Soluzione Progetto 1 ASD a.a. 2019/2020

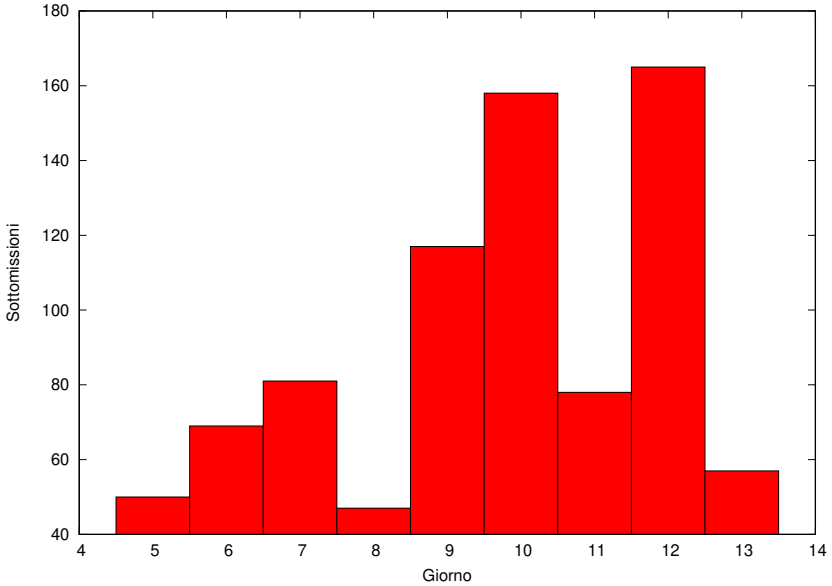
Cicli ciclabili

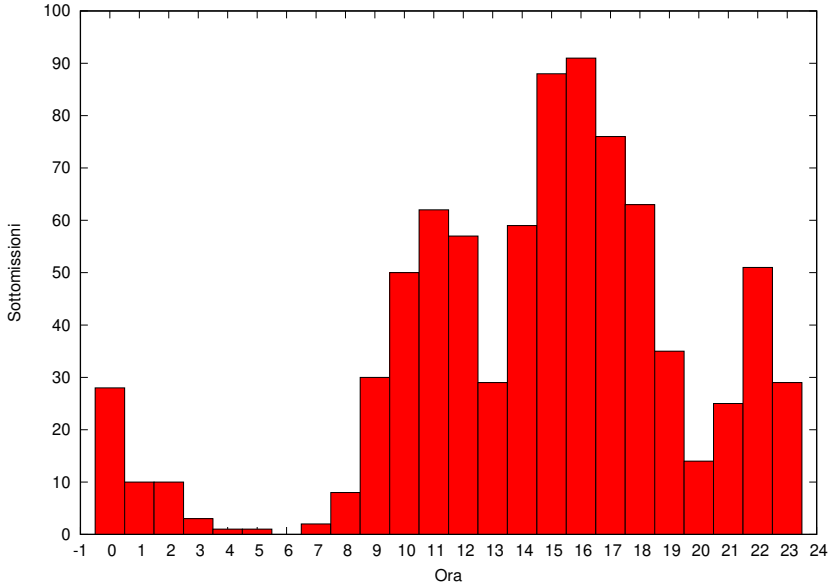
Cristian Consonni, Marta Fornasier e Martin Brugnara

17 dicembre 2019

STATISTICHE

Numero sottoposizioni: 822





- 77 gruppi partecipanti, di cui 73 gruppi hanno fatto almeno una sottoposizione;
- 188 studenti iscritti, di cui 183 appartenenti a gruppi che hanno fatto almeno una sottoposizione;

Risultati

Punteggi

- $P < 30$ $\longrightarrow$ progetto non passato
- $30 \leq P \leq 50$ $\longrightarrow$ 1 punti bonus (23 gruppi)
- $65 \leq P \leq 95$ $\longrightarrow$ 2 punti bonus (19 gruppi)
- $P = 100$ $\longrightarrow$ 3 punti bonus (30 gruppi)

Classifiche e sorgenti sul sito (controllate i numeri di matricola):
`https://judge.science.unitn.it/slides/asd19/classifica_prog1.pdf`

Il problema

Consideriamo un grafo connesso e non orientato $G = (V, E)$ con N nodi ed M archi, in cui vale la seguente proprietà: **l'insieme di nodi in ogni ciclo semplice, costituisce un sottografo completo (cricca)**.

Problema

Date Q coppie di nodi (u, v), per ogni coppia calcolare la distanza $d(u, v)$ tra u e v nel grafo G.

Considerazioni iniziali

- Calcolare una singola distanza tra due nodi è semplice, è sufficiente fare una visita bfs del grafo.
- Ripetere questa operazione per ogni richiesta però non è efficiente...
- ...L'idea fondamentale sarà quella di preprocessare il grafo per rendere il calcolo di ogni distanza più veloce.

Una visita per ogni richiesta

Idea: per ogni richiesta (u, v), si calcola la distanza tra i nodi u e v con una visita bfs del grafo.

