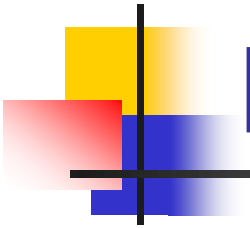
L'ADT grafo

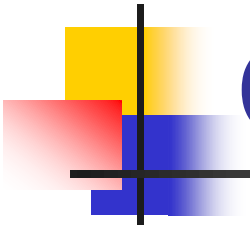
Gianpiero Cabodi e Paolo Camurati

Dip. Automatica e Informatica
Politecnico di Torino



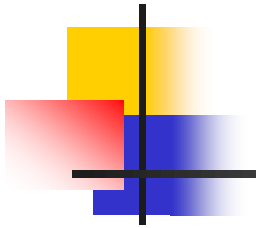
Perché la Teoria dei Grafi?

- Moltissime applicazioni pratiche
- Centinaia di algoritmi
- Astrazione interessante e utilizzabile in molti domini diversi
- Ricerca attiva in Informatica e Matematica discreta.



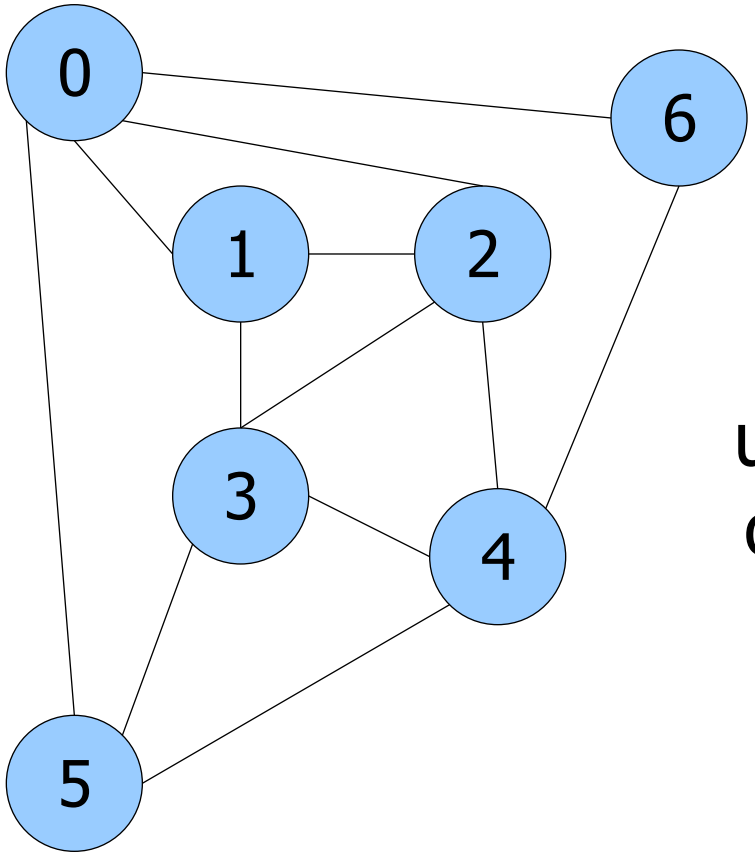
Complessità dei problemi

- problemi facili:
  - determinare se un grafo è connesso
  - determinare la presenza di un ciclo
  - individuare le componenti fortemente connesse
  - individuare gli alberi minimi ricoprenti
  - calcolare i cammini minimi
  - determinare se un grafo è bipartito
  - trovare un cammino di Eulero
- problemi trattabili:
  - planarità di un grafo
  - matching

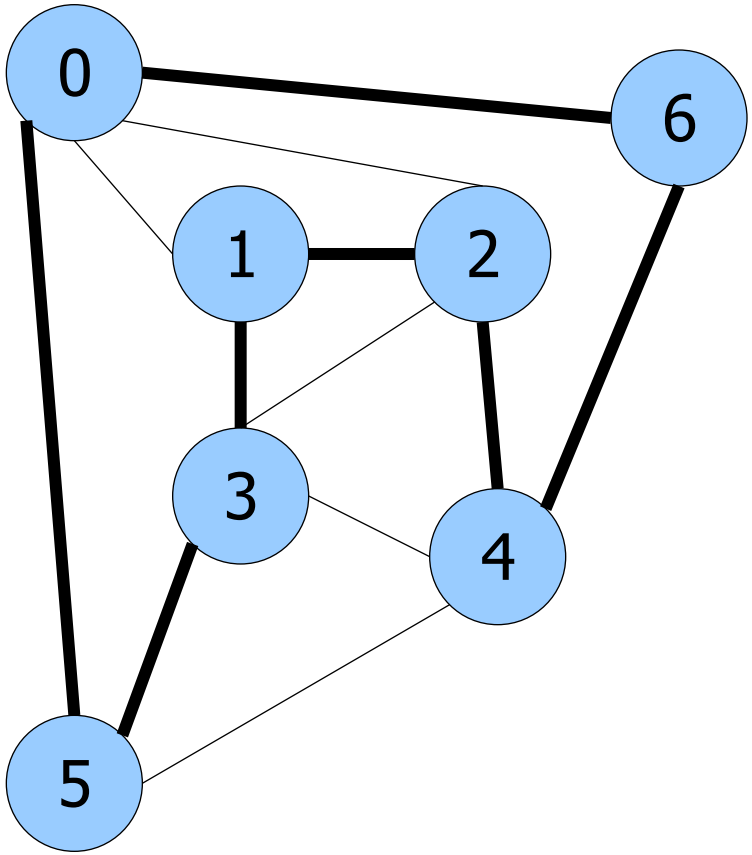


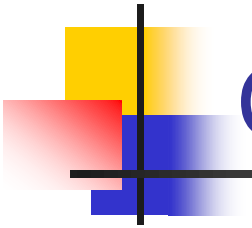
- problemi intrattabili:
  - cammini a lunghezza massima
  - colorabilità
  - clique massimale
  - ciclo di Hamilton
  - problema del commesso viaggiatore
- problemi ignoti:
  - isomorfismo di due grafi

Ciclo di Hamilton



Dato un grafo non orientato G =(V,E), esiste un ciclo semplice che visita ogni vertice una e una sola volta?


0
6
1
2
3
4
5



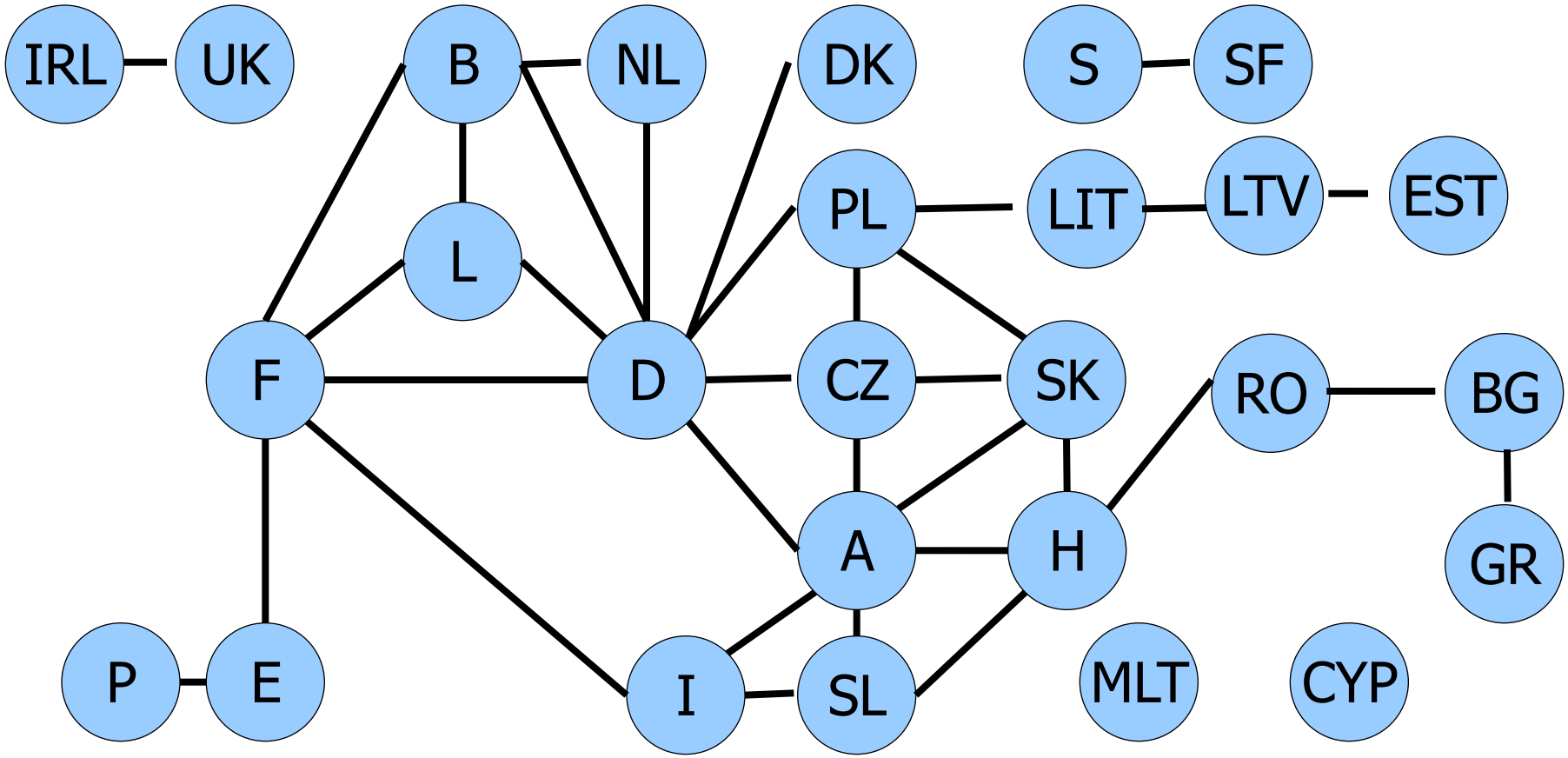
Colorabilità

Dato un grafo non orientato G =(V,E), quale è il minimo numero di colori k necessario affinché nessun vertice abbia lo stesso colore di un vertice ad esso adiacente?

