1 Recap: Balls and Bins

In the previous lecture, we saw the problem of Balls and Bins: given n balls and n bins, we pick a ball and put it into a bin chosen independently and uniformly at random. We have also proven in class that the maximum loaded bin has $\Omega(\frac{\log n}{\log \log n})$ balls in it. This bound is tight with high probability, so the maximum load is unlikely to be larger. However, using the power of two choices, we can reduce the upper bound on maximum load to $O(\log \log n)$.

A natural question to ask would be whether we can further reduce the bound if we increased the number of choices from 2 to any natural number $d > 2$. The answer turns out to be negative, the bound on maximum load being $\Theta(\frac{\log \log n}{\log d})$. Having d choices would also make the algorithm more complex, hence it is better to go with 2 choices.

2 Concentration Bounds

This section is a refresher for probabilistic concentration bounds. We demonstrate their power for coming up with tighter bounds in algorithmic analysis. We limit our discussion to the comparison of Markov's and Bernstein's Inequalities. Although students are encouraged to read further about Chebeyshev's, Jensen's, Han's, Hoeffding's among other inequalities.

Before we begin, let us state the two inequalities:

Markov's Inequality If X is a non-negative random variable, then for an arbitrary $t > 0$ we have

$$\Pr(X \geq t) \leq \frac{\mathrm{E}(X)}{t}$$

Bernstein's Inequality Let $x_1, x_2, \ldots, x_n$ be random variables such that $0 \leq x_i \leq 1$ or they are scaled appropriately and let $\sigma^2 = \sum_{i=1}^{n} Var\{x_i\}$, then for an arbitrary $t > 0$

$$\Pr\left(\sum_{i=1}^{n} x_i - \mathrm{E}(X) \geq t\right) \leq \exp\left(-\frac{t^2}{2(\sigma^2 + \frac{t}{3})}\right)$$

Now, to demonstrate use of these inequalities, Let us spin a biased coin n times. Let p be the probability for it to be head. We take the following three cases to understand the types of bounds we can get from the two inequalities

Case:1 $p_i = \frac{1}{n}$
$\sum p_i = 1$
$\mathrm{E}[\#heads] = 1$
$\sigma^2 = \sum np_i(1 - p_i) < np_i = 1$
By Markov's inequality we can say $\Pr(\#heads \geq 2) \leq \frac{1}{2}$ and
By Bernstein's inequality (using $t = 3\log n$), $\Pr(\#heads - 1 > 6\log n) \leq \exp\left(-\frac{18\log^2 n}{1+2\log n}\right) \leq n^{-9}$.

Case:2 $p_i = \frac{6 \log n}{n}$
$\sum p_i = 6 \log n$
$\mathrm{E}[\#heads] = 6 \log n$
$\sigma^2 = \sum np_i(1-p_i) < np_i = 6 \log n$
By Markov's inequality we can say $\Pr(\#heads \geq 12 \log n) \leq \frac{1}{2}$ and
By Bernstein's inequality (using $t = 6 \log n$), $\Pr(\#heads - 6 \log n \geq 3 \log n) \leq \exp\left(\frac{-\frac{9}{2}\log^2 n}{(6 \log n + \log n)}\right) \leq n^{-\frac{9}{14}}$

Case:3 $p_i = \frac{1}{2}$
$\sum p_i = \frac{n}{2}$
$\mathrm{E}[\#heads] = \frac{n}{2}$
$\sigma^2 = \sum np_i(1-p_i) < np_i = \frac{n}{2}$
By Markov's inequality we can say $\Pr(\#heads \geq n) \leq \frac{1}{2}$ and
By Bernstein's inequality (using $t = \sqrt{6n \log n}$), $\Pr(\#heads - \frac{n}{2} \geq \sqrt{6n \log n}) \leq \exp\left(\frac{-6n \log n}{(n + \frac{2}{3}\sqrt{6n \log n})}\right) \leq n^{-6}$

In all the above 3 cases Markov's inequality gives weak bound on the estimation of concentration. Where as from Bernstein's inequality it is evident that distribution is densely concentrated near the mean. Whereas Markov's inequality doesn't take care of the nature of distribution, Bernstein's inequality considers it (by means of the variance). This results in tighter bounds.