- $\Rightarrow$ soluzione: `bfs.cpp`
- $\Rightarrow$ complessità: $O(Q(N + M))$
- $\Rightarrow$ 30 punti

Sottoproblema: alberi

Idea: se il grafo di input è un albero, si può sfruttare uno degli algoritmi per calcolare il lowest common ancestor (LCA) tra due nodi. Nella versione seguente ha costo di preprocessing $O(N \log N)$ e permette il calcolo di ogni distanza in $O(\log N)$.

Definizione

Dato un albero $T = (V, E)$ e due nodi $u, v \in V$, definiamo $lca(u, v)$ come l'antenato comune di u e v di altezza maggiore nell'albero.

Sottoproblema: alberi

Calcolare il lowest common ancestor di due nodi permette facilmente di calcolarne la distanza.

Osservazione

Dato un albero $T = (V, E)$ e detta, per ogni $w \in V$, $h(w)$ l'altezza del nodo w (la distanza dalla radice), allora per ogni coppia di nodi $u, v \in V$, vale che

$$d(u, v) = h(u) + h(v) - 2h(lca(u, v)).$$

Sottoproblema: alberi (algoritmo)

- Si costruisce una matrice $L[N][\log N]$ in cui $L[i][j]$ è definito come il 2^j-esimo antenato del nodo i. Per costruirla si osserva che

$$L[i][j] = \begin{cases} parent[i] & j = 0 \\ L[L[i][j-1]][j-1] & j > 0 \end{cases}$$

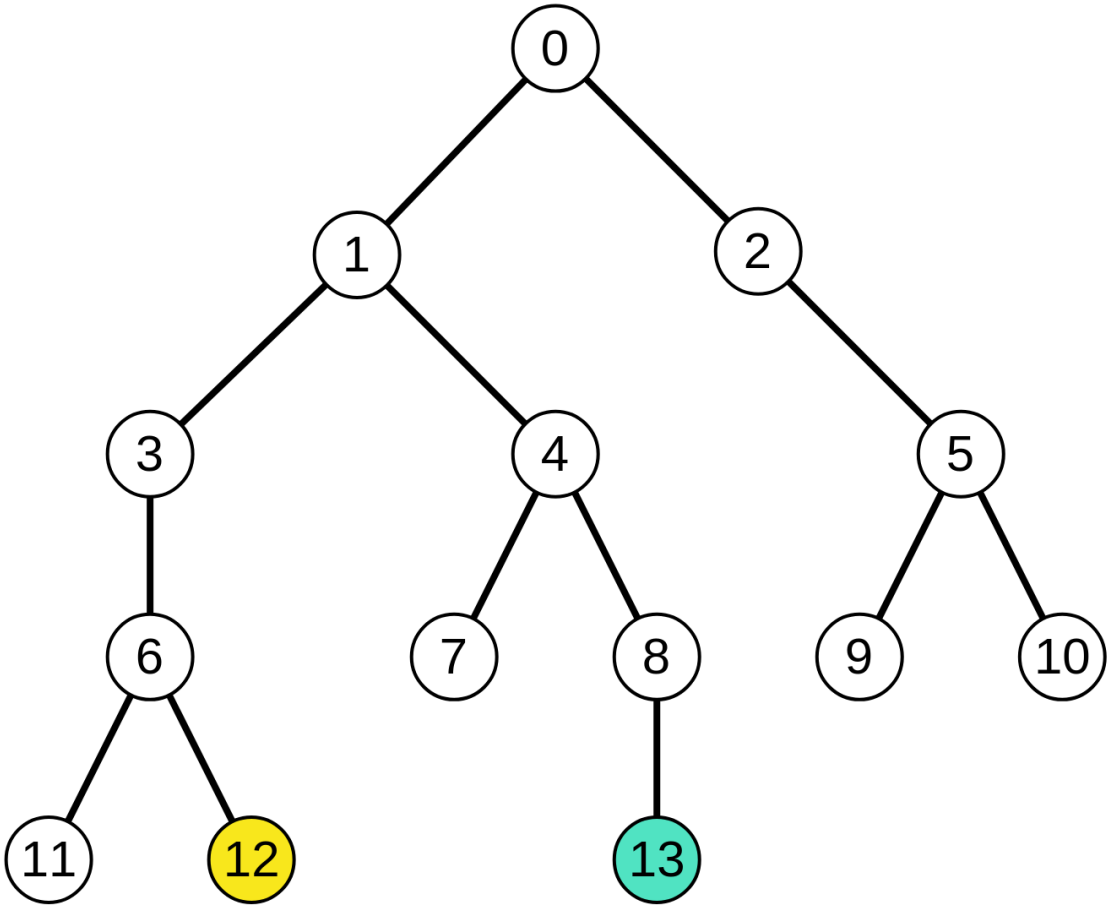
  Il costo di costruzione della matrice è $O(N \log N)$.
- Per calcolare la distanza tra due nodi u e v, si trova $lca(u, v)$ sfruttando la tabella costruita e si utilizza l'osservazione precedente. Il costo di calcolo di ogni distanza è $O(\log N)$.

