Informatica Generale II

Marco Roveri

Trento, 17-12-2004

Nel file `segmenti.dat` è memorizzata una sequenza di segmenti. Ogni segmento è rappresentato mediante una coppia di punti di un piano cartesiano. In particolare, ogni punto è descritto da una coppia di numeri interi positivi: pertanto, ogni segmento è rappresentato nel file da coppie di interi positivi, corrispondenti ad altrettanti punti. Pertanto ogni record del file rappresenta un segmento sul piano cartesiano. Si vogliono estrarre dal file, e memorizzare in una lista, rappresentata mediante record e puntatori, tutti e soli i segmenti che non sono paralleli né all'asse delle ascisse né all'asse delle ordinate.

1. **[Punti 3]** Definire le strutture dati (i nomi sono importanti per il corretto funzionamento delle funzioni/procedure giá fornite):

   - `TipoPunto`: la struttura dovrá avere almeno due campi "`x`" e "`y`" di tipo intero.
   - `TipoSegmento`: la struttura dovrá avere almeno due campi "`primo`" e "`secondo`" di tipo `TipoPunto`.
   - `TipoLista`: la struttura dovrá avere almeno due campi "`segmento`" di tipo `TipoSegmento`, e `next` di tipo puntatore a `TipoLista`. Per il corretto funzionamento delle funzioni/procedure giá fornite, si richiede inoltre che per questa struttura esista almeno un costruttore

     ```
     TipoLista(TipoSegmento s, TipoLista * n)
     ```

     che inizializza il campo `segmento` e il campo `next`.

2. **[Punti 6]** Scrivere una funzione `TogliParalleli` che data la lista dei segmenti letti restituisca la lista costituita da tutti i segmenti che non sono paralleli né all'asse delle ascisse né all'asse delle ordinate. Nota che sia la lista ricevuta in input che la lista costruita dalla funzione `TogliParalleli` devono poter essere deallocate separatamente, ovvero non devono esserci elementi di memoria comuni ad entrambe le liste.

   ```
   TipoLista * TogliParalleli(TipoLista * l) {
     // inserire qui corpo della funzione
   }
   ```

3. **[Punti 6]** Scrivere una funzione che, ricevendo come parametro di input la lista dei segmenti costruita al punto precedente, fornisca la misura e le coordinate dei punti estremi del segmento più lungo.

```
void LunghezzaMaggiore(TipoLista * l, TipoSegmento &s, double & ln) {
  // inserire qui corpo della funzione
}
```

Si ricorda che se gli estremi del segmento sono i punti di coordinate

$$(x_1, y_1) \qquad (x_2, y_2)$$

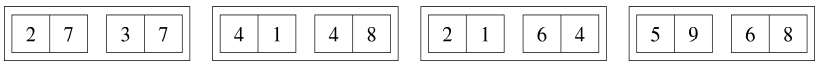
la lunghezza del segmento è pari a

$$\sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$$

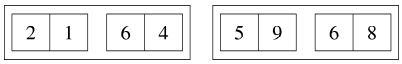
4. **[Punti 5]** Scrivere una funzione che, ricevendo come parametro di ingresso una lista di segmenti, deallochi la lista.

```
void CancellaLista(TipoLista * &l) {
  // inserire qui corpo della funzione
}
```

Ad esempio, se nel file `segmenti.dat` sono rappresentati i seguenti quattro segmenti:



poiché il primo ed il secondo record rappresentano segmenti paralleli, rispettivamente, all'asse delle ascisse e all'asse delle ordinate, nella lista risultante dall'algoritmo di cui al punto (2) saranno inseriti soltanto i seguenti record:



Il segmento più lungo tra quelli presenti nella lista ha coordinate (2,1),(6,4) e lunghezza 5.
Allo studente è fornito un file `main.cpp` nella directory di lavoro in cui troverà le definizioni delle funzioni di cui sopra e che dovrà implementare. Per completezza i file `main.cpp` `segmenti.dat` sono riportati qui di seguito.

N.B.:

1. Lo studente non dovrà per nessun motivo modificare le altre parti del programma tranne il corpo delle funzioni da realizzare e scommentare le parti di programma relative alla domanda risposta nel `main`.

