Дървета на ван Емде Боас

20.11.2020 г.

(van Emde Boas Trees)

Дадено: $U \in \mathbb{N}$.

Динамични заявки към множество $S \subseteq \{0,1,...,U-1\}$

1. $pred_S(x)$
2. $succ_S(x)$
3. $search_S(x)$: "$x \in S$"
   $x \in \{0,...,U-1\}$:
4. $insert(S,x)$: $S \leftarrow S \cup \{x\}$
5. $delete(S,x)$: $S \leftarrow S \setminus \{x\}$

Решение:
Наивно:

Сортиран списък за S

Сортиран масив за S

Балансирани наредени дървета $O(\log|S|)$ за всяка една от заявките.

Ако $x_1, x_2, \dots, x_n$ са произволни (различни) числа може да ги сортираме по следния начин:

$S \leftarrow \emptyset$
for $i = 1$ **to** n **do**
    $S \leftarrow Insert(S, x_i)$
$y_1 \leftarrow succ_S(0)$
for $i = 1$ **to** $n-1$ **do**
    $y_{i+1} \leftarrow succ_S(y_i + 1)$

Ако числата са естествени и различни, то горната процедура ще ги сортира. Операциите $Insert$ и $succ$ може да ги ограничим с $\log(n)$.

В общия случай не може да осигурим нищо по-добро от $O(\log|S|)$ за заявка.

С ограничението обаче за U с дърветата на ван Емде Боас получаваме ограничение за всяка една от двете заявки от $O(\log\log U)$.

$x = \overline{x_1 x_2 \ldots x_u}$

В алгоритъма на Willard направихме двоично търсене по думата образувана от двоичния запис на числото x, за да намерим най-дългия ѝ префикс, който се съдържа в дървото от префикси (се среща в нашето множество). Но нека сега не избързваме толкова с това търсене, а разгледаме само първата му стъпка:

$$\underset{l}{x_1} x_2 \ldots \downarrow \ldots \underset{r}{x_u}$$

$$m = \left[\frac{l+r}{2}\right]$$

1 сл. $x_1 x_2 \ldots x_m$ се среща в дървото, тогава търсим продължение $x_{m+1}, \ldots, x_r$ и
2 сл. $x_1 x_2 \ldots x_m$ не се среща в дървото и тогава търсим съкращаване което се съдържа в $x_1, \ldots, x_m$

Това разделяне на висши и нисши битове ще ни послужи за изграждане на идеята. То както видяхме съответства на делене с остатък.

Основното наблюдение в алгоритъма на ван Емде Боас е следното аритметично наблюдение:
<u>Твърдение</u>: Ако $0 \le x < 2^u$, то x се представя по единствен начин като $x = 2^{u_1} x_a + x_b$, като
1) $u_1 = [u/2]$
2) $0 \le x_b < 2^{u_1}$
3) $0 \le x_a < 2^{u-u_1}$

<u>Доказателство</u>: Делим x с частно и остатък на 2^{u_1}. Тогава $x = 2^{u_1} x_a + x_b$, като $0 \le x_b < 2^{u_1}$.
Накрая, ако $x_a \ge 2^{u-u_1}$, то $x \ge 2^{u_1} x_a + 0 \ge 2^{u_1} 2^{u-u_1} = 2^u$.
$\Rightarrow$ противоречие с $x < 2^{u_1}$.

Идея за търсене $x \in S$ и $0 \le x < 2^u$.

1. Представяме $x = 2^{u_1} a + b$. За всяко $0 \le a < 2^{u-u_1}$:$R[a] = \{0 \le b < 2^{u_1} \mid 2^{u_1} a + b \in S\}$
2. $S' = \{0 \le a < 2^{u-u_1} \mid R[a] \neq \emptyset\}$
   Търсим x_b в множеството $R[x_a]$, като параметъра в този случай ще е $u_1 = [u/2]$

Тоест, ако успеем да организираме нашата структура от данни по начин, по който да имаме тези множества подготвени по удобен начин, така че да може рекурсивно да може да задаваме тези заявки - на всяка стъпка параметъра u, който съответства на броя битове намалява два пъти. В резултат на което имаме логаритмичен процес за числата по броя битове в x, но x ще бъде в зададения интервал, то спрямо U ще имаме $\log \log U$.

