Summer School on Hashing
Theory and Application

Dictionaries with implicit keys

Rasmus Pagh

IT UNIVERSITY OF COPENHAGEN

Supported by:

DET FRIE FORSKNINGSRÅD

erc
European Research Council
Established by the European Commission
Supporting top researchers from anywhere in the world

Agenda

- The retrieval problem - “storing a function”
- Perfect hashing - a recursive construction
  - Static
  - Use with signatures
  - Dynamic
- Retrieval without perfect hashing.
- **Exercise**: Linear data structures for storing sets

Retrieval problem

- For field F and set $S \subseteq F$, represent a function f: $S \to F$.
- Solution: Polynomial degree k-1 hash function:

$$p(x) = \sum_{i=0}^{k-1} a_i x^i$$

Retrieval problem

- For field F and set $S \subseteq F$, represent a function f: $S \rightarrow F$.
- Solution: Polynomial degree k-1 hash function:

$$p(x) = \sum_{i=0}^{k-1} a_i x^i$$

- Space $|S| \log |F|$ bits.
    - Pay only for storing values, not for keys!

Retrieval problem

- For field F and set $S \subseteq F$, represent a function f: $S \rightarrow F$.
- Solution: Polynomial degree k-1 hash function:

$$p(x) = \sum_{i=0}^{k-1} a_i x^i$$

- Space $|S| \log |F|$ bits.
  - Pay only for storing values, not for keys!

Q: Can we map to {0,1} with close to 1 bit/key?

Perfect hashing

- Given set S of size n.
  - Without loss of generality, $2 \log n$ bits/key.
- **Idea**: Carefully choose a hash function h that has:
  - No collisions among items in S, and
  - Hashes to $\{1,\ldots,r\}$, where $r = O(n)$.

Perfect hashing

- Given set S of size n.
  - Without loss of generality, $2 \log n$ bits/key.
- **Idea**: Carefully choose a hash function h that has:
  - No collisions among items in S, and
  - Hashes to $\{1,\ldots,r\}$, where $r = O(n)$.

Such functions are extremely rare!

"Less than perfect" hashing

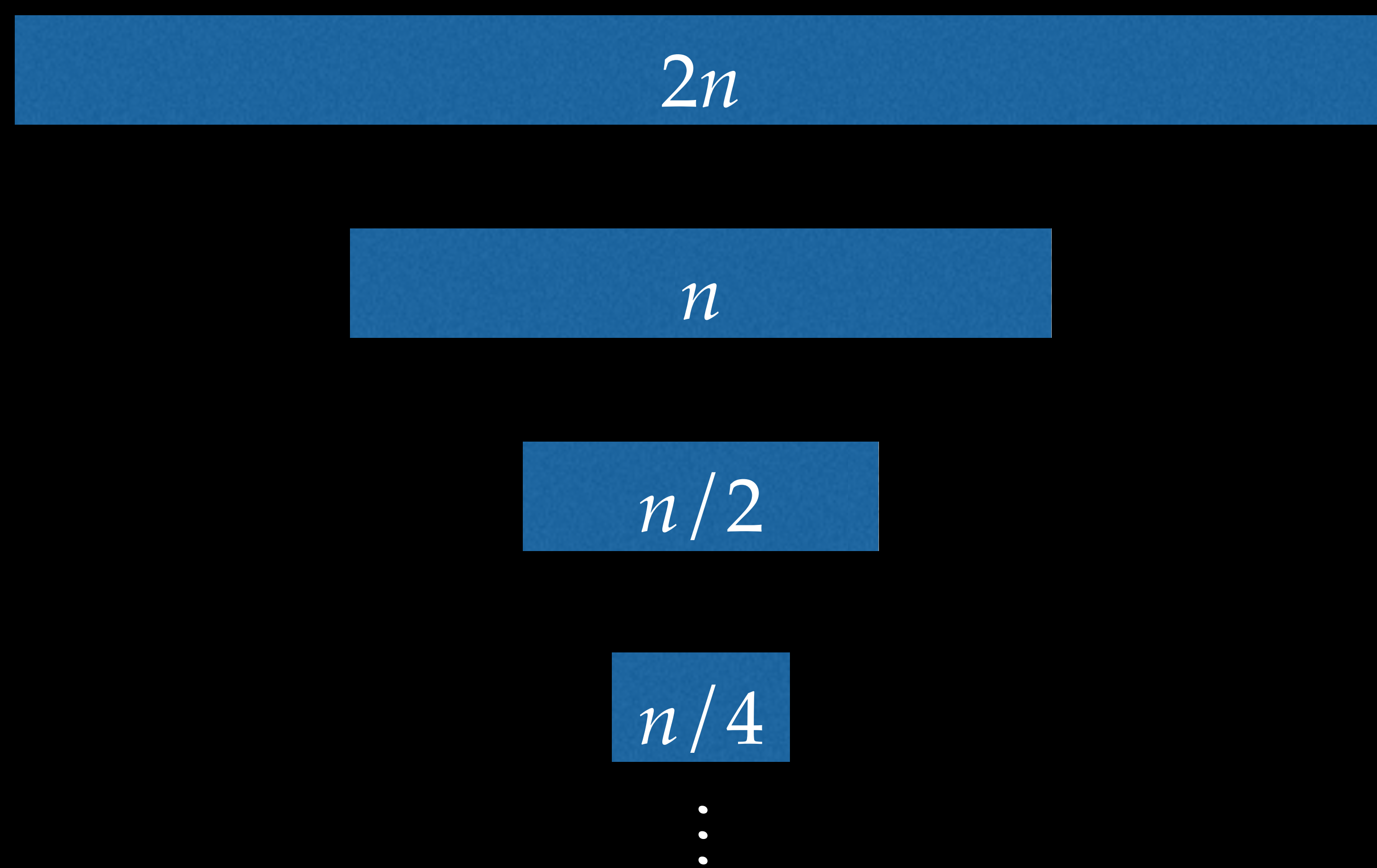
- Suppose $r = 2n$ and h is 2-independent, then the expected number of collisions is:

$$\binom{n}{2}/r < (n^2/2)/(2n) = n/4$$

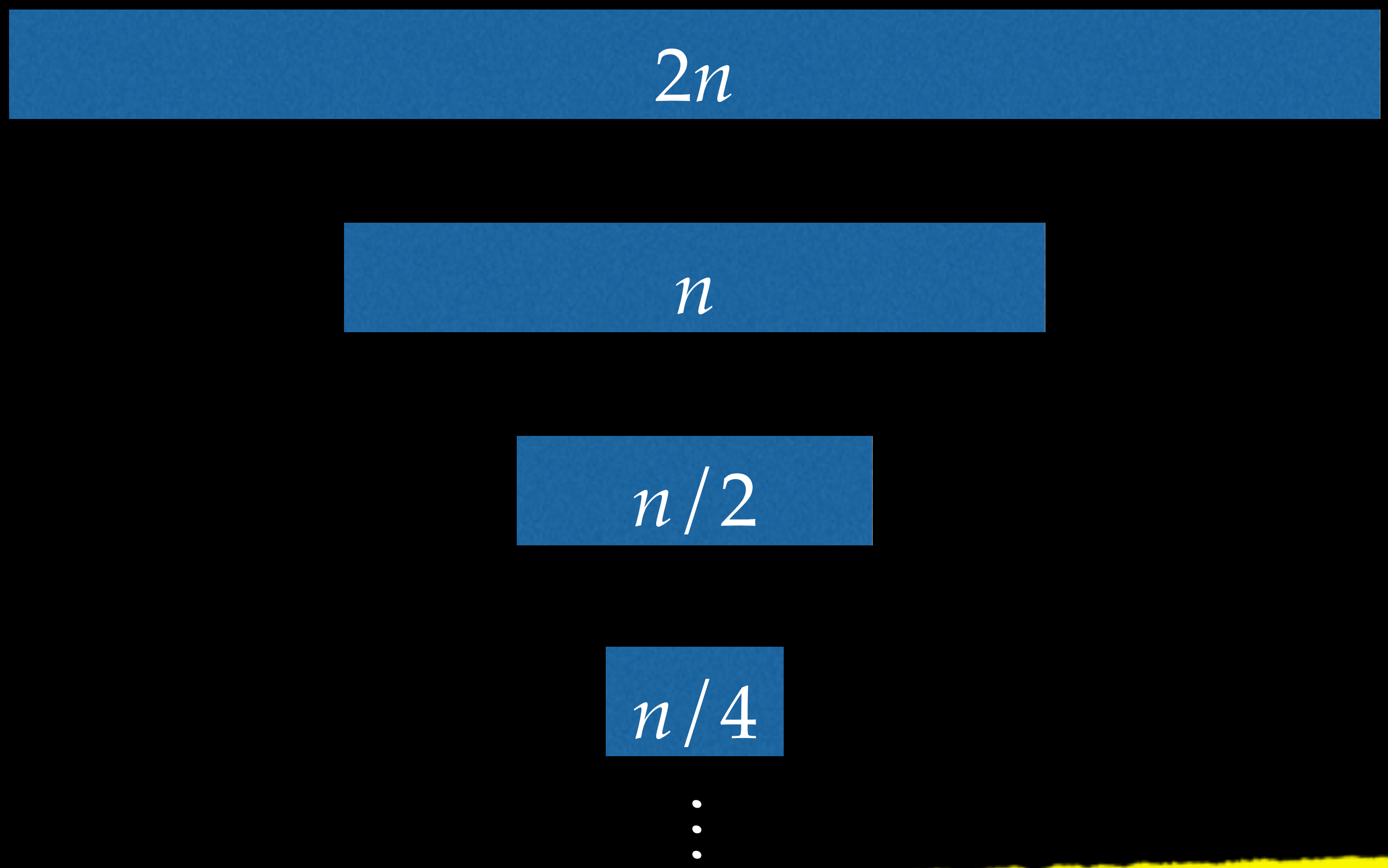
- **Conclusion**: Set $S_1 \subseteq S$ of at least $n/2$ items are not involved in a collision.

