Aufgabe 4 „Schrebergärten“ - Dokumentation

37. Bundeswettbewerb Informatik 2018/19 - 1. Runde

Sebastian Baron, Simon Fiebich, Lukas Rost
Team-ID: 00036

26. November 2018

Lösungsidee

Der einfachste Fall das Problem zu lösen, ist wenn alle Rechtecke die Gleiche Höhe (oder Breite) haben. Dann können einfach die einzelnen Rechtecke nebeneinander (aufeinander) gestellt werden. Da auf diese Weise keine Flächen, die nicht belegt werden, entstehen, ist diese Anordnung optimal. Ausgehend von dieser Idee kann man alle Rechtecke nebeneinander platzieren und das so entstehende Rechteck vorläufig als optimal annehmen. Nun versucht man ein kleineres Rechteck zu finden. Dies kann man systematisch tun, indem man die Breite des bisherigen optimalen Rechtecks schrittweise um 1 verkleinert. Natürlich muss dabei auch die Höhe justiert werden. Um alle Möglichkeiten abzudecken, kann man also alle Höhen zwischen der minimalen Höhe, also der Höhe, die mindestens nötig ist um, mit der momentanen Breite des Rechtecks multipliziert, mindestens die Summe der einzelnen Flächeninhalte der zu platzierenden Rechtecke zu erhalten, und der maximalen Höhe, bei der ihr Produkt mit der momentanen Breite größer wäre, als die Fläche des momentanen optimalen Rechtecks, durchprobieren. Dabei sollte man bei der minimalen Höhe anfangen, da, falls man ein kleineres optimales Rechteck findet, es redundant wird, die größeren Höhen durchzuprobieren.
Die Platzierung erfolgt dabei geordnet nach der Höhe, mit dem höchsten zuerst. Des weiteren wird jedes Rechteck so weit links wie möglich und, sollte es mehrere ‚linke‘ Positionen geben, so weit unten wie möglich platziert.

Umsetzung

Das Programm besteht aus 2 Teilen, die nacheinander ausgeführt werden müssen. Der 1. Teil ist BWInf4.jar. Er wurde in Java geschrieben und ist ein Konsolenprogramm, dass die Eingabe der Rechtecke regelt und die Aufgabe löst. Der 2. Teil ist DrawRectangles.py. Er wurde in Python geschrieben und ist auch ein Konsolenprogramm. Er dient dazu die in Coordinates.txt gespeicherte Lösung, die zuvor von BWInf4.jar ermittelt wurde, graphisch auszugeben.

BWInf.jar besteht aus 3 Klassen.

- **Main**. Diese Klasse ist die Hauptklasse, die den Algorithmus aufruft und Ein- und Ausgabe übernimmt.
- **Rectangles**. Diese Klasse beherbergt den eigentlichen Algorithmus und enthält eine Liste aller Rechtecke, außerdem repräsentiert sie das Rechteck, in dem die anderen angeordnet werden müssen.
- **Rectangle**. Diese Klasse repräsentiert die einzelnen Rechtecke mitsamt deren numerischen Merkmalen.

Zunächst wird beim Aufruf des Programms die main-Methode der Klasse Main aufgerufen. Diese erzeugt ein Objekt der Klasse Rectangles und regelt die Erstellung der einzelnen Rechtecke, sowie die Erstellung eines Arrays, der alle Rechtecke beherbergt.