Разклонеността на дърветата на дърветата на ван Емде Боас е различна във всяко ниво.

Естествения въпрос е каква ще е големината на тази дървовидна структура.

S е множество с елементи от $\{0,1,...,2^u - 1\}$

struct $vEBTree$ {
    int u
    int $U = 2^u$
    int $size$ // $|S|$
    int $\max$ // $\max S$
    int $\min$ // $\min S$

    $vEBTree$ * S' // с параметър $u' = u - [u/2]$
    $vEBTree$ * $R[0...2^{u'-1}]$ // масив от указатели към такива структури като всяко
                    // $R[k]$ е с параметър $[u/2]$
}

Разбира се при $u = 0,1$ поддържаме елементите на S експлицитно в масив с 4 елемента $arr[0..3]$ // $arr[i] = 0 \Leftrightarrow i \notin S, arr[i] = 1 \Leftrightarrow i \in S$.

<u>Твърдение</u>:
За всяко u, празно дърво на ван Емде боас с параметри u, $U = 2^u$ заема памет $O(U)$ и може да се построи за такова време .

<u>Доказателство</u>:
$Space(2^u) \le c + Space(2^{u-[u/2]}) + 2^{u-[u/2]}Space(2^{[u/2]})$. Целта е да покажем, че тази памет е линейна.
С индукция по u може да покажем, че $Space(2^u) \le c_0(2^u - 2)$.
Базата е тривиална, защото лесно може да си изберем c_0, така, че да ни бъде удобно.
Индуктивна стъпка: нека $u_1 = [u/2]$. Тогава

$$Space(2^u) \overset{\substack{induction\\ hypothesys\\ for\ u_1\ and\ u-u_1}}{\le} c + c_0(2^{u-u_1} - 2) + 2^{u-u_1}c_0(2^{u_1} - 2) =$$
$$= c_0 2^u - \cancel{2}c_0 2^{u-u_1} + \cancel{c_0 2^{u-u_1}} - 2c_0 + c = c_0 2^u - 2c_0 + \underbrace{c - c_0 2^{u-u_1}}_{u_1<u} <$$
$$< c_0 2^u - 2c_0 = c_0(2^u - 2).$$

$U \in \mathbb{N}$

динамично $S \subseteq \{0,1,...,U-1\}$

- $search(S,x)\ :\ x \in S?$
- $pred(S,x)\ :\ y = max\{z \in S \,|\, z \leq x\}$
- $succ(S,x)\ :\ y = min\{z \in S \,|\, z \geq x\}$
- $insert(S,x)\ : S \leftarrow S \cup \{x\}$
- $delete(S,x)\ :\ S \leftarrow S \backslash \{x\}$

В основата на алгопритъма на ван Емде Боас е следната фина идея (разделяй и владей относно двоичния запис на числото $x \in \{0,1,...,U-1\}$):

$U \leq 2^u$

$x = a_x 2^{[u/2]} + b_x$, където $0 \leq b_x < 2^{[u/2]}$ и $0 \leq a_x < 2^{u-[u/2]}$

За $S\ : 0 \leq a < 2^{u-[u/2]}$ дефинираме $R[a] = \{b < 2^{[u/2]} \,|\, a2^{[u/2]} + b \in S\}$ и

$$S' = \{a < 2^{u-[u/2]} \,|\, R[a] \neq \emptyset\}$$

Допълнително имахме размера (броя на елементите на множество S):

- $size(S)$

И също така в интерес на ефективността за делението с частно и остатък:

- $u;\ U = 2^u;\ U_1 = 2^{[u/2]}$
- $\max(S);\ \min(S)$

Това е представянето за $u > 2$, докато за $u \leq 2$ правим наивно представяне с масив.

