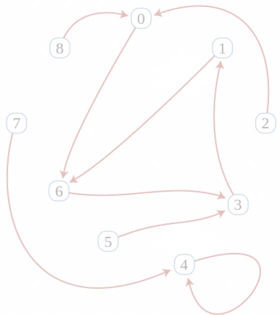
Element Distinctness, Birthday Paradox, and 1-out Pseudorandom Graphs

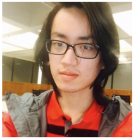


Hongxun Wu

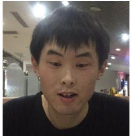
IIIS, Tsinghua University

42	3	23	1	12	30	42	15

Authors of this work



Lijie Chen



Ce Jin

R. Ryan Williams

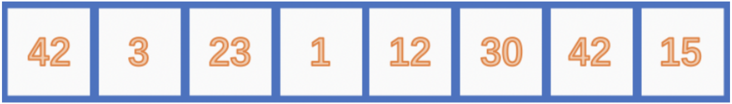
我

Hongxun Wu

Lijie Chen, Ce Jin, and R. Ryan Williams are from MIT.

Element Distinctness

Element Distinctness

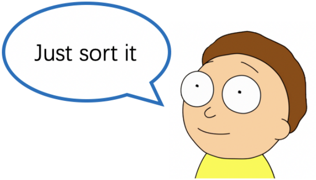


42	3	23	1	12	30	42	15

- INPUT: n positive integers $a_1, a_2, \ldots, a_n$ with $a_i \leq \text{poly}(n)$.
- Decide whether all a's are distinct.

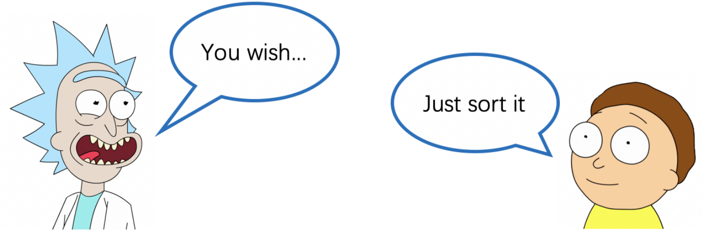
Element Distinctness

1	3	12	15	23	30	42	42

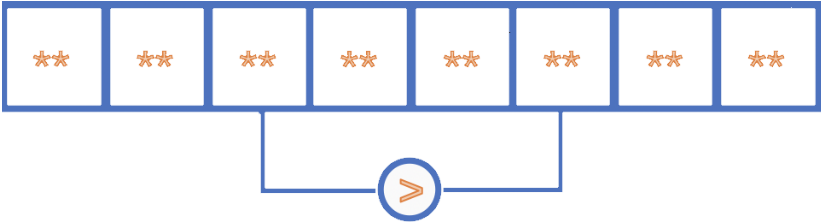


Element Distinctness

1	3	12	15	23	30	42	42

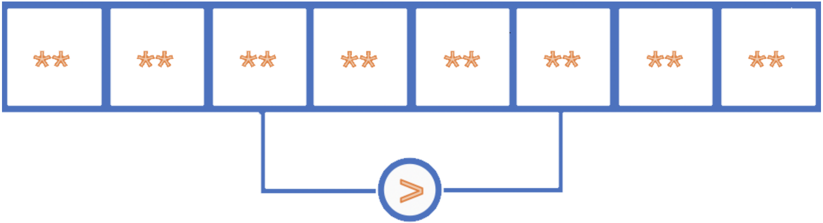


Comparision model



- No direct access to the INPUT a.
- Each query (i, j) returns one of $a_i < a_j$, $a_i = a_j$, $a_i > a_j$.

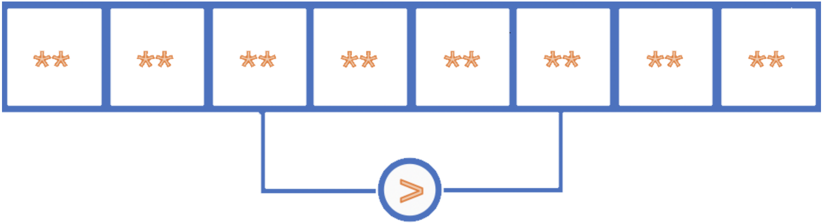
Comparision model



Time-Space tradeoff [BFMADH$^+$87, Yao88]

Element distictness requires $TS = \Omega\left(n^{2-o(1)}\right)$ in Comparision model.

Comparision model

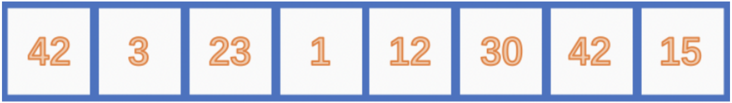


Time-Space tradeoff [BFMADH$^+$87, Yao88]

Element distictness requires $TS = \Omega\left(n^{2-o(1)}\right)$ in Comparision model.

- When $S = O(\text{polylog}\, n)$, $T = \Omega\left(n^{2-o(1)}\right)$.

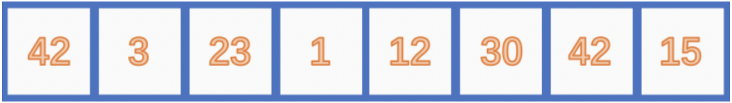
RAM model



42	3	23	1	12	30	42	15

- Random access to read-only input.
- Working memory has a (relatively small) size S.

RAM model

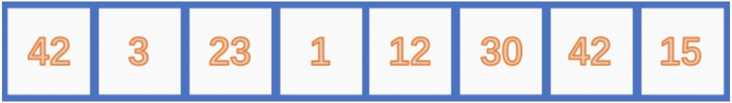


42	3	23	1	12	30	42	15

Time-Space tradeoff [BCM13]

- Assuming the existence of *Random Oracle*, there is an algorithm with $T^2S = \tilde{O}\left(n^3\right)$.

RAM model



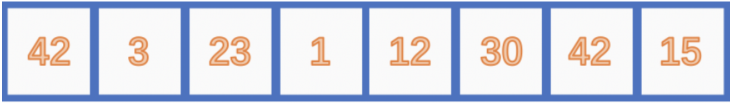
42	3	23	1	12	30	42	15

Time-Space tradeoff [BCM13]

- Assuming the existence of *Random Oracle*, there is an algorithm with $T^2 S = \tilde{O}\left(n^3\right)$.

- When $S = \tilde{O}(1)$, $T = \tilde{O}\left(n^{1.5}\right)$.

RAM model



42	3	23	1	12	30	42	15

Time-Space tradeoff [BCM13]

- Assuming the existence of *Random Oracle*, there is an algorithm with $T^2 S = \tilde{O}\left(n^3\right)$.

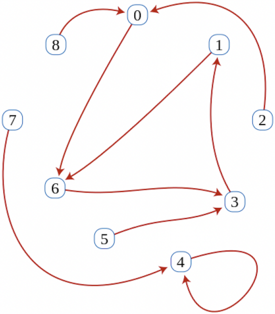
- When $S = \tilde{O}(1)$, $T = \tilde{O}\left(n^{1.5}\right)$.
- In the rest of this talk, we always assume there is only one collision $(a_i = a_j)$.

1-out Graph and Birthday Paradox

Pollard's ρ method [BCM13]

Assuming the existence of *Random Oracle*, when $S = \tilde{O}(1)$, there is an algorithm with $T = \tilde{O}\left(n^{1.5}\right)$.

- For random oracle R, define graph $x \mapsto R(a_x)$ with $x \in [n]$.

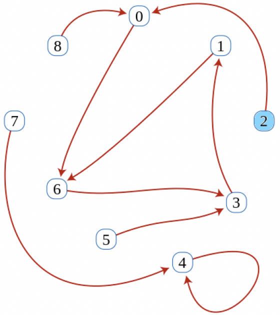


1-out Graph and Birthday Paradox

Pollard's ρ method [BCM13]

Assuming the existence of *Random Oracle*, when $S = \tilde{O}(1)$, there is an algorithm with $T = \tilde{O}\left(n^{1.5}\right)$.

- For random oracle R, define graph $x \mapsto R(a_x)$ with $x \in [n]$.
- Pick a random starting point s.

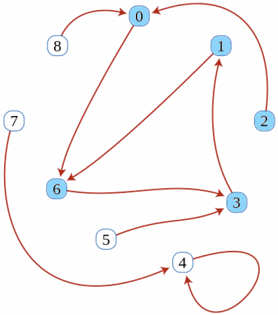


1-out Graph and Birthday Paradox

Pollard's ρ method [BCM13]

Assuming the existence of *Random Oracle*, when $S = \tilde{O}(1)$, there is an algorithm with $T = \tilde{O}\left(n^{1.5}\right)$.

- For random oracle R, define graph $x \mapsto R(a_x)$ with $x \in [n]$.
- Pick a random starting point s.
- Run Floyd's cycle finding.

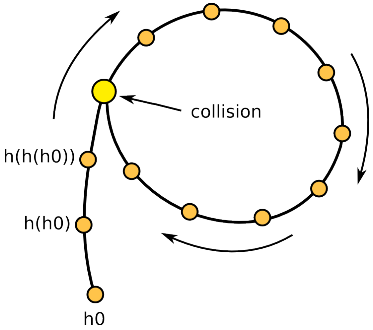


1-out Graph and Birthday Paradox

Birthday Paradox Type Properties [BCM13]

Suppose $f^*(s)$ is the set of vertices reachable from s.

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$

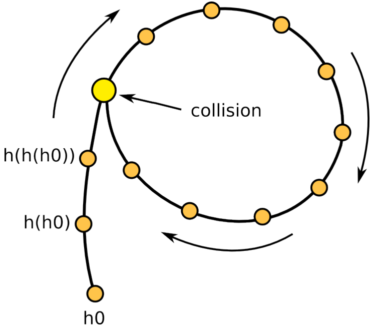


1-out Graph and Birthday Paradox

Birthday Paradox Type Properties [BCM13]

Suppose $f^*(s)$ is the set of vertices reachable from s.

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u, v \in f^*(s)] \geq \Omega(1/n), \ \forall u, v \in [n]$



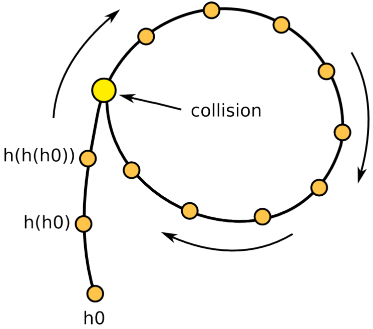
1-out Graph and Birthday Paradox

Birthday Paradox Type Properties [BCM13]

Suppose $f^*(s)$ is the set of vertices reachable from s.

