Fondamenti dell'Informatica
Prof. P.Mentrasti
Anno Accademico 2002-2003

Pattern Matching
(Ricerca di una parola in un testo)

Tesina a cura di:

Silvia Di Silvio

Stefano Caroselli

8 Gennaio 2003

Algoritmo di Boyer - Moore

L'algoritmo di Boyer – Moore è uno dei più efficienti algoritmi usati per la ricerca di un pattern all'interno di un testo perché particolarmente adatto se si ha a disposizione un alfabeto di cardinalità elevata e un testo abbastanza lungo.
Questo algoritmo deve la sua efficienza all'analisi preliminare del pattern dovuta all'uso di due euristiche che gli permette di saltare alcuni confronti inutili che non portano ad alcuna soluzione.
Ma vediamole in dettaglio:

L'euristica del carattere discordante

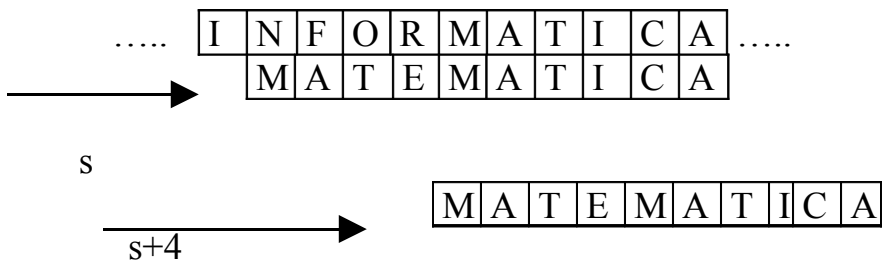
Ad un generico passo dell'algoritmo ci si troverà con uno spostamento s dall'inizio del testo. Si procederà quindi con il confronto del pattern con una porzione di testo (Il confronto si fa a partire da destra verso sinistra) cioè con il confronto di P[1..m] e di T[s+j..s+m]
Se P[m]=T[s+m],.....fino a P[j]<>T[s+j] allora T[s+j] sarà il carattere discordante.
Precedentemente ci saremmo memorizzati in un vettore, per ogni lettera dell'alfabeto, in quale posizione si presenta nel pattern con le seguenti accortezze:

- *se la lettera dell'alfabeto in uso non si presenta nel pattern metteremo nella rispettiva posizione 0*
- *se la lettera dell'alfabeto in uso compare più volte nel pattern memorizzeremo in questo vettore, nella rispettiva posizione, il valore che identifica la posizione dell'ultima occorrenza di quella particolare lettera*

In questo modo, se indichiamo con k la posizione di una lettera nel pattern, questo potrà valere o 0 o k con k:=1..m
e dopo ogni confronto potremmo effettuare uno spostamento di j-k senza perdita di informazioni infatti:

1. se k=0
   Il carattere discordante non si presenterà nel pattern, così si può incrementare con successo s di j senza saltare alcuno spostamento valido

   Esempio:
   R=T[s+4]
   K=0
   j-k=j=4



2. se j>k
   L'occorrenza più a destra del carattere discordante è nel pattern a sinistra della posizione j, così che j-k>0 e il pattern deve essere spostato di j-k caratteri a destra prima che il carattere discordante del testo combaci con qualche carattere del pattern.

3. se j<k
   In questo caso si dovrebbe effettuare uno spostamento negativo (cioè verso sinistra) ma questo non mi fa progredire nella ricerca!A questo punto verrà in soccorso la seconda euristica che ci garantirà un progresso nella ricerca.
   (Quando si dovrà effettuare uno spostamento si prenderà il massimo degli spostamenti suggeriti da ogni euristica e, quello dell'euristica del buon suffisso è sempre positivo!)

