Sufixový strom

Motivácia #1: Ako vyhľadávať vzorky v texte? Najjednoduchšie riešenie je, samozrejme, *triviálne vyhľadávanie*: porovnáme vzorku so všetkými pozíciami v texte, čo trvá $O(m \times n)$, kde n je dĺžka textu a m dĺžka vzorky.

Klasické algoritmy ako Knuth-Morris-Prattov dokážu vzorku predspracovať v čase $O(m)$ a následne prejsť text v lineárnom čase $O(n)$. Praktické algoritmy typu Boyer-Moore či jeho varianty často fungujú ešte lepšie, najmä na prirodzených textoch.

Tieto prístupy však majú spoločnú jednu vec: *predspracúvajú vzorku* a hľadajú ju v pôvodnom texte. To je ideálne, ak vzorka zostáva rovnaká a text sa mení.

Čo ak sme v opačnej situácii? Text T je obrovský a fixný, a chceme v ňom vyhľadávať veľa rôznych vzoriek. Predstavte si napríklad Wikipédiu, ktorá obsahuje desiatky gigabajtov textu a v ktorej chceme rýchlo nájsť dané slová alebo frázy. Alebo ľudskú DNA – približne 3 miliardy „báz“ (`A`, `C`, `G`, `T`) – a chceme v nej nájsť gén dlhý tisíce až desaťtisíce znakov.

Dokázali by sme si vopred predspracovať *text* tak, aby sme ho pri vyhľadávaní nemuseli prechádzať celý?

Ukážeme si, že s pomocou sufixových stromov dokážeme (po úvodnom predspracovaní) vyhľadávať v čase $O(m)$ (!), teda v čase úmernom dĺžke vzorky, nie celého textu.

Motivácia #2: najdlhší spoločný podreťazec. Problém najdlhšieho spoločného podreťazca dvoch reťazcov bol dlhé roky považovaný za ťažký: najlepšie známe riešenia mali zložitosť $O(n \log n)$ a existovali dohady, že lineárny čas je možno nemožný. V roku 1970 Donald Knuth dokonca vyslovil hypotézu, že lineárny algoritmus neexistuje.

Ukážeme si, že ak poznáme sufixové stromy, je táto úloha úplne jednoduchá a elegantné riešenie s lineárnou zložitosťou sa objaví takmer samo.

Keďže oba problémy, o ktorých sme sa zmienili, sú pomerne zložité, skúsme sa najskôr pozrieť na dve jednoduchšie úlohy. Predstavme si, že máme dané texty

$$T_1, \ldots, T_d.$$

Chceme si ich predspracovať tak, aby sme dokázali riešiť nasledujúce otázky:

- *Vyhľadávanie prefixov.* Pre danú vzorku P chceme rýchlo zistiť, ktoré z textov T_i začínajú práve na P.
- *Najdlhší spoločný začiatok.* Chceme nájsť dvojicu textov, ktoré majú najdlhší spoločný prefix.

Tieto dve úlohy sú výrazne jednoduchšie než pôvodné problémy, no ich riešenia nás navedú na cestu k sufixovým stromom a poliam. Skúste sa nad nimi najprv zamyslieť – tieto problémy dokážete vyriešiť aj sami. Až potom pokračujte v čítaní.

Vyhľadávanie prefixov. Ako rýchlo nájsť všetky texty začínajúce na vzorku P? Začnime od najjednoduchšieho riešenia:

- *Triviálne riešenie.* Bez akéhokoľvek predspracovania stačí porovnať vzorku so všetkými textami. To trvá

  $$O(m \times d),$$

  kde m je dĺžka vzorky a d je počet textov.

- *Triedenie + binárne vyhľadávanie.* Ak si texty vopred zotriedime lexikograficky, môžeme následne hľadať pomocou binárneho vyhľadávania. Pri jednom porovnaní dvoch reťazcov prejdeme najviac m znakov, no množinu prehľadávaných textov zmenšíme na polovicu, takže výsledná zložitosť je

  $$O(m \times \log d).$$

- *Písmenkový strom (trie).*[1] Ak si texty uložíme do prefixového stromu, pri vyhľadávaní jednoducho zídeme po ceste zodpovedajúcej vzorke P. Ak cesta existuje, všetky listy v podstrome pod touto cestou predstavujú texty začínajúce na P. Dĺžka zostupu je práve m, takže čas hľadania je

  $$O(m).$$

Najdlhší spoločný začiatok. Pre jednoduchosť predpokladajme, že každý text má dĺžku práve n. Ako nájsť dve slová s najdlhším spoločným prefixom?

- *Triviálne riešenie.* Môžeme porovnať každú dvojicu textov. Takých dvojíc je d^2 a porovnanie dvoch reťazcov trvá v najhoršom prípade n, čiže celková zložitosť je

  $$O(d^2 \times n).$$

- *Triedenie.* Ak si texty najprv zotriedime lexikograficky, stačí porovnať len dvojice, ktoré stoja v utriedenom poradí vedľa seba. Každý text teda porovnávame s najviac dvoma susedmi, takže riešenie beží v čase

  $$O(d \times n).$$

- *Písmenkový strom (trie).* Keď texty vložíme do prefixového stromu, každý text zodpovedá jednej ceste z koreňa do niektorého listu. Dva texty majú spoločný začiatok presne tam, kde sa ich cesty zhodujú. Najdlhší spoločný prefix medzi ľubovoľnými dvoma textami teda nájdeme tak, že v strome hľadáme *najhlbší vrchol, ktorý má aspoň dvoch potomkov.*

  Tento vrchol predstavuje najdlhšiu cestu od koreňa, ktorá je spoločná pre aspoň dva texty, a jeho hĺbka je dĺžkou najdlhšieho spoločného začiatku. Vrchol ľahko nájdeme v lineárnom čase (od veľkosti stromu).

[1] Tieto stromy sú známe pod veľa rôznymi menami: v angličtine trie (zo slova re*trie*val), prefixový strom, písmenkový strom, lexikografický strom...

Štruktúra sufixového stromu

Ako sme videli, písmenkový strom je mimoriadne vhodný na úlohy týkajúce sa prefixov. Ponúka sa teda na prvé počutie šialená myšlienka:

> Každý podreťazec je predsa prefixom nejakého sufixu.

Čo keby sme teda vyrobili písmenkový strom, do ktorého vložíme *všetky sufixy* nášho textu?

Vezmime si napríklad text `MISSISSIPPI$`, ktorý má presne 12 neprázdnych sufixov:

$$\texttt{\$}, \texttt{I\$}, \texttt{PI\$}, \texttt{PPI\$}, \texttt{IPPI\$}, \ldots, \texttt{ISSISSIPPI\$}, \texttt{MISSISSIPPI\$}.$$

Ak zostavíme písmenkový strom obsahujúci všetky tieto reťazce, dostaneme strom zobrazený na obr. 1.

Na obr. 2 a 3 sú dva väčšie príklady: sufixové stromy pre text

$$\texttt{SHE␣SELLS␣SEASHELLS␣BY␣THE␣SEASHORE\$}$$

a pre úsek DNA sekvencie

$$\texttt{ATAGACCGCCATTACATAGATGAGTATAGAGACT\$}.$$

Na koniec textu vždy pridáme špeciálny symbol `$`, ktorý sa nikde inde v reťazci nevyskytuje. Tým zabezpečíme, že každý sufix skončí v samostatnom liste (a nie vo vnútornom vrchole).