$U = 64,\ u = 6$

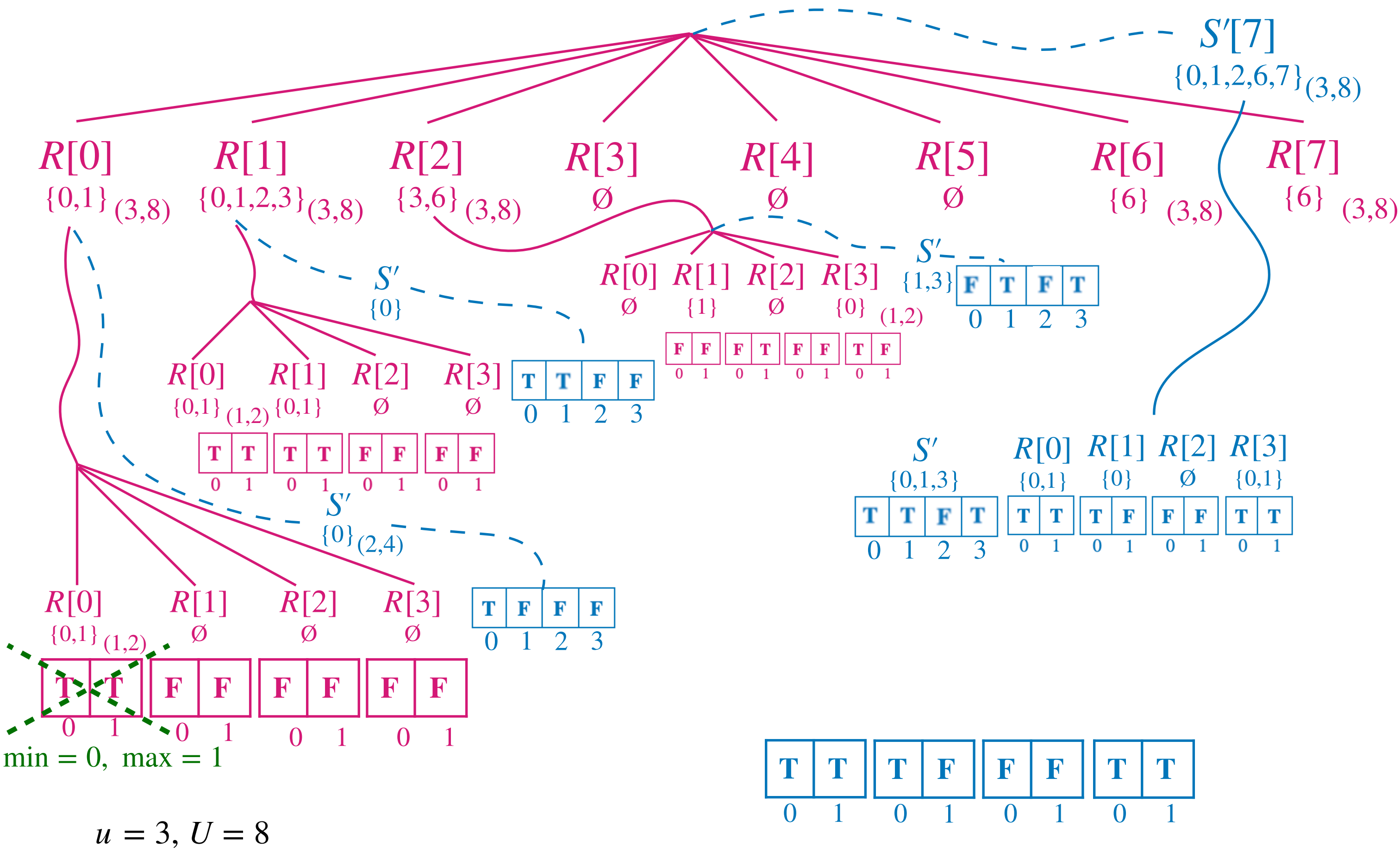
$S = \{0, 1, 8, 9, 10, 11, 19, 22, 54, 62\}$

$u_1 = [u/2] = 3$

$U_1 = 2^{u_1} = 8$

$$\begin{aligned}
0 &= \boxed{0} \times 8 + 1 \\
1 &= \boxed{0} \times 8 + 1 \\
8 &= \boxed{1} \times 8 + 0 \\
9 &= \boxed{1} \times 8 + 1 \\
10 &= \boxed{1} \times 8 + 2 \\
11 &= \boxed{1} \times 8 + 3 \\
19 &= \boxed{2} \times 8 + 3 \\
22 &= \boxed{2} \times 8 + 6 \\
54 &= \boxed{6} \times 8 + 6 \\
62 &= \boxed{7} \times 8 + 6
\end{aligned}$$