3 Cuckoo Hashing

Salient Features of Cuckoo hashing [1] involves :

- Its an adaptive hashing Algorithm.
- Insertion: $O(1)$ expected and $O(\log n)$ worst case w.h.p.
- Lookup: $O(1)$
- Deletion: $O(1)$
- Space: $O(n)$

In Cuckoo's Hashing two universal hash functions $h_1, h_2 : \mathcal{U} \to [m]$ are used, corresponding to each function there is a hash table. Also $|\mathcal{U}| \gg m, n$ and $m = (1+\epsilon)n$, for an arbitrary $\epsilon > 0$. It is assumed that the elements to be inserted are independent and identically distributed. Since the two functions are universal $\Pr[h_1[X] = a \wedge h_2[X] = a] \leq \frac{1}{m^2}$

Lookup: For an element x, this involves finding the corresponding positions in both the tables using h_1, h_2. If the said element is found it is returned.

Deletion: Similar to lookup.

Insertion: To insert an element x into the hash table we look for the its position in table 1 using h_1(lets fix this for every start), if $h_1(x)$ is empty we insert x into it, otherwise we evict the element in $h_1(x)$ and put x in it. The evicted element has to go find its position in table 2 using h_2 hash function. This continues until no element has to be evicted. We will analyse insertion in the forthcoming sections.

Following is a pseudo code for inserting an element in the cuckoo's hash table. Note that $MaxLoop \leftarrow totalElem + 1$. Where $totalElem$ are the number of elements in the connected component.

Algorithm 1. CuckooHash(x)

```
1: loopCount ← 0
2: while loopCount ≠ maxLoop and x ≠ null do
3:     loopCount ← loopCount + 1
4:     if T1[h1(x)] is empty then
5:         T1[h1(x)] ← x
6:         return
7:     else
8:         y ← T1[h1(x)]
9:         T1[h1(x)] ← x
10:        x ← T2[h2(y)]
11:        T2[h2(y)] ← y
12:    end if
13: end while
14: if loopCount = maxCount then
15:    ## Hashing Failed
16:    ReHash(x)
17: end if
```

end

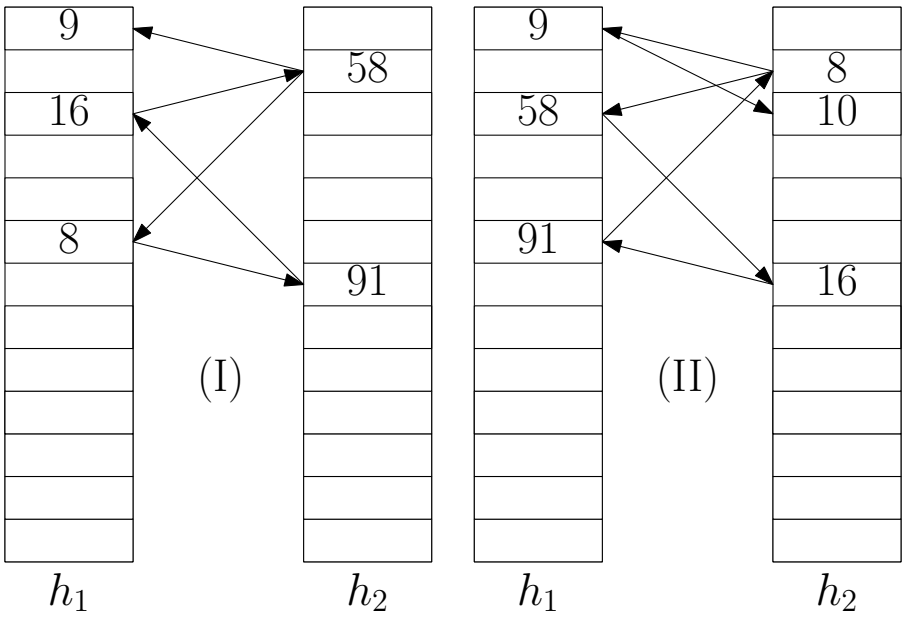


Figure 3.1: Demonstrates Cuckoo hashing. (I) is the state of hash table before inserting 10. Each edge represents an element. Arrows represent the current position of the element in the hash table. Tail represents the corresponding position of the element in other hash table. Apparently inserting 10 is an unlucky case. All the elements gets swapped to the position in the different hash table. Finally 10 finds its position in hash table corresponding to h_2

Definition: Cuckoo Graph is a directed graph $G(V, E)$ where V is the set of all the buckets in the the two hash tables T_1, T_2. If $x \in p_1 = T_1[h_1(x)]$ and $p_2 = T_2[h_2(x)]$ then $(p_1, p_2) \in E$. Similarly if

$y \in q_1 = T_2[h_2(x)]$ and $q_2 = T_1[h_1(x)]$ then $(q_1, q_2) \in E$.

