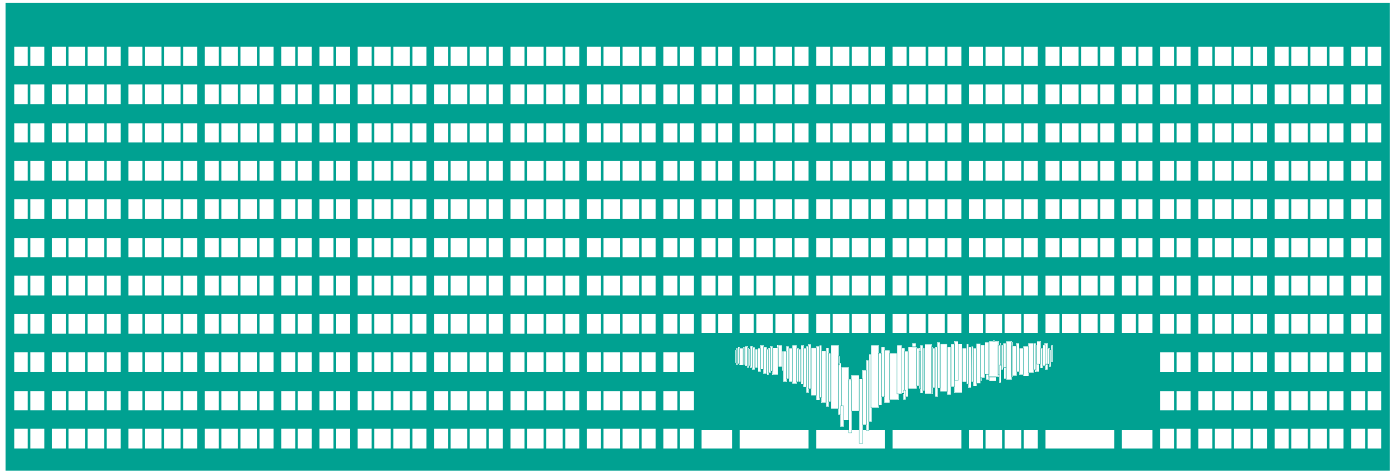
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

Bioinformatika - algoritmy a analýza dat

Indexace Genomu

Michal Vašinek

VŠB – Technická univerzita Ostrava

FEI/EA404

michal.vasinek@vsb.cz

26. října 2022

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Přehled přednášky

- FM-Index
- Porovnání genomů - hledání společných sekvencí

Připomenutí z minula

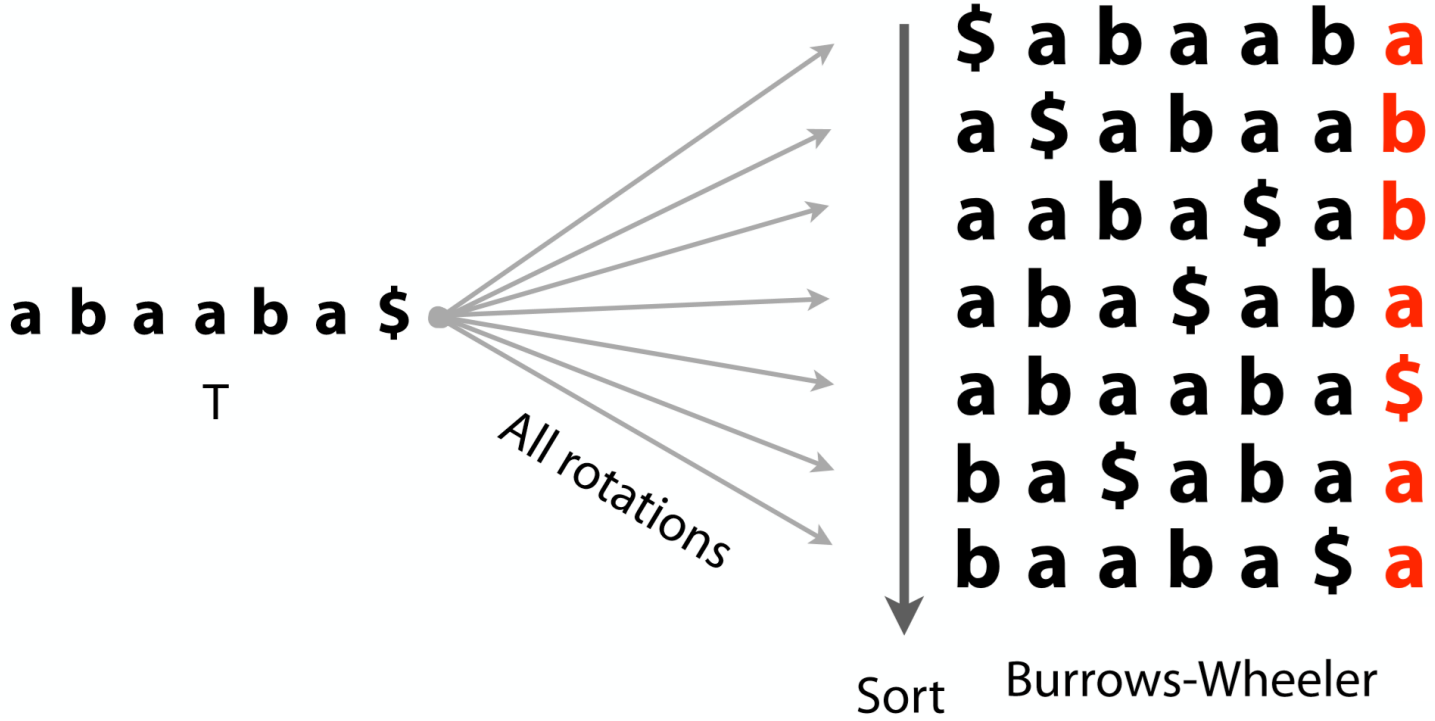
Na vstupu text $T[1..n]$ a sekvence $s[1..m]$.

- **Trie**
  - Ověření výskytu řetězce - $O(m)$.
  - Nalezení k výskytů sekvence - $O(m + k)$.
  - Neumožňuje nalézt přesnou pozici sekvence.
  - René de la Briandais [1959]
- **Suffixový strom**
  - Oproti Trie navíc operace nalezní pozice (Find) v $O(m + k)$.
  - Obrovské paměťové nároky - lidský genom $> 46GB$.
  - Weiner [1973]
- **Suffixové pole**
  - Find v $O(m \log n)$.
  - Lepší paměťové nároky - lidský genom $\approx 12GB$.
  - Manber & Meyers [1990]