Všimnite si, že každá cesta od koreňa k listu zodpovedá jednému sufixu, a teda každá cesta začínajúca v koreni zodpovedá nejakému podreťazcu daného reťazca.

Očividný problém však je, že všetky sufixy majú spolu dĺžku $\Theta(n^2)$. Takýto strom by zaberal príliš veľa pamäte – kvadratickú pamäť si pri textoch dlhých miliardy znakov nemôžeme v žiadnom prípade dovoliť. Zároveň štruktúru, ktorá sama o sebe obsahuje $\Theta(n^2)$ znakov, nedokážeme zostrojiť rýchlejšie než v čase $O(n^2)$, takže aj časová zložitosť by bola zásadný problém.

Na druhej strane, ako vidíme na obr. 1–3, takéto sufixové triese obsahujú množstvo dlhých úsekov, ktoré sa vôbec nerozdeľujú. Prirodzené riešenie je teda jasné: každú takú maximalnú cestu bez vetvenia skomprimujeme a uložíme ako jedinú hranu (pozri obr. 4–6).

Takáto štruktúra sa všeobecne nazýva komprimovaný (alebo kompaktný) písmenkový, lexikografický či prefixový strom; stretnete sa tiež s názvami ako *radixový strom* alebo *Patricia trie*.

Všimnite si, že keďže do stromu vkladáme n reťazcov (všetky sufixy), počet listov je n a teda celý strom má iba lineárne veľa vrcholov aj hrán. To je dobrá správa.

Na druhej strane, zlá správa je, že štruktúra stále zaberá kvadratickú pamäť, pretože sme do nej vložili spolu $\Theta(n^2)$ znakov.

Lenže moment! Všetky reťazce, ktorými sú označené hrany, sú predsa len nejaké úseky pôvodného textu T. Takže namiesto ukladania kópií týchto podreťazcov si na každej hrane stačí zapamätať, *odkiaľ–pokiaľ* daný úsek v texte siaha (pozri obr. 7).

Takto si pri každej hrane v strome stačí zapamätať dve čísla – dva indexy do textu T, ktoré určujú začiatok a koniec príslušného úseku. V tejto reprezentácii má teda každá hrana len konštantnú veľkosť a celková pamäťová zložitosť je zrazu *lineárna*!

Zovšeobecnený sufixový strom. Štruktúru môžeme prirodzene zovšeobecniť aj pre množinu viacerých „dokumentov“

$$\mathcal{D} = \{T_1, T_2, \ldots, T_d\}.$$

Jednoducho zostrojíme sufixový strom, ktorý obsahuje *všetky sufixy všetkých dokumentov* (pozri malý príklad na obr. 8).

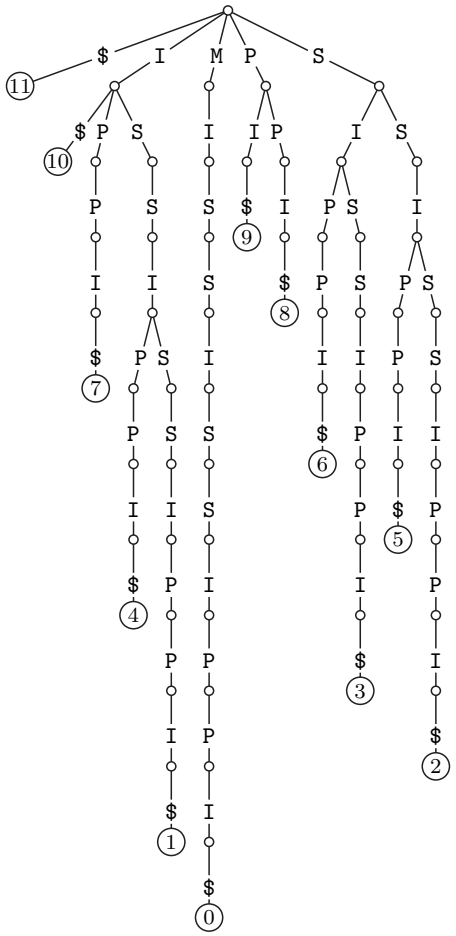
Predstavte si napríklad Wikipédiu, ktorá obsahuje milióny rôznych článkov, knižnicu všetkých kníh sveta, alebo databázy genómov (GenBank, Ensembl) so sekvenciami DNA/RNA najrozličnejších organizmov.

Zovšeobecnený sufixový strom môžeme zostrojiť tak, že každý dokument ukončíme iným špeciálnym znakom, ktorý sa v žiadnom texte nenachádza. Alternatívne môžeme zostrojiť obyčajný sufixový strom pre zreťazený text

