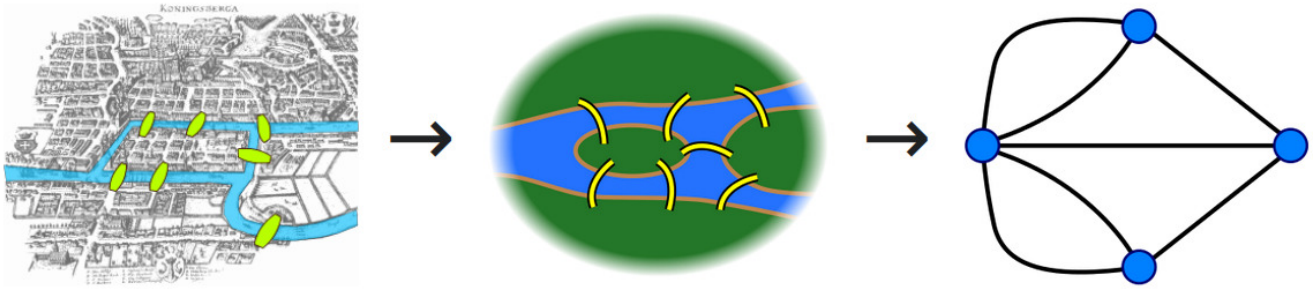
LabASD - Implementazione del tipo “Grafo”

Moreno Marzolla moreno.marzolla@unibo.it (mailto:moreno.marzolla@unibo.it)

Ultimo aggiornamento: 2022-04-08



Il problema dei sette ponti di Königsberg (https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg) risolto da Leonhard Euler (https://en.wikipedia.org/wiki/Leonhard_Euler) nel 1736 ha segnato la nascita della teoria dei grafi

Questo file contiene alcune funzioni per gestire la rappresentazione di grafi mediante liste di adiacenza. È possibile rappresentare sia grafi orientati che non orientati, usando le strutture dati seguenti:

```
typedef struct Edge {
    int src;
    int dst;
    double weight;
    struct Edge *next;
} Edge;

typedef enum { GRAPH_UNDIRECTED, GRAPH_DIRECTED } Graph_type;

typedef struct {
    int n;
    int m;
    Graph_type t;
    Edge **edges;
    int *in_deg;
    int *out_deg;
} Graph;
```

`Edge` rappresenta un arco del grafo. Per grafi non orientati, ogni arco (u, v) deve essere presente due volte: una come (u, v) nella lista di adiacenza di u, e una come (v, u) nella lista di adiacenza di v. Le liste sono concatenate semplici; il campo `next` indica l’arco successivo della lista di adiacenza, oppure `NULL` se è l’ultimo nodo.

La struttura `Graph` rappresenta l’intero grafo; la spiegazione dei vari campi è nel file graph.h (graph.h). Il campo `edges` è un array di lunghezza n di puntatori: `edges[v]` punta all’inizio della lista di adiacenza del nodo v, oppure `NULL` se v non ha archi uscenti. `in_deg` e `out_deg` sono array di lunghezza n; `in_deg[v]` e `out_deg[v]` sono, rispettivamente, il grado entrante e il grado uscente di v. Nel caso di grafi non orientati, si deve avere `out_deg[v] == in_deg[v]`

per ogni v, dato che ogni arco viene considerato sia entrante che uscente su entrambi gli estremi. È necessario mantenere l'informazione sui gradi entranti/uscenti durante la costruzione del grafo.

La Figura 1 mostra un esempio di rappresentazione di un grafo orientato, mentre la Figura 2 mostra un esempio di grafo non orientato; si noti che l'ordine con cui gli archi compaiono nelle liste di adiacenza non è rilevante, e dipende dall'ordine con cui sono stati inseriti nel grafo.

