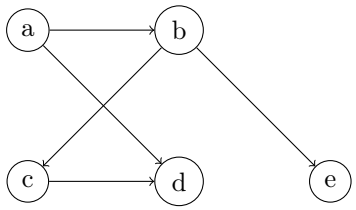
Giorno 7

1 Grafi

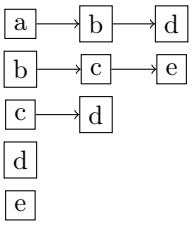
Un grafo è una coppia (V, E), dove V è un insieme di *nodi* o *vertici* ed $E \subseteq V \times V$ è un insieme di *archi*.

1.1 Rappresentazione dei grafi in memoria

I due principali metodi per rappresentare i grafi in memoria sono le liste di adiacenza e le matrici di adiacenza. Le prime sono un metodo più comune, poiché occupano meno spazio in memoria.



Grafo



Lista di adiacenza

	a	b	c	d	e
a	0	1	1	0	0
b	0	0	1	0	1
c	0	0	0	1	0
d	0	0	0	0	0
e	0	0	0	0	0

Matrice di adiacenza

1. Una lista di adiacenza ha complessità in spazio $\Theta(|V| + |E|)$
   - $|V|$ è la cardinalità dell'array che contiene i riferimenti alle liste d'adiacenza;
   - $\sum_{v \in V}$ gradoInUscita(v), che è uguale a $|E|$ per l'HandShaking Lemma, è la somma delle cardinalità delle liste.
2. Una matrice di adiacenza ha ovviamente complessità in spazio $\Theta(|V|^2)$

2 Visite di grafi

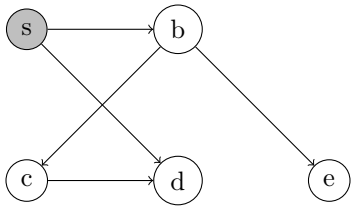
Una visita di un grafo è una procedura che, partendo da un nodo s detto *sorgente* accede a tutti i nodi del grafo (o a tutti quelli raggiungibili da s).

2.1 Visita per ampiezza - BFS

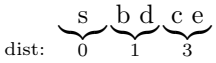
La BFS (breadth-first search) è una visita che accede a tutti i nodi raggiungibili dalla sorgente.

Def. 2.1.0.1. La distanza tra due nodi u e v è $\delta(u, v)$, uguale alla lunghezza del cammino minimo (cammino che attraversa il numero minimo di archi) tra i due nodi.

La caratteristica della BFS è che **l'ordine di visita dei nodi è dato dalla loro distanza da s**;
Si visitano prima tutti i nodi v tali che $\delta(s, v) = 0$ ($\implies v = s$), poi quelli a distanza 1, poi 2 e così via.



Assumendo di dare, a parità di distanza da s, priorità di visita ai nodi in ordine alfabetico, la BFS visita i nodi di questo grafo nell'ordine:



2.1.1 Implementazione della BFS

Ci sono varie possibili implementazioni, ma tutte hanno **un metodo per evitare di visitare archi già visitati**. Si noti, nell'esempio sopra, come il vertice d, che ha grado in entrata 2, è visitato solo una volta.
Questo risultato si può ottenere utilizzando un dizionario che tiene traccia di quali elementi sono già stati raggiunti, oppure si può utilizzare il *metodo da manuale*, che utilizza una colorazione dei nodi per determinare se questi sono stati già visitati o meno.

Ovviamente, dato che vogliamo complicarci la vita senza motivo, utilizzeremo il secondo di questi.

2.1.2 Colorazione dei nodi

Un nodo può essere *bianco*, *grigio* o *nero*.
I nodi *bianchi* non sono ancora stati scoperti dalla visita, quelli *grigi* sono già stati scoperti ma non sono ancora stati scoperti i loro eventuali discendenti, mentre i nodi *neri* sono quelli di cui sono stati scoperti gli eventuali discendenti.

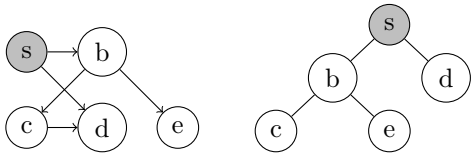
Non è necessario distinguere tra nodi grigi e neri, ma è utile[-citation needed] mantenere la distinzione.

Ad ogni nodo v si assegnano gli attributi $v.color$, $v.d$ (distanza da s), $v.p$ (puntatore al nodo da cui è scoperto v).

```
1  BFS(g, s):
2      Q = new coda;
3      for(v ∈ g.V):
4          v.color = white;
5          v.d = ∞;
6          v.p = nil;
7      s.d = 0;
8      s.color = gray;
9      enqueue(Q, s);
10     while(Q ≠ ∅):
11         y = dequeue(Q);
12         for(u ∈ Adj[y]):
13             if(u.color == white):
14                 u.d = y.d + 1;
15                 u.p = y;
16                 u.color = gray;
17                 enqueue(Q, u);
18         y.color = black;
```

2.1.3 Albero BF

La sequenza di scoperte della BFS può essere rappresentata in un albero, detto albero BF. Questo albero è tale che $v.p$ sia il padre di p, per ogni v.



Dato il grafo a sinistra, l'albero BF associato è quello a destra.

2.1.4 Cammini minimi

Una proprietà importante della BFS è che per ogni v raggiungibile da s, risalendo i riferimenti al nodo scopritore ("padre"): $v.p$, $v.p.p \dots s$ si ottiene un cammino minimo tra v e s.
Questo si dimostra per induzione sulla distanza da s.

Per stampare un cammino minimo tra s e v, dopo aver eseguito la BFS, si usa la seguente procedura:

```
1  PrintPath(v, s):
2      u = v;
3      L = new lista;
4      while(u.p ≠ nil):
5          insert(L, u);
6          u = u.p;
7      return L;
```

2.1.5 Complessità

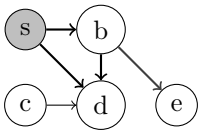
- L'inizializzazione della BFS ha costo $\Theta(|V|)$
- Ogni nodo è inserito e rimosso nella e dalla coda al più una volta, quindi il while viene eseguito al più $|V|$ volte. In ognuna di queste iterazioni si ispeziona ogni elemento della lista di adiacenza di y, quindi sempre per l'HandShaking lemma il costo del while è $O(|E)|$

Quindi BFS ha complessità in tempo $O(|V| + |E|)$, e se il grafo è connesso $\Theta(|V| + |E|)$

2.1.6 Correttezza della BFS

Si osservi la seguente proprietà:

Proposizione 2.1.6.1. Per ogni arco $(u, v) \in E$ vale che $\delta(s, v) \leq \delta(s, u) + 1$



Arco (b, d): $1 = \delta(s, d) \leq \delta(s, b) + 1 = 2$
Arco (b, e): $2 = \delta(s, e) \leq \delta(s, b) + 1 = 2$

Si dimostra anzitutto un limite superiore:

Teorema 2.1.6.2. Al termine di BFS(g, s), per ogni vertice $v \in g.V$ vale che $v.d \geq \delta(s, v)$

Dimostrazione. Induzione sul numero di enqueue:
Hp. $v.d \geq \delta(s, v)$ per ogni $v \in V$.

1. Dopo la prima operazione enqueue, l'Hp. è vera.

$$s.d = 0 = \delta(s, s) \qquad \forall v \in V : v.d = \infty \geq \delta(s, v)$$

2. Sia v un vertice bianco scoperto a partire da u.
   Hp: $u.d \geq \delta(s, u)$.
   L'algoritmo pone $v.d = u.d + 1$.

$$v.d = u.d + 1 \underbrace{\geq}_{Hp} \delta(s, u) + 1 \underbrace{\geq}_{2.1.6.1} \delta(s, v)$$

■

[...]

2.2 Visita in profondità - DFS

La DFS (depth-first search) è una visita che raggiunge tutti i nodi del grafo. Una descrizione informale di DFS è:

DFS(g):

- Per ogni nodo $v \in g.V$ non già scoperto esegui DFS-Visit(v);
- DFS-Visit(s):
  (a) Scopri s;
  (b) Per ogni nodo u di $Adj[s]$:
  Se u non è già stato scoperto esegui DFS-Visit(u);

2.2.1 Implementazione della DFS

Per modellare l'essere o meno scoperto di un nodo useremo di nuovo i colori.
Gli attributi dei nodi sono, per ogni $v \in V$:

- $v.color$;
- $v.p$, che punta al nodo che ha scoperto v;
- $v.d$, da non confondersi con la distanza della BFS, è il *tempo* di scoperta del nodo v;
- $v.f$ è il tempo di fine visita del nodo v.

Si usano i **tempi** di inizio e fine visita per formulare alcune proprietà della DFS. Come tempo si usa una variabile globale `time` inizializzata a zero, che viene incrementata prima di ogni scoperta e di ogni fine visita.

```
1  DFS-Visit(s):
2      time++;
3      s.d = time;
4      s.color = gray;
5      for(u ∈ Adj[s]):
6          if(u.color == white):
7              u.p = s;
8              DFS-Visit(u);
9      time++;
10     s.f = time;
11     s.color = black;
```

```
1  DFS(g):
2      for(v ∈ g.V):
3          v.color = white;
4          v.p = nil;
5      for(v ∈ g.V):
6          if(v.color == white):
7              DFS-Visit(v);
```

Costo di DFS-Visit: $O(\text{gradoInUscita}(s))$
Costo di DFS: $O(|V| + |E|)$

Come per la BFS si poteva definire un albero BF, per la DFS si può definire analogamente un albero DF.

