Предшественик от определено ниво (част 2)

09.10.2020 г.

(Level Ancestor Query)

Рекурсивната схема за отговор на заявка LA, която разглеждахме от миналия път изглеждаше по следния начин:

$LA_Q(v_0, d_0)$

1. $\Pi_0 \longleftarrow$ максималния път, в който се намира върха v_0 $\left(v_0 \in \Pi_0\right)$
2. Ако не намерим отговора за заявката, в максималния път Π_0, т.е. ако $d_1 = depth(v_0) - depth(u_0) < d_0$, където $u_0 = \Pi_0 . last$ е последния елемент от максималния път Π_0, то дефинираме $v_1 \longleftarrow p(u_0)$, където $p(u_0)$ е бащата на връх u_0 и търсим заявката $LA_Q(v_1, d_0 - d_1 - 1)$.

Общ шаблон на рекурсията:

$LA_Q(v_t, d_t)$

1. $\Pi_t \longleftarrow$ пътя на v_t
2. $u_t \longleftarrow$ края на Π_t
3. $v_{t+1} \longleftarrow p(u_t)$
4. $d_{t+1} = depth(v_t) - depth(u_t)$
5. $LA_Q(v_{t+1}, d_t - d_{t+1} - 1)$

Очевидно след като се покачваме чрез преминаване от бащата в друг максимален път, то $height(v_{t+1}) = height(u_t) + 1 \Rightarrow height(u_{t+1}) \geq height(u_t) + 1 \Rightarrow |\Pi_{t+1}| \geq |\Pi_t| + 1 \ (*)$.
Тъй като максималните пътища на едно дърво са негово разбиване, то $\Pi_0, \Pi_1, \ldots, \Pi_t, \ldots$ са два по два непресичащи се.

$|V| \geq \sum_{t=0}^{\infty} |\Pi_t|$. От друга страна $\Pi_0 \geq 1$ и следователно ако Π_{t+1} е дефинирано, то

$\Pi_{t+1} \overset{(*)}{\geq} |\Pi_0| + (t+1) \geq t + 2$. Окончателно, ако T е броя на посетените пътища при

заявката, то $|V| \geq \sum_{t=0}^{T} |\Pi_t| \geq \sum_{t=0}^{T} (t+1) = \dfrac{(T+1)(T+2)}{2} \Rightarrow 2|V| \geq T^2 + 3T + 2$ или $T < \sqrt{2|V|}$.

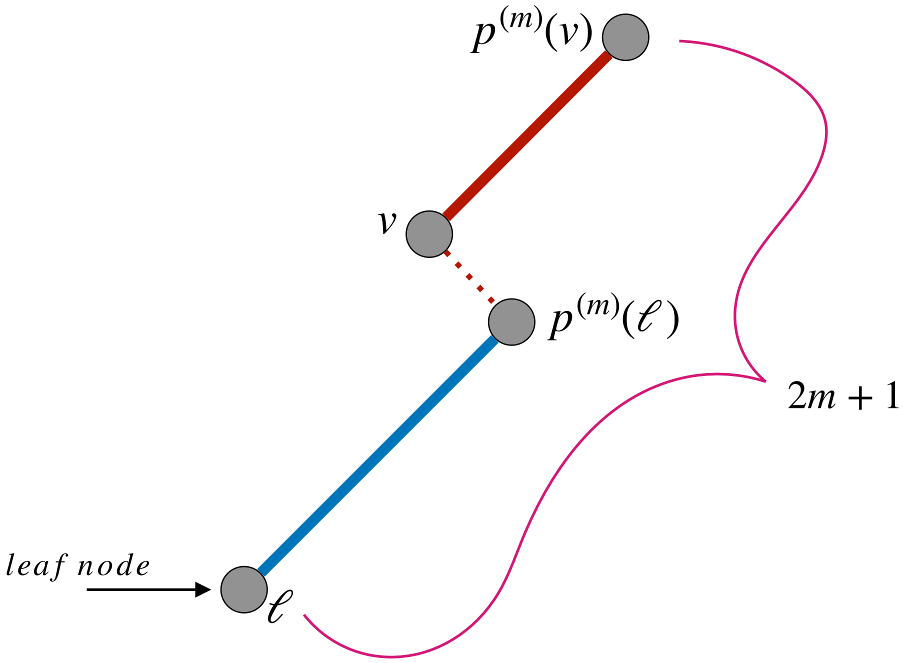
III. Решение на LA-проблема със сложност $< O(n,\ \log(n)) >$ и съответно $< O(nlog(n)), O(1) >$

