Almost random graphs with simple hash functions

Martin Dietzfelbinger

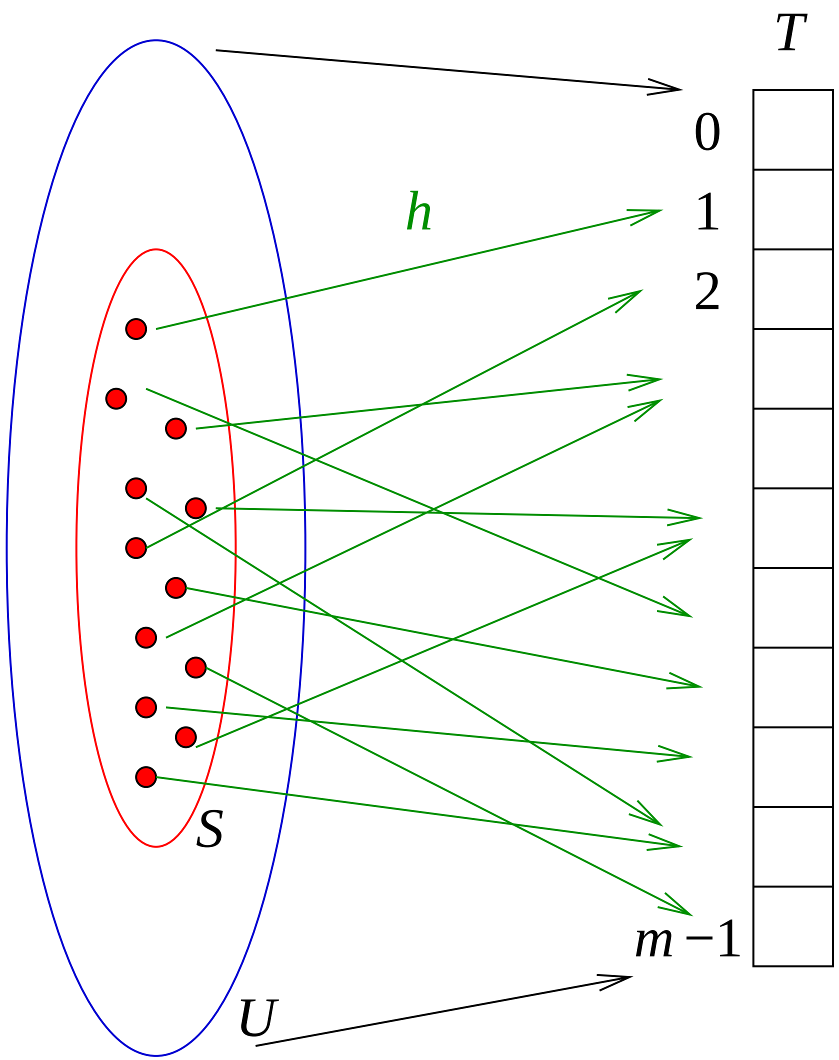
Technische Universität Ilmenau

(Joint work with Philipp Woelfel, Universität Dortmund)

[Appeared: STOC'03]

Version AKA – Hashing, 6.2.2007

Hashing



$h\colon U \to [m]$

U : Universe of all keys

$[m] = \{0, \ldots, m-1\}$:

the range,

indices in table T

Interested in behaviour of h

on $S \subseteq U$, $n = |S|$.

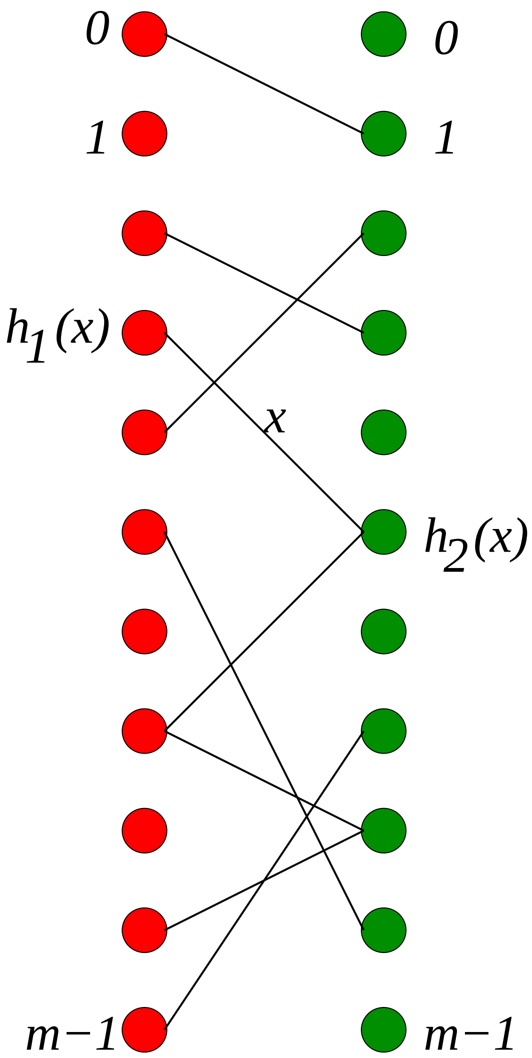
Hashing with two functions

$$h_1, h_2 \colon U \to [m]$$

$[m] = \{0, \ldots, m-1\}$: the range, indices in tables T_1, T_2

Interested in behaviour of h_1, h_2 on $S \subseteq U$.

The graph



Assume h_1, h_2 are "random"

$\longrightarrow$ "random" bipartite graph

$G(S, h_1, h_2)$

edge set:

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

? Randomness properties of $G(S, h_1, h_2)$

Why bother?

Applications (later):

- Cuckoo hashing
- Generating fully random hash functions
- Shared memory simulation
- . . .

Overview

- Universal Hashing
- Structure of function pairs
- Bad substructures of graph, Minimizing
- Probability of bad substructures
- Randomness properties
- Application 1: Cuckoo hashing
- Application 2: Fully random hash functions (whp)
- Conclusion

Universal Hashing [Carter/Wegman 79]

Random experiment:

Choose h at random from a set ("class") $\mathcal{H} \subseteq \{h \mid h\colon U \to [m]\}$

Definition: $\mathcal{H}$ is d-universal if

for each fixed sequence $x_1, \ldots, x_d$ of distinct keys in U

$(h(x_1), \ldots, h(x_d))$ is fully random.

Realization, e.g.:

$\mathcal{H}$ "$=$" all polynomials of degree $< d$ over field U, projected into $[m]$

Space: $\Theta(d)$

Evaluation time: $\Theta(d)$

What if we choose h_1, h_2 from known d-universal classes?

- Simple polynomials:

  constant evaluation time $\Rightarrow d$ constant :

  nothing known about randomness properties of $G(S, h_1, h_2)$.

- n^{ε}-universal hash classes of [Siegel 89]

  (Space $\Theta(n^{\zeta})$, $1 > \zeta > \varepsilon$; evaluation time $O(1)$):

  many properties of truly random graphs hold.

  (Used in many theoretical applications;

  evaluation time unpracticable.)

Our aim: Get good randomness properties in $G(S, h_1, h_2)$

at the (evaluation) cost of low degree polynomials.

Structure of functions

Known [DM90] :

$g: U \longrightarrow [r]$ chosen from a d-universal class,

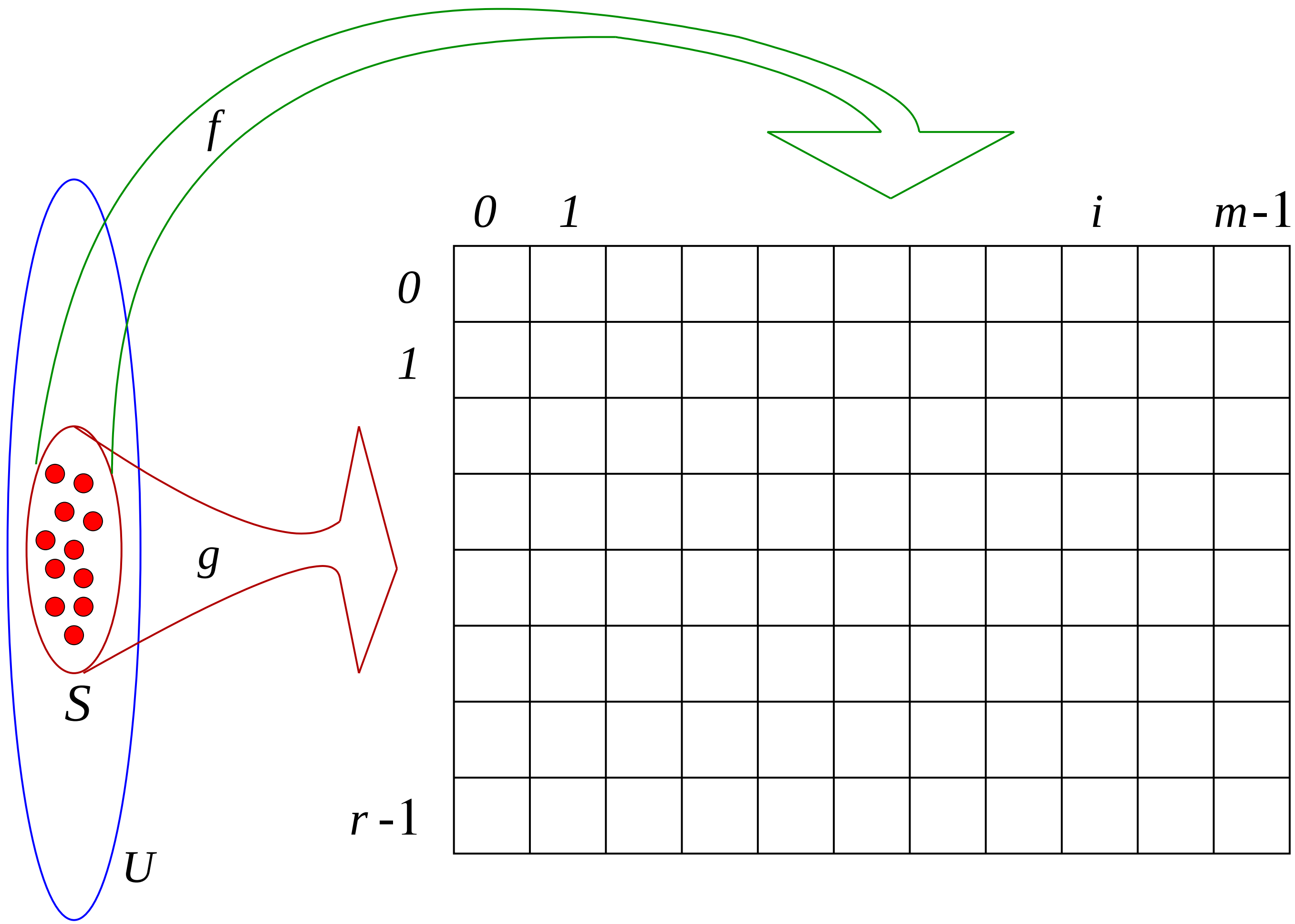
$f: U \longrightarrow [m]$ chosen from a d-universal class,

