Laboratorio di Algoritmi e Strutture Dati

Prof. Aniello Murano

Grafi: Implementazione ed operazioni di base

Corso di Laurea: Laboratorio di Algoritmi e Strutture Dati
Codice insegnamento: 13917
Email docente: murano@na.infn.it
Anno accademico: 2007/2008

Lezione numero: 15

Parole chiave: Grafi, definizione e rappresentazioni

next

Sommario delle lezioni sui grafi

I grafi sono un potente strumento per la rappresentazione di problemi complessi

La soluzione di moltissimi problemi può essere ricondotta alla soluzione di opportuni problemi su grafi.

Nel contesto dei grafi saranno approfonditi i seguenti argomenti:

- Definizioni e rappresentazione di grafi:
  - Lista di adiacenza
  - Matrice di adiacenza
- Algoritmi di base su grafi
  - Ricerca in ampiezza (BFS)
  - Ricerca in profondità (DFS)
- Algortimi avanzati sui grafi
  - Albero minimo di copertura (Minimum Spanning Tree)
  - Percorso minimo tra due vertici

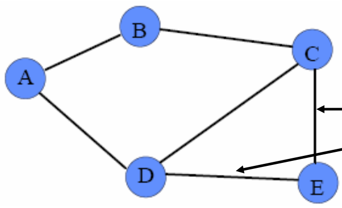
back next

Definizione di Grafo

Un grafo è una coppia **(V, E)**, dove

- **V** è un insieme di nodi, chiamati vertici
- **E** è un insieme di coppie di nodi, chiamati archi
- Un **arco** è una coppia **(v,w)** di vertici in **V**

Esempio:

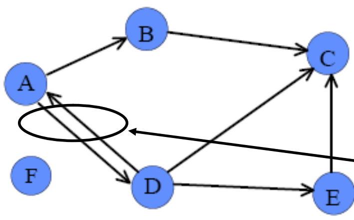


V = {A, B, C, D, E, F}
E = {(A,B), (A,D), (B,C), (C,D), (C,E), (D,E)}

Grafi orientati e non orientati

Un grafo (V,E) è non orientato se l'insieme degli archi E è un insieme di coppie non ordinate

Un grafo (V,E) è orientato se l'insieme degli archi E è una relazione binaria tra vertici.



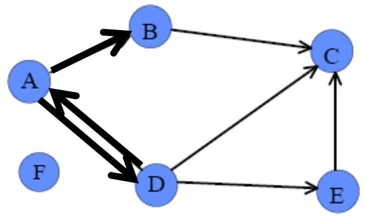
V = {A, B, C, D, E, F}
E = {(A,B), (A,D), (B,C), (C,D), (C,E), (D,E)}

(A,D) *e* (D,A) denotano due archi diversi

Proprietà di un grafo

In un grafo orientato, (w,v) si dice incidente da w a v, e v adiacente a w.

In un grafo non orientato, incidenze e adiacenze sono simmetriche.



- •(*A*,*B*) è ***incidente*** da *A* a ***B***
- •(*A*,*D*) è ***incidente*** da *A* a ***D***
- •(***D***,*A*) è ***incidente*** da ***D*** a *A*

In un grafo non orientato il grado di un vertice è il numero di archi che da esso si dipartono. Per es., A ha grado 2, mentre F ha grado 0.

In un grafo orientato il grado entrante (uscente) di un vertice è il numero di archi incidenti in (da) esso. Per esempio A ha grado uscente 2 e grado entrante 1.