III. 1. Стълби (Ladders Algorithm)

<u>Идея</u>: Тъй като в горното разсъждение имахме зависимостта $|\Pi_{t+1}| \geq |\Pi_t| + 1$ и получихме $O(\sqrt{n})$ сложност за заявка, то ако успеем по някакъв начин да гарантираме че $|\Pi_{t+1}| \geq \underset{const>1}{2} |\Pi_t|$, ще може да заключим, че

$|V| \geq \sum_{t=0}^{T} |\Pi_t| \geq \sum_{t=0}^{T} 2^t = 1 + 2 + \ldots + 2^T = 2^{T+1}$ и следователно

$T \leq \log_2|V| - 1 < \log_2|V|$ и по този начин ще постигнем логаритмична сложност.
Как може синтетично да докараме исканата зависимост?



Синия максимален път от листо ℓ, плюс червеното му продължение с дължина равна на пътя и един преход от син към баща, ще наричаме *стълба* (синьо+червено=стълба). Всеки връх си знае максималния път. Всеки максимален път си има стълба.

Индексираме стълбите (масиви)

(v_t, d_t)

1. $\Pi_t \longleftarrow$ максималния път на v_t
2. $\lambda_t \longleftarrow$ стълбата на Π_t
3. $u_t \longleftarrow$ края на стълбата на Π_t, т.е. последния елемент на λ_t
4. $v_{t+1} \longleftarrow p(u_t)$

(v_{t+1}, d_{t+1})

$|\lambda_t| = 2|\Pi_t| + 1$

$height(u_t) \geq |\lambda_t| + 1 = 2(\Pi_t + 1)$

$|\Pi_{t+1}| \geq 2(|\Pi_t| + 1)$, по индукция може да докажем, че $|\Pi_t| \geq 2^t$.

Дефиниция. Нека $T(V, p, r)$ е кореново дърво. Нека $\Pi = (v_0, v_1, \ldots, v_m)$ е максимален път в T. Стълба $\lambda(\Pi)$ породена от пътя Π в дървото T наричаме пътя:

$$\lambda(\Pi) = \begin{cases} \Pi_0\big(p(v_m), p^{(2)}(v_m), \ldots, p^{(m+1)}(v_m)\big), & \text{ако } d(v_m) \geq m+1 \\ \Pi_0\big(p(v_m), p^{(2)}(v_m), \ldots, \underbrace{p^{(d(v_m))}(v_m)}_{=r}\big), & \text{ако } d(v_m) < m+1 \end{cases}$$

Описание на индекса: $\mathscr{A}_{\mathscr{I}}$

1. Намираме разбиване на T на максимални пътища $\{\Pi_i\}_{i=1}^{I}$
2. За всеки път Π_i намираме λ_i, което е стълбата на пътя Π_i : $\lambda_i = \lambda(\Pi_i)$
   $\big(|\lambda_i| \leq 2|\Pi_i| + 1$ и времето за тази стъпка е $O\big(\sum(\Pi_i + 1)\big) = O(|V| + \underbrace{|I|}_{\leq |V|}) = O(|V|)$
3. Организираме λ_i като масив $\lambda_i = (v_0, \ldots, v_k)$, $len[\lambda_i] = k + 1$

За всеки връх v:

