Алгоритъм на Willard

13.11.2020 г.

(x-fast, y-fast trees)

Дадено: $U \in \mathbb{N}$, $S \subseteq \{0,1,...,U-1\}$
Вход: $x \in \mathbb{N}$
Изход: $\underbrace{predecessor_S(x)}_{pred_S(x)} = \max\{y \in S \,|\, y \leq x\}$

$succ_S(x) = \min\{y \in S \,|\, y \geq x\}$

Цел: Памет за индекс $O(|S|)$ и време за заявка $O(\log \log U)$

Наивен подход:
$\Theta(u)$ - индексиране
$O(1)$ - заявка

Може да заделим масив с u на брой клетки и да заделим в него информацията която ни интересува.
Това е оптимално ако S е от порядъка на u.

По-интересния вариант е когато $|S| << U$, $|S| \in O(|u|)$.
Ще използваме двоично наредено/балансирано дърво.

$O(|S| \log |S|)$ - за индексиране, $O(|S|)$ - памет
$O(\log |S|)$ - за заявка

$Willard$:
$O(|S|)$ - памет
$O(\log \log |u|)$ - време за заявка

Алгоритъма си заслужава да му бъде обърнато сериозно внимание ако $\log u << |S| << u$.

Основна идея на алгоритъма на Willard:

$0 \leq x < U$, $x \in \mathbb{N} \mapsto x_{(2)} = \overline{x_1 x_2 \ldots x_u} \rightarrow x \leq y \overset{0 \leq x,y < U}{\Leftrightarrow} x_1 \ldots x_U \underset{lex}{\preceq} y_1 \ldots y_U$

$S \mapsto S$ - думите от $\{0,1\}^u$ за числата в S за $\tilde{S} - trie$.

Твърдение: Нека $x, y \in \mathbb{N}$ с $x_{(2)} = \overline{x_1 x_2 \ldots x_u}$ и $y_{(2)} = \overline{y_1 y_2 \ldots y_u}$. Тогава $x \leq y \Leftrightarrow x_1 x_2 \ldots x_u \underset{lex}{\preceq} y_1 y_2 \ldots y_u$.

Доказателство: 1. сл. $x_1 x_2 \ldots x_u = y_1 y_2 \ldots y_u \Rightarrow x = y$ и $x_1 \ldots x_u \underset{lex}{=} y_1 \ldots y_u$

2. сл. $x_1 x_2 \ldots x_u \neq y_1 y_2 \ldots y_n$. Без ограничение на общността i е най-малкият индекс, за който $x_i \neq y_i$ и $x_i < y_i \Rightarrow x_i = 0,\ y_i = 1$. Тогава $x = \sum_{j=1}^{u} 2^{u-j} x_j$ и $y = \sum_{j=1}^{u} 2^{u-j} y_j$. От избора на i: $x_j = y_j$ за $j < i$ и следователно $x_1 \ldots x_u \prec y_1 \ldots y_u$. Цел: $x < y$.

Заместваме в двете представяния и получаваме следното:

$$x = \sum_{j=1}^{u} 2^{u-j}x_j = \sum_{j=1}^{i-1} 2^{u-j}x_j + 2^{u-i}\underbrace{x_i}_{=0} + \sum_{j=i+1}^{u} 2^{u-j}\underbrace{x_j}_{\leq 1} \leq \sum_{j=1}^{i-1} x_j 2^{u-j} + 0 + \underbrace{\sum_{j=i+1}^{u} 2^{u-j}}_{\sum_{j=0}^{u-i+1} 2^j = 2^{u-i}-1} = \sum_{j=1}^{i-1} x_j 2^{u-j} + 2^{u-i} - 1$$

$$y = \sum_{j=1}^{u} 2^{u-jy_j} = \sum_{j=1}^{i-1} 2^{u-j}y_j + 2^{u-i}\underbrace{y_i}_{=1} + \sum_{j=i+1}^{u} 2^{u-j}\underbrace{y_j}_{\geq 0} \geq \sum_{j=1}^{i-1} y_j 2^{u-j} + 2^{u-i}$$

Сега, ако сравним двата израза за x и y става ясно, че израза за y е строго по-голям от този за x. Следователно $x < y$.

