

Implementazione del Metodo Ungherese

Corso di Ricerca Operativa 2012-2013

Roberto Roberti

roberto.roberti6@unibo.it

28 Maggio 2013

Introduzione

- Implementare il Metodo Ungherese per il Problema dei Trasporti
 - ▶ Definizione dell'input
 - ▶ Definizione dell'output
 - ▶ Definizione delle strutture dati
 - ▶ Analisi dell'algoritmo
- Implementazione in C in ambiente Visual Studio 2008/2010/2012 (facoltativo l'uso di C++ e/o di un altro IDE)
- Test su 8 problemi test disponibili su <http://campus.unibo.it/> e su problemi generati casualmente

Definizione dell'Input

Istanze Test Fornite

Un esempio di file di input

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

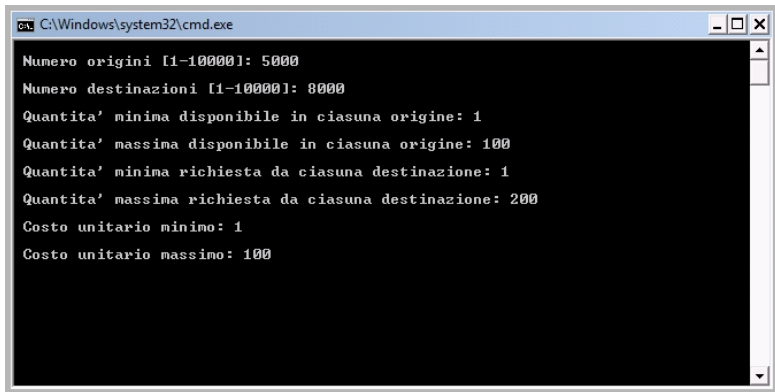
- la prima riga indica il numero di origini ($m = 3$) e di destinazioni ($n = 4$)
- la seconda riga riporta la disponibilità a_i per ogni origine $i = 1, \dots, m$
- la terza riga riporta la richiesta b_j per ogni destinazione $j = 1, \dots, n$
- segue la matrice dei costi, c_{ij} , riportata per righe

Definizione dell'Input

Istanze Random

Prevedere una routine che genera istanze con le seguenti caratteristiche

- numero di origini e numero di destinazioni passate da console
- disponibilità alle origini e richieste delle destinazioni generati casualmente in un range definito da console
- costi random appartenenti a un range definito da console



```
C:\Windows\system32\cmd.exe

Numero origini [1-10000]: 5000
Numero destinazioni [1-10000]: 8000
Quantita' minima disponibile in ciasuna origine: 1
Quantita' massima disponibile in ciasuna origine: 100
Quantita' minima richiesta da ciasuna destinazione: 1
Quantita' massima richiesta da ciasuna destinazione: 200
Costo unitario minimo: 1
Costo unitario massimo: 100
```

Definizione dell'Output

L'applicazione deve restituire in un file di testo le seguenti informazioni

- valore della funzione obiettivo
- valore delle variabili primali positive
- valore di tutte le variabili duali
- esito della verifica della correttezza del risultato
 - ▶ ammissibilità primale
 - ▶ ammissibilità duale
 - ▶ scarti complementari
- numero di cammini aumentanti calcolati
- numero di aggiornamenti della soluzione duale
- tempo di calcolo

Cosa È Richiesto alla Discussione dell'Elaborato

- Il codice sorgente sviluppato e l'eseguibile usato per i test finali, accertandosi che tutto funzioni sui computer del laboratorio
- Una relazione di una facciata con
 - ▶ nome e cognome
 - ▶ matricola
 - ▶ data
 - ▶ linguaggio di programmazione
 - ▶ processore sui cui sono stati svolti i test
 - ▶ una tabella che riporti, per ognuna delle otto istanze
 - ▶ nome dell'istanza
 - ▶ costo della soluzione ottima trovata
 - ▶ tempo di calcolo

