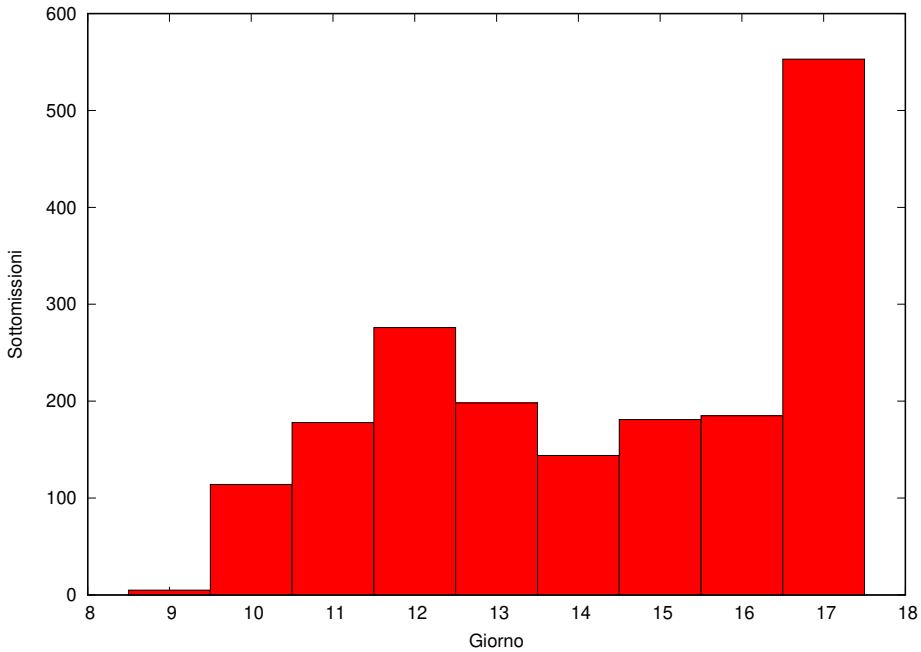
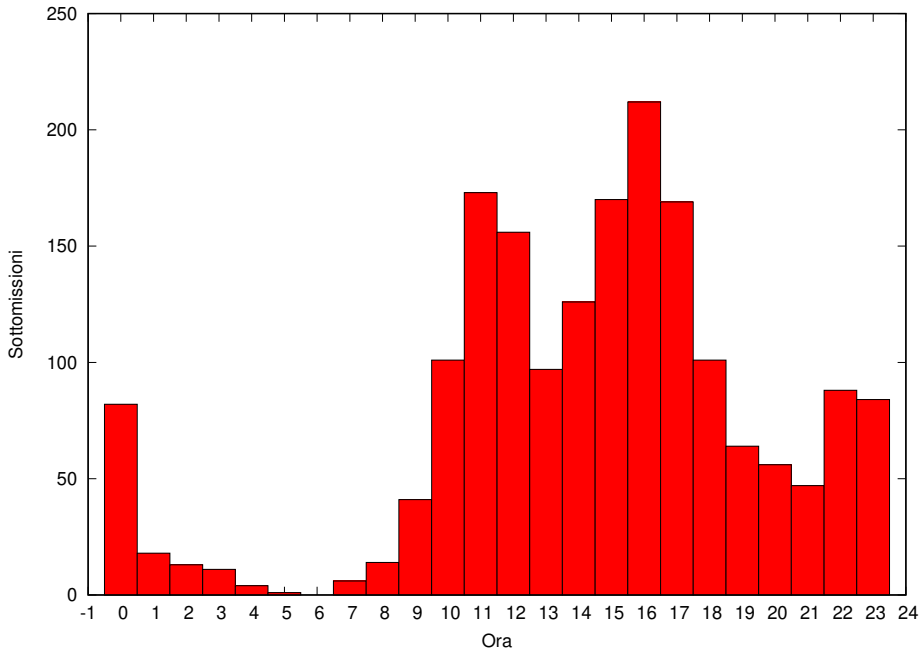
Primo Progetto ASD 2024/2025

Statistiche

Numero sottoposizioni: `1834` (record)





- 66 gruppi hanno fatto almeno una sottoposizione, di cui 61 hanno raggiunto la sufficienza;
- 161 studenti iscritti, di cui 157 appartenenti a gruppi che hanno fatto almeno una sottoposizione;

Risultati

Punteggi

- $P < 30$ $\longrightarrow$ progetto non passato.
- $30 \leq P < 50$ $\longrightarrow$ 1 punto bonus (18 gruppi).
- $50 \leq P < 90$ $\longrightarrow$ 2 punti bonus (19 gruppi).
- $P \geq 90$ $\longrightarrow$ 3 punti bonus (23 gruppi).

Classifiche e sorgenti sul sito (controllate i numeri di matricola):
https://asdlab.science.unitn.it/asd24/classifica_prog1.pdf

Problema

Dato un grafo orientato $G = (V, E)$, dove V sono i vertici rappresentanti le stanze, ed E sono gli archi rappresentanti i corridoi.

Trovare, se esiste, un nodo $v \in V$ contenuto in tutti i cicli di G.

- Se esiste, stampare anche un ciclo che lo contiene.
- Altrimenti, stampare due cicli e un nodo che appartiene solo a uno dei due.

Soluzione 30 punti

Assunzione

Per ogni coppia di percorsi infiniti senza stanze ripetute,
nessuna stanza è contenuta in entrambi i percorsi.

In questi testcase tutti i cicli sono disgiunti:

- Se esiste un solo ciclo, allora ogni suo nodo è una soluzione.
- Se esistono più cicli, allora non c'è un nodo contenuto in tutti i cicli.

Inoltre, con questa assunzione, i cicli corrispondono alle SCC con almeno due nodi.

È sufficiente trovare le SCC e verificare se ne esiste una sola con almeno due nodi. In caso affermativo, ogni suo nodo è una soluzione.

Complessità

⇒ soluzione: `sol-30.cpp`

⇒ complessità:

$$O(N + M)$$

⇒ 30 punti

Soluzione Intermedia (I)

Osservazione

Se un nodo fa parte di tutti i cicli, allora rimuovendo quel nodo il grafo diventa aciclico.

