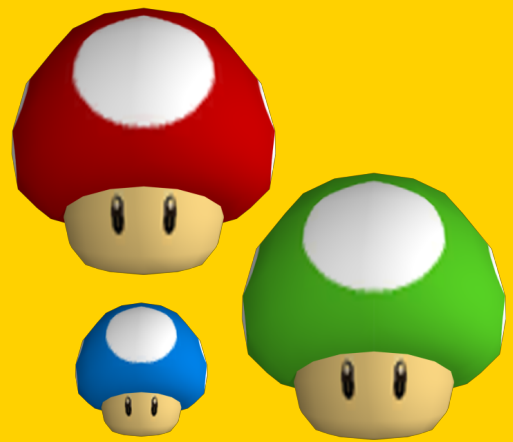
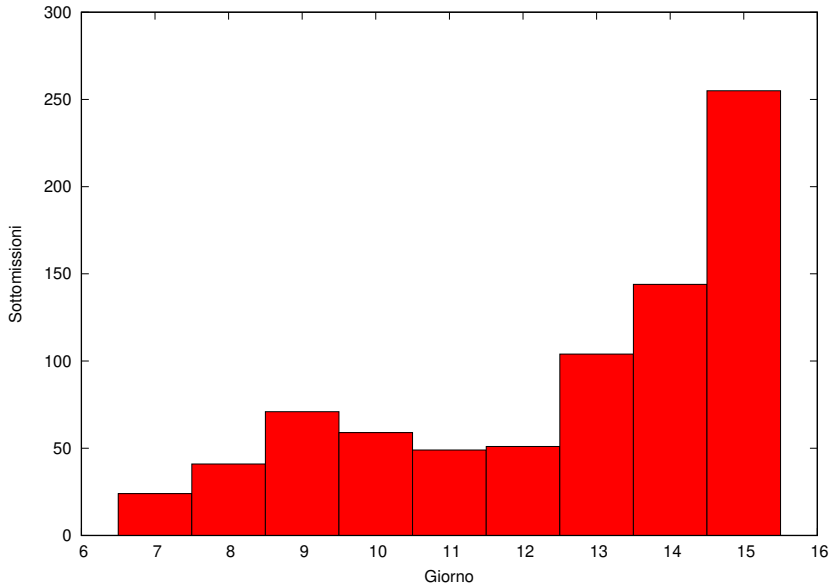
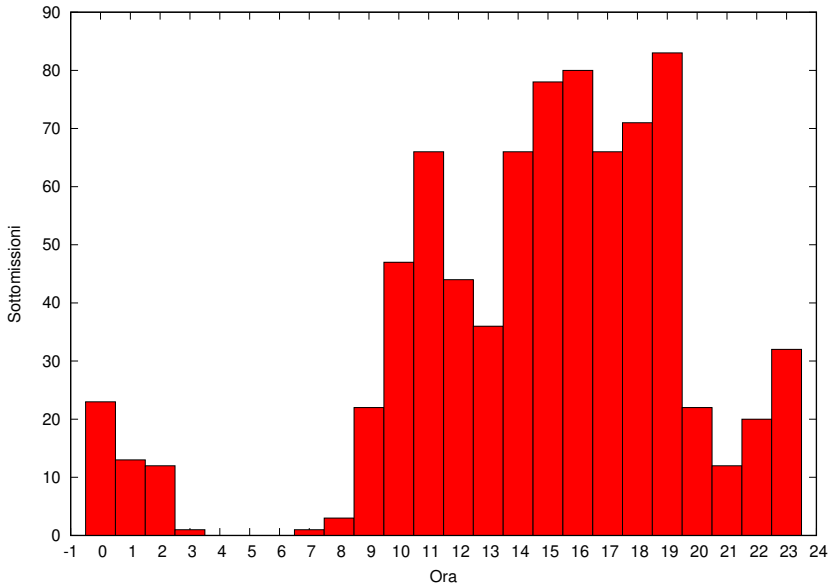
SUPER MARIO GRAPH



Statistiche

Numero sottoposizioni: 798





- 75 gruppi hanno fatto almeno una sottoposizione, di cui 74 hanno raggiunto la sufficienza;
- 151 studenti iscritti, di cui 150 appartenenti a gruppi che hanno fatto almeno una sottoposizione;

Risultati

Punteggi

- $P < 30$ $\longrightarrow$ progetto non passato
- $P = 30$ $\longrightarrow$ 1 punto bonus (28 gruppi)
- $35 \leq P \leq 55$ $\longrightarrow$ 2 punti bonus (21 gruppi)
- $65 \leq P \leq 90$ $\longrightarrow$ 3 punti bonus (13 gruppi)
- $P = 100$ $\longrightarrow$ 3.5 punti bonus (12 gruppi)

Classifiche e sorgenti sul sito (controllate i numeri di matricola):
`https://judge.science.unitn.it/slides/asd21/classifica_prog1.pdf`

Il problema I

Consideriamo un grafo connesso e non orientato $G = (V, E)$ con N nodi ed M archi e P coppie (A, B) di nodi indicanti il fatto che un Power-Up deve essere trasportato dal nodo A e il nodo B.

Ogni Power-Up è consegnato da un Toad Postino, che partendo dal nodo A deve arrivare al nodo B.

Il problema II

Tuttavia, quando un arco viene attraversato in un verso, non può più essere attraversato nel senso inverso.

I Toad devono seguire il percorso migliore, in modo da non impedire agli altri Toad di consegnare i loro Power-Up, se esiste un modo di percorrere gli archi che lo permette.

Problema

Dire se è possibile consegnare tutti i Power-Up.

