Suffixový strom

def: suffixový strom pro řetězec S, $|S| = m$

- kořenový strom s m listy
- vnitřní vrcholy (mimo kořene) mají aspoň 2 potomky
- hrany jsou označeny neprázdnými podřetězci slova S
- pro každý vrchol platí: mají aspoň 2 potomky, liší se označení hran vedoucích do těchto potomků v prvním symbolu

- listy jsou označeny čísly $1 \dots m$. Spojení označení hran na cestě z kořene do listu i je právě $S[i..m]$.

Poznámka: pro některá slova suffixový strom podle předchozí definice nemusí existovat:

pokud je $S[j \dots m] = S[i..k]$ pro $S, |S| = m$ (tj. existuje suffix S, který je prefixem jiného suffixu S), pak nemáme pro $S[j \dots m]$ list!

značení:

řetězce indexujeme od 1

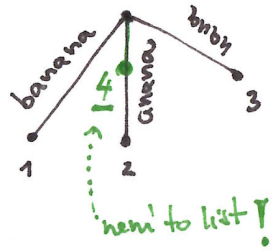
$S[i]$ ... symbol na indexu i, $1 \le i \le |S|$

$S[i..j]$ .. podřetězec, první symbol $S[i]$, poslední symbol $S[j]$

Př. S = banana
1 2 3 4 5 6

$S[4..6]$ = ana

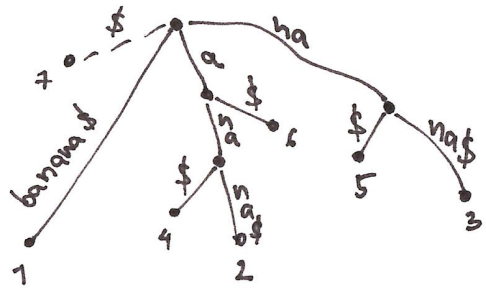
$S[2..4]$ = ana



Proto uvažujme symbol $\$$, který není v S a dáme ho na konec. Dostaneme tedy

S = banana$

a strom:



Využití sufixových stromů (nebo variant)

→ pro algoritmy pracující s řetězci

Př. Problém hledání podřetězce

vstup: S ... řetězec, ve kterém hledáme
P ... který hledáme

výstup: množina indexů, na kterých se P vyskytuje ve S

Existují algoritmy:

1) Předzpracují P v čase $O(|P|)$ a potom umějí vyhledávat v libovolném S v čase $O(S)$.

se sufixovým stromem ale

2) Předzpracují S v čase $O(|S|)$ → vytvoříme sufix. strom. S jeho použitím vyhledáváme pro libovolný P v čase $O(|P|)$ (+ počet výskytů)

Algoritmus:

předp. že: máme sufixový strom. Čteme P a podle označení hran se pohybujeme stromem od kořene.

1) zastavíme a nepřečteme celé P, protože neexistuje vhodná popsaná hrana → P se v S nevyskytuje

2) přečteme celé P, najdeme první následující vrchol w. (= buď skončíme ve vrcholu, pak nedělám nic, nebo skončíme uprostřed hrany a posuneme se až k jejímu vrcholu).

3) označení listů v podstromu s kořenem w jsou žádané indexy výskytů P.

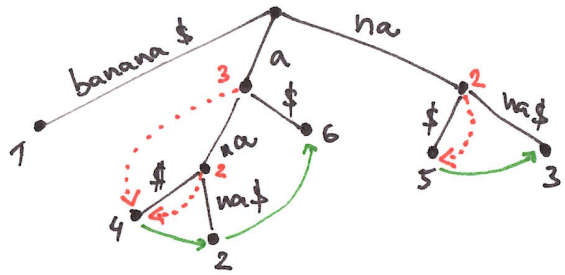
Složitost:

- aby bylo $O(|P|)$, musíme mít strom připraven:

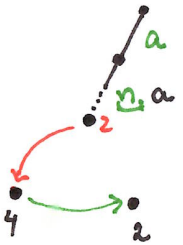
1) v každém vnitřním vrcholu si odkaz na první list v jeho podstromu a jejich počet,

2) listy jsou propojeny do seznamu.

Př. banana $



vyhledáváme an



⇒ výsledek: 4, 2

Korektnost: každý výskyt P je prefixem nějakého sufixu S.

