Hello World!

СУФИКСНИ НИЗ

Петар Величковић

Суфиксни низ

Суфиксни низ неког стринга S, $SA[]$, представља лексикографски сортиран низ свих његових суфикса. На пример, за стринг $S = "banana"$, суфиксни низ је $SA = \{"a","ana","anana","banana","na","nana"\}$. Због потенцијала за меморијски изузетно ефикасно чување, уз методе за оптималну изградњу, и применљивости на широк спектар проблема са стринговима, ова структура података представља важан метод у програмерском арсеналу. Ове карактеристике су везане за оригиналну мотивацију ове структуре: ефикасни рад са ДНК секвенцама (на скали читавих организама, где је дужина секвенце $n \approx 3 \cdot 10^9$). При оваквим величинама улаза, меморијска сложеност постаје критичнија од временске, и у таквим околностима је настао суфиксни низ, као петоструко меморијски ефикаснија структура података од суфиксног стабла.

Анализираћемо једну (разумно ефикасну) методу за изградњу суфиксног низа, као и још једне помоћне структуре података (низ *најдужег заједничког префикса*, $LCP[]$) која драстично повећава њен потенцијал. Коначно, показаћемо како, уз ове две структуре, можемо изузетно брзо решити један наизглед компликован проблем са стринговима.

Изградња суфиксног низа

Најпре приметимо да је, уколико имамо приступ оригиналном стрингу S, изузетно меморијски неефикасно чувати суфиксни низ $SA[]$ на начин приказан у уводном поглављу. Наиме, сасвим је довољно чувати индексе почетних позиција: сами суфикси се могу извести користећи ове позиције и сам стринг. За претходно наведени пример ($S = "banana"$), суфиксни низ (индексиран од јединице) ће бити $SA = \{6, 4, 2, 1, 5, 3\}$.

Наивна изградња суфиксног низа представља дефинисање специјалне релације поретка $\preccurlyeq$ над целим бројевима у $[1, |S|]$, тако да важи $i \preccurlyeq j \Leftrightarrow S[i..|S|] < S[j..|S|]$, где се ова два суфикса пореде лексикограф-

ски. Затим можемо покренути брз алгоритам за сортирање попут *quicksort*-а (сложености $O(n \log n)$ поређења) користећи ову релацију над низом $\{1, 2, \ldots, n\}$ да бисмо добили коначан суфиксни низ. Међутим, с обзиром да једно лексикографско поређење има у најгорем случају сложеност $O(n)$, овим добијамо коначну сложеност $O(n^2 \log n)$.

У остатку овог поглавља, описаћемо методу којом се ова сложеност може поправити на $O(n \log^2 n)$, која је довољно брза за већину стварних примена. Напоменимо да је суфиксни низ могуће изградити још ефикасније --- оптимални алгоритми су потпуно *линеарни*, тј. сложености $O(n)$.

Основна идеја жељеног приступа је да, уместо једног сортирања, обавимо *више* пролаза тако да сваки пролаз користи релацију поретка израчунљиву у $O(1)$, уз својство да после k-тог пролаза имамо низ сортиран лексикографски *по првих* 2^k *слова у суфиксима*. Ово постижемо комбиновањем следећа два својства:

- Сортирање по првих $2^0 = 1$ слова је могуће постићи користећи релацију поретка $i \preccurlyeq_0 j \Leftrightarrow S[i] < S[j]$ (директно поређење слова у стрингу).
- Уколико смо већ сортирали суфиксни низ по првих 2^t слова (користећи релацију поретка $\preccurlyeq_t$), могуће је ефикасно сортирати по првих 2^{t+1} слова користећи следећу релацију:
  - Уколико су суфикси из позиција i и j различити по првих 2^t слова, њихов поредак остаје исти по првих 2^{t+1} слова;
  - У супротном, можемо проверити поредак суфикса из позиција $(i + 2^t)$и $(j + 2^t)$ по првих 2^t слова (уколико оба ова суфикса постоје; у супротном, краћи суфикс је лексикографски мањи).
  - Ово је могуће математички записати као:

$$i \preccurlyeq_{t+1} j \Leftrightarrow i \preccurlyeq_t j \vee (i =_t j \wedge ((i + 2^{t+1} \leq n \wedge j + 2^{t+1} \leq n \wedge i + 2^t \preccurlyeq_t j + 2^t) \vee (i + 2^{t+1} > n \wedge i > j)))$$