We will use Cuckoo Graph to show the necessary and sufficient conditions of hashing to succeed.

Lemma: *Let x be the element to be inserted, further say CC is the connected component corresponding to $T_1[h_1(x)]$ in the corresponding Cuckoo Graph. Hashing fails if CC has two cycles.*
Proof: If $CC(V_{cc}, Ecc)$ has two cycles then, $|E_{cc}| > |V_{cc}|$ thus inducing one more element in CC will lead to overstuffing. Thus hashing fails. Rehash is required.

Lemma: *Similarly hashing will succeed (rehash is not required) if CC has at most one cycle or is a tree.*
Proof: There are three sub cases for this

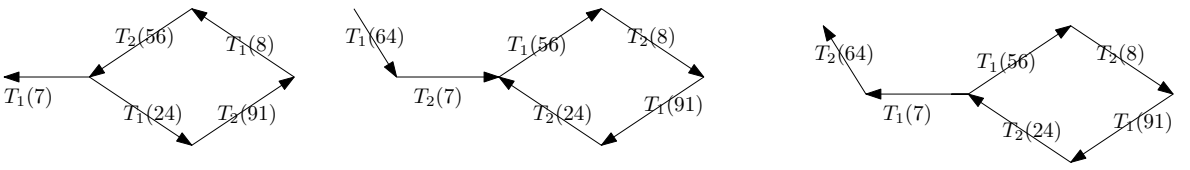
Case1, Cycle is not traversed while inserting This is fairly trivial. Since you don't enter a cycle. the element is accommodated in one of the hash tables without entering a cycle. Thus it cannot loop and hashing succeeds.

Case:2, Cycle is traversed, and finishes inside it Since Cuckoo Graph is Bipartite in nature. Thus the eviction transcends from one table to the other. This finally leads to new position for the element to be inserted. Figure. 3.2 demonstrates this case.

Before Inserting After Inserting

Figure 3.2: Demonstrates Case2. Here we insert 32 in the table. $h_1[32] = h1[5]$, this creates collision. This results in eviction of 5 from T_1. Note that finally 32 finds its right place in T_2.

Case3, Cycle is traversed and finishes outside the cycle New element to be inserted collides with an element outside the cycle, further the elements transcends to both the tables and finally a place for the new entrant is found. It is also intuitive from this case about how $maxLoop \leq 1 + totalElem$.



I) Before Inserting II) Intermediate Step III) After Inserting

Figure 3.3: Demonstrates Case3.

3.1 Analysis of Cuckoo Hashing

We want to show the following:

1. Work done per insertion $\leq$ $2\times$ the size of connected components
2. E[Size of any connected component] $\leq 1 + \frac{1}{\epsilon} \leq O(\frac{1}{\epsilon})$
3. With high probability all connected components have $\leq O(\frac{\log n}{\epsilon})$ size on average.
4. Pr[rehashing] = Pr[getting 2 cycles in same connected component]

The proof of point 4 can be seen in [2]. So we will prove the rest of the three statements here.

Point 1: This is evident from the algorithm previously described. Each connected component is traversed at most twice as edges in a component can be flipped to break a cycle of rehashing. In the graph such a cycle occurs when a vertex has no outgoing edge.

Points 2 and 3: For proving these, we first try to bound $\forall$ u sampled uniformly at random from $V(G)$, E[size of connected component u belongs to]. As the Cuckoo Graph is generated using the hash functions, for ease of understanding, we cheat and view each edge in the cuckoo graph as being generated uniformly at random. We use this to explore the connected components in a BFS manner, starting with an arbitrary vertex, and seeing how many connections it could have, and then exploring all the children.

The stepwise algorithm for constructing this BFS-like random tree corresponding to the Cuckoo Graph is as follows:

- Sample u.
- Sample children v_i of u, such that (v_i, u) or $(u, v_i) \in E(G)$. Random variables $\{Z_i\}_{1\leq i\leq n}$ are sampled. If $h_1(Z_i) = u$ and $h_2(Z_i) = v_i$, then node v_i is added as a child of u. $\{Z_i\}_{1\leq i\leq n}$ are binomial random variables sampled from the distribution Bin$(n, \frac{1}{m})$.
- The above process is continued from each leaf v_i.
- The construction is over when sampling from each current leaf v does not add any vertex, i.e $\forall Z \sim \text{Bin}(n, 1/m),\ h_1(x) \neq v$.

