Breadth-First Search

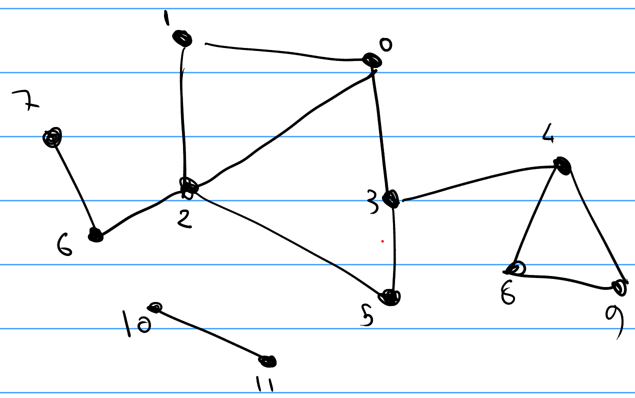
(Visita in ampiezza)

Strategia di esplorazione di un grafo (diretto o semplice)

più tardi vedremo anche un'altra strategia detta DFS.

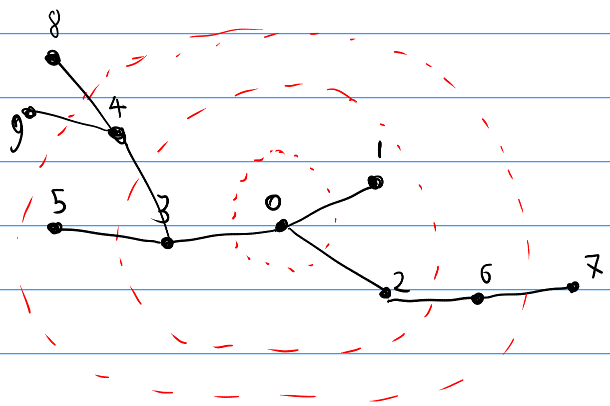
- Strategia BFS

1. Si parte da un vertice
2. si scoprono i suoi vicini
3. si prosegue la visita nei vertici scoperti nei passi precedenti



ESEMPIO QUI

Visita a partire dal vertice 0



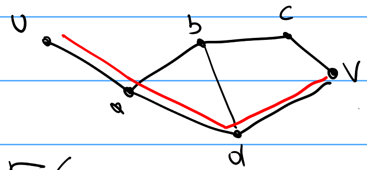
• <u>osservazione</u> Il vertice 2 è già scoperto quando abbiamo esplorato il vertice 0, quindi non dobbiamo rianalizzarlo come vicino del vertice 1 o del 5

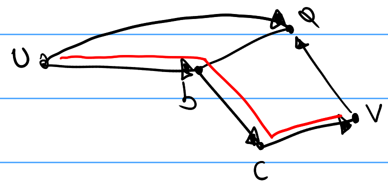
SCOPRIRE UN VERTICE DUE VOLTE È INUTILE

Definizione Dati $u,v \in V(G)$, definiamo $\delta(u,v)$

la DISTANZA di v da u, ovvero la LUNGHEZZA DEL CAMMINO PIÚ BREVE $u \rightsquigarrow v$

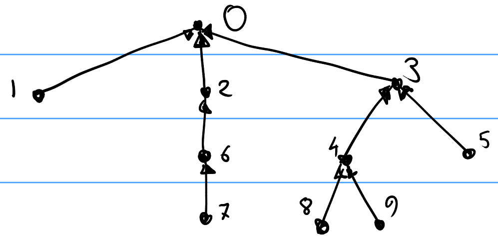
E.s.





Dalla BFS vogliamo

- cammini minimi
- componenti connesse
- alberi di visita



$\delta(0,v)=0$

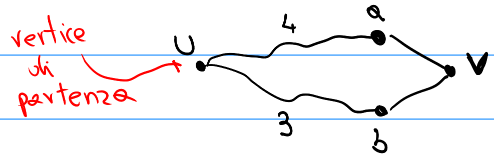
$\delta(0,v)=1$

$\delta(0,v)=2$

$\delta(0,v)=3$

$\delta(0,10)$ e $\delta(0,11)$ non é definita (o ∞ se vogliamo)

OSSERVAZIONE Supponiamo di avere un vertice v connesso a un vertice a distanza 5 e uno a distanza 3



allora v deve essere scoperto come vicino di b piuttosto che come vicino di a.

