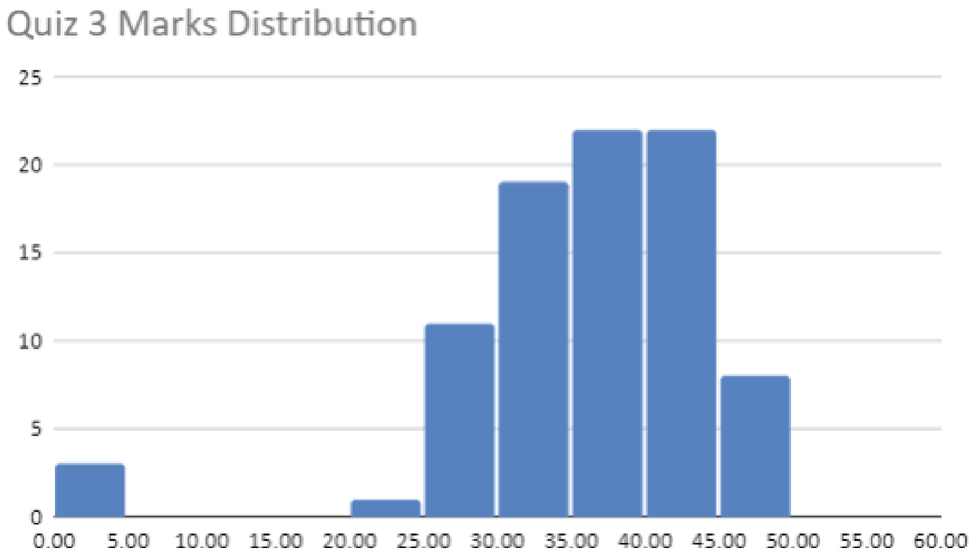
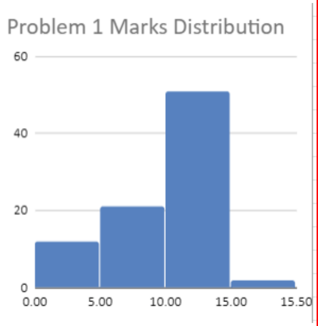
LECTURE 19: CUCKOO HASHING

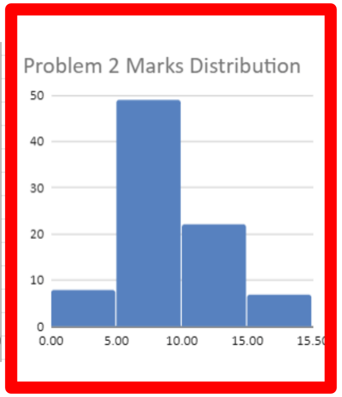
Harold Soh
harold@comp.nus.edu.sg

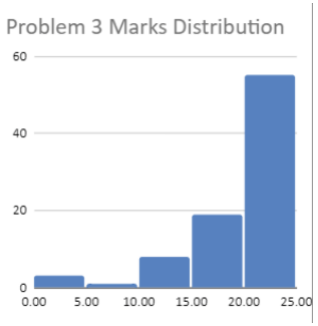
ADMINISTRATIVE ISSUES: QUIZ 3

- Finished grading Quiz 3
- Will be returned on Friday.









PROBLEM 2

Problem 2.b. [10 points] You want to travel from Singapore to Vancouver to attend NewRIPS, the largest AI conference in town. There are no direct flights so, you will have to plan an itinerary involving several flights. Each flight is associated with an origin city, destination city, start time, and flight duration. You need at least 60 minutes to transfer between flights. There are n cities and m flights in total.

Given a set of flights, compute the *earliest* time that you can arrive at the Vancouver given that you start from Singapore. Assume that a flight departing from city A to city B cannot arrive earlier than another flight from A to B that departed earlier. To simplify the problem, flight times are given as integers representing the number of minutes from 12:05pm 30th October 2019. Likewise, flight durations are given in minutes. As an example, consider the flights below:

Origin	Destination	Start-time	Flight Duration
Singapore	Tokyo	120	420
Singapore	Tokyo	230	420
Singapore	Hong Kong	140	240
Singapore	Hong Kong	150	270
Tokyo	San Francisco	500	420
Tokyo	San Francisco	605	420
Hong Kong	Vancouver	220	480
Hong Kong	Vancouver	400	480
Hong Kong	Vancouver	1200	480
San Francisco	Vancouver	200	100
San Francisco	Vancouver	1200	100
San Francisco	Vancouver	2000	100

The earliest you can arrive in Vancouver is 1300, via the following flights:

- Singapore to Tokyo at 120 to arrive at 540
- Tokyo to San Francisco at 605 to arrive at 1025
- San Francisco to Vancouver at 1200 to arrive at 1300.

Note: Flying through Hong Kong is not the fastest valid route. It requires you to take the latest flight that departs at 1200, because you need at least 60 minutes to transfer between flights; the earliest you can arrive at Hong Kong is 380, but you cannot take the flight from Hong Kong to Vancouver at 400.

PROBLEM 2: DISCUSSION

Dijkstra's algorithm with modified relaxation

- Use arrival times instead of distance
- When relaxing, only consider edges (flights) that have start 60 minutes after arrival time.

Common errors/suboptimal solutsions:

- Adding 60 minutes to each flight (possible to do but have to be very careful)
- Complicated graph constructions (that don't work)
- Exploring all paths (e.g., via BFS-like method)

More details at Tutorial on Friday.

FINAL EXAM

4 DEC 2018 (Morning, 9-11am)

2 hours

44 Questions (120 points)

All multiple choice

- Bring pencils to shade in the form

Open-book exam

NEXT WEEK

Lecture on Tuesday

- Final Review + Final Exam ”tips”

No lecture on Wednesday

- Open Office

Poll Everywhere

https://bit.ly/2LvG9bq

QUESTIONS?

THE NEXT 2 DAYS

Probability Review

Bloom Filters

Cuckoo Hashing

CUCKOO HASHING

Did you know?

- Cuckoos lay eggs in other birds nests.
- When the cuckoo bird hatches, it pushes eggs/chicks out of the nest.

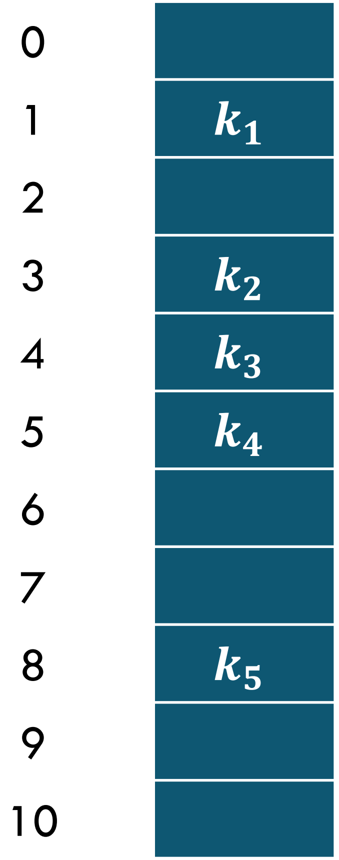
What a neat idea!

- Open addressing policy!
- Described by Rasmus Pagh and Flemming Friche Rodler in 2001.

CUCKOO HASHING: HOW DOES IT WORK?

