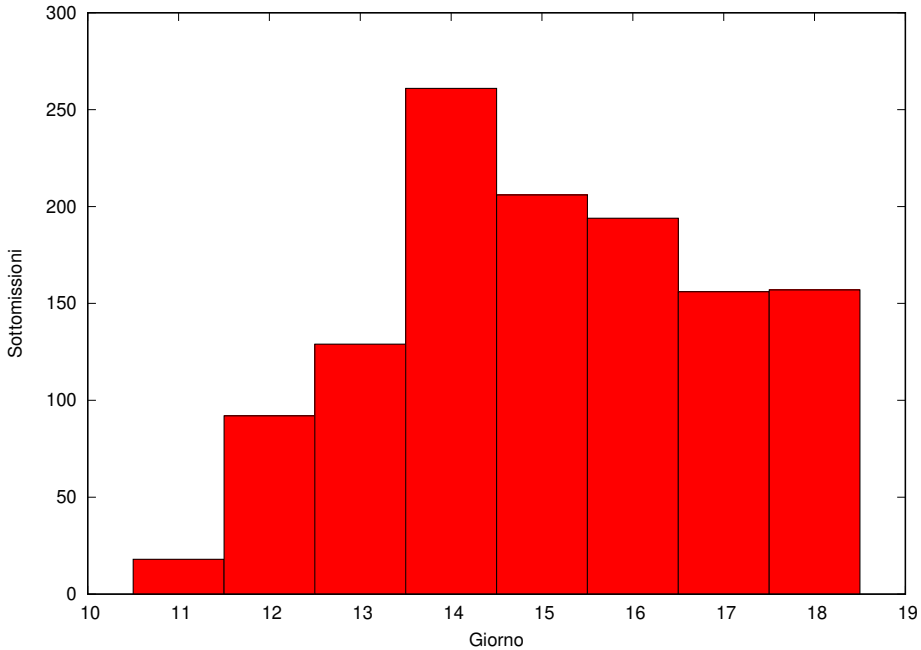
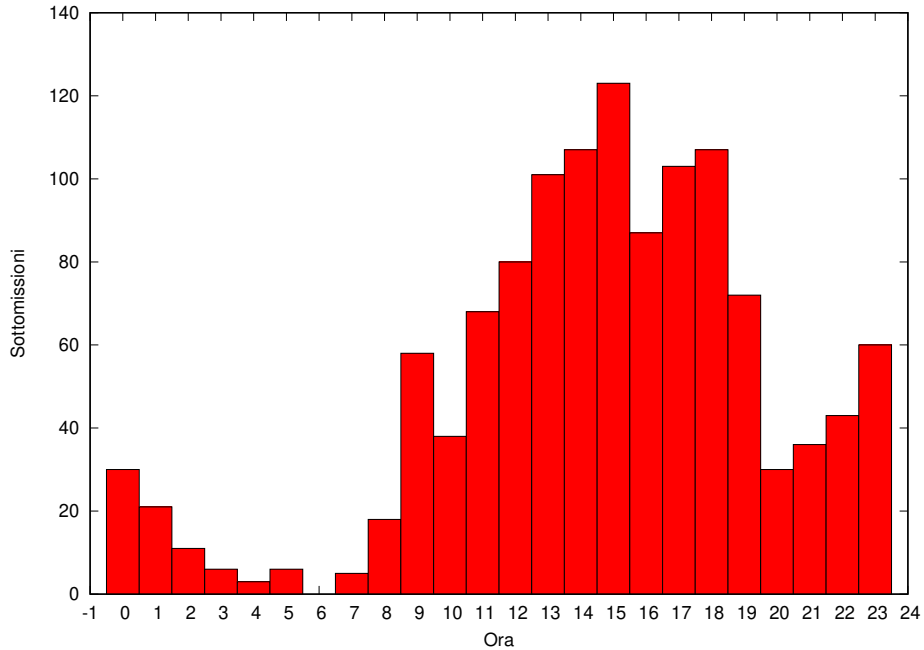
Primo Progetto ASD 2023/2024

Numero sottoposizioni: `1213`





- 74 gruppi hanno fatto almeno una sottoposizione, di cui 74 hanno raggiunto la sufficienza;
- 202 studenti iscritti, di cui 195 appartenenti a gruppi che hanno fatto almeno una sottoposizione (1 ritiro);

Risultati

Punteggi

- $P < 30$ $\longrightarrow$ progetto non passato.
- $30 \leq P < 50$ $\longrightarrow$ 1 punto bonus (9 gruppi).
- $50 \leq P < 80$ $\longrightarrow$ 2 punti bonus (37 gruppi).
- $80 \leq P \leq 100$ $\longrightarrow$ 3 punti bonus (29 gruppi).

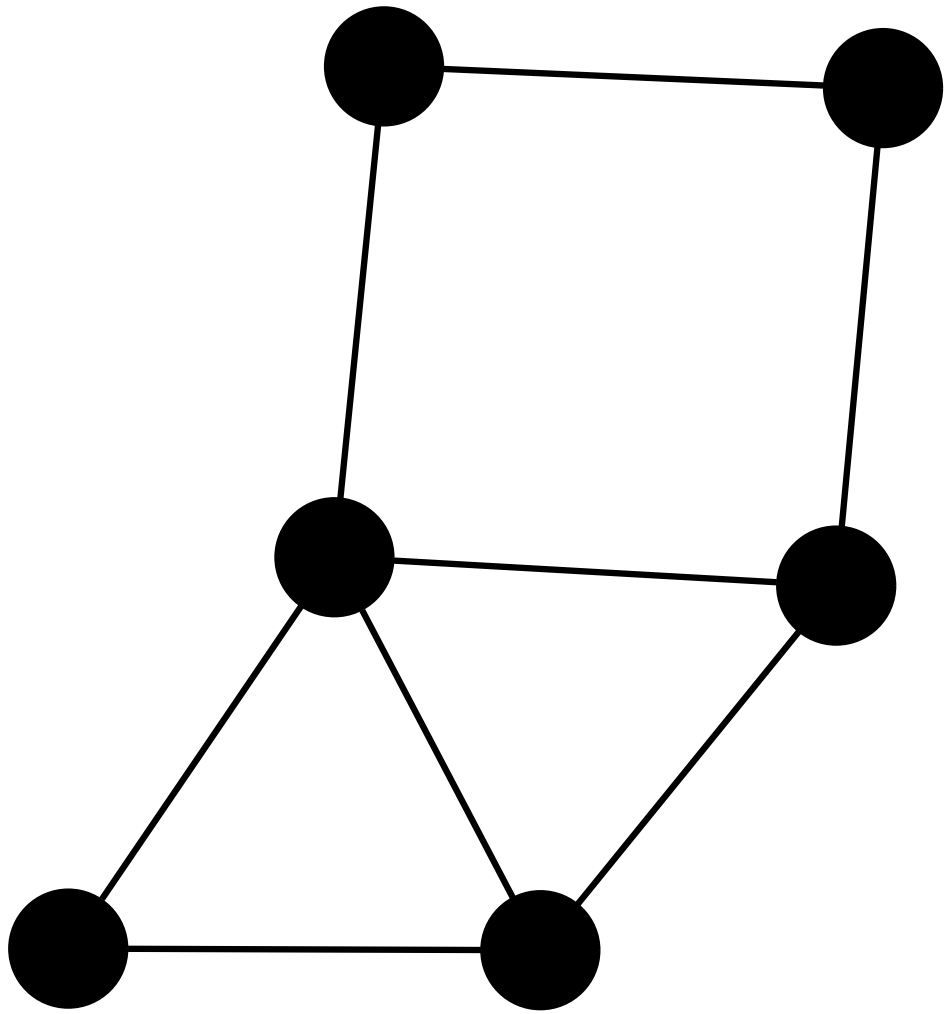
Classifiche e sorgenti sul sito (controllate i numeri di matricola):

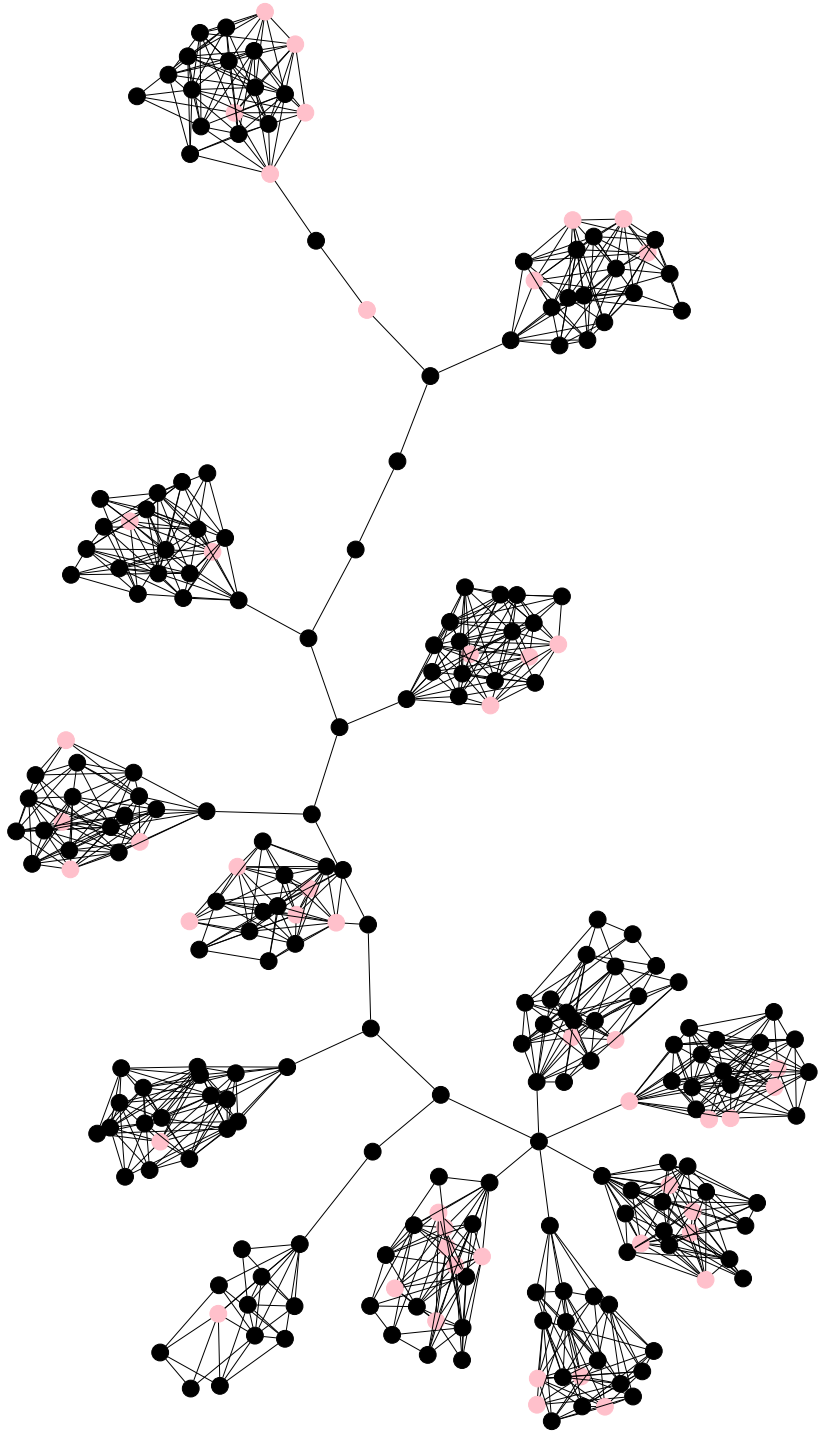
`https://judge.science.unitn.it/slides/asd23/classifica_prog1.pdf`

