Problémy s řetězci

$T \in \Sigma^m$ ... text

$P \in \Sigma^n$ ... hledané slovo

úkol: najdi výskyt / všechny výskyty P v T

Klasika: Aho-Corasick – sestrojím konečný automat z P a spustím ho na T

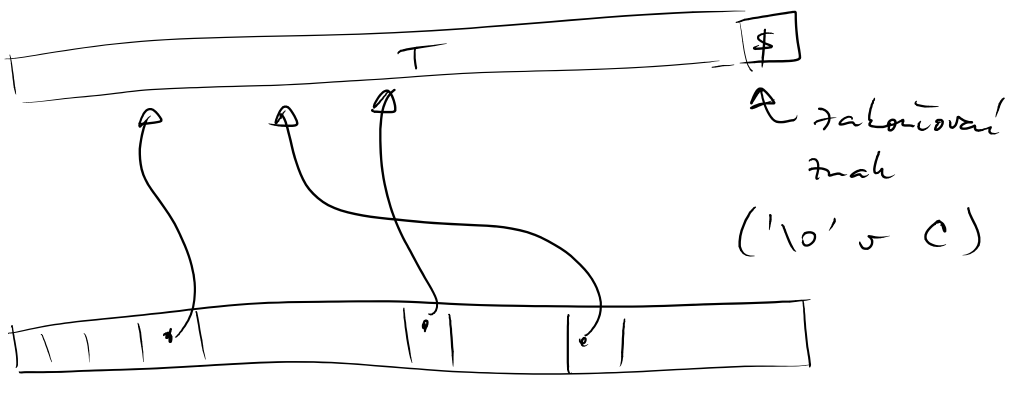
čas $\approx O(m+n)$.

chceme lépe – typicky $m \gg n$ a T je databáze, která se nemění

řešení: sufixové pole, sufixový strom

Sufixové pole

'\$' $<_{lex} \Sigma$



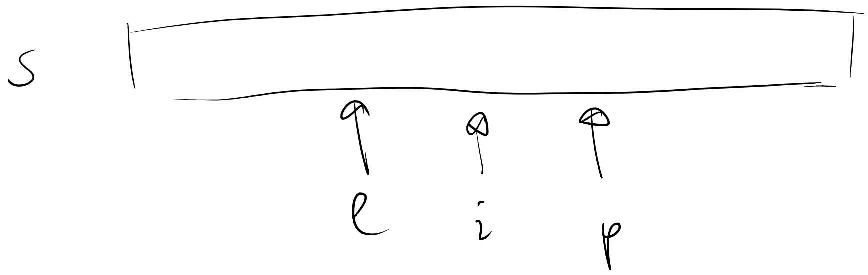
lexikograficky setříděné pole všech sufixů T, tj. pole řetězců $T[1..m]$, $T[2..m]$, ... $T[m..m]$

→ v poli lze binárně vyhledat řetězec P

– hledám lexikograficky nejmenší řetězec větší-rovný P.

· pokud se P shoduje s tímto řetězcem na prvních $|P|$ pozicích, nalezl jsem výskyt P v T, bezprostředně následující řetězce též se shodující s P jsou další výskyty P.

binární vyhledávání – čas $O(n \cdot \log m)$



$l \leftarrow 1,\ p \leftarrow m$ // hledá s přesností ± 1

dokud je $p > l+1$

  $i \leftarrow \left\lfloor \frac{l+p}{2} \right\rfloor$ (*)

  pokud $S[i] <_{lex} P$ pak $l \leftarrow i$

  jinak $p \leftarrow i$.

  return l

end.

... $\log m$ iterací, každé porovnání (*) stojí $O(m)$.