displacements $z_0, \ldots, z_{r-1}$ chosen randomly in $[m]$

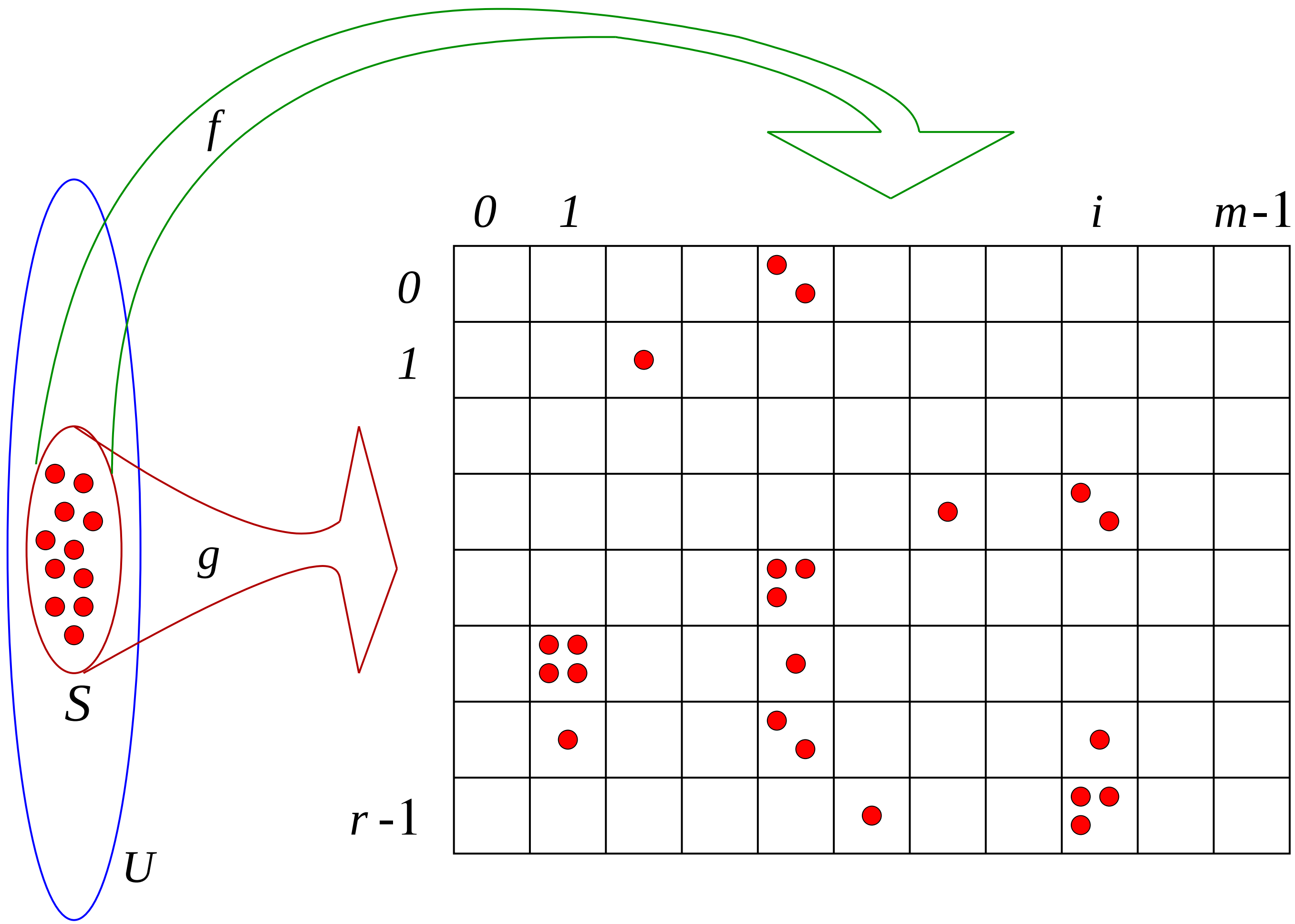
$$h(x) = \big(f(x) + z_{g(x)}\big) \bmod m$$

Constant evaluation time!

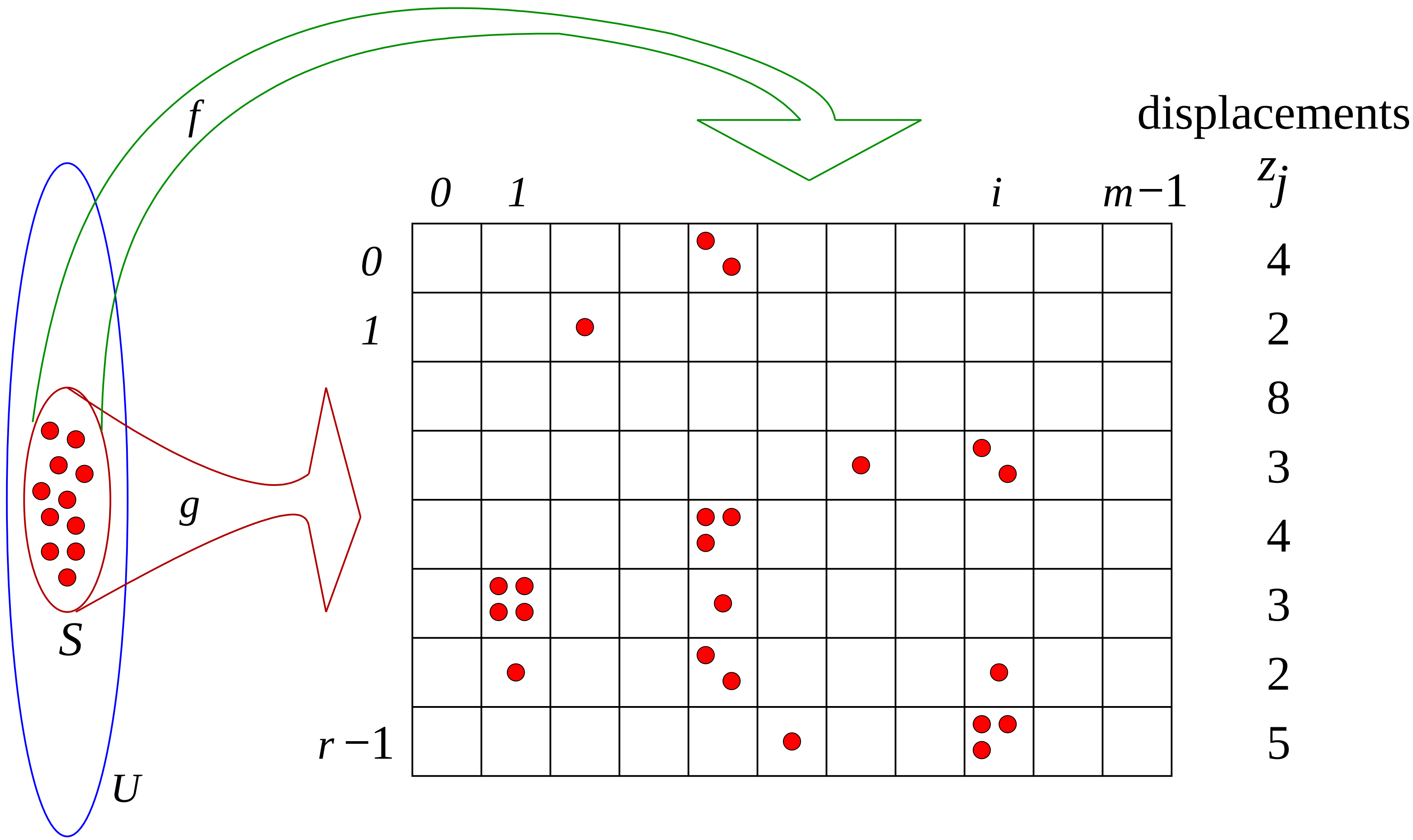
$(h(x))_{x \in S}$ has certain randomness properties.