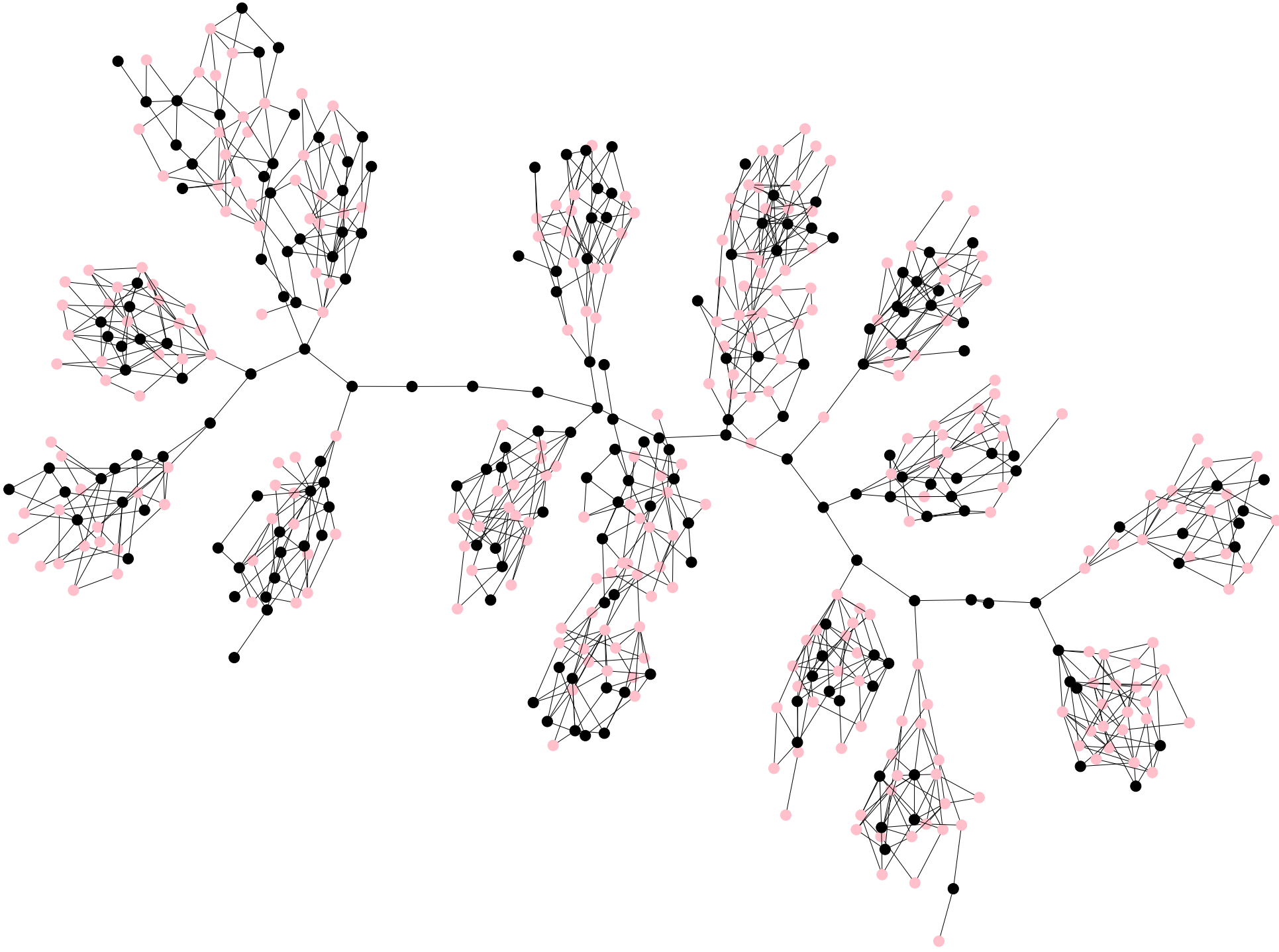
Problema

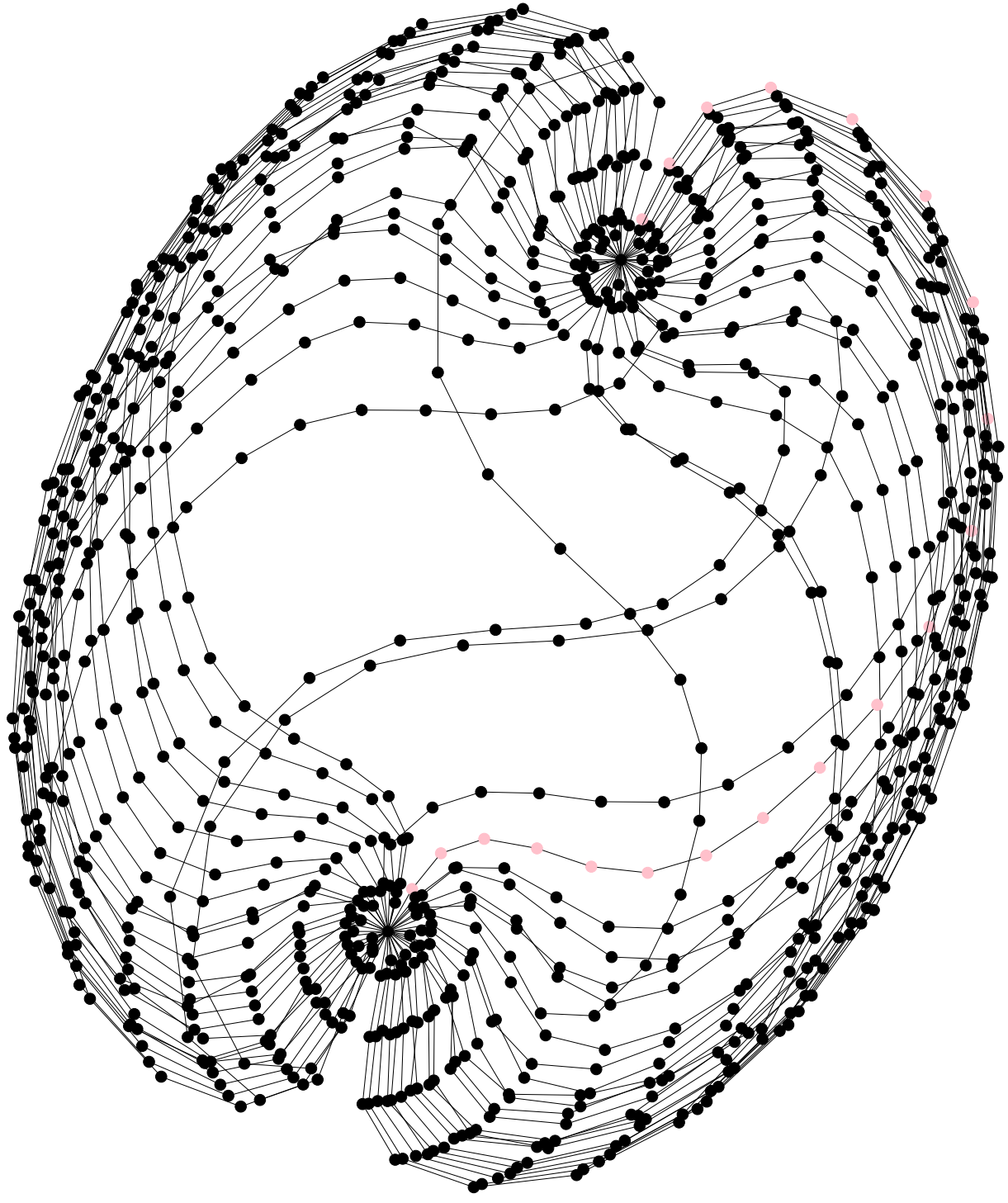
Dato un grafo non orientato pesato $G = (V, E)$.
Dato un sottoinsieme $MATTEL \subseteq V$.

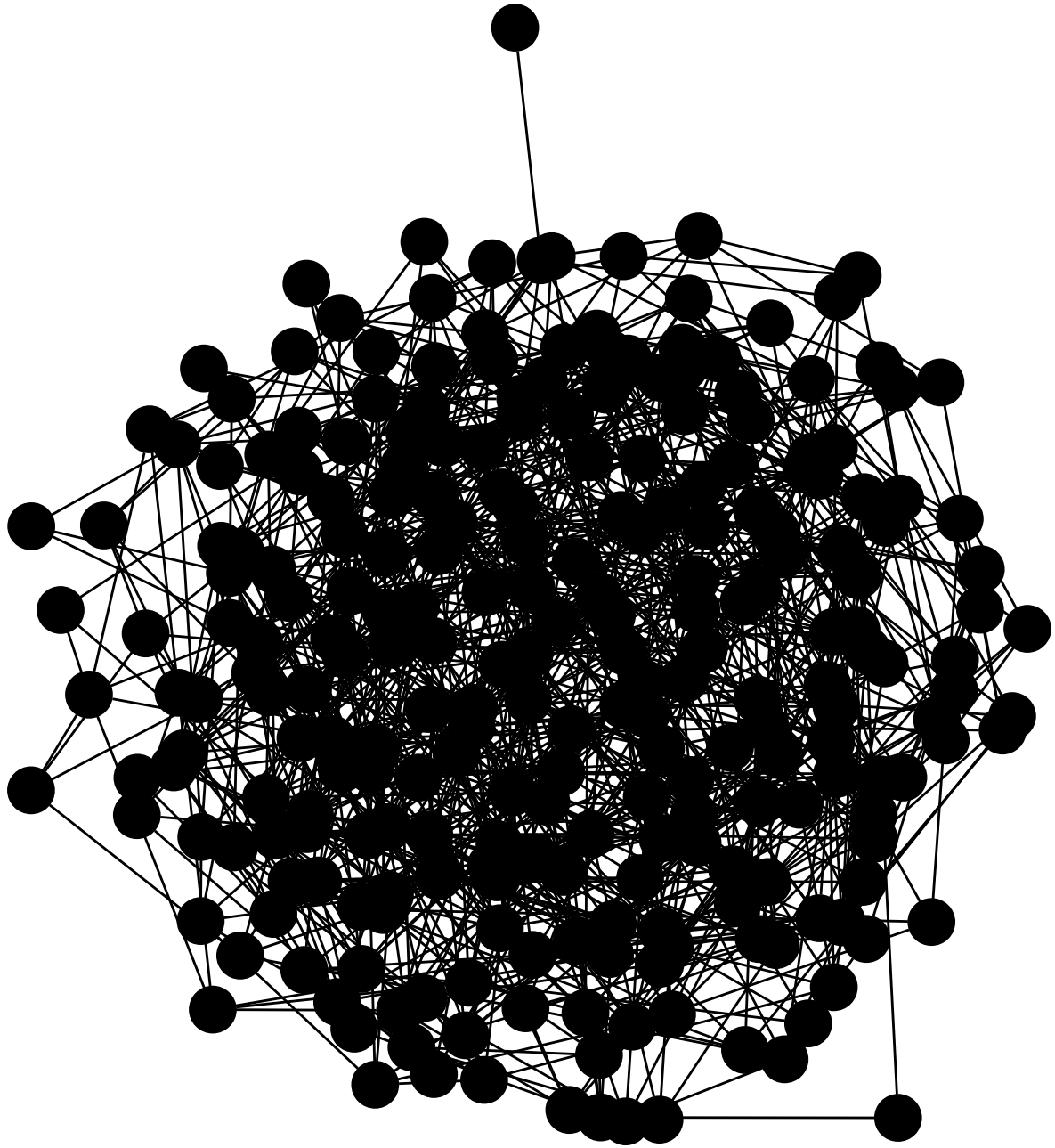
- Qual è il peso massimo K che possiamo aggiungere a tutti gli archi del grafo così da avere che i cammini di peso minimo $[B \dots A]$ passino solo attraverso vertici $V \setminus MATTEL$?
- Qual è un cammino di peso minimo da $[B \dots A]$?