Фиксираме u да бъде броя цифри за записването на числото U в двоичен запис. Т.е. $2^{u-1} \leq U < 2^u$.

Дефинираме множеството $\tilde{S} = \{a_1a_2 \dots a_u \in \{0,1\}^u \,|\, \exists a \in S\ (a_{(2)} = \overline{a_1 \dots a_u})\}$.

<u>Дефиниция</u>: Нека $\sum$ е азбука, а $\mathscr{D} \in \sum^*$ е крайно множество.

$Pref(\mathscr{D}) = \{v \in \sum^* \,|\, \exists w \in \sum^*\ (vw \in \mathscr{D})\}$

$Trie(\mathscr{D})$ ще наречем крайния детерминиран автомат $\left(\sum, Pref(\mathscr{D}), \varepsilon, \delta_{\mathscr{D}}, \mathscr{D}\right)$, където

$$\delta_{\mathscr{D}}(v, \sigma) = v\sigma = \begin{cases} v\sigma, & if \quad v\sigma \in Pref(\mathscr{D}) \\ \neg!, & otherwise \end{cases}$$

За S построяваме $Trie(\tilde{S})$.

<u>Пример</u>: Нека $U = 63$, $u = 6$ и $S = \{13, 22, 5, 31, 59, 19, 20, 50, 23\}$

$\tilde{a}^{(1)} = 13_{(2)} = \overline{001101}$

$\tilde{a}^{(2)} = 22_{(2)} = \overline{010110}$

$\tilde{a}^{(3)} = 5_{(2)} = \overline{000101}$

$\tilde{a}^{(4)} = 31_{(2)} = \overline{011111}$

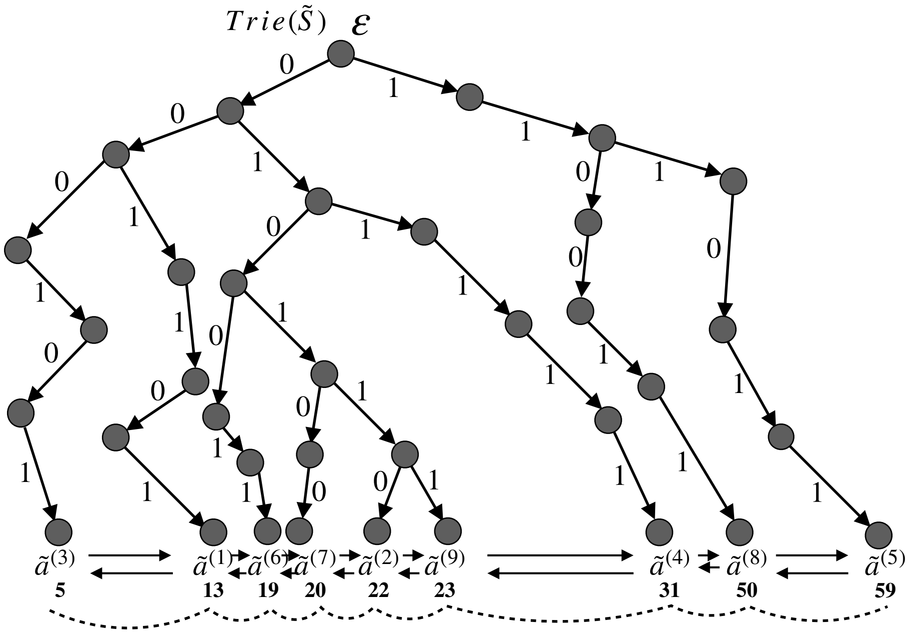
$\tilde{a}^{(5)} = 59_{(2)} = \overline{111011}$