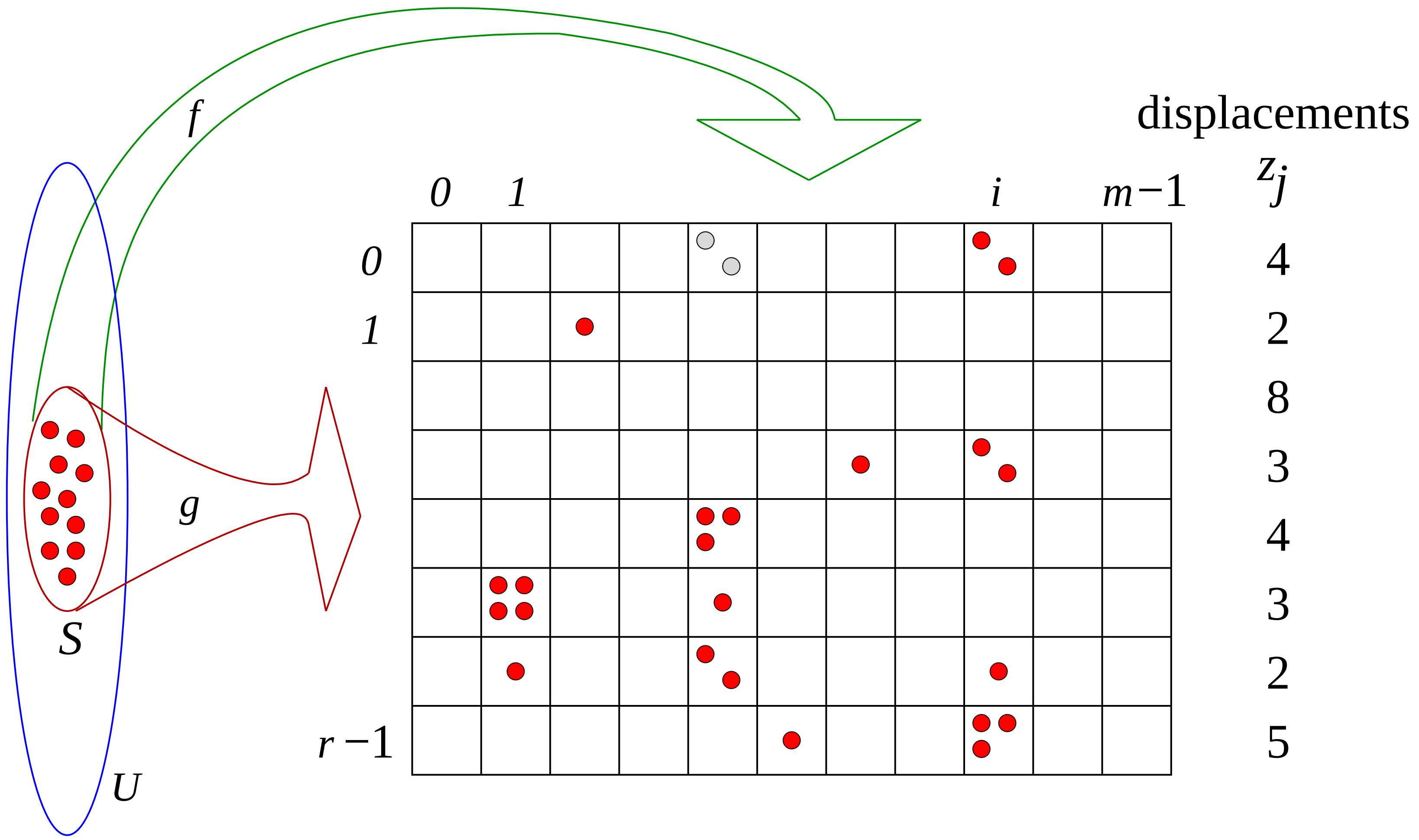
f
0
1
i
m-1
0
1
g
S
r -1
U



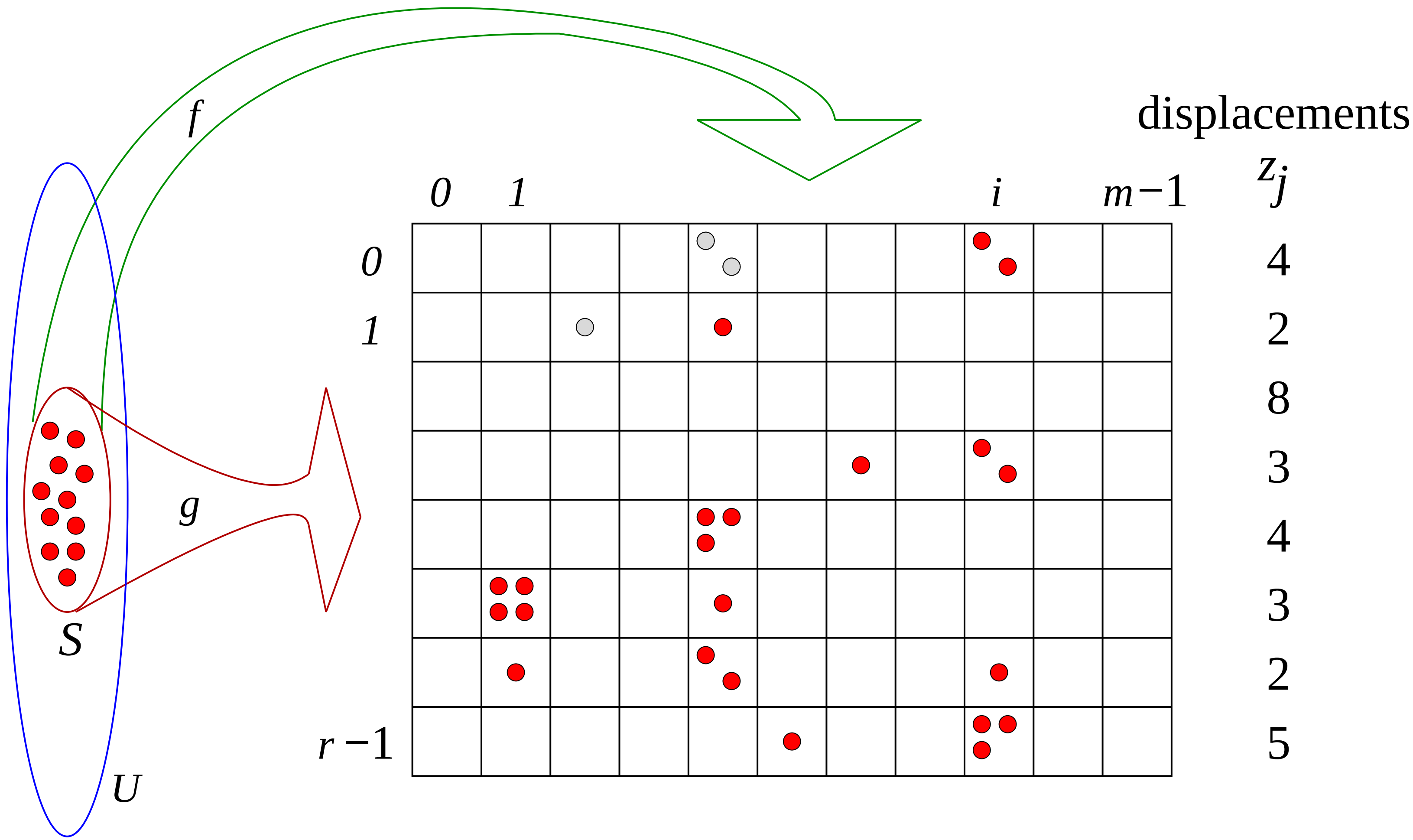
f
0
1
i
m-1
0
1
g
r -1
S
U



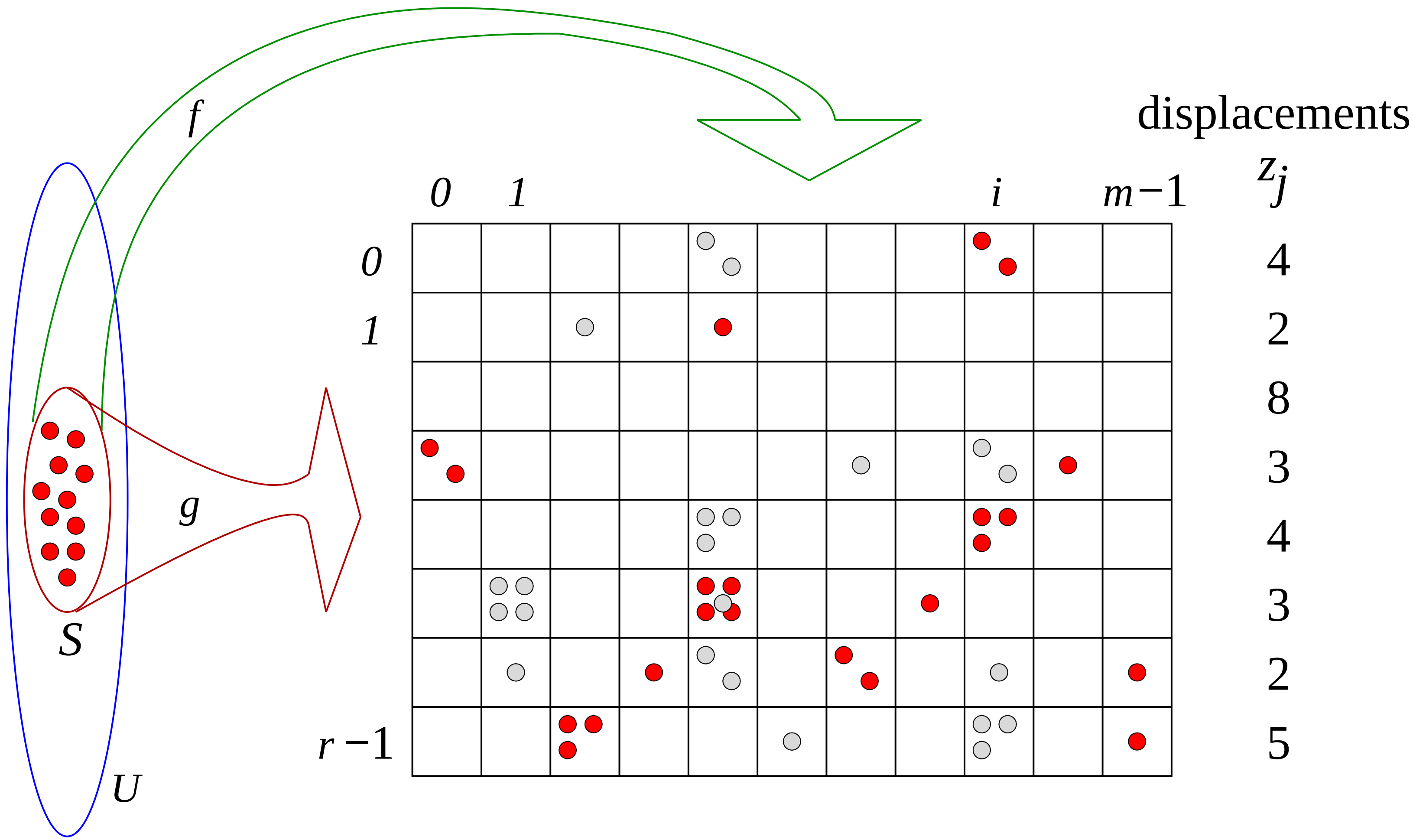
displacements
z_j
f
g
S
U
0
1
i
$m-1$
0
1
$r-1$
4
2
8
3
4
3
2
5



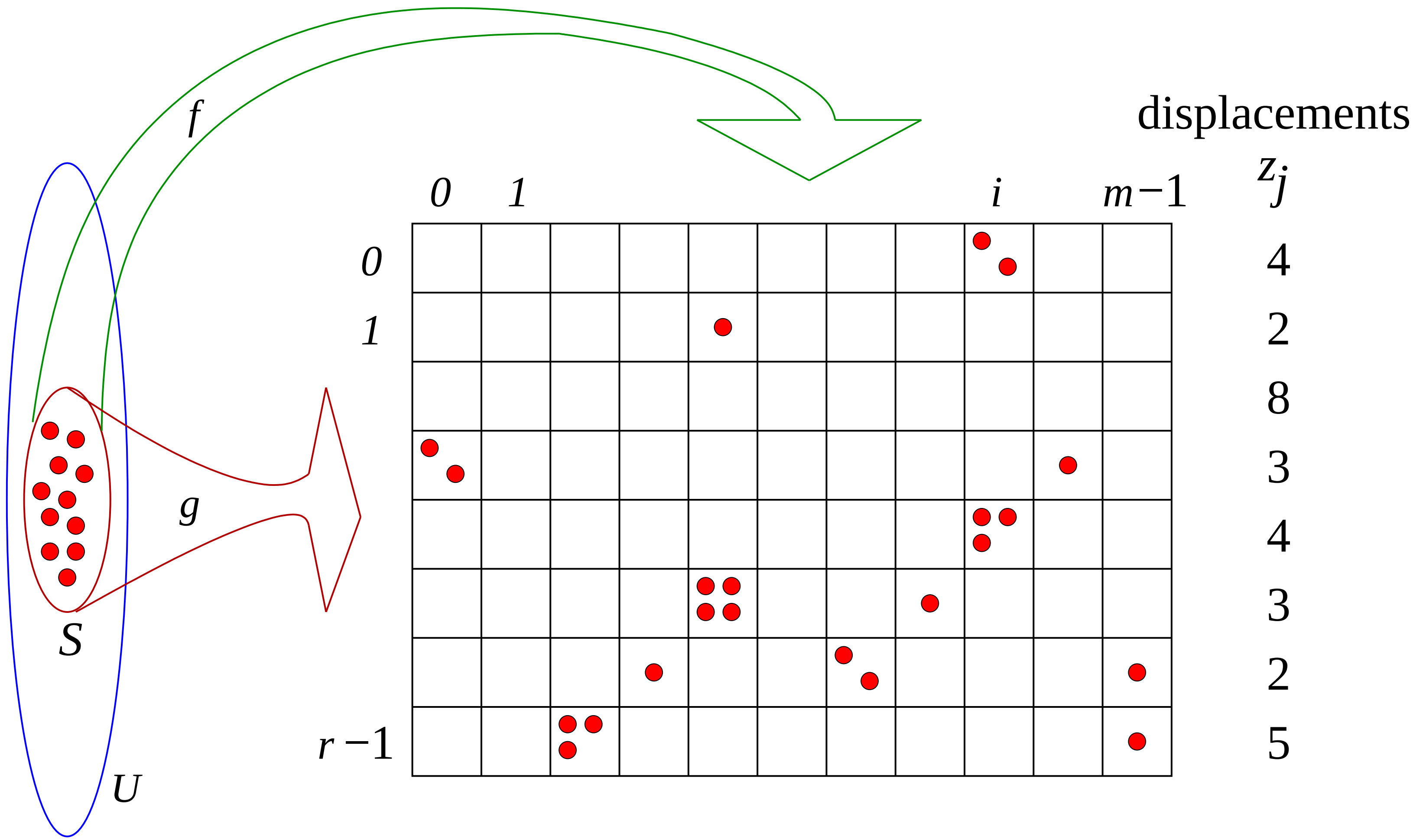
f
g
S
U
displacements
z_j
0
1
i
m−1
0
1
r −1
4
2
8
3
4
3
2
5



