Hashing, Episode 2:

Multiple-choice hashing

Rasmus Pagh
IT University of Copenhagen

MADALGO SUMMER SCHOOL ON DATA STRUCTURES 2013

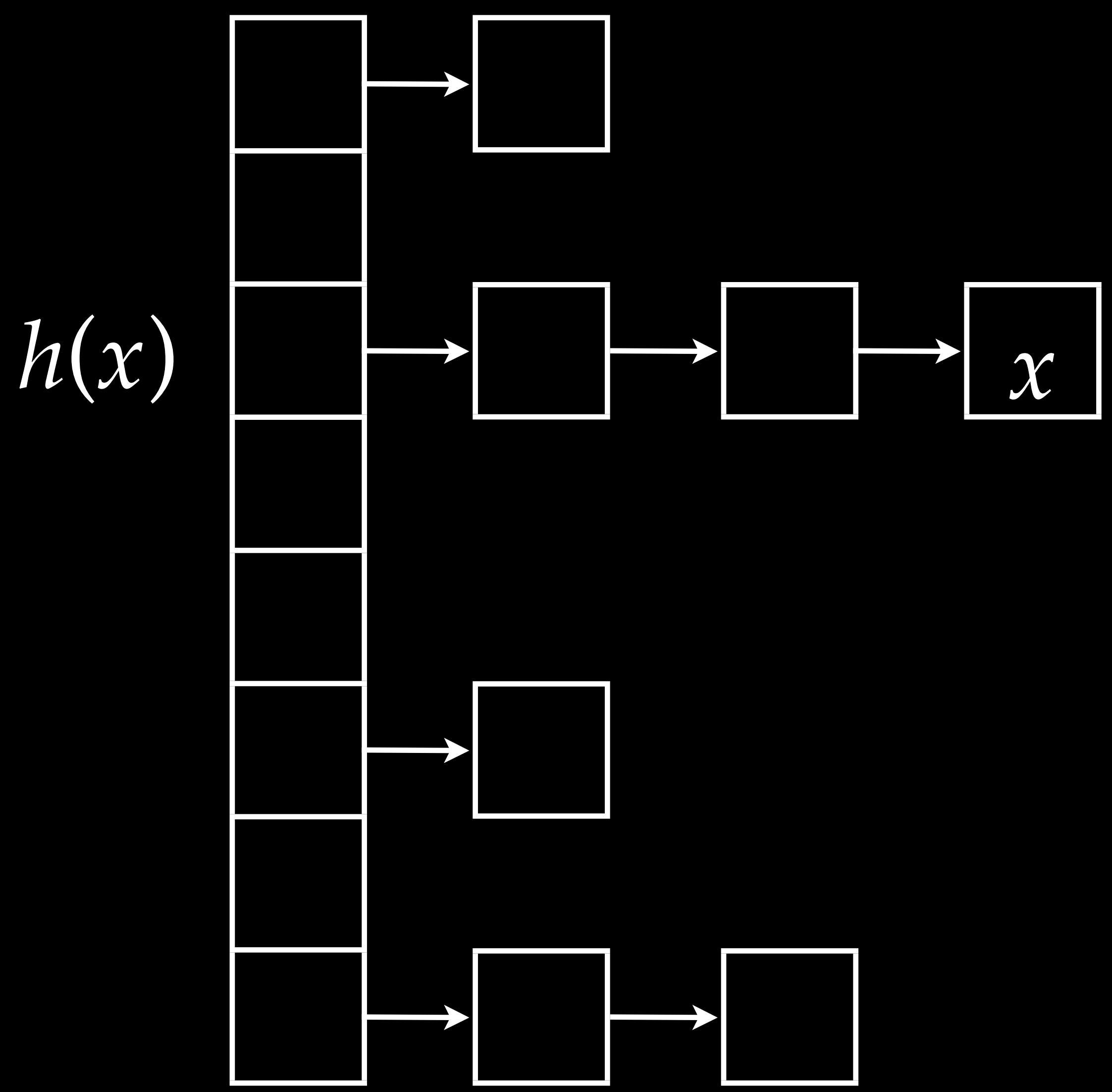
Today: Choice is good!

- In which hash table do you feel most welcome?
- Peeling leaves from trees in random graphs.
- Hash tables without keys.

