Lecture 11

Hashing: *static* perfect hashing via FKS, *dynamic* cuckoo hashing

The Problem: Membership/Dictionary: maintain a set S of n items from a universe U under:
- query(x): $x \in S$? (+ information associated with x)
- insert(x) (dynamic)
- delete(x) (dynamic)

The Solution: A hash function $h : U \to [m]$ for some positive integer $m < |U|$.

- maintain a table $T[1 \dots m]$ of linked lists (chains)
- insert(x): add x to $T[h(x)]$.
- query(x): scan $T[h(x)]$.
- $\forall h$ there exist $x \neq y$ s.t $h(x) = h(y) \Rightarrow$ our goal is short chains.

Theorem 1. *If $m > n$ and h is selected uniformly from all hash functions then insert/delete/query take $O(1)$ expected time.*

However, a random hash function requires $|U| \lg m$ bits to represent $\Rightarrow$ infeasible.

Universal Hashing:

weak universal hashing is enough to obtain $O(1)$ expected time per operation.

Definition 1. *A set $\mathcal{H}$ of hash functions is a weak universal family if for all $x, y \in U$, $x \neq y$,*

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(y)] = \frac{O(1)}{m}.$$

- Sometimes called *d-universal* for probability= $\frac{d}{m}$.
- Why is weak universal enough?
Pick m so that $\frac{n}{m} = O(1)$, and randomly pick $h \in \mathcal{H}$. Let $I_y = 1$ iff $h(x) = h(y)$.

$$E[\text{chain length}] = E\left[\sum_{y \in S} I_y\right] = \sum_{y \in S} E[I_y] = 1 + \sum_{y \neq x} \Pr[h(x) = h(y)] \leq 1 + n \cdot \frac{O(1)}{m} = O(1)$$

- Dictionary construction: randomly choose $h \in \mathcal{H}$ and hash all elements. If there's a chain that is "too long", pick a new h and *rehash* (rebuild the table from scratch, we will use this idea a lot).

- An example weak universal family: $\mathcal{H}_{p,m} = \{h_{a,b} \mid a \in \{1, 2, \dots, p\}, b \in \{0, 1, 2, \dots, p\}\}$, for some prime $p > |U|$, where $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$. Proof in CLRS.

Definition 2. *$\mathcal{H}$ is a strong universal family if for all distinct $x, y \in U$, and for all $a, b \in [m]$,*

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = a \wedge h(y) = b] = \frac{O(1)}{m^2}.$$

Definition 3. *$\mathcal{H}$ is k-independent if for all k distinct items $x_1, \dots, x_k \in U$, and for all $a_1, \dots, a_k \in [m]$,*

$$\Pr_{h \leftarrow \mathcal{H}}[h(x_1) = a_1 \wedge h(x_2) = a_2 \wedge \dots \wedge h(x_k) = a_k] = \frac{O(1)}{m^k}.$$

- An example k-independent family: again, pick some prime $p > |U|$.

$$\mathcal{H} = \{h \mid h(x) = (c_0 + c_1 x + \cdots + c_{k-1} x^{k-1}) \bmod m, \text{ for some } c_0, c_1, \ldots, c_{k-1} \in [p]\}.$$

Theorem 2 (Siegel, 1989). *$\forall \varepsilon > 0$, $\exists$ a $n^{\Omega(1)}$-independent family of hash functions, each represented in n^{ε} space, and evaluated in $O(1)$ time.*

Theorem 3 (Pagh, Ostlin, 2003). *$\exists$ a n-independent family of hash functions, each represented in $O(n)$ words, and evaluated in $O(1)$ time.*

Worst-case Guarantees in Static Hashing:

-Universal hashing gives good performance only in expectation $\Rightarrow$ vulnerable to an adversary.

Theorem 4 (Gonnet, 1981). *Let $\mathcal{H}$ be an n-independent family of hash functions. The expected length of the longest chain is $\Theta\left(\frac{\lg n}{\lg \lg n}\right)$.*

$\Rightarrow$ We can construct a static hash table with $\Theta\left(\frac{\lg n}{\lg \lg n}\right)$ worst-case query time:

- pick a random $h \in \mathcal{H}$, hash every $x \in S$ (in $O(n)$ time).
- if longest-chain $\leq 2\cdot$expected-length then stop.
- otherwise, pick a new h and start over.

Pr(bad hash function) $\leq \frac{1}{2} \Rightarrow O(1)$ trials, $O(n)$ expected construction time.

- Mitzenmacher 1996 [3]: By using two hash functions (insert to the shorter list, search is in both lists) we can get $\Theta(\lg \lg n)$ worst-case query time.

