Data struktures II
NTIN067

Jirka Fink
https://ktiml.mff.cuni.cz/~fink/

Katedra teoretické informatiky a matematické logiky
Matematicko-fyzikální fakulta
Univerzita Karlova v Praze

Summer semester 2018/19
Last change March 18, 2019

Content

Jirka Fink: Data Structures II

General information

E-mail fink@ktiml.mff.cuni.cz
Homepage https://ktiml.mff.cuni.cz/~fink/
Consultations Individual schedule

Content

Computational model Word-RAM

Description

- A word is a w-bit integer
- A memory an array of words indexed by words
- The size of memory is 2^w, so we assume that $w = \Omega(\log n)$
- Operations of words in constant time:
  - Arithmetical operations are $+, -, \star, /,$ mod
  - Bit-wise operations &, $|, \hat{}, >>, <<$
  - Comparisons $=, <, \leq, >, \geq$
- Other operations in constant time: (un)conditional jumps, assignments, memory accesses, etc.
- Inputs and outputs are stored in memory

Static dictionaries

Notations

- Universe U of all elements (words)
- Store $S \subseteq U$ of size n in a data structure
- Using hashing, we store S in a table $M = [m] = \{0, \ldots, m-1\}$ of size m

Goal

Create a data structure determining whether a given element of U belongs to S.

Methods

	Build	Member	
Search tree	$n \log n$	$\log n$	optimal in the comparison model
Cuckoo	n (exp.)	1	$\log n$-independent
FKS	n (exp.)	1	2-independent
	$n \log n$	1	deterministic

Independent repeated trials

Markov inequality

If X is a non-negative random variable and $c > 1$, then $P[X < cE[X]] > \frac{c-1}{c}$.

Expected number of trial using probability

Let V be an event that occurs in a trial with probability p. The expected number of trials to first occurrence of V in a sequence of independent trials is $\frac{1}{p}$.

Expected number of trial using mean

If X is a non-negative random variable and $c > 1$. The expected number of trials to first occurrence of $X \leq cE[X]$ in a sequence of independent trials is at most $\frac{c}{c-1}$.

Example

If the expected number of collisions of a randomly chosen hashing function h is k, then the expected number of independent trials to the first occurrence of a hashing function h with at most $2k$ collisions is 2.

Universal hashing systems

c-universal hashing system

A hashing system $\mathcal{H}$ of functions $h : U \to M$ is c-universal for $c > 1$ if a uniformly chosen h from $\mathcal{H}$ satifies $P[h(x) = h(y)] \leq \frac{c}{m}$ for every $x, y \in U$ and $x \neq y$.

k-independent hashing system

A hashing system $\mathcal{H}$ of functions $h : U \to M$ is k-independent for $k \in \mathbb{N}$ if a uniformly chosen h from $\mathcal{H}$ satifies $P[h(x_i) = z_i \text{ for all } i = 1, \ldots, k] = \mathcal{O}\left(\frac{1}{m^k}\right)$ for all pairwise different $x_1, \ldots, x_k \in U$ and all $z_1, \ldots, z_k \in M$.

Example: System Multiply-mod-prime

- Let p be a prime greater than $|U|$
- $h_{a,b}(x) = (ax + b \bmod p) \bmod m$
- $\mathcal{H} = \{h_{a,b};\ a, b \in [p], a \neq 0\}$
- System $\mathcal{H}$ is 1-universal and 2-independent but it is not 3-independent

Hagerup, Miltersen, Pagh, 2001 [2]

Goal

A static dictionary for n w-bit keys with constant lookup time and a space consumption of $\mathcal{O}(n)$ words can be constructed in $\mathcal{O}(n \log n)$ time on w-word-RAM.
The algorithm is weakly non-uniform, i.e. requires certain precomputed constants dependent on w.

Overview

1. Create a function $f_1 : [2^w] \to [2^{4w}]$ which is an error-correcting code of relative minimum distance $\delta > 0$.
2. Create a function $f_2 : [2^{4w}] \to [\mathcal{O}(n^k)]$ which is an injection on $f_1(S)$
3. Create a function $f_3 : [\mathcal{O}(n^k)] \to [\mathcal{O}(n^2)]$ which is an injection on $f_2(f_1(S))$
4. Create a function $f_4 : [\mathcal{O}(n^2)] \to [\mathcal{O}(n)]$ which is an injection on $f_3(f_2(f_1(S)))$

- $f_4 \circ f_3 \circ f_2 \circ f_1$ can be computed in constant time
- f_2, f_3, f_4 can be found in time $\mathcal{O}(n \log n)$
- f_1 can be precomputed in time $\mathcal{O}(w)$

1. The number of remaining bits is at most r.
2. Since $r \geq \frac{w}{2}$.

Static dictionaries: $[\mathcal{O}(n^2)] \to [\mathcal{O}(n)]$

Lemma

Suppose that (f, g) is q-good and $r \geq 3 + \log n$. Then, for every $v \in [2^r]$ there exists $a_v \in [2^r]$ such that $(x \mapsto g(x) \oplus a_{f(x)}, f)$ is q'-good where $q' = \begin{cases} 0 & \text{if } q \leq n \\ n & \text{otherwise.} \end{cases}$
All values a_v can be computed in expected time $\mathcal{O}(n)$ and space $\mathcal{O}(n)$ worst case.

