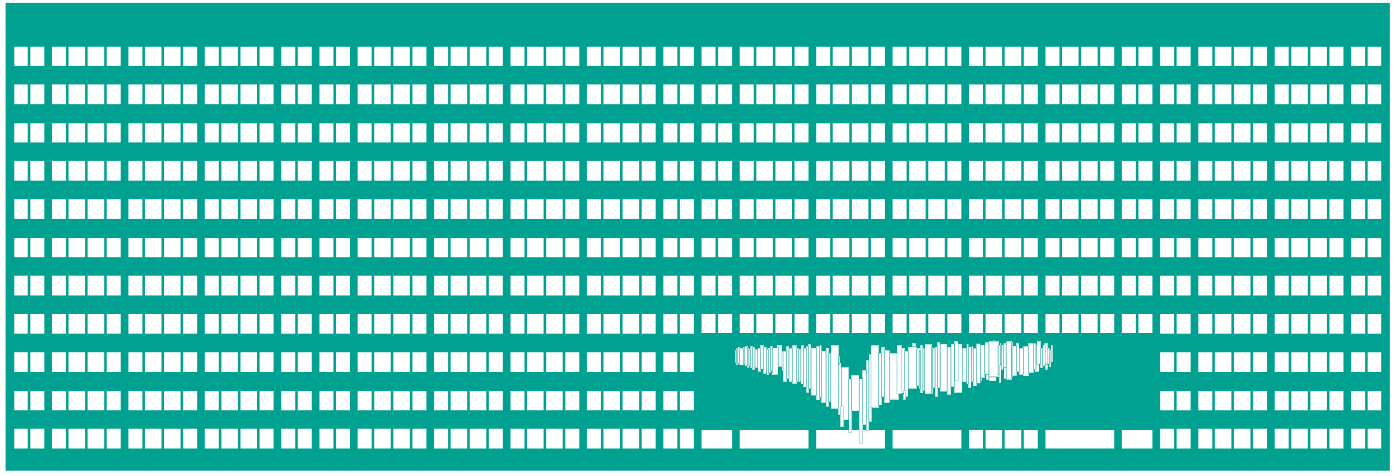
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

Algoritmy pro Bioinformatiku

Indexace Genomu

Michal Vašinek

VŠB – Technická univerzita Ostrava

FEI/EA404

michal.vasinek@vsb.cz

2. října 2019

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Přehled přednášky

- Trie
- Suffixový strom
- Suffixové pole

Vyhledávání řetězců

Mějme text T o délce $n = |T|$ a řetezec P o délce $m = |P|$. Základní úlohy:

- Find - nalezněte pozice všech výskytů P v T.
- Exists - vyskytuje se P v T?
- Occ - Kolikrát se vyskytuje P v T?

Řešení standardními algoritmy pro prohledávání textů:

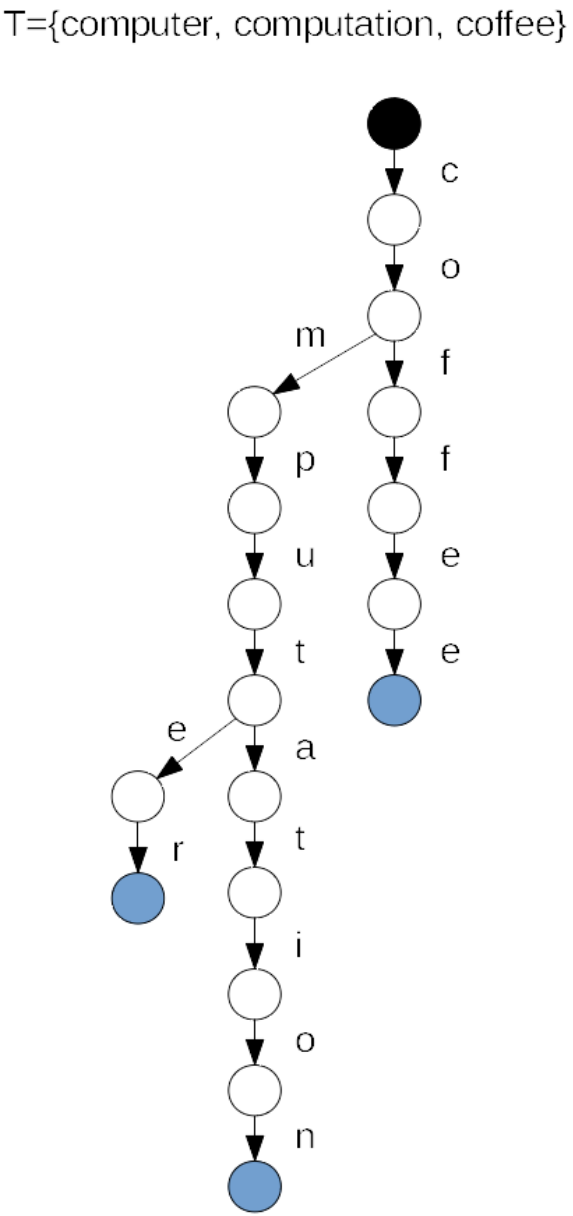
- Hrubou silou $O(nm)$.
- Boyer-Moore algoritmus - předzpracování vzorku $O(n + m)$.

Lidský genom $|G| = 3.2$ mld symbolů. FastQ soubor 1 mil sekvencí. Použitím Boyer-Moorova algoritmu $10^6|G|$ porovnání symbolů.

Indexace Textu Trie

Trie

- Trie (ang. výslovnost "try") je stromová struktura reprezentující množinu řetězců:
  - Každá hrana je označena symbolem $c \in \Sigma$.
  - Pro libovolný uzel stromu platí, že uzel má nanejvýš jednu hranu označenou symbolem c.
  - Vložený řetězec jsme schopni odečíst průchodem od kořene stromu k listu.



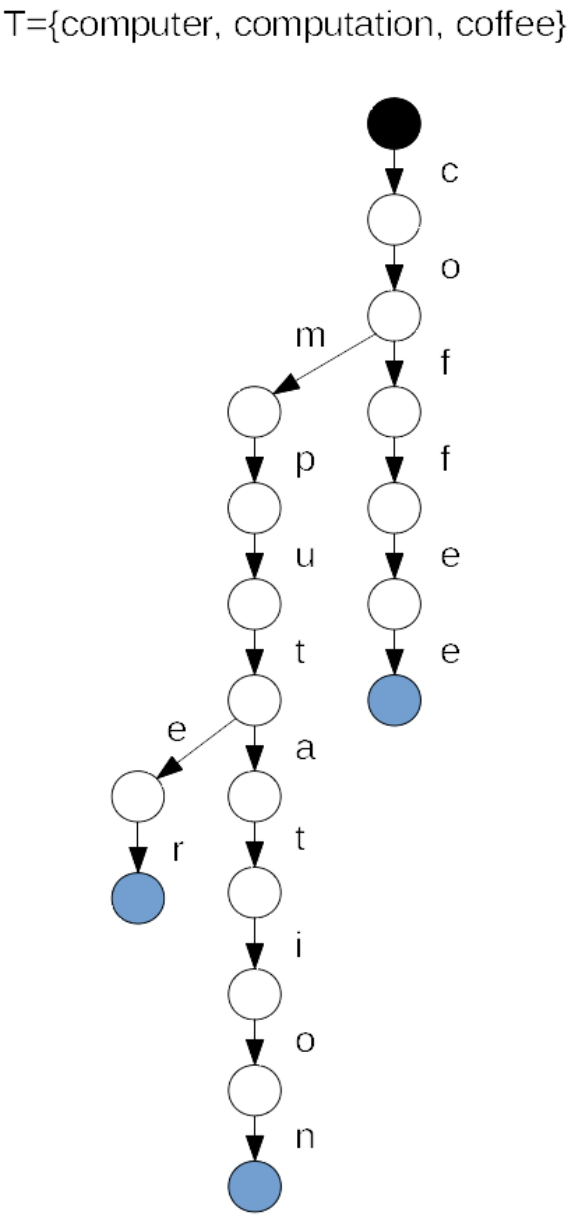
Trie

Použití mimo bioinformatiku:

- Nahrazení hashovací tabulky => vyhledávání klíčů.
- Reprezentace slovníku => doplňování slov v mobilních klávesnicích.

Bioinformatika:

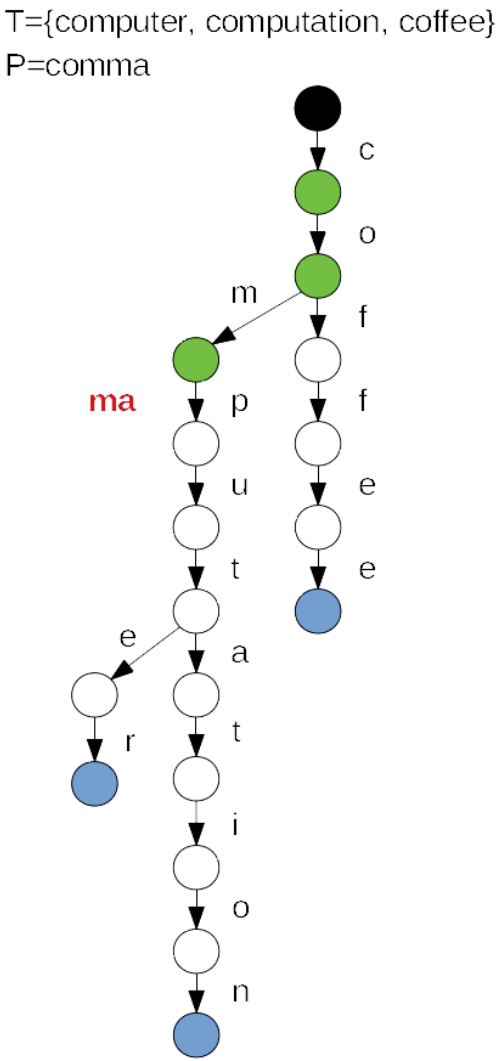
- Indexace suffixů.
- Indexace k-merů.