Use 2 hash functions $h_1(k)$ and $h_2(k)$

So, each key only has **2 possible locations**

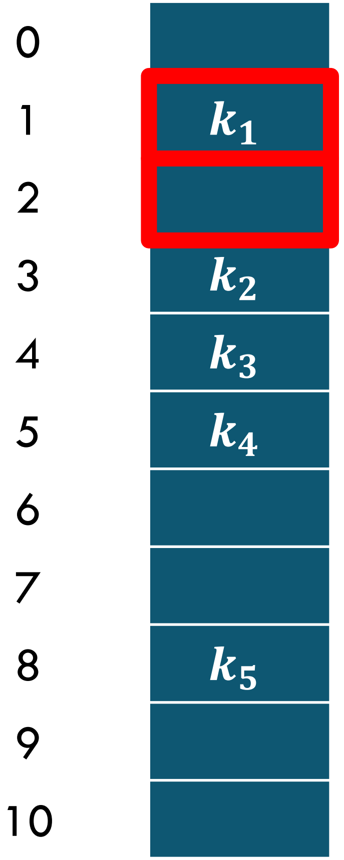


CUCKOO HASHING: HOW DOES IT WORK?

search(k_1)

$h_1(k_1) = 1$

$h_2(k_1) = 2$



Use 2 hash functions $h_1(k)$ and $h_2(k)$

So, each key only has **2 possible locations**

Operations:

- **Lookup:** only check the 2 possible locations
  - $O(1)$ worst-case time!

CUCKOO HASHING: HOW DOES IT WORK?

`delete(`k_1`)`

$h_1(k_1) \;= 1$

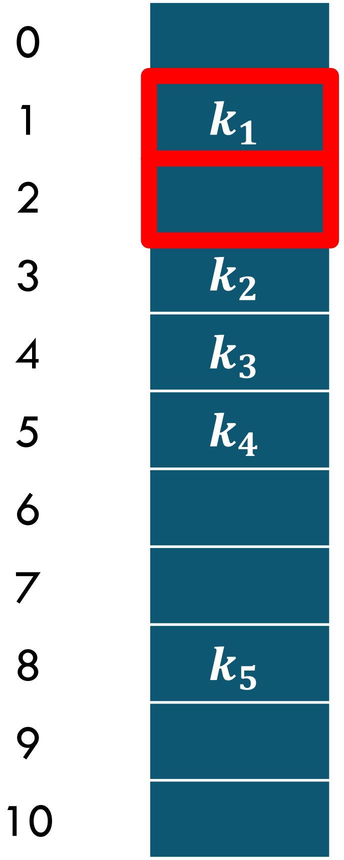
$h_2(k_1) \;= 2$

Use 2 hash functions $h_1(k)$ and $h_2(k)$

So, each key only has **2 possible locations**

Operations:

- **Lookup:** only check the 2 possible locations
  - $O(1)$ worst-case time!
- **Deletion:** only check the 2 possible locations and delete accordingly.
  - Also $O(1)$ worst-case time



CUCKOO HASHING: HOW DOES IT WORK?

`delete(`k_1`)`

$h_1(k_1) = 1$

$h_2(k_1) = 2$

Use 2 hash functions $h_1(k)$ and $h_2(k)$

So, each key only has **2 possible locations**

Operations:

- **Lookup:** only check the 2 possible locations
  - $O(1)$ worst-case time!
- **Deletion:** only check the 2 possible locations and delete accordingly.
  - Also $O(1)$ worst-case time

0	
1	
2	
3	k_2
4	k_3
5	k_4
6	
7	
8	k_5
9	
10	

CUCKOO HASHING: INSERTIONS

insert(k_1)

$h_1(k_1) = 1$

Idea: When inserting, if bucket is taken, push existing key out!

Steps:

- Set $p = h_1(k)$
- Repeat n times
  - If $A[p]$ is empty then
    - $A[p] = k$
    - return
  - $t = A[p]$; $A[p] = k$; $k = t$
  - If $\mathrm{p} = h_1(k)$ then $\mathrm{p} = h_2(k)$ else $\mathrm{p} = h_1(k)$
- rehash(); insert(k)

0	
1	
2	
3	k_2
4	k_3
5	k_4
6	
7	
8	k_5
9	
10	

CUCKOO HASHING: INSERTIONS

insert(k_1)

$h_1(k_1) = 1$

Idea: When inserting, if bucket is taken, push existing key out!

Steps:

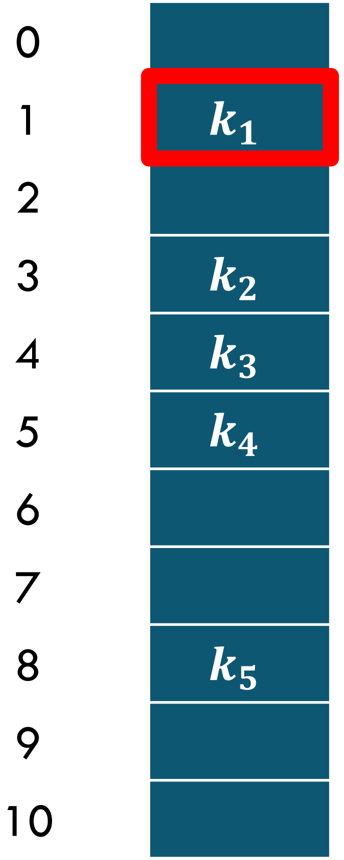
- Set $p = h_1(k)$
- Repeat n times
  - If $A[p]$ is empty then
    - $A[p] = k$
    - return
  - $t = A[p]$; $A[p] = k$; $k = t$
  - If p $= h_1(k)$ then p $= h_2(k)$ else p $= h_1(k)$
- rehash(); insert(k)

0	
1	k_1
2	
3	k_2
4	k_3
5	k_4
6	
7	
8	k_5
9	
10	

CUCKOO HASHING: INSERTIONS

`insert(`k_6`)`

$h_1(k_6) = 1$



Idea: When inserting, if bucket is taken, push existing key out!

Steps:

- Set $p = h_1(k)$
- Repeat n times
  - `If` $A[p]$ `is empty then`
    - $A[p] = k$
    - `return`
  - $t = A[p]$; $A[p] = k$; $k = t$
  - `If` p $= h_1(k)$ `then` p $= h_2(k)$ `else` p $= h_1(k)$
- `rehash(); insert(k)`

CUCKOO HASHING: INSERTIONS

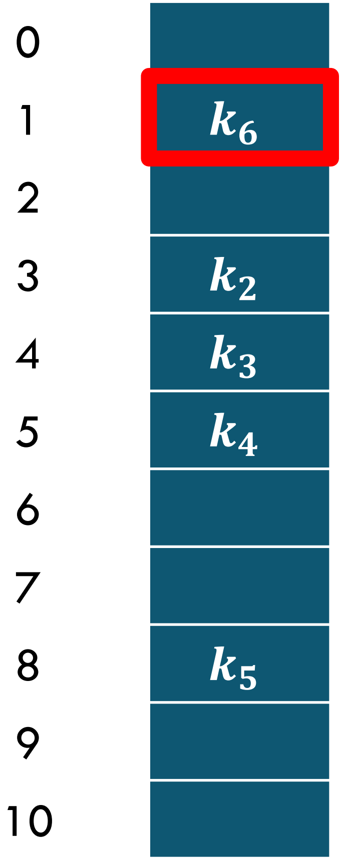
Idea: When inserting, if bucket is taken, push existing key out!

Steps:

- Set $p = h_1(k)$
- Repeat n times
  - `If` $A[p]$ `is empty then`
    - $A[p] = k$
    - `return`
  - $t = A[p]$; $A[p] = k$; $k = t$
  - `If` p $= h_1(k)$ `then` p $= h_2(k)$ `else` p $= h_1(k)$
- `rehash(); insert(k)`

`insert(`k_6`)`

$h_1(k_6) = 1$



k_1 got pushed out!

$h_1(k_1) = 1$

$h_2(k_1) = 2$

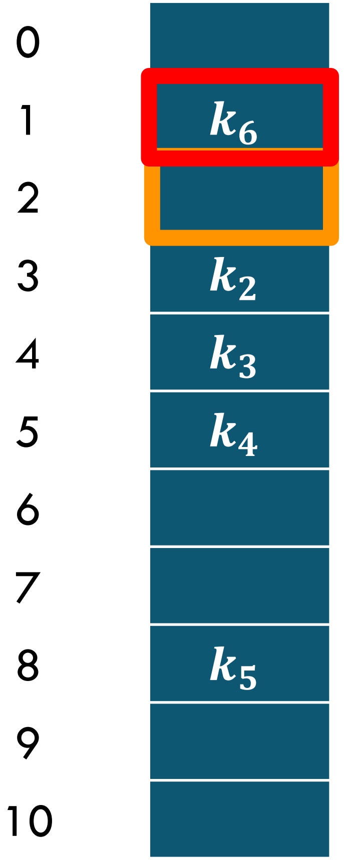
CUCKOO HASHING: INSERTIONS

`insert(`k_6`)`

Idea: When inserting, if bucket is taken, push existing key out!