Proof ($q' \leq n$)

1. Let $h(x) = g(x) \oplus a_{f(x)}$
2. If $x, y \in S$ and $x \neq y$ and $f(x) = f(y)$, then $g(x) \neq g(y)$ and $h(x) \neq h(y)$ ①
3. If $f(x) \neq f(y)$, then $P[h(x) = h(y)] = \frac{1}{2^r}$ where $a_v \sim U[2^r]$ independently for all $v \in [2^r]$ ②
4. $E[\,|\{\{x, y\} \subseteq S;\ h(x) = h(y)\}|\,] \leq \binom{n}{2}/2^r < \frac{n}{16}$ ③
5. The expected number of trials to generate h with at most n collisions is $\mathcal{O}(1)$.

1. Since $x \mapsto (f(x), g(x))$ is perfect on S, $g(x) \neq g(y)$. From $g(x) \oplus a_{f(x)} = g(x) \oplus a_{f(y)} \neq g(y) \oplus a_{f(y)}$ it follows that $h(x) \neq h(y)$.
2. For every $v \in [2^r]$ we randomly and independently choose a_v from the uniform distribution on $[2^r]$. Then,
$$\begin{aligned} h(x) &= h(y) \\ g(x) \oplus a_{f(x)} &= g(y) \oplus a_{f(y)} \\ a_{f(x)} &= g(x) \oplus g(y) \oplus a_{f(y)} \end{aligned}$$
Since $([2^r], \oplus)$ is an Abelian group, $b \mapsto b \oplus c$ is a bijection on $[2^r]$ for every $c \in [2^r]$ and so $a_{f(y)} \sim U[2^r]$, it follows that $g(x) \oplus g(y) \oplus a_{f(y)} \sim U[2^r]$. Since $a_{f(x)}$ and $a_{f(y)}$ are independent, also $a_{f(x)}$ and $g(x) \oplus g(y) \oplus a_{f(y)}$ are independent. Hence, $P[h(x) = h(y)] = \frac{1}{2^r}$.
3. Use the linearity of expectation and substitute r.

Static dictionaries: $[\mathcal{O}(n^2)] \to [\mathcal{O}(n)]$

Lemma

Suppose that (f, g) is q-good and $r \geq 3 + \log n$. Then, for every $v \in [2^r]$ there exists $a_v \in [2^r]$ such that $(x \mapsto g(x) \oplus a_{f(x)}, f)$ is q'-good where $q' = \begin{cases} 0 & \text{if } q \leq n \\ n & \text{otherwise.} \end{cases}$
All values a_v can be computed in expected time $\mathcal{O}(n)$ and space $\mathcal{O}(n)$ worst case.

Proof ($q \leq n$ implies $q' = 0$)

1. Let $S_v = \{x \in S;\ f(x) = v\}$
2. Order S_v by non-increasing size, i.e. $|S_{v_1}| \geq |S_{v_2}| \geq \ldots \geq |S_{v_{2^r}}|$
3. For $j = 1, \ldots, 2^r$ we find a_{v_j} such that h is perfect ①
4. For $a_{v_j} \sim U[2^r]$ it holds $E[\,|\{(x, y) \in S_{v_j} \times S_{<j};\ h(x) = h(y)\}|\,]$
   $\leq |S_{v_j}||S_{<j}|P[h(x) = h(y)]$ ②
   $\leq \sum_{i=1}^{j-1} |S_{v_j}||S_{v_i}|/2^r$ ③
   $\leq \sum_{i=1}^{j-1} |S_{v_i}^2|/2^r$ ④
   $\leq \sum_{i=1}^{j-1} \binom{S_{v_i}}{2}/2^{r-2}$ ⑤
   $\leq q/2^{r-2} \leq \frac{1}{2}$
5. The expected number of trials to generate a_{v_j} such that h has no collision is $\mathcal{O}(1)$. ⑥ ⑦

Static dictionaries: $[\mathcal{O}(n^2)] \to [\mathcal{O}(n)]$

Goal

Find a function $h : U \to [2^r]$ with $U = [\mathcal{O}(n^2)]$ and $r = \max\{\frac{w}{2}, 3 + \log n\}$ s.t.

- h is perfect on $S \subseteq U$ of size n and
- h can be computed in constant time and
- space consumption $\mathcal{O}(n)$ for finding and storing h and
- h can be found in $\mathcal{O}(n \log n)$ worst case time.
  - First, expected $\mathcal{O}(n)$ time,
  - then derandomize to $\mathcal{O}(n \log n)$ worst case time.

$x \in U$ is a point $(\alpha(x), \beta(x))$ in a $(\mathcal{O}(n) \times \mathcal{O}(n))$-table

For $x \in U$, let $\alpha(x)$ denote the first r bits of x and $\beta(x)$ denotes the remaining bits. ①
Then $x \mapsto (\alpha(x), \beta(x))$ is an injection (e.i. perfect on U). ②

Static dictionaries: $[\mathcal{O}(n^2)] \to [\mathcal{O}(n)]$

Definition

For $q \geq 0$ and functions $f, g : U \to [2^r]$, the pair (f, g) is q-good if

- f has at most q collisions and
- $x \mapsto (f(x), g(x))$ is perfect on S.

The number of collisions is the number of pairs $\{x, y\} \subseteq S$ such that $f(x) = f(y)$.