- **Idea**: Store $h(S_1)$ as a bit map, and recurse on $S \setminus S_1$.

Recursive perfect hashing



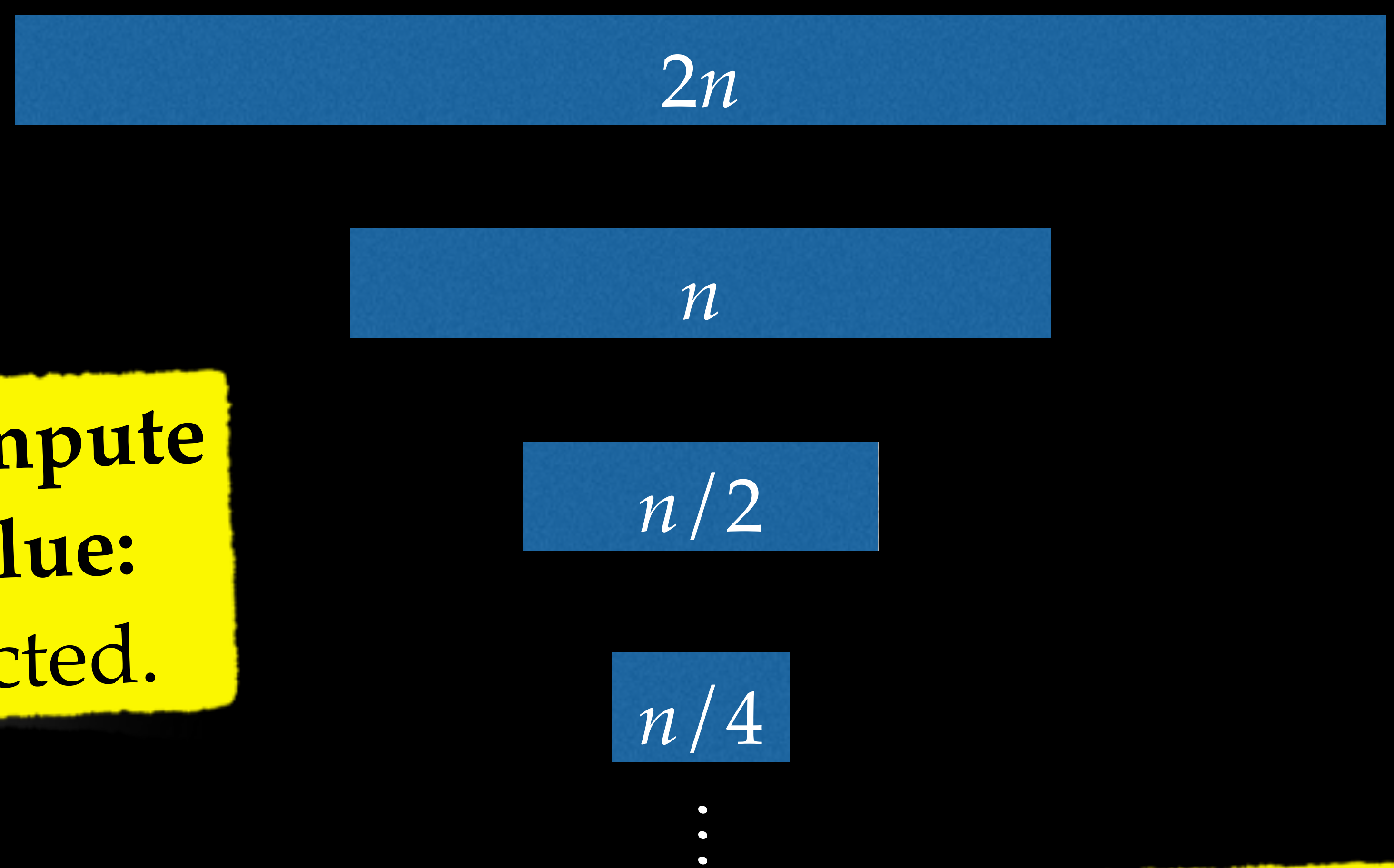
Recursive perfect hashing



Total size: $4n$ bits [improvable to en]

\# **levels**: $\log n$ [improvable to $\log \log n$]

Recursive perfect hashing



Time to compute a hash value: O(1) expected.

Total size: $4n$ bits [improvable to en]

 # levels: $\log n$ [improvable to $\log \log n$]

Recursive perfect hashing

$2n$

n

$n/2$

$n/4$

$\vdots$

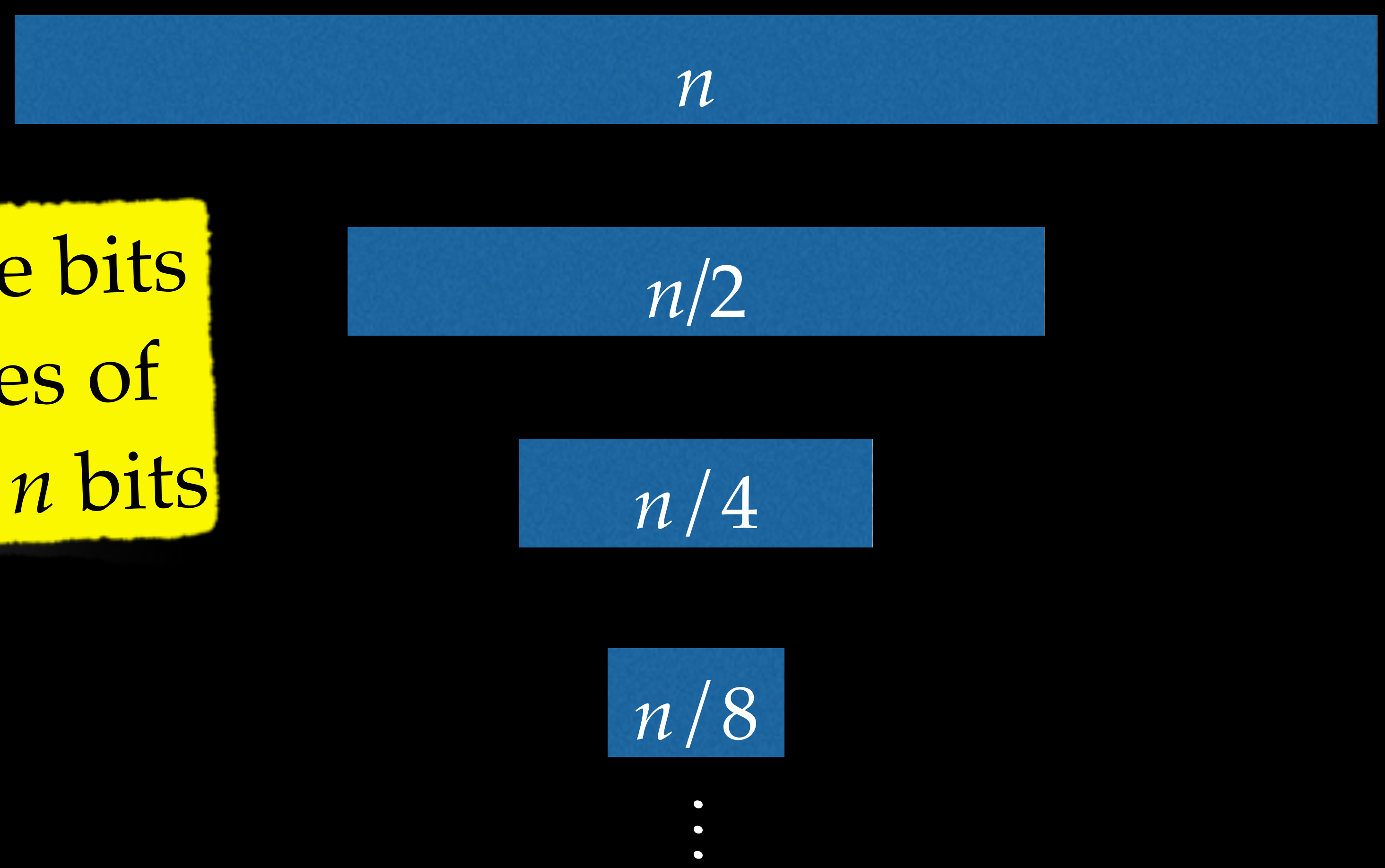
Time to compute a hash value: O(1) expected.