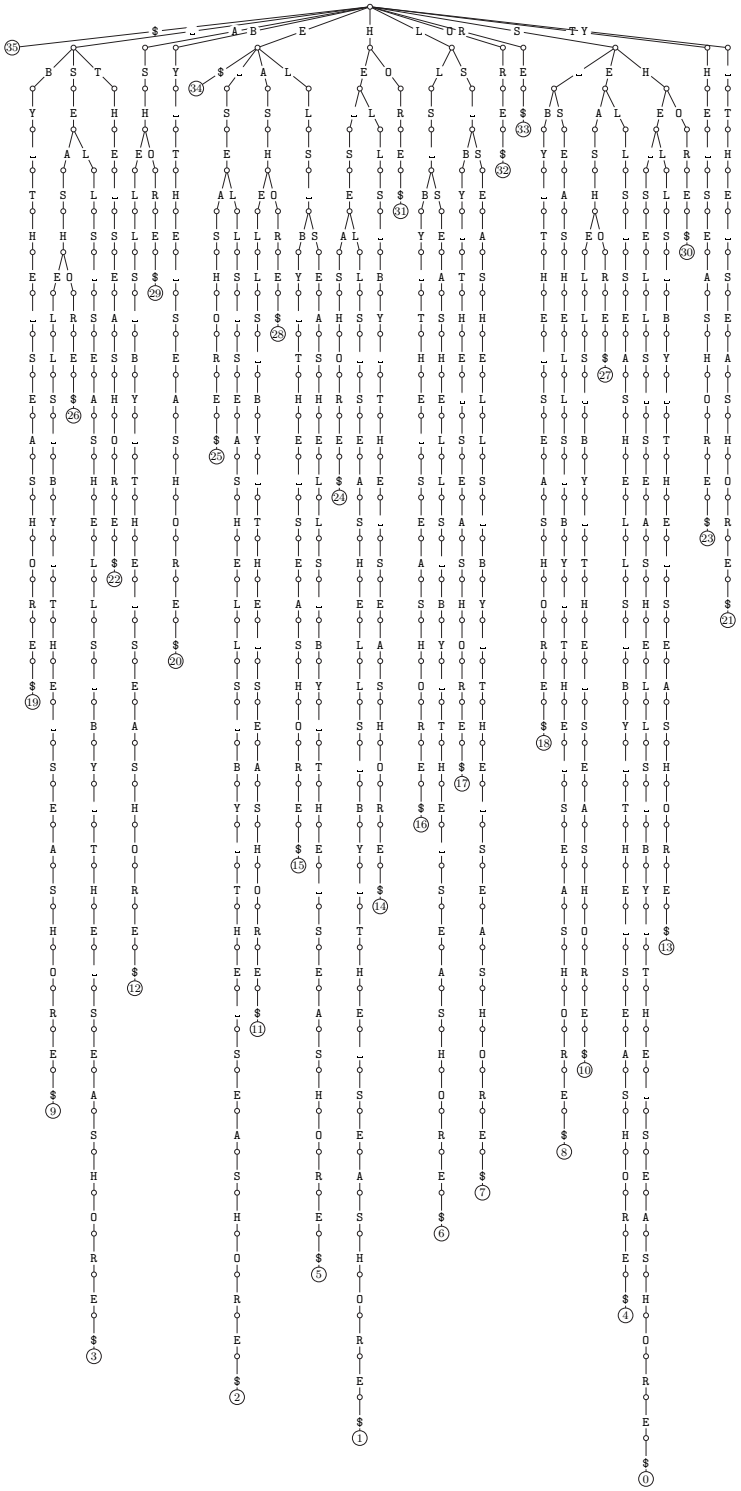
$$T_1\#T_2\#T_3\#\cdots T_d\#\$,$$

kde `#` a `$` sú dva špeciálne ukončovacie symboly, ktoré sa v žiadnom texte nenachádzajú. Jediný drobný rozdiel je v tom, že listy a hrany musia navyše špecifikovať, ktorého dokumentu sa daný sufix (alebo jeho časť) týka.

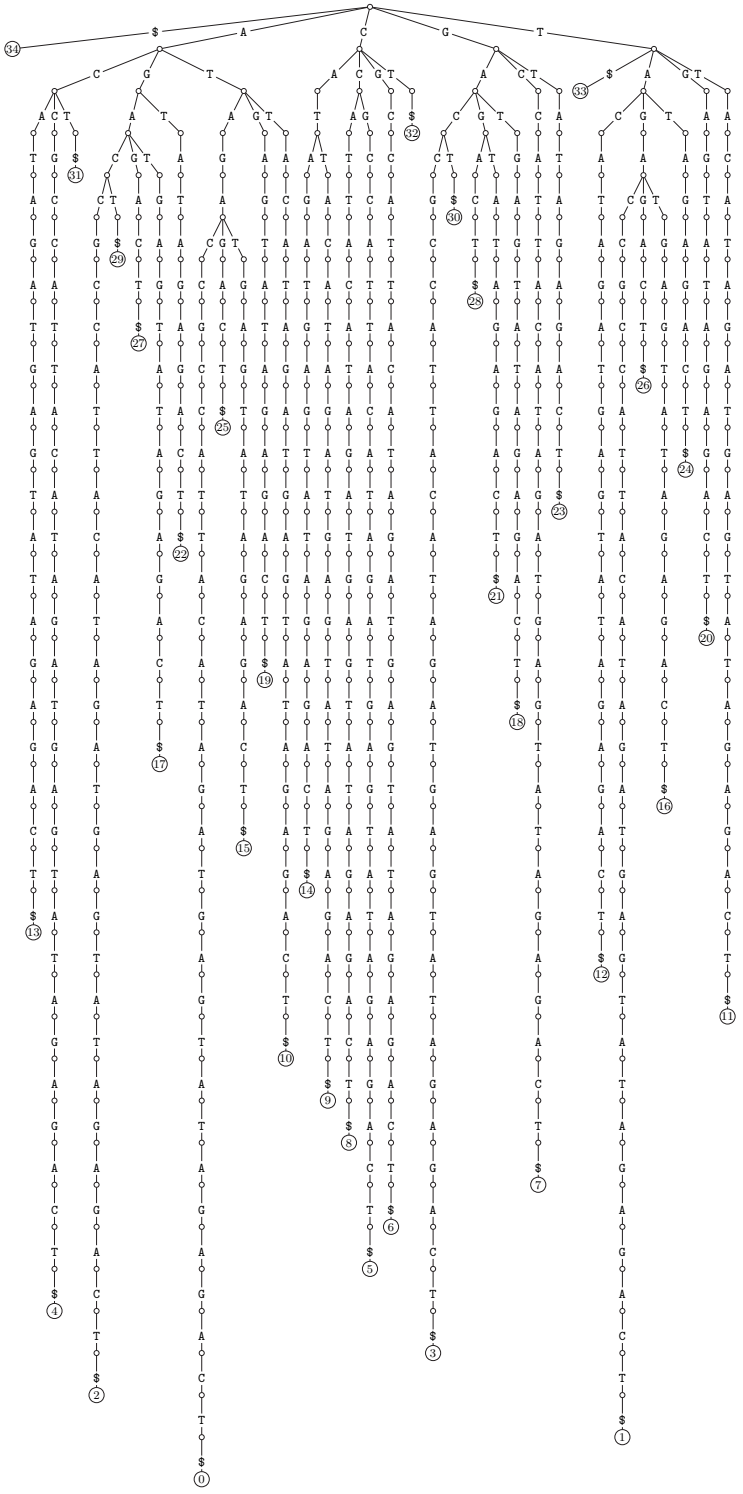
~~Konštrukcia?~~ Aplikácie. Sufixové stromy sa dajú zostrojiť v čase $O(n)$; ako to dosiahnuť, si však vysvetlíme až v kapitole o sufixových poliach. Najprv si však ukážeme, aké užitočné sufixové stromy sú a „milión“ rôznych aplikácií, kde sa dajú využiť. Hovorí sa, že keď má človek kladivo, veľa problémov začne pripomínať klince. A sufixový strom je naozaj veľmi veľké kladivo na najrôznejšie problémy v stringológii.



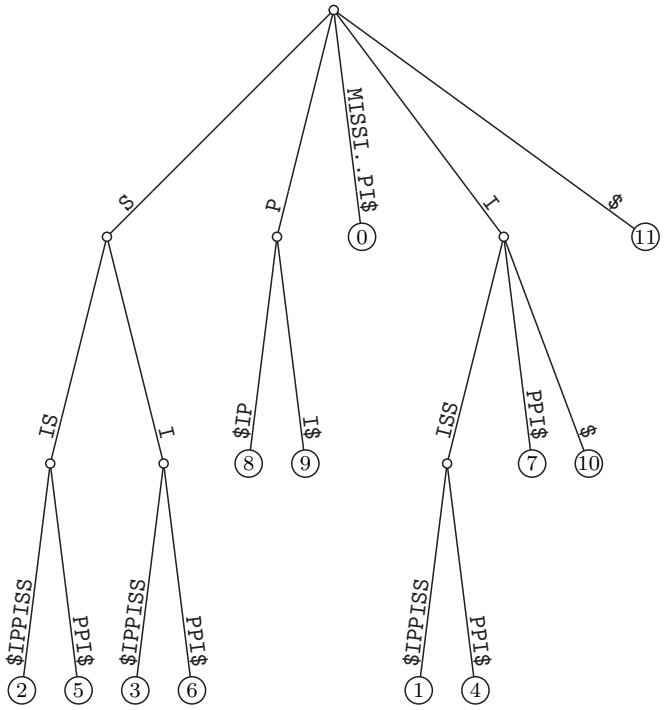
Obr. 1: Sufixový strom pre text „MISSISSIPPI$“.