Lemma

Suppose that (f, g) is q-good and $r \geq 3 + \log n$. Then, for every $v \in [2^r]$ there exists $a_v \in [2^r]$ such that $(x \mapsto g(x) \oplus a_{f(x)}, f)$ is q'-good where $q' = \begin{cases} 0 & \text{if } q \leq n \\ n & \text{otherwise.} \end{cases}$
All values a_v can be computed in expected time $\mathcal{O}(n)$ and space $\mathcal{O}(n)$ worst case.

Application: Randomized construction of a mapping $[\mathcal{O}(n^2)] \to [\mathcal{O}(n)]$

0. (α, β) is $\binom{n}{2}$-good
1. $(x \mapsto \beta(x) \oplus a_{\alpha(x)}, \alpha) =: (\alpha', \beta')$ is n-good
2. $(x \mapsto \beta'(x) \oplus a'_{\alpha'(x)}, \alpha') =: (\alpha'', \beta'')$ is 0-good, so α'' is perfect

1. Note that we must find h without collisions. To be precise, we iteratively find a_{v_j} for j from 1 to 2^r such that it holds $h(x) \neq h(y)$ for every $x \in S_{v_j}$ and $y \in S_{<j}$ where $S_{<j} = \bigcup_{i=1}^{j} S_{v_i}$.
2. Linearity of expectation
3. Definition of $S_{<j}$
4. $|S_{v_i}| \geq |S_{v_j}|$
5. From this point, assume that $|S_{v_j}| \geq 2$.
6. In order to verify that h has no collision, we use a counter $m_v = |\{y \in S_{<j};\ h(y) = v\}|$. For every j we can count the collisions and update m_v in time $\mathcal{O}(|S_i|)$. The expected time to find all a_{v_j} is $\sum_j \mathcal{O}(|S_i|) = \mathcal{O}(n)$.
7. For $S_{v_j} = \{x\}$ we can find v with $m_v = 0$ and set $a_{v_j} = v \oplus g(x)$.

Static dictionaries: $[\mathcal{O}(n^2)] \to [\mathcal{O}(n)]$

Derandomization

Let $L_k(a) = |\{(x,y) \in S_{v_j} \times S_{<j};\ (g(x) \oplus a)_{[k]} = (h(y))_{[k]}\}|$ ①
$(a)_i$ denotes the i-th bit of a and $(a)_M$ denotes the vector of all bits $(a)_i$ for $i \in M \subseteq [r]$ ②

```
for j ← 1 to 2^r do
    a_{v_j} ← 0
    for k ← 0 to r − 1 do
        if L_{k+1}(a_{v_j}) > L_{k+1}(a_{v_j} + 2^k) then
            a_{v_j} ← a_{v_j} + 2^k
```

Proof (goodness of (h, f))

- $L_k(a) + L_k(a \oplus 2^k) = L_{k-1}(a)$ for every $a \in [2^k]$ and $k \in [r]$
- $L_k(a_{v_j}) \leq \frac{L_{k-1}(a_{v_j})}{2} \leq \frac{L_0(a_{v_j})}{2^k} = \frac{|S_{v_j}||S_{<j}|}{2^k}$
- The total number of collision is at most $\sum_j L_r(a_{v_j}) \leq \sum_j 2^{-r}|S_{v_j}||S_{<j}| \leq \sum_{i<j} 2^{-r}|S_{v_j}||S_{v_i}| \leq 2^{-r-1}\left(\sum_i S_{v_i}\right)^2 < \frac{n}{16}$
- If $q \leq n$, then the number of collision with S_{v_j} is $L_r(a_{v_j}) \leq \sum_{i<j} 2^{-r}|S_{v_j}||S_{v_i}| \leq \sum_{i<j} 2^{2-r}\binom{S_{v_j}}{2} \leq 2^{2-r}q \leq \frac{1}{2}$

Static dictionaries: $[\mathcal{O}(n^2)] \to [\mathcal{O}(n)]$

Derandomization

Let $L_k(a) = |\{(x,y) \in S_{v_j} \times S_{<j};\ (g(x) \oplus a)_{[k]} = (h(y))_{[k]}\}|$
$(a)_i$ denotes the i-th bit of a and $(a)_M$ denotes the vector of all bits $(a)_i$ for $i \in M \subseteq [r]$

```
for j ← 1 to 2^r do
    a_{v_j} ← 0
    for k ← 0 to r − 1 do
        if L_{k+1}(a_{v_j}) > L_{k+1}(a_{v_j} + 2^k) then
            a_{v_j} ← a_{v_j} + 2^k
```

Proof (Complexity)

- In order to compute $L_k(a)$, we build a binary tree (trie)
- Every vertex $a \in [2^k]$ of the k-th level has a counter $c_k(a) = |\{y \in S_{<j};\ (h(y))_{[k]} = (a)_{[k]}\}|$
- $L_k(a) = \sum_{x \in S_{v_j}} c_k(g(x) \oplus a)$ can be computed in $\mathcal{O}(|S_{v_j}|)$ time
- After the j-th step, counters can be updated in $\mathcal{O}(|S_{v_j}|r)$ time
- Total time is $\sum_j |S_{v_j}|r = \mathcal{O}(n \log n)$

Static dictionaries: Error-correcting code

Definition

- The Hamming distance between $x \in [2^w]$ and $y \in [2^w]$ is the number of bits in which x and y differ.
- $\psi : [2^w] \to [2^{4w}]$ is an error correcting code of relative minimum distance $\delta > 0$ if the Hamming distance between $\psi(x)$ and $\psi(y)$ is at least $4w\delta$ for every distinct $x, y \in [2^w]$.