Steps:

- Set $p = h_1(k)$
- Repeat n times
  - `If` $A[p]$ `is empty then`
    - $A[p] = k$
    - `return`
  - $t = A[p]$; $A[p] = k$; $k = t$
  - `If` p $= h_1(k)$ `then` p $= h_2(k)$ `else` p $= h_1(k)$
- `rehash(); insert(k)`



k_1 got pushed out!

$h_1(k_1) = 1$

$h_2(k_1) = 2$

CUCKOO HASHING: INSERTIONS

`insert(`k_6`)`

Idea: When inserting, if bucket is taken, push existing key out!

Steps:

- Set $p = h_1(k)$
- Repeat n times
  - `If` $A[p]$ `is empty then`
    - $A[p] = k$
    - `return`
  - $t = A[p];\ A[p] = k;\ k = t$
  - `If` $\mathrm{p} = h_1(k)$ `then` $\mathrm{p} = h_2(k)$ `else` $\mathrm{p} = h_1(k)$
- `rehash(); insert(k)`

0	
1	k_6
2	k_1
3	k_2
4	k_3
5	k_4
6	
7	
8	k_5
9	
10	

k_1 got pushed out!

$h_1(k_1) = 1$

$h_2(k_1) = 2$

CUCKOO HASHING: INSERTIONS

`insert(`k_7`)`

$h_1(k_7) = 4$

Idea: When inserting, if bucket is taken, push existing key out!

Steps:

- Set $p = h_1(k)$
- Repeat n times
  - `If` $A[p]$ `is empty then`
    - $A[p] = k$
    - `return`
  - $t = A[p]$; $A[p] = k$; $k = t$
  - `If` p $= h_1(k)$ `then` p $= h_2(k)$ `else` p $= h_1(k)$
- `rehash(); insert(k)`

0	
1	k_6
2	k_1
3	k_2
4	$\boxed{k_3}$
5	k_4
6	
7	
8	k_5
9	
10	

CUCKOO HASHING: INSERTIONS

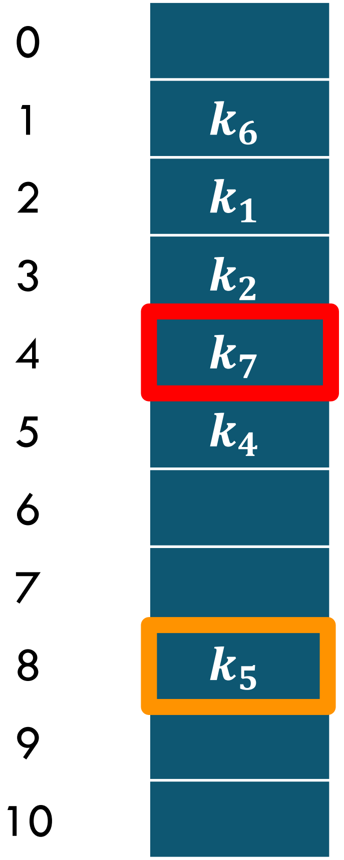
`insert(`k_7`)`

$h_1(k_7) = 4$

Idea: When inserting, if bucket is taken, push existing key out!

Steps:

- Set $p = h_1(k)$
- Repeat n times
  - `If` $A[p]$ `is empty then`
    - $A[p] = k$
    - `return`
  - $t = A[p];\ A[p] = k;\ k = t$
  - `If` $\mathrm{p} = h_1(k)$ `then` $\mathrm{p} = h_2(k)$ `else` $\mathrm{p} = h_1(k)$
- `rehash(); insert(k)`



$\boldsymbol{k_3}$ got pushed out!

$h_1(k_3) = 4$

$h_2(k_3) = 8$

CUCKOO HASHING: INSERTIONS

insert(k_7)

Idea: When inserting, if bucket is taken, push existing key out!

Steps:

- Set $p = h_1(k)$
- Repeat n times
  - `If` $A[p]$ `is empty then`
    - $A[p] = k$
    - `return`
  - $t = A[p]$; $A[p] = k$; $k = t$
  - $\boxed{\texttt{If } p = h_1(k) \texttt{ then } p = h_2(k) \texttt{ else } p = h_1(k)}$
- `rehash(); insert(k)`

0	
1	k_6
2	k_1
3	k_2
4	k_7
5	k_4
6	
7	
8	k_3
9	
10	

k_5 got pushed out!

$\boxed{h_1(k_5) = 0}$

$h_2(k_5) = 8$

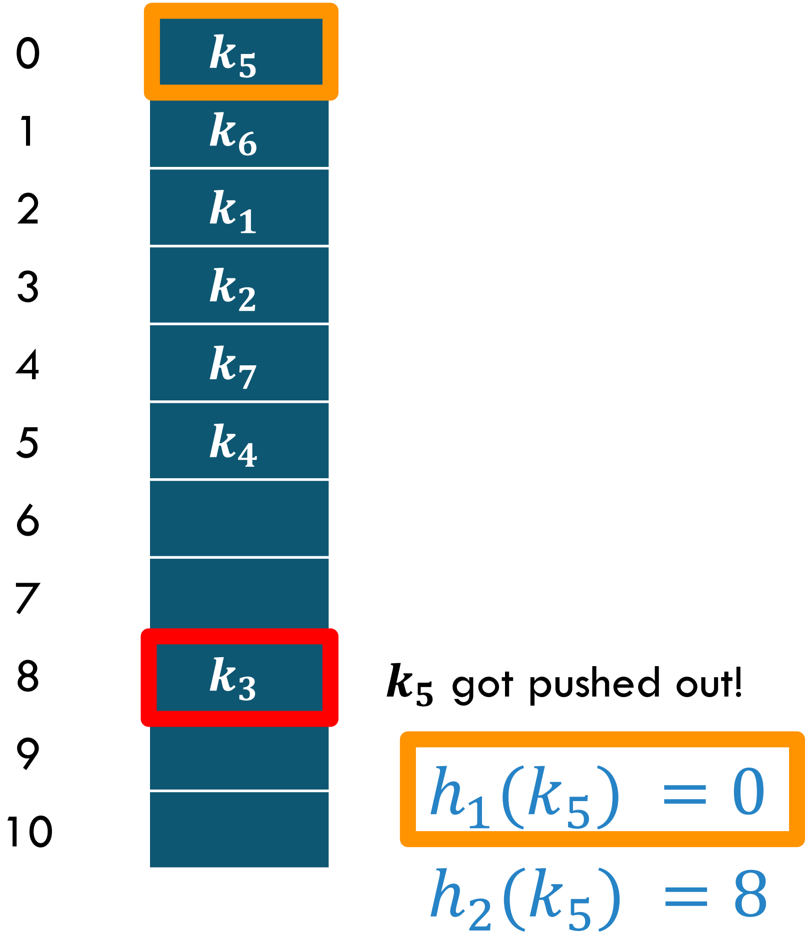
CUCKOO HASHING: INSERTIONS

insert(k_7)

Idea: When inserting, if bucket is taken, push existing key out!

Steps:

- Set $p = h_1(k)$
- Repeat n times
  - If $A[p]$ is empty then
    - $A[p] = k$
    - return
  - $t = A[p]; A[p] = k; k = t$
  - If p $= h_1(k)$ then p $= h_2(k)$ else p $= h_1(k)$
- rehash(); insert(k)



CUCKOO HASHING: PERFORMANCE

Insertions seem to be quite complicated…

But: it takes expected $O(1)$ amortized time!

Analysis requires (a little) graph theory

To be continued in Week 12 … Today!