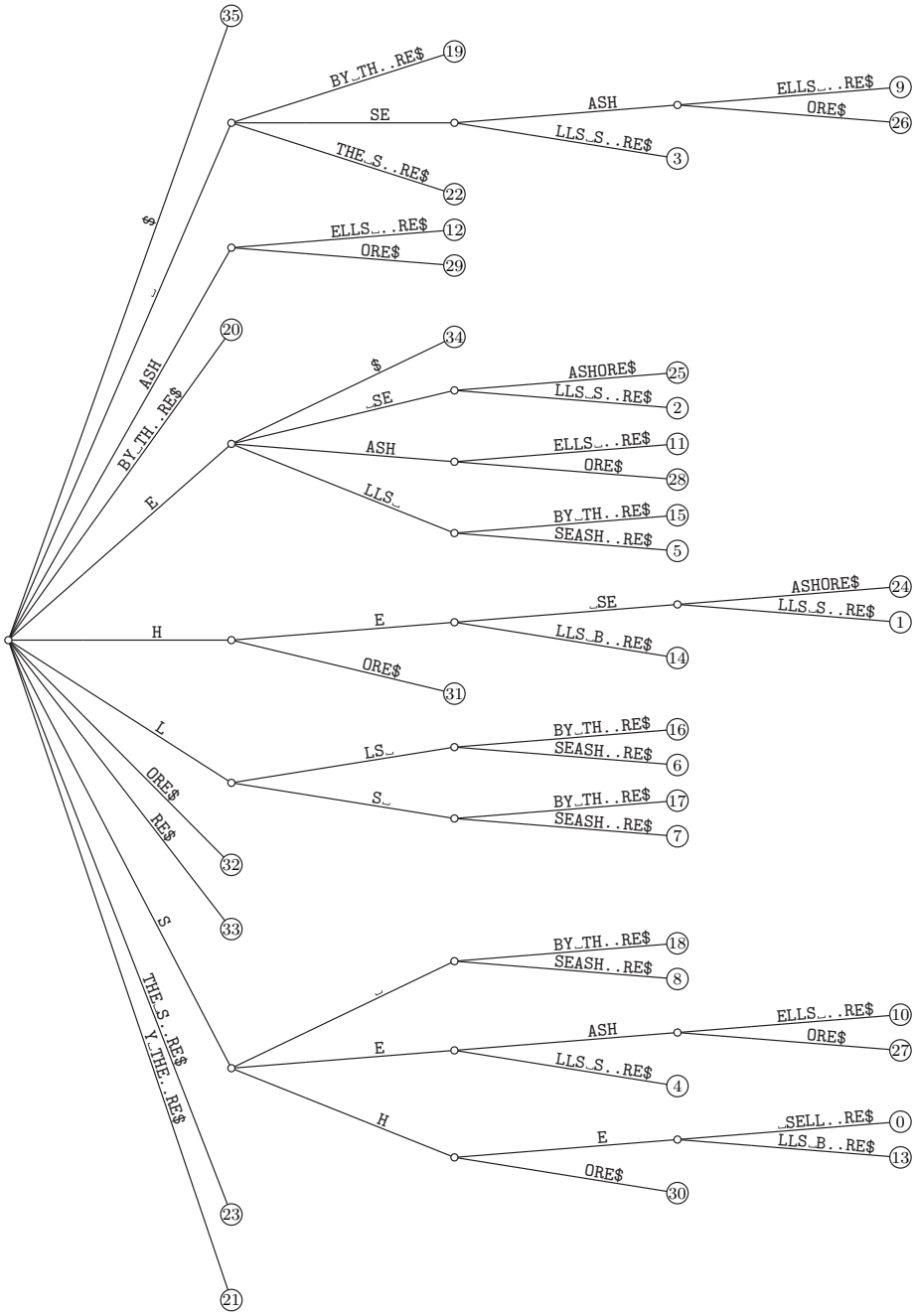
Obr. 2: Sufixový strom pre text „SHE␣SELLS␣SEASHELLS␣BY␣THE␣SEASHORE$“.



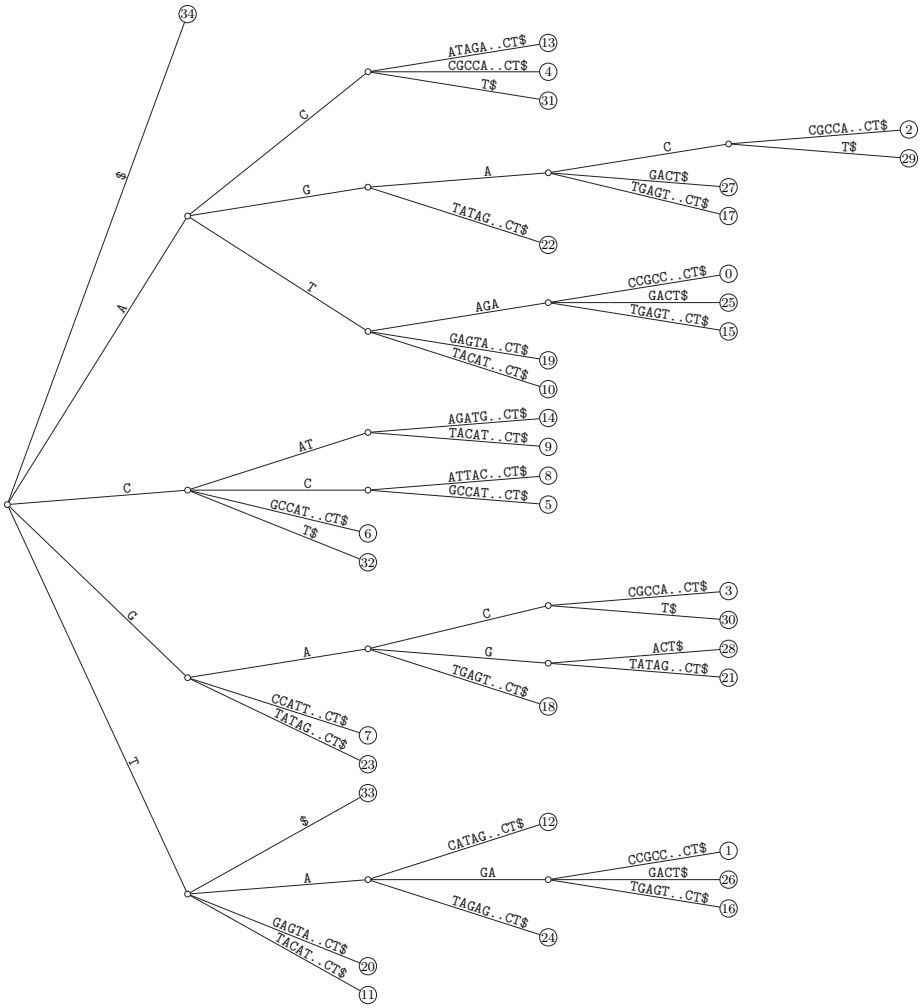
Obr. 3: Sufixový strom pre text „ATAGACCGCCATTACATAGATGAGTATAGAGACT$“.



