correzione del compitino

1 e 2 turno

PRE=(T ha i primi n elementi definiti, n<lim2*lim3, P ha i primi dimP elementi definiti, n_m>0)

```
void trova_colonna(int* T,int n,int lim1,int lim2,int lim3,int* P, int dimP, int n_occ, int & indice_c)
{
   int nru=n/lim3, neur=n%lim3, ntc=lim3;
   if(n<lim3)
     ntc=n;
   bool trovato=false;
   for(int c=0; c<ntc && !trovato; c++)//R
    {
     int lung=nru;
     if(c<neur)
       lung++;
     trovato=checkC(T+c, lung, lim3, P, dimP, n_occ);// attenzione PRE di questa !!
     if(trovato)
      indice_c=c;
    }
```

} POST=(se vediamo T come un array a 2 dimensioni X[lim2][lim3], allora indice_c ha per R-valore l'indice minimo di una colonna di X che contiene **esattamente** n_m match di P[0..dimP-1], anche sovrapposti tra loro; se nessuna colonna soddisfa questa condizione allora indice_c=-1)

R=(esaminate le colonne 0..c-1, da 0 a c-2 nessuna soddisfa la condizione, trovato sse la colonna c-1 soddisfa la condizione,
se trovato, allora indice_c=c-1,
altrimenti indice_c=-1)

PRE=(inizio visto come inizio colonna di array [lim2][lim3], la lunghezza della colonna è lungc, P ha dimP valori, dimP>0, n_occ>0)

```
bool checkC(int* inizio, int lungc, int lim3,int *P,int dimP, int n_occ)
{
  int conta=0;
  for(int i=0; i<lungc-dimP+1  && conta<= n_occ; i++)//R
   {
     if(match(inizio+(i*lim3), lim3, P, dimP))
       {conta++; cout<<"match inizio="<<i<<endl;}
   }
if (conta==n_occ)
      return true;
else
     return false;
}
```

POST=(restituisce true sse la colonna contiene esattamente n_occ match di P)

R=(considerati gli elementi 0..i-1 della colonna come punti di inizio di match con P, conta= n. match di successo)

PRE=(inizio elemento di una colonna di un array [lim2][lim3], P ha dimP valori, dimP>0)

```
bool match(int* inizio, int lim3, int*P, int dimP)
{
  bool ok=true;
  for(int i=0; i<dimP && ok; i++)
   if(inizio[i*lim3]!=P[i])
    ok=false;
  return ok;
}
```

POST=(restituisce true sse inizio[0]=P[0], inizio[lim3]=P[1]...inizio[(dimP-1)*lim3]=P[dimP-1])

errori + frequenti:

1) non usare la PRE che implica che ci sia uno strato solo al massimo tutto definito, ma uno strato solo,
2) accedere valori non definiti :
   --non si calcola la vera lunghezza delle colonne
   --si esce dalle colonne durante il match

3)fare male il match :

usare 2 cicli
```
for(i=0;...;i++)
    for(int j=0; j<dimP;j++)
        if(T[inizio+i]==P[j]).....
```

usare lo stesso ciclo che scorre la colonna:

```
for(i=0;  ...; i=i+lim3)
    if(T[inizio+i]==P[j])
        {   j++;
          if(j==dimP)// match fatto
         }
    else  j=0;   //a volte non c'è
```

così i avanza sempre e in caso di fallimento di un match, i dovrebbe tornare indietro: P=absabx e T=absabsabx il primo tentativo di match porta i fino a 5 e se ripartissi da lì a cercare il prossimo match, non troverei niente. Invece ripartendo da i=3 trovo 1 match in T! E 3 < 5 quindi i deve tornare indietro !

senza else j continua a crescere e quindi considererei match non contigui!

P=aa e T=aba, il primo match va, poi no, ma j=1 e quindi trovo il secondo a alla terza posizione e j=2=dimP ==== Match completo!!! Ma non contiguo.

4) pensare che le colonne siano allocate in memoria come le righe e quindi n/lim2 =n. di colonne piene e n%lim2 = elementi dell'ultima colonna.

5) le condizioni di un'iterazione sono in ordine sbagliato, per esempio A[i]==2 && i<n è in ordine sbagliato perché accede A[i] anche se i>=n.

Invece l'ordine i<n && A[i]==2 sarebbe ok grazie alla valutazione short-cut del C++ (vedi sezione 2.4 del testo).

6) usare break

7) non usare funzioni

Il Turno

PRE=(T ha i primi n elementi definiti, n<=200, lim1*lim2*lim3<=200, P ha i primi dimP elementi definiti, dimP<=20, n_m>0)

```
void trova_strato(int* T,int n,int lim1,int lim2,int lim3,int* P, int dimP, int n_occ, int
& indice_strato)
{
  int nrp=n/lim3, neur=n%lim3,r=0;
  bool  trovato=false;
  while(r<nrp && !trovato)//R
  {
   trovato=checkR(T+r*lim3, lim3, P ,dimP, n_occ);
   if(!trovato) r++;
  }
  if(trovato)
   {indice_strato=r/lim2;}
  else
   if(neur)
     trovato=checkR(T+nrp*lim3,neur,P,dimP,n_occ);
   if(trovato)     indice_strato=nrp/lim2;
}
```

POST=(se vediamo T come un array a 3 dimensioni X[lim1][lim2][lim3], allora indice_strato ha per R-valore l'indice minimo di uno strato di X che contiene una riga che contiene esattamente n_m match di P[0..dimP-1], **NON sovrapposti tra loro**; se nessuno strato soddisfa questa condizione allora indice_strato=-1)

R=(trovato =>la riga T[r*lim3..(r+1)*lim3-1] ha esattamente n_m match non sovrapposti, e nessuna riga j<r ha la stessa proprietà
!trovato=> nessuna riga 0...r-1 ha esattamente n_m match non sovrapposti)

PRE=(T ha dimr elementi definiti, P ne ha dimP)

```
bool checkR(int*T, int dimr, int* P, int dimP, int n_occ)
{
  int inizio=0, conta=0;
  while(inizio<dimr-dimP+1)  //R=(conta ok per 0..inizio-1)
   {
     if(match(T+inizio,P,dimP))
      {conta++;inizio=inizio+dimP;} //attenzione !!
     else
       inizio++;
   }
 if(conta==n_occ)
    return true;
  else
    return false;
}
```

POST=(restituisce true sse in T[0..dimr] ci sono esattamente n_occ match non sovrapposti di P[0..dimP-1])

PRE=(T e P hanno dimP elementi definiti)

```
bool match(int*T, int*P, int dimP)
{
  bool ok=true;
  for(int i=0; i<dimP && ok; i++)  // R=(fino a i-2 bene, ok sse  i-1)
   if(T[i]!=P[i])
    ok=false;
  return ok;
}
```

POST=(restituisce true sse T[0..dimP-1]=P[0..dimP-1])