- Proviamo a togliere ogni nodo e controlliamo se rimangono cicli.

```
for (int u = 0; u < N; u++) {
  vector<int> cycle = find_cycle_without(G, u);
  if (cycle.empty())
    return u;
}
return -1;
```

Soluzione Intermedia (II)

Osservazione

Dato un ciclo C, se esiste un nodo u che fa parte di tutti i cicli, allora u è contenuto in C.

- Troviamo un ciclo e proviamo a togliere solo i nodi di questo ciclo.

```
vector<int> cycle1 = find_cycle_without(G, -1);
for (int u : cycle1) {
  vector<int> cycle2 = find_cycle_without(G, u);
  if (cycle2.empty())
    return u;
}
return -1;
```

Complessità

⇒ soluzione: `sol-naive.cpp`

⇒ complessità:

$$O(N \cdot (N + M))$$

⇒ $\approx$ 90 punti.
$\approx$ 120 punti aggiungendo ottimizzazioni.

- L'obiettivo è minimizzare il numero di nodi da testare.

Verso la soluzione

Idea: dato un ciclo C, possiamo **escludere dei nodi che sicuramente non fanno parte di tutti i cicli**.

Troviamo un ciclo qualsiasi C che chiameremo il "**ciclo principale**" e cerchiamo i **cicli che lo intersecano**, ovvero che hanno almeno un nodo in comune con C.
Solo i nodi che appartengono a tutti questi cicli sono dei candidati ad essere soluzioni.

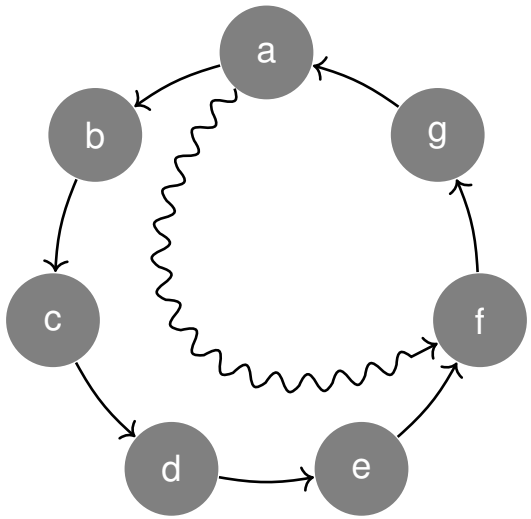
Alla fine, scegliamo uno qualsiasi dei nodi rimasti e controlliamo se è una soluzione, rimuovendolo dal grafo e verificando che non rimangano cicli.
Altrimenti, non esiste una soluzione.

Intersezioni con il ciclo principale

Osservazione

Dato un ciclo principale C, i cicli che lo intersecano:

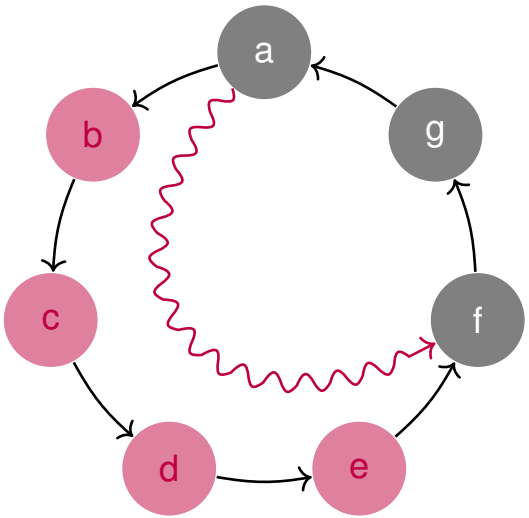
- "Lasciano" il ciclo principale da un nodo s (*source*).
- "Rientrano" nel ciclo principale in un nodo t (*target*).
- Percorrono gli archi di C dal nodo t al nodo s, nella direzione di C, possibilmente prendendo altre "deviazioni" lungo il percorso.



Intersezioni con il ciclo principale

Osservazione

Se un percorso "lascia" il ciclo principale C dal nodo s e rientra nel nodo t, allora tutti i nodi di C tra s e t (esclusi) non appartengono all'intersezione di tutti i cicli.



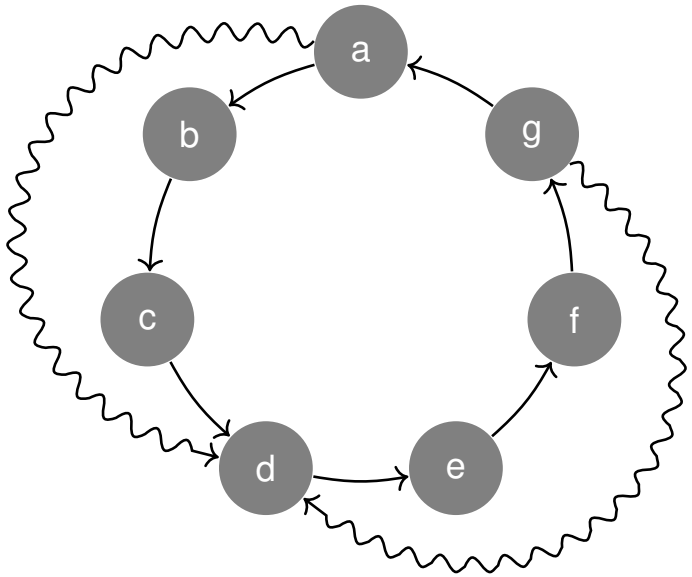
In questo esempio, dato il ciclo principale

$$a \to b \to c \to d \to e \to f \to g \to a$$

Esiste almeno un altro ciclo

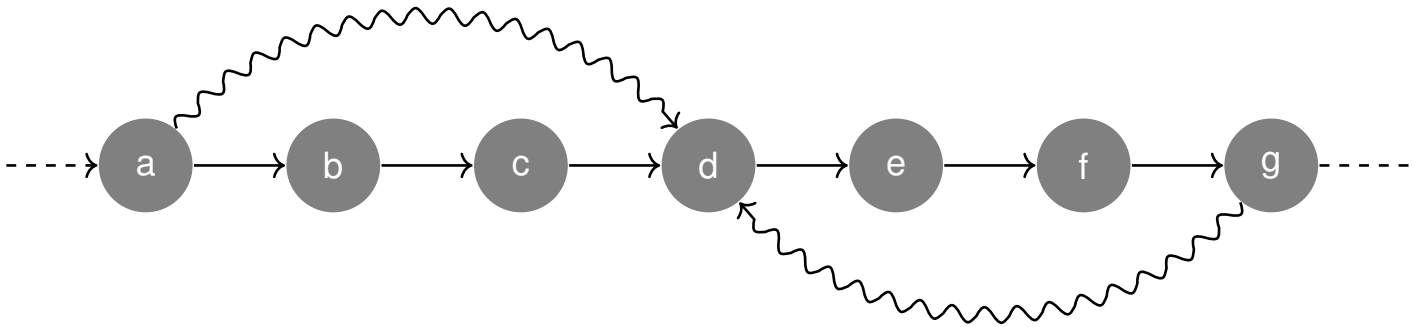
$$a \rightsquigarrow f \to g \to a$$

che non contiene i nodi b, c, d, e.