RECALL: CHAINING

Collision.

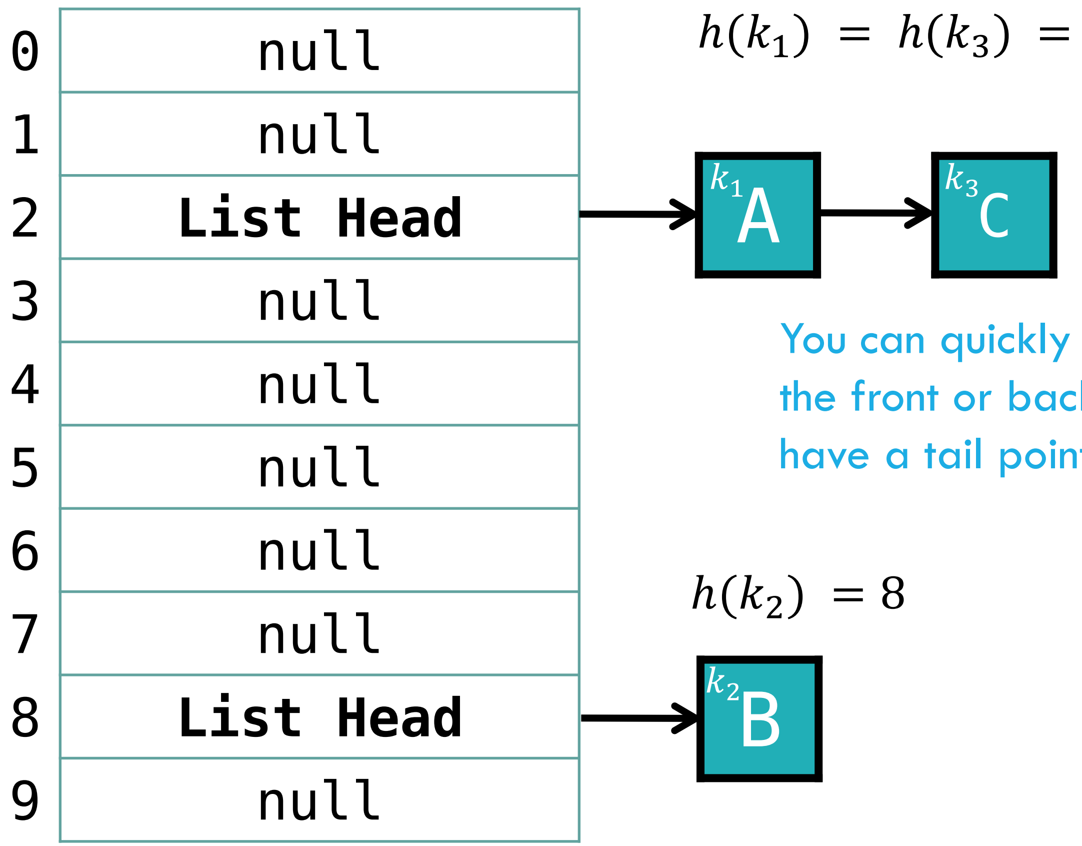
but it's ok!

Idea: Each bucket stores a linked list.

If there is a collision, we add the item to the linked list.

insert(k_1, A)
insert(k_2, B)
insert(k_3, C)

$h(k_1) = h(k_3) = 2$



You can quickly add k_3 to the front or back (if you have a tail pointer)

$h(k_2) = 8$

WHAT IS THE AVERAGE SEARCH TIME…

under the simple uniform hashing assumption (SUHA)

We have:

- m buckets
- n items
- Assume $n = \alpha m$ and $m \geq n$
- α is the "load factor"

Expected search time $= \underbrace{1}_{\text{hashing + array access}} + \underbrace{\text{expected \# items per bucket}}_{\text{linked list traversal}}$

Proof Sketch:

Indicator random variables

$X(i,j) = 1$ if item i is in bucket j
$X(i,j) = 0$ otherwise

Expected number of items in bucket b:

$$\mathbb{E}\left[\sum_i^n X(i,b)\right] = \sum_i^n \mathbb{E}[X(i,b)]$$

$$= \sum_i \frac{1}{m} = \frac{n}{m} = \alpha$$

Since $m > n$

$$\mathbb{E}\left[\sum_i X(i,b)\right] = O(1)$$

INDICATOR RANDOM VARIABLES

Indicator random variable maps every <u>**outcome**</u> to either 0 or 1.

For example: whether you have the disease

$$\Omega = \{(d,\oplus),(\neg d,\oplus),(d,\ominus),(\neg d,\ominus)\}$$

$$X(\omega) = \begin{cases} 1 \text{ if } (d,\oplus) \\ 1 \text{ if } (d,\ominus) \\ 0 \text{ if } (\neg d,\oplus) \\ 0 \text{ if } (\neg d,\ominus) \end{cases}$$

$X = I_\alpha(\omega)$ where $\alpha \in E$ is the event where you have the disease

PROBABILITY: EXPECTATION

The expected or average value of some function $f[x]$ taking into account the distribution of X.

Definition:

$$E\big[f[x]\big] = \sum_{x} f[x]p(x)$$

PROBABILITY: RULES OF EXPECTATION

Rule 1: Expected value of a constant is the constant.

$$E[\kappa] = \kappa$$

Rule 2: Expected value of constant times function is constant times expected value of function.

$$E\left[\kappa f[x]\right] = \kappa E[f[x]]$$

PROBABILITY: RULES OF EXPECTATION

Rule 3: Expectation of sum of functions is sum of expectation of functions.

$$E\big[f[x] + g[x]\big] = E\big[f[x]\big] + E[g[x]]$$

Rule 4: Expectation of product of functions in variables X and Y is product of expectations of functions if X and Y are independent.

$$E\big[f[x]g[y]\big] = E\big[f[x]\big]E[g[y]],$$

if X and Y are independent

WHAT IS THE AVERAGE SEARCH TIME...

under the simple uniform hashing assumption (SUHA)

We have:

- m buckets
- n items
- Assume $n = \alpha m$ and $m \geq n$
- α is the "load factor"

Expected search time $= \underbrace{1}_{\text{hashing + array access}} + \underbrace{\text{expected \# items per bucket}}_{\text{linked list traversal}}$

Proof Sketch:

Indicator random variables

$X(i,j) = 1$ if item i is in bucket j
$X(i,j) = 0$ otherwise

Expected number of items in bucket b:

$$\mathbb{E}\left[\sum_{i}^{n} X(i,b)\right] = \sum_{i}^{n} \mathbb{E}[X(i,b)]$$

$$= \sum_i \frac{1}{m} = \frac{n}{m} = \alpha$$

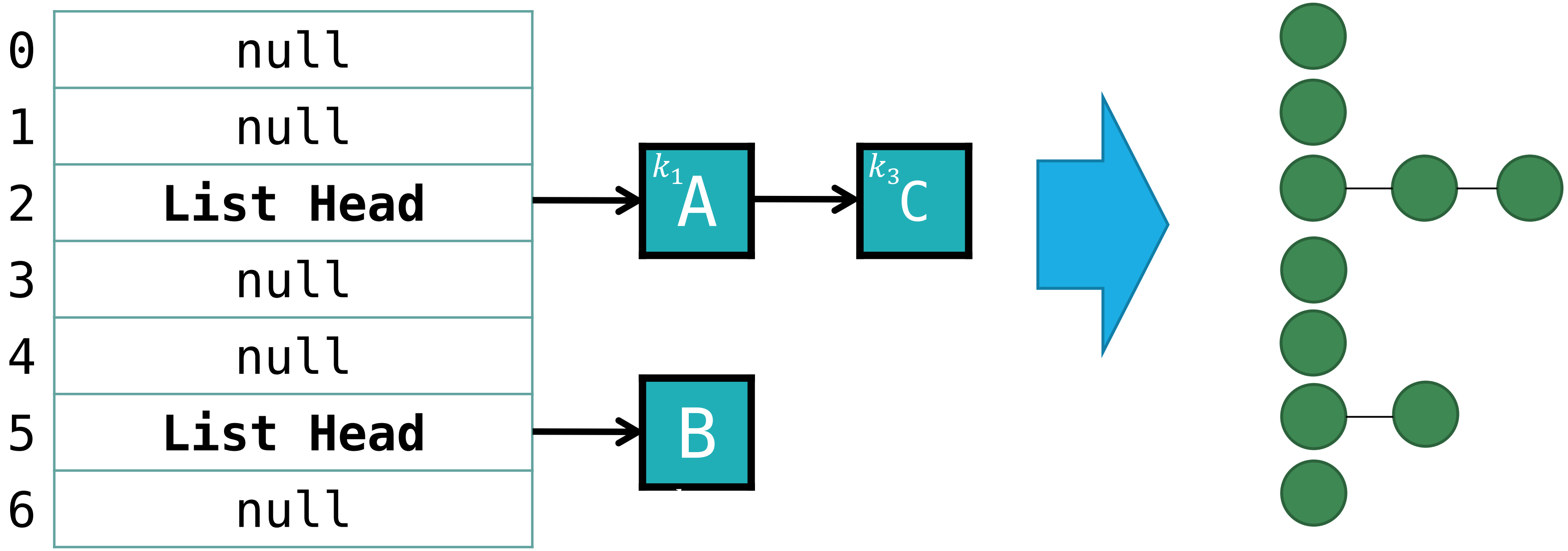
Since $m > n$

$$\mathbb{E}\left[\sum_i X(i,b)\right] = O(1)$$

A SEPARATE CHAINING GRAPH

Forest: A collection of trees

an undirected graph where two nodes are connected by *at most* one path.



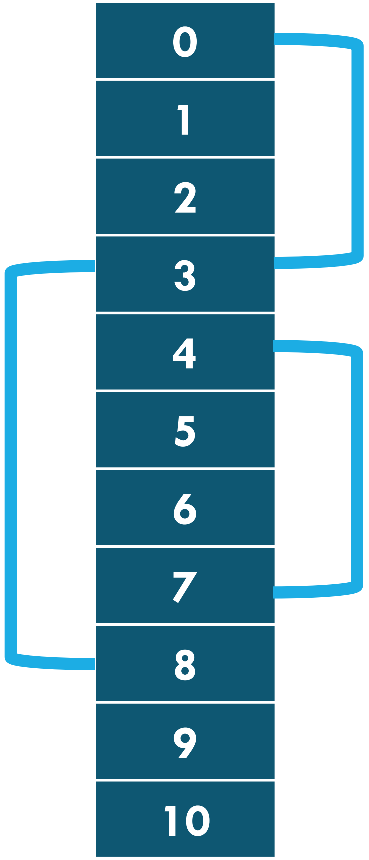
THE CUCKOO GRAPH

Given: a set S of items (keys)

Create a **cuckoo graph** where:

- Every cell is a node (m cells)
- Each key $x \in S$ connects the two cells specified by $h_1(x)$ and $h_2(x)$

THE CUCKOO GRAPH



Given: a set S of items (keys)

Create a **cuckoo graph** where:

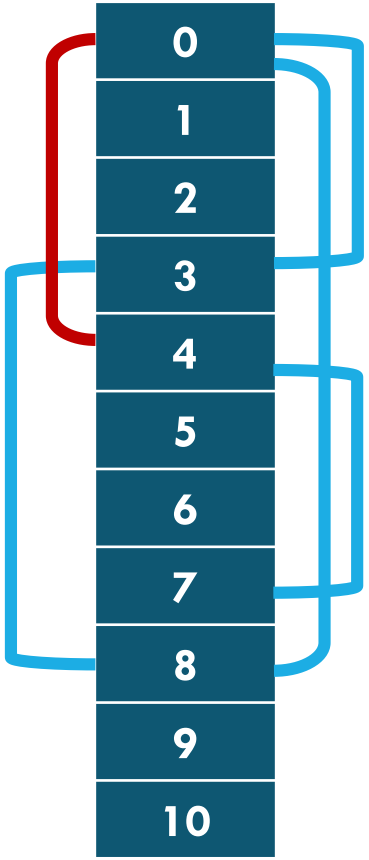
- Every cell is a node (m cells)
- Each key $x \in S$ connects the two cells specified by $h_1(x)$ and $h_2(x)$
- Undirected graph

Example:

Given $S = \{x_1, x_2, x_3\}$

Key	$h_1(x)$	$h_2(x)$
x_1	0	3
x_2	3	8
x_3	4	7

BUCKETS IN THE CUCKOO GRAPH



When inserting x:

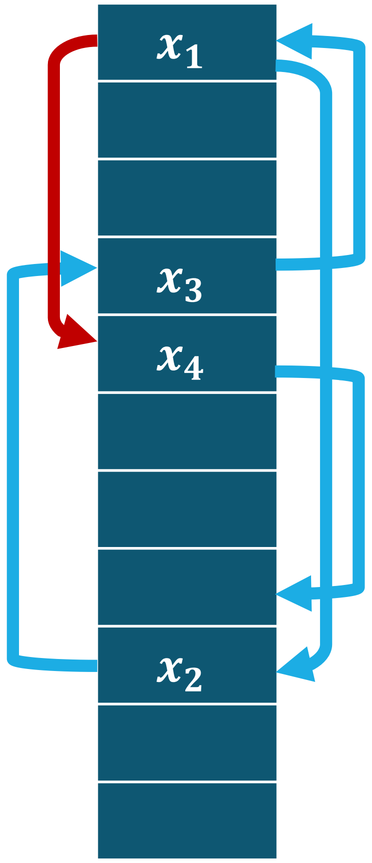
Insertions will only visit positions where there is a path from either $h_1(x)$ or $h_2(x)$

Bucket = {positions visited}

Example:

$h_1(x_5) = 0, h_2(x_5) = 4$

BUCKETS IN THE CUCKOO GRAPH



When inserting x:

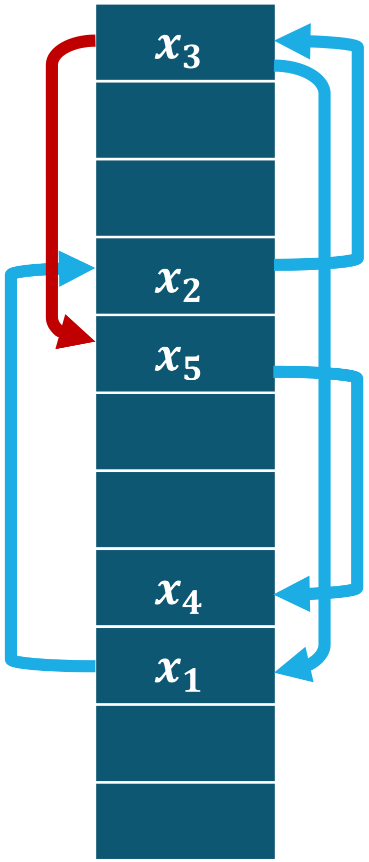
Insertions will only visit positions where there is a path from either $h_1(x)$ or $h_2(x)$

Bucket = {positions visited}

Example:

$h_1(x_5) \ = 0, h_2(x_5) = 4$

BUCKETS IN THE CUCKOO GRAPH



When inserting x:

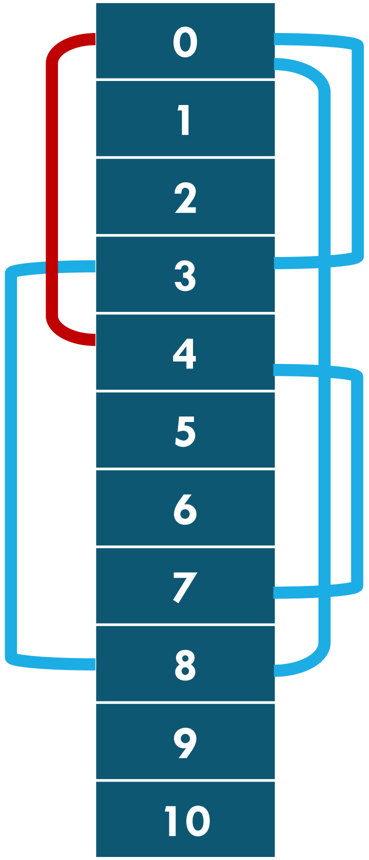
Insertions will only visit positions where there is a path from either $h_1(x)$ or $h_2(x)$