Релација поретка $\preccurlyeq_{t+1}$ је стога, под условом да знамо релацију $<_t$, израчунљива у константном времену. С обзиром на то да при сваком пролазу алгоритма сортирања дуплирамо количину слова по којима поредимо стрингове, могуће је закључити да ће укупан број пролаза бити ограничен са $O(\log n)$. Уколико за сортирање користимо алгоритам попут quicksort-а, добијамо алгоритам жељене сложености, $O(n \log^2 n)$.

Изградња суфиксног низа
Улаз: Стринг S
Излаз: Суфиксни низ $SA[]$

$\preccurlyeq_0 \leftarrow f(i,j) = S[i] < S[j]$
for $i \leftarrow 1$ **to** $|S|$
  $SA[i] \leftarrow i$
$t \leftarrow 0$
while $2^t \leq |S|$
  sort$(SA, \preccurlyeq_t)$
  $\preccurlyeq_{t+1} \leftarrow f(i,j) = (i \preccurlyeq_t j) \vee (i =_t j \wedge$
    $((i + 2^{t+1} \leq n \wedge j + 2^{t+1} \leq n \wedge i + 2^t \preccurlyeq_t j + 2^t) \vee$
    $(i + 2^{t+1} > n \wedge i > j)))$
  $t \leftarrow t + 1$
return SA

Низ најдужег заједничког префикса

Уколико имамо довољно простора за још један низ дужине $|S|$, можемо додатно појачати потенцијал суфиксног низа за ефикасне алгоритме претраге подударања. Ово постижемо изградњом *низа најдужег заједничког префикса* $LCP[]$, тако да је $LCP[i] = LCP(SA[i], SA[i+1])$, тј. овај низ чува дужине најдужих заједничких префикса свих суседних суфикса у суфиксном низу.

Поново се осврћући на пример $S = "banana"$ и суфиксног низа (записаног овако само ради прегледности)

$$SA = \{"a", "ana", "anana", "banana", "na", "nana"\},$$

можемо директно извести да је $LCP = \{1, 3, 0, 0, 2\}$ (нпр. "na" и "$nana$" имају заједнички префикс дужине 2). Наивна изградња овог низа подразумева директно рачунање заједничког префикса за све суседне суфиксе, и захтева временску сложеност $O(n^2)$. Међутим, уколико смо већ израчунали суфиксни низ, LCP низ можемо израчунати у оптималној сложености $O(n)$ уз једну кључну опсервацију: не рачунамо елементе овог низа *редом* (од првог до n-тог) него по позицијама суфикса у почетном стрингу. Уколико смо у оваквој итерацији успешно спарили k слова, знамо да ћемо у следећој моћи дефинитивно спарити барем $k-1$ слово (јер након смањивања тренутно разматраног суфикса за једно слово, можемо смањити и спареног суфикса за једно слово). Ово је најбоље илустровати примером.