FM-Index

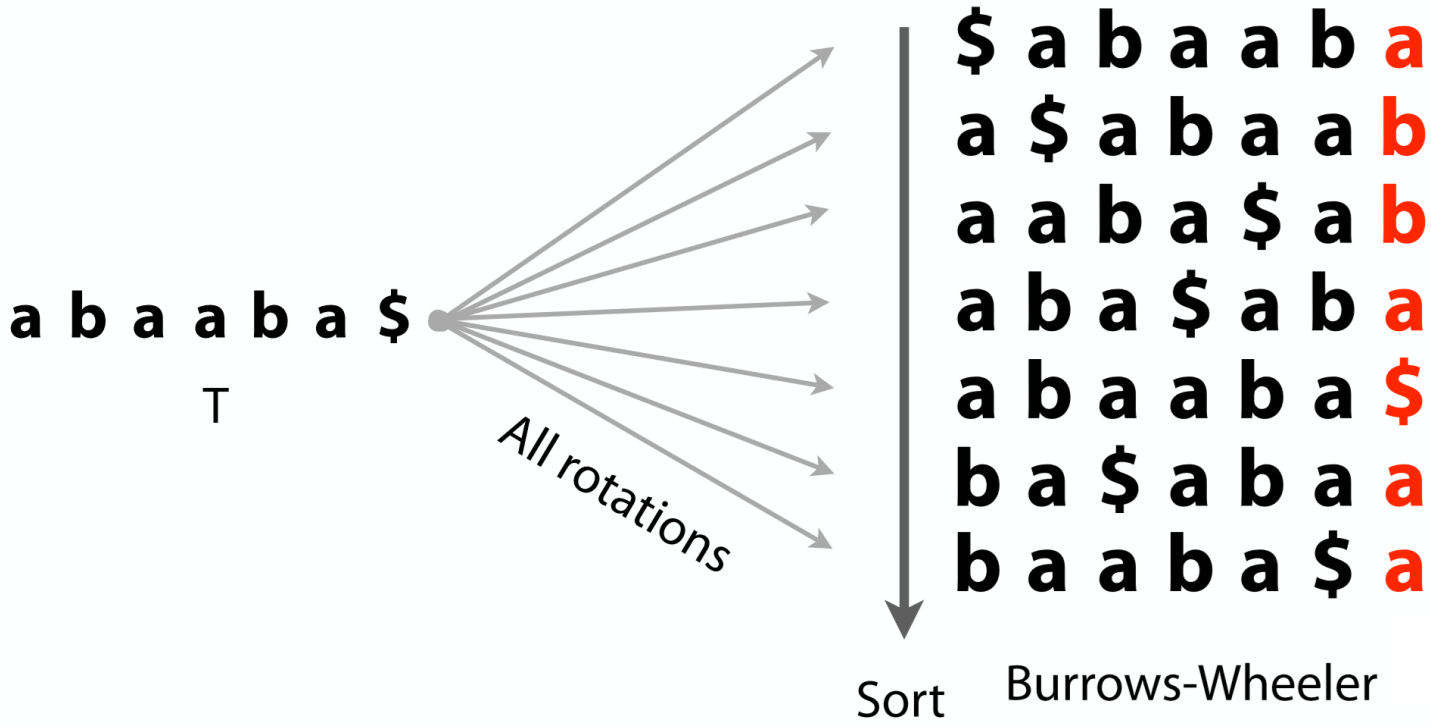
Burrows-Wheelerova transformace

- V roce 1994 Burrows a Wheeler navrhli reversibilní transformaci textu vhodnou pro kompresi dat.
- Používá se v kompresním formátu bzip2.
- Založena na setřizení rotací textu, podobně jako suffixové pole, pořadí řetězců je identické.
- Konec textu opět označen terminálním symbolem \$.



Burrows-Wheelerova transformace

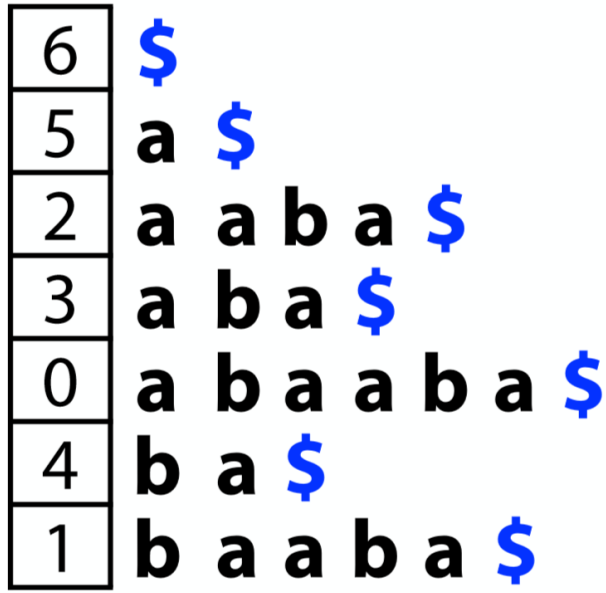
- Výstupem je poslední sloupec Burrows-Wheelerovy matice.
- Komprese dat - v posledním sloupci (ozn. L) shlukování stejných symbolů.
- $BWT(abaaba\$) = abba\$aa$.



BWT a Suffixová pole

- V praxi nevytváříme Burrows-Wheelerovu matici => $O(n^2)$ prostorová náročnost.
- Pomáháme si algoritmy pro sestavení suffixového pole.
- Sloupec L má stejný počet symbolů, jako vstupní text T, navíc je snadno komprimovatelný.

BWM(T)



SA(T)

BWT a Suffixová pole

$$BWT[i] = \begin{cases} T[SA[i]-1] & \text{if } SA[i] > 0 \\ \$ & \text{if } SA[i] = 0 \end{cases}$$

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

BWM(T) SA(T)

LF zobrazení

- LF zobrazení - klíčová vlastnost pro zpětnou transformaci, využívá se i pro vyhledávání sekvencí.
- Popisuje vztah mezi posledním sloupcem L - last a prvním sloupcem F - first.
- Očíslujme každý stejný symbol ve vstupním textu.

F						L
$	a_0	b_0	a_1	a_2	b_1	a_3
a_3	$	a_0	b_0	a_1	a_2	b_1
a_1	a_2	b_1	a_3	$	a_0	b_0
a_2	b_1	a_3	$	a_0	b_0	a_1
a_0	b_0	a_1	a_2	b_1	a_3	$
b_1	a_3	$	a_0	b_0	a_1	a_2
b_0	a_1	a_2	b_1	a_3	$	a_0

LF - proč platí

- Pozorně prozkoumejme druhý a třetí řádek Burrows-Wheelerovy matice.
- Vidíme, že platí:

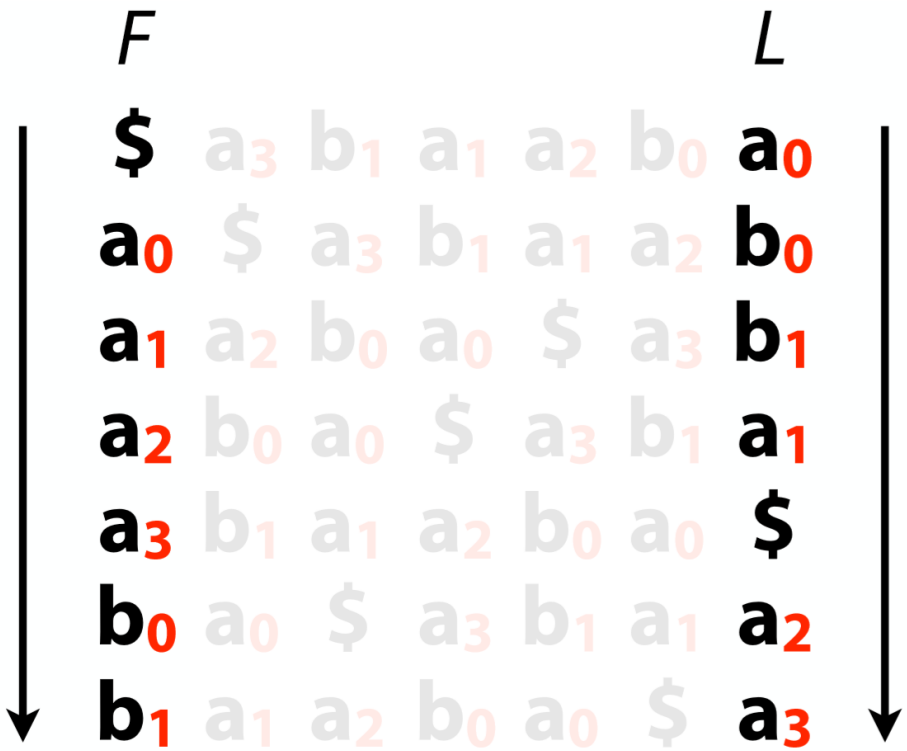
$$\texttt{a\$abaab} \prec \texttt{aaba\$ab}$$

- Pokud přesuneme symbol b z posledního sloupce na začátek, uspořádání pořád platí:

$$\texttt{ba\$abaa} \prec \texttt{baaba\$a}$$

Pořadí symbolů

- V praxi symboly indexujeme podle jejich výskytu ve sloupci L.
- Povšiměnete si, že pro uložení sloupce F nám stačí pouze $\sigma \log n$ bitů => ukládáme pouze četnosti symbolů.