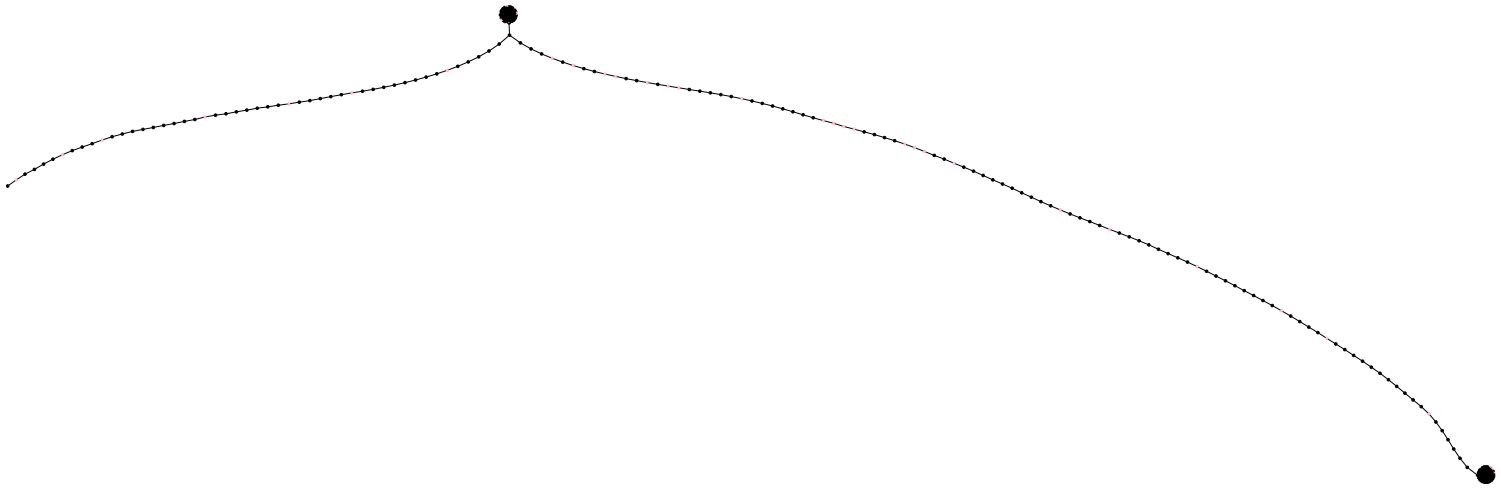


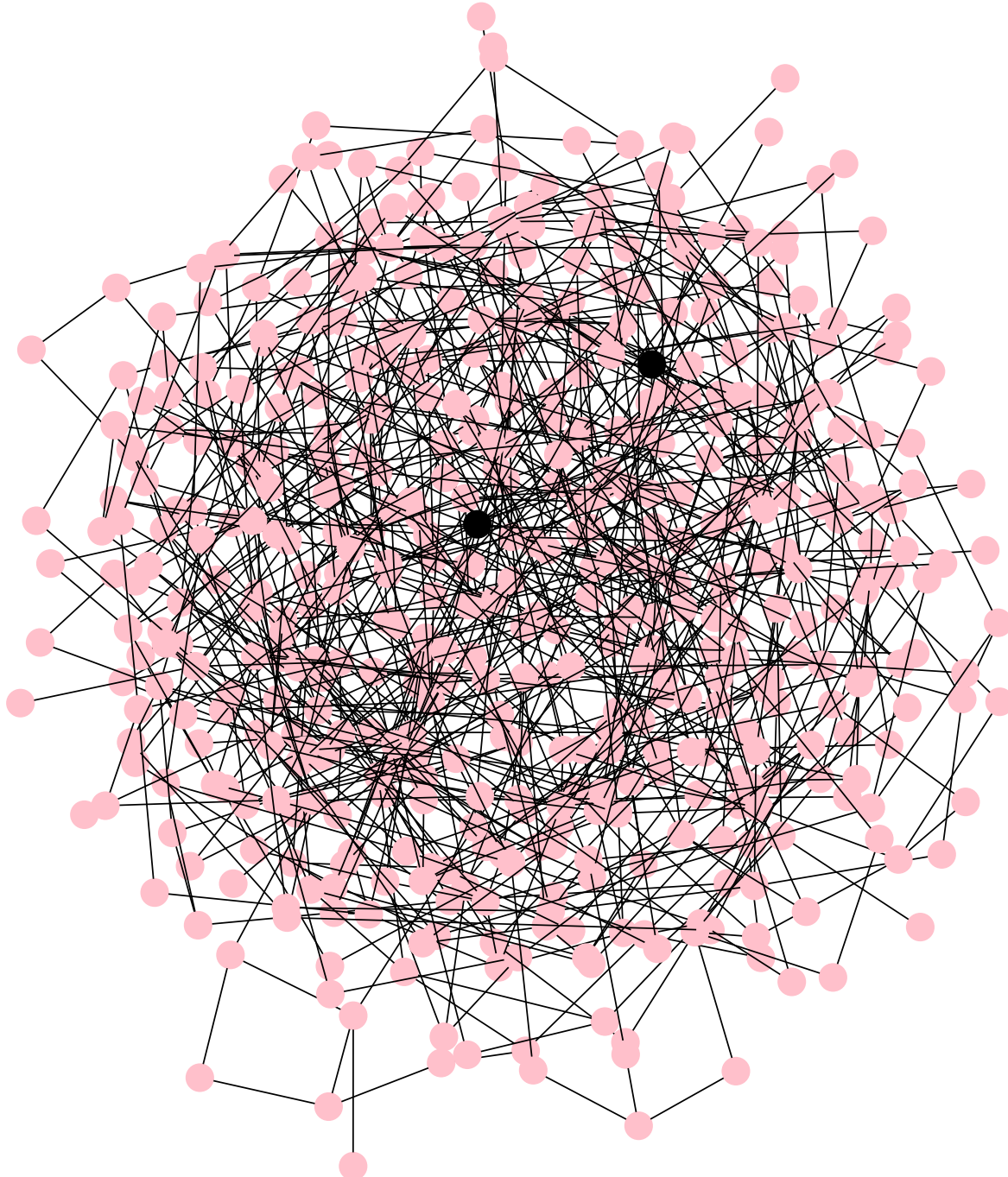


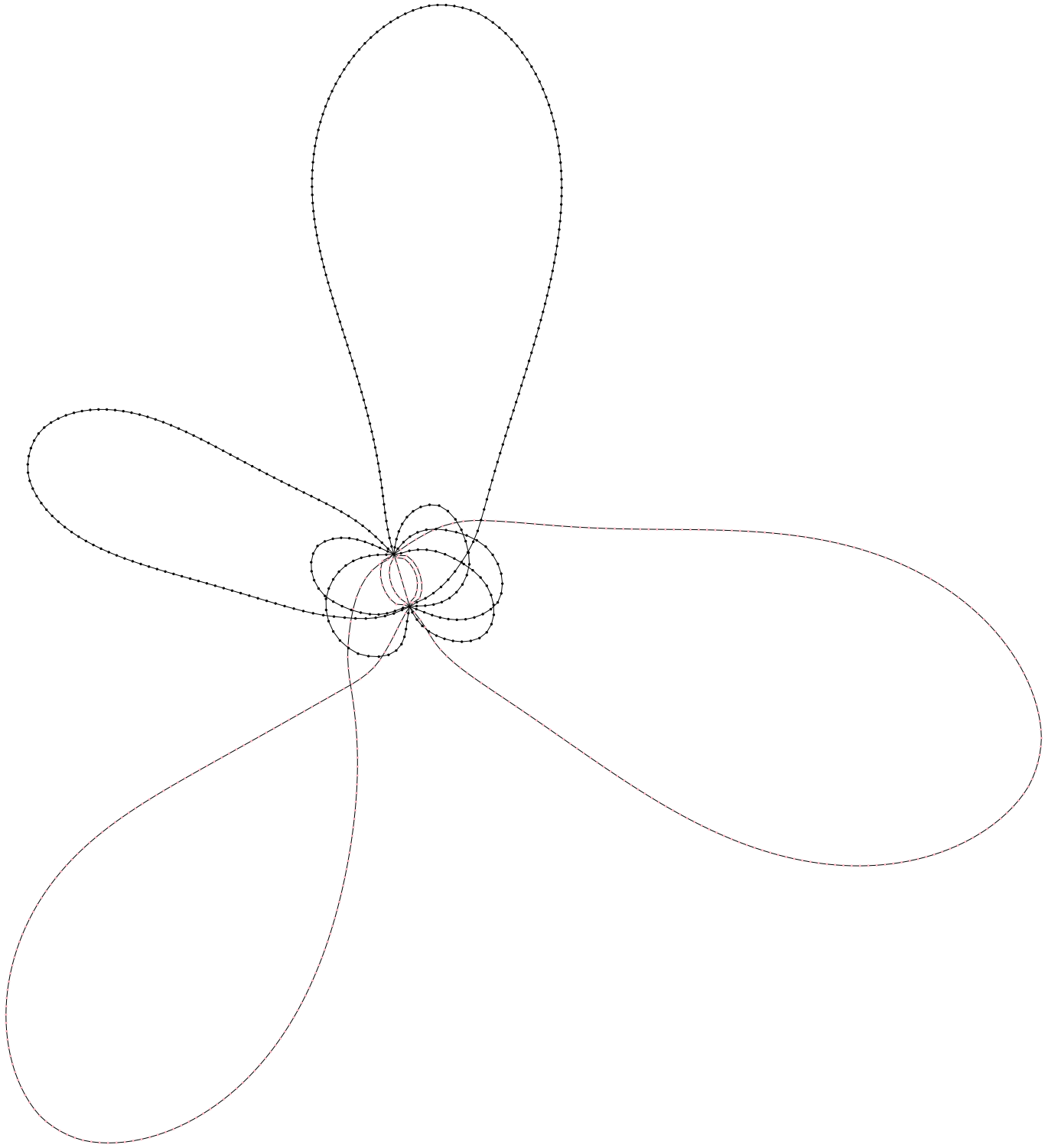


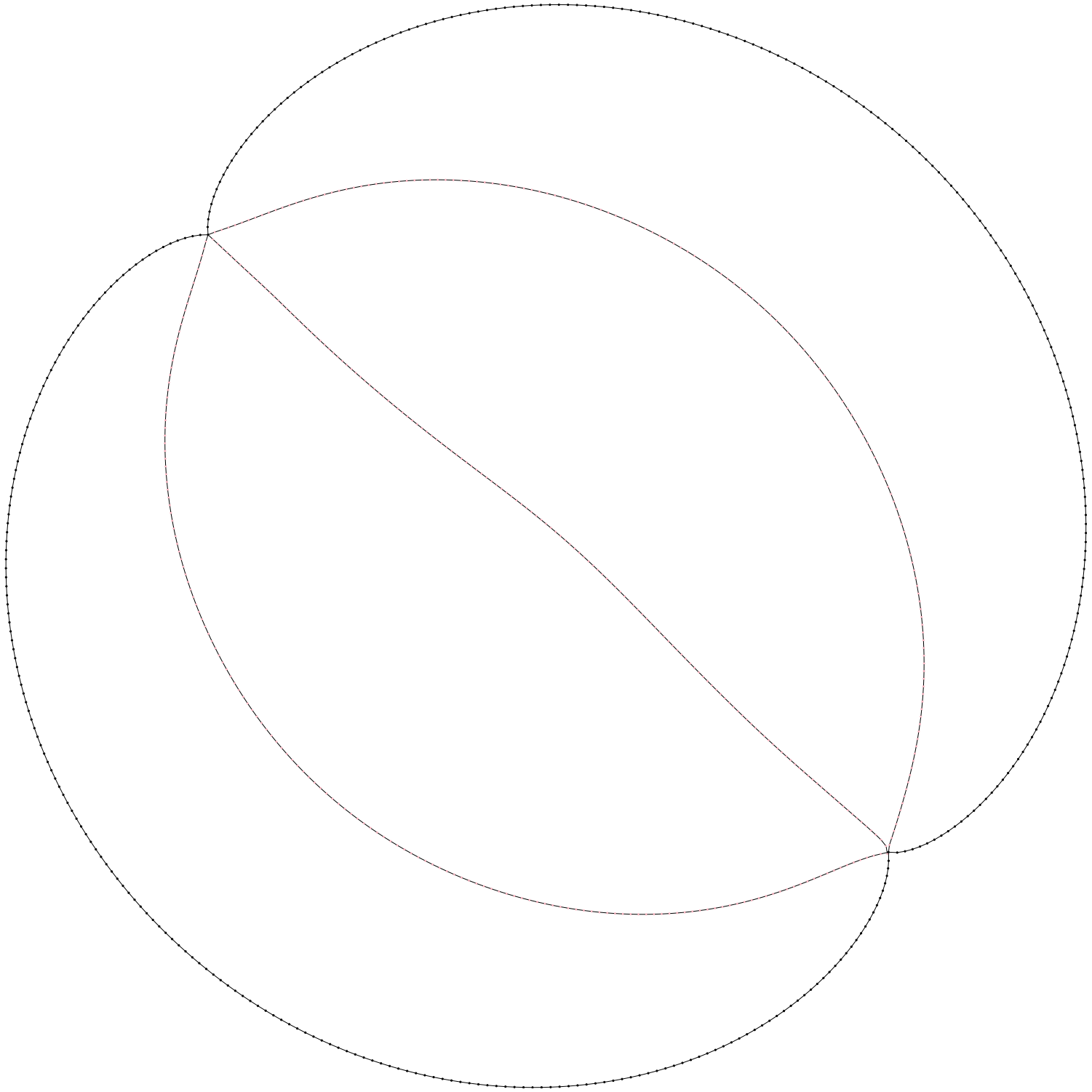


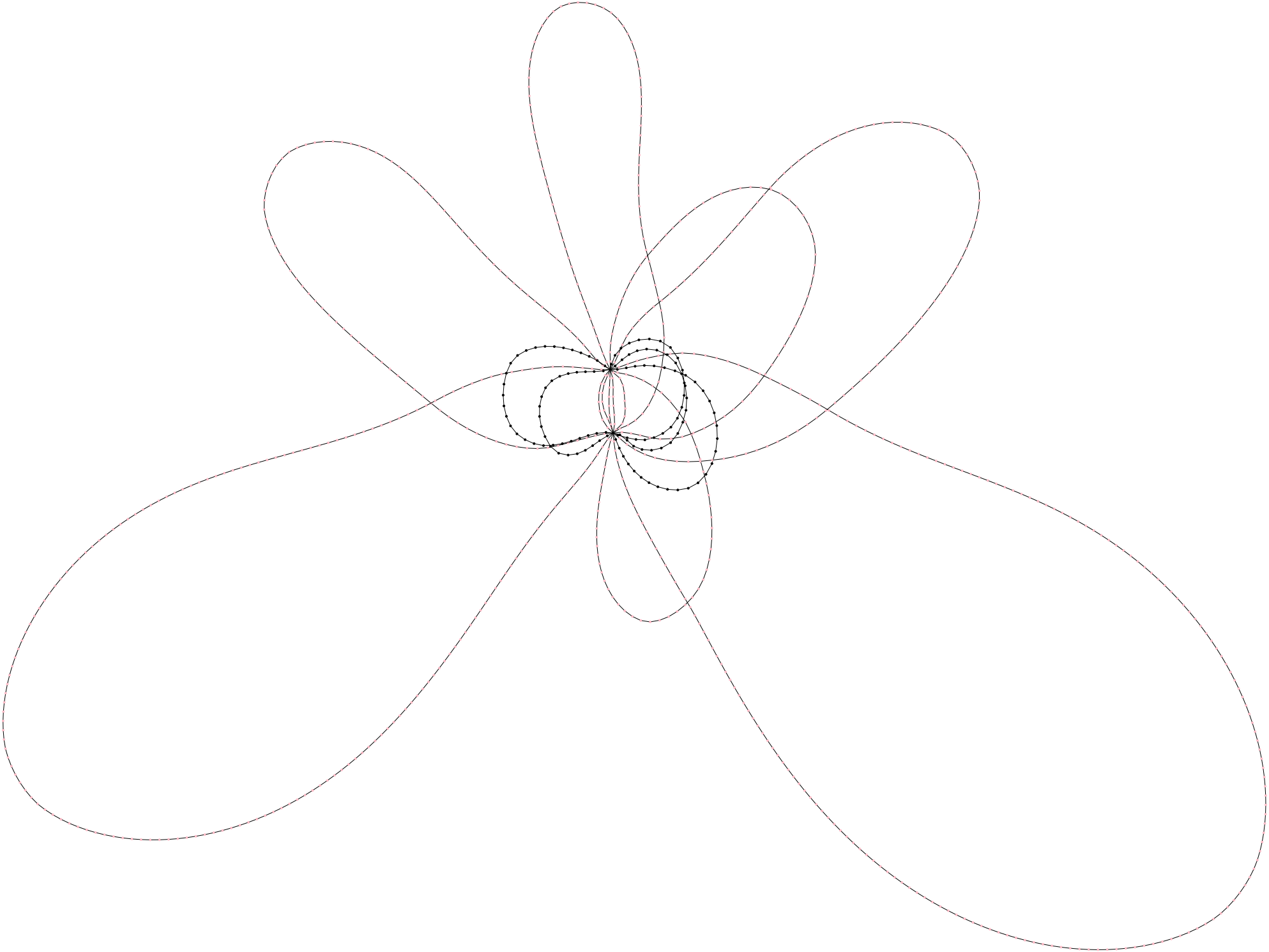


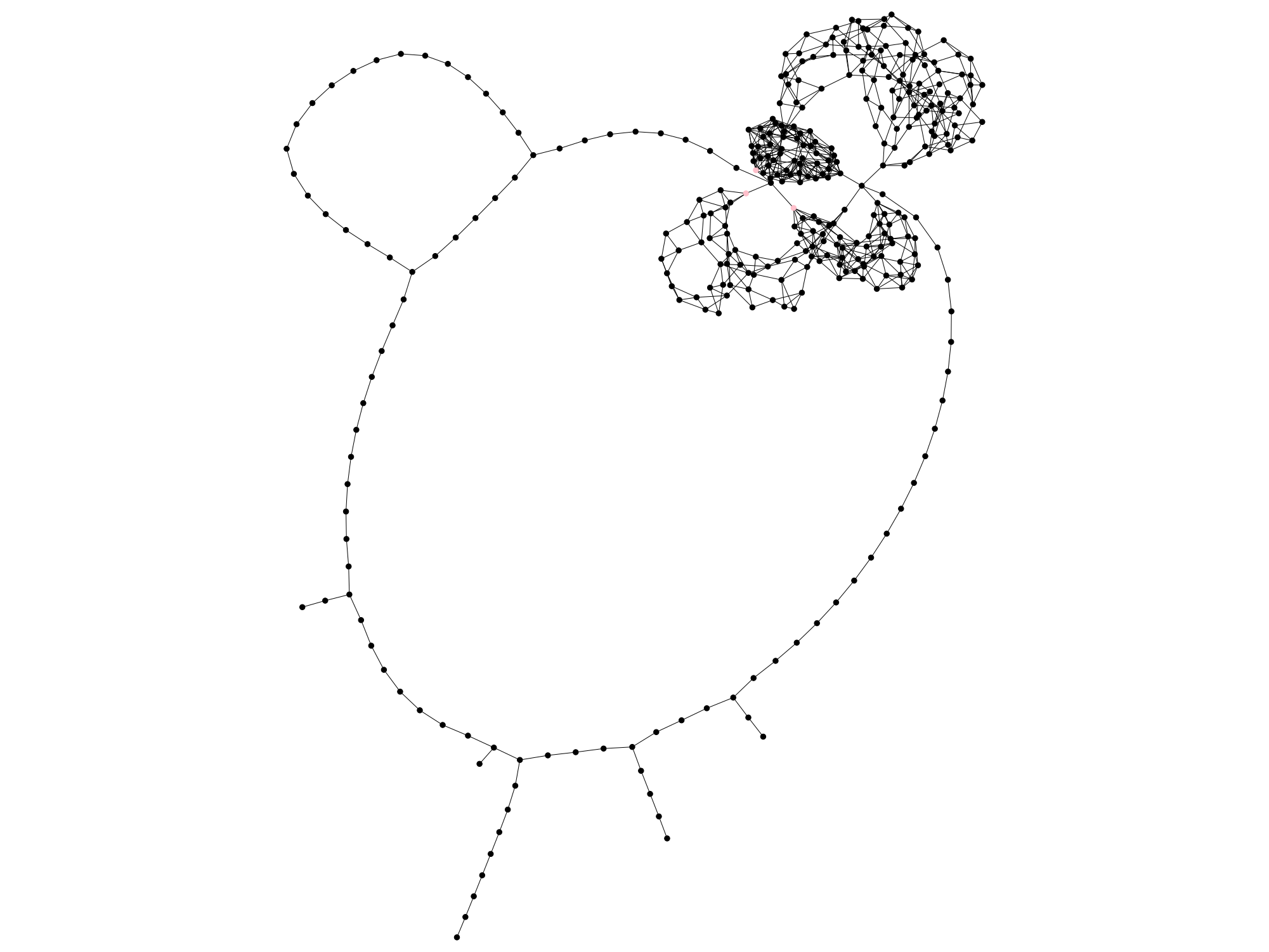


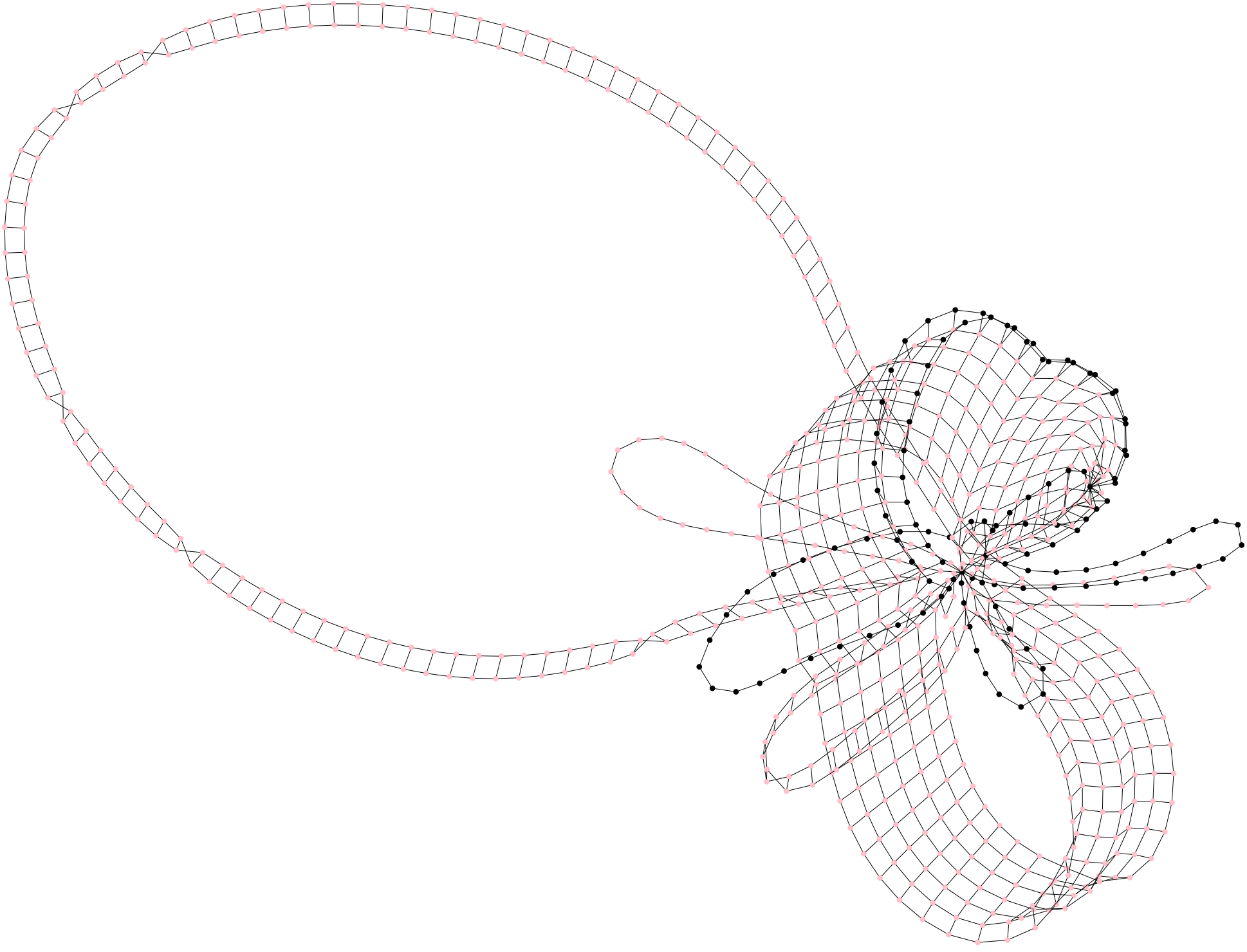


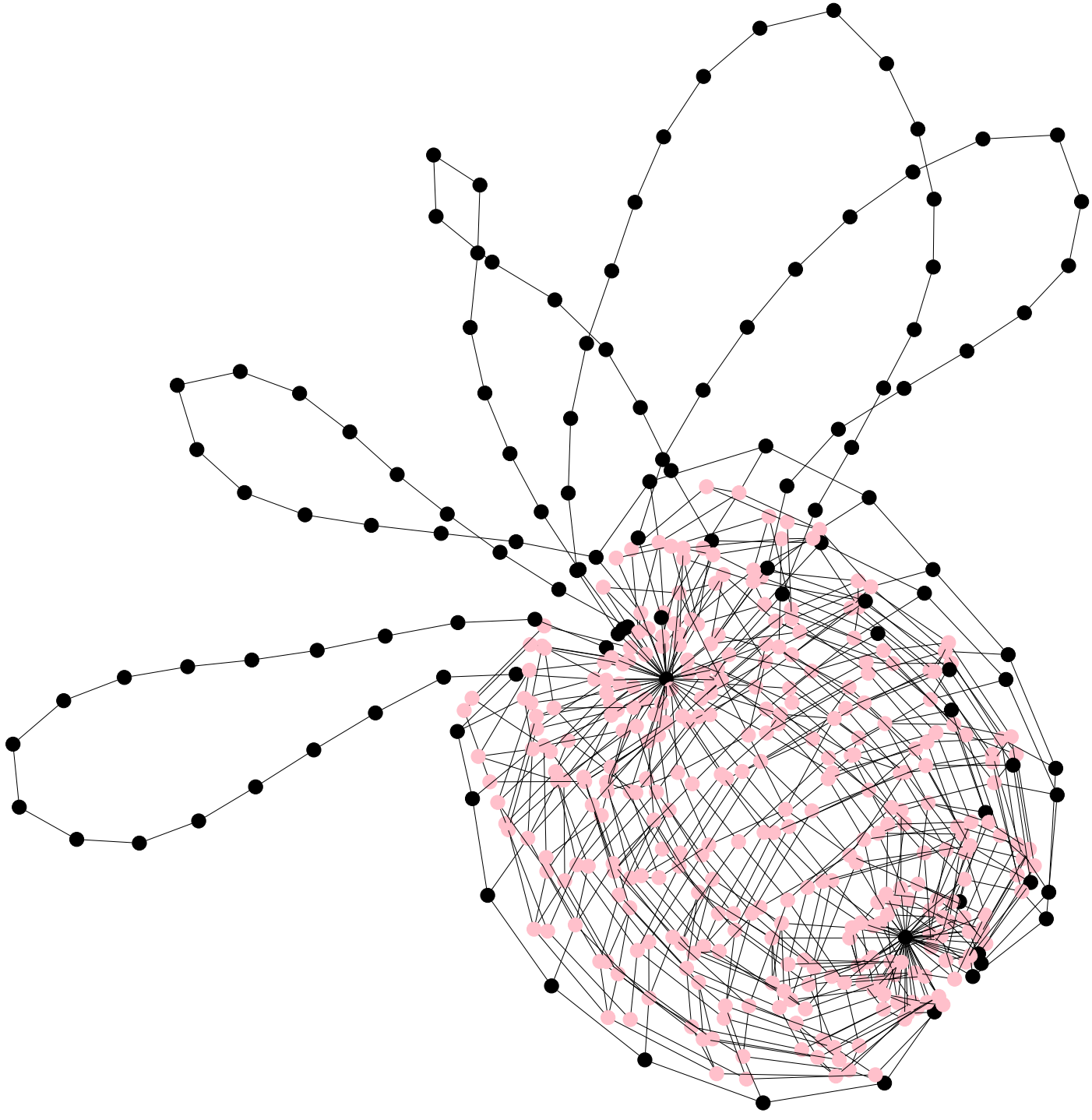


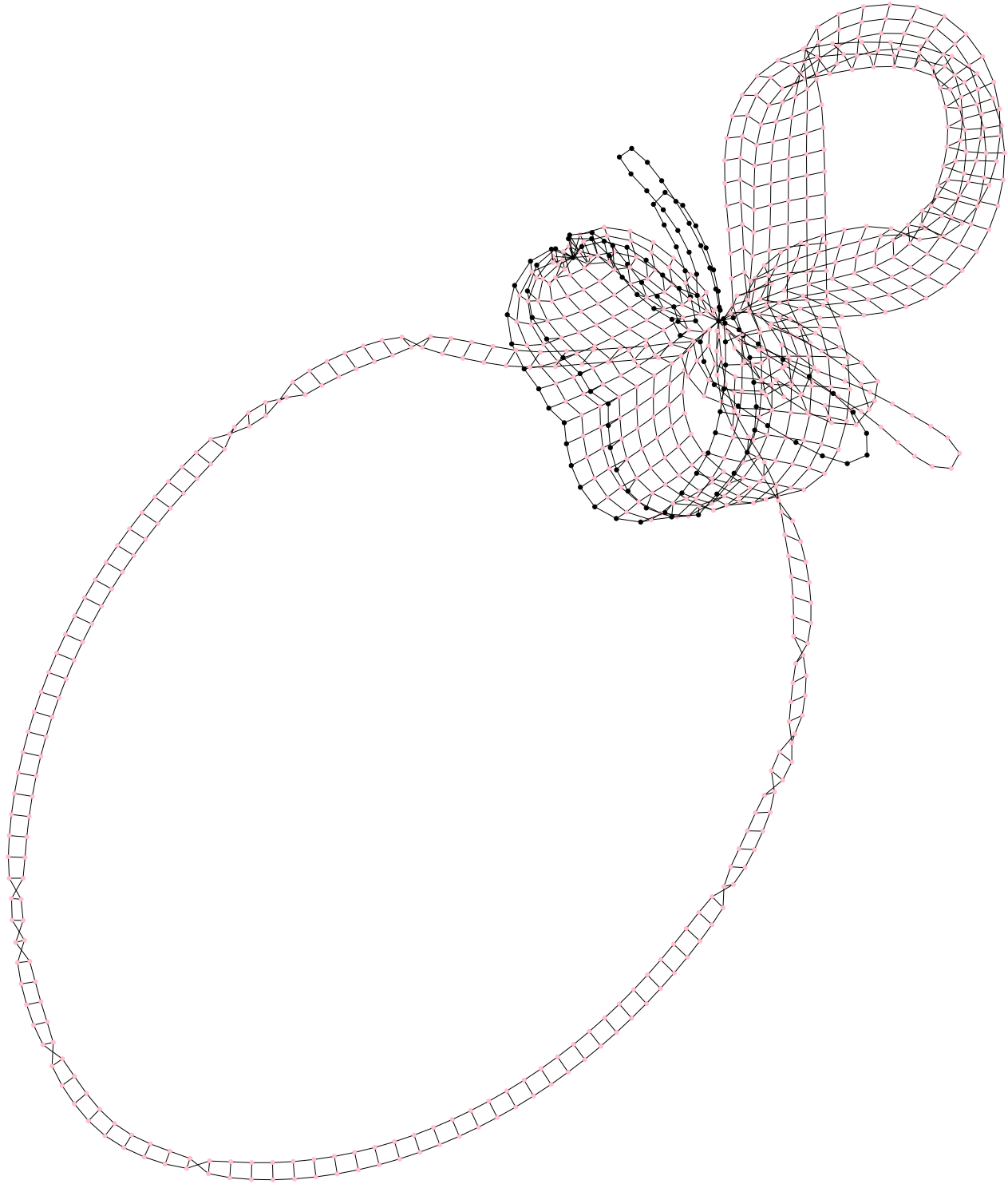


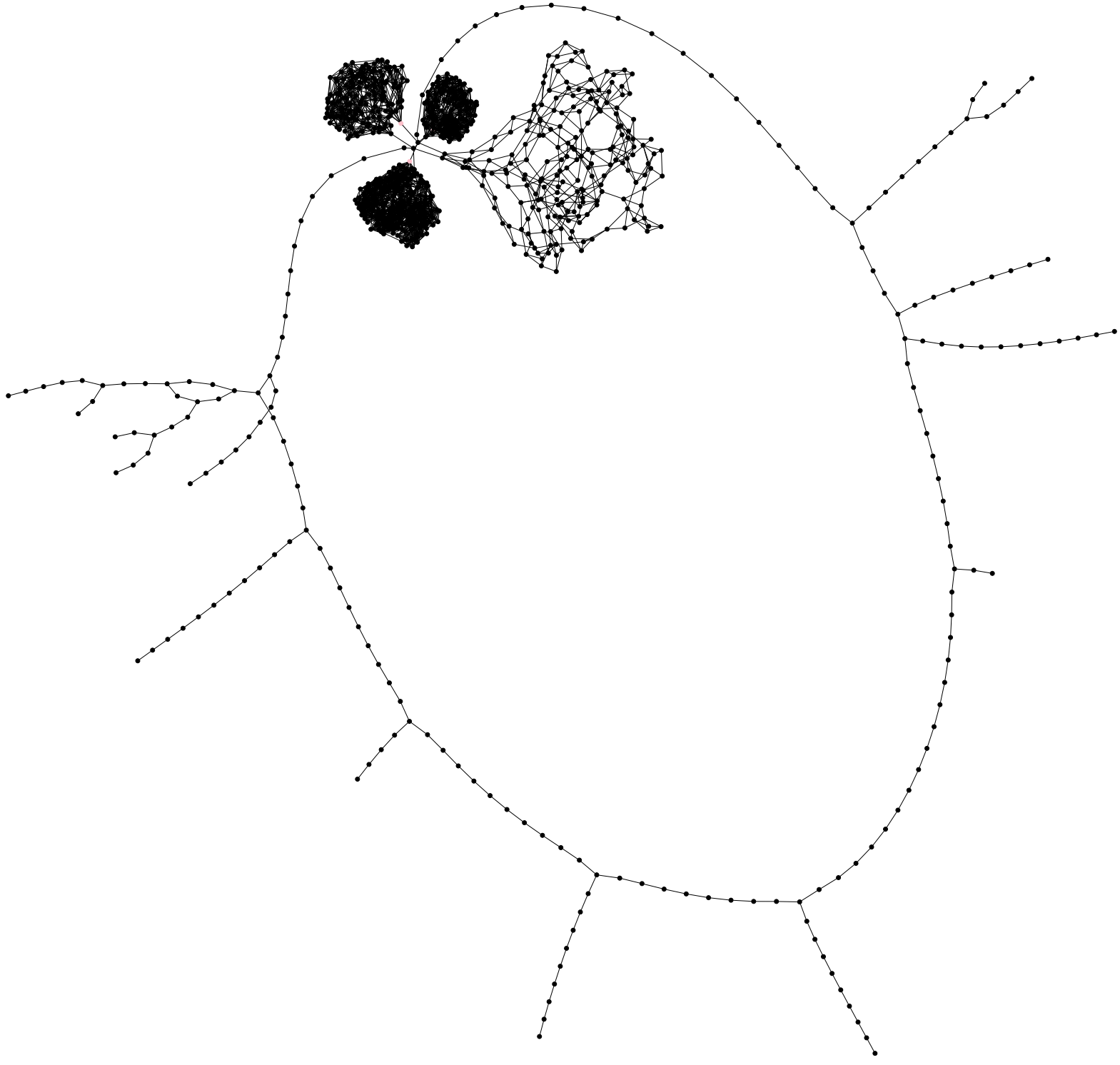












Sottoproblema: stesso tempo di percorrenza

Osservazioni

- Tutte le strade hanno lo stesso tempo di percorrenza
  Tutti gli archi hanno lo stesso peso
- I cammini di peso minimo = cammini che attraversano meno archi
- Se $\exists$ cammino di peso minimo $[B \dots A]$ passante per nodi *MATTEL*, la risposta è IMPOSSIBILE, ILLIMITATO altrimenti

Sottoproblema: stesso tempo di percorrenza

```
vector<node> parent(N, -1); parent[B] = B;
queue<pair<int, node>> with, wout; wout.emplace(0, B);

while (!with.empty() || !wout.empty()) {
  const bool is_with =
    (with.empty() ? INT_MAX : with.front().first) <=
    (wout.empty() ? INT_MAX : wout.front().first);
  queue<pair<int, node>> &q = is_with ? with : wout;