Scivoli arcobaleno

Definizione

Uno **scivolo arcobaleno** è un tubo $u \leftrightarrow v$ che, nel grafo originale, rappresenta **l'unico modo per raggiungere** v **partendo da** u.

In altre parole, uno scivolo arcobaleno è un arco che **non fa parte di nessun ciclo**.

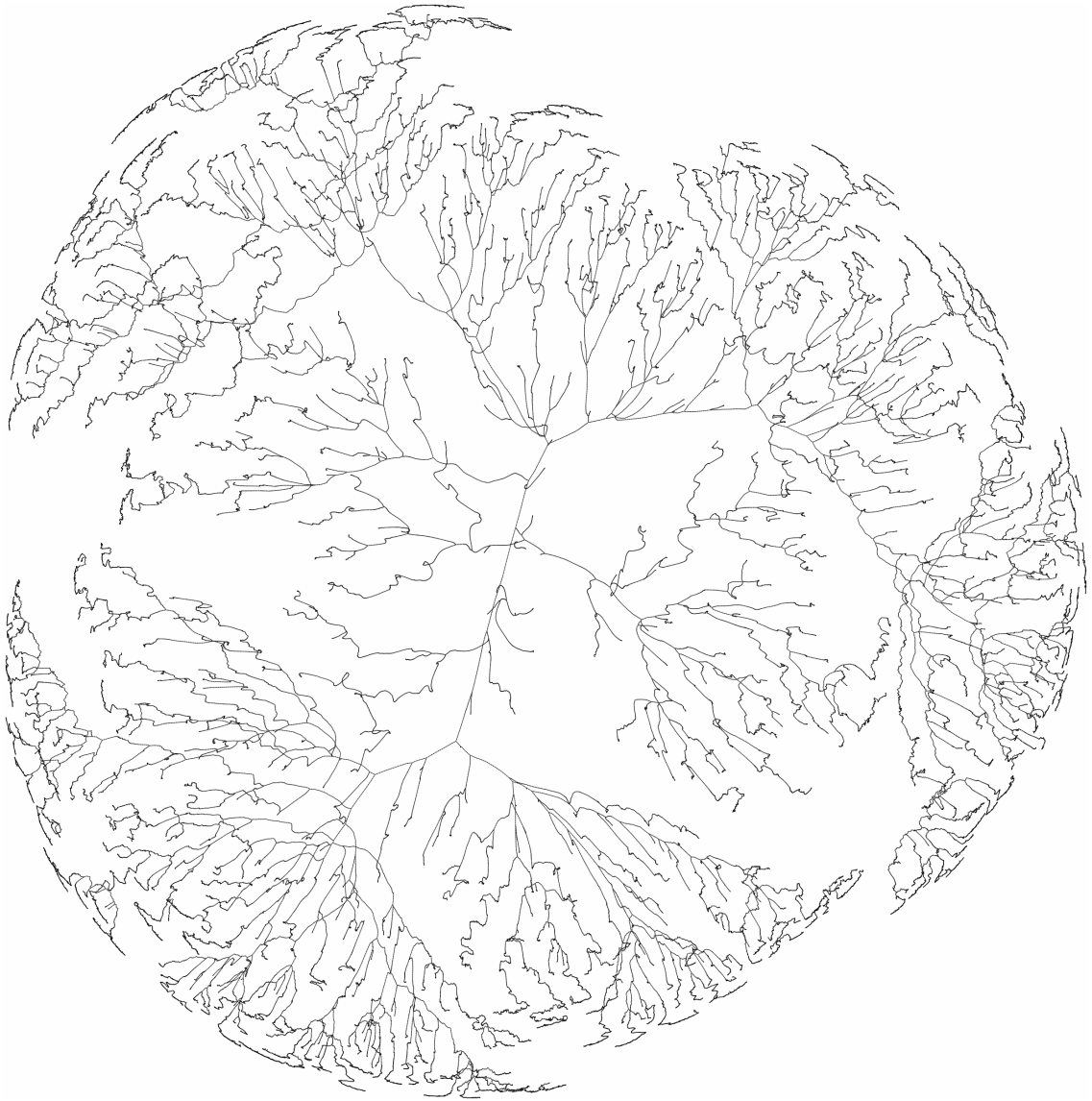
