Aufgabe 4: Streichholzrätsel

Team-ID: 00537

Team-Name: LuC++

Bearbeiter dieser Aufgabe:
Lucas Schwebler

14. November 2020

Inhaltsverzeichnis

1 Lösungsidee

Sei S die Menge der Streichhölzer, die am Anfang vorliegen und Z die des Zielzustandes.

Nun wird über alle Paare von Streichhölzern (u, v) mit $u \in S \wedge v \in Z$ iteriert. Für jedes Paar wenden wir auf alle $x \in Z$ eine Transformation f an, die Streichhölzer verschiebt und dreht, sodass sie die Koordinate k auf k' abbildet und $f(v) = u$. Wir verschieben also alle Streichhölzer aus Z, sodass v auf u verschoben wird. Wenden wir f auf alle diese Streichhölzer an, erhalten wir die Menge Z'. Unser Ziel ist es nun $|S \cap Z'|$, also die Anzahl der Streichhölzer die nach der Verschiebung übereinstimmen, zu maximieren.

Um eine Umlegung auszugeben, ist $A = S \setminus Z'$ die Menge der Streichhölzer, die entfernt werden müssen und $B = Z' \setminus S$ die Menge der Streichhölzer die zu S dazugelegt werden müssen.

Anmerkung zur Aufgabenstellung: Nach dieser soll das Programm eine Anzahl umzulegender Streichhölzer als Eingabe nehmen. Diese Anzahl wird als Obergrenze interpretiert, da man einfach dasselbe Streichholz x den Mengen A und B hinzufügen könnte. Dann würde es erst weggelegt und dann wieder dazugelegt, womit man ein Streichholz mehr „umgelegt“ hätte. Außerdem geht es in Streichholzrätseln klassischerweise darum, eine minimale Anzahl an umzulegenden Streichhölzern zu finden.

Da das Programm aber so unabhängig von der eingegebenen Zahl immer dieselbe Umlegung finden wird, nämlich die minimale, wäre ein solcher Parameter sinnlos. Denn wenn das Programm die minimale Anzahl an Umlegungen ausgibt, so weiß man automatisch, dass es nicht mit weniger Umlegungen geht, aber mit einer Anzahl an Umlegungen, die kleiner gleich einer größeren Obergrenze ist. Daher wird diese Eingabe nicht von dem Programm entgegen genommen.

2 Umsetzung

Im Folgenden soll zunächst die mathematische Umsetzung der Transformation erklärt werden. Die Länge der Streichhölzer wird willkürlich auf 1 festgelegt, um leichter rechnen zu können.

2.1 Koordinaten

Verschiebung von Koordinaten können als Addition mit einem Vektor $\vec{v}$ interpretiert werden. Hat dieser Vektor einen Winkel α zur Horizontalen, so ist $\vec{v} = \begin{pmatrix} cos(\alpha) \\ sin(\alpha) \end{pmatrix}$, eine Verschiebung am Einheitskreis. Da die Streichholzfigur zusammenhängend ist, müssen alle Punkte, die in dieser enthalten sind, über solche Verschiebungen erreichbar sein. Da ferner α ein Vielfaches von 30° sein muss, kommen nur folgende Verschiebungen in Frage:

α	0°	30°	60°	90°	120°	150°	180°	210°	240°	270°	300°	330°	360°
x	1	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$	$-\frac{\sqrt{3}}{2}$	-1	$-\frac{\sqrt{3}}{2}$	$-\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	1
y	0	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	1	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$	$-\frac{\sqrt{3}}{2}$	-1	$-\frac{\sqrt{3}}{2}$	$-\frac{1}{2}$	0

Alle diese Werte sind ganzzahlige Vielfache von entweder $\frac{1}{2}$ oder $\frac{\sqrt{3}}{2}$. Somit lässt sich das Problem von Rundungsfehlern[1], das beim Verwenden von Fließkommazahlen entsteht, umgehen, indem wir alle Koordinaten als Paare $(a; b)$ speichern und diese dann als $a \cdot \frac{1}{2} + b \cdot \frac{\sqrt{3}}{2}$ interpretieren.

2.2 Rotationen

Verschiebungen sind relativ einfach zu implementieren: Wir addieren einfach einen Vektor zu dem Punkt. Bei Rotationen wird es jedoch etwas schwieriger.

2.2.1 Rotation um 90°

Sei $\vec{v}$ ein beliebiger zweidimensionaler Vektor. Dann erhält man $\vec{v'} = \begin{pmatrix} -v_y \\ v_x \end{pmatrix}$, indem man $\vec{v}$ um 90° gegen den Uhrzeigersinn um den Ursprung dreht.

