Strutture Dati

Lezione 16
Grafi

Oggi parleremo di …

- Operazioni
  - visita in profondità (fatto!)
  - visita in ampiezza
  - determinazione delle componenti connesse
  - alberi derivati (spanning tree).
- Problema del percorso minimo
  - Algoritmo di Dijkstra.

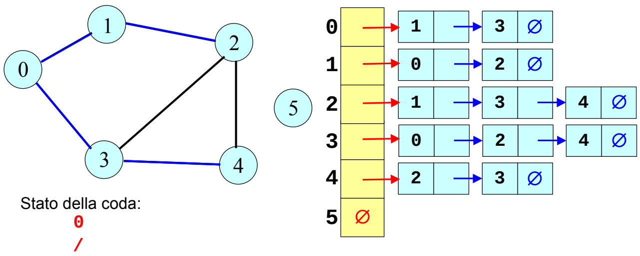
Operazioni sui grafi

- Dato un grafo *G*=(*V*,*L*) e un vertice *v*, visitare tutti i vertici raggiungibili da *v*
  - ricerca in profondità
  - ricerca in ampiezza.
- Ricercare tutte le componenti connesse di un grafo.
- Determinare l'albero derivato (spanning tree) di un grafo.

Visita in ampiezza di un grafo

- La ricerca in ampiezza è simile alla visita per livello di un albero.
- Si visita il nodo di partenza *v*.
- Si visitano tutti i nodi adiacenti a *v*.
- Si visitano tutti i nodi non visitati adiacenti al primo vertice nella lista di adiacenza di *v* e così via sino a quando tutti i vertici del grafo sono stati considerati.
- Quando si visita un vertice, lo si pone in una coda.
- Quando si esamina la sua lista di adiacenza, lo si elimina dalla coda.

Visita in ampiezza di un grafo



Stato della coda:

0
/
1 3
3
3 2
2
2 4
4
/

La sequenza dei vertici raggiungibili da 0 in ampiezza è:

0 1 3 2 4

Visita in ampiezza di un grafo

```
typedef struct coda *coda_pointer;
typedef struct coda {
                 int vertice;
                 coda_pointer link;
                };
```

```
void ram(int v)
{
 void addc(coda_pointer *davanti, coda_pointer *dietro, int item);
 int deletec(coda_pointer  *davanti);
 coda_pointer davanti, dietro;
 nodo_pointer w;

 davanti = dietro = NULL;   /*inizializza la coda*/
 printf("%5d",v);
 visitato[v]=TRUE;
 addc(&davanti,&dietro, v);
 while (davanti) {
    v=deletec(&davanti);
    for (w=grafo[v];w;w=w->link)
         if (!visitato[w->vertice]) {
                 printf("%5d", w->vertice);
                 addc(&davanti, &dietro, w->vertice);
                 visitato[w->vertice]=TRUE;
                 }
    }
}
```

$O(l)$

Componenti connesse

- Tutti i vertici visitati, insieme con i lati incidenti su di essi formano una componente connessa.
- Per verificare se un grafo è connesso basta
  - visitare con `rpr(0)` o `ram(0)`
  - controllare se ci sono vertici non visitati.
- Per determinare tutte le componenti connesse di un grafo basta effettuare chiamate ripetute di `rpr(v)` o `ram(v)` per ogni vertice non visitato `v`.

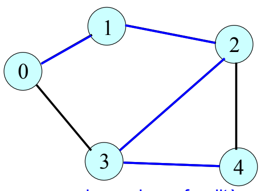
```
void connessa(void)
{
 int i;

 for(i=0;i<n; i++)
    if(!visitato[i])  {
                rpr(i);
                printf("\n");
        }
}
```

$O(n+l)$

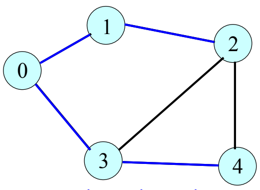
Alberi derivati

- Se G è un grafo connesso, l'insieme dei vertici con i lati attraversati durante una visita formano un albero che include tutti i nodi di G, detto albero derivato (o spanning tree).
  - se la visita è fatta in profondità, l'albero derivato si chiama albero derivato da ricerca in profondità
  - se la visita è fatta in ampiezza, l'albero derivato si chiama albero derivato da ricerca in ampiezza.