- I VICINATI DEI VERTICI A DISTANZA INFERIORE DEVONO ESSERE ESPLORATI PRIMA DEI VICINATI A DISTANZA MAGGIORE

Questo conduce ad un approccio FIFO (first in-first out) ovvero: i vicinati dei vertici vengono analizzati nell'ordine in cui vengono scoperti.

3

for v in G[u] → ciclo su tutti gli adiacenti di u

```python
def BFS(G,x):        # partenza della visita
    Q = deque()
    visited=[False]*len(G)

    Q.append(x)
    visited[x] = True
    while len(Q)>0:
        u = Q.popleft()
        for v in G[u]:
            if not visited[v]:
                Q.append(v)
                visited[v] = True
```

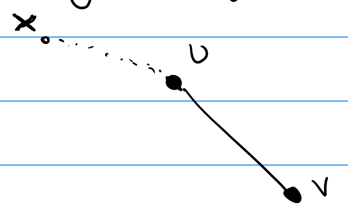
vertici messi nella coda quando vengono scoperti

i vertici vengono analizzati quando vengono estratti dalla coda

i vertici già scoperti (anche se non analizzati) non vengono inseriti nella coda

- Vediamo un esempio precalcolato:

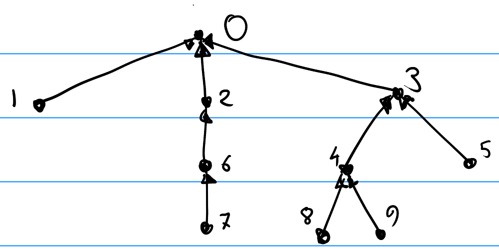
Aggiungiamo dei dettagli



Durante l'analisi del vertice u si scopre il vertice v

- $\delta(x,v) \leqslant \delta(x,u)+1$

- L'algoritmo AFFERMA che $\delta(x,v) = \delta(x,u)+1$ e nell'albero di visita BFS u è il genitore di v



distanza da 0
0
1
2
3

parent

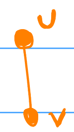
0	1	2	3	4	5	6	7	8	9	10	11
-	0	0	0	3	3	2	6	4	4	-	-

- In BFS e in DFS è utile differenziare
  - vertici sconosciuti - mai visti dall'algoritmo
  - vertici scoperti - trovati durante l'analisi del vicinato
  - vertici analizzati - analisi del vicinato completata


```python
def BFS(G,x):

    Q = deque(),
    P = [None]*len(G)
    dist = [inf]*len(G)
    color = ["white"]*len(G)

    dist[x]=0
    color[x] = "gray"
    Q.append(x)

    while len(Q)>0:
        u = Q.popleft()
        for v in G[u]:
            if color[v]=='white':
                color[v] = "gray"
                dist[v]=dist[u] + 1
                P[v] = u
                Q.append(v)
        color[u]="black"

```

INIZIALIZZAZIONE

LA CODA CONTIENE SOLO VERTICI GRIGI



u diventa il genitore di v nell'albero BFS

QUANDO UN VERTICE È STATO ANALIZZATO, DIVENTA NERO

$\boxed{\text{Vedi esempio precalcolato}}$

Albero di visita BF

La versione di BFS vista produce un albero di tutti i vertici raggiungibili dal vertice di partenza

Possiamo costruire una sequenza di alberi (i.e. una foresta) ognuno delle quali raccoglie tutti i vertici della corrispondente componente connessa

OSSERVAZIONE BFS(G, x) marca come NERI tutti i vertici visitati, e quelli non connessi a x rimangono bianchi

Esercizio: dimostrare questa affermazione

Possiamo trovare una nuova componente connessa prendendo un vertice bianco qualunque e chiamare BFS su di esso

```python
def BFStree(G):
    color = ["white"]*len(G)
    Q = deque(),
    P = [None]*len(G)
    dist = [inf]*len(G)

    for x in range(len(G)):
        if color[x]=='white':
            BFS(G,x,
                color, Q, P, dist)
```