Dato un ciclo principale C, scegliamo un nodo come “primo nodo nel ciclo” e **definiamo un ordine**.

Chiamiamo $pos[n]$ la posizione del nodo n in C, rispetto al nodo scelto come primo.



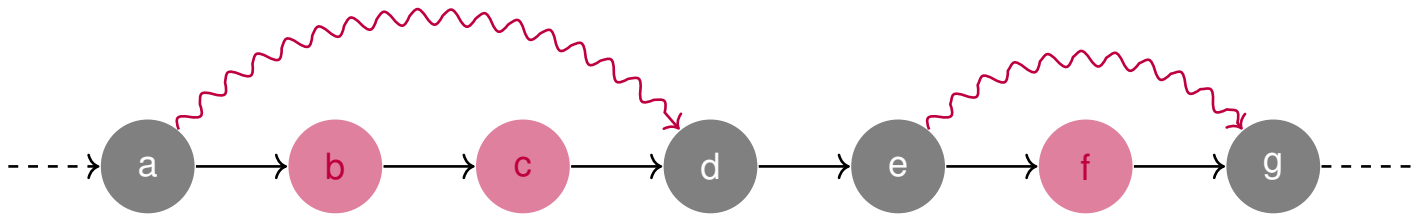
Salti in avanti

Chiamiamo "**salto in avanti**" $s \curvearrowright t$ un percorso che lascia il ciclo principale da un nodo s e rientra in un nodo t tale che $pos[s] < pos[t]$.

Osservazione

Dato $s \curvearrowright t$, **non sono soluzione** tutti in nodi n tali che

$$pos[s] < pos[n] < pos[t].$$



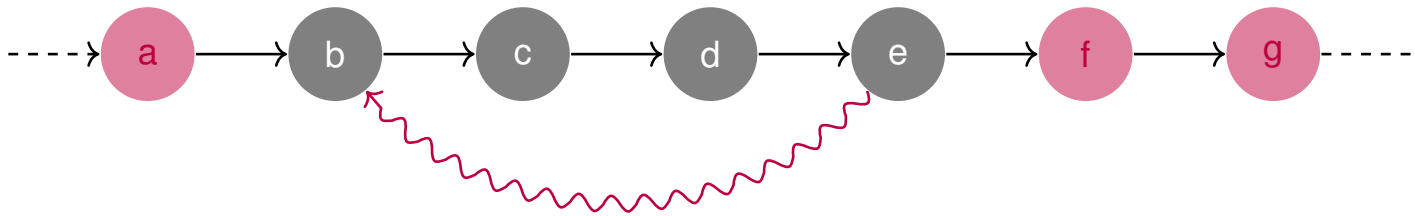
Salti all'indietro

Chiamiamo "**salto all'indietro**" $t \curvearrowleft s$ un percorso che lascia il ciclo principale da un nodo s e rientra in un nodo t tale che $pos[t] \leq pos[s]$.

Osservazione

Dato $t \curvearrowleft s$, **non sono soluzione** tutti in nodi n tali che

$$pos[n] < pos[t] \quad \text{oppure} \quad pos[n] > pos[s].$$



Salti all'indietro

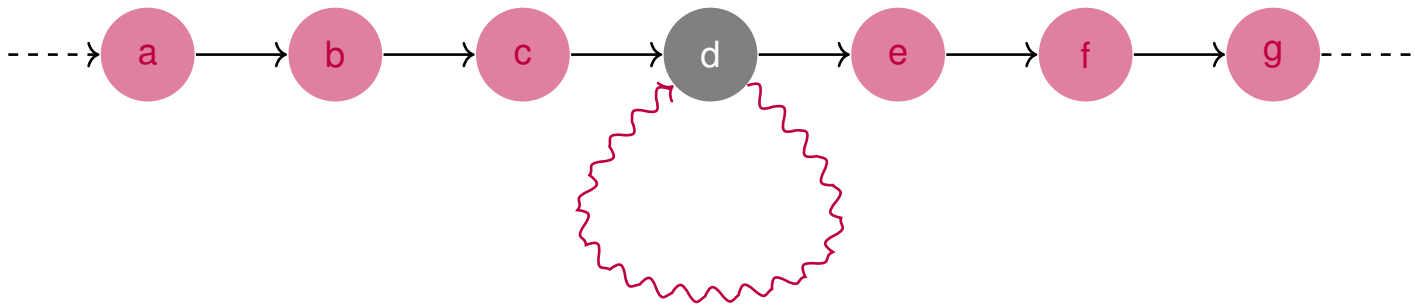
Chiamiamo "**salto all'indietro**" $t \curvearrowleft s$ un percorso che lascia il ciclo principale da un nodo s e rientra in un nodo t tale che $pos[t] \leq pos[s]$.