- $\mathbb{E}[|f^*(s)|] \le O(\sqrt{n})$
- $\Pr[u, v \in f^*(s)] \ge \Omega(1/n), \ \forall u, v \in [n]$

- So each cycle-finding takes $O(\sqrt{n})$ time and finds any collision u, v with probability $\Omega(1/n)$.



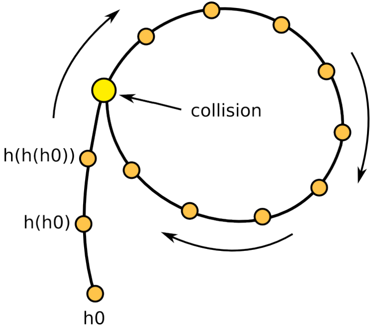
1-out Graph and Birthday Paradox

Birthday Paradox Type Properties [BCM13]

Suppose $f^*(s)$ is the set of vertices reachable from s.

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u, v \in f^*(s)] \geq \Omega(1/n), \ \forall u, v \in [n]$

- So each cycle-finding takes $O(\sqrt{n})$ time and finds any collision u, v with probability $\Omega(1/n)$.
- Repeat $O(n)$ times. It takes $O(n^{1.5})$ time in total.



Our Results

Our Main Lemma

There exsits a family $\{r_{\text{seed}}\}$ of hash functions efficiently samplable with seed length $O(\operatorname{polylog} n)$, and the graph defined by $\{r_{\text{seed}}\}$ (instead of *Random Oracle R*) satisfy

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$

Our Results

Our Main Lemma

There exsits a family $\{r_{\text{seed}}\}$ of hash functions efficiently samplable with seed length $O(\text{polylog}\, n)$, and the graph defined by $\{r_{\text{seed}}\}$ (instead of *Random Oracle R*) satisfy

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u, v \in f^*(s)] \geq \Omega(1/n), \ \forall u, v \in [n]$

Our Results

Our Main Lemma

There exsits a family $\{r_{\text{seed}}\}$ of hash functions efficiently samplable with seed length $O(\operatorname{polylog} n)$, and the graph defined by $\{r_{\text{seed}}\}$ (instead of *Random Oracle* R) satisfy

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u, v \in f^*(s)] \geq \Omega(1/n), \ \forall u, v \in [n]$

Our Result

~~Assuming the existence of *Random Oracle*~~, when $S = O(\operatorname{polylog} n)$, there is a RAM algorithm for Element Distinctness with $T = \tilde{O}\left(n^{1.5}\right)$.

Subset Sum

Low-space Algorithm for Subset Sum [BGNV18]

Assuming the existence of *Random Oracle*, Subset Sum and Knapsack can be solved by a Monte Carlo algorithm in $O^*(2^{0.86n})$ time, with $O(\text{poly}(n))$ space.

Subset Sum

Low-space Algorithm for Subset Sum [BGNV18]

Assuming the existence of *Random Oracle*, Subset Sum and Knapsack can be solved by a Monte Carlo algorithm in $O^*(2^{0.86n})$ time, with $O(\text{poly}(n))$ space.

Our Result

~~Assuming the existence of *Random Oracle*~~, Subset Sum and Knapsack can be solved by a Monte Carlo algorithm in $O^*(2^{0.86n})$ time, with $O(\text{poly}(n))$ space.

Our Results

Our Main Lemma

There exsits a family $\{r_{\text{seed}}\}$ of hash functions efficiently samplable with seed length $O(\text{polylog}\, n)$, and the graph defined by $\{r_{\text{seed}}\}$ (instead of *Random Oracle* R) satisfy

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u, v \in f^*(s)] \geq \Omega(1/n), \ \forall u, v \in [n]$

Our Results

Our Main Lemma

There exsits a family $\{r_{\text{seed}}\}$ of hash functions efficiently samplable with seed length $O(\text{polylog}\, n)$, and the graph defined by $\{r_{\text{seed}}\}$ (instead of *Random Oracle* R) satisfy

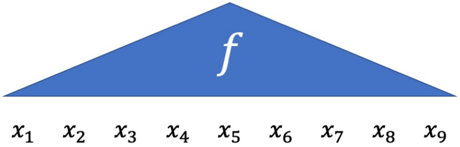
- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \ \forall u \in [n]$

Construction

Random Restriction and Håstad's Switching Lemma

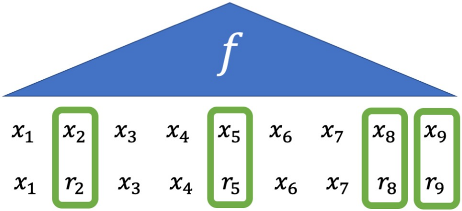
This is Ryan O'Donnell's Youtube lecture which is a masterpiece.

Iterative Restriction



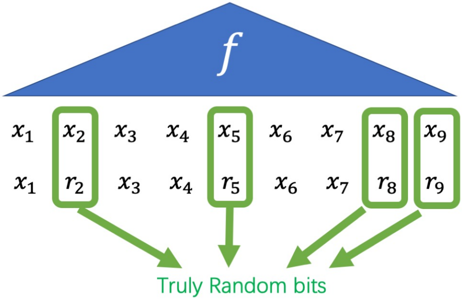
This is the Ajtai-Wigderson Paradigm [AW85] for building PRG.

Iterative Restriction



This is the Ajtai-Wigderson Paradigm [AW85] for building PRG.

Iterative Restriction



f
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
x_1 r_2 x_3 x_4 r_5 x_6 x_7 r_8 r_9
Truly Random bits

This is the Ajtai-Wigderson Paradigm [AW85] for building PRG.

Toy Example: Two levels

Recall the input $a_1, a_2, \ldots, a_n \in [m]$.

Two Level Example

Suppose we have the following:

- $O(\text{polylog } n)$-wise independent functions $g : [m] \to \{0, 1\}$ and $r : [m] \to [n]$.
- Random Oracle R.

Toy Example: Two levels

Recall the input $a_1, a_2, \ldots, a_n \in [m]$.

Two Level Example

Suppose we have the following:

- $O(\text{polylog}\, n)$-wise independent functions $g : [m] \to \{0, 1\}$ and $r : [m] \to [n]$.
- Random Oracle R.

We define the graph $x \mapsto h(a_x)$ with

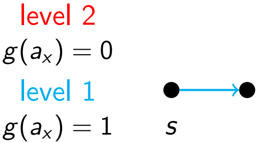
$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

Toy Example: Two levels

Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$



level 2

$g(a_x) = 0$

level 1

$g(a_x) = 1$

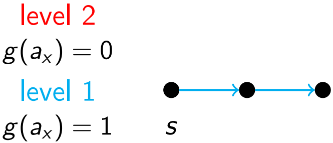
s

Toy Example: Two levels

Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

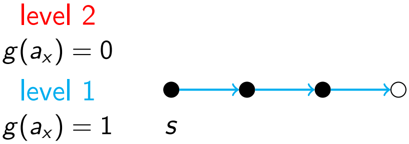


Toy Example: Two levels

Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

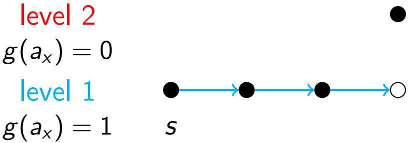


Toy Example: Two levels

Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

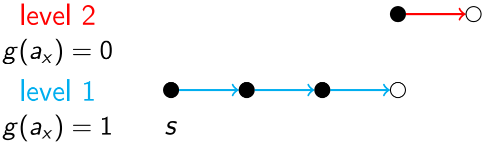


Toy Example: Two levels

Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$



level 2

$g(a_x) = 0$

level 1

$g(a_x) = 1$

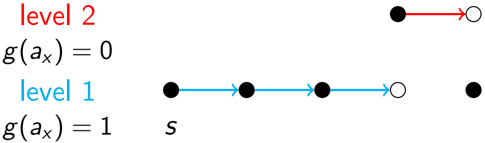
s

Toy Example: Two levels

Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

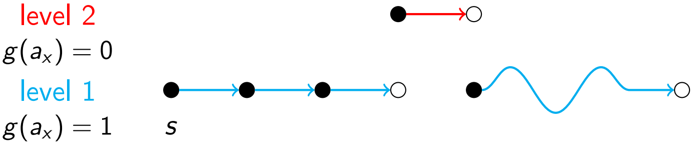


Toy Example: Two levels

Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$



level 2

$g(a_x) = 0$

level 1

$g(a_x) = 1$

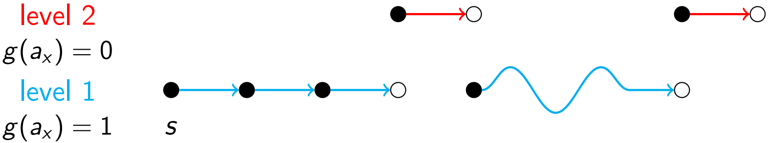
s

Toy Example: Two levels

Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$



level 2

$g(a_x) = 0$

level 1

$g(a_x) = 1$

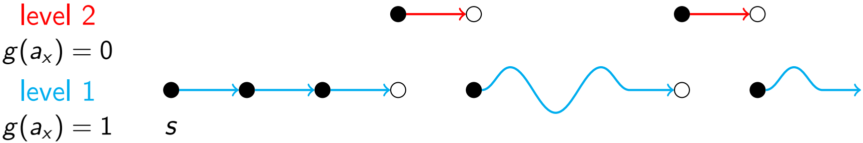
s

Toy Example: Two levels

Two Level Example

We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

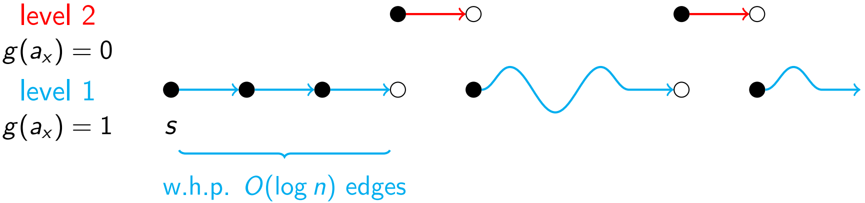


Toy Example: Two levels

Two Level Example

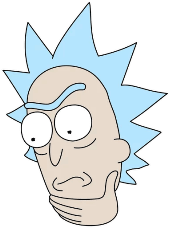
We define the graph $x \mapsto h(a_x)$ with

$$h(a_x) = \begin{cases} R(a_x) & g(a_x) = 0 \\ r(a_x) & g(a_x) = 1 \end{cases}$$

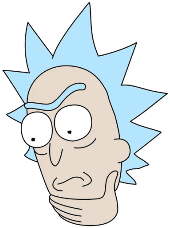


Sanity Check

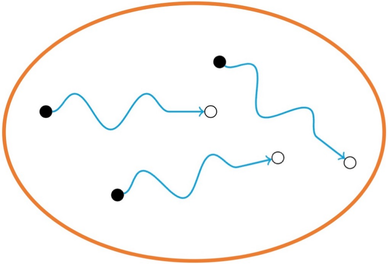
- Why this might be a good idea?



