ESERCIZIO DI ASD DEL 27 APRILE 2009

Diametro

Algoritmi. Ricordiamo che un grafo non orientato, aciclico e connesso è un albero. Un albero può essere pensato come albero radicato una volta che si sia fissato un nodo come radice. Ad esempio il grafo Gr in Figura 1 può essere pensato come albero radicato nel nodo a.

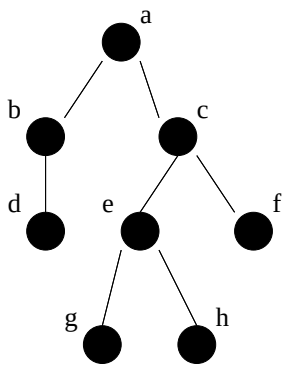

Figura 1. Grafo Gr.

A seconda del nodo che utilizziamo come radice l'albero radicato avrà altezza diversa. L'altezza di un albero radicato non è altro che la massima distanza tra la radice ed una foglia. Il grafo Gr se pensato come albero radicato in a ha altezza 3, mentre se pensato come albero radicato in d ha altezza 5. Se richiamiamo BFS(Gr, a) otteniamo che il nodo a distanza massima da a si trova a distanza 3, mentre se richiamiamo BFS(Gr, d) otteniamo che il nodo a distanza massima da d è a distanza 5.

Il diametro di un grafo G si può quindi trovare richiamando BFS(G, x) per ogni x nodo di G e memorizzando di volta in volta la distanza massima calcolata. Nel grafo Gr otteniamo che il diametro è 5.

Quindi come prima cosa modifichiamo la procedura BFS in modo che al termine ritorni la massima distanza calcolata (Algorithm 1). Si noti che non è necessario usare i colori e neanche costruire l'albero di visita.

A questo punto è sufficiente richiamare BFS_DIAM una volta per ogni nodo e determinare la distanza massima tra tutte le distanze massime restituite (Algorithm 2).

Possiamo utilizzare BFS per ottenere una procedura più efficiente? Dovremmo evitare di richiamare BFS su ogni nodo del grafo e richiamarla sul "pochi" nodi. Ma come? Bisognerebbe riuscire ad indovinare una radice che massimizza l'altezza. Sicuramente non basta utilizzare BFS una sola volta, perchè la prima volta che

Date: 27 Aprile 2009.

Algorithm 1 BFS_DIAM($G = (N, E), x$)

```
1: max ← −1
2: for each v ∈ N do
3:    d[v] ← ∞
4: end for
5: Q ← ∅
6: d[x] ← 0
7: ENQUEUE(Q, x)
8: while Q ≠ ∅ do
9:    v ←HEAD(Q)
10:   for each u ∈ Adj[v] do
11:      if d[u] = ∞ then
12:         d[u] ← d[v] + 1
13:         ENQUEUE(Q, u)
14:      end if
15:   end for
16:   max ← d[v]
17:   DEQUEUE(Q)
18: end while
19: return max
```

Algorithm 2 DIAMETRO_NAIVE($G = (N, E)$)

```
1: diametro ← −1
2: for each x ∈ N do
3:    diametro ← MAX(diametro,BFS_DIAM(G, x))
4: end for
5: return diametro
```

richiamiamo la procedura non sappiamo niente del nostro grafo. Se osserviamo nuovamente il grafo Gr in Figura 1 notiamo che ci sono 3 nodi che ci consentono di calcolare il diametro: il nodo d, il nodo g, ed il nodo h. Due di questi hanno una caratteristica interessante: g ed h sono due nodi a distanza massima da a. In effetti in generale possiamo procedere in questo modo:

- richiamiamo BFS a partire da un nodo $first$ e determiniamo un nodo *second* a distanza massima da $first$;
- richiamiamo BFS a partire da *second*. La distanza massima da *second* sarà il diametro del grafo.