• lze zlepšit na $O(n + \log m)$

$u, v \in \Sigma^*$ $\quad lcp(u, v)$ = délka nejdelšího společného prefixu

předpokládejme, že známe zadarmo

$lcp(S(l), S(i))$ a $lcp(S(p), S(i))$

pro všechny trojice l, p, i (které se mohou vyskytnout během binárního vyhledávání)

↳ těch je pouze $O(m)$, viz níže

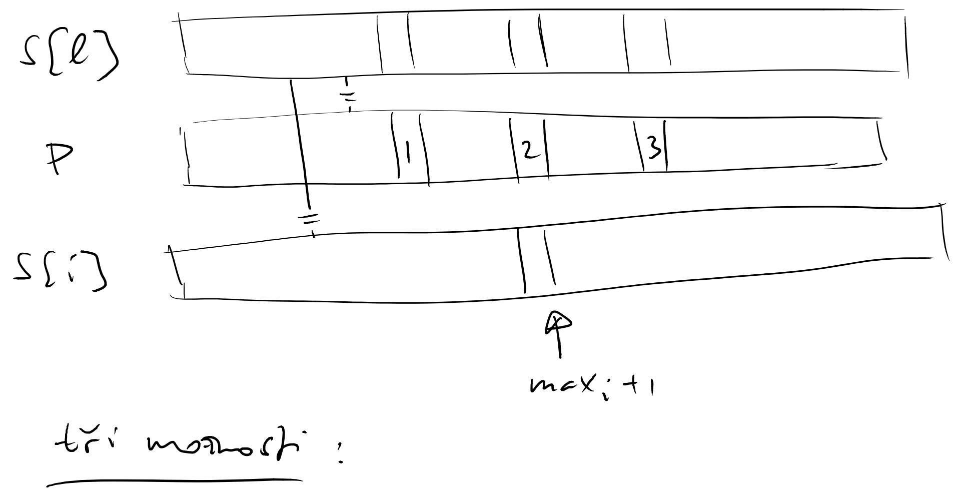
• během binárního vyhledávání si udržuji:

$max_l = lcp(S[l], P)$

$max_p = lcp(S[p], P)$

pokud $max_l > max_p$ porovnání (*) provedu následovně:

$max_i = lcp(S[l], S(i))$



<u>tři možnosti</u>:

3) $max_l > max_i \Rightarrow P <_{lex} S[i]$

neboť $S[l]$ a P se shodují
až do $\boxed{3}$ a $S[i]$ se od nich
liší v $max_i + 1$

$\Rightarrow p \leftarrow i, \; max_p \leftarrow max_i$

2) $max_l = max_i$: najdi první rozdílný znak
mezi $S[i]$ a P za pozicí $max_i \to r$
pokud $S[i][r] < P[r]$

pak $l \leftarrow i, \; max_l \leftarrow r-1$

jinak $p \leftarrow i, \; max_p \leftarrow r-1$

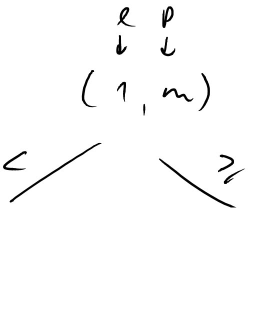
1) $max_l < max_i$: $l \leftarrow i, \; max_l \leftarrow max_i$

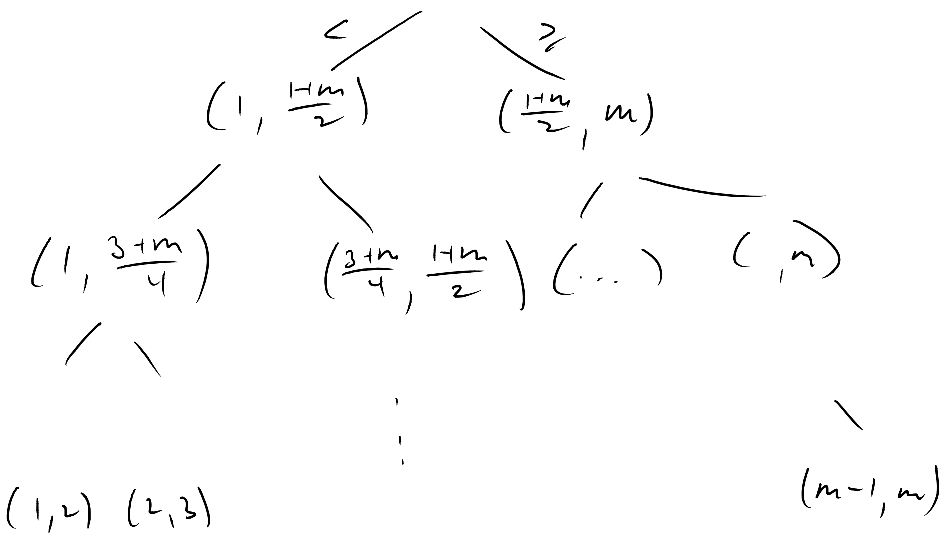
pokud $max_l \leq max_p$ postupuji symetricky
jako při $max_l > max_p$.