Sottoproblema: alberi (algoritmo)

Per ogni coppia (u, v) di nodi, si calcola il lowest common ancestor come segue:

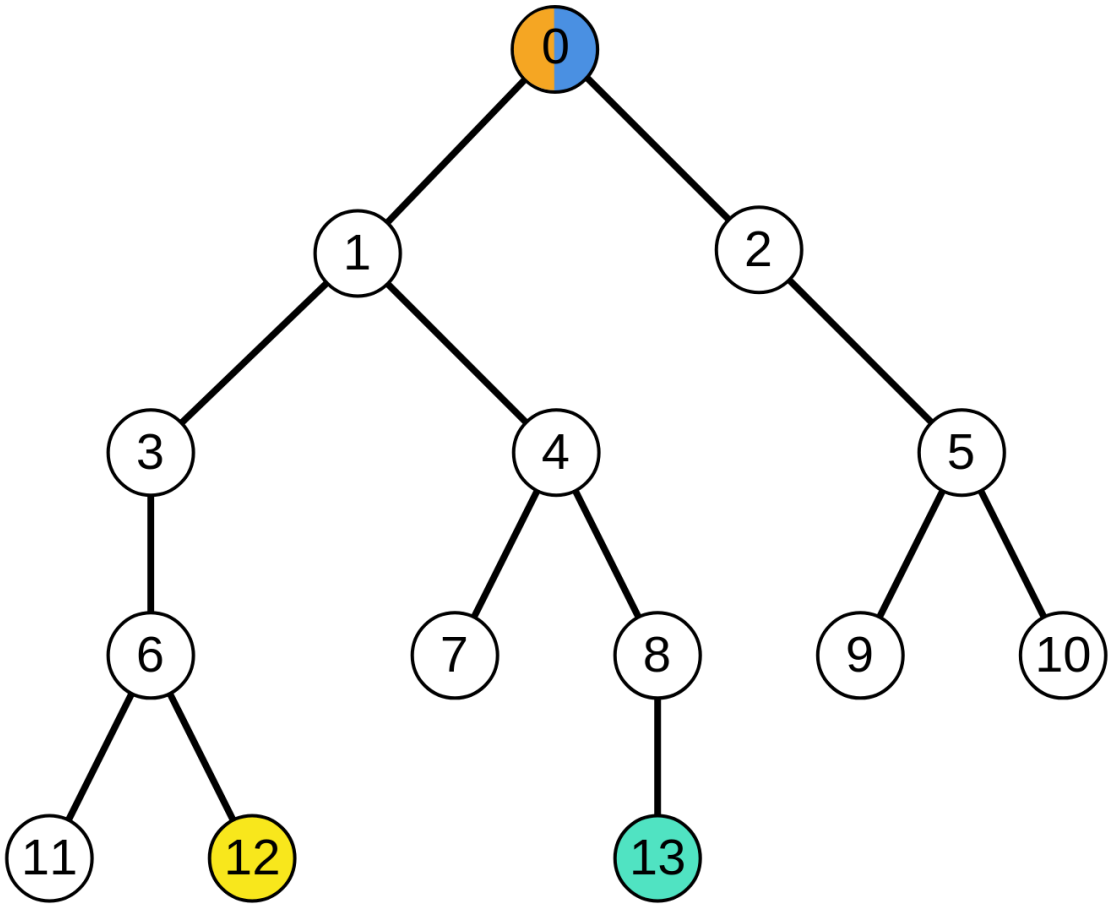
- Se i due nodi non sono alla stessa altezza, si sostituisce il nodo ad altezza maggiore con il suo antenato situato alla stessa altezza dell'altro nodo.
- Per j che varia da $\log N$ a 0 (decrescendo), si confrontano gli antentati 2^j-esimi dei nodi considerati e, se sono distinti, si sostituiscono ai nodi stessi.
- Alla fine dell'algoritmo i due nodi avranno lo stesso padre, che sarà l'antenato cercato.