Výpočet řádku

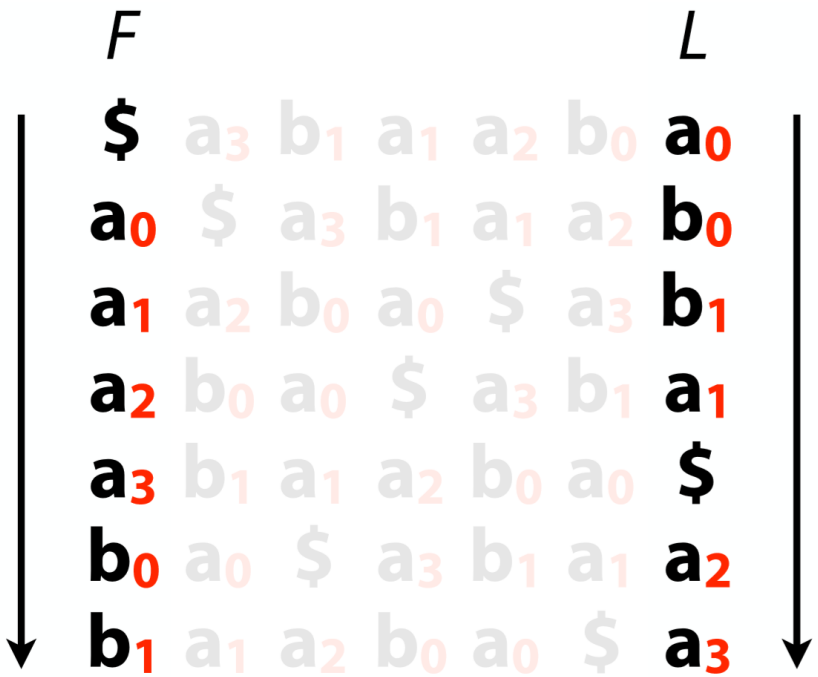
- Uvažujme následující počty symbolů ve sloupci F:

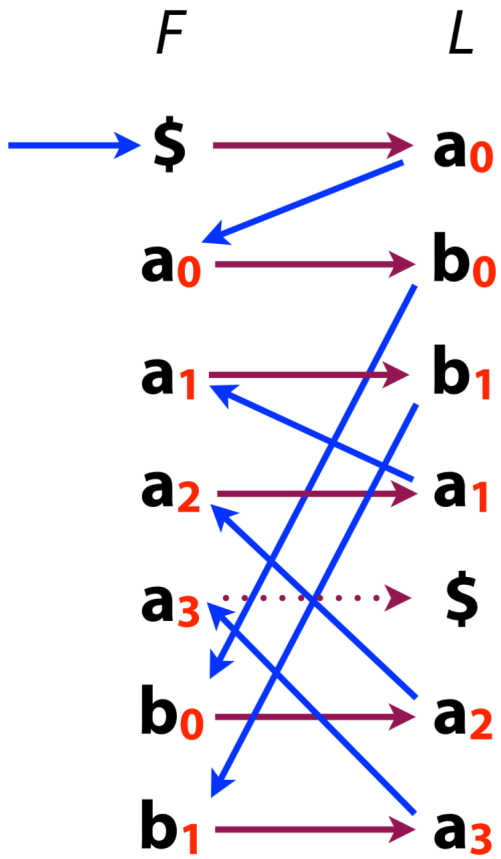
\$	A	C	G	T
1	100	200	300	200

- Který řádek (počítáno od nuly) obsahuje G_{100}?
- $radek(G_{100}) = f(\$) + f(A) + f(C) + 100 = 501$

Zpětný průchod

- Začínáme v prvním řádku: ve sloupci F je \$.
- Symbol ve sloupci L předchází v T symbol ve sloupci F.





BWT a dodatečné pomocné struktury => FM-Index

FM-Index

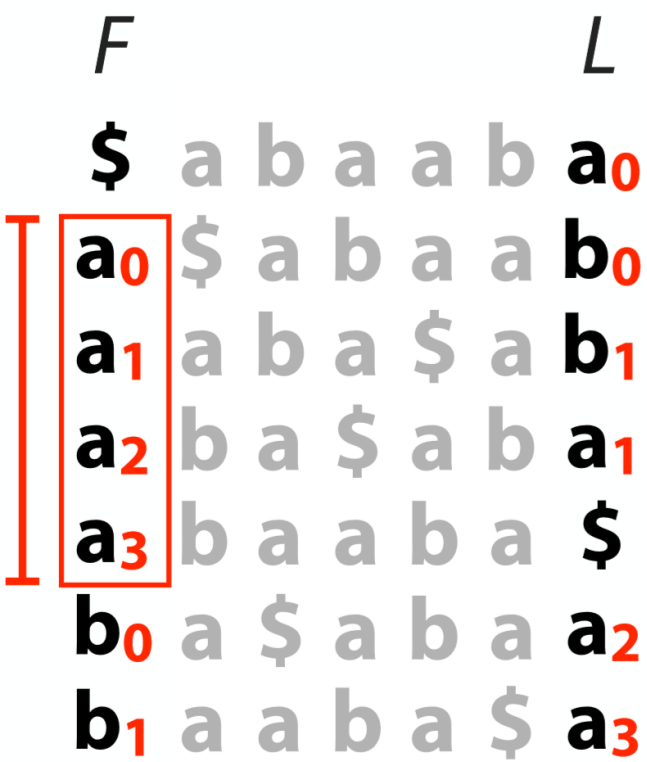
FM-Index je index kombinující BWT s několika pomocnými datovými strukturami.

- Sloupec L - lze zmenšit na $H(T)$.
- Sloupec F - lze zmenšit na $\sigma \log n$.
- Pořadí symbolů - funkce Rank, není třeba ukládat všechny indexy symbolů.
- Suffixové pole - ukážeme si, že není zapotřebí celé suffixové pole.
- Povšimněte si, že nepotřebujeme samotný text T.

FM-Index dotazování - funkce Exists

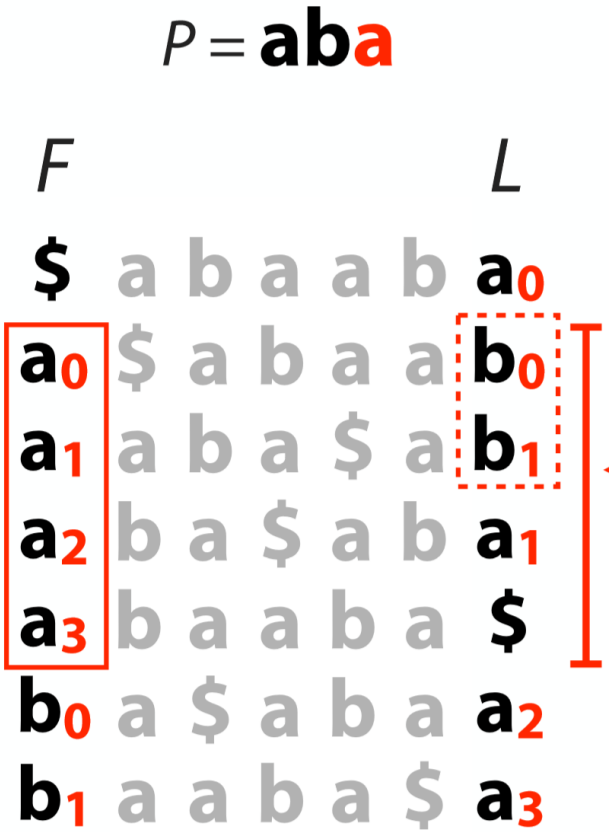
- Hledání začínáme od posledního symbolu hledaného řetězce.
- Ze sloupce F známe pozice výskytů.