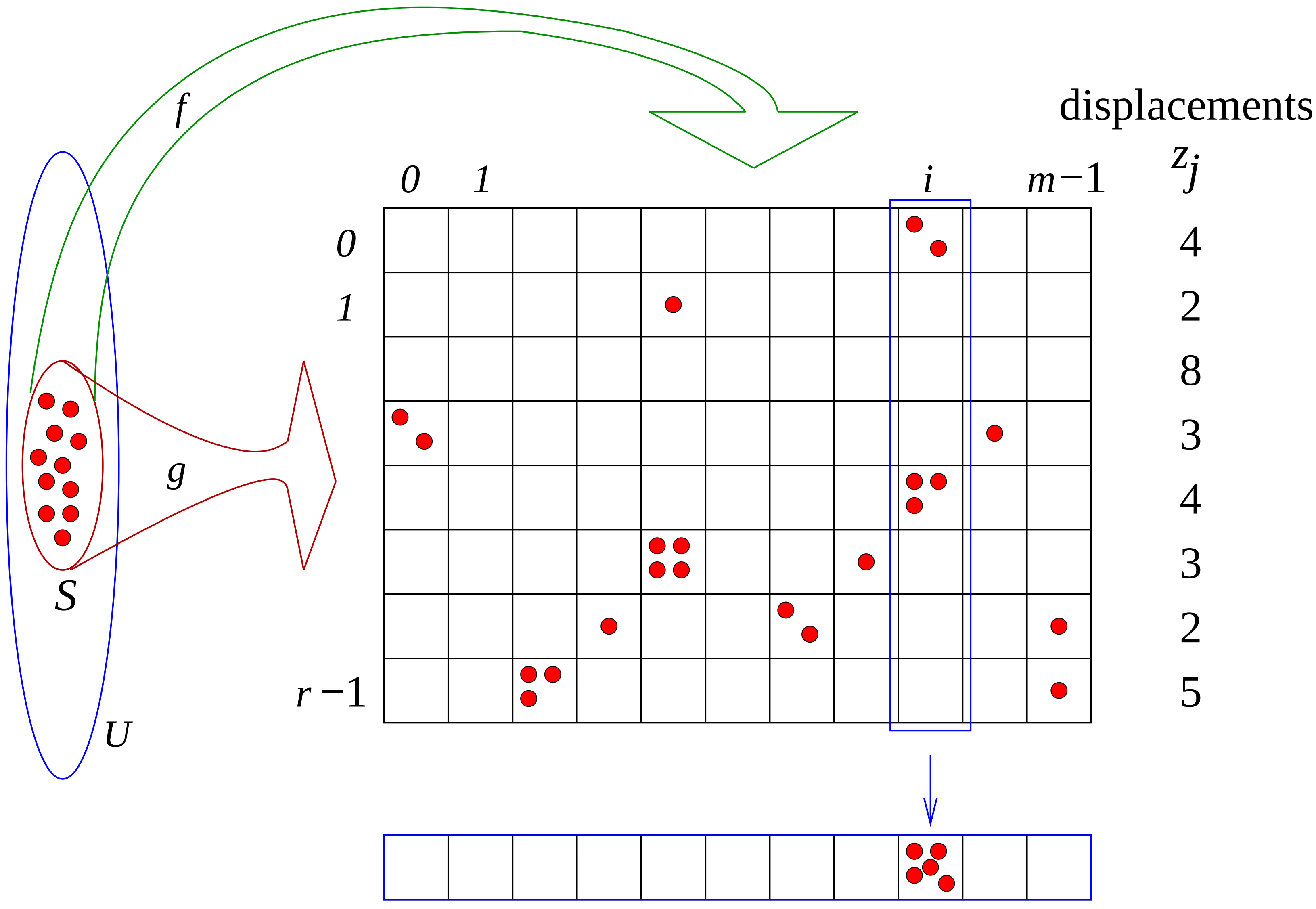
f
displacements
z_j
0
1
i
$m-1$
0
1
$r-1$
4
2
8
3
4
3
2
5
g
S
U



displacements
f
g
S
U
0
1
i
m−1
0
1
r −1
z_j
4
2
8
3
4
3
2
5



displacements
z_j
f
g
S
U
0
1
i
$m-1$
0
1
$r-1$
4
2
8
3
4
3
2
5



f
g
S
U
displacements
0
1
i
m−1
z_j
0
1
r −1
4
2
8
3
4
3
2
5

Structure of functions (cont'd)

$g\colon U \longrightarrow [r]$ chosen from a d-wise independent class,

$f_1, f_2\colon U \longrightarrow [m]$ chosen from a d-wise independent class,

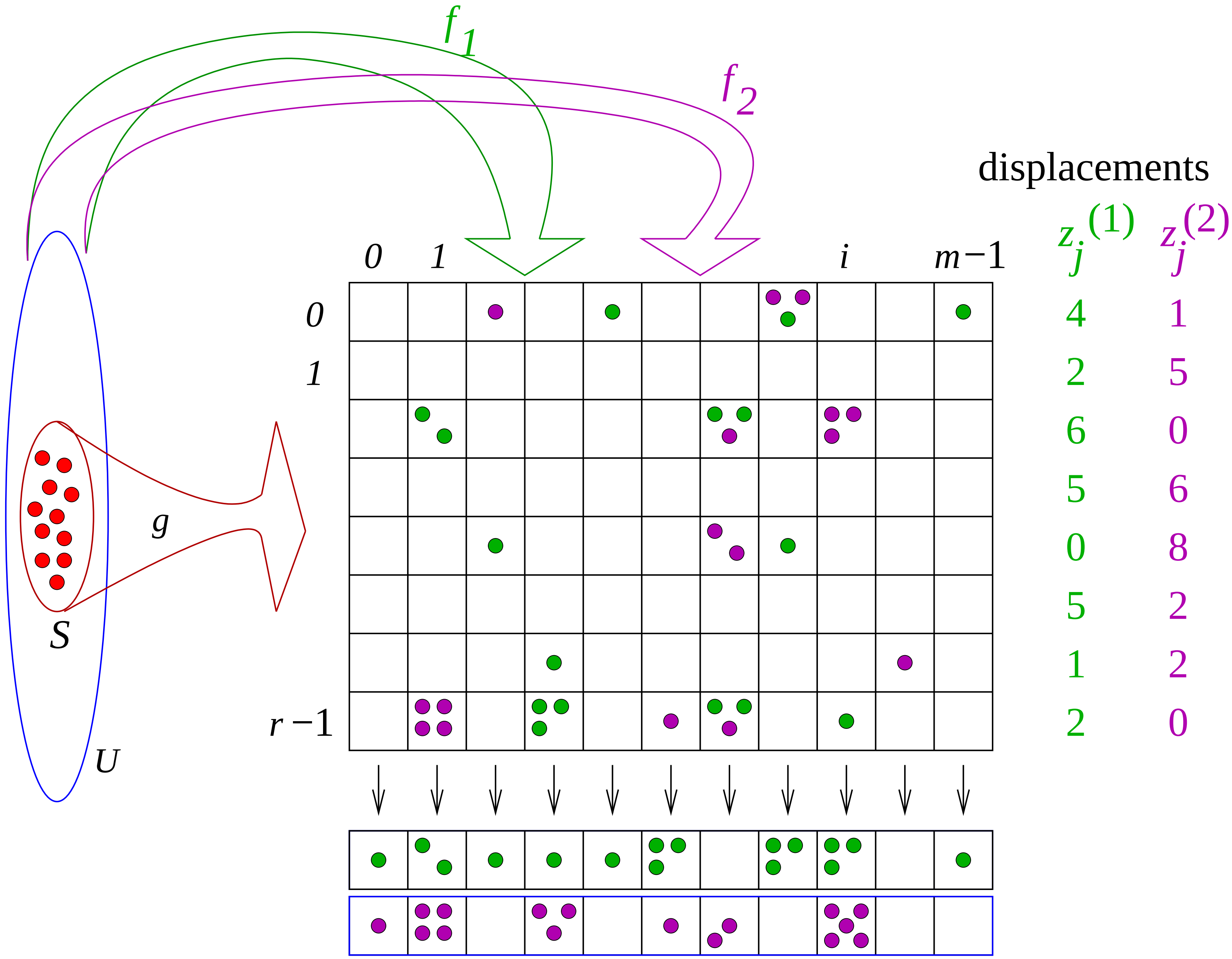
$z_0^{(1)}, \ldots, z_{r-1}^{(1)}$ and $z_0^{(2)}, \ldots, z_{r-1}^{(2)}$ chosen randomly in $[m]$

$$
\begin{aligned}
h_1(x) &= \left(f_1(x) + z_{g(x)}^{(1)}\right) \bmod m \\
h_2(x) &= \left(f_2(x) + z_{g(x)}^{(2)}\right) \bmod m
\end{aligned}
$$

Double DM-construction, but use the same g-function

Constant evaluation time!

Like degree-$(d-1)$-polynomials.



f_1
f_2
displacements
$z_j^{(1)}$
$z_j^{(2)}$
0
1
i
$m-1$
0
1
$r-1$
g
S
U
4 1
2 5
6 0
5 6
0 8
5 2
1 2
2 0

Basic observation:

Let g be given.

Define $B_j = \{x \in S \mid g(x) = j\}$.

Then the $2r$ random vectors

$$(h_1(x))_{x \in B_j}, (h_2(x))_{x \in B_j}, 0 \leq j < r,$$

are independent.

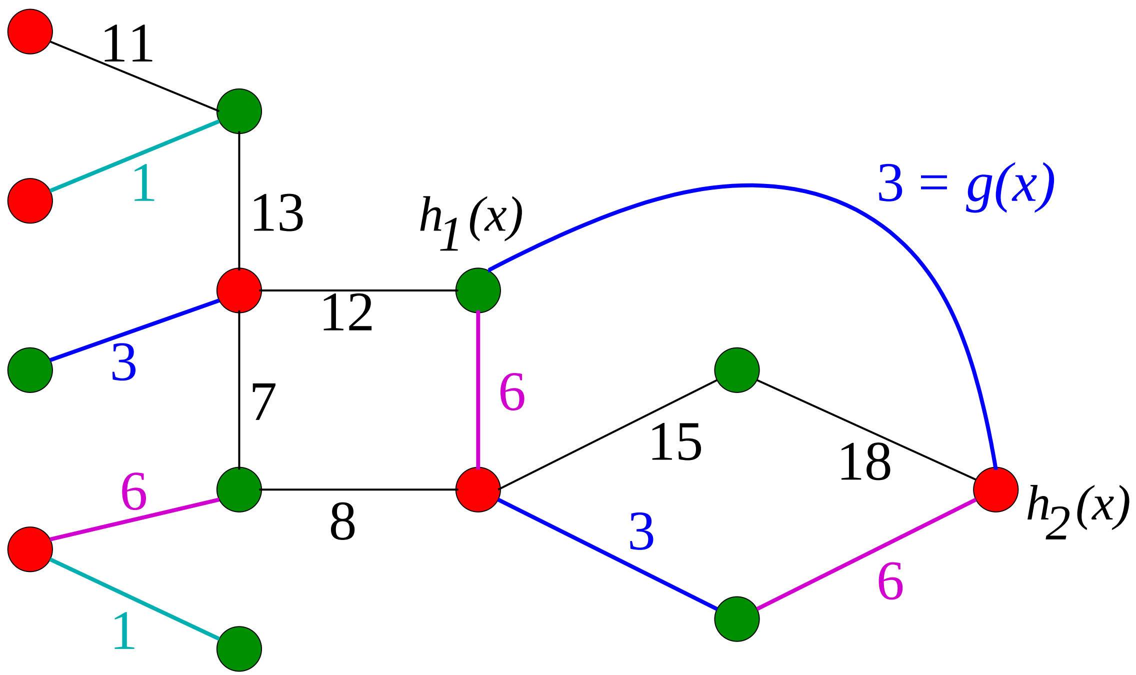
Reason: Random displacements $z_j^{(1)}, z_j^{(2)}$.

Dependencies may exist only among keys inside the same g-row.

Bad substructures

Hope: Inside its connected components graph $G(S, h_1, h_2)$ should behave fully randomly.

Obstructing: $|T| = 16$ keys (edges); $|g(T)| = 11$ used g-values



Connected component in which there are dependencies since the keys of some edges belong to the same g-value.

Measure how far $G(S, h_1, h_2)$ is away from being nice:

Definition: $G = G(S, H_1, h_2)$ is ℓ-bad if

G has a connected component induced by the key set T such that

$$|g(T)| \leq |T| - \ell.$$

(In example: $G(S, h_1, h_2)$ is 5-bad, 4-, 3-, 2-, 1-bad.)

How often do we see ℓ-bad graphs?

If $G(S, h_1, h_2)$ were fully random, there would be no big problem:

Use estimates for the probability that T forms a connected component in a random graph.