To map to range of size n, augment with `rank` data structure.

Total size: $4n$ bits [improvable to en]

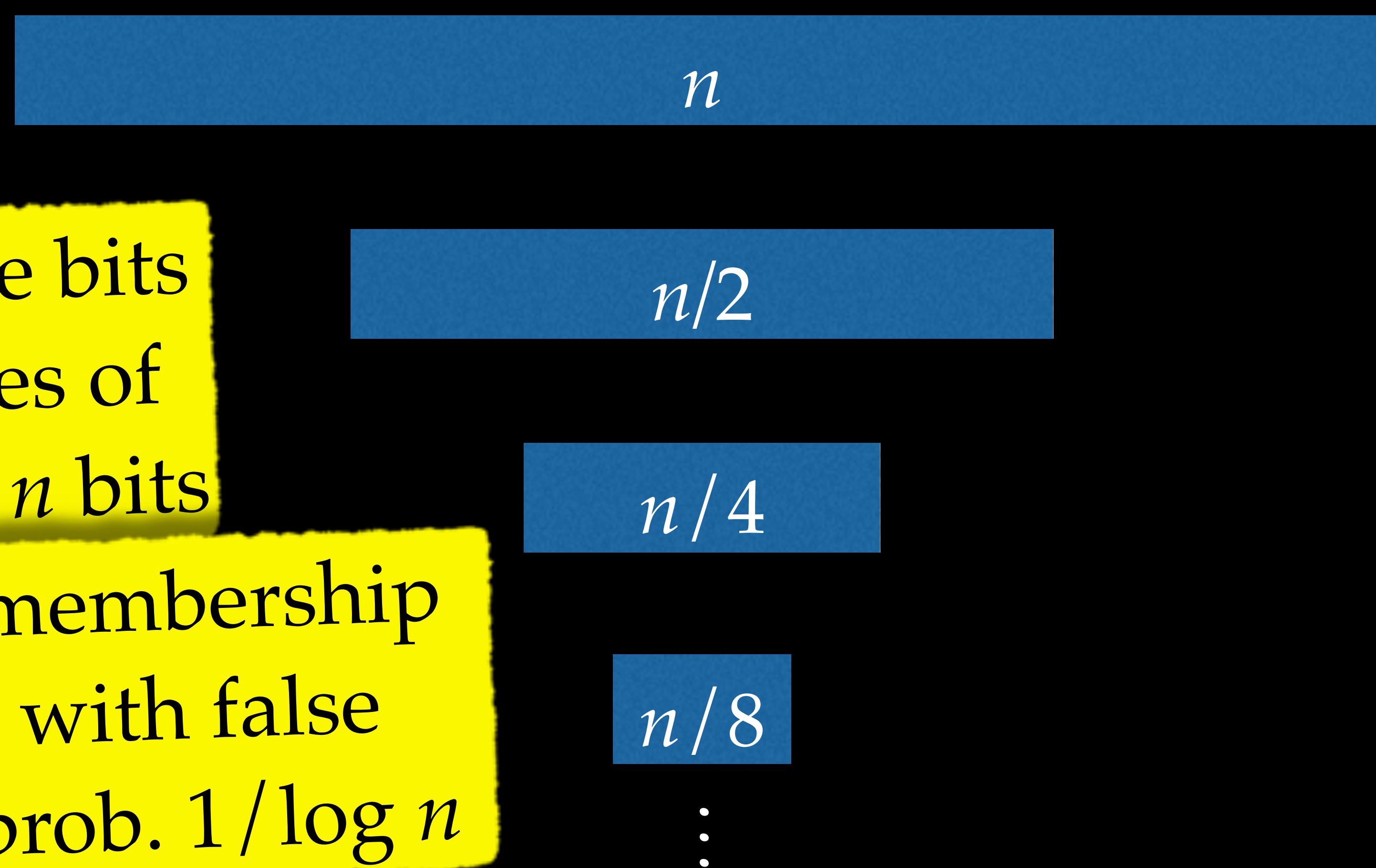
 # levels: $\log n$ [improvable to $\log \log n$]

Adding signatures



Idea: Replace bits by signatures of (e.g.) log log n bits

Adding signatures



Idea: Replace bits by signatures of (e.g.) $\log \log n$ bits

Answer membership queries with false positive prob. $1/\log n$

Adding signatures

n

$n/2$

$n/4$

$n/8$

$\vdots$

Idea: Replace bits by signatures of (e.g.) $\log \log n$ bits

Answer membership queries with false positive prob. $1/\log n$

Insert(x): Search for space to place signature. If hash value **and** signature coincide, store x explicitly.

Adding signatures

n

$n/2$

$n/4$

$n/8$

⋮

Idea: Replace bits by signatures of (e.g.) log log n bits

Answer membership queries with false positive prob. $1/\log n$

Insert(x): Search for space to place signature. If hash value **and** signature coincide, store x explicitly.