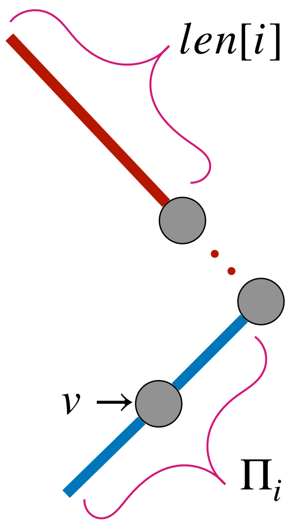
4. Индекса на v: $ind(v) = i \Leftrightarrow v \in \Pi_i$
5. $height(v) = height(T_v)$
6. $depth(v) \longleftarrow$ дълбочината на v в T

За домашно:
С индекса $\mathscr{A}_{\mathscr{I}} : 1 \to 6$ да се опише алгоритъма за заявка $\mathscr{A}_Q$ с времева сложност $O\big(\log|V|\big)$.

Алгоритъм за заявка $\mathscr{A}_Q$ чрез стълби:

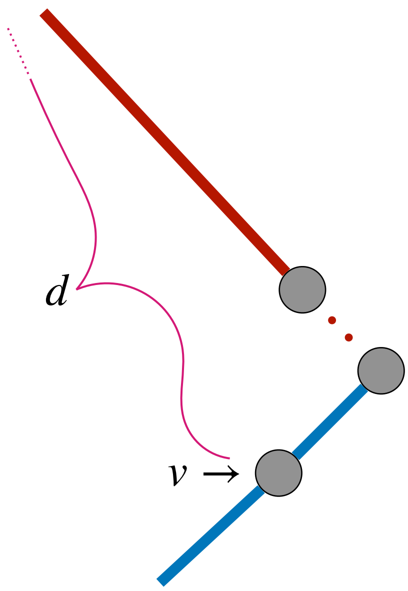
процедура $LA_Q(v, d) \longleftarrow$ връща d-тия предшественик на връх v

1. Ако дълбочината на върха v е по-малка от d, то такъв предшественик няма да съществува и връщаме $nullptr$ или какъвто е стандарта за $\perp$;
2. $u = Ladder[indPath[v]].last()$ - съхраняваме последния връх от стълбата на пътя, в който се намира върха v;
3. Калкулираме разстоянието от търсения връх до края на стълбата $d' = depth(v) - depth(u)$;
4. Ако $d' \geq d$, то търсения предшественик е в текущата стълба и го намираме като пресметнем $depFirst = depth[L[indPath[v]][0]]$ - дълбочината на първия връх в стълбата и $depStart = depFirst - depth(v)$ - позицията, на която се намира елемента v в стълбата. Скед което връщаме върха на разстояние d от върха v в стълбата, към посока корена $\longrightarrow Ladder[indPath[v]][depStart + d]$;
5. Ако $d' < d$, то търсения предшественик го няма в текущата стълба и трябва да преминем към следващата стълба, като отчетем вече изкаченото разстояние в текущата стълба: извикваме рекурсивно заявката за родителя на последния връх от текущата стълба с ново разстояние, което се получава като от старото извадим (разстоянието от върха до края на текущата стълба $+1 = d' + 1$), защото сме се изкачили още веднъж нагоре с операцията син→баща: $LA_Q(parent[u], d - d' - 1)$.



$|\Pi_i| = len[i]$. При заявка (v, d), $d \leq len[i]$, то $|\lambda_i| - |\Pi_i| \geq len[i] + 1$ (освен, ако λ_i не съдържа корена на дървото, в който случай, ще може да отговорим на заявката веднага) $\Rightarrow$ отговора на (v, d) ще е в стълбата λ_i.

