

Алгоритъм на Willard

13.11.2020 г.
(*x-fast, y-fast trees*)

Дадено: $U \in \mathbb{N}, S \subseteq \{0, 1, \dots, U-1\}$

Вход: $x \in \mathbb{N}$

Изход: $\text{predecessor}_S(x) = \max\{y \in S \mid y \leq x\}$

$$\overbrace{\text{predecessor}_S(x)}^{\text{pred}_S(x)} \\ \text{succ}_S(x) = \min\{y \in S \mid y \geq x\}$$

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

Наивен подход:

$\Theta(u)$ - индексирание

$O(1)$ - 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 $|S| \ll U, |S| \in O(|u|)$.

Ще използваме двоично наредено/балансирано дърво.

$O(|S| \log |S|)$ - за индексирание, $O(|S|)$ - 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

$\log u \ll |S| \ll u$.

Основна идея на алгоритъма на Willard:

$$0 \leq x < U, x \in \mathbb{N} \mapsto x_{(2)} = \overline{x_1 x_2 \dots x_u} \rightarrow x \leq y \stackrel{0 \leq x, y < U}{\Leftrightarrow} x_1 \dots x_u \underset{lex}{\leq} y_1 \dots y_u$$

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

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

$$x \leq y \Leftrightarrow x_1 x_2 \dots x_u \underset{lex}{\leq} y_1 y_2 \dots y_u.$$

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

2. сл. $x_1 x_2 \dots x_u \neq y_1 y_2 \dots y_u$. 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 $x_1 \dots x_u < y_1 \dots y_u$. Цел: $x < y$.

Заместваме в двете представяния и получаваме следното:

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

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

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 $\tilde{S} = \{a_1 a_2 \dots a_u \in \{0,1\}^u \mid \exists a \in S (a_{(2)} = \overline{a_1 \dots a_u})\}$.

Дефиниция: Нека Σ е азбука, а $\mathcal{D} \in \Sigma^*$ е крайно множество.

$$Pref(\mathcal{D}) = \{v \in \Sigma^* \mid \exists w \in \Sigma^* (vw \in \mathcal{D})\}$$

$Trie(\mathcal{D})$ ще наречем крайния детерминиран автомат $(\Sigma, Pref(\mathcal{D}), \varepsilon, \delta_{\mathcal{D}}, \mathcal{D})$, където

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

За S построяваме $Trie(\tilde{S})$.

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

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

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

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

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

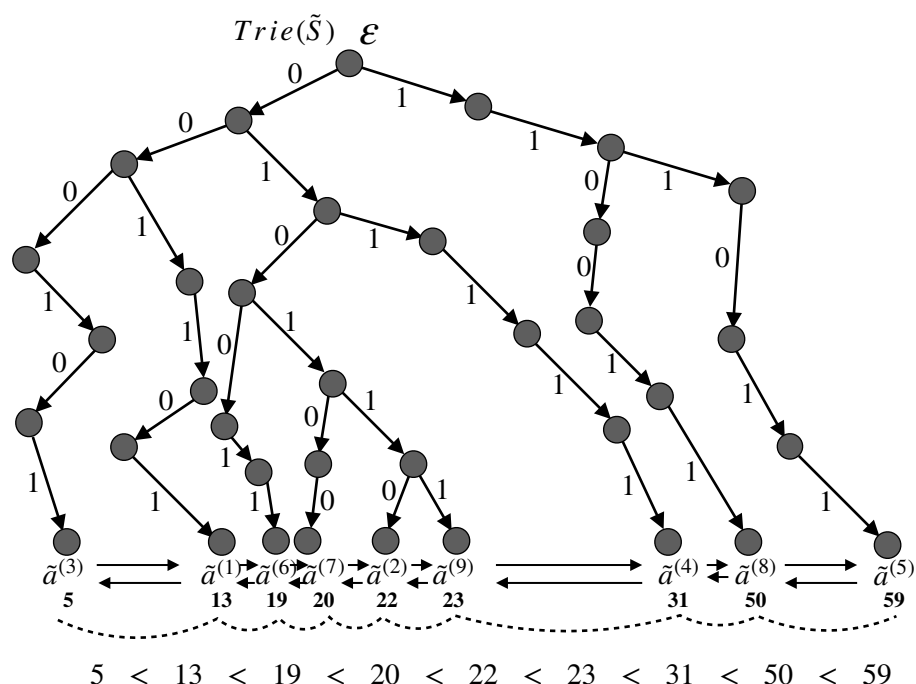
$$\tilde{a}^{(5)} = 59_{(2)} = \overline{111011}$$

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

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

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

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



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

1.) **0**

2.) **1** $\xrightarrow{\text{lex}}$ - наредба на думите \Leftrightarrow 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 листо 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 $v = v_1 v_2 \dots v_i \in \{0,1\}^*$, съпоставяме $val(v) = \sum_{j=1}^i 2^{i-j} v_j$.

