Suffixové stromy

http://jakub.kotrla.net/suffixtrees/

Osnova:

- Motivační příklad
- Přehled možných řešení
- Definice suffixového stromu
- Řešení pomocí suffixových stromů
- Konstrukce suffixového stromu
- Další použití, suffix arrays

Motivační příklad

Výzkum DNA

Máme obrovskou databázi genů (posloupností bazí).
Výzkumníci nám dávají geny či jejich části
a chtějí vědět, zda se vyskytují v naší databázi.

Tedy tzv. „substring problem“

vstup: text T délky m a řetězec P délky n
výstup: zda je P obsažen v T

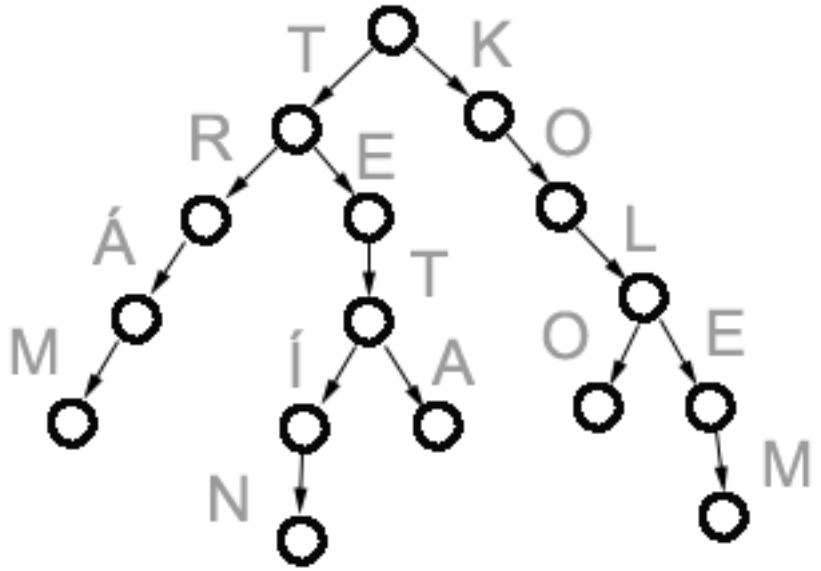
Přehled řešení

- Naivní řešení pracující v čase $O(m^2)$
- Řešení pracující v $O(m+n)$
  - Knuth-Morris-Pratt
  - Moore-Boyle
  - Aho-Corasick
  - Suffixové stromy

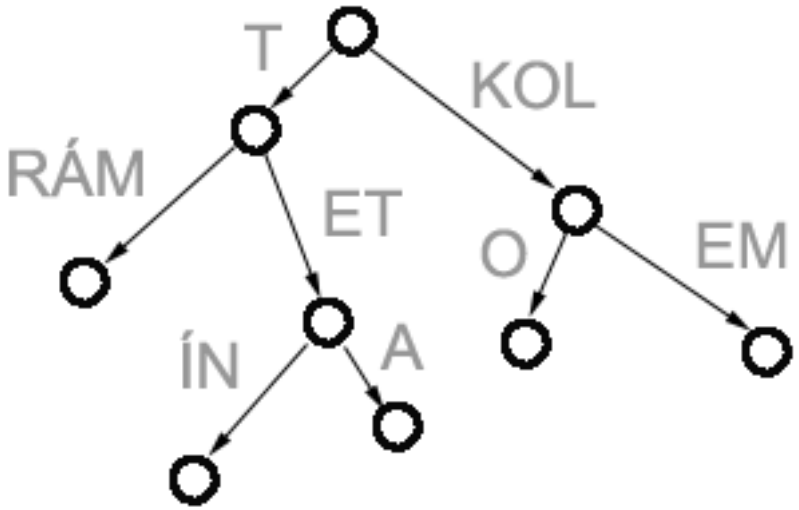
Všechna tato řešení pracují s předpokladem konečné a malé abecedy. Jejich složitost (časová nebo paměťová) je ve skutečnosti $O((m+n)*|\Sigma|)$.

