Мр Драган Урошевић

ПРОБЛЕМ УПАРИВАЊА РЕЧИ

0. Термини и нотација

Реч над неким алфабетом Σ је сваки коначан низ знакова из тог алфабета:

$$s = s_1 s_2 \dots s_n, \quad s_k \in \Sigma.$$

Ако је Σ коначан алфабет, са Σ^* ћемо означавати скуп свих коначних речи формираних од знакова из алфабета Σ. Уместо термина „реч“ у даљем излагању користићемо термин *стринг*. Стринг са 0 знакова зваћемо *празан стринг* и означавати са ε. Број знакова у стрингу x је дужина стринга. Дужину стринга x ћемо означавати са $|x|$. Над стринговима је дефинисана бинарна операција коју називамо *конкатенацијом* (дописивање, надовезивање). Стринг добијен конкатенацијом стрингова x и y обележавамо са xy. Стринг xy има дужину једнаку збиру $|x| + |y|$, а састоји се од знакова из стринга x иза којих следе знаци из стринга y.

Кажемо да је стринг w префикс стринга x и пишемо $w \sqsubseteq x$ ако је $x = wy$ за неки стринг $y \in \Sigma^*$. Слично, кажемо да је w суфикс стринга x и пишемо $w \sqsupseteq x$ ако је $x = yw$ за неки стринг $y \in \Sigma^*$. Јасно, ако су x и y стрингови и a симбол из алфабета Σ, тада је $x \sqsupseteq y$ ако и само ако је $xa \sqsupseteq ya$. Бити префикс и бити суфикс су транзитивне релације. Такође важи следеће: Нека су x, y и z стрингови и нека је $x \sqsupseteq z$ и $y \sqsupseteq z$. Ако је $|x| \leq |y|$ онда је $x \sqsupseteq y$. Ако је $|x| \geq |y|$ онда је $y \sqsupseteq x$. Ако је $|x| = |y|$ онда је $x = y$.

Ако је U узорак, са U^k ћемо означавати префикс узорка U дужине k. Јасно, $U^0 = \varepsilon$. Слично ћемо са ${}^kU$ означити суфикс стринга U кога чине знаци стринга почев од k-тог.

Кажемо да се стринг x појављује у стрингу y ако и само ако постоје стрингови u и v (могу бити и празни) такви да је $y = uxv$. Ако је $s = |u|$ онда се стринг x појављује у стрингу y са *помаком* s. Проблем упаривања речи се може формулисати на следећи начин:

> *Дата су два стринга, узорак U и текст T. Утврдити да ли се узорак појављује у тексту и ако се појављује одредити позиције (помаке) од којих се појављује.*

У наставку ћемо изложити неколико познатих алгоритама за упаривање. Нећемо наводити комплетан програм, већ само процедуру/функцију која служи

за упаривање. Узорак и текст ће бити типа `rec`, који смо ми дефинисали као низ знакова. Такође користимо функцију `len` која треба да одреди дужину речи, а коју можете сами написати.

1. Секвенцијално упаривање стрингова

Ово је најједноставнији (али истовремено и најмање ефикасан) поступак за упаривање стрингова. По том поступку се за сваку позицију $s \leq |T| - |U| + 1$ проверава да ли се узорак поклапа да подстрингом текста дужине $|U|$ почев од позиције s. Тај поступак се може записати на следећи начин:

```
procedure SekvSM(U, T: rec);
var
  s, p: integer;
  Un, Tn: integer;
begin
  Un := Len(U);
  Tn := Len(T);
  for s := 0 to Tn-Un do begin
    p := 1;
    while (p <= Un) and (U[p] = T[s+p]) do inc(p);
    if p > Un then
      writeln('Pojavljuje se uzorak od pozicije ', s+1);
  end;
end;
```

Неефикасност секвенцијалног алгоритма лежи у чињеници да ми не употребљавамо корисне информације које стекнемо у току поређења узорка и текста од позиције s у наставку тј. при поређењу узорка и текста од позиције $s+1$. Наиме, ако смо при поређењу узорка и текста од позиције s утврдили да се поклапа првих q знакова, ми заправо у потпуности познајемо текст од позиције s до позиције $s+q-1$ (јер се поклапа са префиксом U^q) и према томе знамо да ли се може поклапати префикс U^{q-1} са текстом од позиције $s+1$ и тиме мало скратити цео поступак. Сваки напреднији алгоритам покушава да искористи на неки начин те информације и да на тај начин скрати претраживање.