Sanity Check

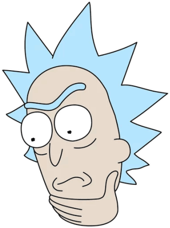


- Why this might be a good idea?

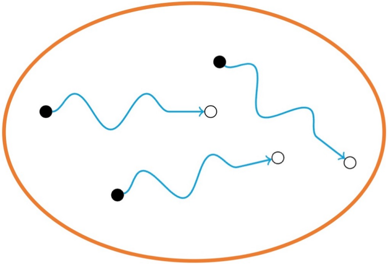


- Each subpath has length $O(\log n)$.

Sanity Check



- Why this might be a good idea?



- Each subpath has length $O(\log n)$.
- Every level 2 edge is an independent sample of a subpath.

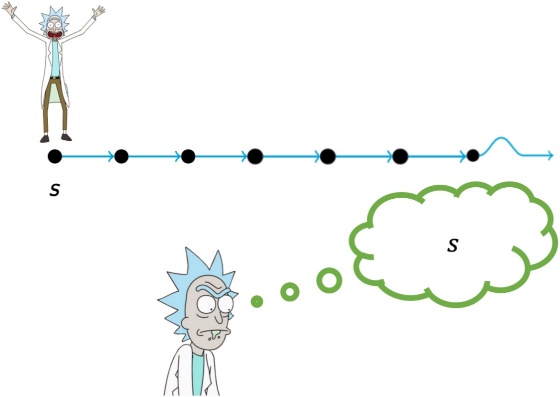
Intuition: Memory Eraser

Recall our goal.

Our Main Lemma

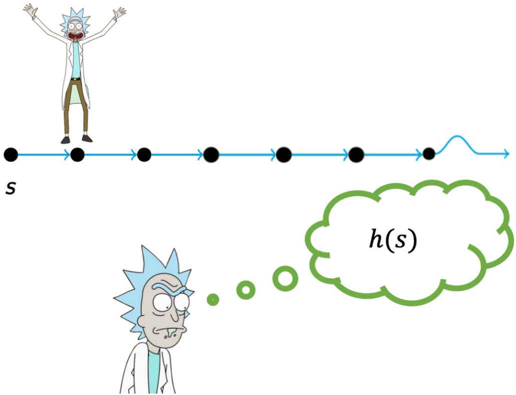
- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \ \forall u \in [n]$

Intuition: Memory Eraser



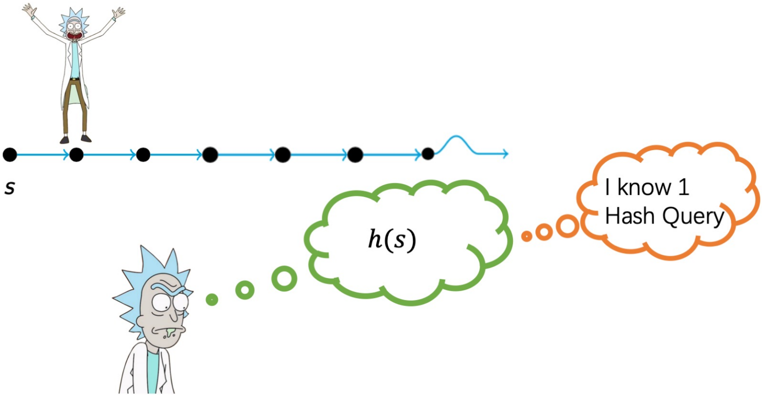
- “Memory” of a random walk: The current vertex it is at.

Intuition: Memory Eraser



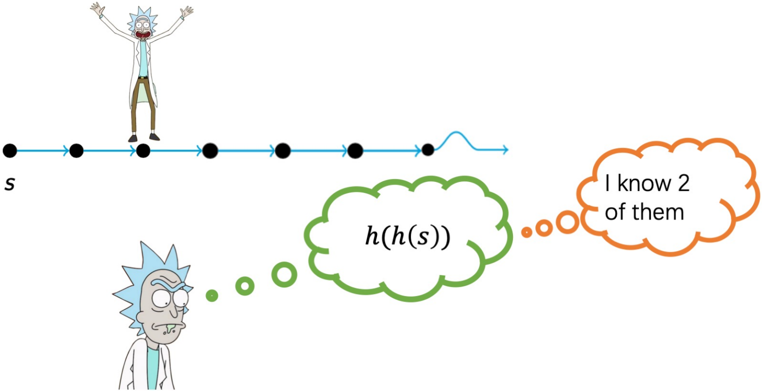
- "Memory" of a random walk: The current vertex it is at.

Intuition: Memory Eraser



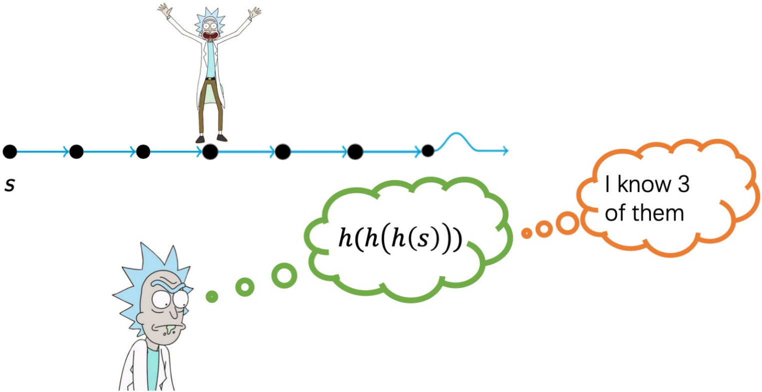
- “Memory” of a random walk: The current vertex it is at.

Intuition: Memory Eraser



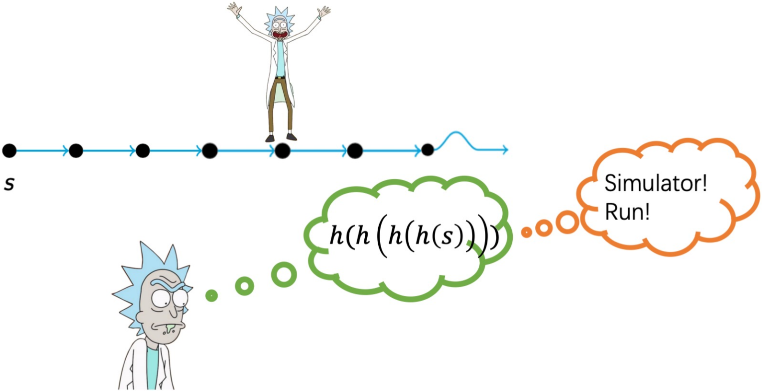
- "Memory" of a random walk: The current vertex it is at.

Intuition: Memory Eraser



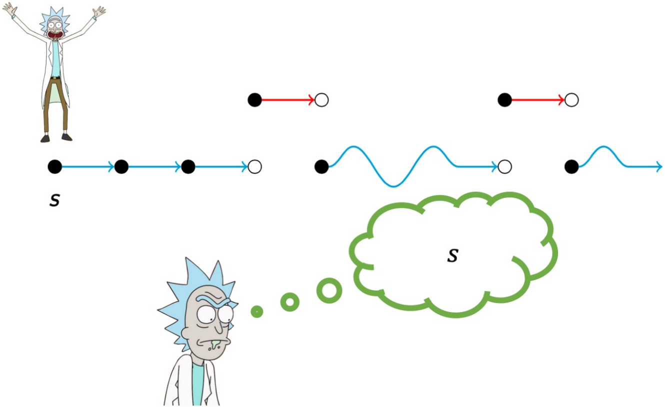
- "Memory" of a random walk: The current vertex it is at.

Intuition: Memory Eraser



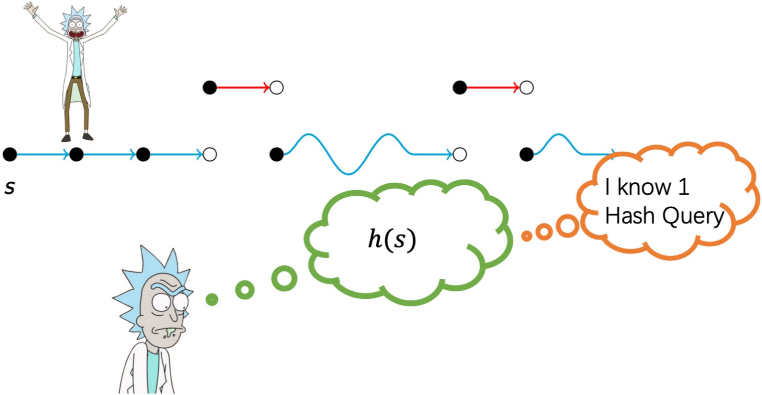
- “Memory” of a random walk: The current vertex it is at.

Intuition: Memory Eraser



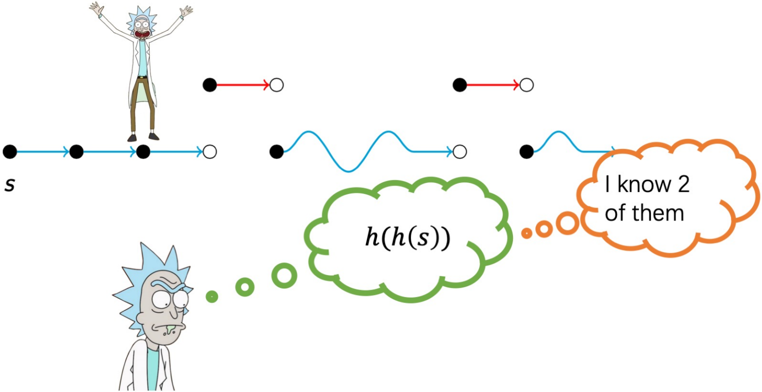
- "Memory" of a random walk: The current vertex it is at.