  const int lv = q.front().first;
  const node x = q.front().second;
  q.pop();
  for (const node adj: neighbours[x]) {
    if (parent[adj] == -1) {
      parent[adj] = x;
      (is_with || is_mattel[adj] ? with : wout)
        .emplace(lv + 1, adj);
    }
  }
}
```

Sottoproblema: stesso tempo di percorrenza

```
const int K = any_of(
    path,
    [&is_mattel](const node x) { return is_mattel[x]; }
  )
  ? IMPOSSIBLE
  : LIMITLESS;

list<node> path;
path.push_front(A);
while (path.front() != B) {
  path.push_front(parent[path.front()]);
}
```

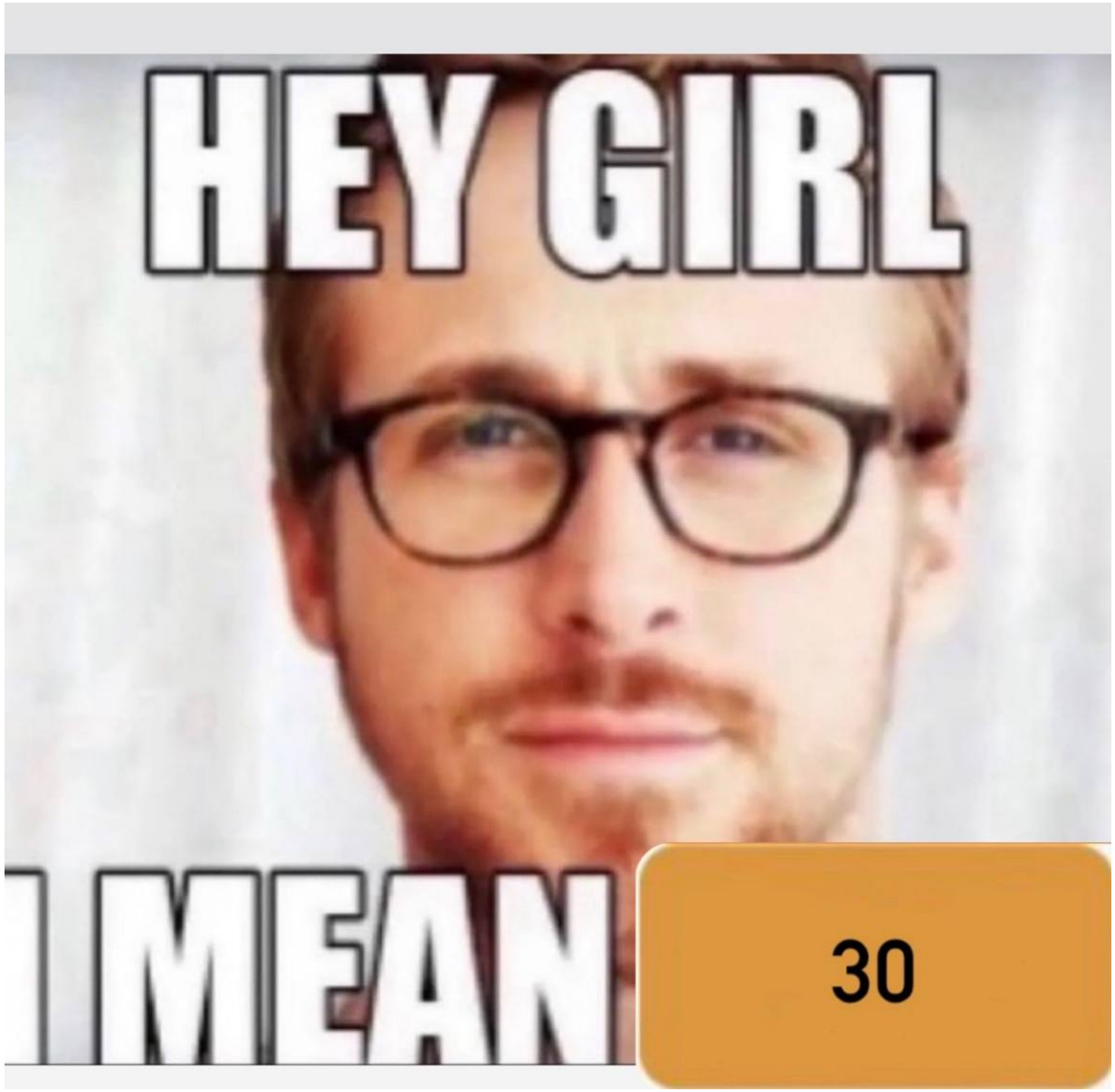
Complessità

- ⇒ soluzione: `sol-30-bfs-with-priority.cpp`
- ⇒ complessità:

$$\mathcal{O}(M + N)$$

- ⇒ 30 punti

It ain't much but it's honest work



Sottoproblema: tutte o nessuna

Osservazioni

- Se $MATTEL = V$, IMPOSSIBILE
- Se $MATTEL = \varnothing$, ILLIMITATO
- Non è più sufficiente una BFS per trovare i percorsi minimi

Sottoproblema: tutte o nessuna

Algoritmo di Dijkstra

L'algoritmo di Dijkstra permette di calcolare le **distanza minime** dei nodi **in un grafo pesato** rispetto a un nodo sorgente.