Osservazione

Dato $t \curvearrowleft s$, **non sono soluzione** tutti in nodi n tali che

$$pos[n] < pos[t] \quad \text{oppure} \quad pos[n] > pos[s].$$

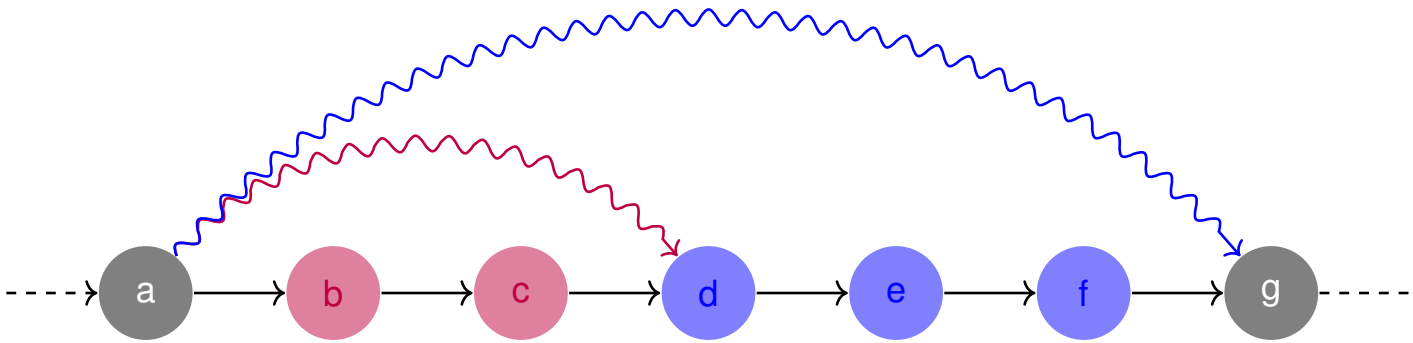
In particolare, se esiste $s \curvearrowleft s$ allora s è l'unico candidato.



TROVARE I SALTI IN AVANTI

Sia $max[n]$ il nodo di C con $pos[\,]$ massima raggiungibile dal nodo n.
Per ogni nodo s di C, se $pos[max[s]] > pos[s]$, allora esiste $s \curvearrowright max[s]$.

È possibile calcolare $max[\,]$ con una visita.



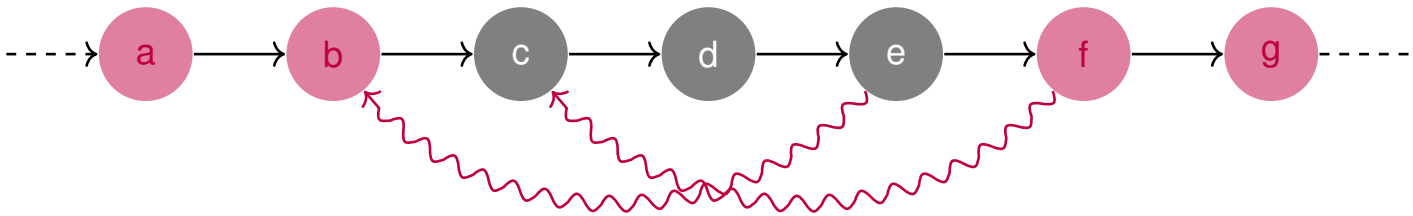
Trovare i salti all'indietro (I)

Per i salti all'indietro, ci basta trovare:

- Il nodo s^* di C con $pos[s^*]$ **minima** tale che $\exists t.\, t \curvearrowleft s^*$.
- Il nodo t^* di C con $pos[t^*]$ **massima** tale che $\exists s.\, t^* \curvearrowleft s$.

Non sono soluzione tutti i nodi n tali che:

$$pos[n] < pos[t^*] \text{ oppure } pos[s^*] < pos[n]$$



Trovare i salti all'indietro (II)

Trovare i nodi da cui partono dei salti all'indietro

Sia $min[n]$ il nodo di C con $pos[\,]$ minima raggiungibile dal nodo n.
Per ogni nodo s di C, se $pos[min[s]] \leq pos[s]$, allora esiste $min[s] \curvearrowleft s$.

È possibile calcolare $min[\,]$ con una visita.

Trovare i nodi in cui rientrano dei salti all'indietro

Un salto all'indietro rientra in un nodo t se t è raggiungibile da un nodo s tale che $pos[t] \leq pos[s]$.

Scorriamo i nodi di C in ordine inverso di $pos[\,]$, e da ognuno facciamo partire una visita, visitando solo i nodi non ancora visitati.
In questo modo, possiamo trovare tutti i nodi raggiungibili da un salto all'indietro, e prendere quello con $pos[\,]$ massima.

RIASSUMENDO

Troviamo un ciclo C.
Inizialmente tutti i nodi di C sono candidati.

- Definiamo un ordine dei nodi di C scegliendone uno come primo.
- Calcoliamo $min[\,]$ e $max[\,]$.
- Escludiamo i nodi saltati dai salti all'indietro:
  i candidati rimanenti sono quelli tra t^* e s^* (inclusi).
- Escludiamo i nodi saltati dai salti in avanti:
  per ogni nodo s in C, escludiamo i nodi tra s e $max[s]$.
- A questo punto, tutti i nodi non esclusi appartengono all'intersezione del ciclo principale con i cicli adiacenti.
  È possibile che esistano altri cicli non adiacenti a quello principale.
  Dobbiamo perciò verificare che togliendo uno qualsiasi dei candidati non rimangano altri cicli. In caso affermativo abbiamo trovato una soluzione, altrimenti non esiste soluzione.

```
int candidates_begin=0, candidates_end=main_cycle.size();
vector<bool> visited(N, false);
// salti all'indietro
for (int s_pos = main_cycle.size()-1; s_pos >= 0; s_pos--){
    int s = main_cycle[s_pos];
    int t = dfs_backward_loops(G, s, s_pos, visited)
    // un salto all'indietro arriva in t
    candidates_begin = max(candidates_begin, t);
    // un salto all'indietro parte da s
    if (min_reach[s] <= s_pos)
        candidates_end = s_pos + 1;
}
if (candidates_begin >= candidates_end) { return -1; }

// salti in avanti
for (int s_pos=0; s_pos < candidates_begin; s_pos++){
    int s = main_cycle[s_pos];
    // salto in avanti da s a max_reach[s];
    candidates_begin = max(candidates_begin, max_reach[s]);
    if (candidates_begin >= candidates_end) { return -1; }
}
return main_cycle[candidates_begin];
```

Complessità

- ⇒ soluzione: `sol-100.cpp`
- ⇒ $\approx$ 140 SLOC.
- ⇒ complessità:

$$O(N + M)$$

- ⇒ 150 punti.

