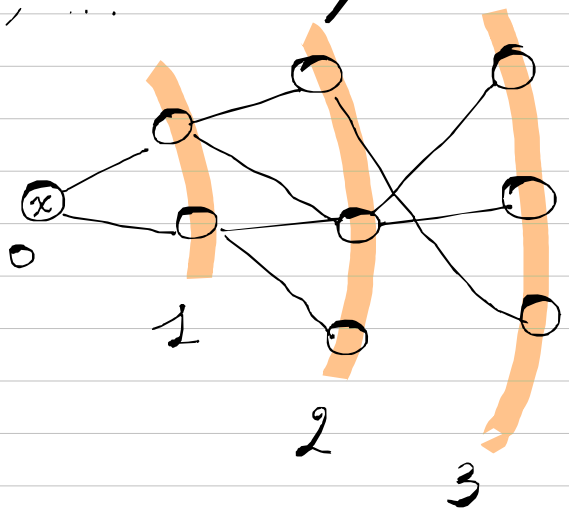
Breadth-first search

→ $d(x,y)$ = lunghezza minima del cammino da x a y

La BFS emula la visita a livello negli alberi, ossia visita prima — scelto uno starting node x — gli y con $d(x,y)=1$ poi quelli con $d(x,y)=2, \dots$



→ per realizzare la BFS si modifica la DFS per avere una queue (FIFO) invece che uno stack

Si definisce DISTANZA MEDIA il valore

$$\sum_{u \neq v} \frac{d(u,v)}{n(n-1)}$$

Il DIAMETRO invece è $\max_{u \neq v} d(u,v)$

$\rightsquigarrow \max_{u \neq v} d(u,v) = \max_{u} \left(\underbrace{\max_{v} d(u,v)}_{\text{tramite BFS}} \right)$

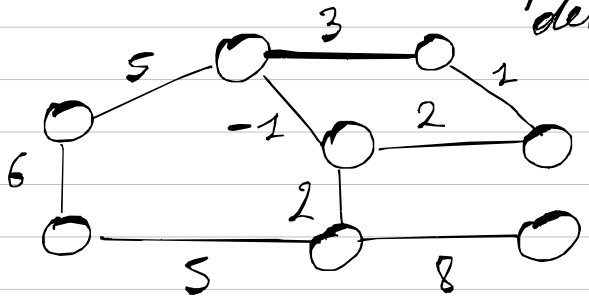
quindi ci vuole $O(n \cdot BFS) = O(n(n+m))$ tempo

Si dice che G è denso se $m \sim n^2$ e sparso se $m \sim n$

$\rightsquigarrow$ analogamente alla DFS, la BFS modella un albero detto BFS-tree che nei grafi non orientati induce una classificazione degli archi in archi del BFS-tree e non-tree tali che collegano nodi la cui diff. di livello è al più 1

Grafi pesati

$\rightsquigarrow G = (V, E, W)$ con $W : E \to \mathbb{R}$, $E \subseteq V \times V$ ↘ funzione dei pesi



Per i grafi pesati si costruisce la matrice di adiacenza come:

$$A_{ij} = \begin{cases} w(i,j) & (i,j) \in E, \\ 0 & \text{altrimenti.} \end{cases}$$

→ in particolare un grafo non pesato è un grafo con $w = 1$.

Anche le liste di adiacenza si adattano facilmente, diventando liste di vettori di coppie ord. in cui la prima coord. è la dest e la seconda è il peso dell'arco.

Si def. PESO DEL CAMMINO $v_1 v_2 \dots v_k$ il valore:

$$\sum_{i=1}^{k-1} w(v_i, v_{i+1}).$$

→ la BFS trova il cammino minimo pesato su grafi con $w = 1$ (o cost) — essa sui grafi non pesati.

Algoritmo di Dijkstra

→ algoritmo per calcolare le distanze pesate minime se $w \geq 0$ (altrimenti è ineff.).

Stesso idea del BFS, ma, al posto di una queue si usa una priority queue (heap min) — a ogni iterazione si aggiorna l'albero dei cammini minimi (SPT tree) usando le condizioni di Bellman:

sulle dist dalla radice!! (altrimenti MST)

$$d(i) + w(i,j) \geq d(j)$$

$$\forall \ i,j \text{ con } (i,j) \in E$$

→ Dijkstra funziona in $O(|E| \log |V|)$ (su costi positivi!!) ed è tale che (in qst cond.) — se i_k è il k-esimo nodo estratto e d_k registra le dist alla k-esima iterazione —

$$d_1(i_1) \leq d_2(i_2) \leq \ldots \leq d_n(i_n)$$

Infatti a ogni it. si sceglie il nodo più vicino al taglio che non gli appartenga (da cui la priority queue).