Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Università degli Studi dell'Aquila

Corso di Algoritmi e Strutture Dati con Laboratorio

GRAFI – I parte

Grafo non orientato (indiretto)

- Un grafo non orientato $G=(V,E)$ è una raccolta di elementi distinti chiamati **vertici** $v \in V$ o **nodi** e coppie di vertici distinte e non ordinate $(u,v) \in E=VxV$ (relazione simmetrica), dette **archi.**
- Graficamente rappresentiamo i vertici come punti o cerchi e gli archi con linee che collegano coppie di vertici

Applicazioni:

- **Facebook**: I vertici sono gli individui e gli archi sono le relazioni di amicizia.
- **Mappa stradale**: I vertici sono le città e gli archi sono le strade (a doppio senso di marcia) che le connettono.

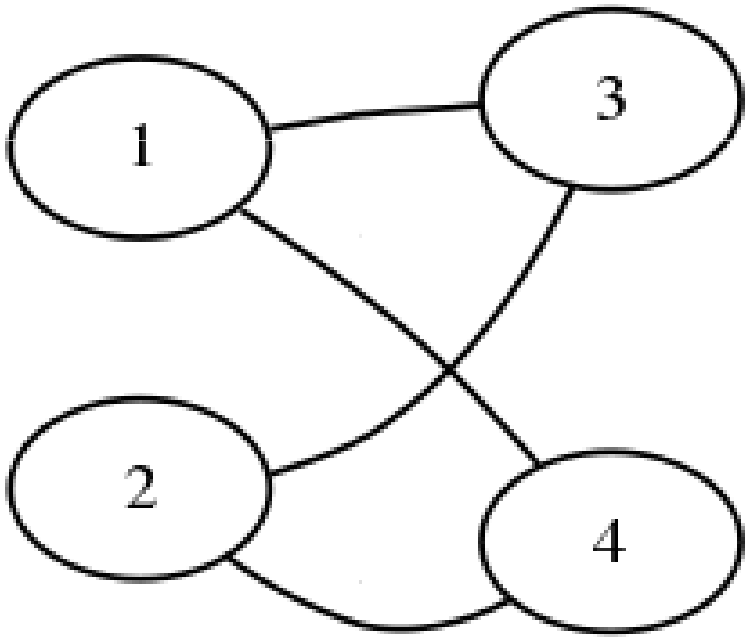
Terminologia

- Due vertici u e v sono **adiacenti** (o vicini) se formano un arco.
- Se $e=(u,v)\in E$ diciamo che e è **incidente** nei nodi u e v.
- Il **grado** di un vertice v è il numero di archi che incidono nel nodo.
- Un nodo si dice **isolato** se il suo grado è zero.

Terminologia

- Un **cammino** è una sequenza di nodi in cui ogni coppia di nodi consecutivi è un arco.
- La **lunghezza di un cammino** è uguale al numero di archi che lo compongono: un cammino di k vertici ha lunghezza $k-1$.
- Un vertice v è **raggiungibile** da u se esiste un cammino da u a v.
- Un **ciclo** è un cammino in cui il primo e l'ultimo nodo coincidono.