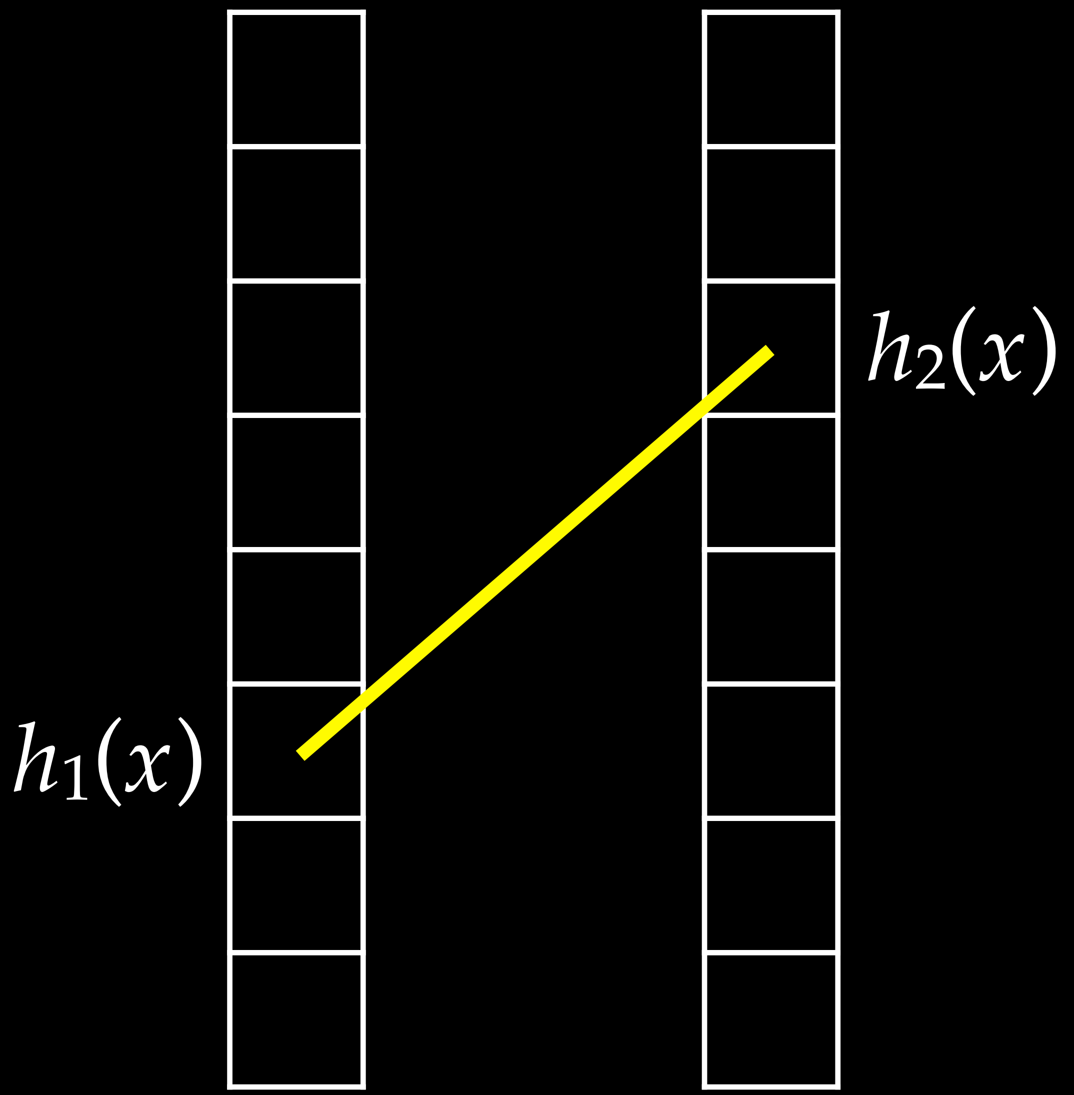
Total size: $2n \log \log n + O(n)$ bits

 # levels: $\log n$ [improvable to $\log \log n$]

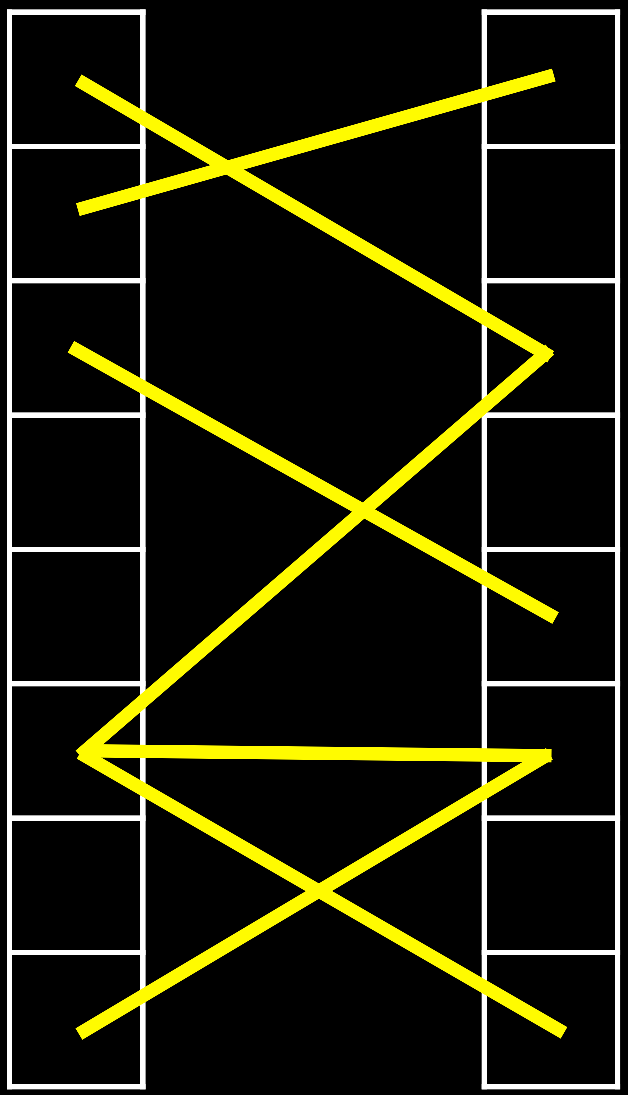
Retrieval w/o perfect hashing

- The best “implementable” solutions of perfect hashing with range n use 2-3 bits/key.
- Lower bound: Need > 1.44 bits/key.
- Is it possible to get rid of this fixed cost per key?