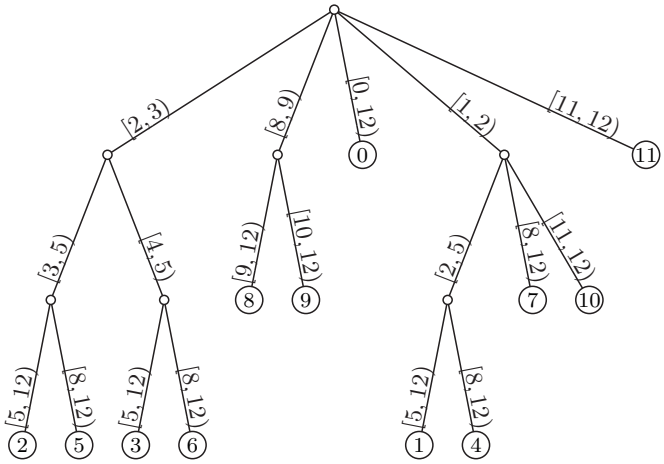
Obr. 4: Sufixový strom pre text „MISSISSIPPI$“. Cesty, ktoré sa nedelili, sme skontrahovali do jednej hrany. Keďže v strome je n sufixov, teda n listov, strom má iba lineárne veľa vrcholov aj hrán. Stále však máme problém, že písmená na jednotlivých hranách zaberajú spolu kvadratický priestor.



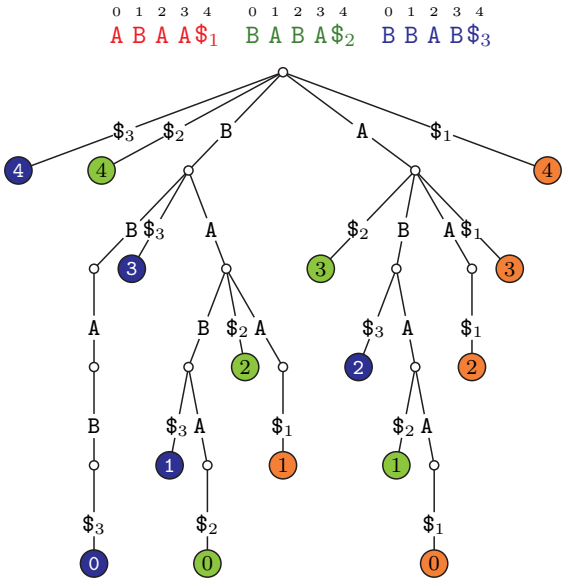
Obr. 5: Sufixový strom (s komprimovanými cestami) pre text „SHE␣SELLS␣SEASHELLS␣BY␣THE␣SEASHORE$“.



Obr. 6: Sufixový strom (s komprimovanými cestami) pre úsek DNA sekvencie „`ATAGACCGCCATTACATAGATGAGTATAGAGACT$`“.



Obr. 7: Sufixový strom pre „MISSISSIPPI$“. Skontrahujeme hrany a namiesto reťazcov na každej hrane použijeme indexy ⟦odkiaľ, pokiaľ) sa nachádza v pôvodnom texte. Takto vieme každú hranu reprezentovať v konštantnej pamäti a tým pádom celý sufixový strom zaberá len lineárnu pamäť.



Obr. 8: Zovšeobecnený sufixový strom môže obsahovať viacero textov (dokumentov) naraz. Listy v tomto prípade predstavujú sufix alebo pozíciu v niektorom texte.

Aplikácie

#1 Vyhľadávanie v texte

Majme text T a vzorku P. Chceme zistiť, či sa P nachádza v T, prípadne nájsť jej prvý výskyt alebo všetky výskyty.

Sufixový strom to umožňuje veľmi jednoducho:

- Stačí zísť z koreňa pozdĺž cesty určenej reťazcom P. Ak sa cesta skončí úspešne v nejakom vrchole, všetky listy v danom podstrome zodpovedajú výskytom P v texte.
- Ak chceme nájsť *prvý* výskyt, predpočítame si pre každý vrchol ukazovateľ na list s najmenším číslom sufixu (alebo priamo pozíciu prvého výskytu). Urobíme to jedným prechodom stromu zdola nahor v čase $O(n)$: pri postorder prechode uložíme do každého vrcholu minimum z jeho detí.
- Ak chceme nájsť *počet* výskytov, stačí si predpočítať počet listov v každom podstrome, opäť jedným postorder prechodom.
- Ak chceme nájsť *všetky* výskyty, jednoducho prehľadáme celý podstrom. Ak je počet výskytov k, podstrom má veľkosť $O(k)$.

Napríklad, vyhľadajme reťazec `BA` vo všeobecnom sufixovom strome na obrázku 8. Začneme v koreni a postupne sledujeme hrany označené `B` a `A`. V podstrome, do ktorého sa takto dostaneme, sa nachádzajú štyri listy: červená 1, zelená 0 a 2 a modrá 1. Tieto listy presne zodpovedajú štyrom výskytom reťazca `BA` v textoch `ABAA`, `BABA` a `BBAB`.

Výsledné zložitosti:

predpočítanie	$O(n)$
prvý výskyt / počet výskytov	$O(m)$
všetky výskyty	$O(m+k)$

kde k je počet výskytov vzorky v texte T.

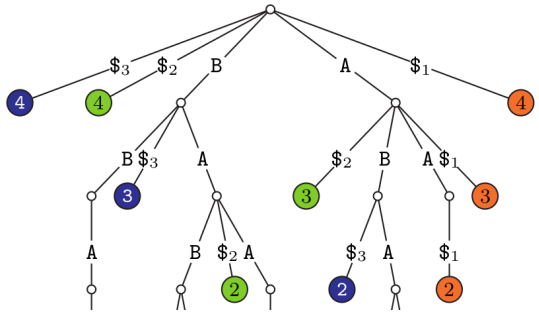
#2 n-gramy

Pod n-gramom rozumieme postupnosť n písmen. Napríklad existuje 8 rôznych trigramov pozostávajúcich iba z písmen `A` a `B`. Ktoré z nich sa vyskytujú v reťazcoch `ABAA`, `BABA` a `BBAB`?

Jeden spôsob, ako to zistiť, je prechádzať reťazce a priebežne si ukladať množinu všetkých trigramov, ktoré sme videli.

Alternatívne sú všetky trigramy priamo viditeľné v zovšeobecnenom sufixovom strome týchto reťazcov: stačí si predstaviť, že strom „rozrežeme“ v hĺbke 3. Potom môžeme odpoveď jednoducho odčítať zo všetkých ciest dĺžky 3 od koreňa:

ABA, BAA, BAB a BBA.



#3 Najdlhší opakujúci sa podreťazec

Aký je najdlhší podreťazec v reťazci MISSISSIPPI, ktorý sa v ňom vyskytuje aspoň dvakrát?

Odpoveď: ISSI.

Ako to však zistíme vo všeobecnosti? Toto je problém, ktorý sme spomínali v úvode kapitoly.