Topics

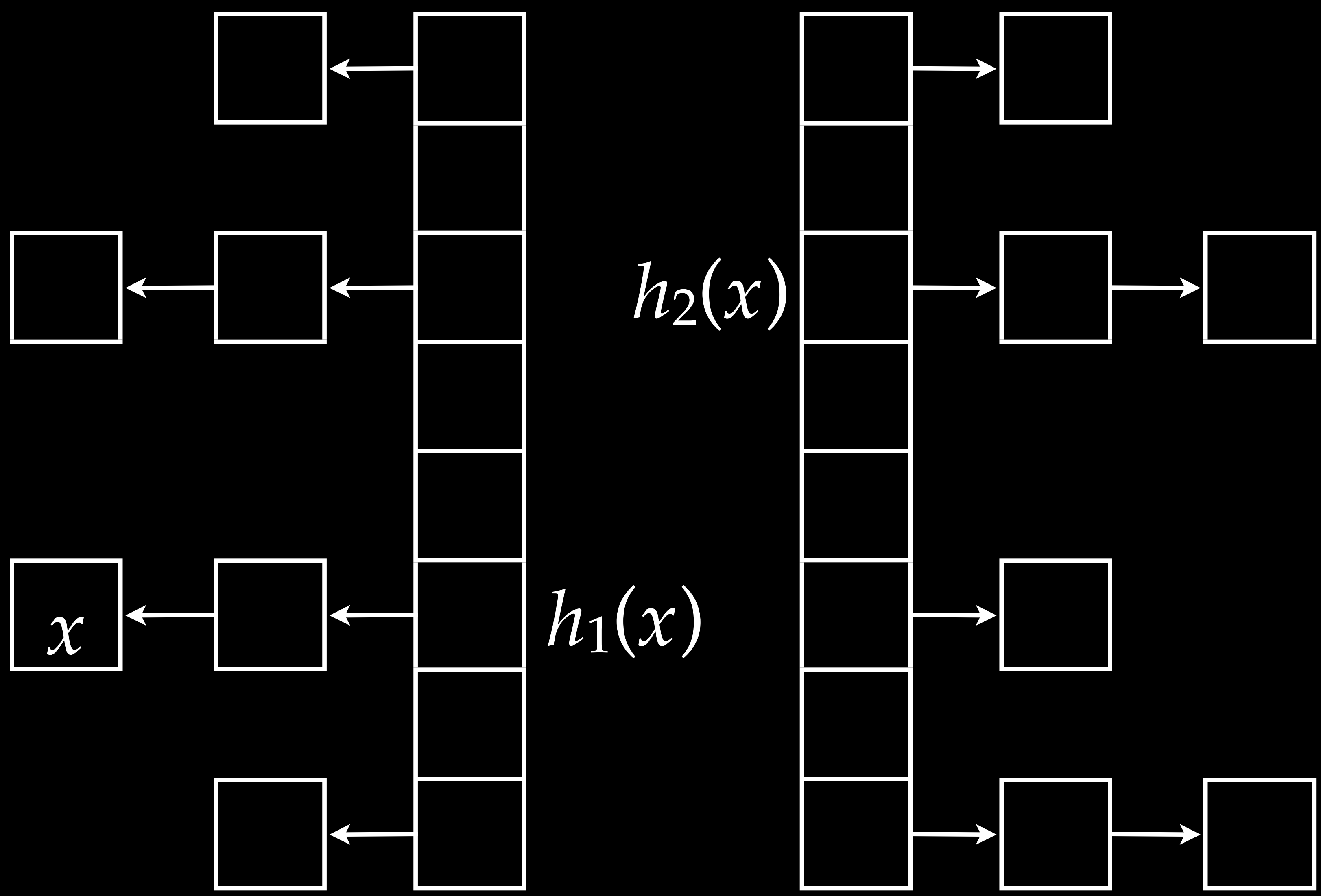
- Two-choice hashing.
- Cuckoo hashing.
- Storing (key,value) pairs without storing any keys.