2.2.2 Proprietà della DFS

Teorema 2.2.2.1 (Delle Parentesi)**.** Dati due nodi u, v, siano gli intervalli $I_u = [u.d, u.f]$, $I_v = [v.d, v.f]$. Si può verificare solo uno tra tre casi:

1. $I_u \subset I_v$, in tal caso u è discendente di v nell'albero DF.
2. $I_v \subset I_u$, in tal caso v è discendente di u nell'albero DF.
3. I due intervali sono disgiunti, in tal caso i due nodi non sono "parenti".

Dimostrazione. Supponiamo *w.l.o.g.* che $u.d < v.d$. Ci sono due casi:

1. Se v è scoperto prima della fine della visita di u, quindi $I_v \subset I_u$, allora u era grigio durante la scoperta di v, e tutti i nodi scoperti mentre u era grigio ne sono discendenti.
2. Se v è stato scoperto dopo la fine della visita di u, cioè se i due intervalli sono disgiunti, allora u era nero, quindi v non può esserne discendente.

■

Teorema 2.2.2.2 (Del Cammino Bianco)**.** Dati due nodi u e v: v è discendente di u se e solo se al momento della scoperta di u esiste un cammino di nodi bianchi tra u e v.

Dimostrazione.

($\Longrightarrow$) Se v è discendente di u:

- $v = u \implies v \rightsquigarrow u$ contiene solo v, che è bianco nel momento in cui è scoperto.
- $v \neq u$, v generico discendente di u, si ha $I_v \subset I_u \implies u.d < v.d \implies v$ è bianco al tempo $u.d$.

($\Longleftarrow$) Esiste un cammino B di nodi bianchi tra u e v.

Assumiamo per assurdo che v sia il primo vertice sul cammino B non discendente di u.

Sia w il predecessore di v, sia quindi w discendente di u. Allora $I_w \subset I_u$, cioè $u.d < w.d < w.f < u.f$

Inoltre $v \in Adj[w]$, quindi la visita di v finisce prima di quella di w. Quindi si ha che:

$$u.d < w.d < v.f < w.f < u.f$$

Per il teorema delle parentesi due intervalli non possono avere elementi comuni senza essere l'uno incluso nell'altro, perciò si ha che $I_v \subset I_u$, quindi v è discendente di u. ($\lightning$)

■

2.3 Classificazione degli archi

Un arco di un grafo può essere:

1. **Arco d'albero**, se compare nell'albero DF;
2. **Arco all'indietro**, se collega un nodo dell'albero DF ad un suo predecessore;
3. **Arco in avanti**, se collega un nodo dell'albero DF ad un suo successore;
4. **Arco trasversale**, se collega un nodo dell'albero DF con un altro che non ne è né predecessore né successore.

In particolare, se facendo la DFS si incontra un nodo *grigio*, si ha un arco all'indietro, se se ne incontra uno *bianco* si ha un arco d'albero, se se ne incontra uno *nero* si può avere sia un arco in avanti che uno trasversale.

Teorema 2.3.0.1. In un grafo non orientato ci sono solo archi d'albero ed archi all'indietro.

Dimostrazione. Quando si scopre un nuovo nodo, può essere già stato esplorato o meno. Se è stato esplorato allora si ha un arco all'indietro, se no si ha un arco d'albero. ■

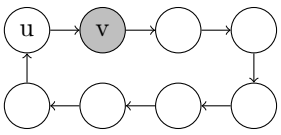
Teorema 2.3.0.2 (Caratterizzazione dei grafi ciclici)**.** Un grafo è ciclico se e solo se presenta archi all'indietro.

Dimostrazione.

($\Longleftarrow$) Se esiste un arco che porta da un nodo v ad un suo predecessore nel'albero DF, allora c'è banalmente un ciclo.

($\Longrightarrow$) Esiste almeno un ciclo.

- Se il grafo non è orientato, allora l'arco che chiude il ciclo non può essere d'albero, quindi è necessariamente all'indietro.
- Se il grafo è orientato:
  Sia v il primo vertice del ciclo ad essere stato scoperto (a diventare grigio). Sia u il vertice che precede v nel ciclo. Al tempo $v.d$, tutti i vertici del ciclo sono bianchi.
  Per il teorema del cammino bianco, u è discendente di v, quindi l'arco (u, v) è un arco all'indietro.



■

2.3.1 Ordinamento topologico

Se si applica DFS ad un DAG e inseriscono i nodi in una lista alla fine della loro visita si ottiene un ordinamento topologico, ossia un ordinamento dei nodi tale che $u \prec v$ solo se non esiste l'arco (v, u).

Dimostrazione. Dobbiamo dimostrare che per ogni $(u, v) \in E$ l'arco (u, v) rispetta l'ordinamento, ossia che l'elemento u è inserito dopo (=viene prima) nella lista rispetto all'elemento v, ossia che $v.f < u.f$:

1. (u, v) può essere un arco d'albero, in tal caso v è successore di u, quindi $v.f < u.f$.
2. (u, v) può essere un arco in avanti; anche in tal caso v è successore di u, quindi $v.f < u.f$.
3. (u, v) non può essere un arco all'indietro poiché il grafo è aciclico!

Quindi la proposizione è dimostrata. ∎