Tomáš Oberhuber

Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague

Video na Youtube

Vyhledávání více vzorů

- pro zadaný text T a množinu vzorů P^i chceme najít všechny výskyty všech vzorů

Example 1

T = `ATGGTCGGT`
P^1 = `GGT`
P^2 = `CGG`
P^3 = `ATG`

- výsledek je $1, 3, 6, 7$

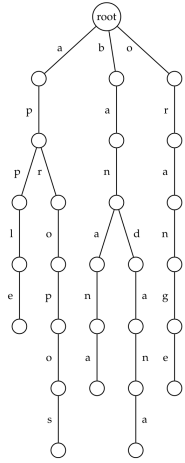
Vyhledávání více vzorů

- klasický postup by vedl na složitost $O(knm)$
- 1975, Alfred Aho, Margaret Corasick – keywords tree
  - algoritmus se složitostí $O(N + m)$, kde $N = \sum_{i=1}^{k} |p_i|$

Example 2

- chceme hledat následující slova:
  - `apple, apropos, banana, bandana, orange`
- vytvoříme si následující strom – *keywords tree*

Vyhledávání více vzorů



Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

Vyhledávání více vzorů

Strom lze definovat takto:

1. každá hrana je označena písmenem
2. dvě hrany jdoucí ze stejného vrcholu mají různá písmena
3. každý řetězec P_i je zapsán na nějaké cestě od kořene do určitého listu

Vyhledávání více vzorů

- při hledání některého vzoru na i-té pozici textu procházíme strom odshora dolu a porovnáváme s příslušnými znaky $T_i, T_{i+1}, \ldots$
- pokud dojdeme až do některého listu stromu, našli jsme příslušný vzor
- tento postup má složitost $O(N + nm)$, kde N je sestrojení stromu
- Aho, Corasick sestrojili algoritmus se složitostí $O(N + m)$
- tento algoritmus zde nebudeme rozebírat

Sufixové stromy

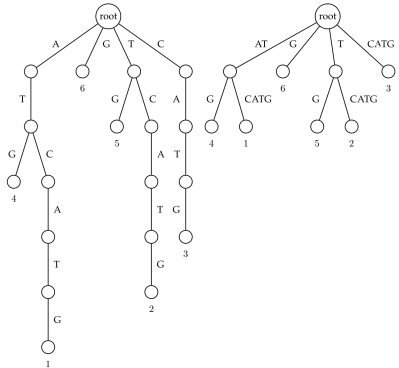
- jde o datovou strukturu, která umožňuje v daném textu hledat se složitostí $O(m)$, tj. nezáleží na délce textu, ve kterém hledáme
- strom lze zkonstruovat se složitostí $O(n)$
- celková složitost je tak $O(m+n)$
- myšlenka je jednoduchá
- vezmeme všechny sufixy daného textu a vytvoříme z nich *keyword tree*

Example 3

- mějme text `ATCATG`
- ten má sufixy `G, TG, ATG, CATG, TCATG, ATCATG`

Sufixové stromy

Example 4



Zdroj: Jones, Pevzner, An introduction to bioinformatics algorithms

- `G, TG, ATG, CATG, TCATG, ATCATG`

Definition 5

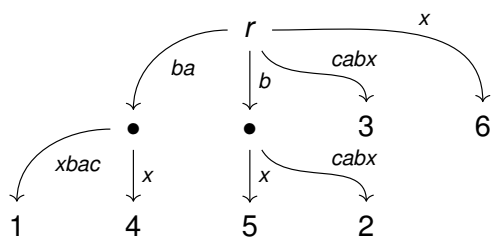
Sufixový strom pro text $T[1 \ldots n]$ splňuje:

1. má přesně n listů očíslovaných $1, \ldots, n$
2. kromě kořenu má každý vnitřní uzel alespoň dva potomky (následovníky)
3. každá hrana je označena nějakým neprázdným podřetězcem textu T
4. každé dvě hrany jdoucí ze stejného vrcholu mají popisky začínající různým znakem
5. řetězec, který získáme spojením všech popisků hran na cestě z kořene do listu i odpovídá sufixu $T[i \ldots n]$

Example 6

T =abcabx

i	$T[i \ldots n]$
1	abcabx
2	bcabx
3	cabx
4	abx
5	bx
6	x



Sufixové stromy

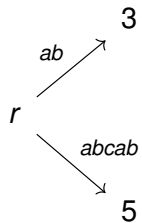
- hledání vzoru P se provádí pomocí procházení stromu od kořene k nějakému listu
- začínáme v kořeni a podle jednotlivých písmen vzoru P se rozhodujeme, které hrany budeme procházet směrem k listům
- jde o tzv. *threading*
- pokud další znak popisku hrany nesouhlasí s dalším znakem vzoru P nebo z uzlu nevede žádná hrana s popiskem začínajícím stejným znakem, který následuje v P, vzor nebyl nalezen - jde o tzv. *incomplete threading*
- pokud takto projdu všechny znaky vzoru P, jde o tzv. *complete threading*

- pokud complete threading končí v listu, jeho index udává pozici výskytu vzoru P v textu T
  - jde totiž o index i sufixu $T[i, \ldots, n]$
- pokud complete threading končí uvnitř stromu, znamená to, že je v textu více výskytů vzoru
- jejich pozici lze dopočítat tak, že dojdu v daném podstromu k jednotlivým listům a v nich najdu index počátku daného sufixu

Problém nastává u řetězců jejichž některý sufix je i jejich prefixem.

Example 7

- T =abcab $\rightarrow$ b, ab, cab, bcab, abcab
- sufix "ab" je i prefixem
- jsou dvě možnosti jak uložit do stromu sufixy "ab" a "abcab"



Nesplňuje 4 Nesplňuje 1,5

- řešením je přidání zakončovacího znaku (*termination character*), který se nevyskytuje nikde v textu T
- budeme ho značit jako dolar \$
- tím pádem už žádný sufix nemůže tvořit prefix jiného sufixu

Definition 8

Popiskem cesty vedoucí z kořene budeme rozumět řetězec poskládaný z popisků hran vedoucích podél cesty.

Sufixové stromy - naivní konstrukce

- naivní konstrukce spočívá ve vkládání jednotlivých sufixů textu T
- $\mathcal{S}(T)$ bude značit sufixový strom k textu T
- $\mathcal{S}_i(T)$ bude značit sufixový strom po vložení i sufixů
- jako první vložíme sufix $T[1 \ldots n]\$$ tj.

$$\mathcal{S}_0(T) \rightarrow \mathcal{S}_1(T)$$

- v kroku $i + 1$ vkládáme $T[i + 1 \ldots n]\$$ tj.

$$\mathcal{S}_i(T) \rightarrow \mathcal{S}_{i+1}(T)$$

Sufixové stromy - naivní konstrukce

Krok $i + 1$ probíhá takto:

- procházíme $\mathcal{S}_i(T)$ od kořene r tak, abychom našli co nejdelší prefix sufixu $T[i + 1 \ldots n]$ odpovídající popisku cesty z kořene do určitého vrcholu v
- pokud žádná takové cesta neexistuje, vytvoříme nový list s s indexem $i + 1$ a spojíme ho s kořenem hranou k níž přiřadíme popisek $T[i + 1 \ldots n]\$$
- pokud taková cesta existuje, pak $T[i + 1 \ldots n]\$$ nemůže být podřetězcem popisku této cesty, protože obsahuje terminální znak \$
- musí tedy existovat prefix $T[i + 1 \ldots k]$ takový, že $T[k + 1]$ neodpovídá pokračování žádné cesty ve stromu $\mathcal{S}_i(T)$

