Progettazione di algoritmi

Angelo Monti

Lezione 8:
Esempi di applicazione della BFS.

Lezione 8: Sommario

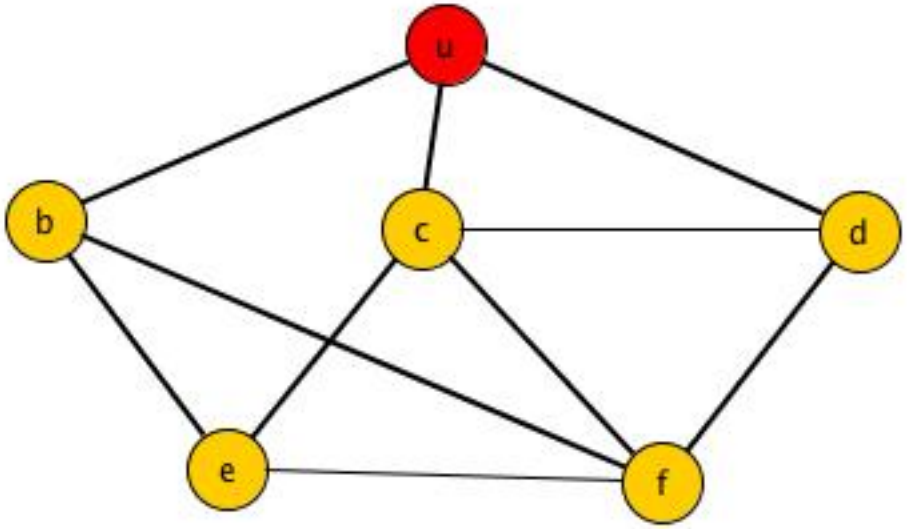
Esempi di applicazione della BFS:

- Contare il numero di cammini minimi
- Trovare la distanza tra due insiemi
- Trovare nodi ad uguale distanza
- Trovare cammini vincolati a passare per determinati nodi
- Trovare le distanze a partire dall'albero dei padri prodotto dalla BFS
- Uguaglianza tra alberi DFS e BFS

ESERCIZIO 1: Numero di cammini minimi. Dato un grafo G ed un suo nodo u, vogliamo sapere per ogni nodo v in G con $v \neq u$ il numero $M[v]$ di cammini minimi che da u porta a v.

- Un nodo x a distanza 1 da u può avere un solo cammino minimo. Ma per un nodo x a distanza 2 possono esserci più cammini minimi.

Ad esempio nel grafo qui sotto il nodo e ha 2 cammini minimi dal nodo u ((u,b,e) e (u,c,e)) mentre f ne ha 3 ((u,b,f), (u,c,f) e (u,d,f)).



- In generale, per un nodo w a distanza k dal nodo di partenza s come possiamo calcolare il numero di cammini minimi?
- Un qualsiasi cammino di lunghezza k da u a w passa per un nodo v (il predecessore di w) che è a distanza $k-1$ da s.
- Se il numero di cammini minimi per v è $M(v)$, allora se ad ognuno di tali cammini aggiungiamo l'arco (v, w) otteniamo $M(v)$ cammini minimi per w.
- Questo ragionamento è valido per un qualsiasi nodo a distanza $k-1$ da u che è adiacente a w. Quindi il numero di cammini minimi di w da u è uguale alla somma $M(v_1) + \ldots + M(v_h)$, dove $v_1, \ldots, v_h$ sono tutti i nodi a distanza $k-1$ che sono adiacenti a w.
- Per implementare tale conteggio basterà modificare la BFS in modo tale da mantenere in un array M per ogni nodo visitato w il numero dei cammini minimi finora trovati $M[w]$ e un array $Dist$ per le distanze.
  - Quando un nodo w è visitato per la prima volta tramite un nodo v, si porrà $M[w] \leftarrow M[v]$ e $Dist[w] \leftarrow Dist[v] + 1$.
  - Poi, ogni volta che si incontra un nodo v' adiacente a w e tale che $Dist[w] = Dist[v'] + 1$, si aggiorna il numero di cammini minimi di w, $M[w] \leftarrow M[w] + M[v']$.

- ecco quindi lo pseudocodice (in rosso le modifiche apportate alla BFS):

$M[1 \ldots n]$ **vettore per il conteggio dei cammini minimi inizializzato a zero**
$Dist[1 \ldots n]$ **vettore delle distanze inizializzato a** $+\infty$
$Dist[s] \leftarrow 0$
$M[s] \leftarrow 1$
$Q \leftarrow$ **coda vuota**
inserisci s **in coda** Q**.**
WHILE $Q \neq \emptyset$ **DO**
    $u \leftarrow$ **estrai nodo dalla coda Q**
    FOR ogni $v \in adj(u)$ **DO**
        IF $(Dist[v] = +\infty)$ **THEN**
            $Dist[v] \leftarrow Dist[u] + 1$
            $M[v] \leftarrow M[u]$
            inserisci v **in coda** Q
        ELSE
            IF $(Dist[v] = Dist[u] + 1)$ **THEN**
                $M[v] \leftarrow M[v] + M[u]$
    ENDFOR
ENDWHILE

- La complessità dell'algoritmo è pari a quella della BFS, cioè $O(n + m)$. Dopo aver ottenuto l'array M, per ogni nodo v, $M[v]$ dà il numero di cammini minimi da s a v.

ESERCIZIO 2: Stessa distanza. Descrivere un algoritmo che, dato un grafo G non diretto e connesso e due suoi nodi u e v, in tempo $O(n+m)$ trova i nodi che hanno la stessa distanza da u e v.

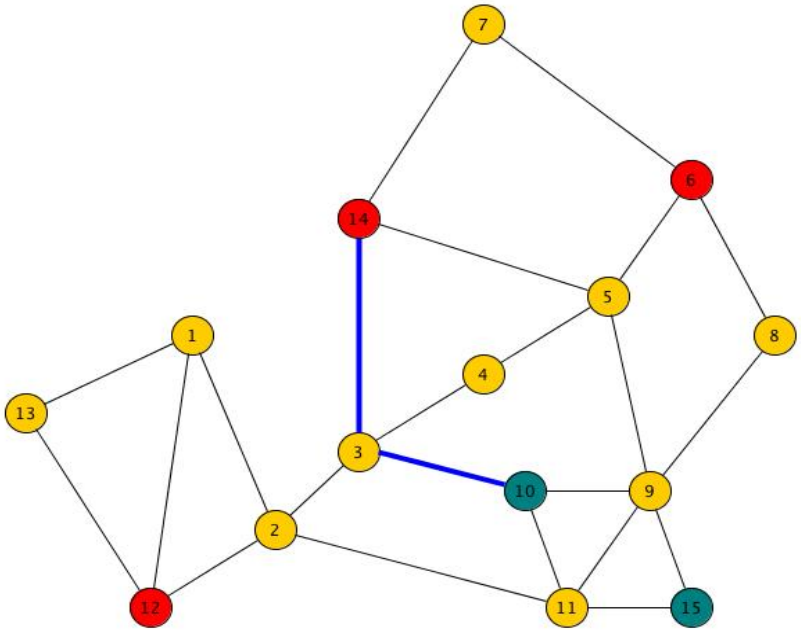
- Per determinare i nodi a uguale distanza dai nodi u e v basterà fare una BFS a partire da u e una BFS a partire da v per ottenere con la prima visita il vettore $Dist_u$ delle distanze da u e con la seconda visita il vettore $Dist_v$ delle distanze da v. A questo punto, per ogni nodo w, si controlla se $Dist_u[w]$ e $Dist_v[w]$ coincidono.

```
SameDist(G: grafo, u, v: nodi)
        Dist_u ← BFS(G, u)
        Dist_v ← BFS(G, v)
        SOL ← ∅
        FOR ogni nodo w DO
                IF (Dist_u[w] = Dist_v[w]) THEN
                        SOL ← SOL ∪ {w}
        ENDFOR
        RETURN SOL
```

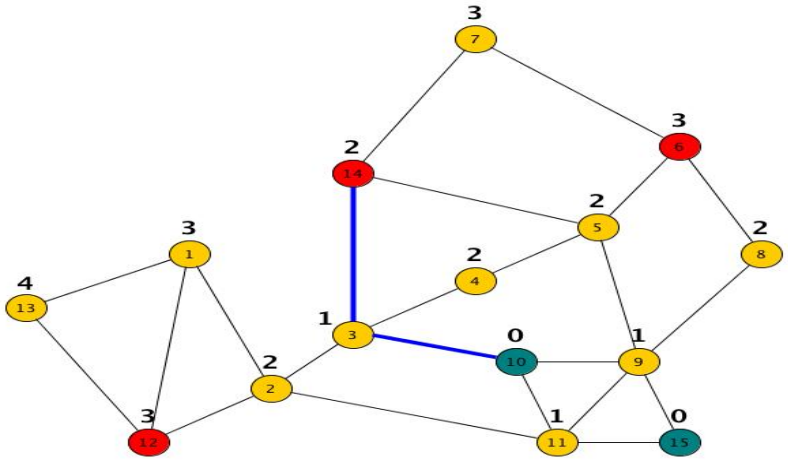
- L'algoritmo restituisce l'insieme SOL con i nodi a uguale distanza da u e v ed ha complessità $O(n+m)$.

ESERCIZIO 3: Distanza tra insiemi. Dato un grafo non diretto G e due sottoinsiemi A e B dei suoi nodi si definisce distanza tra A e B la distanza minima per andare da un nodo in A ad un nodo in B. Se A e B non sono disgiunti, la loro distanza è 0.
Descrivere un algoritmo che, dato G e due sottoinsiemi dei suoi nodi A e B calcola la loro distanza. L'algoritmo deve avere complessità $O(n+m)$.

- Ad esempio per il grafo G in figura, dove i nodi dell'insieme A sono in verde mentre i nodi dell'insieme B sono in rosso, la distanza tra i due insiemi è 2 come evidenziato dal cammino in blu.



- Per determinare le distanze da un insieme di nodi A, piuttosto che da un nodo solo, si può pensare ad una naturale estensione della BFS. Invece di partire da un singolo nodo, la BFS inizia da tutti i nodi dell'insieme A simultaneamente. Così ogni nodo in A avrà la distanza inizializzata a 0. Poi si considera ogni adiacente v di questi ultimi, se v non è già stato visitato, la sua distanza sarà 1. Poi si considerano gli adiacenti dei nodi a distanza 1, quelli non ancora visitati avranno distanza 2, e così via, proprio come nella normale BFS.
- Nel grafo in figura sono riportate le distanze di tutti i nodi del grafo dai nodi dell'insieme A.



- Disponendo del vettore $Dist_A$, dove $Dist_A[u]$ è la distanza del nodo u dall'insieme A. Sarà facile calcolare la distanza dell'insieme B dall'insieme A. Basterà trovare il nodo w di B per cui $Dist_A[w]$ risulta minima.

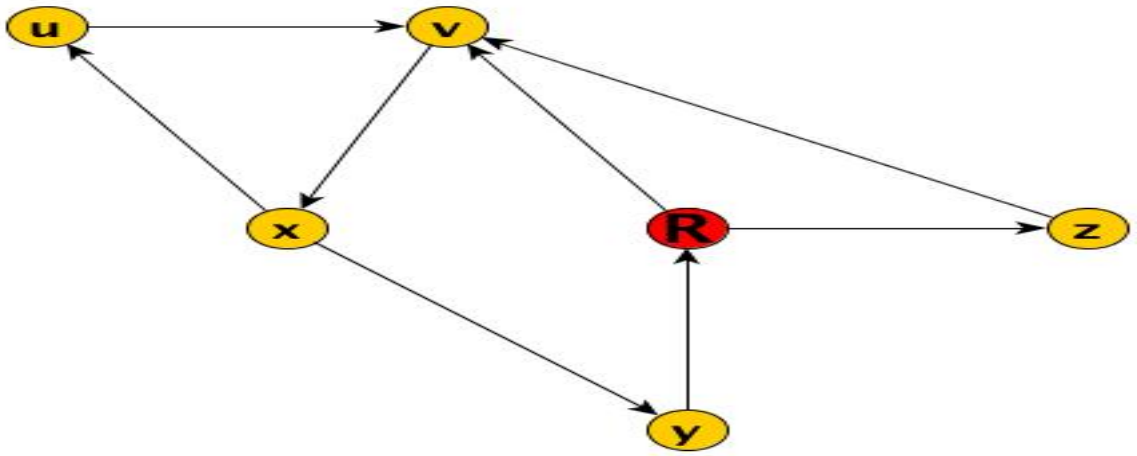
```
DIST(G: grafo, A, B: insiemi di nodi)
        Dist_A ← BFS_SET(G, A)
        d ← +∞
        FOR ogni nodo u in B DO
                IF (Dist_A[u] < d) THEN d ← Dist_A[u]
        ENDFOR
        RETURN d
```

```
BFS_SET(G: grafo, A: insieme di nodi)
    Dist_A ← array delle distanze dall'insieme A, inizializzato a +∞
    Q ← coda vuota
    FOR ogni nodo u in A DO
            Dist_A[u] ← 0
            inserisci u in coda
    ENDFOR
    WHILE Q ≠ ∅ DO
            u ← estrai nodo dalla coda Q
            FOR ogni adiacente v ∈ adj(u) DO
                    IF (Dist_A[v] = ∞) THEN
                            Dist_A[v] ← Dist_A[u] + 1
                            inserisci v in coda Q
            ENDFOR
    ENDWHILE
    RETURN Dist_A
```

- **Il tempo richiesto dall'esecuzione di BFS_SET è $O(n+m)$. Infatti:**
  - **l'inizializzazione di $Dist\_A$ e della coda Q prendono tempo al più lineare in n e poi la visita procede come una normale BFS di costo $O(n+m)$ (anche se la visita parte da un insieme di nodi invece che da un unico nodo).**
- **Una volta ottenuto il vettore $Dist\_A$ delle distanze da A l'esecuzione del FOR richiede $O(n)$.**
- **Quindi l'algoritmo ha complessità $O(n+m)$.**

ESERCIZIO 4: Roma. Descrivere un algoritmo che, dato un grafo diretto e fortemente connesso G ed un suo nodo r, trova tutti i cammini minimi tra tutte le coppie di nodi con il vincolo che questi cammini devono passare per r.

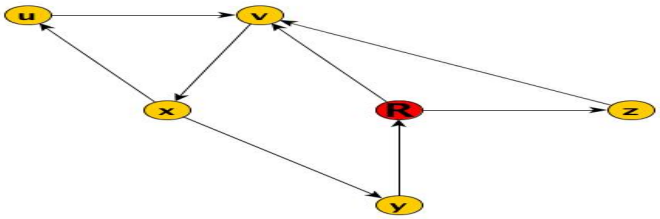
- Ad esempio per il grafo in figura si ha che:
  - il cammino minimo da y a z è y, r, z.
  - il cammino minimo da u a v è u, v, x, y, r, v.



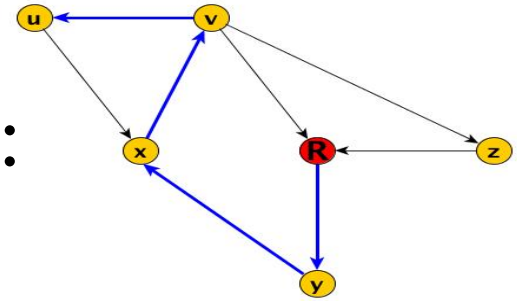
- Si noti che il cammino da una certa sorgente ad una data destinazione, non è necessariamente semplice, cioè può passare due volte per lo stesso nodo (come nel caso del cammino da u a v).

- Nota che il cammino più breve da u a v che passa per r in G è dato dal cammino più breve che passa per u e va a r concatenato al cammino più breve che da r va a v.
- In $O(n+m)$ tempo, con due visite BFS (la prima con nodo sorgente u e la seconda con nodo sorgente r) è possibile dunque calcolare il cammino più breve da u a v che passa per r.
- per calcolare poi il cammino più breve da u' a v' che passa per r avrei bisogno di una terza visita BFS (con sorgente u') mentre l'informazione che mi serve per il cammino più breve da r a v' la ho già calcolata grazie alla seconda visita BFS che usava proprio r come sorgente.
- Dovendo calcolare le informazioni necessarie a trovare cammini minimi tra qualunque coppia di nodi e volendo effettuare **soltanto** due visite posso utilizzare la seguente osservazione:
  - Il cammino più breve da u a r nel grafo G è (a parte la direzione) il cammino più breve da r ad u nel grafo G^T con gli archi invertiti (cioè, il grafo trasposto)
- Dunque per trovare questo cammino minimo vincolato a passare per r, possiamo trovare un cammino minimo C_u da r a u nel grafo trasposto G^T e poi un cammino minimo C_v da r a v (nel grafo originale G). La concatenazione di C_u e C_v ci dà il cammino minimo da u a v che passa per r.
- Possiamo dunque risolvere il problema grazie alle due sole visite BFS, entrambe con sorgente r ma una sul grafo G^T e l'altra sul grafo G.

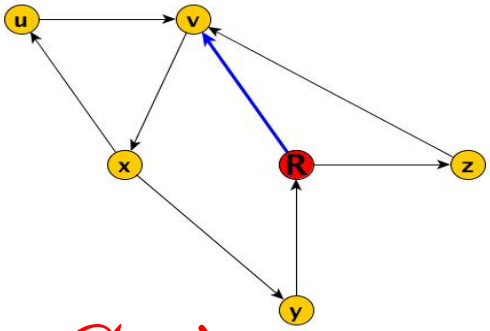
- Alla ricerca del cammino da u a v che passa dal nodo r nel grafo G:
  - calcola il cammino C_u in G^T
  - calcola il cammino C_v in G
  - concatena i due cammini C_u e C_v.



- il cammino C_u in G^T:



- il cammino C_v in G:



- il cammino C_u concatenato C_v è u, x, y, r, v

- Allora, possiamo calcolarci come prima cosa l' albero dei cammini minimi di G radicato in r e poi, in seconda battuta, l'albero dei cammini minimi di G^T radicato in r

$P \leftarrow \text{BFS}(G, r)$ /* Vettore P dei padri dell'albero dei cammini minimi di G da r */
$GT \leftarrow \text{TRASP}(G)$ /* Ritorna il grafo trasposto */
$PT \leftarrow \text{BFS}(GT, r)$ /* Vettore PT dei padri dell'albero dei cammini minimi di GT da r */
RETURN P, PT

- La complessità dell'algoritmo è $O(n + m)$ dato che le due BFS prendono tempo $O(n + m)$ e anche la procedura per il calcolo del grafo trasposto prende tempo $O(n + m)$.

- dati due qualsiasi nodi u e v possiamo costruire il cammino minimo da u a v che passa per r nel seguente modo:

```
C_u ← lista vuota
DO
        inserisci u in coda alla lista C_u
        u ← PT[u]
WHILE u ≠ r
C_v ← lista vuota
WHILE v ≠ r DO
        inserisci v in testa alla lista C_v
        v ← P[v]
ENDWHILE
C ← concatenazione di C_u e C_v
OUTPUT C
```

- Quindi una volta calcolati i vettori dei padri P e PT, un cammino minimo da u a v vincolato a passare per r si può ottenere in tempo lineare nella lunghezza del cammino.

