Деревья поиска для целых чисел

Россия, Санкт-Петербург

20 мая 2020

Описание задачи

- Структура данных хранит подмножество $A \subset U = \{0, 1, 2, \ldots, u-1\}$
  - для простоты можно считать, что $u = 2^b$.
- Запросы:
  - $\textsc{Insert}(x)$ — добавить число x ($A := A \cup \{x\}$);
  - $\textsc{Delete}(x)$ — удалить число x ($A := A \setminus \{x\}$);
  - $\textsc{Next}(x)$ — найти следующее после x ($\min\{y \in A \mid y > x\}$)
  - $\textsc{Prev}(x)$ — найти предыдущее перед x ($\max\{y \in A \mid y < x\}$)
- Умеем все операции за $\mathcal{O}(\log n) = \mathcal{O}(\log u)$.
- Сделаем за $\mathcal{O}(\log \log u)$.

Битовая строка

$A = \{0, 1, 3, 11, 12, 13, 15\}$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$i \in A$	1	1	0	1	0	0	0	0	0	0	0	1	1	1	0	1

- INSERT и DELETE за $\mathcal{O}(1)$;
- NEXT и PREV за $\mathcal{O}(u)$.

Дерево ван Эмде Боаса

$u = 16 = 2^b$, $b = 4$

0				1				2				$3 = \sqrt{u} - 1$			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

- Разбить на кластеры размера $\sqrt{u} = 2^{\frac{b}{2}}$:
  - в каждом кластере $\sqrt{u}$ элементов;
  - номер кластера определяется старшими $\frac{b}{2}$ битами.
- $x = \langle c, i \rangle$;
- $x = 1011$:
  - $c = 10$ — номер кластера;
  - $i = 11$ — номер элемента внутри кластера.

Дерево ван Эмде Боаса

$u = 16 = 2^b$, $b = 4$, $A = \{0, 1, 3, 11, 12, 13, 15\}$

0				1				2				$3 = \sqrt{u} - 1$			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
1	1	0	1	0	0	0	0	0	0	0	1	1	1	0	1
1				0				1				1			

class VEBTree<b>

1: clusters: VEBTree<$\frac{b}{2}$>[$2^{\frac{b}{2}}$]
2: summary: VEBTree<$\frac{b}{2}$>
3: min: int<b>
4: max: int<b>

- min **не** хранится в clusters и summary;
- если min = max, то clusters и summary не создается.
- Peter van Emde Boas (1975).

Поиск следующего: $\textsc{Next}(x)$

$\textsc{Next}(V, x = \langle c, i \rangle)$

1: **if** $x < V.\min$ **then**
2: **return** $V.\min$
3: **if** $i < V.\text{clusters}[c].\max$ **then**
4: **return** $\langle c, \textsc{Next}(V.\text{clusters}[c], i) \rangle$
5: **else**
6: $c' = \textsc{Next}(V.\text{summary}, c)$
7: **return** $\langle c', V.\text{clusters}[c'].\min \rangle$

- $T(b) = T(\frac{b}{2}) + 1 = \mathcal{O}(\log b) = \mathcal{O}(\log \log u)$.

Добавление: INSERT(x)

INSERT($V, x = \langle c, i \rangle$)