Въпросът е какво ще правим в общия случай: когато имаме заявка (v, d) и $d >> len[i]$

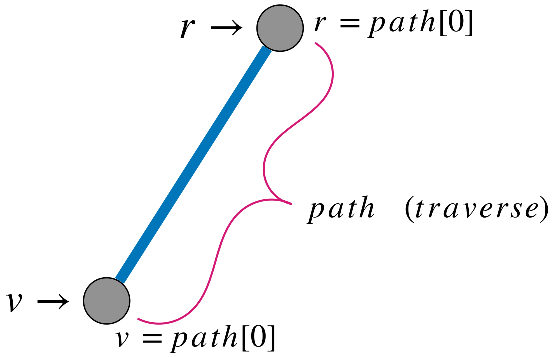


Числото d може да представим като $d = 2^k + d_1$, където $0 \leq d_1 < 2^k$, т.е. $2^k \leq d < 2^{k+1}$. Тогава $p^{2^k}(v)$ има височина поне $2^k \Rightarrow$ неговоят максимален път $\big($максималния път на $p^{2^k}(v)\big)$ е Π_j и е такъв, че $|\Pi_j| \geq 2^k$ и тъй като $d_1 < 2^k$ (защото в противен случай щяхме да имаме, че $d = 2^{k+1} + d_2$), то получаваме, че заявката $(p^{(2^k)}, d_1)$ ще връща връх, който е в рамките на λ_j - стълбата на j-тия максимален път. Тази идея ни навежда на мисълта за намирането на 2^s-тия прародител на даден връх, за константно време. Възможно ли е това и как бихме могли да препроцесираме (направим предварителната подготовка)?

III. 2. Големи подскоци нагоре към корена на дървото (Jump Pointer Algorithm)

За всеки връх v намираме $jump\left[v\right]\left[0\,...\,\lfloor \log[depth(v)]\rfloor\right]$, като $jump[v][k] = p^{(2^k)}(v)$.

<u>Идея</u>: Обхождаме T в дълбочина и в масив $path$ с дължина l пазим пътя до текущия връх v. Тогава може да попълним таблицата по следния начин:



procedure $fillJumps(path, l, v)$
    $k \leftarrow 0$
    $pow \leftarrow 1$
    while $pow \leq l$ **do**
        $jump[v][k] \leftarrow path[l - pow]$
        $k \leftarrow k + 1$
        $pow = 2 \times pow$
    done
done

$O(\log[depth(v)]) = O(|V|)$

procedure $dfs(path, l, v)$
    $path[l] \leftarrow v$
    $fillJumps(path, l, v)$
    for each $u: \ p(u) = v$ **do**
        $dfs(T, path, l + 1, u)$
    done
done

<u>За домашно</u>: Чрез помощта на описания по-горе индекс $\mathscr{A}_{\mathscr{I}}$, да се опише алгоритъм за заявка $\mathscr{A}_Q$ с времева сложност $O\left(\log|V|\right)$.

процедура $LA_{jump}(v, d)$

1. Първо елиминираме случая в който $d = 0$, тъй като съхраненията в $jump$ таблицата започват от първия прародител (бащата), защото $2^0 = 1$.
   Ако $d = 0$ връщаме v;
2. $k \leftarrow flog[d]$, където в $flog[0\,...\,n-1]$ сме преизчислили логаритмите закръглени надолу на всички числа от 2 до $n - 1$ (всевъзможните дълбочини);
3. $v_1 \leftarrow jump[v][k]$, новия връх, до който сме се изкачили с 2^k позиции
   $d_1 \leftarrow d - 2^k$, вече сме изминали 2^k позиции и актуализираме нивото на търсения предшественик. Извикваме $LA_{jump}(v_1, d_1)$.
   Забележка: 2^k го пресмятаме с побитово отместване на 1-ци.

В алгоритъма за заявката LA_{jump} използваме дискретно факта, че всяко естествено може да се представи във вида $d = 2^k + d_1$, където $0 \leq d_1 < 2^k$, т.е. $2^k \leq d < 2^{k+1}$.