FKS - Static Hashing (Fredman, Komlós, Szemerédi [1])

- Construct static hash table with no collisions in expected $O(n)$ time, $O(n)$ worst-case space, and $O(1)$ worst-case query time.
- Requires only a weak universal family $\mathcal{H}$
- Easy to implement.

First attempt: If $m = \Omega(n^2)$ and we randomly pick $h \in \mathcal{H}$ then

$$E[\text{number of collisions}] = \sum_{x,y \in S,\ x \neq y} \Pr[h(x) = h(y)] = \binom{n}{2} \cdot \frac{c}{m} \leq \frac{1}{2}$$

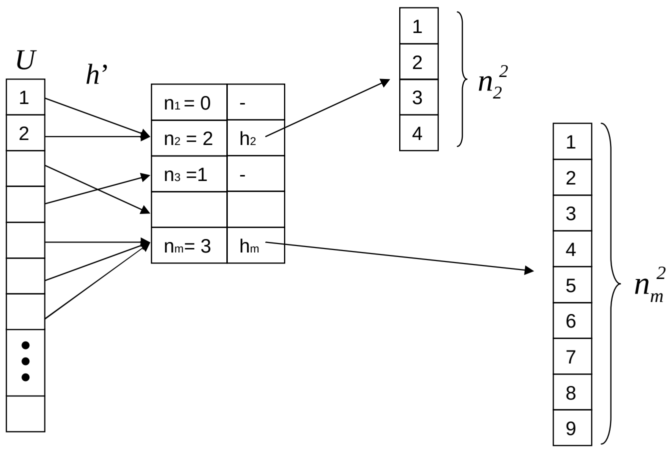
$\Rightarrow$ After expected $O(1)$ trials, we get a collision-free hash function (total time is $O(m) = O(n^2)$).

Second attempt: If $m = n$, the same calculation yields

$$E[\text{number of collisions}] = \binom{n}{2} \cdot \frac{c}{n} = O(n)$$

$\Rightarrow$ After expected $O(1)$ trials, we find a function h' that produces $O(n)$ collisions (total time is $O(n)$).

FKS: Use h' to hash into n buckets, then use h_i's to hash a bucket of size n_i to n_i^2 locations.



Let $n_i = |\{x \in S \mid h'(x) = i\}|$.

(I) The number of collisions is $\sum_{i\in[m]} \binom{n_i}{2} = O(n)$ because we choose h' so. Thus,

$$\sum_{i\in[m]} n_i^2 = O\left(\sum_{i\in[m]} \binom{n_i}{2}\right) = O(n).$$

(II) We can hash n_i elements into a table of size n_i^2 without any collisions in expected $O(n_i^2)$ time.
$\Rightarrow$
- The construction takes $O(n) + O(n_1^2) + \ldots + O(n_m^2) = O(n)$ time in expectation
- Worst-case $O(n)$ space.
- Worst-case $O(1)$ query time (two hashes).

Cuckoo - Dynamic Hashing (Pagh and Rodler 2001 [5])

On the nesting habits of the Cuckoo bird...

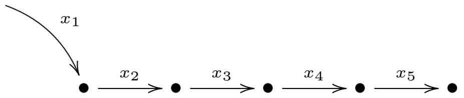
- $O(1)$ expected time for insert
- $O(1)$ worst-case time for queries/deletes.
- Requires two $O(\lg n)$-independent hash functions, h_1 and h_2. (OPEN: same bound using only $O(1)$-independent hash family)
- $m > 2n$ (we will use $m = 4n$).
- Invariant: x is either at $T[h_1(x)]$ or at $T[h_2(x)]$ $\Rightarrow$ query/delete takes worst-case two probes.

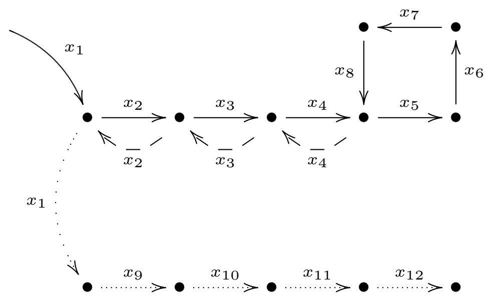
Insertion:

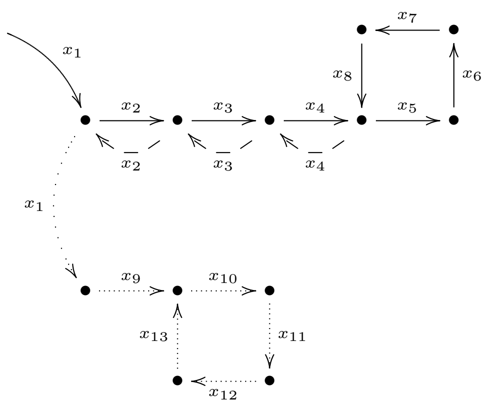
1. Compute $h_1(x)$,
2. If $T[h_1(x)]$ is empty, we put x there, and we are done.
   Otherwise, if $y \in T[h_1(x)]$, we evict y and put x in $T[h_1(x)]$.
3. We find a new spot for y by looking at $T[h_1(y)]$ or $T[h_2(y)]$ (the one that is not occupy by x).

4. Repeat this process. After $6\lg n$ steps stop and rehash.

Let $x_1, x_2, \ldots, x_t$ be the items that are evicted during the process.
- Cuckoo graph $G = (V, E)$, where $V = [m]$ and $(h_1(x), h_2(x)) \in E$ for all $x \in U$. Insertion is one of three possible walks on G:







Key observation: our functions are $O(\lg n)$-independent so we can treat them as truly random functions.

- **No cycle:** $\Pr[1^{st} \text{ eviction}] = \Pr[T[h_1(x_1)] \text{ is occupied }] \leq$

$$\sum_{x \in S, x \neq x_1} (\Pr[h_1(x) = h_1(x_1)] + \Pr[h_2(x) = h_1(x_1)]) < 2n\frac{1}{m} = \frac{2n}{4n} = \frac{1}{2}.$$

  By same reasoning, $\Pr[2^{nd} \text{ eviction}] \leq 2^{-2}$, and $\Pr[t^{th} \text{ eviction}] \leq 2^{-t} \Rightarrow$ the expected running time of this case is $\leq \sum_{t=1}^{\infty} t \cdot 2^{-t} = O(1)$.

  Also, $\Pr[\text{rehash}] \leq 2^{-6\lg n} \leq \frac{1}{n^2}$ $(*)$

- **One cycle:** One of the path parts (solid, dashed or dotted) is at least $t/3$ long.
  $\Rightarrow$ the expected running time of this case is $\leq \sum_{t=1}^{\infty} t \cdot 2^{-t/3} = O(1)$.

Also, $\Pr[\text{rehash}] \leq 2^{-(6 \lg n)/3} = \frac{1}{n^2}$ $(*)$

- **Two cycles:** Counting argument. How many two-cycle configurations are there?
  - The first item in the sequence is x_1.
  - At most n^{t-1} choices of other items in the sequence.
  - At most t choices for where the first loop occurs, t choices for where this loop returns, and t choices for when the second loop occurs.
  - We also have to pick $t-1$ hash values to associate with the items.

  $\Rightarrow$ At most $t^3 n^{t-1}(4n)^{t-1}$ configurations.

  The probability that a specific configuration occurs is $2^t(4n)^{-2t}$. Why?
  $\Rightarrow$ The probability that some two-cycle configuration occurs is at most

$$\frac{t^3 n^{t-1}(4n)^{t-1}2^t}{(4n)^{2t}} = \frac{t^3}{4n^2 2^t}.$$

  $\Rightarrow$ The probability that a two-cycle occurs at all is at most

$$\sum_{t=2}^{\infty} \frac{t^3}{4n^2 2^t} = \frac{1}{4n^2} \sum_{t=2}^{\infty} \frac{t^3}{2^t} = \frac{1}{2n^2} \cdot O(1) = O\left(\frac{1}{n^2}\right) . (*)$$

By $(*)$'s, $\Pr[\text{insertion causes rehash}] \leq O(1/n^2)$.
$\Rightarrow \Pr[n \text{ insertions cause rehash}] \leq O(1/n)$.
$\Rightarrow$ Rehashing (n insertions) succeeds with prob. $1 - O(1/n)$, so after constant number of trials.
  - A trial takes $n \cdot O(1) + O(\lg n) = O(n)$ time in expectation.

$\Rightarrow$ Rehashing takes $O(n)$ time in expectation.
$\Rightarrow$ The expected running time of an insertion is $O(1) + O(1/n^2) \cdot O(n) = O(1) + O(1/n) = O(1)$.

References

1. M. Fredman, J. Komlós, E. Szemerédi, *Storing a Sparse Table with $O(1)$ Worst Case Access Time*, Journal of the ACM, 31(3):538-544, 1984.
2. G. Gonnet, *Expected Length of the Longest Probe Sequence in Hash Code Searching*, Journal of the ACM, 28(2):289-304, 1981.
3. M. Mitzenmacher, *The Power of Two Choices in Randomized Load Balancing*, Ph.D. Thesis 1996.
4. A. Ostlin, R. Pagh, *Uniform hashing in constant time and linear space*, 35th STOC, p. 622-628, 2003.
5. R. Pagh, F. Rodler, *Cuckoo Hashing*, Journal of Algorithms, 51(2004), p. 122-144.
6. A. Siegel, *On universal classes of fast hash functions, their time-space tradeoff, and their applications,* 30th FOCS, p. 20-25, Oct. 1989.