Pregatirea lotului national de informatica

Alba Iulia, 2004

SUFFIX ARRAYS

Prof. Emanuela Cerchez

Liceul de Informatica “Grigore Moisil” Iasi

Problema

Fie $A = a_0 a_1 \ldots a_{N-1}$ un sir de lungime N peste alfabetul S.

`Query(W):`

Fie $W = w_0 w_1 \ldots w_{P-1}$ un sir de lungime P peste alfabetul S. Sa se determine toate aparitiile sirului W in sirul A.

Observatie

Sirul A este fixat si trebuie sa suporte numeroase interogari.

Solutii

- **Cautare naiva** - O(N*P)
- **KMP** - O(N+P)
- **Suffix trees** - O(N) complexitate timp pentru constructia arborelui si O(N|S|) complexitate spatiu, apoi O(P) complexitate timp pentru query => complexitate spatiu prohibitiva. Prin compresie - O(Nlog|S|) complexitate timp pentru constructie arbore si O(N) complexitate spatiu, apoi O(Plog|S|) complexitate timp pentru query.

Suffix arrays

- Este o structura de date proiectata pentru a raspunde eficient la interogari pentru un sir `A` fixat.
- Complexitate spatiu: `O(N)`
- Complexitate timp pentru interogari in cazul cel mai defavorabil $P+[\log_2(N-1)]$
- Complexitate timp pentru constructie suffix-array: `O(NlogN)` sau `O(N)`, folosind structuri de date suplimentare.

Cautare

- Notam A_i=sufixul lui A care incepe la pozitia i $=a_i a_{i+1} \ldots a_{N-1}$.
- Asociem sirului A un vector Pos, de lungime N. $Pos[i]$=pozitia de inceput in A al celui de-al k-lea cel mai mic sufix dintre $\{A_0, A_1, \ldots, A_{N-1}\}$ (k-smallest), relativ la relatia de ordine lexicografica.
- $A_{Pos[0]} < A_{Pos[1]} < \ldots < A_{Pos[N-1]}$ (unde < indica relatia de ordine lexicografica).

Exemplu

- Fie `A=aabaabab` un sir de lungime `8`.
- Cele 8 sufixe ale lui `A` sunt: A_0=aabaabab, A_1=abaabab, A_2=baabab, A_3=aabab, A_4=abab, A_5=bab, A_6=ab, A_7=b.
- Sufixele in ordine lexicografica sunt: {A0, A3, A6, A1, A4, A7, A2, A5}.
- Vectorul `Pos` este (0,3,6,1,4,7,2,5).

Notatii

- Fie x un sir. Notam x^p prefixul de lungime p al lui x (adica $x_0x_1 \ldots x_{p-1}$).
- Fie x si y doua siruri $x <_p y$ daca si numai daca $x^p < y^p$ (se compara lexicografic prefixele de lungime p).
- Analog se pot defini $\geq_p$, $\leq_p$, $>_p$.

Observatie. Toate sufixele care au p-prefixe egale apar in Pos pe pozitii consecutive.

Alte notatii

- $L_w = \min\{k \mid W \leq_P A_{Pos[k]}\}$ sau N
- $R_w = \{\max\{k \mid A_{Pos[k]} \leq_P W\}$ sau -1
- Deoarece Pos este ordonat si dupa relatia $\leq_P$ deducem ca W apare incepand cu pozitia i in A ("se potriveste" cu $a_i a_{i+1} \ldots a_{i+p-1}$) daca si numai daca $i = Pos[k]$ unde $k \in [L_w, R_w]$. Deci numarul de aparitii ale lui W in A este $R_w - L_w + 1$.

Algoritmul de determinare L_w si R_w

- Vectorul `Pos` fiind ordonat in raport cu relatia $\leq_P$, putem determina L_w si R_w modificand algoritmul de cautare binara. Prin urmare sunt necesare `O(logN)` comparatii de siruri de lungime `P`, deci complexitatea este `O(P log N)`.