Test Case con Cicli Disgiunti (I)

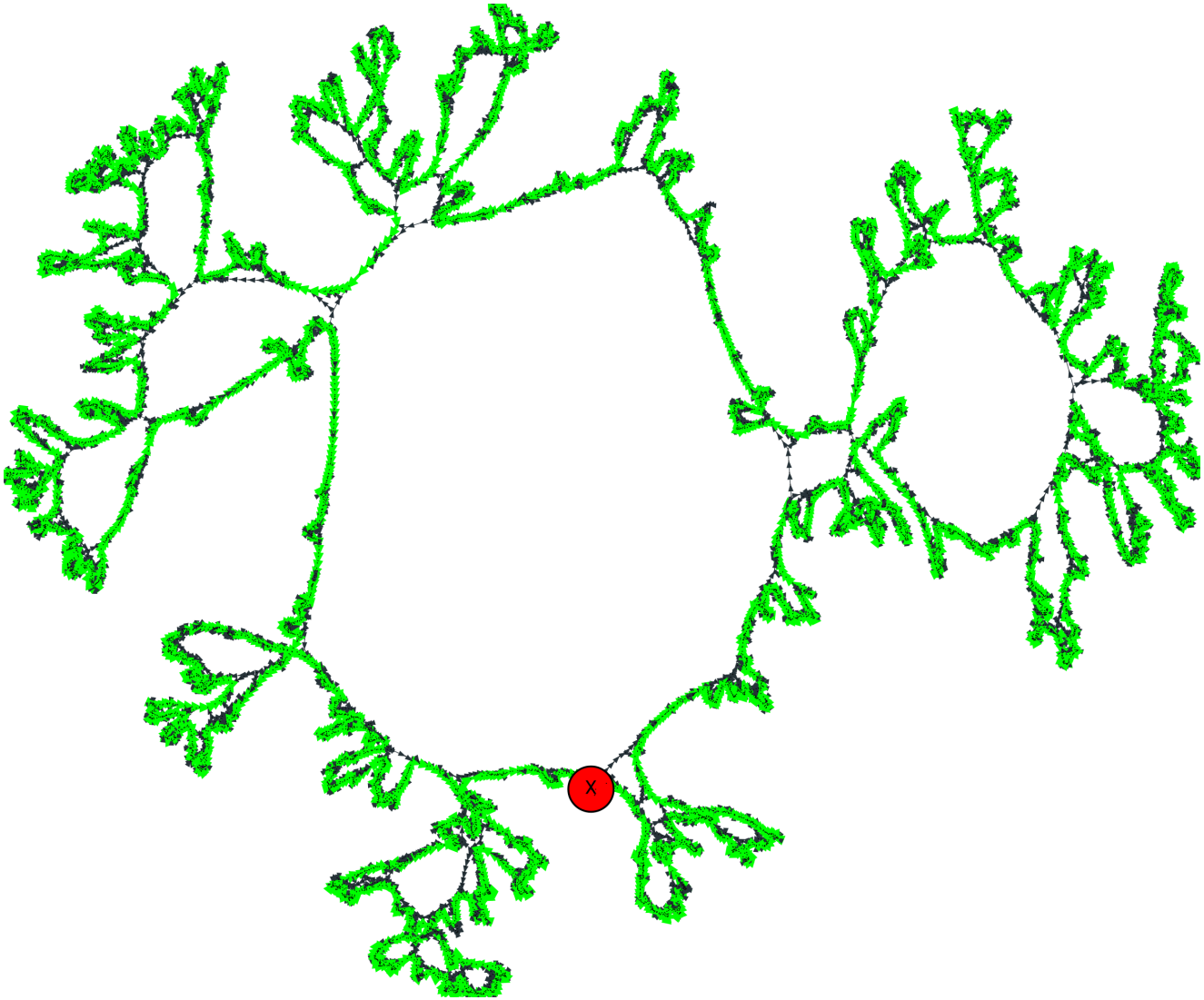


Figura: “Albero + ciclo di foglie”

Test Case con Cicli Disgiunti (II)