2. Lo studente **deve** inserire il proprio nome, cognome, numero di matricola e numero della postazione utilizzata.

main.cpp

```
/*
  NOME:
  CONGNOME:
  MATRICOLA:
  DOCUMENTO:
  CODICE CALCOLATORE:
*/

#include <iostream>
#include <cmath>

using namespace std;

// ************************************************
// Prima domanda
// ************************************************
struct StructPunto {
  // inserire qui corpo della struttura

} TipoPunto;

struct TipoSegmento {
  // inserire qui corpo della struttura

};

struct TipoLista {
  // inserire qui corpo della struttura

};

// ************************************************

TipoLista * ReadFile(void) {
  TipoLista * result = NULL;
  TipoSegmento s;
  char c;
  c = cin.get();
  while ( (c != EOF) && !cin.eof() ) {
    cin.unget();
    cin >> s.primo.x;
    cin >> s.primo.y;
    cin >> s.secondo.x;
    cin >> s.secondo.y;
    cin >> c;
    result = new TipoLista(s, result);
  }
  return result;
```

```
}

void PrintList(TipoLista * l) {
  while (l != NULL) {
    cout << l->segmento.primo.x << " ";
    cout << l->segmento.primo.y << " ";
    cout << l->segmento.secondo.x << " ";
    cout << l->segmento.secondo.y << endl;
    l = l->next;
  }
}

void PrintMaggiore(TipoSegmento s, double l) {
  cout << "Segmento piu' lungo: ";
  cout << s.primo.x << " ";
  cout << s.primo.y << " ";
  cout << s.secondo.x << " ";
  cout << s.secondo.y << endl;
  cout << "Lunghezza = " << l << endl;
}

// ************************************************
// seconda domanda
// ************************************************
TipoLista * TogliParalleli(TipoLista * l) {
  // inserire qui il corpo della procedura
}
// ************************************************


// ************************************************
// terza domanda
// ************************************************
void LunghezzaMaggiore(TipoLista * l, TipoSegmento &s, double & ln) {
  // inserire qui il corpo della procedura
}
// ************************************************

// ************************************************
// quarta domanda
// ************************************************
void CancellaLista(TipoLista * & l) {
  // inserire qui corpo della funzione

}


int main(int argc, char *argv[])
{
```

```
  TipoLista * lista = NULL;
  TipoLista * lista1 = NULL;
  TipoSegmento s;
  double lenght;

  lista = ReadFile();

  // Domanda 2:
  // scommentare le quattro righe di codice seguente
  // lista1 = TogliParalleli(lista);
  // cout << "Lista segmenti non paralleli:" << endl;
  // PrintList(lista1);

  // Domanda 4
  CancellaLista(lista);

  // Domanda 3:
  // Scommentare le tre righe di codice seguente
  // LunghezzaMaggiore(lista1, s, lenght);
  // PrintMaggiore(s, lenght);

  // Domanda 4
  //CancellaLista(lista1);
}
```

segment.dat

```
2 7 3 7
4 1 4 8
2 1 6 4
5 9 6 8
```

Implementazione richiesta al punto 1

Siccome non sono richiesti costruttori per le strutture richieste, a parte il caso della struttura `TipoLista`, una possibile risposta alla prima domanda é qui di seguito riportata.

```
// ************************************************
// Prima domanda
// ************************************************
struct StructPunto {
  // inserire qui corpo della struttura
  int x, y;
} TipoPunto;

struct TipoSegmento {
  // inserire qui corpo della struttura
  TipoPunto primo, secondo;
};

struct TipoLista {
  // inserire qui corpo della struttura
  TipoSegmento segmento;
  TipoLista * next;
  TipoLista(TipoSegmento s, TipoLista * n) {
    segmento = s;
    next = n;
  };
};
// ************************************************
```

Funzione richiesta al punto 2

Prima di tutto é da ricordare che un segmento é parallelo all'asse delle ascisse se i due punti estremi caratterizzanti il segmento hanno le stesse coordinate Y (il campo y dei due punti estremi é lo stesso). Dualmente, un segmento é parallelo all'asse delle ordinate se i due punti caratterizzanti il segmento hanno le stesse coordinate X.

Per risolvere questo punto occorre notare che é necessario scorrere la lista in input e per ogni elemento verificare che non sia parallelo ad uno dei due assi. Se é parallelo lo ignoriamo, altrimenti inseriamo il segmento in questione nella lista risultato. Siccome l'ordine non é importante, possiamo inserire il nuovo nodo in testa alla lista risultato (che quindi risulterá rovesciata rispetto alla lista di partenza). Siccome le due liste non devono avere memoria in comune, occorrerá creare un nuovo nodo con valore copia del valore del nodo di partenza per inserirlo nella lista risultato.

```
ALGORTHM elimina segmenti paralleli
  Crea lista risultato vuota
  WHILE (la lista in input non e' vuota) DO
    IF (l'elemento corrente non e' parallelo ad un asse) THEN
       Inserisci l'elemento corrente nella lista risultato
```

```
      ENDIF
      Avanza all'elemento successivo della lista di input
    END
    RETURN nuova lista costruita
  ENDALGORITHM
```

