4. Şiruri de caractere

- Marea majoritate a celor preocupaţi de activitatea de programare sunt familiarizaţi cu şirurile de caractere întrucât aproape toate limbajele de programare includ **şirul** sau **caracterul** printre elementele predefinite ale limbajului.
- În prima parte a capitolului se va preciza **tipul de date abstract şir**.
- Apoi vor fi abordate **modalităţile majore de implementare**, prin intermediul **tablourilor** respectiv al **pointerilor**.
- În ultima parte a capitolului vor fi precizate câteva din **tehnicile de căutare** în şiruri.

4.1. Tipul de date abstract şir

- Un şir este o colecţie de caractere, spre exemplu "`Structuri de date`".
- În toate limbajele de programare în care sunt definite şiruri, acestea au la bază **tipul primitiv** `char`, care în afara literelor şi cifrelor cuprinde şi o serie de alte caractere.
- Se subliniază faptul că într-un şir, **ordinea caracterelor** contează. Astfel şirurile `"CAL"` şi `"LAC"` deşi conţin aceleaşi caractere sunt diferite.
- De asemenea, printr-un uşor **abuz** de notaţie se consideră că un *caracter* este interschimbabil cu *un şir* constând dintr-un singur caracter, deşi strict vorbind ele sunt de tipuri diferite.
- Asemeni oricărui tip de date abstracte, definirea precisă a **TDA şir** necesită:
    - Descrierea **modelului** său **matematic**,
    - Precizarea **operatorilor** care acţionează asupra elementelor tipului.
- Din punct de vedere **matematic**, elementele tipului de date abstract şir pot fi definite ca **secvenţe finite de caractere** $(c_1, c_2, ..., c_n)$ unde c_i este de tip caracter, iar n precizează lungimea secvenţei.
- Cazul în care n este egal cu zero, desemnează **şirul vid**.
- În continuare se prezintă un posibil set de operatori care acţionează asupra TDA şir.
- Ca şi în cazul altor structuri de date, există practic o libertate deplină în selectarea acestor operatori motiv pentru care setul prezentat are un caracter orientativ.

```
------------------------------------------------------------
TDA Şir

Modelul matematic:secvenţă finită de caractere.

Notaţii: s,sub,u - siruri;
         c - valoare de tip caracter;
         b - valoare booleană;
         poz,lung - întregi pozitivi.        [4.1.a]
```

```
Operatori:
      CreazaSirVid(s) - procedură ce creează şirul vid s;
      b:=SirVid(s) - funcţie ce returnează true dacă
            şirul este vid;
      b:=SirComplet(s) - funcţie booleană ce returnează
            valoarea true dacă şirul este complet;
      lung:=LungSir(s) - funcţie care returnează numărul
            de caractere ale lui s;
      poz:=PozitieSubsir(sub,s) - funcţie care returnează
            poziţia la care subşirul sub apare prima dată
            în s. Dacă sub nu e găsit în s, se returnează
            valoarea 0. Poziţiile caracterelor sunt numerotate
            de la stânga la dreapta începând cu 1;
      ConcatSir(u,s) - procedură care concatenează la
            sfârşitul lui u atâtea caractere din s, până
            când SirComplet(u) devine true;
      CopiazaSubsir(u,s,poz,lung) - procedură care-l face
            pe u copia subşirului din s începând cu poziţia
            poz, pe lungime lung sau până la sfârşitul lui s;
            dacă poz>LungSir(s) sau poz<1, u devine şirul
            vid;
      StergeSir(s,poz,lung) - procedură care şterge din s,
            începând cu poziţia poz, subşirul de lung
           caractere. Dacă  poz este invalid (nu aparţine
            şirului), s rămâne nemodificat;
      InsereazaSir(sub,s,poz) - procedură care inserează în s,
            începând de la poziţia poz, şirul sub;
      c:=FurnizeazaCarSir(s,poz) - funcţie care returnează
            caracterul din poziţia poz a lui s. Pentru poz
            invalid, se returnează caracterul nul;
      AdaugaCarSir(s,c) - procedură care adaugă caracterul
            c la sfârşitul şirului s;
      StergeSubsir(sub,s,poz) - procedură care şterge prima
            apariţie a subşirului sub în şirul s şi returnează
            poziţia poz de ştergere. Dacă sub nu este găsit, s
            rămâne nemodificat iar poz este poziţionat pe 0;
      StergeToateSubsir(s,sub) - şterge toate apariţiile
            lui sub în s.
------------------------------------------------------------
```