$P = \mathbf{aba}$



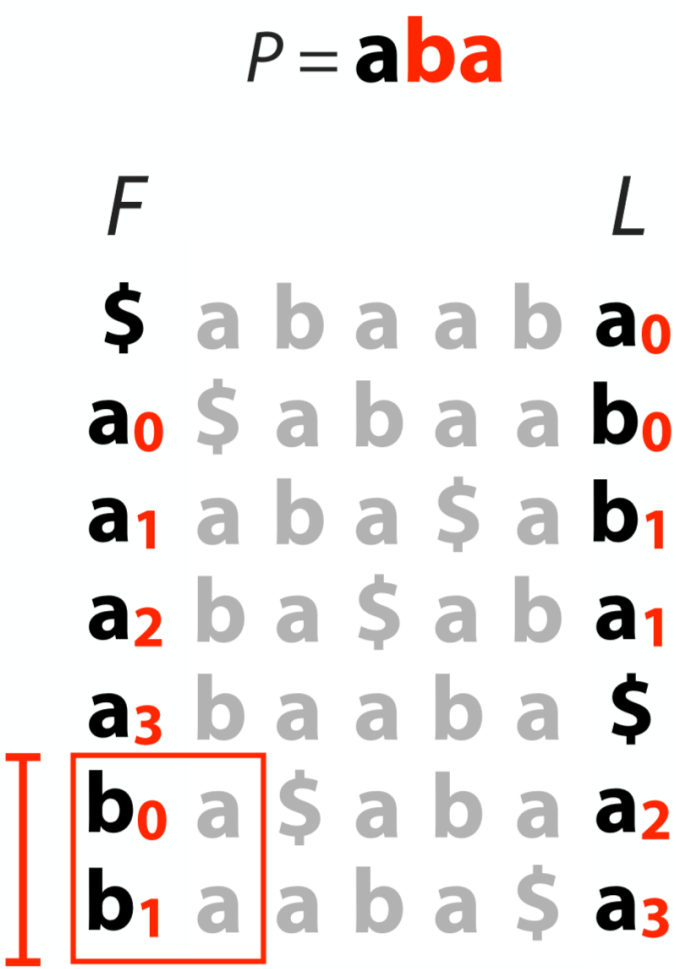
FM-Index dotazování - funkce Exists

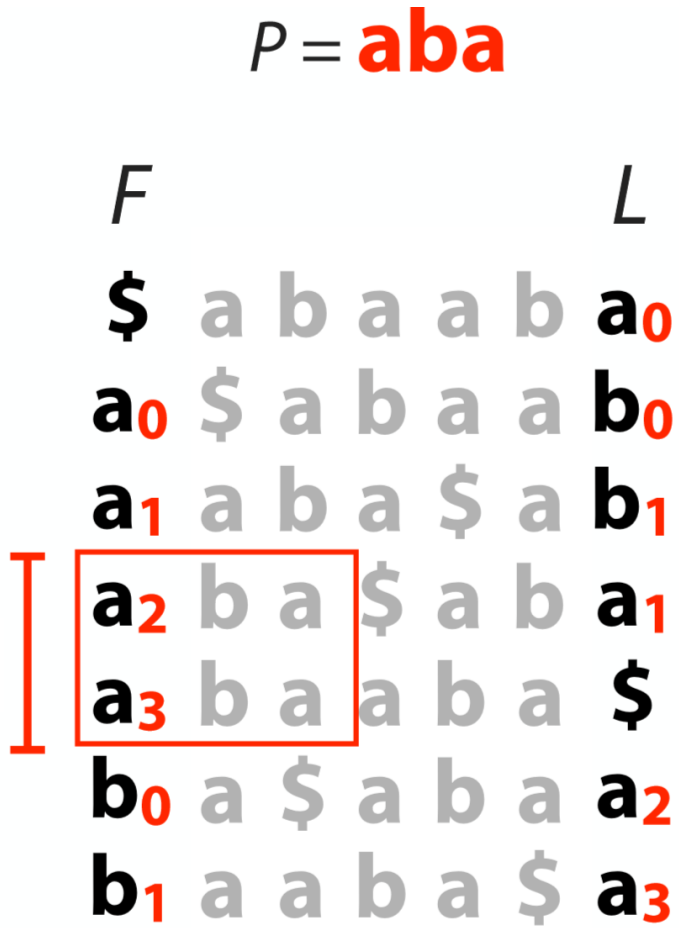
- Použijeme LF mapování.
- Ve sloupci L symboly, které předcházejí symbol a. Nalezneme symbol b s nejnižším a nejvyšším indexem (rank).



FM-Index dotazování - funkce Exists

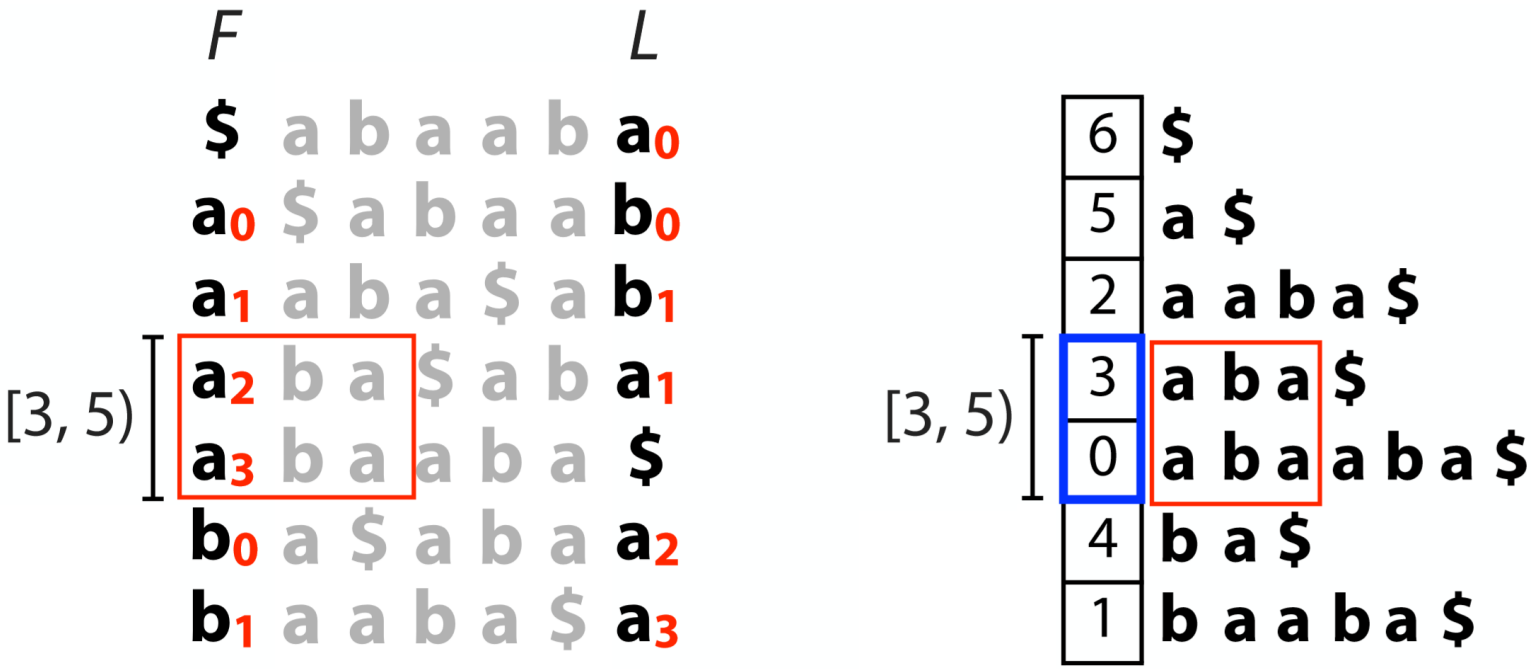
- Symbolům b_0 a b_1 odpovídají symboly a_2 a a_3 ve sloupci L.
- Nalezli jsme hledaný vzor $P = aba$.





FM-Index - Exists, Count a Find

- Ve chvíli, kdy zjistíme, zda se hledaný vzor v textu vyskytuje, okamžitě zjišťujeme také počet jeho výskytů => rozsah intervalu funkce Rank pro poslední nalezený symbol.
- Oproti suffixovému poli však zatím neznáme přesnou polohu v T.
- Nabízí se použít jako doplňkovou strukturu suffixové pole => Find.



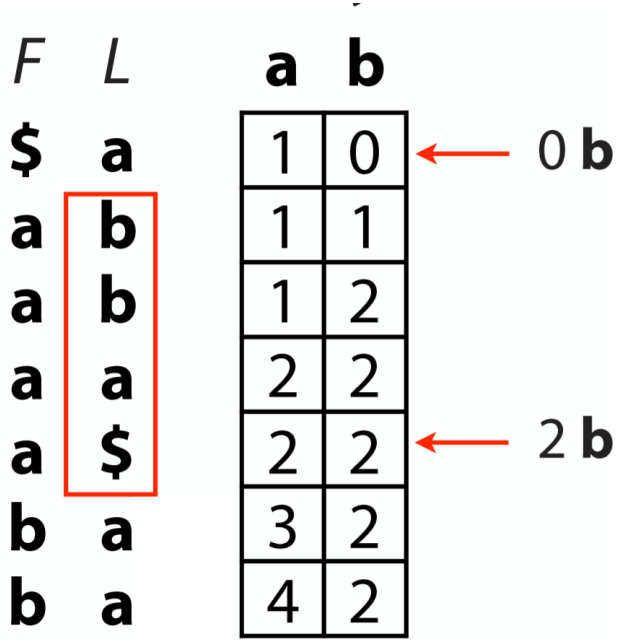
FM-Index - průchod sloupcem L je pomalý

- Pokud bychom pokaždé procházeli sloupec L abychom zjistili pořadí daného symbolu, museli bychom L projít v nejhorším případě m-krát.
- Řešení: předpočítejme si funkci Rank.