Each node in this tree has at most n children with probability $1/m$ for each edge. From each node, all n r.vs are resampled. Hence the sampled vertices are binomial random variables of the form Bin$(n, \frac{1}{m})$. As $\frac{1}{m} = \frac{1}{(1+\epsilon)n}$,

$$\forall Z_i,\ \text{E}[Z_i] = \frac{1}{1+\epsilon},$$

i.e on average, there is < 1 children attached to the root u.

Let X_i denothe the number of leaves after depth i. Hence,
$X_0 = 1$
$X_{i+1} = \sum_{j=1}^{X_i} Z_j$, such that $Z_j = \text{Bin}(n, \frac{1}{m})$.

The randomized algorithm terminates at depth l when $X_l = 0$. The size of the tree is $\sum_{i=0}^{l-1} X_i$. Now, $\forall i, Z_i = Z$ such that $Z = \text{Bin}(n, \frac{1}{m})$.

Claim 3.1. $\mathrm{E}[X_i] = \mathrm{E}[Z]^i$

Proof. (*By Induction*)
Base Case: $\mathrm{E}[X_0] = 1 = \mathrm{E}[Z]^0$ as $X_0 = 1$.

Assuming inductively for i,
$\mathrm{E}[X_{i+1}] = \mathrm{E}[\sum_{j=1}^{X_i} Z_j]$.
By Wald's equation, $\mathrm{E}[\sum_{j=1}^{X_i} Z_j] = \mathrm{E}[X_i]\mathrm{E}[Z]^i$
Using the inductive hypothesis, $\mathrm{E}[\sum_{j=1}^{X_i} Z_j] = \mathrm{E}[Z]^{i+1}$.

□

Proof. *(Point 2)*

$$\begin{aligned}
\mathrm{E}[\text{size of any connected component}] =& \mathrm{E}[X_0 + X_1 + \cdots] \\
=& 1 + \mathrm{E}[Z] + \mathrm{E}[Z]^2 + \cdots \text{ (by above claim)} \\
=& \frac{1}{1 - \mathrm{E}[Z]} \\
=& \frac{1}{1 - \frac{1}{1+\epsilon}} \\
=& 1 + \frac{1}{\epsilon} \;=\; O(\frac{1}{\epsilon})
\end{aligned}$$

□

To analyse the total size of all connected components, we need to count the number of connected components. A stepwise procedure for that would be as follows:

1. $C = 1$ initially, where C is the counter for connected components. Let root = unspent.
2. Run algorithm 3.1, introducing $z_1 \sim Z = \mathrm{Bin}(n, \frac{1}{m})$ many unspent vertices.
3. $C \leftarrow C + z_1 - 1$, since the root and its children form a single connected component.

We continue above steps until $X_l = 0$. If the total number of steps taken is l, then for t steps, $0 \le t \le l$, we want to show, with high probability,

$$C_t = 1 + \sum_{i=1}^{t} (z_i - 1) < 1$$

Hence, the probability that size of all connected components is small can be represented by:

$$\begin{aligned}
&\Pr[C_t \ge 1] \text{ is small} \\
&= \Pr[1 + \sum_{i=1}^{t} (z_i - 1) \ge 1] \text{ is small} \\
&= \Pr[\sum_{i=1}^{t} z_i \ge t] \text{ is small}
\end{aligned}$$

We will estimate this probability using concentration bounds.

$$\Pr[\text{There will be unspent vertices after } t \text{ explored vertices}]$$
$$= \Pr[\sum_{i=1}^{t} z_i \geq t]$$

Using Bernstein's Bound, $\Pr[C_t > \mathrm{E}[C_t] + t] < exp(\frac{-t^2/3}{\sigma^2 + t/3})$, and given that $\mathrm{E}[C_t] = t\mathrm{E}[Z] = \frac{t}{1+\epsilon}$, and substituting $t = \frac{10 \log n}{\epsilon}$, we get,

$$\Pr[C_t \geq \frac{20 \log n}{\epsilon}] \leq \frac{1}{n^4}$$

which proves Point 3.

References

[1] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004. 3

[2] Reinhard Kutzelnigg. Bipartite random graphs and cuckoo hashing. In *Fourth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities*, pages 403–406. Discrete Mathematics and Theoretical Computer Science, 2006. 3.1