Percorsi sui Grafi

- Sia G = (V, E) un grafo. Un percorso nel grafo è una sequenza di vertici $<w_1, w_2,...,w_n>$ dove per ogni i, (wi, w_{i+1}) è un arco di E
- La lunghezza del percorso è il numero totale di archi che connettono i vertici nell'ordine della sequenza.
- Un percorso si dice semplice se tutti i suoi vertici sono distinti (compaiono una sola volta nella sequenza), eccetto al più il primo e l'ultimo che possono coincidere.
- Se esiste un percorso p tra i vertici v e w, si dice che w è raggiungibile da v tramite p.
- Un ciclo in un grafo è un percorso $<w_1,\ldots,w_n>$ tale che $w_1 = w_n$
- Un grafo senza cicli è detto aciclico.
- Un grafo è completo se ha un arco tra ogni coppia di vertici.
- Maggiori dettagli sulle slide di teoria del Prof. Benerecetti……

Implementazione di grafi

Per rappresentare un grafo si può utilizzare:

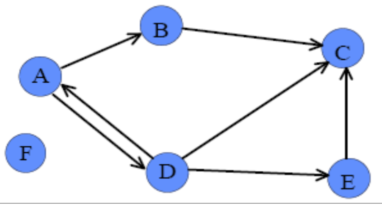
- Una Lista di Adiacenza
- Una matrice di Adiacenza.

Matrice di adiacenza

Supponiamo di avere un Grafo G di n nodi 0,1,…,n e di volerlo rappresentare con una matrice di adiacenza. Si definisce allora una matrice M[n,n] riempita utilizzando la seguente regola

$$M(v, w) = \begin{cases} 1 & \text{se } (v, w) \in E \\ 0 & \text{altrimenti} \end{cases}$$

Per esempio, il seguente grafo costituito da 6 nodi è rappresentato dalla seguente matrice M[6,6]



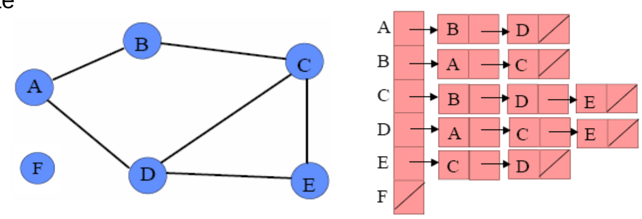
	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	0	0	0	1	0
D	1	0	1	0	0	0
E	0	0	0	1	0	0
F	0	0	0	0	0	0

Liste di adiacenza

Dovendo rappresentare un grafo G con n nodi ordinati con una lista di adiacenza, si definiscono n liste (una per ogni vertice) riempite nel modo seguente. Per ogni nodo v del grafo,

$$L(v) = \text{lista di } w \text{, tale che } (v, w) \in E,$$

Per esempio, il seguente grafo di 6 nodi è rappresentato nel modo seguente



Complessità

Matrice di adiacenza

- **Spazio richiesto O(|V|^2)**
- **Verificare se i vertici *u* e *v* sono adiacenti richiede tempo O(1)**
- **Molti 0 nel caso di *grafi sparsi***

Liste di adiacenza

- **Spazio richiesto O(|E|+|V|)**
- **Verificare se i vertici *u* e *v* sono adiacenti richiede tempo O(|V|).**

Interrogazione di un Grafo 1/2

Di seguito elenchiamo alcune delle operazioni più comuni di interrogazioni su un grafo G:

- Isempty(G): restituisce TRUE se il il grafo è vuoto.
- numVertices(G): restituisce il numero di vertici.
- numEdges(G): restituisce il numero di archi.
- endVertices(G,e): Restituisce le due estremità dell'arco e.
- Grado(G,v): Restituisce il grado di un nodo v.

Interrogazione di un Grafo 2/2

Di seguito elenchiamo alcune delle operazioni più comuni di interrogazioni su un grafo G:

- Adiacente(G,v): Restituisce i vertici adiacenti al vertice v.
- Incidente (G,v): Restituisce i vertici incidenti sul vertice v.
- SonoAdiacenti(G, v, w): Restituisce TRUE se i vertici v e w sono adiacenti.
- Completo(G): Restituisce TRUE se G è completo.
- Stampa(G): Stampa il grafo.

