Предшественик от определено ниво (част 1)

02.10.2020 г.

(Level Ancestor Query)

лектор: Стефан Герджиков
e-mail: stefangerdzikov@fmi-uni-sofia.bg

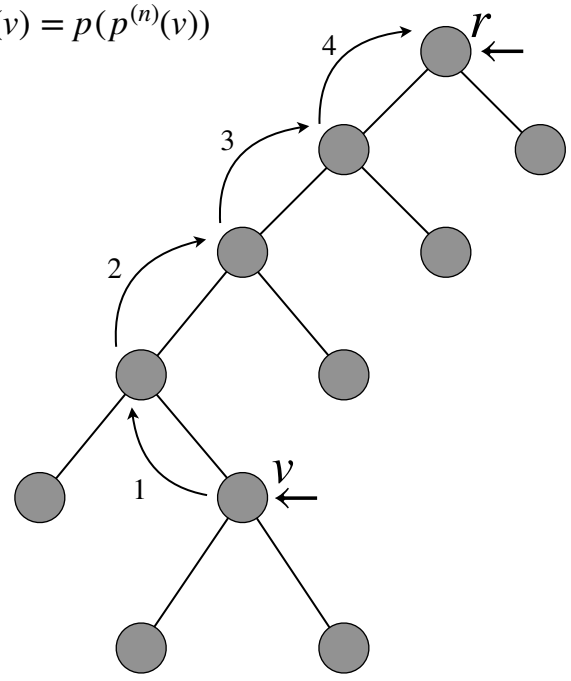
Коренови дървета:
$T = (V, p, r)$, където V е множество от върхове, $p : V \to V$ е функция на бащите (функция на прекия/непосредствен предшественик, наричан още *баща*), а r е корена на дървото.

Рекурсивна дефиниция на кореново дърво:
- $T = \big(\{r\}, \emptyset, r\big)$ е кореново дърво
- Ако $T = (V, p, r)$ е кореново дърво и $u \in V, v \notin V$, то $T = (V \cup \{v\}, p', r)$ също е кореново дърво, където $p'(x) = \begin{cases} p(x), & x \neq v \\ u, & x = v \end{cases}$

Някои означения:
За връх $u \in V$ на кореновото дърво $T = (V, p, r)$, ще означаваме с $d(u)$ дълбочината на върха u в T, където под дълбочина на даден връх ще разбираме разстоянието от този връх до корена на дървото. Функцията $d(u) : V \to V$ може да дефинираме рекурсивно по следния начин:
- $d(r) = 0$
- $d(u) = d(p(u)) + 1$

Височина на кореново дърво $T = (V, p, r)$ ще означаваме с $height(T) = \max\limits_{v \in V} d_T(v)$.

Дефиниция:
$T = (V, p, r)$ е кореново дърво. Дефинираме $p^{(n)} : V \to V$ за $n \in \mathbb{N}$ рекурсивно по n:
1. $p^{(0)}(v) = v$, за $v \in V$
2. $p^{(n+1)}(v) = p(p^{(n)}(v))$



$$n = 4$$
$$p^{(4)}(v) = r$$

Дефиниция: LA-проблем (предшественик от определено ниво) *Level Ancestor*.
Проблем: $T = (V, p, r)$ е дадено кореново дърво.
Вход: $v \in V, d \in \mathbb{N}$.
Изход: $p^{(d)}(v)$, ако $d(v) \geq d$ и $\perp$, иначе ($\perp$ е символа за *bottom* (дъно) или в програмния случай ще е *nullptr* или някаква стойност с която е недопустимо да се определи връх от дървото).

Дефиниция: Решение на LA-проблема ще наричаме двойка от алгоритми $\mathscr{A}_I$ и $\mathscr{A}_Q$, такива че:

1. $\mathscr{A}_I(T) = I$ (по дадено дърво $T = (V, p, r)$ - построява индекс I)
2. $\mathscr{A}_Q(I, v, d) = \begin{cases} p^{(d)}(v), & d \leq d(v) \\ \perp, & d > d(v) \end{cases}$

Сложност на решението $(\mathscr{A}_I, \mathscr{A}_Q)$ ще наричаме двойката от функции (f_I, f_Q), такива, че времевата сложност на $\mathscr{A}_I$ е $O(f_I(|V|))$, а на $\mathscr{A}_I$ е $O(f_Q(|V|))$.

