Scritto di Programmazione del 30 marzo 2009

E' indispensabile consegnare un testo leggibile. Evitare di spezzare una funzione su più pagine. In caso serva si può usare il foglio di protocollo aperto a giornale. Elaborati non leggibili non verranno corretti.

Chi copia perde il diritto di accedere all'appello di luglio.

PROBLEMA:

Dato un array di interi P[col*rig] che va interpretato come un array P[rig][col], un cammino in P è una sequenza C=[r0,r1,…,r(col-1)] di col valori in [0,rig-1], tale che, per ogni j in [0,col-2], tra rj ed r(j+1) una delle seguenti 3 possibilità vale:
(i) r(j+1)=rj-1,
(ii) r(j+1)=rj,
(iii) r(j+1)=rj+1.
Quindi il cammino da rj deve avanzare alla prossima colonna o passando alla riga precedente (caso (i)), o restando sulla stessa riga (caso (ii)), oppure spostandosi sulla riga successiva (caso (iii)). Ovviamente se rj=0 allora il caso (i) non si applica,mentre se rj=rig-1 allora il caso (iii) non si applica.
Un tale cammino C rappresenta un cammino che attraversa P e che è costituito dai seguenti elementi di P[rig][col]: (r0,0),(r1,1),…,(r(col-1),col-1).
Il **valore** di un cammino C=[r0,r1,…,r(col-1)] consiste nella somma P[r0][0]+P[r1][1]+…+P[r(col-1)][col-1]. Il problema da risolvere è trovare un cammino di massimo valore in P.

***Esempio 1**: sia P[4*4] che interpretata come il int[4][4] è:*
1021
2003
0112
1211
Un cammino attraverso P è: C=[1,1,0,1] e il valore di C è 2+0+2+3=7.

ATTENZIONE: P è un array di interi con [rig*col] elementi, ma va usato come int[rig][col]

La strategia per calcolare il cammino con valore massimo è la seguente:
a) se col=1 allora il cammino a valore massimo è semplicemente il valore massimo dell'unica colonna.
b) se col=2, allora dobbiamo considerare ogni elemento della colonna 0 e calcolare quale tra le 3 scelte possibili per il prossimo passo conviene fare per ottenere il valore massimo. Per esempio se consideriamo P[1][0] potremmo continuare con P[0][1], oppure con P[1][1] oppure con P[2][1].
In questo modo calcoliamo il cammino massimo che parte da P[1][0].
c) Se col= 3, allora dopo avere calcolato le colonne 2 e 1 come descritto in (a) e (b) possiamo calcolare la colonna 0 nello stesso modo.
d) Il procedimento appena descritto si può ripetere per qualsiasi numero di righe.

La strategia appena descritta consente di calcolare per ogni elemento (i,j) di P un valore che è quello del cammino di valore massimo che parte da P[i][j] e termina su un elemento dell'ultima colonna di P. Anziché modificare P, questi nuovi valori andranno inseriti in un array M che ha la stessa dimensione di P, cioè [rig*col] elementi interi e che va interpretato come int [rig][col] esattamente come P. Cosa va inserito in M è spiegato precisamente nel seguente Esempio.

***Esempio 2**: riprendiamo l'Esempio 1. Con il procedimento appena descritto calcoliamo la seguente matrice M[4*4]che interpretiamo come [4][4] come segue:*
6551
7533
6542
7631
La colonna 3 di M è uguale alla colonna 3 di P (vedi (a)), la colonna 2 di M è calcolata seguendo il metodo spiegato in (b), e lo stesso vale per le colonne 1 e 0.
Quindi il valore massimo dei cammini di P è 7 e ci sono 2 punti di partenza per ottenere un cammino di valore 7: (1,0) e (3,0).

Parte iterativa: scrivere una funzione iterativa F che abbia il seguente prototipo :
void F(int *P, int *M int rig, int col) e che esegua il seguente calcolo:
riceve l'array ad una dimensione P e lo interpreta come un array int [rig][col] (completamente definito) e riempie M (anch'esso interpretato come int [rig][col]) nel modo spiegato nell'Esempio 2.

ATTENZIONE: M, esattamente come P, è un array di interi con [rig*col] elementi, ma va usato come int[rig][col]. In F può convenire usare funzioni iterative ausiliarie.

