Relazione di progetto

Algoritmi e Strutture Dati

Sessione estiva 2013/2014

Julian Sparber
matr. no.: 260324

1 Specifica del problema

Si supponga di elaborare i dati relativi ad un grafo. Le informazioni associate al problema siano: un insieme di vertici (con nomi specifcati da stringhe prive di spazi) e un insieme di archi caratterizzati da una tripla di distanze d1, d2, d3 (una tripla di numeri reali).

1.1

Acquisisce da file le informazioni relative al grafo. Il formato del file è del tipo:

<No. totale dei vertici>				
<No. di vertici collegati al vertice A>				
<vertice_A>	<vertice_B>	<d1>	<d2>	<d3>
<vertice_A>	<vertice_M>	<d1>	<d2>	<d3>
...				
<vertice_A>	<vertice_Z>	<d1>	<d2>	<d3>
<No. di vertici collegati al vertice B>				
<vertice_B>	<vertice_C>	<d1>	<d2>	<d3>
<vertice_B>	<vertice_X>	<d1>	<d2>	<d3>
...				

Ad esempio:

```
5
3
v_a v_b 8.2 5.3 9.7
v_a v_c 2.5 1.4 3.2
v_a v_e 3.6 5.0 2.7
4
v_b v_a 1.4 5.2 0.1
v_b v_c 8.5 11.4 0.2
v_b v_d 6.0 2.2 2.7
v_b v_e 6.9 2.4 2.8
2
v_c v_d 2.7 6.2 1.1
v_c v_e 3.8 4.4 3.4
3
v_d v_a 18.2 7.3 19.7
v_d v_c 12.5 1.6 5.4
v_d v_e 11.6 3.2 12.7
1
v_e v_a 12.6 16.2 14.1
```

1.2

Inserisce i dati acquisiti in una opportuna struttura dati.

1.3

Dati un vertice sorgente, uno destinazione e una tipologia di distanze (d1 oppure d2 oppure d3) inseriti dall'utente, calcola il percorso più breve tra sorgente e destinazione, mostrando a monitor tale percorso e la relativa distanza.

1.4

Dato un vertice specificato dall'utente, calcola la media e la mediana della distanze minime che separano tale vertice da tutti gli altri vertici del grafo, in base alle tipologie di distanza d1, d2 e d3.

Per quanto riguarda l'analisi teorica si devono studiare le complessità degli algoritmi di acquisizione del file (punto 2), calcolo del percorso più breve tra due vertici (punto 3) e calcolo di media e mediana (punto 4).
Per quanto riguarda il punto 4 si deve anche verificare sperimentalmente la complessità delcalcolo di media e mediana, generando casualmente una sequenza di distanze (di N numeri reali) da fornire come input all'algoritmo per valori crescenti di N.

2 Analisi del problema

2.1 INPUT

Un vertice sorgente, un vertice destinazione e una tipologia di distanze (d1 oppure d2 oppure d3) inseriti dall'utente.
Un file sorgete che contine tutte le informazioni sui nodi.

2.2 OUTPUT

All'utente viene mostrato il percorso più breve tra i nodi sorgente e destinazione definiti dall'utente e anche la media e la mediana delle distanze minime dal nodo sorgente a tutti gli altri vertici del grafo.

2.3 PERCORSO

I valori in questo grafo sono distanze qundi non possono essere negativi.

2.4 MEDIA

La media è la somma di tutti i valori divisa per il numero di valori.

2.5 MEDIANA

La mediana è l'elemento centrale di una lista ordinata (o se la lista è ha un numero pari di elementi la media dei due elementi in mezzo).

3 Progettazione dell'algoritmo

Ogni carattere letto dal file viene validato immediatamente. Il nodo e l'arco preso da ognuna delle righe viene salvato in una struttura dinamica di questo tipo:

{
    nome del nodo,
    archi del nodo,
    minima distanza,
    parente del nodo,
    nodo successivo;
}

e gli archi vengono salvati in una struttura di questo tipo:

{
    nome del nodo di destinazione,
    nodo di destinazione,
    la tripla della distanza,
    arco successivo;
}