Aggiornamento di un Grafo

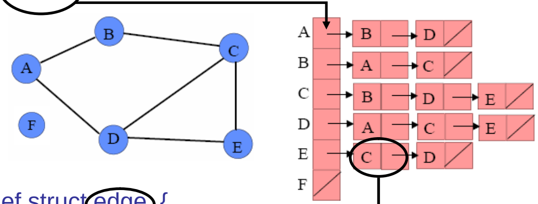
Di seguito elenchiamo alcune delle operazioni più comuni sulla modifica di un grafo G:

- Crea_grafo(n)
- Aggiungi_vertice(G,v)
- Aggiungi_arco(G,e)
- Rimuovi_vertice(G,v)
- Rimuovi_arco(G,e)
- Free(G)

Implentazione di un grafo

Implementazione della rappresentazione di un grafo orientato tramite lista di adiacenze

```
typedef struct graph {
    int     nv;   /* numero di vertici del grafo */
    edge **adj;   /* vettore con le liste delle adiacenze */ } graph;
```

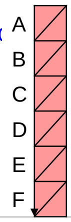



```
typedef struct edge {
    int key;
    struct edge *next; } edge;
```

Creazione di un grafo

Sia G un grafo non orientato di n elementi 0,1,…,n-1 senza archi, da rappresentare con liste di adiacenza. La seguente funzione crea e restituisce un puntatore ad una struttura dati grafo con n liste di adiacenza vuote. Si noti come la presenza di ogni nodo **i** è implicita nella allocazione di memoria per la lista (vuota) dei nodi adiacenti ad **i**.

```
graph *g_empty(int n)
{graph *G;  int i;
  G = (graph*)malloc(sizeof(graph));
  if (G==NULL) printf("ERRORE: impossibile allocare memoria per il
    grafo\n");
  else {
     G->adj = (edge**)malloc(n*sizeof(edge*));
     if (G->adj==NULL)  {
        printf("ERRORE: imp. allocare mem. per la lista del graf
        free(G);
          G=NULL;}
     else {
          G->nv = n;
          for (i=0; i<n; i++)
                G->adj[i]=NULL;}}
  return(G); }
```



Stampa di un grafo

La seguente funzione controlla se un grafo è vuoto:

```
int is_empty(graph *G)  { return (G==NULL); }
```

La seguente funzione serve a stampare un grafo G non orientato. Si ricordi che il grafo ha n nodi ordinati 0,1,…,n-1

```
void g_print(graph *G)
{    int i, ne=0;
     edge *e;
     if (!is_empyt(G))
     {            printf("\n Il grafo ha %d vertici\n", G->nv);
                  for (i=0; i<G->nv; i++)
                  {            printf("nodi adiacenti al nodo %d -> ", i);
                               e=G->adj[i];
                               while (e!=NULL)
                               {            printf("%d ", e->key); ne=ne+1; e=e->next;   }
                               printf("\n");
                  }
                  printf("\n Il grafo ha %d archi \n", ne); } }
```

Aggiunta di un arco

Mostriamo una funzione che inserisce l'arco (u,v) in un grafo G.

Precondizioni (da controllare in fase di implementazione!!!):

- G diverso da NULL; u e v vertici (compresi tra 0 e G->nv-1); l'arco (u,v) non è già nel grafo

```
void g_add(graph *G, int u, int v)
{
  edge *new, *e;
  [.... Aggiungere frammento di codice per i controlli vari]
  new = (edge*)malloc(sizeof(edge));
  if (new==NULL) printf("ERRORE: impossibile allocare memoria \n");
  else {
          new->key=v; new->next=NULL;
          if (G->adj[u] == NULL) //u non ha archi //
              G->adj[u] = new;
          else {
              e=G->adj[u];
              while (e->next!=NULL) e=e->next;
              e->next=new; }              }           }
```

Si può anche aggiungere il nodo in testa. Si deve comunque scorrere tutta la lista per controllare se il nodo è già presente.