Lemma

Let $\mathcal{H}$ be a 2-universal hashing system of function $[2^w] \to [2^{4w}]$. For every δ with $1/4w < \delta \leq 1/2$, the probability that $h \sim U(\mathcal{H})$ is an error correcting code of relative minimum distance $\delta > 0$ is at least $1 - ((\frac{e}{\delta})^{4\delta}/4)^w$. ①

Proof

- For $x \in [2^{4w}]$ the number of y within Hamming distance k is at most $(\frac{4ew}{k})^k$. ②
- For $x \neq y$, $P(\text{Hamming distance between } x \text{ and } y\ \leq k) \leq 2^{1-4w}(\frac{4ew}{k})^k$
- The probability that this happens for any of the $\binom{2^w}{2} < 2^{2w-1}$ pairs is at most $((\frac{e}{\delta})^{4\delta}/4)^w$ ③

Static dictionaries: $[2^w] \to [\mathcal{O}(n^k)]$

Lemma

Let $\psi : [2^w] \to [2^{4w}]$ be an error correcting code of relative minimum distance $\delta > 0$ and $S \subseteq U = [2^w]$ of size n. There exists a set $D \subseteq [4w]$ with $|D| \leq 2 \log n / \log \frac{1}{1-\delta}$ such that for every pair x, y of distinct elements of S it holds $(\psi(x))_D \neq (\psi(y))_D$.

Proof

- For $D \subseteq [4w]$ and $v \in [2^{|D|}]$ let $C(D, v) = \{x \in S;\ (\psi(x))_D = v\}$ ①
- The set of colliding pairs of D is $B(D) = \bigcup_{v \in [2^{|D|}]} \binom{C(D,v)}{2}$
- We construct $D_0 \subseteq D_1 \subseteq \ldots \subseteq D_k$ such that $|D_i| = i$ and $|B(D_i)| < (1-\delta)^i n^2/2$ ②
- Let $I(d) = \{\{x, y\} \in B(D_i);\ (\psi(x))_d = (\psi(y))_d\}$ be the colliding pairs indistinguishable by $d \in [4w] \setminus D_i$
- Let $I = \sum_{d \in [4w] \setminus D_i} |I(d)|$
- Every pair $\{x, y\} \in B(D_i)$ contributes to I by at most $4w - i - 4w\delta < 4w(1-\delta)$, so $I \leq 4w(1-\delta)|B(D_i)|$
- By averaging, there exists $d \in [4w] \setminus D_i$ such that $|I(d)| \leq \frac{I}{4w-i} \leq (1-\delta)|B(D_i)|$ ③
- Let $D_{i+1} = D_i \cup \{d\}$. Hence, $|B(D_{i+1})| = |I(d)| \leq (1-\delta)|B(D_i)|$
- By setting $k = \left\lfloor 2 \log n / \log \frac{1}{1-\delta} \right\rfloor$ we obtain $|B(D_k)| < 1$.

1. Where $k \in [r]$ and $a \in [2^k]$
2. Our goal is to iteratively and deterministically compute a_{v_j} for j from 1 to 2^r. The value of a_{v_k} is computed by bits from the least significant to the most significant bit. $L_k(a)$ determines the number of collision between S_{v_j} and $S_{<j}$ if we consider only last k bits.

Static dictionaries: $[\mathcal{O}(n^k)] \to [\mathcal{O}(n^2)]$

Approach

- Every $x \in U = [\mathcal{O}(n^k)]$ can be regarded as constant-length string over an alphabet of size n
- Build n-way branching compressed trie of string S
- The number of leaves is $|S| = n$, so the total number of vertices is at most $2n$
- Build static $[\mathcal{O}(n^2)] \to [\mathcal{O}(n)]$ dictionary for pairs (vertex of the trie, letter) which returns a child of the vertex
- One polynomial-size-universe lookup is evaluated using a constant number of quadratic-size-universe lookups
- Space complexity is $\mathcal{O}(n)$ and this dictionary is constructed in $\mathcal{O}(n \log n)$ time

1. For $\delta < \frac{1}{4w}$ it holds that $4w\delta < 1$ and the identity is an error correcting code of relative minimum distance δ.
2. The number of $y \in [2^{4w}]$ within Hamming distance $k \geq 1$ from a fixed $x \in [2^{4w}]$ is $\sum_{i=0}^{k} \binom{4w}{i} \leq (\frac{4w}{k})^k \sum_{i=0}^{k} \binom{4w}{i}(\frac{k}{4w})^i \leq (\frac{4w}{k})^k (1 + \frac{k}{4w})^{4w} \leq (\frac{4w}{k})^k e^k \leq (\frac{4ew}{k})^k$ using the binomial theorem.
3. By setting $k = \lfloor 4w\delta \rfloor$ we obtain $2^{2w-1}2^{1-4w}(\frac{4ew}{k})^k \leq 2^{-2w}(\frac{4ew}{4w\delta})^{4w\delta} = (2^{-2}(\frac{e}{\delta})^{4\delta})^w$

