De Dictionariis Dynamicis Pauco Spatio Utentibus

(*lat.* On Dynamic Dictionaries Using Little Space)

Erik D. Demaine[1] Friedhelm Meyer auf der Heide[2]

Rasmus Pagh[3] Mihai Pătraşcu[1]

[1]MIT

[2]University of Paderborn

[3]IT University of Copenhagen

LATIN, March 2006

Dictionaries

Maintain $S \subset [u], |S| \le n$ under:

INSERT(x) : $S \leftarrow S \cup \{x\}$

DELETE(x) : $S \leftarrow S \setminus \{x\}$

MEMBER(x) : is $x \in S$?

OPUS CLASSICUM : [FKS'82]

query: $O(1)$ worst-case

update: $O(1)$ expected $\longrightarrow$ w.h.p. [DMadH'90]

space: $O(n \lg u)$ bits $\longrightarrow$ $O(n \lg \frac{u}{n})$ bits [RR'03]

We achieve both, through new techniques.

Dictionaries

Maintain $S \subset [u], |S| \le n$ under:

- INSERT(x) : $S \leftarrow S \cup \{x\}$
- DELETE(x) : $S \leftarrow S \setminus \{x\}$
- MEMBER(x) : is $x \in S$?

OPUS CLASSICUM : [FKS'82]

query: $O(1)$ worst-case

update: $O(1)$ expected $\longrightarrow$ w.h.p. [DMadH'90]

space: $O(n \lg u)$ bits $\longrightarrow$ $O(n \lg \frac{u}{n})$ bits [RR'03]

We achieve both, through new techniques.

Dictionaries

Maintain $S \subset [u], |S| \le n$ under:

INSERT(x) : $S \leftarrow S \cup \{x\}$

DELETE(x) : $S \leftarrow S \setminus \{x\}$

MEMBER(x) : is $x \in S$?

$\mathcal{OPUS\ CLASSICUM}$: [FKS'82]

query: $O(1)$ worst-case

update: $O(1)$ expected $\longrightarrow$ w.h.p. [DMadH'90]

space: $O(n \lg u)$ bits $\longrightarrow$ $O(n \lg \frac{u}{n})$ bits [RR'03]

We achieve both, through new techniques.

Dictionaries

Maintain $S \subset [u], |S| \leq n$ under:

INSERT(x) : $S \leftarrow S \cup \{x\}$

DELETE(x) : $S \leftarrow S \setminus \{x\}$

MEMBER(x) : is $x \in S$?

OPUS CLASSICUM : [FKS'82]

query:	$O(1)$ worst-case			
update:	$O(1)$ expected	$\longrightarrow$	w.h.p.	[DMadH'90]
space:	$O(n \lg u)$ bits	$\longrightarrow$	$O(n \lg \frac{u}{n})$ bits	[RR'03]

We achieve both, through new techniques.

Dictionaries

Maintain $S \subset [u], |S| \leq n$ under:

INSERT(x) : $S \leftarrow S \cup \{x\}$

DELETE(x) : $S \leftarrow S \setminus \{x\}$

MEMBER(x) : is $x \in S$?

$\mathcal{OPUS\ CLASSICUM}$: [FKS'82]

query: $O(1)$ worst-case

update: $O(1)$ expected $\longrightarrow$ w.h.p. [DMadH'90]

space: $O(n \lg u)$ bits $\longrightarrow$ $O(n \lg \frac{u}{n})$ bits [RR'03]

We achieve both, through new techniques.

Maintain $S \subset [u], |S| \le n$ under:

INSERT(x) : $S \leftarrow S \cup \{x\}$

DELETE(x) : $S \leftarrow S \setminus \{x\}$

MEMBER(x) : is $x \in S$?

OPUS CLASSICUM : [FKS'82]

query: $O(1)$ worst-case

update: $O(1)$ expected $\longrightarrow$ w.h.p. [DMadH'90]

space: $O(n \lg u)$ bits $\longrightarrow$ $O(n \lg \frac{u}{n})$ bits [RR'03]

We achieve both, through new techniques.

Basic Idea

How to get w.h.p. from expected bounds? independence

Idea: $\sqrt{n}$ independent tables

1. permute universe randomly: π
2. distribute elements by $hi(\pi(x))$
3. store $lo(\pi(x))$ in hash table

Permutation needed!