Trie - Operace s řetězci

Ověření existence řetězce:

- Začneme v kořenovém uzlu.
- V potomcích aktuálního uzlu hledáme aktuálně zpracovávaný symbol.
- Pokud z aktuálního uzlu nevede hrana s požadovaným symbolem, řetězec se v trie nenachází.
- Pokud jsme pro každý symbol v hledaném řetězci nalezli cestu skrz trie, pak se řetězec v trie vyskytuje.

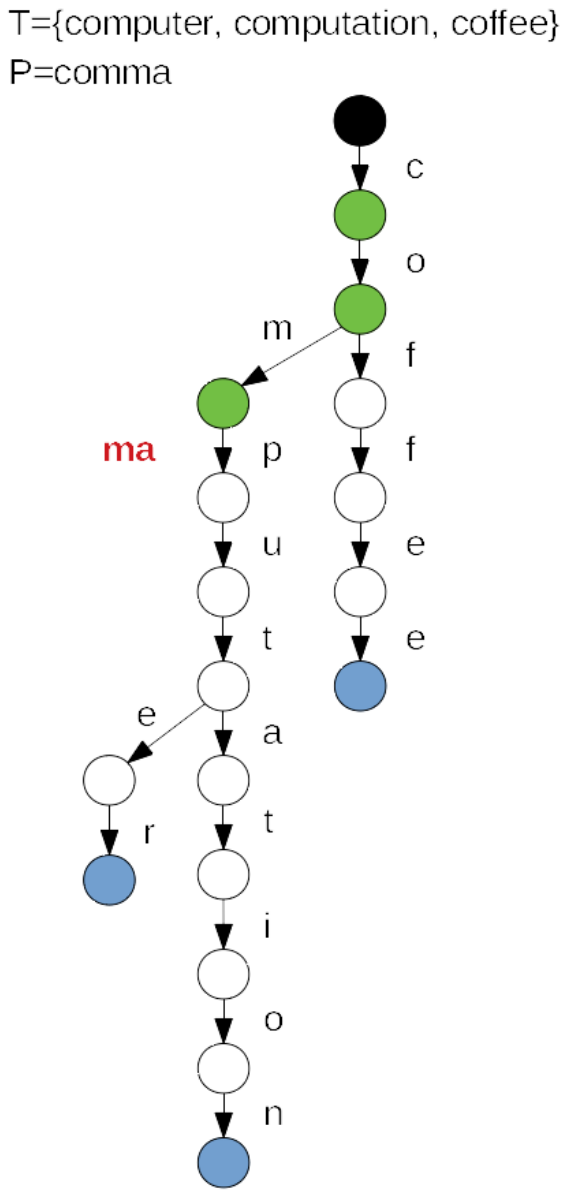


Trie

- Při ověření existence vzoru P o délce $|P| = m$ procházíme nejvýše m hran.
- Pokud do trie vložíme w řetězců o celkové délce W, pak počet hran v trie je nejvýše W.

Implementace:

- Hashovací tabulka => předpokládáme složitost $O(1)$ pro nalezení symbolu v uzlu, pak vyhledání řetězce má složitost $O(m)$.
- Setřizené pole => binárním vyhledáváním nalezneme symbol v čase $\log \sigma$, tudíž řetězec nalezneme v čase $O(m \log \sigma)$.



Trie - indexace suffixů

- Dosud jsme do trie vkládali oddělené řetězce.
- Pro vyhledávání v dlouhém textu T vkládáme do trie všechny suffixy textu T.

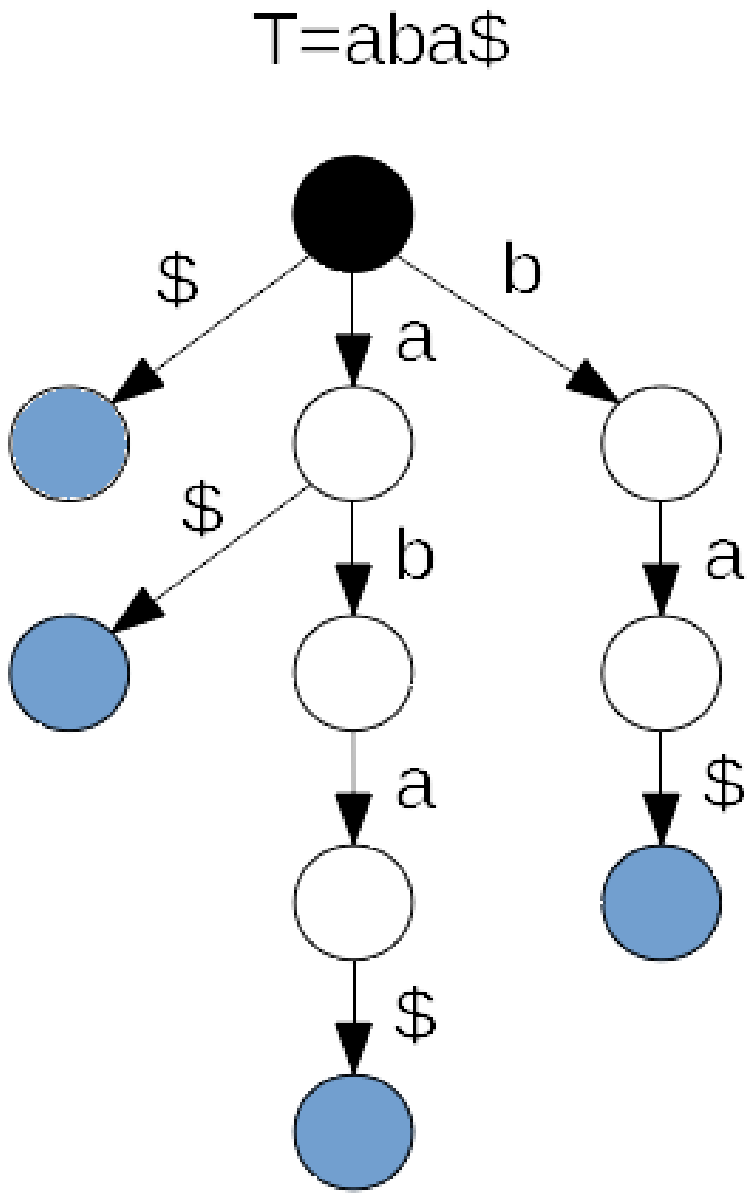
T:	c	o	f	f	e	e	$
	c	o	f	f	e	e	$
		o	f	f	e	e	$
			f	f	e	e	$
				f	e	e	$
					e	e	$
						e	$
							$

Celkem mají všechny suffixy $n(n+1)/2$ symbolů. Prostorová složitost $O(n^2)$.

Každý podřetězec T je prefixem nějakého suffixu.

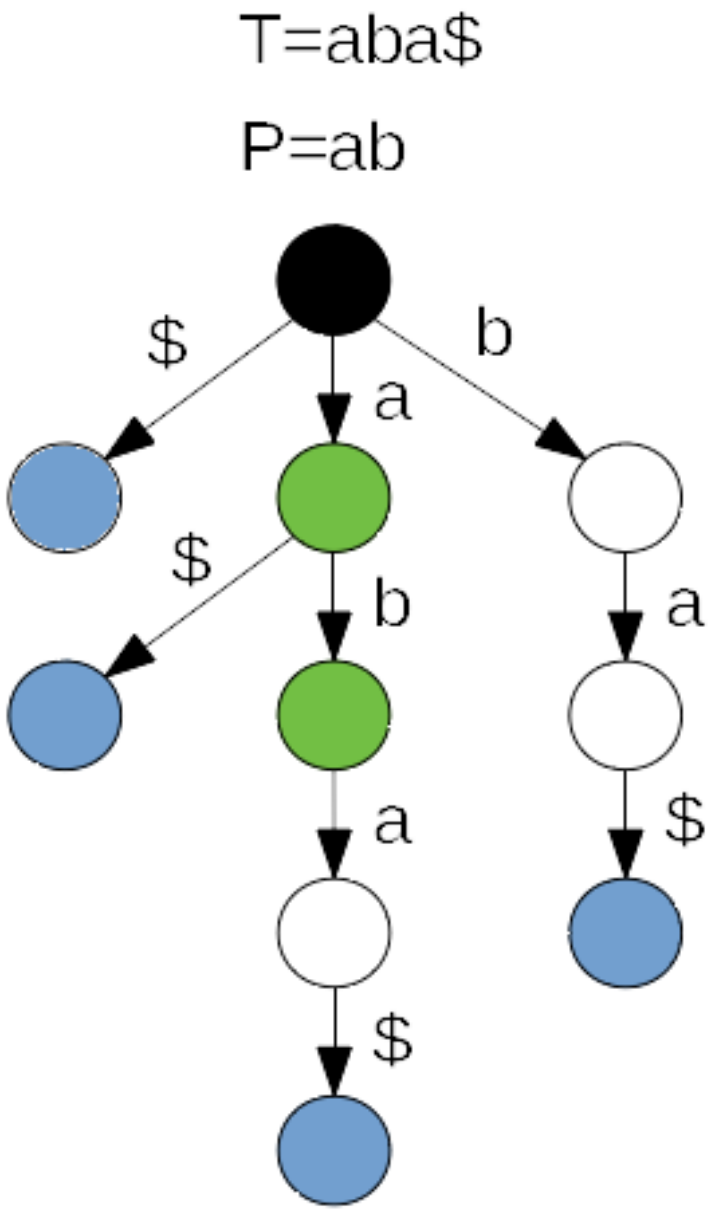
Trie - sestavení

- K textu T přidáváme na konec speciální terminální symbol \$, menší, než všechny ostatní symboly.
- Speciální symbol \$ nám zajistí dvě důležité vlastnosti:
  - Kratší řetězce začínající stejným prefixem jsou při setřízení vepředu. např "e"a "ee"ve slově coffee.
  - Žádný suffix T nemůže být zároveň prefixem jiného suffixu.
- Trie sestavujeme od nejdelšího po nejkratší suffix.



Trie - Find

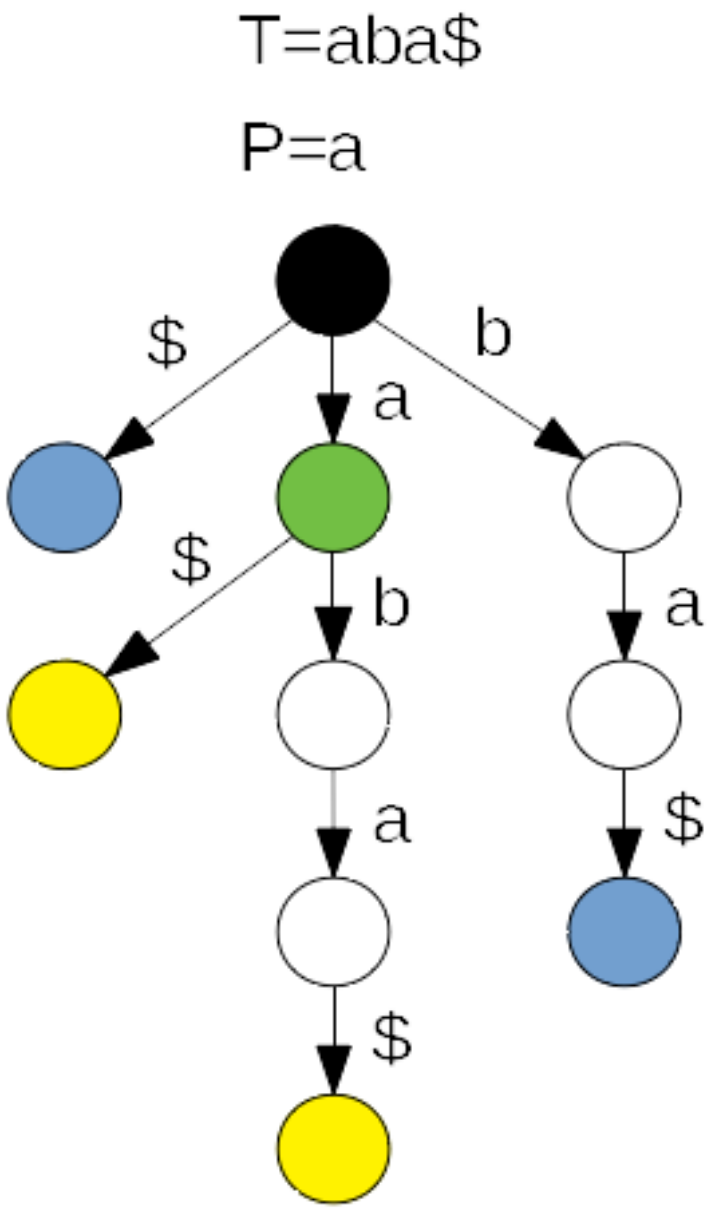
- Každá cesta od kořene k listu reprezentuje jeden suffix.
- Ověření výskytu řetězce P v T provedeme opět průchodem od kořene.



Trie - Occ

Výpočet počtu výskytů:

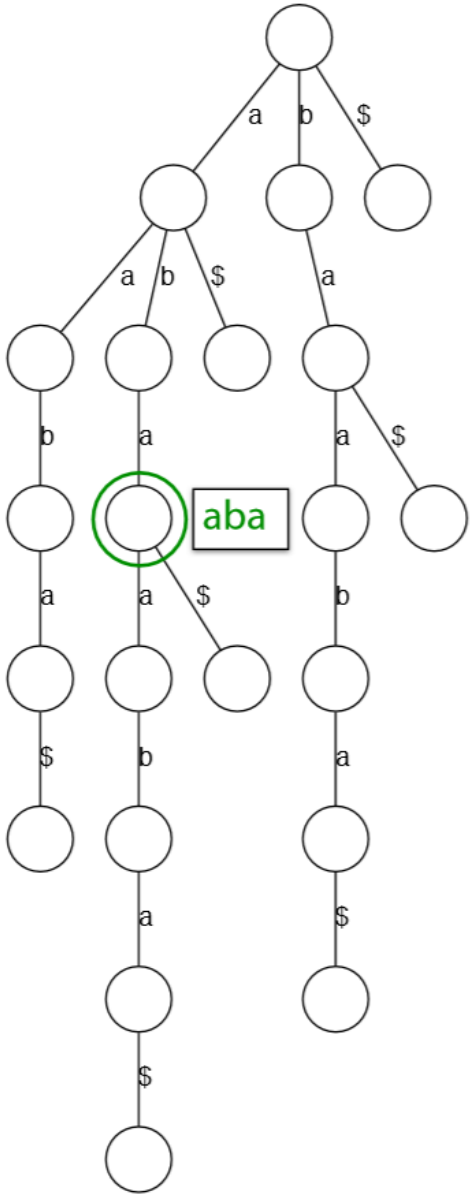
- Postupujeme od kořene k uzlům.
- Pokud se řetězec P v trie nevyskytuje, pak je počet roven 0, jinak je počet výskytů P dán počtem listových uzlů odpovídajících podstromu začínajícímu v uzlu, do kterého jsme dorazili průchodem přes symboly z P.
- Průchod do hloubky.



Trie - nejdelší opakující se podřetězec

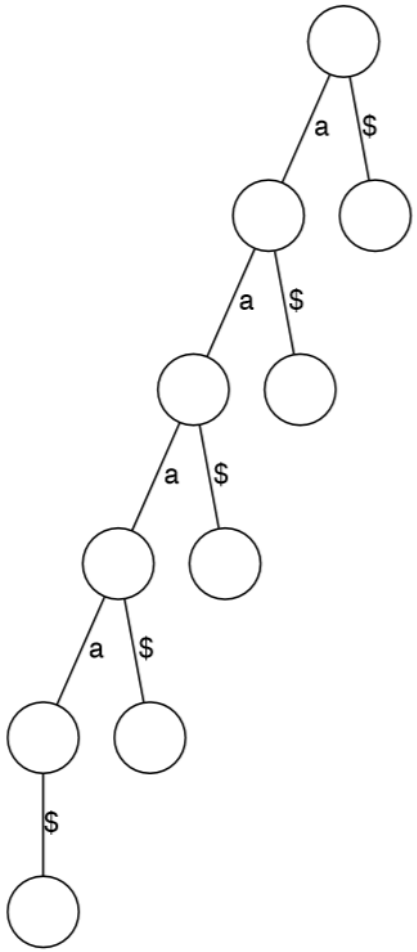
Výpočet nejdelšího opakujícího se podřetězce:

- Najděte uzel s alespoň dvěma potomky, ke kterému je nejdelší cesta od vrcholu.



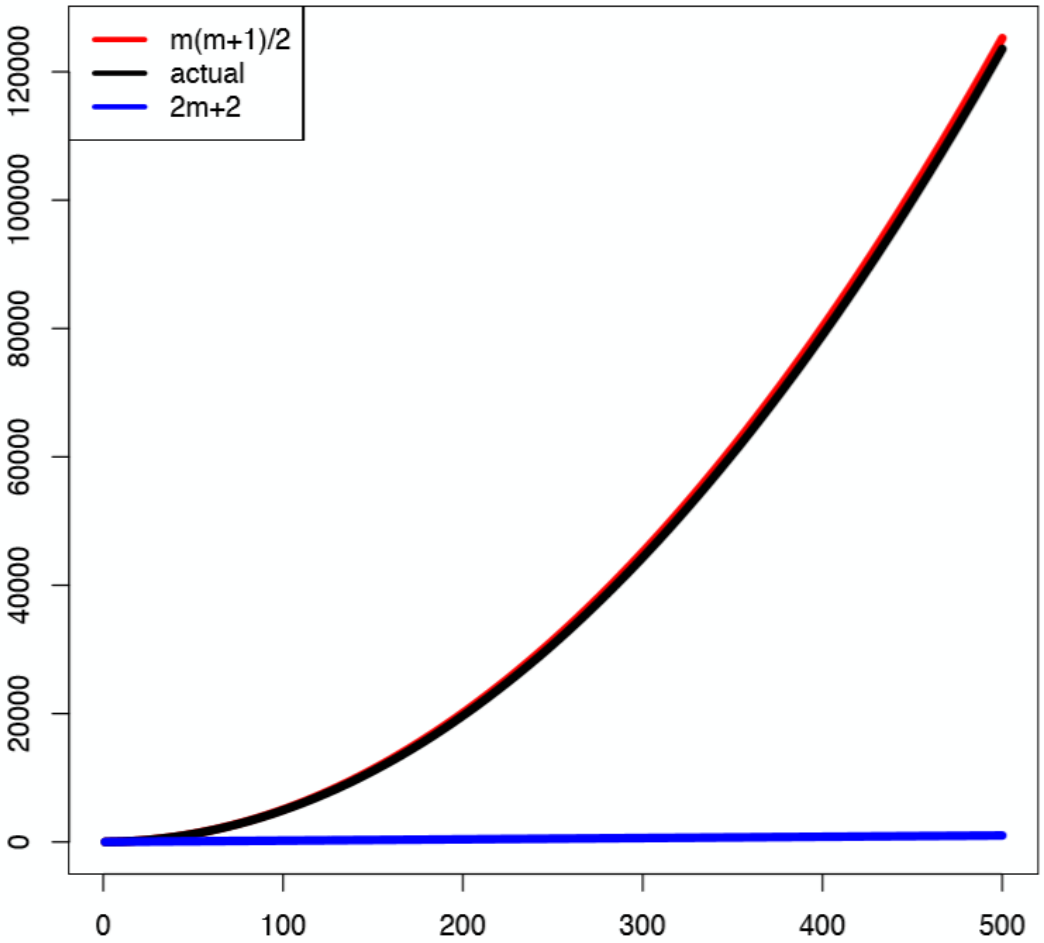
Trie - analýza

- Uvažujme řetězec $T = a^n$, tedy n-opakování symbolu a.
- Počet uzlů:
  - 1 kořenový
  - n uzlů se vstupující hranou "a".
  - $n + 1$ uzlů se vstupující hranou "$".
  - Celkem $2n + 2$ uzlů.
- V nejhorším případě $n(n + 1)/2 = O(n^2)$ uzlů.



Trie - prostorová složitost

Lambda fág genom. Počet uzlů roste přibližně rychlostí nejhoršího případu.



Suffixový strom

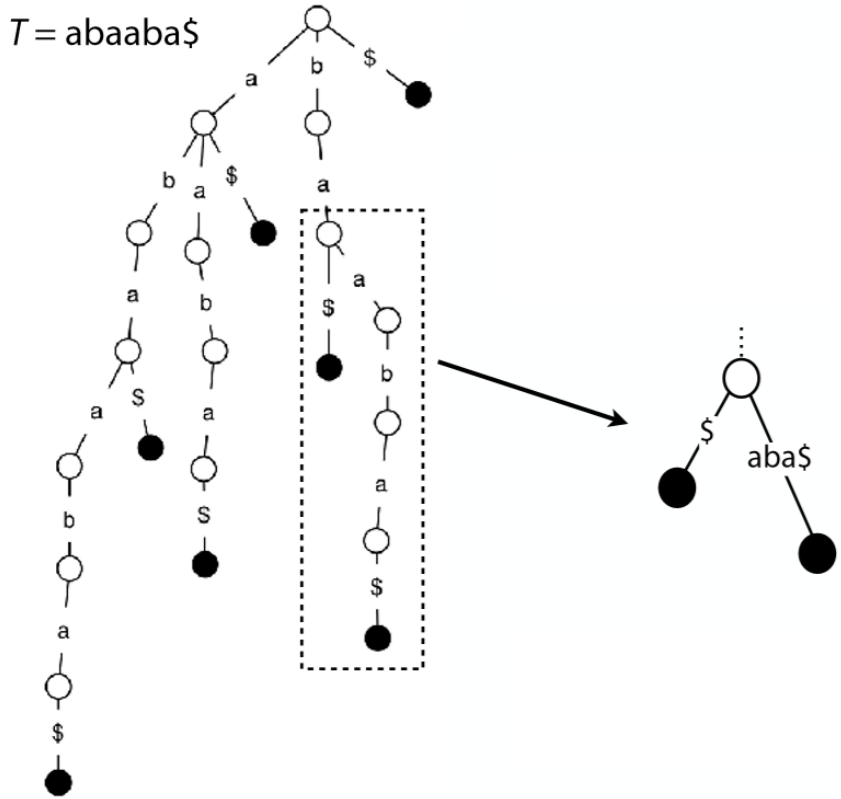
Suffixový strom

- Víme, že ověření zda řetězec P je obsažen v T lze pomocí trie vyřešit v $O(m)$.
- Prostorová složitost nicméně $O(n^2)$. Lidský genom $|G| = 3.2 \times 10^9$ => $|G|^2$.
- Cíl: zmenšit paměťové nároky => suffixový strom.

Myšlenka 1

Spojit nevětvící se cesty do jedné cesty nadepsané spojením symbolů v dané cestě.

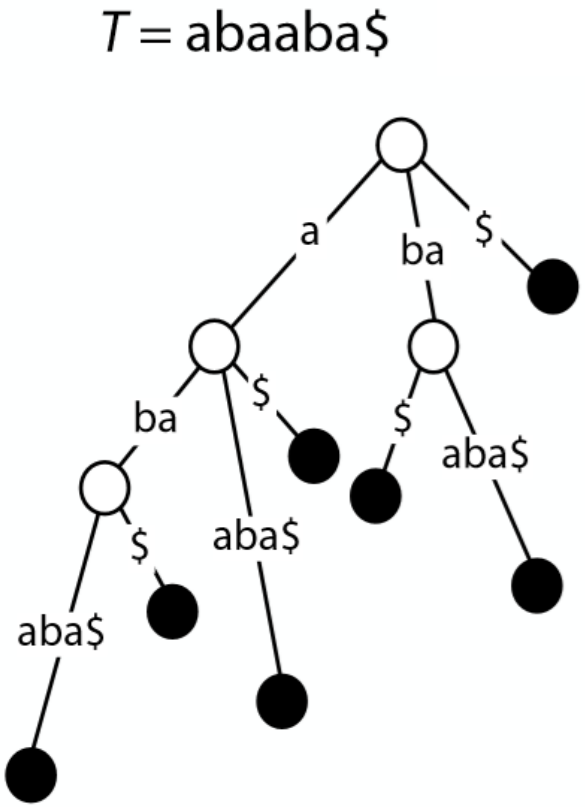
- Redukce počtu uzlů.
- Každý vnitřní uzel má alespoň dva potomky.
- Počet symbolů popisujících hrany zůstává stejný.



Myšlenka 1

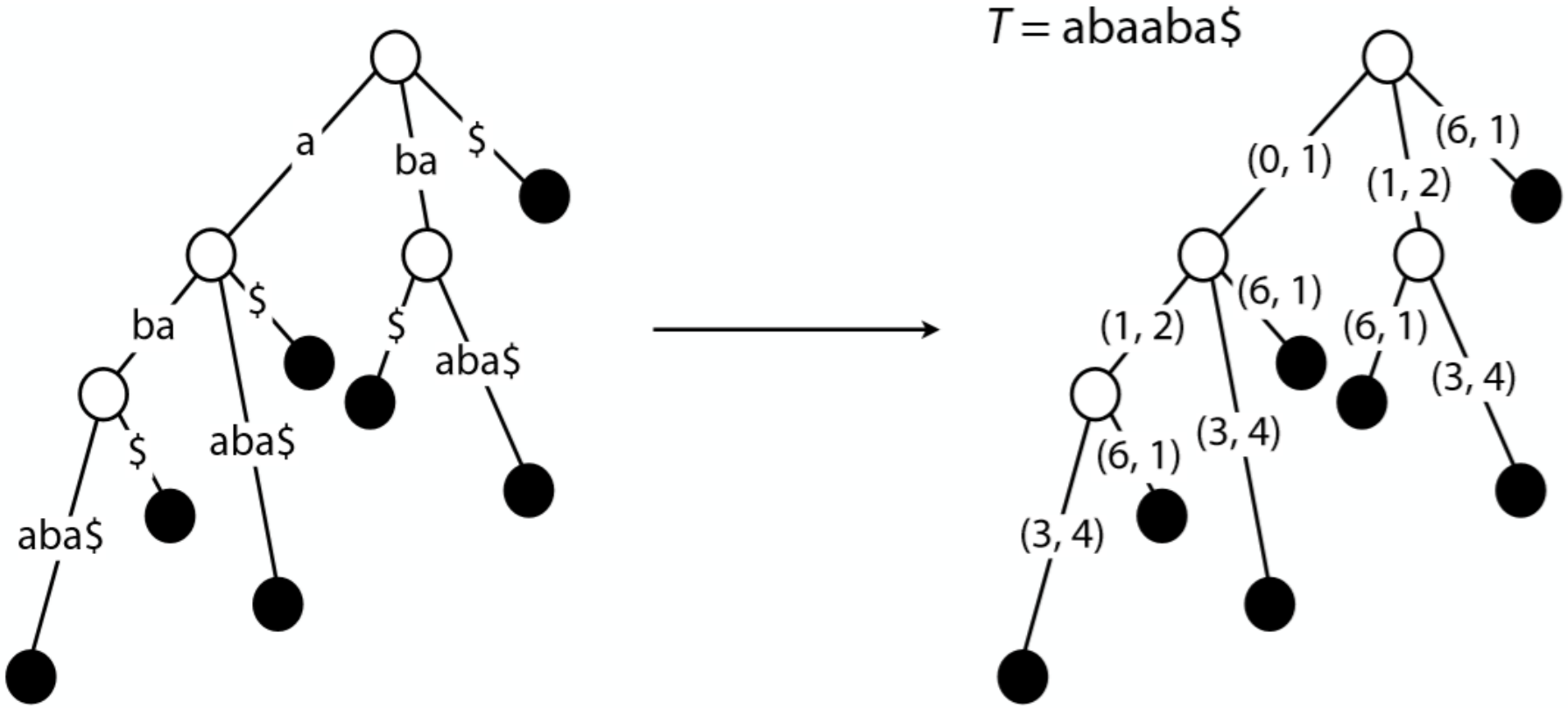
Spojit nevětvící se cesty do jedné cesty nadepsané spojením symbolů v dané cestě.

- Počet listových uzlů n.
- Nelistové uzly nejvýše $n - 1$.
- Pamětová složitost $2n - 1 = O(n)$
- Počet symbolů roste stále s $O(n^2)$.



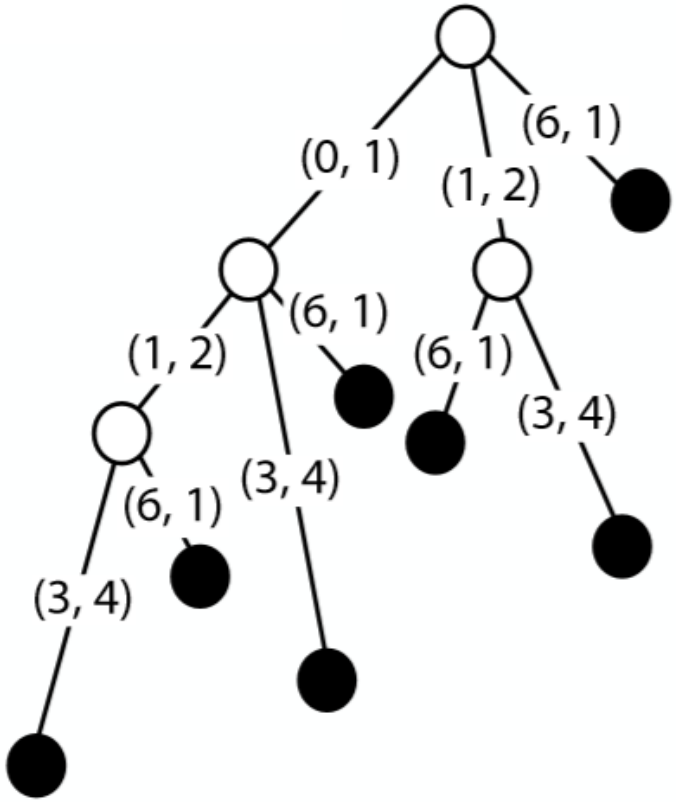
Myšlenka 2

Uložit si společně se stromovou strukturou samotný text T a převést popisky hran na dvojice (offset,délka) v T.



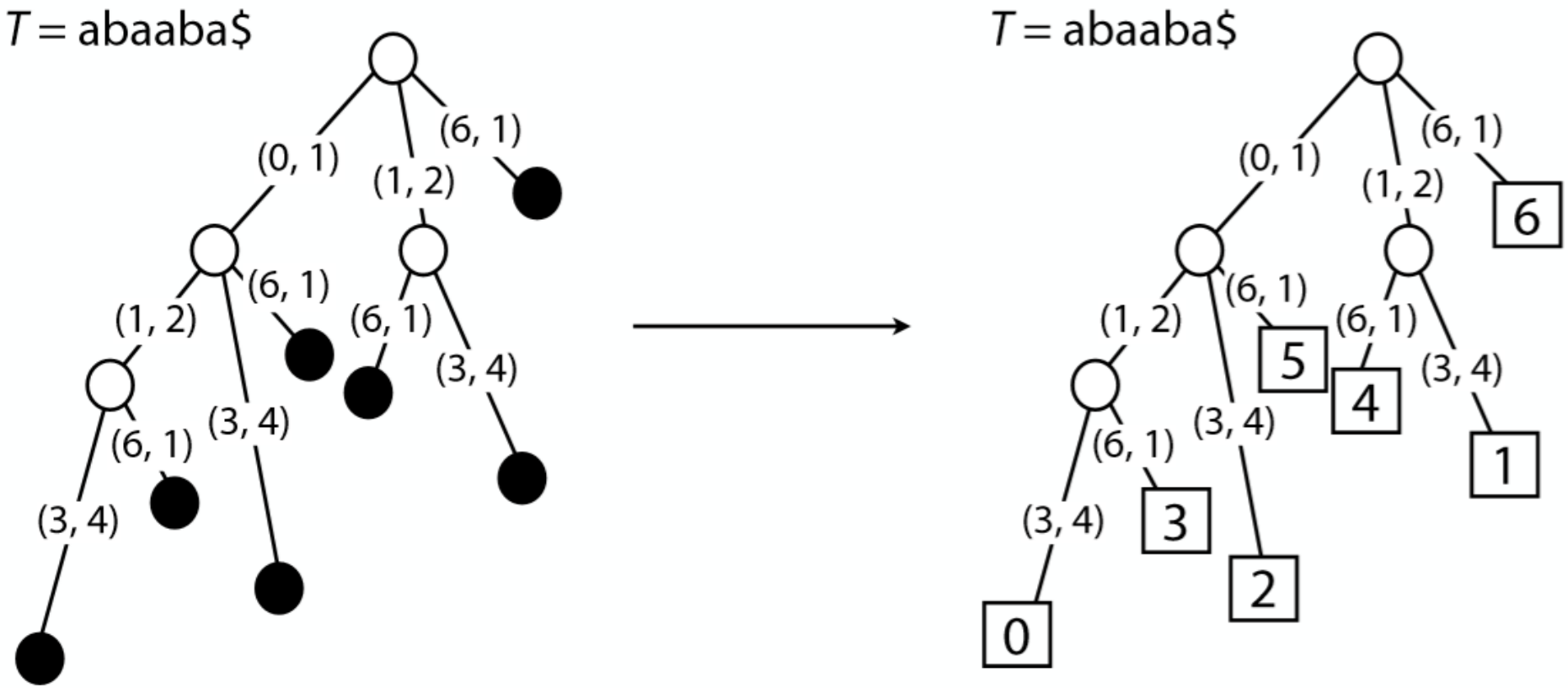
Suffixový strom

- Offset a délka nabývají maximální hodnoty $|T| = n$ a lze je uložit pomocí $2\lceil \log n \rceil$ bitů.
- Celková prostorová složitost: $2n - 1$ uzlů vede k uchování $2n - 2$ hran. Počet dvojic (offset,délka) => $(2n - 2)2 \log n = O(n \log n)$.

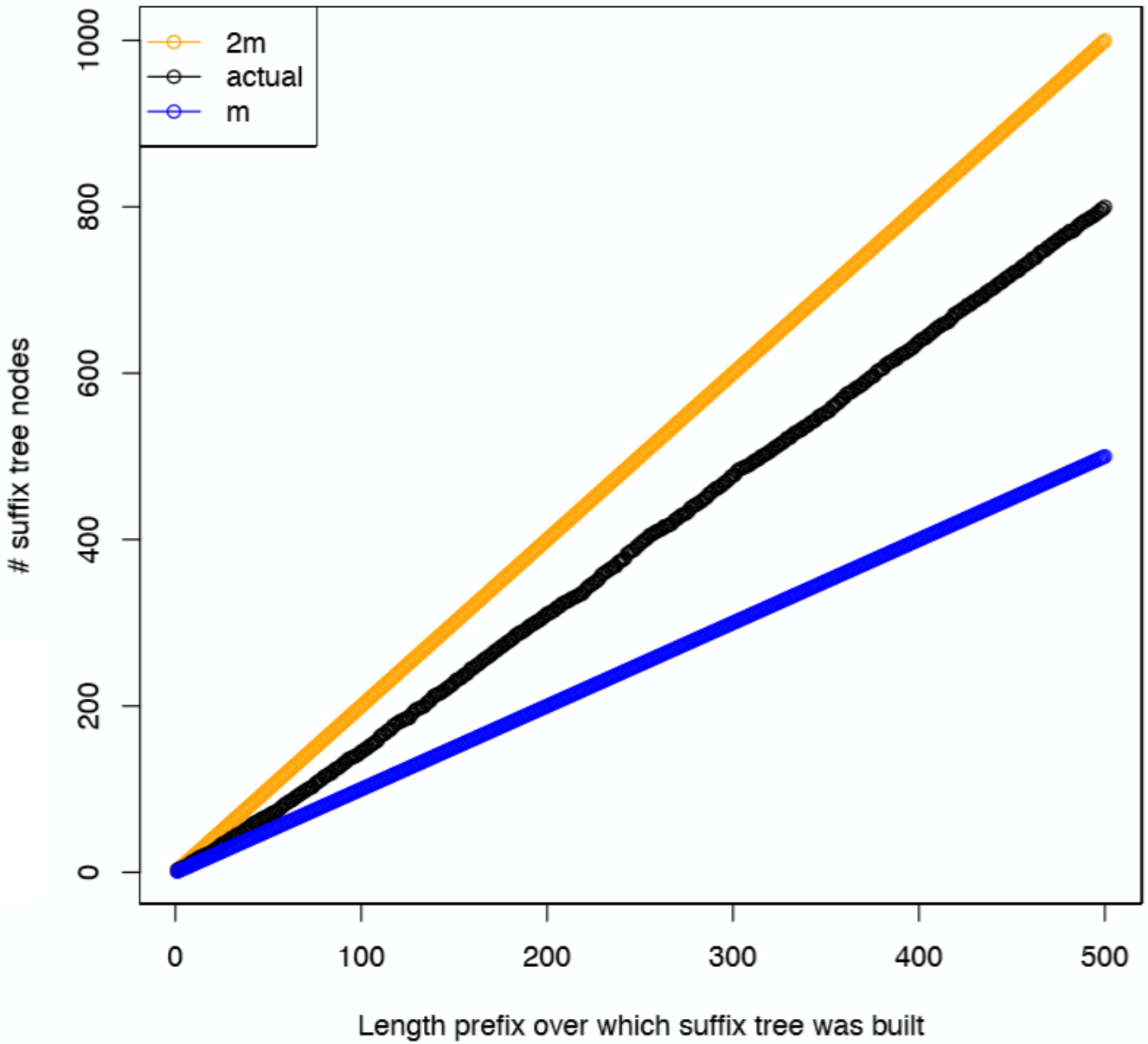


Myšlenka 3

Do listových uzlů uložíme začátek suffixu v textu T.



Suffixový strom - prostorová složitost

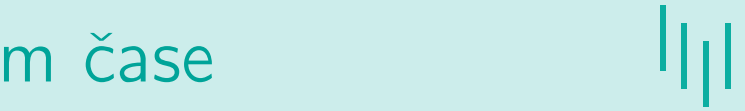


Suffixový strom - sestavení

Dva algoritmy k sestavení:

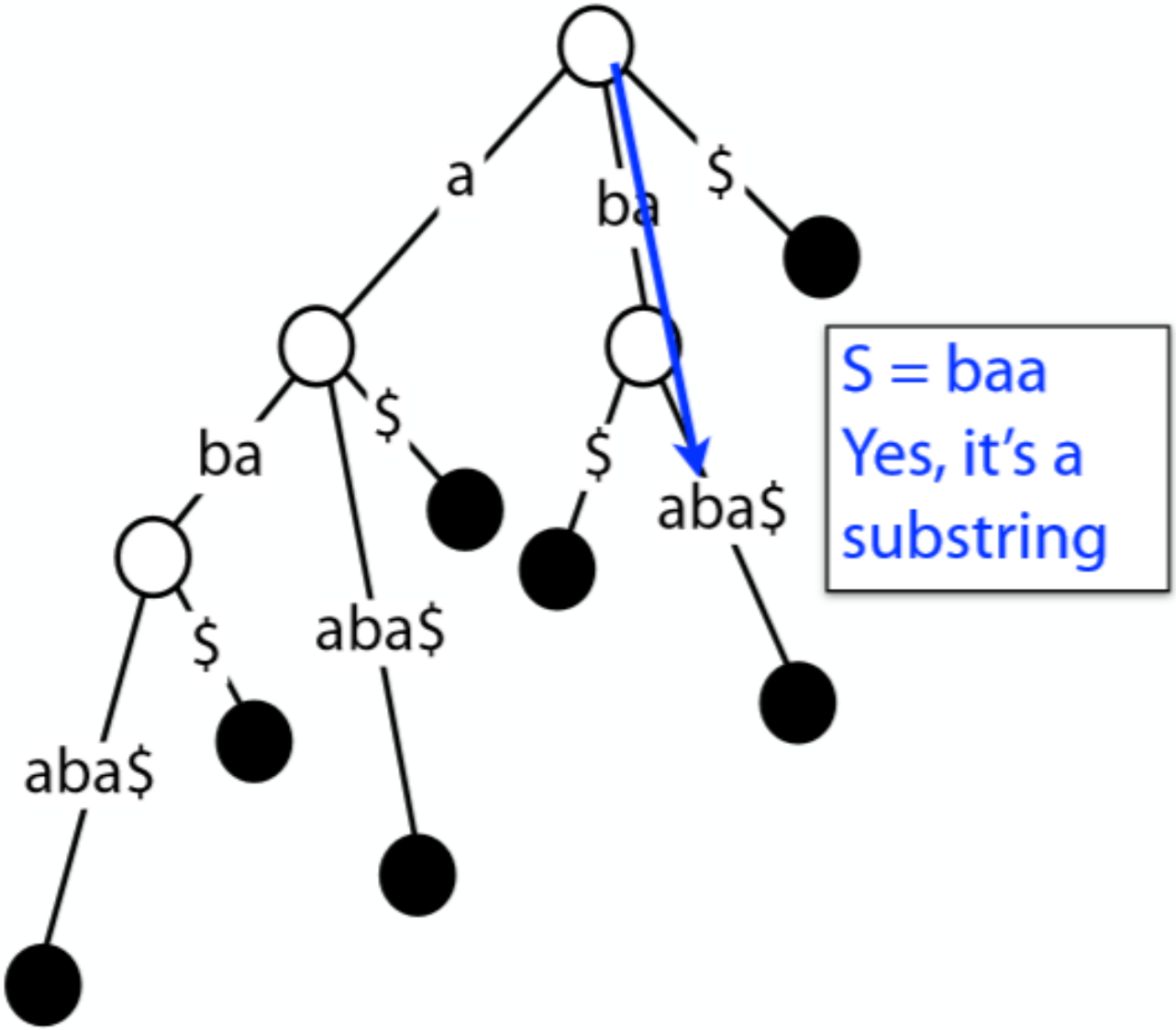
- Trie => Suffixový strom - $O(n^2)$ čas a prostor.
  1. Sestavit trie
  2. Spojit nevětvící se uzly a přeznačit označení hran.
- Přímé sestavení suffixového stromu - $O(n^2)$ čas, $O(n\sigma \log n)$
  1. Sestavte strom pro nejdelší suffix.
  2. Upravit strom, aby obsahoval druhý nejdelší suffix.
  3. Upravit strom, aby obsahoval třetí a další suffixy...

Suffixový strom - sestavení v lineárním čase

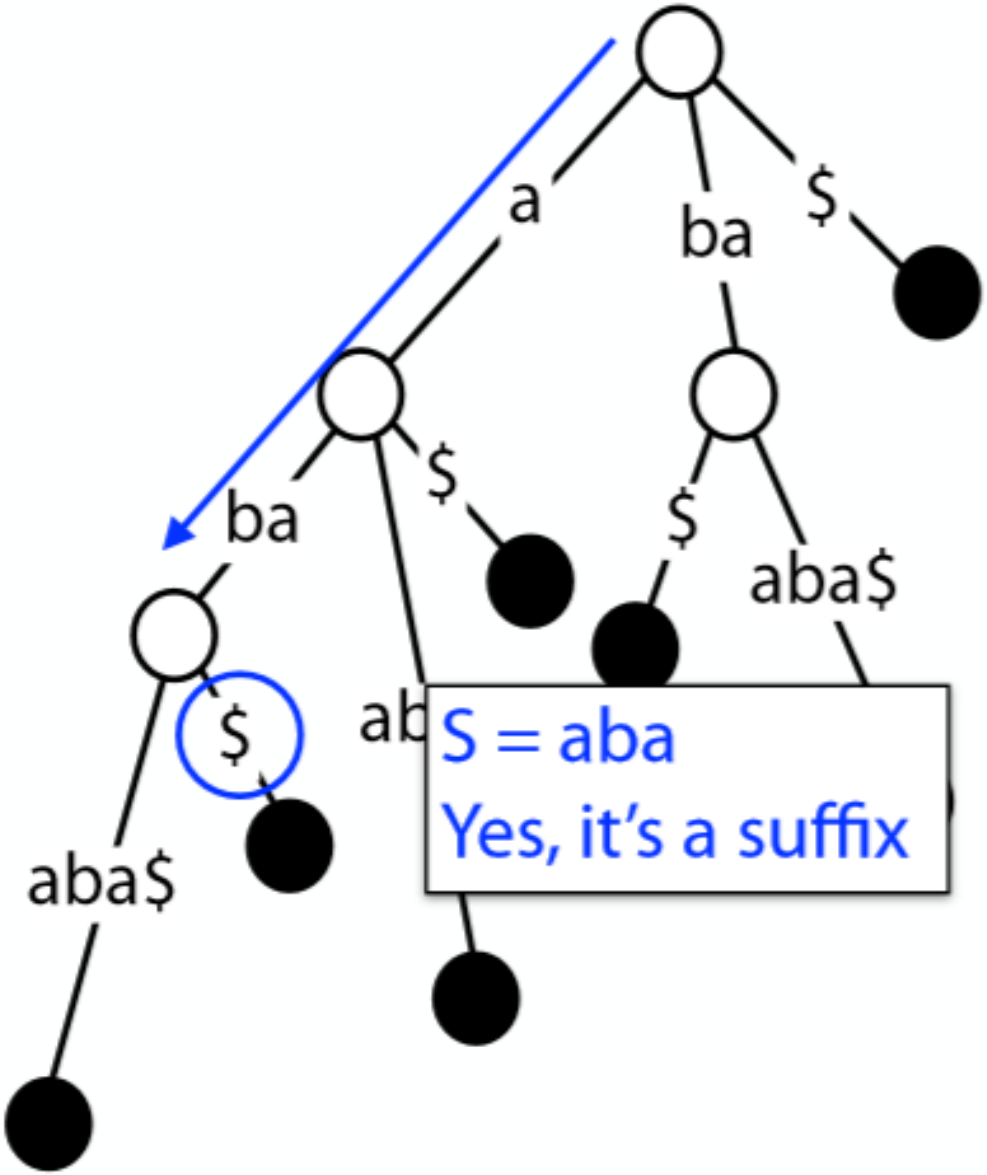


- Weiner's algorithm [1973]
- McCreight [JACM 1976]
- Ukkonen [Algorithmica 1995] - $O(m)$ čas a prostor.
- Farach [FOCS 1997]
- Hon, Sadakane, Sung [FOCS 2003]

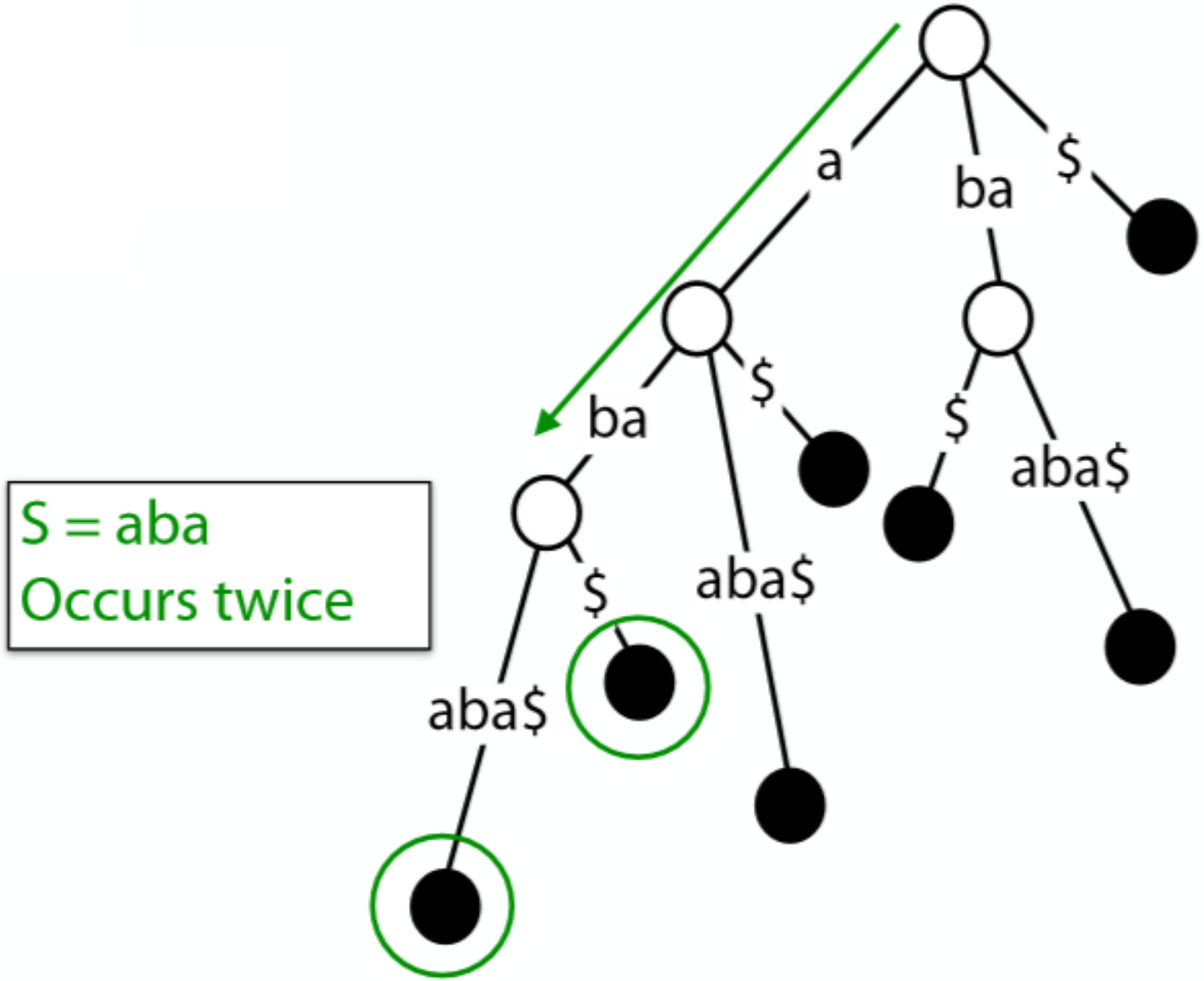
Suffixový strom - operace exists



Suffixový strom - operace isSuffix

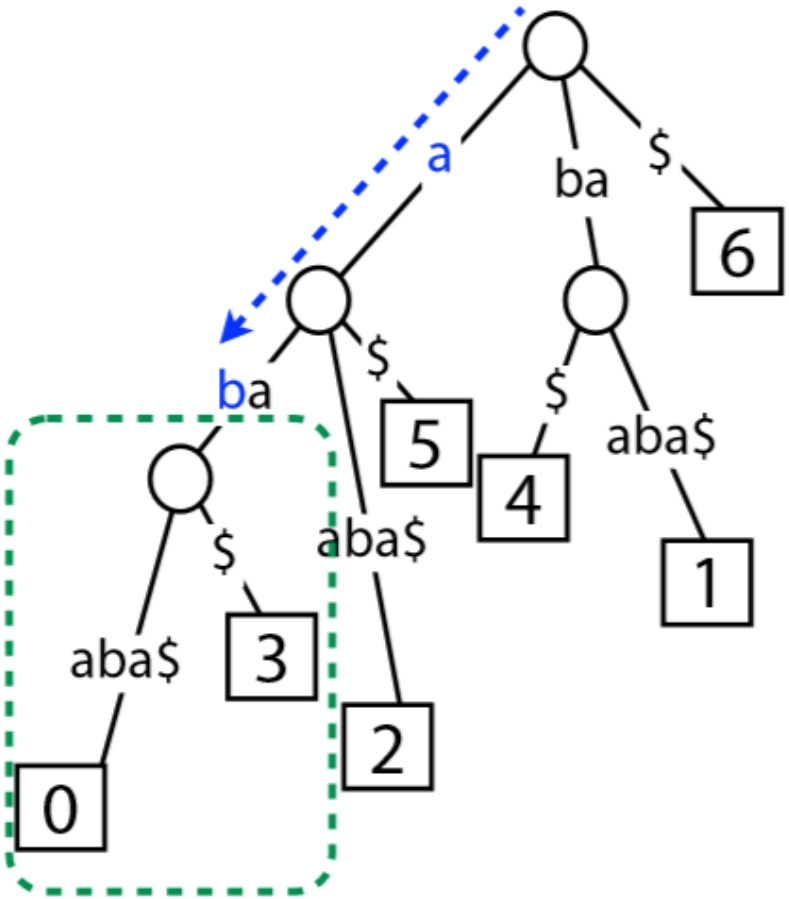


Suffixový strom - operace count



Suffixový strom - operace find

- Nalezneme podstrom odpovídající hledaném vzoru. $P = ab$. $O(n)$.
- Projdeme podstrom do hloubky až projdeme všechny listové uzly, těch je k. Vnitřních uzlů je nejvýše $k - 1$.
- Celková složitost $O(n + 2k - 1) = O(n + k)$.



Suffixové pole

Suffixové pole

- Suffixový strom pro lidský genom je sice teoreticky $O(m)$, nicméně v realitě to znamená > 45GB.

Suffixové pole

Suffixové pole je pole kladných celých čísel, které reprezentují počáteční pozice suffixů v lexikografickém pořadí.

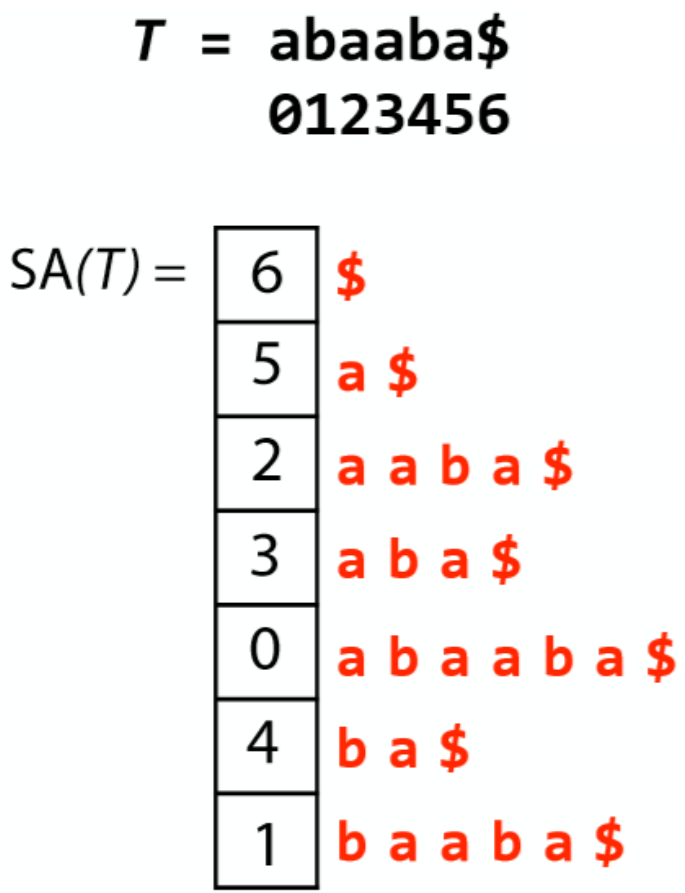
T = abaaba$
0123456

SA(T) =

6	$
5	a $
2	a a b a $
3	a b a $
0	a b a a b a $
4	b a $
1	b a a b a $

Suffixové pole

- K řetězci o délce n přidáváme opět terminální symbol.
- Pole tak obsahuje $n + 1$ indexů.
- Každý index vyžaduje $\lceil \log n \rceil$ bitů.
- Prostorová složitost $O(n \log n)$.
- Lidský genom cca 12 GB plus původní text.



Suffixové pole - vyhledávání

- Hledáme vzor P.
- Jestliže se P vyskytuje v T, pak P musí být prefixem nějakého suffixu.
- Navíc, suffixy sdílející stejný prefix jsou setřizeny v suffixovém poli za sebou.
- Nalezení P binárním vyhledáváním.

T = abaaba$
0123456

SA(T) =

6	$
5	a $
2	a a b a $
3	a b a $
0	a b a a b a $
4	b a $
1	b a a b a $

Suffixové pole - vyhledávání příklad

- $P = ba$.
- Začneme na pozic $SA[n/2] = 3$.
  $T[SA[3]..] = aba\$$.
- $aba\$ < ba$ prohledáváme spodní polovinu suffixového pole.
- $SA[(6+4)/2] = 5$.
  $T[SA[5]..] = ba\$$. Shoda P a suffixu od pozice 5.
- Prohledáme pozice nad a pod indexem číslo pět, abychom zjistili rozsah pro prefix ba => [4,1] indexy v textu.

T = abaaba$
0123456

SA(T) =	
6	$
5	a $
2	a a b a $
3	a b a $
0	a b a a b a $
4	b a $
1	b a a b a $

Suffixové pole - vyhledávání analýza

- Provádíme $\log n$ půlení intervalu.
- V každém kroku navíc provedeme až m porovnání symbolů => $O(m)$ při každém půlení
- Nalezení prvního výskytu je tedy operace s $O(m \log n)$ časovou složitostí.
- Nalezení všech k výskytů vyžaduje dalších až $k + 2$ porovnání, tudíž dohromady $m \log n + mk$ => $O(m(k + \log n))$ složitost.