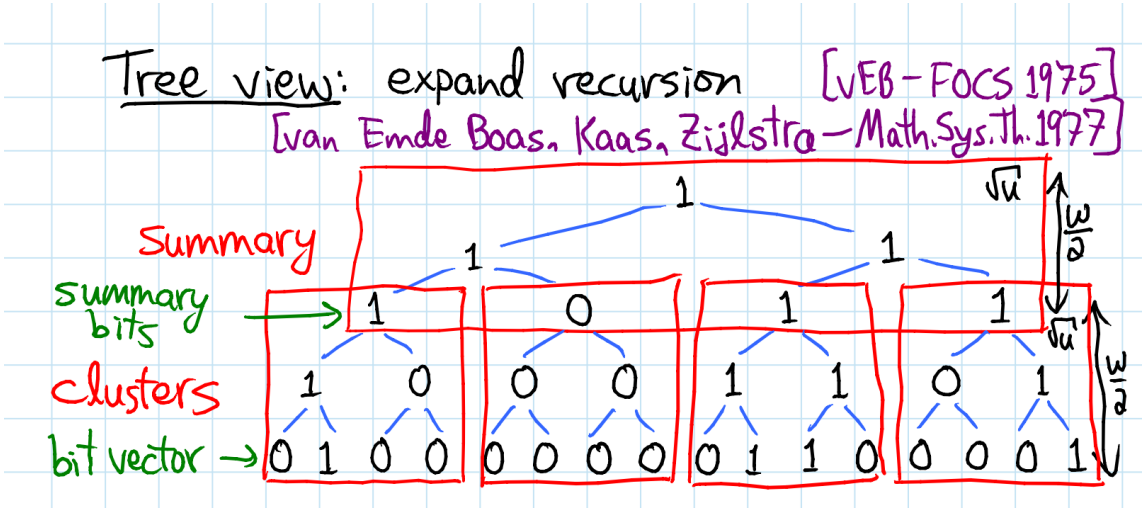
1: **if** V.min $= \perp$ **then**
2: $\quad V$.min $= V$.max $= x$
3: $\quad$ **return**
4: **if** $x < V$.min **then**
5: $\quad$ swap $x \leftrightarrow V$.min
6: V.max $= \max(V$.max$, x)$
7: **if** V.clusters$[c]$.min $= \perp$ **then**
8: $\quad$ INSERT(V.summary$, c$) *// следующий вызов за $\mathcal{O}(1)$*
9: INSERT(V.cluster$[c], i$)

- $T(b) = T(\frac{b}{2}) + 1 = \mathcal{O}(\log b) = \mathcal{O}(\log \log u)$.

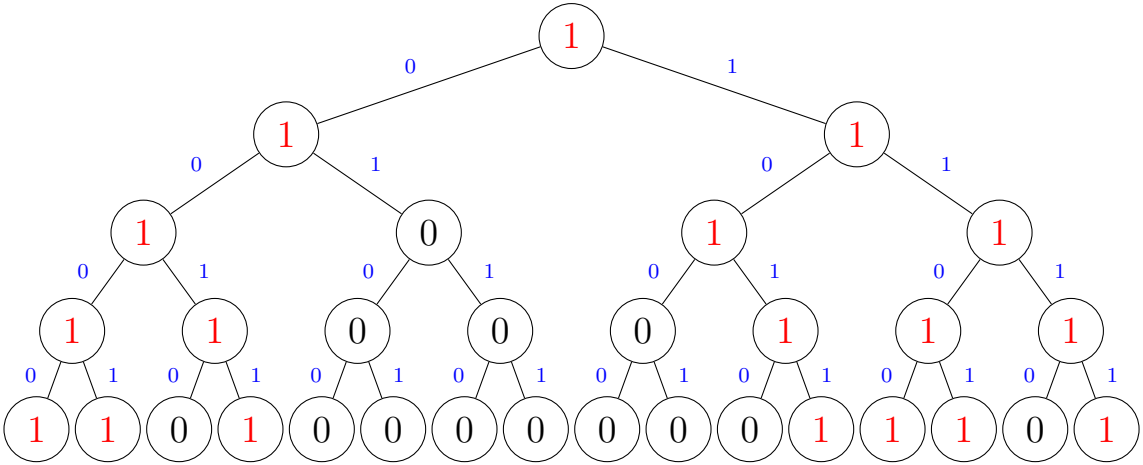
Удаление: DELETE(x)

```
DELETE(V, x = ⟨c, i⟩)
 1: if x = V.min then
 2:     c = V.summary.min                          // первый непустой кластер
 3:     if c = ⊥ then
 4:         V.min = V.max = ⊥                      // удалился последний элемент
 5:         return
 6:     x = V.min = ⟨c, i = V.clusters[c].min⟩     // удаляем новый минимум
 7: DELETE(V.clusters[c], i)
 8: if V.clusters[c].min = ⊥ then
 9:     DELETE(V.summary, c)                       // кластер стал пустым
10: if V.summary.min = ⊥ then
11:     V.max = V.min                              // все кластеры пустые
12: else
13:     c' = V.summary.max                         // ищем новый максимум
14:     V.max = ⟨c', V.clusters[c'].max⟩
```