A questo punto ci serve una variante di BFS che restituisca un nodo a distanza massima (Algorithm 3). L'ultimo nodo ad uscire dalla coda sarà sicuramente un nodo a distanza massima.

Ora non resta che combinare BFS_DIAM e BFS_NODO per ottenere una procedura efficiente per il diametro.

Correttezza. La correttezza della procedura DIAMETRO_NAIVE segue immediatamente dalla definizione di diametro, dalla correttezza della procedura BFS e dal fatto che BFS(G, x) calcola le distanze da x in ordine crescente, quindi l'ultima distanza calcolata sarà la distanza massima di un nodo dal nodo x.

Algorithm 3 BFS_NODO$(G = (N, E), x)$

```
1: nodo ← NIL
2: for each v ∈ N do
3:    d[v] ← ∞
4: end for
5: Q ← ∅
6: d[x] ← 0
7: ENQUEUE(Q, x)
8: while Q ≠ ∅ do
9:    v ←HEAD(Q)
10:   for each u ∈ Adj[v] do
11:      if d[u] = ∞ then
12:         d[u] ← d[v] + 1
13:         ENQUEUE(Q, u)
14:      end if
15:   end for
16:   nodo ← v
17:   DEQUEUE(Q)
18: end while
19: return nodo
```

Algorithm 4 DIAMETRO$(G = (N, E))$

```
1: first ←PICK(N) //sceglie un nodo qualsiasi in N
2: second ←BFS_NODO(G, first)
3: return BFS_DIAM(G, second))
```

Dimostriamo la correttezza della procedura DIAMETRO. Diamo per buona la correttezza della procedura BFS_DIAM(G, x) che calcola la distanza massima di un nodo da x e della procedura BFS_NODO(G, x) che determina un nodo a distanza massima da x.

Theorem 1. *La procedura* DIAMETRO(G) *termina sempre ed al termine restituisce il diametro di* G*.*

Dimostrazione. Dalla correttezza di BFS_DIAM$(G, second)$ abbiamo che la procedura restituisce un valore d tale che esiste un nodo z tale che $\delta(second, z) = d$.

Dalla definizione di diametro sappiamo che sicuramente vale $d = \delta(second, z) \leq d(G)$.

Quindi dobbiamo solo dimostrare che $d(G) \leq \delta(second, z) = d$, ovvero che

$$\forall a, b \in N(\delta(a, b) \leq \delta(second, z))$$

Consideriamo l'albero di BFS generato a partire dal nodo $first$. In questo albero troviamo sicuramente l'unico cammino che esiste in G tra a e b. Questo cammino risalirà da a verso la radice $first$ fino ad arrivare ad un nodo t e poi riscenderà verso b. In altri termini t è il più giovane antenato comune tra a e b nell'albero BFS di $first$. Indichiamo con $p(first, t)$ il cammino tra $first$ e t e con $p(first, second)$ il cammino tra $first$ e $second$. Distinguiamo due casi.

Caso 1. $p(first, t)$ e $p(first, second)$ non hanno archi in comune. Si veda la Figura 2. In tal caso abbiamo che

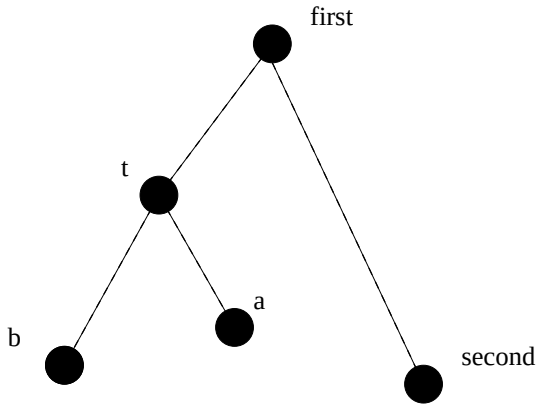

FIGURA 2. Caso 1.

$$\begin{array}{lcll} \delta(second, b) & = & \delta(second, first) + \delta(first, t) + \delta(t, b) & \\ & \geq & \delta(second, first) + \delta(t, b) & \text{ho tralasciato un addendo} \\ & \geq & \delta(a, first) + \delta(t, b) & second \text{ era a distanza massima da } first \\ & \geq & \delta(a, t) + \delta(t, b) & t \text{ é sul cammino tra } a \text{ e } b \\ & = & \delta(a, b) & \end{array}$$

Quindi abbiamo $\delta(second, z) \geq \delta(second, b)$ in quanto z è a distanza massima da $second$ e $\delta(second, b) \geq \delta(a, b)$. Possiamo concludere $\delta(second, z) \geq \delta(a, b)$.

Caso 2. $p(first, t)$ e $p(first, second)$ hanno in comune tutti gli archi che si trovano tra $first$ ed un nodo r. Se $second$ si trova nel sottoalbero radicato in t (ovvero se $r = t$), allora potrebbe esserci una parte di cammino tra t e $second$ in comune con il cammino tra t ed a o con quello tra t e b, ma non con entrambi, visto che questi due non hanno archi in comune. In tal caso non è restrittivo supporre che ci potrebbero essere archi in comune con il cammino tra t ed a. Si veda la Figura 3. Quindi abbiamo che

$$\begin{array}{lcll} \delta(second, b) & = & \delta(second, r) + \delta(r, b) & \\ & \geq & \delta(a, r) + \delta(r, b) & (*) \\ & = & \delta(a, b) & \end{array}$$

dove in (*) abbiamo potuto rimpiazzare $\delta(second, r)$ con $\delta(a, r)$ in quanto $second$ è a distanza massima da $first$ e quindi la distanza tra $second$ e r è maggiore di quella tra a ed r.

Quindi abbiamo $\delta(second, z) \geq \delta(second, b)$ in quanto z è a distanza massima da $second$ e $\delta(second, b) \geq \delta(a, b)$. Possiamo concludere $\delta(second, z) \geq \delta(a, b)$. □

Complessità. Le procedure BFS_DIST e BFS_NODO hanno complessità $\Theta(|V| + |E|) = \Theta(|E|)$, in quanto non sono altro che banali modifiche di BFS ed il grafo in input è connesso.

Di conseguenza la procedura DIAMETRO_NAIVE ha complessità pari a $\Theta(|V| * |E|)$, in quanto BFS_DIST viene richiamata $|V|$ volte, mentre DIAMETRO ha complessità $\Theta(|E|)$, in quanto si richiama una volta BFS_NODO ed una volta BFS_DIST.

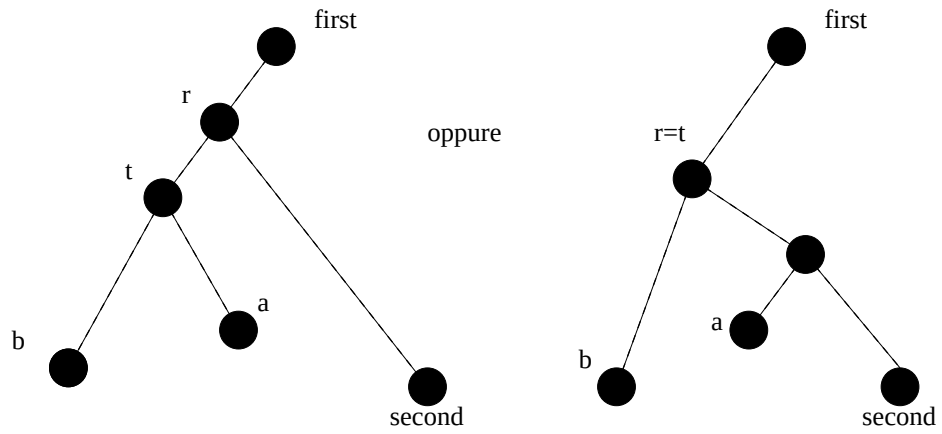

FIGURA 3. Caso 2.