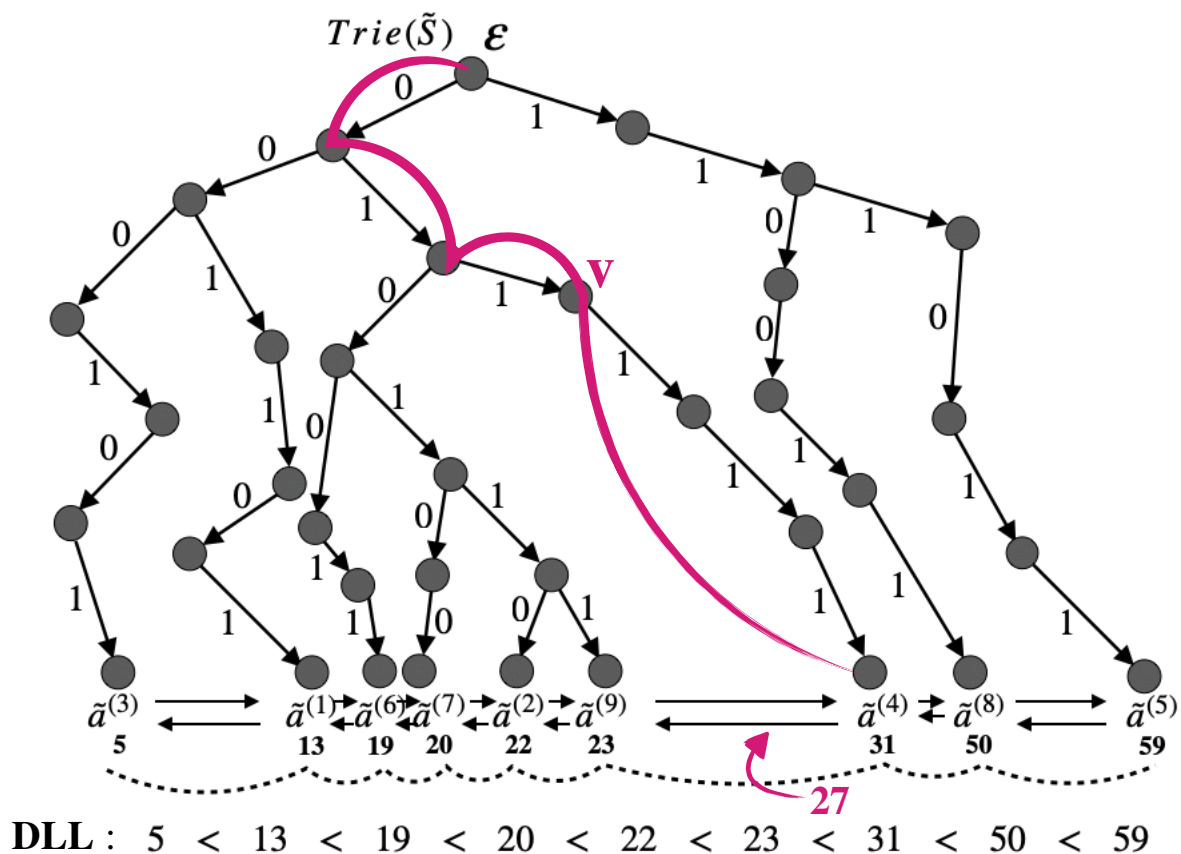
5. За всяко ниво \mathcal{L}_i на дървото T поддържа (построяваме) перфектен хеш

$\mathcal{H}_i = \{val(v), v \mid v \in Pref(\tilde{S}), |v| = i\}$. 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

$prev_S(x), succ_S(x)$.