Si dice «planare» un grafo che, se disegnato su di un piano, non ha archi che si intersecano.

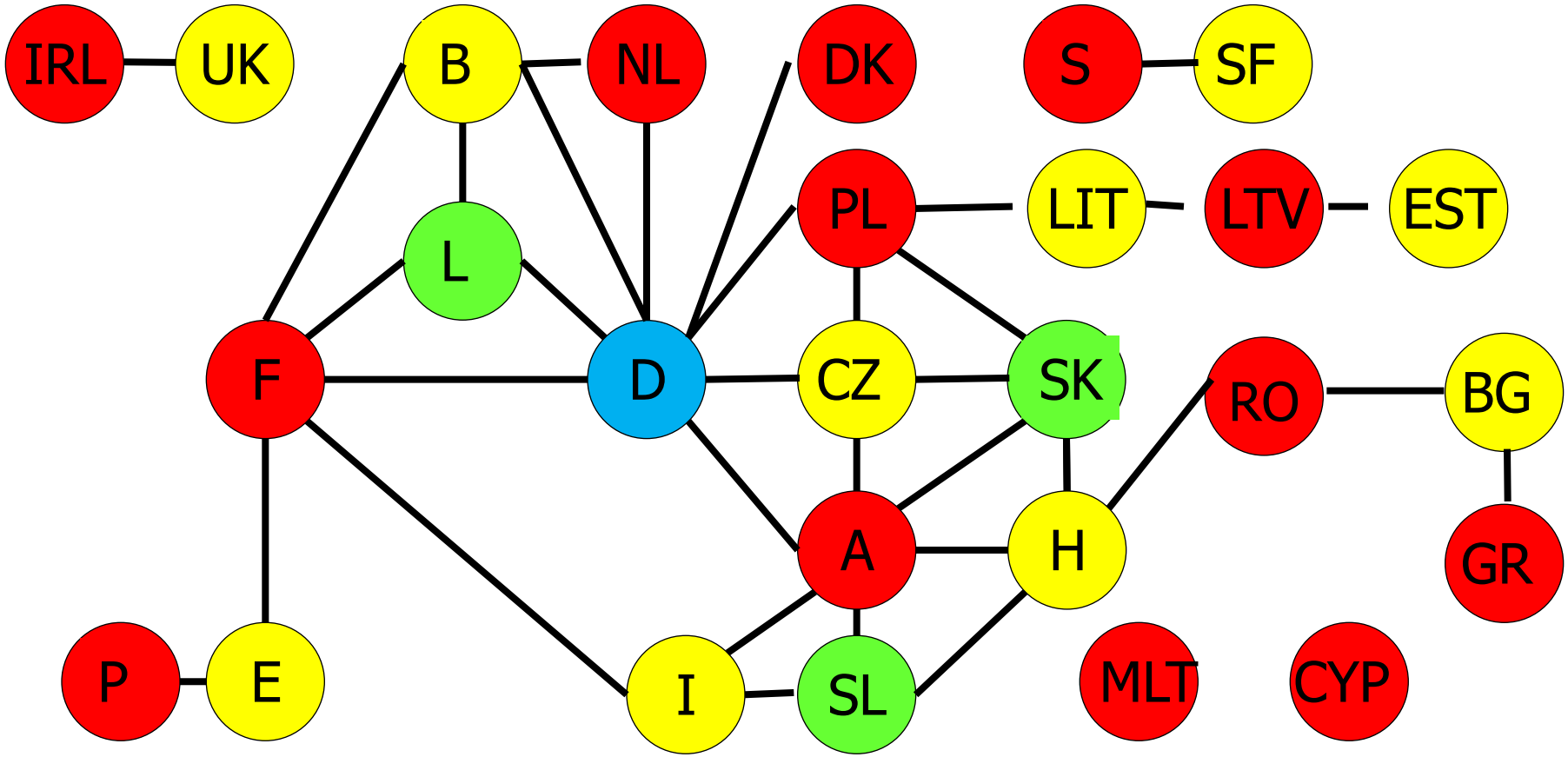
Le mappe geografiche sono modellate come grafi planari.

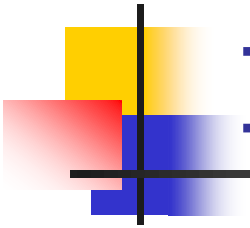
L'UE a 27 stati



Colorabilità: k=4

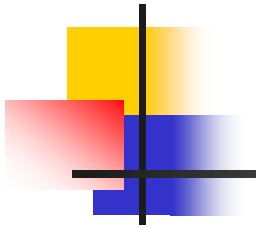
Per i grafi planari si può dimostrare che servono al più 4 colori.



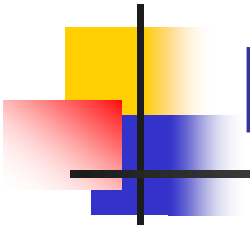


Ipotesi

- Grafi statici: non si aggiungono né si cancellano vertici, si possono aggiungere o cancellare archi
- Numero di vertici V noto, calcolabile o sovrastimabile
  - compare come intero sulla prima riga del file da cui si legge
  - si calcola leggendo il file avvalendosi di una tabella di simboli per identificare i vertici distinti
  - si sovrastima come 2 volte il numero di archi letti da file, ipotizzando che ogni arco connetta vertici distinti.



- I vertici sono interi tra 0 e V-1, così da permettere un accesso diretto tramite indice in un vettore
- Se in origine i vertici non sono interi ma stringhe, ci si avvale di una tabella di simboli per metterli in corrispondenza biunivoca con gli interi tra 0 e V-1. La funzione `STindex(char *key)` ritorna l'intero associato con la chiave se già presente, altrimenti inserisce la chiave in una tabella che contiene N simboli e associa N+1 alla chiave.

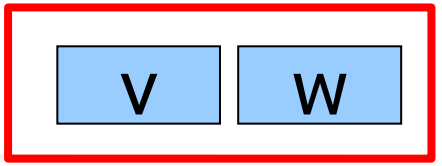


L’ADT grafo

- Interfaccia
- Rappresentazione:
  - matrice delle adiacenze
  - lista delle adiacenze
  - (elenco di archi)

ADT Graph

Edge



Graph.h

```
typedef struct { int v; int w; } Edge;
Edge EDGE(int, int);

typedef struct graph *Graph;

Graph GRAPHinit(int);
void  GRAPHinsertE(Graph G, Edge e);
void  GRAPHremoveE(Graph G, Edge e);
int   GRAPHedges(Graph G, Edge a[]);
void  GRAPHshow(Graph G);
```

Funzione di creazione di un arco da una coppia di interi

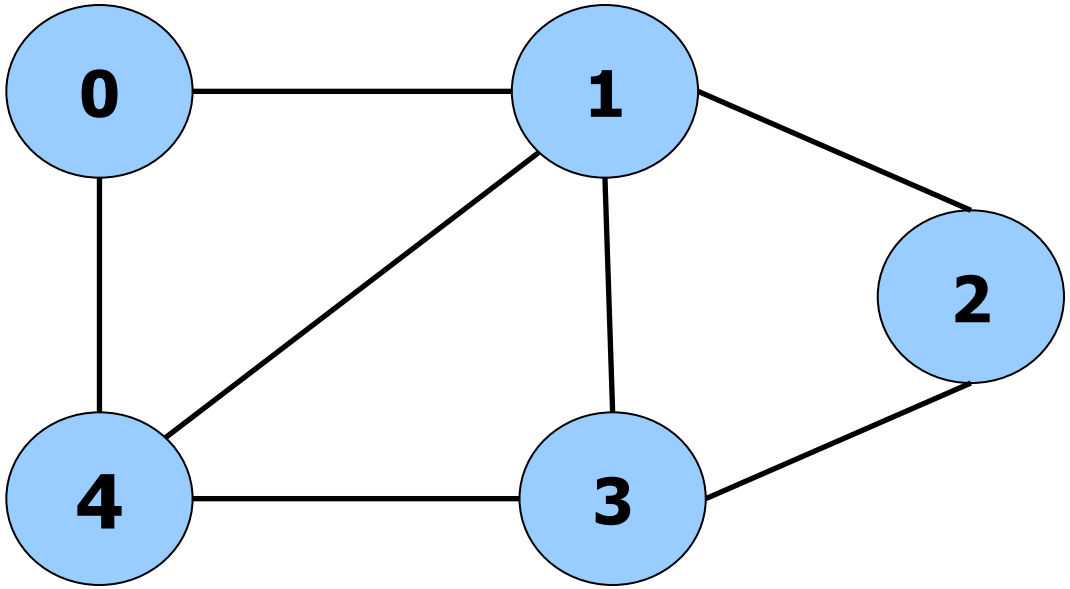
Rappresentazione

Matrice di adiacenza

Dato G = (V, E), la matrice di adiacenza è:

- matrice A di |V| x |V| elementi
- $A[i,j] = \begin{cases} 1 \text{ se } (i, j) \in E \\ 0 \text{ se } (i, j) \notin E \end{cases}$
- grafi non orientati: A simmetrica.

