Scritto di programmazione 17 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 al II appello scritto.

Parte iterativa) si tratta di scrivere una funzione void F(int * T, int dim,int rig, int col, int k, int & strato, int & riga) che riceve in T un array contenente dim interi e deve trattarlo come un array int[][rig][col] e deve calcolare lo strato di T tale che contiene la riga che presenta il massimo numero di valori distinti che appaiono esattamente k volte nella riga stessa e deve restituire al chiamante l'indice dello strato trovato e anche l'indice della riga, all'interno dello strato, che ha la proprietà appena descritta. I due indici vanno restituiti attraverso i 2 parametri di F passati per riferimento *strato* e *riga*. Qualoro ci fossero più strati a pari merito, F deve restituire quello con indice minimo (e la sua riga).

Esempio*: sia T=[1,2,3,1, 1,1,1,1, 0,2,1,2, 1,0,1,3, 3,3,1,1, 1,2,3,1, 2,2,1,3, 3,3] con rig=2, col=4 e k=2. T va quindi visto come un array di 4 strati, di cui i primi 3 sono pieni (cioè con 2 righe di 4 elementi) mentre l'ultimo strato ha una riga piena e la seconda riga con solo 2 elementi. Per facilitare la comprensione gli strati sono separati da spazi.*

Lo strato 0*: nella prima riga solo 1 occorre esattamente due volte. Nella seconda riga nessun elemento occorre esattamente due volte.*

Lo strato 1*: nella prima riga solo 2 occorre esattamente due volte. Nella seconda riga 1 occorre esattamente due volte.*

Lo strato 2*: nella prima riga sia 3 che 1 occorrono esattamente due volte. Nella seconda riga 1 appare esattamente due volte.*

Lo strato 3*: nella prima riga 2 appare esattamente due volte. Nella seconda riga (parziale) 3 appare esattamente 2 volte.*

Quindi, in questo caso, F dovrebbe ritornare al chiamante con strato = 2 e riga = 0.

Attenzione: l'ultimo strato potrebbe essere incompleto.

Consiglio: conviene introdurre funzioni che si occupano di sottoproblemi.

Parte ricorsiva) Data una lista concatenata x, i cui nodi contengono caratteri, e dato un pattern P, vogliamo effettuare un pattern matching di P sulla lista x, staccando dalla lista originale i nodi che partecipano al match di P. Quindi la funzione ricorsiva richiesta deve restituire due liste: quella dei nodi che partecipano al match e quella dei nodi che non partecipano al match.

Esempio: *sia x= d->v->a->r->w->t->h->e->q->z e P=[vhqwz], allora la funzione deve restituire due liste:*

L1: d->a->r->w->t->e->z dei nodi che non partecipano al match e la lista L2: v->h->q dei nodi che invece partecipano al match.

Importante: il match può essere anche parziale (come nell'esempio). Osservare il fatto che le 2 liste da restituire sono composte dai nodi della lista originale. **Nessun nuovo nodo va allocato!**

La funzione ricorsiva deve avere il seguente prototipo: *doppio* PM(nodo*x, char*P, int dimP); in cui x, P, e dimP hanno il significato consueto ed il valore restituito è di un tipo strutturato, chiamato *doppio*, che deve contenere 2 liste concatenate (cioè due puntatori a nodo) che naturalmente dovranno essere le 2 liste richieste come risultato (L1 ed L2 nell'esempio). La definizione del tipo *doppio* va specificata e vanno anche specificati i costruttori del tipo *doppio* che avrete usato nel codice di PM.

Attenzione: abbiate cura di specificare chiaramente qual'è il campo di *doppio* destinato a contenere la lista del match e quindi qual'è il campo destinato a contenere l'altra lista.

Correttezza:

Specificare la post-condizione della funzione ricorsiva PM e dimostrare la sua correttezza secondo lo schema induttivo.

Soluzione parte iterativa:

```
bool presente(int x, int*A, int top)
{
  bool trovato=false;
  for(int i=0; i<top && ! trovato; i++)
    if(A[i]==x)
      trovato=true;
  return trovato;
}

int nocc(int x, int*T, int dim)
{
  int c=0;
  for(int i=0; i<dim; i++)
    if(T[i]==x)
      c++;
  return c;
}

int F_riga(int*T,int dim,int k)
{
  int A[]=new int[dim];
  int top=0, v=0;
  for(int i=0; i< dim; i++)
    {
      if(!presente(T[i],A,top))
            {
              int c=nocc(T[i], T, dim);
              if(c==k)
                v++;
              A[top]=T[i];
              top++;
            }
    }
  delete [] A;
  return v;
}

int F_strato(int*T, int dim, int col, int k, int &riga)
{
  int nrp=dim/col, eur=dim%col, max=0;
  riga=0;
  for(int i=0; i<nrp; i++)
    {
      int m=F_riga(T+i*col,col,k);
      if(m>max)
            {
              max=m;
              riga=i;
            }
```

```
   }
  if(eur>0)
   {
    int m=F_riga(T+nrp*col,eur,k);
    if(m>max)
          {
           max=m;
           riga=nrp;
          }
   }
  return max;

}

void F(int *T, int dim, int rig, int col, int k, int & strato, int & riga)
{
  int nsp=dim/rig*col, eus=dim%rig*col, max=0, rigat;
  strato=0;
  riga=0;
  for(int i=0; i<nsp; i++)
   {
    int t=F_strato(T+i*(col*rig),rig*col, col, k, rigat);
    if(t > max)
          {
           max=t;
           strato=i;
           riga=rigat;
          }
   }
  if(eus>0)
   {
    int t=F_strato(T+nsp*(rig*col), eus, col, k, rigat);
    if(t > max)
          {
           max=t;
           strato=nsp;
           riga=rigat;
          }
   }
  }
```

Soluzione parte ricorsiva:

```
doppio PM(nodo*x, char*P, dimP)
{
 if(!x)
   return doppio(0,0);
 if(!dimp)
   return doppi0(0,x);
 if(x->info==*P);
 {
   doppio y=PM(x->next,P+1,dimP-1);
   x->next=y.primo;
   return doppio(x,y.secondo);
 }
 else
   {
     doppio y=PM(x->next,P, dimP);
     x->next=y.secondo;
     return doppio(y.primo,x);
   }
}
```