Esempio



V = {1, 2, 3, 4}
E = {(1,3), (1,4), (2,3), (2,4)}

- 1 e 3 sono adiacenti (vicini)
- 1 e 2 non sono adiacenti

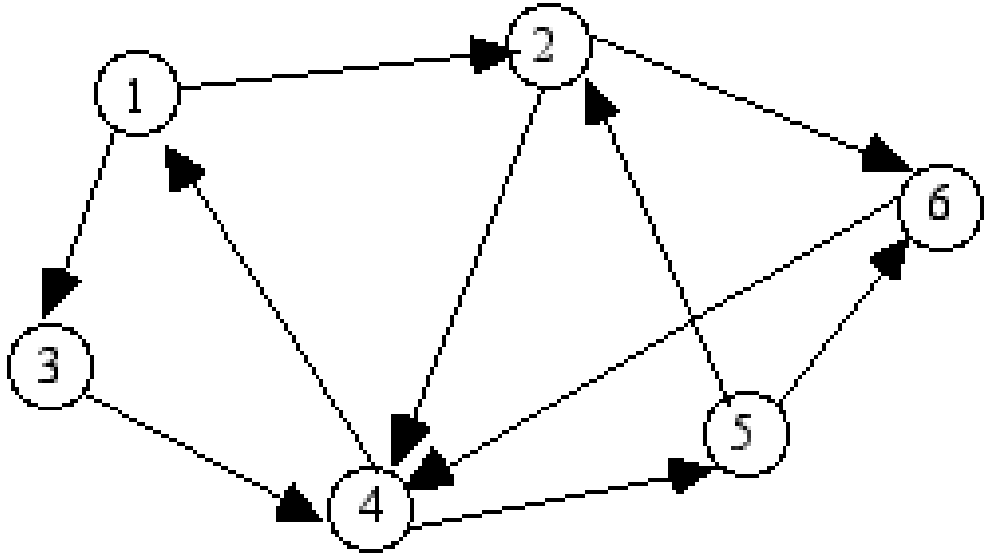
- Cammino: 1,3,2,4 (lungo 3)
- Ciclo: 3,2,4,1,3

Terminologia

- Un grafo è **completo** se ha il massimo numero di archi possibile: un grafo completo di n nodi ha $n(n-1)/2$ archi
- Un grafo è **connesso** se per ogni coppia di nodi distinti esiste un cammino che li connette.
- Le **componenti connesse** di un grafo sono le classi di equivalenza dei vertici secondo la relazione “è raggiungibile da”.

Grafo orientato

- Un grafo orientato o diretto G=(V,E) (digrafo) è una raccolta di vertici (o nodi) v∈V e archi <u,v> ∈E=VxV (relazione non simmetrica), in cui gli archi sono **coppie ordinate di nodi.**
- Rappresentiamo graficamente gli archi con frecce che vanno dal primo al secondo nodo



Terminologia

In un digrafo:

- Se e=<u,v> è un arco, diciamo che u è **adiacente** a v (il viceversa non è implicato).
- Se e=<u,v> è un arco, diciamo che e **esce** da u ed **entra** in v.
- Il **grado uscente** di un vertice v è il numero di archi che escono da v; Il **grado entrante** di v è il numero di archi che entrano in v; Il **grado** di v è la somma del grado entrante e del grado uscente.
- un **cammino** segue la direzione delle frecce.
  - Esempio: 1,3,4,5

Connessione

- Un digrafo è (**fortemente**) **connesso** se, presi comunque due nodi u e v, esiste almeno un cammino da u a v e viceversa
- Nel caso di grafi non orientati il "viceversa" è scontato…
- Le **componenti fortemente connesse** di un digrafo sono le classi di equivalenza dei vertici secondo la relazione "sono mutuamente raggiungibili"
- Un digrafo è **debolmente connesso** se il grafo non orientato ottenuto ignorando l'orientamento degli archi è connesso.

Alberi

- Un **albero non orientato** è un grafo non orientato, connesso e aciclico in cui un nodo viene designato come radice
- Un **albero** (orientato o generico) è un grafo orientato che è vuoto o ha un nodo radice t.c.
  - Non ci sono archi entranti in radice
  - Ogni nodo non radice ha esattamente un arco entrante
  - Per ogni nodo non radice esiste un cammino che va dalla radice al nodo stesso
- Un albero binario non è semplicemente un albero orientato in cui ogni nodo ha al più due figli ! Perché?

Reti

- Un **grafo etichettato** (orientato e non) è un grafo in cui ad ogni arco è associata un'informazione aggiuntiva detta etichetta
- Una **rete** è un grafo etichettato sugli archi con numeri non negativi detti **pesi.**
- Dato un cammino in una rete, il **peso totale del cammino** è la somma dei pesi degli archi nel cammino.