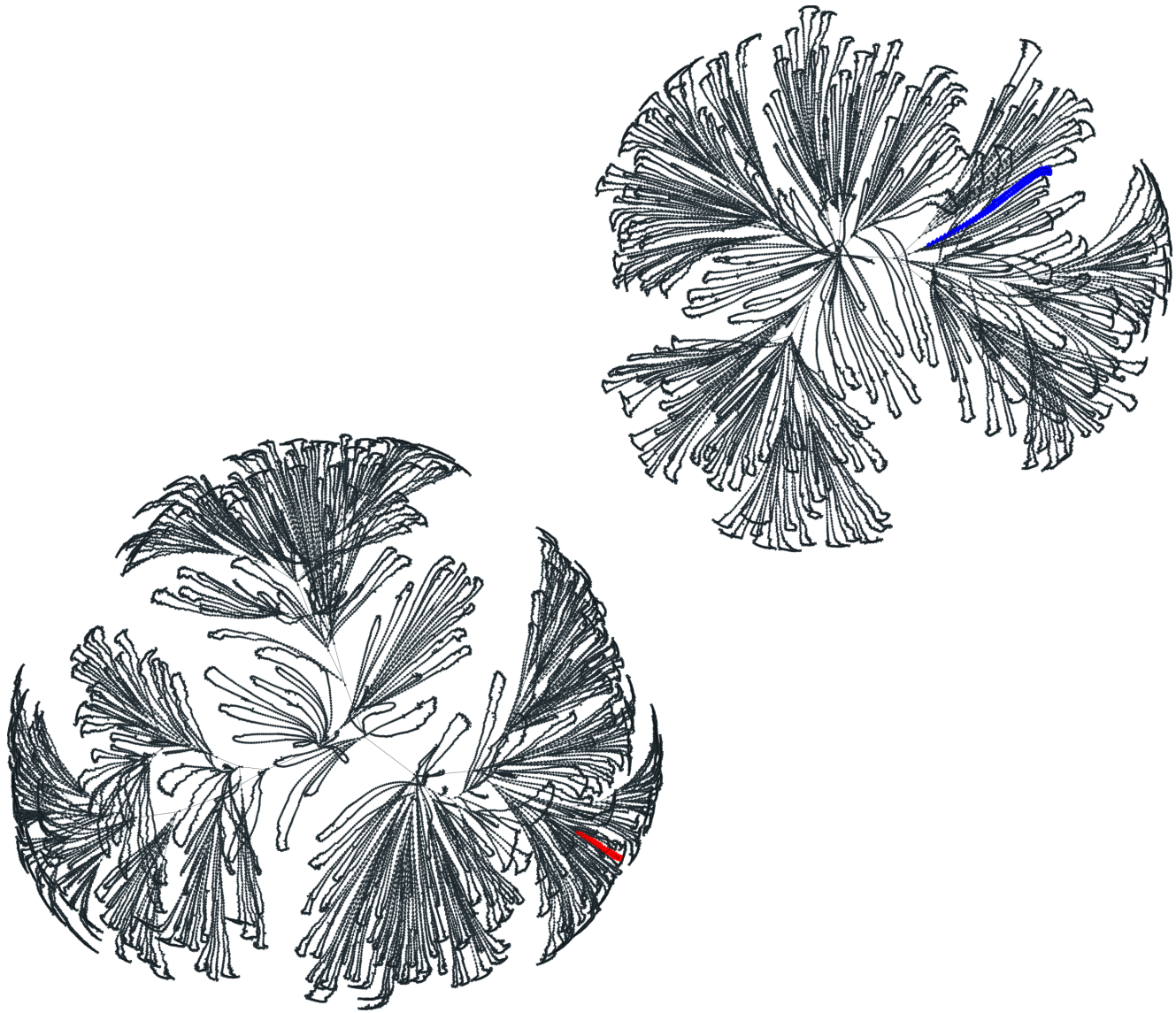


Figura: “Foresta di alberi con cicli alle foglie” (impossibile)

Test Case con Cicli Disgiunti (III)

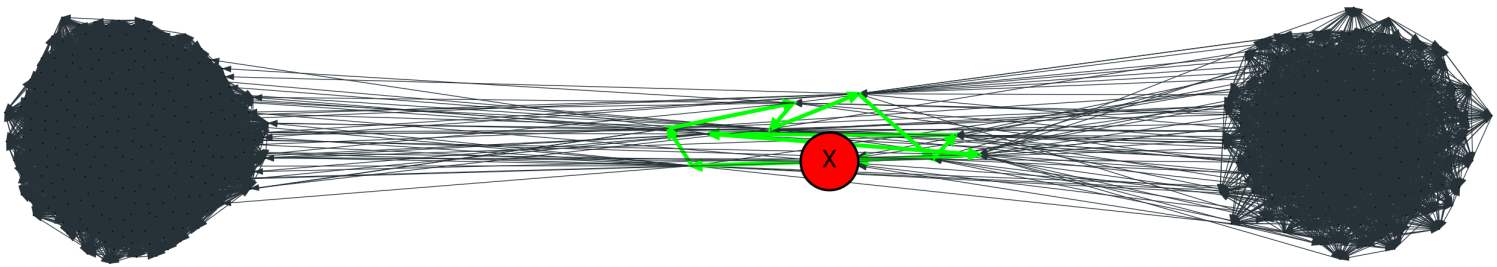
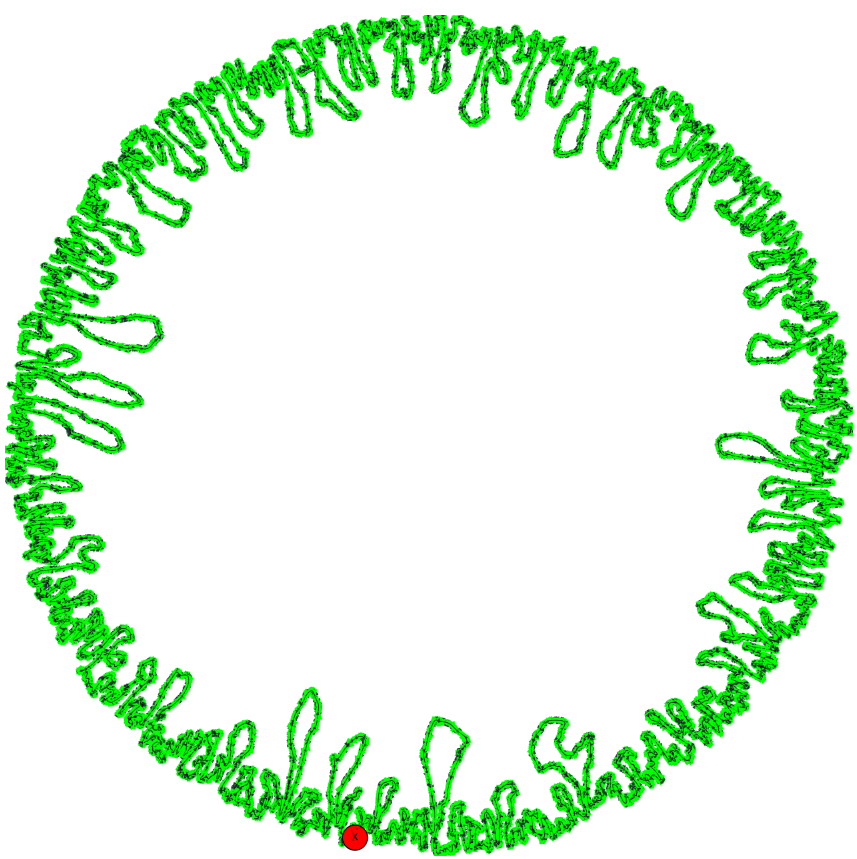
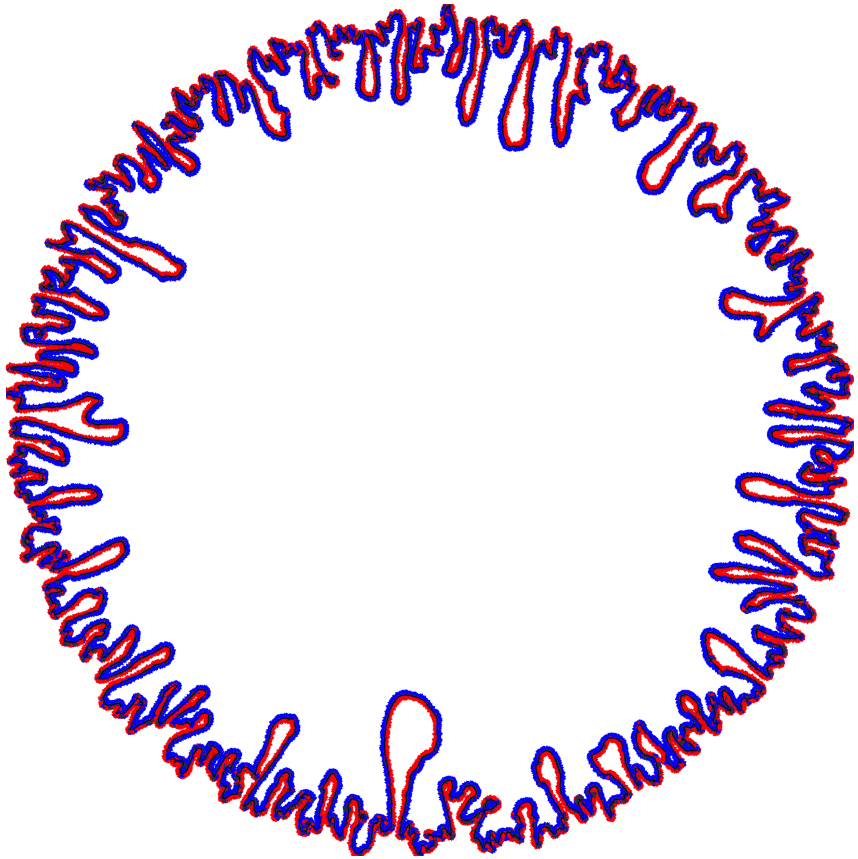