Intuition: Memory Eraser



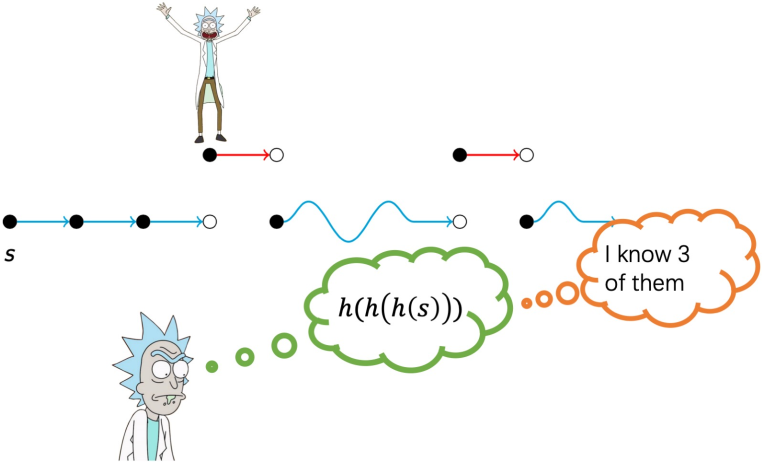
- "Memory" of a random walk: The current vertex it is at.

Intuition: Memory Eraser

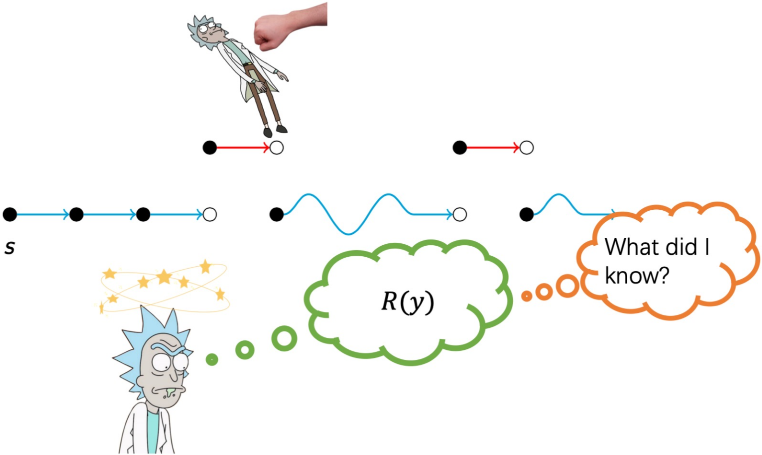


- “Memory” of a random walk: The current vertex it is at.

Intuition: Memory Eraser



- "Memory" of a random walk: The current vertex it is at.



- "Memory" of a random walk: The current vertex it is at.

Our Construction via Iterative Restriction

Our Construction

Now we sample $O(\log n)$ many hash functions $\{r_i, g_i\}_{i \in [\ell]}$.
Each $r_i : [m] \to [n]$ and $g_i : [m] \to [2]$ are $O(\log n)$-wise independent.

Our Construction via Iterative Restriction

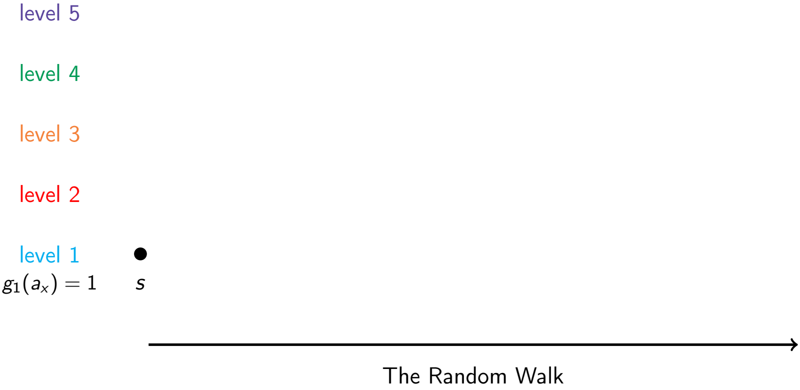
Our Construction

Now we sample $O(\log n)$ many hash functions $\{r_i, g_i\}_{i\in[\ell]}$.
Each $r_i : [m] \to [n]$ and $g_i : [m] \to [2]$ are $O(\log n)$-wise independent.
Then we set $h_{\ell+1}(a_x) = \perp$ and

$$h_i(a_x) = \begin{cases} h_{i+1}(a_x) & g_i(a_x) = 0 \\ r_i(a_x) & g_i(a_x) = 1 \end{cases}$$

Finally, we set $h = h_1$.