Choice graph



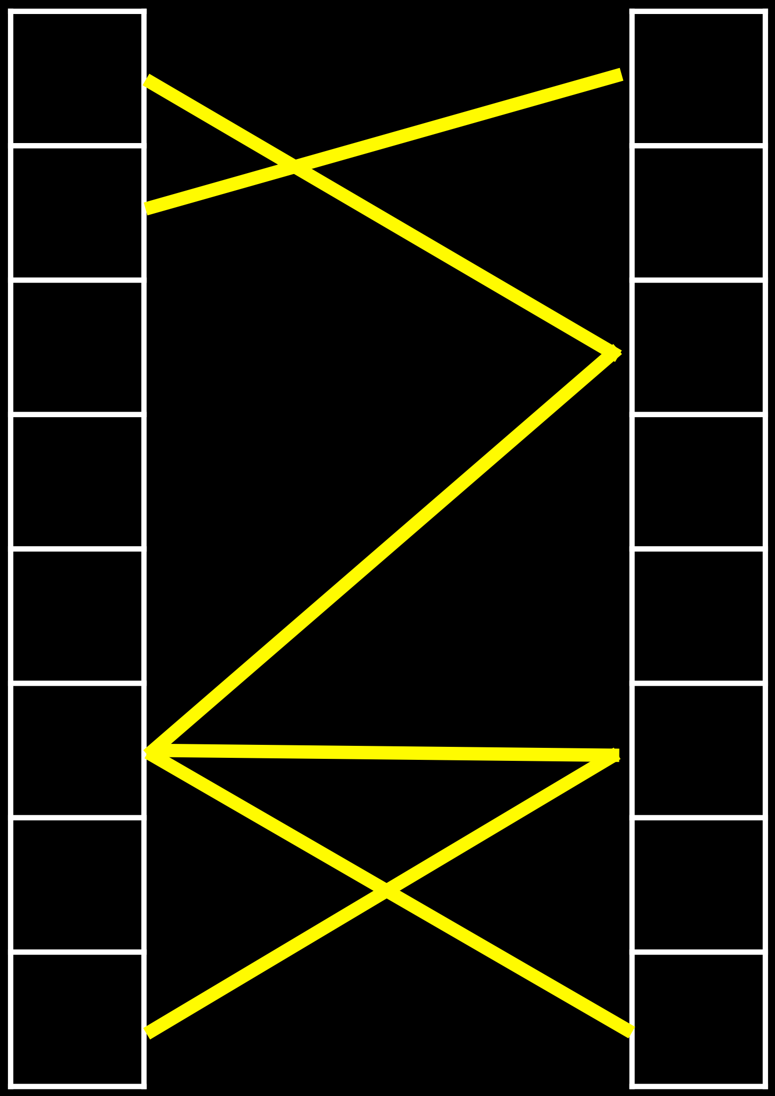
Choice graph



$$V = \{1, \ldots, r/2\} \cup \{r/2 + 1, \ldots, r\}$$

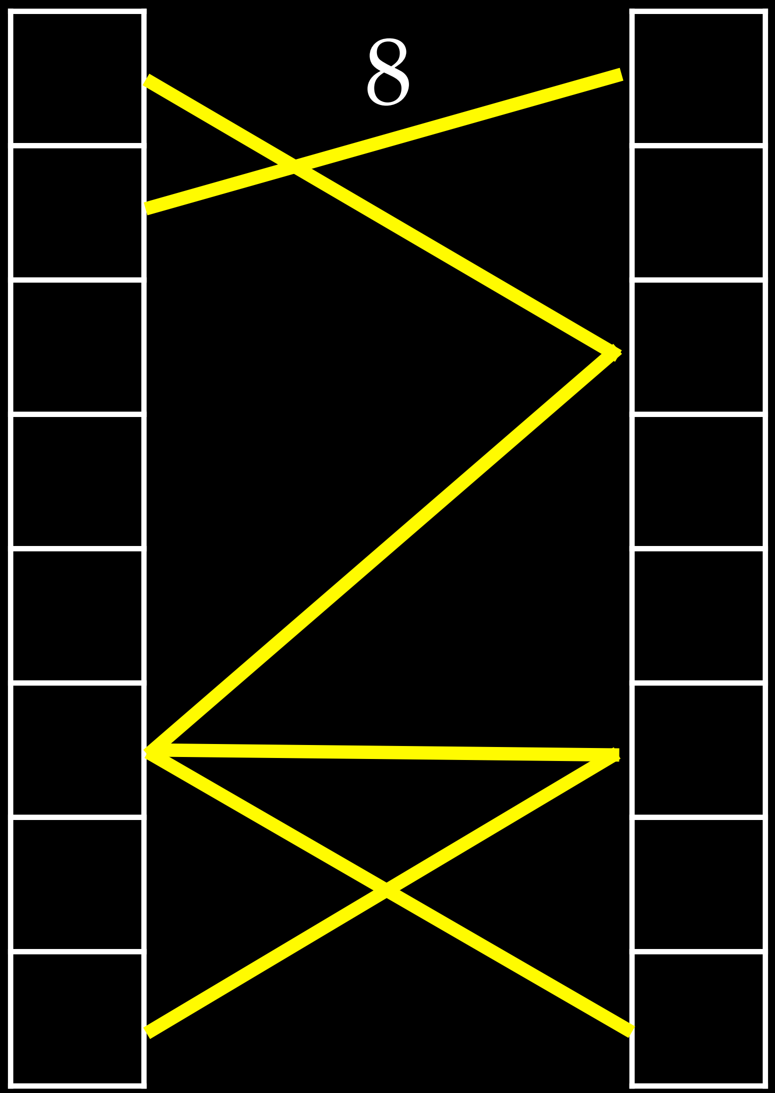
$$E = \{\{h_1(x), h_2(x)\} \mid x \in S\}$$

Associating values with edges



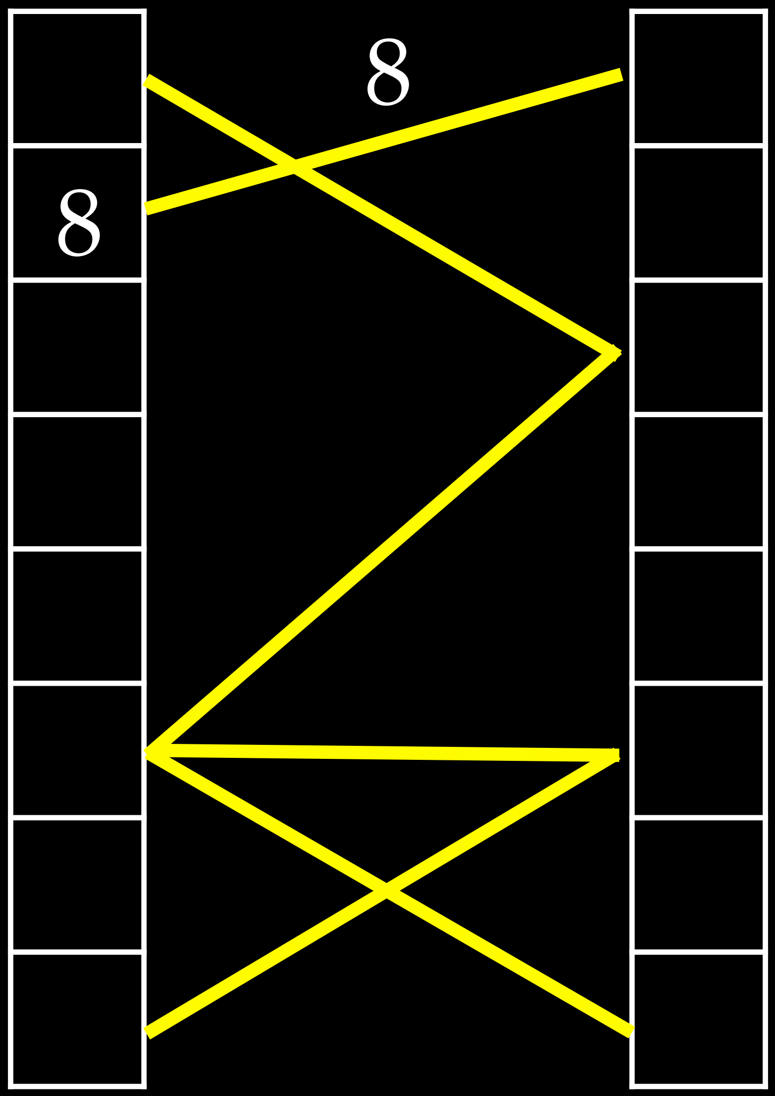
Idea: Choose hash table entries s.t $f(x)=T[h_1(x)]+T[h_2(x)]$

Associating values with edges



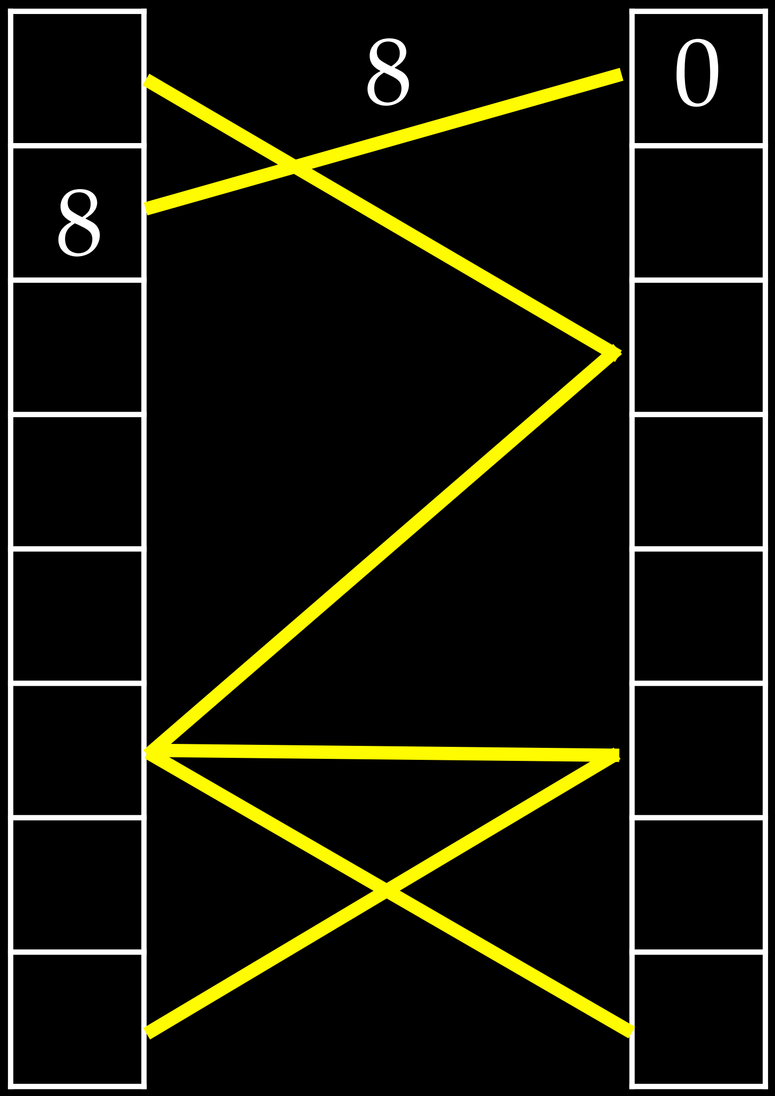
Idea: Choose hash table entries s.t $f(x)=T[h_1(x)]+T[h_2(x)]$