$S = \{0,\ 1,\ 8,\ 9,\ 10,\ 11,\ 19,\ 22,\ 54,\ 62\}$



$u = 3,\ U = 8$

$u_1 = 1,\ U_1 = 2$

$R[0]$	$R[1]$	$R[2]$	$\cdots$	S'
$0 = 0 \times 2 + 0$	$0 = 0 \times 2 + 0$	$3 = 1 \times 2 + 1$		$0 = 0 \times 2 + 0$
$1 = 0 \times 2 + 1$	$1 = 0 \times 2 + 1$	$6 = 3 \times 2 + 0$		$1 = 0 \times 2 + 1$
	$2 = 0 \times 2 + 2$			$2 = 1 \times 2 + 0$
	$3 = 0 \times 2 + 3$			$6 = 3 \times 2 + 0$
				$7 = 3 \times 2 + 1$

<u>Забележка</u>: Височината на дърво на ван Емде Боас за $U = 2^u$ (макс. бр. ел.) е $O(\log u) = O(\log\log U)$, т.е. дървото е доста плитко. Това се дължи на разклоняването с високи темпове по етажите на дървото.

procedure $search(S, x)$ {
    if $u(S) \leq 2$ **then**
        return $arr_S[x]$
    $a_x \leftarrow [X/U_1(S)]$
    $b_x \leftarrow x \mod (U_1(S))$
    return $search(R[a_x], b_x)$
}

Сложност: $O(\log u(S)) = O(\log\log U(S)) = O(\log\log U)$

Ясно е че двете заявки съотв. за предшественик и наследник са интересни само когато елемента, за който ги търсим не е в дървото, тъй като тогава отговора ще е тривиален и реално на него ще е също толкова ефективно да се отговори и със заявката за $search/member$.

Пример: $13 = 1 \times 8 + 5$
$pred_S(13)$
$5 = 2 \times 2 + 1$
2 – към S'
Търсим предшественика във S' на 2: $pred_{S'}(2) = 1$
$pred_{R[0]}(5) = 1 \times 2 + \max \underbrace{\underset{=1}{R[1]}}_{lower\ R[1]} = 1 \times 2 + 1 = 3$

$\Rightarrow 1 \times 8 + 3 = 11.$

- $pred(S, x)$
- $pred\_naive(S, x)$
  $U(S) \leq 4$

procedure $pred(S, x)$ {
  if $size(S) = 0$ **then**
    return -1
  if $x \geq \max(S)$ **then**
    return $\max(S)$
  if $x < \min(S)$ **then**
    return -1
  if $U(S) \leq 4$ **then**
    return $pred\_naive(S, x)$
  $a_x \leftarrow [x/U_1(S)]$
  $b_x \leftarrow x \mod U_1(S)$
  if $size(S.R[a_x]) \neq 0$ **and** $\min(S.\widetilde{R}[a_x]) \leq b_x$ **then**
    $a' \leftarrow a_x$
    $b' \leftarrow prev(S.\widetilde{R}[a_x], b_x)$
  else
    $a' \leftarrow prev(S.\widetilde{S}', a_x - 1)$
    $b' \leftarrow \max(\widetilde{R}[a'])$
  return $a'.U_1(S) + b'$
}

Знаем, че:
$\min(S) \leq x < \max(S)$
$x = a_x U_1(S) + b_x$

if $a' = -1$ **then**
  return $\min(S)$

Сложност: Сложността на алгоритама е очевидно толкова колкото е дълбочината на дървото, тъй като имаме толкова рекурсивни извиквания (най-много едно за всяко ниво) колкото е височината на дървото, а останалите съпровождащи операции се изпълняват за константно време. Броят на битовете в $u(S)$ намалява с единица след всяко извикване на рекурсивната функция.

Следователно сложността на процедурата за намиране на предшественик на x е $O(\log u(S)) = O(\log \log U(S)) = O(\log \log U)$.

Коректност: Ще подходим индуктивно по нивата на дървото.