L	a	b
a	**1**	0
b	1	**1**
b	1	**2**
a	**2**	2
$	2	2
a	**3**	2
a	**4**	2

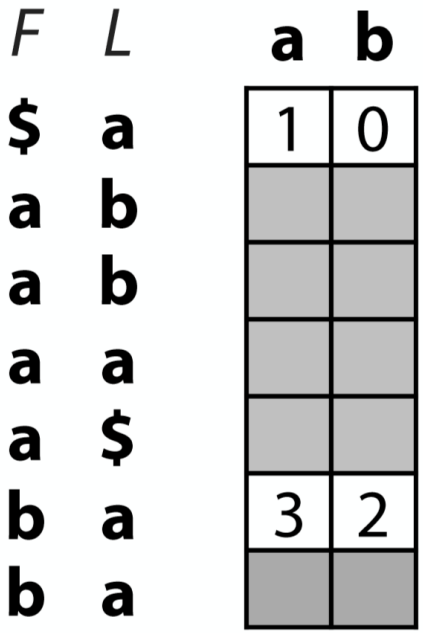
Funkce Rank

- Předpočítaný Rank použijeme k určení minimálního a maximálního indexu dalšího symbolu.
- Podíváme se do sloupce odpovídajícího danému symbolu, pak minimální index nalezneme na pozici o jedna menší, než je aktuální dolní mez intervalu a maximální index nalezneme na pozici odpovídající konci intervalu.



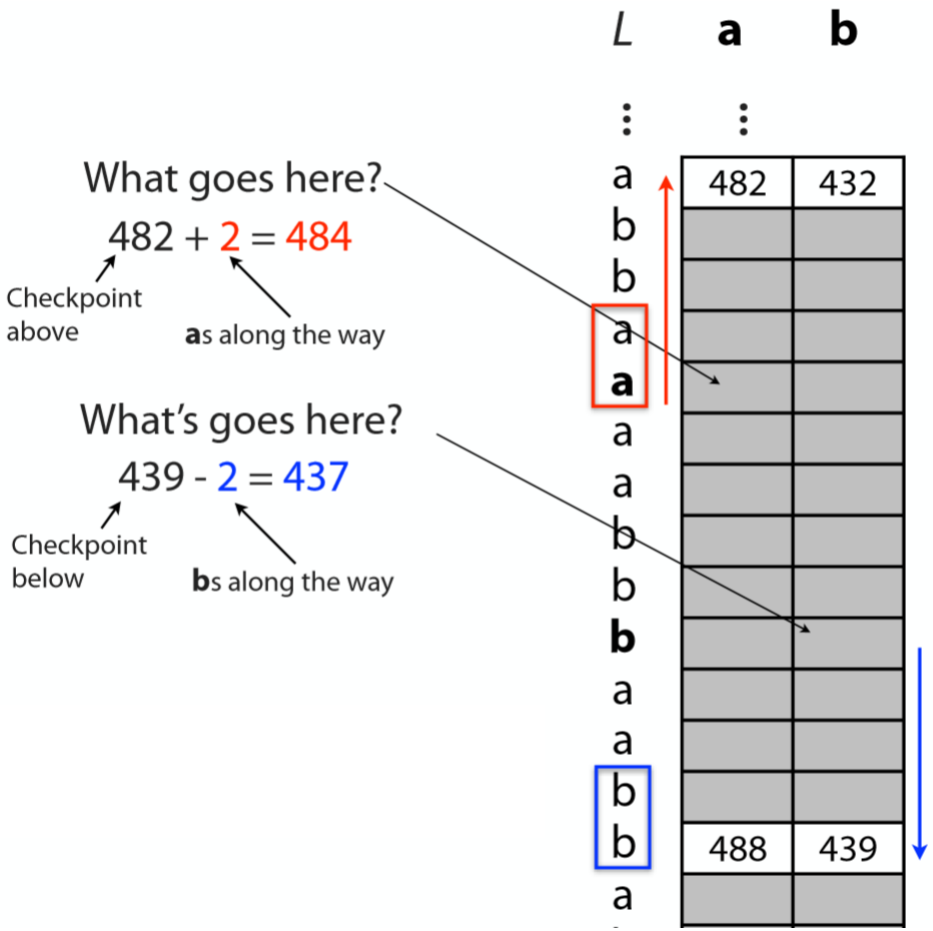
Dopočet z omezeného množství hodnot funcke Rank

- Funkce Rank vyžaduje uložení dalších σn hodnot.
- Řešení: uložit Rank pouze pro vybrané řádky, např. každý pátý řádek a další hodnoty dopočítat za běhu.



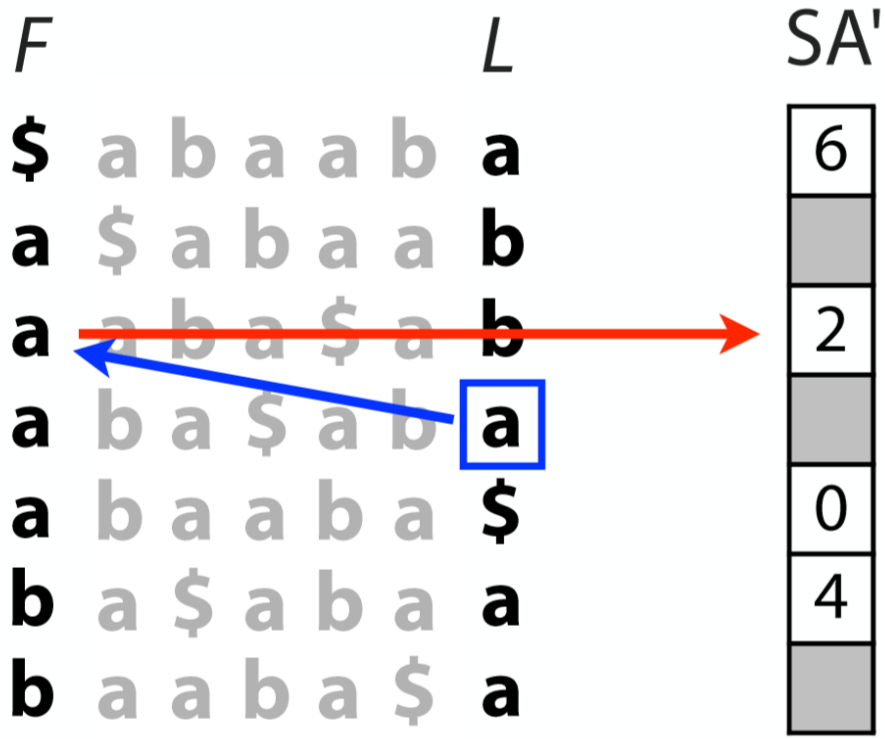
Dopočet z omezeného množství hodnot funcke Rank

- Pro hledaný symbol/řádek nalezneme nejbližší uložený Rank, průchodem sloupcem L spočítáme výskyty symbolu a ty připočteme/odečteme k hodnotě v předuloženém Ranku.
- Protože nejvyšší počet porovnání ve sloupci L je dán vzdáleností uložených bodů, která je pro daný výpočet konstatní, můžeme operaci Rank řešit s $O(1)$ časovou složitostí.



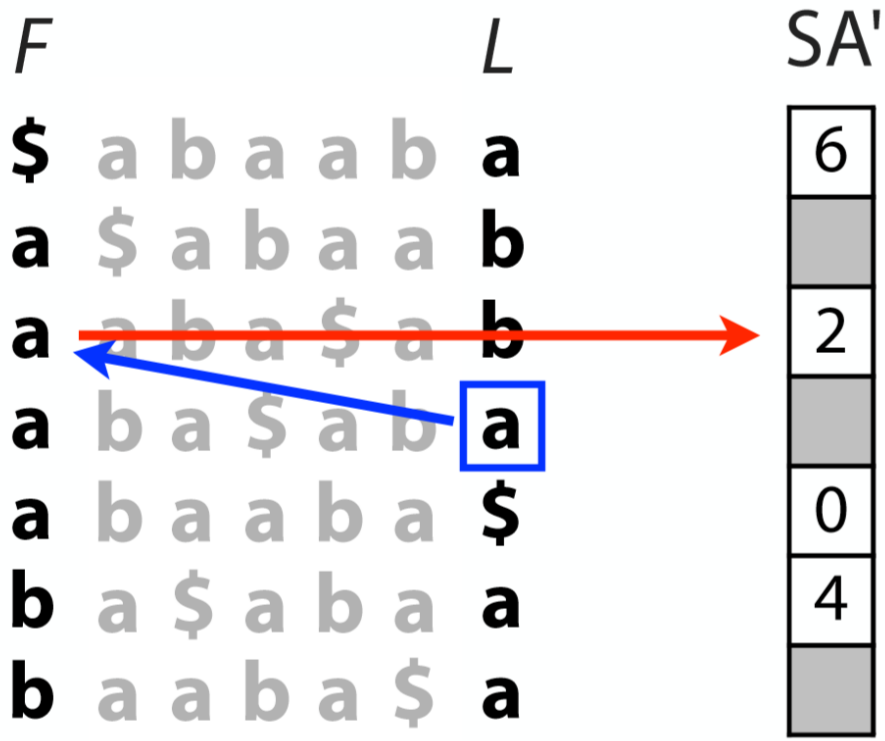
Dopočet indexů do T

- Stejný problém, jako s funkcí Rank máme i s indexy v suffixovém poli => suffixové pole je značně velké: $4n$ bytů pro lidský genom.
- Řešení: opět vynechání vybraných indexů.