To be information-efficient, need to store just $lo(\pi(x))$.

Basic Idea

How to get w.h.p. from expected bounds? independence

Idea: $\sqrt{n}$ independent tables

1. permute universe randomly: π
2. distribute elements by $hi(\pi(x))$
3. store $lo(\pi(x))$ in hash table

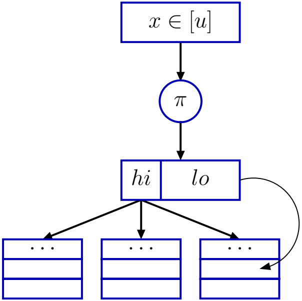
Permutation needed!

To be information-efficient, need to store just $lo(\pi(x))$.

Basic Idea

How to get w.h.p. from expected bounds? independence

Idea: $\sqrt{n}$ independent tables



1. permute universe randomly: π
2. distribute elements by $hi(\pi(x))$
3. store $lo(\pi(x))$ in hash table

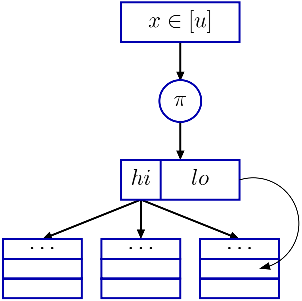
Permutation needed!

To be information-efficient, need to store just $lo(\pi(x))$.

Basic Idea

How to get w.h.p. from expected bounds? independence

Idea: $\sqrt{n}$ independent tables



1. permute universe randomly: π
2. distribute elements by $hi(\pi(x))$
3. store $lo(\pi(x))$ in hash table

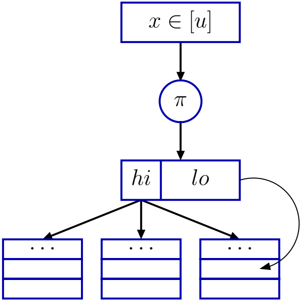
Permutation needed!

To be information-efficient, need to store just $lo(\pi(x))$.

Basic Idea

How to get w.h.p. from expected bounds? independence

Idea: $\sqrt{n}$ independent tables

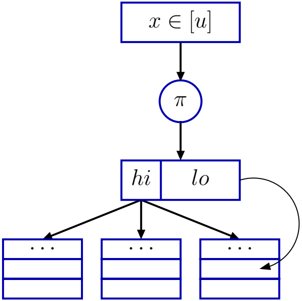


1. permute universe randomly: π
2. distribute elements by $hi(\pi(x))$
3. store $lo(\pi(x))$ in hash table

Permutation needed!

To be information-efficient, need to store just $lo(\pi(x))$.

How to get w.h.p. from expected bounds? independence



Idea: $\sqrt{n}$ independent tables

1. permute universe randomly: π
2. distribute elements by $hi(\pi(x))$
3. store $lo(\pi(x))$ in hash table

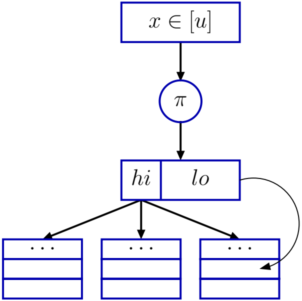
Permutation needed!

To be information-efficient, need to store just $lo(\pi(x))$.

Basic Idea

How to get w.h.p. from expected bounds? independence

Idea: $\sqrt{n}$ independent tables



1. permute universe randomly: π
2. distribute elements by $hi(\pi(x))$
3. store $lo(\pi(x))$ in hash table

Permutation needed!

To be information-efficient, need to store just $lo(\pi(x))$.

We need k-independent permutations.

Wait in line. . .

Idea: construct permutations with good k^{th} moment bounds.

Tools:

1. Siegel's highly independent hash functions
2. many tricks for reusing randomness

Highly Independent Permutations?

We need k-independent permutations.

Wait in line. . .

Idea: construct permutations with good k^{th} moment bounds.

Tools:

1. Siegel's highly independent hash functions
2. many tricks for reusing randomness

Highly Independent Permutations?

We need k-independent permutations.

Wait in line. . .

Idea: construct permutations with good k^{th} moment bounds.

Tools:

1. Siegel's highly independent hash functions
2. many tricks for reusing randomness

Highly Independent Permutations?

We need k-independent permutations.

Wait in line. . .