Algoritm 1

```
if (W≤PAPos[0])Lw=0;
    else
    if (W>PAPos[N-1]) Lw=N;
        else
        {L=0; R=N-1;
         while (R-L>1)
                {M=(L+R)/2;
                 if (W ≤PAPos[M]) R=M;
                     else L=M;
                }
         Lw=R;}
```

Exemplu

- Sa consideram sirul `A` din exemplul precedent (`A= aabaabab`) si `W=ab`. Vectorul `Pos` determinat este `(0,3,6,1,4,7,2,5)`.
- $L_w = \min\{k \mid ab \leq_2 A_{Pos[k]}\} = 2$.
- $R_w = \max\{k \mid A_{Pos[k]} \leq_2 ab\} = 4$
- Numarul de aparitii ale lui `W` este `4-2+1=3` si aparitiile incep pe pozitiile `Pos[2]=6`, `Pos[3]=1` si `Pos[4]=4`.

Optimizari

- Timpul de executie a algoritmului precedent poate fi imbunatatit, observand ca nu este necesar ca de fiecare data sa executam toate comparatiile aferente testului $\leq_P$ de la inceput. Putem utiliza informatiile din comparatia curenta pentru a optimiza comparatiile urmatoare.

Optimizare 1

- Fie v si w doua siruri. Notam cu $lcp(v,w)$ lungimea celui mai lung prefix comun al sirurilor v si w. Observati ca $lcp(v,w)$ poate fi determinat pe parcursul compararii sirurilor v si w.
- Putem modifica algoritmul 1., retinand in variabila $l=lcp(W,A_{Pos[L]})$ si in variabila $r=lcp(W,A_{Pos[R]})$
- Initial l este setat in urma compararii dintre W si $A_{Pos[0]}$, iar r este setat in urma compararii dintre W si $A_{Pos[N-1]}$.
- In urma comparatiei efectuate intre W si $A_{Pos[M]}$ va fi actualizat l sau r.
- Retinand valorile l si r, nu mai este necesar sa efectuam primele $h=min\{l,r\}$ comparatii de simboluri atunci cand comparam W cu $A_{Pos[M]}$, deoarece $A_{Pos[L]}=_l W=_r A_{Pos[R]}$ implica faptul ca $A_{Pos[k]}=_h W$, pentru orice $L<=k<=R$. Aceasta modificare va reduce numarul de comparatii de simboluri, dar in cazul cel mai defavorabil complexitatea ramane $O(Plog\ N)$.

Optimizare 2

- Pentru a reduce numarul de comparatii de simboluri la $P+[\log_2(N-1)]$ vom precalcula informatiile referitoare la prefixe.
- Sa consideram toate tripletele de forma (L, M, R) care pot aparea in algoritmul 1 ($0<=L<M<R<=N-1$). Exista $N-2$ triplete, fiecare avand un unic punct de mijloc $0<M<N-1$. Fie (L_M, M, R_M) tripletul care are mijlocul M.
- Vom utiliza doi vectori suplimentari $Llcp$ si $Rlcp$ avand $N-2$ componente (cate una pentru fiecare triplet).
- $Llcp[M]=lcp(A_{Pos[M]}, A_{Pos[LM]})$
- $Rlcp[M]=lcp(A_{Pos[M]}, A_{Pos[RM]})$

Exemplu

- Sa consideram sirul din exemplu precedent `A=aabaabab` cu vectorul `Pos (0,3,6,1,4,7,2,5)`.
- Cele 6 triplete `(L,M,R)` care pot aparea in timpul executiei algoritmului 1 sunt: `(0,3,7), (0,1,3), (3,5,7), (1,2,3),(3,4,5), (5,6,7)`.
- $Llcp[M]=lcp(A_{Pos[M]}, A_{Pos[LM]})$ (pentru $M=1,6$).
- $Llcp[1]=lcp(A_{Pos[1]}, A_{Pos[0]})=4$
- $Llcp[2]=lcp(A_{Pos[2]}, A_{Pos[1]})=1$
- $Llcp[3]=lcp(A_{Pos[3]}, A_{Pos[0]})=1$
- $Llcp[4]=lcp(A_{Pos[4]}, A_{Pos[3]})=3$
- $Llcp[5]=lcp(A_{Pos[5]}, A_{Pos[3]})=0$
- $Llcp[6]=lcp(A_{Pos[6]}, A_{Pos[5]})=1$