Our Construction via Iterative Restriction



level 5

level 4

level 3

level 2

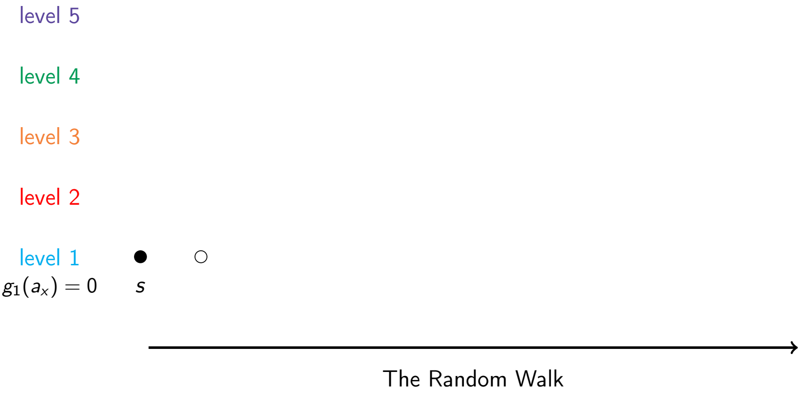
level 1

$g_1(a_x) = 1$

s

The Random Walk

Our Construction via Iterative Restriction



level 5

level 4

level 3

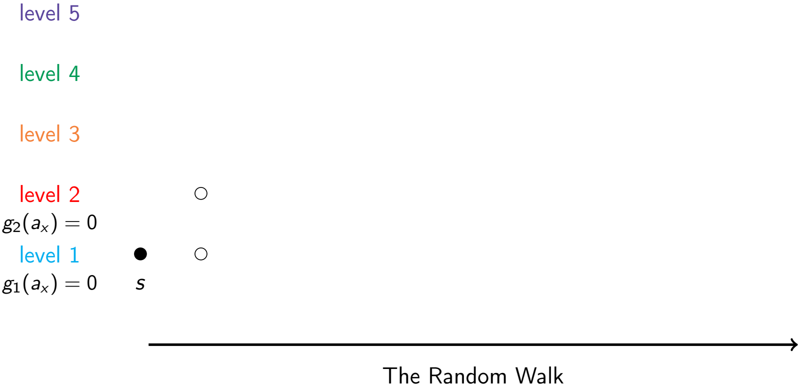
level 2

level 1

$g_1(a_x) = 0$ s

The Random Walk

Our Construction via Iterative Restriction



level 5

level 4

level 3

level 2

$g_2(a_x) = 0$

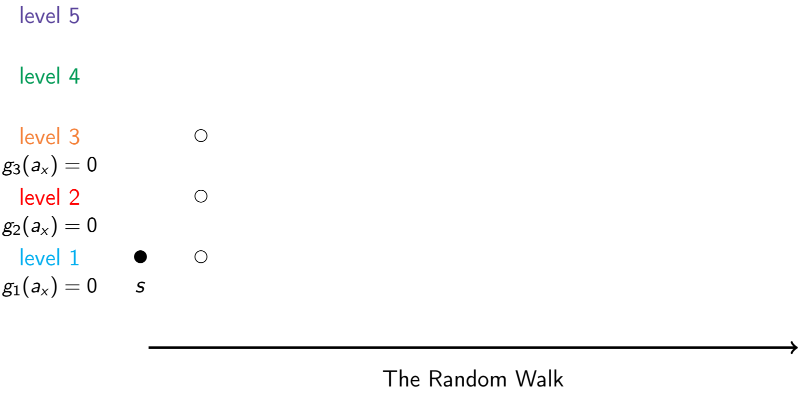
level 1

$g_1(a_x) = 0$

s

The Random Walk

Our Construction via Iterative Restriction



level 5

level 4

level 3

$g_3(a_x) = 0$

level 2

$g_2(a_x) = 0$

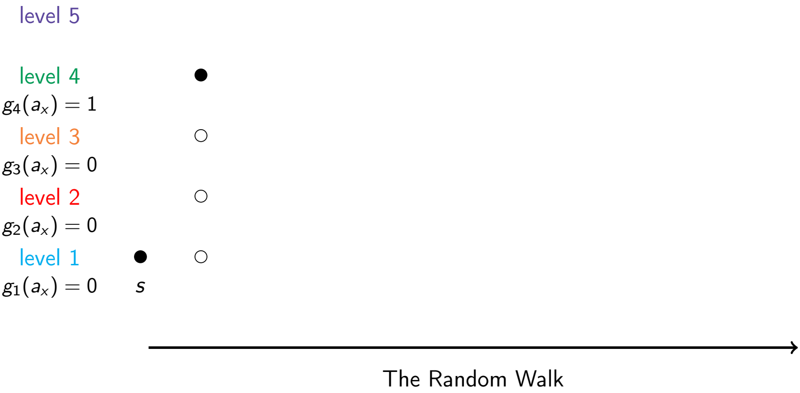
level 1

$g_1(a_x) = 0$

s

The Random Walk

Our Construction via Iterative Restriction



level 5

level 4

$g_4(a_x) = 1$

level 3

$g_3(a_x) = 0$

level 2

$g_2(a_x) = 0$

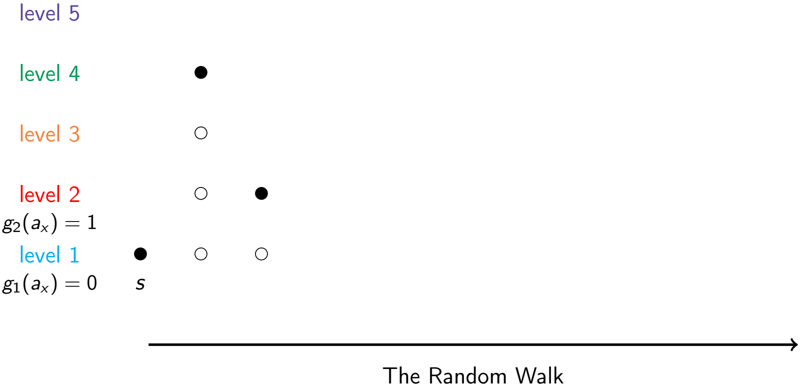
level 1

$g_1(a_x) = 0$

s

The Random Walk

Our Construction via Iterative Restriction



level 5

level 4

level 3

level 2

$g_2(a_x) = 1$

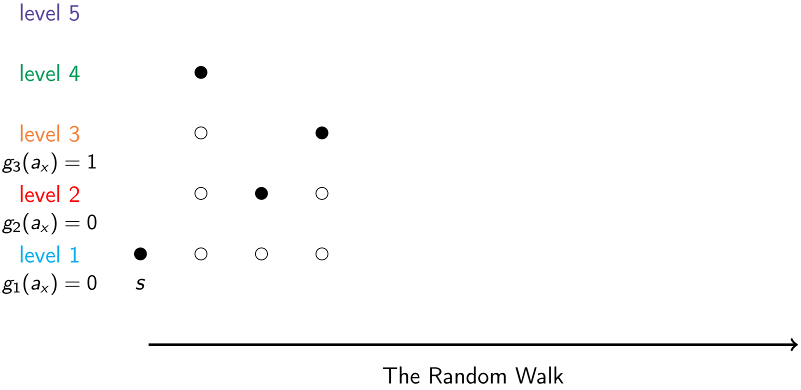
level 1

$g_1(a_x) = 0$

s

The Random Walk

Our Construction via Iterative Restriction



level 5

level 4

level 3

$g_3(a_x) = 1$

level 2

$g_2(a_x) = 0$

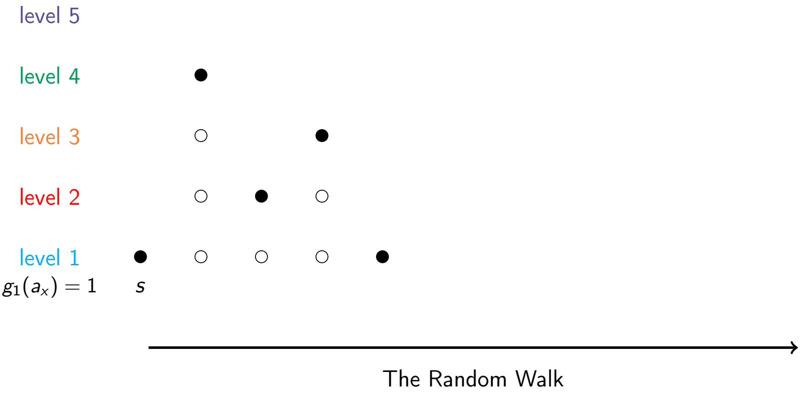
level 1

$g_1(a_x) = 0$

s

The Random Walk

Our Construction via Iterative Restriction



level 5

level 4

level 3

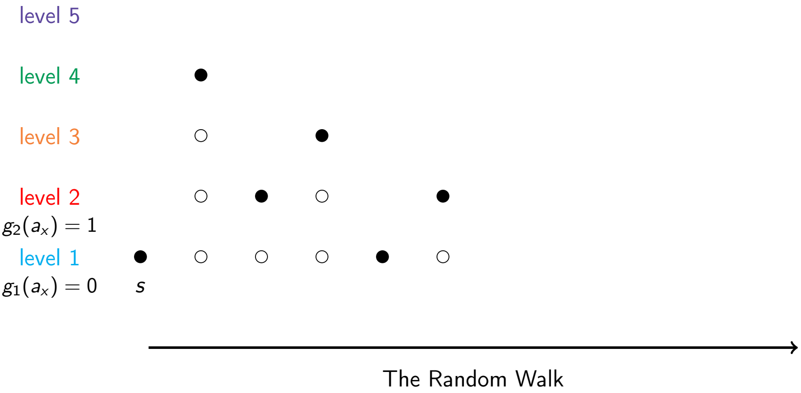
level 2

level 1

$g_1(a_x) = 1$ s

The Random Walk

Our Construction via Iterative Restriction



level 5

level 4

level 3

level 2

$g_2(a_x) = 1$

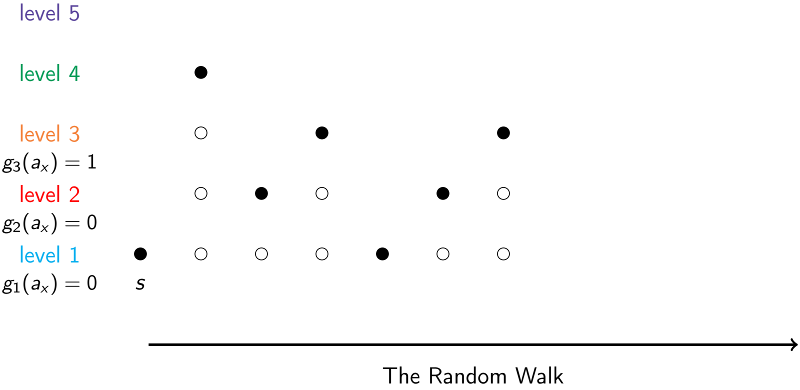
level 1

$g_1(a_x) = 0$

s

The Random Walk

Our Construction via Iterative Restriction



level 5

level 4

level 3

$g_3(a_x) = 1$

level 2

$g_2(a_x) = 0$

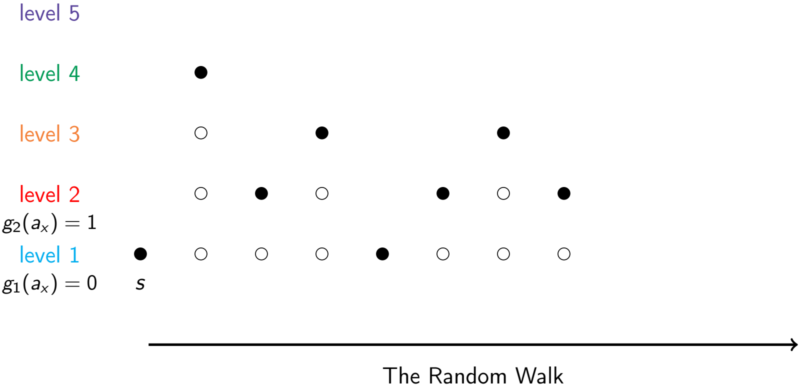
level 1

$g_1(a_x) = 0$

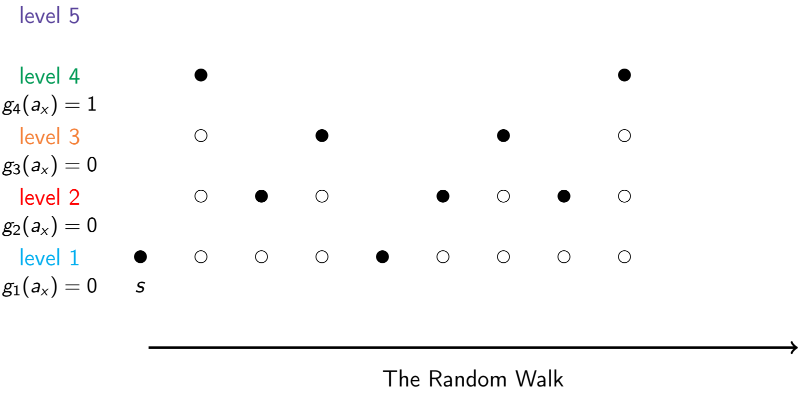
s

The Random Walk

Our Construction via Iterative Restriction