Esempio



- $u = 12$, $v = 13$
- $j = 2$, $h = 2^j = 4$

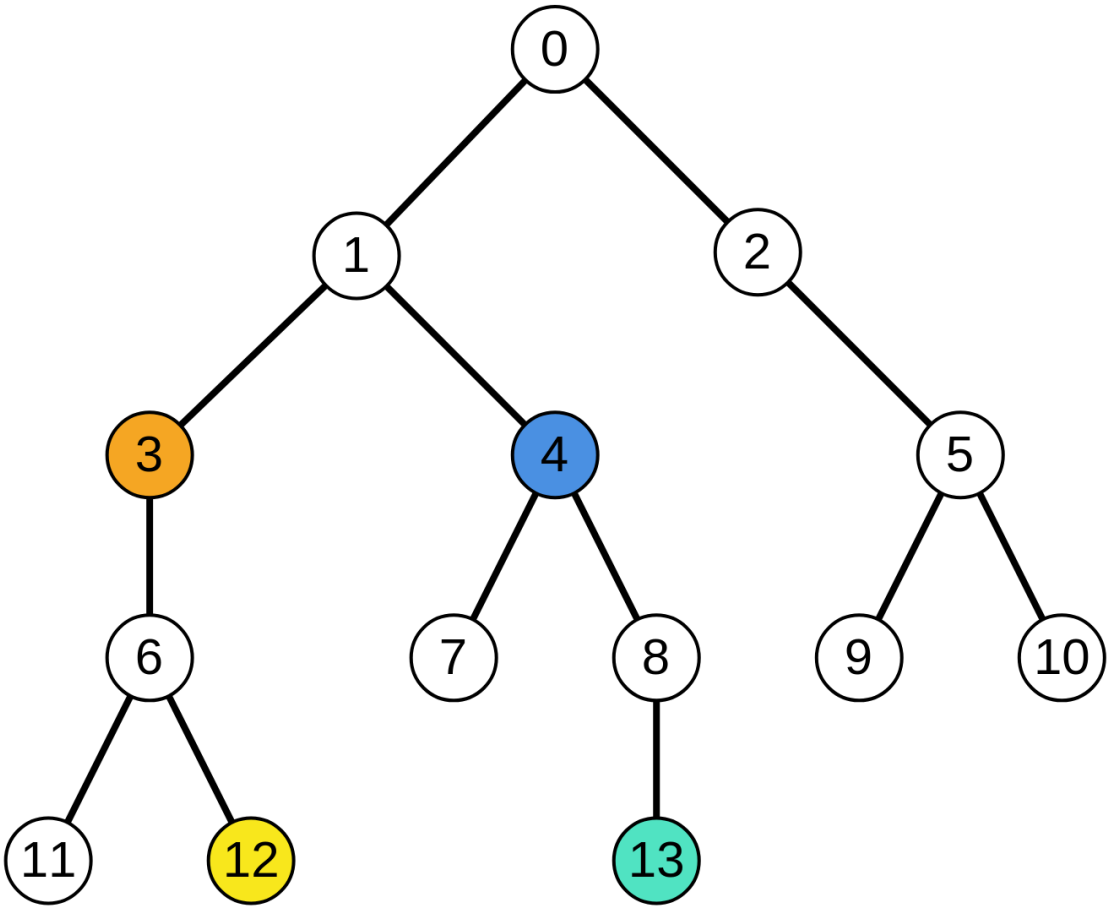
Esempio



- $u = 12$, $v = 13$
- $j = 2$, $h = 2^j = 4$
- $lca[u][j] = 0$, $lca[v][j] = 0$

$\Rightarrow$ u e v non vengono modificati

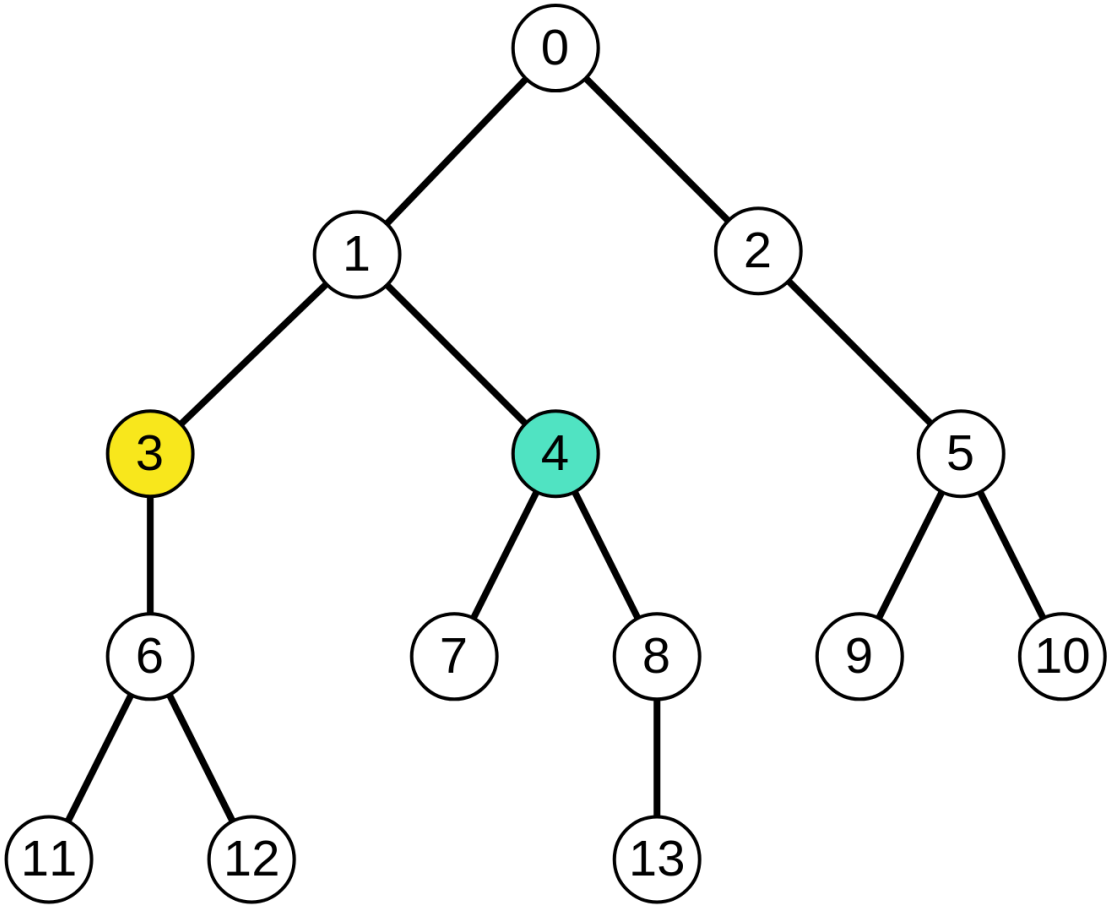
Esempio



- $u = 12$, $v = 13$
- $j = 1$, $h = 2^j = 2$
- $lca[u][j] = 3$,
  $lca[v][j] = 4$

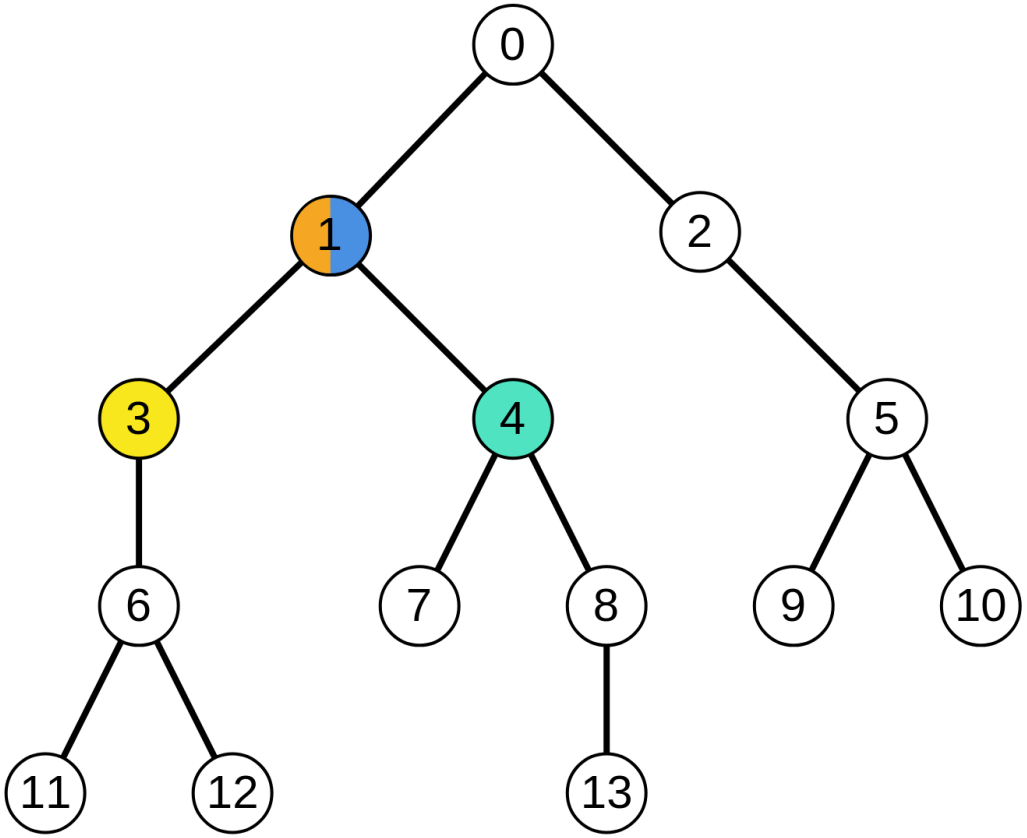
$\Rightarrow$ facciamo "salire" i nodi:
$u = 3$ e $v = 4$

Esempio



- $u = 3$, $v = 4$

Esempio



- $u = 3$, $v = 4$
- $j = 0$, $h = 2^j = 1$
- $lca[u][j] = 1$, $lca[v][j] = 1$

$\Rightarrow$ u e v non vengono modificati

L'antenato cercato è il padre degli attuali u e v, quindi $lca(12, 13) = 1$.
La distanza cercata è
$d(12, 13) = h(12) + h(13) - 2h(lca(12, 13)) = 4 + 4 - 2 = 6$.

Complessità

- ⇒ soluzione: `tree.cpp`
- ⇒ complessità: $O(N \log N + Q \log N)$
- ⇒ 50 punti

Nota

Ci sono diverse implementazioni possibili di lca, ad esempio si veda: `https://www.topcoder.com/community/competitive-programming/tutorials/range-minimum-query-and-lowest-common-ancestor`.