Узмимо, као и увек, $S = "banana"$. Рачунаћемо LCP низ за позиције које одговарају постепено смањујућим суфиксима стринга S:

1. $S_1 = "banana"$. Индекс овог суфикса је 4, и суседни суфикс му је "na"; њихов заједнички префикс је дужине 0. $LCP = \{-, -, -, 0, -\}$.
2. $S_2 = "anana"$. Индекс овог суфикса је 3, и суседни суфикс му је "$banana$"; њихов заједнички префикс је дужине 0. $LCP = \{-, -, 0, 0, -\}$.
3. $S_3 = "nana"$. Индекс овог суфикса је 6, и суседни суфикс не постоји.
4. $S_4 = "ana"$. Индекс овог суфикса је 2, и суседни суфикс му је "$anana$"; њихов заједнички префикс је дужине 3. $LCP = \{-, 3, 0, 0, -\}$.
5. $S_5 = "na"$. Индекс овог суфикса је 5, и суседни суфикс му је "$nana$". Одмах знамо да ће прва два $(3 - 1)$ слова обавезно бити спарена, и можемо започети проверу тек од трећег слова (које не постоји у S_5). $LCP = \{-, 3, 0, 0, 2\}$.
6. $S_6 = "a"$. Индекс овог суфикса је 1, и суседни суфикс му је "ana". Одмах знамо да ће прво $(2 - 1)$ слово обавезно бити спарено, и можемо започети проверу тек од другог слова (које не постоји у S_6). $LCP = \{1, 3, 0, 0, 2\}$.

Изградња низа најдужег заједничког префикса
Улаз: Стринг S, суфиксни низ $SA[]$
Излаз: Низ најдужег заједничког префикса $LCP[]$

```
k ← 0
for i ← 1to |S|
        Дефинишимо p преко SA[p] = i
        if p ≠ |S|
                j ← SA[p + 1]
                while S[i + k] = S[j + k]
                        k ← k + 1
                LCP[p] ← k
                if k > 0
                        k ← k − 1
return LCP
```

Овај алгоритам је (оптималне) сложености $O(n)$, зато што ће свако слово у стрингу бити разматрано не више од двапут (једном кад се буде спаривало, и једном када се буде уклањало).

Најдужи понављајући подстринг

На крају, осврнућемо се на примену ове две структуре података на решавање једног, наизглед не толико једноставног, проблема са стринговима: за дати стринг S, потребно је одредити најдужи подстринг који се понавља бар два пута у стрингу (дозвољена су преклапања). За стринг $S = "banana"$, решење је $"ana"$ (понављања од позиција 2 и 4).

Под овом дефиницијом, понављајући подстринг дужине L имплицира постојање два суфикса овог стринга чији је заједнички префикс дужине L. Уколико можемо ефикасно наћи највеће такво L, и одредити позицију на којој се одговарајући суфикси појављују у стрингу, проблем је ефективно решен.

За коначно решење треба нам још једна опсервација: најсличнији суфикси улазног стринга (по префиксима, због лексикографског поретка) се налазе на суседним позицијама у суфиксном низу. Самим тим, уколико можемо наћи суседне позиције у суфиксном низу где је поклапање највеће, пронашли смо тражено решење. Међутим, поклапања суседних чланова суфиксног низа можемо ефикасно представити користећи низ најдужег заједничког префикса! Највеће такво поклапање биће управо највећи елемент низа LCP. Након што знамо позицију у суфиксном низу са највећим поклапањем, можемо лако реконструисати тражени подстринг:

Најдужи понављајући подстринг
Улаз: Стринг S
Излаз: Најдужи понављајући подстринг у S

```
Изградити SA[] и LCP[] од S
L ← 0
ind ← 0
for i ← 1 to |S| − 1
        if LCP[i] > L
                L ← LCP[i]
                ind ← i
x ← SA[ind]
return S[x..x + L − 1]
```

Задаци

1. Како бисте изградили суфиксни низ у сложености $O(n \log n)$?
2. За два задата стринга, X и Y, одредите све позиције на којима се Y налази у X. На пример, за $X = "aaabbcaa"$ и $Y = "aa"$, решење је $\{1, 2, 7\}$.
3. За два задата стринга, X и Y, одредите њихов најдужи заједнички подстринг. На пример, за $X = "abcdefgh"$ и $Y = "aaabbbcccdefffff"$, решење је $"cdef"$.

Статијата прв пат е објавена во списанието Тангента на ДМ на Србија во 2016/17 година