$\tilde{a}^{(6)} = 19_{(2)} = \overline{010011}$

$\tilde{a}^{(7)} = 20_{(2)} = \overline{010100}$

$\tilde{a}^{(8)} = 50_{(2)} = \overline{110010}$

$\tilde{a}^{(9)} = 23_{(2)} = \overline{010111}$



$5 < 13 < 19 < 20 < 22 < 23 < 31 < 50 < 59$

$DFS\big(Trie(\tilde{S})\big)$:

1.) $\mathbf{0}$

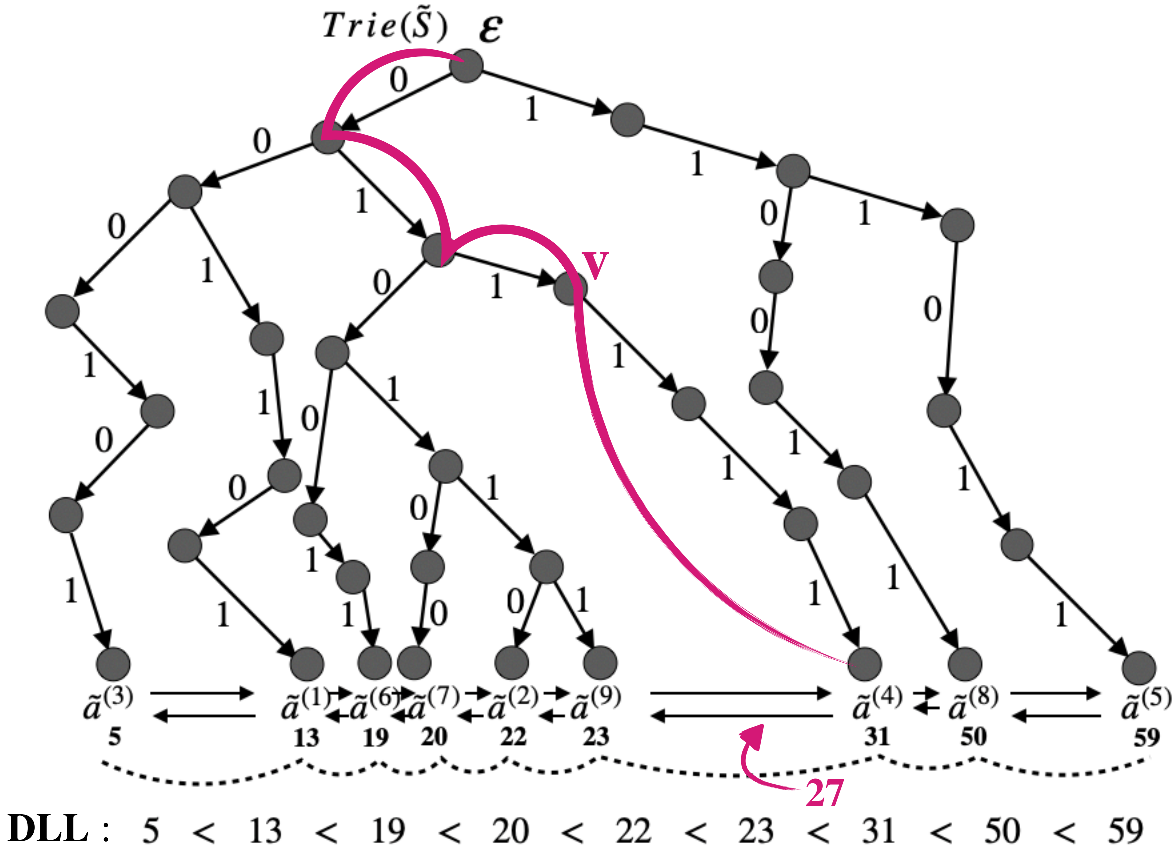
2.) $\mathbf{1}$ $\quad$ lex - наредба на думите ⇔тв. естествена наредба в $\mathbb{N}$.