1 сл. Да допуснем, че $x < \max(S)$ и $x \geq \min(S)$

$R[a_x] \neq \emptyset$
$b_x \geq \min(R[a_x])$

От индукционната хипотеза имаме, че $b = pred_{R[a_x]}(b_x)$.
Твърдим, че $a_x U_1(S) + b' = a' U_1(S) + b' = pred_S(x)$.
От $b' \leq b_x \Rightarrow a_x U_1(S) + b' \leq x$.
Да допуснем, че има елемент в нашата структура между горните два, т.е.:
$a_x U_1(S) + b' \leq a'' U_1(S) + b'' \leq x = a_x U_1(S) + b_x$ и това което знаем е, че
$0 \leq b', b_x < U_1(S) \Rightarrow$ делейки целочислено на $U_1(S)$ получаваме, че
$a_x \leq a'' \leq a_x \Rightarrow a'' = a_x$ и ако $a'' U_1(S) + b'' \in S \Leftrightarrow b'' \in R[a_x]$. Сега вече е ясно, че
$a_x U_1(S) + b' \leq \underset{a_x}{a''} U_1(S) + b'' \leq a_x U_1(S) + b_x \Leftrightarrow b' \leq b'' \leq b_x$. Но
$b' = pred_{R[a_x]}(b) \Rightarrow \underset{\mathbf{if}\ a''U_1(S)+b''\in S}{b'' = b'}$.

2 сл. Отново може да предполагаме, че $\min(S) \leq x < \max(S)$, но този път се е случило така, че $R[a_x] = \emptyset$ или $b_x < \min(R[a_x])$
$a' = pred_{S'}(a_x - 1)$
$b' = \max(R[a'])$
Твърдим, че $a' U_1(S) + b' = pred_S(x)$
1). $a' < a_x$ и $b' < U_1(S) \Rightarrow$
$a' U_1(S) + b' \leq (a_x - 1) U_1(S) + b' < (a_x - 1) U_1(S) + U_1(S) = a_x U_1(S) \leq x$.

Нека $y \in S$, $y = a'' U_1(S) + b''$, $0 \leq b'' < U_1(S)$ и
$a' U_1(S) + b' \leq y \leq x \Rightarrow a' U_1(S) + b' \leq a'' U_1(S) + b'' \leq a_x U_1(S) + b_x \Rightarrow$

$\underset{b',\, b'',\, b_x < U_1(S)}{\overset{\mathbf{divide\ to}\ U_1(S)}{\Rightarrow}}$ $a' \leq a'' \leq a_x$. Но $a' = pred_{S'}(a_x - 1)$ и $a'' \in S' \Rightarrow$

$\Rightarrow$1.) $a'' = a'$ $(a'' \leq a_x - 1)$ или 2.) $a'' = a_x$

1 сл. $a'' = a'$: $b'' \in R[a''] = R[a'] \Rightarrow$
$\Rightarrow b'' \leq \max(\underbrace{R[a'']}_{=R[a']}) = b' \Rightarrow a'' U_1(x) + b'' \leq a' U_1(x) + b' \Rightarrow$
$\Rightarrow y = a' U_1(x) + b'$.

2 сл. $a'' = a_x : b'' \in R[a_x]$. Тъй като $y \leq x \Rightarrow b'' \leq b_x < \min(R[a_x]) \Rightarrow$ ↯ противоречие с това, че $b_x \in R[a_x]$!

- Добавяне на елемент

  $x \notin S\ ;\ x = a_x U_1(S) + b_x.$

$R[a] = \{b < U_1(S) \,|\, aU_1(S) + b \in S\}$
$S' = \{a \,|\, R[a] \neq \emptyset\}$

  - ако $R[a_x] \neq \emptyset$, добавяме b_x към $R[a_X]$;
  - ако $R[a_x] = \emptyset$; първо добавяме a_x към S' и второ добавяме b_x към $R[a_x]$.