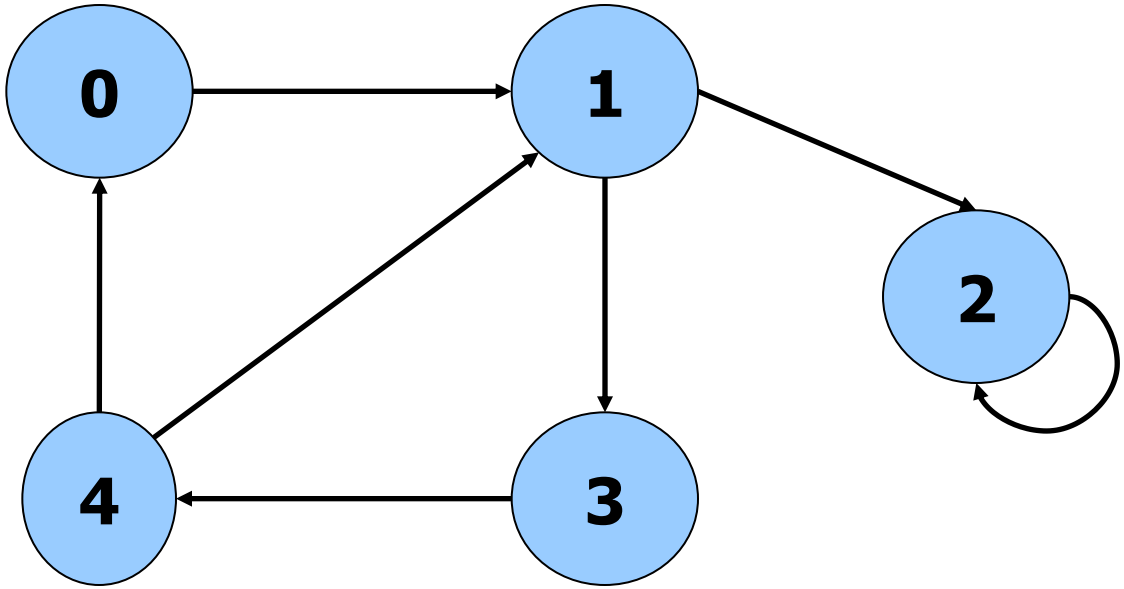
Esempio: grafo non orientato



A

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

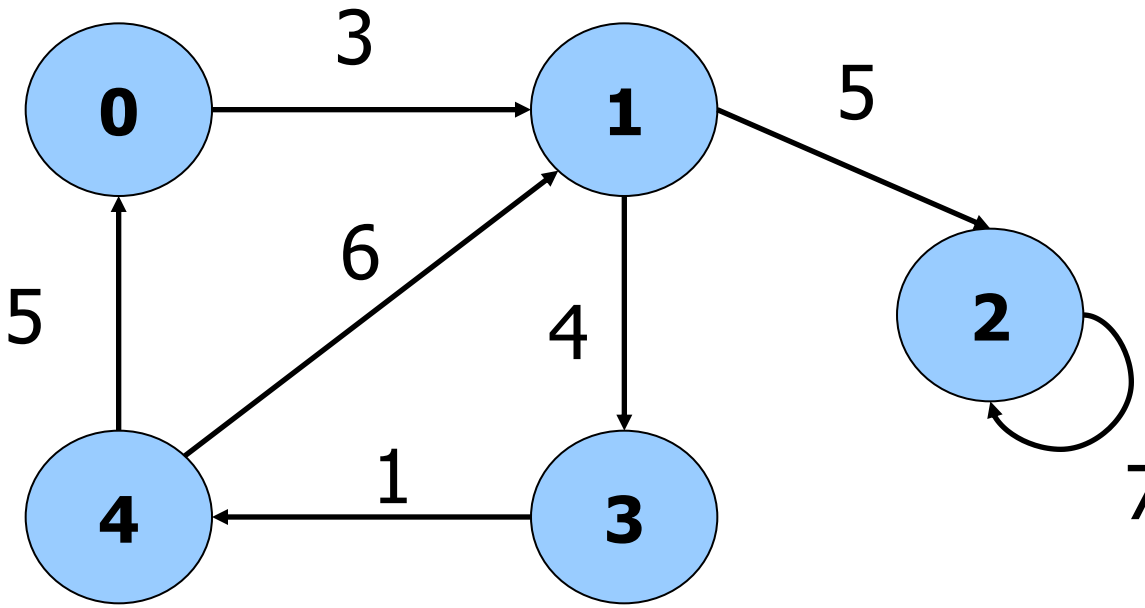
Esempio: grafo orientato



A

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

Esempio: grafo orientato pesato



A

	0	1	2	3	4
0	0	3	0	0	0
1	0	0	5	4	0
2	0	0	7	0	0
3	0	0	0	0	1
4	5	6	0	0	0

Graph.c

```
#include <stdlib.h>
#include <stdio.h>
#include "Graph.h"

struct graph {int V; int E; int **adj; };

Edge EDGE(int v, int w) {
  Edge e;
  e.v = v;
  e.w = w;
  return e;
}
int **MATRIXint(int r, int c, int val) {
  int i, j;
  int **t = malloc(r * sizeof(int *));
  for (i=0; i < r; i++)
    t[i] = malloc(c * sizeof(int));
  for (i=0; i < r; i++)
    for (j=0; j < c; j++)
      t[i][j] = val;
  return t;
}
```

Numero di vertici

Matrice

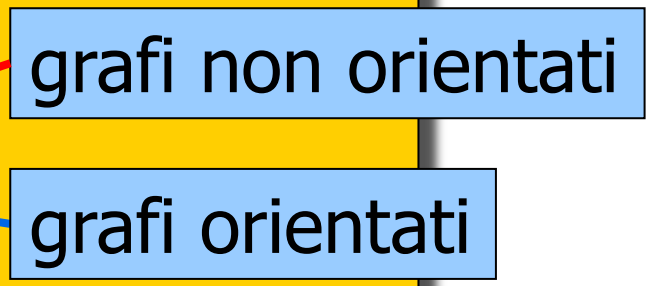
Numero di archi

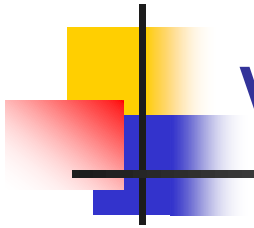
Attenzione: si possono generare cappi!

```
Graph GRAPHinit(int V) {
  Graph G = malloc(sizeof *G);
  G->V = V;
  G->E = 0;
  G->adj = MATRIXint(V, V, 0);
  return G;
}
void  GRAPHinsertE(Graph G, Edge e) {
  int v = e.v, w = e.w;
  if (G->adj[v][w] == 0)
    G->E++;
  G->adj[v][w] = 1;
  G->adj[w][v] = 1;
}
void  GRAPHremoveE(Graph G, Edge e) {
  int v = e.v, w = e.w;
  if (G->adj[v][w] == 1)
    G->E--;
  G->adj[v][w] = 0;
  G->adj[w][v] = 0;
}
```

Attenzione: solo per grafi non orientati

```
int   GRAPHedges(Graph G, Edge a[]) {
  int v, w, E = 0;
  for (v=0; v < G->V; v++)
    for (w=v+1; w < G->V; w++)
    for (w=0; w < G->V; w++)
      if (G->adj[v][w] == 1)
        a[E++] = EDGE(v, w);
  return E;
}

void  GRAPHshow(Graph G) {
  int i, j;
  printf("%d vertices, %d edges \n", G->V, G->E);
  for (i=0; i < G->V; i++) {
    printf("%2d:", i);
    for (j=0; j < G->V; j++)
      printf("%2d", G->adj[i][j]);
    printf("\n");
  }
}
```





Vantaggi/svantaggi

- Complessità spaziale

  $S(n) = \Theta(|V|^2)$

  $\Rightarrow$ vantaggiosa SOLO per grafi densi
- No costi aggiuntivi per i pesi di un grafo pesato
- Accesso efficiente (O(1)) alla topologia del grafo.