Invece di una semplice coda, usiamo una **coda con priorità**, che ci consente di **estrarre il nodo** non ancora visitato con **distanza minima**.

SOTTOPROBLEMA: TUTTE O NESSUNA

```
void dijkstra(const vector<pair<int, int>> adj[],
              const int n, const int x, const int y,
              stack<int> & min_path){


    vector<bool> vis(n, false);
    priority_queue<pair<long long, int>> pq;
    vector<long long> dist(n, LONG_LONG_MAX);
    vector<int> prec(n, -1);

    pq.push({0, x});
    dist[x]=0;
```

SOTTOPROBLEMA: TUTTE O NESSUNA

```
while(!pq.empty()){
    auto act=pq.top().second;
    pq.pop();
    if(act==y){
        visited[act]=true;
        break;
    }else if(!visited[act]){
        visited[act]=true;
        for(int i=0; i<adj[act].size(); i++){
            if( dist[adj[act][i].first] >
                    dist[act]+adj[act][i].second){
                dist[adj[act][i].first] =
                    dist[act]+adj[act][i].second;

                prec[adj[act][i].first]=act;
                pq.push({-dist[adj[act][i].first],
                    adj[act][i].first});
}}}}
```

SOTTOPROBLEMA: TUTTE O NESSUNA

```
    int act=y;
    while(act!=-1){
