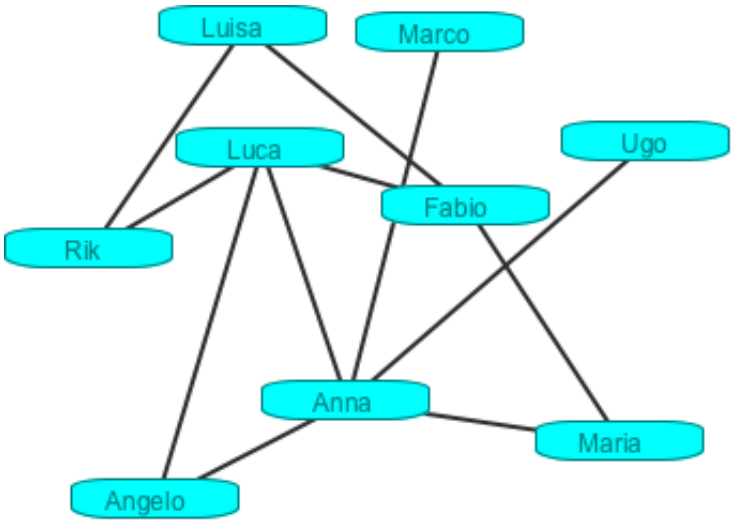
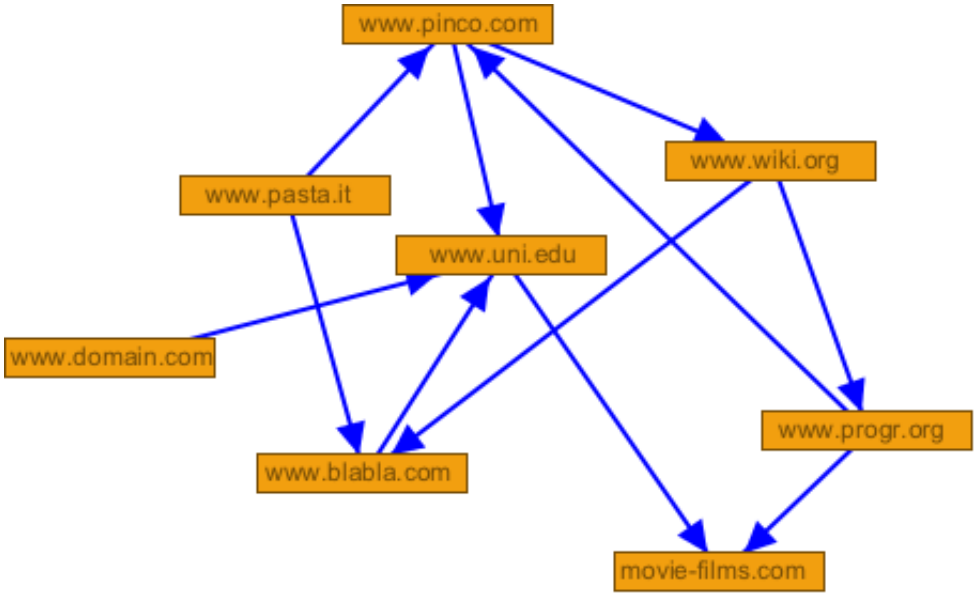
Progettazione di algoritmi

Grafi

Un *grafo* è una collezione di elementi con una relazione binaria tra di essi. Le situazioni che possono essere modellate dai grafi sono innumerevoli e provengono dai campi più disparati: fisica, biologia, informatica, sistemi informativi e sociali, ecc. Gli elementi di un grafo sono detti **nodi** o **vertici** (*nodes* o *vertices* nella terminologia inglese). Le relazioni tra i nodi possono essere simmetriche, come ad es. nel caso della relazione di amicizia tra persone:



Oppure possono essere asimmetriche, come nel caso dei link tra le pagine del Web:



