

Parte 5 - Grafi

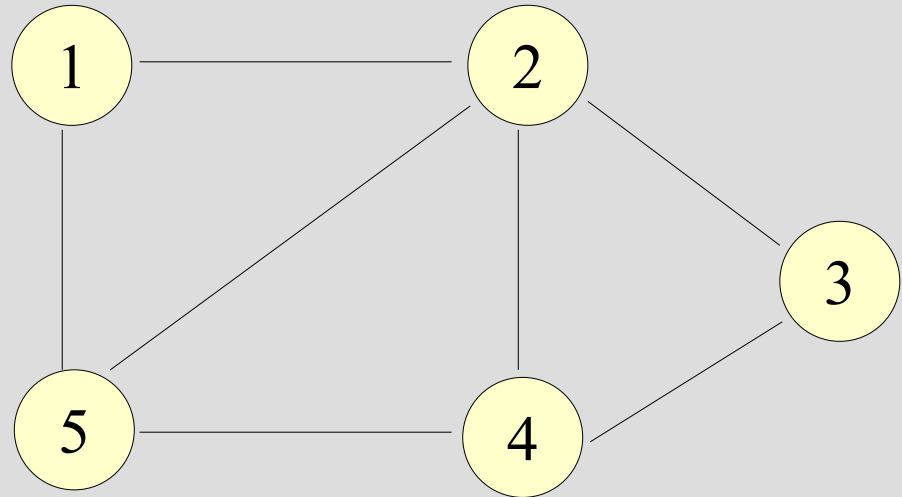
Rappresentazione dei grafi



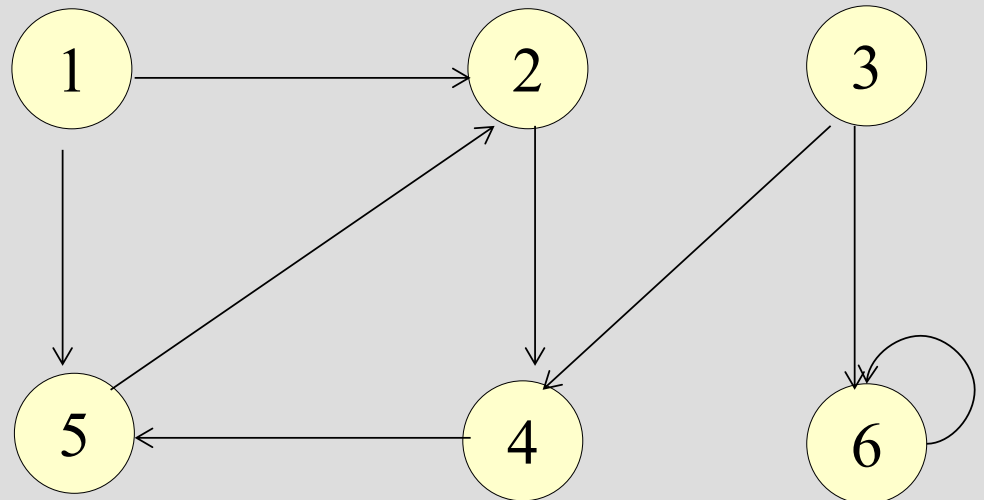
Rembrandt *Ronda di notte* 1642

Grafo

$G = (V, E)$
non orientato



$G = (V, E)$
orientato



Rappresentazione

Grafo $G = (V, E)$

Assumiamo che ogni vertice in V sia univocamente identificato da un *identificativo* nel range $[1, |V|]$