        min_path.push(act);
        act=prec[act];
    }
}
```

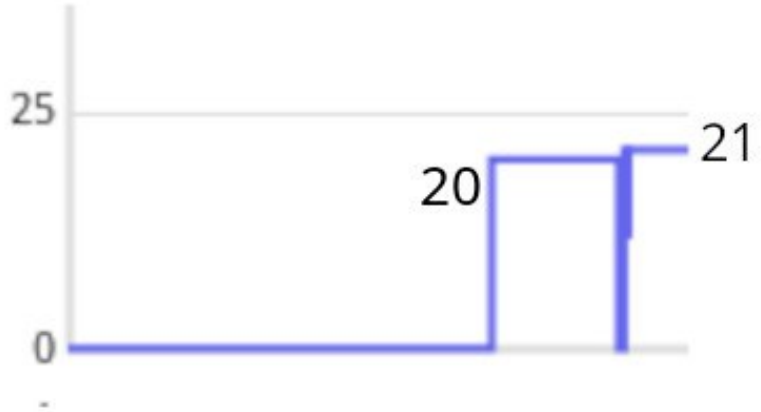
Complessità

- ⇒ soluzione: `sol-50-dijkstra.cpp`
- ⇒ complessità:

$$\mathcal{O}(N + M \log N)$$

- ⇒ 50 punti

Un piccolo passo per un gruppo di informatici
Un grande passo per una barbie



OSSERVAZIONI

- Sia Π l'insieme di cammini $[B \dots A]$
  Sia $\Pi_{MATTEL} \subseteq \Pi$ l'insieme di cammini $[B \dots A]$ tali che:
  $\pi \in \Pi_{MATTEL}$ sse π visita un nodo *MATTEL*
  Il problema diventa:

$$\max K : \exists s \in \Pi \setminus \Pi_{MATTEL}.$$
$$\forall r \in \Pi_{MATTEL}.$$
$$w_s + n_s \cdot K < w_r + n_r \cdot K$$

Cammini disgiunti

$$\max K: \quad \exists s \in \Pi \setminus \Pi_{MATTEL}. \quad \forall r \in \Pi_{MATTEL}. \quad w_s + n_s \cdot K < w_r + n_r \cdot K$$

Osservazioni

- Dato che tutti i cammini sono disgiunti, con una qualsiasi visita (in ampiezza, in profondità, con dijkstra, ...) troviamo w_s e n_s per ogni cammino $\in \Pi$, in $\mathcal{O}(N + M)$.
- I cammini sono al massimo N, quindi compariamo tutti i cammini $\in \Pi \setminus \Pi_{MATTEL}$ con tutti i cammini $\in \Pi_{MATTEL}$ in $\mathcal{O}(N^2)$.