Our Construction via Iterative Restriction



level 5

level 4

$g_4(a_x) = 1$

level 3

$g_3(a_x) = 0$

level 2

$g_2(a_x) = 0$

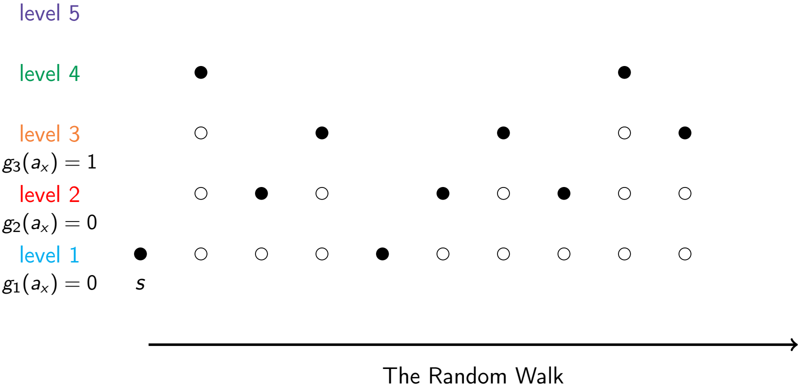
level 1

$g_1(a_x) = 0$

s

The Random Walk

Our Construction via Iterative Restriction



level 5

level 4

level 3

$g_3(a_x) = 1$

level 2

$g_2(a_x) = 0$

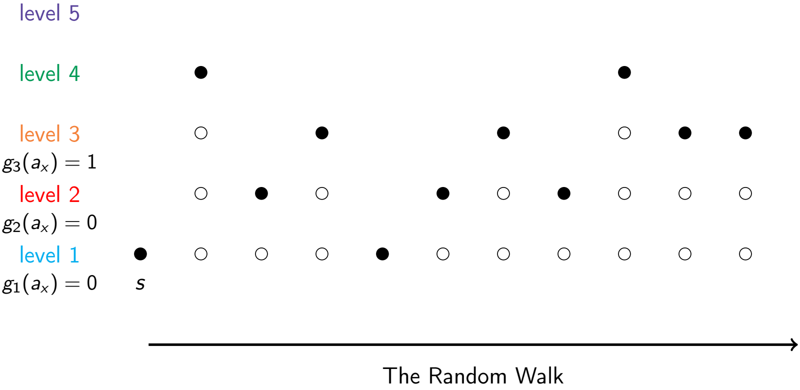
level 1

$g_1(a_x) = 0$

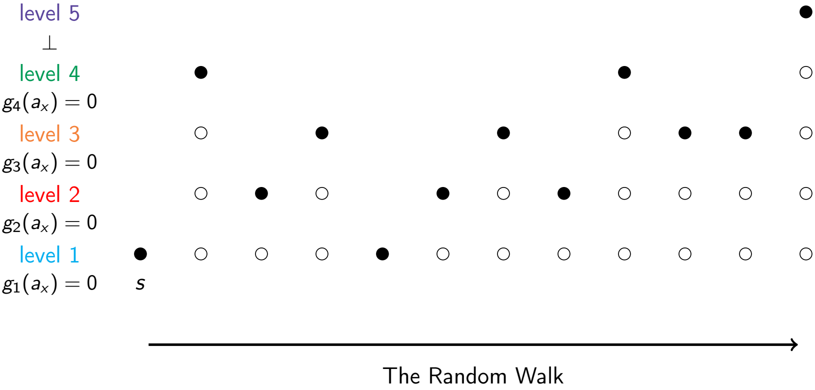
s

The Random Walk

Our Construction via Iterative Restriction

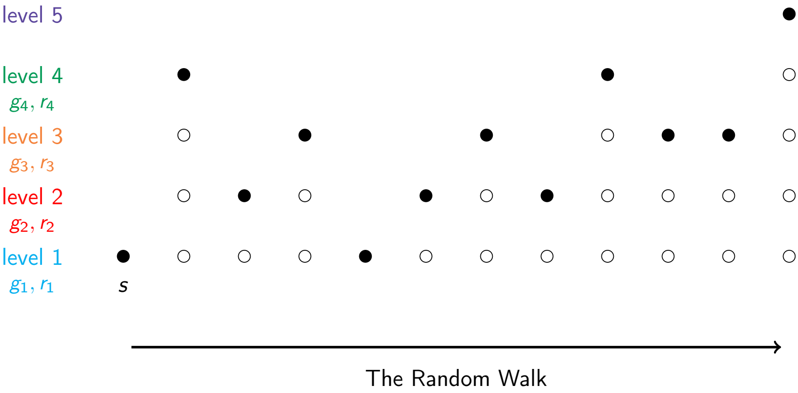


Our Construction via Iterative Restriction

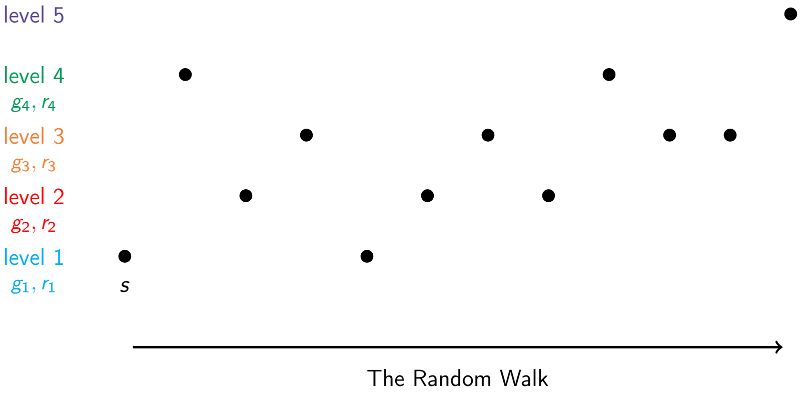


Key Ideas in Our Analysis

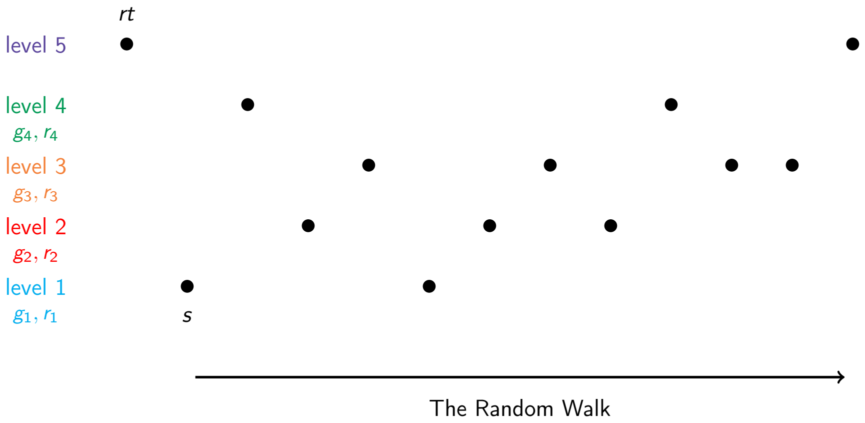
Dependency Tree



Dependency Tree



Dependency Tree



rt

level 5

level 4

g_4, r_4

level 3

g_3, r_3

level 2

g_2, r_2

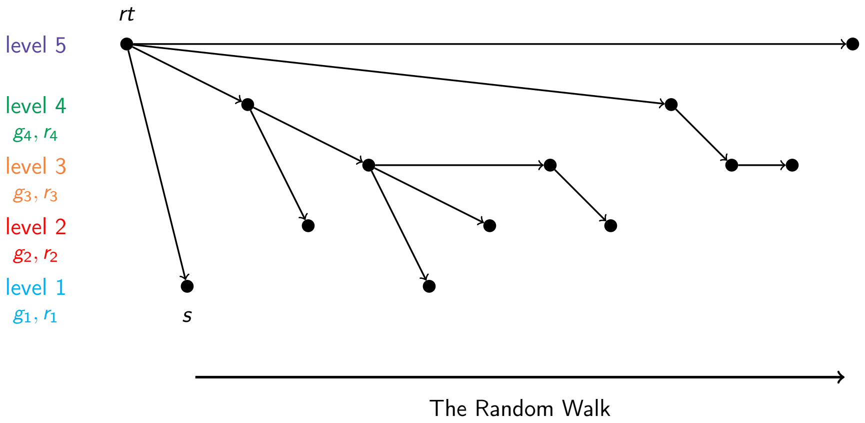
level 1

g_1, r_1

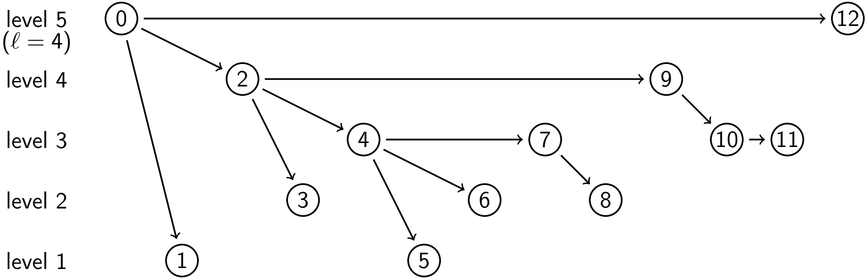
s

The Random Walk

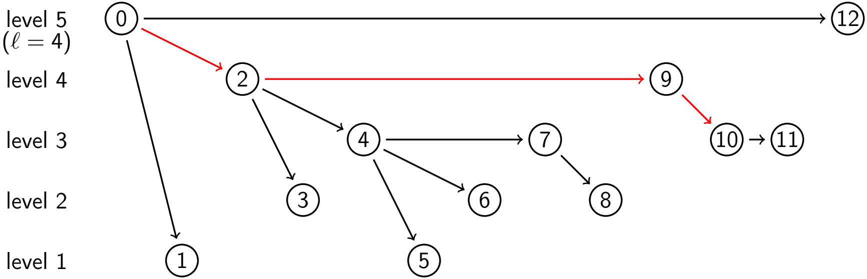
Dependency Tree



Dependency Tree

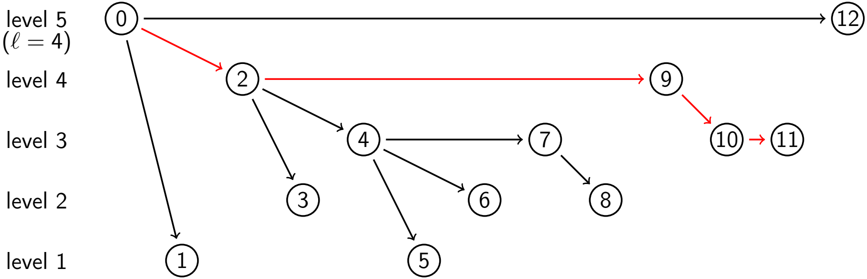


Dependency Tree



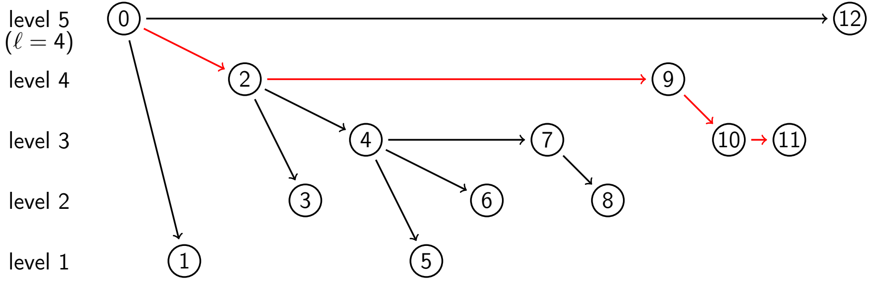
- We index a node by the shape of its path, e.g. $\vec{k}_{10} = (0, 0, 1, 2)$.

Dependency Tree



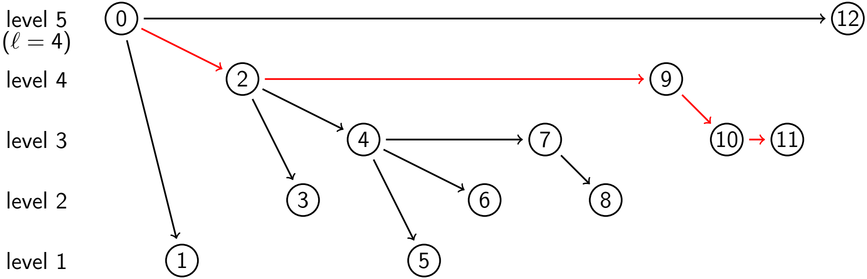
- We index a node by the shape of its path, e.g. $\vec{k}_{11} = (0, 0, 2, 2)$.

Dependency Tree



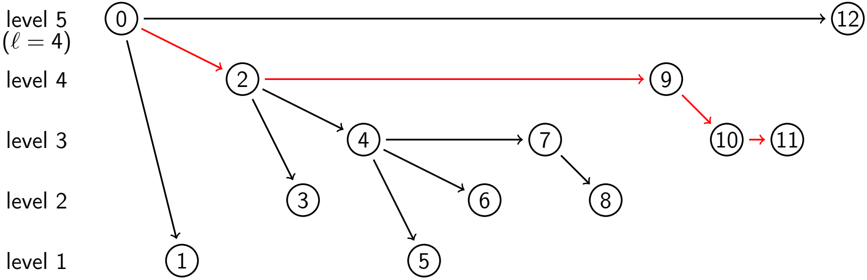
- We index a node by the shape of its path, e.g. $\vec{k}_{11} = (0, 0, 2, 2)$.
- Consider $\vec{k}_x$. Fix x, $\vec{k}$ is a random variable. Fix $\vec{k}$, x is a random variable.

Dependency Tree



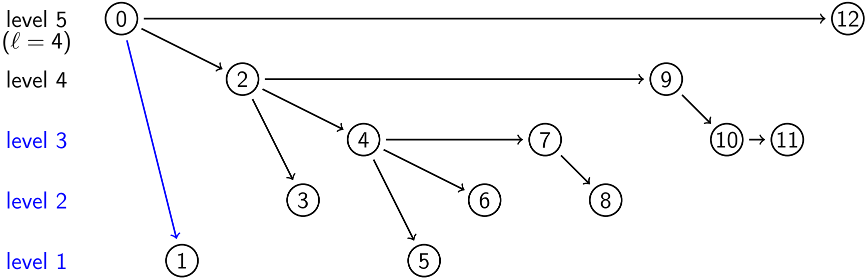
- We index a node by the shape of its path, e.g. $\vec{k}_{11} = (0, 0, 2, 2)$.
- Consider $\vec{k}_x$. Fix x, $\vec{k}$ is a random variable. Fix $\vec{k}$, x is a random variable.
- We fix index $\vec{k}$ and let x be the random variable (which may not exist).