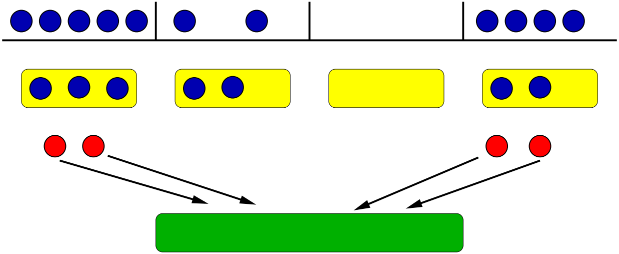
Idea: construct permutations with good k^{th} moment bounds.

Tools:

1. Siegel's highly independent hash functions
2. many tricks for reusing randomness

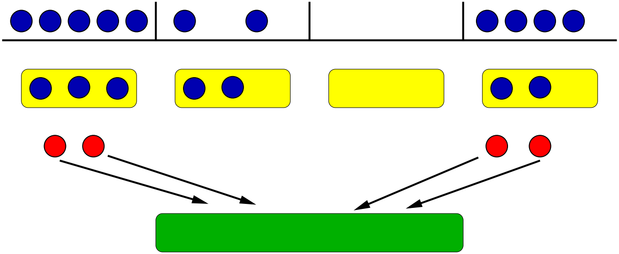
The Dictionary Structure

1. break universe into $n/\lg n$ segments
2. minidictionary in each segment with capacity $2\lg n$
3. fall back to high performance dictionary:



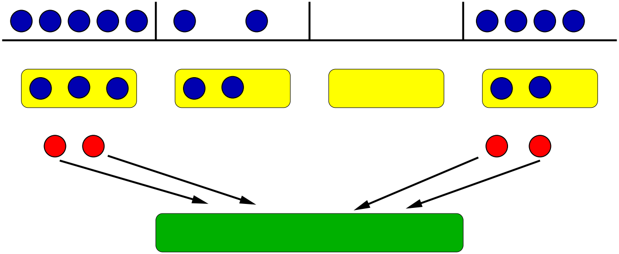
The Dictionary Structure

1. break universe into $n/\lg n$ segments
2. minidictionary in each segment with capacity $2\lg n$
3. fall back to high performance dictionary:



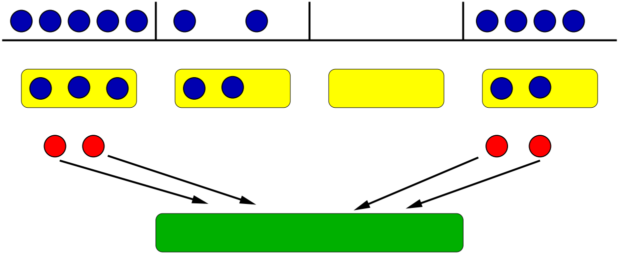
The Dictionary Structure

1. break universe into $n/\lg n$ segments
2. minidictionary in each segment with capacity $2\lg n$
3. fall back to high performance dictionary:



The Dictionary Structure

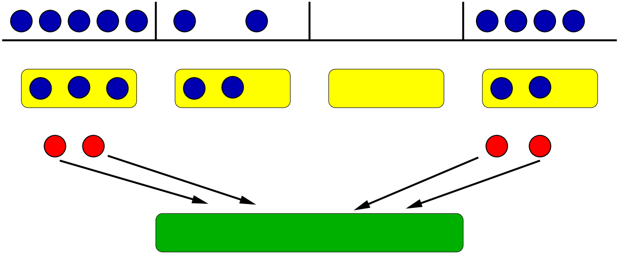
1. break universe into $n/\lg n$ segments
2. minidictionary in each segment with capacity $2\lg n$
3. fall back to high performance dictionary:
   - elements overflowing capacity of minidictionary
   - elements for which the minidictionary failed



The Dictionary Structure

1. break universe into $n/\lg n$ segments
2. minidictionary in each segment with capacity $2\lg n$
3. fall back to high performance dictionary:

exp. $O(1)$ bad elements from each segment
$\Rightarrow O(n/\lg n)$ total w.h.p.



