

Stringológia

Mišo Forišek

<misof@mfnotes.ksp.sk>
 Department of Computer Science
 Faculty of Mathematics, Physics, and Informatics
 Comenius University, Bratislava, Slovakia

jeseň 2003

1 KMP: Knuth-Morris-Pratt

Daný je pattern P dĺžky p , text T dĺžky t .

1.1 Triviálne riešenie

V čase $O(pt)$ všetky možnosti. V praxi $p \sim 10^3$, $t \sim 10^6$.

1.2 ± 1 chyby

Algoritmy kritické na ± 1 chyby: napr. qsort, bsearch, aj KMP. Riešenie: presne sformulovaný invariant, jasná predstava, čo ktorá hodnota znamená.

Príklad: bsearch. Problém: koniec, napr. zacyklíme sa pre $r = l + 1$. Riešenie: Pridáme $A[N+1] = \infty$, položíme $l = 1$, $r = N + 1$. Invariant: $A[l] \leq x < A[r]$. Ak $r = l + 1$, našli sme, ak nie, je $|r - l| \geq 2$, zoberieme stred intervalu, podľa výsledku sa vnoríme, tým sa interval určite skrúti – nezacyklíme sa.

Pri KMP skúsime začať názornou predstavou, aby sme vedeli, čo nami spočítané hodnoty predstavujú.

1.3 Nedeterministický automat

Predstavme si takúto hračku: Máme $p+1$ miestností (očíslovaných 0 až p) v rade zľava doprava, sú pospájané dverami. Na dverách medzi miestnosťami $i-1$ a i je písmeno $P[i]$. Každú sekundu prečítame jedno písmeno x textu T , čím sa všetky dvere označené x otvoria a ostatné sa zavrú.

Hráme nasledujúcu hru: V miestnosti 0 je lemming. V ľubovoľnom okamihu mu môžeme povedať, nech sa vyberie (rýchlosťou 1 miestnosť za sekundu) doprava. Ak musí zastať, zahynie a prehrali sme, ak sa mu podarí prejsť až na koniec, vyhrali sme.

Zjavne hra sa dá vyhrať iff P sa nachádza v T . Vševedúca víla Amálka by nám vedela dokázať, že P je v T tak, že vyhrá hru. Ale čo my chudáci, čo nevieme kúzlom zistiť, kde sa P v T nachádza?

1.4 Deterministický automat

Budeme podvádzať. Povieme lemmingovi, nech dovedie celú rodinu, očísľujeme ich a bez ostychu každú sekundu jedného pošleme dnu. Čísla tých, čo prejdú, nám určia, kde všade sa P v T nachádza. (Aby sa nám nehromadili v miestnosti p , nech aj tam pekne po sekunde hynú, nie je ich škoda.)

Pozrime sa po niekoľkých sekundách na situáciu. V niektorých miestnostiach sú lemmingovia, v niektorých nie. Koľko je možných rozostavení? Zdalo by sa, že 2^p – v každej z miestností 1 až p buď je alebo nie je lemming. ALE. Všimnime si miestnosť, kde je najpravejší lemming, nech je to k . Potom ale vieme, akých bolo doteraz posledných $k - 1$ písmen T a tým je jednoznačne určené, kde (v miestnostiach naľavo od k) sú a kde nie sú lemmingovia.

(V miestnosti i je lemming iff string $P[1..i]$ je sufixom stringu $P[1..j]$.)

Explicitná konštrukcia automatu v $O(p^2|\Sigma|)$, vyhľadávame v $O(t)$ – máme prvý rýchly vyhľadávací algoritmus.

1.5 Finta

Nebudeme sledovať všetkých lemmingov, iba najpravejšieho. Potrebujeme sa ale vedieť presunúť na nasledujúceho, keď nám najpravejší zakape. Preto si pre každú miestnosť j spočítame hodnotu $h(j)$: „ak je v j lemming, kde smerom doľava je najbližší ďalší?“

Zjavne $h(j) = \max\{i \mid i < j \wedge P[1..i] \text{ je sufixom } P[1..j]\}$.