Implementazione in C++.

```
TipoLista * TogliParalleli(TipoLista * l) {
// La lista risultato, inizialmente vuota
  TipoLista * result = NULL;

  // Finche' la lista in input non e' vuota
  while(l != NULL) {
    // Se il segmento non e' parallelo ad un asse cartesiano
    if (l->segm.primo.x != l->segm.secondo.x &&
        l->segm.primo.y != l->segm.secondo.y) {
      // Creiamo un nuovo nodo in cui il campo info e' lo stesso
      // del nodo considerato.
      // Notare che:
      // 1) stiamo effettuando una copia.
      // 2) per efficienza effettuiam una inserzione in testa.
      result = new TipoLista(l->segm, result);
    }
    // avanziamo al prossimo segmento della lista
    l = l->next;
  }
  // ritorniamo la lista risultato
  return result;
}
```

Funzione richiesta al punto 3

Per rispodere alla domanda del punto 2, prima di tutto risulta necessario definire una funzione che dati due punti calcoli la distanza tra questi due punti. Questa funzione calcola quindi la lunghezza del segmento se applicata ai due punti caratterizzanti di un segmento.

La parte principale della domanda consiste nel determinare il segmento all'interno della lista che ha la lunghezza maggiore. Questo lo si ottiene scorrendo la lista. Supponiamo che il primo elemento della lista sia l'elemento massimo cercato (assumiamo quindi che la lista contenga almeno un elemento). Cominciamo quindi a scorrere la lista dal secondo elemento confrontando la lunghezza del segmento corrente con il segmento candidato ad essere il massimo elemento. Se la lunghezza del segmento corrente e' maggiore della lunghezza del segmento candidato allora, il segmento corrente diventa il nuovo segmento candidato e procediamo con l'elemento successivo. Altrimenti procediamo con l'elemento successivo.

```
ALGORITHM Calcola elemento massima lunghezza
  IF (lista segmenti vuota) ERROR
  Calcolo lunghezza primo elemento
```

```
   Elezione primo elemento ad essere il candidato
   Avanziamo al secondo elemento della lista
   WHILE (la lista non e' vuota) DO
     Calcola linghezza segmento corrente
     IF (lughezza segmento corrente maggiore della
          lunghezza del candidato) THEN
        segmento corrente eletto a nuovo candidato
     ENDIF
     Avanziamo a segmento successivo
   ENDWHILE
 ENDALGORITHM
```

Implementazione in C++.

```
double Lunghezza(TipoSegmento s) {
  double result = 0.0;
  int diffx = s.primo.x - s.secondo.x;
  int diffy = s.primo.y - s.secondo.y;
  result = sqrt(diffx*diffx + diffy*diffy);
  return result;
}

void LunghezzaMaggiore(TipoLista * l, TipoSegmento &s, double & ln) {
  if (l == NULL) {
    cerr << "ERRORE: lista vuota" << endl;
    exit(1);
  }
  else {
   // Estrazione primo segmento e candidatura ad essere il nodo cercato
    s = l->segm;
    // Calcolo lunghezza primo segmento
    ln = Lunghezza(s);
    // Avanzamento al segmento successivo
    l = l->next;
    // Finche' ci sono ancora segmenti
    while (l != NULL) {
      // Calcolo lunghezza segmento corrente
      double lnt = Lunghezza(l->segm);
      // Se la lunghezza e' maggiore della lunghezza del segmento candidato
      if (lnt > ln) {
        // il segmento corrente diventa il nuovo candidato
        ln = lnt;
        s = l->segm;
      }
      // Procediamo con l'elemento successivo
      l = l->next;
    }
  }
}
```

Funzione richiesta al punto 4

Per rispondere a questa domanda, basta scorrere la lista fintanto che non é vuota. Ad ogni iterazione memorizzo il segmento corrente. Avanzo all'elemento successivo e cancello l'elemento corrente salvato.

```
ALGORITHM Cancella lista
  WHILE (la lista non e' vuota) DO
    Memorizza nodo della lista corrente
    Avanziamo a segmento successivo
    Deallochiamo nodo corrente memorizzato
  ENDWHILE
ENDALGORITHM
```

Implementazione in C++ L'implementazione é riportata qui di seguito.

```
void CancellaLista(TipoLista * & l) {
  // inserire qui corpo della funzione
  while(l != NULL) {
    TipoLista * l1 = l;
    l = l->next;
    delete l1;
  }
}
```

Notare che prima di deallocare il nodo corrente `l1` salvato, vanziamo a considerare il nodo successivo. Se avessimo deallocato prima di avanzare, non avremmo potuto farlo, in quanto l'area di memoria del nodo corrente non avrebbe fatto piú parte della memoria del programma.