Associating values with edges



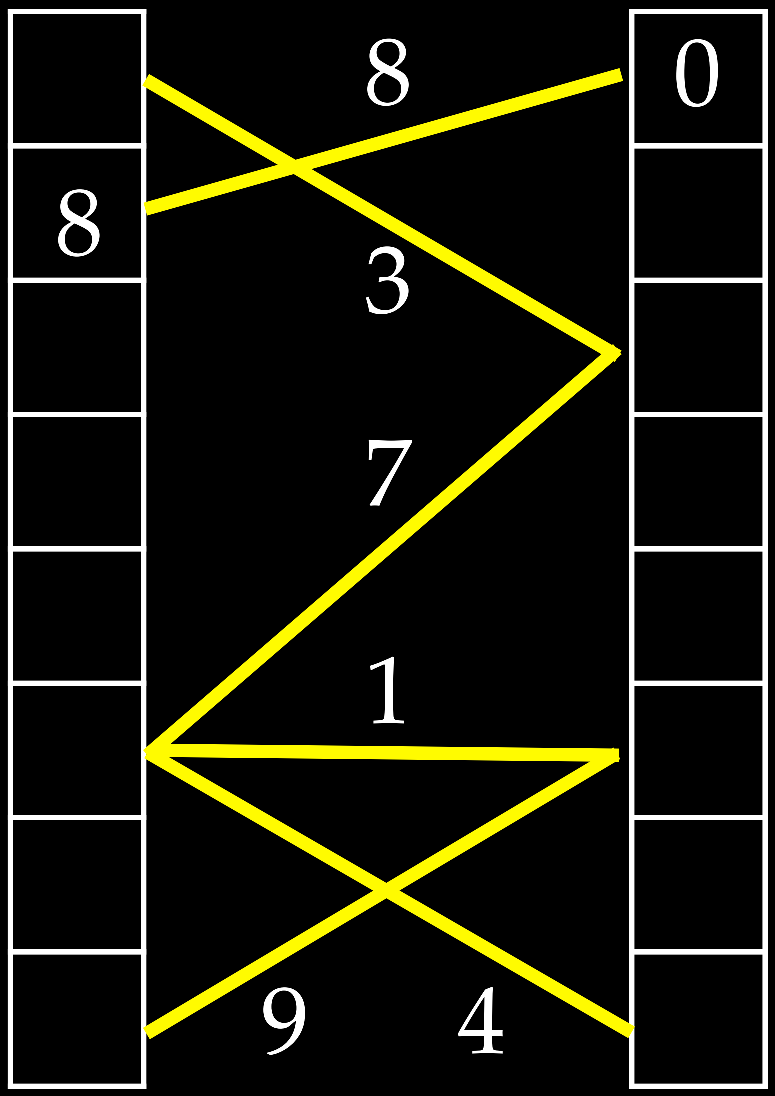
Idea: Choose hash table entries s.t $f(x)=T[h_1(x)]+T[h_2(x)]$

Associating values with edges



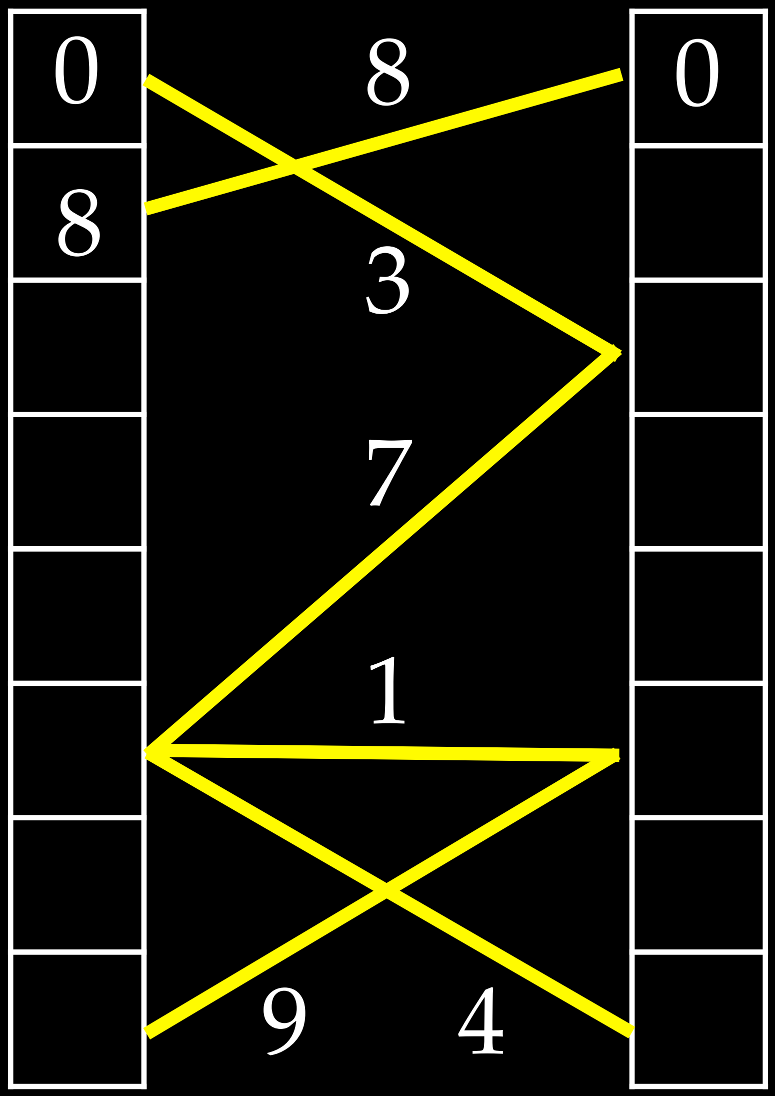
Idea: Choose hash table entries s.t $f(x)=T[h_1(x)]+T[h_2(x)]$

Associating values with edges



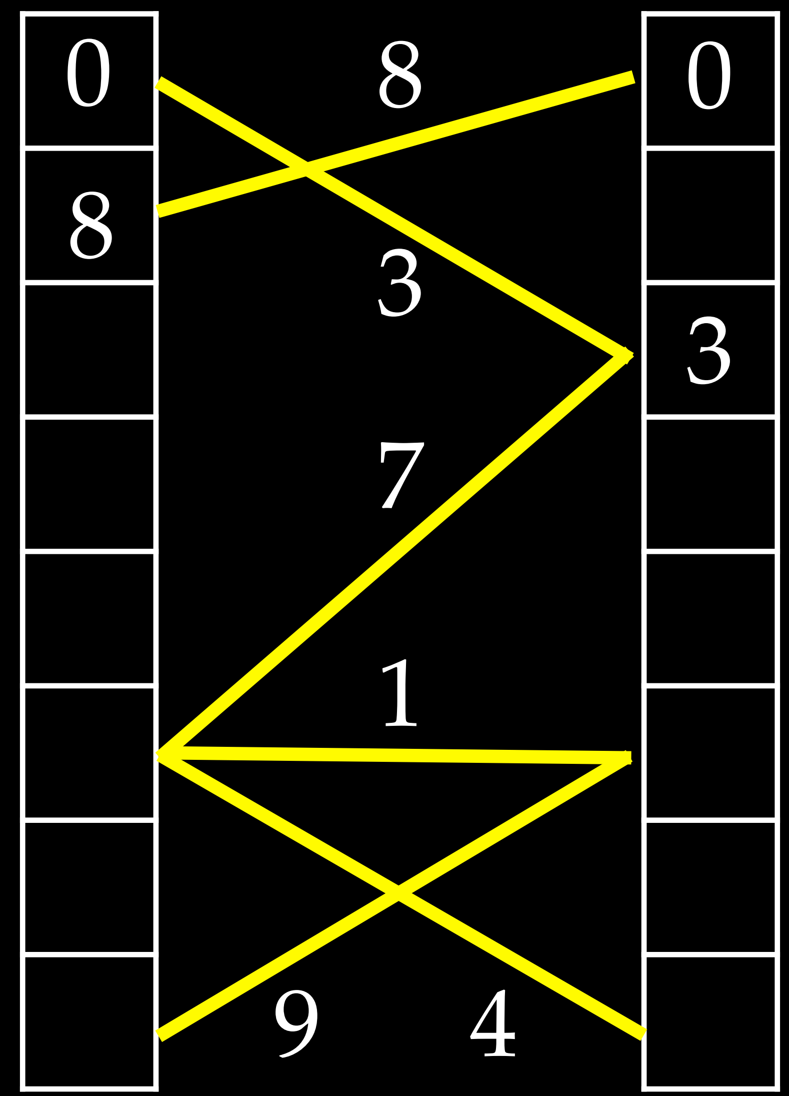
Idea: Choose hash table entries s.t $f(x)=T[h_1(x)]+T[h_2(x)]$