Rappresentazione

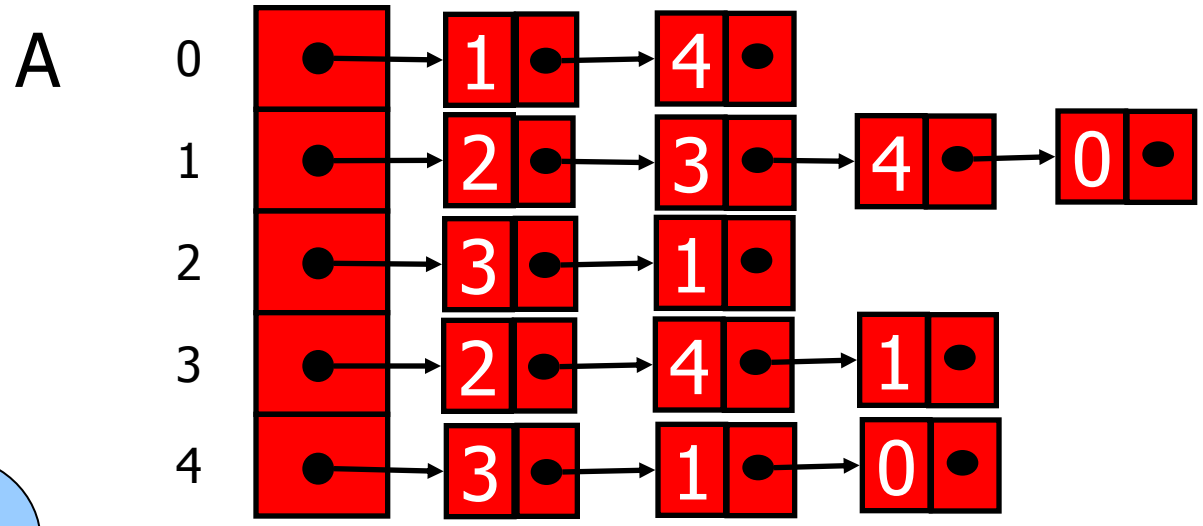
Se conosco il numero di vertici, si può implementare con un vettore di liste, altrimenti lista di liste

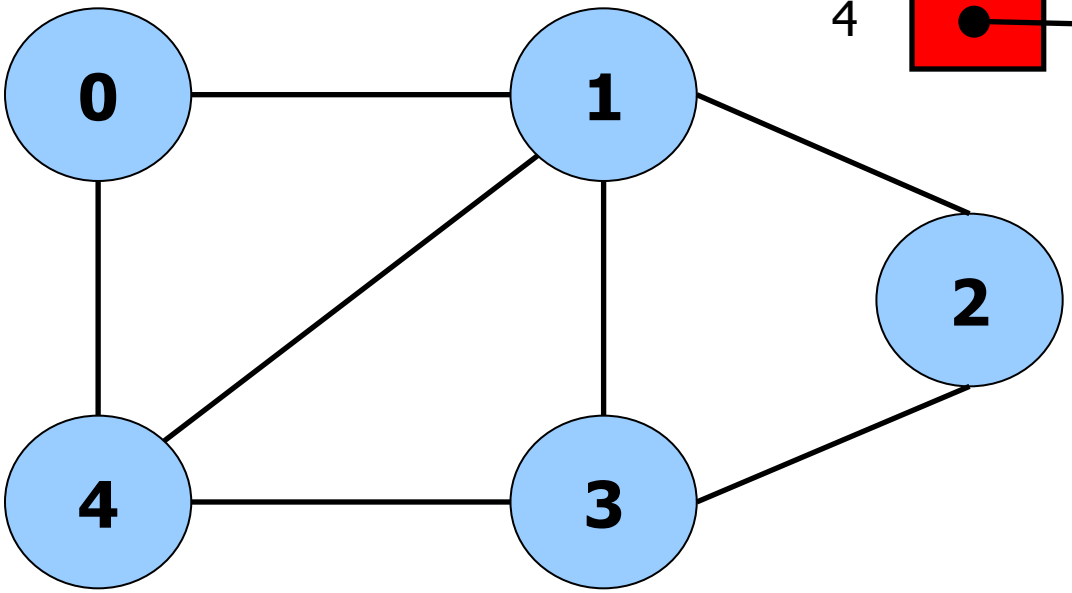
Lista di adiacenza

Dato G = (V, E), la lista di adiacenza è:

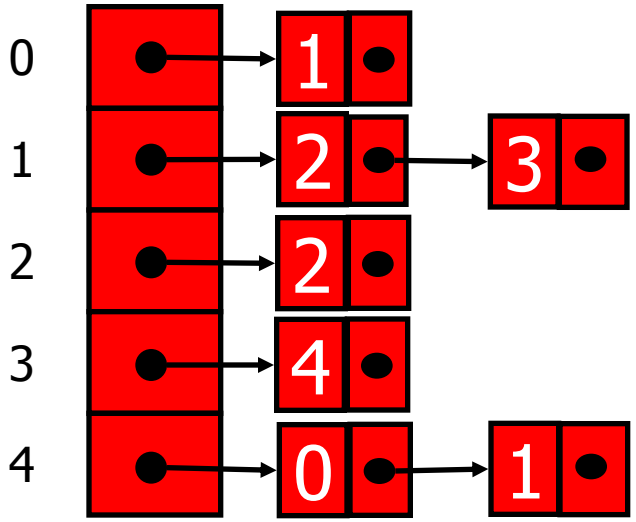
- un vettore A di |V| elementi. Il vettore è vantaggioso per via dell'accesso diretto
- dove A[i] contiene il puntatore alla lista dei vertici adiacenti a i.

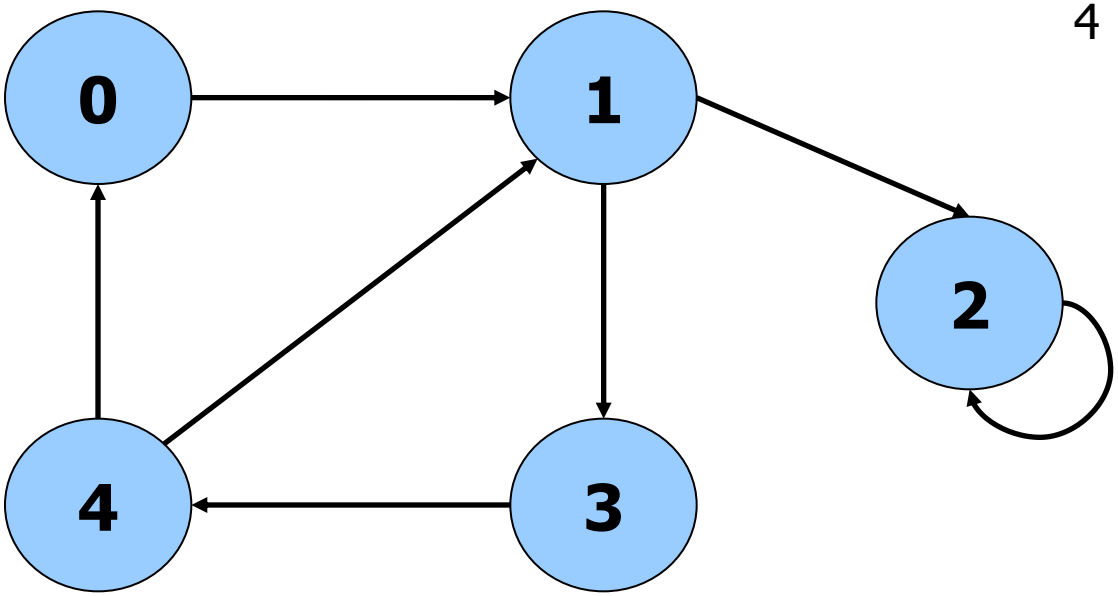
Esempio: grafo non orientato



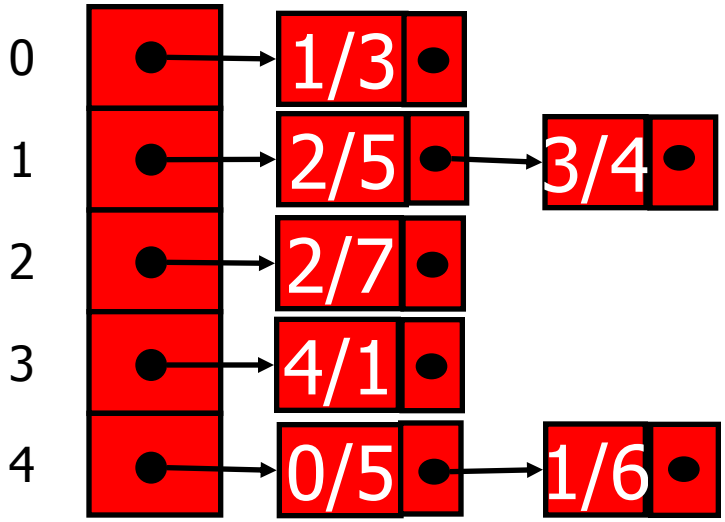


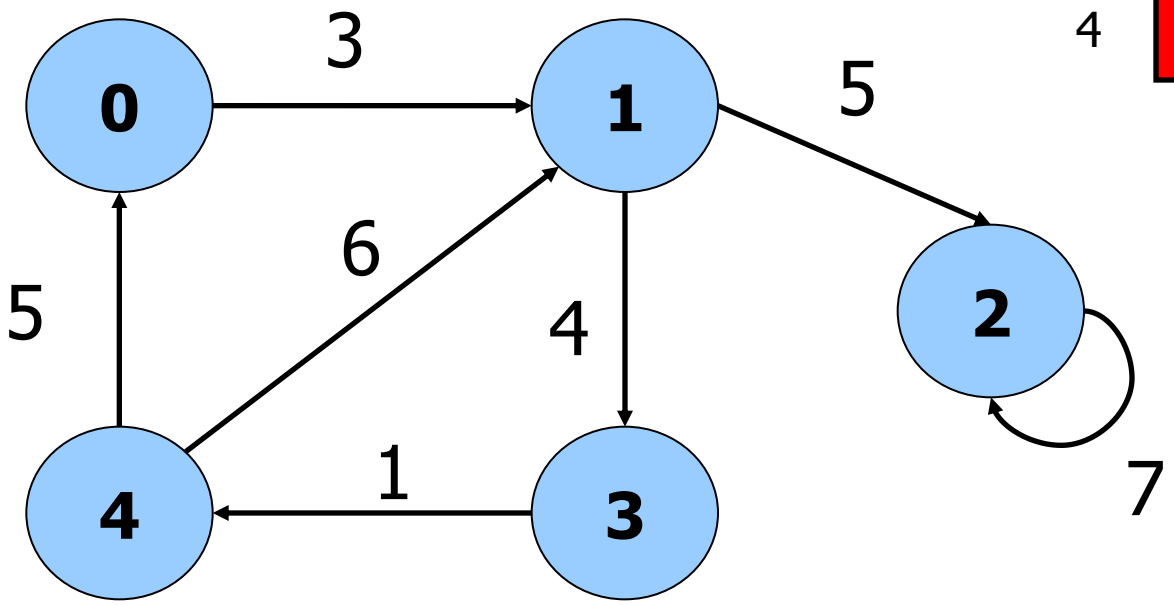
Esempio: grafo orientato





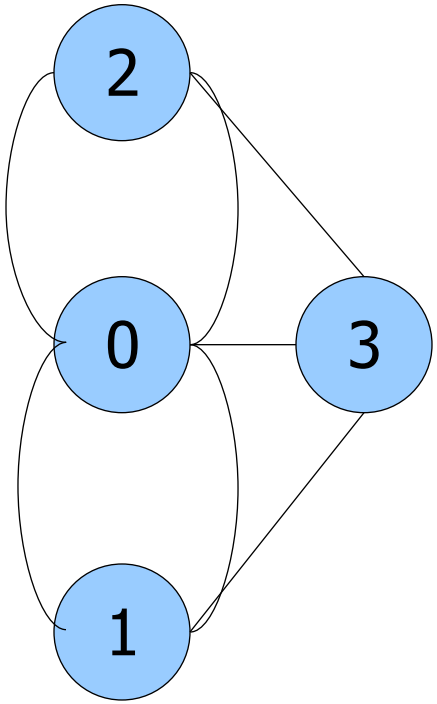
Esempio: grafo orientato pesato

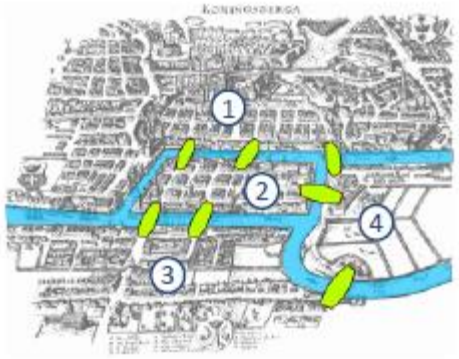


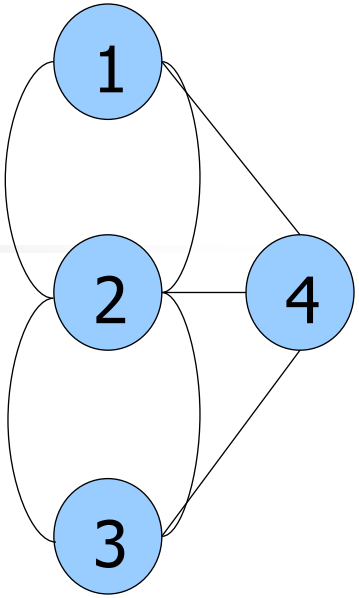


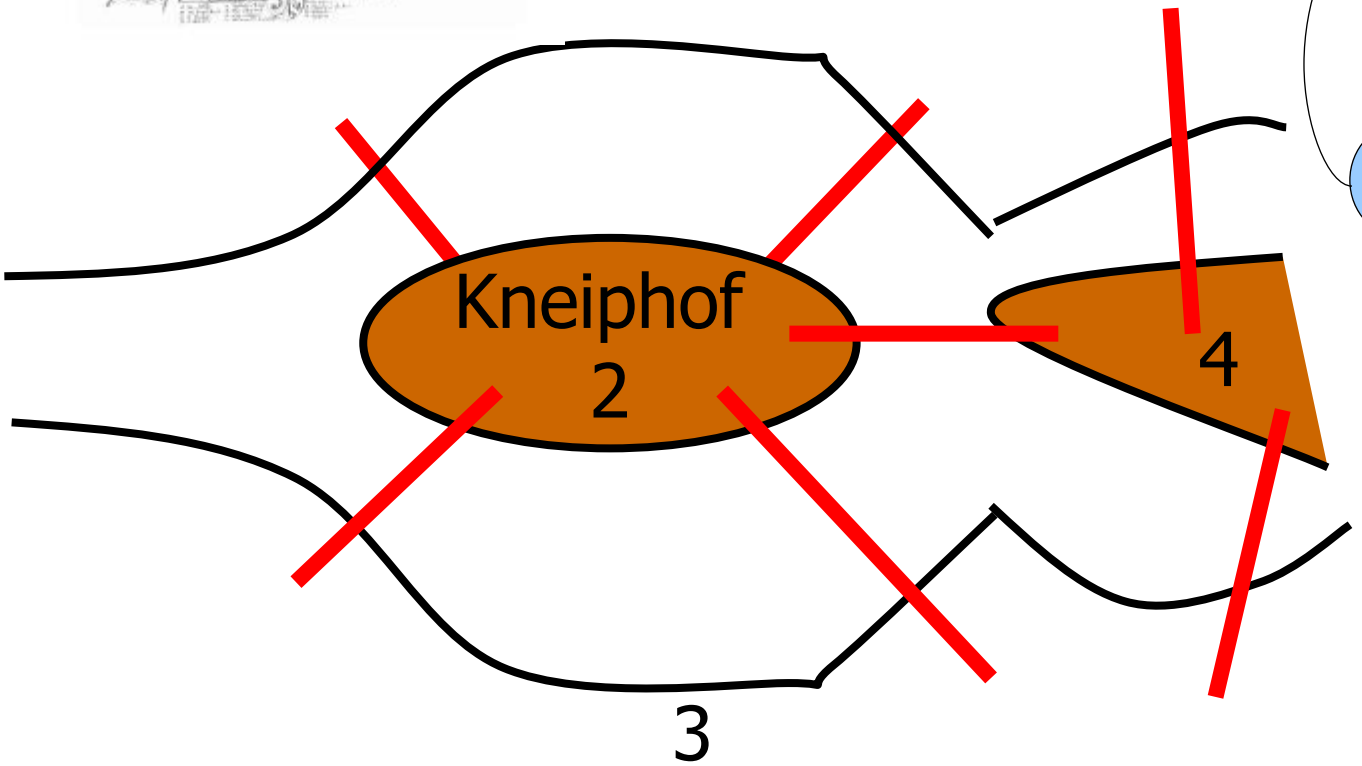
Esempio: il multigrafo

Multigrafo: archi multipli che connettono la stessa coppia di vertici. Si può generare se in fase di inserzione degli archi non si scartano quelli già esistenti.









I ponti di Königsberg
(Eulero, 1736)

Graph.c

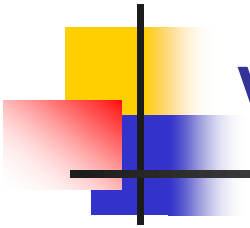
```
#include <stdlib.h>
#include <stdio.h>
#include "Graph.h"
typedef struct node *link;
struct node { int v; link next; } ;
struct graph { int V; int E; link *adj; } ;

link NEW(int v, link next) {
  link x = malloc(sizeof *x);
  x->v = v; x->next = next; return x;
  }
Edge EDGE(int v, int w) {
  Edge e; e.v = v; e.w = w;  return e;
}
Graph GRAPHinit(int V) {
  int v;
  Graph G = malloc(sizeof *G);
  G->V = V; G->E = 0;
  G->adj = malloc( V  * sizeof(link));
  for (v = 0; v < V; v++)
    G->adj[v] = NULL;
  return G;
}
```

Attenzione: si possono generare sia cappi sia archi ripetuti!

```
void  GRAPHinsertE(Graph G, Edge e) {
  int v = e.v, w = e.w;
  G->adj[v] = NEW(w, G->adj[v]);
  G->adj[w] = NEW(v, G->adj[w]);
  G->E++;
}
int   GRAPHedges(Graph G, Edge a[]) {
  int v, E = 0; link t;
  for (v=0; v < G->V; v++)
    for (t=G->adj[v]; t != NULL; t = t->next)
      if (v < t->v)
        a[E++] = EDGE(v, t->v);
  return E;
}
void  GRAPHshow(Graph G) {
  int v; link t;
  printf("%d vertices, %d edges \n", G->V, G->E);
  for (v=0; v < G->V; v++) {
    printf("%2d:", v);
    for (t=G->adj[v]; t != NULL; t = t->next)
      printf("%2d", t->v);
    printf("\n");
  }
}
```

Attenzione: solo per grafi non orientati

Attenzione: solo per grafi non orientati



Vantaggi/svantaggi

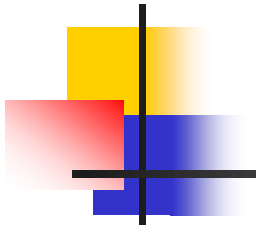
- Grafi non orientati:

  elementi complessivi nelle liste = 2|E|
- Grafi orientati:

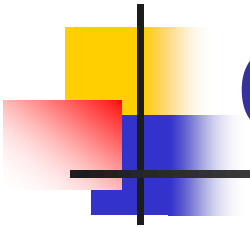
  elementi complessivi nelle liste = |E|
- Complessità spaziale

  S(n) = O(max(|V|, |E|)) = O(|V+E|)

  $\Rightarrow$ vantaggioso per grafi sparsi



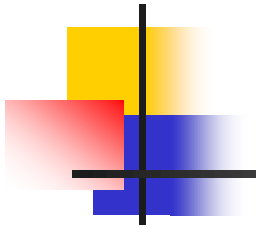
- Svantaggi:
  - verifica dell'esistenza di arco (v,w) mediante scansione della lista di adiacenza di v
  - uso di memoria per i pesi dei grafi pesati.



