Perfect Hashing, Cuckoo Hashing

Birthday (non)Paradox

- **Q:** How many people must be in a room for it to be more likely than not that two have ***the same birthday***?
- **A:** 23. $\mathcal{E}$ = event that all birthdays are distinct.
  - $\Pr(\mathcal{E}) = 1\left(1-\frac{1}{365}\right)\left(1-\frac{2}{365}\right)\cdots\left(1-\frac{22}{365}\right) < 1/2.$
- $S \subset [u]$ is a set of n keys.
- $h\colon [u] \to [m]$ is a uniformly random hash function.
- **Q:** How big must m be for h to be *injective* on S with constant probability?
- **A:** $m = n^2$ will do.
  - And this is true even if h is only pairwise independent!

$$\prod_{i=1}^{2n} \left(1 - \frac{1}{365}\right) \approx \prod_{i=1}^{2n} e^{-\frac{1}{365}} = e^{\sum_{i=1}^{2n} -\frac{1}{365}} = e^{-\frac{2n}{365}}$$

X = # of collisions

(i,j) $\quad i,j \in S$ $\quad P(i) = P(j)$

$$P(X \geq 1) \leq \frac{E(X)}{1} < \frac{1}{2}$$

$X = X_1 + X_2 + \cdots + X_{\binom{n}{2}}$

$$E(X) = \sum_{i=1}^{\binom{n}{2}} X_i = \binom{n}{2} E(X_i) = \binom{n}{2}\left(\frac{1}{m}\right) \qquad m = n^2$$

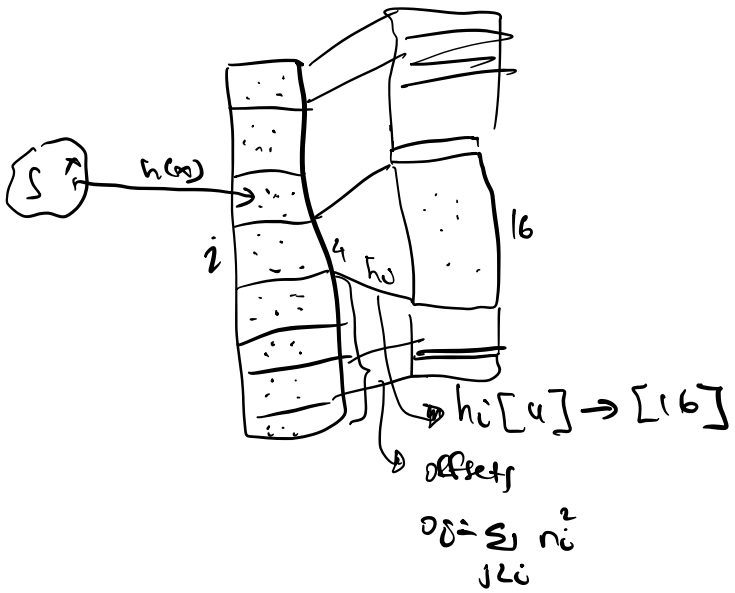
↳ pairwise independence $= \binom{n}{2} P(h(x) = h(y))$ $\Rightarrow \dfrac{\frac{n(n-1)}{2}}{n^2} < \frac{1}{2}$

Data Structures

- The ***dictionary*** problem: maintain a dynamic set S of ***keys*** (with associated data)
  - Insert(x) : add x to S
  - Delete(x) : delete x from S
  - Lookup(x) : is $x \in S$? (Return ptr. to associated data if yes.)

- ***Static case***: S is given in advance; we only need to support Lookup(x) operations.

Perfect hashing in $O(n^2)$ space. 2-wise indep h, alloc $A[0, \ldots, n^2-1]$
$S \subseteq [u]$ Store $x \in S$ in $A[h(x)]$. 50% chance h will work, can keep picking h's. But space is large



$h:[u] \to [n]$

$S_i = \{x \in S \mid h(x) = i\}$

$n_i = |S_i|$

$|A| = \sum_{i=0}^{n-1} n_i^2$

Space: $h, h_1, \dots, h_n$, hash fns

$\sum n_i^2$ slots, extra array

$O_1, \dots, O_{n-1}$ offsets

$O(n)$

$\delta_i = \sum_{j<i} n_i^2$

Back to before, $m = n$

X = # collisions

(i,j) $h(i) = h(j)$ $i \neq j$

$E(X) = \frac{\binom{n}{2}}{n} = \frac{n-1}{2}$ $\quad n_i = |\{i \mid S_i = \{x \mid h(x) = i\}$

$X = \sum_i \binom{n_i}{2} = \sum_{i=1}^{n} \frac{n_i(n_i-1)}{2} \leq \frac{n}{2} + \sum \frac{n_i^2}{2} \leq n + 2X$