Pred zavedením sufixových stromov bol dlho považovaný za ťažký: existuje triviálny $O(n^3)$ algoritmus a o niečo lepší $O(n^2)$ algoritmus založený na dynamickom programovaní. Dlhý čas nebolo jasné, či sa dá nájsť aj lineárny algoritmus a dokonca sa objavili hypotézy, že žiadny takýto neexistuje.

Prekvapivo však s pomocou sufixových stromov dostaneme riešenie, ktoré je úplne jednoduché:

> *najdlhší opakujúci sa podreťazec zodpovedá najhlbšiemu vnútornému vrcholu sufixového stromu.*

Vnútorný vrchol sufixového stromu vždy reprezentuje reťazec, ktorý sa v texte vyskytuje aspoň dvakrát: pod daným vrcholom sa totiž nachádzajú aspoň dva listy, teda dva sufixy začínajúce rovnakým prefixom. Počet listov v podstrome zároveň udáva počet výskytov tohto podreťazca.

Pre každý vrchol si môžeme predpočítať jeho *textovú hĺbku (string depth)*, t.j. *počet znakov* na ceste od koreňa do daného vrcholu. (Pozor, nejde o klasickú hĺbku vrcholu v zmysle počtu hrán, ale o dĺžku textu uloženého na hranách po ceste od koreňa.)

Najdlhší opakujúci sa podreťazec zodpovedá vnútornému vrcholu s maximálnou hodnotou *textovou hĺbkou*. Takýto vrchol vieme nájsť v čase $O(n)$ jedným prechodom stromu.

Skúste si rozmyslieť, ako by ste riešili mnohé ďalšie príbuzné úlohy ako:

- najdlhší opakujúci sa podreťazec, ktorého výskyte sa neprekrývajú,
- najkratší podreťazec, ktorý sa v texte vyskytuje len raz,
- najčastejšie sa vyskytujúci reťazec dĺžky aspoň k.

#4 Najdlhší spoločný podreťazec

Podobná úloha ako predošlá, ale tentoraz majme dva texty T_1 a T_2 a chceme nájsť ich najdlhší spoločný podreťazec.

Stačí zostrojiť zovšeobecnený sufixový strom pre množinu $\{T_1, T_2\}$. Listy teraz *ofarbíme dvoma farbami* podľa toho, ku ktorému textu príslušný sufix patrí. Cieľom je nájsť taký vnútorný vrchol, pod ktorým sa nachádzajú listy oboch farieb, a ktorý zároveň maximalizuje svoju *textovú hĺbku*.

Pri prechode stromom zdola nahor si pre každý vrchol predpočítame, či jeho podstrom obsahuje iba listy jednej farby (a ktorú), alebo listy oboch farieb. Vnútorné vrcholy, ktoré majú pod sebou listy oboch farieb, reprezentujú všetky spoločné podreťazce oboch textov. Najdlhší z nich je ten s najväčšou hodnotou textovou hĺbkou.

#5 Maximálne *repeat*y

Majme text T a chceme nájsť všetky *maximálne repeat*y, t.j. také dvojice výskytov podreťazca, že

$$T[i \ldots i+k] = T[j \ldots j+k],$$

ale podreťazec sa už nedá predĺžiť ani doľava, ani doprava:

$$T[i-1] \neq T[j-1] \qquad \text{a} \qquad T[i+k+1] \neq T[j+k+1].$$

Intuitívne hľadáme všetky opakujúce sa podreťazce, ktoré sú „maximálne“ v tom zmysle, že ich ďalšie rozšírenie by už viedlo k nezhode.

V sufixovom strome je riešenie opäť jednoduché. Každý opakujúci sa podreťazec zodpovedá vnútornému vrcholu. Aby bol repeat maximálny, stačí skontrolovať, či jeho výskyty nemožno predĺžiť o jeden znak doľava alebo doprava.

Na tento účel si pri konštrukcii stromu stačí pri každom liste, ktorý reprezentuje sufix začínajúci na pozícii i, poznačiť znak, ktorý sa nachádza bezprostredne *pred ním,* t.j. znak $T[i-1]$ (pre $i = 0$ môžeme použiť špeciálny symbol).

Potom pre každý vnútorný vrchol, pod ktorým sa nachádzajú najmenej dva takto označené listy, môžeme ľahko overiť, či sa daný reťazec dá (alebo nedá) predĺžiť doľava či doprava. Vnútorné vrcholy, ktoré zároveň spĺňajú tieto obmedzenia, zodpovedajú práve maximálnym repeatom v texte.

LCA a RMQ

V nasledujúcich aplikáciách budeme potrebovať vedieť riešiť dve základné podúlohy: *LCA (Lowest Common Ancestor)* a *RMQ (Range Minimum Query).*

LCA: Majme daný zakorenený strom a dva jeho vrcholy u a v. Úlohou je nájsť ich *najnižšieho spoločného predka*, t.j. taký vrchol, ktorý je predkom oboch a zároveň leží najhlbšie v strome. Ak si predstavíme cestu od koreňa k u a v, tak $\mathrm{LCA}(u, v)$ je posledný vrchol na ich spoločnej ceste.

RMQ: Majme pole čísel $A[0 \ldots n-1]$. Dotaz $\mathrm{RMQ}(i,j)$ má vrátiť pozíciu najmenšej hodnoty v intervale $A[i \ldots j]$.

Ako tieto dve úlohy efektívne riešiť si ukážeme v nasledujúcej kapitole. Zatiaľ budeme predpokladať, že ak si vstupný strom/pole vhodne predspracujeme, dokážeme odpovedať na otázky o LCA a RMQ v konštantnom čase.

#6 Najdlhší spoločný prefix

Majme dve pozície i a j v texte T a chceme zistiť najdlhší spoločný prefix sufixov $T[i \ldots]$ a $T[j \ldots]$, t.j. dĺžku najdlhšej počiatočnej zhody:

$$\mathrm{LCP}(i,j) = \max\{\, k : T[i \ldots i+k-1] = T[j \ldots j+k-1] \,\}.$$

Triviálne riešenie. Jednoduchý spôsob je porovnávať znaky jeden po druhom, kým nenájdeme prvý nesúlad. Toto trvá $O(k)$, kde $k = \mathrm{LCP}(i,j)$.

Riešenie pomocou LCA. V sufixovom strome má každý sufix svoj list. Najdlhší spoločný prefix dvoch sufixov zodpovedá *textovej hĺbke* ich najnižšieho spoločného predka:

$$\mathrm{LCP}(i,j) = \text{string-depth}\big(\mathrm{LCA}(\mathrm{list}(i), \mathrm{list}(j))\big).$$

Ak teda vieme LCA počítať v čase $O(1)$, vieme aj $\mathrm{LCP}(i,j)$ určiť v čase $O(1)$.

#7 Približné výskyty

Majme text T dĺžky n, vzorku P dĺžky m a chceme nájsť všetky pozície, kde sa P „približne“ nachádza v T, pričom tolerujeme, ak sa vzorka od výskytu líši najviac v k znakoch.

Triviálne riešenie. Priložíme vzorku P ku každej pozícii v texte a spočítame, v koľkých znakoch sa líši. Toto trvá $O(n \times m)$, pretože pre každú pozíciu porovnávame po znakoch až kým nedosiahneme m.