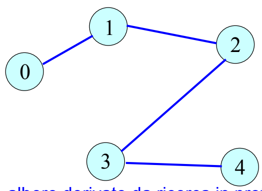
Alberi derivati



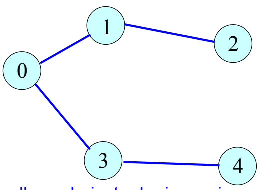
ricerca in profondità



ricerca in ampiezza



albero derivato da ricerca in profondità

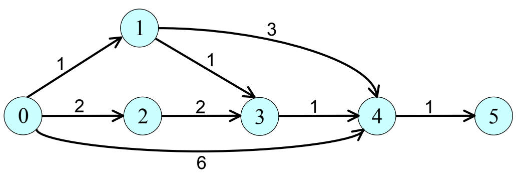


albero derivato da ricerca in ampiezza

Problema del percorso minimo

- Sia G un grafo orientato pesato
  - ad ogni lato a è associato un peso intero positivo p_a.
- Ai percorsi (orientati) nel grafo è assegnato un peso, dato dalla somma dei pesi degli archi che lo compongono.
- Dati due nodi x e y, il problema del percorso minimo consiste nel fornire un percorso da x a y di peso minimo.
- Se tutti i lati sono pesati 1, il problema si risolve con una visita in ampiezza.
- E se i pesi sono diversi da 1?

Problema del percorso minimo



Quale è il cammino minimo da 0 a 5?

- Una visita in ampiezza non risolve il problema
  - assegneremmo al nodo 5 un cammino di peso 5!
- E' necessario visitare per primi i nodi che hanno cammini minimi da 0 più brevi.
- E' necessario determinare i cammini in ordine crescente di lunghezza.

Algoritmo di Dijkstra

- Si visitano i nodi del grafo come in una ricerca.
- Ad ogni istante, l'insieme N dei nodi del grafo è suddiviso in tre parti:
  - l'insieme dei nodi visitati V
  - l'insieme dei nodi di frontiera F (successori dei nodi visitati)
  - l'insieme dei nodi da esaminare.
- Per ogni nodo z, l'algoritmo tiene traccia del valore d_z, inizialmente posto a ∞, e di un nodo u_z, inizialmente indefinito.

Algoritmo di Dijkstra

- Si sceglie dall'insieme F un nodo z con d_z minimo.
- Si sposta tale nodo z da F in V.
- Si spostano tutti i successori di z sconosciuti in F.
- Per ogni successore w di z si aggiornano i valori d_w e u_w secondo la regola seguente

$$d_w = \min\{d_w, d_z + p_a\}$$

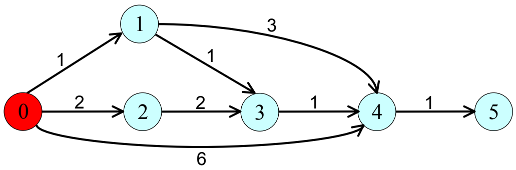
dove p_a è il peso dell'arco a che collega z a w.

- Se il valore di d_w è stato modificato, allora u_w viene posto uguale a z.
- Si ripetono i passi precedenti sino a raggiungere il nodo destinazione (se raggiungibile).

Algoritmo di Dijkstra

- L'algoritmo inizia con $V = \varnothing$, $F = \{x\}$ e $d_x = 0$.
- Si prosegue sino a raggiungere il nodo y o finchè $F = \varnothing$.
- Al termine dell'algoritmo
  - d_z contiene, per ogni nodo z, il peso del cammino minimo da x a z
  - il vettore u consente di ricostruire l'albero dei cammini minimi con origine in x.

Algoritmo di Dijkstra: esempio

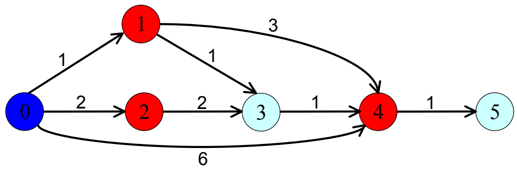


$V = \varnothing$ $F = \{0\}$

	0	1	2	3	4	5
d:	0	∞	∞	∞	∞	∞

	0	1	2	3	4	5
u:	?	?	?	?	?	?

Algoritmo di Dijkstra: esempio



$V = \{0\}$ $F = \{1, 2, 4\}$ $d_w = \min\{d_w, d_z + p_a\}$

Stato precedente

	0	1	2	3	4	5
d:	0	∞	∞	∞	∞	∞

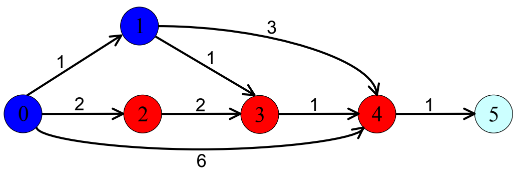
	0	1	2	3	4	5
u:	?	?	?	?	?	?

Stato corrente

	0	1	2	3	4	5
d:	0	1	2	∞	6	∞

	0	1	2	3	4	5
u:	?	0	0	?	0	?

