44. Bundeswettbewerb Informatik - 1. Runde / 1. Aufgabe

Luna Hagemann (Team-ID 00008 (Einzelteam), Teilnahme-ID 78245)

17. November 2025

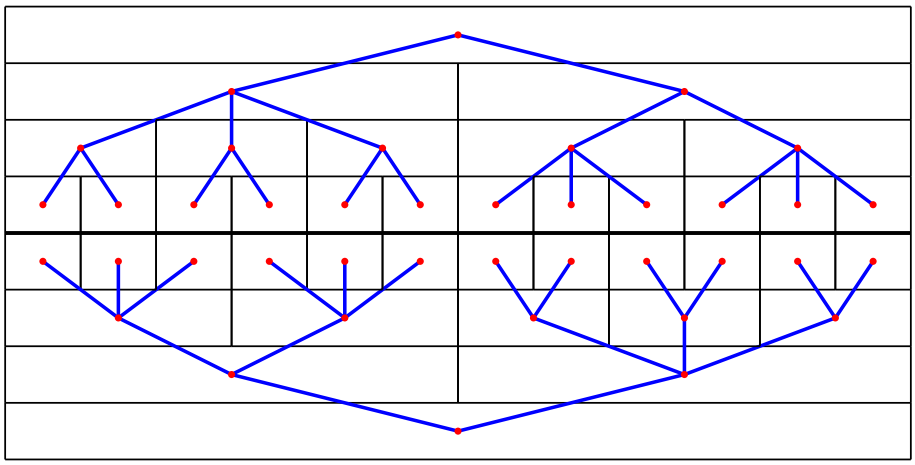
Inhaltsverzeichnis

1 Lösungsideen

1.1 1. Idee: string-comparison

Zuerst hatte ich die Idee den string genauer anzugucken, um dort direkt eine Drehfreudigkeit abzuleiten. So war meine Idee, den string zu spiegeln, wobei öffnende Klammern zu schließenden Klammern sowie schließende Klammern zu öffnenden Klammern umgewandelt werden sollten und dann der string Zeichen für Zeichen gespiegelt werden sollte. Dann wollte ich einen Vergleich durchführen, ob diese strings zu 100% übereinstimmen. Dieses Konzept funktioniert zwar bei einfachen Bäumen, jedoch nicht bei allen Bäumen wie im folgenden Beispiel, wo der Baum zwar drehfreudig ist, aber von dieser Methode nicht so gekennzeichnet werden würde:

normal: ((((() ()) (() ()) (() ())) ((() () ()) (() () ()))
gedreht: ((((() () ()) (() () ())) ((() ()) (() ()) (() ()))



1.2 2. Idee: rekursive Funktion

Deswegen musste der etwas komplexere Weg genommen werden, die Größe aller Blätter zu errechnen.

1.2.1 Erkennung der Drehfreudigkeit

Da die Eingabedatei nahezu einem Python tuple entspricht, habe ich zuerst die Eingabe in einen tuple umgewandelt. Durch diesen tuple kann ich nun rekursiv durchgehen, wobei ich immer sobald ich an einem Blatt bin die Breite und Tiefe dieses Blattes notiere, bzw. zurück gebe. Dadurch gibt es dann am Ende eine Reihenfolge an Blättern, zusammen mit ihrer Breite und Tiefe. Wenn diese Liste nun gespiegelt wird und die gleiche Liste herauskommt, ist dies das Gleiche wie eine Drehung, wodurch der Baum dann drehfreudig ist. Ist dies nicht der Fall, ist der Baum dementsprechend nicht drehfreudig.

Warum die Tiefe ausreichend ist Obwohl die Höhe übereinstimmen muss, damit der Baum drehfreudig ist, reicht es ebenfalls die Tiefe des Blattes zu bestimmen und diese als Alternative für die Höhe verwenden. Die Höhe von Blättern kann auch durch die maximale Tiefe minus der Tiefe des aktuellen Blattes ausgerechnet werden ($max(t_{\text{Blätter}}) - t_{\text{Blatt}} = h_{\text{Blatt}}$) dies kann auch für das gegenüberliegende Blatt von dem gedrehten Baum errechnet werden. $h_{\text{Blatt}} \stackrel{?}{=} h'_{\text{Blatt}}$ bestimmt, ob das Blatt drehfreudig ist. Nun können wir dafür die Formeln von oben einsetzen:

$$max(t_{\text{Blätter}}) - t_{\text{Blatt}} \stackrel{?}{=} max(t'_{\text{Blätter}}) - t'_{\text{Blatt}} \tag{1}$$

Da die maximale Tiefer der Blätter von beiden Bäumen gleich sein muss, können wir sie kürzen und die Vorzeichen tauschen ($| - max(t_{\text{Blätter}})$ & $| * -1$):

$$t_{\text{Blatt}} \stackrel{?}{=} t'_{\text{Blatt}} \tag{2}$$

Daraus ist klar, dass die Tiefe des Blattes auch ausreicht und nicht die Höhe benötigt wird.

1.2.2 Erstellung des Bilds

Nun kann das Bild erstellt werden, mit dem die Drehfreudigkeit gezeigt oder wiederlegt wird. Dafür wird die davor generierte Liste der Blätter ebenfalls übergeben, aber nur um zu prüfen was die maximale Tiefe ist um dafür die entsprechenden Kästchen von Blättern weit genug zu zeichnen. Sonst gibt es hierfür eine weitere rekursive Funktion, welche das SVG für jeden Knoten/Blatt generiert und dann in eine gemeinsame Datei hinzufügt. Die Elemente für Punkte auf dem Bild werden aber zuerst separat gespeichert, um diese in der SVG-Datei am Ende zu platzieren. Dadurch kann gewährleistet werden, dass die Punkten immer über allen Linien sind und Linien Punkte nicht teilweise verdecken. Über eine SVG-Gruppe kann dies dann einmal um 180° gedreht geklont werden, um die Drehfreudigkeit zu zeigen.

2 Umsetzung

2.1 Erkennung der Drehfreudigkeit

Die Verarbeitung des Inputs kann hier gut von Python übernommen werden. Nach jeder geschlossenen Klammer wird ein Komma hinzugefügt um einen tuple-like string zu erhalten. Nun kann die eingebaute Funktion eval() verwendet werden, um diesen string in einen echten nested tuple zu verwandeln. (*Hinweis:* eval() kann ebenfalls schädliche Befehle ausführen, dies passiert aber in diesem Rahmen nicht, da nur kontrollierte Inputs verarbeitet werden. Für mehr Sicherheit kann auch die Funktion literal_eval () des Paketes ast verwendet werden oder eine weiter rekursive Funktion kann programmiert werden, um dies selber zu implementieren (dann ist der vorherigen Schritt (das Hinzufügen von Kommas) obsolet.). Ich habe mich gegen literal_eval () entschieden, da hier keine Gefahr besteht und dafür keine neue Bibliothek importiert werden musste.)

Nun kann durch eine rekursive Funktion mit einer initialen Breite von 1 und einer Tiefe von 0 durch jeden Knoten/Blatt durchgegangen werden, solange bis der Knoten keine weiteren tuples mehr in sich hat, was bedeutet, dass dieser Knoten ein Blatt sein muss, wofür dann in eine Liste die Breite und Tiefe gespeichert werden kann. Falls der Knoten kein Blatt ist, sondern weitere tuples in sich hat, wird für diese jeweils nochmal die Funktion selber aufgerufen, diesmal mit einer Tiefe, die um eins höher ist als die vorherige Tiefe (s. 1.2.1) und einer Breite von der aktuellen Breite / die Anzahl der Kinder dieses Knotens.

Wurde nun diese Liste erstellt, kann diese einmal gespiegelt (bzw. um 180° gedreht) werden und nur wenn die beiden Listen übereinstimmen ist der gegebene Baum drehfreudig.

2.2 Erstellung des Bilds

Eine weitere rekursive Funktion wird benutzt, um für jeden Knoten die zugehörigen SVG-Elemente zu zeichnen. Dazu wird von der Berechnung der Drehfreudigkeit die maximale Tiefe eines Knotens genommen, um immer zu wissen, wie weit ein Kästchen gezeichnet werden muss.

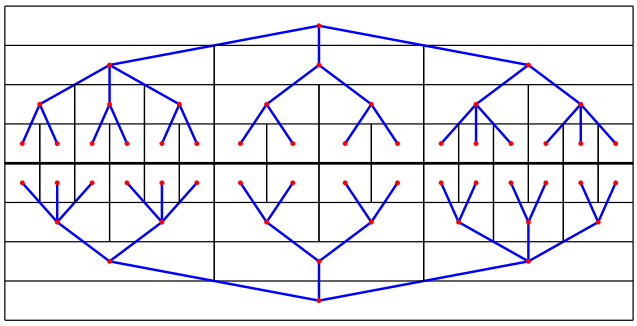
Jeder Knoten erstellt, je nachdem ob er ein Blatt ist oder nicht, das entsprechende SVG und speichert es zuerst in einer Variable. Bei Blättern ist die Besonderheit, dass das Kästchen bis zur maximalen Tiefe gezogen wird. Wenn es kein Blatt ist, wird erstmal rekursiv das SVG für alle Kind-Elemente generiert. Dann wird noch, falls in der Funktion eine Position des Eltern-Knoten übergeben wurde, eine Verbindungslinie zu diesem Knoten gezogen.

Zuletzt werden alle generierten Elemente in eine SVG-Gruppe gepackt, welche dann über ein <use>-tag geklont und gedreht wird.

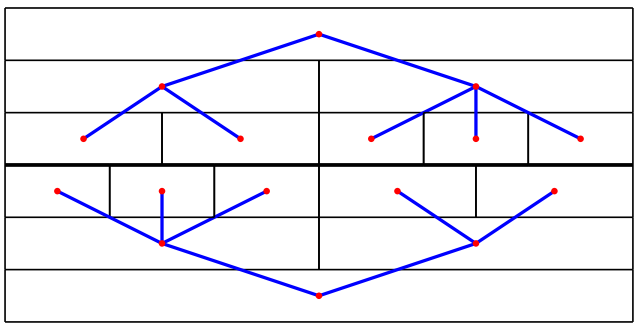
3 Beispiele

Beispiel	Drehfreudig?
drehfreudig01	Drehfreudig!
drehfreudig02	Nicht drehfreudig.
drehfreudig03	Nicht drehfreudig.
drehfreudig04	Drehfreudig!
drehfreudig05	Nicht drehfreudig.
drehfreudig06	Nicht drehfreudig.
drehfreudig07	Nicht drehfreudig.
drehfreudig08	Nicht drehfreudig.
drehfreudig09	Nicht drehfreudig.
drehfreudig10	Drehfreudig!
drehfreudig11	Nicht drehfreudig.
drehfreudig12	Nicht drehfreudig.
drehfreudig13	Nicht drehfreudig.
drehfreudig14	Nicht drehfreudig.
drehfreudig15	Nicht drehfreudig.

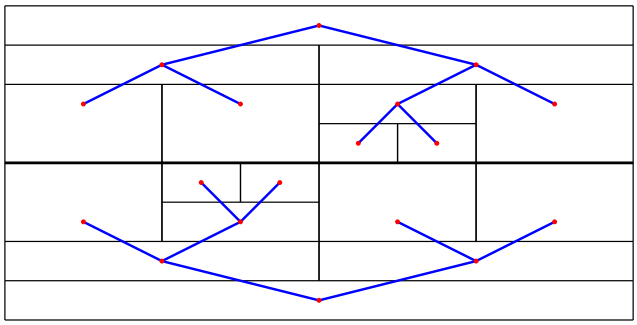
3.0.1 drehfreudig_01



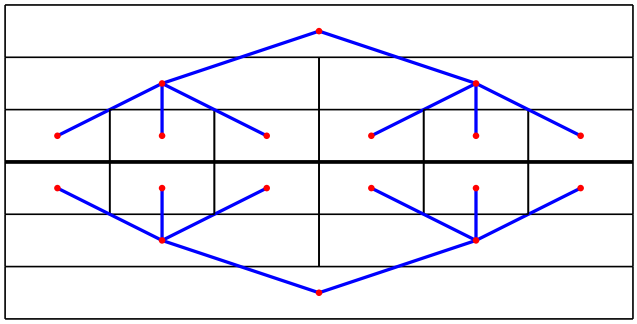
3.0.2 drehfreudig_02



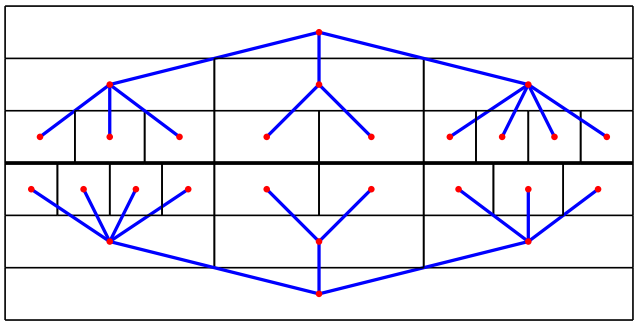
3.0.3 drehfreudig_03



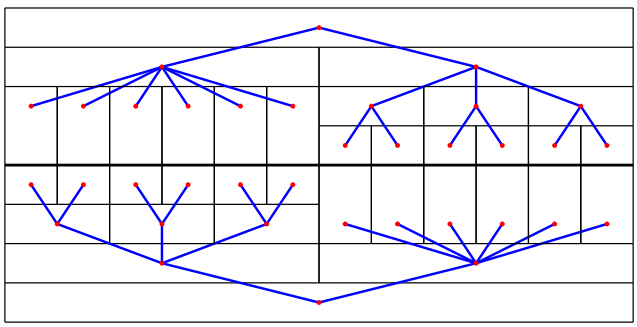
3.0.4 drehfreudig_04



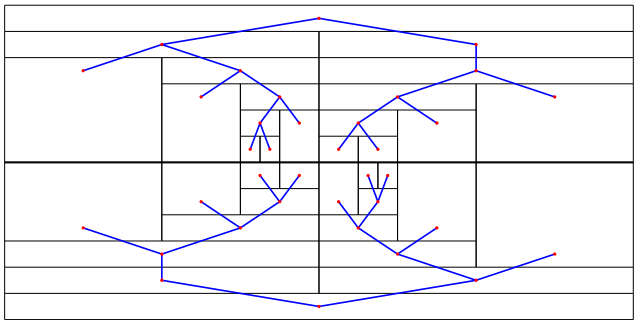
3.0.5 drehfreudig_05



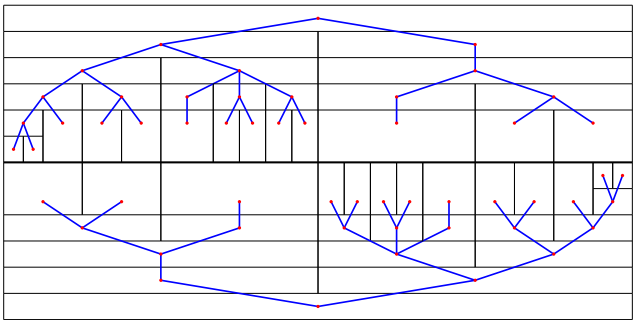
3.0.6 drehfreudig_06



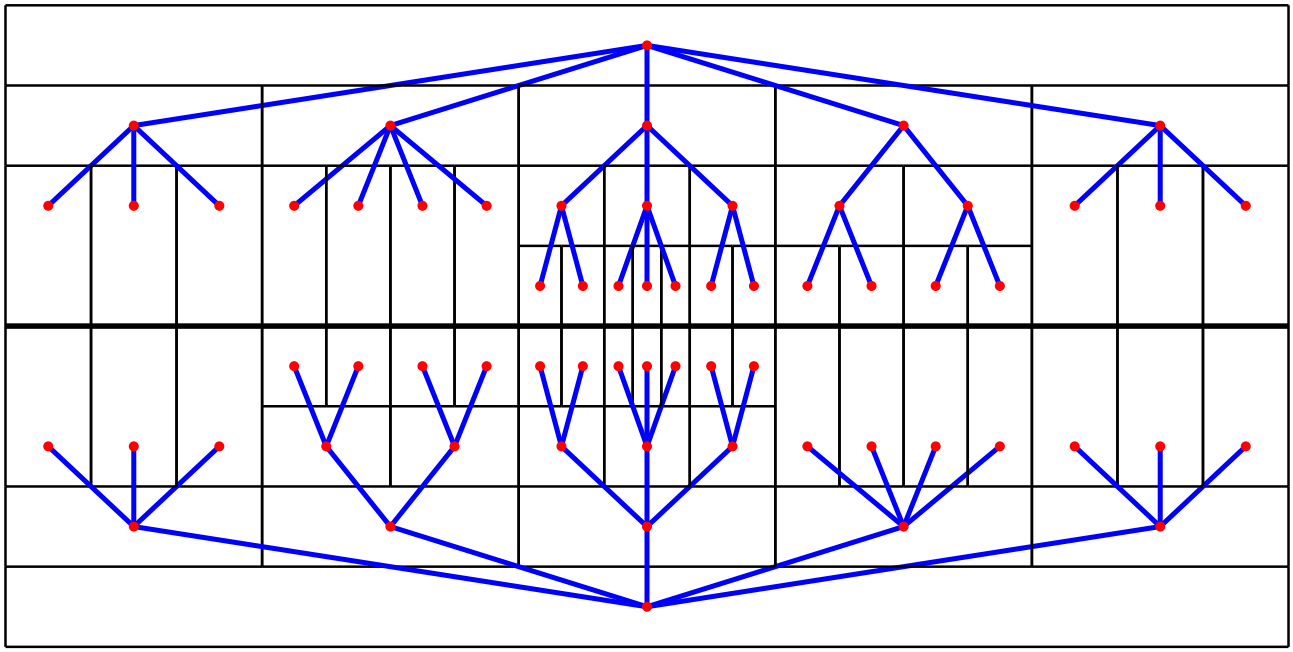
3.0.7 drehfreudig_07



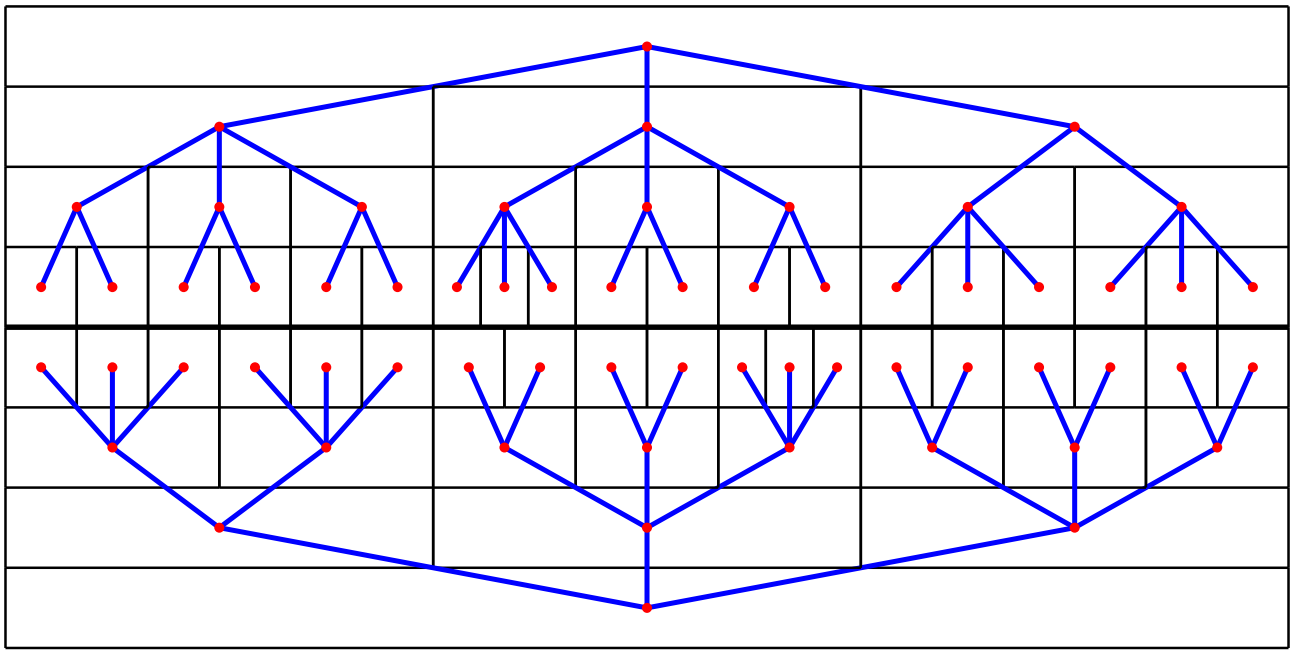
3.0.8 drehfreudig_11



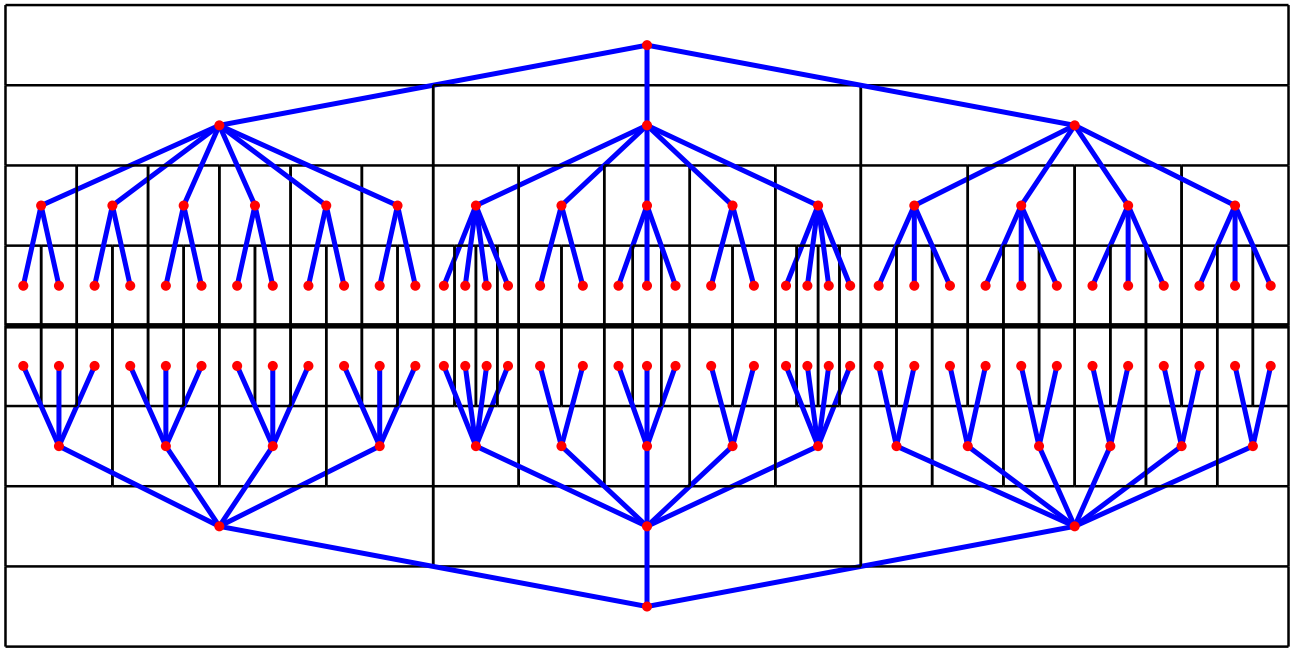
3.0.9 drehfreudig_08



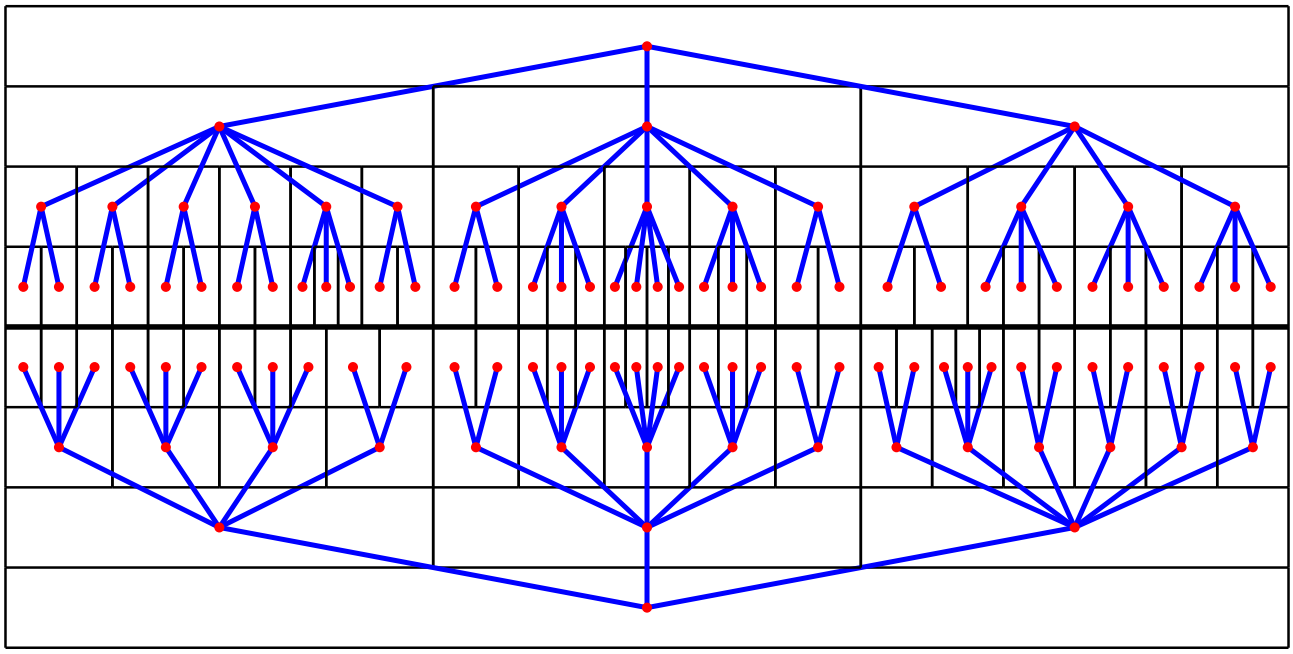
3.0.10 drehfreudig_09



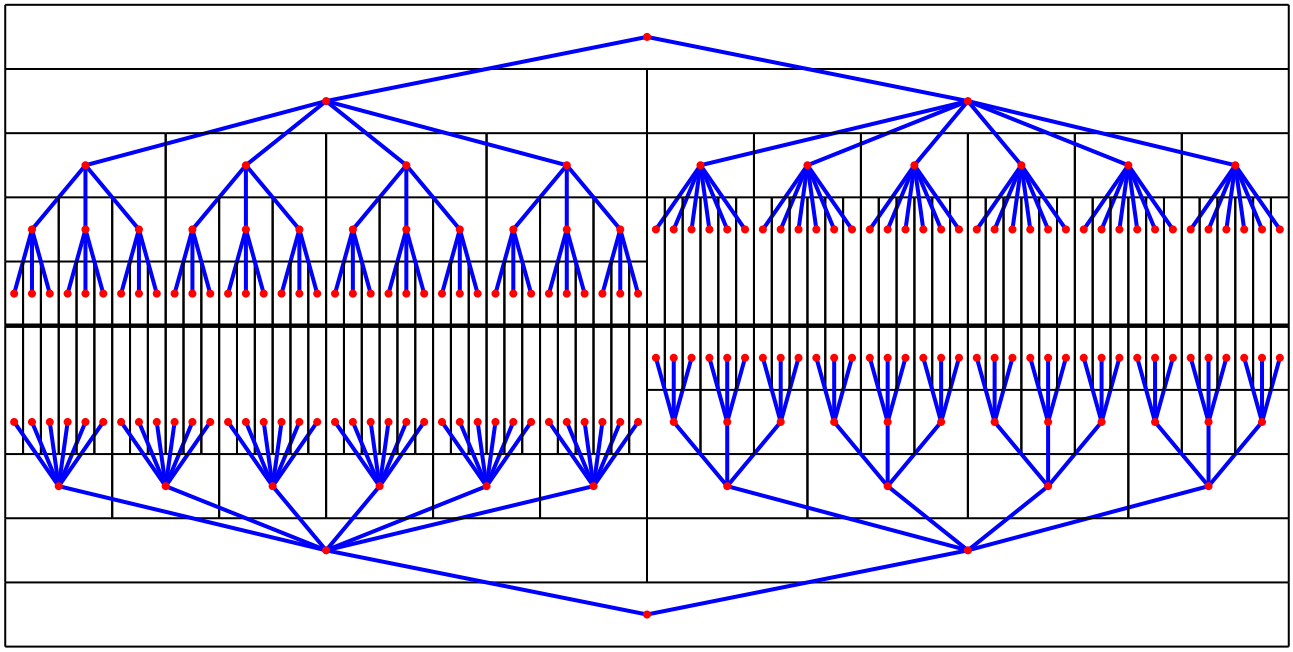
3.0.11 drehfreudig_10



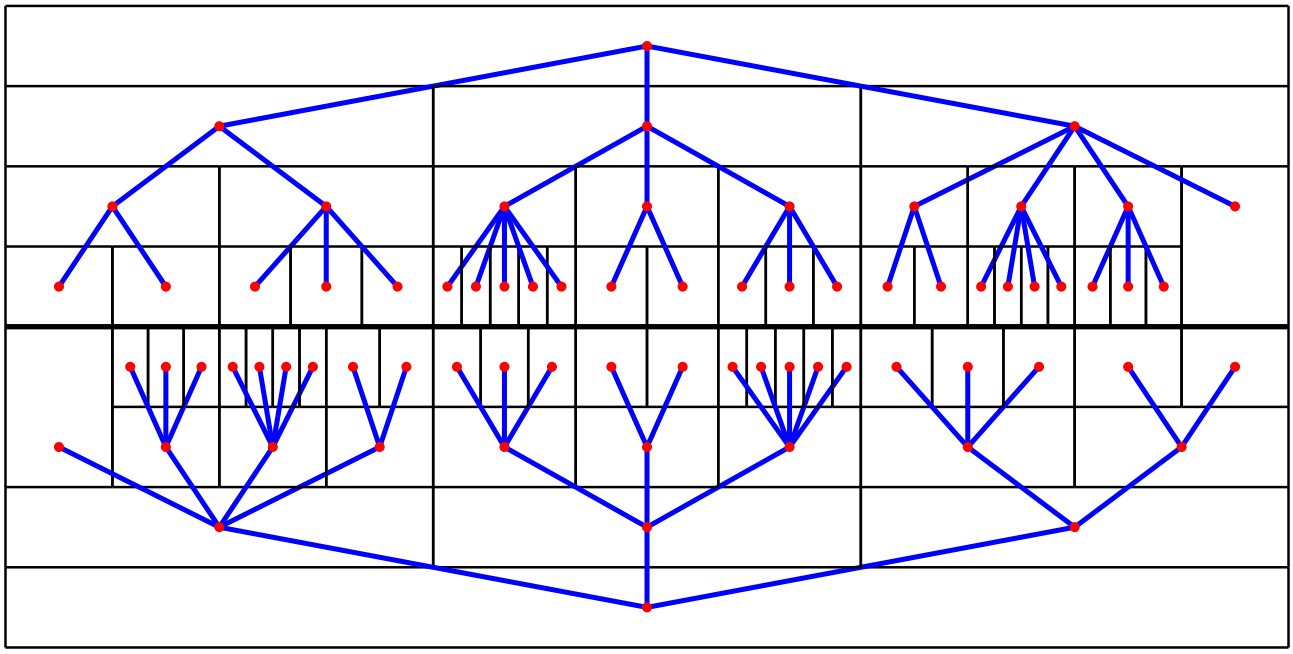
3.0.12 drehfreudig_12



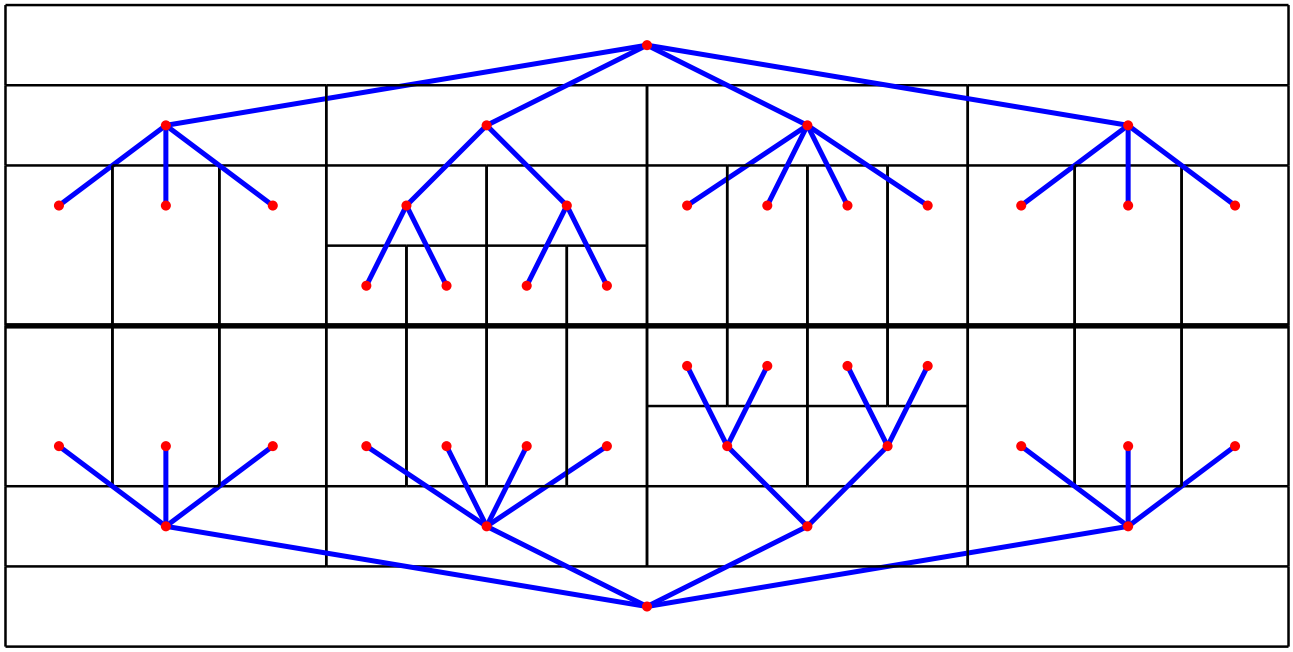
3.0.13 drehfreudig_13



3.0.14 drehfreudig_14



3.0.15 drehfreudig_15



4 Quelltext

```
def main():
    with open('./inp/drehfreudig16.txt', 'r') as f:
        txt = f.read().strip()
    tree = eval(txt.replace(')', '),').strip(',')) # use ast.literal_eval() for safer alternative
    leafs = explore_leafs(tree)
    print('Drehfreudig!' if leafs == leafs[::-1] else 'Nicht drehfreudig.')
    csvg(tree, leafs)


def explore_leafs(tree: tuple[tuple | None], width: float = 1, depth: int = 0) -> tuple[tuple[float, int]]:
    if not tree:
        return ((width, depth),)
    else:
        leafs = []
        for node in tree:
            leafs.extend(explore_leafs(node, width / len(tree), depth + 1))
        return tuple(leafs)


def rcsvg(tree: tuple[tuple | None], width: float = 1, depth: int = 0, maxd: int = 0, pos: tuple[float, float] = (0, 0), ppos: tuple[float, float] | None = None) -> tuple[str, str]:
    if tree:
        ls = f'''
    <line x1="{pos[0]}" y1="{pos[1]}" x2="{pos[0] + width}" y2="{pos[1]}" stroke="#000000" stroke-width="0.03125"></line>
    <line x1="{pos[0]}" y1="{pos[1]}" x2="{pos[0]}" y2="{pos[1] + 1}" stroke="#000000" stroke-width="0.03125"></line>
    <line x1="{pos[0] + width}" y1="{pos[1]}" x2="{pos[0] + width}" y2="{pos[1] + 1}" stroke="#000000" stroke-width="0.03125"></line>
    '''.strip().replace('\n', '').replace('\t', '')
        cs = f'''
```

```
        ls += f'''<circle cx="{pos[0] + width / 2}" cy="{pos[1] + .5}" r="0.0625" fill="#ff0000"></circle>
        '''.strip().replace('\n', '').replace('\t', '')
        xo = 0
        for node in tree:
            gen = rcsvg(node, width = width / len(tree), depth = depth + 1, maxd = maxd, pos = (pos[0] + xo, pos[1] + 1), ppos = (pos[0] + width / 2, pos[1] + .5))
            ls += gen[0]
            cs += gen[1]
            xo += width / len(tree)
    else:
        ls = f'''
        <line x1="{pos[0]}" y1="{pos[1]}" x2="{pos[0] + width}" y2="{pos[1]}" stroke="#000000" stroke-width="0.03125"></line>
        <line x1="{pos[0]}" y1="{pos[1]}" x2="{pos[0]}" y2="{maxd + 1}" stroke="#000000" stroke-width="0.03125"></line>
        <line x1="{pos[0] + width}" y1="{pos[1]}" x2="{pos[0] + width}" y2="{maxd + 1}" stroke="#000000" stroke-width="0.03125"></line>
        <line x1="{pos[0]}" y1="{maxd + 1}" x2="{pos[0] + width}" y2="{maxd + 1}" stroke="#000000" stroke-width="0.0625"></line>
        '''.strip().replace('\n', '').replace('\t', '')
        cs = f'''
        <circle cx="{pos[0] + width / 2}" cy="{pos[1] + .5}" r="0.0625" fill="#ff0000"></circle>
        '''.strip().replace('\n', '').replace('\t', '')
    if ppos: # ppos: parent position
        ls += f'<line x1="{ppos[0]}" y1="{ppos[1]}" x2="{pos[0] + width / 2}" y2="{pos[1] + .5}" stroke="#0000ff" stroke-width="0.0625"></line>' # move to fg / topmost layer?
    return ls, cs # ls: line elements; cs: circles (dots)


def csvg(tree: tuple[tuple | None], leafs: tuple[tuple[float, int]]):
    y = (max(leaf[1] for leaf in leafs) + 1) * 2
    x = len(leafs)
    gen = '\n'.join(rcsvg(tree, width = x, maxd = max(leaf[1] for leaf in leafs)))
    svg = f'''
    <svg width="{x}cm" height="{y}cm" viewBox="0, 0, {x}, {y}" version="1.1" xmlns="http://www.w3.org/2000/svg">
        <g id="graph">
            {gen}
        </g>
        <use href="#graph" y="{y / 2}" transform="translate(0, {y / 2}) rotate(180, {x / 2}, {y / 2})"></use>
    </svg>
    '''.strip().replace('\n', '').replace('\t', '') # BUG: transform doesn't work yet

    with open('./output.svg', 'w') as f:
        f.write(svg)


if __name__ == '__main__':
    main()
```