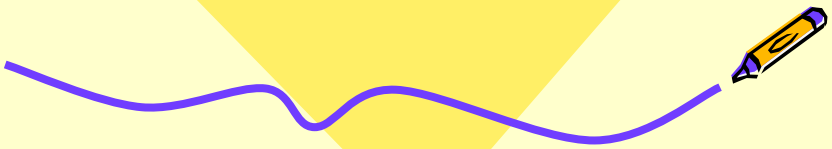
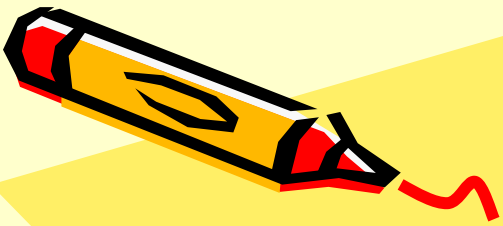
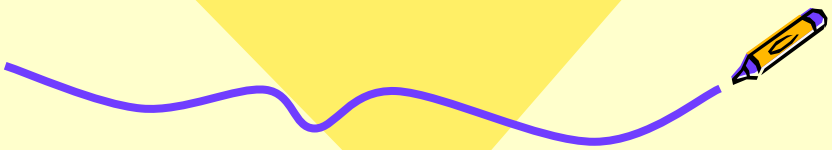
Laboratorio di Algoritmi e Strutture Dati

Aniello Murano
http://people.na.infn.it/~murano/





Introduzione ai Grafi: Implementazione e operazioni di base



Informazione Generali (1)

- Orario lezioni:
  - Giovedì ore 11-13 aula F1
  - Venerdì ore 14-16 Laboratorio
- Durata complessiva del modulo B:
  - 4 lezioni frontali e 5 di laboratorio.
- Ricevimento
  - Giovedì ore 15:00 – 16:00
  - Venerdì ore 16:00 – 18:00
  - Ufficio 2Ma10 Dip. Fisica (2 piano edificio M)

Informazione Generali (2)

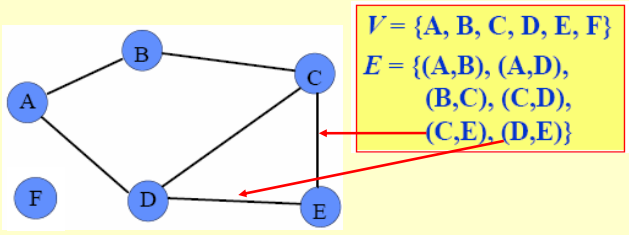
- Lucidi del Corso:
  - Di volta in volta verranno messi a disposizione al seguente URL: http://people.na.infn.it/~murano/Didattica.html
- Indirizzo e-mail:
  - murano@na.infn.it

Proposta di Programma del modulo B

- 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.
  - Algoritmi di base su grafi
    - ✓ Ricerca in ampiezza (BFS)
    - ✓ Ricerca in profondità (DFS)
  - Algortimi avanzati sui grafi
    - ✓ Albero minimo di copertura (Minimum Spanning Tree)
    - ✓ Ricerche dei Percorsi Minimi in un grafo
    - ✓ Percorso minimo tra due vertici
    - ✓ Percorsi minimi tra tutti i vertici
  - Programmazione dinamica.

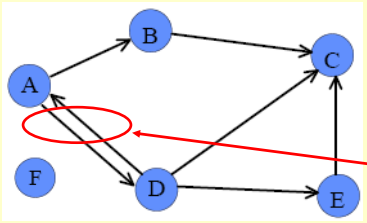
Definizione di Grafo

- 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:



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.

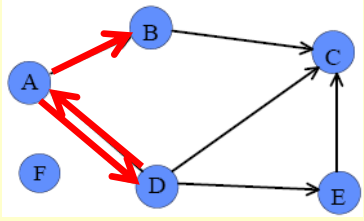


$V = \{A, B, C, D, E, F\}$
$E = \{(A,B), (A,D), (B,C), (C,D), (C,E), (D,E)\}$

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

Proprietà di un grafo

- In un grafo orientato, un arco **(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 **<w1, w2,...,wn>** dove per ogni i, **(wi, wi+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 **<w1,…,wn>** tale che **w1 = wn.**
- 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

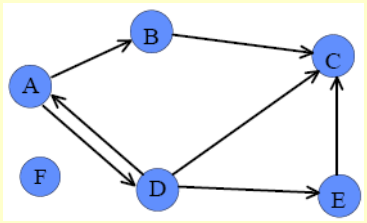
- Per rappresentare un grafo si può utilizzare:
- Una Lista di Adiacenza
- Una matrice di Adiacenza.

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]



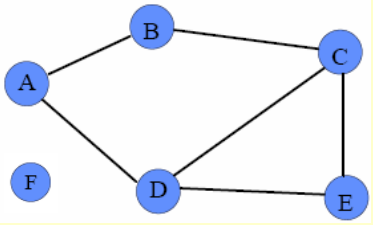
	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	0	0	0	1	0
D	1	0	1	0	0	0
E	0	0	0	1	0	0
F	0	0	0	0	0	0

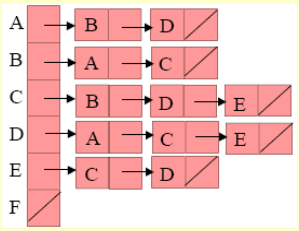
Matrice di adiacenza

- 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

- 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.
  - 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.
  - Fortemente_connesso(*G*): valuta se *G* è fortemente connesso.
  - Stampa(*G*): Stampa il grafo.

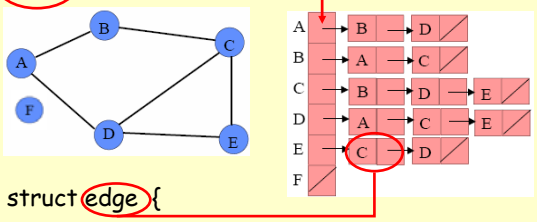
Aggiornamento di un Grafo

- 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 ;
```

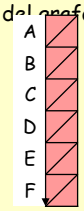



```
typedef struct edge {
   int key;
   struct edge *next; } edge;
```

Creazione di un grafo

Si supponga di avere un grafo non orientato di n elementi 0,1,…,n-1 senza archi e di volerlo rappresentare con liste di adiacenza. La seguente funzione crea e restituisce un puntatore ad una struttura dati grafo che conserva il numero dei vertici e definisce n liste di adiacenza vuote. Si noti come la presenza di ogni singolo nodo i è implicitamente definita 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) && (n>0)) {
          printf("ERRORE: impossibile allocare memoria per la lista del grafo\n");
          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);                }
  return;                                                               }
```

Aggiunta di un arco

- Mostriamo una funzione che inserisce l'arco (u,v) in un grafo G.
- Precondizioni:
  - G diverso da NULL; u e v vertici del grafo (compresi tra 0 e G->nv-1); l'arco (u,v) non è già presente nel grafo

```
void g_add(graph *G, int u, int v) {
   edge *new, *e;
   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) //il nodo u non ha archi //
                        G->adj[u] = new;
                        else {
                        e=G->adj[u];
                        while (e->next!=NULL) e=e->next;
                        e->next=new; }}
 return;}
```


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);
   return;}
```