2 metodi standard per la rappresentazione

- **Liste di adiacenza**
- **Matrici di adiacenza**

Entrambi validi sia per **grafi orientati** che **non orientati**

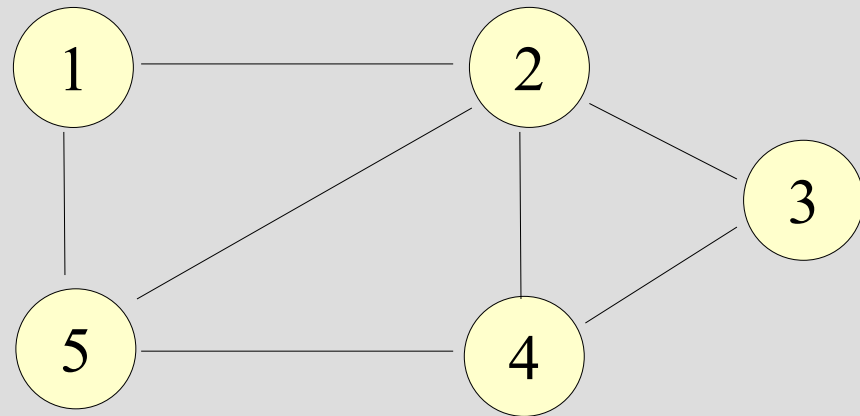
Liste di adiacenza

Grafo $G = (V, E)$

- Array **Adj** di $|V|$ liste, una per ogni vertice in **V**
- Per ogni vertice **u** in **V**, **Adj[u]** contiene tutti i vertici **v** in **V** tali che esista un arco **(u,v)** in **E** (tutti i **vertici adiacenti** a **u** in **G**, memorizzati in **ordine arbitrario**)
- A livello implementativo, una soluzione è che **Adj[u]** contenga un **puntatore** alla testa della lista tali vertici

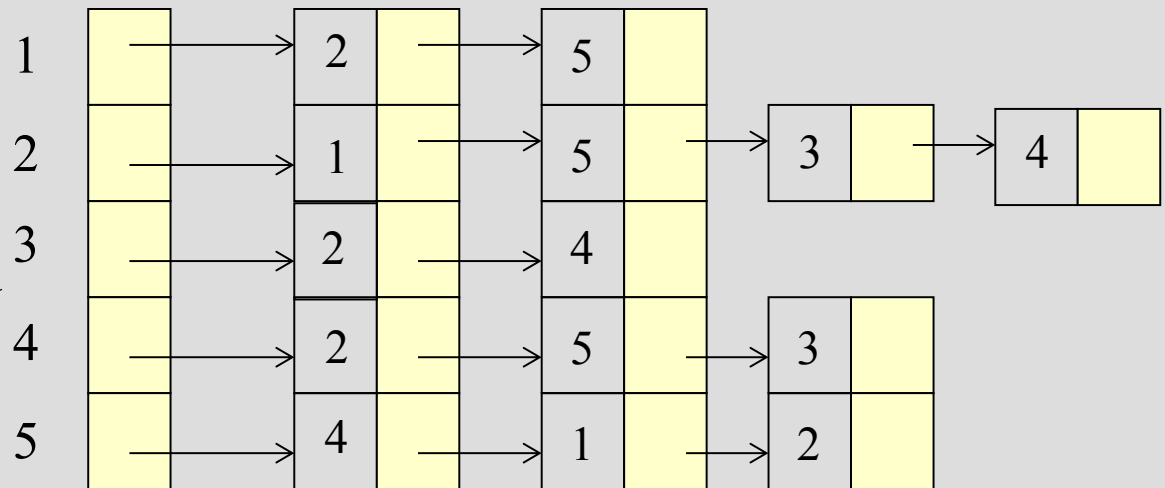
Grafo non orientato

$G = (V, E)$
non orientato



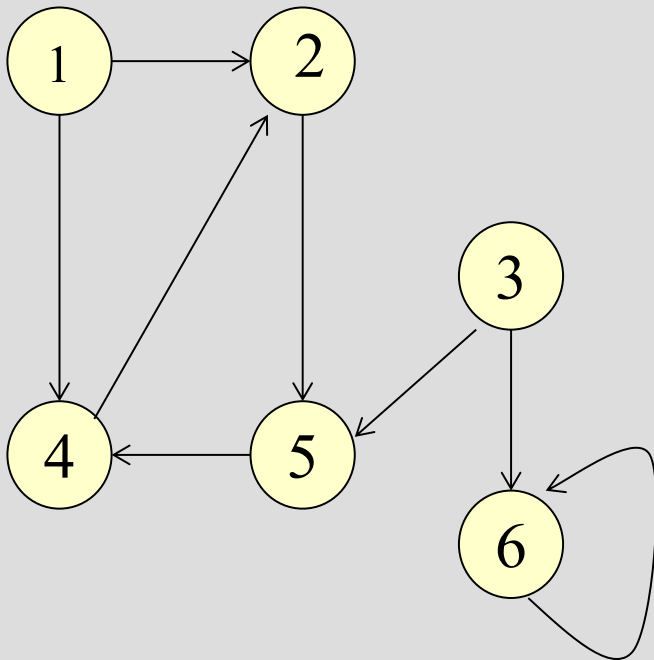
Lista di adiacenza:

Ogni arco (u,v) è memorizzato nella lista di adiacenza di u e nella lista di adiacenza di v