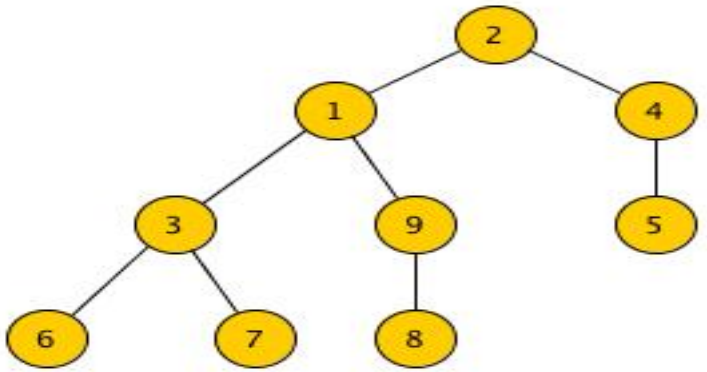
ESERCIZIO 5: **Dato un vettore dei padri P che rappresenta l'albero di una BFS a partire da un nodo u, dare un algoritmo che calcola il corrispondente vettore $Dist$ delle distanze da u in tempo $O(n)$.**

Ad esempio:

Per $P =$

2	2	1	2	4	3	3	9	1

si ha $u = 2$ **e l'albero dei cammini minimi è**



L'algoritmo deve restituire il vettore

$Dist =$

1	0	2	1	2	3	3	3	2

Sia v un nodo qualsiasi, possiamo calcolare la distanza da u tramite il vettore dei padri P come segue:

$d \leftarrow 0$
$w \leftarrow v$
WHILE $P[w] \neq w$ **DO**
    $d \leftarrow d + 1$
    $w \leftarrow P[w]$
ENDWHILE
OUTPUT d

È chiaro che questo richiede tempo lineare nella distanza d, cioè $O(d)$. Se lo ripetiamo per tutti i nodi avremo un algoritmo che richiede tempo lineare nella somma di tutte le distanze.
In generale, tale algoritmo ha una complessità superiore a $O(n)$. Ad esempio, se il grafo è un cammino e quindi anche l'albero rappresentato da P è un cammino, la complessità è $O(n^2)$.

Ma se nell'effettuare il calcolo delle distanze non ricalcoliamo le distanze che abbiamo già calcolato, potremmo risparmiare parecchio tempo.

```
Dist : vettore delle distanze inizializzato a −1
FOR ogni nodo w DO
        IF (Dist[w] = −1) THEN Dist[w] ← DIST(w)
ENDFOR
RETURN Dist

DIST(v)
      IF (P[v] = v) THEN RETURN 0
      IF (Dist[P[w]] ≠ −1) THEN RETURN Dist[P[w]] + 1
       h ← DIST(P[w])
      RETURN h + 1
```

La funzione DIST(v) calcola la distanza di v solo se non è già stata calcolata e sfrutta le distanze già calcolate. Quindi il numero totale di chiamate ricorsive effettuate in tutte le chiamate a DIST della funzione DISTANZE non può superare il numero di nodi. Questo proprio perché per ogni nodo sarà effettuata la chiamata ricorsiva solo la prima volta, quando la sua distanza non è stata ancora calcolata. Siccome ogni chiamata a $DIST$ costa tempo costante, escluso il tempo preso dall'eventuale chiamata ricorsiva, l'algoritmo ha complessità lineare nel numero di nodi, cioè $O(n)$.

ESERCIZIO 6: Dimostrare che perché le visite DFS e BFS producano alberi uguali il grafo G visitato deve essere è un albero.

Prova (per assurdo)

Sia G un grafo tale che, partendo da un nodo u, DFS e BFS producono la stesso albero di visita T radicato in u.

- Supponiamo che G non sia un albero, deve dunque esistere in G un arco $e = \{v, w\}$ che non appartiene a T.
- Sappiamo che nella DFS e sarà classificato come arco all'indietro, cioè v è un antenato di w in T.
- Per ipotesi T è anche l'albero di visita della BFS e siccome v è un antenato di w e l'arco $\{v, w\}$ non appartiene a T, deve essere $d(u, w) \geq d(u, v) + 2$.
- Ma questo è impossibile perché w è adiacente a v e quindi deve essere $d(u, w) \leq d(u, v) + 1$.

Perciò concludiamo che un arco non appartenente a T non può esistere. Quindi i due alberi di visita sono uguali solo quando il grafo G è un albero.