Trie, patricia

Trie pro slova:
trám, teta, tetín, kolem, kolo



Patricia
(Practical Algorithm to Retrieve Information Coded in Alphanumeric, D.R.Morrison (1968))
pro ta samá slova



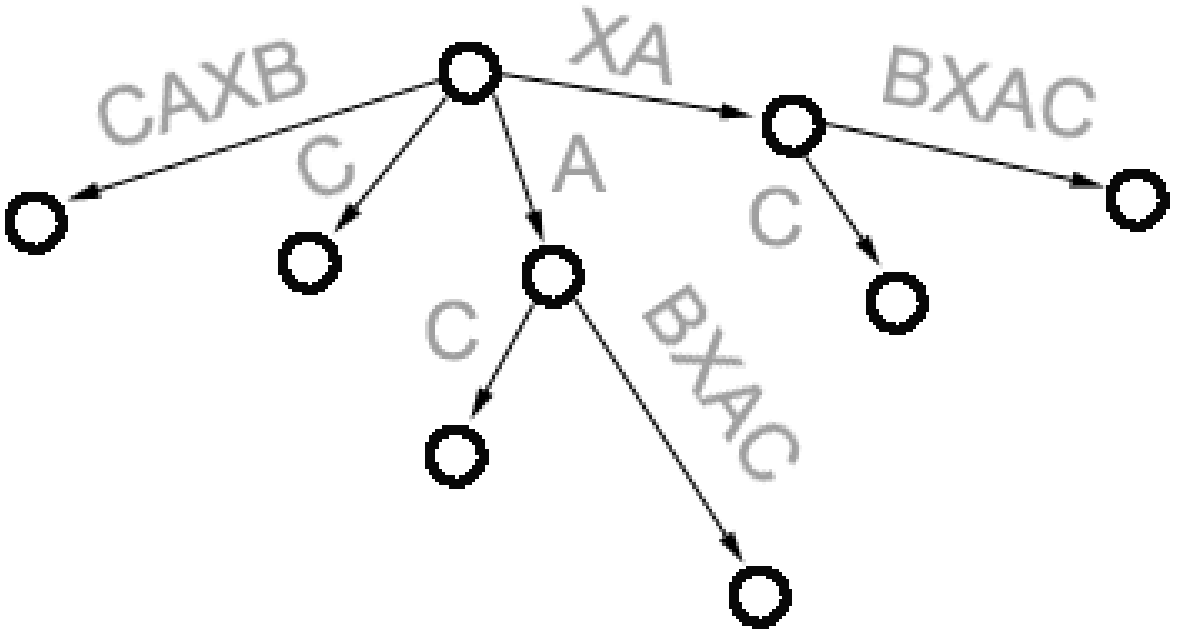
Suffixový strom - definice

Suffixový strom pro text T délky m je zakořeněný strom s m listy číslovanými $1..m$. Každý vnitřní bod mimo kořen má alespoň dva syny, každá hrana je označená podřetězcem T. Všechny hrany vedoucí ze stejného vrcholu začínají různým písmenem.
Spojení podřetězců na cestě z kořene do listu i je suffix T začínající na pozici i – $T[i..m]$.

Suffixový strom pro text T je vlastně patricia všech suffixů textu T.

Tato definice nezaručuje, že pro každý text T existuje suffixový strom. Pokud nějaký suffix je prefix jiného suffixu, jeho cesta nekončí v listu.