Rýchlejšie riešenie pomocou sufixového stromu. Môžeme dosiahnuť čas $O(n \times k)$, ak využijeme sufixový strom a najmä možnosť počítať najdlhší spoločný prefix dvoch sufixov v čase $O(1)$.

Myšlienka je jednoduchá: skúsime priložiť vzorku P ku každej pozícii i v texte ako predtým, ale namiesto porovnávania znakov jeden po druhom sa vždy v konštantnom čase posunieme na *najbližšiu chybu*. To urobíme pomocou výpočtu

$$\mathrm{LCP}(P[j \ldots], T[i+j \ldots]).$$

Ak je napríklad priloženie presné na prvých x znakov, LCP nám v $O(1)$ povie hodnotu x, a my okamžite preskočíme o x znakov ďalej, až k najbližšej možnej chybe.

Opakujeme to najviac $(k+1)$-krát, keďže pri viac než k chybách môžeme pozíciu rovno zamietnuť. Na každej pozícii teda strávime $O(k)$ času a celkovo tak dostávame časovú zložitosť $O(n \times k)$.

Poznámka: Tento algoritmus je skôr teoretický.

#8 Počítanie dokumentov

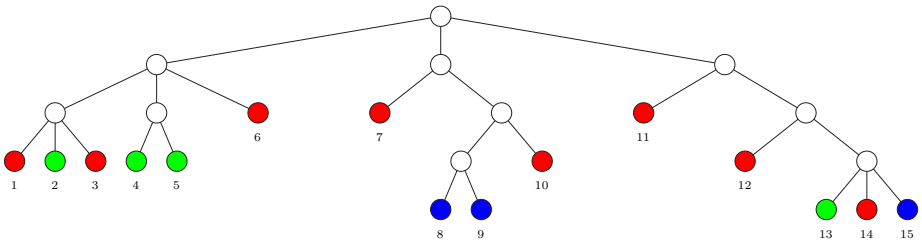
Majme množinu dokumentov

$$\mathcal{D} = \{T_1, \ldots, T_d\}.$$

Označme n_i dĺžku i-teho textu a $N = \sum_i n_i$ celkovú dĺžku všetkých textov dokopy. Pre danú vzorku P chceme spočítať v koľkých rôznych dokumentoch sa vyskytuje.

Pri ôsmej aplikácií snáď neprekvapí, že budeme pracovať so zovšeobecneným sufixovým stromom pre danú množinu dokumentov. Každý list zafarbíme farbou dokumentu, z ktorého pochádza príslušný sufix.

Ak úlohu preformulujeme do reči stromov: Máme d rôznych farieb a pre každý vrchol v strome chceme určiť, *koľko rôznych farieb* (teda dokumentov) sa nachádza v jeho podstrome.



Priamočiare riešenie. Pre vrchol, v ktorom končí vyhľadávaná vzorka, stačí prehľadať celý jeho podstrom a zozbierať farby všetkých listov. To trvá

$$O(m + \#\text{počet listov v podstrome}) = O(m + \#\text{výskytov}).$$

Dá sa to lepšie? V dokumentoch sa hľadaná vzorka môže vyskytovať veľakrát a teda počet výskytov môže byť oveľa–oveľa väčší ako je počet rôznych dokumentov, ktoré ju obsahujú.

Prvý pokus o zrýchlenie. Pre každý vrchol môžeme predpočítať množinu farieb v jeho podstrome. To by však znamenalo pamäť aj čas na predpočítanie

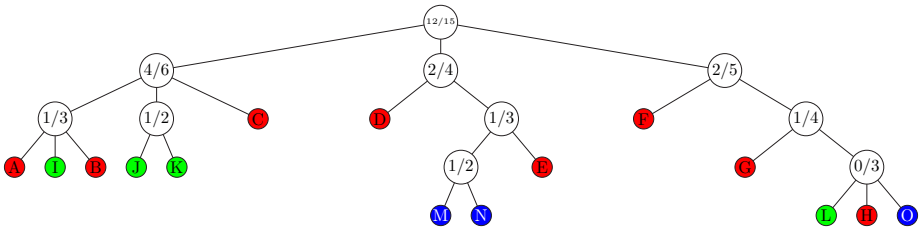
$$O(N \times d),$$

čo je dosť veľa.

Ak je d malé, praktické riešenie je reprezentovať množiny ako bitvektory: jednotkový bit na pozícii j znamená, že podstrom obsahuje j-ty dokument. Pri zjednocovaní množín detí potom stačí z-OR-ovať jednotlivé bitvektory.

Lepšie riešenie pomocou LCA. Finta je v tom, že sa vyhneme explicitnému ukladaniu množín. Fixujme si nejaký konkrétny podstrom (vrchol v). Dva listy a a b patria do podstromu práve vtedy, keď ich najnižší spoločný predok $\mathrm{LCA}(a, b)$ leží v podstrome v.

To nám umožní počítať opakovania farieb nasledovne: ak sa v podstrome nachádza napríklad r listov červenej farby, tak existuje presne $r - 1$ po sebe idúcich dvojíc červených listov, ktorých LCA leží v tomto podstrome.



Namiesto toho, aby sme zisťovali počet rôznych farieb, spočítame najprv *počet opakovaní* farieb. Vo vrchole s podstromom obsahujúcim r listov rovnakej farby má táto farba $r - 1$ "extra výskytov". Ak vieme pre všetky farby spočítať, kde sa takéto opakovania nachádzajú, vieme pre daný vrchol určiť:

$$\#\text{farieb v podstrome } v \;=\; (\#\text{listov v podstrome } v) - (\#\text{opakovaní v podstrome } v).$$

Ako zistíme opakovania?

1. Pre každú farbu si vezmime všetky listy tejto farby a zotrieďme ich podľa poradia v Eulerovskom prechode (zľava doprava).
2. Pre každú dvojicu po sebe idúcich listov a, b danej farby spočítame $\mathrm{LCA}(a, b)$.
3. Pre vrchol $u = \mathrm{LCA}(a, b)$ zaznačíme, že v jeho podstrome začína jedno opakovanie.
4. Nakoniec urobíme jeden DFS prechod zdola nahor. V každom vrchole sčítame všetky hodnoty svojich detí a tak zistíme celkový počet opakovaní v jeho podstrome.

Potom už len odpočítame počet opakovaní od počtu listov a získame počet rôznych farieb. Celú štruktúru vieme predpočítať v čase $O(n)$, ak máme k dispozícii LCA v čase $O(1)$. Po tomto predspracovaní vieme pre každý dotaz zodpovedať v čase $O(1)$.

#9 Hľadanie dokumentov

Uvažujme opäť zovšeobecnený sufixový strom nad množinou dokumentov $\mathcal{D} = \{T_1, \ldots, T_d\}$. Tentokrát budeme chcieť nielen *počet* dokumentov, ktoré obsahujú vzorku P, ale aj vypísať, ktoré to sú. Tak ako pri predchádzajúcej úlohe, si

môžeme vrchol zodpovedajúci sufixu z dokumentu T_i označiť farbou i a úlohou bude vypísať všetky farby v danom podstrome.

