Azzolini Riccardo 2019-03-18

Visita di grafi

1 Attraversamento in ampiezza

Problema:

- *Input*: Un grafo $G = \langle V, E \rangle$, non orientato e connesso, e un nodo di partenza $s \in V$.
- *Output*: Una sequenza che contiene, una e una sola volta, tutti i nodi del grafo, ordinati rispetto alla distanza da s (cioè prima s, poi i nodi adiacenti a s, poi quelli adiacenti a questi ultimi, ecc.). A parità di distanza, l'ordine non è specificato, quindi è determinato dall'implementazione.

Soluzione:

- si sceglie di rappresentare G tramite liste di adiacenza, ordinate per etichette (in questo modo viene fissato anche l'ordine dei nodi alla stessa distanza);
- si usa una coda in cui salvare i nodi da visitare.

1.1 Implementazione

```
public void Ampiezza(s) {
    Queue<Nodo> q = new Queue<Nodo>();
    List<Lato> u = new List<Lato>();

    for (v in V) nuovo[v] = 1;
    q.enqueue(s); nuovo[s] = 0;

    while (!q.isEmpty()) {
        v = q.front(); q.dequeue();
        v.visita();

        for (w in L(v))
            if (nuovo[w]) {
                q.enqueue(w); nuovo[w] = 0;
                u.insert({v, w}, 1);
            }
    }
}
```

- `s` è il nodo di partenza.
- `q` è la coda in cui vengono salvati i nodi da visitare.
- `u` è una lista contenente i lati "percorsi" durante l'attraversamento, che formano un *albero di copertura* del grafo. Poiché questa lista rappresenta un insieme, l'ordine degli elementi è irrilevante, quindi l'inserimento viene effettuato in testa per minimizzarne il costo.
- `nuovo` è un vettore che contiene 1 per i nodi ancora da visitare, e 0 in corrispondenza dei nodi già visitati, o anche solo inseriti nella coda (per i quali è stata "prenotata" la visita).
- `V` è l'insieme dei nodi del grafo.
- `L(v)` è la lista di adiacenza del nodo `v`.

1.2 Complessità

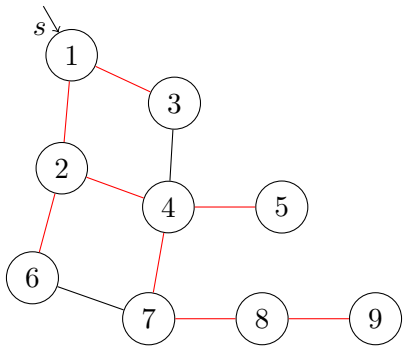
Dato il grafo $G = \langle V, E \rangle$, siano $n = |V|$ e $m = |E|$.

- Lo *spazio* richiesto, senza contare la rappresentazione del grafo, è $\Theta(n)$, perché
  - il vettore `nuovo` ha n elementi: $\Theta(n)$
  - la lista `u` contiene, alla fine dell'attraversamento, gli $n - 1$ lati di un albero con n nodi: $\Theta(n)$
  - la coda `q` assume durante l'esecuzione una lunghezza massima che varia da 1 (nel caso migliore, l'attraversamento di un albero degenere) a n (nel caso peggiore di un *grafo a stella*, in cui tutti i nodi sono adiacenti a s): $O(n)$

  e, in totale, $\Theta(n) + \Theta(n) + O(n) = \Theta(n)$.
- Il *tempo* (numero di istruzioni) necessario è $\Theta(n + m)$, che si può considerare equivalente a $\Theta(m)$ perché, in un grafo connesso, si ha sempre $m \geq n - 1 \implies m = \Omega(n)$.

  Le operazioni che contribuiscono alla complessità sono principalmente:
  - il ciclo di inizializzazione del vettore `nuovo`: $\Theta(n)$;
  - la visita di ciascun nodo, $O(1) \cdot n = \Theta(n)$ in totale, e la lettura della sua lista di adiacenza, che complessivamente (su tutti i nodi) ha costo $\Theta(m)$.

1.3 Esempio



	nuovo	
Visitato	1 2 3 4 5 6 7 8 9	Coda q
	0,1,1,1,1,1,1,1,1	1
1	0,0,0,1,1,1,1,1,1	2,3
2	0,0,0,0,1,0,1,1,1	3,4,6
3	0,0,0,0,1,0,1,1,1	4,6
4	0,0,0,0,0,0,0,1,1	6,5,7
6	0,0,0,0,0,0,0,1,1	5,7
5	0,0,0,0,0,0,0,1,1	7
7	0,0,0,0,0,0,0,0,1	8
8	0,0,0,0,0,0,0,0,0	9
9	0,0,0,0,0,0,0,0,0	

2 Attraversamento in profondità

A ogni passo si cerca di prolungare il cammino (semplice) corrente, visitando un nodo non ancora visitato. Se tutti i nodi adiacenti a quello corrente sono già stati visitati, si torna indietro di un passo e si riprova.

Soluzione:

- come per l'attraversamento in ampiezza, si rappresenta G tramite liste di adiacenza;
- si utilizza la ricorsione, oppure una pila in cui salvare i nodi da visitare.

2.1 Implementazione ricorsiva

```
public void ProfondRic(v) {
    v.visita(); nuovo[v] = 0;
```

```
    for (w in L(v))
        if (nuovo[w]) {
            u.insert({v, w}, 1);
            ProfondRic(w);
        }
}
```

- `v` è il nodo corrente, e `L(v)` è la sua lista di adiacenza.
- Come nell'attraversamento in ampiezza, `nuovo` è un vettore che indica i nodi ancora da visitare e `u` è una lista che memorizza i lati dell'*albero di copertura* risultante. In quest'implementazione, però, tali variabili devono essere definite e inizializzate fuori dalla funzione `ProfondRic`, dato che essa viene eseguita più volte.

2.1.1 Complessità

Dato il grafo $G = \langle V, E \rangle$, siano $n = |V|$ e $m = |E|$.

- Lo *spazio* necessario è $\Theta(n)$, composto da:
  - $\Theta(n)$ per il vettore `nuovo`;
  - $\Theta(n)$ per la lista `u`;
  - $O(n)$ per lo stack delle chiamate.
- Il *tempo* impiegato è $\Theta(n + m)$, equivalente a $\Theta(m)$ perché $m = \Omega(n)$.

  Siccome ogni nodo viene visitato una e una sola volta, vengono eseguite n chiamate. Ciascuna chiamata effettua

  - alcune operazioni di costo $O(1)$ per la visita del nodo, che sommando tutte le chiamate hanno costo complessivo $\Theta(n)$
  - una lettura della lista di adiacenza del nodo, che ha costo variabile, ma la somma su tutte le chiamate è $\Theta(m)$

  per un costo totale di $\Theta(n + m)$.

2.2 Implementazione iterativa

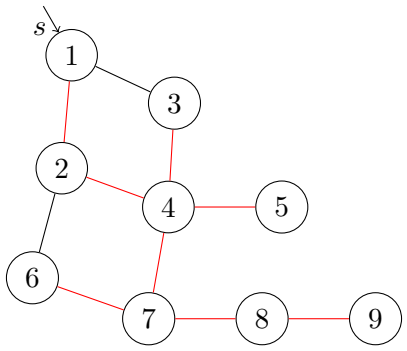
```
public void ProfondIt(v) {
    Stack<Nodo> s = new Stack<Nodo>();
    List<Lato> u = new List<Lato>();

    for (x in V) nuovo[x] = 1;
    v.visita(); nuovo[v] = 0;
    s.push(v);

    do {
        while (esiste w in L(v) && nuovo[w]) {
            w.visita(); nuovo[w] = 0;
            u.insert({v, w}, 1);
            s.push(w); v = w;
        }

        s.pop();
        if (!s.isEmpty()) v = s.top();
    } while (!s.isEmpty());
}
```

- `v` è il nodo corrente.
- `s` è la pila che tiene traccia del cammino dal nodo di partenza a quello corrente, sostituendo la ricorsione.
- `nuovo`, `u` e `L(v)` hanno il solito significato.
- La condizione `esiste w in L(v) && nuovo[w]` del ciclo `while` cerca tra i nodi adiacenti a `v` uno che non sia ancora stato visitato:
  - se esiste, viene assegnato alla variabile `w` e si esegue il corpo del ciclo;
  - altrimenti, il ciclo termina.

2.2.1 Complessità

La complessità è pari alla versione ricorsiva, sia per quanto riguarda il *tempo*, $\Theta(n + m)$, $m = \Omega(n) \implies \Theta(m)$, che per lo *spazio*, $\Theta(n)$: la pila occupa una quantità di memoria asintoticamente uguale ai record di attivazione per le chiamate ricorsive.

2.3 Esempio



Visitato	nuovo 1 2 3 4 5 6 7 8 9	Chiamate o pila s
	1,1,1,1,1,1,1,1,1	
1	0,1,1,1,1,1,1,1,1	1
2	0,0,1,1,1,1,1,1,1	1,2
4	0,0,1,0,1,1,1,1,1	1,2,4
3	0,0,0,0,1,1,1,1,1	1,2,4,3
	0,0,0,0,1,1,1,1,1	1,2,4
5	0,0,0,0,0,1,1,1,1	1,2,4,5
	0,0,0,0,0,1,1,1,1	1,2,4
7	0,0,0,0,0,1,0,1,1	1,2,4,7
6	0,0,0,0,0,0,0,1,1	1,2,4,7,6
	0,0,0,0,0,0,0,1,1	1,2,4,7
8	0,0,0,0,0,0,0,0,1	1,2,4,7,8
9	0,0,0,0,0,0,0,0,0	1,2,4,7,8,9
	0,0,0,0,0,0,0,0,0	1,2,4,7,8
	0,0,0,0,0,0,0,0,0	1,2,4,7
	0,0,0,0,0,0,0,0,0	1,2,4
	0,0,0,0,0,0,0,0,0	1,2
	0,0,0,0,0,0,0,0,0	1
	0,0,0,0,0,0,0,0,0	