Strutture Dati

```
1 #define MAX_M      5000
2 #define MAX_N      5000
3
4 // Dati di input
5 int m, n;           // #origini, #destinazioni
6 int a[MAX_M];       // disponibilita alle origini
7 int b[MAX_N];       // richieste alle destinazioni
8 int c[MAX_M][MAX_N]; // matrice dei costi
9
10 // Variabili primali e duali
11 int x[MAX_M][MAX_N]; // Matrice delle variabili primali
12 int u[MAX_M];        // Variabili duali vincoli origini
13 int v[MAX_N];        // Variabili duali vincoli destinazioni
14
15 // Matrice dei costi ridotti
16 int cr[MAX_M][MAX_N]; // matrice dei costi ridotti
17
18 // Variabili del primale ristretto
19 int xo[MAX_M];       // Variabili artificiali origini
20 int xd[MAX_N];       // Variabili artificiali destinazioni
21
22 // Variabili per l'etichettamento
23 int po[MAX_M];       // Padri per origini
24 int pd[MAX_N];       // Padri per destinazioni
```

Main

```
1 void main(void ) {  
2     /* leggi istanza (una delle tre seguenti istruzioni) */  
3     problema_slide();  
4     // leggi_da_file();  
5     // genera_random();  
6  
7     Ungherese();  
8  
9     flag = VerificaScarti();  
10    /* stampe finali */  
11 }
```


Inizializza Istanza Lucidi

```
1 void problema_slide(void ) {  
2     m = 4;  
3     n = 5;  
4  
5     a[1] = 4; a[2] = 5; a[3] = 7; a[4] = 4;  
6     b[1] = 3; b[2] = 6; b[3] = 4; b[4] = 5; b[5] = 2;  
7  
8     c[1][1] = 23; c[1][2] = 18; c[1][3] = 9; c[1][4] = 30; c[1][5] = 26;  
9     c[2][1] = 19; c[2][2] = 21; c[2][3] = 7; c[2][4] = 23; c[2][5] = 29;  
10    c[3][1] = 18; c[3][2] = 16; c[3][3] = 14; c[3][4] = 20; c[3][5] = 24;  
11    c[4][1] = 15; c[4][2] = 20; c[4][3] = 6; c[4][4] = 18; c[4][5] = 25;  
12  
13    for ( i = 1 ; i <= m ; ++i )  
14        for ( j = 1 ; j <= n ; ++j )  
15            cr[i][j] = c[i][j];  
16 }
```

Metodo Ungherese

```
1 void Ungherese(void ) {  
2     int s, t, delta;  
3  
4     soluzione_iniziale_duale();  
5     soluzione_iniziale_primale_ristretto();  
6  
7     while ( ( s = origine_esposta() ) >= 1 ) {  
8         if ( ( t = cammino_aumentante(s) ) >= 1 ) {  
9             delta = incremento_flusso(s,t);  
10            aggiorna_primale_ristretto(s,t,delta);  
11        } else {  
12            aggiorna_soluzione_duale();  
13        }  
14    }  
15 }
```

Soluzione Iniziale Duale

```
1 void soluzione_iniziale-duale(void ) {  
2  
3     for ( i = 1 ; i <= m ; ++i ) {  
4         u[i] = cr[i][1];  
5         for ( j = 2 ; j <= n ; ++j )  
6             if ( cr[i][j] < u[i] )  
7                 u[i] = cr[i][j];  
8     }  
9  
10    for ( i = 1 ; i <= m ; ++i )  
11        for ( j = 1 ; j <= n ; ++j )  
12            cr[i][j] -= u[i];  
13  
14    for ( j = 1 ; j <= n ; ++j ) {  
15        v[j] = cr[1][j];  
16        for ( i = 2 ; i <= m ; ++i )  
17            if ( cr[i][j] < v[j] )  
18                v[j] = cr[i][j];  
19    }  
20  
21    for( i = 1 ; i <= m ; ++i )  
22        for ( j = 1 ; j <= n ; ++j )  
23            cr[i][j] -= v[j];  
24 }
```