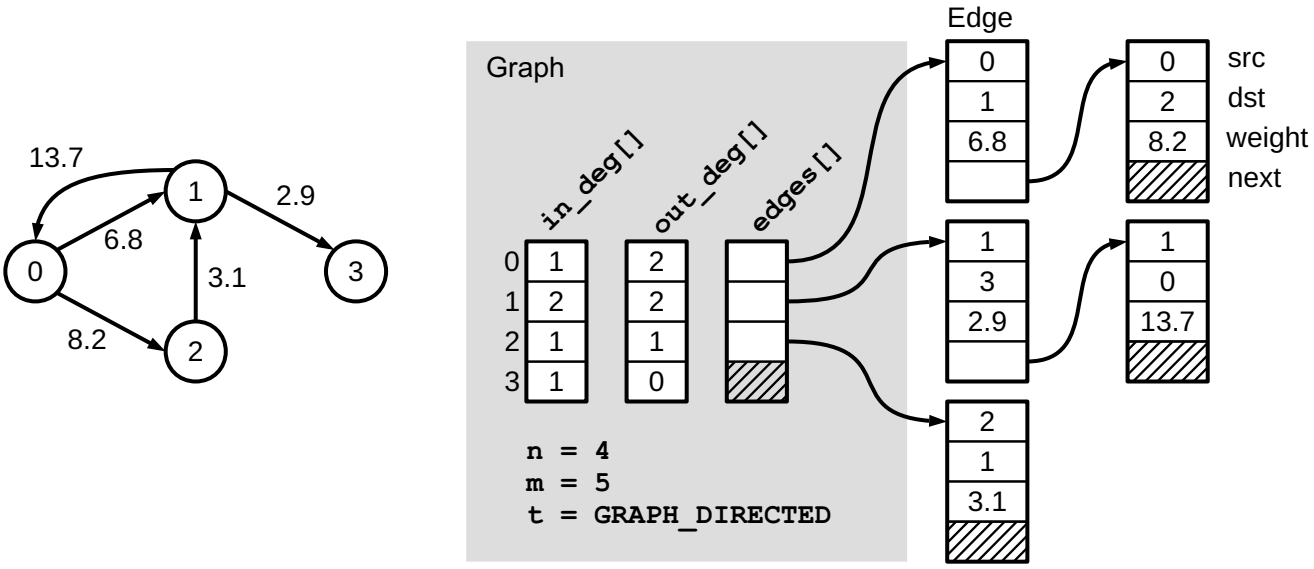


Figura 1: Rappresentazione di un grafo orientato

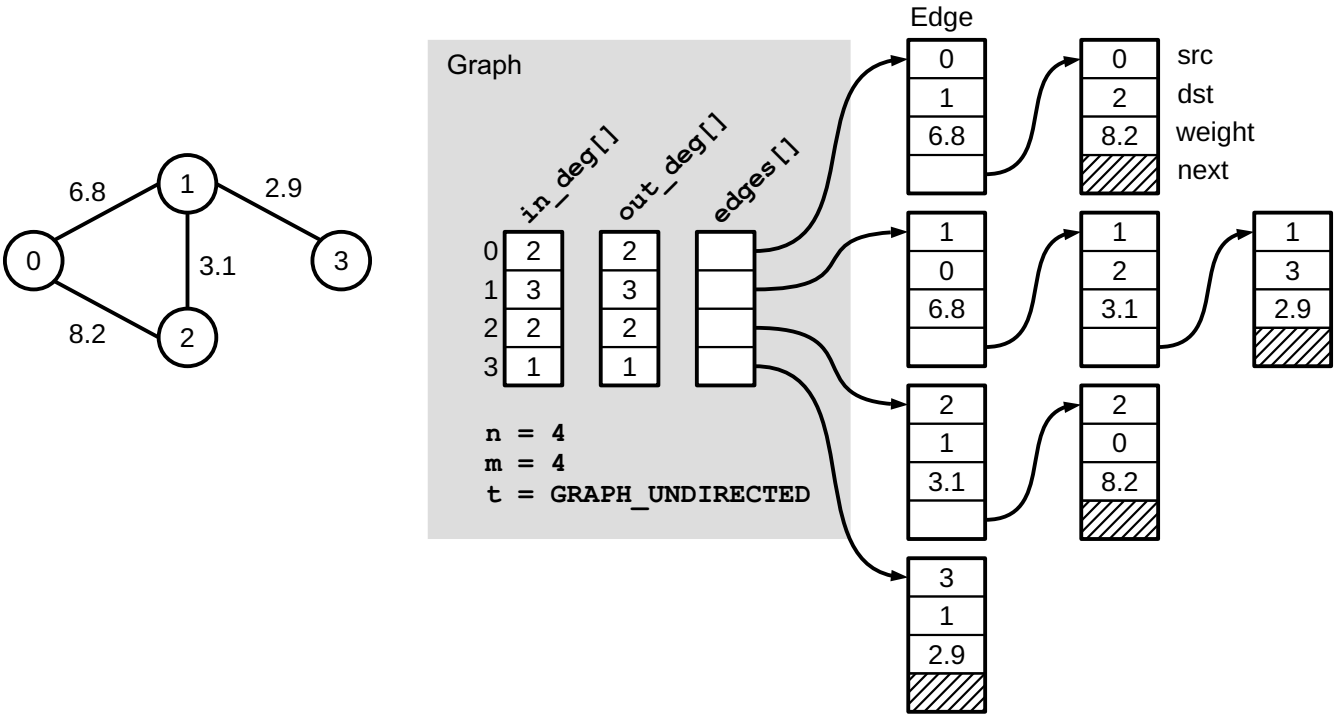


Figura 2: Rappresentazione di un grafo non orientato

L'elenco delle funzioni con la descrizione dei parametri e del comportamento atteso è presente nel file graph.h (graph.h).

Il file graph-main.c (graph-main.c) contiene una funzione `main()` che legge un grafo da file, il cui nome va specificato sulla riga di comando, e ne stampa il contenuto a video. Per rispondere ai quiz di autovalutazione sulla piattaforma "Virtuale", è possibile modificare il programma per stampare il grado entrante/uscente dei nodi con le funzioni `graph_in_degree()` oppure

`graph_out_degree()`.

Per compilare:

```
gcc -std=c99 -Wall -Wpedantic graph.c graph-main.c -o graph-main
```

Per eseguire:

```
./graph-main graph10.in
```

Formato di input/output

Le funzioni `graph_read_from_file()` e `graph_write_to_file()` utilizzano un semplice formato testuale per rappresentare un grafo:

```
n m type
s[0]    d[0]    w[0]
s[1]    d[1]    w[1]
...
s[m-1]  d[m-1]  w[m-1]
```

dove:

- *n* è il numero di nodi; i nodi sono rappresentati dagli interi $0, \ldots, (n-1)$;
- *m* è il numero di archi;
- *type* può valere 0 (grafo non orientato), oppure 1 (grafo orientato)
- $s[i], d[i], w[i]$ indicano, rispettivamente, il nodo sorgente, il nodo destinazione e il peso dell'arco i-esimo. I pesi sono valori reali, anche negativi.

File

- graph.c (graph.c)
- graph.h (graph.h)
- graph-main.c (graph-main.c)
- graph10.in (graph10.in) (output atteso (graph10.out); l'ordine con cui compaiono gli archi nelle liste di adiacenza è irrilevante)
- graph100.in (graph100.in)
- graph1000.in (graph1000.in)