BFS & DFS

corso di laurea in **Matematica**

Informatica Generale, Esercitazione 8

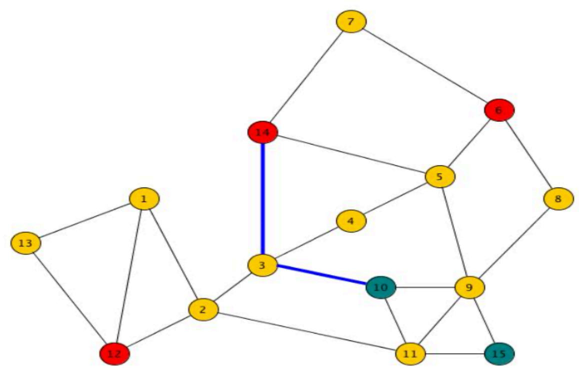
Ivano Salvo

SAPIENZA UNIVERSITÀ DI ROMA

Esercitazione 8, Lezione 22bis

1. Dato un grafo $G = (V, E)$ non orientato e connesso e due insiemi di nodi $W, Z \subseteq V$ calcolare la distanza tra W e Z, dove la distanza tra due insiemi di nodi è data da $\min_{w \in W, z \in Z} d(w, z)$. Una buona soluzione dovrebbe avere complessità $\Theta(n+m)$.

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.



2. [Scritto 19 settembre 2021] Dato un grafo G non orientato connesso restituire una passeggiata (come lista di nodi) che attraversa tutti gli archi di G esattamente 2 volte. Ciascun arco verrà percorso una volta in una direzione e l'altra volta nella direzione opposta.

SUGG: risolvere lo stesso problema con un albero binario modificando la visita in profondità. E poi applicare ai grafi.

Cosa dovete fare

1. Proporre una struttura dati per memorizzare il grafo $G = (V, E)$ e l'insieme dei nodi verdi e rossi.
2. Utilizzando una funzione *dist*(grafo G, nodo s) che restituisce un vettore indicizzato sui nodi con tutte le distanze (in numero di archi) dal nodo s, scrivere un algoritmo che calcola la distanza tra gli insiemi V_1 e V_2.
3. Assumendo che dist abbia complessità $\Theta(m+n)$, con $n = |V|$ e $m = |E|$, calcolare la complessità del vostro algoritmo.
4. ★Scrivere un algoritmo alternativo *distVerdiRossi*(grafo G, V_1, V_2) che fa un'unica visita del grafo G e quindi ha complessità $\Theta(m+n)$ [ovviamente l'input degli insiemi V_1 e V_2 dipende dal punto 1.]

Soluzione 1

Gli insiemi V_1 e V_2 possono essere rappresentati semplicemente con un vettore *colore* indicizzato sui nodi che contiene i valori verde, rosso, grigio: *colore*[*u*] è **verde** se $u \in V_1$, è **rosso** se $u \in V_2$, e **grigio** altrimenti. Ovviamente si può usare una codifica a valori interi (per esempio, 17, 73 e 42 😉 rispettivamente).

Nel primo caso, dovendo usare la funzione *dist*(*G*, *s*), possiamo, per ogni nodo s tale che *colore*[*s*] è verde, calcolare il minimo valore del vettore *d*, sui nodi rossi, cioè $\min_{u \in V2} d[u]$

```
fun distVerdiRossi(grafo G=(V,E), array colore):
  dist = |V|
  forall u ∈ V do
     if colore[u]=VERDE then
        d = dist(G, u)          /* O(m+n) */
        for i=1 to n do          /* O(n) */
           if colore[i]=ROSSO and d[i] < dist
              then dist = d[i]
  return dist
```

$\mathcal{O}(n^2+nm)$

Soluzione 1bis

Potendo modificare la funzione *dist*, ma seguendo la stessa idea, possiamo calcolare la distanza dal primo nodo rosso, interrompendo la BFS non appena si incontra un nodo verde e restituendo un solo valore.

Osservate che il tempo di esecuzione cala vistosamente, ma non la complessità asintotica globale. Chiamiamo *distR* la nuova funzione (**vedi prossima slide**).

```
fun distVerdiRossi(grafo G=(V,E), array colore):
   dist = |V|
  forall  u ∈ V do
     if colore[u]=VERDE then
        d = distR(G, u, colore)          /* O(m+n) */
        if d < dist  then dist = d
  return dist
```

$\mathcal{O}(n^2+nm)$

Soluzione 1bis

A titolo di ripasso, vediamo come modificare la BFS per ottenere la funzione *distR*.

Osservare che il vettore *dist* viene usato anche per marcare i nodi già visitati.

```
fun distR(grafo G=(V,E), nodo s, array colore):
  alloca(dist)  /* dist array indicizzato sui nodi */
  forall u ∈ V do dist[u]=+∞
  Q=emptyQueue();
  enqueue(Q, s)
  dist[s]=0
 while  notEmpty(Q) do
    v = dequeue(Q)
      /* se dist[v]<+∞ allora u è già stato visitato */
    if dist[v] ≠ +∞ then
       forall u ∈ adj(v) do
          if colore[u] = ROSSO then return dist[v] + 1
          dist[u]=dist[v]+1 /* ↑ esco al primo rosso*/
          enqueue(Q, u)
 return +∞
```

$\mathcal{O}(n+m)$

Soluzione punto 4

Possiamo modificare la BFS come segue: mettiamo nella coda **tutti i nodi di V_1** con distanza 0. Procedendo come in una normale BFS, dopo aver estratto e processato tutti i nodi di V_1 avremmo trovato tutti i nodi di G che distano 1 da V_1 e così via finchè non incontriamo il primo nodo di V_2: la sua distanza sarà la distanza di V_2 da V_1. Il tutto **con una sola BFS**!

```
fun distVerdiRossi(grafo G=(V,E), array colore):
  alloca(dist)  /* dist array indicizzato sui nodi */
  Q=emptyQueue();
  forall u ∈ V1 do enqueue(Q, u); dist[u] = 0
  while notEmpty(Q) do
     v = dequeue(Q)
     if dist[v] ≠ +∞ then
           forall u ∈ adj(v) do
           if colore[u]=ROSSO then return dist[v]+1
           dist[u]=dist[v]+1
           enqueue(Q, u)
 return +∞
```

Inserisco tutti i nodi di V_1 nella coda con dist=0

$\mathcal{O}(n+m)$