- Operatorii definiţi pentru un TDA-şir pot fi împărţiţi în două categorii:
  - Operatori **primitivi** care reprezintă un set minimal de operaţii strict necesare în termenii cărora pot fi dezvoltaţi operatorii nonprimitivi.
  - Operatori **nonprimitivi** care pot fi dezvoltaţi din cei anteriori.
- Această divizare este oarecum **formală** deoarece, de obicei e mai uşor să defineşti un operator neprimitiv direct decât să-l defineşti în termenii primitivelor.
  - Spre **exemplu** operatorul `InsereazăŞir` poate fi definit în termenii operatorilor `CreazăŞirVid` şi `AdaugăCar`.
  - Algoritmul este simplu: se construieşte un şir de ieşire temporar (`rezultat`) căruia i se adaugă pe rând:
    - (1) caracterele şirului sursă până la punctul de inserţie
    - (2) toate caracterele şirului de inserat (`subşir`)
    - (3) toate caracterele şirului sursă de după punctul de inserţie.

- Şirul astfel construit înlocuieşte şirul iniţial (`sursa`) (secvenţa [4.1.b]).

```
------------------------------------------------------------------
/*Implementarea operatorului Inserează_Şir utilizând operatorii
Crează_Sir_Vid, Adaugă_Car_Sir şi Furnizeaza_Car_Sir */

void insereaza_sir(tipsir subsir, tipsir* sursa,tipindice p)
  /*insereaza subsirul în sursa între pozitiile p si p+1*/
{
    tipsir rezultat; tipindice i;

  if ((p<1) || (p>lung_sir(*sursa))) ;
      /*eroare (poziţie ilegală în inserţie)*/
    else
      {                                          /*[4.1.b]*/
        creaza_sir_vid(rezultat);
        for( i= 1; i <= p-1; i++)
          adauga_car_sir(rezultat,
furnizeaza_car_sir(*sursa,i));
        for( i=1; i <= lung_sir(subsir); i++)
          adauga_car_sir(rezultat,
furnizeaza_car_sir(subsir,i));
        for( i=p; i <= lung_sir(*sursa); i++)
          adauga_car_sir(rezultat,
furnizeaza_car_sir(*sursa,i));
        creaza_sir_vid(*sursa);
        for( i=1; i <= lung_sir(rezultat); i++)
          adauga_car_sir(*sursa,
furnizeaza_car_sir(rezultat,i));
      }
}
------------------------------------------------------------------
```

4.2. Implementarea tipului de date abstract şir

- Sunt cunoscute două tehnici majore de implementare a şirurilor:
  - Implementarea bazată pe **tablouri**
  - Implementarea bazată pe **pointeri**

4.2.1. Implementarea şirurilor cu ajutorul tablourilor

- Cea mai simplă implementare a TDA-şir se bazează pe două elemente:
  - (1) Un **întreg** reprezentând lungimea şirului
  - (2) Un **tablou** de caractere care conţine şirul propriu-zis.
- Un exemplu de astfel de implementare este următorul [4.2.1.a].

```
------------------------------------------------------------------
/* Exemplu de implementare a TDA Sir utilizând tablouri */

enum { lungime_max = 100};
typedef unsigned char tiplungime;

typedef unsigned char tipindice;
                                          /*[4.2.1.a]*/
```

```
typedef struct tipsir {
  tiplungime lungime;
  char sir[lungime_max];
} tipsir;
tipsir s;
----------------------------------------------------------------
```