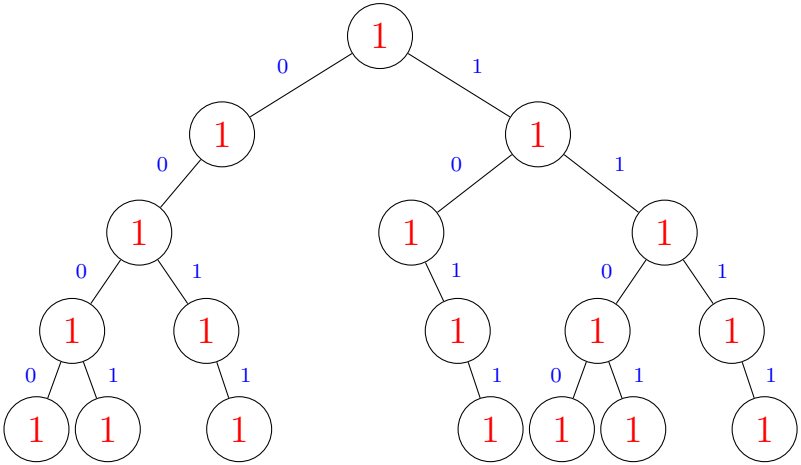
X-fast tree


Tree view: expand recursion
[vEB – FOCS 1975]
[van Emde Boas, Kaas, Zijlstra – Math.Sys.Th. 1977]
Summary
summary bits
clusters
bit vector
1
1 1
1 0 1 1
1 0 0 0 1 1 0 1
0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1
√u
w/2
√u
w/2

X-fast tree

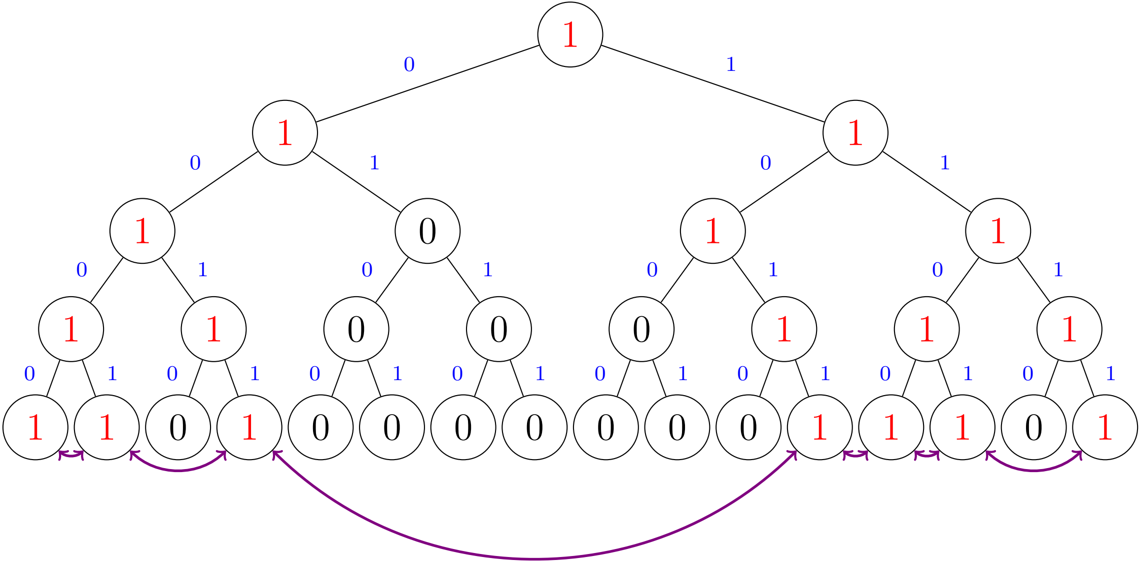

1
0
1
1
0
1
1
0
1
1
0
1
1
0
1
1
0
1
1
0
1
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
1
1
0
1
0
0
0
0
0
0
0
1
1
1
0
1

X-fast tree



Dan E. Willard (1982) (aka X-fast trie)

X-fast tree


1
0 1
1 1
0 1 0 1
1 0 1 1
0 1 0 1 0 1 0 1
1 1 0 0 0 1 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 1 0 1 0 0 0 0 0 0 0 1 1 1 0 1