Beweis. Das Kreuzprodukt $\vec{v} \times \vec{v'} = v_x \cdot v'_y - v_y \cdot v'_x = v_x \cdot v_x - v_y \cdot (-v_y) = v_x^2 + v_y^2 = |\vec{v}|^2$.
Allgemein gilt: $|\vec{u} \times \vec{v}| = |\vec{u}||\vec{v}|sin(\theta)$.
Da $|\vec{v}| = |\vec{v'}|$, folgt $|\vec{v}||\vec{v'}| = |\vec{v}|^2$ und somit $sin(\theta) = 1$, was bedeutet, dass $\theta = 90°$. □

2.2.2 Rotation um andere Winkel

Kennt man einen senkrechten Vektor, so lassen sich auch Vektoren mit anderen Rotationswinkeln bestimmen:

Sei $\vec{v}$ der zu rotierende Vektor, $\vec{v}_\perp$ der senkrechte Vektor (siehe 2.2.1) und $\vec{v'}$ der resultierende Vektor nach Rotation um den Winkel θ.

Dann bilden die Vektoren $\vec{a}$ und $\vec{b}$, die dieselbe Richtung wie $\vec{v}$ bzw. $\vec{v}_\perp$ besitzen, zusammen mit $\vec{v'}$ ein rechtwinkliges Dreieck (siehe Abbildung 1). Es gilt:

$$|\vec{a}| = cos(\theta) \cdot |\vec{v'}| \ \wedge \ \vec{a} = \frac{|\vec{a}|}{|\vec{v}|} \cdot \vec{v} \quad \Rightarrow \quad \vec{a} = \frac{|\vec{v'}| \cdot cos(\theta)}{|\vec{v}|} \cdot \vec{v} = cos(\theta) \cdot \vec{v}$$

Analog ist $\vec{b} = sin(\theta) \cdot \vec{v_\perp}$ und somit $\vec{v'} = \vec{a} + \vec{b} = cos(\theta) \cdot \vec{v} + sin(\theta) \cdot \vec{v_\perp}$, womit die gesuchte Formel gefunden wurde.

2.2.3 Rotation um beliebige Punkte

Soll nun nicht mehr um den Ursprung rotiert werden, sondern um den Punkt mit Ortsvektor $\vec{r}$, so lässt sich der zu rotierende Ortsvektor $\vec{v}$ als Summe $\vec{v} = \vec{r} + \vec{a}$ schreiben. Da $\vec{r}$ das Zentrum der Rotation ist, bleibt dieser Summand unverändert. $\vec{a'}$ können wir bereits mit dem Ansatz aus 2.2.2 bestimmen, da $\vec{a}$ um den Ursprung rotiert wird.

[1] Dieses Problem könnte man auch durch das Einführen eines ϵ Wertes bei Test auf Gleichheit umgehen, allerdings geht es hier auch mit ganzen Zahlen.

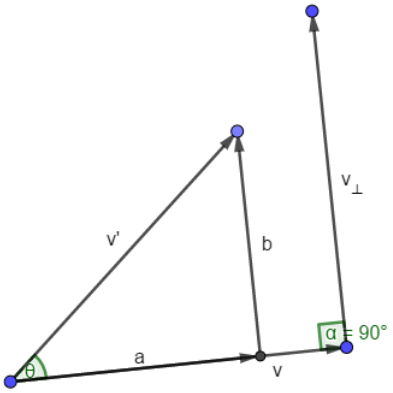

Abbildung 1: Rotation um beliebigen Winkel

2.3 Format

Für die Input- / Outputdateien wird ein Format verwendet, das für Menschen schwer zu lesen ist, mit dem das Programm aber sehr gut arbeiten kann. Ein Streichholz wird beschrieben durch 2 Koordinaten, von denen jede aus 2 ganzen Zahlen besteht (siehe 2.1), sowie eine Richtung r. Der Winkel zur Horizontalen $\alpha = r \cdot 30°$. Zu Beginn der Datei steht eine ganze Zahl n, die die Anzahl der Streichhölzer angibt. Es folgen n Zeilen, die die Streichhölzer der Startkonfiguration beschreiben und darauf von einer leeren Zeile getrennt n weiter Zeilen, die die Streichhölzer der Endkonfiguration angeben.

Die Outputdatei steht zunächst die Anzahl an umgelegten Streichhölzern und dann erst die entfernten Streichhölzer, wonach die dazugelegten Streichhölzer aufgeführt werden.

