_{poly(k)}|} \\ &= \frac{1}{p^k}\end{aligned}$$

Clearly the denominator is p^k, but to see why the number of h's s.t. $\forall j\ h(i_j) = y_j$ is 1, we can note that this is essentially a k degree polynomial in our finite field and we want it to go through k points. There is only one way to do this.

Finally, we may want get around the condition that $m = U$. We still assume that $U = p$ which is some prime. Then we define $\hat{H}_{poly(k)} = \{h(x) : h(x) = (\sum_{i=0}^{k-1} a_i x^i)(mod\ p)) \pmod m\}$. This works almost as well since we get that $|\hat{H}_{poly(k)}| = m^k$ which gives us the same complexity as before.

4 Linear Probing Analysis

4.1 Dictionary Review

Recall the problem from last lecture, the dictionary problem on a universe of size u.

In hashing with chaining; we initialize m "bins" and $h(x)$ tells you which bin the item should go in. If there is a hashing collision, where two items hash to the same thing, then we instead create a linked list with both the items. To query, you have to walk along the linked list to find your queried item.

Claim 4.1. For all $x \in [u]$, the expected time to query x is $O(1 + \frac{n}{m})$.

In static dictionary, there is a known data structure to take linear space and have constant time query. However, there is no known algorithm for this regime in the dynamic problem, nor is there a lower bound disallowing it.

4.2 Linear Probing

However, this approach is not great for cache reasons, so instead we use linear probing. We still keep an array of size m, but when inserting x and finding a collision, we start at $h(x)$ and continue along in the array until we find an empty space. We do a similar walk for a query.

Definition 4.2. An interval $I \subseteq [m]$ in our array is *full* if the number of keys in the database hashing to I is $\geq |I|$

Lemma 4.3. *Suppose query(x) took k steps. Then $h(x)$ is contained in $\geq k$ full intervals of all different lengths.*

Proof. Since we know that $query(x)$ took k steps, it must be that $x, x+1, ..., x+k-1$ are all full. Say that $x-j$ is the first empty slot before x. Then we know that the interval $x-j+1, ..., x$ must be queried at least j ties since $x-j$ is empty, but $x-j+1, ..., x$ is full.

Similarly for all l such that $0 \leq l \leq k-1$, we have that $x-j+1, ..., x+l$ must have been queried $l+j$ times. This proves the claim. □

4.3 Analysis

Today, we will do the analysis assuing fully independent hashing. Next time we will to it for 7-wise and 5-wise independent hashing. Recall that last time we talked about the famous theorem by Donald Knuth:

Theorem 4.4 (Knuth [1])**.** *In a hash table with linear probing with $m = (1+\epsilon)n$, then*

$$\mathbb{E}(\textit{query time}) = O(1/\epsilon^2)$$

Today, we will show a slightly weaker version of it:

Theorem 4.5. *In a hash table with linear probing with $m = 2n$, then*

$$\mathbb{E}(\textit{query time}) = O(1)$$

Proof. Note that for some interval I, $\mathbb{E}[\text{items that hash to } I] = \frac{|I|}{2}$ since $m = 2n$. Thus, by the Chernoff bound we have that $\mathbb{P}[\text{a length k interval is full}] \leq e^{-\Omega(k)}$

The number of probes to query(x) is $\leq \sum_{k=1}^{\infty} \mathbb{1}_{\exists \text{ length k full interval containing } h(x)}$. Thus, we have that:

$$
\begin{aligned}
\mathbb{E}[\#\text{ probes to } query(x)] &\leq \sum_{i=1}^{\infty} \mathbb{P}[\exists \text{ length k full interval containing } h(x)] \\
&\leq \sum_{i=1}^{\infty} k\, \mathbb{P}[\text{a specific length k inteval containing h(x) is full}] \text{ by Union Bound} \\
&\leq \sum_{i=1}^{k} k e^{-\Omega(k)} \text{ by the Chernoff bound} \\
&= O(1)
\end{aligned}
$$

□

Note that the sum $\sum_{i=1}^{k} k e^{-\Omega(k)}$ actually converges faster than in needs to in order to get the necessary bound. This gives intuition for how we are going to show this for 7-wise and 5-wise independent hashing next time.

References

[1] Donald Knuth. Notes on "open" addressing, 1963. URL: http://jeffe.cs.illinois.edu/teaching/datastructures/2011/notes/knuth-OALP.pdf.