Bridge

Definizione

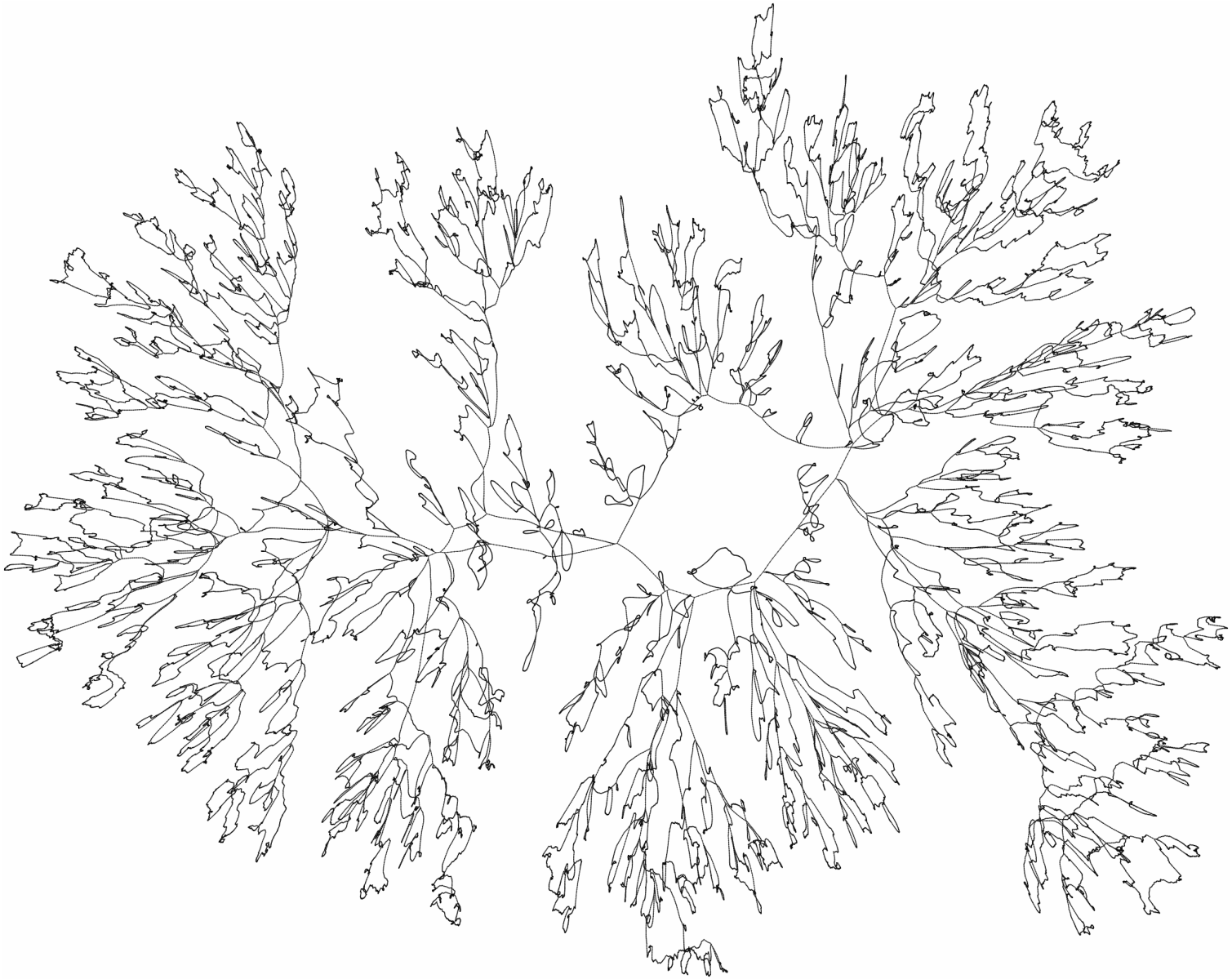
Un **bridge** è un arco del grafo G che, se rimosso, aumenta il numero di componenti connesse di G. Equivalentemente, un arco è un bridge **se e solo se non è contenuto in nessun ciclo**. [1]

[1] https://it.wikipedia.org/wiki/Ponte_(teoria_dei_grafi)