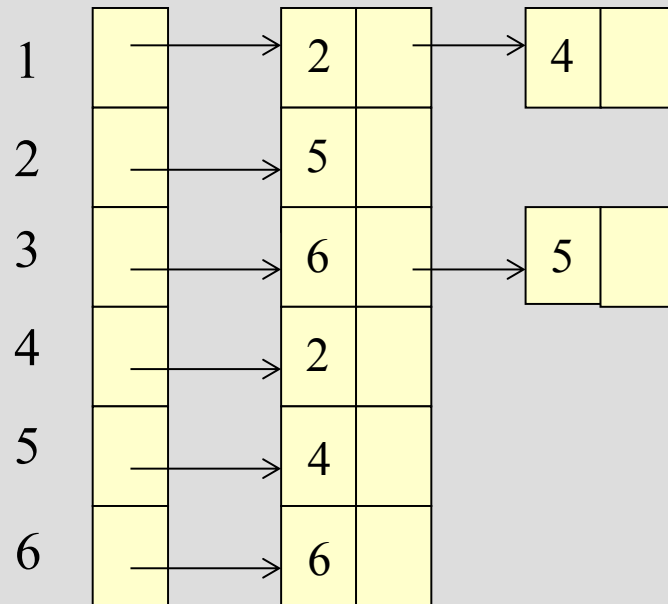
Grafo orientato

$G = (V, E)$
orientato



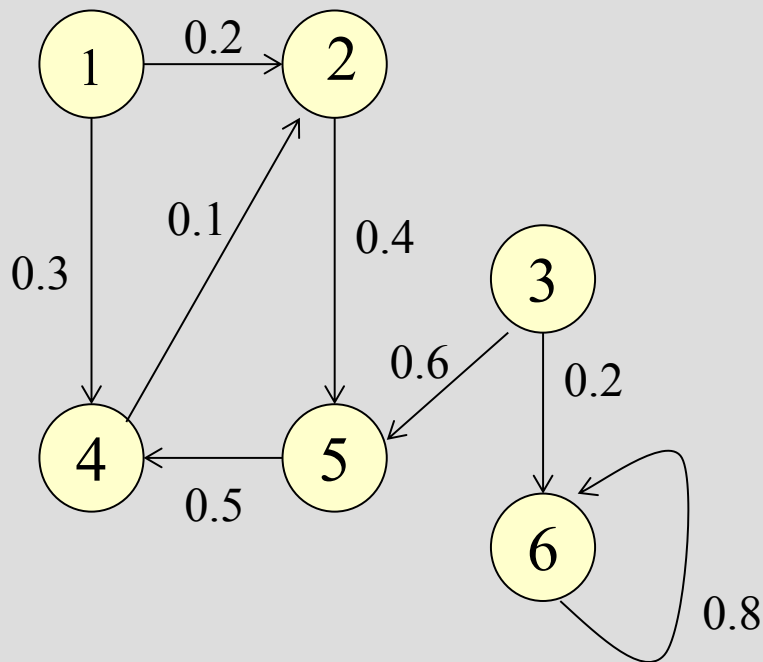
Lista di adiacenza:

La somma delle lunghezze
di tutte le liste di adiacenza
è pari ad $|E|$

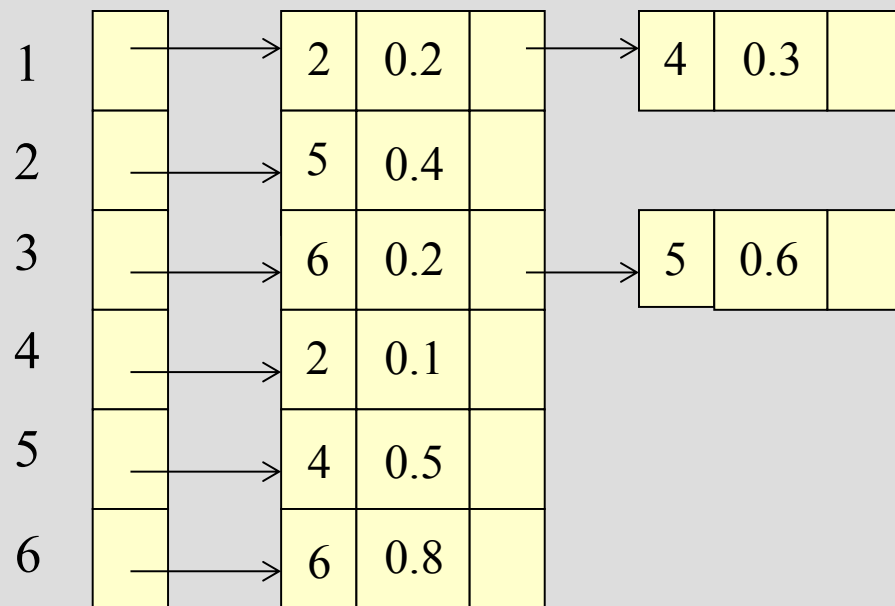


Grafo orientato e pesato

$G = (V, E)$
orientato e pesato



Lista di adiacenza:
il peso dell'arco (u,v) è
memorizzato col vertice
 v nella lista di u