Bucket = {positions visited}

Example:

$h_1(x_5) = 0, h_2(x_5) = 4$

BUCKETS IN THE CUCKOO GRAPH



When inserting x:

Insertions will only visit positions where there is a path from either $h_1(x)$ or $h_2(x)$

Bucket = {positions visited}

Example:

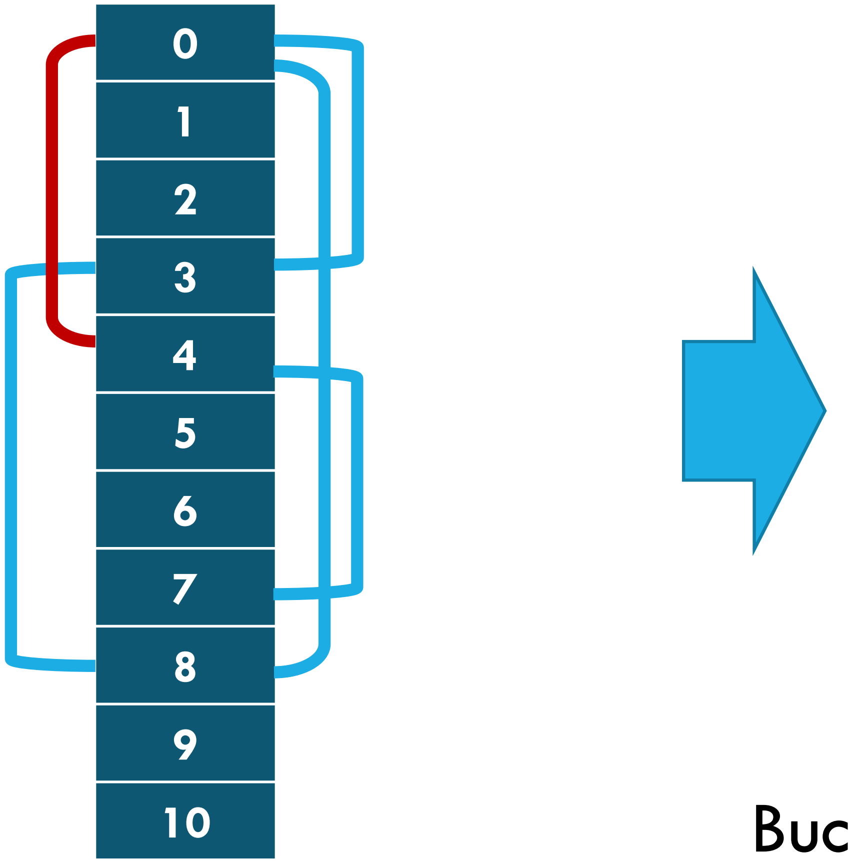
$h_1(x_5) = 0, h_2(x_5) = 4$

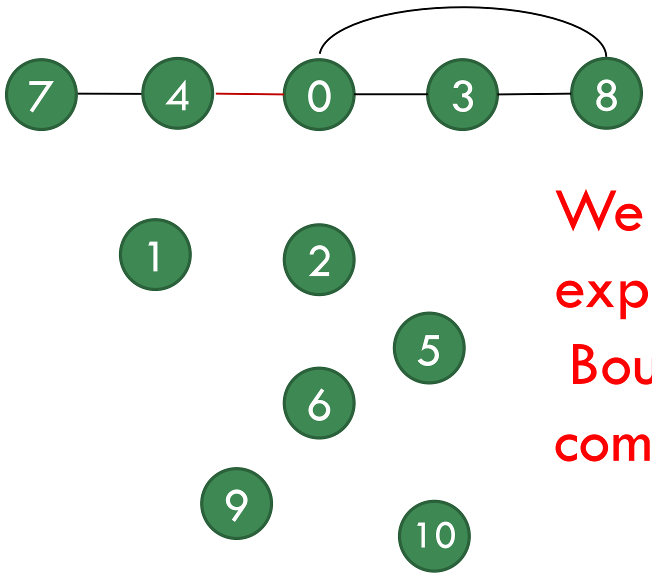
Bucket: $B(x_5) = \{0,3,8,4,7\}$

BUCKETS IN THE CUCKOO GRAPH

Insertion succeeds when the induced cuckoo graph is a PseudoForest

an undirected graph where every connected component has *at most* 1 cycle





We want to bound expected bucket size $\rightarrow$ Bound the expected component size.

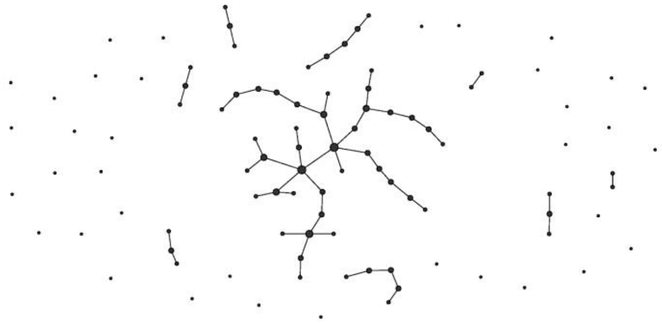
Bucket: $B(x_5) = \{0,3,8,4,7\}$

RANDOM GRAPHS

The combination of a table and random set of items produces a **random graph**.

Specifically a Erdős-Rényi Graph

- Fixed number of m nodes
- Nodes are connected with some random probability p

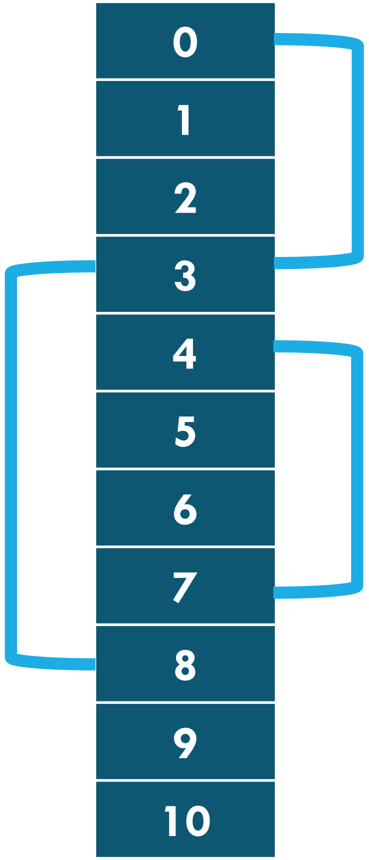


https://en.wikipedia.org/wiki/Erdős–Rényi_model

Paul Erdős
1913-1996

"a mathematician is a machine for turning coffee into theorems"

THE PROB. OF PATHS OF A CERTAIN LENGTH



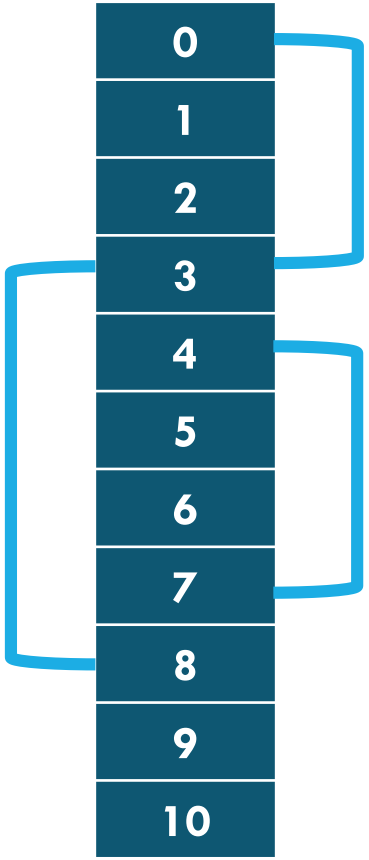
y is in the bucket of x if:

- there is a path between $\{h\_1(x), h\_2(x)\}$ and the position of y in the graph.