Exemplu - continuare

- $Rlcp[M]=lcp(A_{Pos[M]}, A_{Pos[LM]})$ (pentru $M=1,6$).
- $Rlcp[1]=lcp(A_{Pos[1]}, A_{Pos[3]})=1$
- $Rlcp[2]=lcp(A_{Pos[2]}, A_{Pos[3]})=2$
- $Rlcp[3]=lcp(A_{Pos[3]}, A_{Pos[7]})=0$
- $Rlcp[4]=lcp(A_{Pos[4]}, A_{Pos[5]})=0$
- $Rlcp[5]=lcp(A_{Pos[5]}, A_{Pos[7]})=1$
- $Rlcp[6]=lcp(A_{Pos[6]}, A_{Pos[7]})=2$

Utilizarea vectorilor `Llcp` si `Rlcp`

- Constructia vectorilor `Llcp` si `Rlcp` se poate face in etapa de sortare a sufixelor.
- Sa consideram tripletul `(L, M, R)` corespunzator iteratiei curente, `h=max(l,r)`.
- In analiza noastra sa presupunem fara a restrange generalitatea ca `r<=l=h`. Comparand `h` cu `Llcp(M)` obtinem urmatoarele cazuri:

Cazuri

- Cazul 1: `Llcp(M)>l`

In acest caz, $A_{Pos[M]} =_{l+1} A_{Pos[L]} \neq_{l+1} W$ deci `W` trebuie sa fie in jumatatea din dreapta iar `l` ramane neschimbat.

- Cazul 2: `Llcp(M)=l`

In acest caz stim ca primele `l` simboluri din $A_{Pos[M]}$ si `W` sunt egale; deci trebuie sa comparam simbolurile `l+1`, `l+2`, ... pana cand gasim simbolul `l+j` a.i. $W \neq_{l+j} A_{Pos[M]}$. Simbolul `l+j` va stabili daca `LW` este in partea stanga sau in partea dreapta. In oricare dintre cele doua situatii stim noua valoare pentru `l` sau pentru `r` (`l+j`).

- Cazul 3: `Llcp(M)<l`

In acest caz `W` se potriveste cu primele `l` simboluri din `L` si `<l` simboluri din `M`, deci L_W va fi in partea stanga, iar noua valoare a lui `r` va `fi Llcp[M]`.

- *Observatie*. Pentru cazul `l<r` se analizeaza in acelasi mod, utilizand `Rlcp`.

Cautare in O(P+log N) pentru LW

```
l=lcp(W,A_Pos[0]); r=lcp(W,A_Pos[N-1]);
if (l=P || W[l]<=A[Pos[0]+l]) LW=0;
   else
   if (r<P || W[r]<=A[Pos[N-1]+r]) LW=N;
      else
      {L=0; R=N-1;
       while (R-L>1)
             {M=(L+R)/2;
              if (l>=r)
                 if (Llcp[M]>=l)
                     m=l+lcp(A_Pos[M]+l,W_l)
                     else m=Llcp[M]
```

Cautare in O(P+log N) pentru LW – continuare

```
    else //l<r
     if (Rlcp[M]>=r) m=r+lcp(A_Pos[M]+r, W_r)
        else m=Rlcp[M];
 if (m==P || W[m]<=A[Pos[M]+m])    {R=M; r=m;}
     else {L=M; l=m;}
 }
LW=R;
}
```

Sortarea sufixelor

- Sortarea va fi realizata in $[\log_2(N+1)]$ etape, prin distribuire (*buckets sort*). Numerotam etapele 1, 2, 4, 8, ...
- Vom utiliza |S| galeti (*buckets*), cate una pentru fiecare simbol din alfabet.
- In prima etapa, vom distribui sufixele in galeti in functie de prima litera (sufixul A_i va fi plasat in galeata corespunzatoare lui A[i]). Deci dupa prima etapa, fiecare galeata va contine sufixele care incep cu acelasi simbol (deci sunt sortate in raport cu $\leq_1$).
- Rezultatul sortarii este memorat in vectorul Pos. In plus vom utiliza un vector boolean BH care va indica modul de distribuire a sufixelor in cele m_1 galeti, astfel BH[i]=1 daca si numai daca Pos[i] indica primul sufix dintr-o galeata.