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

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

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

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

23 = $\text{pred}_S(27)$

Твърдение: 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 $x \in S$; $\text{prev}_S(x) = \text{succ}_S(x) = x$
- 2) $i < u$, то $\delta(v, x_{i+1})$ не е дефинирана и нещо повече:
 - 2.1) ако $x_{i+1} = 0$, то $lml(v) = \text{succ}_S(x)$
 - 2.2) ако $x_{i+1} = 1$, то $rml(v) = \text{prev}_S(x)$

Доказателство:

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:

1 сл.: $\tilde{w}_1 \tilde{w}_2 \dots \tilde{w}_i = v \Rightarrow \tilde{w} \in T_v$. От дефиницията на най-ляво листо $lml(v)$ следва, че лексикографски $\tilde{w} \geq_{lex} lml(v)$. 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 **1**-ца и следователно

$$lml(v) \geq_{lex} x_1 x_2 \dots x_u \Rightarrow w \geq_{lex} \text{val}(lml(v)) > x.$$

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 $\tilde{w} <_{lex} v \Leftrightarrow \tilde{w} <_{lex} x$ и по същия начин $\tilde{w} <_{lex} v \Leftrightarrow \tilde{w} <_{lex} lml(v) \Rightarrow$

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

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/върх 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 хешове, които дефинирахме за нивата и съответно стойностите на върховете v в тях. Това би ни позволило да намерим най-дулгия префикс на x , който се среща в нашето поддърво за време пропорционално на $\log(Height(T)) = \log \log U$.

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 \mathcal{L}_j на T , използвайки хешовете \mathcal{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 , $x \mapsto$ (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

$x = \sum_{j=1}^u 2^{u-i} x_j = 2^{u-i} \cdot val(x_1 \dots x_i) + \sum_{j=i+1}^u 2^{u-j} x_j < 2^{u-i} (val(x_1 \dots x_i) + 1)$. Т.е. за да

намерим стойността на съответния префикс е достатъчно да разделим:

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

Вече сме готови да реализираме двоичното търсене.

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

```

procedure query( $T, \mathcal{H}, DLL, x, u$ ) {
  if  $x \in \mathcal{H}_u$  then
    return (  $\begin{smallmatrix} x \\ \text{pred}_S(x) \end{smallmatrix}, \begin{smallmatrix} x \\ \text{succ}_S(x) \end{smallmatrix}$  )
  if  $x > U$  then
    return ( $\text{val}(\text{rml}(\varepsilon)), \infty$ )
  //  $x < U$  и  $x_{(2)} = \overline{x_1 \dots x_u}$ 
   $l \leftarrow 0, r \leftarrow u$  //  $x_1 \dots x_r \notin \mathcal{L}_r, x_1 \dots x_l \in \mathcal{L}_l \leftarrow \text{invariant}$ 
  while ( $l - r > 1$ ) do
     $m \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$ 
     $y \leftarrow \left\lfloor \frac{x}{2^{u-m}} \right\rfloor$ 
    if  $y \in \mathcal{H}_m$  then
       $l \leftarrow m$ 
    else
       $r \leftarrow m$ 
  done
   $v : \left( \left\lfloor \frac{x}{2^{n-l}} \right\rfloor, v \right) \in \mathcal{H}_l$ 
  if  $\delta(v, 1)$  is defined then // т.е.  $x_{l+1} = 0$  и  $\delta(v, 0)$  не е дефинирано
     $s \leftarrow \text{lml}(v)$ 
     $p \leftarrow \text{Pred}(DLL, s)$ 
  else
     $p \leftarrow \text{rml}(v)$ 
     $s \leftarrow \text{Succ}(DLL, p)$ 
  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 промяна 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.

Намаляване на паметта.

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

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

3. За всяко k поддържаме масив $B[k] = \{a^{(ku+1)}, \dots, a^{(k+1)u}\}$

4. Намираме индекса за S' (\tilde{S}' , $T' = \text{Trie}(\tilde{S}')$, \mathcal{H}'_i , DLL_i)
(Паметта, която е необходима е $|S'|u \leq n' \cdot u + u \in O(n)$)

Заявка за $x \in \mathbb{N}$.

1. Намираме $(k, k+1) : \text{pred}_{S'}(x) = a^{(ku+1)} \leq x < a^{((k+1)u+1)}$

2. С двоично търсене в масива $B[k]$ намираме най-големия индекс $i \leq u$, за който $a^{(ku+i)} \leq x$. $O(\log(u)) = O(\log \log(u))$

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 $i = u : a^{((k+1)u)} \leq x < a^{((k+1)u+1)}$ - това е случаят, в който предшественика и наследника на се намират на границата на един от масивите $B[k]$.