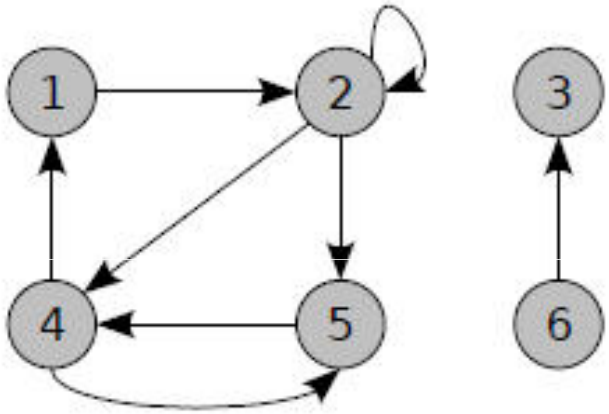
Applicazione:

- Mappa con strade (anche a senso unico) etichettate dalle distanze tra le città

Rappresentazione di grafi

- Nella memoria di una macchina un grafo G=(V,E) può essere rappresentato mediante:
- Liste di adiacenza
- Matrici di adiacenza
- …

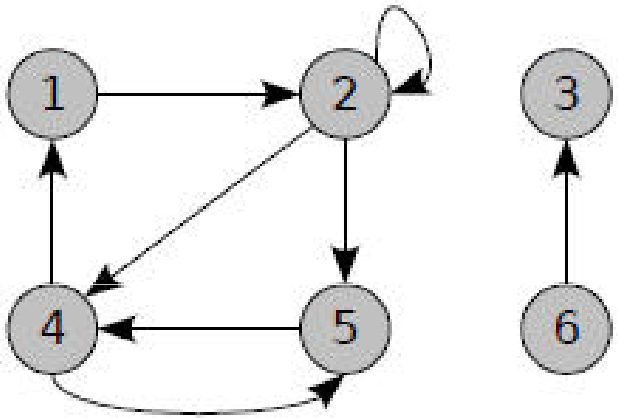
Consideriamo il grafo



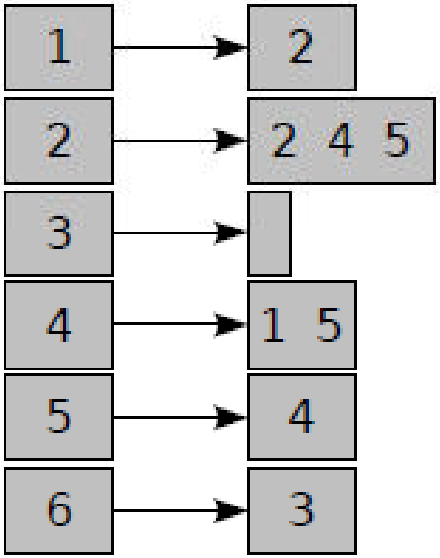
La sua rappresentazione mediante matrice di adiacenza è la seguente

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	0	1	1	0
3	0	0	0	0	0	0
4	1	0	0	0	1	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0

Consideriamo il grafo



La sua rappresentazione mediante liste di adiacenza è la seguente



Matrici di adiacenza

- La matrice di adiacenza di un grafo richiede una memoria pari a $\Theta(|V|^2)$, indipendentemente dal numero di archi del grafo stesso. Possiamo però stabilire se un certo $(u, v) \in E$ in un tempo costante
- È particolarmente adatta per rappresentare **grafi densi**, ossia grafi in cui $|E|$ è prossimo a $|V \times V|$
- Se G è pesato con funzione peso w, il peso $w(u, v)$ dell'arco (u, v) viene memorizzato nella posizione $M_{u,v}$ della matrice di adiacenza

