KMP, AHO-CORASICK

Petr Ryšavý

20. září 2023

Katedra počítačů, FEL, ČVUT

Knutt-Morris-Pratt

LINEÁRNÍ VYHLEDÁVÁNÍ V TEXTU

Hledání v textu

Definice *Nechť* T *a* p *jsou slova nad abecedou* Σ. *Nalezněte všechna* $i \in \{1, 2, \ldots, |T| - |p| + 1\}$, *že* $T_i^{i+|p|-1} = p$.

- Neformálně: naleznete všechny výskyty slova p v textu T.

Naivní přístup

- Porovnáváme každý znak textu s každým znakem patternu.
- Až $\mathcal{O}(|T| \cdot |p|)$.

Naivní přístup

- Porovnáváme každý znak textu s každým znakem patternu.
- Až $\mathcal{O}(|T| \cdot |p|)$.

- Chtěli bychom získat ideálně lineární čas běhu. Jak ho docílit?

KNUTT-MORRIS-PRATT (KMP) ALGORITMUS

Potenciál pro zlepšení

- Při každém posunutí patternu zapomeneme na všechnu práci, kterou jsme doposud provedli.
- Využijme úspěšných zarovnání, která jsme zatím provedli.

Prefix-suffix pole

- Máme zarovnaný prefix, známe tedy předcházející znaky. Hledejme konec tohoto zarovnání, který je prefixem p, abychom o tuto znalost nepřišli.
- Formálně pro každou pozici $i \in \{1, 2, \ldots, |p|\}$ hledáme suffix p_1^i, který je „proper" prefix p_1^i.
- Nikdy nejdeme v textu zpět

KMP Algoritmus

function KMP(T,p) **returns** All occurences of p in T
    lps $\leftarrow$ Longest-Prefix-Suffix(p)
    $i \leftarrow 0$, $j \leftarrow 0$
    while $i < |T|$ **do**
        if $p_j = T_i$ **then** ▷ Match on this position, extend
            $i \leftarrow i + 1, j \leftarrow j + 1$
            if $j = |p|$ **then** ▷ Match found!
                register match
                $j \leftarrow lps[j - 1]$
            end if
        else ▷ Mismatch on this position, shift the pattern
            if $j \neq 0$ **then** $j = lps[j - 1]$ ▷ Do not increase i
            else $i \leftarrow i + 1$
        end if
    end while
end function

PŘEDZPRACOVÁNÍ PATTERNU

Předzpracování patternu

- Použijme dynamické programování
- Využijme toho, že známe výsledek pro kratší pattern

Příklad

Předzpracování patternu

```
function LONGEST-PREFIX-SUFFIX(p) returns lps
    len ← 0
    lps[0] ← 0
    i ← 1                                          ▷ For iteration over p
    while i < |p| do
        if p_i = p_len then                        ▷ Match, extend the prefix
            len ← len + 1
            lps[i] ← len
            i ← i + 1
        else
            if len ≠ 0 then                        ▷ Some prefix matches, check shorter
                len ← lps[len − 1]                 ▷ Do not increase i
            else
                lps[i] = 0
                i ← i + 1
            end if
        end if
    end while
end function
```

ČAS BĚHU

Vyhledávání

- Na každý match se posuneme o jedna v textu a nikdy se nevracíme.
- Na každý mismatch posuneme pattern alespoň o jedna.
- Dohromady tedy nejhůře $\mathcal{O}(|T| + |T| - |p| + 1) = \mathcal{O}(|T|)$

Tvorba prefix-suffix pole

- Zpracováváme pattern délky $|p|$.
- Aby nedošlo k inkrementaci i, musela být nejdříve zvýšena hodnota len
- K inkrementaci i tedy nedojde nejvýše $|p|$-krát
- Dohromady tedy $\mathcal{O}(2|p|) = \mathcal{O}(|p|)$

Čas běhu celkem

$$\mathcal{O}(|T| + |p|)$$

Aho-Corasick

VYHLEDÁVÁNÍ MNOŽINY ŘETĚZCŮ

Vyhledávání množiny řetězců

Definice *Nechť* T *je slovo a* P *je množina slov nad abecedou* Σ*.*
Nalezněte všechna $i \in \{1, 2, \ldots, |T|\}$*, že* $T_i^{i+|p|-1} = p$ *pro nějaké* $p \in P$*.*

- Neformálně: naleznete všechny výskyty slov z množiny P v textu T.