Chained hashing



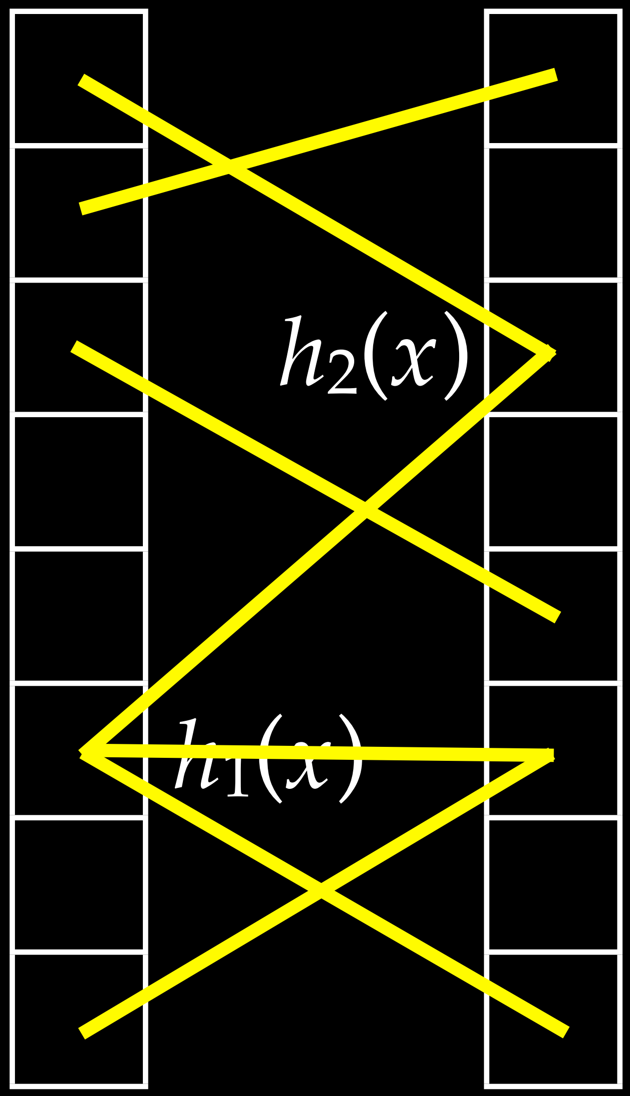
Max search time ~ $\log(n)/\log\log(n)$

Two-choice chaining



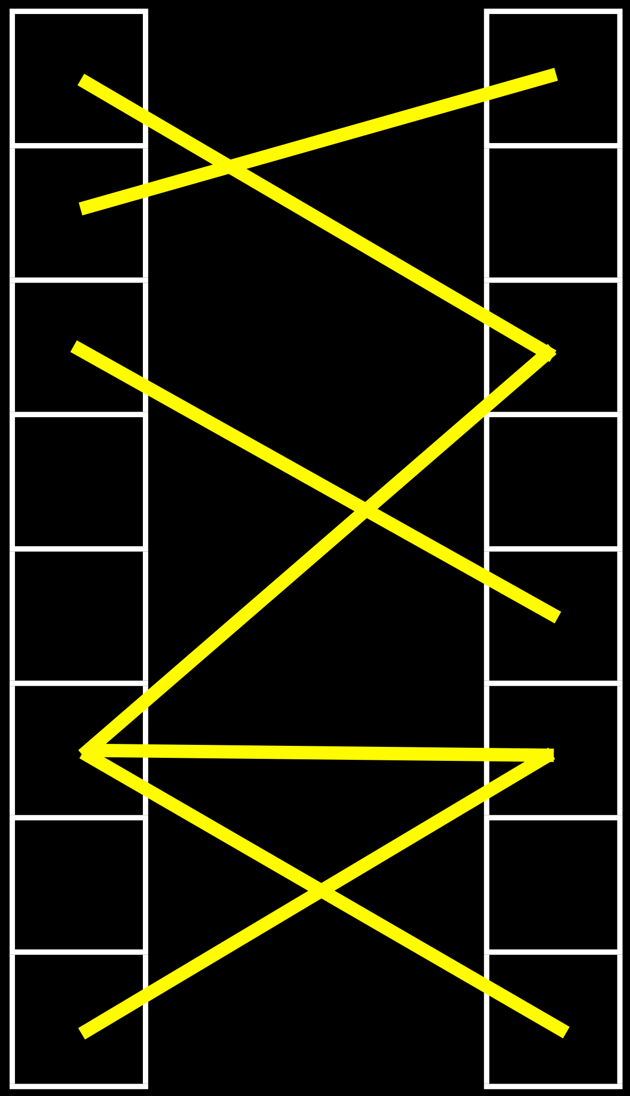
Max search time ~ log log(n) !