Sottoproblema: alberi con una cricca

Idea: cercare l'unica cricca presete nel grafo, eliminarla come descriveremo nel caso generale ed utilizzare lo stesso algoritmo precedente (lca) per rispondere alle richieste.

Ricerca della cricca: la cricca può essere trovata utilizzando uno degli algoritmi per la ricerca delle componenti fortemente connesse sul grafo generato dalla visita bfs.

Complessità

⇒ soluzione: `treewithclique.cpp`

⇒ complessità: $O(M + N \log N + Q \log N)$

⇒ 70 punti

Caso generale

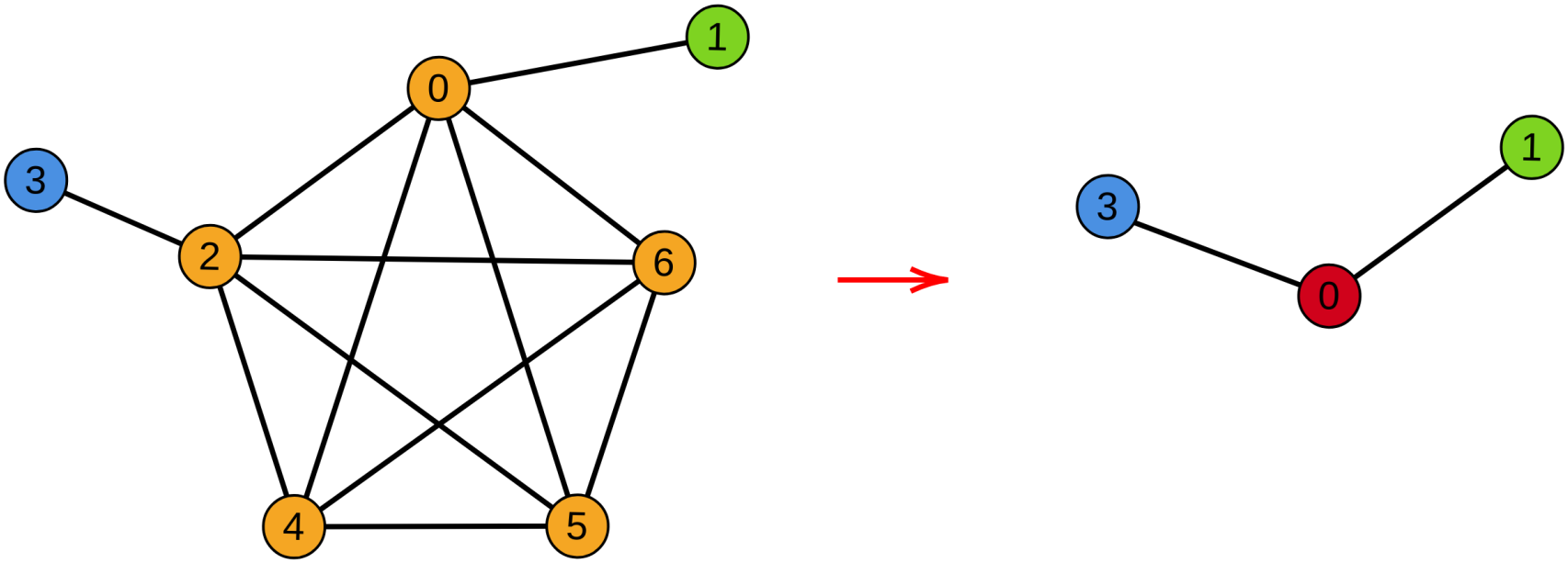
Idee:

- Eliminare le cricche massimali, in modo da rendere il grafo un albero.
- Una volta costruito l'albero, utilizzare l'algoritmo di LCA.