Pozn: pokud zjistím, že P je prefix $S[i]$, vyhledávání zastavím a našel jsem výskyt. Během binárního vyhledávání tak můžu předpokládat, že P není prefixem ani $S[l]$ ani $S[p]$.

Čas na $(*)$ je pak v součtu $O(n + \log m)$, neboť v P se posunuji pouze doprava

- předpoklad, že známe $lcp(S[l], S[i])$ a $lcp(S[p], S[i])$:
  - ty si předpočítáme, je jich $O(m)$, tudíž je lze skladovat v rozumné paměti

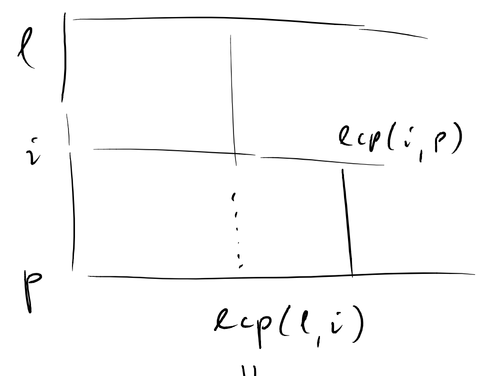




hranice l, p, i se posouvají jako při průchodu jednou z větví. Strom má $\leq m$ listů a $\leq m$ vnitřních uzlů $\Rightarrow$ $\leq 2m$ kombinací l, p, i.

potřebné hodnoty lcp lze spočítat odspodu nahoru, pokud známe $lcp(i, i+1)$ $\forall i$. Platí totiž

$$lcp(l,p) = \min_{i \in [l,p)} lcp(i, i+1)$$



$$\Downarrow$$

$$lcp(l,p) = \min(lcp(l,i), lcp(i,p))$$

- $lcp(i, i+1)$ $\forall i = 1..m-1$ lze spočítat v čase $O(m \lg m)$

- sufixové pole lze zkonstruovat v čase $O(m \cdot \log m)$:
  - varianta bucket-sortu [Karp-Miller-Rosenberg '72]
  - $\log m$ fází
    - v nulté fázi rozdělím řetězy do bucketů podle prvního symbolu.
    - ve fázi H, přerozdělím řetězy do bucketů podle prvních $2H$ symbolů.

      → na začátku fáze se řetězy v každém bucketu shodují v prvních H znacích na konci v prvních $2H$ symbolech (buckety se podrozdělí)