Choice graph



$$E = \{\{h_1(x), h_2(x)\} \mid x \in S\}$$

Choice graph



What is the highest load we could possibly get?

Choice graph

$$E = \{\{h_1(x), h_2(x)\} \mid x \in S\}$$

Lemma. If E is acyclic, the maximum bucket size in a connected component $C \subseteq E$ is bounded by $\lceil \log_2(|C| + 1) \rceil$.

How large connected components do we expect?
Are all components (close to) being trees?

Random graph theory

Assuming highly random hash functions

- **Some answers:**
  - Largest component *either* has O(log n) ***or*** $\theta(n)$ edges, whp. ($n = |E|$)
  - Depends on whether n/r exceeds a certain *threshold*.
  - Below the threshold components are all pseudotrees (trees + 1 edge) whp., and have average constant size.

Below the threshold

- We argue that $r > 2.1n$ suffices.
- 1st step: Connected components are small.

Lemma. The expected number of vertices in the choice graph at distance ℓ from a given vertex u is at most $(2n/r)^{\ell}$.

Consequences of lemma

- With high probability, all components have size O(log n).
- The average component size is O(1).

Lemma. For $r > 2.1n$ the probability that a connected component of the choice graph with t vertices contains more than t edges is $O(t^4/r^2)$.

More balls, more tables?

- What if we throw n balls in $b < n$ bins?
  - **Answer**: Max. load n/b+O(log log n).
- What if we use $d > 2$ hash functions?
  - **Answer**: Depending on tie-breaking rule max. load O($\log_d$ log n) or O((log log n)/d)

Choice graph becomes hypergraph

Many more tables?

- Curiosity:
  If $S \subseteq U$ and we use $O(\log |U|)$ hash fct., the error probability can be made *zero*!
- Caveat:
  No known good construction of the deterministic hash functions (= an unbalanced expander).

Cuckoo hashing

- Answer to a natural question:
  "Can we reduce maximum query time by moving keys between the tables?"
- **Lemma**: If the choice graph consists of trees and pseudotrees, it is possible to place the keys with 1 key per bucket.

Cuckoo approach to getting a nest

procedure insert(x)
    pos $\leftarrow h_1(x)$;
    loop n **times** {
        if $T[\text{pos}] = \texttt{NULL}$ **then** { $T[\text{pos}] \leftarrow x$; **return**};
        $x \leftrightarrow T[\text{pos}]$;
        if pos$= h_1(x)$ **then** pos$\leftarrow h_2(x)$ **else** pos$\leftarrow h_1(x)$;}
    rehash(); insert(x)
end

Cuckoo hashing analysis

- Assume $r > 2.1n$ so lemma holds whp.
- Failure probability
  = 1 - probability of pseudotree
  = $\tilde{O}(1/r^2)$.
- Expected insertion time
  = size of connected component
  = O(1).

Generalized cuckoo hashing

- $d > 2$ hash functions - much fuller table:

d	2	3	4	5	6
α	0.500	0.917	0.976	0.992	0.997

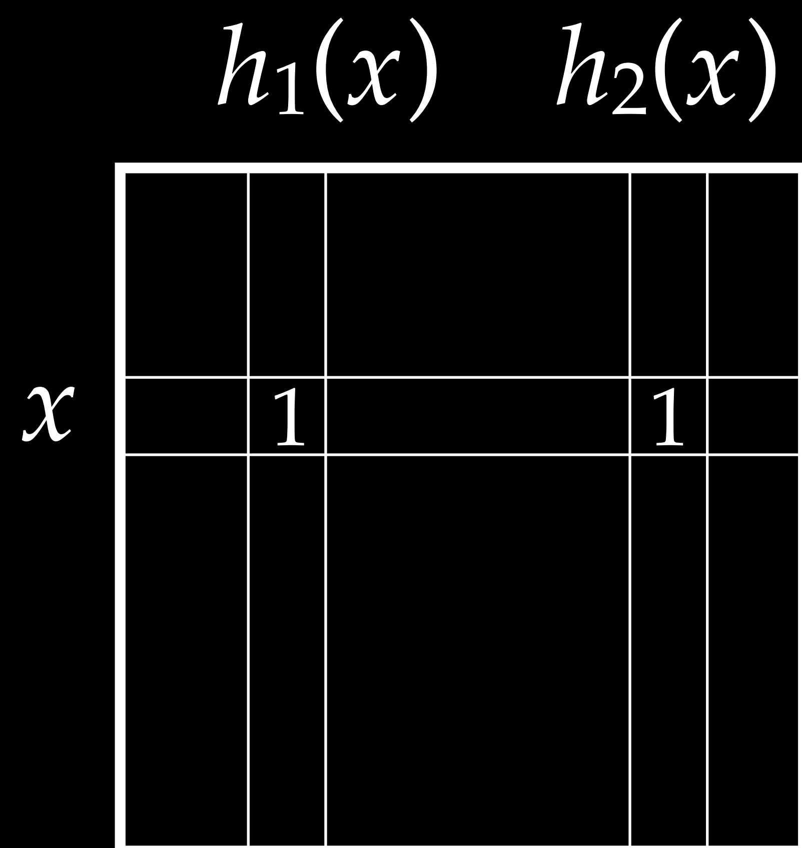
- $b > 1$ keys in each position - similar effect:

b	1	2	3	4	5	6
α	0.500	0.897	0.959	0.980	0.989	0.994

Open problem:
Efficient insertions

Choice matrix

- The choice graph as a sparse 0-1 matrix:



- Row v_x = the set of hash values of a key x. Generalizes to $k > 2$ hash functions.

Choice matrix properties

- **Lemma**: If the choice graph is acyclic, the choice matrix A has full rank (in any field).
- **Proof**: The linear system $A\boldsymbol{y}=\boldsymbol{b}$ can be solved greedily by *peeling* nodes/variables.
- Ratio r/n needed for peelability:

k	2	3	4	5	6	7
r/n	2.000	1.222	1.295	1.425	1.570	1.721

Retrieval

- **Problem:**
  Given keys $x_1,\ldots,x_n$ and values $b_1,\ldots,b_n$.
  Store a function f that such that $f(x_i)=b_i$.
- **Solution**:
  - Choose h_1,h_2 so that A has full rank.
  - Solve the linear system $A\boldsymbol{y}=\boldsymbol{b}$, store $\boldsymbol{y}$.
  - Let $f(x) = y_{h_1(x)} + y_{h_2(x)}$

Addition in *some* group
(e.g. bitwise xor, arithmetic mod p)

Retrieval - space usage

- **Need to store** (3 hash functions)**:**
  - The vector y of 1.23 n values.
  - Description of the hash functions.
- No space needed for storing $x_1,\ldots,x_n$!
- By increasing the number of hash functions, the size of y can be made arbitrarily close to n.

Application:

Approximate membership

- Let $s(x)$ be a $\log_2(1/\varepsilon)$-bit signature of x.
- Create retrieval function f with $f(x_i)=s(x_i)$.
- Return 'yes' on input x iff $f(x)=s(x)$.
- With ≥ 3 hash functions this uses less space than Bloom filters.

Some open questions

- An elementary analysis of the "heavily loaded case" of 2-choice chaining.
- Analyzing the insertion time for cuckoo hashing generalizations (several algorithms, average and worst-case bounds).
- Good unbalanced expander graphs.

Some references

- Azar et al.: Balanced Allocations
  http://www.cs.tau.ac.il/~azar/box.pdf
- Pagh and Rodler: Cuckoo Hashing
  http://www.itu.dk/people/pagh/papers/cuckoo-jour.pdf
- Dietzfelbinger and Weidling: Balanced allocation and dictionaries with tightly packed constant size bins
  http://dl.acm.org/citation.cfm?id=1244728
- Dietzfelbinger and Pagh: Succinct Data Structures for Retrieval and Approximate Membership
  http://www.itu.dk/people/pagh/papers/bloomier.pdf
- Mitzenmacher: Some open problems related to cuckoo hashing
  http://www.eecs.harvard.edu/~michaelm/postscripts/esa2009.pdf