Hľadáme teda dĺžku najdlhšieho reťazca, ktorý je (vlastným) prefixom aj sufixom $P[1..j]$.

1.6 Počítame $h(j)$

Zjavne $h(1) = 0$. Nech vieme $h(1)$ až $h(j)$, spočítajme $h(j + 1)$. Najpravejší lemming práve prešiel z j do $j + 1$. Keď ešte stál na j , mal najbližšieho suseda na políčku $h(j)$. Ak aj tento mohol prejsť doprava, sme done ($h(j + 1) = h(j) + 1$). Inak sused zakapal, potrebujeme nájsť ďalšieho kandidáta. Nič ľahšie. Ďalším kandidátom je samozrejme susedov sused. Ak aj ten zakape, tak jeho sused, atď. Všetkých už vieme hľadať v $O(1)$.

Príklad: *ababacababab*. Lemmingovia stoja na 1, 3, 5, 11. Vieme už, že $h(11) = 5$, $h(5) = 3$, $h(3) = 1$, atď. Keď prečítame *b*: lemming z 11 prejde do 12, jeho sused z 5 zakape, toho sused z 3 ide na 4 a je novým susedom lemminga z 12. Preto $h(12) = 4$. (Už nás nezaujímá, čo robí lemming z miestnosti 1.)

1.7 Je to lineárne?

Ukazujme počas celého rátania $h(j)$ prstom na druhého najpravejšieho lemminga. V každom kroku algoritmu prst posúvame – ak lemming zakape, ide prst doľava, ak nie, zväčšíme j a prst

ide o 1 doprava. Doprava posunieme prst dokopy o N , potom ale doľava ho môžeme posunúť najviac N -krát \Rightarrow náš algoritmus spraví najviac $2N$ krokov.

1.8 Ako vyhľadávame?

Presne rovnako ako sme rátali $h(j)$. Sledujeme, kde je najpravejší lemming. Ak na aktuálne písmeno môže ísť, ide a čítame ďalšie, inak zakape a presunieme sa na najbližšieho suseda.

Rovnaká argumentácia ako vyššie, že je to lineárne.

1.9 Klasická definícia KMP

Uvedomme si, že spočítané hodnoty sú dĺžky najdlhšieho sufixu, ktorý je aj prefixom dotyčnej časti. Posúvanie patternu, príklad na $T = xyabcxabcxabcex$, $P = abcxabce$.

1.10 Použitie

Vyhľadávanie.

Cvičenie: Daný je písmenkový strom, nájdite všetky výskyty patternu vo všetkých koreň-list cestách.

Periódá. Dá sa def. tak, že je to najmenší posun, po ktorom string pasuje sám na seba. Vtedy ale vlastne nejaký sufix T pasuje na prefix T . Preto dĺžka periódy je $N - h(N)$.

Rotácie. Nech $|u| = |v|$, potom u je rotácia v iff u sa nachádza v vv .

Cvičenie: Navrhňte algoritmus na nájdenie patternu na kružnici.

2 Aho-Corasick

Máme množinu patternov \mathcal{P} s celkovou dĺžkou m , text T dĺžky t

2.1 Opäť nedeterministický automat

Písmenkový strom (trie, keyword tree). Môžeme sa naň opäť dívať ako na budovu s lemmingami – z niektorých miestností síce vedie viac dvier, ale naraz budú otvorené max. jedny.

Príklad: trie pre potato, tattoo, other, theater.

2.2 Opäť deterministický automat

Osvedčená metóda – cpeme veľa lemmingov, sledujeme najhlbšieho. Potrebujeme pointer na najbližšieho „plytšieho“. (Uvedomiť si, že v každej hĺbke je najviac jeden.)