$\sum n_i^2 \leq n + 2\sum \binom{n_i}{2}$

$E(\sum n_i^2) \leq E(n + 2\sum_i \binom{n_i}{2}) \leq E(n+2X)$

$\leq n + 2\left(\frac{n-1}{2}\right) = 2n$

1) Pick main h w/ $< 50\%$ collisions

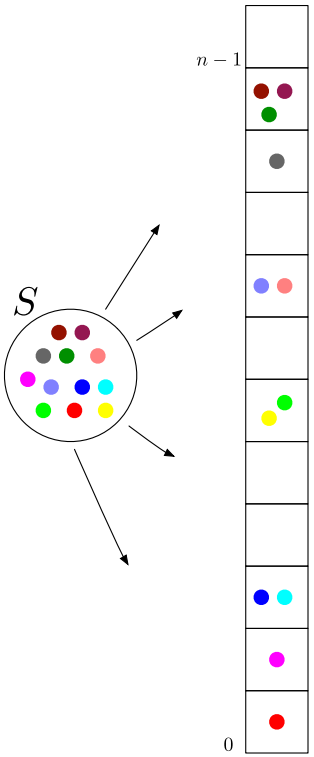
2) Then pick each h_i until each i has an injective h_i. ($\approx O(1)$ time)

3) Overall $O(n)$ space.

Perfect Hashing in $O(n)$ space

[Ajtai, Komlos, Szemeredi]

- $S =$ set of n keys is *fixed*.
- Pick $h: [u] \to [n]$
  - $E[\#\text{ collisions among } S] = \frac{\binom{n}{2}}{n} = \frac{n-1}{2}$.
- $S_i = \{x \in S \mid h(x) = i\}; \;\; n_i = |S_i|$.
- Pick $h_0, \ldots, h_{n-1}$ s.t.
  - $h_i : [u] \to [n_i^2]$
  - and h_i is collision-free on S_i.

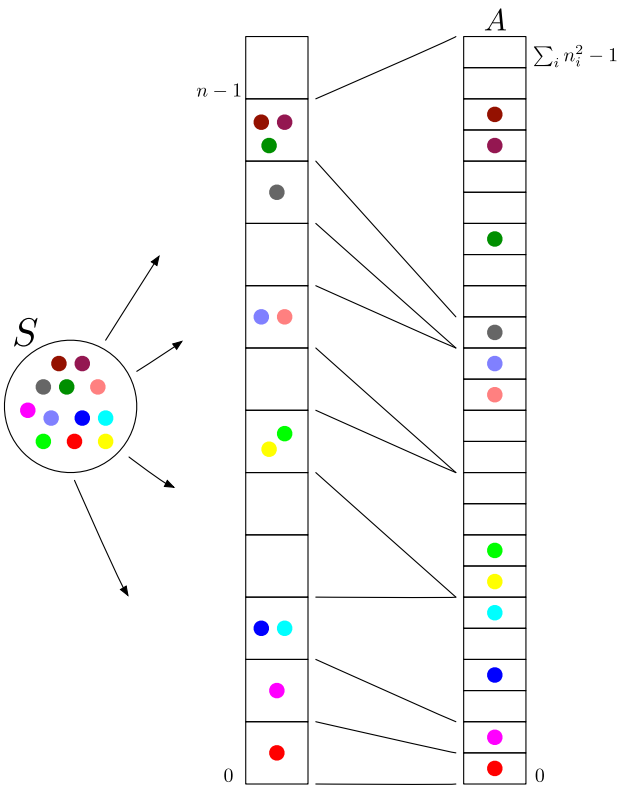


Perfect Hashing in $O(n)$ space

[Ajtai, Komlos, Szemeredi]

- $S =$ set of n keys is *fixed*.
- Pick $h: [u] \to [n]$
  - $E[\#\text{ collisions among } S] = \frac{\binom{n}{2}}{n} = \frac{n-1}{2}$.
- $S_i = \{x \in S \mid h(x) = i\}; \;\; n_i = |S_i|$.
- Pick $h_0, \ldots, h_{n-1}$ s.t.
  - $h_i : [u] \to [n_i^2]$
  - and h_i is collision-free on S_i.
- Allocate array A, $|A| = \sum_i n_i^2$.
- Store S_i in n_i^2 consecutive cells of A.
- The data structure consists of $h, A,$

  $h_0, \ldots, h_{n-1}$, and the offsets $o_i = \sum_{0 \le j < i} n_j^2$.



Cuckoo Hashing

- Hashing with chaining:
  - $O(m+n)$ space.
  - No need to know maximum value of "n" in advance.
  - ***Inserts***, ***Deletes***, ***Lookups*** in $O(1+n/m)$ time in expectation. ***No worst-case time bound.***
- Static perfect hashing:
  - $O(n)$ space.
  - No ***Inserts***/***Deletes*** allowed. ***Lookups*** in $O(1)$ worst-case time.
- **Cuckoo Hashing**
  - $O(n)$ space.
  - ***Deletes*** and ***Lookups*** in $O(1)$ worst-case time.
  - ***Inserts*** in $O(1)$ time *in expectation*.