1. Note that for every $D \subseteq [4w]$ the set S is split into $2^{|D|}$ disjoint clusters $C(D, v)$ for $v \in [2^{|D|}]$.
2. For $i = 0$ it holds that $D_0 = \emptyset$ and $B(D_0) = \binom{n}{2} < \frac{n^2}{2}$.
3. A bit $d \in [4w] \setminus D_i$ with $|I(d)| \leq (1-\delta)|B(D_i)|$ can be found in $\mathcal{O}(wn)$ time as follows. We a list of all clusters $C(D_i, v)$ of size at least two. Every cluster has a list of all elements. So, $I(d)$ for one $d \in [4w] \setminus D_i$ can be determined in $\mathcal{O}(n)$ time and we can process all d in $\mathcal{O}(wn)$ time. Then, all lists can be updated in $\mathcal{O}(n)$ time. Using word-level parallelism, the time complexity can be improved to $\mathcal{O}(n)$.

Content

Word-RAM as a vector computer

Theorem (Linear compression and speed up

An algorithm running on a $\mathcal{O}(w)$-word-RAM with time complexity $T(n)$ and space complexity $S(n)$ can be simulated on w-word-RAM with time complexity $\mathcal{O}(T(n))$ and space complexity $\mathcal{O}(S(n))$ assuming $S(n) = o(2^w)$. ① ②

Proof

- Every word of $\mathcal{O}(w)$-word-RAM is replaced by $\mathcal{O}(1)$ words on w-word-RAM, so the space complexity is $\mathcal{O}(T(n))$.
- Every instruction on $\mathcal{O}(w)$-word-RAM is replaced by $\mathcal{O}(1)$ instruction on w-word-RAM. ③

1. This theorem is similar to the linear speed theorem for Turing machines.
2. The assumption $S(n) = o(2^w)$ is needed to ensure a sufficient amount of memory for linear data decompression.
3. Bit-wise operations and comparisons are trivial. In order to simulate arithmetical operations, every $\mathcal{O}(w)$ bits word is split into $\frac{w}{2}$ bits words to handle integer overflows and carry.

RAM with uninitialized memory

Theorem

Every program with time complexity $T(n)$ and space complexity $S(n)$ which assumes initialized memory by zeros can be simulated by a program with time complexity $\mathcal{O}(T(n))$ and space complexity $S(n)$ which do not assume initialized memory. ①

Proof

Simulation uses the following variables:

- M: An array of the original memory
- N: The number of already initialized cells by the original program
- I: An array of indices of all initialized cells
- X: A reverse table of I, i.e. if j is initialized, then $I[X[j]] = j$

A read cell j is initialized if and only if $X[j] < N$ and $I[X[j]] = j$.

1. Reading initialized memory must be permitted but an arbitrary value can be read.

Integer data structures

Goal

Create a dynamic data structure storing a set S of intergers from a universe U with is can find the predecessor and the successor of a element $x \in U$.

- Predecessor: $\textsc{Pred}(x \in U) = \max\{y \in S;\ y < x\}$
- Successor $\textsc{Succ}(x \in U) = \min\{y \in S;\ y > x\}$

An invalid element is returned if no such element exists.

Our knowledge

$\mathcal{O}(\log n)$ query and update using search trees

van Emde Boas tree [3]

Notation

For a t-bit integer $x \in U$ let

- $r(x)$ be the left (upper) $\lceil t/2 \rceil$ bits of x and
- $l(x)$ be the left (lower) $\lceil t/2 \rceil$ bits of x.

For a set of elements S let

- $S_i = \{r(x) \in S \setminus \min S;\ l(x) = i\}$ for $i \in l(U)$
- $P = \{S_i;\ S_i \neq \emptyset\}$

A node of van Emde Boas tree contains

- Pointers to the minimum and maximum of S
- An array of vEB trees for all sets S_i
- A primary vEB tree of elements P for finding non-empty buckets

Space complexity

- Terrible: $\mathcal{O}(U)$, but we will improve it later.
- Requires initialized memory

van Emde Boas tree: Operation FIND(x)

Algorithm

```
1 Procedure FIND(x, S)
2     if S = ∅ then
3         return None
4     if x = min(S) then
5         return min(S)
6     return FIND(r(x), S_{l(x)})
```

Analysis

- The height of the tree is $\mathcal{O}(\log \log U)$
- Time complexity is $\mathcal{O}(\log \log U)$

van Emde Boas tree: Operation SUCC(x)

Algorithm

```
1  Procedure SUCC(x, S)
2      if S = ∅ or x ≥ max(S) then
3          return None
4      if x < min(S) then
5          return min(S)
6      i ← I(x)
7      if S_i ≠ ∅ and x < max(S_i) then
8          return SUCC(r(x), S_i)
9      i ← SUCC(i, P)
10     return min(S_i)
```

Analysis

- The recursion is called at most once for every level of the tree
- Time complexity is $\mathcal{O}(\log \log U)$

van Emde Boas tree: Operation INSERT(x)

Algorithm

```
1  Procedure INSERT(x, S)
2      if S = ∅ then
3          min(S) ← max(S) ← x and return
4      if x < min(S) then
5          swap(x, min(S))
6      if x > max(S) then
7          max(S) ← x
8      i ← I(x)
9      if S_i = ∅ then
10         INSERT(i, P)
11     INSERT(r(x), S_i)
```