2. Рабин-Карпов алгоритам

За овај алгоритам је карактеристично да у току провере да ли се поклапају узорак и текст од неке позиције s не поредимо обавезно знак по знак. Наиме, ако је рецимо $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$, онда су стрингови који се пореде бројеви. Такви стрингови су једнаки ако су одговарајући бројеви једнаки. Поступак се може уопштити на произвољан алфабет тако што ћемо сматрати да је сваки симбол из тог алфабета нека цифра у систему са основом d, где је d број симбола у алфабету ($d = |\Sigma|$).

Нека је дат узорак U са укупно Un знакова и текст T са укупно Tn знакова. Можемо произвести низ Un-цифрених бројева t_s^* формираних помоћу цифара

(симбола) $T_s, T_{s+1}, \ldots, T_{s+Un-1}$ $(s = 1, 2, \ldots, Tn - Un + 1)$. Број t_s^* $(s > 1)$ се лако рачуна на основу t_{s-1}^* по формули:

$$t_s^* = (t_{s-1}^* - T_{s-1} * d^{Un-1}) * d + T_{s+Un-1}.$$

Ако са u означимо број чији се запис поклапа са узорком U онда се узорак појављује у тексту само од позиције s за коју важи да је $u = t_s^*$. У складу са овом причом значајно можемо скратити поступак упаривања. Израчунамо u и t_1^*. На основу израчунатог t_1^* лако рачунамо $t_2^*, t_3^*, \ldots, t_{Tn-Un+1}^*$ и сваки од њих поредимо са u. Невоље настају када је величина алфабета d и/или дужина узорка Un релативно велика тако да бројеви које производимо у поступку упаривања могу бити толико велики да се не могу регистровати помоћу стандардних целобројних типова (integer, longint).

Тада не региструјемо праве вредности броја u и бројева t_s^* већ бројеве $u \ mod \ p$ и $t_s^* \ mod \ p$ где је p неки природни број (најчешће неки прост број). Да би се поклапали узорак U и текст од позиције s мора обавезно бити $u \ mod \ p = t_s^* \ mod \ p$. Међутим то није довољно. Да бисмо били сигурни да се поклапају морамо проверити да ли се поклапају одговарајући знаци. Провером једнакости по модулу вероватно смо одбацили неке могуће помаке s и тиме скратили поступак упаривања.

Поступак можемо записати следећом функцијом:

```
procedure RKSM(U, T: rec; d: integer; pp: longint);
(* d je velicina alfabeta, a pp je prost broj koji se
poredi pri proveri jednakosti dva stringa odnosno podstringa *)
var
  s, p: integer;
  um, tm: longint;
  Un, Tn: integer;
function stp(x, n: integer; pp: longint): longint;
var
  k: integer;
  rez: longint;
begin
  rez := 1;
  for k := 1 to n do
    rez := (rez*x) mod pp;
  stp := rez;
end;
begin
  Un := Len(U);
  Tn := Len(T);
  um := 0; tm := 0;
  for p := 1 to Un do begin
    um := (um*d+ord(U[p])) mod pp;
    tm := (tm*d+ord(T[p])) mod pp;
  end;
  for s := 0 to Tn-Un do begin
```

```
    if um = tm then begin
      p := 1;
      while (p <= Un) and (U[p] = T[s+p]) do inc(p);
      if p > Un then
writeln('Pojavljuje se uzorak od pozicije ', s+1);
    end;
    tm := (tm-ord(T[s+1])*stp(d, Un-1, pp)) mod pp;
    tm := (tm*d+ord(T[s+Un+1])) mod pp;
    if tm < 0 then tm := tm+pp;
  end;
end;
```

3. Упаривање коришћењем детерминистичких аутомата

Детерминистички аутомат је уређена петорка $A = (\Sigma, Q, \delta, s, T)$ где је Σ неки алфабет, Q скуп стања аутомата, s почетно стање, T скуп завршних стања, а $\delta\colon Q \times \Sigma \to Q$ пресликавање које паровима облика (p, x) где је $x \in \Sigma$ и $p \in Q$ додељује неко стање. Ако је $\delta(p, a) = q$ кажемо да аутомат из стања p прелази у стање q при наиласку слова a. Функција аутомата је да врши препознавање речи које јесу у неком језику L. У овом случају треба да препознаје само једну реч а то је узорак U. Препознавање је омогућено одређеним скупом стања која на неки начин треба да памте већ проучени (проанализирани) део речи. Када аутомат добије наредно слово речи он реагује тако што прелази у ново стање а то ново стање зависи и од претходног дела речи и од тог последњег знака. Ако аутомат пређе у завршно стање, кажемо да је реч исправна, тј. препозната од стране аутомата.

У случају упаривања речи аутомат има $|U| + 1$ стање (у даљем тексту ћемо користити ознаку Un за дужину узорка, тј. $|U| = Un$). Аутомат се налази у стању k ако се последњих k анализираних слова у тексту поклапа са префиксом U^k и не постоји нити једно $m > k$ такво да се m последњих анализираних слова поклапа са префиксом U^m. Ако се аутомат налази у стању Un по претходној причи последњих Un слова текста се поклапа са префиксом дужине Un а то је заправо комплетан узорак, тј. пронађена је једна појава узорка у тексту. Ако се аутомат налази у стању k а следеће слово у тексту је x онда постоје два случаја:

1. Ако је $x = U_{k+1}$ онда се префикс U^{k+1} поклапа са последњих $k+1$ знакова текста те аутомат прелази у стање $k + 1$.

2. Ако је $U_{k+1} \neq x$ онда слово x не одговара у смислу да се сигурно не поклапа последњих $k + 1$ знакова текста и префикс U^{k+1}, али се зато анализира реч $r = U^k x$.

Ако се последњих $m < k + 1$ знакова текста поклапа са префиксом U^m онда мора бити $U^m \sqsupset r$. Да не бисмо прескочили неку појаву узорка у тексту морамо направити што мањи помак по тексту а то значи да m мора бити што веће. Зато одређујемо највеће могуће m за које важи да је $U^m \sqsupset r$ и то је стање у које прелази аутомат у овом случају.

Поступку упаривања помоћу детерминистичких аутомата претходи конструкција (одређивање) функције прелаза за аутомат и то се ради на основу узорка.

```
procedure DASM(u, t: rec);
var
  p, st: integer;
  Un, Tn: integer;
procedure OdrediDA;
var
  z: char;
  k1, k2: integer;
  pi: array[0..MAXL] of integer;
begin
  Un := Len(u);
  pi[0] := 0;
  pi[1] := 0;
  for k1 := 0 to Un do
    for z := chr(0) to chr(255) do delta[k1,z] := 0;
  for k1 := 2 to Un do begin
    k2 := pi[k1-1];
    while (k2 <> 0) and (u[k2+1] <> u[k1]) do k2 := pi[k2];
    if u[k2+1] = u[k1] then inc(k2);
    pi[k1] := k2;
  end;
  writeln(pi[Un]);
  delta[0,u[1]] := 1;
  for k1 := 1 to Un do begin
    if k1 < Un then
      delta[k1, U[k1+1]] := k1+1;
    k2 := pi[k1];
    while k2 <> 0 do begin
      z := u[k2+1];
      if delta[k1,z] = 0 then delta[k1,z] := k2+1;
      k2 := pi[k2];
    end;
    z := u[1];
    if delta[k1,z] = 0 then delta[k1,z] := 1;
  end;
end;
begin
  Un := Len(u);
  Tn := Len(t);
  OdrediDA;
  st := 0;
  for p := 1 to Tn do begin
    st := delta[st,t[p]];
    if st = Un then begin
      writeln('Pojavljuje se uzorak od pozicije ', p-Un+1);
    end;
  end;
end;
```

4. Кнут-Морис-Пратов алгоритам

Ако смо при поређењу узорка и текста од позиције s утврдили да се првих p знакова поклапа, онда ми знамо у потпуности део текста дужине p знакова од позиције s. Тако знамо да је $U_1 = T_s, U_2 = T_{s+1}, U_3 = T_{s+2}, \ldots U_p = T_{s+p-1}$. Ако претпоставимо да се узорак појављује у тексту од позиције s' ($s' - s = q$), онда мора бити $U_1 = T_{s'} = T_{s+q} = U_{q+1}, U_2 = T_{s'+1} = T_{s+q+1} = U_{q+2}$ итд. Другим речима префикс узорка U дужине $p - q$ мора истовремено бити суфикс префикса узорка U дужине p. Према томе, ако U^{p-1} није суфикс од U^p нема разлога да проверавамо да ли се узорак појављује од позиције $s+1$. Слично, ако U^{p-2} није суфикс од U^p нема разлога ни проверавати од позиције $s+2$. Прва позиција $s' = s + q$ од које треба проверавати јесте она за коју важи да је U^{p-q} суфикс од U^p. Зато дефинишемо функцију π: $\{0, 1, 2, \ldots, Un\} \to \mathbf{N}_0$ тако да је $\pi(k_1)$ највећи број $k_2 < k_1$ за који важи да је U^{k_2} суфикс од U^{k_1}:

$$\pi(k_1) = \max\{k_2 : U^{k_2} \sqsupset U^{k_1}\}.$$

Како израчунати функцију π?. Може се израчунати рекурзивно. Приметимо да је $\pi(1) = 0$ и да је $\pi(0) = 0$. Даље приметимо да ако је $U^q \sqsupset U^p$ онда је $U^{q-1} \sqsupset U^{p-1}$ и $U_q = U_p$. Ту чињеницу ћемо искористити да бисмо израчунали $\pi(k)$. Пре свега, низ

$$k_1 = \pi(k-1), k_2 = \pi^2(k-1) = \pi(\pi(k-1)), \ldots, k_m = \pi^m(k-1) = \pi(\pi^{m-1}(k-1)), \ldots$$

монотоно опада тако да ће у једном тренутку бити изједначен са нулом. Такође, за свако k_i важи да је $U^{k_i} \sqsupset U^{k-1}$. Међутим, у том низу су побројани сви бројеви q за које важи да је $U^q \sqsupset U^{k-1}$. Природно је очекивати да не важи свако q да је истовремено и $U^q \sqsupset U^{k-1}$ и да је $U_{q+1} = U_k$. Ми ћемо одабрати највеће за које то важи и то $q+1$ ће бити вредност функције π:

$$\pi(k) = \begin{cases} \max\{\pi^m(k-1) | U_{\pi^m(k-1)+1} = U_k\} + 1, & \text{ако такав } m \text{ постоји} \\ 0, & \text{у супротном.} \end{cases}$$

Процедура се врло једноставно пише:

```
procedure RacunajPi(u: rec; var pi: niz);
var
  k1, k2, Un: integer;
begin
  pi[0] := 0;
  pi[1] := 0;
  Un := Len(u);
  for k1 := 2 to Un do begin
     k2 := pi[k1-1];
     while (k2 <> 0) and (u[k2+1] <> u[k1]) do k2 := pi[k2];
     if u[k2+1] <> u[k1] then pi[k1] := 0 else pi[k1] := k2+1;
  end;
end;
```

Улога функције π је да одреди померај по тексту. Тако, ако смо утврдили да се првих p знакова узорка поклапа са текстом од позиције s, онда следећи пут

вршимо поређење од позиције $s + p - \pi(p)$ (ако је $p = 0$ онда се померамо за 1). Тако се процедура за упаривање може записати у следећем облику:

```
procedure KMPSM(u, t: rec);
var
  Tn, Un: integer;
  pi: niz;
  p, s: integer;
begin
  Un := Len(u);
  Tn := Len(t);
  Racunajpi(u, pi);
  s := 0;
  p := 1;
  while s <= Tn-Un do begin
    while (p <= Un) and (u[p] = t[s+p]) do inc(p);
    if p > Un then writeln('Poklapa se od pozicije ', s);
    if p > 1 then begin
      s := s+p-pi[p-1]; p := p-pi[p-1];
    end else begin
      inc(s); p := 1;
    end;
  end;
end;
```

Друга варијанта за запис исте процедуре је следећа. Надамо се да ће корисник разумети приложени код:

```
procedure KMPSM(u, t: rec);
var
  Tn, Un: integer;
  pi: niz;
  p, s: integer;
begin
  Un := Len(u);
  Tn := Len(t);
  Racunajpi(u, pi);
  p := 0;
  for s := 1 to Tn do begin
    while (p <> 0) and (u[p+1] <> t[s]) do p := pi[p];
    if u[p+1] = t[s] then p++;
    if p = Un then begin
      writeln('Poklapaju se od pozicije ', s-Un+1);
      p := pi[p];
    end;
  end;
end;
```

5. Сандијев алгоритам

Сандијев алгоритам за упаривање речи је настао комбиновањем лош знак хеуристике и добар суфикс хеуристике. Међутим узорак и текст се не пореде редом. Знаци од којих је састављен узорак сортирају се по фреквентности у језику или узорку. Та фреквентност одређује редослед поређења тако да се прво пореди знак са најмањом фреквенцијом у узорку и знак који би њему требало да одговара у тексту, затим знак са следећом фреквенцијом итд.

Померај по тексту је одређен на основу два критеријума. Први критеријум је лош знак хеуристика. Ако смо проверавали да ли се узорак U дужине Un појављује у тексту од позиције s онда знак T_{s+Un} одређује за колико ћемо се померити по тексту. Наиме, по тексту се померамо тако да се знак на позицији $s+Un$ поклопи са последњом појавом тог знака у узорку. Ако се тај знак уопште не појављује у узорку, онда се померамо по тексту на позицију $s+Un+1$ и то је идеална ситуација.

Добар суфикс хеуристика је заснована на чињеници да се по тексту померамо за неко d тако да се већ анализирани знаци узорка (за које самим тим већ знамо где се налазе и колика им је вредност) не разликују од знакова у узорку који се налазе на том растојању d. Другим речима, ако већ знамо да је $U_k = T_{s+k}$ и ако се померамо за d места по тексту, онда мора бити $U_k = T_{s+k} = T_{(s+d)+(k-d)} = U_{k-d}$ (под претпоставком да је $k \geq d$). То наравно мора важити за свако k, за које смо већ констатовали да је $U_k = T_{s+k}$. Зато ће d бити најмањи број d који је већи од нуле и који задовољава тај услов да је $U_k = U_{k-d}$ (наравно, ако је $k \geq d$).

Програм се може записати на следећи начин.

```
(* pomocni tip cija je uloga da pripamti znake
uzorka i pozicije znakova u samom uzorku *)
const
  LMAX = 1000;
type
  uzorak = record
    zn: char; poz: integer;
  end;
  nizu = array[1..LMAX] of uzorak;
  nizp = array[char] of integer;
  rec = array[1..LMAX] of char;
procedure SandiSM(u, t: rec);
var
  pp, frek: nizp;
  (* niz za pozicije poslednjih pojava i frekvencije znaka u uzorku *)
  uzr: nizu;
  dsd: array[0..LMAX] of integer;
  s, p: integer;
  Un, Tn: integer;
function CMPuzorak(uzr1, uzr2: uzorak): integer;
begin
```

```
  if frek[uzr1.zn] > frek[uzr2.zn] then CMPuzorak := 1
  else
    if frek[uzr1.zn] < frek[uzr2.zn] then CMPuzorak := -1
    else
      CMPuzorak := uzr1.poz-uzr2.poz;
end;
procedure UrediUzorak(U: rec);
var
  ch: char;
  tuzr: uzorak;
  Un: integer;
  i, j, k: integer;
begin
  Un := Len(u);
  for k := 1 to 255 do
    frek[chr(k)] := 0;
  for i := 1 to Un do inc(frek[u[i]]);
  for i := 1 to Un do begin
    uzr[i].poz := i; uzr[i].zn := u[i];
  end;
  for i := 1 to Un do
    for j := i+1 to Un do
      if CMPUzorak(uzr[i], uzr[j]) > 0 then begin
        tuzr := uzr[i]; uzr[i] := uzr[j]; uzr[j] := tuzr;
      end;
end;
procedure RacunajPP(u: rec; pp: nizp);
var
  z: char;
  k, Un: integer;
begin
  Un := Len(u);
  for z := chr(0) to chr(255) do pp[z] := Un+1;
  for k := 1 to Un do pp[u[k]] := Un-k+1;
end;
function OdrediDSD(u: rec; uk, d: integer): integer;
var
  q1, q2: boolean;
  i, j, k, m, Un: integer;
begin
  Un := Len(u);
  q1 := true;
  while q1 and (d <= Un) do begin
    i := uk;
    q2 := true;
    while q2 and (i > 0) do begin
      j := uzr[i].poz-d;
      if (j > 0) and (uzr[i].zn <> u[j]) then q2 := false else dec(i);
```

```
    end;
    if q2 then q1 := false else inc(d);
  end;
  OdrediDSD := d;
end;
procedure RacunajDSD(u: rec);
var
  q: boolean;
  Uk, Un, d, i: integer;
begin
  Un := Len(u);
  dsd[0] := 1;
  d := 1;
  for uk := 1 to Un do begin
    d := OdrediDSD(u, uk, d);
    dsd[uk] := d;
  end;
  dsd[un+1] := dsd[un];
  for uk := 1 to Un do begin
    d := dsd[uk];
    q := true;
    while q and (d <= un) do begin
      i := uzr[uk].poz-d;
      if (i < 1) or (uzr[uk].zn <> u[i]) then q := false
      else begin
        inc(d);
        OdrediDSD(u, uk, d);
      end;
    end;
    dsd[uk] := d;
  end;
end;
begin
  Un := Len(u); Tn := Len(t);
  UrediUzorak(u);
  RacunajPP(u, pp);
  RacunajDSD(u);
  s := 0;
  while s <= tn-un do begin
    p := 1;
    while ((p <= un) and (uzr[p].zn = t[s+uzr[p].poz])) do inc(p);
    if (p > un) then writeln('Poklapaju se od pozicije ', s+1);
    s := s+max(dsd[p], pp[t[s+un+1]]);
  end;
end;
```