Cammini disgiunti

$$s \in \Pi \setminus \Pi_{MATTEL}. \quad \forall r \in \Pi_{MATTEL}. \quad w_s + n_s \cdot K < w_r + n_r \cdot K$$

Osservazioni

$$w_s + n_s \cdot K < w_r + n_r \cdot K$$
$$(n_s - n_r) \cdot K < w_r - w_s$$

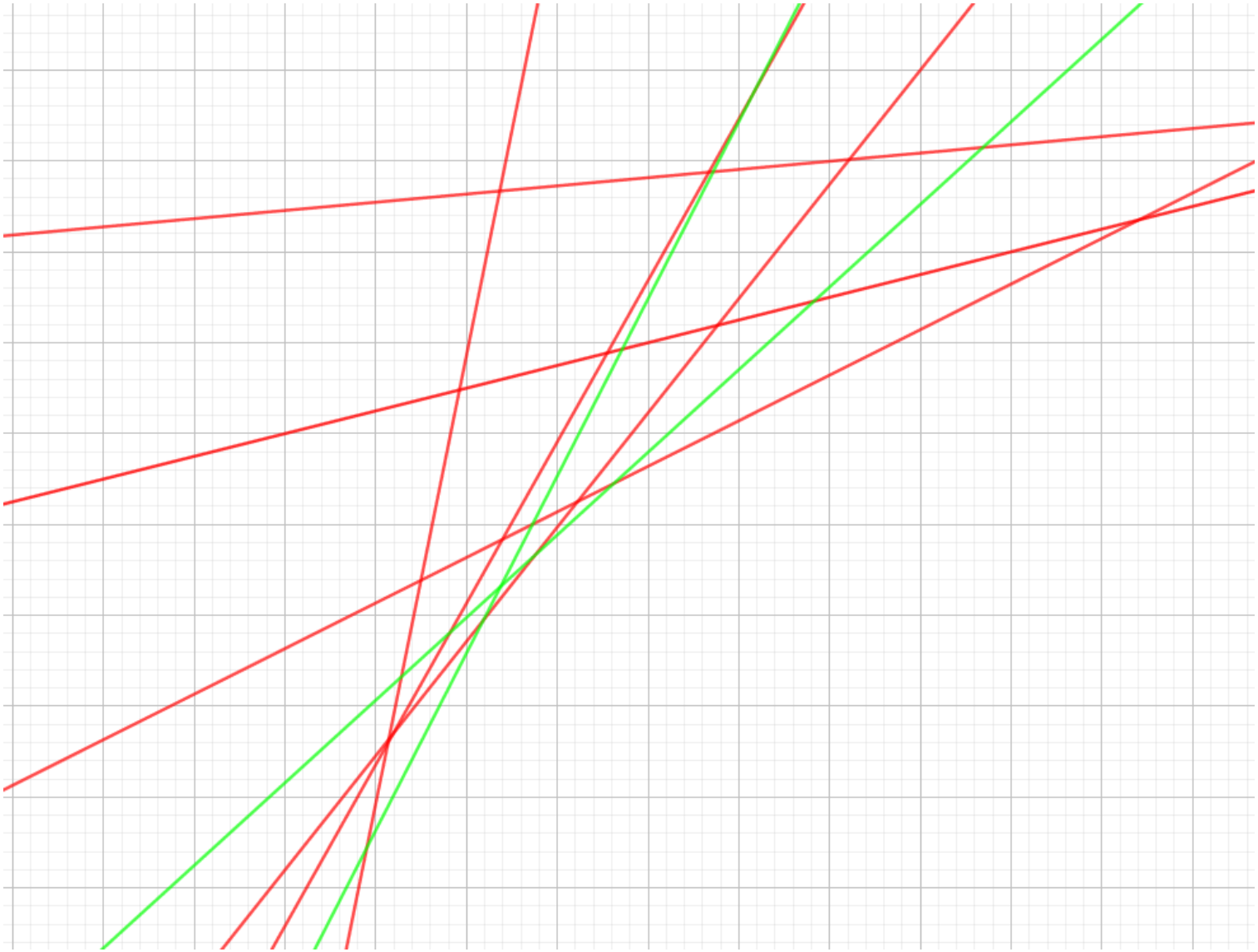
Quando $n_s - n_r > 0$:
Upper bound

$$K < \frac{(w_r - w_s)}{(n_s - n_r)}$$

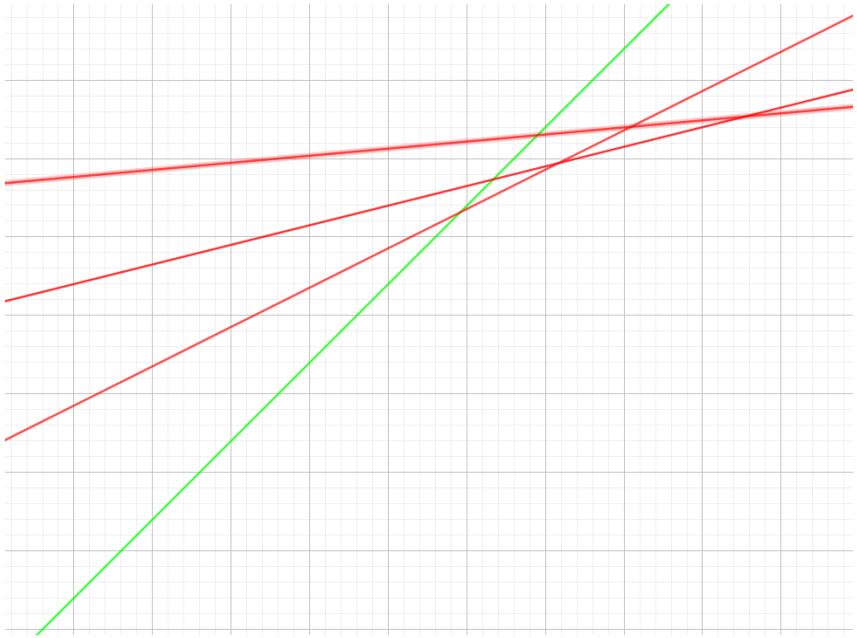
Quando $n_s - n_r < 0$:
Lower bound

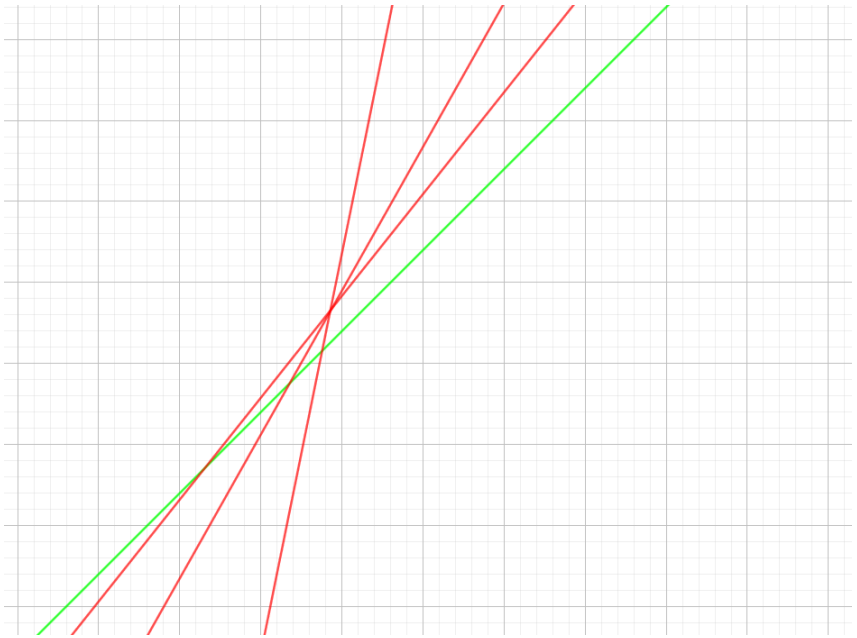
$$K > \frac{(w_r - w_s)}{(n_s - n_r)}$$

Cammini disgiunti

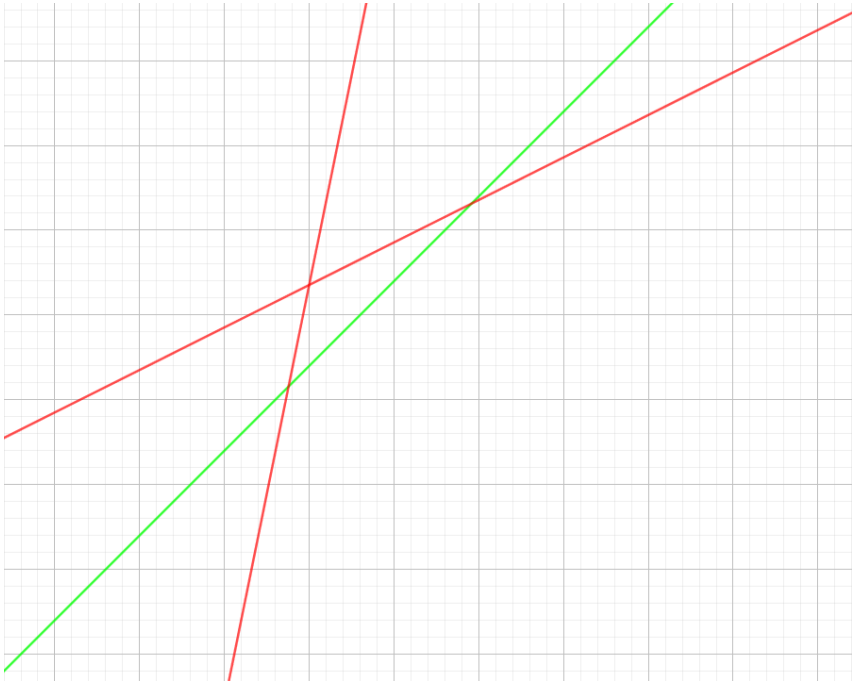


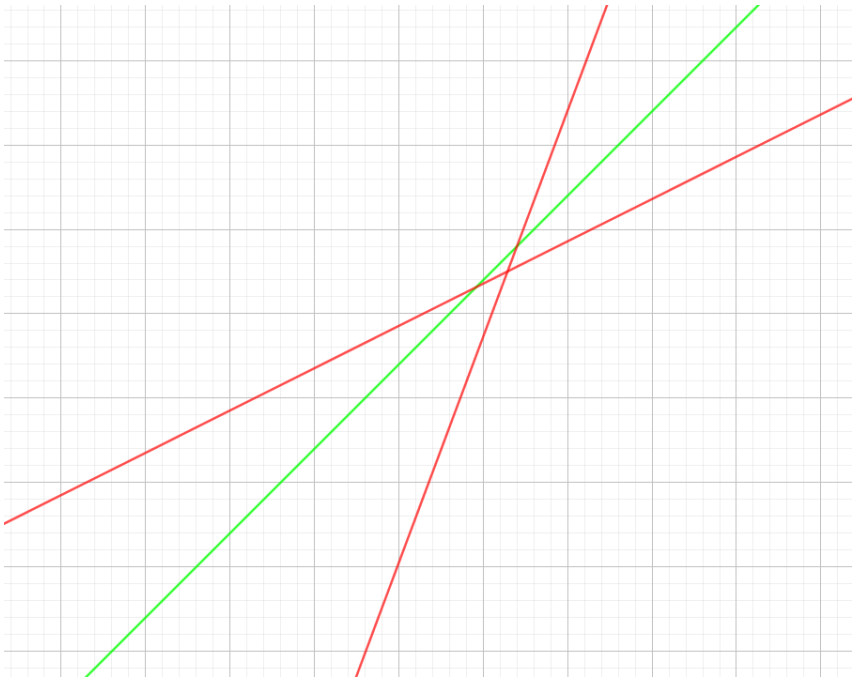
Cammini disgiunti



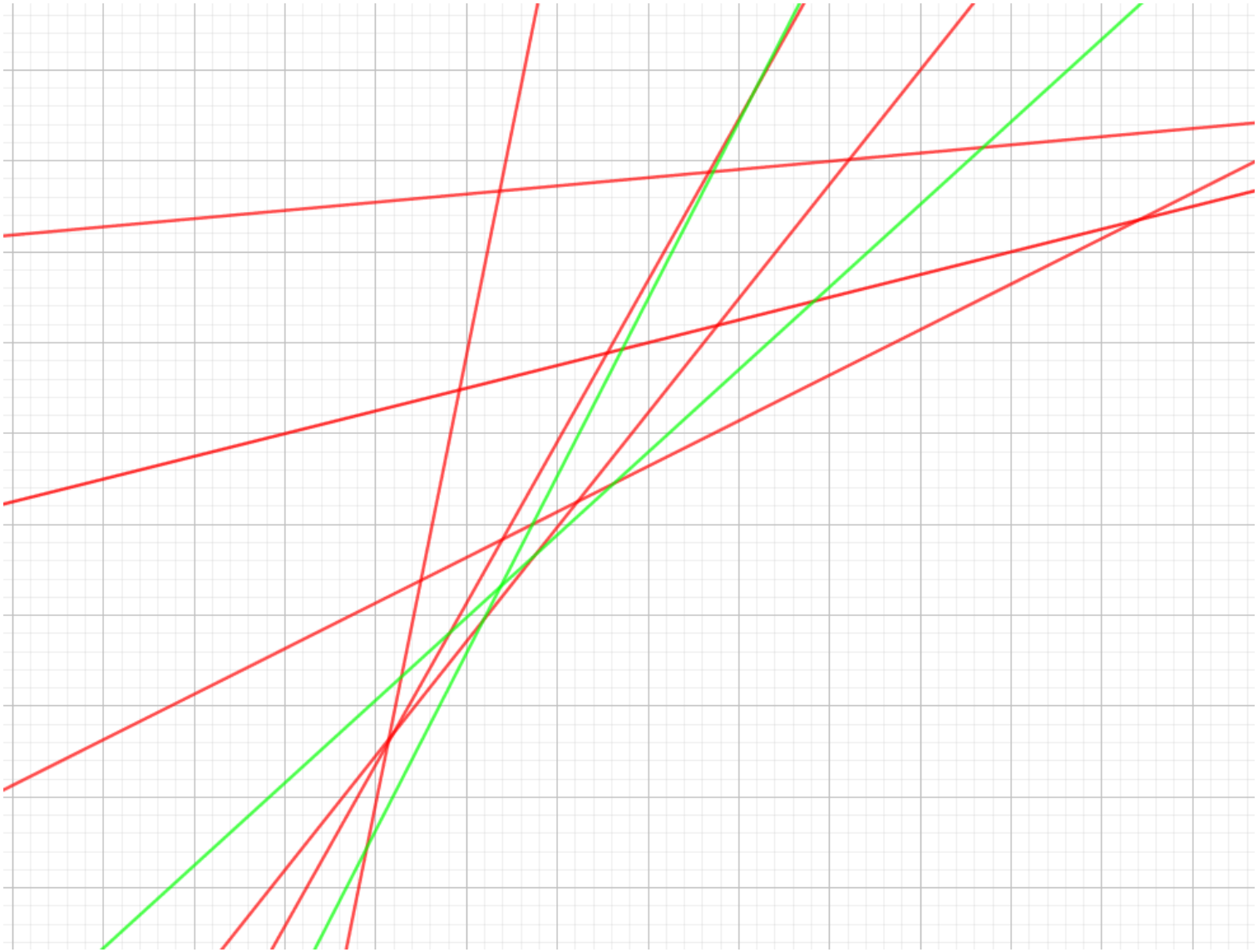


Cammini disgiunti





Cammini disgiunti



CAMMINI DISGIUNTI

```
long long ComparaIntervalli(
        const vector<pair<int, long long>> &min_dist,
        const vector<pair<int, long long>> &min_dist_safe) {
        //entrambi ordinati in base al numero di archi dei
            percorsi
  if (min_dist_safe.size() == 0) {
    return -2;
  } else if (min_dist.size() == 0) {
    return -1;
  }
  // .first = numero di archi, .second = costo con k nullo
  int a = min_dist_safe[0].first;
  int b = min_dist[0].first;

  if (a < b || (a == b && min_dist_safe[1].second <
                          min_dist[1].second)) {
    return -1;
  }

  long long max_k = -2;
```

Cammini disgiunti

```
for (int i = 0; i < min_dist_safe.size(); i++) {
  long long max_k_act = LONG_LONG_MAX;
  int j;
  for (j = 0; j < min_dist.size() && min_dist[j].first <
                              min_dist_safe[i].first; j++) {
    if ((min_dist[j].second - min_dist_safe[i].second) %
        (min_dist_safe[i].first - min_dist[j].first) == 0) {
      max_k_act =
        min((min_dist[j].second - min_dist_safe[i].second) /
            (min_dist_safe[i].first - min_dist[j].first) - 1,
                max_k_act);
    } else {
      max_k_act =
        min((min_dist[j].second - min_dist_safe[i].second) /
            (min_dist_safe[i].first - min_dist[j].first),
               max_k_act);
  }}
```

Cammini disgiunti

```
if (max_k_act <= 0) {
  max_k_act = -2;
  continue;
}

if (j < min_dist.size() &&
    min_dist[j].first == min_dist_safe[i].first &&