III. 3. Съчетание на голямо скачане към върха на дървото със стълба (Jump pointer + ladder)

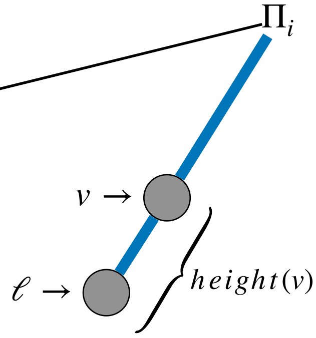
Да се върнем отново на алгоритъма за намиране на индекса $\mathcal{A}_{\mathcal{J}}$. Необходимо ли е да намираме скоковете за всички върхове в дървото? Може да намерим скоците само за листата. По този начин като получим заявка за връх v ще намерим в кой максимален път се съдържа, ще вземем първия елемент на този път, който ще е листо и ще скочим от него. Но, всеки максимален път си има стълба, която е с дължина колкото $2*k+1$, където k е дължината на максималния път с която е асоциирана стълбата (забележете, че асоциираме стълба към път, а не към връх, тъй като всеки връх има точно един максимален път, но не и една стълба, той може да участва в няколко стълби). Първия скок ще е с дължина 2^m, тоест върха в който ще скочим, ще участва в максимален път, който е с дължина поне 2^m (2. от дефиницията за максимален път). Ако допуснем, че стълбата на този максимален път не стига до корена, то първия скок ще е бил на 2^{m+1}, тъй като стълбата има дължина два пъти дължината на максималния път $+1$ по конструкция (или по-малка, но само ако е стигнала корена). Това е противоречие с допускането, че първия скок е с дължина 2^m. Следователно в стълбата ще се съдържа корена и може веднага да върнем отговор на заявката.

За всяко листо ℓ_i на T намираме скоковете $jump\left[\ell_i\right]\left[0 \dots flog[depth(\ell_i)]\right]$: $jump[\ell_i][k] = p^{(2^k)}(\ell_i)$.

Времето което ще ни е необходимо, за да го осъществиме това е от порядъка на $\underbrace{O(|V|)}_{\text{време извън } fillJumps} + \underbrace{O\left(|Leaves| \times \log |V|\right)}_{\text{време за } fillJumps}$. Общо за $\mathcal{A}_{\mathcal{J}}$: $O\left(|V| + |Leaves| \times \log|V|\right)$.

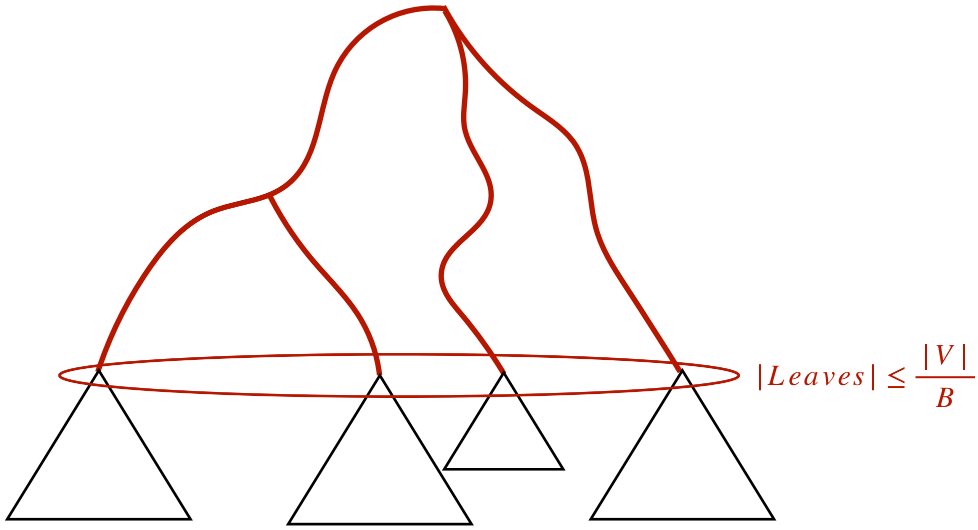
procedure $LA_{leaf}(\mathcal{A}_{\mathcal{J}}, \ell, d)$
  if $d > depth(\ell)$ **then return** $\perp$
  $k \leftarrow flog[d]$ // $2^k \leq d < 2^{k+1}$
  $pow \leftarrow pow[k]$ // $= 2^k$ (тук може да използваме и побитово изместване)
  $u \leftarrow jump[\ell][k]$ // $u = p^{(2^k)}(\ell)$
  $i \leftarrow ind(u)$ // $u \in \Pi_i$, $\lambda(\Pi_i) = \lambda_i$
  return $\lambda_i[height(u) + d - pow]$
done