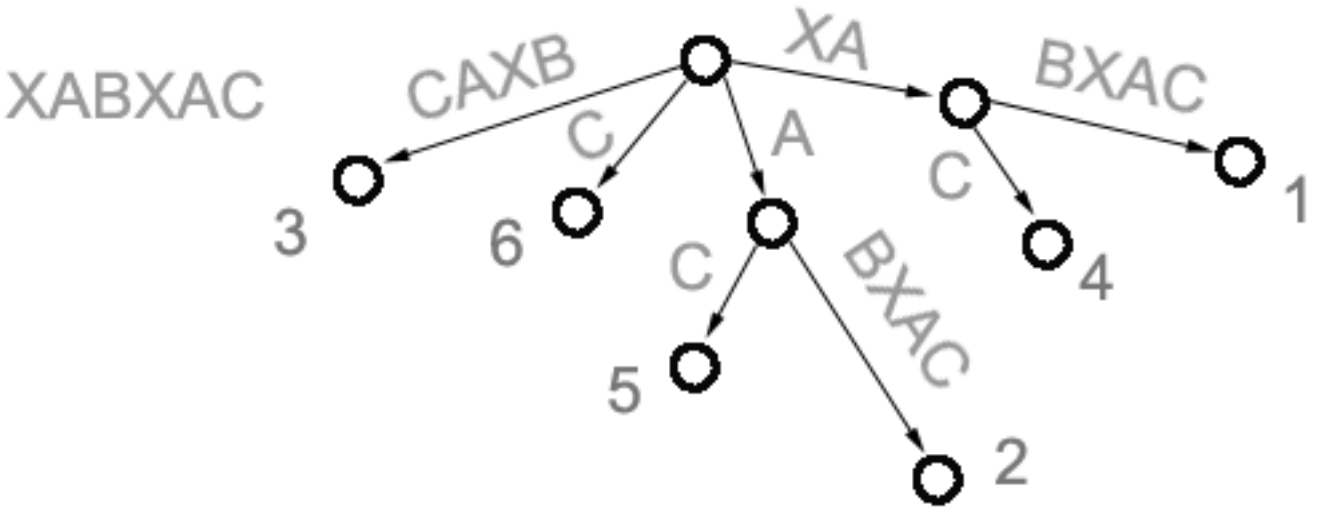
Suffixový strom - příklad



Suffixový strom pro slovo **xabxac.**

Pokud bychom odebrali poslední znak (*T*=**xabxa**), pak suffix **xa** je prefix suffixu **xabxa** a jeho cesta nekončí v listu. Proto je nutné, aby se poslední znak *T* nikde jinde v *T* nevyskytoval (přidáme speciální ukončovací znak - $).

Suffixový strom - vyhledávání



Vyhledání řetězce *P* v *T* je průchod suffixovým stromem. Porovnáváme postupně znaky z *P* při průchodu stromem, časem nastane jedna z možností:

- znak z *P* neodpovídá žádné hraně z aktuálního vrcholu
  (tedy *P* není v *T*)
- Projdeme celý řetězec *P*
  (všechny listy pod aktuálním vrcholem odkazují na výskyt *P* v *T*)

Srovnání řešení

Máme-li text T délky m a řetězec S délky n, pak všechny předcházející metody (mimo naivní) pracují v čase $O(n+m)$.

Suffixové stromy

předzpracují T v $O(m)$
(postaví suffixový strom).
Vyhledání řetězce pak trvá $O(n)$.

Ostatní metody

předzpracují P v $O(n)$
a vyhledání v T pak
trvá $O(m)$.

Pokud je m výrazně větší než n a chceme-li vyhledávat mnoho řetězců v pevném textu, jsou suffixové stromy lepší řešení.

Suffixové stromy - historie

První algoritmus s lineární časovou složitostí, jak postavit suffixový strom, uveřejnil v roce 1973 Weiner.

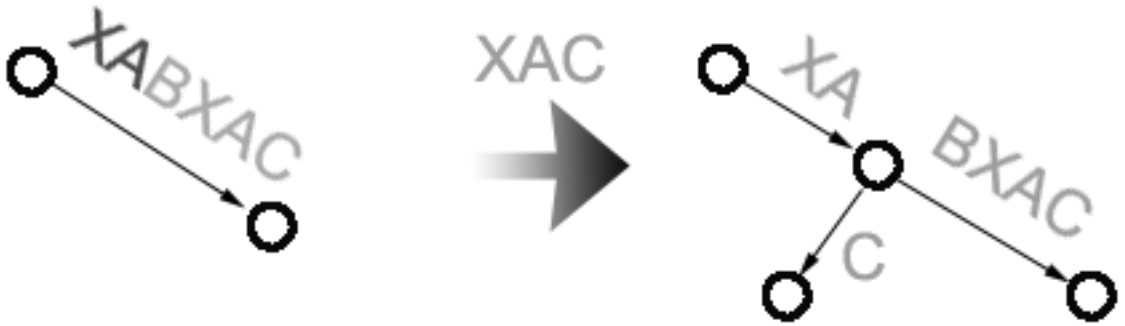
V roce 1976 McCreight popsal algoritmus, který je čitelnější a má menší paměťové nároky.