Pointre spočítame prehľadávaním stromu do šírky analogicky ako u KMP.

2.3 Vyhľadávanie

Problém: ak nekončí keyword v liste, môžeme ho preskočiť.

Príklad: potato, tattoo, ta, at, po prečítaní potat failure linky preskočia ta.

Jednoduchší príklad: abacc, ba, pattern abacx.

Presnejšia definícia problému: ak jeden pattern je substring iného. (Neskôr idúci lemming zakape skôr, ako si ho všimneme.)

Patch-Gusfield: Všimnime si nasledovnú vec. Sme v nejakom vrchole. Ak odtiaľto vieme po failure linkoch prísť do vrcholu zodp. niektorému patternu, tak ten pattern práve skončil. A naopak, ak práve skončil pattern, buď sme v príslušnom vrchole, alebo sa doň vieme dostať po failure linkoch.

Patch-ja: V každom kroku môže na niektorý pattern prísť aj jeden z „vyšších“ lemmingov, musíme zisťovať, či to nenastalo.

2.4 Efektívny patch

Ako to efektívne implementovať? (Lepšie ako pozerať vždy všetkých lemmingov?) Okrem failure linkov budeme mať aj output linky, ktoré vedú len po tých lemmingoch, ktorí sú na konci patternu.

Výsledný čas je $O(m)$ na predspracovanie, $O(t+v)$ hľadanie (kde v je počet výskytov patternov).

2.5 Použitie

Osemsmerovka. K slovám zostrojíme keyword tree, potom už len preliezame a škrtáme.

2D pattern matching. Nech sú všetky riadky patternu rôzne. Zostrojíme keyword tree, pobe-
háme „text“, na každom políčku, kde začína niektorý (najviac jeden!) riadok, si zapíšeme
jeho číslo. Teraz len potrebujeme nájsť v niektorom stĺpci postupnosť $1, 2, \dots, N$.

Ak sú niektoré riadky rovnaké (zistíme pri konštrukcii keyword tree), všetky dostanú
to isté číslo, potom v stĺpcoch potrebujeme nájsť napr. $1, 2, 3, 1, 5, 3, 2, 3$, hľadáme (ako
ináč) pomocou KMP.

3 Suffix array

Pokec o dôležitosti sufixov (keď sa vyznáme v sufixoch, vyznáme sa vlastne vo všetkých sub-
stringoch).

SA: pre každý sufix si pamätáme jeho poradie po utriedení všetkých sufixov.

3.1 Konštrukcia

Triviálne v $O(t^2 \log t)$ – `qsort()` a `strcmp()`.

Menej triviálne v $O(t^2)$ – radix sort podľa jednotlivých písmeniek.

Fintou v $O(t \log t)$ – keď máme utriedené sufixy podľa prvých 2^k znakov, vieme ich utriediť podľa prvých 2^{k+1} – ak sú dva zatiaľ rovnaké, potrebujeme porovnať ich ďalších 2^k znakov, tie ale už máme (pre iné dva sufixy) porovnané.

Lineárne zo sufixového stromu.

3.2 Vyhľadávanie

Binary search na zotriedených sufixoch, priamočiara implementácia v $O(p \log t)$.

4 Suffixové stromy

Definícia: zobrať suffix trie, kontrakcia vrcholov s outdegree 1.

Presnejšie: Ako suffixový strom definujem to, čo Gusfield volá implicit suffix tree, t.j. nepožadujem, aby sufixy končili v listoch.

Veľkosť suffixového stromu je lineárna, lebo má najviac N listov.

Z každého vrcholu každým písmenom začína max. 1 hrana.

4.1 Triviálny kvadratický algoritmus

Postupne vkladám sufixy, potom skontražujem a mám.