Algoritmy pro sestavení sufixového stromu

$ už v řetězci nesmí nikde nastat, že suffix je prefix of suffix

Naivní algoritmus pro $S\$$, $|S| = m$

posloupnost suff. stromů $N_1, \dots N_m$, kde

N_i obsahuje suffixy $S[1..m], S[2..m] \dots S[i..m]$
(se přidaným $ na konci)

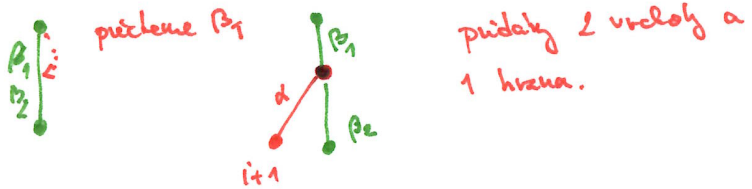
$N_1 =$ 

Postup pro $N_i \to N_{i+1}$:

- vkládáme $S[i+1..m]\$$
- čteme $S[i+1]\dots m]$ od kořene s pomocí označení hran tak daleko, jak to jde. zbude α.

Skončíme:

→ uprostřed hrany:
→ přidáme na toto místo nový vrchol, jako jeho potomka přidáme nový list s označením i+1, hranu do něj označíme α.

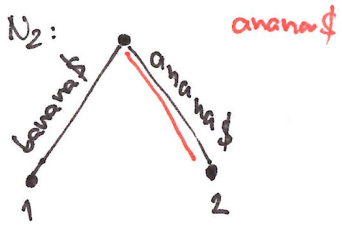


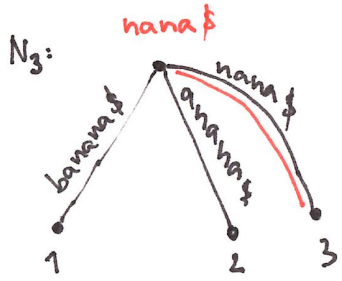
→ ve vrcholu:
→ přidáme tomuto vrcholu nového potomka s označením i+1 a hranu do něj označíme α

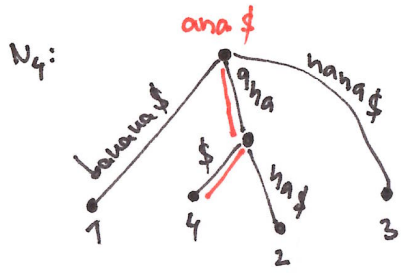
↦ čtení v přidaném kroku neskončí v listu: Žádný sufix S$ není prefixem jiného sufixu S$.

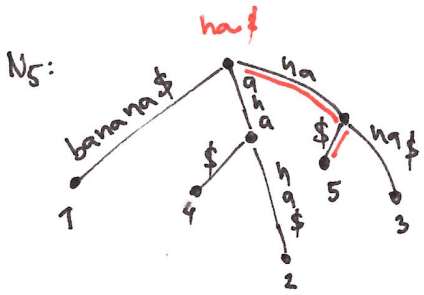
Př. banana$

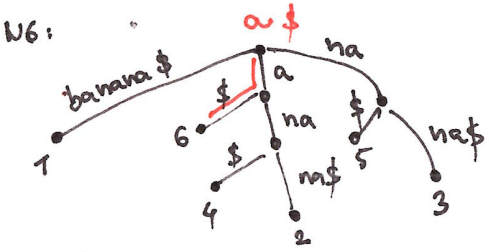
N_1: 

N_2: 

N_3: 

N_4: 

N_5: 

N_6: 

Složitost

$(m-1) + (m-2) + (m-3) + (\dots) + T \in O(m^2)$

počty porovnání symbolů = čtení sufixu

$\Theta(m^2)$

Algoritmy pracující v lineárním čase

- existují tři:
  - Weiner
  - McCreight
  - Ukkonen ← Ten si řekneme

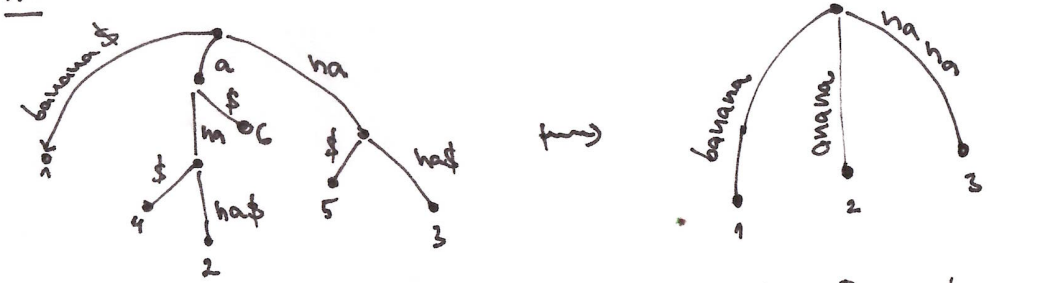
Ukkonenův algoritmus

Ukážeme neefektivní formu, kterou optimalizujeme až na $O(m)$. ($|S| = m$).