Cuckoo Hashing

- $n =$ maximum number of elements in set S.
- $r = 2n$
- Allocate arrays $A_1[0..r-1]$ and $A_2[0..r-1]$
- Choose **two** hash functions $h_1, h_2 : [u] \to [r]$

The only invariant:

If $x \in S$ then $A_1[h_1(x)] = x$ or $A_2[h_2(x)] = x$ (but not both)

- Lookup(x) : `return(A1[h1(x)] == x || A2[h2(x)] == x)`
- Delete(x) : `if (A1[h1(x)] == x) A1[h1(x)] = NULL;`
  `if (A2[h2(x)] == x) A2[h2(x)] = NULL;`
- Insert(x) : *the only non-trivial one…*

Cuckoo Hashing

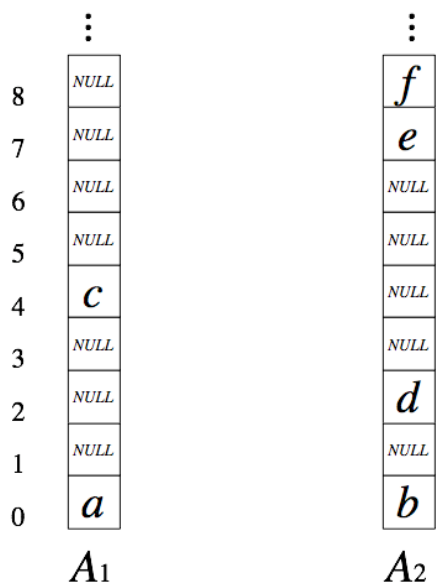
- Lookup(x) : `return(A1[h1(x)] == x || A2[h2(x)] == x)`
- Delete(x) : `if (A1[h1(x)] == x) A1[h1(x)] = NULL;`
  `if (A2[h2(x)] == x) A2[h2(x)] = NULL;`
- Insert(x) : (first check that x ∉ S)

```
z = x;
while(should something go here?) {
    swap(z,A1[h1(z)]);
    if z==NULL return;
    swap(z,A2[h2(z)]);
    if z==NULL return;
}
```

"swap(u,v)" just swaps the values of u and v.

Cuckoo Hashing

- Suppose a, b, c, d, e, f are already in the table
- $h_1(a) = 0, \quad h_1(c) = 4$
- $h_2(b) = 0, \quad h_2(d) = 2, \quad h_2(e) = 7, \quad h_2(f) = 8$

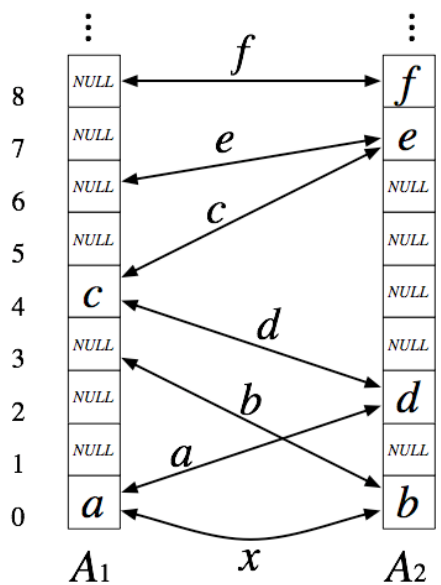


Cuckoo Hashing

- Suppose a, b, c, d, e, f are already in the table
- $h_1(a) = 0, \quad h_1(b) = 3, \quad h_1(c) = 4, \quad h_1(d) = 4, \quad h_1(3) = 6, \quad h_1(f) = 8$
- $h_2(a) = 2, \quad h_2(b) = 0, \quad h_2(c) = 2, \quad h_2(d) = 2, \quad h_2(e) = 7, \quad h_2(f) = 8$

- Suppose we try Insert(x) :