Oba sice pracovaly v lineárním čase, ale zpracovávaly *T* zprava - odzadu. Až v roce 1995 vymyslel Esto Ukkonen koncepčně odlišný algoritmus, který zpracovává *T* zleva.

Suffixový strom - pomalá konstrukce

Suffixový strom můžeme postavit jako patricii, tedy postupně přidávat suffixy $T[i..m]$ pro $i = 1..m$. Jako N_i označme dočasný strom, který obsahuje suffixy $1..i$.

N_1 je jediná hrana mezi kořenem a listem 1. Konstrukce N_{i+1} z N_i: při průchodu N_i postupně porovnáváme znaky z přidávaného suffixu, až žádná další shoda není možná. V tom okamžiku jsme ve vrcholu nebo na hraně (vrchol si musíme vytvořit) a z něj vytvoříme novou hranu a list.



Tento postup má časovou složitost $O(m^2)$.

Implicitní suffixový strom

Ukkonenův algoritmus konstruuje posloupnost implicitních suffixových stromů.

Jestliže ze suffixového stromu pro T\$ odstraníme všechny výskyty ukončovacího znaku \$ a vyčistíme jej, získáme implicitní suffixový strom pro T.

Implicitní suffixový strom pro $T[1..i]$ označme I_i.

Ukkonenův algoritmus - pohled shora

Ukkonenův algoritmus má *m* fází. Ve fázi *i* postaví strom I_i z I_{i-1}. Každá fáze *i* se dělí do *i* rozšíření (extension).

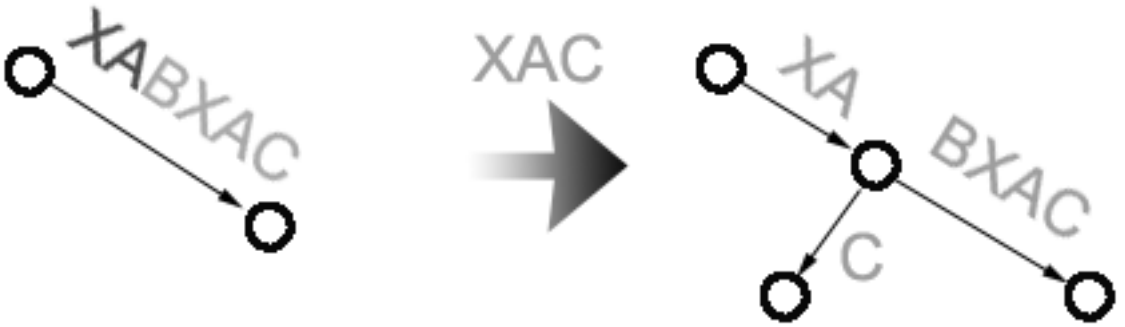
Ve fázi *i* a v rozšíření *j* algoritmus přidá do stromu *T*[*j*..*i*]. Tedy nejdříve najde ve stromě konec cesty *T*[*j*..*i*-1]. Nyní nastane jeden ze tří případů (použijeme jedno z pravidel):

- cesta končí v listu, tedy přidáme znak *T*[*i*] na konec cesty

Ukkonenův algoritmus - pohled shora 2

- cesta končí uprostřed hrany nebo ve vnitřním vrcholu a z něj nevede žádná hrana začínající znakem $T[i]$,

  Vytvoříme novou hranu označenou $T[i]$ (a list), pokud cesta končí uprostřed hrany, musíme ji rozdělit novým vrcholem



- cesta končí ve vnitřním vrcholu a z něj vede hrana začínající znakem $T[i]$, tedy tento suffix už ve stromě je a neděláme nic.