На пръв поглед може да си кажем, че вместо едно добавяне правим две добавяния и промяната е дребна. Но тези добавяния са рекурсивни операции и не са атомарни, което означава, че ако запишем рекурентното уравнение което съответства на времевата сложност във втория случай, бихме получили:
$Time\big(U(S)\big) = 2Time\left(\dfrac{U(S)}{2}\right)$, което е ралично от $Time\big(U(S)\big) = Time\left(\dfrac{U(S)}{2}\right) + 1$, както то е при първия случай и там получаваме логаритмична сложност, а в първото уравнение ще получим линейна сложност.

Как ще се справим с този проблем? Справянето с този проблем е свързан с едно простичко наблюдение: **ако се налага да добавяме $\mathbf{a_x}$, то $\mathbf{R[a_x]} = \emptyset$ е празно**. При това положение b_x ще бъде единствения елемент в $R[a_x]$ след добавянето и той ще бъде и $\max$ и $\min$.

$\tilde{R}[a] = \{b < U_1(S) \,|\, aU_1(S) + b \in S \setminus \min(S)\ \&\ \max(S)\}$
$\tilde{S} = \{a \,|\, \tilde{R}[a] \neq \emptyset\}$

Може да се наложи да добавим x в $S : x < \underbrace{\min(S)}_{y}$ или $x > \max(S)$

$S \cup \{x\} = S.\ \big(S \backslash \{y\} \cup \{x\}\big) \cup \{y\}$

В случая когато $y = \min(S)$, оперативно това математическо равенство може да се изрази по следния начин:

$$
\begin{array}{l}
insert(S, x)\{ \\
\qquad \left.\begin{array}{l} y \leftarrow \min(S) \\ \min(S) \leftarrow x \end{array}\right\} \ \big(S \backslash \{y\} \cup \{x\}\big) \\
\qquad insert(S, y) \\
\}
\end{array}
$$

В общия случай, когато $\min(S) < x < \max(S)$:

$x = a_x U_1(S) + b_x$
- $\tilde{R}[a_x] \neq \emptyset;\ \tilde{R}[a_x] \cup \{b_x\}$
- $\tilde{R}[a_x] = \emptyset;\ \tilde{S}' \cup \{a_x\}$

  $\max(R[a_x]) = \min(\tilde{R}[a_x]) = b_x$

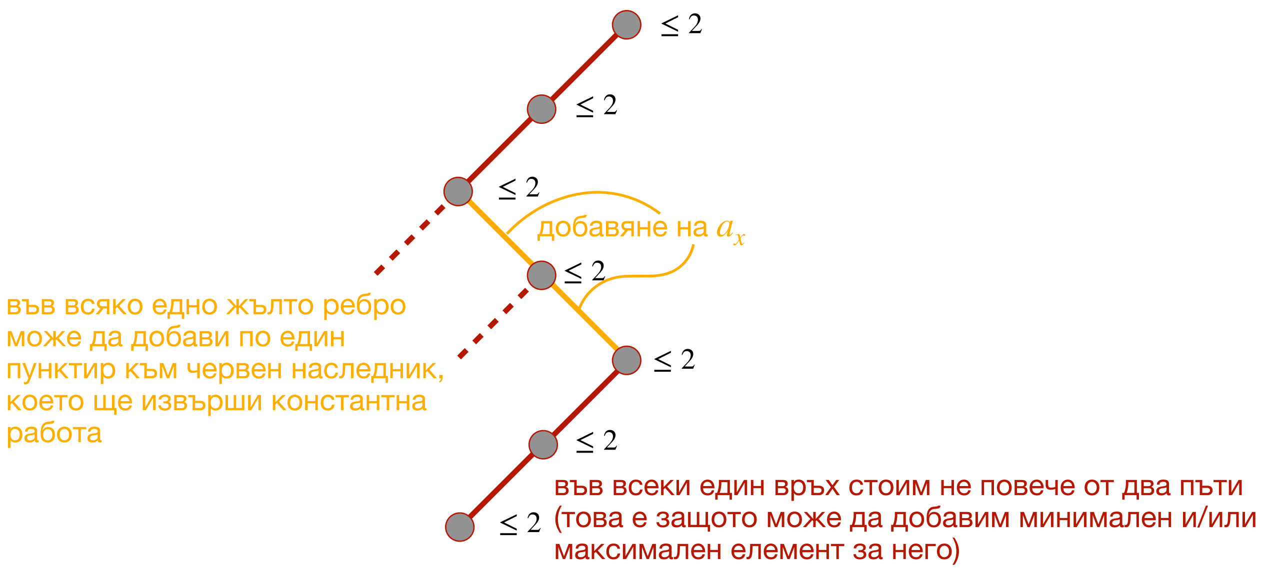
Предлагаме:
$insert\_naive(S, x)$ за $U(S) \leq 4$