  $h_1(x) = h_2(x) = 0$



Cuckoo Hashing

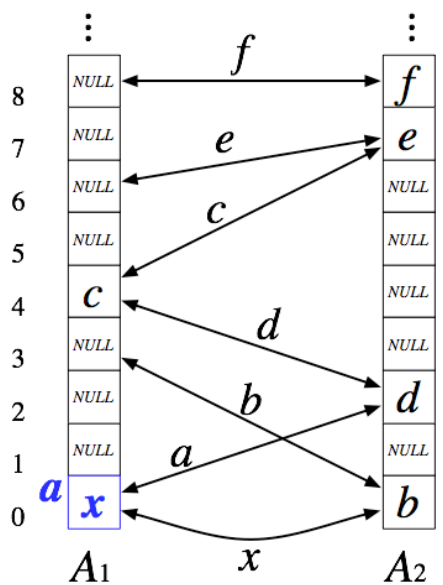
- Suppose a, b, c, d, e, f are already in the table
- $h_1(a) = 0, \quad h_1(b) = 3, \quad h_1(c) = 4, \quad h_1(d) = 4, \quad h_1(3) = 6, \quad h_1(f) = 8$
- $h_2(a) = 2, \quad h_2(b) = 0, \quad h_2(c) = 2, \quad h_2(d) = 2, \quad h_2(e) = 7, \quad h_2(f) = 8$

- Suppose we try Insert(x) :

  $h_1(x) = h_2(x) = 0$

  z = x;

  swap(z,A_1[0]); (now z=a)



Cuckoo Hashing

- Suppose a, b, c, d, e, f are already in the table
- $h_1(a) = 0, \quad h_1(b) = 3, \quad h_1(c) = 4, \quad h_1(d) = 4, \quad h_1(3) = 6, \quad h_1(f) = 8$
- $h_2(a) = 2, \quad h_2(b) = 0, \quad h_2(c) = 2, \quad h_2(d) = 2, \quad h_2(e) = 7, \quad h_2(f) = 8$

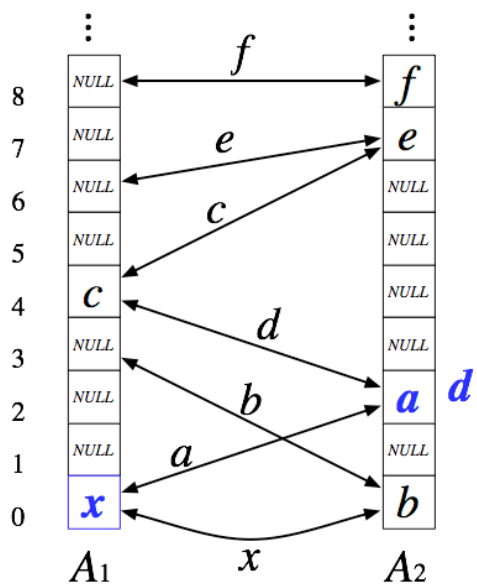
- Suppose we try Insert(x) :

  $h_1(x) = h_2(x) = 0$

  z = x;

  swap(z,A_1[0]); (now z=a)

  swap(z,A_2[2]); (now z=d)



Cuckoo Hashing

- Suppose a, b, c, d, e, f are already in the table
- $h_1(a) = 0, \quad h_1(b) = 3, \quad h_1(c) = 4, \quad h_1(d) = 4, \quad h_1(3) = 6, \quad h_1(f) = 8$
- $h_2(a) = 2, \quad h_2(b) = 0, \quad h_2(c) = 2, \quad h_2(d) = 2, \quad h_2(e) = 7, \quad h_2(f) = 8$

- Suppose we try Insert(x) :

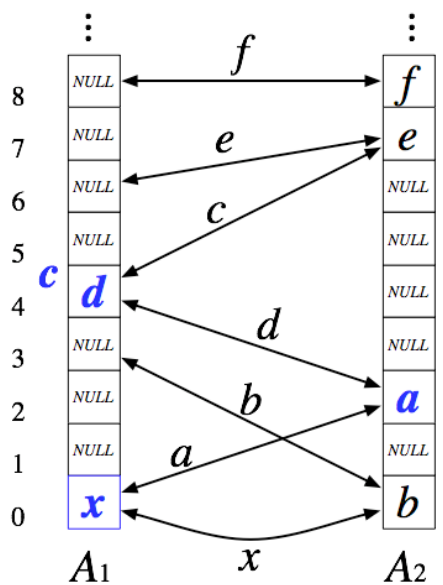
  $h_1(x) = h_2(x) = 0$

  z = x;

  swap(z,A_1[0]); (now z=a)

  swap(z,A_2[2]); (now z=d)

  swap(z,A_1[4]); (now z=c)



Cuckoo Hashing

- Suppose a, b, c, d, e, f are already in the table
- $h_1(a) = 0, \quad h_1(b) = 3, \quad h_1(c) = 4, \quad h_1(d) = 4, \quad h_1(3) = 6, \quad h_1(f) = 8$
- $h_2(a) = 2, \quad h_2(b) = 0, \quad h_2(c) = 2, \quad h_2(d) = 2, \quad h_2(e) = 7, \quad h_2(f) = 8$

- Suppose we try Insert(x) :