Analysis

- If $S_i = \emptyset$, then INSERT$(r(x), S_i)$ only sets $\min(S_i)$ and $\max(S_i)$
- A non-trivial recursion is called at most once for every level of the tree
- Time complexity is $\mathcal{O}(\log \log U)$

van Emde Boas tree: Operation DELETE(x)

Algorithm ①

```
1  Procedure DELETE(x, S)
2      if S = ∅ or x = min(S) = max(S) then
3          S ← ∅ and return
4      if x = min(S) then
5          min(S) ← DELETEMIN(S) and return
6      i ← I(x)
7      if x = min(S_i) = max(S_i) then
8          DELETE(i, P)
9      DELETE(r(x), S_i)
10     max(S) ← max(S_max(P))
11 Procedure DELETEMIN(S) ②
12     i ← min(P)
13     if min(S_i) < max(S_i) then
14         return DELETEMIN(S_i)
15     S_i ← ∅
16     i ← DELETEMIN(P)
17     return min(S_i)
```


1. Time complexity is also $\mathcal{O}(\log \log U)$
2. Function DELETEMIN also returns the new minimal value.

x-fast trie (Willard [4])

First step

- A set S is represented using an array A such that x is stored at $A[x]$
- Non-empty positions of A are interconnected using a linked list
- Build a binary tree T whose leaves are cells of A
- Every node v of T stores the minimum and maximum of the subtree of v

Operation SUCC(x)

- If $A[x]$ is non-empty, then return $A[x] \to next$
- Find the largest empty subtree v containing x and its parent p
- If v is a left child of p, then return $p \to min$
- Return $p \to max \to next$

Complexity

- Space: $\mathcal{O}(U)$
- SUCC: $\mathcal{O}(\log U)$
- INSERT, DELETE: $\mathcal{O}(\log U)$

x-fast trie: Improvements

Second step: Speed up

- The largest empty subtree v containing x can be found using binary search
- Time complexity of SUCCis $\mathcal{O}(\log \log U)$

Third step: Space reduction

- The number of nodes of T with non-empty subtree is at most nw
- Store them in a hash table (e.g. Cuckoo)

x-fast trie

- Space complexity $\mathcal{O}(nw)$
- SUCC: $\mathcal{O}(\log \log U)$ worst case
- INSERT, DELETE: $\mathcal{O}(\log U)$ expected amortized

y-fast trie (Willard [4])

Structure

- Split the ordered set S into $\Theta(n/B)$ blocks S_i of size $\Theta(B)$ where $B = \log U$
- For every block S_i choose a representative r_i and let R be the set of all representatives
- Create x-fast trie for representatives R, called a primary tree
- For every block S_i, create a balanced search tree T_i, called block trees

Space complexity

- Every elements of S is stored in one node of one block tree
- Space complexity of al block trees is $\mathcal{O}(n)$
- Space complexity of the primary tree is $\mathcal{O}(|R|w) = \mathcal{O}(n)$ since $|R| = \Theta(n/\log U)$ and $|w| = \Theta(\log U)$

y-fast trie: Operations FIND and SUCC

Operation FIND(x)

- Find the predecessor r_i and the successor r_j representatives of x in the primary tree
- Element x lives in T_i or T_j, so search both block trees ①

Time complexity

- Representative are found in time $\mathcal{O}(\log \log U)$
- Searching block trees takes $\mathcal{O}(\log |B|) = \mathcal{O}(\log \log U)$
- Time complexity of operation FINDis $\mathcal{O}(\log \log U)$

Operation SUCC(x)

- If $x \notin S$, then FIND(x) finds both the predecessor and the successor of x
- If $x \in S$, we use a linked list to find the successor
- Time complexity of operation SUCCis $\mathcal{O}(\log \log U)$

1. If x is a representative, we search only one tree.

y-fast trie: Operations INSERT and DELETE

Without balancing sizes of blocks

- Find the proper block tree: $\mathcal{O}(\log \log U)$ time
- Update the block tree: $\mathcal{O}(\log B) = \mathcal{O}(\log \log U)$ time
- However, the size of blocks must be $\mathcal{O}(\log B)$ and number of blocks must be $\mathcal{O}(n/\log B)$

Balancing sizes of blocks

- If a block is too large, split it
- If a block is too small, merge it with a sibling or move $\Omega(B)$ elements from a sibling
- Updates blocks trees takes $\mathcal{O}(B)$ time
- Updates the primary tree takes $\mathcal{O}(\log U)$ time
- Homework: Create rules which ensures that balancing occur once every $\Omega(B)$ updates
- Amortized cost of balancing is $\mathcal{O}(1)$
- Time complexity is $\mathcal{O}(\log \log U)$ expected amortized ①

1. Remind that x-fast trie uses Cuckoo hashing

RAM as a vector computer

Representation of a vector

- Consider b-bits integers $x_0, \ldots, x_{n-1}$
- Every integers is prepended by 0 and concatenated, i.e. a vector is represented as $\sum_{i=0}^{n} x_i 2^{i(b+1)}$
- Using linear compression theorem, it suffices to assume that $nb = \mathcal{O}(w)$

Operation Read(x,i)

- Returns the i-th integers from a vector x
- Read(x,i) = $x >> 2^{i(b+1)} \& 1^b$
- where 1^b is a binary number with b ones, i.e. $2^{b+1} - 1$

Operation Write(x,i,a)