Figura: “Ciclo con DAG in ingresso e uscita”

Test Case Ciclo con Scorciatoie

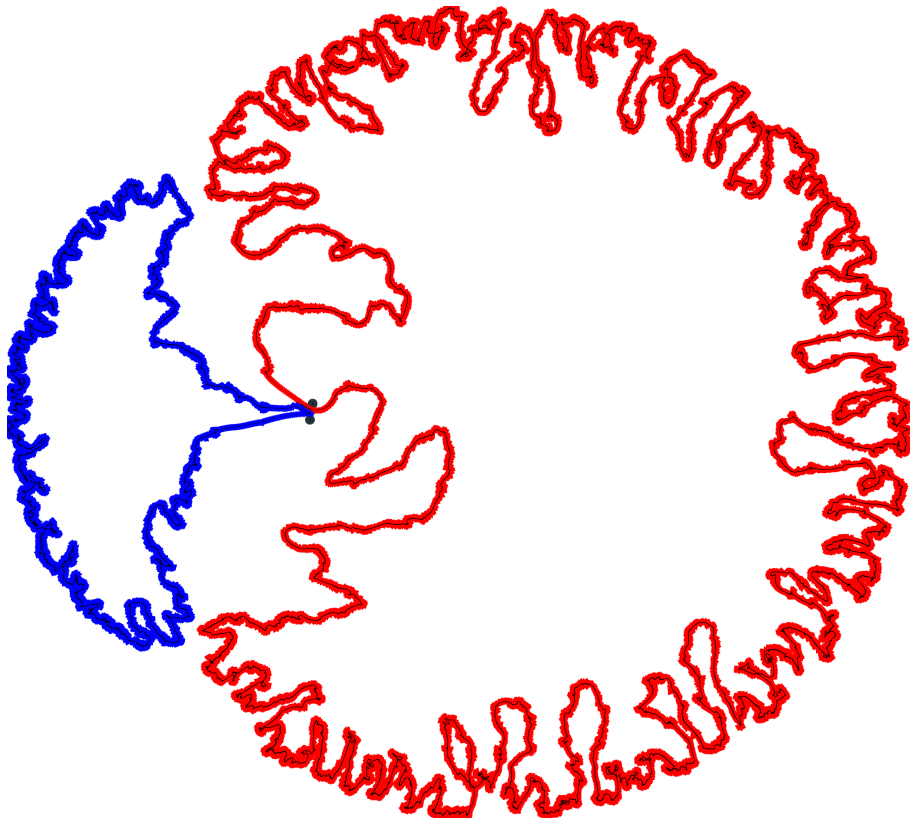


(A) Possibile

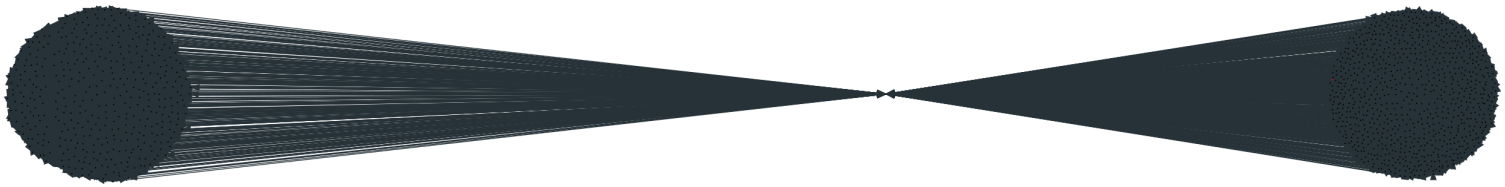


(B) Impossibile

Test Case Impossibili

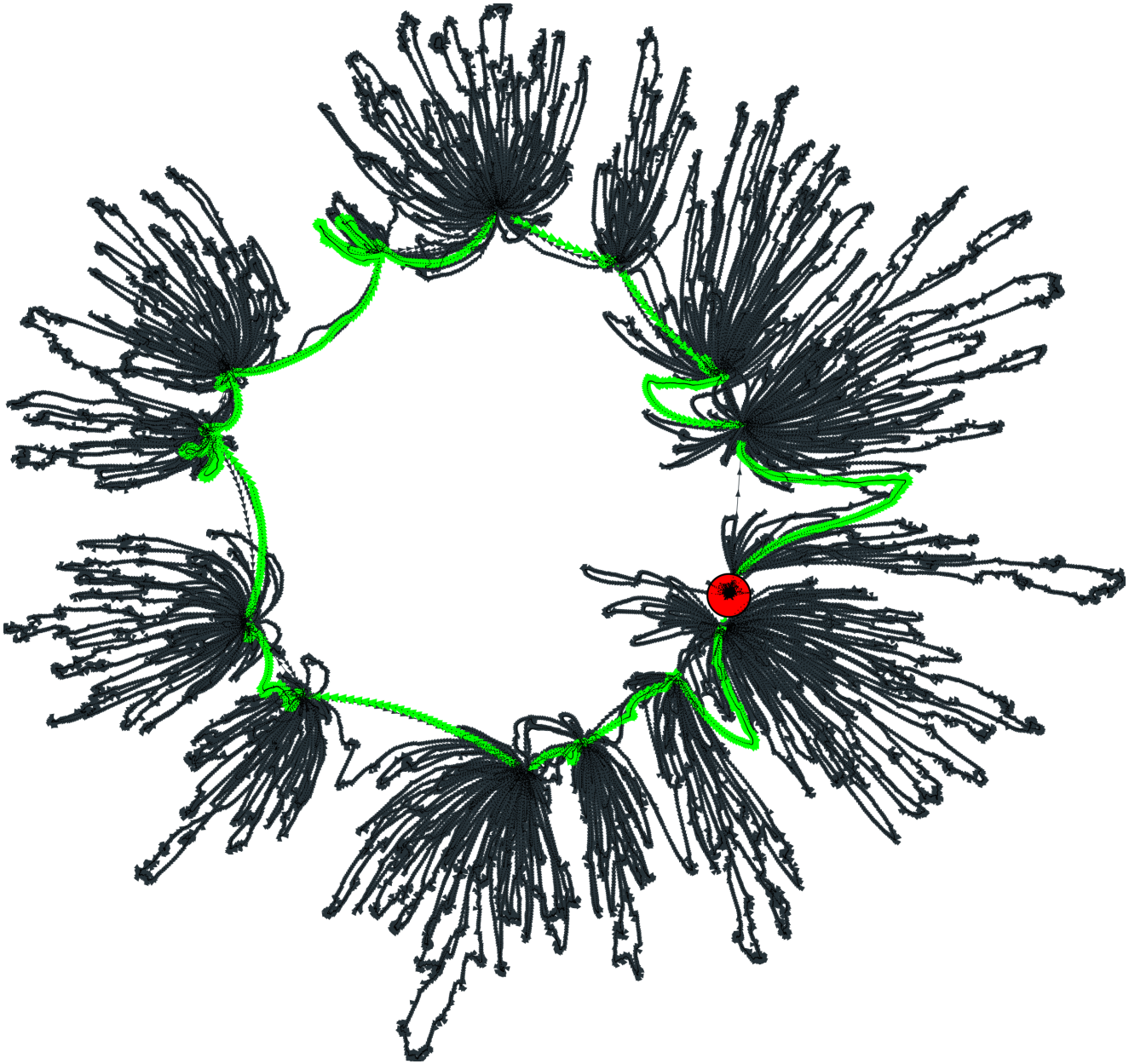