Associating values with edges



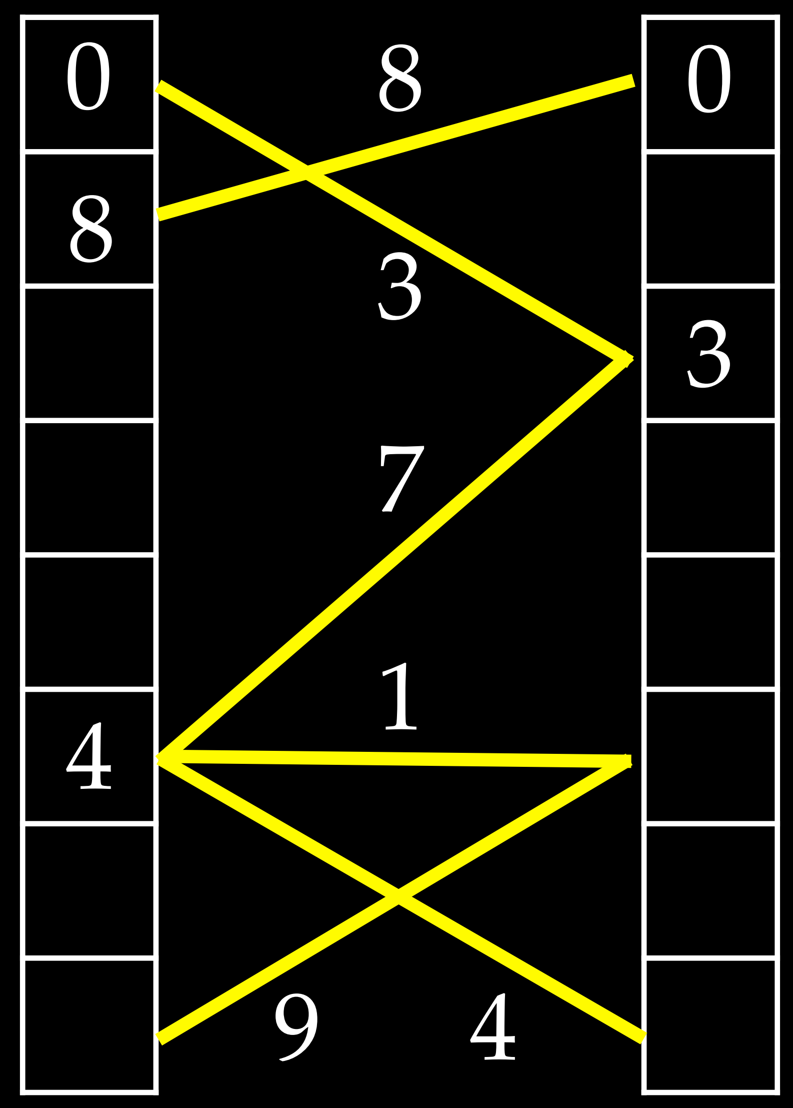
Idea: Choose hash table entries s.t $f(x)=T[h_1(x)]+T[h_2(x)]$