Soluzione Iniziale Primale Ristretto

```
1 void soluzione_iniziale_primale_ristretto(void) {  
2     for ( i = 1 ; i <= m ; ++i )  xo[i] = a[i];  
3  
4     for ( j = 1 ; j <= n ; ++j )  xd[j] = b[j];  
5  
6     for ( i = 1 ; i <= m ; ++i )  
7         for ( j = 1 ; j <= n ; ++j )  
8             if ( cr[i][j] == 0 ) {  
9                 x[i][j] = MIN( xo[i], xd[j] );  
10                xo[i]    -= x[i][j];  
11                xd[j]    -= x[i][j];  
12            }  
13 }
```

Origine Esposta

```
1 | int origine_esposta(void ) {  
2 |     for ( i = 1 ; i <= m ; ++i )  
3 |         if ( xo[i] > 0 )  
4 |             return i;  
5 |  
6 |     return 0;  
7 | }
```

Cammino Aumentante

```
1 int cammino_aumentante(int s) {
2     po[s] = 0;
3
4     while ( /* esiste k da espandere */ ) {
5         if ( /* k origine */ ) {
6             for ( j = 1 ; j <= n ; ++j )
7                 if ( ( pd[j] == NIL ) && ( cr[k][j] == 0 ) ) {
8                     pd[j] = k;
9                     if ( xd[j] == 0 ) /* aggiungi j ai nodi da espandere */;
10                    else return j;
11                }
12            } else { /* k destinazione */
13                for ( i = 1 ; i <= m ; i++ )
14                    if ( ( po[i] == NIL ) && ( x[i][k] > 0 ) && ( cr[i][k] == 0 ) ) {
15                        po[i] = k;
16                        /* aggiungi i ai nodi da espandere */
17                    }
18            }
19        }
20
21        return 0;
22    }
```

Incremento del Flusso

```
1 int incremento_flusso(int s, int t) {  
2     delta = MIN( xo[s], xd[t] );  
3  
4     k = t;  
5     while ( 1 ) {  
6         k = pd[k];  
7         if ( k == s ) break;  
8         if ( x[k][po[k]] < delta ) delta = x[k][po[k]];  
9         k = po[k];  
10    }  
11  
12    return delta;  
13 }
```

Aggiorna Primale Ristretto

```
1 void aggiorna_primale_ristretto(int s, int t, int delta) {  
2     xo[s] -= delta;  
3     xd[t] -= delta;  
4  
5     k = t;  
6     while ( 1 ) {  
7         x[pd[k]][k] += delta;  
8         k = pd[k];  
9         if ( k == s ) break;  
10        x[k][po[k]] -= delta;  
11        k = po[k];  
12    }  
13 }
```


Aggiorna Soluzione Duale

```
1 void aggiorna_soluzione_duale(void )
2 {
3     for ( i = 1 ; i <= m ; ++i )
4         if ( /* i origine etichettata (i in I+) */ )
5             for ( j = 1 ; j <= n ; ++j )
6                 if ( /* j destinazione non etichettata (j in J-) */ )
7                     if ( cr[i][j] < delta )
8                         delta = cr[i][j];
9
10    for ( i = 1 ; i <= m ; ++i )
11        if ( /* i origine etichettata (i in I+) */ )
12            u[i] += delta;
13
14    for ( j = 1 ; j <= n ; ++j )
15        if ( /* j destinazione etichettata (j in J+) */ )
16            v[j] -= delta;
17
18    for ( i = 1 ; i <= m ; ++i )
19        for ( j = 1 ; j <= n ; ++j )
20        {
21            if ( /* i etichettata e j non etichettata (i in I+, j in J-) */ )
22                cr[i][j] -= delta;
23
24            if ( /* i non etichettata e j etichettata (i in I-, j in J+) */ )
25                cr[i][j] += delta;
26        }
27 }
```

Verifiche Scarti

```
1 | int VerificaScarti(void )  
2 | {  
3 |     /* vincoli del primale sono rispettati? */  
4 |     /* > dall'origine i partono a[i] unita? */  
5 |     /* > alla destinazione j arrivano b[j] unita? */  
6 |  
7 |     /* vincoli del duale sono rispettati? */  
8 |     /* >  $c[i][j] \geq u[i] + v[j]$  */  
9 |  
10 |    /* condizioni di ottimalita sono rispettate? */  
11 |    /* > gli scarti complementari sono rispettati? */  
12 | }
```