Имаме S, u, U

1. Намираме $\tilde{S} \in \{0,1\}^u$
2. Строим $T = Trie(\tilde{S})$
3. Обхождаме T в дълбочина, като ребро със стойност 0 е с приоритет пред ребро със стойност 1 и организираме листата на T в двусвързан списък DLL според това обхождане.
4. За всеки връх от дървото, $\forall v \in T$, пресмятаме:
   $lml(v)$ - най-ляво листо в поддървото T_v;
   $rml(v)$ - най-дясно листо в поддървото T_v;
   $(lml = left - most - leaf, rml = right - most - leaf)$
   Ако $v = v_1 v_2 \ldots v_i \in \{0,1\}^*$, съпоставяме $val(v) = \sum_{j=1}^{i} 2^{i-j} v_j$.
5. За всяко ниво $\mathcal{L}_i$ на дървото T поддържаме (построяваме) перфектен хеш $\mathcal{H}_i = \{val(v), v \,|\, v \in Pref(\tilde{S}), |v| = i\}$. Ключовете са стойностите на върховете v, а това което хешираме са самите върхове v.

Това са основните структури от данни, които ще ни трябват. Сега да видим защо те ще ни трябват и как ще ни помогнат.

Нека сега имаме една конкретна заявка $x = 27$. Това число в двоична бройна система би се записало по следния начин $\tilde{a}_x = 27_{(2)} = \overline{011011}$. Искаме да намерим $prev_S(x)$, $succ_S(x)$.



$$lml(\mathbf{v}) = rml(\mathbf{v}) = \tilde{a}^{(4)} = \mathbf{31}$$

$\tilde{a}_x = 27_{(2)} = \overline{011011}$

твърдението е, че $lml(\mathbf{v}) = \tilde{a}^{(4)} = 31$ е наследника на $\tilde{a}_x = 27$.

$\mathbf{31} = \mathbf{succ_S(27)}$, а пък $Prev(\tilde{a}^{(4)}) = Prev(31) = 23 = \tilde{a}^{(9)}$ е предшественика на $\tilde{a}_x$.

$\mathbf{23} = \mathbf{pred_S(27)}$

<u>Твърдение</u>: Нека $x \in \mathbb{N}$ със запис $x_{(2)} = \overline{x_1 x_2 \dots x_u}$ и нека $\mathbf{v} = x_1 x_2 \dots x_i$ е най-дългия префикс на x, за който $v \in T$. Тогава:

1) ако $i = u$, то $x \in S$; $prev_S(x) = succ_S(x) = x$

2) $i < u$, то $\delta(v, x_{i+1})$ не е дефинирана и нещо повече:

2.1) ако $x_{i+1} = 0$, то $lml(v) = succ_S(x)$

2.2) ако $x_{i+1} = 1$, то $rml(v) = prev_S(x)$

<u>Доказателство</u>:

Първото твърдение е ясно. Разглеждаме втората ситуация. 2)

Твърдението, че $\delta(v, x_{i+1})$ не е дефинирано следва тривиално от максималността на i. Без ограничение на общността разглеждаме случая, в който $x_{i+1} = 0$. Тогава:

Нека $\tilde{w} \in \tilde{S}$ $(w \in S)$ е произволно. Тогава има два случая:

<u>I сл.</u>: $\tilde{w}_1 \tilde{w}_2 \dots \tilde{w}_i = v \Rightarrow \tilde{w} \in T_v$. От дефиницията на най-ляво листо $lml(v)$ следва, че лексикографски $\tilde{w} \underset{lex}{\geq} lml(v)$. Тъй като v не е листо и $\delta(v, x_{i+1}) = \delta(v, 0)$ не е дефинирано, то $lml(v)$ на позиция $i+1$ има $\mathbf{1}$-ца и следователно

$lml(v) \underset{lex}{\geq} x_1 x_2 \dots x_u \Rightarrow w \geq val\big(lml(v)\big) > x$.