(A) Due cicli

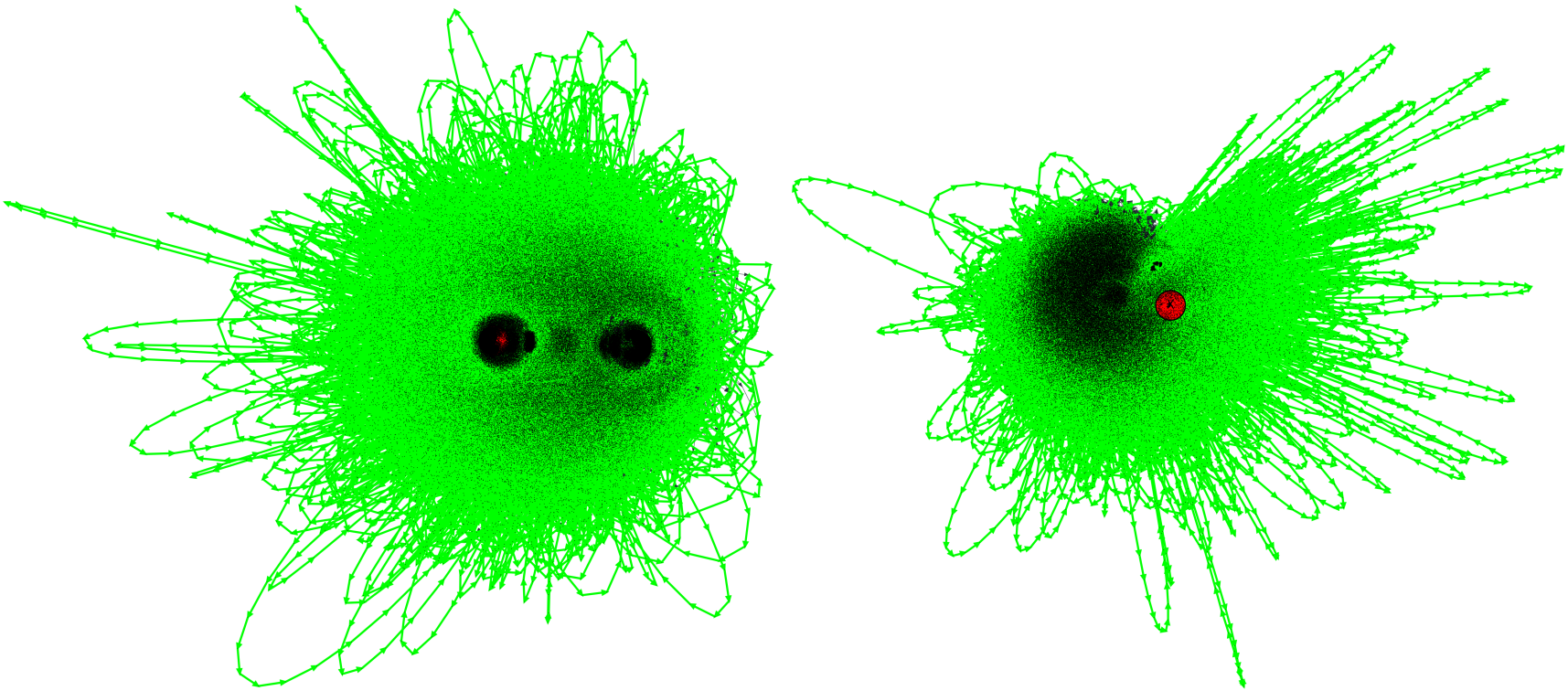


(B) Due cricche

Test Case ciclo con Percorsi Alternativi



Test Case Black Hole



Complimenti!

Il Professor A. è stato fermato! Ci vorrà un po' di tempo per riprendersi...