Multiply by the probability that there are colliding g-values.

Does not work, because $G(T, h_1, h_2)$ is not random.

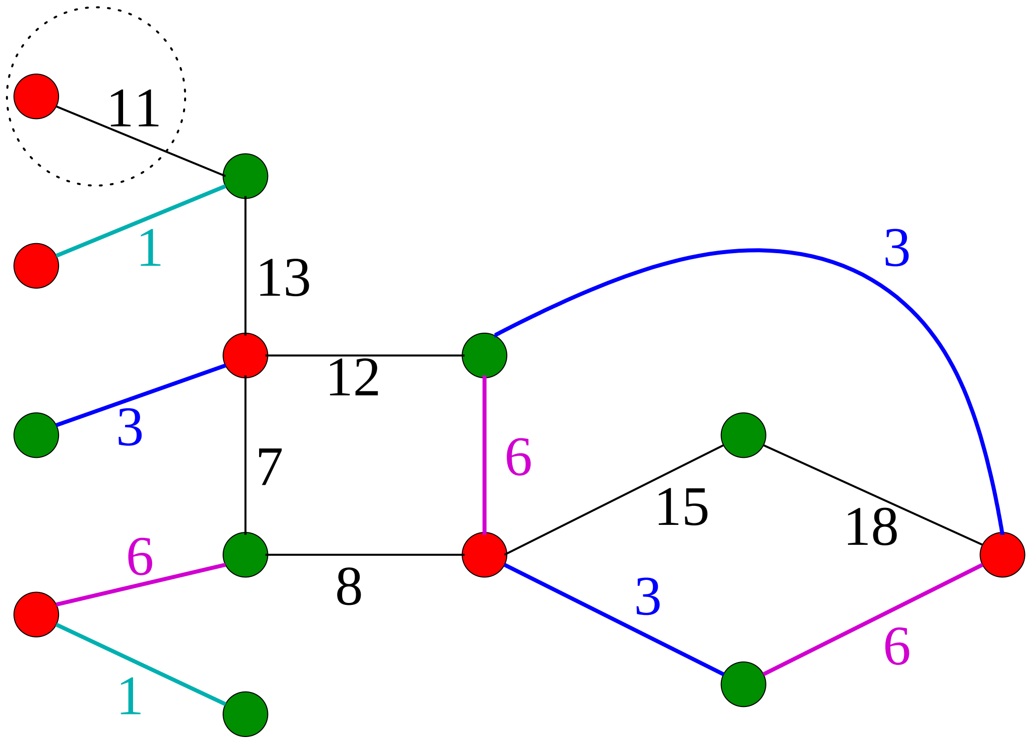
Minimizing obstructing substructures

Assume $G(S, h_1, h_2)$ has a connected component induced by $T \subseteq S$ that makes it ℓ-bad.

Peel!

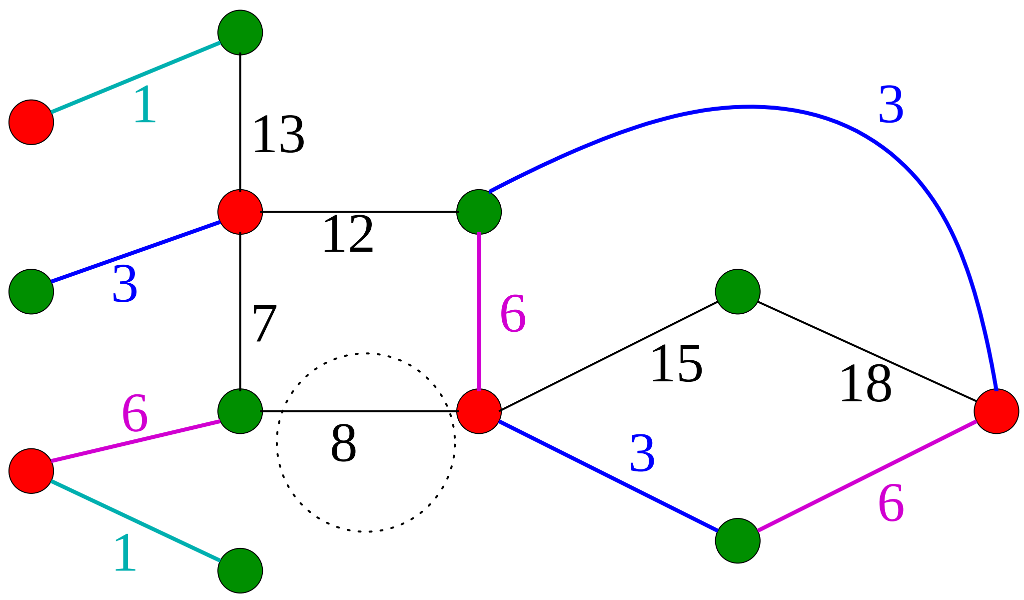
Take out edges (keys) so as to retain a connected, ℓ-bad subgraph.

Aim: Reduce, stay 4-bad.



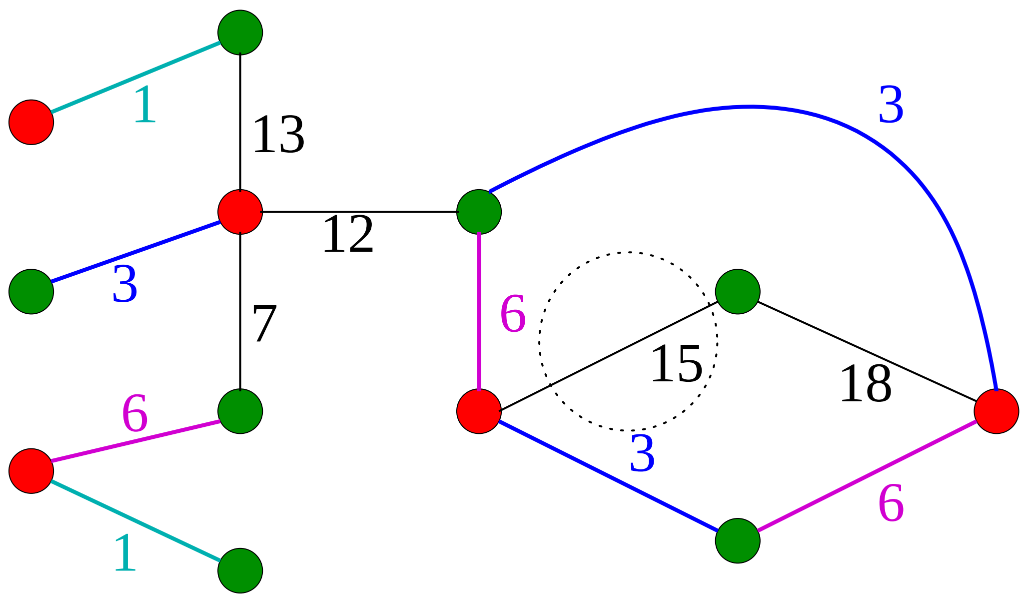
Remove leaf with key that is not g-colliding.

Aim: Reduce, stay 4-bad.



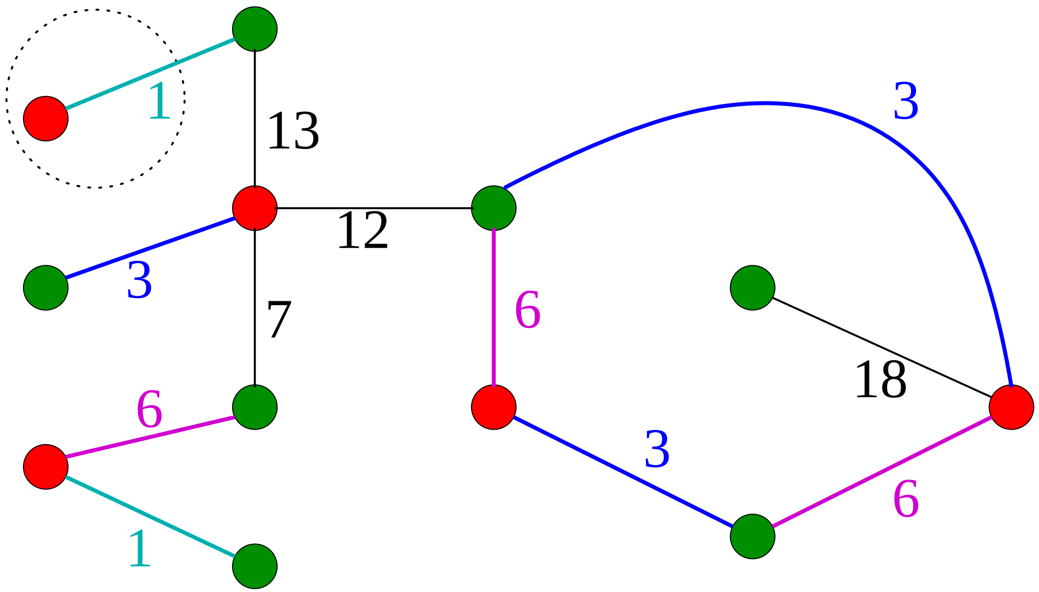
Remove cycle edge with key that is not g-colliding.

Aim: Reduce, stay 4-bad.



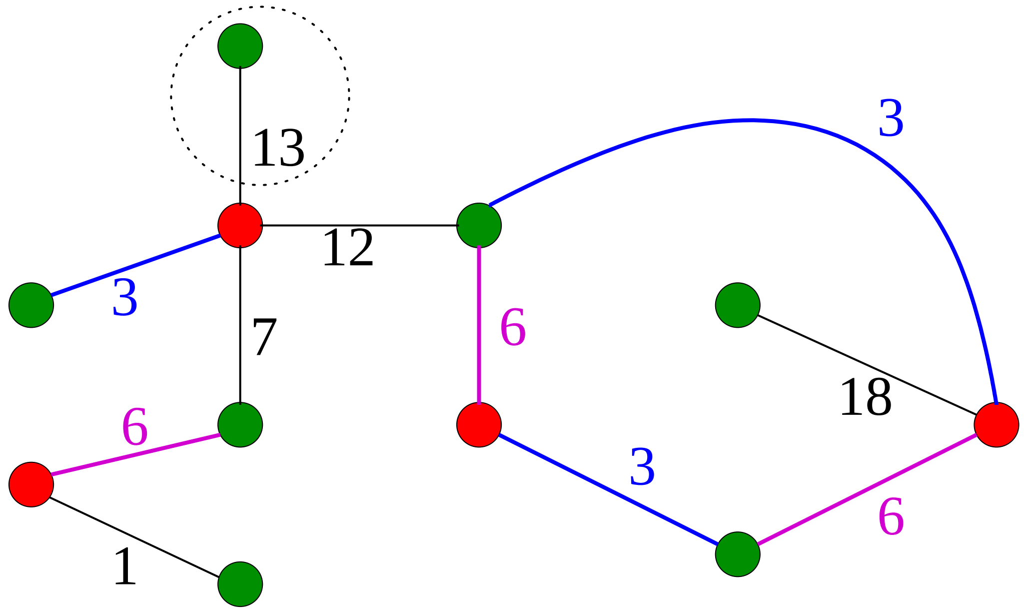
Remove cycle edge with key that is not g-colliding.

Aim: Reduce, ℓ-bad, $\ell = 4$.