procedure $LA_{const}(\mathcal{A}_{\mathcal{J}}, v, d)$
  if $d = 0$ **then return** v
  $i \leftarrow ind(v)$ // $v \in \Pi_i$
  $\ell \leftarrow \lambda_i[0]$
  return $LA_{leaf}(\mathcal{A}_{\mathcal{J}}, \ell, height(v) + d)$
done



IV. **Решение на LA-проблема със сложност** $< O(n, log(1)) >$ **.**

1. $T = (V, p, r)$ ще го представим като дърво $T_\mu = (V_\mu, p_\mu, r)$ с $\leq O\left(\frac{|V|}{\log |V|}\right)$ листа и много „малки“ дървета с $O(\log |V|)$ върха.



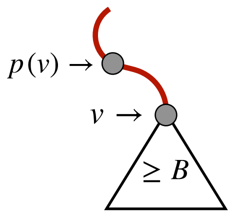
2. За „малките“ дървета ще направим следното: ще ги групираме според тяхната топология (те ще изглеждат по един и същ начин при обхождане в дълбочина). Ще изберем един (каноничен) представител от всяка група, който ще индексираме с решениет $O(n^2,1)$.
   Тогава общата сложност за индексиране на малките дървета ще бъде
   $\underbrace{O(|V|)}_{\text{биекции към „канонични“}} + (\#\text{канонични}) \times (\text{размер на канонично дърво})^2$

<u>Дефиниция</u>: За кореново дърво $T = (V, p, r)$ и константа $B \in \mathbb{N}$ казваме, че:

1. Връх $v \in V$ е микро-връх, ако поддървото T_v има по-по малко върхове от B: $|T_v| < B$
2. $v \in V$ е макро-връх, ако $|T_v| \geq B$
3. $v \in V$ е макро-листо, ако v е макро-връх и $T_v \backslash \{v\}$ са микро-върхове.
4. $v \in V$ е макро-корен, ако v е микро-връх, но $p(v)$ е макро-връх.

<u>Свойства</u>:

1. Ако $v \in V$ е макро-връх, то $p(v)$ също е макро-връх.

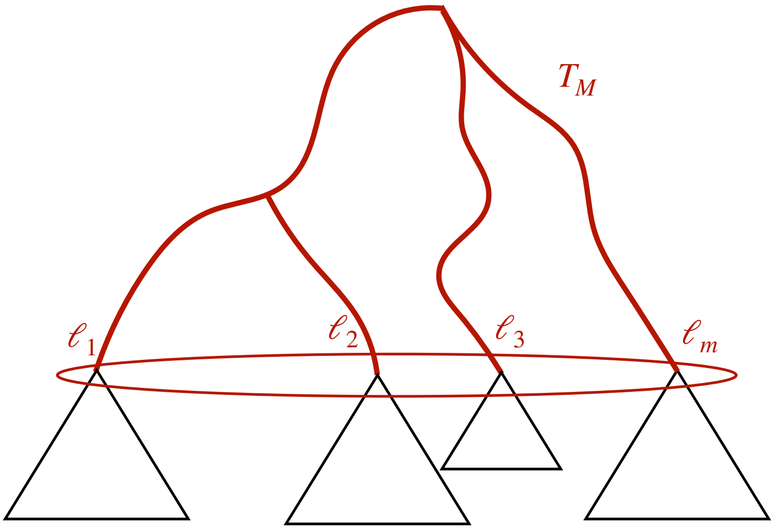


2. Ако $v \in V$ е микро-връх и u е син на v, то u е микро-връх.