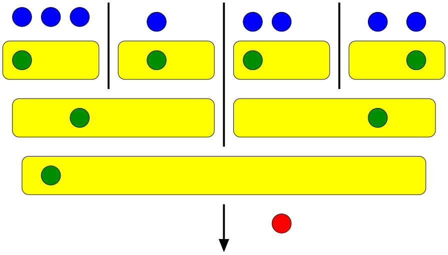
Life is tough

Minidictionary stores $O(\lg n)$ elements in arbitrary order

$\Rightarrow \Omega(\lg\lg n)$ bits per element!

For small universes, need to do something crazy...

$O(\lg\lg n)$ levels of filters
on disjoint segments of the universe } all packed in a word!



Dictionary? What dictionary?

What is the query?

membership: is $x \in S$?

requires $\geq \lg \binom{u}{n} = \Omega(n \lg \frac{u}{n})$ bits

retrieval: return $data(x)$ if $x \in S$

1.

2.

perf hashing: return unique, immutable ID for any $x \in S$

roughly same as retrieval

Dictionary? What dictionary?

What is the query?

membership: is $x \in S$?

requires $\geq \lg \binom{u}{n} = \Omega(n \lg \frac{u}{n})$ bits

retrieval: return $data(x)$ if $x \in S$

1. very useful without membership (trust me)
2. membership lower bound does not apply!

optimal space $\Theta(n \lg\lg \frac{u}{n})$ [sublinear!]

perf hashing: return unique, immutable ID for any $x \in S$

roughly same as retrieval

What is the query?

membership: is $x \in S$?

requires $\geq \lg \binom{u}{n} = \Omega(n \lg \frac{u}{n})$ bits

retrieval: return $data(x)$ if $x \in S$

1. very useful without membership (trust me)
2. membership lower bound does not apply!

optimal space $\Theta(n \lg \lg \frac{u}{n})$ [sublinear!]

perf hashing: return unique, immutable ID for any $x \in S$

roughly same as retrieval

Dictionary? What dictionary?

What is the query?

membership: is $x \in S$?

requires $\geq \lg \binom{u}{n} = \Omega(n \lg \frac{u}{n})$ bits

retrieval: return $data(x)$ if $x \in S$

1. very useful without membership (trust me)
2. membership lower bound does not apply!

optimal space $\Theta(n \lg \lg \frac{u}{n})$ [sublinear!]

perf hashing: return unique, immutable ID for any $x \in S$

roughly same as retrieval

Dictionary? What dictionary?

What is the query?

membership: is $x \in S$?

requires $\geq \lg \binom{u}{n} = \Omega(n \lg \frac{u}{n})$ bits

retrieval: return $data(x)$ if $x \in S$

1. very useful without membership (trust me)
2. membership lower bound does not apply!

optimal space $\Theta(n \lg \lg \frac{u}{n})$ [sublinear!]

perf hashing: return unique, immutable ID for any $x \in S$

roughly same as retrieval

Dictionary? What dictionary?

What is the query?

membership: is $x \in S$?

requires $\geq \lg \binom{u}{n} = \Omega(n \lg \frac{u}{n})$ bits

retrieval: return $data(x)$ if $x \in S$

1. very useful without membership (trust me)
2. membership lower bound does not apply!

optimal space $\Theta(n \lg \lg \frac{u}{n})$ [sublinear!]

perf hashing: return unique, immutable ID for any $x \in S$

roughly same as retrieval

Dictionary? What dictionary?

What is the query?

membership: is $x \in S$?

requires $\geq \lg \binom{u}{n} = \Omega(n \lg \frac{u}{n})$ bits

retrieval: return $data(x)$ if $x \in S$

1. very useful without membership (trust me)
2. membership lower bound does not apply!

optimal space $\Theta(n \lg \lg \frac{u}{n})$ [sublinear!]

perf hashing: return unique, immutable ID for any $x \in S$

roughly same as retrieval

Dictionary? What dictionary?

What is the query?

membership: is $x \in S$?

requires $\geq \lg \binom{u}{n} = \Omega(n \lg \frac{u}{n})$ bits

retrieval: return $data(x)$ if $x \in S$

1. very useful without membership (trust me)
2. membership lower bound does not apply!

optimal space $\Theta(n \lg \lg \frac{u}{n})$ [sublinear!]

perf hashing: return unique, immutable ID for any $x \in S$

roughly same as retrieval

Recapitulare

1. independence through permutation hashing
2. multilevel word-packed dictionaries
3. dictionaries without membership

FINIS

GRATIAS AGO VOBIS