- Write a into the i-th position of x
- Write(x,i,a) = $(x \& M_i) | (a << 2^{i(b+1)})$
- where $M_i = 1^{(b+1)(n-i-1)} 0^{b+1} 1^{(b+1)i}$ is a mask cleaning the i-th position

RAM as a vector computer

Replicate(a)

- Create n copies of a
- Replicate(a) = $a \cdot (0^b 1)^n$

Sum(x)

- Return the sum $\sum_{i=0}^{n-1} x_i$
- Sum(x) = $x \mod 2^{b+1} - 1$
- $x = \sum_{i=0}^{n-1} x_i 2^{b+1} \equiv \sum_{i=0}^{n-1} x_i \mod 2^{b+1} - 1$

Cmp(x,y)

- return z such that $z_i = 1$ if $x_i \geq y_i$ and 0 otherwise
- in the difference $(1x_i) - (0y_i)$, the first bit is 1 if and only if $x_i \geq y_i$
- Cmp(x,y) = $(((x|(10^b)^n) - y) >> b) \& (0^b 1)^n$

Min(x,y)

- $m = Cmp(x, y) \cdot 1^b$
- m filters positions of x where $x_i \geq y_i$
- Min(x,y) = $(x \& \sim m) | (y \& m)$

RAM as a vector computer

Rank(x,a)

- The number of elements of x smaller than a
- Rank(x,a) = Sum(Cmp(x,Replicate(a)))

Insert(x,a)

- Insert a into a sorted vector x
- i = Rank(x,a) is the possition for a in x
- Filter and move $x_i, \ldots, x_{n-1}$ to left

RAM as a vector computer

Unpack(a)

- Create a vector x such that x_i equals to the i-th bit a_i of a
- y = Replicate(a) $\&(2^{b-1}, \ldots, 1)$, i.e. $y_i = 2^i a_i$
- Unpack(a) = Cmp(y, $(0^b 1)^n$)
- Unpack$_\pi$(a) according to to a permutation π

Pack(a)

- Consider that b is decreased by one and apply Sum
- Since the vector is not properly formatted, modulo cannot be used
- Pack(a) = $((a \cdot (0^{b-1} 1)^n) >> bn) \& 1^b$

RAM as a vector computer assuming $w \geq \Omega(b^2)$

Weight(a)

- The Hamming weight of a
- Weight(a) = Sum(Unpack(a))

Permute$_\pi$(a)

- Permute bits in a according to a permutation π
- Permute$_\pi$(a) = Pack(Unpack$_\pi$(a))

LSB(a)

- The least significant bit in a
- LSB(a) = Weight($a \oplus (a-1)$)-1

MSB(a)

- MSB(a) = b-1-LSB(Permute$_{revers}$(a))

RAM as a vector computer

MSB(a) on $\mathcal{O}(w)$ bits

- $b = \lfloor \sqrt{w} \rfloor$
- Split a into l block of b bits and padding
- $x = (a\&(01^b)^l)|((a\&(10^b)^l) >> b)$
- Now, padding is zero and $x_i = 0$ if and only if i-th block is zero
- y = Cmp(x,$(0^b1)^n$)
- $y_i = 1$ if if and only if i-th block is non-zero
- j = MSB(Pack(y))
- j is the first non-empty block
- $z = (a >> (b+1)j)\&1^{b+1}$
- z is the content of the j-th block
- MSB(a) = $(b+1)j$+ MSB(z)

Fusion trees (Fredman, Willard, 1993 [1])

Fusion node

- Let $k = \mathcal{O}\left(w^{1/5}\right)$
- A fusion node stores keys $x_0, \ldots, x_{k-1}$
- A fusion node can find Rank, Succ and Pred in $\mathcal{O}(1)$
- Construction time is Poly(k)

Fusion tree

- B-tree for $B = k$
- Nodes are Fusion nodes
- The depth of the tree is $\mathcal{O}(\log_B(n)) = \mathcal{O}\left(\frac{\log n}{\log w}\right)$
- Time complexity of Rank, Succ and Pred is $\mathcal{O}\left(\frac{\log n}{\log w}\right)$

Fusion trees

Fusion node

- Consider a trie on keys $x_0, \ldots, x_{k-1}$
- The number of leaves is k, so the number of splitting nodes is $k-1$ ①
- Hence, at most $k-1$ levels has a splitting node ②
- Let $s(x)$ select all splitting bits from the binary code of x ③
- Observe that $s(x_0) < s(x_1) < \cdots < s(x_{k-1})$
- $S = (s(x_0), \ldots, s(x_{k-1}))$
- The number of bits of S is at most $k^2 = \mathcal{O}\left(w^{2/5}\right)$ so S can be stored in a single w-word
- Rank(x) can be computed using Rank(s(x),S) in $\mathcal{O}(1)$
- However, we need to compute $s(x)$ in $\mathcal{O}(1)$
- We find $a(x)$ of length $\mathcal{O}(k^4)$ containing $s(x)$ with extra zeros on the same position of all x

1. Splitting node is a node of degree 2.
2. We call them splitting levels.
3. Note that every level of a trie corresponds to one bit in the binary code. Bits corresponding to splitting levels are called splitting bits. Since only splitting bits are significant, $s(x)$ select these splitting bits from x.