procedure $insert(S, x)$ {
  if $U(S) \leq 4$ **then**
    return $inert\_naive(S, x)$
  if $size(S) < 2$ **then** {
    if $size(S) = 0$ **then** {
      $\min(S) \leftarrow x$
      $\max(S) \leftarrow x$
    } **else** { // $|S| = 1$, $min(S) = \max(S)$
$O(1)$ **if** $\min(S) < x$ **then**
        $\max(S) \leftarrow x$
      else
        $\min(S) \leftarrow x$
    }
    $size(S) = size(S) + 1$
    return
  }
$\min(S) < y < \max(S)$
  if $\max(S) < x$ **then**
    $y \leftarrow \max(S)$
    $\max(S) \leftarrow x$
    return $insert(S, y)$
  if $\min(S) > x$ **then**
    $y \leftarrow \min(S)$
    $\min(S) \leftarrow x$
    return $insert(S, y)$
  $a_x \leftarrow [x / U_1(S)]$
  $b_x \leftarrow x \mod U_1(S)$
  if $size(S\,.\,\tilde{R}[a_x]) = 0$ **then**
    $insert(S\,.\,\tilde{S}, a_x)$
  $insert(S, \tilde{R}[a_x], b_x)$
  $size(S) \leftarrow size(S) + 1$
}

Анализ на времето: Основния инвариант, който наблюдаваме е $u(S)$. Да видим какво се случва с него. В първите няколко **if**-а (първите 2) не се случва нищо и директно приключваме с процедурата за константно време. В трети и четвъртия **if** оператори, напървите редове имаме $O(1)$, но както отбелязахме и в двата случая $\min(S) < y < \max(S)$. Следователно, въпреки че броя на битовете не се променя при тези стъпки, рекурсивното извикване на функцията с y ще прескочи първата и втората част от програмата и ще се засили да изпълнява последната част от тази процедура. Така, че:

- за всяко S има най-много две последователни извиквания $insert(S, \cdot\,)$
- $u(\tilde{S}) \leq \left\lceil \dfrac{u(S)+1}{2} \right\rceil$

Рекурсивния процес може да се разглежда като разходка в дървото, като преминаваме от връх на връх. В един връх се задържаме най-много два поредни пъти след което продължаваме надолу в поддървото и изминаваме точно един път от корен до листо В някой случай този път може да бъде разклонен, но тогава пътя който ще получим ще изглежда по следния начин:



Общия брой преходи, които ще извърви нашия алгоритъм е еквивалентен на времевата сложност и тя е не повече от три пъти дължината на пътя от корена до листо в това дърво. А този път както видяхме има дължина от $O\big(\log u(S)\big) = O(\log\log U)$. Така, че най-много $3 \times \log\log U$ е сложността на алгоритъма, който описахме. Основната печалба идва от това, че минимума и максимума на нашето множество не се пазят експлицитно два пъти както беше в първоначалната реализация, а се пазят само по веднъж в самия връх.

Изтриването е дуална операция на вмъкването. Трудността ще дойде от това, че от време на време ще трябва да актуализираме минималния и максималния елемент в нашето множество.

Изтриване на елемент.

Предполагаме, че $x \in S$

- $delete\_naive(S, x)$ // $U(S) \leq 4$

  Отново представяме x по следния начин:

  $x = a_x U_1(S) + b_x$
- $\min(S) < x < \max(S)$

  $S \setminus x \sim \cdot\, \tilde{R}[a_x] \setminus \{b_x\}$

  $\cdot\, b_x$ - може да се окаже единствен в $\tilde{R}[a_x]$ и ако това се случи, тогава трябва да изтрием a_x от S': $S' \setminus \{a_x\}$
- $x = \min(S)$

  $\cdot\,\boxed{y = \min(S \setminus \{x\})\ ?}$ - в общия случай тази функция ще отнеме $\log\log U$ и ще получим две рекирсивни извиквания, което е лошо.