  $h_1(x) = h_2(x) = 0$

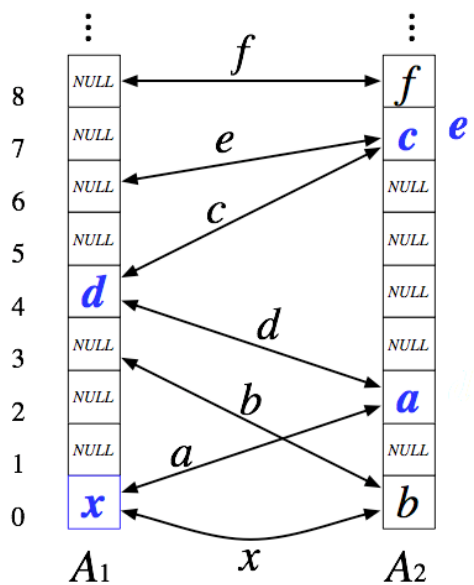
  z = x;

  swap(z,A_1[0]); (now z=a)

  swap(z,A_2[2]); (now z=d)

  swap(z,A_1[4]); (now z=c)

  swap(z,A_2[7]); (now z=e)



Cuckoo Hashing

- Suppose a, b, c, d, e, f are already in the table
- $h_1(a) = 0, \quad h_1(b) = 3, \quad h_1(c) = 4, \quad h_1(d) = 4, \quad h_1(3) = 6, \quad h_1(f) = 8$
- $h_2(a) = 2, \quad h_2(b) = 0, \quad h_2(c) = 2, \quad h_2(d) = 2, \quad h_2(e) = 7, \quad h_2(f) = 8$

- Suppose we try Insert(x) :

  $h_1(x) = h_2(x) = 0$

  z = x;

  swap(z,A_1[0]); (now z=a)

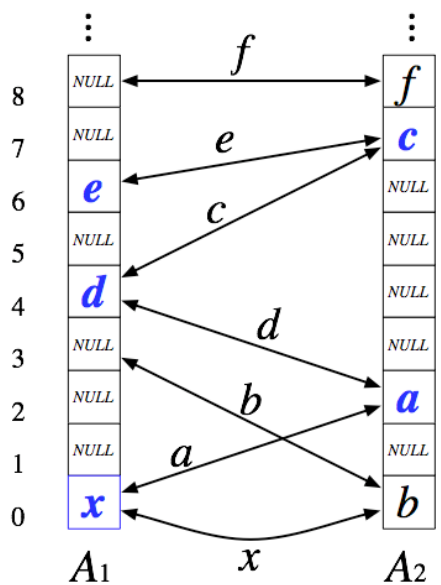
  swap(z,A_2[2]); (now z=d)

  swap(z,A_1[4]); (now z=c)

  swap(z,A_2[7]); (now z=e)

  swap(z,A_1[6]);

  z==NULL and we're done



What's the worst that could happen?

- Insert(x) : (first check that x ∉ S)

```
z = x;
while(should something go here?) {
    swap(z,A1[h1(z)]);
    if z==NULL return;
    swap(z,A2[h2(z)]);
    if z==NULL return;
}
```

What's the worst that could happen?

- Insert(x) : (first check that x ∉ S)

```
z = x;
while(we havn't seen a "closed loop") {
    swap(z,A1[h1(z)]);
    if z==NULL return;
    swap(z,A2[h2(z)]);
    if z==NULL return;
}
pick new hash functions h1,h2
rebuild the whole data structure
```

This takes $\Omega(n)$ time! We need to guarantee this doesn't happen too often

The analysis

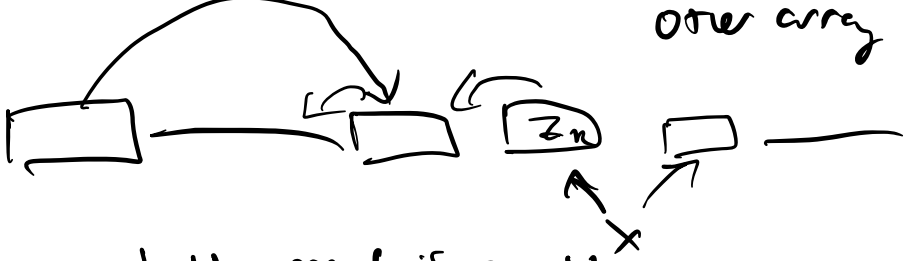
- Review of parameters
  - $\boldsymbol{S} \subseteq [u]$ = set of keys in data structure,
  - $\boldsymbol{u}$ = universe size.
  - $|S| \leq \boldsymbol{n}$ at all times.
  - $\boldsymbol{r} = 2n$. A_1 and A_2 both have length r.
  - Assume the ***IDEAL hash model***:

$\boldsymbol{h_1}, \boldsymbol{h_2}$ are selected <u>*uniformly at random*</u>

from ALL functions from $[u] \to [r]$

Cwas holding z_j, now z_{j-1}

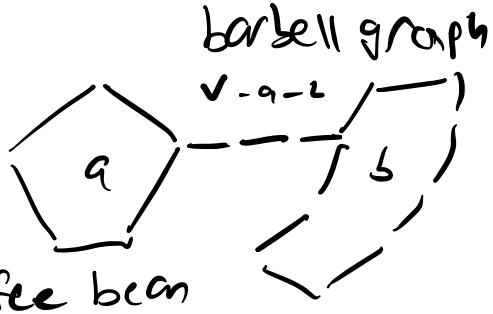
existence of loop is OK, switch back things to prev slot, put x in the



loop on both arrays is a problem.