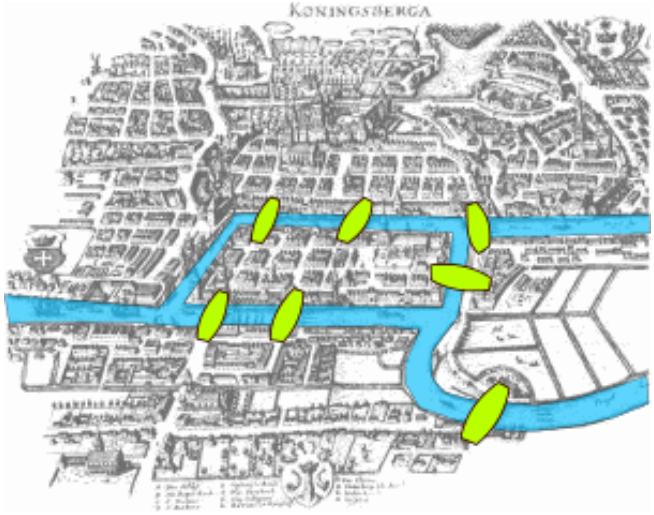
La relazione tra due nodi *u* e *v* si chiama **arco** (in inglese *edge*). Se la relazione tra *u* e *v* è simmetrica l'arco è detto *non diretto* ed è denotato con {*u*, *v*}, cioè, come un insieme di due

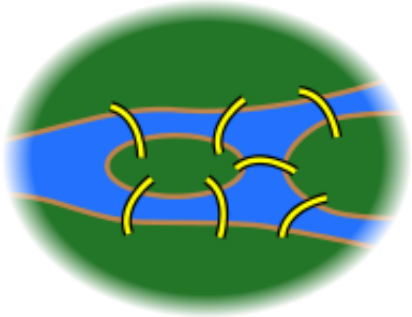
elementi. Se invece la relazione è asimmetrica, l'arco è detto *diretto* o *orientato* ed è denotato con (*u*, *v*), cioè come una coppia di elementi, e il verso dell'arco, in questo caso, è da *u* a *v*. Un grafo con solamente archi non diretti si dice *grafo non diretto* (o *non orientato*) e un grafo con archi diretti si dice *grafo diretto*.

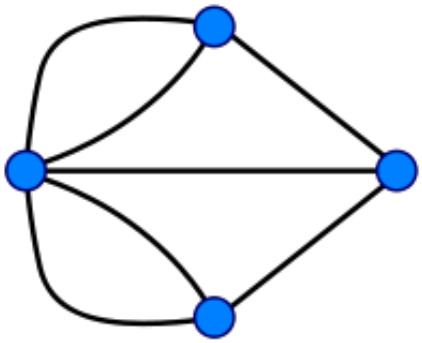
Se c'è un arco tra *u* e *v* si dice che sono **adiacenti** o **direttamente connessi** (*adjacent* o *directly connected*). Un arco che ha come uno dei suoi estremi un nodo *u* si dice **incidente** in *u* (*incident*). Il numero di tutti gli archi indicenti in un nodo *u* è detto il **grado** di *u* (*degree*). Nel caso di grafi diretti il numero di archi uscenti da un nodo *u* è detto il **grado uscente** di *u* (*outdegree*) e il numero di quelli entranti è il **grado entrante** di *u* (*indegree*).

Le origini

Come molto spesso accade, una nuova teoria nasce dal tentativo di risolvere un nuovo tipo di problema. Il problema dei sette ponti di Königsberg: è possibile attraversare tutti e sette i ponti passando una sola volta su ogni ponte?





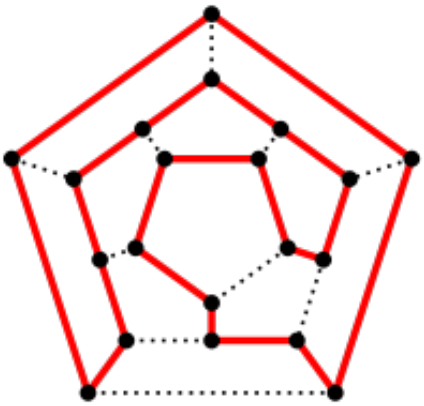


Il grande matematico *Leonhard Euler* risolvendo questo problema nel 1735 pose le basi della teoria dei grafi.

La soluzione al problema è negativa. Un tale attraversamento non esiste perché un nodo che non è né l'inizio né la fine della passeggiata deve avere grado pari (se si entra nel nodo da un ponte si deve uscire da un altro ponte) ma tutti e quattro i nodi hanno grado dispari.

Per un grafo qualsiasi il problema chiede se esiste una sequenza di nodi adiacenti che attraversa tutti gli archi del grafo senza mai passare due volte per lo stesso arco. Una sequenza di nodi adiacenti è detta **cammino** (*path* o *walk*). Il problema generale si chiama problema del *cammino Euleriano*. Nel caso si richieda che il cammino parta e finisca nello stesso nodo si chiama problema del *circuito Euleriano*. Un cammino che inizia e finisce nello stesso nodo è detto **circuito** o **ciclo** (*circuit* o *cycle*). Se il cammino o il ciclo non passano mai due volte per lo stesso nodo sono detti **cammino semplice** e **ciclo semplice**.

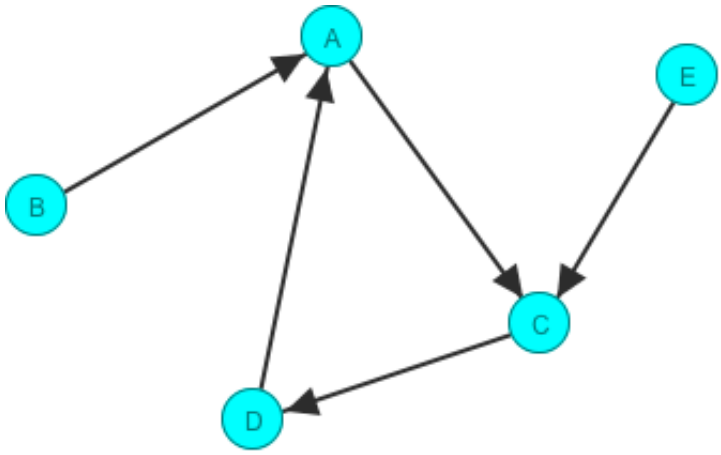
Esistono algoritmi efficienti per trovare cammini o circuiti Euleriani (se esistono). Ma questo non è vero per un problema apparentemente molto simile. Dato un grafo esiste un cammino che passa per tutti vertici senza mai passare due volte per lo stesso vertice? Questo problema fu proposto come rompicapo dal fisico-matematico *William Rowan Hamilton* a metà dell'ottocento relativamente al grafo dei vertici e spigoli del dodecaedro:



Il problema generale si chiama problema del *cammino Hamiltoniano* (o del *circuito Hamiltoniano*, se si chiede di trovare un ciclo) ed è un problema molto più difficile di quello relativo ai cammini Euleriani.

Applicazioni

Le applicazioni dei grafi sono tantissime. In informatica possono essere usati per modellare reti di computer, organizzazione dei dati, flussi di computazioni, ecc. Ad esempio, in un sistema operativo o in un DBMS più processi devono condividere delle risorse (se sono transazioni di un DBMS, devono condividere la scrittura di una base di dati). In ogni istante certi processi detengono l'uso esclusivo di certe risorse mentre altri sono in attesa di accedere a tali risorse. La situazione può essere rappresentata da un grafo del tipo:



dove un arco diretto (*B*, *A*) significa che il processo *B* è in attesa di accedere ad una risorsa attualmente detenuta da *A*. In queste situazioni, la presenza di cicli diretti indica la presenza di stalli o deadlock.

I grafi sono usati in moltissime altre aree. In chimica e fisica sono usati per studiare la struttura delle molecole. In biologia e etologia per studiare i flussi migratori degli animali. In linguistica computazionale per rappresentare i significati delle parole tramite reti semantiche. In sociologia e informatica, per lo studio delle cosidette reti sociali.

Rappresentazioni dei grafi

Un grafo generalmente viene denotato come una coppia $G = (V, E)$, dove V è l'insieme dei nodi ed E è l'insieme degli archi. Di solito n denota il numero dei nodi e m il numero degli archi. In linea di principio, ci sono moltissimi modi per rappresentare un grafo nella memoria di un computer ma nella pratica sono solamente due i metodi più usati. Il primo si chiama **matrice di adiacenza** e usa una matrice nxn M tale che

$M[u][v]$ = *true* se l'arco (u, v) (o l'arco $\{u, v\}$) appartiene a E, altrimenti è *false*

Chiaramente se il grafo non è diretto la M è una matrice simmetrica (cioè, $M[u][v] = M[v][u]$). Questa rappresentazione è molto efficiente nel determinare se un arco è presente o meno. Però risulta essere molto dispendiosa in termini di memoria se il grafo è sparso, cioè, ha relativamente pochi archi. Infatti, lo spazio richiesto è $O(n^2)$, indipendente dal numero degli archi. Inoltre, se il grafo è sparso scandire tutti gli adiacenti di un dato nodo diventa un'operazione costosa perché richiede $O(n)$ tempo per scorrere una intera riga o colonna della matrice, anche se il grado del nodo è molto più piccolo di n.

Il secondo metodo si chiama **liste di adiacenza** e mantiene per ogni nodo u la lista dei suoi adiacenti. Ad esempio, nel caso del primo grafo relativo alle amicizie la rappresentazione tramite liste di adiacenza sarebbe:

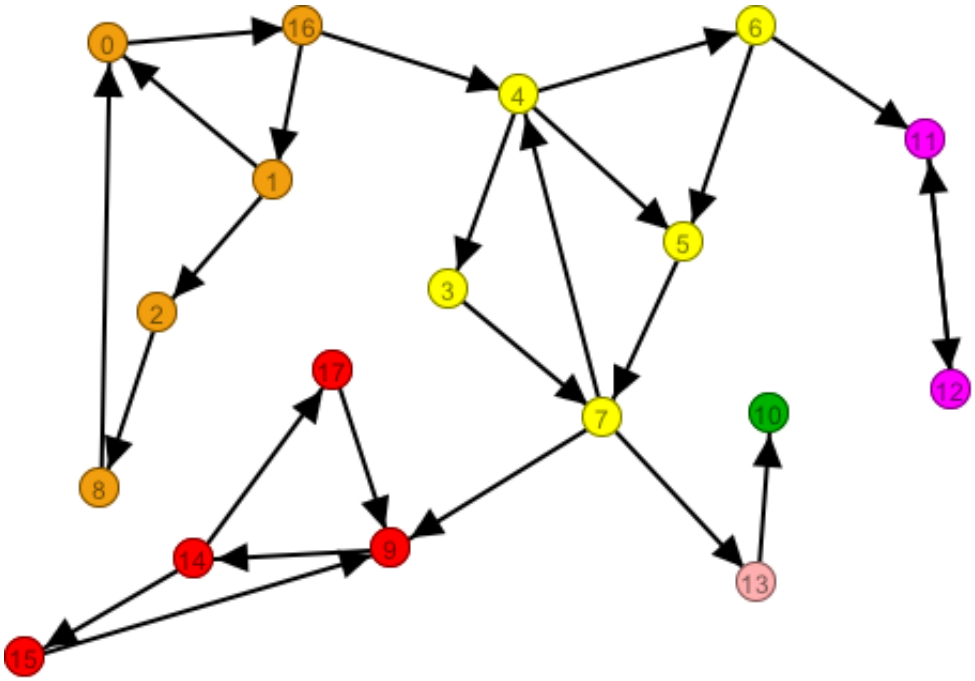
```
Luisa  -> Rik, Fabio
Marco  -> Anna
Luca   -> Rik, Fabio, Angelo, Anna
Ugo    -> Anna
Rik    -> Luisa, Luca
Fabio  -> Luisa, Luca, Maria
Anna   -> Luca, Marco, Ugo, Angelo, Maria
Angelo -> Luca, Anna
Maria  -> Anna, Fabio
```

Il costo in termini di spazio di memoria è O($n + m$) e quindi per grafi sparsi e preferibile alla rappresentazione tramite matrice di adiacenza. In generale, non permette di effettuare il test di esistenza di un arco in tempo costante O(1) ma permette di scandire la lista degli adiacenti di un nodo u in tempo O(d), dove d è il grado di u. Come vedremo questa è l'operazione più frequente in molti algoritmi sui grafi. Pertanto, si può tranquillamente dire che le liste di adiacenza sono il metodo di rappresentazione più usato.

Connettività

Una delle proprietà fondamentali della struttura di un grafo è la connettività: dati due nodi u e v sono connessi? Cioè, esiste un cammino che li connette. Di solito la relazione di connessione è pensata in modo simmetrico. Questo ovviamente è immediato nel caso di grafi non diretti: se esiste un cammino da u a v, il cammino rovesciato è anche un cammino da v a u. Un grafo è detto **connesso** (*connected*) se comunque presi due nodi u e v esiste un cammino che li connette. Ma per grafi diretti la situazione è un po' più complicata. Un cammino orientato da u a v non necessariamente implica l'esistenza di un cammino orientato da v a u. Per questo nel caso di grafi diretti si parla di **connessione forte**. Un grafo diretto è **fortemente connesso** (*strongly connected*) se per ogni coppia di nodi u e v esiste sia un cammino orientato da u a v che un cammino orientato da v a u.

Se un grafo non è connesso, è partizionato in parti che sono connesse. In un grafo non diretto G una **componente connessa** (*connected component*) è un sottografo di G tale che è connesso e che non è estendibile senza distruggere la connessione. Questo significa che se G non è connesso, è partizionato in due o più componenti connesse e non ci sono archi che collegano nodi appartenente a componenti connesse differenti. In un grafo diretto G una **componente fortemente connessa** (*strongly connected component*) è un sottografo di G che è fortemente connesso e che non è estendibile senza distruggere la connessione forte. Quindi se G non è fortemente connesso, è partizionato in due o più componenti fortemente connesse e gli archi tra queste non possono formare cicli, cioè non esistono cicli che contengono nodi appartenenti a componenti differenti. Ad esempio, nel grafo qui sotto ci sono 6 componenti fortemente connesse evidenziate dai colori dei nodi:



Come vedremo per determinare la connessione di un grafo ed eventualmente le sue componenti (fortemente) connesse si può fare una visita del grafo, ovvero una esplorazione di tutti i nodi e gli archi in un certo ordine.

Esempio

Prima di passare a vedere i metodi di vista di un grafo consideriamo un piccolo problema. Abbiamo un puzzle meccanico formato da 9 tessere, numerate da 1 a 9, incasellate in un quadrato 3x3. La configurazione iniziale è la seguente:

```
-------------
| 1 | 2 | 3 |
|---|---|---|
| 4 | 5 | 6 |
|---|---|---|
| 7 | 8 | 9 |
-------------
```

La meccanica del puzzle permette di fare solamente due tipi di mosse:

- shift di una posizione a destra di una delle tre righe. Ad esempio, se alla configurazione iniziale applichiamo tale mossa alla seconda riga otteniamo:

```
-------------
| 1 | 2 | 3 |
|---|---|---|
| 6 | 4 | 5 |
|---|---|---|
| 7 | 8 | 9 |
-------------
```

- shift di una posizione in alto di una delle tre colonne. Ad esempio, se alla configurazione iniziale applichiamo tale mossa alla prima colonna otteniamo:

```
-------------
| 4 | 2 | 3 |
|---|---|---|
| 7 | 5 | 6 |
|---|---|---|
| 1 | 8 | 9 |
-------------
```

Data una qualsiasi configurazione mescolata vogliamo trovare una sequenza di mosse che la riporta nella configurazione iniziale. Ad esempio, qual'è una sequenza per la seguente configurazione?

```
-------------
| 7 | 3 | 9 |
|---|---|---|
| 2 | 5 | 1 |
|---|---|---|
| 4 | 8 | 6 |
-------------
```

Come possiamo risolvere questo problema in generale per una qualsiasi data configurazione? Si tenga presente che alcune configurazioni potrebbero non essere valide, nel senso che non permettono di raggiungere la configurazione iniziale.

Esercizio [Acqua]

Ci sono tre contenitori di capienza 10, 7 e 4 litri, rispettivamente. Inizialmente, quelli da 4 e 7 litri sono pieni d'acqua e quello da 10 è vuoto. Possiamo fare un solo tipo di operazione: versare l'acqua da un contenitore in un altro fermandoci solo quando o il contenitore sorgente è vuoto o quello destinazione è pieno. C'è una sequenza di operazioni di versamento che termina lasciando esattamente 2 litri o nel contenitore da 7 o in quello da 4? Cosa c'entra questo problema con i grafi?