Visto che i valori in questo grafo sono distanze, non possono essere negativi. Per questa ragione l'algoritmo di Dijkstra è una buona scelta.

```
void inizializza(vertice_grafo_t *grafo_p,
          vertice_grafo_t *sorgente_p)
{
  vertice_grafo_t *vertice_p;
  for (vertice_p = grafo_p;
    (vertice_p != NULL);
    vertice_p = vertice_p->vertice_succ_p)
  {
    vertice_p->distanza_min = INFINITO;
    vertice_p->padre_p = NULL;
  }
  sorgente_p->distanza_min = 0.0;
}
```

```
void riduci(arco_grafo_t *arco_p, vertice_grafo_t *vertice_p)
/* vertice da cui l'arco esce */
{
  if (arco_p->vertice_adiac_p->distanza_min >
      vertice_p->distanza_min + arco_p->peso)
  {
    arco_p->vertice_adiac_p->distanza_min =
      vertice_p->distanza_min + arco_p->peso;
    arco_p->vertice_adiac_p->padre_p = vertice_p;
  }
}

void dijkstra(vertice grafo t *grafo p, vertice grafo t *sorgente p)
{
  vertice grafo t *vertice p;
  arco grafo t
  *arco p;
  inizializza(grafo p, sorgente p);
  <costruisci un insieme per i vertici
  già considerati (inizialmente vuoto)>
  <costruisci una struttura
    per i vertici da considerare (inizialmente tutti)>
  while (<la struttura non e vuota>)
  {
    <rimuovi dalla struttura
    il vertice vertice p con distanza min minima>
    <inserisci vertice p nell insieme dei vertici già considerati>
    for (arco p = vertice p->lista archi p;
    (arco p != NULL);
    arco p = arco p->arco succ p)
      if (<arco p->vertice adiac p
          non è nell'insieme dei vertici già considerati>)
        riduci(arco p, vertice p);
  }
}
```

La mediana viene calcolata con quickselect, un'algoritmo randomizzato che trova il k-esimo elemento di una struttura disordinata con n elementi eseguendo in $O(n^2)$ nel caso pessimo e O(n) nel caso ottimo.

Quickselect è relativamente semplice da implementare, e se implementato correttamente, leggermente più veloce di heapselect.

```
algoritmo Quickselect(array A, intero K) -> elemento_array

  seleziona un elemento_array x in A
  partiziona A rispetto ad x calcolando:

  A1 = { y appartenente ad A : y < x }
  A2 = { y appartenente ad A : y = x }
  A3 = { y appartenente ad A : y > x }

  se ( k < |A1| )
    allora ritorna Quickselect(A1,k)

  altrimenti se ( k > |A1| + |A2| )
    allora ritorna Quickselect(A3, k - |A1| - |A2|)

  altrimenti ritorna x
```

4 Implementazione dell'algoritmo

4.1 Strutture

Le strutture usate sono:

```
typedef struct tArc {
    char node[STRING_LENGTH];
    double distance[3];
    struct tArc *next;
    struct tNode *dest;
  } tArc;

typedef struct tNode {
    char node[STRING_LENGTH];
    tArc *arc;
    struct tNode *next;
    double minDistance;
    struct tNode *parent;
  } tNode;

typedef struct tList {
    tNode *node;
    struct tList *next;
  } tList;
```

I nomi dei vertici devono iniziare con "v" e possono avere al massimo 255 charateri (STRING_LENGTH).

4.2 Main()

Nella funzione main vengono chiamate tutte le varie funzioni.

4.3 Percorso

Inizialmente viene cercato nel grafo l' indirizzo del nodo di partenza e del nodo di destinazione.

Per l' algoritmo di Dijkstra non viene utilizato l' insime per i vertici già considerati perchè sarebbe ridondante in questo caso.

4.4 Mediana

Prima del calcolo della mediana (e media) viene costruita una lista da tutti i nodi. Così possiamo ordinare la lista senza distruggere l'ordine iniziale.

Calcolo della mediana:
(Le righe non significatve sono state rimosse)

```
double calcMedian(tList *nodeList) {
  int length;
  double median;
  length = listLength(nodeList);
  if (length % 2 == 0)
    median = (findElement(nodeList, length/2) +
      findElement(nodeList, length/2 + 1)) / 2;
  else
    median = findElement(nodeList, length/2 + 1);
  return median;
}
```

L'implementazione di quickselect:

```
 double findElement(tList *nodeList, int median) {
  int x = listLength(nodeList);
  int lengthMinorList;
  int lengthEqualList;
  double randomMinDistance =
    getMinDistance(nodeList,
      (int) (x * ((double)rand() / RAND_MAX)));
  double result;
  tList * minorList = NULL;
  tList * equalList = NULL;
  tList * majorList = NULL;
  tList * listEl;

  for (listEl = nodeList; listEl != NULL; listEl = listEl->next) {
    if (listEl->node->minDistance < randomMinDistance)
      addList(&minorList, listEl->node);
    else if (listEl->node->minDistance == randomMinDistance)
      addList(&equalList, listEl->node);
    else if (listEl->node->minDistance > randomMinDistance)
      addList(&majorList, listEl->node);
  }
  lengthMinorList = listLength(minorList);
```

```
  lengthEqualList = listLength(equalList);
  if (median <= lengthMinorList)
    result = findElement(minorList, median);
  else if (median > lengthMinorList + lengthEqualList)
    result = findElement(majorList,
      median - lengthMinorList - lengthEqualList);
  else
    result = randomMinDistance;

  return result;
}
```