Всички листа, които са в поддървото на v ще бъдат по-големи или равни от заявката в този случай.
II сл.: $\tilde{w}_1 \dots \tilde{w}_i \neq v$ и нека j е първата позиция, на която $\tilde{w}$ и v се различават. Тогава е ясно, че $\tilde{w}_1 \dots \tilde{w}_{j-1} = v_1 v_2 \dots v_{j-1} = x_1 \dots x_{j-1}$ и освен това $w_j \neq v_j = x_j$, но това означава, че $\tilde{w} \underset{lex}{\prec} v \Leftrightarrow \tilde{w} \underset{lex}{\prec} x$ и по същия начин $\tilde{w} \underset{lex}{\prec} v \Leftrightarrow \tilde{w} \underset{lex}{\prec} lml(v) \Rightarrow$

$w < x \Leftrightarrow w < val(lml(v)) \Rightarrow$ от I и II може да заключим, че стойността на това листо е точно наследника на елемента x от множеството S: $val\big(lml(v)\big) = succ_S(x)$.

Случаят когато $x_{i+1} = 1$ е дуален. Тогава просто ще получим предшественика на x, вместо наследника, намирайки най-дясното листо на лявото поддърво с ребро/връх 0.

С тези забележки вече сме готови да покажем алгоритъма на Willard в неговата първа част.

Идеята сега ще бъде да спестим това наивно траверсиране със заявки към перфектните хещове, които дефинирахме за нивата и съответно стойностите на върховете v в тях. Това би ни позволило да намерим най-дългия префикс на x, който се среща в нашето поддърво за време пропорционално на $\log\big(Height(T)\big) = \log \log U$.

Заявки:
Идеята е да правим двоично търсене използвайки хешовете.
Идея: Търсим най-дълъг префикс $x_1 \dots x_i \in T$ с двоично търсене по нивата $\mathscr{L}_j$ на T, използвайки хешовете $\mathscr{H}_j$.
Какво ни пречи правенето на такова търсене? Единственото което ни трябва е по даден номер i на префикс да може ефективно да сметнем коя е стойността на този префикс. Т.е. Дадено ни е i, x $\nrightarrow$ (трябва да стигнем до) $val(x_1 \dots x_i) = \sum_{j=1}^{i} 2^{i-j} x_j$, като това което имаме е $x = \sum_{j=1}^{u} 2^{u-i} x_j$, но

$x = \sum_{j=1}^{u} 2^{u-i} x_j = 2^{u-i} . val(x_1 \dots x_i) + \sum_{j=i+1}^{u} 2^{u-j} x_j < 2^{u-i}\big(val(x_1 \dots x_i) + 1\big)$. Т.е. за да намерим стойността на съответния префикс е достатъчно да разделим:

$val(x_1 \dots x_i) = \left\lfloor \dfrac{x}{2^{u-1}} \right\rfloor$.

Вече сме готови да реализираме двоичното търсене.

Пресмятаме стойностите 2^{u-i} предварително т.е.$\rightarrow$ степените на двойката.

$\textbf{procedure } query(T, \mathscr{H}, DLL, x, u)$ {

$\quad$ **if** $x \in \mathscr{H}_u$ **then**

$\qquad$ **return** $(\underset{pred_S(x)}{x}, \underset{succ_S(x)}{x})$

$\quad$ **if** $x > U$ **then**

$\qquad$ **return** $(val(rml(\varepsilon)), \infty)$

$\quad$ // $x < U$ и $x_{(2)} = \overline{x_1 \dots x_u}$

$\quad$ $l \leftarrow 0, r \leftarrow u$ // $x_1 \dots x_r \notin \mathscr{L}_r, x_1 \dots x_l \in \mathscr{L}_l \leftarrow invariant$