Ukkonenův algoritmus - zatím $O(m^3)$

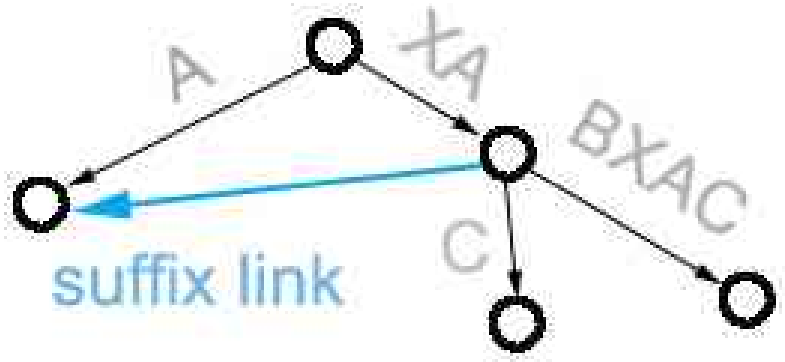
Algoritmus má m fází, v každé nejvýše m rozšíření. Jedno rozšíření znamená najít konec cesty a přidat znak $T[i]$. Nalezení znaku zvládneme v $O(m)$, přidání znaku už je konstanta.

Zatím tedy máme algoritmus pracující v $O(m^3)$. Zkusíme ho zrychlit.

První zrychlení - suffix links

Nechť $x\alpha$ je řetězec, kde x je znak a α podřetězec (může být prázdný). Nechť v je vnitřní vrchol označený cestou $x\alpha$, pokud existuje vrchol $s(v)$ označený cestou α, pak ukazatel z v do $s(v)$ je *suffix link*.

Každý vnitřní vrchol má suffix link, kromě posledního vytvořeného (ten ho dostane do konce přístího rozšíření).



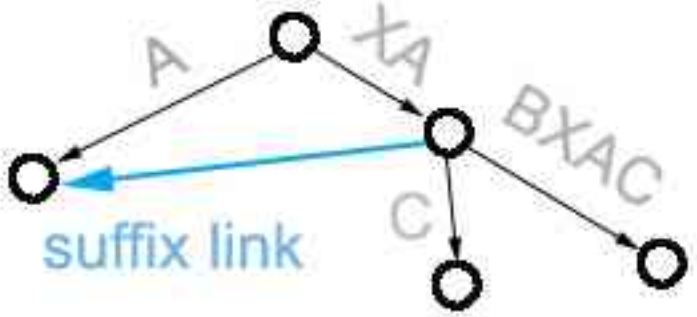
SEA – single extension algorithm

- Najdi na konci cesty $T[j\text{-}1..i]$ nebo nad ním vrchol v, který má suffix link nebo je kořen. To znamená jít po nejvýše jedné hraně. Nechť α je řetězec, který jsme přešli (může být prázdný).

- Pokud v není kořen, jdi přes suffix link do $s(v)$ a odtamtud po cestě pro řetězec α. Pokud v je kořen, jdi po cestě pro řetězec $T[j..i]$.

- Použij jedno z pravidel pro rozšíření.

