

Informatica Generale

Ivano Salvo/Giancarlo Bongiovanni

Test Finale di Autovalutazione

Corso di Laurea in Matematica, II anno



SAPIENZA
UNIVERSITÀ DI ROMA

Lezione 30, 10 giugno 2021

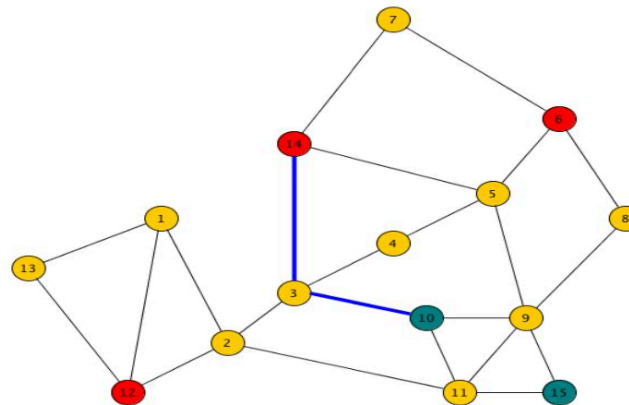
Distanze tra insiemi di nodi

Esercizio 9

Dato un grafo G e due sottoinsiemi V_1 e V_2 dei suoi vertici si definisce distanza tra V_1 e V_2 la distanza minima per andare da un nodo in V_1 ad un nodo in V_2 . Nel caso V_1 e V_2 non sono disgiunti allora il valore 0.

Descrivere un algoritmo che, dato un grafo G e i due sottoinsiemi dei vertici V_1 e V_2 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.



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 $dist(\text{grafo } G, \text{nodo } s)$ 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 $distVerdiRossi(\text{grafo } G, V_1, V_2)$ 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 $\min_{u \in V_2} d[u]$

```
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
```

$O(n^2+nm)$

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 rosso 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  $u \in V$  do  
        if colore[ $u$ ]=VERDE then  
             $d = \text{distR}(G, u, \text{colore})$       /*  $O(m+n)$  */  
            if  $d < \text{dist}$  then  $\text{dist} = d$   
    return dist
```

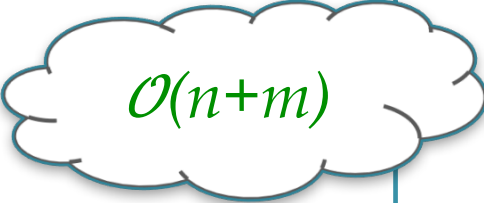
$O(n^2+nm)$

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  $u \in V$  do  $dist[u]=+\infty$   
  Q=emptyQueue();  
  enqueue(Q, s)  
   $dist[s]=0$   
  while notEmpty(Q) do  
     $v = dequeue(Q)$   
    /* se  $dist[v] < +\infty$  allora  $u$  è già stato visitato */  
    if  $dist[v] \neq +\infty$  then  
      forall  $u \in adj(v)$  do  
        if  $colore[u] = \text{ROSSO}$  then return  $dist[v] + 1$   
         $dist[u]=dist[v]+1$  /* ↑ esco al primo rosso */  
        enqueue(Q, u)  
  return  $+\infty$ 
```



$O(n+m)$

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 */  
   $Q = \text{emptyQueue}()$ ;  
  forall  $u \in V_1$  do enqueue( $Q, u$ );  $dist[u] = 0$   
  while notEmpty( $Q$ ) do  
     $v = \text{dequeue}(Q)$   
    if  $dist[v] \neq +\infty$  then  
      forall  $u \in \text{adj}(v)$  do  
        if  $colore[u] = \text{ROSSO}$  then return  $dist[v] + 1$   
         $dist[u] = dist[v] + 1$   
        enqueue( $Q, u$ )  
  return  $+\infty$ 
```

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

$O(n+m)$

Lezione 20

That's all Folks...

Grazie per l'attenzione...

...Domande?