Fusion trees

Lemma

For every $b_0 < \cdots < b_{k-1}$ there exist $m_0 < \cdots < m_{k-1}$ such that

- $b_i + m_j$ are pair-wise different for all i and j
- $b_0 + m_0 < \cdots < b_{k-1} + m_{k-1}$
- $(b_{k-1} + m_{k-1}) - (b_0 + m_0) = \mathcal{O}(k^4)$

Splitting bits with extra zeros

- $y = x\& \sum_i 2^{b_i}$ filters all splitting bits
- $z = y \cdot \sum_j 2^{m_j} = \sum_i \sum_j x_{b_i} 2^{b_i + m_j}$
- There is no carry since $b_i + m_j$ are pair-wise different
- $a(x) = z\&(\sum_i 2^{b_i+m_i}) >> (b_0 + m_0)$

Content

Path queries in trees

Goals

- Store a tree with weights on vertices
- Query: Find the minimum on the path between given two vertices
- Local update: Change the weight on a given vertex
- Path update: Increase weights on all vertices on the path between given two vertices
- Cut: Split the tree by removing a given two vertices
- Link: Join two trees by adding an edge between two given vertices
- Evert: Change the root of a tree and the orientation of all edges on the path between old and new vertices

Queries in sequences

Static representation

- Store a sequence of weights $w_1, \ldots, w_n$ in an array
- Build a balances binary search tree with leaves corresponding to the weights
- Every vertex stores the minimum of all weights in its subtree

Query: Find the minimum in a given subsequence $w_i, \ldots, w_j$

- Process $\mathcal{O}(\log n)$ nodes on paths from w_i and w_j to the root
- Time complexity is $\mathcal{O}(\log n)$

Local update of a given weight w_i

- Update the minimum in all vertices on the path from w_i to the root
- Time complexity is $\mathcal{O}(\log n)$

Queries in sequences

Increase all weights in a subsequence $w_i, \ldots, w_j$

- Lazy evaluation: Vertex can store an increment of all weight in its subtree
- Invariant: The actual weight w_k is the sum of the value stored in k-th leave and all increments on the path to the root
- Increment propagation: The increment stored in a vertex u can be added to increments in children and cleared in u
- All operations propagate increment on accessed vertices
- Subsequence increase: Set the increase in nodes on paths from w_i and w_j to the root
- Time complexity is $\mathcal{O}(\log n)$

Heavy-light decomposition of rooted trees

Heavy and light edges

- Let $s(v)$ be the number of vertices in the subtree of v (including v)
- Let u be a vertex and v its child
- The edge uv is called *heavy* if $s(u) \geq s(v)/2$; and *light* otherwise

Observations

- Every vertex u has at most one child v such that uv is heavy
- Heavy edges form a set paths
- Every vertex lies on a heavy path since trivial paths are allowed
- For every vertex u the path from the root to u contains at most $\log n$ light edges and most $\log n$ prefixes of heavy paths

Heavy-light decomposition of rooted trees

Representation

Every vertex v stores:

- Its parent u and the list of its children
- Heavy/light flag of the edge uv
- Minimum of weights in the subtree
- First vertex of the heavy path containing v
- Order of v on the heavy path containing v
- Every heavy path has a DS for sequences
- In static version: Minimum of the prefix of heavy path to v

Heavy-light decomposition of rooted trees

Lowest Common Ancestor (LCA)

For given two vertices u and v find the last common vertex on path from the root to u and v

- On the path from u to the root, mark all heavy paths
- Find the first marked heavy path p from v to the root
- Using the order of vertices on p, determine the LCA
- Time complexity is $\mathcal{O}(\log n)$

Path query

Find the minimal weight on the path between given two vertices u and v

- Find $z = \text{LCA}(u,v)$
- The path $u \ldots z \ldots v$ contains:
  - $\mathcal{O}(\log n)$ light edges
  - $\mathcal{O}(\log n)$ prefixes of heavy paths
  - At most one subpath of a heavy path
- Time complexity
  - Static version: $\mathcal{O}(\log n)$
  - Dynamic version: $\mathcal{O}\left(\log^2 n\right)$

Link-Cut trees (Sleator, Tarjan)

Definition

- Trees are rooted
- Every edges is thick or thin
- Every vertex u has at most one child v such that uv is thick
- Thick edges form a set paths
- Every vertex lies on a thick path since trivial paths are allowed

Operations

- Expose(v): Makes the path from v to the root thick and all children of v thin
- Find the parent of v and the root of v
- Cut: Split the tree by removing a given two vertices
- Link: Join two trees by adding an edge between two given vertices
- Evert: Change the root of a tree and the orientation of all edges on the path between old and new vertices
- Find the weight of a path
- Find the minimum of all weight on a path
- Change the weight of vertex
- Increase all weights of vertices on a path

Content

Content

Content

Content

Content

[1] Michael L Fredman and Dan E Willard.
Surpassing the information theoretic bound with fusion trees.
Journal of computer and system sciences, 47(3):424–436, 1993.

[2] Torben Hagerup, Peter Bro Miltersen, and Rasmus Pagh.
Deterministic dictionaries.
Journal of Algorithms, 41(1):69–85, 2001.

[3] Peter van Emde Boas.
Preserving order in a forest in less than logarithmic time.
In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 75–84. IEEE, 1975.

[4] Dan E Willard.
Log-logarithmic worst-case range queries are possible in space θ (n).
Information Processing Letters, 17(2):81–84, 1983.