L'euristica del buon suffisso

Algoritmo

```
Program Boyer_Moore_Matcher;

uses crt;

const dim = 30;

type vet = array[0..dim] of integer;
     vetc= array['0'..'z'] of integer;
     pattern = string[dim];

var i,j,m,n,s : integer;
    T         : string;
    P         : pattern;
    l         : vetc;
    g,q       : vet;
    trovato   : boolean;


Function max(x,y:integer):integer;
  begin
    if x>y then max:=x else max:=y;
  end;

Function inv(s:string;m:integer):string;
  var i:integer;
  begin
    inv:=s;
    for i:=1 to m do inv[i]:=s[m-i+1];
  end;
```

```
Procedure UltimaOcc(P:pattern; m:integer; var l:vetc);
  var a:char; j:integer;
  begin
    for a:='0' to 'z' do l[a]:=0;
    for j:=1 to m do l[P[j]]:=j;
  end;


Procedure Prefisso(P:pattern; var pr:vet);
  var k,q:integer;
  begin
    pr[1]:=0;
    k:=0;
    for q:=2 to m do
      begin
        while (k>0) and (P[k+1]<>P[q]) do k:=pr[k];
        if (P[k+1]=P[q]) then k:=k+1;
        pr[q]:=k;
      end;
  end;


Procedure BuonSuff(P1:pattern; m:integer; var g:vet);
var i,j:integer;
    q1,q2:vet;
    P2:pattern;
    begin
    Prefisso(P1,q1);
    P2:=inv(P1,m);
    Prefisso(P2,q2);

    for j:=0 to m do g[j]:=m-q1[m];
    for i:=1 to m do
      begin
        j:=m-q2[i];
        if (g[j]>i-q2[i]) then g[j]:=i-q2[i];
      end;
  end;


Begin
  clrscr;
  writeln('Algoritmo di Boyer-Moore');
  writeln('Inserire il testo da esaminare [numeri o lettere, max 255 caratteri]
: ');
    readln(T);
  writeln('Inserire la chiave di ricerca [max ',dim,' caratteri] : ');
    readln(P);
  trovato:=false;
  n:=length(T);
  m:=length(P);
  UltimaOcc(P,m,l);
  BuonSuff(P,m,g);
  s:=0;
  while (s<=n-m) do
    begin
      j:=m;
      while (j>0) and (P[j]=T[s+j]) do j:=j-1;
      if (j=0)
        then begin
             writeln('La chiave appare con spostamento : ',s);
```

```
            s:=s+g[1];
            trovato:=true;
            end
        else s:=s+max(g[j], j-l[T[s+j]]);
    end;
if trovato=false then writeln('Il pattern non compare nel testo');
  readln;
End.
```

Primo metodo per mettere l'algoritmo in parallelo

Questo è un semplice metodo per simulare un calcolo in parallelo tramite un ciclo *for*: una volta deciso il numero di processori, ad ogni passo del ciclo una porzione del testo è esaminata da un processore. Ad esempio se vi sono m processori, ognuno esaminerà 1/m dell'intero testo.

```
Begin {main}
  clrscr;
  writeln('Algoritmo di Boyer-Moore');
  writeln('Inserire il testo [numeri o lettere, max 255 caratteri]:');
  readln(T);
  writeln('Inserire la chiave di ricerca [max ',dim,' caratteri]:');
  readln(P);
  n:=length(T);
  m:=length(P);
  UltimaOcc(P,m,l);
  BuonSuff(P,m,g);
  write('Numero di processori [da 1 a ',n div m,']:');
  readln(proc);
  a:=n div proc;
  for i:=1 to proc do   {□ questo "for" simula il calcolo in parallelo}
    begin
      s:=a*(i-1);
      while (s<=min(n-m,a*i)) do
        begin
          j:=m;
          while (j>0) and (P[j]=T[s+j]) do j:=j-1;
          if (j=0)
            then
              begin
                writeln(' - La chiave appare con spostamento:',s);
                s:=s+g[1];
              end
            else s:=s+max(g[j], j-l[T[s+j]]);
        end;
      readln;
    end;
End.
```

Secondo metodo per mettere l'algoritmo in parallelo

L'idea alla base di questo metodo è quella di identificare all'interno del testo alcuni punti in cui sia possibile effettuare un taglio nel testo al fine di avere p processori che ricercano l'intero pattern ma in una porzione di esso.
I tagli però devono essere fatti opportunamente perchè se si procedesse con un taglio casuale, potrebbe succedere che il pattern si presenti a cavallo di quel taglio e che quindi non venga riconosciuto da nessun processore.
Grazie all'euristica del carattere discordante si è in grado di sapere quali sono le lettere che sono presenti nel testo ma non nel pattern.
Bisognerà quindi andare a vedere ognuna di queste lettere in che posizione si presenta nel testo e memorizzarla in un vettore.
A questo punto il vettore apparirà in modo disordinato.
Bisognerà quindi ordinarlo con un sort e si avranno così in ordine i vari punti di taglio.
L'utente potrà scegliere un numero di processori da far lavorare in parallelo pari alla lunghezza di questo vettore.
L'algoritmo sarà modificato come segue nel main e verrà aggiunta la procedura di un sort.
Questo metodo non è buono per pattern che presentano tutte le lettere dell'alfabeto in uso e inoltre aumenta la complessità dell'algoritmo che, senza la messa in parallelo, ha un costo pari a *O((n-m+1)m+card)*
e invece così andrà prima inizializzato il vettore proc (costa *n*) poi si ha un for dentro l'altro (quindi si ha *card * n*) e inoltre si ha il costo del sort. Qui verrà usato il selection sort.

```
Begin (*MAIN*)
  clrscr;
  writeln('Algoritmo di Boyer-Moore');
  writeln('Inserire il testo da esaminare [numeri o lettere, max 255 caratteri]
: ');
    readln(T);
  writeln('Inserire la chiave di ricerca [max ',dim,' caratteri] : ');
    readln(P);
  trovato:=false;
  n:=length(T);
  m:=length(P);
  UltimaOcc(P,m,l);
  BuonSuff(P,m,g);
  for i:=1 to n do proc[i]:=100;
  proc[0]:=0;
  h:=0;
  for b:='0' to 'z' do if l[b]=0 then begin
                              for j:=1 to n do if t[j]=b then begin
                                                              h:=h+1;
                                                              proc[h]:=j;
                                                              end;
                                        end;
writeln('Il numero max di processori che possono lavorare in parallelo e''
',h+1);
writeln('Quanti processori devono lavorare in parallelo?');
```

```
readln(pr);
sort(proc,h);
if pr>h+1 then writeln('ERRORE')
          else begin
               c:=h div pr;
               w:=c;
               while proc[w]<=n do BEGIN

  s:=proc[w-c];
  while (s<=proc[w]-m) do
    begin
      j:=m;
      while (j>0) and (P[j]=T[s+j]) do j:=j-1;
      if (j=0)
        then begin
             writeln('La chiave appare con spostamento : ',s);
             s:=s+g[1];
             trovato:=true;
             end
        else s:=s+max(g[j], j-l[T[s+j]]);
    end;

    w:=w+c;
                                     END;
                end;

if trovato=false then writeln('Il pattern non compare nel testo');
  readln;
End.



Procedure SORT (var x:vet; var n:integer);
var i, j: integer;

Procedure SCAMBIA (var a, b:integer);
var c:integer;
begin
c:=a;
a:=b;
b:=c;
end;

begin
for i:=1 to n-1 do begin
                   for j:=i+1 to n do
                   if x[i]>x[j] then scambia (x[i],x[j])
                   end
end;
```

Programma in C

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define dim 100
#define max(x,y) ( (x)>(y) ? x : y )

typedef char str[dim];
int q[dim],g[dim],L[256];
str S,T,P;

void UltimaOcc(str P, int m, int l[])          // P=pattern
  { for (int a=0; a<=255; a++) l[a]=0;         // l=ult.occ. (funz.lambda)
    for (int j=0; j<m; j++) {char c=P[j]; l[c]=j+1;} }

void Prefisso(str P, int m, int pr[])          // pr=prefisso (funz.pi-greco)
  { pr[1]=0;  int k=0;
    for (int q=2; q<=m; q++)
      { while ((k>0) && (P[k]!=P[q-1])) k=pr[k];
        if (P[k]==P[q-1]) k++;
        pr[q]=k; }  }

void BuonSuff(str P1, int m, int h[])      // P1=pattern  q1=prefisso di P1
  { int q1[dim],q2[dim];                   // P2=inverso  q2=prefisso di P2
    Prefisso(P1,m,q1);
    str P2; strcpy(P2,P1); strrev(P2);     // creazione dell'inverso
    Prefisso(P2,m,q2);
    for(int j=0; j<=m; j++) h[j]=m-q1[m];
    for(int l=1; l<=m; l++)
      { int j=m-q2[l]; if (h[j]>l-q2[l]) h[j]=l-q2[l]; }   }

int main()
{ printf("Algoritmo di Boyer-Moore \n");
  printf("Inserire il testo da esaminare
            [max 255 caratteri, senza spazi vuoti] : \n");
    scanf("%s",&T);
  printf("Inserire la chiave di ricerca [max ",dim," caratteri] : \n");
    scanf("%s",&P);
  int n=strlen(T);
  int m=strlen(P);
  UltimaOcc(P,m,L);
  BuonSuff(P,m,g);

  int proc;
  printf("Inserire il numero di processori da 1 a %d : ",n);
    scanf("%d",&proc);
  int size=n/proc;
  for(int i=0; i<proc; i++)          // ciclo simulante la parallelizzazione
   { int s=i*size;
     while (s < min((i+1)*size, n-m+1))
       { int j=m;
         while ((j>0) && (P[j-1]==T[s+j])) j--;
         if (j==0)
            {printf(" - la chiave compare con spostamento %d \n",s+1);
         s+=g[1];}
         else s+=max(g[j], j-L[T[s+j]]);  }    }
  getchar();
  return 0; }
```