13 Grafi

Per la definizione di grafo si rimanda agli appunti di Fondamenti dell'Informatica. Ricordiamo solo le caratteristiche principali:

- **V** rappresenta l'insieme di vertici
- **E** rappresenta l'insieme di archi
- Un grafo si dice **denso** quando $|E| \approx |V|^2$
- Un grafo si dice **sparso** quando $|E| \approx |V|$
- Un grafo può essere **orientato** o **non orientato**
- Il grafo è **pesato** quando viene associato un *peso* agli archi o ai nodi

13.1 Rappresentazione

Dato un grafo con n vertici tale che $V = \{1, 2, \ldots, n\}$ abbiamo due possibili modi di rappresentarlo.

13.1.1 Matrice di adiacenza

Una **matrice di adiacenza** rappresenta il grafo come una **matrice** A di dimensione $n \times n$. In questa matrice se un arco (i, j) esiste allora l'elemento $A[i, j]$ sarà valorizzato a 1, altrimenti sarà 0. Lo *spazio occupato* da questa matrice è $|V|^2$ il che la rende efficiente solo per grafi piccoli.

Note 13.1.1. *Si noti che se il grafo è* ***non orientato*** *serve solo metà matrice, in quanto* $(i, j) = (j, i)$.

13.1.2 Lista di adiacenza

Una **lista di adiacenza** rappresenta il grafo come un array di liste la cui dimensione dipende dal numero di archi presenti. Occupa $n + m$ spazio.

13.2 Ricerca in un grafo

Per cercare un elemento in un grafo dobbiamo esplorare ogni vertice e arco a partire da una sorgente s. Come risultato otteniamo un **albero** basato sul grafo iniziale che ha la sorgente come radice (o una foresta se il grafo non è connesso).

Definizione 13.1 (Percorso minimo)**.** *Un percorso minimo in un grafo G tra due vertici s, v è un percorso da s a v che contiene il minimo numero di archi. La sua lunghezza è detta* ***distanza minima*** *e viene indicata come $\delta(s, v)$.*
Una proprietà importante dato un arco tra due nodi (u, v) e un nodo v è la seguente:

$$\delta(s, v) \leq \delta(s, u) + 1$$

13.2.1 Breadth-First Search

Questo algoritmo esplora il grafo un vertice alla volta espandendo la frontiera dei vertici esplorati in **ampiezza**. Per evitare di attraversare più volte nodi già visitati associamo alcuni "colori" ad ogni nodo:

- **Bianco**: vertici non ancora visitati
- **Grigio**: vertici visitati ma non ancora esplorati
- **Nero**: vertici completamente esplorati

Il concetto quindi è di esplorare i vertici scorrendo le *liste di adiacenza* dei vertici grigi. Ogni vertice da esplorare viene aggiunto in una **queue** (coda) e appena vengono terminati e diventano neri vengono rimossi. Di seguito una possibile implementazione dell'algoritmo:

```
BFS(G, s) {
  inizializza vertici; // Vertici inizializzati ad ∞
  Q = {s}; // Q coda inizializzata con s e 0
  while (Q non vuota) {
    u = RemoveTop(Q);
    for each v ∈ u->adj {
      if (v->d == infinito)
      v->d = u->d + 1;
      v->p = u;
      Enqueue(Q, v);
    }
  }
}
```

Listing 28: Implementazione di BFS

La **complessità** in tempo di questo algoritmo è $O(V + E)$ mentre quella in spazio è $O(V)$.
L'algoritmo in questione genera un **albero breadth-first**, dove i cammini verso la radice rappresentano i cammini minimi nel grafo G.

Definizione 13.2 (Albero breadth-first)**.** *Dato un grafo $G = \langle V, E\rangle$ con sorgente s, un albero breadth-first è un albero $G' = \langle V', E'\rangle, V' \subseteq V, E' \subseteq E$ tale che:*

- *G' è un* ***sottografo*** *di G*
- *Un vertice v appartiene all'albero se e solo se quel vertice è* ***raggiungibile*** *dalla sorgente nel grafo G*
- *Per ogni vertice dell'albero il percorso dalla sorgente è* ***minimo***

13.2.2 Depth-First Search