Цел: Търсим решение на LA-проблема със сложност $(f_I(n), f_Q(v, d)) = <O(n), O(1)>$, което означава че, $f_I(n) \in O(n)$, $f_Q(v, d) \in O(1)$.

I. Решение на LA-проблема със сложност $< O(n^2), O(1) >$

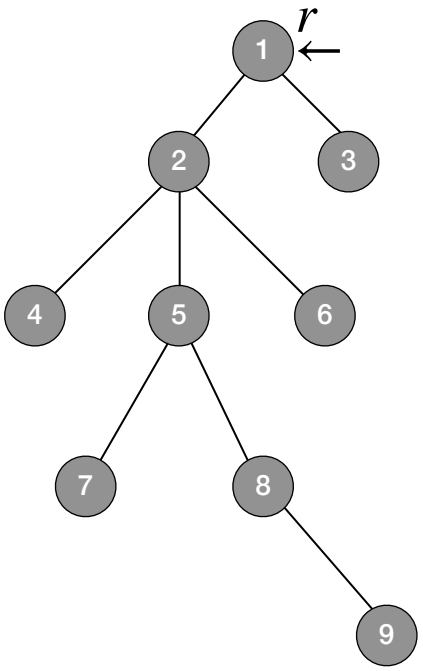
Идея: Ще попълним таблица $table[v][d]$ за $v \in V$ и $d \leq |V|$ по следния начин:

$$table[v][d] = \begin{cases} p^{(d)}(v), & d \leq d(v) \\ \perp, & d > d(v) \end{cases}, \text{ тогава}$$

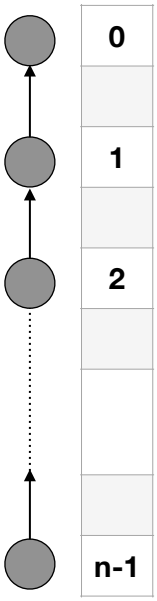
$$\mathscr{A}_Q(v, d) = \begin{cases} table[v][d], & d \leq |V| \\ \perp, & d > |V| \end{cases}$$

Това лесно може да стане по следния начин:

Кода по-долу е имплементиран на програмния език C++ и чака заявки от вида $(v, d) \in \mathbb{N}_0^2$ за следното дърво:



Да разгледаме един частен случай на дърво, а именно такова в което има само едно листо, тоест може да се разглежда като свързан списък. С други думи, тъй като няма да добавяме или изваждаме върхове в дървото (тъй като няма такова изисване за операции в условието), то това изродено дърво (списък) може да се разглежда и като масив.

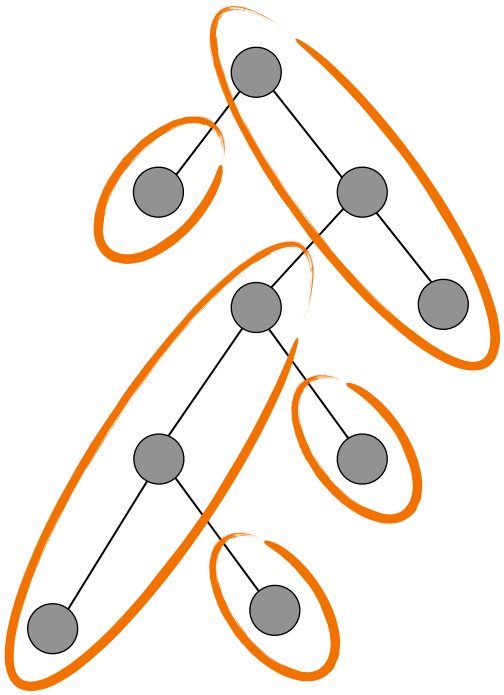


В такъв случай, може да използваме индексирането на масива като дълбочина на връх, тъй като дълбочината на даден връх също е естествено число. Следователно може да създадем следната структура от данни:

$A_I(T)$:
for each v in V
  A[d(v)]=v;

И да я използваме по следния начин: $A_Q(A, v, d) = \begin{cases} A[d(v) - d], & d \leq d(v) \\ \perp, & d > d(v) \end{cases}$

В този случай имахме линейна сложност по памет и константна сложност за отговаряне на всяка заявка. Може ли по някакъв начин да използваме тази идея от частния случай и да я пренесем и в общия случай?



Дефиниция: Максимален път за кореново дърво $T = (V, p, r)$ ще наричаме $\Pi = (v_0, v_1, \ldots, v_k)$, за който е изпълнено:

1. $p(v_{i-1}) = v_i$, за $1 \leq i \leq k$
2. $height(T_{v_i}) = i$, където T_{v_i} е поддърво на дървото T, което има за корен v_i

Следствие: Ако Π е максимален, то $height(T_{v_0}) = 0$ и v_0 е листо в дървото T.

Дефиниция: Разбиване на $T = (V, p, r)$ на максимални пътища е множеството $\Pi = \left\{ \Pi_i \right\}_{i=1}^{I}$, което е такова че:

1. Π_i е максимален за $\forall i$
2. $V = \bigcup_{i=1}^{I} \Pi_i$
3. $\Pi_i \cap \Pi_j = \emptyset$ за $i \neq j$

Тук просто приложихме дефиницията за разбиване от ДС 1, Теория 1, дефиниция 3 - https://github.com/andy489/Discrete_Structures/blob/master/DS%201%20Theory/Theory%201.pdf

II.1. Намиране на разбиване на максимални пътища за времева сложност с порядък принадлежащ на $O(|V|)$.

Предположения:
1.) Знаем колекцията $Leaves$ от листа на дървото;
2.) За всеки връх знаем неговата дълбочина: $d(v)$ за $\forall v \in V$.

II.2. Сортиране на *листата* по тяхната дълбочина $d(v)$:

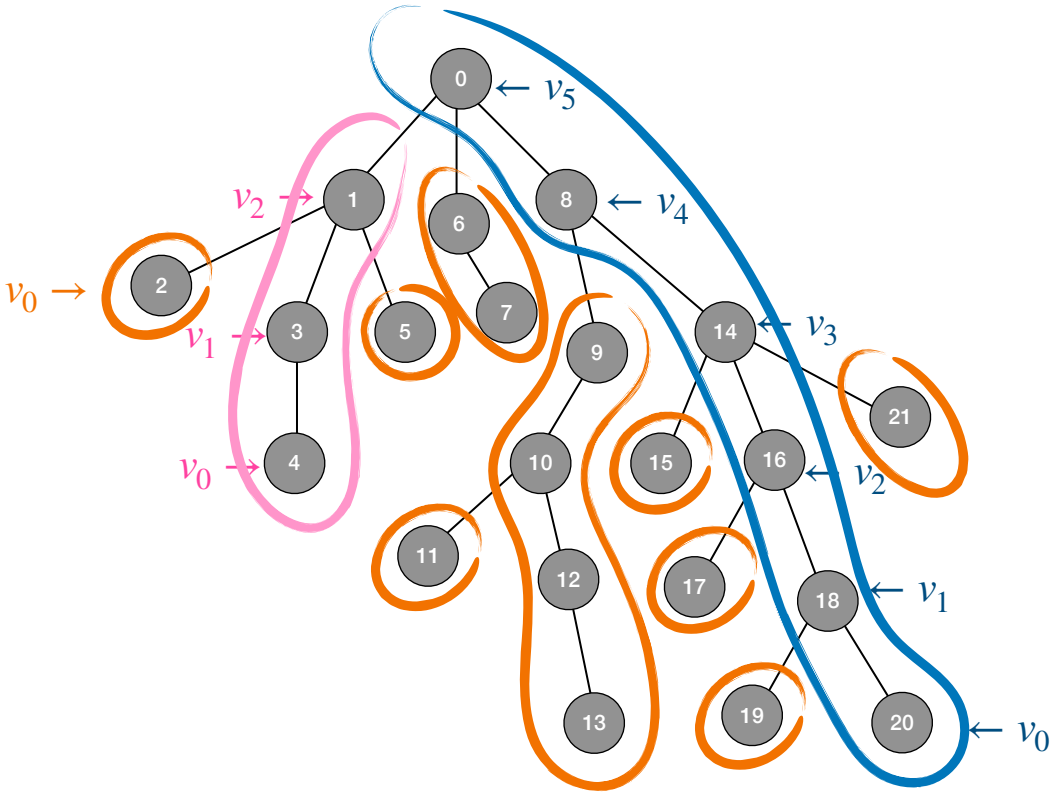
За реализирането на този подалгоритъм ще използваме идеята на *counting sort*.
1. Създаваме масив $AL[0 \ldots n-1]$ от $|V| = n$ на брой списъка
2. За $\forall$ листо $\ell \in Leaves$: $AL[d(\ell)].insert(\ell)$
3. Създаваме сортирания масив $SL[0 \ldots |Leaves| - 1]$ от листа

Псевдо код:

```
int k = 0
for d = |V| - 1 down to 0 do
  while AL[d] ≠ Ø do
     ℓ = AL[d].extract()
     SL[k++]=ℓ
  done
done
```

II.3. Разбиване на $T = (V, p, r)$ на максимални пътища:

1. Намираме $SL[0 \ldots |Leaves| - 1]$ от сортирани листа
2. $\prod \leftarrow \emptyset$ \\ множество от максимални пътища
   $mark[0 \ldots |V| - 1]$
   $for\ v = 0\ to\ |V| - 1$
   $mark[v] \leftarrow false$ // маркираме, че не участва в максимален път
3. $for\ \ell = 0\ to\ |Leaves - 1|\ do$
   $\Pi \leftarrow \emptyset$
   $v \leftarrow SL[\ell]$
   $while\ v\ is\ defined\ and\ mark[v]\ equals\ false$
   $\Pi \leftarrow \Pi \cup v$
   $mark[v] \leftarrow true$
   $v \leftarrow p(v)$
   $done$
   $\prod \leftarrow \prod \cup \Pi$
   $done$



Времева сложност на описания по горе алгоритъм: $O(|V|)$
Обяснение на сложността: Всяка стъпка от тялото на while цикъла съответства на промяна от масива, в който отбелязваме посетените върхове от $mark[v] = false$ към $mark[v] = true$. Нещо повече - веднъж вдигнат флага $mark[v]$ на $true$, никъде след това не се променя. Следователно броят пъти, които се изпълнява while цикъла е $O(|V|)$. Освен това за всяко листо имаме константна работа в тялото на for цикъла, но извън тялото на while цикъла. Тоест общо $O(|V| + |Leaves) = O(|V|)$, тъй като листата са от порядъка на всички върхове в дървото (не може да са повече).

Разглеждане на инварианти на изградената структура от максимални пътища:
Нека Π_j е пътя построен от нашия алгоритъм при j-тата итерация от for-цикъла.
1.) Ако $v \in \Pi_j$, то $mark[v] = true$ след j-тата итерация на for-цикъла.
2.) Ако $mark[v] = true$ след j-тата итерация на for цикъла, то $mark[p(v)]$ също ще е равно на $true$ след j-тата итерация на for цикъла.
3.) Ако $v \in \Pi_j$, то $p(v) \in \bigcup_{i=0}^{j} \Pi_i$.
4.) Всеки от пътищата Π_j е максимален.

Нека $v \in \Pi_j \Rightarrow mark[v] = true$ при j-тата итерация и освен това $mark[v]$ е $false$ при всяка от итерациите $0, 1, \ldots, j-1$.
Нека ℓ е произволно листо в поддървото T_v. Ако $\ell \in \Pi_i$ с $i < j$, то от 3.) следва, че $p(\ell), p^{(2)}(\ell), \ldots \in \bigcup_{k=0}^{i} \Pi_k$, в частност $v \in \bigcup_{k=0}^{i} \Pi_k \Rightarrow mark[v] = true$ след i-тата итерация на for цикъла. Но $i < j$, което е противоречие с допускането, с допускането, че $i < j$ и с което доказваме, че всяко листо $\ell \in T_v$ има индекс $\geq j$ в масива SL и от това, че масива е сортиран $\Rightarrow d(\ell) \leq d(SL[j])$.
Но за върха $\ell \in T_k$, $d_{T_k}(\ell) = d(\ell) - d(v) \Rightarrow d_{T_v}(SL[j]) = \max_{\ell \in T_v} T_v \Rightarrow$ $height(T_v) = d(SL[j]) - d(v)$, но това е точно позицията в Π_j, на която се записва v.

II.4. Решение на LA-проблема със сложност $< O(n), O(\sqrt{n}) >$

Нека $\prod = \{\Pi_i\}_{i=1}^{I}$ е разбиване на максимални пътища на входното дърво.

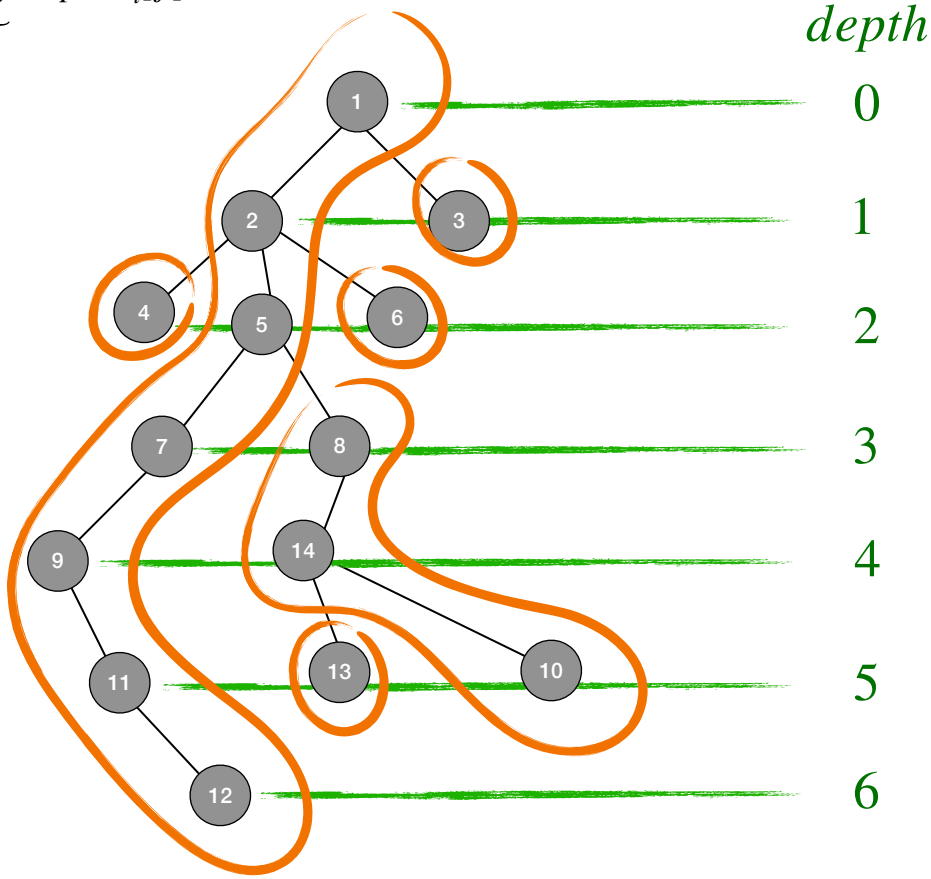
Ще опишем структурата която ще създадем, както и начина, по който ще я използваме:

За всеки път $\Pi_i = (v_0, v_1, \ldots, v_k)$ поддържаме масиви

$path_i[0 \ldots k]$ и
$path_i[j] = v_j$.

За всеки връх $v \in V$:

$index_p(v) = i \Rightarrow v \in \Pi_i$
$\underbrace{index(v) = j \Rightarrow path_i[j] = v}_{height(v)}$



$LAQ_2(v, d)$
  $if\ d > d(v)\ then\ return\ \perp$
  $i \leftarrow index_p(v)$
  $u \leftarrow path_i(len[i] - 1)$
  $d' \leftarrow height(v) - height(u)$ // *height(v) е височината на поддървото с корен v*
  $if\ (d \leq d')\ then\ return\ path_i[height(v) + d]$
  $else\ return\ LAQ_2(p(u), d - d' - 1)$

Времева сложност на алгоритъма: $O(\sqrt{n})$
За домашно - да се помисли и да се аргументира защо е такава сложността. Насоки: да се помисли в следната посоката - какво се случва с дължината на пътя $len[i]$, където $i = index(v)$?