$\quad$ **while** $(l - r > 1)$ **do**

$\qquad$ $m \leftarrow \left\lceil \frac{l+r}{2} \right\rceil$

$\qquad$ $y \leftarrow \left\lfloor \frac{x}{2^{u-m}} \right\rfloor$

$\qquad$ **if** $y \in \mathscr{H}_m$ **then**

$\qquad\quad$ $l \leftarrow m$

$\qquad$ **else**

$\qquad\quad$ $r \leftarrow m$

$\quad$ **done**

$\quad$ $v : \left(\left\lfloor \frac{x}{2^{n-l}} \right\rfloor, v \right) \in \mathscr{H}_l$

$\quad$ **if** $\delta(v,1)$ $is\ defined$ **then** // т.е. $x_{l+1} = 0$ и $\delta(v,0)$ не е дефинирано

$\qquad$ $s \leftarrow lml(v)$

$\qquad$ $p \leftarrow Pred(DLL, s)$

$\quad$ **else**

$\qquad$ $p \leftarrow rml(v)$

$\qquad$ $s \leftarrow Succ(DLL, p)$

$\quad$ **return** (p, s)

}

Целия този цикъл в алгоритъма завършва за време $O(\log u) = O(\log \log U)$.

Остана само проблема с паметта. Броя на върховете в дървото което имаме не може да го ограничим с броя на листата (не може да го ограничим с константа по броя на листата). Проблема е, че често може да имаме дълги верижки без разклонения, т.е. да имаме малко разклонения.

Памет $O(|S| \log U)$. Трябва да смачкаме фактора от $\log$ без да разваляме времето за заявка.

Отново ще използваме идеята на блоковете, но този път блоковете ще са от листа.

<u>Намаляване на паметта</u>.

1. Сортираме S във възходящ ред $\big($тъй като ако построим директно $Trie$-а, ще изгубим веднага памет $n \log(u)\big)$: $a^{(1)} < a^{(2)} < \dots < a^{(n)}$ - за това ни трябва време $O\big((nlog(n)\big)$ (може и за време $O\big(n \log(u)\big)$, но това не е целта) и памет $O(n)$.
2. Избираме блоковете с големина $\log(u)$.
   $n = n'u + r, r < u$

Пресмятаме множеството $S' = \{a^{(ku+1)} \,|\, 0 \leq k \leq n'\}$, което ще бъде от минималните елементи на всеки един блок с дължина u.

3. За всяко k поддържаме масив $B[k] = \{a^{(ku+1)}, \ldots, a^{(k+1)u}\}$
4. Намираме индекса за S' $\left(\tilde{S}', T' = Trie(\tilde{S}'), \mathscr{H}'_i, DLL_i\right)$

   $\big($Паметта, която е необходима е $|S'|u \leq n'.u + u \in O(n)\big)$

Заявка за $x \in \mathbb{N}$.

1. Намираме $(k, k+1) : pred_{S'}(x) = a^{(ku+1)} \leq x < a^{\left((k+1)u+1\right)}$
2. С двоично търсене в масива $B[k]$ намираме най-големия индекс $i \leq u$, за който $a^{(ku+i)} \leq x$. $\qquad O\big(\log(u)\big) = O\big(\log\log(u)\big)$
3. Ако $i < u : a^{(ku+i)} \leq x < a^{(ku+i+1)}$, т.е. отговора се намира изцяло в масива $B[k]$ и имаме двата последователни елемента, между които е заключен x и те са съответно неговия предшественик и наследник. Ако имаме равенство, то предшественика и наследника съвпадат. Ако $i = u : a^{\left((k+1)u\right)} \leq x < a^{\left((k+1)u+1\right)}$ - това е случаят, в който предшественика и наследника на се намират на границата на един от масивите $B[k]$.