Cricca massimale

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

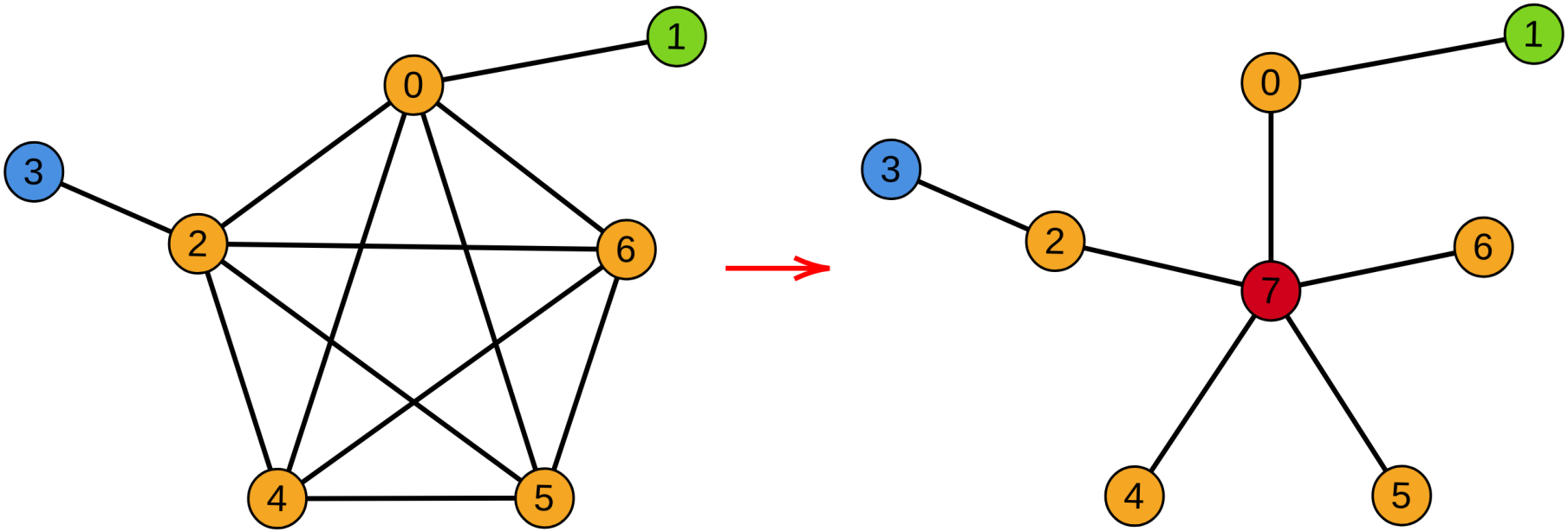
Esempio (eliminazione cricche sbagliata)



Idea: identificare con un unico nodo tutti i nodi parte di una cricca massimale.

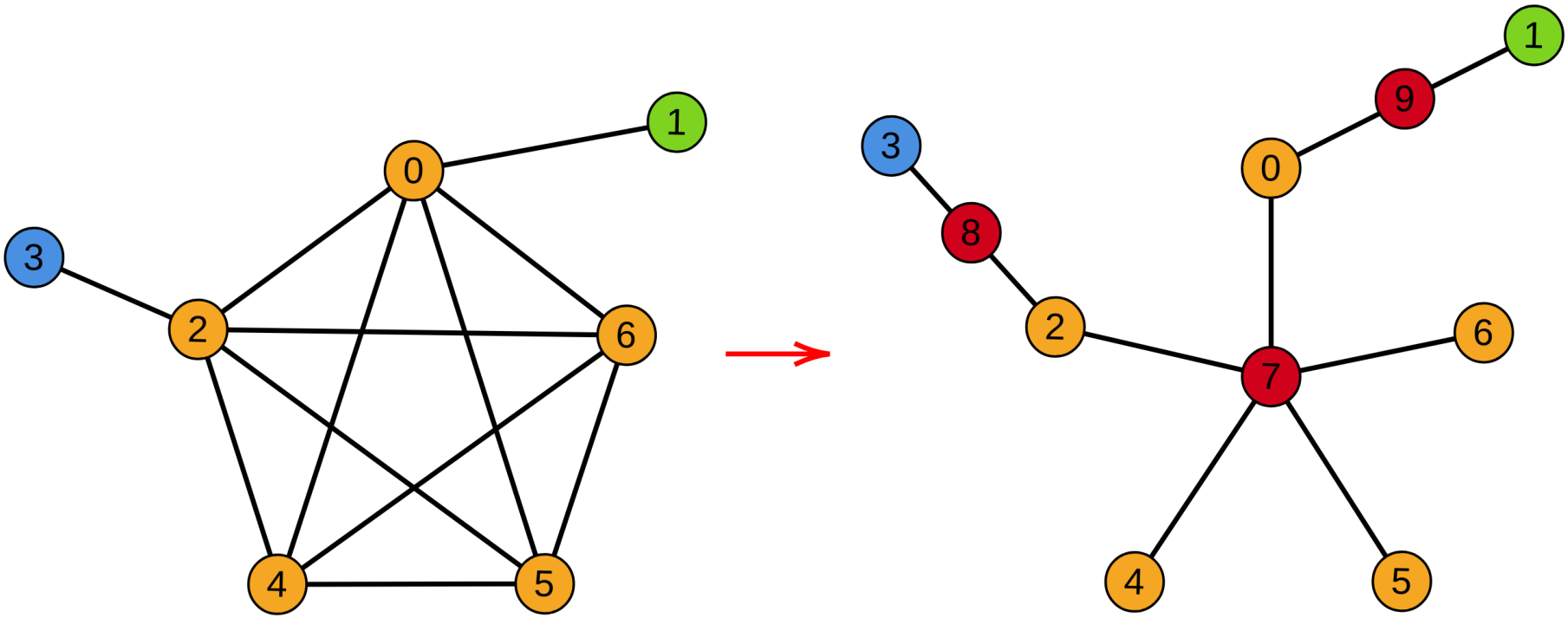
Problema: le distanze tra i nodi sono state modificate e non è più possibile recuperarle.