"closed loop" vertices - array locs
edges - keys

≡ a minimal graph w/ v keys / edges
v-1 array locs / vertices

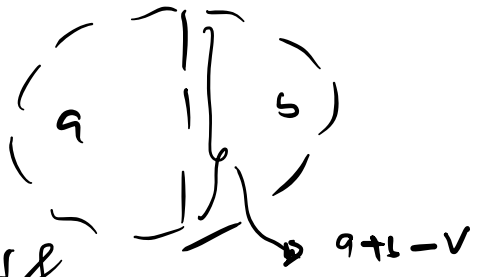


Error Prob

P(finding a closed loop involving X)

1. Choose c ($\leq v^2$) topologically
2. representation in array
3. Choose keys & assignment to edges

Since each vertex is on array cell



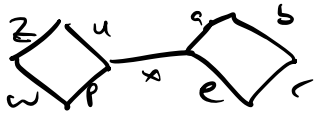
$$= \sum_{V \geq 3} \sum_{\substack{\text{closed loops} \\ c \text{ w/ size } V}} P(c \text{ exists})$$

$\leq r^{v-1}$ r choices of cells each time

$\binom{n}{v-1} v!$ ↖ assign edges

↑ choose keys

$$= \sum_{v \geq 3} v^2 \binom{n}{v-1} r^{v-1} v! \cdot r^{-2v}$$



$$\Phi_{h_1,h_2}\left(\begin{array}{l} h_1(x) = \\ h_2(x) = \\ h_2(a) = \\ h_2(a) = \\ \vdots \end{array} \right) = r^{-2v}$$

$$= \sum_{v \geq 3} v^3 \frac{n^{v-1}}{(v-1)!} r^{v-1} v! \, r^{-2v}$$

$$= \frac{1}{n} \cdot \sum_{v \geq 3} v^3 \left(\frac{n}{r}\right)^v \approx O\left(\frac{1}{n}\right)\left(\sum_{v \geq 3} v^3 \left(\frac{1}{2}\right)^v\right)$$

since $r = 2n$

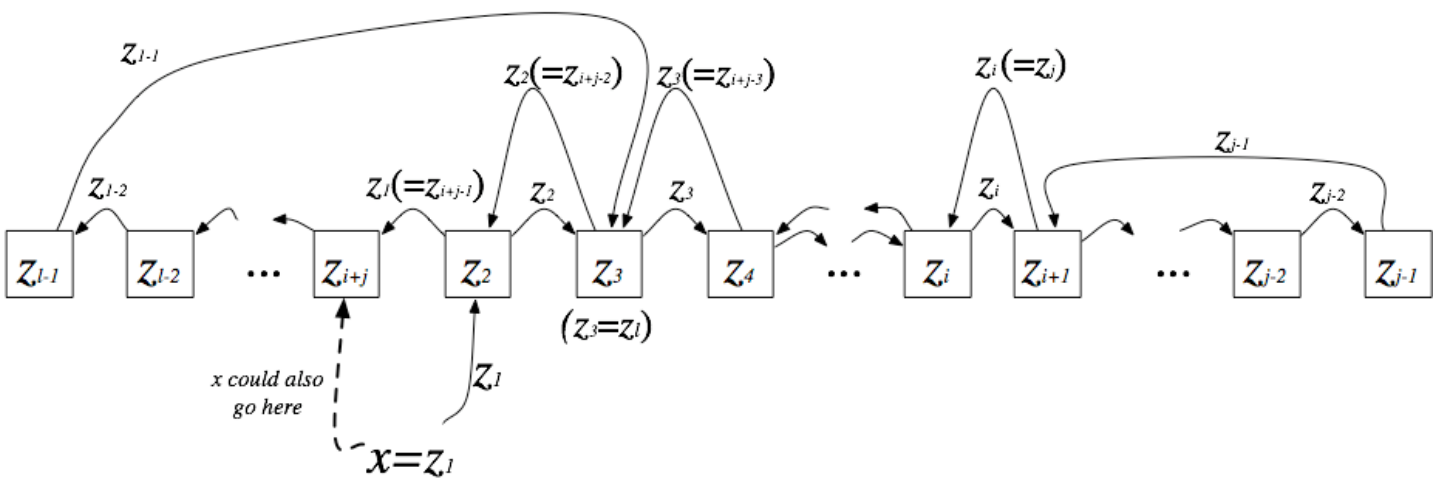
$$= O\left(\frac{1}{n^2}\right)$$

geometric sum converges to constant, $O(1)$

Upper bound on prob that when you insert an edge it is the last piece of a closed loop.