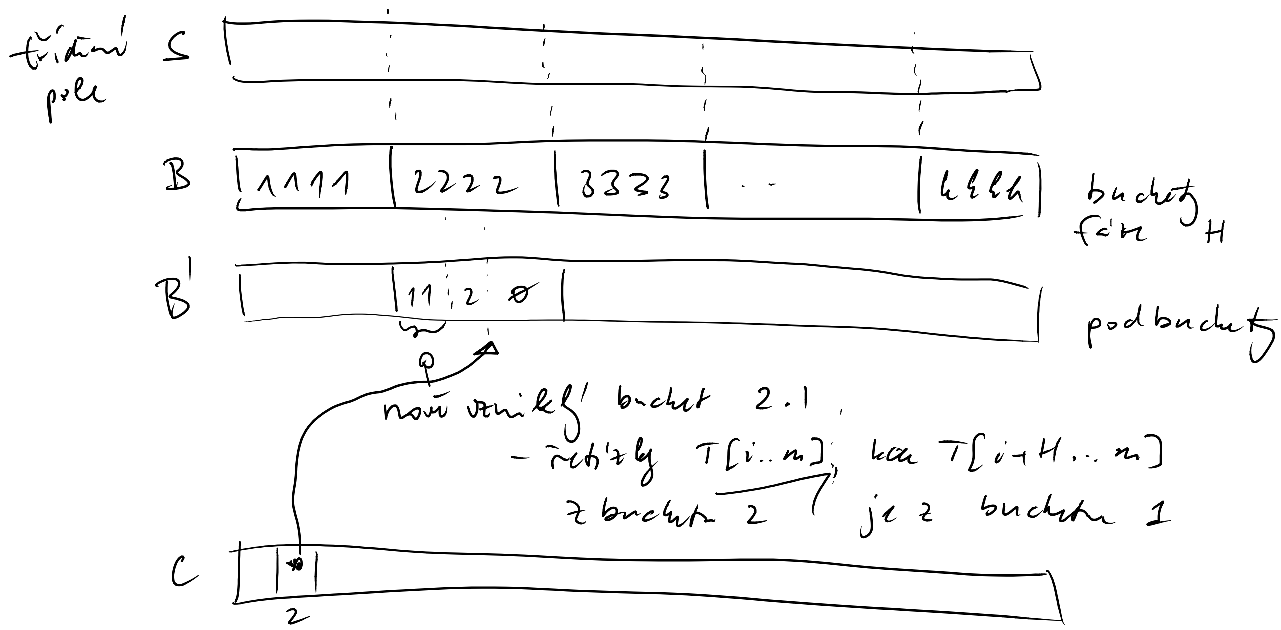
      → fáze trvá $O(m)$ čas.

      ↓

      využívám, že již znám roztřídění řetězců podle H-symbolů
      Pro porovnání $T[i,m]$ a $T[j,m]$ se stejnými prvními H symboly lze použít informaci o bucketech $T[i+H,m]$ a $T[j+H,m]$.

    - skutečná implementace bere buckety v lex-pořadí, v rámci bucketu jde řetězec $T[i..m]$ po řetězcu a přesune řetězec $T[i-H..m]$

(když $i-H>0$) v rámci jeho bucketu na začátek do nově odděleného bucketu [vše je uloženo v poli.]



ukazatel na ještě nezpracovaný záznam v daném bucketu, kde vznikl nový podbucket

– na konci fáze přečíslují vzniklé buckety od 1, 2, ...

Sufixový strom: trie sestavená z $T[1..n], T[2..n], \dots T[n..n]$

– koncepčně přímočařejší než sufixové pole, ale sufixové pole je kompaktnější, z hlediska času obě řešení jsou srovnatelná.

- mnoho různých algoritmů na konstrukci suf. polí & stromů.
- komprimovaná trie, kde se místo řetězců na hranách pouze odkazují na podřetězce v T potřebují pouze $O(n)$ místa.

Užití:

1) Hledání P v T

2) Zobecnění: Hledáme P v $T_1, T_2, \dots T_k$

3) nejdelší společný podřetězec (podinterval)

T_1 a T_2 ... $O(|T_1| + |T_2|)$

(ignorujeme "log"y)

a mnoho dalších

Samoupravující se seznamy

- Množina prvků $x_1, \dots x_n$
- chci reprezentovat spojovým seznamem
- prst. $p_1, p_2, \dots p_n$, kde p_i je pravděpodobnost, že Find bude hledat x_i.

Otázka: Optimální uspořádání prvků v seznamu?

$\longrightarrow$ podle klesající pravděpodobnosti p_i

Bůno: $p_1 \geq p_2 \geq p_3 \dots \geq p_n$

očekávaný čas operace Find (x_i)

$$E[T] = \sum_{i=1}^{n} \underline{i \cdot p_i}$$

s prst. p_i hledám x_i a musím tedy přelézt i prvků.

Problém: většinou neznám $p_1, p_2, \dots p_n$ dopředu