Dopočet indexů do T

- Problém: pozice v SA není uložena.
- Použijeme LF mapování, abychom se dostali do pozice v SA, která je uložena. K nalezené pozici přičteme počet kroků pro její nalezení. Zde v $SA[2] = 2$, provedli jsme jeden krok, tudíž $SA[3] = 2 + 1 = 3$.



FM-Index - Lidský genom

- Nechť $a = 1/32$ je poměr hodnot SA, které zanecháme v indexu.
- Nechť $b = 1/128$ je poměr řádků funkce Rank, které zanecháme v indexu.
- Sloupec F - σ integerů => A,C,G,T => 4*32 = 16 bajtů.
- Sloupec L - n znaků => 2 bity na znak, 3miliardy znaků = 750 MB.
- Suffixové pole => 3 miliardy znaků, každý index 32 bitů (4 bajty) / 32 $\approx$ 400 MB.
- Rank => 3 miliardy znaků, abeceda o 4 znacích, každý počet 32 bitů (4 bajty) / 128 $\approx$ 400 MB.
- Dohromady 1.5GB.

Porovnání se suffixovým stromem a suffixovým polem

	Suffix tree	**Suffix array**	**FM Index**
Time: Does P occur?	$O(n)$	$O(n \log m)$	$O(n)$
Time: Count k occurrences of P	$O(n + k)$	$O(n \log m)$	$O(n)$
Time: Report k locations of P	$O(n + k)$	$O(n \log m + k)$	$O(n + k)$
Space	$O(m)$	$O(m)$	$O(m)$
Needs T?	*yes*	*yes*	*no*
Bytes per input character	>15	~4	~0.5

$m = |T|, n = |P|, k =$ # occurrences of P in T

Porovnání genomů

Porovnání genomů

- Zajímáme se o podobnosti v genomech.
- Které sekvence jsou u různých oprganismů stejné či podobné.
- Identifikovat sekvence, které se zachovávají napříč evolucí.
- 2018 - pouze 3500 eukaryotických organismů bylo osekvenováno, cca 0.2 %. Pouze 100 v kvalitě referenční sekvence.
- 2018/listopad - Earth Biogenome Project[1] - cíl: osekvenování všech organismů.

[1] `https://www.earthbiogenome.org/roadmap`

Proč nepoužívat SW

- Známe algoritmus pro výpočet podobných sekvencí => Smith-Waterman (SW)
- SW je navržen pro porovnání dvou genů nebo proteinů.
- Je příliš pomalý pro porovnání genomů.
- SW nachází pouze lokální řešení, není schopen pojmout celkovou podobnost dovu genomů.

Obecný framework a principy

Předpoklad

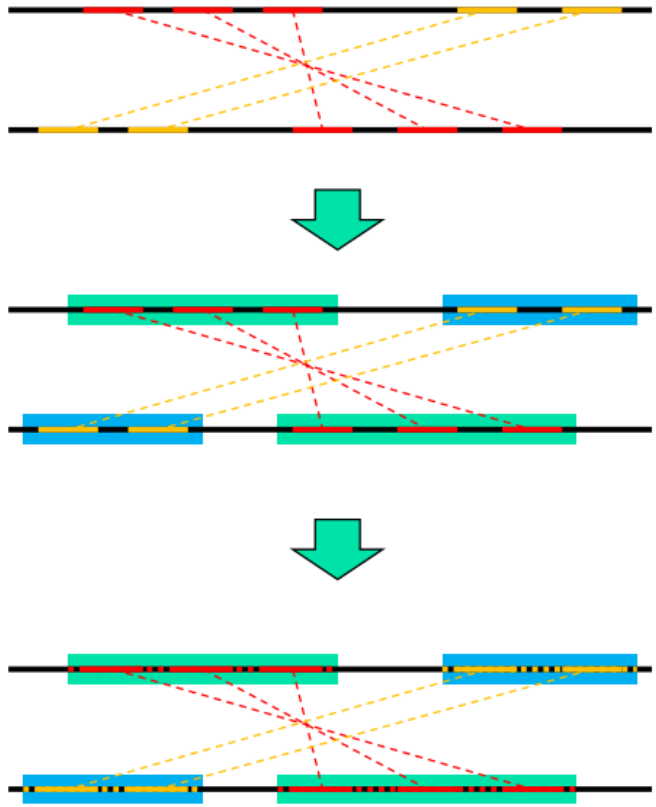
Každé dva genomy sdílejí určitou množinu stejných (evolučně zakonzervovaných, neměnných) sekvencí.

Princip

Nejdříve vzájemně přiřadíme neměnné sekvence, poté řešení rozšíříme o proměnlivé sekvence.

Obecný framework a principy

1. Identifikace potenciálních překryvů.
2. Identifikace ko-lineárních překryvů, které se stanou základem pro vzájemný alignment.
3. Vyplnit mezery mezi překryvy.



Maximální unikátní shodný podřetězec

Z angl. Maximal Unique Match (MUM)

Maximální unikátní shodný podřetězec

MUM je společný podřetězec dvou genomů, takový, že jej nelze prodloužit ani z jedné strany a navíc je v obou sekvencích unikátní.

- Vždy stanovujeme nějakou limitní délku podřetězce MUM, obvykle 20-50 nt.
- Téměř všechny geny sdílené mezi ogranismy obsahují krátké společné podsekvence.

Maximální unikátní shoda - příklad

- GCTA je společný podřetězec, který je unikátní, ale není maximální.
- GCTAC je maximální unikátní společný podřetězec.
- TAC je podřetězec, který není unikátní.
- GCTACT není společný podřetězec.

```
Genome 1: ACGACTCAGCTACTGGTCAGCTATTACTTACCGC
Genome 2: ACTTCTCTGCTACGGTCAGCTATTCACTTACCGC
```

MUM - hrubou silou

- Na vstupu máme dvě sekvence $S[1..m_1]$ a $T[1..m_2]$.
- Řešení hrubou silou:

  for Pro každou pozici i v sekvenci S **do**
    for Pro každou pozici j v sekvenci T **do**
      Nalezněte nejdelší společný prefix P v $S[i..m_1]$ a $T[j..m_2]$
      Ověřte zda $|P| \geq d$ a zda je P unikátní v obou genomech.
    end for
  end for

- Toto řešení vede na $O(m_1 m_2)$ minimální časovou složitost.