Parte ricorsiva: scrivere una funzione R con prototipo void R(int*M, int *C, int rig, int col) che riceve in M l'array prodotto da F ed inserisce in C (che ha col elementi) i valori che definiscono un cammino di valore massimo che attraversa P, come descritto nel seguente esempio:

***Esempio 3**: da M dell'Esempio 2, un cammino C di valore massimo che attraversa P è: C=[1,0,0,1]. Ovviamente questo non è l'unico cammino di valore massimo (nell'Esempio 1 ne mostriamo un altro), ma R deve inserire in C gli indici di uno qualsiasi dei cammini a valore massimo.*

ATTENZIONE: R potrebbe essere non ricorsiva, ma in ogni caso dovrà espletare il suo compito invocando essenzialmente funzioni ricorsive. E' possibile usare anche qualche funzione iterativa (possibilmente usate per la parte iterativa), ma solo a condizione che esse diano un contributo piccolo al calcolo richiesto a R.

Ogni funzione ricorsiva deve venire corredata da Pre e soprattutto da Post-condizione e dalla dimostrazione della sua correttezza basata sull'induzione.

Criterio di valutazione: il risultato dipenderà strettamente dalla chiarezza e semplicità dei programmi prodotti

Soluzione

```
// parte iterativa
// point serve a contenere una coppia di indici
struct point{int r,c;
  point(int a=0, int b=0)
  {r=a; c=b;}
};
```

```
//OK controlla  che un point sia un elemento di una  matrice rig X col

bool ok(point x, int rig, int col)
{
  return (x.r>=0 && x.r<rig) && (x.c >=0 && x.c<col);
}

// previous, dato un point x costruisce in M  i 3 points della colonna successiva  e riga sopra, uguale
// sotto, qualcuno di questi  point potrebbe essere fuori dalla matrice,  ma useremo OK per controllarlo
void previous(point x, point* M)
{
  M[0]=point(x.r-1,x.c+1);
  M[1]=point(x.r,x.c+1);
  M[2]=point(x.r+1,x.c+1);
}

void F(int* P, int*M, int rig, int col)
{

  for(int i=0;i<rig; i++)
    M[i*col+col-1]=P[i*col+col-1];
  for(int i=col-2; i>=0; i--)
    for(int j=0;j<rig; j++)
     {
          point K[3];
          previous(point(j,i),K);
          int mx=INT_MIN;
          point max(0,0);
          for(int k=0;k<3;k++)
            if(ok(K[k],rig,col))
              {
                if(M[K[k].r*col+K[k].c]>mx)
                      {
                        mx=M[K[k].r*col+K[k].c];
                        max=K[k];
                      }
              }

          M[j*col+i]=M[max.r*col+max.c]+P[j*col+i];
     }
}

// parte ricorsiva, R usa R1 e max che sono ricorsive

void R1(int*M, int*C, int c, int rig, int col)
{
  if(c<col-1)
    {
      int r=C[c];
      point k[3];
      previous(point(r,c),k);
      int max=INT_MIN, index=-1;
      for(int j=0;j<3;j++)
```

```
          if(ok(k[j],rig,col))
            if(M[k[j].r*col+k[j].c]>max)
             {
               max=M[k[j].r*col+k[j].c];
               index=j;
             }
    C[c+1]=k[index].r;
    R1(M,C,c+1,rig,col);
   }
}
int max(int* M, int r, int rig, int col)
{
 if(r==rig-1)
   return rig-1;
 int a=max(M,r+1,rig,col);
 if(M[a*col]>M[r*col])
   return a;
 else
   return r;
}

void R(int*M,int*C,int rig, int col)
{
 int m=max(M,0, rig,col);
 C[0]=m;
 R1(M,C,0,rig,col);

}
// prova le funzioni F ed R sull'esempio dell'esame
main()
{
 int P[]={1,0,2,1,2,0,0,3,0,1,1,2,1,2,1,1};
 int M[16];
 F(P,M,4,4);
 for(int i=0; i<4; i++)
  {
    for(int j=0; j<4; j++)
          cout<< M[i*4+j]<<' ';
    cout<<endl;
  }

 int C[4];
 R(M,C,4,4);
 for(int i=0; i<4; i++)
  cout<<C[i]<<' ';
 cout<<endl;

}
```