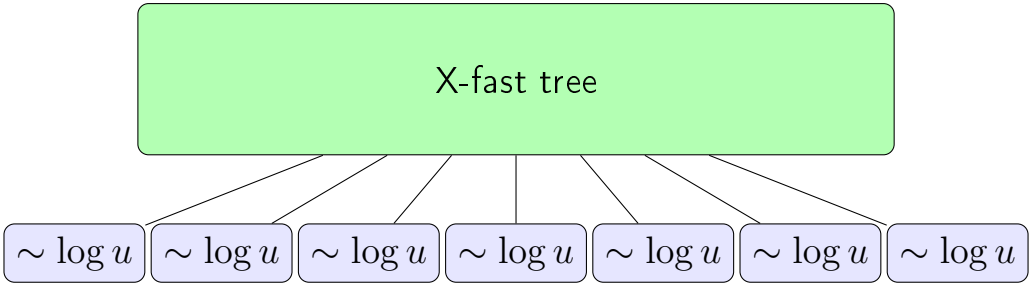
X-fast tree

- Каждая вершина — число длины не больше $b = \log_2 u$;
  - числа меньше $u = 2^b$.
- Сохраним все вершины в хеш-таблицу:
  - число в хеш-таблице $\implies$ вершина есть в дереве.
- Добавление и удаление просто спуском:
  - $\log_2 u$ добавлений или удалений в хеш-таблице.
- Двусвязный список на существующих листьях.
- Поиск пред/след $\iff$ поиск существующего предка;
  - в другом поддереве максимум/минимум $\iff$ пред/след;
  - в каждой вершине храним мин/макс лист в поддереве.
- Поиск другого соседа: по ссылке от найденного.

X-fast tree

- Поиск след/пред $\mathcal{O}(\log \log u)$;
- Добавление $\mathcal{O}(\log u)$;
- Удаление $\mathcal{O}(\log u)$;
- Память $\mathcal{O}(n \log u)$.

Y-fast tree



- Разобьем множество A на подряд идущие блоки:
  - размер блоков от $\frac{1}{2}\log_2 u$ до $2\log_2 u \implies$ всего $\sim \frac{n}{\log u}$ блоков.
- Из каждого блока выберем один элемент, например, минимальный;
  - построим X-fast tree из этих $\sim \frac{n}{\log u}$ элементов;
  - дерево поиска для каждого блока: каждая операция за $\mathcal{O}(\log \log u)$.
- Dan. E. Willard (1982) (aka Y-fast trie)

Y-fast tree: INSERT и DELETE

- Добавление:
  - найти блок, в который добавить:
    - $\textsc{Prev}(x)$ в X-fast tree $\mathcal{O}(\log \log u)$;
  - добавить в этот блок: $\mathcal{O}(\log \log u)$ — INSERT в дерево размера $\sim \log u$.
  - Может нарушится инвариант, размер блока стал $> 2 \log_2 u$:
    - разделить на два блока размера от $\log_2 u$ до $2 \log_2 u$ за $\mathcal{O}(\log u)$;
    - добавить элемент в X-fast tree за $\mathcal{O}(\log u)$;
    - исправление инварианта: амортизированно за $\mathcal{O}(1)$.
- Удаление, аналогично:
  - найти блок, в котором элемент:
    - $\textsc{Prev}(x)$ в X-fast tree $\mathcal{O}(\log \log u)$;
  - удалить из него: $\mathcal{O}(\log \log u)$ — DELETE из дерева размера $\sim \log u$.
  - Может нарушиться инвариант, размер блока стал $< \frac{1}{2} \log_2 u$:
    - объединить с одним из соседних блоков;
    - сделать один или два блока размерами: $> \log_2 u$;
    - удалить (и добавить) элемент в X-fast tree за $\mathcal{O}(\log u)$;
    - исправление инварианта: амортизированно за $\mathcal{O}(1)$.

Y-fast tree: PREV и NEXT

- NEXT(x):
  - Найти в каком блоке: PREV(x) в X-fast tree: $\mathcal{O}(\log \log u)$;
  - Найти следующий в этом блоке $\mathcal{O}(\log \log u)$
    - в дереве поиска размера $\sim \log u$.
  - Если следующего нет, то искать минимум в следующем блоке;
    - $\mathcal{O}(\log \log u)$, минимум в дереве поиска размера $\sim \log u$.
- PREV(x), аналогично.
- Память $\mathcal{O}(n)$:
  - X-fast tree: $\mathcal{O}(\frac{n}{\log u} \log u) = \mathcal{O}(n)$;
  - Деревья поиска: $\mathcal{O}(n)$ суммарно.