Memory Eraser on Dependency Tree



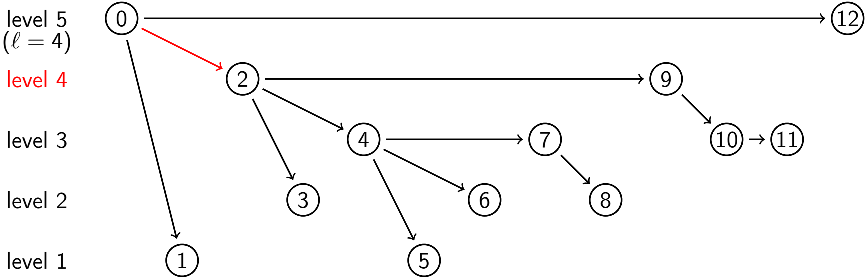
- Fix $\vec{k} = (0, 0, 2, 2)$.

Memory Eraser on Dependency Tree



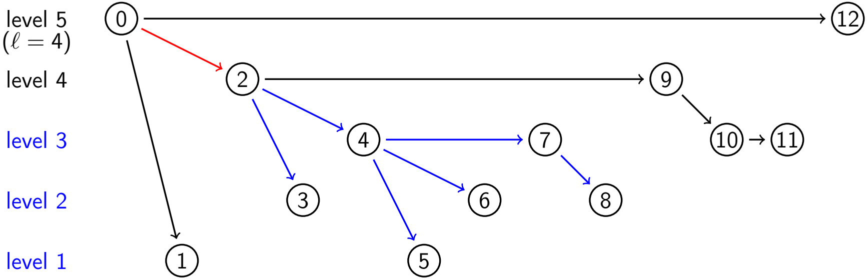
- Fix $\vec{k} = (0, 0, 2, 2)$.

Memory Eraser on Dependency Tree



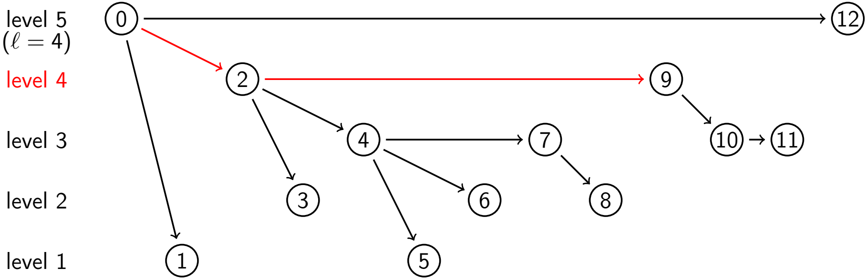
- Fix $\vec{k} = (0, 0, 2, 2)$.

Memory Eraser on Dependency Tree



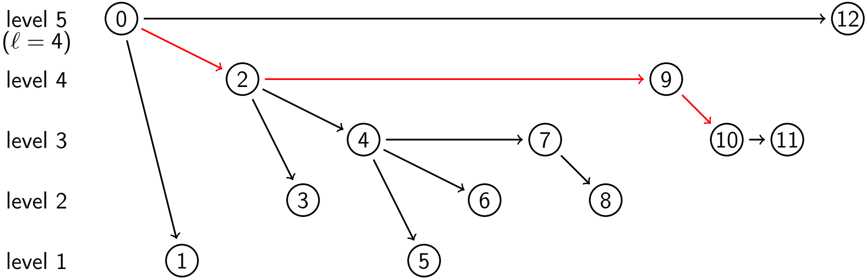
- Fix $\vec{k} = (0, 0, 2, 2)$.
- Blue part is a random variable. But it will finally end up with a node with level ≥ 4.

Memory Eraser on Dependency Tree



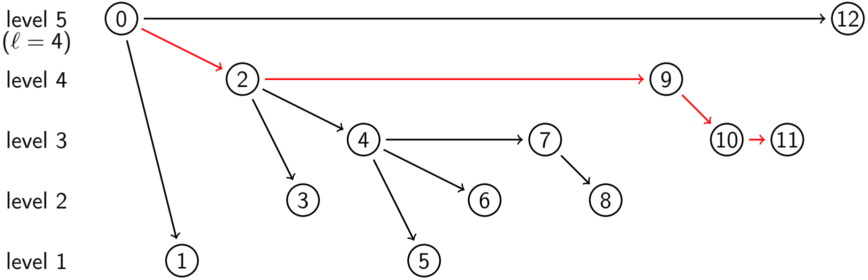
- Fix $\vec{k} = (0, 0, 2, 2)$.
- Blue part is a random variable. But it will finally end up with a node with level ≥ 4.

Memory Eraser on Dependency Tree



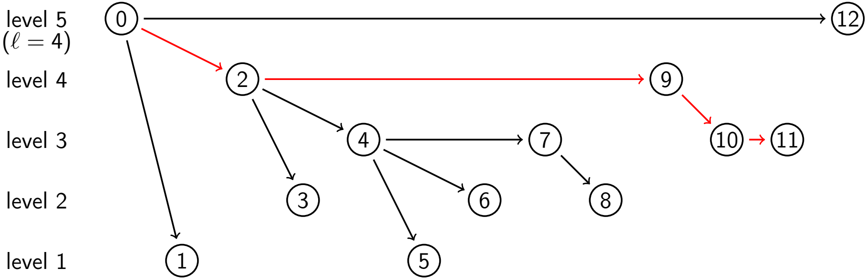
- Fix $\vec{k} = (0, 0, 2, 2)$.
- Blue part is a random variable. But it will finally end up with a node with level ≥ 4.

Memory Eraser on Dependency Tree



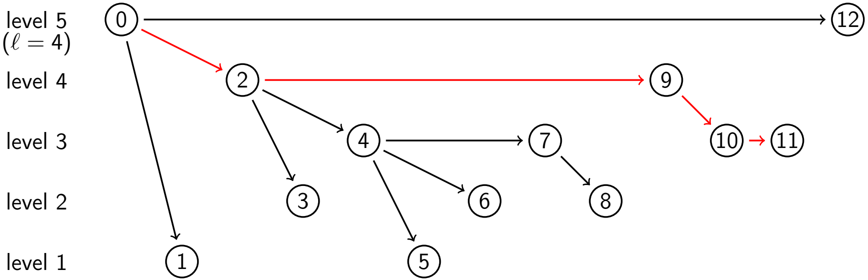
- Fix $\vec{k} = (0, 0, 2, 2)$.
- Blue part is a random variable. But it will finally end up with a node with level ≥ 4.

Memory Eraser on Dependency Tree



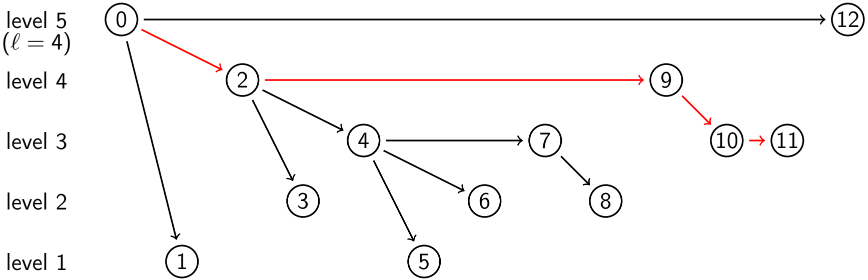
- Fix $\vec{k} = (0, 0, 2, 2)$.
- Blue part is a random variable. But it will finally end up with a node with level ≥ 4.
- One issue: What if $a_{w_2} = a_{w_9}$?

(Locally Simulatable) Extended Walk



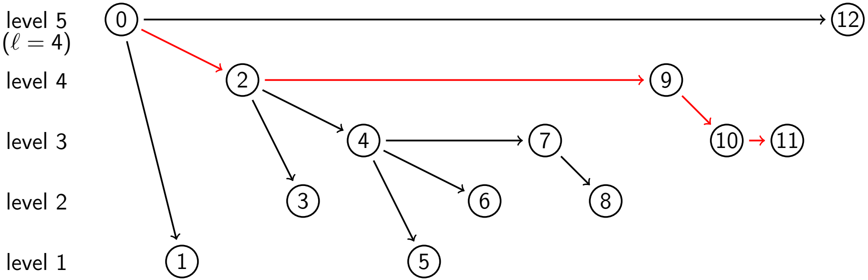
- Instead of original walk w, we look at extended walk w^*.

(Locally Simulatable) Extended Walk



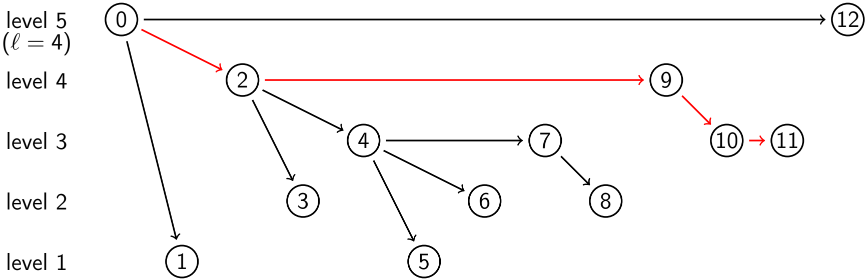
- Instead of original walk w, we look at extended walk w^*.
- Once a position in our memory is queried twice, we replace it with true randomness.

(Locally Simulatable) Extended Walk



- Instead of original walk w, we look at extended walk w^*.
- Once a position in our memory is queried twice, we replace it with true randomness.
- w^* is *locally simulatable* in the sense that each query position can be uniquely determined by memory.

(Locally Simulatable) Extended Walk



- Instead of original walk w, we look at extended walk w^*.
- Once a position in our memory is queried twice, we replace it with true randomness.
- w^* is *locally simulatable* in the sense that each query position can be uniquely determined by memory.
- w and w^* agree if w^* has no collision $a_{w_i^*} = a_{w_j^*}$.

Good = All - Bad

Recall our goal.

Our Main Lemma

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \ \forall u \in [n]$

- w and w^* agree if w^* has no collision $a_{w_i^*} = a_{w_j^*}$.

Good = All - Bad

Recall our goal.

Our Main Lemma

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \ \forall u \in [n]$

- w and w^* agree if w^* has no collision $a_{w_i^*} = a_{w_j^*}$.
- Good: $E[\#\{t|w_t^* = u, w^* \text{ has no collision}\}] \geq \Pr[\exists t, w_t = u]$.

Good = All - Bad

Recall our goal.