Liste di adiacenza

- Le liste di adiacenza consentono di rappresentare in modo compatto grafi **sparsi**, ossia grafi per cui $|E|$ (il numero di archi) è molto più piccolo di $|V \times V|$
- Se il grafo è orientato, la somma delle lunghezze di tutte le liste di adiacenza è $|E|$, perchè un arco (u, v) viene rappresentato inserendo v in $Adj[u]$ (lista di adiacenza di u)
- Se il grafo è non orientato, la somma delle lunghezze di tutte le liste di adiacenza è $2|E|$, perchè se un arco (u, v) è non orientato, allora v compare nella lista di adiacenza di u e viceversa
- Le liste di adiacenza possono essere adattate per rappresentare **grafi pesati**, cioè grafi per i quali ogni arco ha associato un **peso**. I pesi associati agli archi vengono tipicamente rappresentati mediante una funzione peso $w : E \rightarrow \mathbb{R}$
- Se G è pesato con funzione peso w, il peso $w(u, v)$ dell'arco (u, v) viene memorizzato insieme a v nella lista di adiacenza di u

Le liste di adiacenza

- L'unico svantaggio è che non esiste un modo veloce per stabile se un certo arco $(u, v) \in E$; possiamo solo cercare v nella lista di adiacenza di u (il che richiede un tempo non necessariamente costante)
- Gli algorimi di visita (in ampiezza ed in profondità) di un grafo assumono una rappresentazione del grafo in input mediante liste di adiacenza

Algoritmi per grafi

- Un'operazione che costituisce un prerequisito per altri algoritmi è la possibilità di effettuare la scansione di un grafo:
- Visita "in ampiezza" (breadth-first iterator)
- Visita "in profondità" (depth-first iterator)

Visita in ampiezza

Breadth-First Iteration
(also known as Breadth-First Search):

Dato un **vertice iniziale:**

- Visita per primo il vertice iniziale.
- Marca tutti i vicini (non marcati) del vertice visitato come "**raggiungibili**"
- Visita i vertici raggiungibili nell'ordine in cui sono stati marcati come tali.
- Continua finché ci sono vertici raggiungibili ancora da visitare.

BreadthFirstIterator()

- Dopo che un nodo è stato contrassegnato come raggiungibile, prima o poi sarà visitato
- Memorizziamo i vertici contrassegnati come raggiungibili in una raccolta
- Dato che vogliamo visitare (estrarre) i vicini del nodo attualmente esaminato nell'ordine in cui i vicini sono stati inseriti nella raccolta, usiamo una coda.
- Vogliamo tenere traccia del nodo attualmente esaminato.

```
public BreadthFirstIterator (Vertex start)
{
  for every vertex in the network:
     mark that vertex as not reachable.
  mark start as reachable.
  queue.add (start);
} // algorithm for constructor
```

```
public boolean hasNext( )
{
  return !queue.isEmpty( );
} // algorithm for hasNext
```

```
public Vertex next( )
{
 current = queue.remove();
 for each vertex that is a neighbor of
  current:
 if that vertex has not yet been marked as
  reachable
   {
      mark that vertex as reachable;
      add that vertex to queue;
   } // if
return current;
} // algorithm for method next
```

DepthFirstIterator()

- Un iteratore che procede in profondità è una generalizzazione dell'attraversamento in pre-ordine negli alberi binari
- Un scansione in profondità di un grafo procede in modo analogo ad una scansione in ampiezza, ma il nodo successivo da visitare è quello dichiarato **raggiungibile più recentemente**
- Usiamo dunque uno stack

```
public DepthFirstIterator (Vertex start)
{
  for every vertex in the network:
     mark that vertex as not reachable.
  mark start as reachable.
  stack.push (start);
} // algorithm for constructor
```

```
public Vertex next( )
{
  current = stack.pop();
  for each vertex that is a neighbor of
  current:
      if that vertex has not yet been marked
      as reachable
      {
            mark that vertex as reachable;
            stack.push (vertex);
       } // if
  return current;
} // algorithm for method next
```