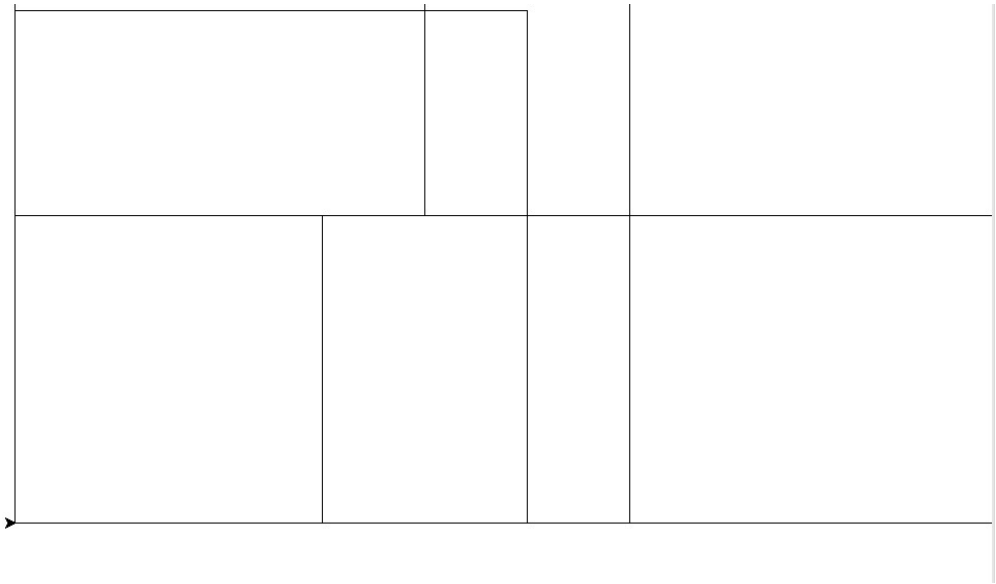
Nachdem alle Rechtecke erstellt wurden, werden diese mithilfe der sort() Methode der Klasse Rectangles sortiert. Danach wird der eigentliche Algorithmus aufgerufen. Dieser überprüft systematisch alle Rechtecke, die kleiner als das momentane optimale Rechteck sind, darauf, ob sie alle zu platzierenden Rechtecke beherbergen können. Sollte dies der Fall sein, wird das momentane optimale Rechteck durch das neu gefundene ersetzt und der Algorithmus wird fortgesetzt. Wenn dies geschehen ist, gibt die Main-Methode die Fläche des ermittelten Rechtecks aus und überschreibt Coordinates.txt mit den Koordinaten der einzelnen Rechtecke, so dass diese von DrawRectangles.py genutzt werden können.

DrawRectangles.py besteht nur aus der Main-Methode, die Coordinates.txt in eine Liste einliest und mithilfe des Pythonmoduls turtle die einzelnen Rechtecke zeichnet.

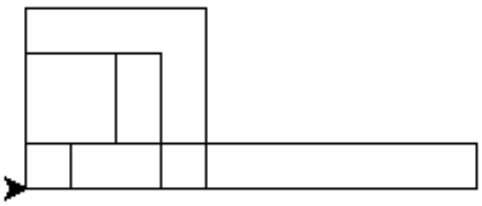
Hinweis: Die Implementierung funktioniert im aktuellen Zustand nicht. Es ist jedoch leider zeitlich nicht mehr möglich, diese fertigzustellen.

Beispiele

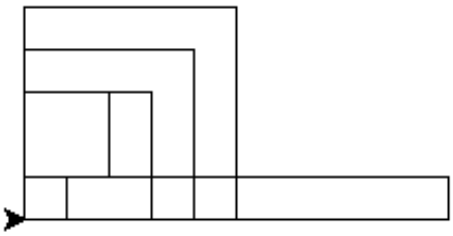
Beispiel 1



Beispiel 2



Beispiel 3



Quelltextauszüge

DrawRectangles.py

```
import turtle
import time

f = open ("coordinates.txt", "r")
coordinates = f.readlines()

number = ""
enclose = []
for a in coordinates[0]:
    if a != "," and a != "\n":
        number = number + a
    elif a == ",":
        enclose.append(int(number)*20)
        number = ""

turtle.forward(enclose[0])
turtle.left(90)
turtle.forward(enclose[1])
turtle.left(90)
turtle.forward(enclose[0])
turtle.left(90)
turtle.forward(enclose[1])
turtle.left(90)
enclose = []

for b in coordinates:
    for a in b:
        if a != "," and a != "\n":
            number = number + a
        elif a == ",":
            enclose.append(int(number)*20)
            number = ""
    if len(enclose) > 2:
        turtle.penup()
        turtle.setposition(enclose[0], enclose[1])
        turtle.pendown()
        turtle.forward(enclose[2])
        turtle.left(90)
        turtle.forward(enclose[3])
        turtle.left(90)
```

```
        turtle.forward(enclose[2])
        turtle.left(90)
        turtle.forward(enclose[3])
        turtle.left(90)
    enclose = []

time.sleep(10)
```