Matrici di adiacenza

Grafo $G = (V, E)$

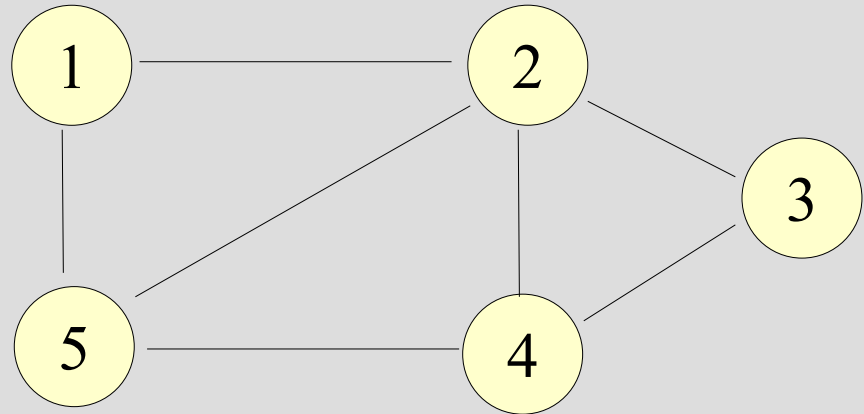
- Matrice $A = (a_{ij})$ di dimensione $|V| \times |V|$

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \text{ appartiene a } E \\ 0 & \text{altrimenti} \end{cases}$$

- Per *archi pesati* memorizzo il peso anziché il valore 1

Grafo non orientato

$G = (V, E)$
non orientato

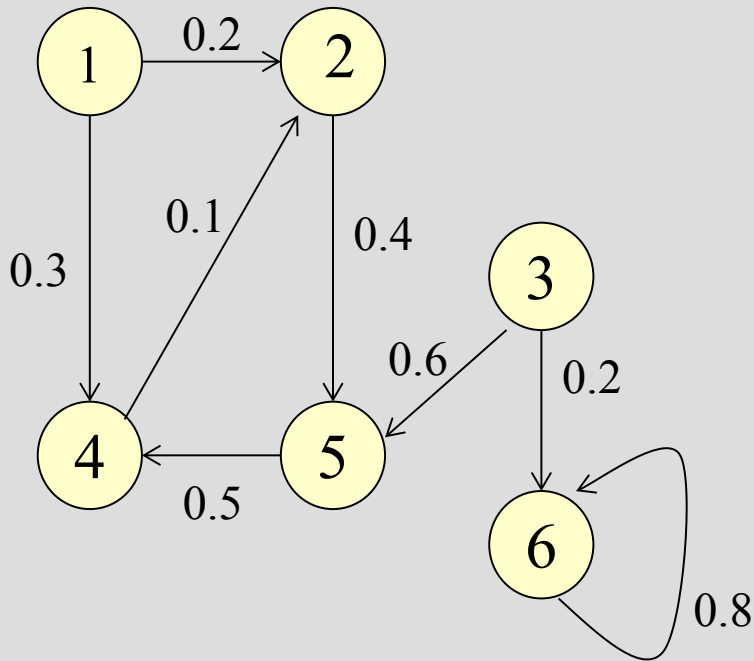


**Matrice di
adiacenza
(simmetrica)**

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

Grafo orientato (e pesato)

$G = (V, E)$
orientato e pesato



**Matrice di adiacenza
(asimmetrica)**

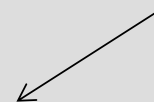
	1	2	3	4	5	6
1	0	.2	0	.3	0	0
2	0	0	0	0	.4	0
3	0	0	0	0	.6	.2
4	0	.1	0	0	0	0
5	0	0	0	.5	0	0
6	0	0	0	0	0	.8

Implementazione con liste

Struttura Lista di adiacenza

```
struct adj_node {  
    int node;  
    float weight;  
    adj_node* next;  
};
```