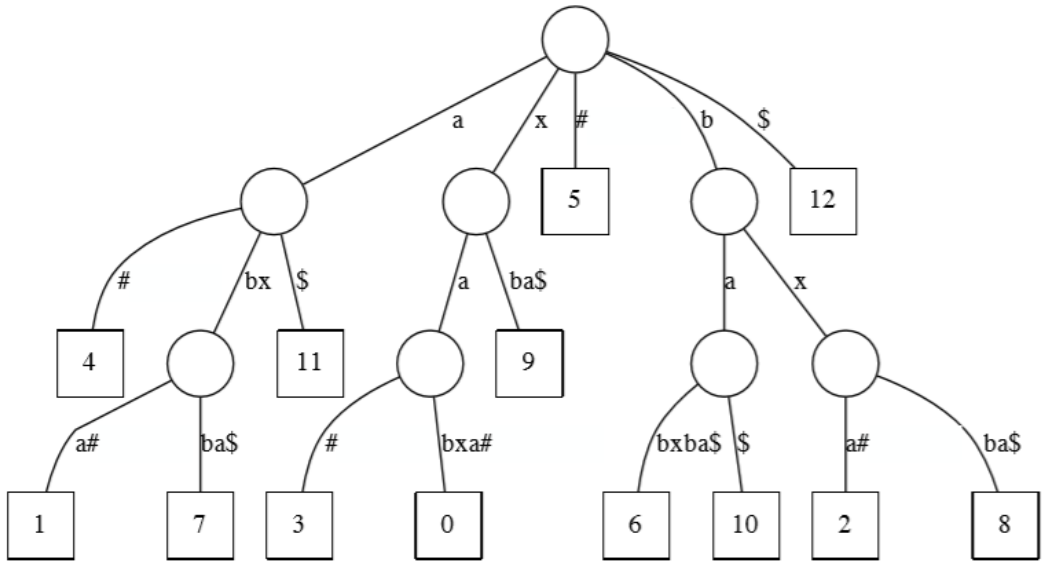
Konstrukce zobecněného suffixového stromu

- Zobecněný suffixový strom umožňuje vyhledávání ve více než jedné sekvenci.
- n sekvencí oddělíme pomocí n terminálních symbolů a vložíme do stromu.

$$X = xabxa$$

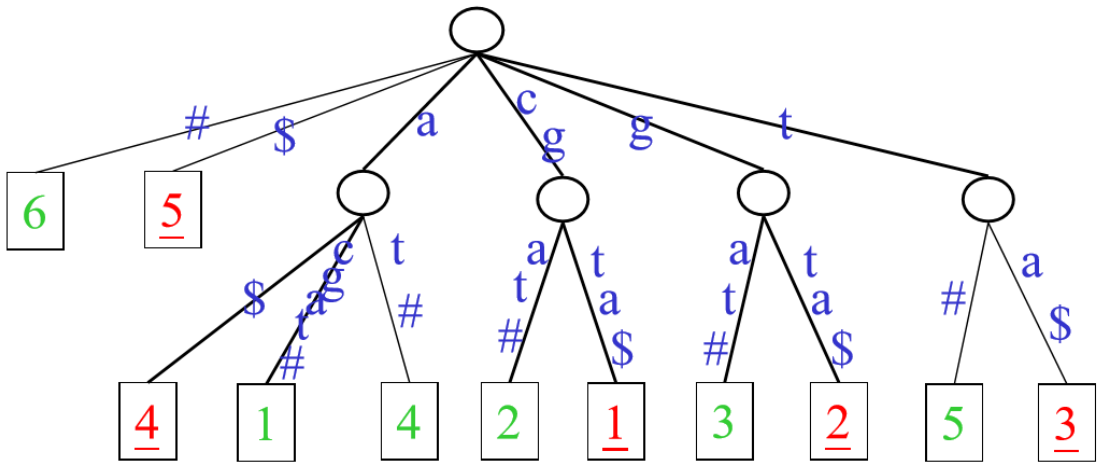
$$Y = babxba$$

$$X\#Y\$ = xabxa\#babxba\$$$



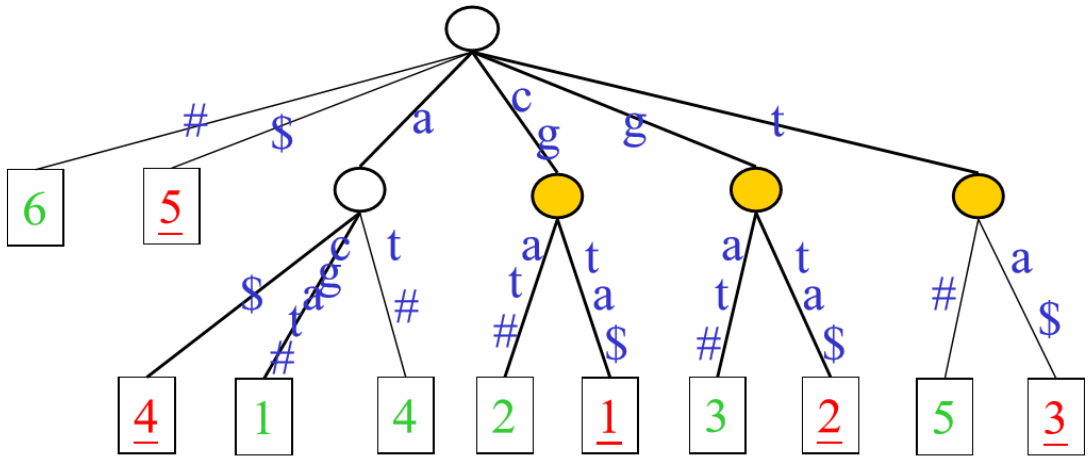
MUM - suffixovým stromem - příklad

- Uvažujme dvě sekvence $S = acgat\#$ a $T = cgta\$$.
- Krok 1: Sestavíme zobecněný suffixový strom pro S a T.



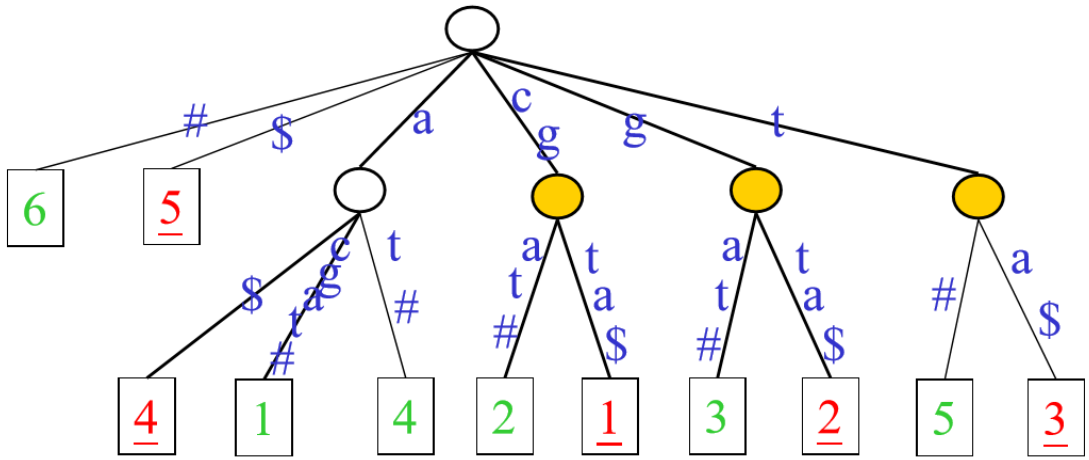
MUM - suffixovým stromem - příklad

- Krok 2: Označíme všechny vnitřní uzly, které mají právě dva potomky reprezentující suffixy S a T.
- Tím zajistíme unikátnost podřetězce.



MUM - suffixovým stromem - příklad

- Krok 3: Předpokládejme, že označenému uzlu odpovídají i-tý suffix v S a j-tý suffix v T. Ověříme, zda $S[i-1] \neq T[j-1]$, pokud jsou odlišné, pak se jedná o MUM.
- Zajistíme, že se jedná o maximální podřetězec.
- Pro $S = acgat\#$ a $T = cgta\$$ nám vyjde $\{cg, t\}$



Časová analýza

- Krok1: Sestavení suffixového stromu $O(m_1 + m_2)$.
- Krok2: Označení vnitřních uzlů $O(m_1 + m_2)$.
- Krok3: Ověření existence MUM $O(m_1 + m_2)$.
- Celkově tedy algoritmus s $O(m_1 + m_2)$ časovou složitostí.

Příklad člověk a myš

1. Překvapivě člověk a myš si jsou geneticky poměrně blízké organismy. Sdílejí celou řadu genů.
2. Tyto geny lze identifikovat pomocí MUM sekvencí. Téměř 100% známých párů genů jsou nalezeny jako MUM.
3. Na druhou stranu většina identifikovaných MUM sekvencí netvoří geny.

