P1 - Esercizio 5

dal post di discussione su Moodle aperto da Filè

Estratto Approfondito

Oggi 04 nov 2014, nella seconda ora del laboratorio, ho presentato una soluzione dell'esercizio 5 con gli invarianti dei 2 cicli annidati.

La soluzione presentata è la seguente:

```
#include<iostream>
using namespace std;

int main()
{   //PRE
  int T[100], P[20], dimT, dimP;
  cin >> dimT >> dimP;

  int i=0;
  while(i<dimT)          //riempio T[0..dimT-1]
  {
    cin >> T[i];
    i++;
  }

  int j=0;
  while(j<dimP)          //riempio P[0..dimP-1]
  {
    cin >> P[j];
    j++;
  }


  int bestIn=-1, bestMIS=dimP;                    //"bestIn=-1" posiz inesistente; "bestMIS=dimP" basta che sia abbastanza grande (>dimP-1)
  i=0;                                            //reset indice i per scorrere T
  while(i<dimT-dimP+1)              //R            //in alternative "(i<=dimT-dimP)"
  {
    int mis=0, m=i;                               //variabili ausiliarie
    bool fine=false;                              //ferma il ciclo non appena ho due mis consecutivi
    j=0;                                          //reset indice j per scorrere P
    while(j<dimP && !fine && m<dimT)          //R1
    {
      if(P[j]==T[m])
      {
        j++;
        m++;
      }
      else
        if(m+1<dimT && P[j]==T[m+1])
        {
          j++;
          m=m+2;
          mis++;
        }
        else
          fine=true;
    }
```

```
    if(j==dimP && mis<bestMIS)
    {
      bestIn=i;
      bestMIS=mis;
  }
    i++;
  }

  cout<<bestIn<<" "<<bestMIS<<endl;
}  //POST



/*
PRE=(cin contiene dimT e dimP (0<dimT<=100 e 0<dimP<=20), seguiti da dimT+dimP valori interi qualsiasi).

R=(ho considerato i punti di inizio T[0..i-1]) && (0<=i<=dimT):
- (bestIn=-1 => non esistono M1MIS che iniziano in [0..i-1])
- (bestIn!=-1 => bestIn e bestMIS sono l'inizio e il MIS del match migliore che inizia in T[0..i-1]).

R1=(!fine <=> a partire da i, in T[i..m-1] ho un M1MIS di P[0..j-1] con MIS=mis).

POST=(se c'è almeno un M1MIS di P in T, il programma stampa l'inizio del M1MIS a MIS minimo (e quello che inizia prima se ce ne fossero diversi minimi)
e il valore del suo MIS) && (se non esiste M1MIS di P in T allora il programma scrive su cout "nessun M1MIS").
*/
```

Il codice è disponibile anche qui:
http://pastebin.com/8R1WnTCk

A lezione ho anche detto che questo programma, sebbene calcoli la soluzione GIUSTA per tutti i test (ed in particolare per quello automatico), ha un errore: considera dei match che non dovrebbe considerare.
Ho invitato gli studenti a scoprire dove sta l'errore.

PRIMA:

Penso di aver trovato il problema di cui lei parlava oggi a lezione riguardante l'esercizio numero 5, ma non ne sono del tutto sicuro. All'interno del primo while lei ha dichiarato le variabili j, mis, m e fine, in questo modo dalla seconda iterazione del ciclo in poi verranno azzerate tutte le variabili, e inoltre verranno dichiarate nuovamente occupando più spazio di memoria.

La congettura non è corretta. È vero che ad ogni iterazione le variabili dichiarate nel corpo del ciclo vengono create come nuove, ma è anche vero che le variabili dell'iterazione precedente vengono deallocate (cioè la memoria che occupano viene liberata e quindi è disponibile per le nuove variabili della prossima iterazione). Quindi non si somma l'occupazione di memoria delle diverse iterazioni. Inoltre è corretto che ad ogni iterazione del ciclo esterno j, mis ed m vengano re-inizializzate perché, ad ogni iterazione del ciclo esterno, i avanza di 1 ed il ciclo interno inizia a cercare un nuovo M1MIS a partire da i.

SECONDA:

Citando la post “(se non esiste M1MIS di P in T allora il programma scrive su cout "nessun M1MIS")" cosa che questo programma non fa, in quanto se non trova alcun M1MIS ovvero un qualunque match di P in T il programma stampa "-1 dimP".
Esempio di input: "5 2 3 3 3 3 3 1 1" output del programma così com'è "-1 2".

Non è qui l’errore; il problema indicato è dovuto al mio tentativo di semplificare la soluzione durante la spiegazione di oggi.
Si risolve sostituendo "`cout<<bestIn<<' '<<bestMIS<<endl;`" con:

```
if(bestIn!=-1)
cout<<bestIn<<" "<<bestMIS;
else cout<<"nessun M1MIS";
```

TERZA:

Dal testo dell'esercizio, si capisce (o almeno io ho capito così) di trovare il primo match completo di P in T, in poche parole la lunghezza dei match che devo cercare è di dimP elementi se il match è continuo o al massimo può essere di dimP+1 elementi dove il +1 e un miss. Il programma se trova subito un match di P in T contiguo continua a cercare altri match anche se ha già trovato il primo match che ha la lunghezza minore in assoluto ovvero dimP elementi e mis=0. Quindi il programma dovrebbe fermarsi appena trova il primo match contiguo di P in T.
Esempio di input: "14 3 1 2 3 4 1 2 4 1 2 3 4 1 2 4 1 2 4" in questo caso il programma trova un mismatch in posizione 0, un match contiguo in posizione 4 ed un altro mismatch in posizione 7, e ancora un atro match in posizione 11 che è contiguo. In questo caso una volta che il programma ha trovato il match contiguo in posizione 4 non ha più senso che cerchi altri match in quanto è il match che ha indice minore tra tutti i match di P in T che hanno bestMIS=0 ovvero il più basso valore di mis che si può trovare e se continua a cercare match può trovarne solo match con bestMIS=0 con indice sicuramente maggiore di 4 o può trovare match che hanno bestMIS=1.

Non è un errore, ma una possibile ottimizzazione che si ottiene inserendo la condizione “&& bestMIS>0” nella condizione di permanenza del ciclo while esterno. Oltre a ciò, nel ragionamento mi pare che ci sia qualcosa che non va: in un M1MIS ci può essere più di 1 mismatch. Ci possono essere fino a dimP-1 mismatch.

QUARTA:

Il problema è che se il primo valore considerato di T (diciamo T[i]) non matcha con P[0] il programma esegue un confronto fra P[0] ed il valore successivo T[i+1], questo è un errore perchè si potrebbe avere un match di P in T dove il primo confronto è un miss, il programma quindi restituirebbe come posizione di inizio del match il valore i quando invece dovrebbe essere i+1.

Si tratta proprio di quello che dici, ma pensi che il programma possa restituire risultati sbagliati a causa di questo problema?

No, il programma restituirebbe comunque il match che parte da i+1 perchè contenente un numero di mis minore rispetto a quello che parte da i.

Alla luce di quanto emerso dalla discussione, segue una riproposta del codice con le seguenti aggiunte e/o ottimizzazioni:
- output corretto (SECONDA obiezione).
- ottimizzazione while esterno (TERZA obiezione).
- un condizionale (da attivare) dopo il while interno che mostra il vero problema (QUARTA obiezione).

```
#include<iostream>
using namespace std;

int main()
{   //PRE
  int T[100], P[20], dimT, dimP;
  cin >> dimT >> dimP;

  int i=0;
  while(i<dimT)          //riempio T[0..dimT-1]
  {
    cin >> T[i];
    i++;
  }

  int j=0;
  while(j<dimP)          //riempio P[0..dimP-1]
  {
    cin >> P[j];
    j++;
  }


  int bestIn=-1, bestMIS=dimP;                    //"bestIn=-1" posiz inesistente; "bestMIS=dimP" basta che sia abbastanza grande (>dimP-1)
  i=0;                                            //reset indice i per scorrere T
  while(i<dimT-dimP+1 && bestMIS>0)       //R      //in alternativa "(i<=dimT-dimP && bestMIS>0)"; "bestMIS>0" ferma al primo pattern senza mis (di meglio non c'è)
  {
    int mis=0, m=i;                               //variabili ausiliarie
    bool fine=false;                              //ferma il ciclo non appena ho due mis consecutivi
    j=0;                                          //reset indice j per scorrere P
    while(j<dimP && !fine && m<dimT)      //R1
    {
      if(P[j]==T[m])
      {
        j++;
        m++;
      }
      else
        if(m+1<dimT && P[j]==T[m+1])
        {
          j++;
          m=m+2;
          mis++;
        }
        else
          fine=true;
    }

  /*
    // il seguente condizionale (se abilitato) mostra come il programma considera "M1MIS" anche quelli con un mis prima del pattern (evidenziati da: "<- THIS").
    if(j==dimP)
```

```
    {
      if(T[i]!=P[0])
        cout<<"start at: "<<i<<" con "<<mis<<" miss."<<" <- THIS"<<endl;
      else
        cout<<"start at: "<<i<<" con "<<mis<<" miss."<<endl;
    }
  */

    if(j==dimP && mis<bestMIS)
    {
      bestIn=i;
      bestMIS=mis;
  }
    i++;
  }

  if(bestIn==-1)
    cout<<"nessun M1MIS"<<endl;
  else
    cout<<bestIn<<" "<<bestMIS<<endl;

}  //POST



/*
PRE=(cin contiene dimT e dimP (0<dimT<=100 e 0<dimP<=20), seguiti da dimT+dimP valori interi qualsiasi).

R=(ho considerato i punti di inizio T[0..i-1]) && (0<=i<=dimT):
- (bestIn=-1 => non esistono M1MIS che iniziano in [0..i-1])
- (bestIn!=-1 => bestIn e bestMIS sono l'inizio e il MIS del match migliore che inizia in T[0..i-1]).
NB: da rifare/integrare alla luce dell'aggiunta di "&& bestMIS>0"

R1=(!fine <=> a partire da i, in T[i..m-1] ho un M1MIS di P[0..j-1] con MIS=mis).

POST=(se c'è almeno un M1MIS di P in T, il programma stampa l'inizio del M1MIS a MIS minimo (e quello che inizia prima se ce ne fossero diversi minimi)
e il valore del suo MIS) && (se non esiste M1MIS di P in T allora il programma scrive su cout "nessun M1MIS").
*/
```

Il codice è disponibile anche qui:
http://pastebin.com/ynFYDGb9

Il seguente codice (che eredita le ottimizzazioni dal precedente), tenta di risolvere la QUARTA obiezione; tuttavia introduce un ulteriore piccolo “difetto”: il controllo “T[i]==P[0]” viene eseguito consecutivamente due volte.
Ciò non è un problema, viene testato all’ingresso nel while interno e dal primo condizionale del while interno, ciò non inficia in alcun modo la funzionalità del programma, ma è logicamente sbagliato ri-controllare un’espressione che ho appena controllato e non può essere cambiata nel mentre.

```
#include<iostream>
using namespace std;

int main()
{   //PRE
  int T[100], P[20], dimT, dimP;
  cin >> dimT >> dimP;

  int i=0;
  while(i<dimT)          //riempio T[0..dimT-1]
  {
    cin >> T[i];
    i++;
  }

  int j=0;
  while(j<dimP)          //riempio P[0..dimP-1]
  {
    cin >> P[j];
    j++;
  }


  int bestIn=-1, bestMIS=dimP;                    //"bestIn=-1" posiz inesistente; "bestMIS=dimP" basta che sia abbastanza grande (>dimP-1)
  i=0;                                            //reset indice i per scorrere T
  while(i<dimT-dimP+1 && bestMIS>0)        //R     //in alternativa "(i<=dimT-dimP && bestMIS>0)"; "bestMIS>0" ferma al primo pattern senza mis (di meglio non c'è)
  {
    int mis=0, m=i;                               //variabili ausiliarie
    bool fine=false;                              //ferma il ciclo non appena ho due mis consecutivi
    j=0;                                          //reset indice j per scorrere P
    while(j<dimP && !fine && m<dimT && T[i]==P[0])  //R1
    {
      if(P[j]==T[m])                              //NB: qui la prima volta (m=i e j=0) viene ripetuto il controllo T[i]==P[0]; si può evitare ma complica il codice
      {
        j++;
        m++;
      }
      else
        if(m+1<dimT && P[j]==T[m+1])
        {
          j++;
          m=m+2;
          mis++;
        }
        else
          fine=true;
    }

    if(j==dimP && mis<bestMIS)
    {
```

```
        bestIn=i;
        bestMIS=mis;
    }
      i++;
    }

    if(bestIn==-1)
      cout<<"nessun M1MIS"<<endl;
    else
      cout<<bestIn<<" "<<bestMIS<<endl;

}   //POST



/*
PRE=(cin contiene dimT e dimP (0<dimT<=100 e 0<dimP<=20), seguiti da dimT+dimP valori interi qualsiasi).

R=(ho considerato i punti di inizio T[0..i-1]) && (0<=i<=dimT):
- (bestIn=-1 => non esistono M1MIS che iniziano in [0..i-1])
- (bestIn!=-1 => bestIn e bestMIS sono l'inizio e il MIS del match migliore che inizia in T[0..i-1]).
NB: da rifare/integrare alla luce dell'aggiunta di "&& bestMIS>0"

R1=(!fine <=> a partire da i, in T[i..m-1] ho un M1MIS di P[0..j-1] con MIS=mis).
NB: da rifare/integrare alla luce dell'aggiunta di "&& T[i]==P[0]"

POST=(se c'è almeno un M1MIS di P in T, il programma stampa l'inizio del M1MIS a MIS minimo (e quello che inizia prima se ce ne fossero diversi minimi)
e il valore del suo MIS) && (se non esiste M1MIS di P in T allora il programma scrive su cout "nessun M1MIS").
*/
```

Il codice è disponibile anche qui:
http://pastebin.com/VvfM68fv

Possibile soluzione al piccolo “difetto del doppio controllo”:
In prima battuta si potrebbe semplicemente inizializzare j a 1 e m a i+1 ad ogni iterazione del while esterno, valido dato che ho già controllato il caso j=0 e m=i ed eviterebbe il doppio controllo di cui sopra, ma aggiungerebbe un caso degenere: dimP=1; in tale situazione infatti non si entrerebbe mai nel while interno con il risultato di falsare il condizionale successivo con effetti negativi, ad esempio: input “3 1 1 5 1 5” darà come output “0 0” che è errato (dovrebbe restituire “1 0”).
Pertanto o si “accetta” che avvenga un “doppio controllo” oppure prima del while interno occorre gestire anche il caso degenere dimP=1 in modo che in tale situazione sia preservato l’ingresso nel ciclo ad esempio sostituendo:

```
int mis=0, m=i;                 //variabili ausiliarie
j=0;                            //reset indice j per scorrere P
bool fine=false;                //ferma il ciclo non appena ho due mis consecutivi
```

con:

```
int mis=0, m;                   //variabili ausiliarie
if(dimP==1)
{                               //j=0 e m=i il controllo T[i]==P[0] avverrà 2 volte ma è necessario nel caso degenere in cui dimP=1
  m=i;
  j=0;                          //reset indice j per scorrere P
}
else
{                               //se dimP!=1 allora j=1 e m=i+1 così evita il doppio controllo T[i]==P[0]
  m=i+1;
  j=1;                          //reset indice j per scorrere P
}
bool fine=false;                //ferma il ciclo non appena ho due mis consecutivi
```

così facendo solo nel caso dimP=1 il ciclo while interno eseguirà il “doppio controllo”.
(PS: Si potrebbe evitare anche quest’ulteriore doppio controllo che avviene solo se dimP=1, ma per farlo occorrerebbe complicare il condizionale appena qui sopra imponendo dei drastici cambi a bestIn e bestMIS, de-facto gestendo i casi 0 e 1 fuori dal ciclo interno).

Con quanto detto fin qui si potrebbe adattare nuovamente il solito codice anche con questa novità…
Tuttavia segue un codice che differisce da quello ufficiale di Filè, è molto simile in alcune parti ma ha delle differenze in altre, la riporto come soluzione alternativa, ed implementa anche la possibile soluzione al “doppio controllo” (in versione semplice, cioè tollerando il doppio controllo qualora dimP=1); in particolare non usa allo stesso modo l’indice “m” e non fa uso di così tante variabili ausiliarie.

```
#include<iostream>
using namespace std;

int main()
{   //PRE
  int T[100], P[20], dimT, dimP;
  cin >> dimT >> dimP;

  int t=0;
  while(t<dimT)          //riempio T[0..dimT-1]
  {
    cin >> T[t];
    t++;
  }

  int p=0;
  while(p<dimP)          //riempio P[0..dimP-1]
  {
    cin >> P[p];
    p++;
  }

  t=0;                                  //reset indice t per scorrere T
  int MIS=dimT, posiz=-1;               //MIS=dimT perchè sia un valore molto alto, va bene qualsiasi>dimP-1; posiz=-1 perchè tale posizione in T non esiste
```

```
  bool stopP;
  while(t<=dimT-dimP && MIS>0)   //R          //"!(dimT-t<dimP)" inutile entrare se dimT<dimP e/o se non avanzano abbastanza elementi in T per poter contenere P
                                              //da "t<dimT && !(dimT-t<dimP)" con DeMorgan si ha: "t<=dimT-dimP"
  {                                           //se c'è match perfetto (MIS=0) di meglio non c'è quindi mi posso fermare
    if(dimP==1)                               //reset indice p per scorrere P
      p=0;                                    //p=0 il controllo T[t]==P[0] avverrà 2 volte ma è necessario nel caso degenere in cui dimP=1
    else
      p=1;                                    //nel caso in cui dimP!=1 -> p=1 così evita il doppio controllo T[t]==P[0]
    int m=t;                                  //indice ausiliario (essendo che "salta" i mismatch isolati funge anche da contatore (modulo t))
    stopP=false;                              //stopP blocca se per 2 elem consecutivi di T NON c'è match con l'elem di P in esame
    while(p<dimP && !stopP && m+p<dimT && T[t]==P[0])   //R1                  //m+p<dimT bound check per non sforare nella lettura degli elem di T
    {
      if(T[m+p]==P[p])                        //gli elem matchano? bene passiamo ai prossimi
        p++;
      else
      {
        m++;                                  //controlliamo se dopo il mismatch dell'elem di P in esame con un elem di T ne segue subito un altro
        if(m+p<dimT && T[m+p]==P[p])           //m+p<dimT bound check, va fatto come primo controllo, altrimenti potremmo leggere elementi fuori da T
          p++;                                //questo è quello buono, posso quindi andare avanti
        else                                  //ho trovato in T due mismatch consecutivi per l'elem di P in esame -> non c'è M1MIS -> fermo il ciclo interno
          stopP=true;
      }
    }

    if(p==dimP && m-t<MIS)                    //se mi fermo perchè p==dimP ho matchato tutto P ed ho m-t MIS -> aggiorno la posiz d'inizio del match se ho un MIS minore
    {
      posiz=t;
      MIS=m-t;
    }
    t++;
  }

  if(posiz==-1)
    cout<<"Match (M1MIS) NON trovato."<<endl;
  else
    cout<<"Match (M1MIS) trovato."<<endl<<"inizio: "<<posiz<<" con "<<MIS<<" miss."<<endl;
}  //POST



/*
PRE=(cin contiene dimT e dimP (0<dimT<=100 e 0<dimP<=20), seguiti da dimT+dimP valori interi qualsiasi).

R=().

R1=().

POST=(se c'è almeno un M1MIS di P in T, il programma stampa l'inizio del M1MIS a MIS minimo (e quello che inizia prima se ce ne fossero diversi minimi)
e il valore del suo MIS) && (se non esiste M1MIS di P in T allora il programma scrive su cout "nessun M1MIS").
*/
```

Il codice è disponibile anche qui:
http://pastebin.com/1fYDesTT