Beispiel:

```
streichholz0.txt (in)
7
0 0 0 0 0
0 0 0 0 3
2 0 0 0 3
0 0 2 0 0
0 0 2 0 2
2 0 2 0 4
1 0 2 1 3

0 0 0 0 3
0 0 0 0 8
0 0 0 0 10
0 0 2 0 8
0 0 2 0 10
-1 0 0 -1 3
1 0 0 -1 3

streichholz0.txt (out)
3
0 0 0 0 0
0 0 2 0 0
1 0 2 1 3

1 0 0 1 3
1 0 0 1 8
1 0 0 1 10
```

Damit aber auch Menschen in der Lage sind die Ergebnisse anschaulich nachzuvollziehen, enthält die Einsendung eine kleine Javascript Anwendung, die ein passendes Bild erzeugt. Rote Streichhölzer wurden beim Umlegen entfernt, grüne dazugelegt. Abbildung 2 zeigt das entsprechende Bild für die erste Datei.

3 Beispiele

Die Outputdateien werden hier nicht aufgeführt, stattdessen die Bilder. Allerdings sind hier zu jeder der Dateien die Anzahl an umzulegenden Streichhölzern zu finden:

```
streichholz0.txt: 3
streichholz1.txt: 4
streichholz2.txt: 3
streichholz3.txt: 8
streichholz4.txt: 2
streichholz5.txt: 4
```

Nun folgen die Bilder zu den Dateien:

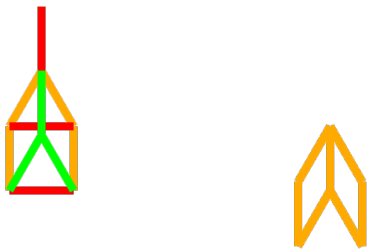

Abbildung 2: Lösung zu streichholz0.txt

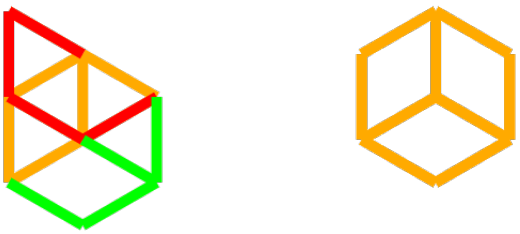

Abbildung 3: Lösung zu streichholz1.txt

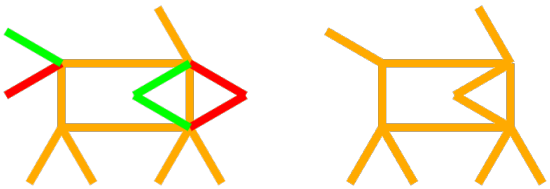

Abbildung 4: Lösung zu streichholz2.txt

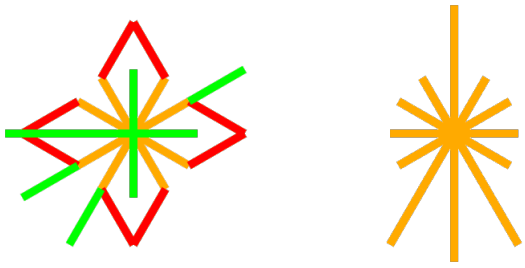

Abbildung 5: Lösung zu streichholz3.txt

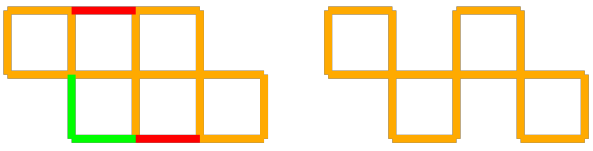

Abbildung 6: Lösung zu streichholz4.txt

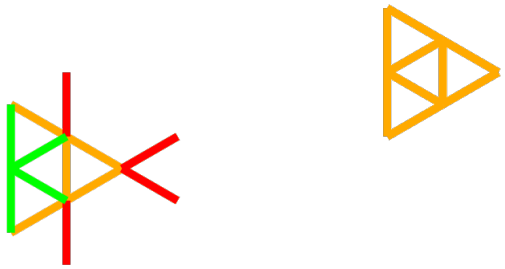

Abbildung 7: Lösung zu streichholz5.txt

4 Quellcode

Eigentliche Lösung:

```
void solve()
{
    int best = 0;
    // Itereiere über alle Paare (u, v)
    for(auto u : start)
    {
        for(auto v : ziel)
        {
            int match = 0;
            set<pair<Vec, int>> start_match;
            set<pair<Vec, int>> no_match;

            // lambda Funktion, die überprüft, ob ein Streichholz enthalten ist
            auto check = [&](pair<Vec, int> holz)
            {
                // Jedes Streichholz kann auf 2 Arten angegeben werden, indem man das
                // andere Ende als Vektor angibt und die Richtung invertiert
                pair<Vec, int> holz2 = {holz.first.move(holz.second), (holz.second + 6) % 12};
                // prüfe daher beide Orientierungen
                if(start.count(holz))
                {
                    match++;
                    start_match.insert(holz);
                }
                else if(start.count(holz2))
                {
                    match++;
                    start_match.insert(holz2);
                }
                else
                {
                    no_match.insert(holz);
                }
            };

            auto update_best = [&]()
            {
                if(match > best)
                {
                    best = match;
                    add = no_match;
                    rem.clear();
                    for(auto a : start)
                    {
                        if(!start_match.count(a)) rem.insert(a);
                    }
                }
            };