  Но ако за момент си мислим, че имаме y, тогава:

  $S \setminus \{x\} = S \setminus \{x, y\} \cup \{y\} = \big(S \setminus \{y\}\big) \setminus \{x\} \cup \{y\}$

  $delete(S, x)$

  $delete(S, y)$

  $\min(S) \leftarrow y$ // $\setminus \{x\} \cup \{y\}$

procedure $fast\_succ\_min(S)$ { // предполагаме, че $|S| \geq 2$
    if $size(S) = 2$ **then** // $S = \{\min(S),\ \max(S)\}$
        return $\max(S)$
    if $size(S) \geq 3$ **then** // $|S| \geq 3$
        $a \leftarrow \min(S.\tilde{S})$ // $\tilde{R}[a] \neq \emptyset$
        $b \leftarrow \min(S.\tilde{R}[a])$
    return $aU_1(S) + b$
}

Коректност: $\min(S) = a_x U_1(S) + b_x$. Останалите елементи на S ще имат или по-голямо a или същото a, но по-голямо b. Намирайки най-малкото a в множеството $\tilde{S}$ - намираме всъщност най-малкото a на елемент в множеството S, който е различен от минималния и е различен и от максималния, тъй като те са единствените, които не са представени по рекурсивен начин. Тъй като множеството S има поне три елемента - намираме най-малкото a, което е по-голямо или равно от a_x. И от тези, които имат едно и също a взимаме това с най-малкото b. Ясно е, че елемента, който ще получим ще бъде строго по-голям от минималния в множеството и други между него и минималния няма да има, защото те или биха имали по-малко a, което ще доведе до противоречие.

Дуално за предшественик на максимален елемент:

procedure $fast\_pred\_max(S)$ {
    if $size(S) = 2$ **then**
        return $\min(S)$
    $a \leftarrow \max(S.S')$
    $b \leftarrow \max(S.\tilde{R}[a])$
    return $aU_1(S) + b$
}

procedure $delete(S, x)$ {

    if $U(S) \leq 4$ **then**

        $delete\_naive(S, x)$

        return

    if $U(S) \leq 2$ **then** // $S = \{\min, \ \max\}$

        if $x = \min(S)$

            $\min(S) \leftarrow \max(S)$

        else

            $\max(S) \leftarrow \min(S)$

        $size(S) \leftarrow size(S) - 1$

        return

    // **if** $size(S) = 0$ **then** $\max(S) = -1; \ \min(S) \leftarrow +\infty$

    // $|S| \geq 3$

    if $\max(S) = x$ **then**

        $y \leftarrow fast\_pred\_max(S)$ // $\min(S) < y < \max(S)$

        $delete(S, y)$

        $\max(S) \leftarrow y$

        return

    if $\min(S) = x$ **then**

        $y \leftarrow fast\_succ\_min(S)$

        $delete(S, y)$

        $\min(S) \leftarrow y$

        return

    // ако сме стигнали до тук, чначи $\min(S) < x < \max(S)$

    $a_x \leftarrow [x / U_1(S)]$

    $b_x \leftarrow x \mod U_1(S)$

    // $b_x \in S.\tilde{R}[a_x]; \ a_x \in S.\tilde{S}'$

    $delete(S.\tilde{R}[a_x], b_x)$ // ($\star$)

    if $size(S.\tilde{R}[a_x]) = 0$ **then**

        $delete(S.\tilde{S}', a_x)$ // ако това се случи, то в ($\star$) сме имали времева сл. $O(1)$

    $size(S) \leftarrow size(S) - 1$

    return

}

$S \subseteq \{0,1,...,U-1\}$

$U = U(S)$

$u(S) = \log U(S)$

$U_1(S) = 2^{[u(S)/2]}$

$\min(S)$

$\max(S)$

$\tilde{R}[a] = \{b < U_1(S) \mid aU_1(S) + b \in S \setminus \{\min(S), \ \max(S)\}\}$

$\tilde{S}' = \{a \mid \tilde{R}[a] \neq \emptyset\}$