- Acest mod de implementare al şirurilor **nu** este unic.
- Se utilizează **tablouri** întrucât tablourile ca şi şirurile sunt **structuri liniare**.
- Câmpul `lungime` este utilizat deoarece şirurile au lungimi diferite în schimb ce tablourile au lungimea fixă.
- Implementarea se poate dispersa de câmpul `lungime`, caz în care se poate utiliza un caracter convenit pe post de **santinelă de sfârşit** (**marker**).
    - În această situație operatorul `LungimeŞir` va trebui să contorizeze într-o manieră liniară caracterele până la detectarea markerului.
    - Din acest motiv este preferabil ca lungimea să fie considerată un element **separat** al implementării.
- În contextul implementării şirurilor cu ajutorul tablourilor se defineşte noțiunea de **şir complet**
    - **Şirul complet** este şirul care are lungimea egală cu `lungime_maximă`, adică egală cu **dimensiunea** tabloului definit spre a-l implementa.
    - Şirurile implementate în acest mod **nu** pot depăşi această lungime, motiv pentru care în acest context operează operatorul boolean `Şir_Complet`.
- În accepțiunea modelului anterior prezentat, în continuare se prezintă o implementare a operatorilor primitivi [4.2.1.b,c,d,e].

```
----------------------------------------------------------------
void creaza_sir_vid(tipsir* s)                       /*O1*/
  {
    s->lungime= 0;                                 /*[4.2.1.b]*/
  }
----------------------------------------------------------------
tiplungime lung_sir(tipsir* s)                      /*O1*/
  {
    tiplungime lung_sir_result;
    lung_sir_result= s->lungime;                    /*[4.2.1.c]*/
    return lung_sir_result;
  }
----------------------------------------------------------------
char furnizeaza_car(tipsir s,tipindice poz)
  {                                               /*O1*/
    char furnizeaza_car_result;
    if ((poz<1) || (poz>s.lungime))
        {                                         /*[4.2.1.d]*/
          /*eroare(poziție incorectă);*/
          furnizeaza_car_result= '/0'; /*caracterul vid*/
        }
      else
        furnizeaza_car_result= s.sir[poz-1];
    return furnizeaza_car_result;
  }
```

```
---------------------------------------------------------------
void adauga_car_sir(tipsir* s, char c)            /*O1*/
  {
    if (s->lungime==lungime_max) ;
       /*eroare(se depaşeşte lungimea maximă a şirului)*/
      else
        {
          s->lungime= s->lungime+1;            /*[4.2.1.e]*/
          s->sir[s->lungime-1]= c;
        }
  }
---------------------------------------------------------------
```

- Se observă că toate aceste operații rulează în O*(1)* unități de timp indiferent de lungimea şirului.

- Procedurile `CopiazăSubŞir`, `ConcatŞir`, `ŞtergeŞir` şi `InsereazaŞir` se execută într-un interval de timp liniar O*(n)*, unde *n* este după caz lungimea subşirului sau a şirului de caractere.

  - Spre **exemplu** procedura `CopiazăSubşir (u,s,poz,lung)` returnează în `u` subşirul având `lung` caractere începând cu poziția `poz`.

  - Accesul la elementele subşirului se realizează direct `(s.şir[poz], s.şir[poz+1],...,s.şir[poz+lung-1])`, astfel încât consumul de timp al execuției este dominat de mutarea caracterelor [4.2.1.f].

```
---------------------------------------------------------------
/*Implementarea operatorului CopiazăSubşir*/      /*O(n)*/

void copiazasubsir(tipsir* u, tipsir* s, tiplungime poz,
tiplungime lung)
{
    tiplungime indexsursa,indexcopiere;

  if ((poz<1) || (poz>s->lungime))
      u->lungime= 0;
    else
      {                                          /*[4.2.1.f]*/
        indexsursa= poz-1;
        indexcopiere= 0;
        while ((indexsursa<s->lungime) &&
                 (indexcopiere<u->lungime))  {
            indexsursa= indexsursa+1;
            indexcopiere= indexcopiere+1;
            u->sir[indexcopiere-1]= s->sir[indexsursa-1];
          }    /*WHILE*/
        u->lungime= indexcopiere;
      }
}
---------------------------------------------------------------
```

  - Se observă faptul că în interiorul buclei **`WHILE`** există 3 atribuiri, care pentru o lungime `lung` a sub şirului determină `3*lung + 3` atribuiri.

  - Se observă de asemenea că procedura `CopiazăSubşir` este liniară în raport cu lungimea `lung` a subşirului.