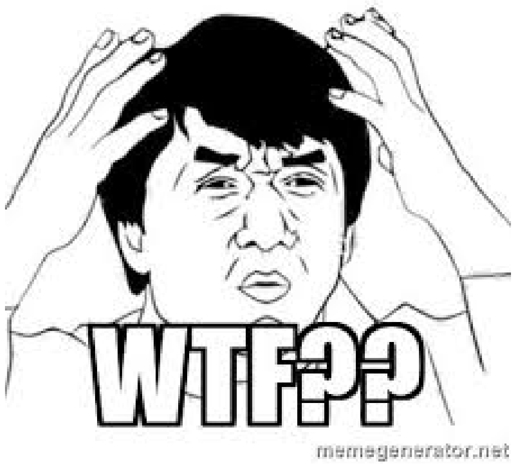
Want: the probability of a path between two locations i and j

Lemma 1: For any i, j and $c > 1$,
if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

THE PROB. OF PATHS OF A CERTAIN LENGTH

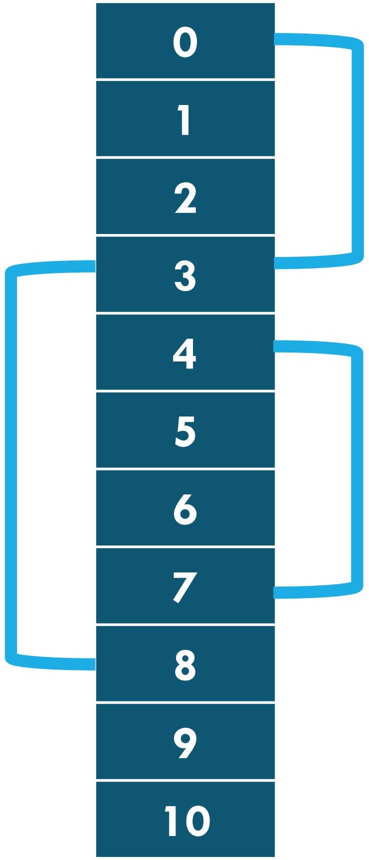


Lemma 1: For any i, j and $c > 1$,
if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$



what does this mean?!

THE PROB. OF PATHS OF A CERTAIN LENGTH

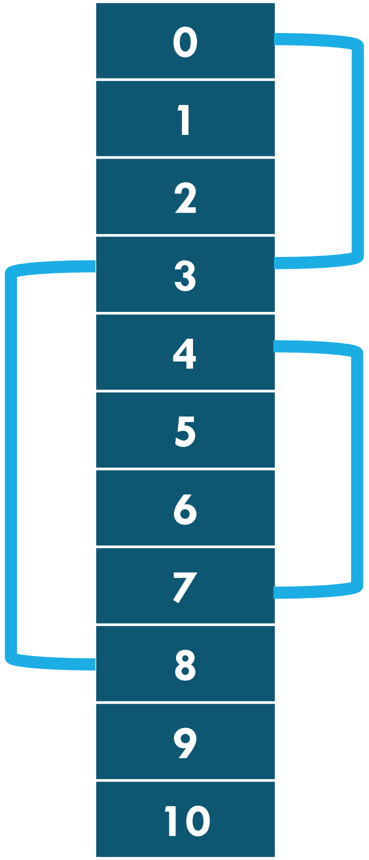


Lemma 1: For any i, j and $c > 1$, if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

Intuition: "For any i, j and $c > 1$, if $m \geq 2cn$ …"

- m is the number of cells (nodes)
- n is the number of items (edges)
- $m > 2cn$ means **low load factor** α!
    - If $c = 2$ then $\alpha = 0.25$

THE PROB. OF PATHS OF A CERTAIN LENGTH



Lemma 1: For any i, j and $c > 1$,
if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

Intuition: if low load factor:

- $p(i$ and j connected by a path) is *small.*
- $p(i$ and j connected by a path of length $l) = O(1/m)$
- The probability the shortest path of length l exists decreases *exponentially* in l

If p(any two nodes are connected) is small, the size of components (buckets) should also be small.

PROOF SKETCH (BY INDUCTION)

Lemma 1: For any i, j and $c > 1$, if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

Proof Sketch:

Induction on l (length of path)

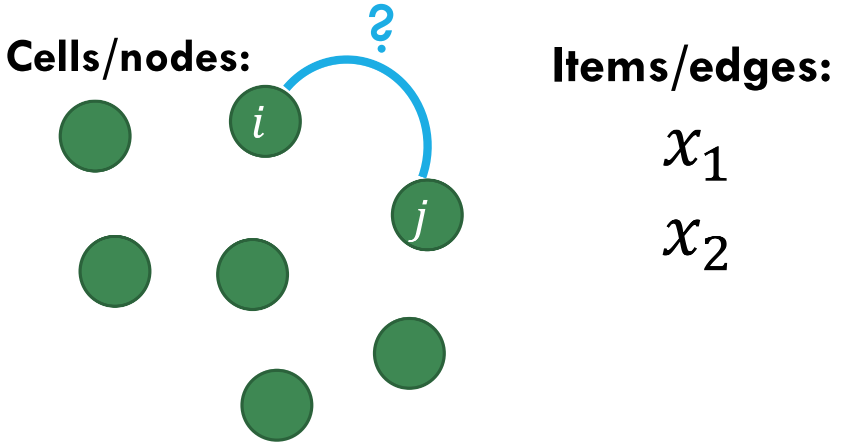
Lemma 1: For any i, j and $c > 1$, if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

PROOF SKETCH (BY INDUCTION)

Base case: $l = 1$

Edge exists if key x has:

- $h_1(x) = i$ and $h_2(x) = j$ **OR**
- $h_1(x) = j$ and $h_2(x) = i$
- So, prob. of edge between i & j at most $\frac{2}{m^2}$

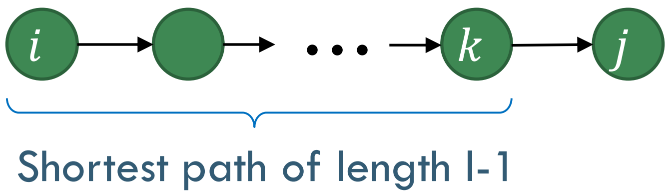


$$p(i, j \text{ connected by path } l = 1) \leq \sum_{x \in S} \frac{2}{m^2} = \frac{2n}{m^2} = \frac{c^{-1}}{m}$$

Lemma 1: For any i, j and $c > 1$, if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

PROOF SKETCH (BY INDUCTION)

Cells/nodes:



Inductive hypothesis:

- Lemma holds for length $l - 1$

Inductive step:

Goal: Bound probability there is a shortest path between i and j of length l:

- **Event A:** there exists a shortest path of length $l - 1$ from i to k that does not go through j (probability $\leq \frac{c^{-(l-1)}}{m}$ by inductive hypothesis)
- **Event B:** There exists an edge from k to j (probability given first condition is true?)
- Want $p(A \text{ and } B) = p(B|A)p(A)$

FROM YESTERDAY: SUM AND PRODUCT RULES

Sum rule:

$$p(x) = \sum_y p(x, y)$$

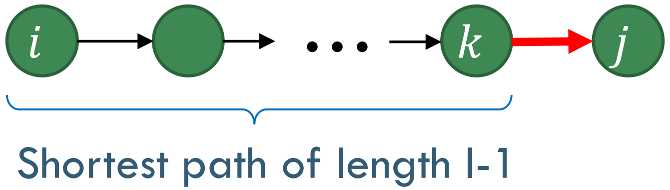
Product/Chain rule:

$$p(x, y) = p(x|y)p(y)$$

Lemma 1: For any i, j and $c > 1$, if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