3. Нека $V_M = \{v \in V \mid v$ е макро-връх$\}$ и да допуснем, че $|V| \geq B$, т.е. $r \in V_M$. Тогава $p_M = p \restriction V_M$, т.е. $p_M(v) = p(v)$ за $v \in V_M$ е функция от V_M във V_M и $T_M = (V_M, p_M, r)$ е дърво. T_M наричаме макро-дърво за T (то е едно).

Твърдение: Листата на T_M са точно върховете на T, които са макро-листа и техния брой е $\leq \frac{|V|}{B}$.

Доказателство: $v \in V_M$ е листо в $T_M \Leftrightarrow$ няма $u \in V_M : \ p_M(u) = v \Leftrightarrow$ за всяко $u \in V$ с $p(u) = v \notin V_M \Rightarrow$ всички синове на v са микро-върхове $\underbrace{\Rightarrow}_{\text{свойство 2}}$ $T_v \backslash \{v\}$ не съдържа макро-върхове.



Нека $\ell_1, \ell_2, \ldots, \ell_m$ са всички макро-листа. Тогава (по дефиниция) единствения макро връх в дървото T_{ℓ_i} е корена му $\ell_i \Rightarrow$ за $i \neq j : \ \ell_j \notin T_{\ell_i} \Rightarrow T_{\ell_i} \cap T_{\ell_j} = \emptyset$.

$$|V| \geq \left| \bigcup_{i=1}^{m} T_{\ell_i} \right| = \sum_{i=1}^{m} |T_{\ell_i}| \geq \sum_{i=1}^{m} B = B \times m \Rightarrow m \leq \frac{|V|}{B}.$$

Следствие:
Може да приложим решението от III върху дървото T_M и то ще даде време за индексиране $O(|V| + \frac{|V|}{B} \times \log|V|, 1)$. Остава да се справим с микро-дърветата. Т.е. дърветата T_u, за които u е микро-корен.

Знаем, че:
1. Размера на едно микро-дърво е $|T_u| \leq B - 1$
2. Колко са различните дървета с $\leq B - 1$ върха, защото тогава ще получим оценката за време $O\big((B-1)^2 \times$ #различни дървета с $\leq (B-1)$върха$\big)$ за времевата сложност.

Ойлерово обхождане на коренови дървета $T = (V, p, r)$: $EulerDFS(T, v)$ // резултатът е дума $w(T_v) \in \{0, 1\}^*$

procedure $EulerDFS(T, v)$

    $w \leftarrow \varepsilon$ (празната дума)

        for u : $p(u) = v$ **do**

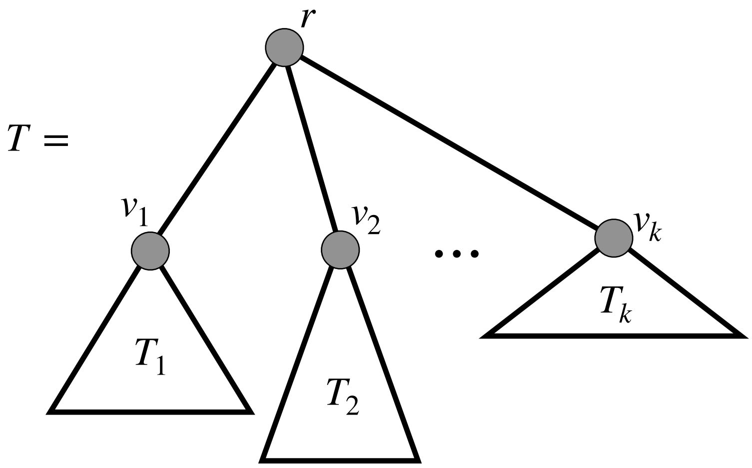
            $w \leftarrow w \circ 0 \circ EulerDFS(T, u) \circ 1$ // $\circ \leftarrow$ конкатенация