```

```
            // u - v    (so gewählt, dass v + translate = u)
            Vec translate = u.first.add(v.first.multiply(-1));
            // Winkel zwischen v und u (gegen den Uhrzeigersinn) in 30° Schritten
            int steps = (u.second - v.second + 12) % 12;
            // Wende die Transformation auf alle z element ziel an
            for(auto z : ziel)
            {
                // Verschiebe z um translate
                // Rotiere z um u um den Winkel alpha = steps * 30°
                Vec v = z.first.add(translate).rotate(u.first, steps);
                // Die Richtung von z'
                int dir = (z.second + steps) % 12;
                check({v, dir});
            }
            update_best();

            match = 0;
            start_match.clear();
            no_match.clear();

            // Wenn u und v übereinstimmen, können alle Streichhölzer
            // der Menge Z' gespiegelt werden, wonach u und v immernoch
            // übereinstimmen. Diese Variante muss also auch betrachtet werden:
            Vec trans2 = u.first.move(u.second).add(v.first.multiply(-1));
            int steps2 = (steps + 6) % 12;
            for(auto z : ziel)
            {
                Vec v = z.first.add(trans2).rotate(u.first, steps2);
                int dir = (z.second + steps2) % 12;
                check({v, dir});
            }
            update_best();
        }
    }
}
```

Sonstiger Code, auf den die Lösung zugreift:

```
int sign(int x)
{
    return x >= 0 ? 1 : -1;
}

struct PosComponent
{
    // PosComponent{a, b} = a * 1/2 + sqrt(3)/2 * b
    int a, b;
    PosComponent multiply(int x)
    {
        return {a * x, b * x};
    }
    PosComponent multSqrt3()
    {
        // sqrt(3) * (a * 1/2 + b * sqrt(3) / 2) = 3*b * 1/2 + a * sqrt(3)/2
        return {3 * b, a};
    }
    PosComponent divide2()
    {
        return {a / 2, b / 2};
    }
    PosComponent add(PosComponent x)
    {
        return {a + x.a, b + x.b};
    }
    bool operator < (const PosComponent &x) const
    {
        // 1/2 * a + sqrt(3)/2 * b < 1/2 * x.a + sqrt(3)/2 * x.b
        // a + sqrt(3) * b < x.a + sqrt(3) * x.b
        // (a - x.a) < sqrt(3) * (x.b - b)
        // sign(a - x.a) * (a - x.a)^2 < 3 * (x.b - b)^2 * sign(x.b - b)
        return sign(a - x.a) * (a - x.a) * (a - x.a) < 3 * (x.b - b) * (x.b - b) * sign(x.b - b);
    }
```

```
};

struct Vec
{
    PosComponent x, y;
    Vec multiply(int k)
    {
        return {x.multiply(k), y.multiply(k)};
    }
    Vec multSqrt3()
    {
        return {x.multSqrt3(), y.multSqrt3()};
    }
    Vec add(Vec v)
    {
        return {x.add(v.x), y.add(v.y)};
    }
    Vec move(int dir)
    {
        Vec v = *this;
        return v.add(Vec{{2, 0}, {0, 0}}.rotate(dir));
    }
    Vec divide2()
    {
        return {x.divide2(), y.divide2()};
    }
    Vec rotate(int steps)
    {
        Vec v = *this;
        Vec senkrecht = {y.multiply(-1), x};
        if(steps % 3 == 1)
        {
            v = v.multSqrt3().add(senkrecht).divide2();
            steps -= 1;
        }
        if(steps % 3 == 2)
        {
            v = senkrecht.multSqrt3().add(v).divide2();
            steps -= 2;
        }
        senkrecht = {v.y.multiply(-1), v.x};
        if(steps == 0) return v;
        if(steps == 3) return senkrecht;
        if(steps == 6) return v.multiply(-1);
        if(steps == 9) return senkrecht.multiply(-1);
        // ERROR
        assert(false);
        return *this;
    }
    Vec rotate(Vec center, int steps)
    {
        return this->add(center.multiply(-1)).rotate(steps).add(center);
    }

    bool operator < (const Vec& v) const
    {
        if(x < v.x) return true;
        if(v.x < x) return false;
        return y < v.y;
    }
};
```