Rimozione di un arco

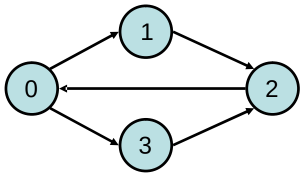
Di seguito mostriamo una funzione per la rimozione di un arco

Prerequisiti: G non NULL; u e v vertici del grafo (compresi tra 0 e G->nv-1); l'arco (u,v) esiste

```
void  g_remove_edge(graph *G, int u, int v)
{
    edge *prev; /* l'arco precedente a quello da togliere nella lista */
    edge *e; /* l'arco da togliere dalla lista */
    e=G->adj[u];
    if (e->key == v)
       G->adj[u] = e->next;
    else
    {  prev=e;
       while (prev->next->key != v)    prev=prev->next;
       e=prev->next;
       prev->next=e->next;
    }
    free(e);
}
```

Grafo Trasposto

- Sia G=(V,E) un grafo orientato. Il trasposto di G è un grafo G'=(V,E'), dove E' è l'insieme degli archi (v,u) tali che (u,v) è un arco di E.
- Dunque, il grafo trasposto G' è il grafo G con tutti i suoi archi invertiti.
- Per esempio, si consideri il seguente grafo, la sua rappresentazione e quella del suo trasposto con matrici di adiacenza:



	0	1	2	3
0	0	1	0	1
1	0	0	1	0
2	1	0	0	0
3	0	0	1	0

	0	1	2	3
0	0	0	1	0
1	1	0	0	0
2	0	1	0	1
3	1	0	0	0

- Nel caso di rappresentazione con matrice di adiacenza, il grafo G' è rappresentato dalla trasposta di G che è dunque calcolata in O(V).
- Esercizio: Implementare un algoritmo efficiente per la computazione del trasposto di un grafo rappresentato con liste di adiacenza e confrontare la complessità asintotica dell'algoritmo con quella dell'algoritmo precedente.

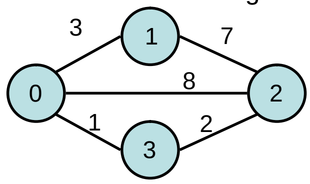
Grafi pesati

La struttura di un grafo può essere generalizzata associando un valore numerico, detto peso, ai suoi archi. Un grafo di tale genere si dice pesato. I grafi pesati vengono rappresentati mediante

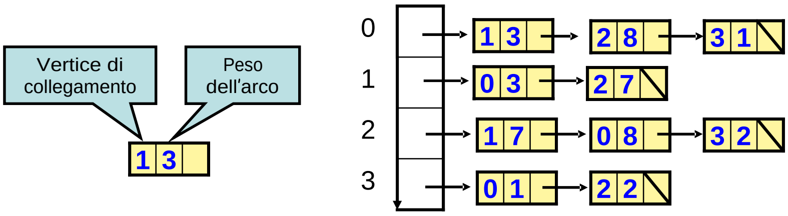
- Matrici di adiacenza: se l'elemento di indici i, j della matrice di adiacenza è un valore diverso da 0 esso è il peso dell'arco (i,j), altrimenti non esiste un arco fra i nodi i e j;
- Liste di adiacenza: un elemento della lista di adiacenza al nodo i contiene un campo per memorizzare il nome del nodo adiacente (ad esempio, j), un campo per memorizzare il peso dell'arco (nell'esempio, il peso dell'arco (i,j)) ed un campo per memorizzare il puntatore all'elemento successivo nella lista.

Rappresentazione con liste

Si consideri di nuovo il grafo dell'esempio precedente:

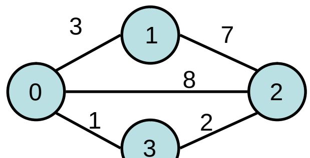



```
typedef struct edge
{     int key;
      int peso;
      struct edge *next;} edge;
```



Rappresentazione con matrice

Per esempio, si consideri il seguente grafo:



Matrice di adiacenza

	0	1	2	3
0	0	3	8	1
1	3	0	7	0
2	8	7	0	2
3	1	0	2	0

Matrice come vettore di puntatori