Our Main Lemma

- $\mathbb{E}[|f^*(s)|] \le O(\sqrt{n})$
- $\Pr[u \in f^*(s)] \ge \Omega(1/\sqrt{n}), \ \forall u \in [n]$

- w and w^* agree if w^* has no collision $a_{w_i^*} = a_{w_j^*}$.
- Good: $E[\#\{t | w_t^* = u, w^* \text{ has no collision}\}] \ge \Pr[\exists t, w_t = u]$.
- All: $E[\#\{t | w_t^* = u\}]$

Good = All - Bad

Recall our goal.

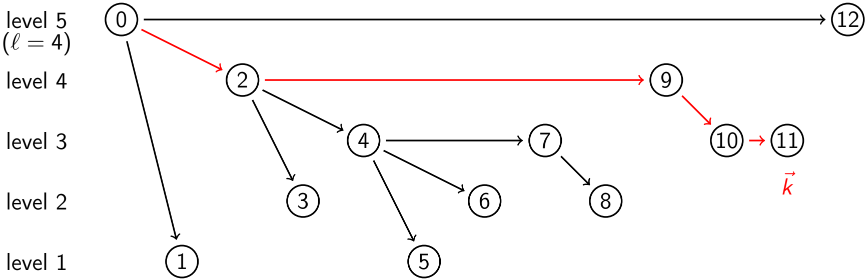
Our Main Lemma

- $\mathbb{E}[|f^*(s)|] \leq O(\sqrt{n})$
- $\Pr[u \in f^*(s)] \geq \Omega(1/\sqrt{n}), \ \forall u \in [n]$

- w and w^* agree if w^* has no collision $a_{w_i^*} = a_{w_j^*}$.
- Good: $E[\#\{t|w_t^* = u, w^* \text{ has no collision}\}] \geq \Pr[\exists t, w_t = u]$.
- All: $E[\#\{t|w_t^* = u\}]$
- Bad:

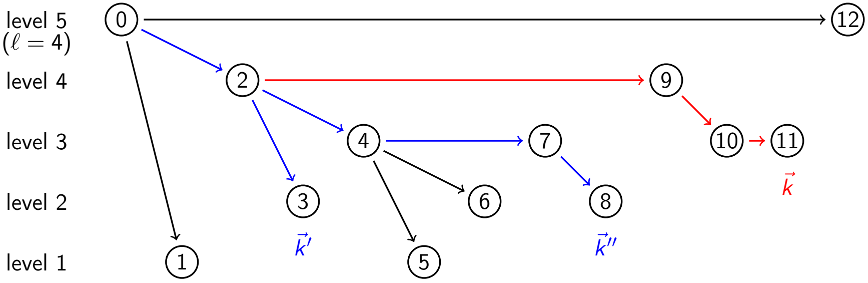
$$E[\#\{t|w_t^* = u, \exists t' \neq t'', a_{w_{t'}^*} = a_{w_{t''}^*}\}]$$
$$\leq E[\#\{t, t' \neq t''|w_t^* = u, a_{w_{t'}^*} = a_{w_{t''}^*}\}]$$

Good = All - Bad



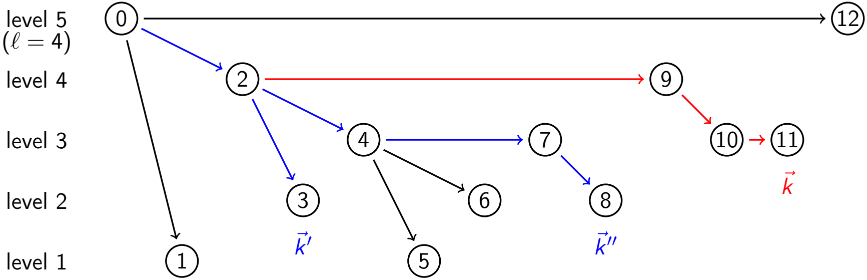
$$E[\#\{t | w_t^* = u\}] = \sum_{\vec{k}} \frac{2^{-(k_1+k_2+\cdots+k_\ell)}}{n}$$

Good = All - Bad



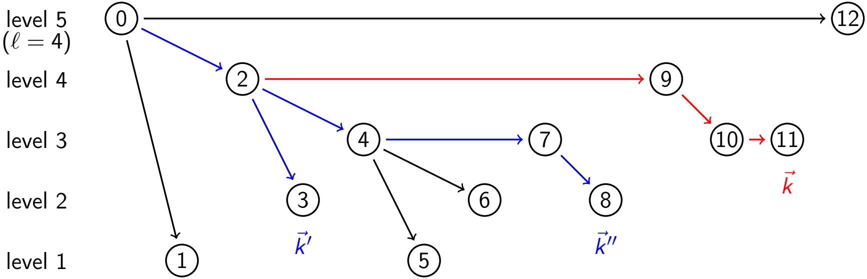
$$E[\#\{t, t' \neq t'' | w_t^* = u, a_{w_{t'}^*} = a_{w_{t''}^*}\}] = \sum_{\vec{k}, \vec{k}', \vec{k}''} \frac{2^{-\|\vec{k}\|_1 - \|\vec{k}'\|_1 - \|\vec{k}''\|_1}}{n^2}$$

Good = All - Bad



$$\text{Good} = \sum_{\vec{k}} \frac{2^{-(k_1+k_2+\cdots+k_\ell)}}{n} - \sum_{\vec{k},\vec{k}',\vec{k}''} \frac{2^{-\|\vec{k}\|_1 - \|\vec{k}'\|_1 - \|\vec{k}''\|_1}}{n^2}$$

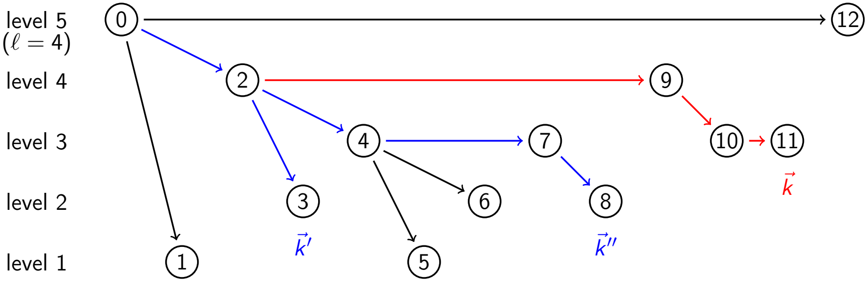
Good = All - Bad



$$\text{Good} = \sum_{\vec{k}} \frac{2^{-(k_1+k_2+\cdots+k_\ell)}}{n} - \sum_{\vec{k},\vec{k}',\vec{k}''} \frac{2^{-\|\vec{k}\|_1-\|\vec{k}'\|_1-\|\vec{k}''\|_1}}{n^2}$$

$$\sum_{\vec{k}} \frac{2^{-(k_1+k_2+\cdots+k_\ell)}}{n} = \frac{1}{n}\prod_{i=1}^{\ell}\sum_{k_i=0}^{\infty} 2^{-k_i} = \frac{2^\ell}{n}$$

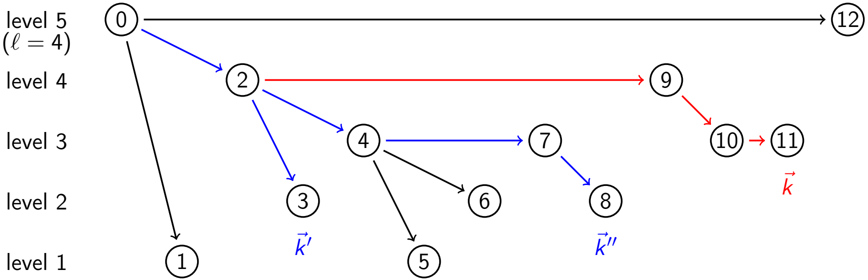
Good = All - Bad



$$\text{Good} = \sum_{\vec{k}} \frac{2^{-(k_1+k_2+\cdots+k_\ell)}}{n} - \sum_{\vec{k},\vec{k}',\vec{k}''} \frac{2^{-\|\vec{k}\|_1-\|\vec{k}'\|_1-\|\vec{k}''\|_1}}{n^2}$$

$$\sum_{\vec{k},\vec{k}',\vec{k}''} \frac{2^{-\|\vec{k}\|_1-\|\vec{k}'\|_1-\|\vec{k}''\|_1}}{n^2} = \frac{8^\ell}{n^2}$$

Good = All - Bad



$$\text{Good} = \sum_{\vec{k}} \frac{2^{-(k_1+k_2+\cdots+k_\ell)}}{n} - \sum_{\vec{k},\vec{k}',\vec{k}''} \frac{2^{-\|\vec{k}\|_1-\|\vec{k}'\|_1-\|\vec{k}''\|_1}}{n^2} = \frac{2^\ell}{n} - \frac{8^\ell}{n^2}$$

Let $\ell \leftarrow \frac{1}{2}\log n - 100$. $\frac{2^\ell}{n} - \frac{8^\ell}{n^2} = \frac{2^{-100}}{\sqrt{n}} - \frac{2^{-300}}{\sqrt{n}} = \Omega\left(\frac{1}{\sqrt{n}}\right)$.

Warning

Our Main Lemma

- $\mathbb{E}[|f^*(s)|] \le O(\sqrt{n})$
- $\Pr[u \in f^*(s)] \ge \Omega(1/\sqrt{n}),\ \forall u \in [n]$

- Even for this simple case, there is so much more technical challenges that is hidden in this talk.

Open Problems

Open Problems

- **Time-space Tradeoffs**
  In this work, we only solved the case when $S = \tilde{O}(1)$. Can we extend it to the full tradeoff?
- **Shorter Seed Length**
  In this work, our seed length is $O(\log^3 n \log\log n)$. Can this be improved?

References I

Miklos Ajtai and Avi Wigderson.
Deterministic simulation of probabilistic constant depth circuits.
In 26th Annual Symposium on Foundations of Computer Science (sfcs 1985), pages 11–19. IEEE, 1985.

Paul Beame, Raphaël Clifford, and Widad Machmouchi.
Element distinctness, frequency moments, and sliding windows.
In 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, pages 290–299. IEEE, 2013.

Allan Borodin, Faith Fich, F Meyer Auf Der Heide, Eli Upfal, and Avi Wigderson.
A time-space tradeoff for element distinctness.
SIAM Journal on Computing, 16(1):97–99, 1987.

References II

Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas.
Faster space-efficient algorithms for subset sum, k-sum, and related problems.
SIAM Journal on Computing, 47(5):1755–1777, 2018.

Andrew Chi-Chih Yao.
Near-optimal time-space tradeoff for element distinctness.
In FOCS, pages 91–97, 1988.