Sortarea sufixelor - continuare

- Sa presupunem ca dupa H etape, sufixele sunt partitionate in m_H galeti, fiecare galeata continand sufixe care au acelasi H-prefix (adica sunt sortate in raport cu relatia $\leq_H$).
- In etapa $2H$ prefixele vor fi sortate in raport cu relatia $\leq_{2H}$.
- Fie A_i si A_k doua sufixe care dupa etapa H sunt in aceeasi galeata, adica $A_i =_H A_k$. Pentru a determina ordinea $\leq_{2H}$ dintre A_i si A_k va trebui sa comparam urmatoarele H simboluri din aceste doua sufixe. Dar urmatoarele H simboluri din A_i sunt primele H simboluri din A_{i+H}, iar urmatoarele H simboluri din A_i sunt primele H simboluri din A_{k+H} (iar pentru aceste sufixe cunoastem deja ordinea $\leq_H$).

Sortarea sufixelor - continuare

- Sa consideram A_i, primul sufix din prima galeata (adica $Pos[0]=i$) si sa consideram A_{i-H} (daca $i-H<0$, ignoram A_i si trecem la sufixul indicat de $Pos[1]$, s.a.m.d)
- Deoarece A_i incepe cu cel mai mic H-prefix, deducem ca A_{i-H} trebuie sa fie primul sufix din galeata sa de la pasul $2H$. Prin urmare vom muta pe A_{i-H} la inceputul galetii sale si vom marca acest lucru.
- Pentru fiecare galeata va fi necesar sa cunoastem numarul de sufixe care au fost deja mutate la inceputul galetii (adica sunt ordonate in raport cu relatia $\leq_{2H}$).
- Practic, vom parcurge toate sufixele in ordinea $\leq_H$, si pentru fiecare sufix A_i vom muta sufixul A_{i-H} (daca exista) pe urmatoarea pozitie disponibila din galeata sa de la pasul H, obtinand astfel sufixele sortate in ordinea $\leq_H$.

Structuri de date necesare

- Vom utiliza 3 vectori de cate `N` componente, avand semnificatiile:
- `Pos[i]`=pozitia de inceput al celui de al `i`-lea cel mai mic sufix (in raport cu relatia $\leq_H$).
- `Prm[i]`=inversul lui `Pos[i]` (`Prm[Pos[i]]=i`).
- `BH[i]=1` daca si numai daca `Pos[i]` indica primul sufix (cel mai din stanga) dintr-o galeata (adica $A_{Pos[i]} \neq_H A_{Pos[i-1]}$).
- De asemenea utilizam doi vectori auxiliari:
- `Count[i]`=numarul de sufixe care au fost mutate la inceputul galetii care contine al `i`-lea sufix.
- `B2H[i]=1` daca si numai daca al `i`-lea sufix a fost mutat (deci este deja ordonat in raport cu $\leq_{2H}$).
- Putem implementa galetile ca o lista simplu inlantuita alocata static. `Bucket[x]=-1` daca galeata corespunzatoare simbolului `x` este vida sau pozitia de inceput in sirul `A` a primului sufix din galeata;
- `Link[i]`=pozitia de inceput in `A` a urmatorului sufix din galeata.

Descrierea etapei 2H

- Initializam vectorul `Count` cu `0`, si vectorul `Prm` a.i. `Prm[i]` indica cea mai din stanga pozitie din galeata continand al `i`-lea sufix.
- Parcurgem vectorul `Pos` in ordine crescatoare, cate o galeata la un pas. Fie (`l`, `r`) limitele stanga-dreapta ale galetii curente.
- Sa notam `T[i]=Pos[i]-H`.
- Pentru orice `i`, `l<=i<=r`, incrementam `Count[Prm[T[i]]`, setam `Prm[T[i]]=Prm[T[i]]+Count[Prm[T[i]]-1` si setam `B2H[Prm[T[i]]=1`.
- Inainte de a trece la urmatoarea galeata, vom pastra valoarea `1` in `B2H` numai pentru cea mai din stanga componenta din galeata, celelalte componente din galeata le vom marca in `B2H` cu `0` (astfel in `B2H` va ramane marcat corect numai inceputul galetii).
- Dupa parcurgerea tuturor galetilor, construim `Pos` (pe baza vectorului `Prm`) si `BH=B2H`.

Sortarea sufixelor - etapa 1

```
//initializare
for a∈S Bucket[a]=-1;
//distribuim sufixele in galeti, dupa prima litera
for(i=0;i<N;i++)(Bucket[A[i]],Link[i])=(i,Bucket[A[i]])
//construim vectorii Prm, Pos, BH
c=0;
for a∈S
    {i=Bucket[a];
     while (i!=-1) //parcurgem galeata
           {j=Link[i]; Prm[i]=c;
            if (i==Bucket[a]) //primul sufix din galeata
                BH[c]=1;
                else BH[c]=0;
            c++; i=j;}
BH[N]=1;
for (i=0; i<N; i++) Pos[Prm[i]]=i;
```

Sortarea sufixelor - continuare

```
for (H=1; H<N; H*=2) //sufixele sunt ordonate dupa <=H
   {//parcurgem vectorul Pos galeata cu galeata
    for ∀galeata[l, r] //l, r sunt capetele galetii
       {Count[l]=0;
        for (c=l; c<=r; c++) Prm[Pos[c]]=l;
        d=N-H; e=Prm[d]; Count[e]++; B2H[Prm[d]=1;
        for ∀galeata[l, r]
            {//distribuim sufixele in 2H ordine
              for (c=l; c<=r; c++)
                 {d=Pos[c];
                  if (d∈[0,N-1])
                      {e=Prm[d];
                       Prm[d]=e+Count[e];
                       Count[e]++;
                       B2H[Prm[d]]=1;}
```

Sortarea sufixelor - continuare

```
      for (c=l; c<=r; c++)
           {d=Pos[c];
            if (d∈[0,N-1])
                if (B2H[Prm[d]])
                  {e=min{j|j>Prm[j] && (BH[j] || ! B2H[j])}
                    for (x=Prm[d]+1; x<e; x++) B2H[x]=0;}
      }
for (i=0; i<N; i++) Pos[Prm[i]]=i;
for (i=0; i<N; i++) BH[i]=B2H[i];
}
```

Calcularea lcp

- Cautarea in $O(P+\log N)$ necesita precalcularea vectorilor $Llcp$ si $Rlcp$.
- Sa notam A_i si A_j doua sufixe din galeti consecutive.
- Initial (cand sufixele sunt distribuite in galeti dupa primul simbol), $lcp(Ai,Aj)=0$.
- Dupa pasul $2H$ sufixele sunt distribuite in galeti dupa primele $2H$ simboluri. Daca A_i si A_j sunt doua sufixe care ajung in galeti adiacente la acest pas (deci erau in aceeasi galeata la pasul H), $H \le lcp(A_i,A_j) < 2H$. Mai mult, $lcp(A_i, A_j)=H+lcp(A_{i+H}, A_{j+H})$.
- Se stie ca $lcp(A_{i+H}, A_{j+H}) < H$. Problema este A_{i+H} si A_{j+H} s-ar putea sa nu fie in galeti consecutive.

Calcularea lcp - continuare

- Daca $A_{Pos[i]}$ si $A_{Pos[j]}$ ($i<j$) au $lcp<H$ si Pos este $\leq_H$ ordonat, atunci $lcp(A_{Pos[i]}, A_{Pos[j]})$ este minimul dintre lcp dintre perechile din galeti adiacente dintre $Pos[i]$ si $Pos[j]$:
- $lcp(A_{Pos[i]},A_{Pos[j]})=min\{lcp(A_{Pos[k]},A_{Pos[k+1]}|i\leq k<j\}$
- Din pacate, utilizarea formulei precedente necesita prea mult timp. Pentru a determina lcp in $O(log\ N)$ puteti utiliza un arbore de intervale.

Bibliografie

- Udi Manber & Gene Myers - “Suffix arrays- A new method for on-line string searches.”