    min_dist[j].second <= min_dist_safe[i].second) {
  max_k_act = -2;
  continue;
}
```

Cammini disgiunti

```
for (; j < min_dist.size(); j++) {
  if (min_dist[j].second <= min_dist_safe[i].second) {
    long long min_k_act =
          ((min_dist_safe[i].second - min_dist[j].second) /
           (min_dist[j].first - min_dist_safe[i].first)) + 1;
    if (min_k_act > max_k_act) {
      max_k_act = -2;
      break;
    }
  }
}
max_k = max(max_k, max_k_act);
```

Complessità

$\Rightarrow$ complessità:

$$\mathcal{O}(M + N + N^2) = \mathcal{O}(N^2)$$

infatti se i cammini sono disgiunti

$$M = \mathcal{O}(N)$$

$\Rightarrow$ 70 punti

Troppi percorsi, troppe scelte possibili

Caso generale

Osservazioni

- Per risolvere il caso generale è quindi necessario generare i costi dei cammini minimi per ogni possibile lunghezza in archi
- Possiamo raggiungere questo risultato modificando diversi algoritmi, uno di questi è Bellman-Ford

Caso generale

```
v<v<p>> SP(C, v<p>(C, {.w=INFTY,.mattel=false,.parent=MAX_N}));
SP[0][B] = {.w=0,.mattel=is_mattel[B],.parent=B};

for (int steps = 1; steps < C; ++steps) {
  for (const auto e: E) {
    const node from = e.first.first, to = e.first.second;
    const int w = e.second;
    const int attempt = SP[steps - 1][from].w + w;
    if (attempt <= SP[steps][to].w)
      SP[steps][to] = {
            .w=attempt,
            .mattel=SP[steps - 1][from].mattel
                    || is_mattel[to]
                    || (attempt == SP[steps][to].w
                                && SP[steps][to].mattel),
            .parent=from
      };
} }
```

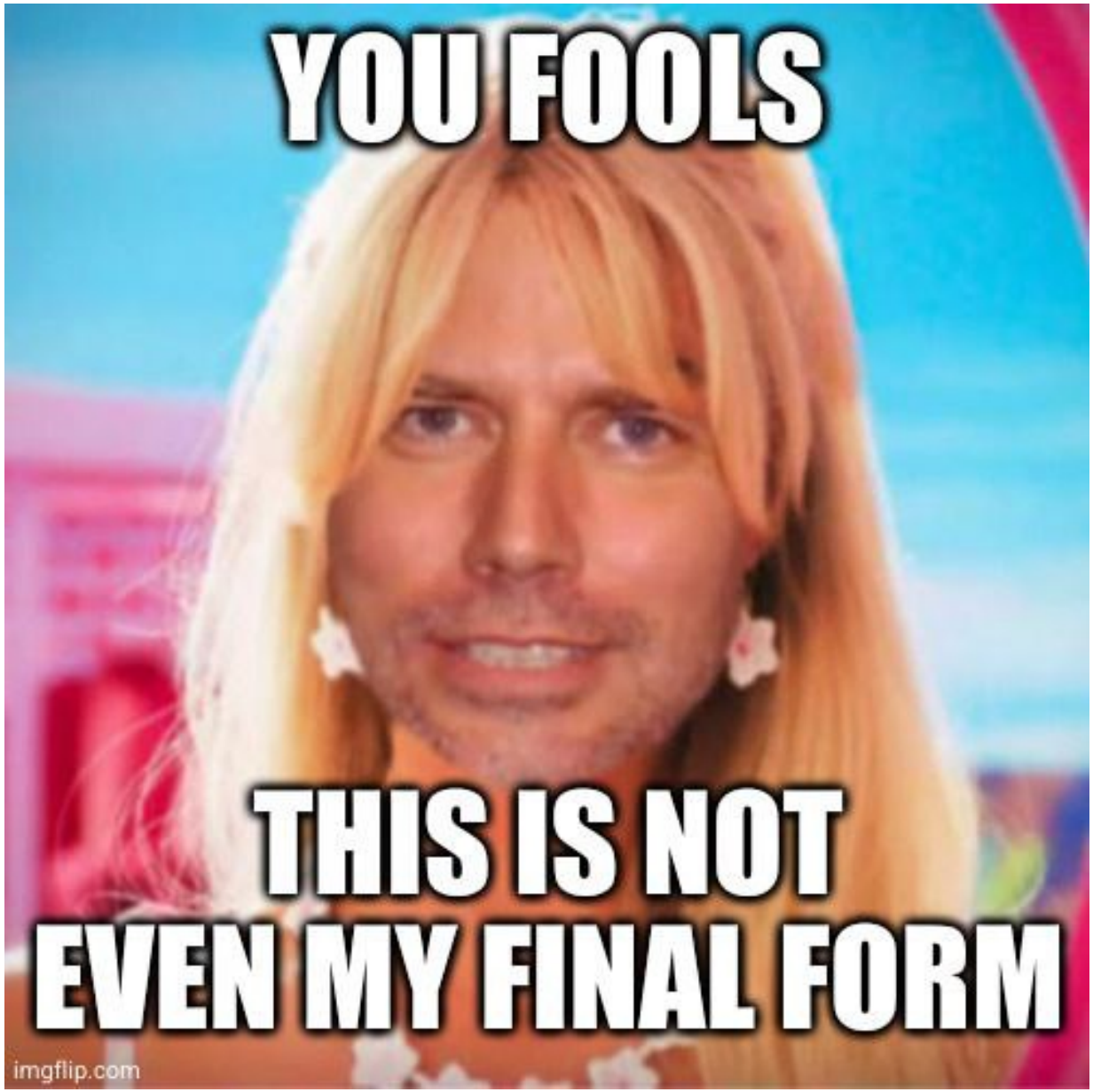
Complessità

- ⇒ soluzione: `sol-100-inequalities.cpp`
- ⇒ complessità:

$$\mathcal{O}(M \cdot N + N^2)$$

- ⇒ 100 punti

Si, sembrava difficile, ma siamo stati gentili



Potevamo farvi soffrire

- Si possono generare i cammini più velocemente?
- A livello di complessità no (o almeno così crediamo)
- Ma si possono usare algoritmi con performance migliori

Ma farvi soffrire davvero tanto

- Si possono comparare i cammini più velocemente?
- Sì, in

$$\mathcal{O}(N)$$

Però vi vogliamo bene <3

- Ci interessa che lo sappiate fare?
- Come obbiettivo del progetto no
- Ma provarci non vi farà perdere troppa sanità mentale