Tak ako v predchádzajúcej aplikácii môžeme bez predpočítania jednoducho prejsť celý podstrom a zozbierať farby listov. Časová zložitosť tohto riešenia je však

$$O(m + \#\text{vrcholov v podstrome}).$$

My by sme ideálne chceli algoritmus, ktorý má zložitosť

$$O(m + \#\text{rôznych farieb v podstrome}).$$

Kľúčová myšlienka. Zavedieme si pole $A[1..n]$, kde $A[i]$ je index *predošlého listu s rovnakou farbou* ako list i. Ak je list i prvý zo svojej farby, nech $A[i] = 0$.

Pozri príklad na obr. 9.

Listy zodpovedajúce výskytom vzorky tvoria *súvislý interval* $[i, j]$ v tomto poli. Chceme vypísať všetky *rôzne farby* v tomto intervale.

Farba na pozícii k sa v intervale $[i, j]$ vyskytuje prvýkrát práve vtedy, keď

$$A[k] < i.$$

Takže problém sa redukuje na nasledovný:

$$\text{Vypísať všetky } k \in [i, j] \text{ také, že } A[k] < i.$$

Tieto pozície vždy prvým výskytom vzorky P v každom dokumente. Ako ich nájsť efektívne?

Nad poľom A si predpočítame dátovú štruktúru pre *range minimum query* (RMQ) v čase $O(n)$, s dotazmi v čase $O(1)$.

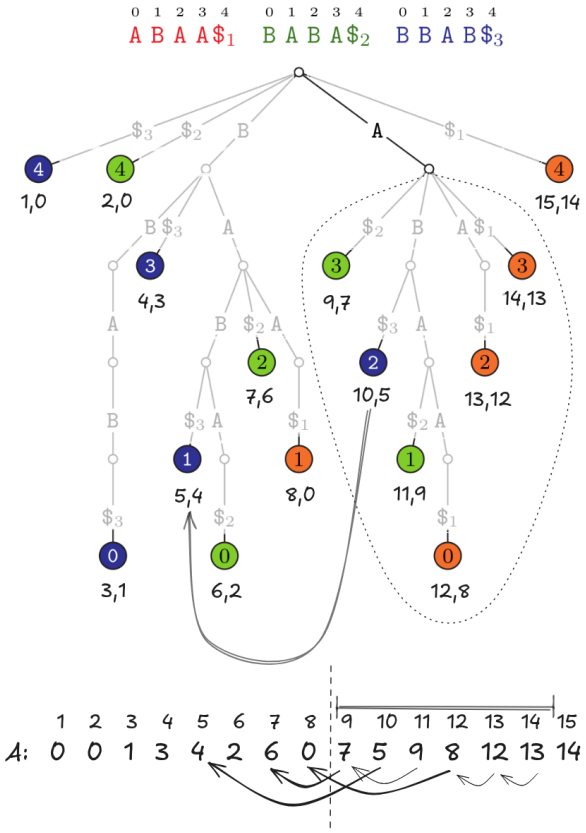
Potom postupujeme rekurzívne:

1. Nájdeme
$$k = \text{argmin}(A[i..j]).$$
2. Ak $A[k] < i$, tak pozícia k reprezentuje nejaký dokument – vypíšeme farbu listu k.
3. Rekurzívne pokračujeme v intervaloch $[i, k-1]$ a $[k+1, j]$.

Rekurzia sa zastaví vždy, keď minimum v intervale už nespĺňa podmienku $A[k] < i$.

Výsledná zložitosť. Predspracovanie poľa A (výpočet A a RMQ) trvá $O(n)$.

Ak si predstavíme strom rekurzívnych volaní (koreň je minimum intervalu, synovia sú minimá ľavého a pravého podintervalu), prechádzame vlastne časť tzv. Kartézského stromu pre dané pole. Každý vnútorný vrchol je jedna farba, ktorú vypíšeme a listy reprezentujú časti poľa, kde rekurziu môžeme ukončiť, pretože všetky prvky sú už väčšie alebo rovné i. Keďže listov je len o 1 viac ako vnútorných vrcholov a každý vrchol zodpovedá len jednému volaniu RMQ, čo je $O(1)$, celkový čas je $O(\#\text{dokumentov})$.



Obr. 9: Pre každý list si poznačíme dvojicu (i, p), kde i je poradové číslo listu zľava doprava a p je číslo *predošlého listu s rovnakou farbou*. Napríklad pod modrým vrcholom 2 máme poznačené $(10, 5)$, pretože je to 10-ty vrchol v poradí a najbližší modrý list vľavo je 5-ty v poradí (ako znázorňuje šípka medzi modrými listami $2 \to 1$). Tieto dvojice tvoria pole A dolu.
Povedzme, že v strome hľadáme vzorku `A` – prejdeme od koreňa po hrane označenej `A` a zakrúžkovaný podstrom zodpovedá všetkým výskytom `A` vo všetkých dokumentoch. Tento podstrom zodpovedá intervalu od 9-teho po 14-ty vrchol. Všimnite si, že prvý výskyt v každom dokumente sú práve tie listy, ktorých predchodca je *skôr ako* 9-ty v poradí. Naopak, opakované výskyty majú predchodcu v intervale $[9, 14]$.
Úloha sa nám teda redukuje na nájdenie tých pozícii k v intervale $[9, 14]$, kde $A[k] < 9$.

Zhrnutie

Sufixový strom je písmenkový strom (trie), ktorý obsahuje všetky sufixy daného reťazca. Zovšeobecnený sufixový strom obsahuje všetky sufixy viacerých reťazcov. Zaberá iba $O(n)$ pamäte, ak použijeme kompaktnú reprezentáciu, pričom cesty, ktoré sa nedelia, skomprimujeme do jednej hrany a každú hranu reprezentujeme len dvoma indexmi do pôvodného textu. Možno ho zostrojiť v čase $O(n)$ (konštrukciu si ukážeme v kapitole o sufixových poliach).

Je veľmi užitočný v stringológii, pretože odhaľuje veľa štruktúry o podreťazcoch daného textu a mnoho problémov nad reťazcami sa dá pomocou neho preložiť na problémy nad stromami. Preto sa oplatí mať poruke malý „prekladový slovník“ medzi svetom reťazcov a svetom stromov:

Stringológia		*Stromy*
pozícia i v texte	$\longleftrightarrow$	list i v sufixovom strome
i-ty sufix	$\longleftrightarrow$	cesta od koreňa k listu i
podreťazec $T[i..j]$	$\longleftrightarrow$	začiatok cesty od koreňa k listu i dĺžky $j-i+1$;
výskyty vzorky P	$\longleftrightarrow$	listy v podstrome pod cestou označenou P
všetky podreťazce dĺžky k v texte	$\longleftrightarrow$	„prerezanie“ stromu v hĺbke k
dokument	$\longleftrightarrow$	farba listu
dokumenty obsahujúce P	$\longleftrightarrow$	rôzne farby listov pod cestou P
spoločný prefix dvoch podreťazcov	$\longleftrightarrow$	spoločná cesta od koreňa smerom k dvom vrcholom
najdlhší spoločný prefix dvoch sufixov	$\longleftrightarrow$	LCA príslušných dvoch listov.