        done

    return w

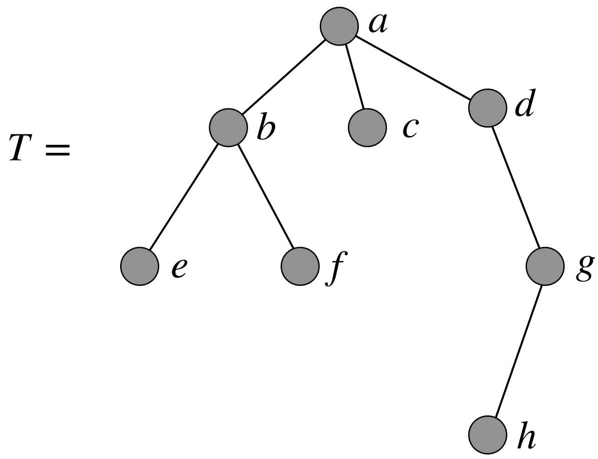
От математическа гледна точка дефинираме $w(T)$ с $\{0,1\}^*$ рекурсивно за всяко наредено кореново дърво T.

$T = (\{r\}, \emptyset, r), w(T) = \varepsilon$



$$w(T) = 0 \circ w(T_1) \circ 10 \circ w(T_2) \circ 10 \circ \cdots \circ 10 \circ w(T_k) \circ 1 = \prod_{i=1}^{k} \big(0 \circ (T_i) \circ 1\big).$$

<u>Пример</u>:



$$w(T) =$$
$$= \overset{0}{(a,b)} \circ \overset{0}{(v,e)} \circ \overset{1}{(e,b)} \circ \overset{0}{(b,f)} \circ \overset{1}{(f,b)} \circ \overset{1}{(b,a)} \circ \overset{0}{(a,c)} \circ \overset{1}{(c,a)} \circ \overset{0}{(a,d)} \circ \overset{0}{(d,g)} \circ \overset{0}{(g,h)} \circ \overset{1}{(h,g)} \circ \overset{1}{(g,d)} \circ \overset{1}{(d,a)} =$$
$$= \underbrace{00101101000111}_{2n-2}.$$

Свойства:

1. $w(T)$ е с дължина $2|V|-2$;
2. $\underbrace{|w(T)|_0}_{\text{брой нули}} = \underbrace{|w(T)|_1}_{\text{брой единици}} = \underbrace{|E(T)|}_{\text{брой ребра в } T} = |V|-1$;
3. Във всеки префикс $u \preccurlyeq pref\ w(T): \ |u|_0 \geq |u|_1$;
4. Нека $T_1 = (V_1, p_1, r_1)$ и $T_2 = (V_2, p_2, r_2)$ са (наредени) коренови дървета. Казваме, че T_1 и T_2 са изоморфни (с *еднаква* *(за нашите цели)* форма), ако:
   - има биекция $f: \ V_1 \to V_2$, за която
     - $f(r_1) = f(r_2)$
     - $f(p_1(v_1)) = p_1(f(v_1))$ за всяко $v_1 \in V_1$
     - Ако u_1 е i-тия син на v_1, то $f(u_1)$ е i-тия син на $f(v_1)$

Ако $w(T_1) = w(T_2)$, то T_1 е изоморфно на T_2. И обратното: ако T_1 и T_2 са наредени коренови дървета, които са изоморфни, то $w(T_1) = w(T_2)$.

Така получаваме, че броят на различните наредени коренови дървета с B върха са толкова колкото и думите $w \in \{0,1\}^{2(B-1)}$ за които $|w|_0 = |w|_1$ и $|pref\ w|_1 \preccurlyeq |pref\ w|_0$

Във всички случай броя на различните наредени коренови дървета с B върха е по-малък от $2^{2(B-1)}$.

Забележка: Ако $f: \ V_1 \to V_2$ е изоморфизъм на $T_1 = (V_1, p_1, r_1)$ и $T_2 = (V_2, p_2, r_2)$, то $f\left(p_1^{(d)}(v_1)\right) = p_2^{(d)}\left(f(v_1)\right), p_1^{(d)}(v_1) = f^{-1}\left(p_2^{(d)}\left(f(v_1)\right)\right)$.