První nápad - $|P|$ běhů KMP

- Nejpřímočařejší řešení je pustit $|P|$ krát KMP algoritmus
- Čas běhu je pak $\mathcal{O}(|P|(|p| + |T|))$
- Lze lépe?

Využití automatů - KMP jako automat?

Aho-Corasickův algoritmus

Jeden automat pro všechna slova

- Sestrojme trii z databáze slov

Jeden automat pro všechna slova

- Sestrojme trii z databáze slov
- Doplňme *failure links*
  - Podobně jako v KMP pole `lps` představují nejdelší proper prefix nějakého slova, který je suffixem toho co zatím máme zarovnané

Jeden automat pro všechna slova

- Sestrojme trii z databáze slov
- Doplňme *failure links*
  - Podobně jako v KMP pole `lps` představují nejdelší proper prefix nějakého slova, který je suffixem toho co zatím máme zarovnané
- Průchod je podobný průchodu KMP

Jeden automat pro všechna slova

- Sestrojme trii z databáze slov
- Doplňme *failure links*
  - Podobně jako v KMP pole `lps` představují nejdelší proper prefix nějakého slova, který je suffixem toho co zatím máme zarovnané
- Průchod je podobný průchodu KMP
- Potřebujeme *output links*

Čas běhu - vyhledávání

- Podobné jako v KMP - v každém kroku se posuneme o 1 v textu, nebo
- trie se posune o 1 vpravo
- Potřebujeme tedy $\mathcal{O}(|T| + z)$ + čas na sestrojení trie
- z je počet výskyt slov z P v T (může jich být až kvadraticky)

LINEÁRNÍ VYBUDOVÁNÍ LINKŮ V TRII

Fail links

- Postupujeme BFS od kořene dolů
- Potomci kořene v případě chyby odkazují na kořen
- Pro každý uzel u hledáme fail nodes jeho potomků v
  - využijeme znalosti fail node f uzlu u
  - hledáme hranu z f ohodnocenou stejně jako je hrana (u, v)
  - neexistuje-li, pokražujeme z fail nodu uzlu f rekurzivně

Čas konstrukce

- Zaměřme se na jedno slovo z trie
- Výpočet jeho fail nodes je obdobný výpočtu fail nodes v KMP
- Nelze jít vícekrát zpět než jsme měli hran doprava v trii
- Celkem tedy $\mathcal{O}(\sum_{p \in P} |p|)$

Poslední krok - output links

- Potřebujeme vyřešit situace, kdy máme slova jako $\{\text{he}, \text{she}\}$
- Při tvorbě fail links ověříme, zda je je fail link na koncový vrchol f
- Pokud ano, nastavme output link na f
- Jinak nastavme output link na output link f
- Neovlivníme čas běhu

Reference - KMP

- Pěkná implementace (a zdroj pro pseudokód)
  https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/
- https://www.youtube.com/watch?v=BXCEFAzhxGY

Reference - Aho-Corasick

- https://www.youtube.com/watch?v=ePafMI_rSJg, https://www.youtube.com/watch?v=qPyhPXPl3T4, https://www.youtube.com/watch?v=IcXimoT_YXA
- Přednáška ze Stanfordu: http://web.stanford.edu/class/archive/cs/cs166/cs166.1166/lectures/02/Small02.pdf

Děkuji za pozornost.

Čas na otázky!