Generazione dei grafi

Tecnica 1: archi casuali

- Vertici come interi tra 0 e |V|-1
- generazione di un grafo casuale a partire da E coppie casuali di archi (interi tra 0 e |V|-1)
  - possibili archi ripetuti (multigrafo) e cappi
  - grafo con |V| vertici e |E| archi (inclusi cappi e archi ripetuti)



```
int randV(Graph G) {
  return G->V * (rand() / (RAND_MAX + 1.0));
}

Graph GRAPHrand(Graph G, int V, int E) {
  while (G->E < E)
    GRAPHinsertE(G, EDGE(randV(G), randV(G)));
  return G;
}
```

Tecnica 2: archi con probabilità p

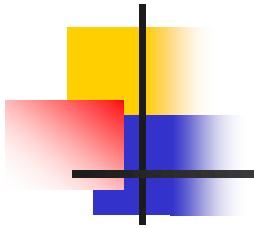
- si considerano tutti i possibili archi |V|*(|V|-1)/2
- tra questi si selezionano quelli con probabilità p
- p è calcolato in modo che sia

$$|E| = p*(|V|*(|V|-1)/2)$$

quindi

$$p = 2 * |E| / (|V| * (|V| - 1))$$

- si ottiene un grafo con in media |E| archi
- non ci sono archi ripetuti.



```
Graph GRAPHrand(Graph G, int V, int E) {
  int i, j;
  double p = 2.0 * E / (V * (V-1));
  for ( i = 0; i < V; i++)
    for ( j = i+1; j < V; j++)
      if (rand() < p * RAND_MAX)
        GRAPHinsertE(G, EDGE(i, j));
  return G;
}
```

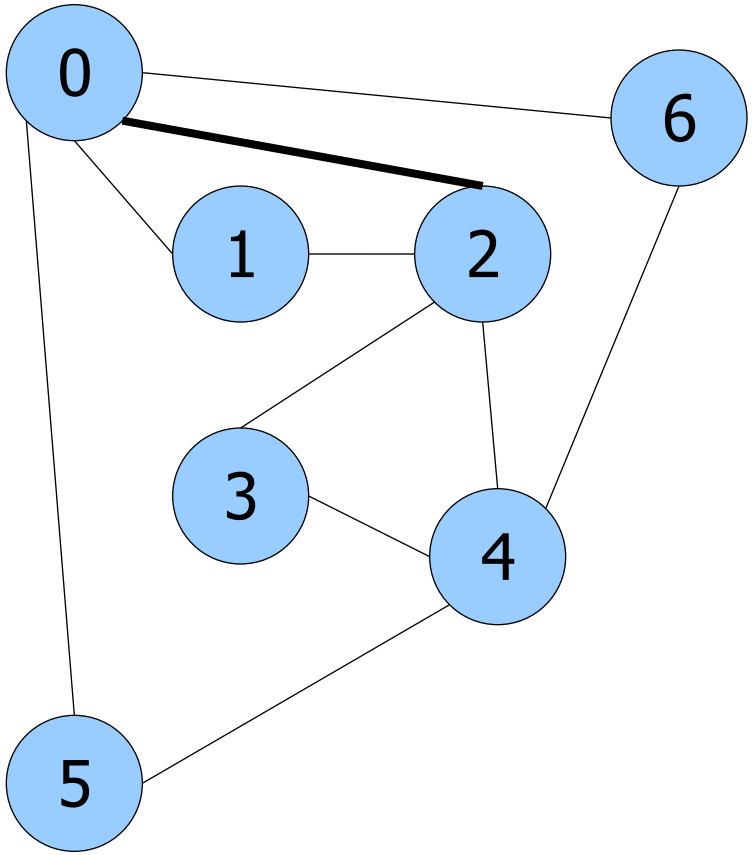
Cammino semplice

Dato un grafo non orientato G =(V, E) e 2 suoi vertici v e w, esiste un cammino semplice che li connette?

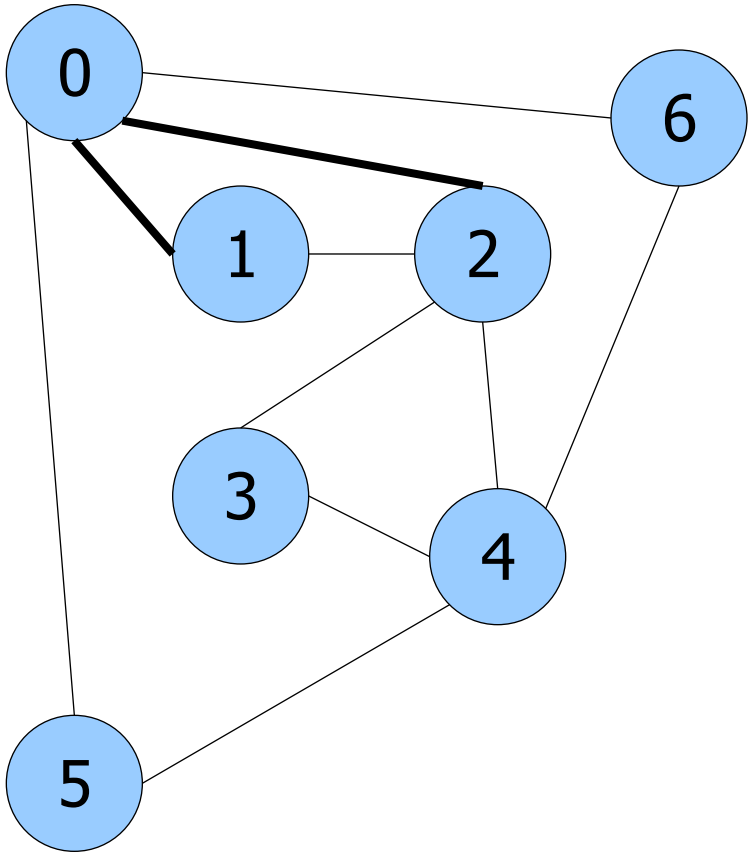
$\exists$ p: $v \rightarrow_p w$

- $\forall$ vertice t adiacente al vertice corrente v, determinare ricorsivamente se esiste un cammino semplice da t a w
- array `visited` per marcare i nodi già visitati
- cammino visualizzato in ordine inverso
- complessità T(n) = O(|V+E|)

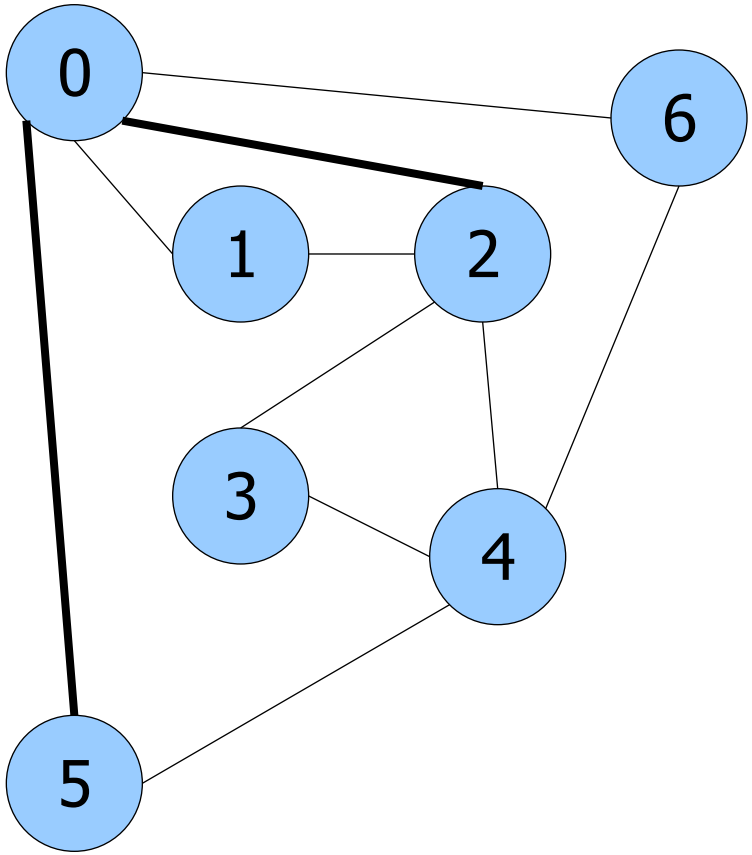
Esempio



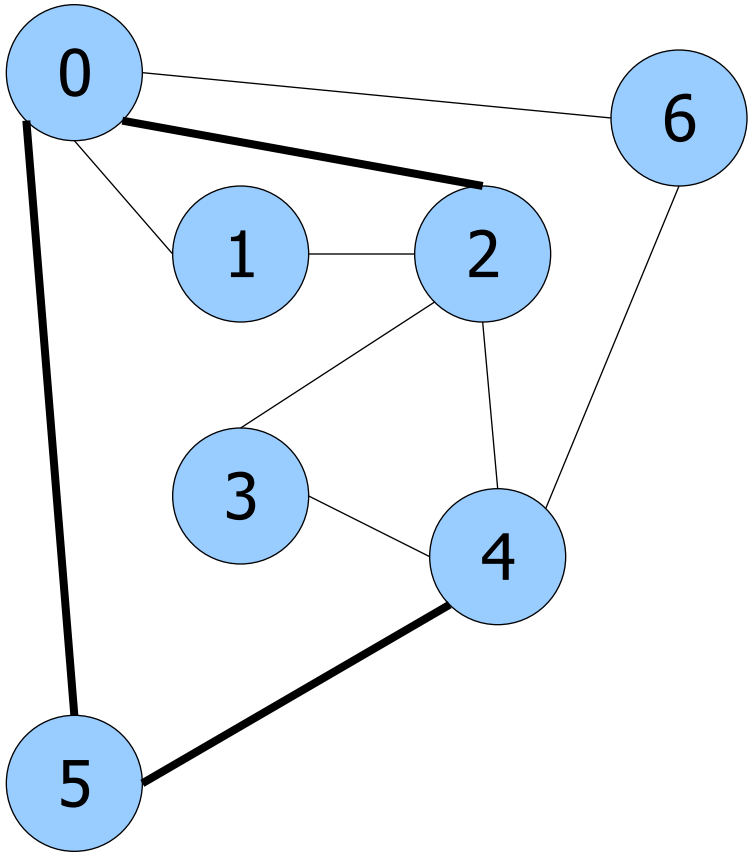
$\exists p: \ 2 \rightarrow_p 6?$
passo 1:
vertici adiacenti a 2
non ancora visitati:
0, 1, 3, 4
seleziono 0



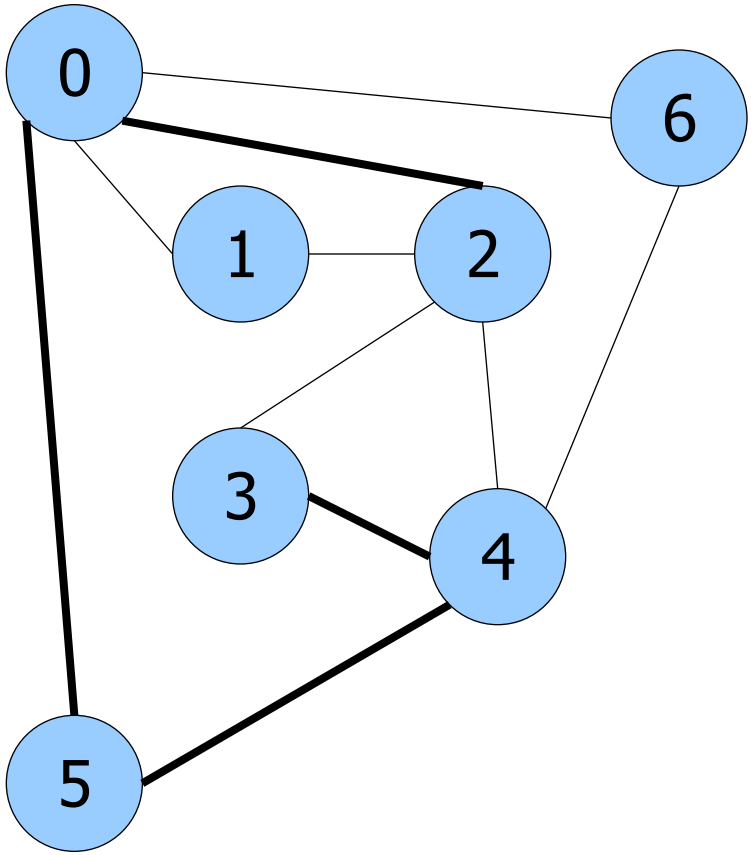
$\exists p: \ 0 \rightarrow_p 6?$
passo 2:
vertici adiacenti a 0
non ancora visitati:
1, 5, 6
seleziono 1



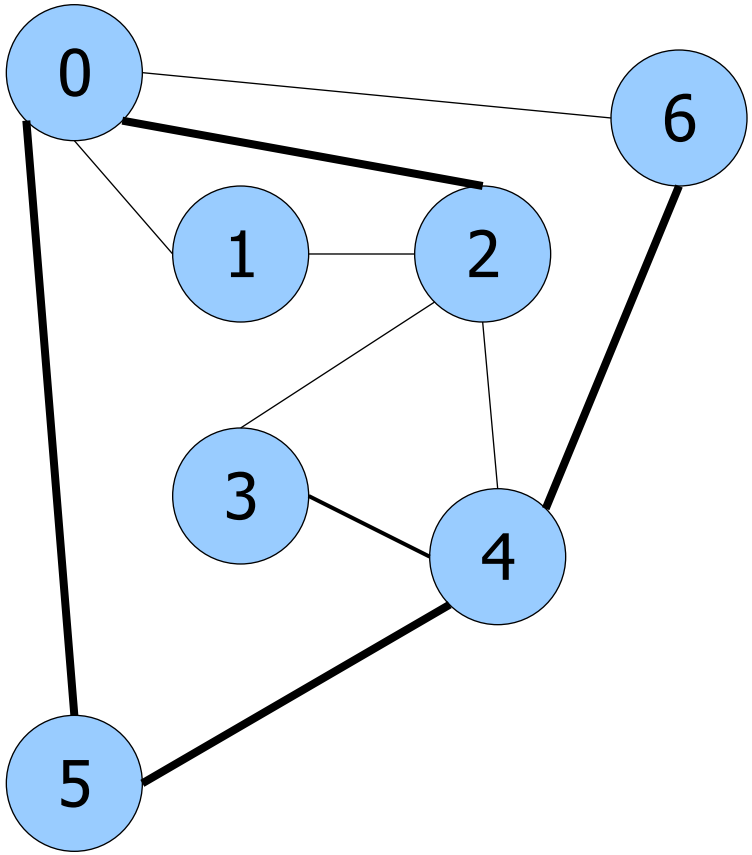
$\exists p: 1 \rightarrow_p 6?$
passo 3:
vertici adiacenti a 1
non ancora visitati:
nessuno
La ricorsione torna a
0 e seleziona 5



$\exists p: \ 5 \rightarrow_p 6?$
passo 4:
vertici adiacenti a 5
non ancora visitati:
4
seleziono 4



$\exists p: \ 4 \rightarrow_p 6?$
passo 5:
vertici adiacenti a 4
non ancora visitati:
3, 6
seleziono 3

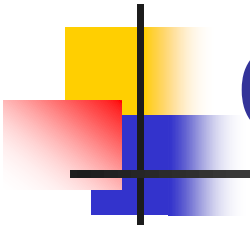


$\exists p: \ 3 \rightarrow_p 6?$
passo 6:
vertici adiacenti a 3
non ancora visitati:
nessuno
La ricorsione torna a
4 e seleziona 6

Matrice delle adiacenze

Stampa gli archi del cammino in ordine inverso

```
int pathR(Graph G, int v, int w) {
  int t;
  if (v == w) return 1;
  visited[v] = 1;
  for ( t = 0 ; t < G->V ; t++)
    if (G->adj[v][t] == 1)
      if (visited[t] == 0)
        if (pathR(G, t, w)) {
          printf("(%d, %d) \n", v, t);
          return 1;
        }
  return 0;
}
int GRAPHpath(Graph G, int v, int w) {
  int t;
  for ( t = 0 ; t < G->V ; t++) visited[t] = 0;
  return pathR(G, v, w);
}
```

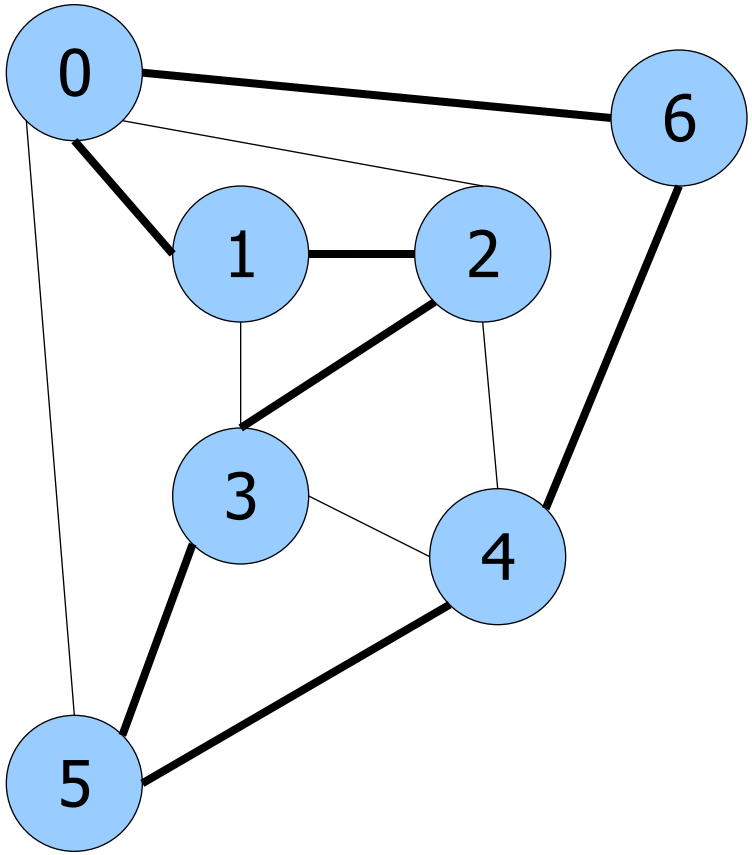


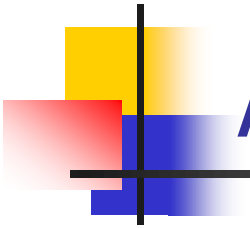
Cammino di Hamilton

Dato un grafo non orientato G =(V, E) e 2 suoi vertici v e w, se esiste un cammino semplice che li connette visitando ogni vertice una e una sola volta, questo si dice cammino di Hamilton.

Se v coincide con w, si parla di ciclo di Hamilton.

Esempio: ciclo di Hamilton





Algoritmo

Cammino di Hamilton tra v e w:

- $\forall$ vertice t adiacente al vertice corrente v, determinare ricorsivamente se esiste un cammino semplice da t a w
- ritorno con successo se e solo se la lunghezza del cammino è |V|-1
- set della cella dell'array `visited` per marcare i nodi già visitati
- reset della cella dell'array `visited` quando ritorna con insuccesso
- complessità esponenziale!

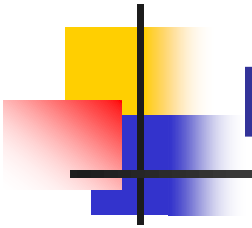
```
int pathRH(Graph G, int v, int w, int d) {
  int t;
  if (v == w) {
    if (d == 0) return 1;
    else return 0;
  }
  visited[v] = 1;
  for ( t = 0 ; t < G->V ; t++)
  if (G->adj[v][t] == 1)
    if (visited[t] == 0)
      if (pathRH(G, t, w, d-1)){
        printf("(%d, %d) in Hpath \n", v, t);
        return 1;
      }
  visited[v] = 0;
  return 0;
}
int GRAPHpathH(Graph G, int v, int w) {
  int t;
  for ( t = 0 ; t < G->V ; t++) visited[t] = 0;
  return pathRH(G, v, w, G->V-1);
}
```

Intero che indica quanto manca a un cammino lungo |V|-1

Cammino di Eulero

Dato un grafo non orientato G =(V, E) e 2 suoi vertici v e w, si dice cammino di Eulero un cammino (anche non semplice) che li connette attraversando ogni arco una e una sola volta.

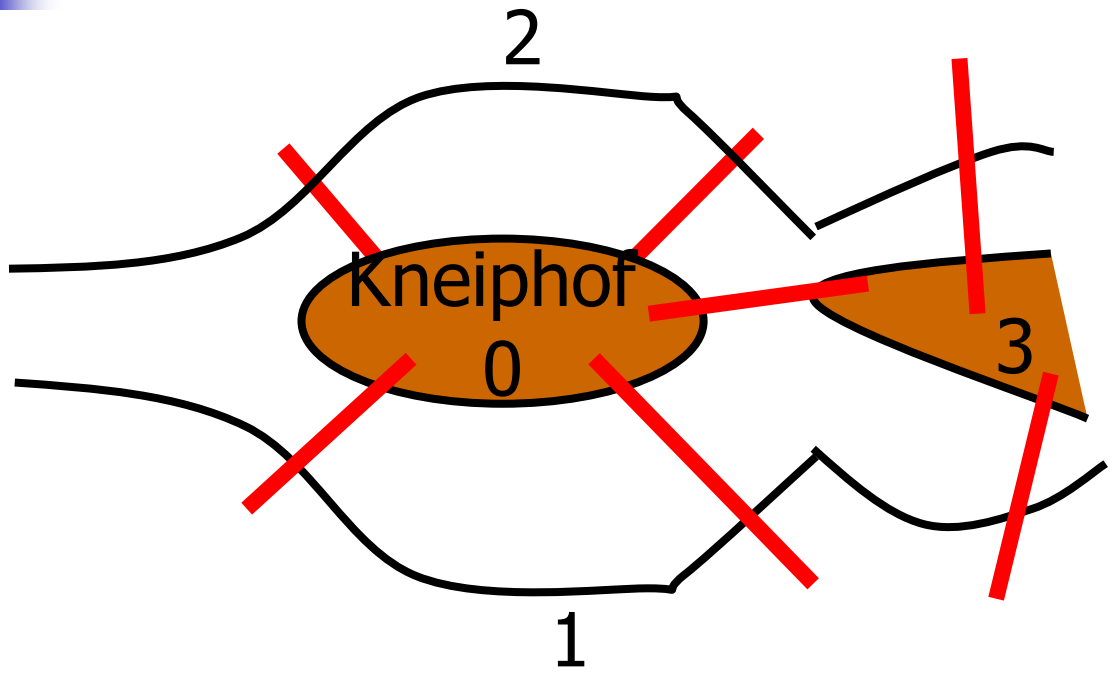
Se v coincide con w, si parla di ciclo di Eulero.

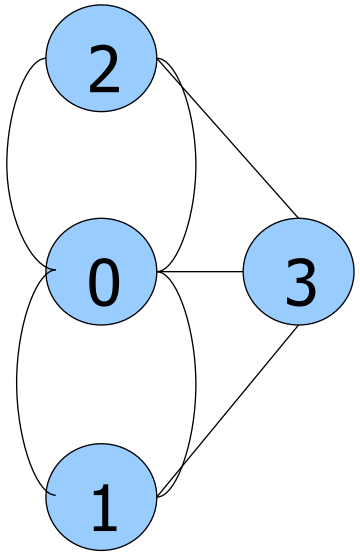


Lemmi

- Un grafo non orientato ha un ciclo di Eulero se e solo se è connesso e tutti i suoi vertici sono di grado pari
- Un grafo non orientato ha un cammino di Eulero se e solo se è connesso e se esattamente due vertici hanno grado dispari.

I ponti di Königsberg



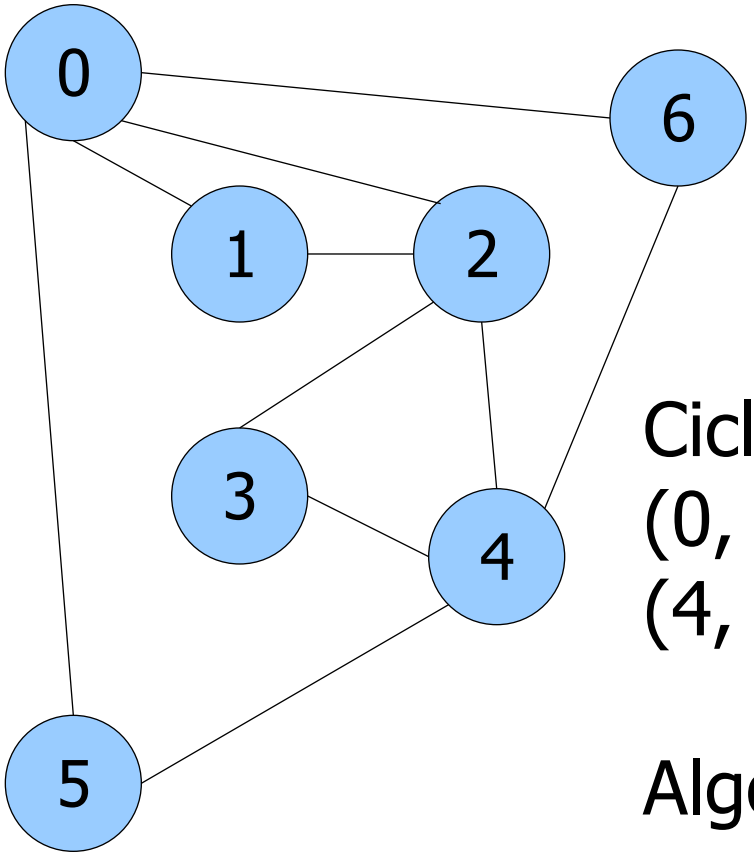


Multigrafo: archi multipli che connettono la stessa coppia di vertici

$\nexists$ né cammini, né cicli di Eulero

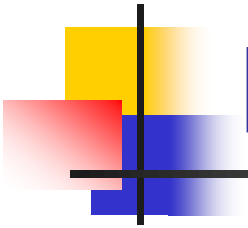
Esempio: verifica della esistenza di ciclo di Eulero

nodo	deg
0	4
1	2
2	4
3	2
4	4
5	2
6	2



Ciclo di Eulero:
(0, 1), (1, 2), (2, 0), (0, 6), (6, 4)
(4, 3), (3, 2), (2, 4), (4, 5), (5, 6)

Algoritmo di complessità O(|E|)



Riferimenti

- L'ADT grafo non orientato:
  - Sedgewick Part 5: 17.2
- Rappresentazione dei grafi:
  - Sedgewick Part 5: 17.3, 17.4
  - Cormen 23.1
- Generazione di grafi:
  - Sedgewick Part 5: 17.6
- Cammini semplici, di Hamilton, di Eulero:
  - Sedgewick Part 5: 17.6
  - Cormen 36.2, 36.5.4