Implicitní suffixový strom pro S ← teď není $ součástí S

Suffixový strom pro S$
- odstraníme $ z označení hran
- odstraníme hrany bez označení (+ izolované uzly/vrcholy)
- odstraníme vnitřní uzly/vrcholy s méně než dvěma potomky

Př.



suffix t. pro banana

⟹

impl. s.t. pro banana

Pozn:
- impl. s.t. obsahuje všechny suffixy (= cestou z kořene k pomocí označení hran přečtené), které nekončí v listu.

- pokud se poslední symbol S nevyskytuje na jiném místě S, je implicitní suffix t. celý suffix t.
  (tj. no suffix is a prefix of another suffix)

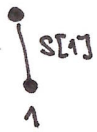
Ozn:

I_j .... implicitní suffix stromu pro $S[1 \dots j]$

Ukkonenův algoritmus konstruuje posloupnost

$$I_1 \xrightarrow{\text{fáze 2}} I_2 \xrightarrow{\text{fáze 3}} I_3 \longrightarrow \dots \xrightarrow{\text{fáze m}} I_m$$

Strom I_1 je triviální:



Ve fázi $(i+1)$ se konstruuje strom I_{i+1} z I_i pomocí $i+1$ operací expanze. Tj. pro $j = 1 \dots i+1$ zajistíme, aby byl suffix $S[j \dots i+1]$ ve stromu I_{i+1}.

Tj. algoritmus je dvojitý cyklus. Vnější cyklus je přes fáze, vnitřní cyklus je přes expanze.

Pravidla pro expanzi

$I_i \longrightarrow I_{i+1}$ $\exists\, 1 \le j \le i+1$, zajistíme, že $S[j \dots i+1]$ je ve stromu

označ. $\beta = S[j..i]$, ($S[i+1, i] = \varepsilon$!)

1. Najdeme konec β ve stromu (β patří do I_i !)

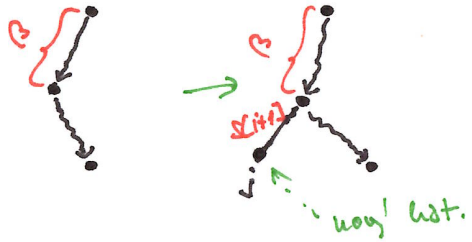
   z tohoto místa provedeme rozšíření o symbol $S[i+1]$

Pravidlo 1: β končí v listu

→ přidáme $S[i+1]$ k označení poslední hrany na cestě

Pravidlo 2: z konce β nepokračuje cesta symbolem $S[i+1]$, (ale nejsme v listu).

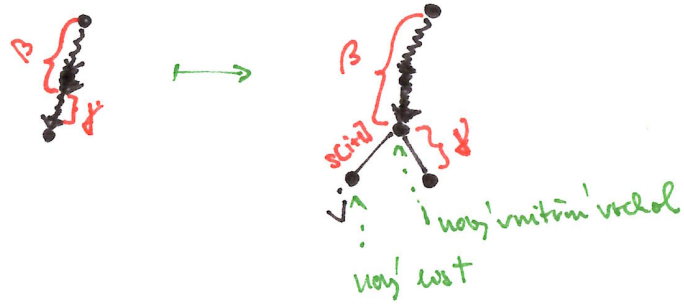
→ konec je vrchol. Přidáme nového potomka a hranu s označením $S[i+1]$. Potomek je označen j



[TADY PŘIDÁVÁME LIST, NEPŘIDÁVÁME VNITŘNÍ VRCHOL]

→ konec není vrchol.

Přidáme nový vnitřní vrchol na místo konce β a upravíme označení hran. Potom postupujeme jako v předchozím bodě.