VENGONO INIZIALIZZATI UNA VOLTA SOLA

VENGONO CONDIVISI E RIUTILIZZATI DA TUTTE LE CHIAMATE

Vedi ESEMPIO PRECALCOLATO

COMPLESSITÀ DI BFS

- Vediamo che l'algoritmo, dato un vertice, deve elencare tutti i vertici adiacenti

  questo suggerisce l'uso di LISTE di ADIACENZA

- Consideriamo $G = (V, E)$

Prop Un vertice entra nella coda al massimo una volta

dim solo i vertici bianchi sono presi in considerazione per entrare nella coda, e prima di essere inseriti, diventano grigi

Prop Ogni lista di adiacenza del grafo viene analizzata al più una volta, e per ogni vicino elencato nella lista di adiacenza si fanno un numero costante di operazioni

Per ogni vertice v_i

- $O(1)$ operazioni di entrata/uscita dalla coda
- $O(\deg(v_i))$ operazioni di analisi della lista di adiacenza

In totale $O(|V| + |E|)$ operazioni

$\sum_{v_i} \deg(v_i) =$
- $2|E|$ grafi semplici
- $|E|$ grafi diretti

BFS ha complessità $\Theta(|V|+|E|)$

perché Θ?
Non è difficile vedere che l'algoritmo legge tutto il grafo

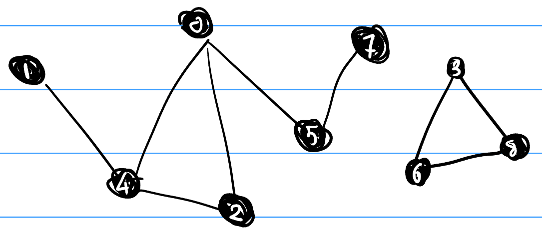
Esercizio: (es. 22.2-4)
Quale sarebbe la complessità nel caso venisse utilizzata la rappresentazione di G con MATRICI DI ADIACENZA

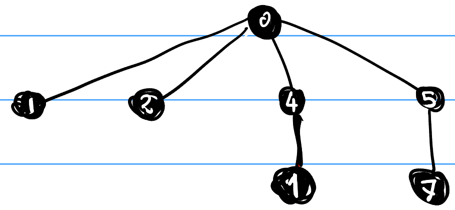
ESERCIZIO: (es. 22.2-5)
Osservate che l'esecuzione dell'algoritmo BFS(G,x) dipende dall'ordine con cui sono memorizzati i vicini nelle liste di adiacenza. Mostrare che

- L'albero della visita può cambiare
- Le distanze $\delta(x,v)$ sono le stesse indipendentemente da queste differenze nella rappresentazione (come è giusto che sia)

ESERCIZIO L'algoritmo BFS(G,x) produce un array P che rappresenta l'albero di visita BF da x.

E.g.





0	1	2	3	4	5	6	7	8
-	4	0	-	0	0	-	5	-

Trovate un algoritmo efficiente che dato P e un vertice v stampi

- se v è connesso a x, e in quel caso stampi un cammino $x \rightsquigarrow v$

ESERCIZIO Modificate l'algoritmo a pag. 5 in modo tale che BFS(G) produca un array C tale che

- dati C, u, v si possa determinare in tempo $O(1)$ se u e v sono nella stessa componente connessa

---

Correttezza dell'algoritmo BFS(G, x)

Thm Sia G un grafo semplice o un grafo diretto e sia $x \in V(G)$, allora durante la sua esecuzione

- $dist[v] = \delta(x, v)$ per ogni $v \in V(G)$
- per ogni v raggiungibile da x esiste un cammino minimo da x a v il cui ultimo arco è $(P[v], v)$

[cioè $dist[v] = \infty$ per i vertici irraggiungibili da x e uguale alla lunghezza del cammino minimo per quelli raggiungibili

(Questo è il teorema 22.5 nel libro di testo)