PROOF SKETCH (BY INDUCTION)

Cells/nodes:



Given a shortest path from $i \to k$,

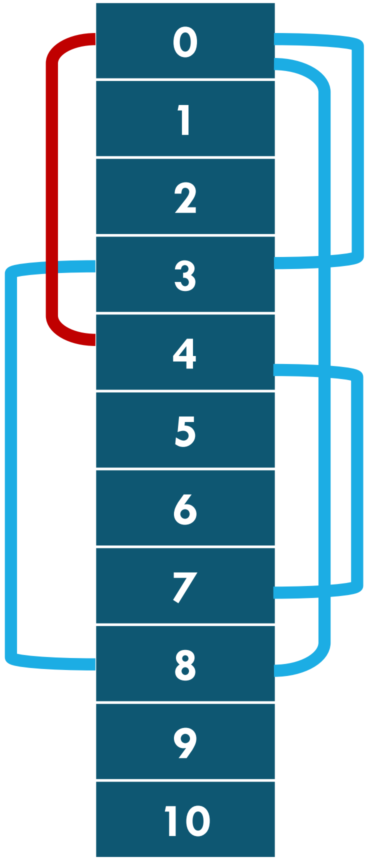
Probability there exists an edge from k to j?

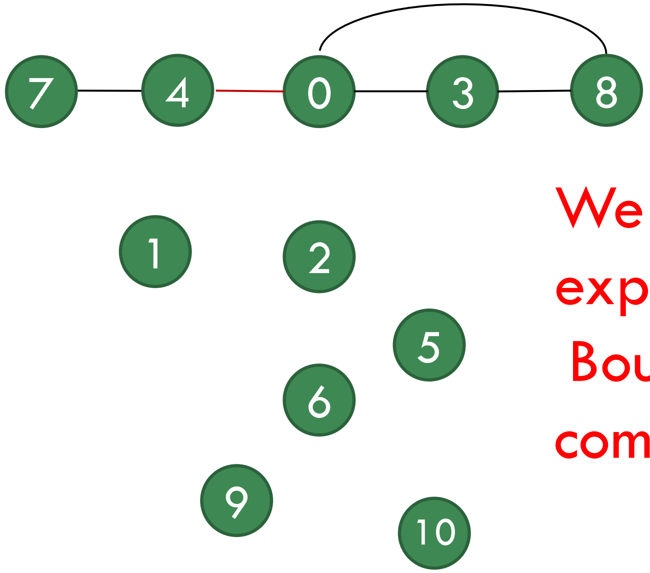
- Recall: Edge exists if key has:
  - $h_1(x) = k$ and $h_2(x) = j$ OR
  - $h_1(x) = j$ and $h_2(x) = k$
  - So, probability $\frac{2}{m^2}$ per key
- So, for $n - l$ keys, probability is $\leq \frac{2(n-l)}{m^2} \leq \frac{c^{-1}}{m}$

Putting both together:

$$\sum_k p(A \text{ and } B) = \sum_k p(B|A)p(A) \leq \sum_k \frac{c^{-1}}{m} \times \frac{c^{-(l-1)}}{m} = \sum_k \frac{c^{-l}}{m^2} \leq \frac{c^{-l}}{m} \blacksquare$$

SUMMARY SO FAR

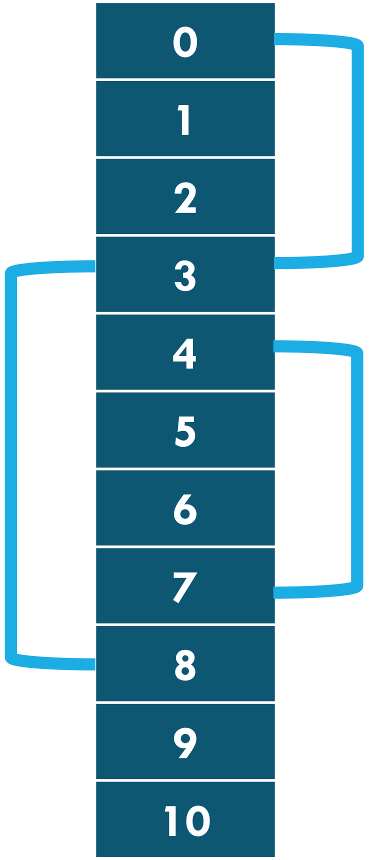




We want to bound expected bucket size → Bound the expected component size.

Bucket: $B(x_5) = \{0,3,8,4,7\}$

SUMMARY SO FAR:



y is in the bucket of x if:

- there is a path between $\{h\_1(x), h\_2(x)\}$ and the position of y in the graph.

Want: the probability of a path between two locations i and j

Lemma 1: For any i, j and $c > 1$,
if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

Lemma 1: For any i, j and $c > 1$, if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

BOUNDING THE BUCKET SIZE

$$\mathbb{E}[|B(x)|] = \sum_{y \in S} p(y \text{ is in bucket } B(x))$$

Lemma 1: For any i, j and $c > 1$, if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

BOUNDING THE BUCKET SIZE

$$\mathbb{E}[|B(x)|] = \sum_{y \in S} p(y \text{ is in bucket } B(x))$$

$$= \sum_{y \in S} p(\mathrm{e}_y \text{ is in bucket } B(x)) = np\left(\mathrm{e}_y \text{ is in bucket } B(x)\right)$$

Lemma 1: For any i, j and $c > 1$, if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

BOUNDING THE BUCKET SIZE

$$\mathbb{E}[|B(x)|] = \sum_{y \in S} p(y \text{ is in bucket } B(x))$$

$$= \sum_{y \in S} p(\mathrm{e}_y \text{ is in bucket } B(x)) = np\left(\mathrm{e}_y \text{ is in bucket } B(x)\right)$$

$$\leq n \sum_{l=1}^{\infty} \frac{c^{-l}}{m} = \frac{n}{m}\,\hat{c} = O(1) \;\blacksquare$$

(Since $m > n$)

Lemma 1: For any i, j and $c > 1$, if $m \geq 2cn$ then the probability that there exists a *shortest* path in the cuckoo graph from i to j of length l, is at most $\frac{c^{-l}}{m}$

BOUNDING THE BUCKET SIZE

$$\mathbb{E}[|B(x)|] = \sum_{y \in S} p(y \text{ is in bucket } B(x))$$

$$= \sum_{y \in S} p(\mathrm{e}_y \text{ is in bucket } B(x)) = np\left(\mathrm{e}_y \text{ is in bucket } B(x)\right)$$

$$\leq n \sum_{l=1}^{\infty} \frac{c^{-l}}{m} = \frac{n}{m} \hat{c} = O(1) \blacksquare$$

(Since $m > n$)

Insertions take O(1) time on average!

CUCKOO HASHING: SUMMARY

Use 2 hash functions $h_1(k)$ and $h_2(k)$

So, each key only has **2 possible locations**

Operations:

- **Lookup:** only check the 2 possible locations
  - $O(1)$ worst-case time!
- **Deletion:** only check the 2 possible locations and delete accordingly.
  - Also $O(1)$ worst-case time
- **Insertion:**
  - $O(1)$ average, amortized

0	
1	
2	
3	k_2
4	k_3
5	k_4
6	
7	
8	k_5
9	
10	

FURTHER EXPLORATION

Rehashing:

- What happens when we need to rehash
- Amortized cost is $O(1)$

In practice:

- Nice worst-case lookups!
- Can be slower than linear probing

Further reading:

- http://www.it-c.dk/people/pagh/papers/cuckoo-undergrad.pdf
- https://web.stanford.edu/class/cs166/lectures/13/Small13.pdf

Poll Everywhere

https://bit.ly/2LvG9bq

QUESTIONS?

NEXT WEEK

Lecture on Tuesday

- Final Review + Final Exam ”tips”

No lecture on Wednesday

- Open Office

See you next week!