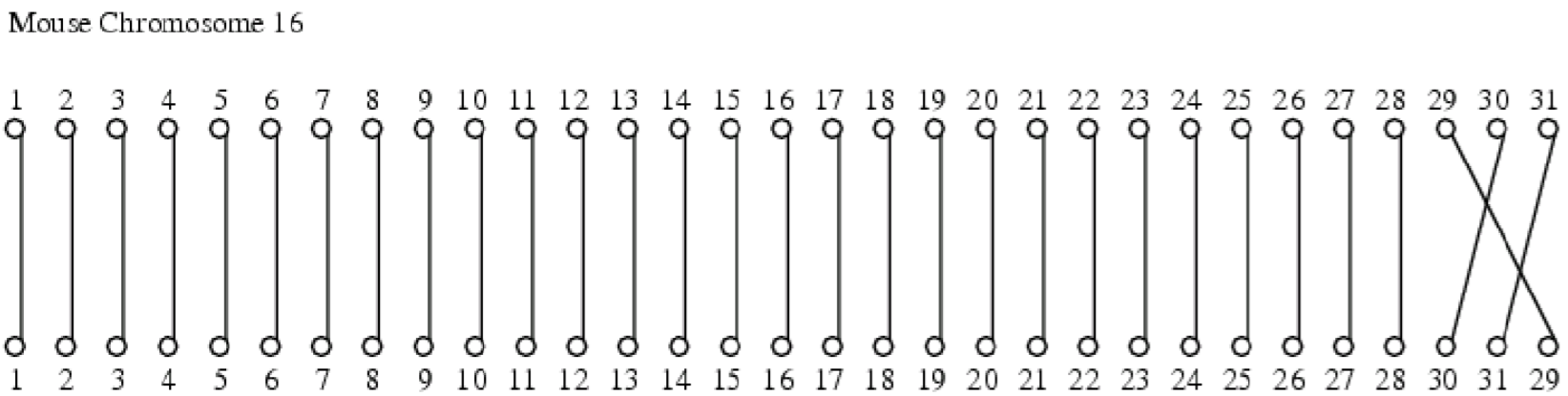
Mouse Chr No.	Human Chr No.	# of Published Gene Pairs
2	15	51
7	19	192
14	3	23
14	8	38
15	12	80
15	22	72
16	16	31
16	21	64
16	22	30
17	6	150
17	16	46
17	19	30
18	5	64
19	9	22
19	11	93

2

[2] https://www.ncbi.nlm.nih.gov/projects/homology/maps/human/chr1/

Identifikace ko-lineárních MUM

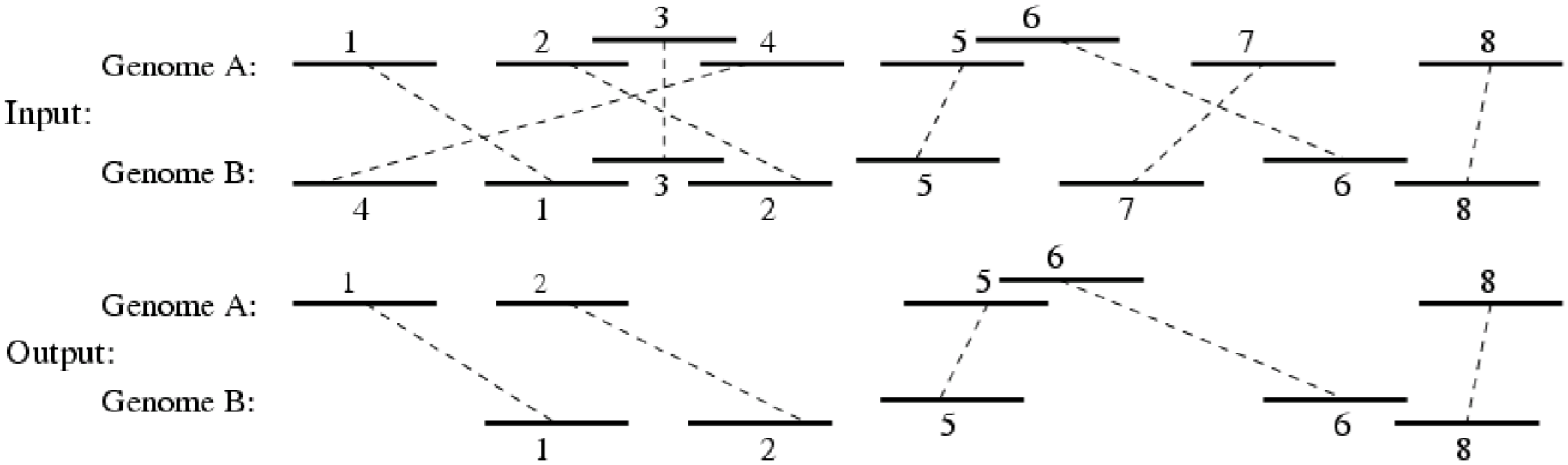
- Hypotéza: Dva blízké organismy by měly zachovávat pořadí evolučně neměnných genů.



(a) There are 31 conserved gene pairs found in Mouse chromosome 16 and Human chromosome 16. The genes are labeled according to the positions of the genes in the mouse chromosome from the 5' end. The corresponding genes in human are drawn according to their relative positions from the 5' end of human chromosome.

Identifikace ko-lineárních MUM

- Ze všech MUM sekvencí spočítáme nejdelší společnou podsekvenci (Longest Common Subsequence - LCS).
- Dále budeme uvažovat pouze MUM sekvence, které jsou součástí LCS.



12345678
41325768

→

12345678
41325768

Nejdelší společná podsekvence

LCS

- Předpokládejme existenci n MUM sekvencí.
- Vstup: mějme dvě sekvence $A[1..n] = a_1 a_2 \ldots a_n$ a $B[1..n] = b_1 b_2 \ldots b_n$. Obě sekvence jsou složeny z odlišných symbolů (čísel).
- Výstup: nejdelší společná podsekvence - LCS

Nejdelší společná podsekvence - dynamické programování

- Nechť $C_i[j]$ je délka nejdelší společné podsekvence $A[1..i]$ a $B[1..j]$.
- Nechť $\delta(i)$ je index symbolu v B, takový že $a_i = b_{\delta(i)}$.
- Intuitivně $C_i[0] = C_0[j] = 0$
- Další prvky se počítají pomocí rekurentní formule:

$$C_i[j] = max \begin{cases} C_{i-1}[j] \\ 1 + C_{i-1}[\delta(i) - 1] \quad \text{if } j > \delta(i) \end{cases}$$

- Nejdelší společný podřetězec je pak dán:

$$LCS(A, B) = C_n[n]$$

- Pomocí dynamického programování je tato úloha řešitelná v čase $O(n^2)$.

Nejdelší společná podsekvence - dynamické programování

- A[1..8]=12345678
- B[1..8]=41325768
  - $C_i[0]=0$ and $C_0[j]=0$

C	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0								
2	0								
3	0								
4	0								
5	0								
6	0								
7	0								
8	0								

Nejdelší společná podsekvence - dynamické programování

- A[1..8]=12345678
- B[1..8]=41325768

$$C_i[j] = \max\begin{cases} C_{i-1}[j] \\ 1 + C_{i-1}[\delta(i) - 1] \quad if\ j \geq \delta(i) \end{cases}$$

C	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	1					
3	0								
4	0								
5	0								
6	0								
7	0								
8	0								

$C_2[4]$=LCS(A[1..2],B[1..4]).

$C_2[4]$=max{$C_1[4]$, 1+$C_1[3]$}=2

Nejdelší společná podsekvence - dynamické programování

- A[1..8]=12345678
- B[1..8]=41325768

$$C_i[j] = \max\begin{cases} C_{i-1}[j] \\ 1 + C_{i-1}[\delta(i) - 1] \quad if\ j \geq \delta(i) \end{cases}$$

LCS(A,B)=$C_8[8]$=5.

C	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2
3	0	0	1	2	2	2	2	2	2
4	0	1	1	2	2	2	2	2	2
5	0	1	1	2	2	3	3	3	3
6	0	1	1	2	2	3	3	4	4
7	0	1	1	2	2	3	4	4	4
8	0	1	1	2	2	3	4	4	5

Vyplnění mezer mezi překryvy

- Použitím Smith-Watermanova algoritmu doplníme chybějící:
  - Inserce/Delece
  - Opakující se sekvence
  - Krátké zmutované oblasti.
  - SNPs.
- Program MUMmer1 - Nalezení MUM, sestavení LCS a doplnění mezer pomocí SW.

Příště uvidíte

- Principy algoritmů pro detekci variant.

DÄŻkuji za pozornost

Michal Vašinek

VŠB – Technická univerzita Ostrava

FEI/EA404

michal.vasinek@vsb.cz

26. října 2022

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY