Cicli ciclabili

Una soluzione lineare

Marco Rubini Dumitru Damaschin

Università di Trento

Il problema

Consideriamo un grafo connesso e non orientato $\mathcal{G} = (V , E)$ con N nodi ed M archi, in cui vale la seguente propriet: linsieme di nodi in ogni ciclo semplice, costituisce un sottografo completo (cricca).

Problema

Dato l'insieme $\mathcal{Q}$ di coppie di nodi (u, v), per ogni coppia calcolare la distanza d(u, v) tra u e v in $\mathcal{G}$.

Strategia generale

- Consideriamo $\mathcal{T}$ un albero orientato ottenuto da una visita in profondit di $\mathcal{G}$.
- Caratteriziamo i nodi di $\mathcal{T}$ che appartengono a una stessa cricca in $\mathcal{G}$.
- Per rispondere a una query (u, v) troviamo la distanza tra u e v in $\mathcal{T}$, poi la modifichiamo considerando le cricche rimosse.
- Siccome conosciamo l'intero insieme di query, rispondiamo a tutte le query con una singola visita del grafo.

Osservazioni

Definizione cricca massimale

Una cricca massimale un sottografo completo che non contenuto in nessun altro sottografo completo.

Osservazione 1

Due cricche massimali distinte con almeno 3 nodi non possono avere due nodi in comune.

Dimostrazione: Siano $C1$, $C2$ due cricche massimali, assumiamo $x \in (C1 \cap C2)$, $y \in (C1 \cap C2)$, $x \neq y$. Siano $u \in (C1 \setminus C2)$, $v \in (C2 \setminus C1)$. $x \to u \to y \to v \to x$ un ciclo per ogni scelta di u e v, quindi esiste una cricca che contiene tutti i nodi di $C1$ e $C2$, ma questo non pu essere perch $C1$ e $C2$ sono massimali.

Osservazioni

Osservazione 2

Se (u, v) un cross-edge in $\mathcal{T}$ allora esiste una e unica cricca massimale in $\mathcal{G}$ che contiene u e v.

Se (u, v) un cross-edge in $\mathcal{T}$, allora (u, v) appartiene a un ciclo in $\mathcal{G}$. Quindi u e v sono in una stessa cricca. Per l'osservazione 1, due cricche massimali non possono condividere due nodi, quindi la cricca anche unica.

Costruzione di $\mathcal{T}$

Per costruire $\mathcal{T}$ effettuiamo una visita alternatamente in profondit e per livello partendo da un nodo qualsiasi di $\mathcal{G}$. Durante la costruzione assegnamo ad ogni nodo un colore, tale che se $(\mathtt{u},\ \mathtt{v})$ un cross edge, allora $\mathtt{u}$ e $\mathtt{v}$ hanno lo stesso colore. All'inizio dell'algoritmo, tutti i nodi hanno un colore diverso.

Algorithm 1 $\mathcal{T}$ construction

Require: the graph $\mathcal{G}$

1: $\mathtt{visited} \leftarrow \mathtt{empty\ set}$
2: $\mathtt{colors} \leftarrow \{\mathtt{0},\ \mathtt{1},\ \ldots,\ \mathtt{size}(\mathcal{G})\mathtt{-1}\}$
3: $\mathcal{T} \leftarrow \mathtt{empty\ tree}$
4: `DFS1(0, -1)`