4.2 Menej triviálny, ale kubický :)

String spracúvame po písmenkách. Keď nám príde ďalšie, potrebujeme:

- všetky už vložené sufixy o 1 predĺžiť
- vložiť jeden nový sufix dĺžky 1 (triviálne)

Pri predlžovaní sufixu α na αx môžu nastať 3 prípady:

Case 1. Cesta pre α končí v liste – len tam pridáme x .

Case 2. Z miesta, kde sa dostaneme na α sa na x nedá pokračovať. Vtedy pridáme do stromu [vrchol a] novú hranu s písmenom x .

Case 3. Cesta pre αx už v strome existuje – nerobíme nič.

Takže: $\forall i$ od 1 do $N - 1$ (
 prečítame $(i + 1)$. písmeno
 $\forall j$ od 1 do $i + 1$ (
 nájdeme kde v strome končí $S[j..i]$
 pridáme znak $S[i + 1]$
)
)

Fáza „nájdeme, kde v strome končí“ trvá $O(i)$, nasčítava sa nám to na $O(N^3)$.

4.3 . . . kvadratický . . .

Možnosť a) Pre každý sufix si pamätáme najbližší vrchol stromu k jeho koncu. Problém: Hranu od tohto vrcholu ku koncu sufixu nám môže niečo rozdeliť, čím vznikne nový bližší vrchol. Riešenie: Keď zliezame ku koncu, vždy si posúvame aj tento pointer po prípadných nových vrcholoch. Nový problém: Akú to má preboha zložitosť?

Možnosť b) Suffix linky.

Suffix link vždy existuje: ak existuje vrchol aw , tak existuje aj w (alebo aw práve vznikol a w vznikne pri pridávaní o 1 kratšieho sufixu).

Lozenie po nich: Sme na konci sufixu. Hore do najbližšieho vrcholu, po suffix linku, dole po (aj niekoľkých) hranách.

Skip/count trick: Keď ideme dole po tých hranách, stačí sa rozhodovať podľa prvého písmena, totiž tá cesta tam určite je, prvé písmeno jednoznačne určí, ktorú hranu vybrať.

Prečo je to kvadratické? Vrcholová hĺbka $s(v) \geq$ vrcholová hĺbka v mínus 1. (Všetky vrcholy na ceste do $s(v)$ sú aj na ceste do v , môže ich byť aj viac – vid' $issi \rightarrow ssi$.) Tzn. pri každom lezení ideme max. o 2 hore a potom dole. Celkovo ideme hore (teda aj dole) najviac o $2N$.

4.4 . . . lineárny!

Labely hrán aby bola skutočne lineárna veľkosť.

Case 3 končí fázu.

Case 1 robíme implicitne – label ∞ .

Všetko čo boli casy 1, 2, budú v budúcej fáze casy 1 – netreba sa nimi zaoberať.

4.5 Úplný

Ako posledný znak vložíme zarážku \$, teraz každému sufixu zodpovedá list.

4.6 A zovšeobecnený

Keď máme slová w_i , vytvoríme slovo $w_1\$1 \dots w_k\k , preň suffix tree, každú hranu do listu urežeme tak, aby končila pri prvom endmarkri.

4.7 Použitie

Cvičenie: Redukcia pamäťových nárokov Aho-Corasicka na lineárne.

Vyhľadávanie viacerých vzoriek.

Najdlhší (k -krát sa) opakujúci substring.

Najdlhší spoločný substring dvoch stringov. (Ide aj pre k , buď v $O(km)$ si po strome posielame, ktoré stringy majú list v danom podstrome alebo použijeme LCA.)

Cvičenie: Pre daných k stringov zistite, či je niektorý z nich substring iného.

Cvičenie: Nájdite najdlhšie v také, že dané slovo w začína vv .

Cvičenie: Maximálny repeat v reťazci w je trojica (i, j, d) taká, že $w[i..(i+d-1)] = w[j..(j+d-1)]$ a $w[i+d] \neq w[j+d]$. Dokážte, že maximálnych repeatov je najviac $|w|$.