- Un alt exemplu îl reprezintă implementarea funcției `PozițieSubşir` [4.2.1.g].

```
---------------------------------------------------------------
/*Implementarea operatorului PozițieSubşir*/

tiplungime pozitiesubsir(tipsir* sub,tipsir* s)
 {
     tipindex mark, /*index pentru punctul de start al unei
comparatii*/
     indexsub,           /*index subsir*/
     indexsir;           /*index sir*/
     tiplungime poz;     /*pozitia lui sub în s*/
     boolean stop;  /*devine adevarat când elementele
             corespunzatoare din s si sub nu sunt egale*/

   tiplungime pozitiesubsir_result;
   indexsir= 1;
   poz= 0;
   do {
     indexsub= 1;
     mark= indexsir;
     stop= false;
     while ((! stop) && (indexsir<=s->lungime) &&
              (indexsub<=sub->lungime))
       if (s->sir[indexsir-1]==sub->sir[indexsub-1])
           {
             indexsir= indexsir+1;
             indexsub= indexsub+1;             /*[4.2.1.g]*/
           }
         else
           stop= true;  /*WHILE*/
     if (indexsub>sub->lungime)
            poz= mark; /*potrivire*/
       else
         indexsir= mark+1;
   } while (!((poz>0) ||
           (indexsir+sub->lungime-1>s->lungime)));
   pozitiesubsir_result= poz;
   return pozitiesubsir_result;
 }
---------------------------------------------------------------
```

- Complexitatea funcției `PozițieSubşir(sub,s)` este $O(lungime * s.lungime)$ unde `lungime` este lungimea modelului iar `s.lungime` este lungimea şirului în care se face căutarea.

- Există însă şi alte metode mult mai **performante** de căutare care fac obiectul unui paragraf ulterior.

4.3. Tehnici de căutare în şiruri

- Una din operațiile cele mai frecvente care se executată asupra şirurilor este **căutarea**.

- Întrucât **performanța** acestei operații are o importanță covârşitoare asupra marii majorități a prelucrărilor care se realizează într-un sistem de calcul, studiul **tehnicilor de căutare performante** reprezintă o **preocupare permanentă** a cercetătorilor în domeniul programării.

* În cadrul acestui paragraf vor fi trecute în revistă câteva dintre cele mai cunoscute tehnici de căutare în şiruri de caractere.

4.3.1. Căutarea de şiruri directă

* O manieră frecvent întâlnită de căutare este aşa numită **căutare de şiruri directă** ("**string search**").
* **Formularea problemei**:
    * Se dă un tablou `s` de `n` elemente şi un tablou `p` de `m` elemente, unde `0<m<n` declarate astfel [4.3.2.a]:

```
------------------------------------------------------------
typedef unsigned char boolean;
char s[n];
char p[m];                              /*[4.3.2.a]*/
------------------------------------------------------------
```

* Căutarea se şiruri are drept scop stabilirea **primei apariţii** a lui `p` în `s`.
* De regulă, `s` poate fi privit ca un **text**, iar `p` ca un cuvânt **model** ("pattern") a cărui **primă apariţie** se caută în textul `s`.
* Aceasta este o operaţie **fundamentală** în toate sistemele de prelucrare a textelor şi în acest sens este de mare interes găsirea unor algoritmi cât mai eficienţi.
* Cea mai **simplă** metodă de căutare este aşa numita **căutare de şiruri directă.**
* **Rezultatul** unei astfel de căutări este un indice `i` care precizează apariţia unei **coincidenţe** între model şi şir.
* Acest lucru este formalizat de predicatul **`P`**`(i,j)` [4.3.2.b].

$$\mathbf{P}(i,j) \Leftarrow \mathbf{A}_k : 0 \le k < j : s_{i+k} = p_k \qquad [4.3.2.b]$$

----

* Predicatul **`P`**`(i,j)` precizează faptul că există o **coincidenţă** între:
    * Şirul `s` (începând cu indicele `i`)
    * Şirul `p` (începând cu indicele `0`) pe o lungime de `j` caractere (fig. 4.3.2.a).

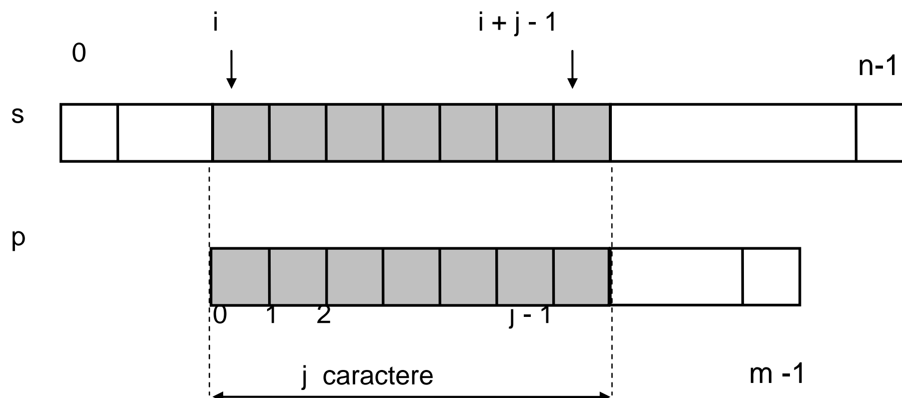


Fig.4.3.2.a. Reprezentarea grafică a predicatului **`P`**`(i,j)`

- Este evident că indexul `i` care rezultă din **căutarea directă de şiruri** trebuie să satisfacă predicatul **P**`(i,m)`.
- Această condiţie **nu** este însă suficientă.
- Deoarece căutarea trebuie să furnizeze **prima** apariţie a modelului, **P**`(k,m)` trebuie să fie fals pentru toţi $k < i$.
- Se notează această condiţie cu **Q**`(i)` [4.3.2.c].

$$Q(i) = A_k:\ 0 \le k < i:\ \sim P(k,m) \qquad [4.3.2.c]$$

- Punerea problemei sugerează formularea căutării directe de şiruri ca şi o iteraţie de comparaţii redactată în **termenii** predicatelor **Q** respectiv **P**.
- Astfel implementarea lui **Q**`(i)` conduce la secvenţa [4.3.2.d]:

```
{Implementarea predicatului Q(i)}
i:= -1;
REPEAT
  i:= i+1;                                       [4.3.2.d]
  gasit:= P(i,m)
UNTIL gasit OR (i=n-m);
```

- Calculul lui **P** rezultă în continuare ca şi o iteraţie de comparaţii de caractere individuale.
- Rafinarea secvenţei anterioare conduce de fapt la **implementarea căutării directe de şiruri** ca o repetiţie într-o altă repetiţie [4.3.2.e].

```
/*{Implementarea căutării directe de şiruri*/

boolean cautaredirecta(int* poz)
  {
     int i,j;   /*i parcurge caracterele din sir,
                      j parcurge caracterele din model*/
     boolean cautaredirecta_result;
     i= 1;
     do {                                          /*[4.3.2.e]*/
       i= i+1; j= 0;    /*Q(i)*/
       while ((j<m) && (s[i+j]==p[j]))
           j= j+1;  /*P(i,m)*/
     } while (!((j==m)||(i==n-m)));
     *poz=i;
     cautaredirecta_result= j==m;
     return cautaredirecta_result;
  }
```

- Termenul `j = m` din condiţia de terminare, corespunde lui **găsit** deoarece el implică **P**`(i,m)`.
- Termenul `i=n-m` implică **Q**`(n-m)`, deci **non** existenţa vreunei coincidenţe în cadrul şirului.
- **Analiza căutării de şiruri directe.**

- Algoritmul lucrează destul de **eficient** dacă se presupune că nepotrivirea în procesul de căutare apare cel mult după câteva comparaţii în cadrul buclei interioare.
- Astfel pentru un set de 128 de caractere se poate presupune că nepotrivirea apare după inspectarea a 1 sau 2 caractere.
- Cu toate acestea în cazul cel mai nefavorabil, degradarea performanţei este îngrijorătoare.
    - Astfel dacă spre exemplu şirul `s` este format din `n-1` caractere 'A' urmate de un singur 'B',
    - Iar modelul constă din `m-1` caractere 'A' urmate de un 'B',
    - În acest caz este necesar un număr de comparaţii de ordinul `n * m` pentru a găsi coincidenţa la sfârşitul şirului.
    - Din fericire există metode care îmbunătăţesc radical comportarea algoritmului în această situaţie.
    - Tehnicile de căutare care sunt prezentate în continuare materializează acest deziderat.

4.3.2. Căutarea de şiruri Boyer-Moore

- Metoda de căutare inventată în 1975 de R.S. Boyer şi J.S. Moore este una dintre cele mai ingenioase şi în acelaşi timp una dintre cele mai performante.
- Căutarea BM, după cum mai este numită, este bazată pe ideea **neobişnuită** de a începe compararea caracterelor de la **sfârşitul** modelului şi **nu** de la început.
- Modelul este **precompilat** în prealabil într-un tablou `d`.
    - Astfel, pentru fiecare caracter `x` al setului de caractere, se notează cu d_x distanţa dintre cea mai din dreapta apariţie a lui `x` în cadrul modelului şi sfârşitul modelului (fig.4.3.4.a.).

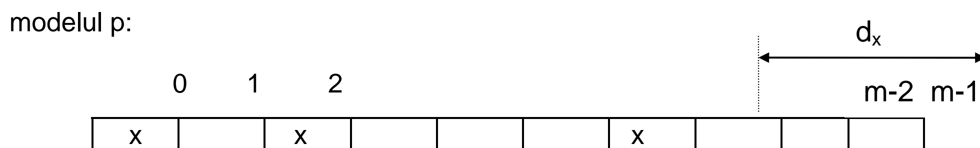


Fig. 4.3.4.a. Determinarea valorii d_x corespunzătoare caracterului `x` al modelului

- Valoarea d_x se trece în tabloul `d` în poziţia corespunzătoare caracterului `x`.
- Pentru caracterele care **nu** sunt în model respectiv pentru ultimul caracter al modelului, d_x se face egal cu **lungimea totală** a modelului.
- În continuare, se presupune că în procesul de **comparare** al şirului cu modelul a apărut o **nepotrivire** între caracterele corespunzătoare.
    - În această situaţie modelul poate fi imediat **deplasat** spre dreapta cu $d[p_{m-1}]$ poziţii, valoare care este de regulă mai mare ca 1.

- Se precizează faptul că p_{m-1} este caracterul din şirul baleat `s`, corespunzător **ultimului** caracter al modelului la momentul considerat, **indiferent** de locul în care s-a constatat nepotrivirea (figura 4.3.4.b.).

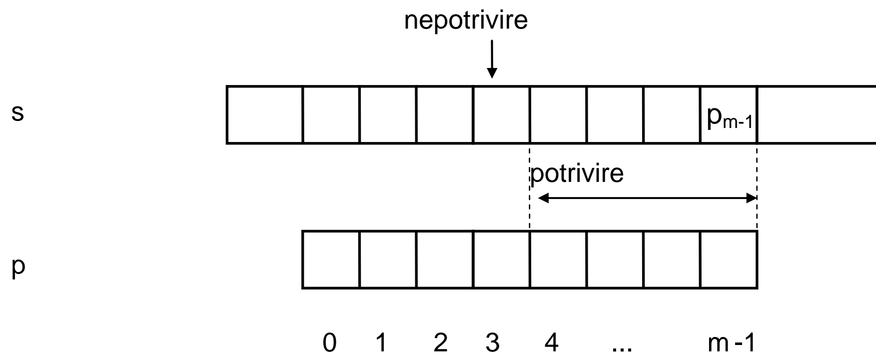
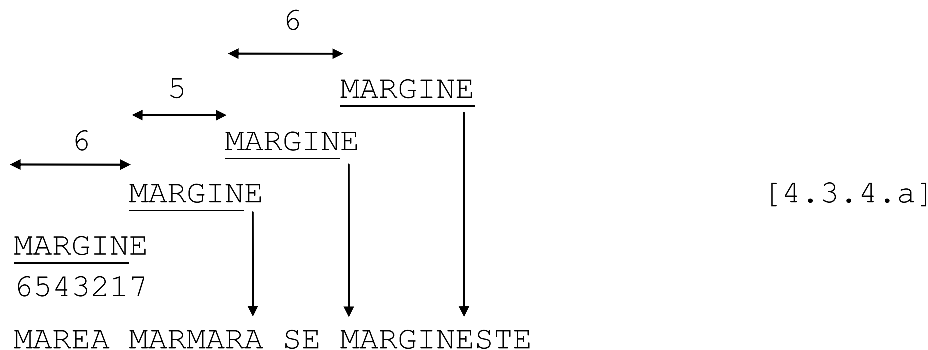


Fig. 4.3.4.b. Comparare şir - model pentru determinarea lui p_{m-1}

- Dacă caracterul p_{m-1} **nu** apare în model, deplasarea este mai mare şi anume cu întreaga lungime a modelului.
- Exemplul din secvenţa [4.3.4.a.] evidenţiază acest proces.



- Implemetarea căutării Boyer-Moore apare în secvenţa [4.3.4.c.]

```
/*căutare Boyer-Moore */
  i= m; j= m;
  while ((j>0) && (i<n))
    {      /*Q(i-m)*/                         /*[4.3.4.c]*/
    j= m; k= i;
    while ((j>0) && (s[k-1]==p[j-1]))
      {      /*P(k-j,j)&(k-j=i-m)*/
        k= k-1;
        j= j-1;
      }
    i= i+d[s[i-1]];
}      /*cautare Boyer-Moore*/
```

- Indicii implicaţi satisfac următoarele relaţii: `0 < j < m` şi `0 < i, k < n`.

- Astfel **terminarea** algoritmului cu `j = 0` implică o **potrivire** începând de la poziția `k` spre dreapta, de `m` caractere, unde `k = i - m`.
- **Terminarea** algoritmului cu `j > 0` implică `i = n` element care indic **absența** potrivirii.

- Programul următor [4.3.4.d] implementează strategia Boyer-Moore într-un context mai general.

```
------------------------------------------------------------------
/* Căutarea Boyer-Moore -varianta C */

typedef unsigned char boolean;
#define true          (1)
#define false         (0)

enum { mmax = 100  /*lungime maxima model*/,
      nmax = 200} /*lungime maxima sir sursa*/; /*[4.3.4.c]*/
int m/*lungime model*/,n/*lungime sir*/;
char p[mmax];          /*model*/
char s[nmax];          /*sir*/
int d[256];        /*tabela de deplasari*/
boolean cautarebm(int* poz)
  {
     int i,j,k;

    /*citire sir;    n este lungimea curenta a sirului
    /*citire model;  m este lungimea curenta a modelului*/
     boolean cautarebm_result;
     for( i=0; i <= 255; i++) d['i']= m; /*compilare model*/
     for( j=0; j <= m-2; j++) d[p[j]]= m-j-1;
     i=m; j=m;   /*cautare model*/
     while ((j>0) && (i<=n))
       {
         j= m;
         k= i;
         while ((j>0) && (s[k-1]==p[j-1]))
           {
             k= k-1;
             j= j-1;
           }
         if (j>0)
          i=i+d[s[i-1]];
       }
     *poz=i-m; /*poz=k*/
     cautarebm_result= j==0;
     return cautarebm_result;
  }
------------------------------------------------------------------
```

- **Analiza căutării Boyer-Moore**

- Autorii căutării BM, au demonstrat proprietatea remarcabilă că în **toate** cazurile, cu excepția unora special construite, **numărul** de **comparații** este substanțial mai **redus** decât `n`.

- În cazul cel mai **favorabil** când ultimul caracter al modelului nimereşte întotdeauna în şir peste un caracter diferit de cele ale modelului, **numărul** de **comparații** este `n/m`.

4.4. Rezumat

- Un **şir** este o colecţie de caractere. În toate limbajele de programare în care sunt definite şiruri, acestea au la bază tipul primitiv `char`.
- Asemeni oricărui tip de date abstracte, **definirea TDA şir** necesită pe de o parte descrierea **modelului său matematic**, iar pe de altă parte precizarea **operatorilor** care acţionează asupra elementelor tipului.
- Cea mai cunoscută modalitate de implementare a TDA şir o constiuie **tablourile liniare**.
- Dintre operaţiile cele mai frecvente care se execută asupra şirurilor de caractere este **căutarea**.
- Cea mai frecventă metodă de căutare este **căutarea de şiruri directă ("string search").**
- O metodă de căutare cu performanţe mult mai bune este **căutarea de şiruri Boyer-Moore**.

4.5. Exerciţii

1) Se cere să se precizeze *modelul matematic* şi *setul de operatori* pentru *TDA şir de caractere.*

2) Se cere să se realizeze o implementare bazată pe tablouri a următorilor operatori: *CreazăŞirVid, LungŞir, InsereazăŞir, ConcatŞir, PoziţieSubşir*.

3) Se cere să se redacteze o funcţie care implementează *căutarea directă de şiruri*.

4) Care este *principiul căutării Boyer-Moore*? Cum se realizează *precompilarea modelului*?

5) Se cere să se redacteze o funcţie care implementează *căutarea de şiruri Boyer-Moore*.

6) Se cere să se realizeze un program C care:

- citeşte un şir de caractere `s` de la tastatură
- citeşte un şir de caractere model `p`
- caută în şirul `s` pe `p` utilizând căutarea directă de şiruri
- caută în şirul `s` pe `p` utilizând căutarea de şiruri BM

Obs. Se vor utiliza funcţiile dezvoltate în exerciţiile anterioare.