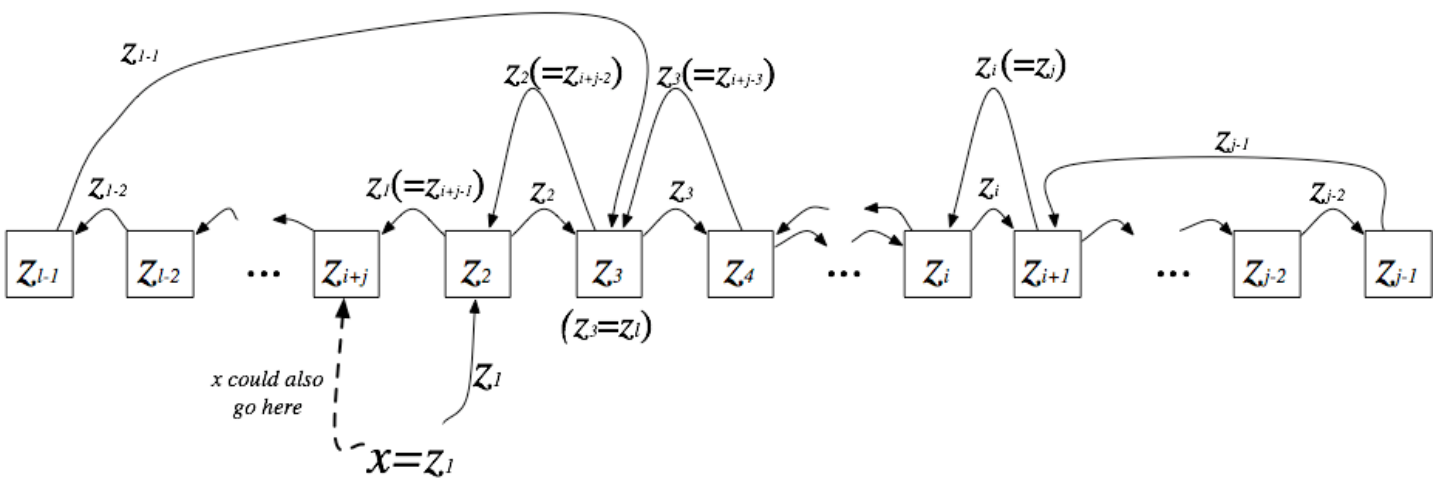
What does a “closed loop” look like?

- Let z_g be the g^{th} value of var. z during an insert.
- Boxes = table cells (alternating A_1 and A_2)
- Boxes labeled with the z_g *originally* there (before the insert)
- Arrow labels = the z_g being relocated during an insert
- By def: $z_j = z_i$ is the *first* key seen twice
- By def: z_l is the *next* key seen twice where $l \geq i+j$.
- If l exists then we have a ***closed loop***.



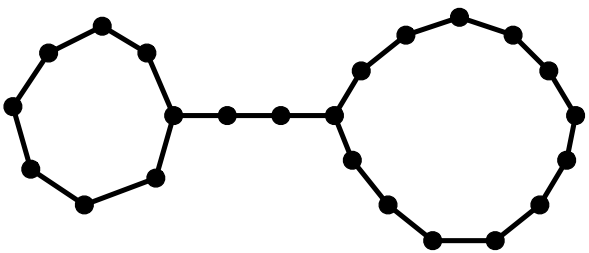
and why is this a bad situation??

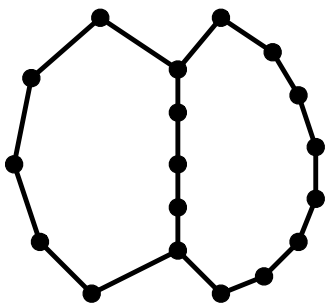
- There are l-i-1 distinct keys ($z_1,\ldots,z_{j-1},z_{i+j},\ldots,z_{l-1}$).
- They cannot be put in only l-i-2 table locations.
- In the analysis "v" = l-i-1 is the number of keys involved in the closed loop.



Closed Loops as Graphs

- Vertices : array cells; Edges : elements in S.
- Closed Loops are minimal graphs with more edges than vertices. They come in two varieties:
- “Bar Bells” and “Coffee Beans”





- How many topologically distinct closed loops are there with v edges and $v - 1$ vertices?

Closed loops are possible, but likely?