T = abaaba$
0123456

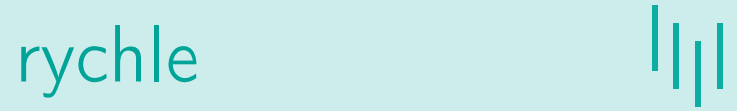
SA(T) =

6	$
5	a $
2	a a b a $
3	a b a $
0	a b a a b a $
4	b a $
1	b a a b a $

Suffixové pole - sestavení

- S použitím běžných třídících algoritmů $O(n^2 \log n)$.
- V praxi 80% řetězců délky 20 jsou unikátní => $O(20n \log n) = O(n \log n)$
- Například Quicksort, Heapsort.
- Karkainnen [2003] - $O(n)$ algoritmus.
- Ko, Aluru [2003] - SAIS, $O(n)$ algoritmus.

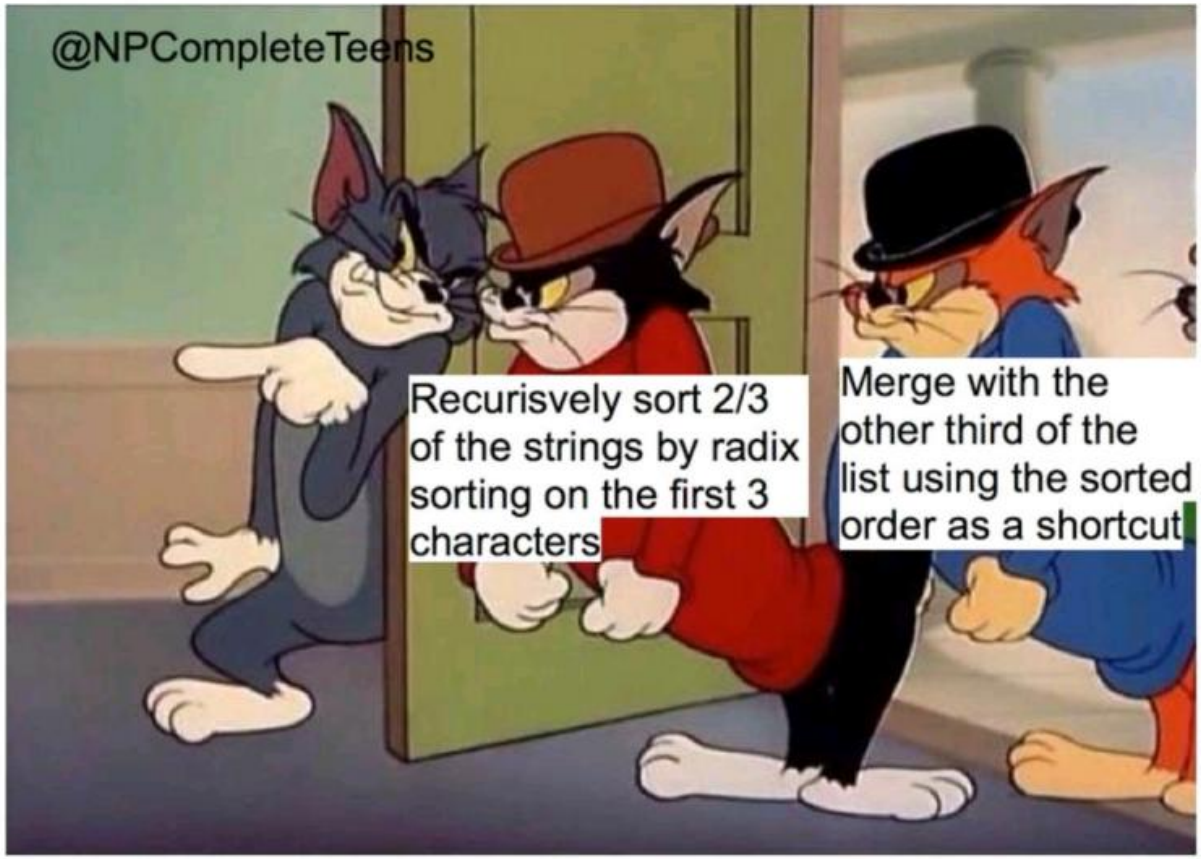
Suffixové pole - sestavení dostatečně rychle



- Spočítáme si počty stejných prefixů o konstatní délce p u všech suffixů. $O(pn) = O(n)$
- Rozdělíme si suffixové pole do přihrádek podle počtu suffixů se stejným prefixem - RadixSort.
- Každou přihrádku, pak setřídíme pomocí Quicksortu.
- Dostatečně rychlé řešení pro indexaci genomu.
- Teoreticky nejlepší výkon pro $p = 3$, cca 10x rychlejší, než samotný Quicksort.

Suffixové pole - :)

When you need to make a lexicographically ordered suffix array

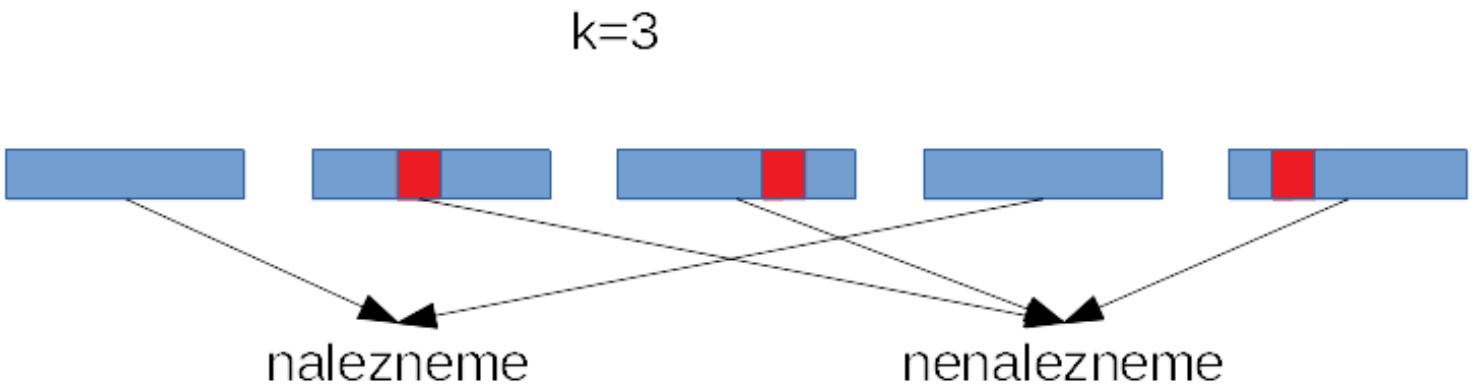


Dirichletův princip

- Struktury umožňují nalézt pouze exaktní shody vzoru a textu => mutace?.
- V angl. literatuře Pigeonhole principle.
- Používá se např. v aligneru Bowtie.

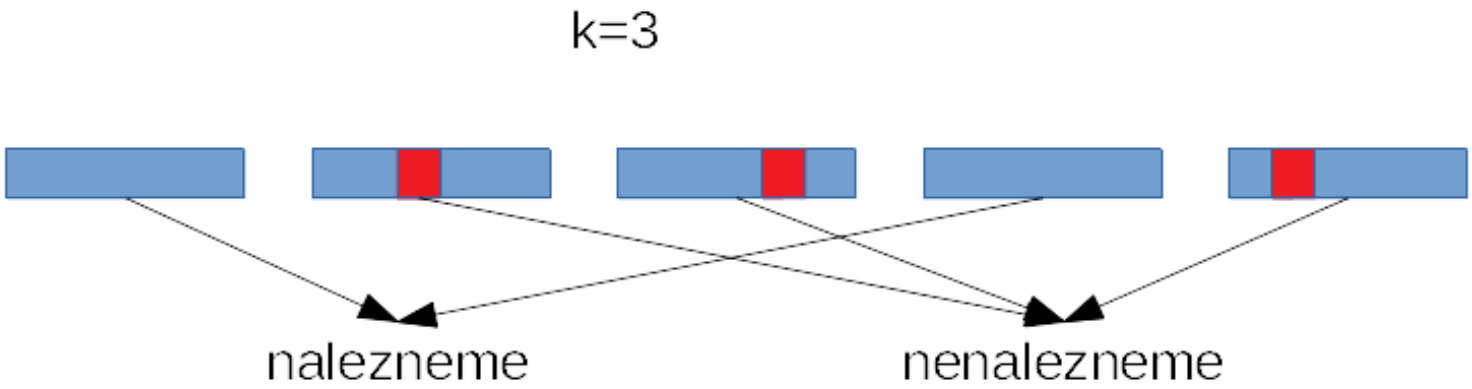
Dirichletův princip

Uvažujme, že v našem systému tolerujeme k mutací. Pak pokud vstupní sekvenci rozdělíme na $k + l$ nepřekrývajících se řetězců, máme jistotu, že alespoň l řetězců nebude obsahovat mutaci.



Dirichletův princip

- 80% sekvencí délky 20nt jsou unikátní.
- Každou hledanou sekvenci rozdělíme na $k + l$ řetězců.
- Každý řetězec zkusíme nalézt v referenčním genomu a pozice jeho výskytu si uložíme do seznamu.
- Seznamy spojíme do jednoho seznamu/pole a setřídíme.
- Pokud se nějaké pozice vyskytují alespoň l-krát, pak na danou sekvenci od dané pozice aplikujeme dynamické programování a vybereme pozici s nejvyšším skóre.



Pro zájemce - témata projektů

- Ukkonenův algoritmus pro sestavení suffixového stromu v $O(n)$.
- SAIS algoritmus pro sestavení suffoxového pole v $O(n)$.

Příště uvidíte

- Backtracking v suffixovém poli
- FM-Index
- Dokončení alignmentu - kombinace FM-Indexu a dynamického programování.

DÄŻkuji za pozornost

Michal Vašinek

VŠB – Technická univerzita Ostrava

FEI/EA404

michal.vasinek@vsb.cz

2. října 2019

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY