il sito **The Oracle of Bacon**

- **mantiene un grafo** con attori legati da un arco se hanno recitato in uno stesso film
- dato un attore ne **calcola** il **numero di Bacon**, i.e. **lunghezza del cammino minimo** che lo separa da Kevin Bacon (***breadth-first search*** *algorithm* starting from Bacon)

- un **grafo**
- un **algoritmo** che risolve un problema:
  dati due nodi, calcola il cammino minimo

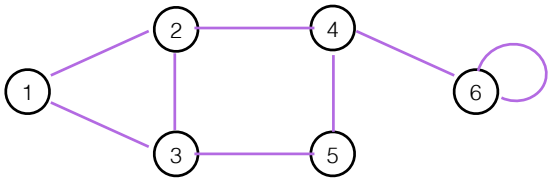
i dati del problema

i dati del programma

- una **struttura dati** per rappresentare un grafo
- **implementazione di un algoritmo** su un grafo

rappresentazione di un grafo

Un **grafo** è un insieme di vertici/**nodi** e un insieme di **archi** tra coppie di nodi:



G=(V,E) con

V={1,2,3,4,5,6}

E={(1,2), (1,3), (2,3), (2,4), (4,6), (3,5), (4,5), (6,6) }

Come rappresentiamo un grafo in un programma ?

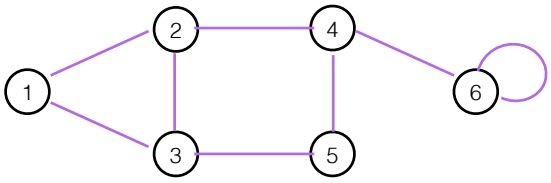
- si potrebbe implementare un grafo come **due liste**:

```
vertici = [1,2,3,4,5,6]
archi = [[1,2],[1,3],[2,3],[2,4],[4,6],[3,5],[4,5],[6,6]]
```

- basta **un intero** e una **lista**:

```
n_vertici = 6
archi = [[1,2],[1,3],[2,3],[2,4],[4,6],[3,5],[4,5],[6,6]]
```

rappresentazione di un grafo




```
n_vertici = 6
archi = [[1,2],[1,3],[2,3],[2,4],[4,6],[3,5],[4,5],[6,6]]
```

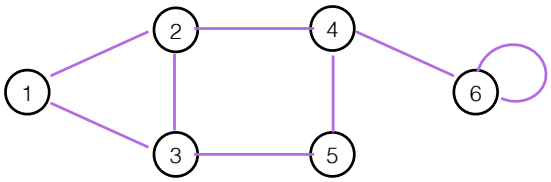
- come calcolare quanti archi escono da nodo 3?
- come calcolare se c'è un arco che collega 1 e 6?

devo cercare in tutto la lista `archi` !

```
quanti_da_3 = 0
for x in archi:
    if x[0]==3 or x[1]==3:
        quanti_da_3 = quanti_da_3+1
```

```
arco_1_6 = False
for x in archi:
    if (x[0]==1 and x[1]==6) or
       (x[0]==6 and x[1]==1):
          arco_1_6 = True
          break
```

rappresentazione di un grafo



V={1,2,3,4,5,6}

E={(1,2), (1,3), (2,3), (2,4), (4,6), (3,5), (4,5), (6,6) }

Come rappresentiamo un grafo in un programma?

matrice di adiacenza

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

in posizione i,j c'è 1 se esiste l'arco (i,j) altrimenti c'è 0

lista di adiacenza

1 : 2, 3
2 : 1, 3, 4
3 : 1, 2, 5
4 : 2, 5, 6
5 : 3, 4
6 : 4, 6

in posizione i c'è l'elenco di nodi collegati ad i con un arco

- come calcolare quanti archi escono da nodo 3?
- come calcolare se c'e' un arco che collega 1 e 6?

è immediato!

matrice di adiacenza

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

lista di adiacenza

1 : 2, 3
2 : 1, 3, 4
3 : 1, 2, 5
4 : 2, 5, 6
5 : 3, 4
6 : 4, 6

- come calcolare quanti archi escono da nodo 3?
- come calcolare se c'e' un arco che collega 1 e 6?
  è immediato!

```
matrice = [[0,1,1,0,0,0],
           [1,0,1,1,0,0],
           [1,1,0,0,1,0],
           [0,1,0,0,1,1],
           [0,0,1,1,0,0],
           [0,0,0,1,0,1]]
```

```
lista = [[2,3],
         [1,3,4],
         [1,2,5],
         [2,5,6],
         [3,4],
         [4,6]]
```

```
# la riga del nodo 3 è matrice[2]
quanti_da_3 = 0
for i in matrice[2]: #somma
    quanti_da_3 = quanti_da_3 + i


arco_1_6 = (matrice[0][5] == 1)
```

```
quanti_da_3 = 0
for i in lista[2]:  #lunghezza
   quanti_da_3=quanti_da_3 + 1

arco_1_6 = False
for n in lista[0]:
    if n==6:
        arco_1_6 = True
        break
```

Esercizio

Si consideri quanto visto a lezione:

```
matrice = [[0,1,1,0,0,0],
           [1,0,1,1,0,0],
           [1,1,0,0,1,0],
           [0,1,0,0,1,1],
           [0,0,1,1,0,0],
           [0,0,0,1,0,1]]
```

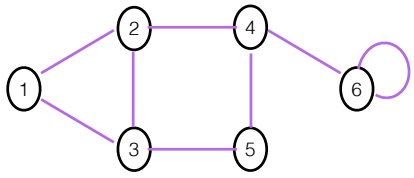
```
lista = [[2,3],
         [1,3,4],
         [1,2,5],
         [2,5,6],
         [3,4],
         [4,6]]
```

```
# la riga del nodo 3 è matrice[2]
quanti_da_3 = 0
for i in matrice[2]: #somma
    quanti_da_3 = quanti_da_3 + i


arco_1_6 = (matrice[0][5] == 1)
```

```
quanti_da_3 = 0
for i in lista[2]:  #lunghezza
   quanti_da_3=quanti_da_3 + 1

arco_1_6 = False
for n in lista[0]:
    if n==6:
        arco_1_6 = True
        break
```

1. **Tracciare il flusso di controllo dei due programmi**.
2. **Contare quante istruzioni vengono eseguite in totale** dai due programmi. Qual è quello che ***termina prima***, i.e. esegue meno istruzioni?

Esercizio



Si consideri il programma che usa la seguente rappresentazione dello stesso grafo:

```
n_vertici = 6
archi = [[1,2],[1,3],[2,3],[2,4],[4,6],[3,5],[4,5],[6,6]]
```

```
1  quanti_da_3 = 0
2  for x in archi:
3      if x[0]==3 or x[1]==3:
4          quanti_da_3 = quanti_da_3+1
5
6  arco_1_6 = False
7  for x in archi:
8      if (x[0]==1 and x[1]==6) or
9         (x[0]==6 and x[1]==1):
10           arco_1_6 = True
11           break
```

Tracciare il flusso di controllo del programma e contare quante istruzioni vengono eseguite in totale.

14 passi

```
matrice = [[0,1,1,0,0,0],
           [1,0,1,1,0,0],
           [1,1,0,0,1,0],
           [0,1,0,0,1,1],
           [0,0,1,1,0,0],
           [0,0,0,1,0,1]]

riga_nodo_3 = matrice[2] #indici da 0
quanti_da_3 = 0
for i in riga_nodo_3: #somma elementi
    quanti_da_3 = quanti_da_3 + i

arco_1_6 = (matrice[0][5] == 1)
```

12 passi

```
lista = [[2,3],
         [1,3,4],
         [1,2,5],
         [2,5,6],
         [3,4],
         [4,6]]

quanti_da_3 = 0
for i in lista[2]:  #lunghezza
   quanti_da_3=quanti_da_3 + 1

arco_1_6 = False
for n in lista[0]:
    if n==6:
        arco_1_6 = True
        break
```

- calcolare quanti archi escono da nodo 3
- calcolare se c'e' un arco che collega 1 e 6

37 passi

```
n_vertici = 6
archi = [[1,2],[1,3],[2,3],[2,4],[4,6],[3,5],[4,5],[6,6]]

quanti_da_3 = 0
for x in archi:
    if x[0]==3 or x[1]==3:
        quanti_da_3 = quanti_da_3+1
arco_1_6 = False
for x in archi:
    if (x[0]==1 and x[1]==6) or
       (x[0]==6 and x[1]==1):
          arco_1_6 = True
          break
```

lo **stesso problema**, sullo **stesso grafo** viene risolto con **efficienza molto diversa** a seconda del modo che scelgo per **rappresentare** i ***dati del problema*** (il grafo) in ***dati del programma***

Esercizio

1. Comparare il numero di passi svolti dai tre programmi su un grafo iniziale con **più nodi** e **più archi**

2. Rifare l'esercizio precedente su un grafo che ha **molti più nodi che archi**

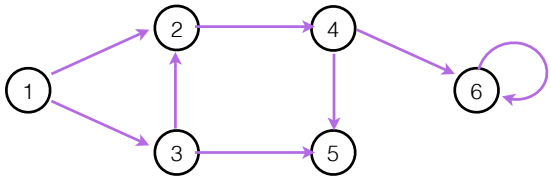
3. Rifare l'esercizio precedente su un grafo che ha **molti più archi che nodi**

4. osservare se ci sono differenze tra qual è la rappresentazione del grafo più efficiente nel caso 2 e 3.

Esercizio

- Considerare il grafo G composto dai vertici={1,2,3,4,5,6} e dagli archi {(1,2),(2,4),(1,4),(3,4),(5,3),(4,5),(5,6),(6,1)}
- Disegnare il grafo
- Scrivere la corrispondente matrice di adiacenza e la corrispondente lista di adiacenza
- Scrivere un programma che **calcola se escono più archi dal nodo 6 oppure dal nodo 1**.
  - confronta il numero di istruzioni eseguite dal programma che usa la matrice di adiacenza rispetto a quello che usa la lista di adiacenza
- Scrivere un programma che trova **qual è il nodo da cui escono più archi**
  - confronta il numero di istruzioni eseguite dal programma che usa la matrice di adiacenza rispetto a quello che usa la lista di adiacenza

rappresentazione: grafo orientato



matrice di adiacenza *n x n*

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

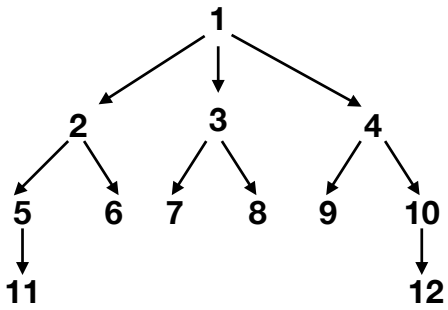
quali archi **escono** da nodo 5

quali archi **entrano** in nodo 5

lista di adiacenza m

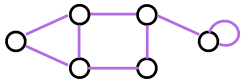
1 : 2, 3
2 : 1, 3, 4
3 : 1, 2, 5
4 : 2, 5, 6
5 : 3, 4
6 : 4, 6

Esercizio



- Considerare il grafo **orientato** in figura
- Scrivere la corrispondente matrice di adiacenza e la corrispondente lista di adiacenza
- Scrivere un programma che calcola quanti archi **escono** dal nodo 2 e quanti archi **entrano** nel nodo 2
  - confronta il numero di istruzioni eseguite dal programma che usa la matrice di adiacenza rispetto a quello che usa la lista di adiacenza
- Scrivere un programma che calcola **quanti nodi sono raggiungibili a partire dal nodo 4**
  - è più chiaro il codice che risolve l'esercizio usando la matrice di adiacenza oppure quello che usa la lista di adiacenza?

rappresentazione di un grafo



la ***scelta è fortemente dipendente*** da:

- che caratteristiche voglio evidenziare,
- che uso devo farne,
- quanto è grande il grafo da memorizzare,
- qual è la sua densità

grafi sociali **non** stanno su memoria di una singola macchina!

matrice di adiacenza

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

lista di adiacenza

1 : 2, 3
2 : 1, 3, 4
3 : 1, 2, 5
4 : 2, 5, 6
5 : 3, 4
6 : 4, 6