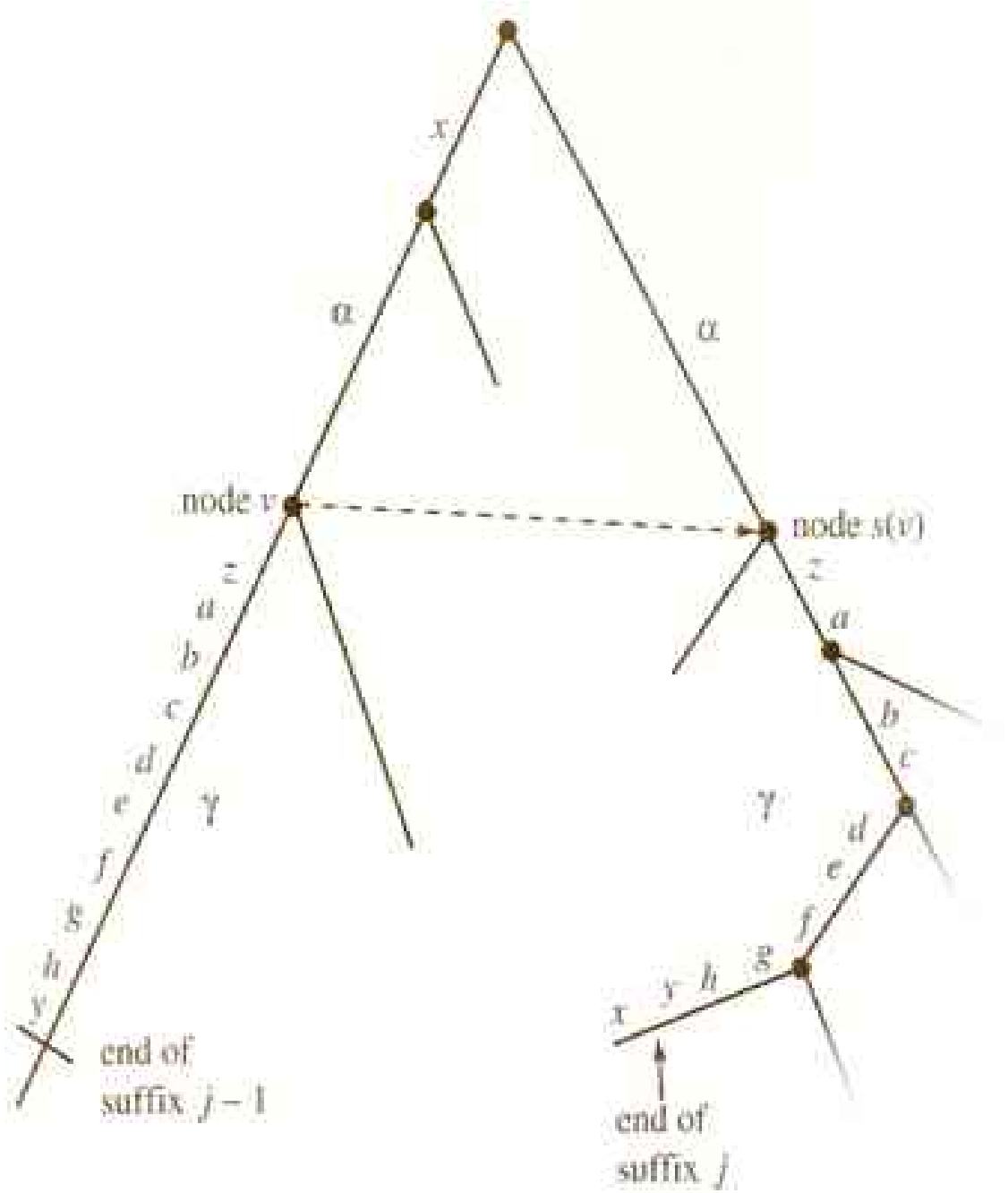
- Pokud byl v minulém rozšíření vytvořen vnitřní vrchol w, vytvoř suffix link.



Skip/count trik

Algoritmus jde v každém rozšíření po cestě pro řetězec α. To znamená porovnávat znak po znaku a zabere to čas odpovídající $|\alpha|$.

Pokud budeme znát délku řetězce na každé hraně a délku α, můžeme vynechat některá porovnávání, takže čas nutný na cestu dolů bude odpovídat počtu počtu vrcholů na této cestě.



Ukkonenův algoritmus - už $O(m^2)$

Díky skip/count triku bude chůze po cestě dolů v rámci celé fáze trvat $O(m)$.

Pomocí suffix links a skip/count triku trvá každá fáze $O(m)$.

Fází je nejvýše m, takže celý algoritmus pracuje v čase $O(m^2)$.

Jak zapsat O(m^2) dat v čase O(m)

Na každé hraně máme uložený podřetězec textu T. Suffixový strom může takto zabrat až O(m^2) místa. Abychom ho mohli postavit v O(m), musíme změnit způsob reprezentace stromu.

Místo abychom k hraně ukládali podřetězec, zapíšeme k ní pouze dvě čísla – začátek a konec jejího podřetězce v T. Na jiném a jediném místě si pak bude algoritmus uchovávat celý T.

Paměťová složitost algoritmu pak bude O(m).

Stop pravidlo

Pokud ve fázi i v rozšíření j použijeme pravidlo 3 (cesta končí ve vnitřním vrcholu a z něj vede hrana začínající znakem $T[i]$, neděláme nic), použijeme ho i pro všechny následující rozšíření v této fázi.

Cesta $T[j..i]$ pokračuje znakem $T[i]$, takže cesta $T[j+1..i]$ také.