Sufixové stromy - naivní konstrukce

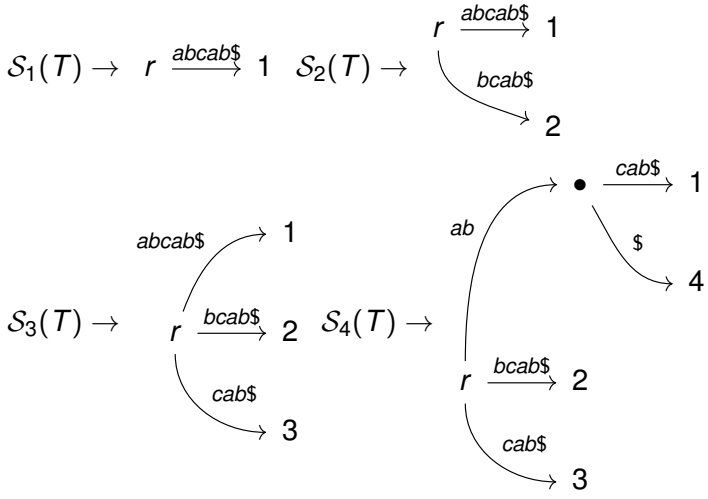
- buď (u, v) hrana, kde končí cesta s popiskem $T[i + 1 \dots k]$
- pokud cesta končí uprostřed hrany
  - vytvoříme nový uzel w a hranu (u, v) nahradíme hranami (u, w) a (w, v)
  - hraně (u, w) přiřadíme část popisku hrany (u, v) končící na pozici k a hraně (w, v) zbytek popisku hrany (u, v)
  - vytvoříme list s s indexem $i + 1$ a vytvoříme hranu (w, s) s popiskem $T[k + 1 \dots n]\$$
- pokud cesta končí ve vrcholu v
  - vytvoříme list s s indexem $i + 1$ a vytvoříme hranu (v, s) s popiskem $T[k + 1 \dots n]\$$
- algoritmus končí vložením znaku \$

Example 9

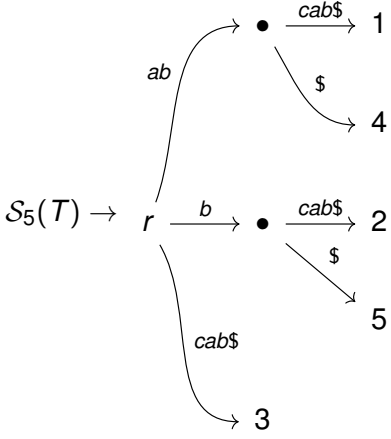
T =abcab\$

i	$T[i \ldots n]$\$
1	abcab\$
2	bcab\$
3	cab\$
4	ab\$
5	b\$
6	\$

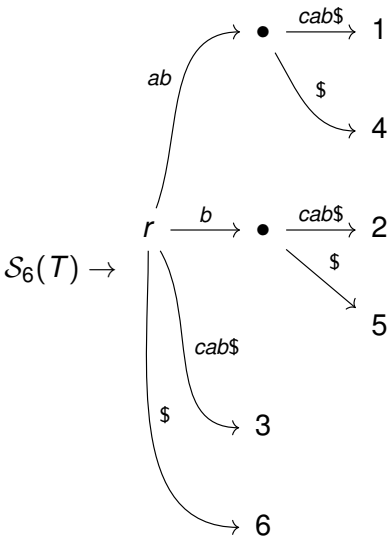
Sufixové stromy - naivní konstrukce



Sufixové stromy - naivní konstrukce



Sufixové stromy - naivní konstrukce



Sufixové stromy - naivní konstrukce

- náročnost vložení sufixu o délce i je $O(i)$
  - musíme postupně porovnat všechny znaky sufixu a popisky hran v grafu
- celková složitost je tedy $O(n^2)$
- Ukkonenův algoritmus dokáže konstruovat strom se složitostí $O(n)$