Puntatore al prossimo elemento
della lista di adiacenza



Lista di adiacenza

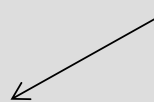


```
typedef adj_node* adj_list;
```

Grafo

```
typedef struct {  
    adj_list* nodes;  
    int dim;  
} graph;
```

Array dinamico di *dim* liste di
adiacenze, una per vertice



Numero dei vertici



Osservazione sull'implementazione

- Le teste delle liste di adiacenza vengono memorizzate in un vettore dinamico della dimensione corrispondente al numero dei vertici

```
graph g;  
g.dim = ...;  
g.nodes = new adl_list[g.dim];
```

- I vertici del grafo nell'implementazione a lista di adiacenza sono quindi identificati dagli indici $0 \dots g.dim-1$
- L'identificativo dei nodi V di un grafo $G = (V,E)$ è nel range $[1,|V|]$
- Le primitive per l'accesso e la manipolazione di liste di adiacenza associate ai nodi dovranno occuparsi della *conversione* da identificativo del nodo a indice nel vettore

Creazione e aggiornamento di un grafo

```
graph new_graph(int n)
```

Restituisce la rappresentazione di un grafo di n vertici identificati univocamente da 1 a n attraverso n liste di adiacenza

```
void add_arc(graph& g,int s,int d,float w)
```

Aggiunge l'arco orientato (s,d) con peso w alla lista di adiacenza del nodo s

```
void add_edge(graph& g,int s,int d,float w)
```

Aggiunge l'arco non orientato (s,d) con peso w alla lista di adiacenza del nodo s e del nodo d

Primitive di accesso ad un grafo

`int get_dim(graph)`

Restituisce il numero n dei nodi del grafo

`adjlist get_adjlist(graph, int)`

Restituisce la *testa* della lista di adiacenza del nodo con identificativo in ingresso

`int get_adjnode(adj_node*)`

Restituisce l'identificativo del nodo contenuto nell'elemento della lista di adiacenza

`adj_list get_nextadj(adj_list)`

Restituisce il prossimo elemento della lista di adiacenza

Il modulo «grafo»

- Creare il modulo «grafo» che implementa un grafo attraverso liste di adiacenza
- Il modulo deve contenere la definizione dei tipi associati all'implementazione
- Il modulo deve contenere le primitive per la creazione e l'aggiornamento del grafo

SOLUZIONE Vedi cartella *grafo*

Lettura grafo da file

Vedi cartella *file-grafo*

graph1 e *graph2*: **file di input** che contengono un **elenco di archi**

Es.

7 <

Numero di nodi componenti il grafo

1 2

1 3

2 3

2 4

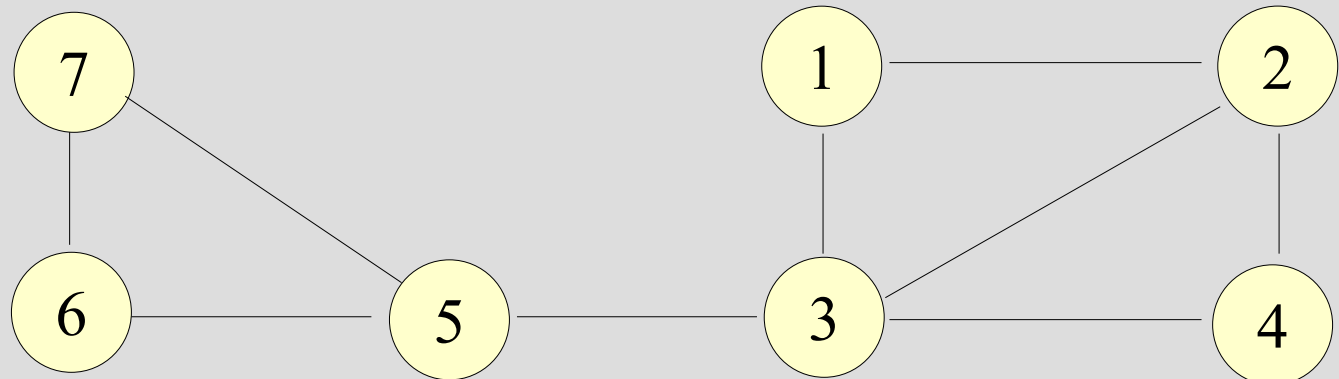
4 3

3 5

5 6

5 7

7 6



Nota: archi interpretabili come orientati o non orientati

Lettura grafo da file

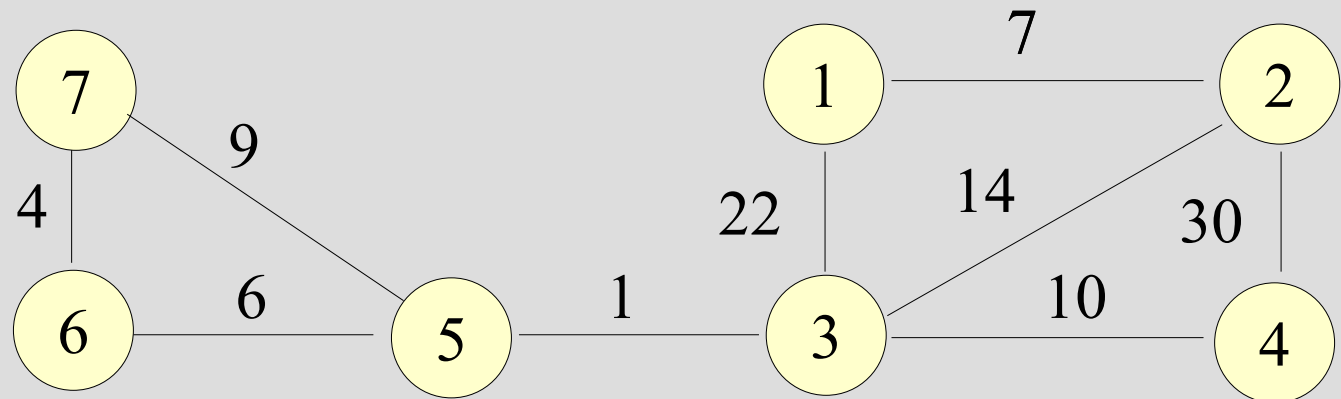
Vedi cartella *file-grafo*

graph1w e *graph2w*: **file di input** che contengono un **elenco di archi pesati**

Es.

7
1 2 7
1 3 22
2 3 14
2 4 30
4 3 10
3 5 1
5 6 6
5 7 9
7 6 4

Numero di nodi componenti il grafo



Nota: archi interpretabili come orientati o non orientati

ESERCIZIO

- Scrivere un programma che consenta di creare un grafo leggendolo da un file
- I file che contengono i grafi si trovano nella cartella *file-grafo*
- Il nome del file, la tipologia di grafo, orientato/non orientato, pesato/non pesato, vengono passati come argomenti alla chiamata del programma

ESEMPIO

```
graph graph1 1 0
```

Richiama l'eseguibile `graph` passandogli come parametri il file «graph1» e specificando che il grafo è orientato e non pesato

Lettura argomenti

Come leggere tali argomenti nei nostri programmi C++?

- Eseguire un programma equivale a chiamare la funzione **main**

```
int main(int argc, char *argv[])
```

- riceve in ingresso due argomenti:
- un intero **argc** che corrisponde al numero di argomenti
- un array di stringhe **argv** che contiene gli argomenti, uno per stringa

Lettura argomenti

- Per convenzione `argv[0]` contiene il nome con il quale il programma è stato invocato

Quindi

- `argc` vale sempre almeno 1
- Gli argomenti passati al programma sono memorizzati nelle stringhe `argv[1]... argv[argc-1]`

ESEMPIO

```
int main(int argc, char *argv[])
{ /* main che stampa gli argomenti */
    for (int i = 0 ; i < argc ; i++)
        cout<<argv[i]<<endl ;
    return 0 ;
}
```

Esercizio (cont.)

Per realizzare il programma usare il modulo «grafo» e scrivere il file `main.cc` che contiene le funzioni:

- `graph g_build(ifstream &g, bool d, bool w)` che legge dallo stream `g` il numero dei nodi (prima riga), genera il corrispondente grafo e lo popola aggiungendo gli archi specificati nelle righe successive del file.
`g_build` deve creare un grafo orientato o meno in base al parametro booleano `d` e deve leggere i pesi dal file se `w` è `TRUE`
- `int main(int argc, char *argv[])` che legge gli argomenti passati alla chiamata del programma, segnala un errore se il numero degli argomenti non è corretto, apre il file con il nome indicato nel primo argomento, richiama `g_build` per costruire il grafo e stampa il contenuto del grafo: per ogni nodo – identificativo del nodo e identificativo dei nodi adiacenti

SOLUZIONE Vedi sorgente `graph_sol.cc`