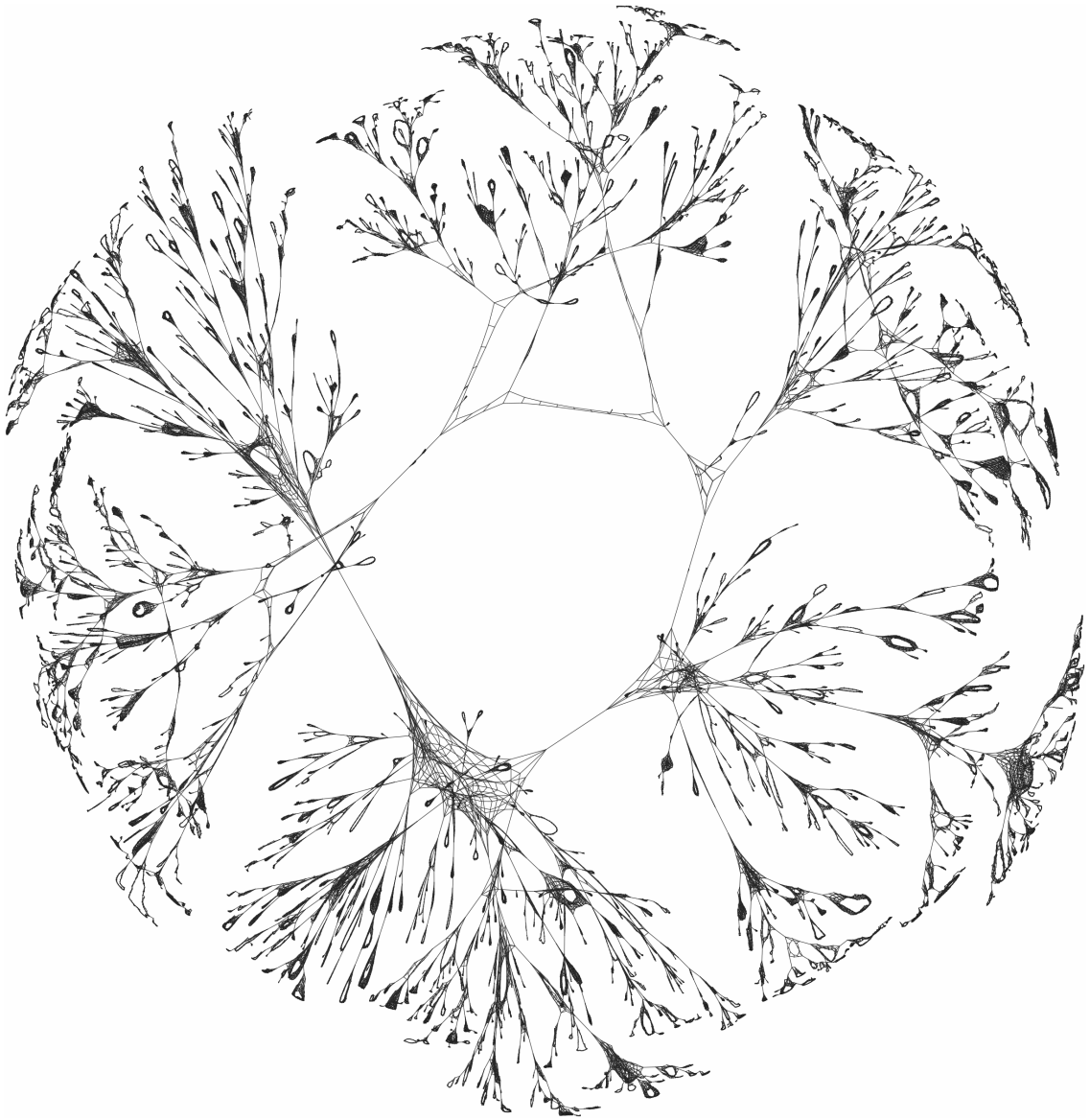
Testcase - Alberi con rami lunghi



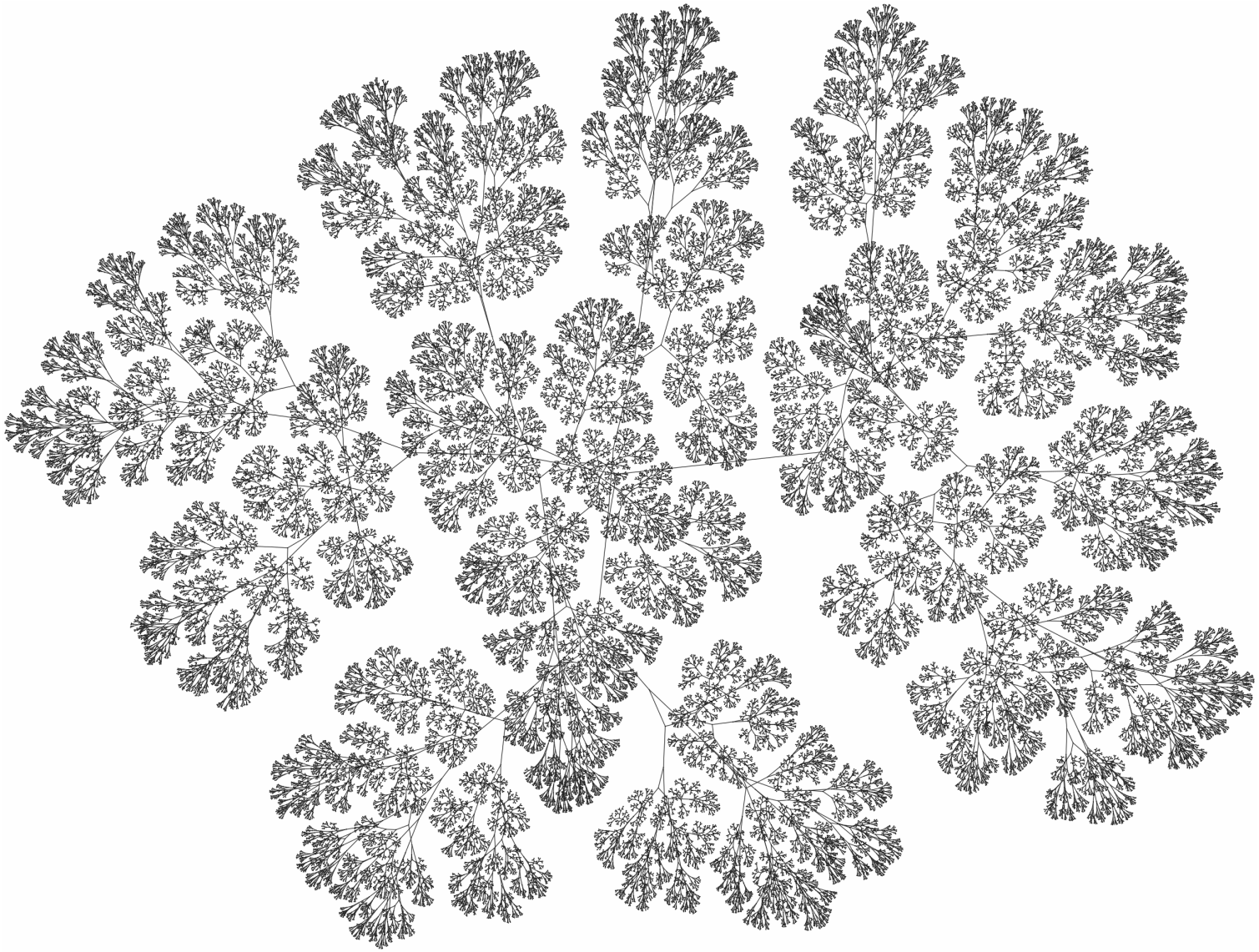
Testcase - Cicli ad anello



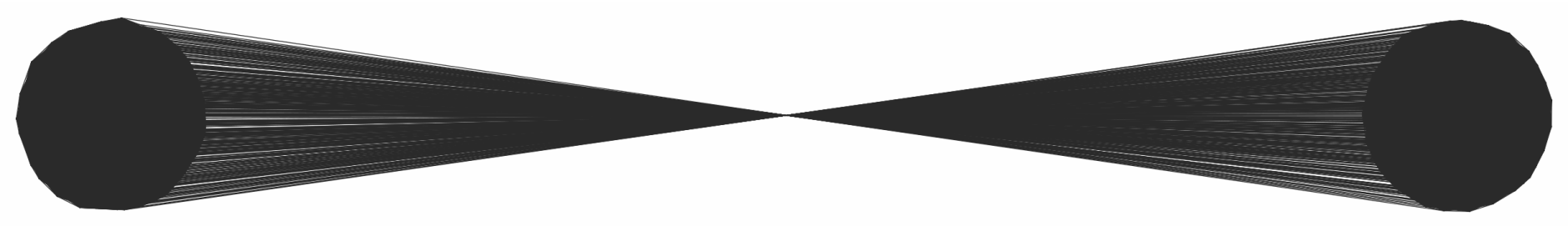
Testcase - Cicli a ragnatela



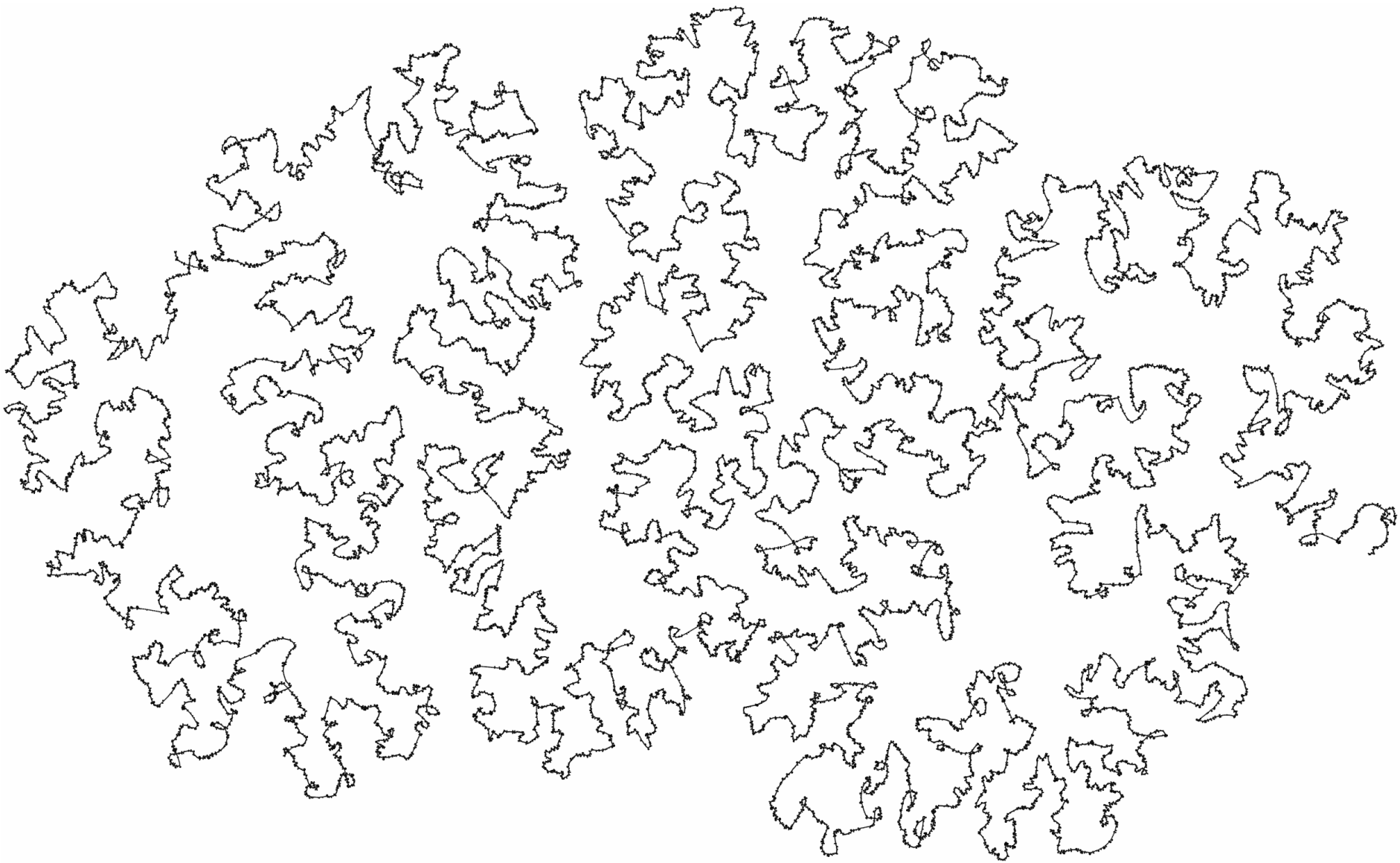
Testcase - Alberi bilanciati



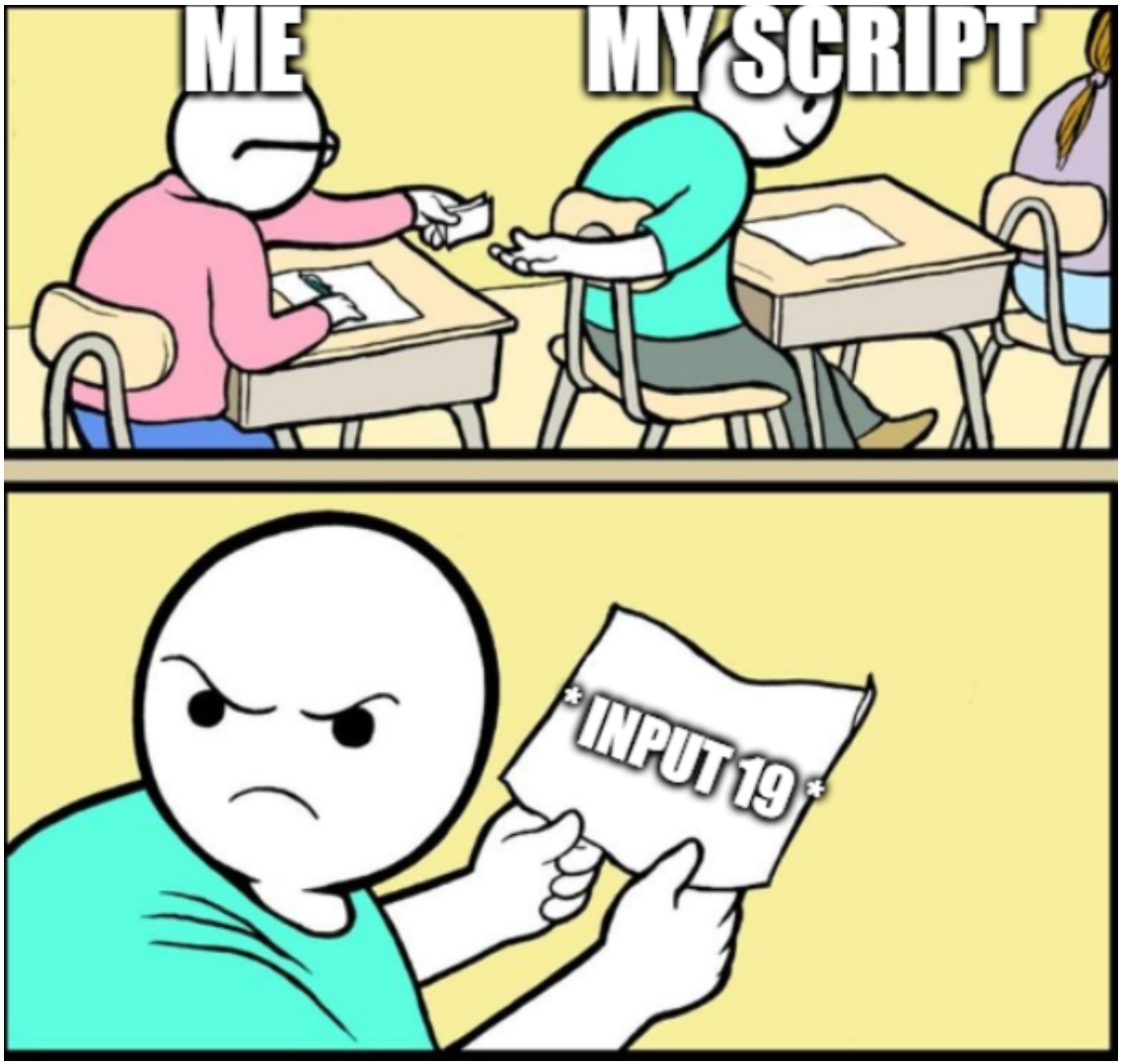
Testcase - Grafi senza bridge



Testcase - Alberi caterpillar



Testcase - Alberi caterpillar



Sottoproblema: alberi con $P = 1$

Osservazione

Se il grafo in input è un **albero**, tra ogni coppia di nodi esiste **un solo percorso**. Perciò, **tutti gli archi sono scivoli arcobaleno**.

Quindi, se abbiamo un solo Power-Up da consegnare dal nodo A al nodo B, è sufficiente fare una visita per trovare il percorso tra A e B e stampare gli archi attraversati.