5 Testing del programma

5.1 Output del programma se vengono utilizzati i dati dall'esempio

```
./distance
loaded this data from 'database.txt':
Arcs of: v_a
  v_b with distances: [0] 8.20, [1] 5.30, [2] 9.70
  v_c with distances: [0] 2.50, [1] 1.40, [2] 3.20
  v_e with distances: [0] 3.60, [1] 5.00, [2] 2.70
Arcs of: v_b
  v_a with distances: [0] 1.40, [1] 5.20, [2] 0.10
  v_c with distances: [0] 8.50, [1] 11.40, [2] 0.20
  v_d with distances: [0] 6.00, [1] 2.20, [2] 2.70
  v_e with distances: [0] 6.90, [1] 2.40, [2] 2.80
Arcs of: v_c
  v_d with distances: [0] 2.70, [1] 6.20, [2] 1.10
  v_e with distances: [0] 3.80, [1] 4.40, [2] 3.40
Arcs of: v_d
  v_a with distances: [0] 18.20, [1] 7.30, [2] 19.70
  v_c with distances: [0] 12.50, [1] 1.60, [2] 5.40
  v_e with distances: [0] 11.60, [1] 3.20, [2] 12.70
Arcs of: v_e
  v_a with distances: [0] 12.60, [1] 16.20, [2] 14.10
...
```

5.2 Output del programma per il percorso da v_a a v_b e tipologia 0

```
...
Enter the start node
v_a
Enter the end node
v_b
Enter the typology
0
Start and end Node found!
Start routing
The route between v_a and v_b is direct.
The distance between v_a and v_b is 8.200000
Average distance of v_a with all other nodes is: 4.875000
Median distance of v_a with all other nodes is 4.400000
```

5.3 Output del programma per il percorso da v_e a v_d e tipologia 0

```
...
Enter the start node
v_e
Enter the end node
v_d
Enter the typology
0
Start and end Node found!
Start routing
The route between v_e and v_d is
  v_a with distance 12.600000
  v_c with distance 15.100000
The distance between v_e and v_d is 17.800000
Average distance of v_e with all other nodes is: 16.575000
Median distance of v_e with all other nodes is 16.450000
```

5.4 Output del programma per il percorso da v_d a v_e e tipologia 0

```
...
Enter the start node
v_d
Enter the end node
v_e
Enter the typology
0
Start and end Node found!
Start routing
The route between v_d and v_e is direct.
The distance between v_d and v_e is 11.600000
Average distance of v_d with all other nodes is: 17.175000
Median distance of v_d with all other nodes is 15.350000
```

5.5 Output del programma per il percorso da v_d a v_c e tipologia 1

```
...
Enter the start node
v_d
Enter the end node
v_c
Enter the typology
1
Start and end Node found!
Start routing
The route between v_d and v_c is direct.
The distance between v_d and v_c is 1.600000
Average distance of v_d with all other nodes is: 6.175000
Median distance of v_d with all other nodes is 5.250000
```

5.6 Output del programma per il percorso da v_a a v_b e tipologia 2

```
...
Enter the start node
v_a
Enter the end node
v_b
Enter the typology
2
Start and end Node found!
Start routing
The route between v_a and v_b is direct.
The distance between v_a and v_b is 9.700000
Average distance of v_a with all other nodes is: 4.975000
Median distance of v_a with all other nodes is 3.750000
```

5.7 Output del programma per il percorso da v_b a v_a e tipologia 2

```
...
Enter the start node
v_b
Enter the end node
v_a
Enter the typology
2
Start and end Node found!
Start routing
The route between v_b and v_a is direct.
The distance between v_b and v_a is 0.100000
Average distance of v_b with all other nodes is: 1.100000
Median distance of v_b with all other nodes is 0.750000
```

6 Valutazione della complessità del programma

Visto che l' algoritmo viene applicato su struttura a grafo, la complessità può essere espressa in funzione del numero di vertici e di archi del grafo stesso. A questo scopo verrà indicato con |V| il numero di vertici e con |E| (edges) il numero totale di archi.

La complessità dell'algoritmo di acquisizione:

$$T(|V|, |E|) = 1+ |V| + |E| = O(|V| + |E|)$$

La complessità dell'algoritmo per calcolare il percorso più breve tra due vertici (Algoritmo di Dijkstra):

$$T\ (|V|, |E|) = O(|V| \text{ x } \log|V| + |E|)$$

La complessità dell'algoritmo per calcolare la media:

$$T(|V|, |E|) = O(|V|)$$

La complessità dell'algoritmo per calcolare la mediana (quickselect):

Caso ottimo:

$$T(|V|, |E|) = |V| + |V| + T(|V| \ / \ 2) = O(|V|)$$

Caso pessimo:

$$T(|V|, |E|) = |V| + |V| + T(|V| - 1) = O(|V|^2)$$

6.1 Valutazione sperimentale della complessità del calcolo di media e mediana

Questi grafi risultano quando viene fornito come input dell'algoritmo una sequenza di distanze (di N numeri reali), generata casualmente, per valori crescenti di N.