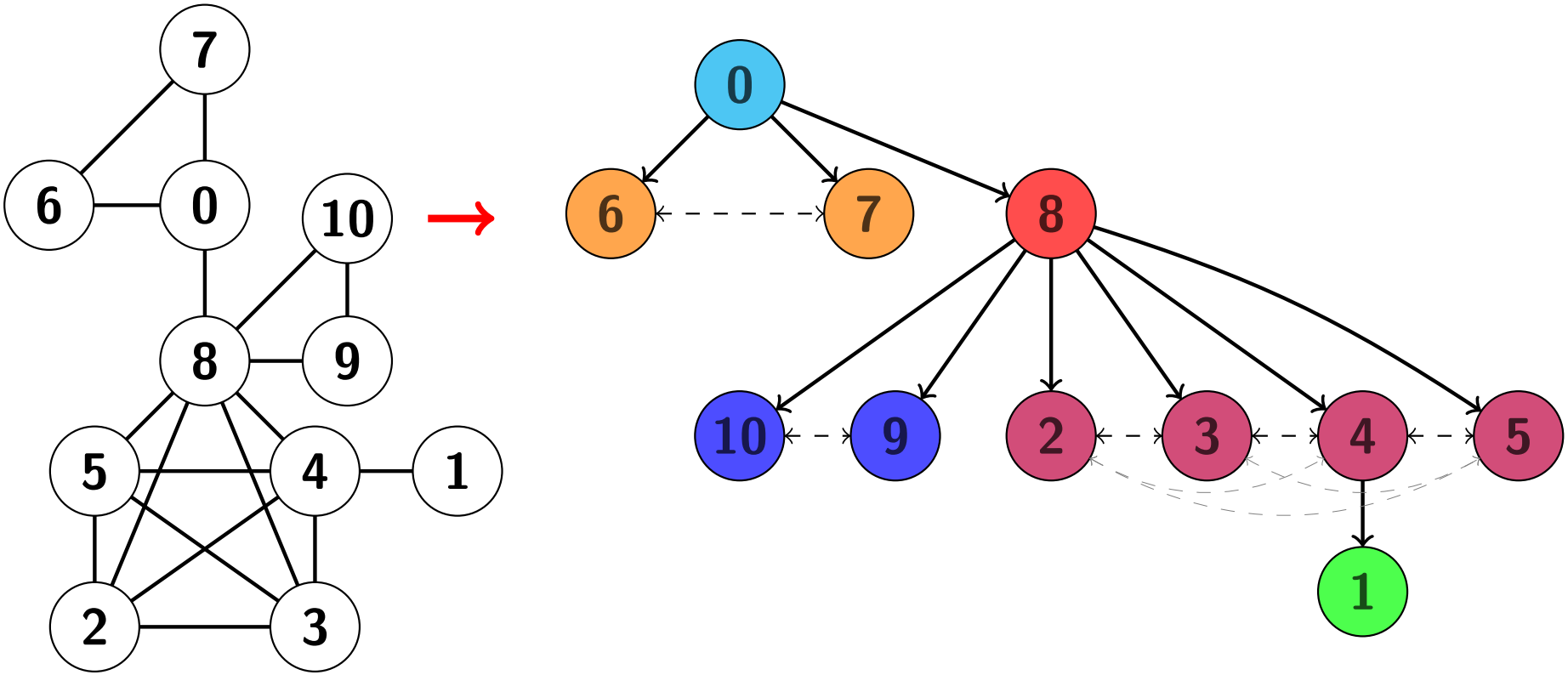
Costruzione di $\mathcal{T}$

Algorithm 2 $\mathcal{T}$ construction helper

Require: $\mathcal{G}$, $\mathcal{T}$, colors, visited

```
 1: procedure DFS1(curr, parent)
 2:     visited ← visited ∪ {curr}
 3:     for each e ∈ N_G(curr) if e.dest ≠ parent do
 4:         if e.dest ∈ visited then                    ▷ e un cross edge
 5:             colors[e.dest] ← colors[curr]
 6:         else                                        ▷ e un tree edge
 7:             visited ← visited ∪ {e.dest}
 8:             parents_T[e.dest] ← curr
 9:         end if
10:     end for
11:     for each e ∈ N_G(curr) if parents_T[e.dest] = curr do
12:         DFS1(e.dest, curr)
13:     end for
14: end procedure
```

Esempio



Propriet della visita

Properiet 1

Se (u, v) un cross edge, $\texttt{parents}_{\mathcal{T}}[u] = \texttt{parents}_{\mathcal{T}}[v]$

Propriet 2

$\texttt{color}[u] = \texttt{color}[v] \iff (u, v)$ un cross edge

Riduzione a LCA

Definizione LCA

Dato un albero $T = (V, E)$ e due nodi $u, v \in V$, $\texttt{LCA}(u, v)$ l'antenato comune di u, v di altezza maggiore nell'albero.

Distanza in un albero

Dato un albero $T = (V, E)$ e i nodi $u, v \in V$, $w = \texttt{LCA}(u, v)$
$d_T(u, v) = \texttt{depth}_T(u) + \texttt{depth}_T(v) - 2 \cdot \texttt{depth}_T(w)$

Per rispondere a una query (u, v) troviamo $w = \texttt{LCA}(u, v)$ in $\mathcal{T}$, calcoliamo $d_{\mathcal{T}}(u, v)$ e sottraiamo 1 se il percorso che collega u, v in $\mathcal{T}$ passa per 3 nodi della stessa cricca in $\mathcal{G}$.

Riduzione a LCA

Definizione Below-LCA

Siano $u, v \in \mathcal{T}, u \neq v$.
$BLCA_u(u, v) = x \mid \texttt{parents}_{\mathcal{T}}[x] = \texttt{LCA}(u, v) \wedge$ x pi vicino a u
$BLCA_v(u, v) = x \mid \texttt{parents}_{\mathcal{T}}[x] = \texttt{LCA}(u, v) \wedge$ x pi vicino a v

Per poter implementare l'algoritmo efficientemente, dobbiamo trovare per ogni query la sua coppia di `BLCA`. I `BLCA` fanno parte del percorso $u \to v$ in $\mathcal{T}$, e se sono diversi e fanno parte della stessa cricca, anche il loro parente $\texttt{LCA}(u, v)$ ne fa parte, quindi dobbiamo sottrarre 1 alla soluzione.

Ricerca di tutti i BLCA

Per trovare la coppia di BLCA per ogni query usiamo l'algoritmo di Tarjan per il calcolo di LCA offline, che ha complessit lineare nella dimensione del grafo e nel numero di query.
https://en.wikipedia.org/wiki/Tarjan%27s_off-line_lowest_common_ancestors_algorithm

Ricerca di tutti i BLCA

Algorithm 3 Ricerca dei BLCA per ogni query

```
1:  procedure DFS2(curr)
2:      path.push_back(curr)
3:      makeset(curr)
4:      for each e ∈ N_T(curr) do
5:          DFS2(e.dest)
6:      end for
7:      for each (u, v) ∈ Q | curr ∈ {u, v} do          ▷ query contenenti curr
8:          w ← other endpoint of the query
9:          if w ∈ closed then
10:             blca_w ← highest_closed_ancestor[findset(w)]
11:             if depth_T(blca_w) > depth_T(curr) then      ▷ curr  ancestor di w
12:                 blca_curr ← blca_w
13:             else
14:                 blca_curr ← path[depth_T(blca_w)]
15:             end if
16:         end if
17:     end for
```

Ricerca di tutti i BLCA

```
18:     for each e ∈ 𝒩_𝒯(curr) do
19:         joinset(curr, e.dest)
20:     end for
21:     highest_closed_ancestor[findset(curr)] ← curr
22:     path.pop_back()
23:     closed ← closed ∪ {curr}
24: end procedure
```

Complessit

Sorgente: `https://gist.github.com/marcorubini/cb4dbae0ae99c34a2fee106b03182bee`
Complessit: $\mathcal{O}((N+M)\cdot\mathcal{A}(M+N)+\text{size}(\mathcal{Q}))$
Dove $\mathcal{A}$ l'inversa della funzione di Ackermann. Il fattore moltiplicativo deriva dall'utilizzo di una implementazione semplice della struttura dati union-find.