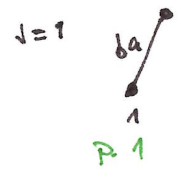
Pravidlo 3: z konce β pokračuje cesta symbolem $S[i+1]$

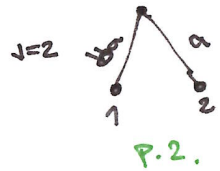
→ nedělá se nic, $S[j \dots i+1]$ už ve stromu je

Př. banana

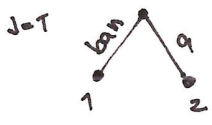
I_1:

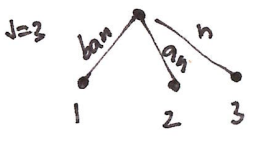
[ba]
$I_1 \to I_2$





[ban]
$I_2 \to I_3$





Optimalizace: krok 7 (Rychlé hledání konců suffixů)

def: v... vrchol

- path-label(v) ... řetězec, který dostaneme spojením označení hran na cestě z kořene do v

- v... vnitřní ~~uzel~~ vrchol, path-label(v) = xα, x symbol, α řetězec (i prázdný)

  vrchol $\bar{s}(v)$ ... vrchol s path-label($\bar{s}(v)$) = α. Pointer z v do $\bar{s}(v)$ je suffix-link.

Pozn: Pokud α = ε, ... $\bar{s}(v)$ kořen.

(Teď ani nevíme, jestli $\bar{s}(v)$ existuje, ... Předtím od v k $\bar{s}(v)$ ... při běžném operace)

Lemma: ~~Předp. že jsme ve fázi algoritmu a~~

Pokud během expanze ~~rozšíření~~ j (v nějaké fázi algoritmu) vytvoříme vnitřní vrchol v, path-label(v) = xα, potom:

→ buď existuje vnitřní vrchol $\bar{s}(v)$, path-label($\bar{s}(v)$) = α (a)

↘ nebo $\bar{s}(v)$ bude vytvořen v ~~rozšíření~~ expanzi j+1 téže fáze. (b)

Důkaz:

Předp. že jsme ve fázi i+1 (tj. $I_i \to I_{i+1}$)

Vnitřní vrchol v vytvořen pravidlem 2, tj. že strom ~~nerozšiřitelně~~ už obsahoval xαc pro $c \neq S[i+1]$.

(ostatní případy v téže fázi jsem dělal)

Současně xαc patřil do I_i a tedy $\overset{\alpha c}{\text{~~I~~}}_i$ a proto v rozšíření j+1 obsahuje abd. strom slovo αc. Tedy máme 2 možnosti:

→ α pokračuje pouze c.
Potom pravidlo 2 vytvoří nový vnitřní vrchol. Tento vrchol má path-label = α a tj. tedy $\bar{s}(v)$ (b)

→ α pokračuje i jiným symbolem.
Potom ale musí α končit ve vrcholu s path-label(v) = α, který je tedy $\bar{s}(v)$. (a).

Důsledek: Na konci fáze mají všechny ~~vnitřní~~ vnitřní vrcholy v suffix-link do $\bar{s}(v)$.

Důkaz:

Plyne přímo z předchozího lemmatu a toho, že poslední rozšíření během fáze nemůže vytvořit nový vnitřní vrchol. (protože S[i+1, i] = ε). (tam se dělá jinak při odbočce)

Použití suffix-linků

(implicitní stromy)
$I_i \rightarrow I_{i+1}$

zafixujeme jednu fázi, řekněme $i+1$.

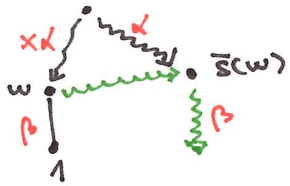
→ V první ~~externí~~ expanzi potřebujeme najít konec $S[1..i]$.
To je ovšem list označený 1. Tj. konec najdeme v konstantním čase [např. si udržujeme v kořeni pointer na list 1].

→ V druhé expanzi jsme v listu 1. Přejdeme do rodiče w tohoto listu. ~~označíme jej~~ Předp. že hrana z w do listu 1 je označena řetězcem β.

Máme 2 možnosti

1) w je kořen → najdeme konec $S[2..i]$ stejně jako v naivním algoritmu.

2) w není kořen
potom $S[1..i] = x\alpha\beta$
a $S[2..i] = \alpha\beta$



OBR ⊛

Podle předpokladu ~~hashky~~ důsledku existuje $\bar{s}(w)$. Přes tento suffix-link půjdeme do ~~uz~~ vrcholu $\bar{s}(w)$.
Z $\bar{s}(w)$ pokračujeme pomocí β jako v naivním algoritmu a najdeme konec $S[2..i]$

→ V obecné expanzi (pro $j>2$) postupujeme analogicky jako v druhé expanzi, s malým rozdílem:

Pokud $S[j-1..i]$ končí ve vnitřním vrcholu z, a tento vrchol má suffix-link, nevracíme se zpět k w jako v druhé expanzi, ale použijeme suffix-link vrcholu z.

Vrchol z ~~nemusí mít~~ nemusí mít suffix-link, pokud byl vytvořen v $j-1$ expanzi pravidlem 2.

Vytváření nových suffix-linků

Pokud v expanzi $j-1$ vytvoříme pravidlem nový vnitřní vrchol z, a potom v expanzi j nalezneme ~~vrchol~~ (nebo vytvoříme) vrchol w s path-label = $S[j..i]$, zapojíme suffix-link z vrcholu z do vrcholu w.

Rychlý pohyb stromem (fixed 2)

situace: - jsme ve vrcholu v, (tj. je to kořen našeho podstromu)
- víme, že z v lze pokračovat (dolů k listům) pomocí účězce β

(✱ odpovídá situaci, kdy na předešlé stránce hledáme konec β z vrcholu $\bar{s}(w)$, viz obr ✱)

Chceme zajistit, aby složitost nalezení konce β závisela na počtu vrcholů, přes které musíme projít, nikoliv na $|\beta|$.

→ na hranách si pamatujeme délku zpětných označení

Jsme-li ve vrcholu w, potom podle stovnání prvních symbolů označení hran vedoucích z w do potomků poznáme, po které hraně se vydat. Označená vybraná hrana (w,z).

Je-li délka označení této hrany g a délka β je h (tj. $|\beta|=h$) pak

- pokud $g>h$: konec β je na h-tém symbolu označení hrany
- pokud $g<h$: konec β nalezneme popsaným postupem pro vrchol z a příslušný suffix β délky $h-g$
- pokud $g=h$: konec β je vrchol z.

Předchozí algoritmus tedy zajistí, že složitost nalezení konce β je lin. závislá na počtu vrcholů, které musíme ~~projít~~. Navštívit.

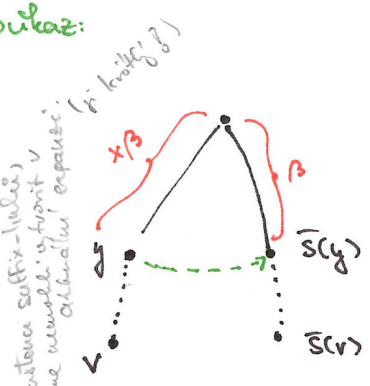
Lemma: $(v, \bar{s}(v))$ ... suffix-link

V čase, kdy přidáváme přes suffix tuto link platí $\text{depth}(v) \le \text{depth}(\bar{s}(v)) + 1$.

[depth(x) je hloubka vrcholu x]

v může být o 1 hlouběji

Důkaz:



Předp. existuje vnitřní vrchol y, který je předchůdcem v a máme path-label$(y) = x\beta$.

Potom má y suffix-link do $\bar{s}(y)$ a path-label $\bar{s}(y)$ je β.

Ale $x\beta$ je prefix path-label(y), $[x \in \Sigma, \beta \in \Sigma^*]$

tj. β je prefix path-label$(\bar{s}(v))$ a tedy $\bar{s}(y)$ je předek $\bar{s}(v)$.

Tj. počet předků v je nejvýše počet předků $\bar{s}(v)$, zvětšený o 1, pokud má v předka s path-labelem $x \in \Sigma$ (= jedním symbolem) a $\beta = \varepsilon$, suffix-link tohoto předka je kořen.

□

Lemma: S použitím suffix-linků zabere jedna fáze čas $O(m)$. // m je délka řetězce

Důkaz:

Mimo počtu operací průchodu od vrcholu k jeho potomku je vše v čase $O(m)$.

Kolik je průchodů k potomku? [= kolikrát můžeme snížit hloubku akt. vrcholu]

Alg. si udržuje akt. vrchol [Mění se průchody po suffix linkách a hledáním konců řetězců].

Během jedné expanze:

→ snížíme hloubku akt. vrcholu nejvýše o 1
(když hledáme vrchol, ze kterého půjdeme přes suffix-link)

→ průchod přes suffix-link sníží hloubku nejvýše o 1 (viz předchozí lemma.)

Hloubka vrcholu je z intervalu $[0, m]$ a proto hloubku akt. vrcholu během algoritmu (jedné fáze) můžeme zvýšit méně než $3m$ krát.

Důsledek: Ukonečný algoritmus se suffix-linky má složitost $O(m^2)$.

Další zrychlení

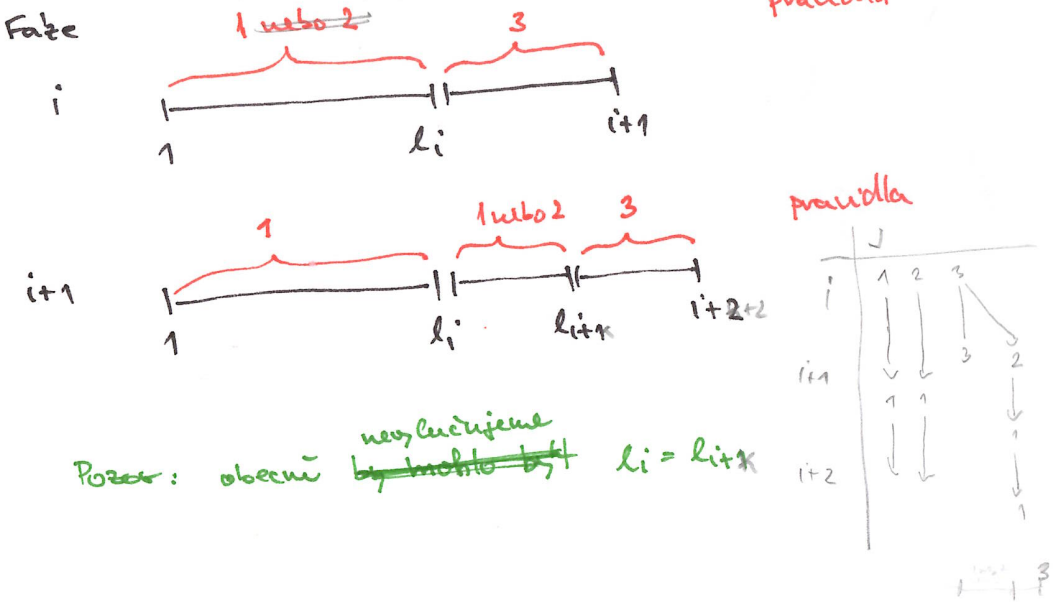
A) pokud ve fázi i během expanze j použijeme pravidlo 3, ve sledujících expanzích $j+1, \dots, i+1$ se také použije pravidlo 3

(suffixy určitě jsou) ⇒ I_i byl implicitně v stromu

B) je-li pravidlem 2 vytvořen nový ~~uzel~~ list, zůstává tento vrchol listem až do konce algoritmu. (Pokud byl vytvořen v expanzi j, je v budoucích fázích během expanze j použito pravidlo 1.)

obecněji, pokud je v expanzi j fáze i použito 1, je použito ve expanzi j fáze $i+1, i+2, \dots$

TEDY:



Pozor: obecně ~~by mohlo být~~ nevyužíváme $l_i = l_{i+1}$

Použití pozorování:

→ označení hran děláme pomocí dvojice indexů $[i,j]$, to odpovídá úřezu $S[i,j]$

→ zavedeme globální proměnnou e, hrany vedoucí do listů mají označení $[i,e]$ (pro nějaký index i)

[Trik je v tom, že můžeme provést všechny expanze pravidla 1 pomocí inkrementace e o 1]

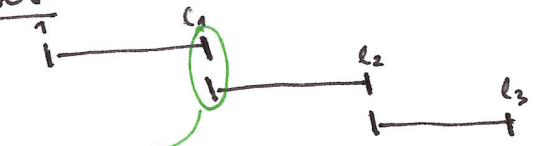
Algoritmus pro Fázi $i+1$

1. inkrementujeme e o 1 — [tím se vyřídí pravidlo 1]

2. počítáme ~~extenze~~ expanze l_i až l^*, kde l^* je první expanze používající pravidlo 3, nebo poslední expanze v této Fázi ($i+2$) — [dělat pravidla 1 nebo 2]

3. nastavíme l_{i+1} na l^*-1.

Obrázek:



během sdílené expanze Pze projít přes suffix-link.

Věta:

Ukončení algoritmus s vylepšeními má složitost $O(n)$.

Důkaz:

Vynechán.

□