6.1.1 media

Nei risultati della valutazione sperimentale si vede che la complessità del calcolo della media è O(|V|).

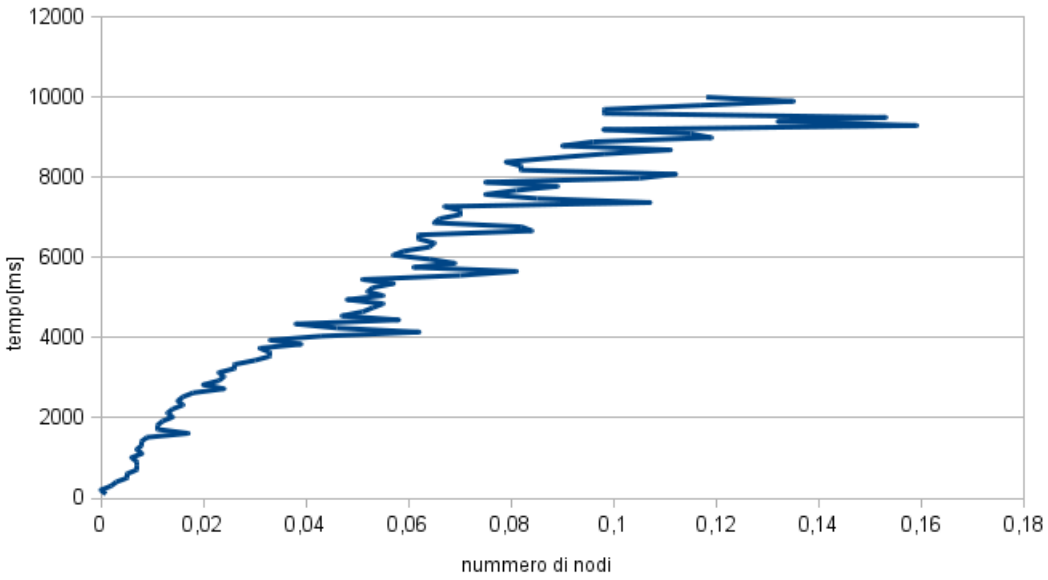

Figure 1: complessità del calcolo della media

6.1.2 mediana

Anche il risultato dalla valutazione sperimentale del calcolo della mediana non sorprende. Qui si vede il confronto dei calcoli di media e di mediana. La complessità varia molto perchè quickselect è casuale quindi dipende molto dal vertice di partenza.

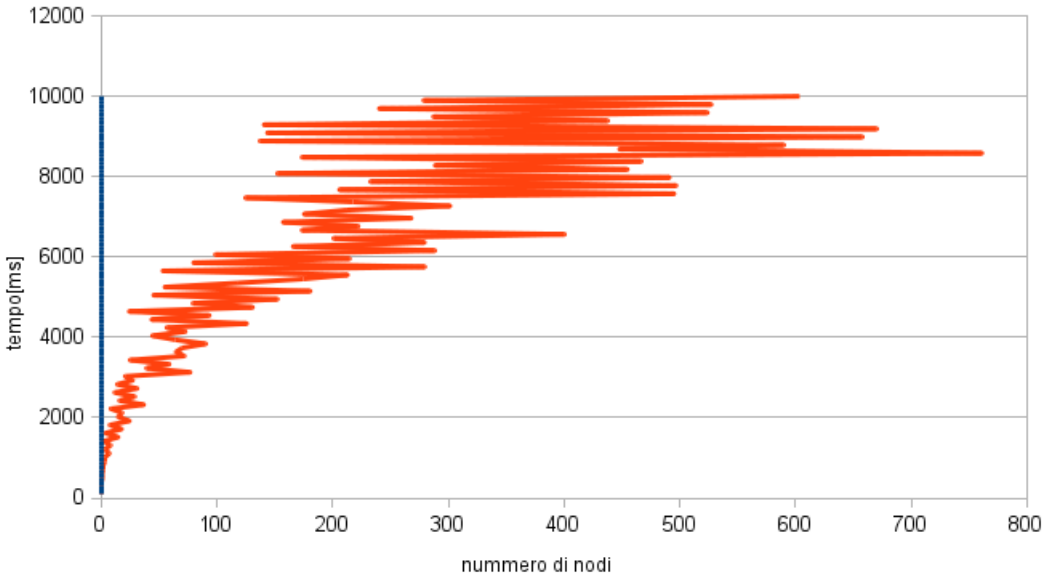

Figure 2: complessità del calcolo di media (blu) e mediana (rosso)