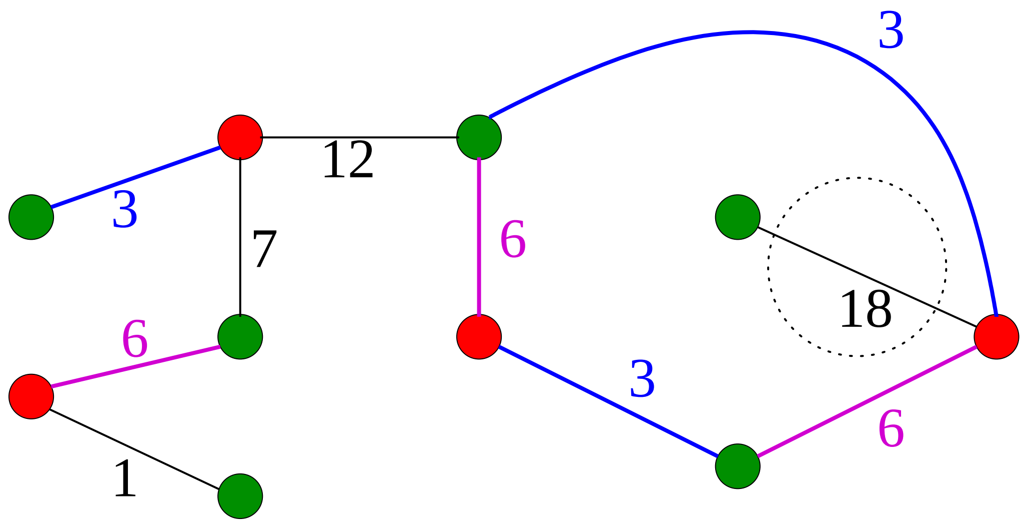
Remove leaf edge with g-colliding key, if $|g(T)| < |T| - \ell$.

Aim: Reduce, stay ℓ-bad, $\ell = 4$.



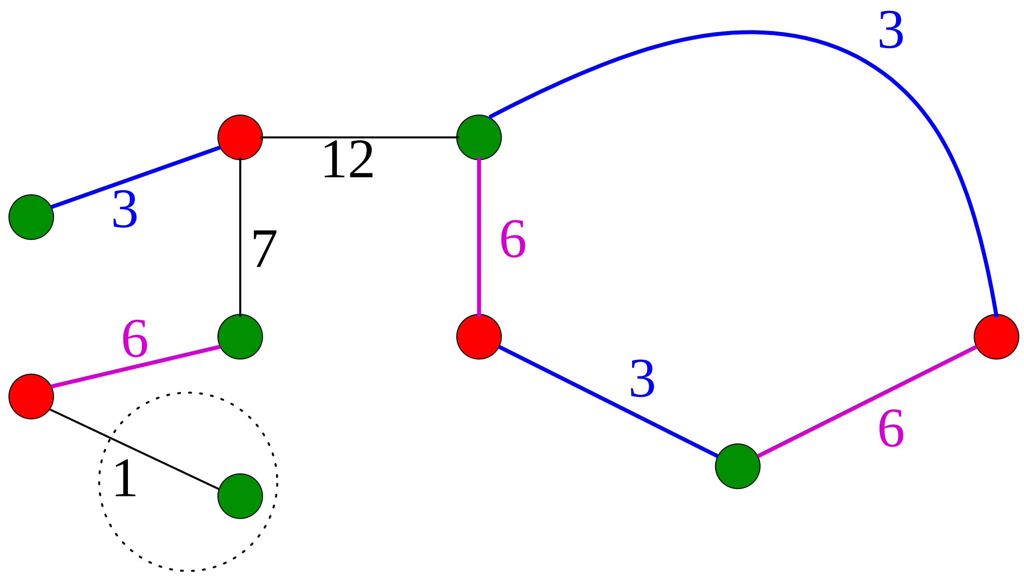
Remove leaf with key that is not g-colliding.

Aim: Reduce, stay ℓ-bad, $\ell = 4$.



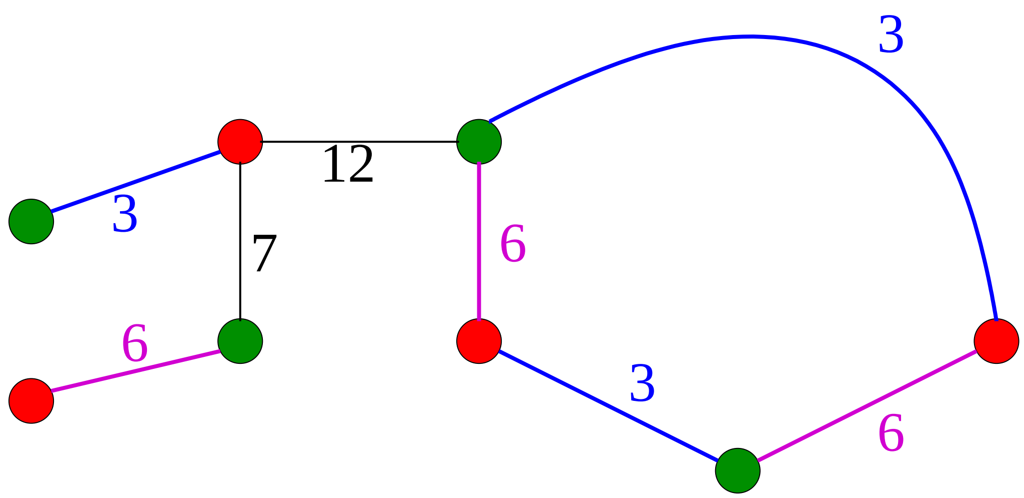
Remove leaf with key that is not g-colliding.

Aim: Reduce, stay ℓ-bad, $\ell = 4$.



Remove leaf with key that is not g-colliding.

Aim: Reduce, stay ℓ-bad, $\ell = 4$.



No more possible moves:

minimal ℓ-bad structure.

General: repeat throwing away:

- non-g-colliding leaf and cycle edges
- g-colliding leaf and cycle edges, as long as $|g(T)| < |T| - \ell$.

Resulting connected minimal structure has at most 2ℓ leaf and cycle edges, and at most 2ℓ g-colliding keys

$\Rightarrow$ can count these structures

Now:

$$\begin{aligned}
&\Pr(G(S,h_1,h_2) \text{ has } \ell\text{-bad component}\\
&\quad\le\quad \Pr(\exists T\colon G(T,h_1,h_2) \text{ is connected, } \ell\text{-bad, minimal})\\
&\quad\le\quad \sum_{T\subseteq S} \Pr(G(T,h_1,h_2) \text{ is connected, } \ell\text{-bad, minimal})
\end{aligned}$$

Nice:

if f_1, f_2 are 2ℓ-wise independent, then within minimal ℓ-bad substructures the dependence produced by keys in the same g-row is made up for by independence via f_1, f_2

$\Rightarrow$ the hash values are fully independent

$\Rightarrow$ we may use known estimates from random graph theory.

Theorem 1

If f_1, f_2, g are 2ℓ-universal, and $m \geq (1+\varepsilon)n$, then $\Pr(B) = \Pr(G$ is ℓ-bad$) = O(n/r^\ell)$.

Example: Use $\ell = 2$, hence 4-universal classes, and $r = n^{3/4}$.

Randomness properties I

For $T \subseteq S$ let $R^*(T) =$ the event that $|g(T)| \geq |T| - \ell$.

Theorem 2

If f_1, f_2, g are 2ℓ-universal, and $m \geq (1+\varepsilon)n$, then for all $T \subseteq S$ we have:

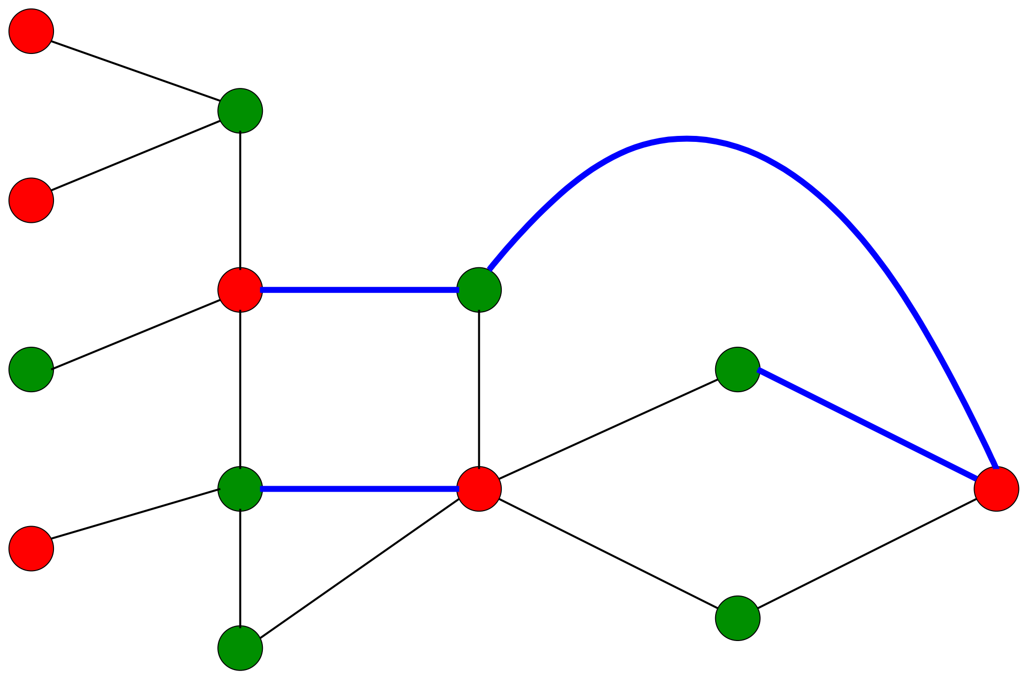
- $R^*(T)$ happens $\Rightarrow$ h_1, h_2 are perfectly random on T.
- $R^*(T)$ does not happen and $G(T, h_1, h_2)$ is within a connected component of $G(S, h_1, h_2)$
  $\Rightarrow G(S, h_1, h_2)$ is ℓ-bad

Intuition:

Apart from a small bad part (probability $O(n/r^\ell)$) everything inside connected components of G is fully random.

Definition: The cyclomatic number of a connected graph $G = (V, E)$ with N vertices and M edges is $M - N + 1$, i.e. the number of edges that are not contained in a (any) spanning tree of G.

Example: 13 nodes, 16 edges, cyclomatic number 4



Randomness properties II

Theorem 3

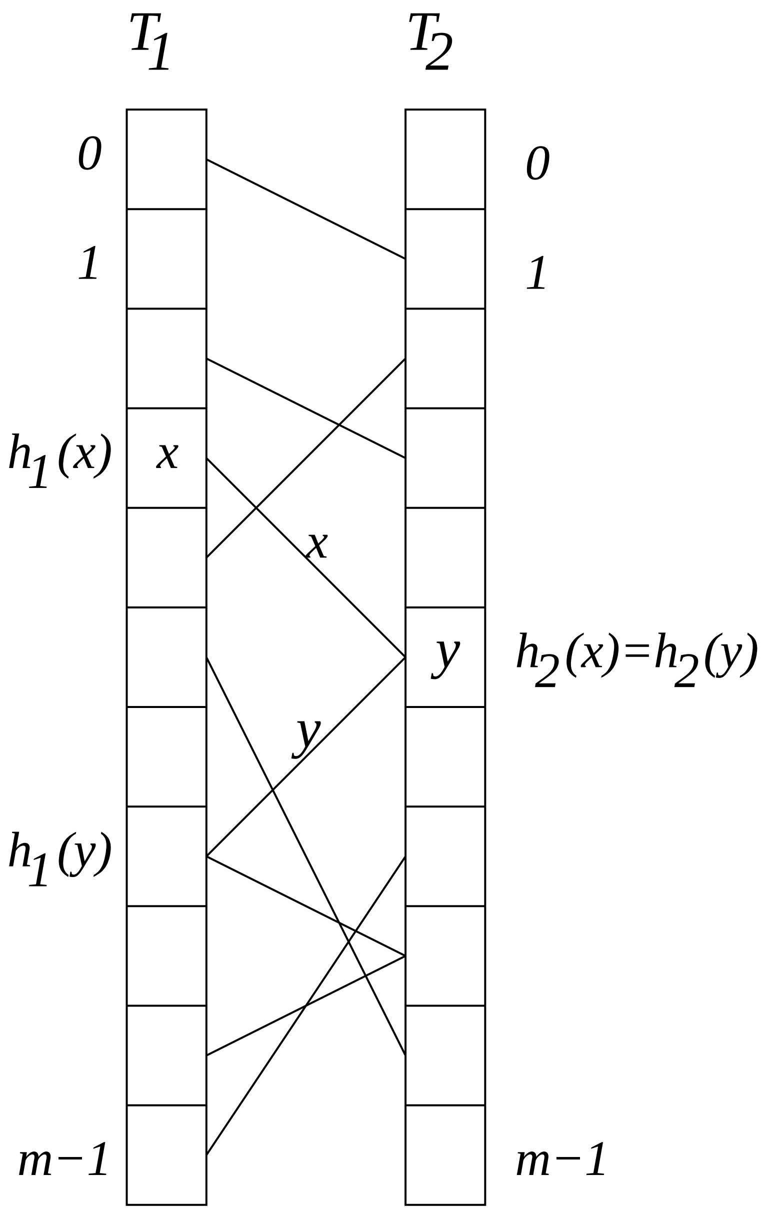
If f_1, f_2, g are 2ℓ-universal, and $m \geq (1+\varepsilon)n$, then

$$\begin{aligned} &\Pr(G(S,h_1,h_2) \text{ has c. c. with cyclomatic number } \geq q) \\ &= \; O(n/r^\ell) + O(n^{1-q}). \end{aligned}$$

(For random graphs with the same edge density:

$\ldots = O(n^{1-q})$.)