Associating values with edges



Idea: Choose hash table entries s.t $f(x)=T[h_1(x)]+T[h_2(x)]$

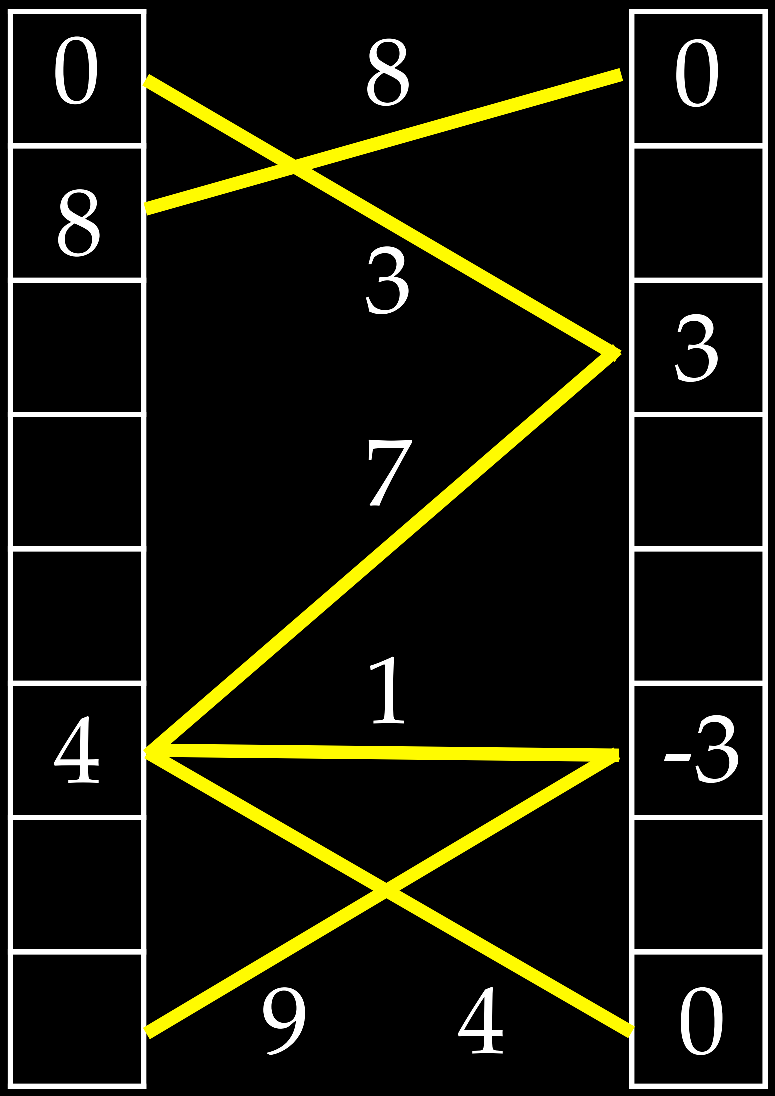
Associating values with edges



Idea: Choose hash table entries s.t $f(x)=T[h_1(x)]+T[h_2(x)]$

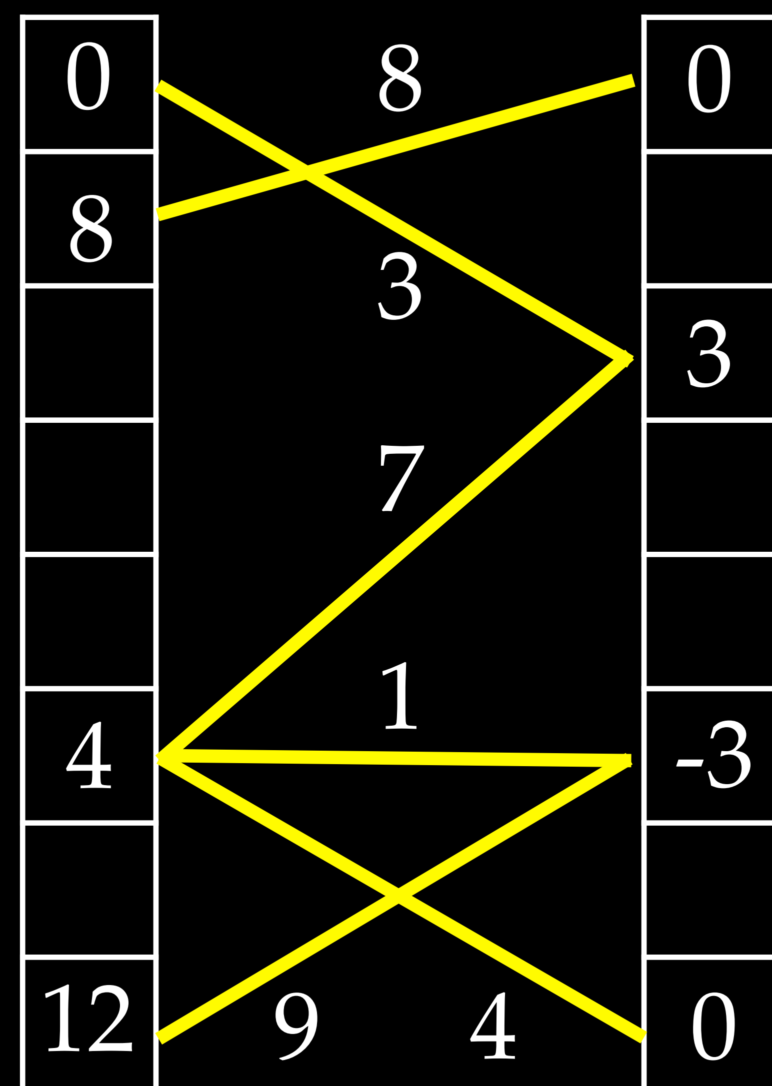
Associating values with edges

Idea: Choose hash table entries s.t $f(x)=T[h_1(x)]+T[h_2(x)]$



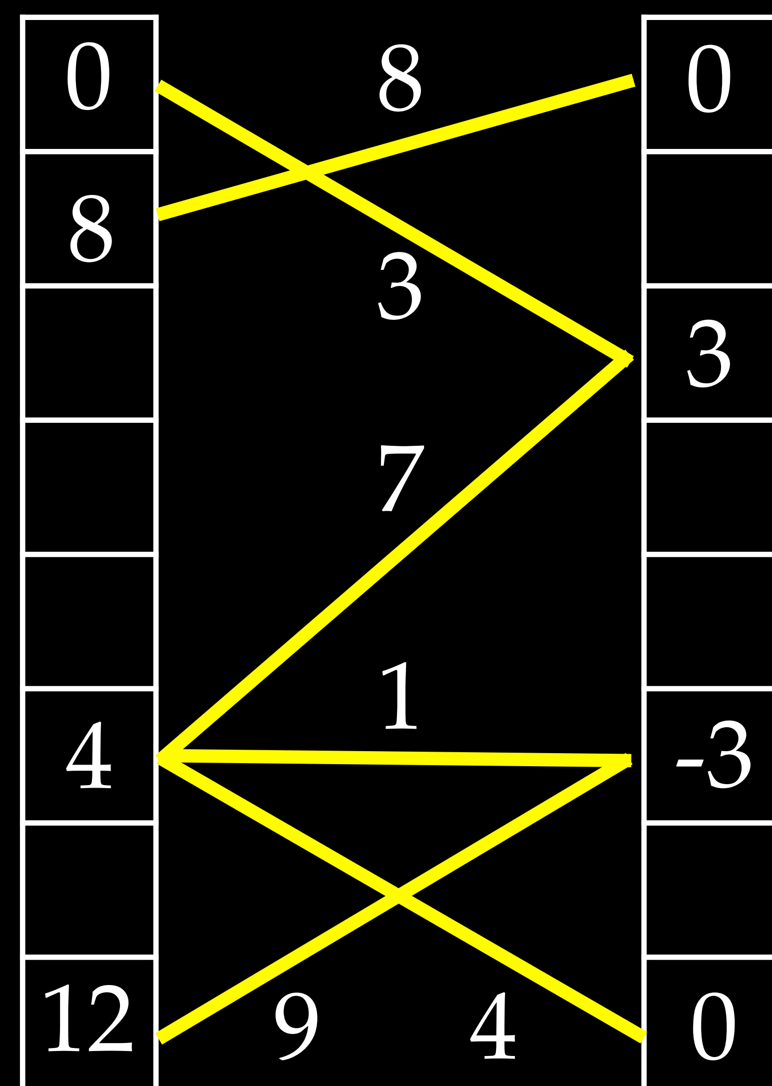
Associating values with edges

Idea: Choose hash table entries s.t $f(x) = T[h_1(x)] + T[h_2(x)]$



Associating values with edges

Idea: Choose hash table entries s.t $f(x)=T[h_1(x)]+T[h_2(x)]$



Works if choice graph is acyclic!

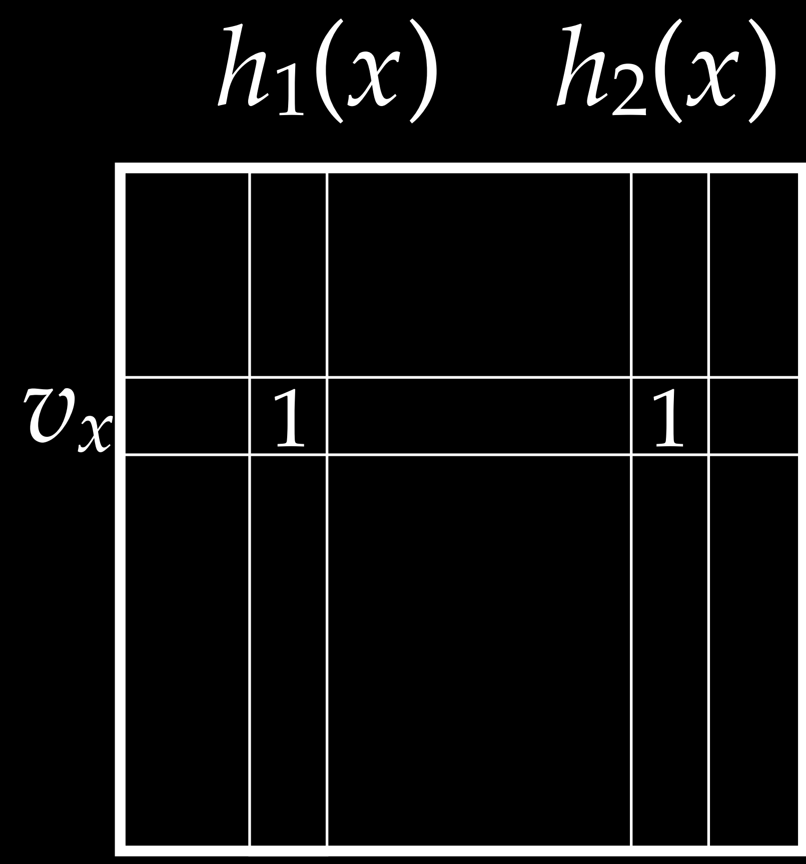
Random graph theory

Assuming fully random hash functions

- Kind of random graph depends on whether $n < r/(2+\varepsilon)$ (*threshold*).
- Below threshold: Connected components are all *pseudotrees* (trees + at most 1 edge) with high probability.
- In fact, *acyclic* with constant probability.

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 choice hypergraph is acyclic, choice matrix A has full rank (any field).
- 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

- For k=3: Enough to store 1.23 values/key

Choice matrix properties

- **Lemma**: If choice hypergraph is acyclic, choice matrix A has full rank (any field).
- 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

- For k=3: Enough to store 1.23 values/key

By increasing k, value of r/n can be made arbitrarily close to 1.

Choice matrix properties

- **Lemma**: If choice hypergraph is acyclic, choice matrix A has full rank (any field).
- 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

- For k=3: Enough to store 1.23 values/key

By increasing k, value of r/n can be made arbitrarily close to 1.

Drawback: Solving the linear systems becomes more demanding.

Some references

- Broder & Karlin: Multilevel Adaptive Hashing
  http://dl.acm.org/citation.cfm?id=320181 (behind paywall)

- Lu, Prabhakar, & Bonomi: Perfect Hashing for Network Applications
  web.stanford.edu/~balaji/papers/06perfecthashing.pdf

- Dietzfelbinger & Pagh: Succinct Data Structures for Retrieval and Approximate Membership
  www.itu.dk/people/pagh/papers/bloomier.pdf

- Mortensen, Pagh & Patrascu: On Dynamic Range Reporting in One Dimension [section 2]
  http://www.itu.dk/people/pagh/papers/dyn1d.pdf

- Pagh, Segev, & Wieder. How to Approximate A Set Without Knowing Its Size in Advance
  http://www.itu.dk/people/pagh/papers/dynbloom.pdf