Esempio (eliminazione cricche corretta)



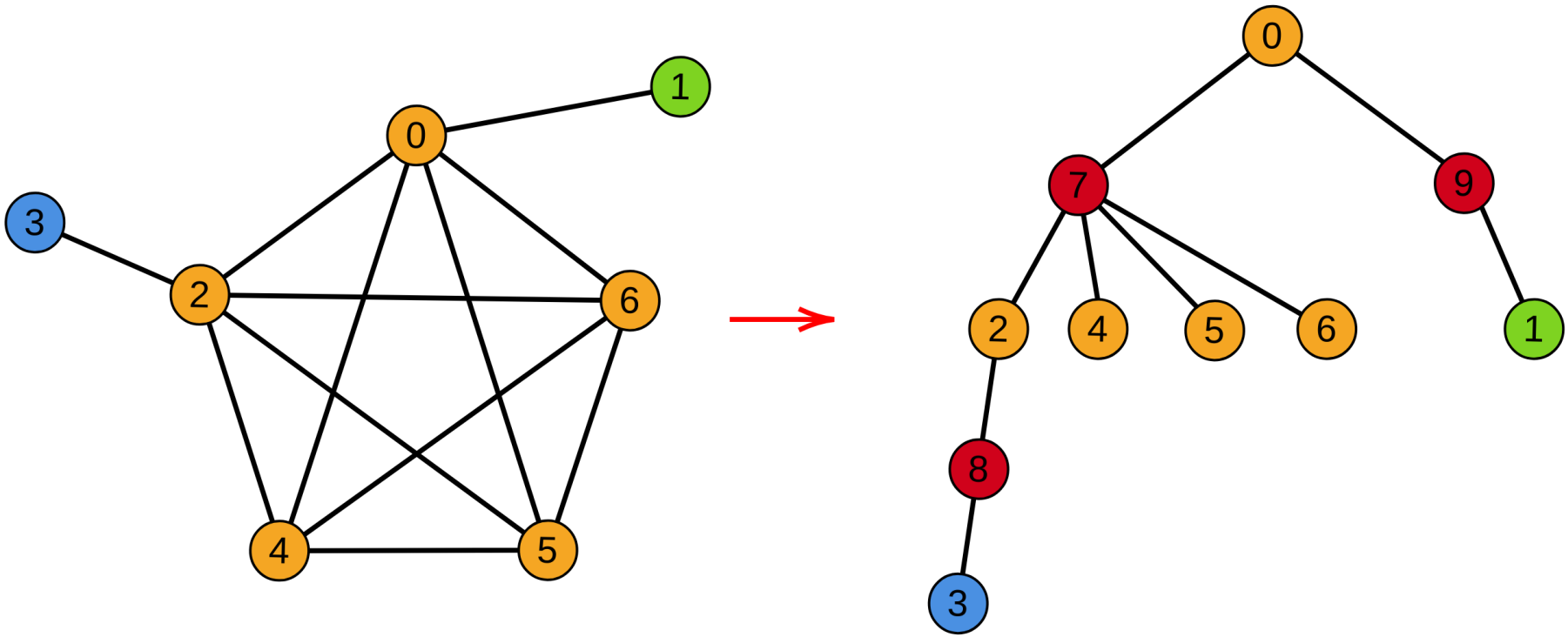
Idea: costruire un grafo in cui si collegano tutti i nodi parte della stessa cricca massimale ad un nuovo nodo "centrale".
Osservazione: ora le distanze tra i nodi della cricca sono raddoppiate!

Esempio (costruzione albero)



Le cricche massimali possono essere costituite anche da due soli elementi. Se aggiungiamo un nodo "centrale" in **ogni** cricca massimale... **Tutte** le distanze tra i nodi del grafo iniziale vengono raddoppiate!

Esempio (costruzione albero)



Abbiamo così trasformato il grafo $G = (V_G, E_G)$ di partenza in un albero $T = (V_T, E_T)$ con la seguente proprietà.

$$d_G(u, v) = \frac{1}{2} d_T(u, v) \quad \forall u, v \in V_G.$$

Soluzione

- Per ogni cricca massimale rimuoviamo tutti gli archi tra i suoi nodi e li colleghiamo ad un nuovo singolo nodo.

$\Rightarrow$ Questa operazione è ben definita perché due cricche massimali non possono avere archi in comune. Infatti, se due cricche hanno un arco in comune, allora una è contenuta nell'altra.

- Il grafo che otteniamo è un albero, in cui vale la seguente proprietà:

$$d_G(u, v) = \frac{1}{2} d_T(u, v) \quad \forall u, v \in V_G.$$

- Utlizziamo l'algoritmo visto per il caso dell'albero (LCA con preprocessing) e rispondiamo alle richieste.

Costruzione dell'albero

L'unico problema che rimane la costruzione dell'albero in modo efficiente. Le cricche possono essere trovate con un'unica visita, utilizzando l'algoritmo di Tarjan per la ricerca dei bridges: `https://judge.science.unitn.it/slides/asd16/sol_prog1.pdf`.

Complessità

- ⇒ soluzione: `sol.cpp`
- ⇒ complessità: $O(M + N \log N + Q \log N)$
- ⇒ 100 punti