Pravidlo 3 říká, že v tom případě nemusíme nic dělat. Navíc nový suffix link musíme přidat pouze po použití pravidla 2.

Takže můžeme zrychlit algoritmus heuristikou, která říká:
pokud použiješ pravidlo 3, ukonči aktuální fázi. Další rozšíření již není nutné dělat, jsou udělány implicitně.

List zůstává listem

Pokud během algoritmu vytvoříme nový list, zůstane listem ve všech dalších stromech. Algoritmus nikdy nepřekročí list, jen rozšiřuje hranu, která k němu vede.

Na začátku každé fáze je posloupnost rozšíření, které použijí pravidlo 1 nebo 2. Nechť j_i je poslední takové rozšíření.

Platí $j_i \leq j_{i+1}$. Všem těmto rozšířením se můžeme vyhnout a nechat je provést implicitně.

Pokud ve fázi i vytvoříme novou hranu, označíme ji T[p..i] resp. indexy (p, i). Místo toho ji označme (p,e), kde e je globální index označující aktuální konec.

V každé fázi pak pouze zvýšíme index e a implicitně tak provedeme rozšíření 1.. j_i.

SPA – single phase algorithm

Algoritmus pro fázi i:

- Zvyš e na i. (tím implicitně provedeme rozšíření $1..j_i$)
- Proveď rozšíření (pomocí SEA) od j_i+1 až do rozšíření j^*, které použije pravidlo 3 nebo až budou všechna rozšíření v této fázi vyčerpána (implicitně provedeme rozšíření $j^*..i$)
- Polož $j_{i+1} = j^* - 1$.

Každé dvě následující fáze mají nejvýše jeden společný index rozšíření. Počet fází je m, takže algoritmus provede nejvýše $2m$ explicitních rozšíření.
Navíc fáze i+1 ví, kde skončila předchozí, a první rozšíření tak nemusí hledat konec cesty. První rozšíření v každé fázi trvá konstantní čas.

Ukkonenův algoritmus - konečně O(m)

Algoritmus provede nejvýše $2m$ explicitních rozšíření. Jedno rozšíření je konstanta a čas na přechod cestou dolů, úměrný počtu vrcholů na cestě.

Uvažujme h jako aktuální vrcholovou hloubku. Ta se nemění mezi jednotlivými rozšířeními (ani když jsou v jiných fázích).

V každém rozšíření se nejdříve sníží nejvýše o dva (krok nahoru a suffix link). Při krocích dolů po cestě se zvýší o jedna za každý vrchol na cestě. Nejvyšší možná hodnota h je m a počet explicitních rozšíření je $2m$. Takže počet vrcholů přeskočených cestou dolů je za celý běh algoritmu nejvýše $3m$.

Celý algoritmus tedy pracuje v čase O(m).

Z implicitního na opravdový

Dokážeme zkonstruovat implicitní suffixový strom v čase O(m).
Ve stejném čase ho dokážeme převést na suffixový strom.

Nejdříve přidáme k T ukončovací znak \$ a provedeme ještě jednu fázi. Pak bude každý suffix končit v listě.

Ještě musíme ve všech listech nahradit globální index číslem m.

Výsledný strom bude opravdový suffixový strom postavený v čase O(m).

Další použití

Kromě nalezení řetězce délky n v čase O(n) (a varianty pro více řetězců či textů) nám suffixové stromy umožňují řešit mnoho jiných složitějších problémů s řetězci. Například:

- nejdelší společný podřetězec dvou i více řetězců
- nejdelší palindrom (P = reverse(P)) T
- nejdelší opakovaný podřetězec v T

Dalším krokem jsou suffixové tabulky (suffix arrays), které jsou méně náročné na paměť, především při veliké abecedě.

5: a
4: ha
3: iha
1: kniha
2: niha

kniha

Literatura

Dan Gusfield
Algorithms on Strings, Trees, and Sequences:
Computer Science and Computational Biology.
Cambridge University Press, 1997, 1st edition, ISBN 0521585198.

Hezký úvod
http://www.dogma.net/markn/articles/suffixt/suffixt.htm

Ukázka v JS, návaznost na další problémy
http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Tree/Suffix/

Implementace v ANSI C
http://cs.haifa.ac.il/~shlomo/suffix_tree/