Cuckoo hashing [Pagh/Rodler 2001]



Implementation of dynamic dictionary:

Two tables T_1, T_2 of size m each

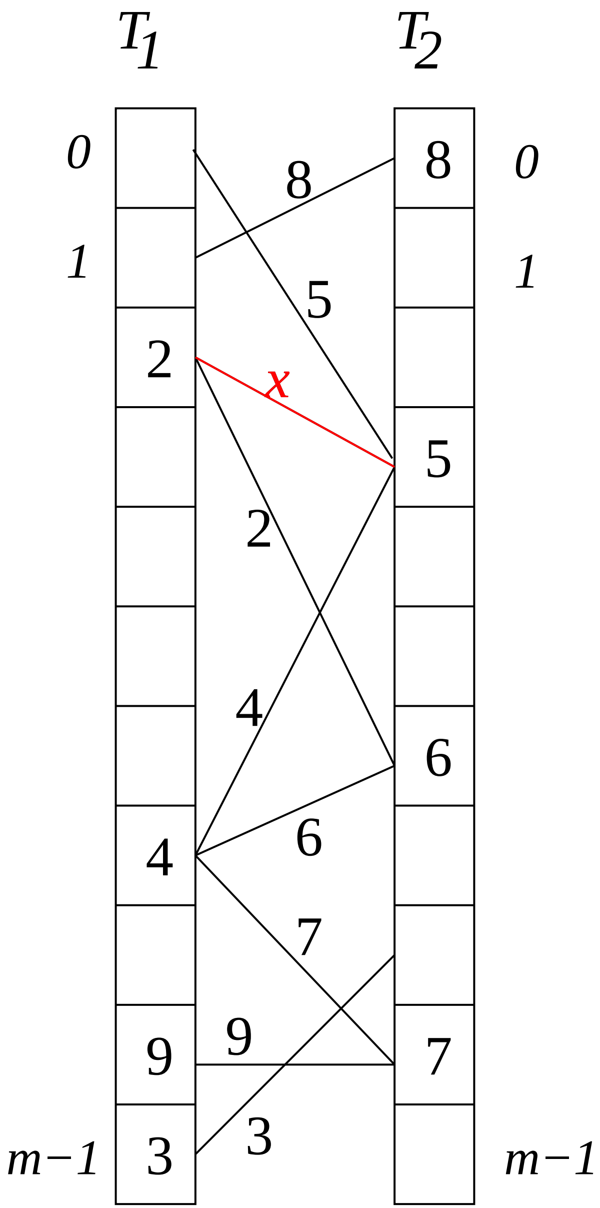
$x \in S$ may be stored in $T_1[h_1(x)]$ or in $T_2[h_2(x)]$.

$\Rightarrow$ Constant access time in the worst case.

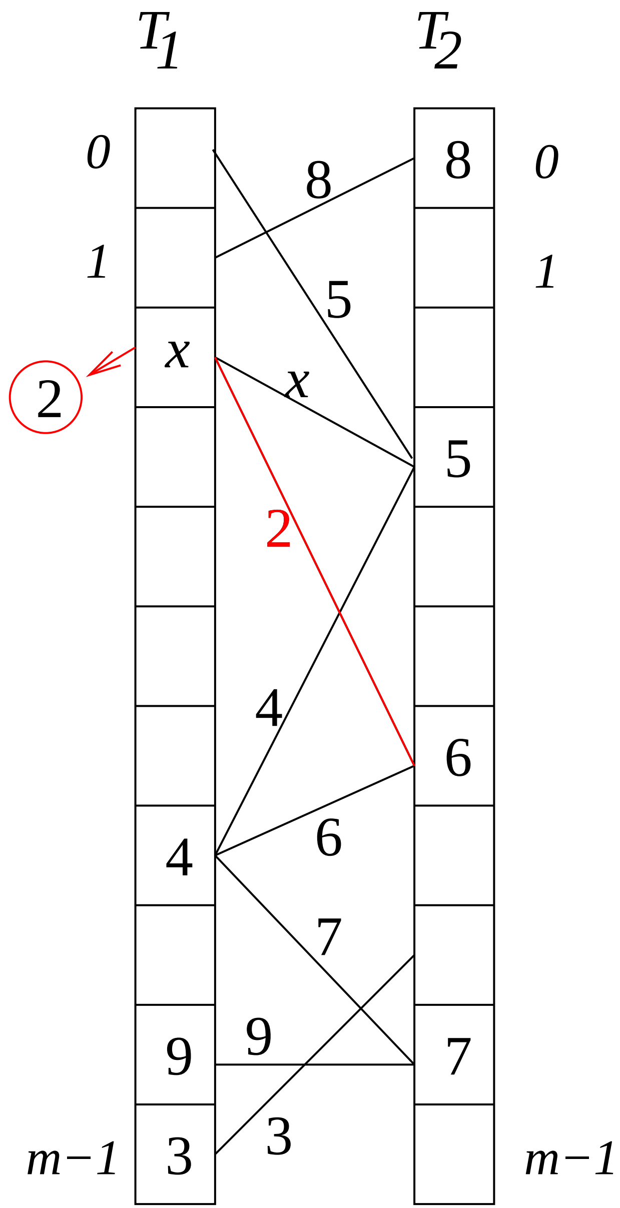
“Cuckoo hashing”

because of interesting insertion procedure.

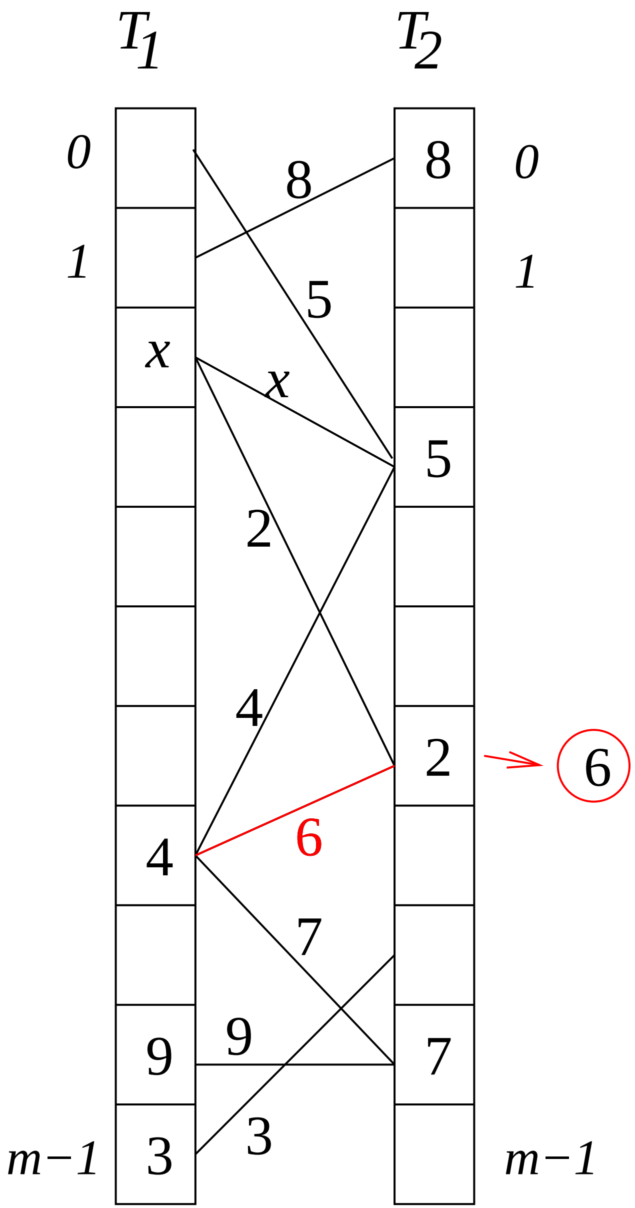
Key x that wants to be placed in the table may kick out another key y that sits in $T_1[h_1(x)]$ or $T_2[h_2(x)]$.



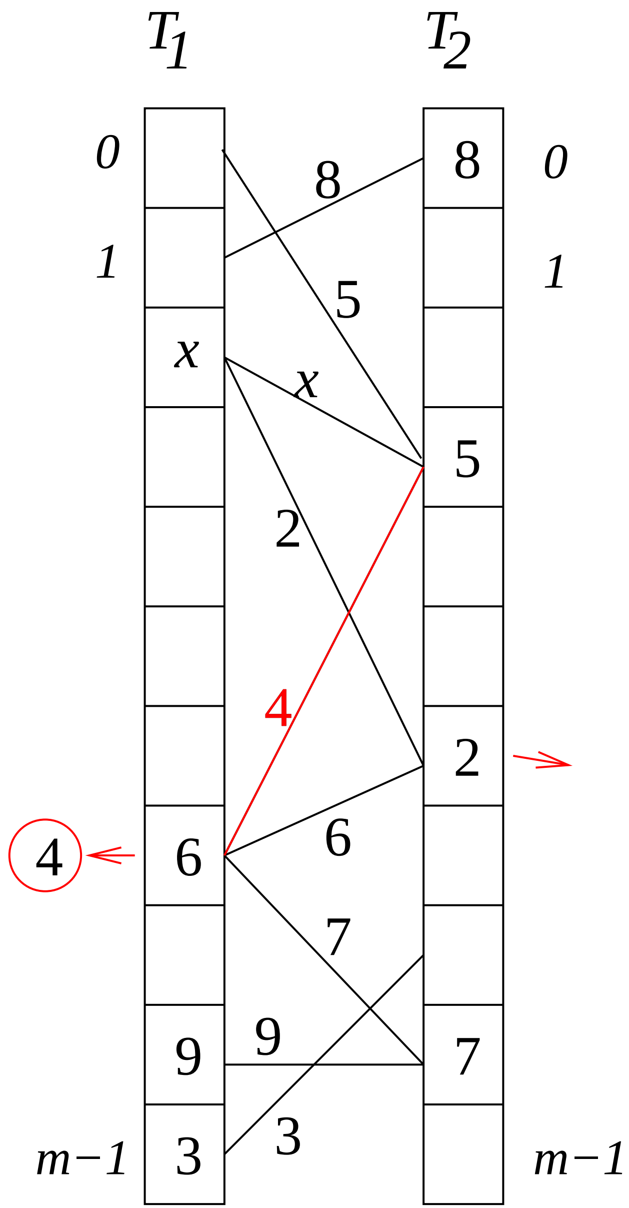
Aim: Insert x. Try $T_1[h_1(x)]$. Occupied!