Algoritmo di Dijkstra: esempio



$V = \{0, 1\}$ $F = \{2, 4, 3\}$ $d_w = \min\{d_w, d_z + p_a\}$

Stato precedente

	0	1	2	3	4	5
d:	0	1	2	∞	6	∞

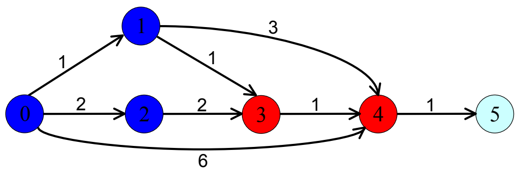
	0	1	2	3	4	5
u:	?	0	0	?	0	?

Stato corrente

	0	1	2	3	4	5
d:	0	1	2	2	4	∞

	0	1	2	3	4	5
u:	?	0	0	1	1	?

Algoritmo di Dijkstra: esempio



$V = \{0, 1, 2\}$ $F = \{4, 3\}$ $d_w = \min\{d_w, d_z + p_a\}$

Stato precedente

	0	1	2	3	4	5
d:	0	1	2	2	4	∞

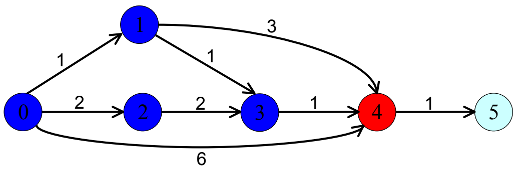
	0	1	2	3	4	5
u:	?	0	0	1	1	?

Stato corrente

	0	1	2	3	4	5
d:	0	1	2	2	4	∞

	0	1	2	3	4	5
u:	?	0	0	1	1	?

Algoritmo di Dijkstra: esempio



$V = \{0, 1, 2, 3\}$ $F = \{4\}$ $d_w = \min\{d_w, d_z + p_a\}$

Stato precedente

	0	1	2	3	4	5
d:	0	1	2	2	4	∞

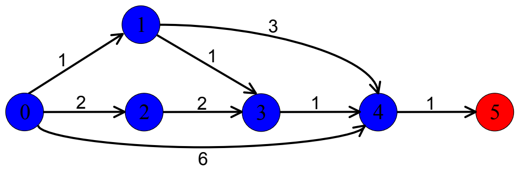
	0	1	2	3	4	5
u:	?	0	0	1	1	?

Stato corrente

	0	1	2	3	4	5
d:	0	1	2	2	3	∞

	0	1	2	3	4	5
u:	?	0	0	1	3	?

Algoritmo di Dijkstra: esempio



$V = \{0, 1, 2, 3, 4\}$ $F = \{5\}$ $d_w = \min\{d_w, d_z + p_a\}$

Stato precedente

	0	1	2	3	4	5
d:	0	1	2	2	3	∞

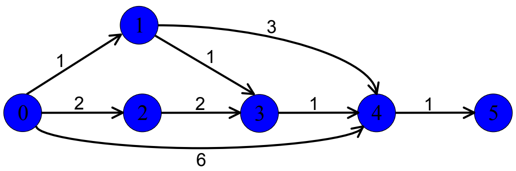
	0	1	2	3	4	5
u:	?	0	0	1	3	?

Stato corrente

	0	1	2	3	4	5
d:	0	1	2	2	3	4

	0	1	2	3	4	5
u:	?	0	0	1	3	4

Algoritmo di Dijkstra: esempio



$V = \{0, 1, 2, 3, 4, 5\}$ $F = \{ \varnothing \}$

	0	1	2	3	4	5
d:	0	1	2	2	3	4

	0	1	2	3	4	5
u:	?	0	0	1	3	4

Il cammino minimo da 0 a 5 è: `0, 1, 3, 4, 5`

Algoritmo di Dijkstra

```
void percorso_min(int v,int costo[ ][MAX_VERTICI], int distanza[ ], int n,
   short int trovato[ ])
{
 /* distanza[i]  rappresenta il percorso minimo dal vertice v a i;
    trovato[i]   contiene 0 se il percorso minimo dal vertice i non e'
                 stato trovato, altrimenti 1;
    costo        e' la matrice di adiacenza. */

   int i, u, w;

   for (i = 0; i < n; i++) {
        trovato[i] = FALSE;
        distanza[i] = costo[v][i];
        }
   trovato[v] = TRUE;
   distanza[v] = 0;
   for (i = 0; i < n-2; i++) {
        u = scegli(distanza, n, trovato);
        trovato[u] = TRUE;
        for (w = 0; w < n; w++)
             if(!trovato[w])
                 if (distanza[u] + costo[u][w] < distanza[w])
                           distanza[w] = distanza[u] + costo[u][w];
   }
}
```

$O(n^2)$

Algoritmo di Dijkstra

```
int scegli(int distanza[ ], int n, short int trovato[ ])
{
 /* trova la distanza piu' piccola non ancora controllata */

 int i, min, min_pos;

 min = INT_MAX;
 min_pos = -1;
 for (i = 0; i < n; i++)
   if(distanza[i] < min && !trovato[i]) {
         min = distanza[i];
         min_pos = i;
   }
 return min_pos;
}
```