Cvičenie: Nájdite najdlhší nepretínajúci sa repeat.

Circular stringy.

Palindrómy (s LCA).

Cvičenie: Zostrojte zo sufixového stromu sufixové pole v lineárnom čase.

5 Palindrómy

Kubicky: poviem si začiatok, koniec, overím.

Kvadraticky: poviem si stred, idem kým sa dá.

Neprijemnosť: dva typy stredov (na písmene, medzi písmenami). Riešenie: medzi každé dve písmená vložiť „medzeru“, už nás trápia len palindrómy so stredom nad písmenom.

Myšlienka k lineárnemu riešeniu: Keď rátame dĺžku palindrómu s daným stredom, využívať už spočítané info.

Príklad: $abcba\underline{x}abcba\underline{x}b$. Spracúvame podčiarknuté c . Už ale vieme, že v podčiarknutom x má stred palindróm polomeru 6. To ale znamená, že reťazec $T[1..5]$ je reverzom reťazca $T[7..11]$. Už vieme, že na políčku 3 má stred palindróm s polomerom 3. Tie isté písmená sú ale aj okolo práve spracúvaného políčka 9 – aj tu je palindróm polomeru 3 (možno aj dlhší).

Lineárne: Pre každé spracované písmeno si pamätáme polomer palindrómu, ktorý má na ňom stred. Okrem toho si pamätáme najďalej doprava siahajúci palindróm P , ktorý sme našli. Keď čítame písmeno, najskôr sa pomocou tohto palindrómu pozrieme, čo vieme o jeho okolí. Ak je v ňom palindróm, ktorý siaha až po (za) kraj P , ďalšie znaky overujeme lineárne, kým nenastane mismatch.

Prečo je to lineárne: každé porovnanie znakov nám posunie pravý kraj P .

6 Suffix arrays revisited

6.1 Upgradujeme vyhľadávanie

Nech L a R sú aktuálne hraničné sufixy. Nech už vieme, že L matchuje P na l a R na r znakoch. Označme $k = \text{lcp}(L, R) = \min(l, r)$. Potom hľadaný string stačí porovnávať od pozície $k + 1$.

Updatovať si tieto hodnoty pomôže, ale nestačí.

Majme l, r . Už sme určite videli $\max(l, r)$ znakov z P a už ich nechceme vidieť znova. Ako na to? Ak $l = r$, easy. Nech BUNV $l > r$, označme stredný sufix M .

Ak $\text{lcp}(L, M) > l$, keďže $L < P$, aj $M < P$, preto $L \leftarrow M$ a ideme ďalej.

Ak $\text{lcp}(L, M) < l$, tak $P < M$ (líšia sa na $\text{lcp}(L, M) + 1$. znaku), teda $R \leftarrow M$ a ideme ďalej.

Ak $\text{lcp}(L, M) = l$, ideme po M od $l + 1$. znaku kým nenájdeme mismatch alebo nie sme done.

(V poslednom prípade sa síce pozrieme na 1 znak P , ktorý sme už videli, ale určite odlezieme ďalej.)

6.2 Odkiaľ vezmeme lcp?

Pri binary searchi je len $O(N)$ dvojíc (L, R) , na ktoré môžeme teoreticky naraziť. (Nakresliť strom pre 8.) Pre $SA[R] = SA[L] + 1$ (teda susedné) vieme lcp zo suffix tree, zvyšné zistíme preliezaním BS stromu ako minimum v danom podstromu (t.j. minimum synov).

Disclaimer. Tieto poznámky môžete voľne používať na ľubovoľné nekomerčné účely. Na akékoľvek komerčné využitie je potrebný súhlas autora. Ak v mojich poznámkach objavíte nejakú chybu, prípadne ich nejakým spôsobom viete doplniť, budem rád, ak mi dáte vedieť.

Pre potreby prípadného citovania má tento kus poznámok evidenčné číslo MF-0002.