Complessità

⇒ soluzione: `sol_30.cpp`

⇒ complessità: $O(N + M)$, 39 SLOC

⇒ 30 punti

Problema generale: Grafi

Nel caso il grafo in input non sia un albero, abbiamo un problema: per ogni PowerUp (A, B) può esserci più di un percorso tra A e B.

Come possiamo fare per essere sicuri che i percorsi che scegliamo siano sempre ottimi?

Osservazione

All'interno di un ciclo composto dai nodi V, è sempre possibile direzionare gli archi in modo che ogni nodo $v \in V$ possa raggiungere ogni altro nodo $u \in V$.

Problema generale: Grafi

Osservazione

All'interno di un ciclo composto dai nodi V, è sempre possibile direzionare gli archi in modo che ogni nodo $v \in V$ possa raggiungere ogni altro nodo $u \in V$.

Più in generale:

Robbins' theorem

È possibile orientare ogni arco di un (sotto)grafo non diretto G e renderlo **un'unica componente fortemente connessa** se e solo se G è connesso e **non contiene bridge**. [1]
È sufficiente orientare gli archi secondo il verso di percorrenza di una DFS partendo da un nodo qualsiasi.

[1] https://en.wikipedia.org/wiki/Robbins%27_theorem

Problema generale: Grafi

Come possiamo sfruttare il Robbins' theorem?

- orientiamo il grafo tramite una DFS partendo da un nodo qualsiasi;
- calcoliamo le **componenti fortemente connesse** dell'albero costruito dalla DFS (tramite Kosaraju o Tarjan);
- gli archi che collegano tra loro le componenti fortemente connesse sono **bridge nel grafo originale**.

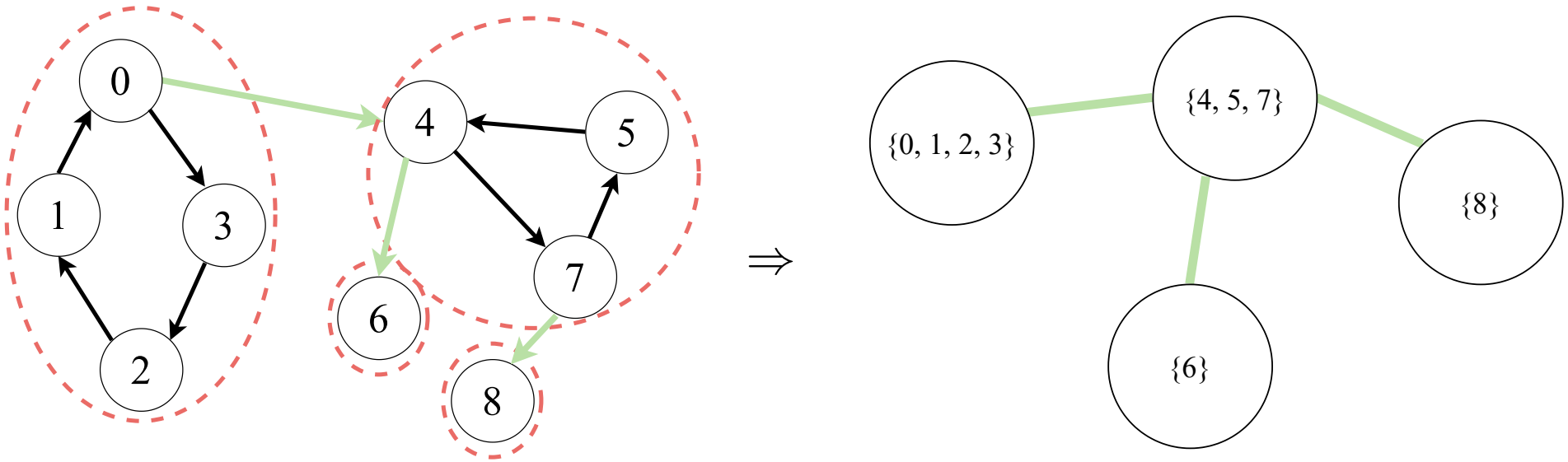
Alternativamente possiamo usare l'algoritmo di Tarjan per trovare i bridge in un grafo non orientato. [1]

Entrambi questi algoritmi hanno complessità $O(N + M)$.

[1] https://it.wikipedia.org/wiki/Ponte_(teoria_dei_grafi)#Algoritmo_per_la_ricerca_di_ponti

Bridge Tree

Chiamiamo Bridge Tree il nuovo albero formato dalle componenti fortemente connesse e dai bridge che le collegano.



Problema generale: Grafi

Ora che abbiamo il *Bridge Tree* del grafo originale, siamo ritornati al problema su alberi. Possiamo quindi sfruttare il fatto che, per ogni PowerUp (A, B), c'è **un solo percorso** tra A e B.

Questo ci obbliga a percorrere quel percorso, e non dobbiamo più preoccuparci di sapere se il nostro percorso è quello giusto o se avessimo potuto sceglierne uno migliore.

Quindi, per ogni Power-Up (A, B), percorriamo il percorso da A a B e controlliamo di volta in volta di non passare su archi direzionati nel verso opposto.

Complessità

⇒ soluzione: `sol_NP_simul_on_tree.cpp`

⇒ complessità: $O(P \cdot (N + M))$, 208 SLOC

⇒ 75 punti

Si può fare meglio di così?

Soluzione generale

Il nostro obiettivo è quello di migliorare la complessità della soluzione precedente. Vogliamo evitare di dover fare una visita dell'intero albero per ogni Power-Up.

Per prima cosa, scegliamo un nodo radice, in modo da avere un albero radicato.

Soluzione generale

Idea

Dato un nodo n, se sappiamo:

- (**a**) Quanti PU devono partire da un nodo del suo sottoalbero;
- (**b**) Quanti PU devono arrivare in un nodo del suo sottoalbero;
- (**c**) Quanti percorsi sono interamente contenuti nel suo sottoalbero.

Allora sappiamo come orientare l'arco da n verso il parent:

Soluzione generale

Idea

Dato un nodo n, se sappiamo:

- (**a**) Quanti PU devono partire da un nodo del suo sottoalbero;
- (**b**) Quanti PU devono arrivare in un nodo del suo sottoalbero;
- (**c**) Quanti percorsi sono interamente contenuti nel suo sottoalbero.

Allora sappiamo come orientare l'arco da n verso il parent:

- (**a** $>$ **c** & **b** $<$ **c**) c'è almeno un PU che deve uscire dal sottoalbero $\rightarrow$ oriento l'arco verso l'alto;

Soluzione generale

Idea

Dato un nodo n, se sappiamo:

- (**a**) Quanti PU devono partire da un nodo del suo sottoalbero;
- (**b**) Quanti PU devono arrivare in un nodo del suo sottoalbero;
- (**c**) Quanti percorsi sono interamente contenuti nel suo sottoalbero.

Allora sappiamo come orientare l'arco da n verso il parent:

- (**a** $>$ **c** & **b** $<$ **c**) c'è almeno un PU che deve uscire dal sottoalbero $\rightarrow$ oriento l'arco verso l'alto;
- (**a** $<$ **c** & **b** $>$ **c**) c'è almeno un PU che deve entrare nel sottoalbero $\rightarrow$ oriento l'arco verso il basso;

Soluzione generale

Idea

Dato un nodo n, se sappiamo:

- (**a**) Quanti PU devono partire da un nodo del suo sottoalbero;
- (**b**) Quanti PU devono arrivare in un nodo del suo sottoalbero;
- (**c**) Quanti percorsi sono interamente contenuti nel suo sottoalbero.

Allora sappiamo come orientare l'arco da n verso il parent:

- (**a** $>$ **c** & **b** $<$ **c**) c'è almeno un PU che deve uscire dal sottoalbero $\rightarrow$ oriento l'arco verso l'alto;
- (**a** $<$ **c** & **b** $>$ **c**) c'è almeno un PU che deve entrare nel sottoalbero $\rightarrow$ oriento l'arco verso il basso;
- (**a** $>$ **c** & **b** $>$ **c**) ci sono sia PU che devono entrare, sia PU che devono uscire $\rightarrow$ l'arco verso il parent ha un conflitto;

Soluzione generale

Idea

Dato un nodo n, se sappiamo:

- (**a**) Quanti PU devono partire da un nodo del suo sottoalbero;
- (**b**) Quanti PU devono arrivare in un nodo del suo sottoalbero;
- (**c**) Quanti percorsi sono interamente contenuti nel suo sottoalbero.

Allora sappiamo come orientare l'arco da n verso il parent:

- ($\mathbf{a} > \mathbf{c}$ & $\mathbf{b} < \mathbf{c}$) c'è almeno un PU che deve uscire dal sottoalbero $\rightarrow$ oriento l'arco verso l'alto;
- ($\mathbf{a} < \mathbf{c}$ & $\mathbf{b} > \mathbf{c}$) c'è almeno un PU che deve entrare nel sottoalbero $\rightarrow$ oriento l'arco verso il basso;
- ($\mathbf{a} > \mathbf{c}$ & $\mathbf{b} > \mathbf{c}$) ci sono sia PU che devono entrare, sia PU che devono uscire $\rightarrow$ l'arco verso il parent ha un conflitto;
- ($\mathbf{a} = \mathbf{c}$ & $\mathbf{b} = \mathbf{c}$) nessuno PU deve uscire o entrare $\rightarrow$ è indifferente come oriento l'arco, non viene percorso.

Soluzione generale

Come ricaviamo queste informazioni?

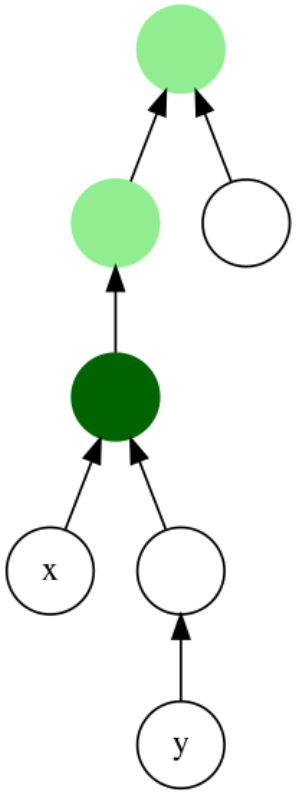
Idea

Dato un nodo n:

- (**a**) Quanti PU devono partire da un nodo del suo sottoalbero $\rightarrow$ **facile**, per ogni nodo u del sottoalbero di n sommiamo le volte in cui u è la partenza di un PU;
- (**b**) Quanti PU devono arrivare in un nodo del suo sottoalbero $\rightarrow$ **facile**, simmetrico ad (**a**);
- (**c**) Quanti percorsi sono interamente contenuti nel suo sottoalbero $\rightarrow$ **difficile**.

Soluzione generale

Abbiamo ridotto il problema da risolvere al seguente:
Quanti percorsi sono interamente contenuti nel sottoalbero del nodo *n*?



Lowest Common Ancestor

In un albero, il *Lowest Common Ancestor* di due nodi *x* e *y* è il nodo più profondo che ha *x* e *y* come discendenti.

`https://en.wikipedia.org/wiki/Lowest_common_ancestor`

Soluzione generale

Per ogni nodo u del sottoalbero di n calcoliamo quante volte u è il Lowest Common Ancestor del percorso di un PU.

Assumiamo di sapere quante volte u è LCA di un PU...

(c) Quanti percorsi sono interamente contenuti nel sottoalbero radicato in $n \rightarrow$ **facile**, per ogni nodo u del sottoalbero di n sommiamo le volte in cui u è LCA del percorso di un PU.

Abbiamo ritrasformato il nostro problema! Come facciamo a sapere quante volte un nodo è LCA di un PU?

SOLUZIONE GENERALE

```
vector<int> a(N, 0), b(N, 0), c(N, 0);
for (pair<int, int>& p: powerups) {
    a[p.first]++;
    b[p.second]++;
    c[lca(p.first, p.second)]++;
}
dfs(root, -1);
```

```
void dfs(int u, int parent) {
    for (int v : tree[u]) {
        if (v != parent) {
            dfs(v, u);
            a[u] += a[v]; b[u] += b[v]; c[u] += c[v];
        }
    }
    // qui posso orientare l'arco verso il parent
}
```

Soluzione generale

Prima avevamo un problema, poi l'abbiamo trasformato. Ora lo riduciamo ancora.

- Come capisco in che verso orientare gli archi?
- Quanti percorsi sono contenuti nel sottoalbero di un nodo n?
- Come facciamo a sapere quante volte un nodo è LCA di un PU?
- Come troviamo gli LCA di ogni PU in modo efficiente?

Soluzione Generale

Come troviamo gli LCA in modo efficiente senza visitare l'albero per ogni PU?
Esistono diversi modi per farlo: se ben implementati, tutti i seguenti algoritmi garantivano 100 punti:

- LCA off-line di Tarjan [2] $\rightarrow O(N + P)$
- LCA con Sparse Table RMQ[3] $\rightarrow O(N \cdot \log N + P)$
- LCA Binary Lifting[4] $\rightarrow O(N \cdot \log N + P \cdot \log N)$

Nota: con N si intende il numero di nodi del bridge-tree.

[2] https://cp-algorithms.com/graph/lca_tarjan.html
[3] https://cp-algorithms.com/data_structures/sparse-table.html
[4] https://cp-algorithms.com/graph/lca_binary_lifting.html

Complessità

- ⇒ soluzione: `sol_N+P_lca_offline.cpp`
- ⇒ complessità: $O(N + M + P)$ (0.56s, 27MB), 228 SLOC
- ⇒ soluzione: `sol_NlogN+P_lca_rmq.cpp`
- ⇒ complessità: $O(N \cdot \log N + M + P)$ (0.94s, 44MB), 278 SLOC
- ⇒ 100 punti