- The strategy:

  (1) Enumerate EVERY POSSIBLE closed loop involving x.

  (2) Calculate the probability that this closed loop occurs.

  (3) Sum all probabilities from (2). This is an ***upper bound*** on the probability of there being ***any*** closed loop involving x.

$$\Pr(\exists \text{ a closed loop}) \leq \sum_{v \geq 3} \sum_{\substack{every\ closed \\ loop\ C\ with\ v\ keys}} \Pr(C \text{ exists})$$

What to do now:

Count the number of closed loops with v keys

Calculate the probability that such a loop exists.

Prob comp

counting component

$$\Pr(\exists \text{ a closed loop}) \leq \sum_{v \geq 3} \overbrace{\left(v^2 \binom{n}{v-1} r^{v-1} v! \right)}^{\text{counting component}} \overbrace{(r^{-2v})}^{\uparrow}$$

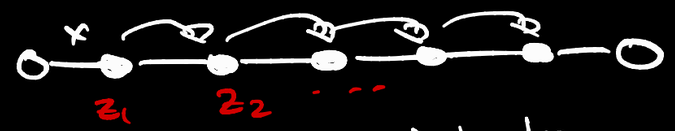
$$\leq \sum_{v \geq 3} \left(v^2 \frac{n^{v-1}}{(v-1)!} \cdot r^{v-1} v! \right) (r^{-2v})$$

$$\leq O(1) \cdot \sum_{v \geq 3} v^3 n^{v-1} r^{v-1} r^{-2v}$$

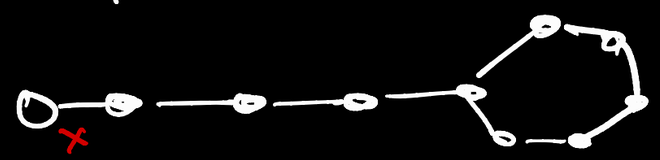
$$= O\left(\frac{1}{nr}\right) \cdot \sum_{v \geq 3} v^3 \left(\frac{n}{r}\right)^v = O\left(\frac{1}{n^2}\right) \sum_{v \geq 3} v^3 2^{-v} = O\left(\frac{1}{n^2}\right)$$

Expected time for insert

→ path with $r \geq 1$ (edges) and v vertices



→ lollipop with $r \geq 1$ keys edges & v vertices



→ with r keys / cells how many of them shape?

↳ paths → 1

↳ lollipops → r

Expected Time for Insert

$\Pr(\exists$ insertion path with size v containing $x)$

$$\leq \binom{n}{v-1} r^v (v-1)! \cdot r^{-(2v-1)}$$

$$\leq \left(\frac{n}{r}\right)^{v-1} = \left(\frac{1}{2}\right)^{v-1}$$

$\Pr(\exists$ lollipop with size v containing $x)$

$$\leq v \binom{n}{v-1} r^v v! \cdot r^{-2v}$$

$$\leq v^2 n^{v-1} r^{-v} = O\left(\frac{v^2}{n}\right)\left(\frac{1}{2}\right)^{v-1}$$

bound

quite small $r \ll n$

→ choose keys $\binom{n}{r-1}$

→ choose edges $\leq r^v$

→ assign keys → $(r-1)!$

→ prob to get this (x is second free)

↳ $r^{-(2v-1)}$

extra r come from here

Exp search time

Expected Time for Insert

- $E(\text{insert time} \mid \text{no closed loop})$

$$= \sum_{v \geq 1} O(v) \cdot \Pr(\exists \text{ size } v \text{ path/lollipop})$$

$$= \sum_{v \geq 1} O(v) \cdot \left(\frac{1}{2}\right)^{v-1} \left(1 + 0\left(\frac{v^2}{n}\right)\right)$$

$$= O(1)$$

$$\Pr(\#\text{insert swaps} > 2\log n) \leq 0\left(\frac{\log^2 n}{n^2}\right).$$

- Insert(x) : (first check that x ∉ S)

```
z = x;
while(#swaps ≤ 2log n) {
   swap(z,A1[h1(z)]);
   if z==NULL return;
   swap(z,A2[h2(z)]);
   if z==NULL return;
}
pick new hash functions h1,h2
rebuild the whole data structure
```

The Punchline

- If insertion time is capped at $O(\log n)$, then all events in the analysis involve $O(\log n)$ keys.
  - [Pagh, Rodler 2001] If h_1, h_2 are **$O(\log n)$-wise independent**, then insertions take $O(1)$ time in expectation and fail with probability $\tilde{O}(n^{-2})$.
  - [Cohen, Kane 2009] ***5-wise independent*** hash functions are ***not enough***. Closed loops appear with high probability.
  - [Open Problem] Is any k-wise independence enough, for some $k = O(1)$?

This is when you should implement exponential search for insertion