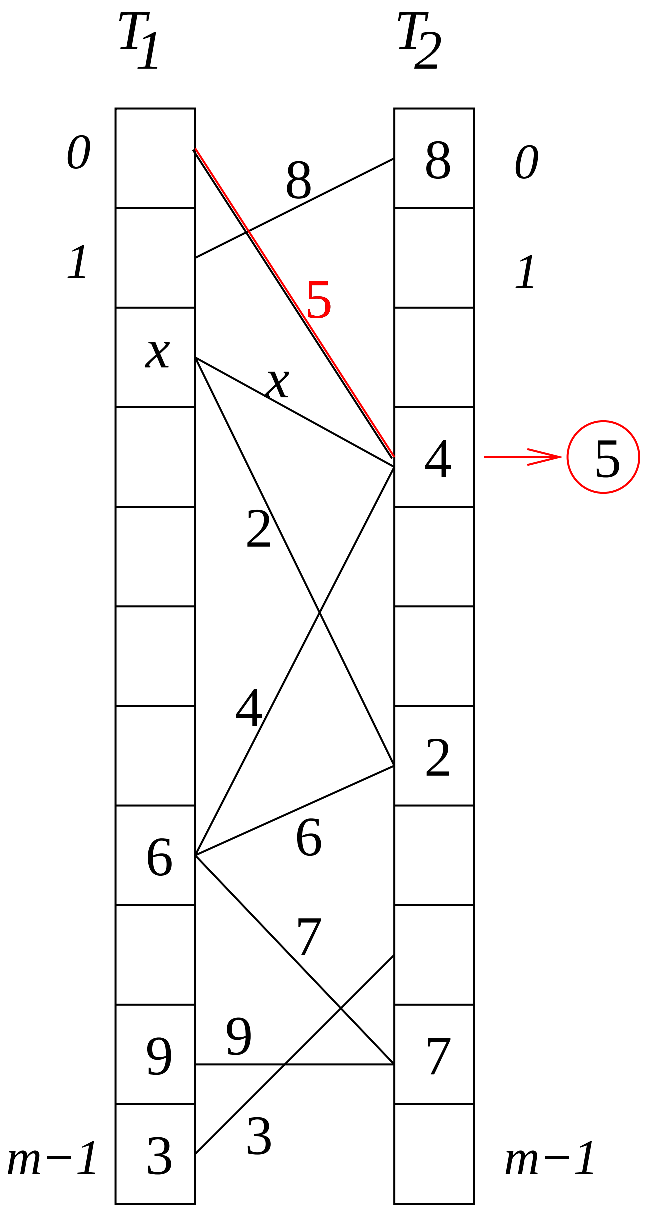
Kick out 2 from T_1. Now 2 “nestless”. $T_2[h_2(2)]$ occupied!



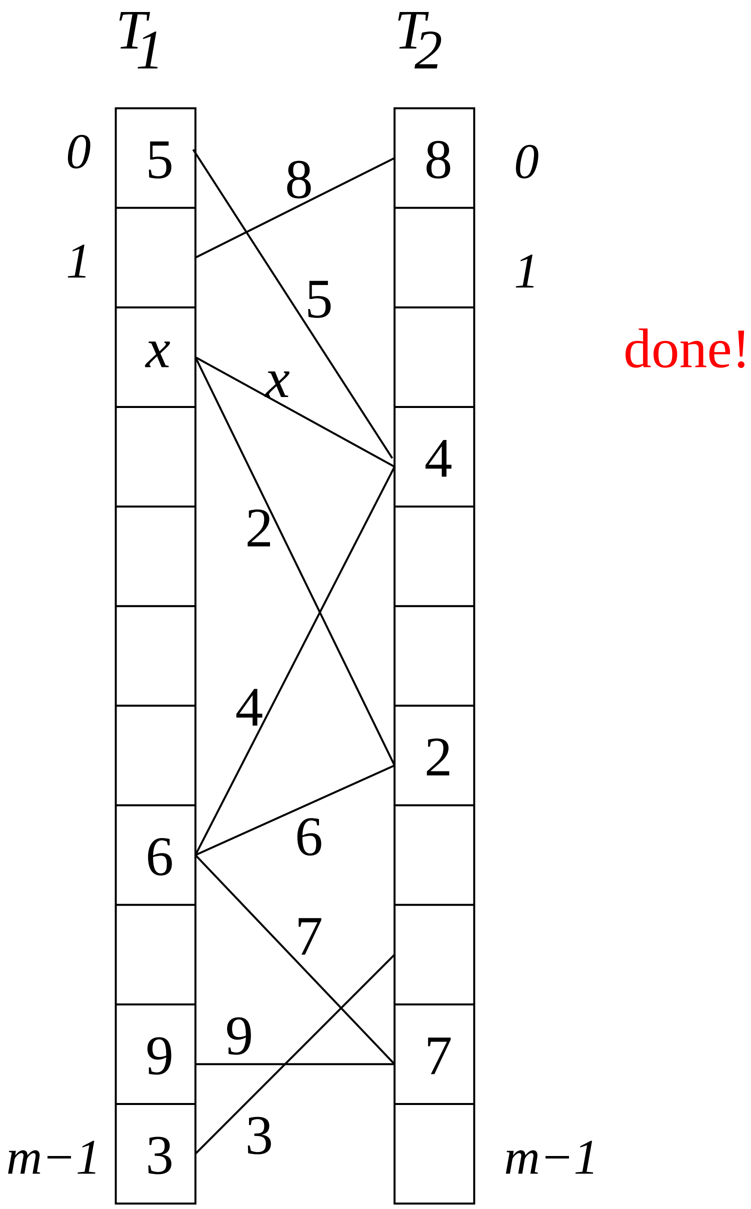
Kick out 6 from T_2. Now 6 “nestless”. $T_1[h_1(6)]$ occupied!



Kick out 4 from T_1. Now 4 “nestless”. $T_2[h_2(4)]$ occupied!



Kick out 5 from T_2. Now 5 "nestless". $T_1[h_1(5)]$ empty!



Place 5 in $T_1[h_1(5)]$.

Original analysis [PR01]:

If $S \subseteq U$ is the set of keys in the table, $|S| = n$, and

- $m \geq (1+\varepsilon)n$ and
- h_1, h_2 are from a $c \log n$-universal class,
  $c > 0$ constant, sufficiently large,

then

- with probability $1 - O(\frac{1}{n})$ all S may be stored as required
  (obstructing: connected component with cyclomatic number ≥ 2)
- a single insertion attempt succeeds with probability $1 - O(\frac{1}{n^2})$
  within $O(\log n)$ kick-out moves; the expected number of kick-out moves is constant.

If something goes wrong: start anew with new hash functions.

Drawback:

Need strong randomness assumptions about h_1, h_2:

$$c \log n\text{-universality.}$$

($c > 0$ constant.)

Achievable with polynomials of degree $c \log n$ or with Siegel's class.

Solution:

Use h_1, h_2 as described above.

Under the assumption that $G(S, h_1, h_2)$ is not ℓ-bad,

the analysis of [PR01] goes through.

Essential: With probability $O(n/r^\ell) + O(1/n)$, all connected components of $G(S, h_1, h_2)$ have cyclomatic number at most 1 (at most one extra edge in addition to a spanning tree).

E.g., can use degree-3-polynomials for g, f_1, f_2 and $2r = 2n^{3/4}$ random displacements $z_j^{(1/2)}$.

Simulating uniform hashing

[Östlin/Pagh 2003]: One can initialize a data structure D that involves in essence $O(n)$ random numbers in $[t]$ so that D allows computing a hash function $h\colon U \longrightarrow [t]$, with the following property:

- D is built obliviously of the keys it will be applied to
- for each $S \subseteq U$, $|S| = n$, the probability of a "bad event" B_S in D when applied to S is $O(1/n^k)$
- under the condition that B_S does not occur,

$$h(x), x \in S,$$

  is perfectly random.

Very interesting consequences for data structures (eliminating idealizing assumptions for the analysis of many hashing procedures), balanced allocation, ...

Drawback:

Construction requires $c \log n$-universal hash classes.

Achievable with polynomials of degree $c \log n$ or with Siegel's class.

Pay with high evaluation time.

Alternative:

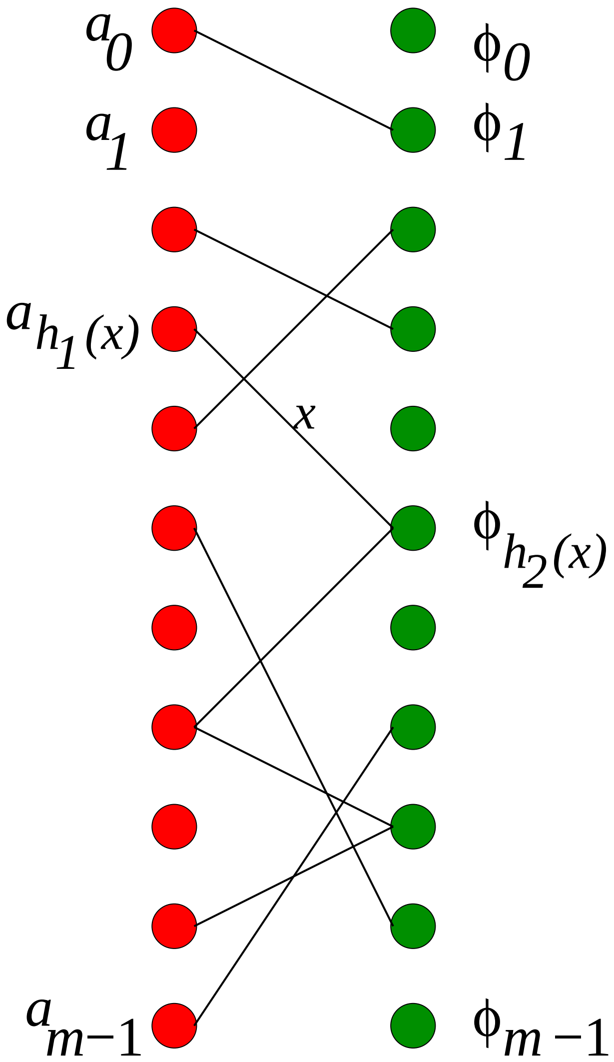
Let

$$\big(h(x) = a_{h_1(x)} + \phi_{h_2(x)}(x)\big) \bmod t,$$

where

- h_1 and h_2 are functions chosen as described above, range $[m]$ with $m \geq (1+\varepsilon)n$,
- $a_0, \ldots, a_{m-1}$ chosen at random from $[t]$,
- $\phi_0, \ldots, \phi_{m-1}$ are chosen at random from a $2q$-universal class of functions from U to $[t]$.

The labeled graph



Bipartite graph

$G(S, h_1, h_2)$

with node labels:

a_j and ϕ_j.

$$h(x) =$$
$$(a_{h_1(x)} + \phi_{h_2(x)}(x)) \bmod t$$

Theorem 4

Then, for each $S \subseteq U$, $|S| = n$, apart from a bad event B_S that has probability $O(n/r^\ell) + O(n^{1-q})$,

$$h(x), x \in S$$

is fully random on S.

Essence of proof:

For $h(x)$ to be fully random on S

it is sufficient

that no connected component of $G(S, h_1, h_2)$ has cyclomatic number $> q$.

Conclusion, Open Problems

- Graphs that behave randomly within connected components, with hash functions that are very fast to evaluate.
- Cuckoo hashing and simulation of uniform hashing with fast functions.
- What about denser graphs ($m < n$) ?
- Hypergraphs (3 or more functions)
- Analyze graphs obtained from simple d-universal hash functions.