Rectangles.java

```
public class Rectangles {


    public Rectangle[] rectangles;
    protected int[][] coordinates;
    public int x = 0;
    public int y;
    public int area;
    public int area_min;
    public int x_temp;
    public int y_temp;

    public Rectangles(int x, Rectangle... rects){
        rectangles = rects;
    }

    public void sort(){
        Rectangle rectangle;
        for (int b = 0; b<rectangles.length-1; b++){
            for (int a = 0; a<rectangles.length-1; a++){
                if (rectangles[a].height > rectangles[a+1].height){
                    rectangle = rectangles[a];
                    rectangles[a] = rectangles[a+1];
                    rectangles[a+1] = rectangle;
                }
            }
        }
        y = rectangles[0].height;
        y_temp = y;
        for (int a = 0; a<rectangles.length-1; a++){
            x = x + rectangles[a].width;
            area_min = area_min + rectangles[a].area;
        }
        x_temp = x;
        area = x * y;

    }

    public void solve(){
        if (area_min == area){
            x = x_temp;
            y = y_temp;
        } else if (x_temp == 1) {
        }else {
            x_temp = x_temp-1;
            y_temp = area_min/x_temp;
            int z = 0;
            while (z== 0){
                if (x_temp*y_temp >= area){
                    solve();
                    z=1;
                } else {
```

```
                    if (x_temp * y_temp < area){
                        if (place(0)){
                            x= x_temp;
                            y= y_temp;
                            area = x*y;
                            coordinates = new int[rectangles.length][2];
                            int[] coordinate;
                            int t=0;
                            for (Rectangle rectangle: rectangles){
                                coordinate = new int[]{rectangle.x,
rectangle.y};
                                coordinates[t]= coordinate;
                                t++;
                            }
                        }

                    }

                }
                y_temp = y_temp +1;
            }
        }
    }

    public Boolean place(int a){
        int x_min = area;
        int y_min = area;
        int x_min_temp = 0;
        int y_min_temp = 0;
        int z = 0;
        if (a != 0){
            for (int b= 0; b<a; b++){
                y_min_temp = rectangles[b].y + rectangles[b].height;
                if (y_min_temp + rectangles[a].height > y_temp){
                    y_min_temp = 0;
                } else if (!check(rectangles[b].x, y_min_temp, b)){
                    y_min_temp = 0;
                } else {
                    if (rectangles[b].x < x_min) {
                        x_min = x_min_temp;
                        y_min = y_min_temp;
                        z=1;
                    } else if (rectangles[b].x == x_min){
                        if (y_min_temp < y_min){
                            x_min = x_min_temp;
                            y_min = y_min_temp;
                            z=1;
                        } else {
                            y_min_temp = 0;
                        }
                    } else {
                        y_min_temp = 0;
                    }
                }
            }
            if (z==1){
                rectangles[a].x = x_min;
                rectangles[a].y = y_min;
            }
            for (int b= 0; b<a; b++){
                x_min_temp = rectangles[b].x + rectangles[b].width;
                if (x_min_temp + rectangles[a].width > x_temp){
                    x_min_temp = 0;
                } else if (!check(x_min_temp, rectangles[b].y, b)){
```

```
                    x_min_temp = 0;
                } else {
                    if (rectangles[b].y < y_min) {
                        x_min = x_min_temp;
                        y_min = y_min_temp;
                        z=1;
                    } else if (rectangles[b].y == y_min){
                        if (x_min_temp < x_min){
                            x_min = x_min_temp;
                            y_min = y_min_temp;
                            z=1;
                        } else {
                            x_min_temp = 0;
                        }
                    } else {
                        x_min_temp = 0;
                    }
                }
            }
            if (z==1){
                if (x_min < rectangles[a].x){
                    rectangles[a].x = x_min;
                    rectangles[a].y = y_min;
                    return place(a+1);
                }  else if (x_min == rectangles[a].x){
                    if (y_min < rectangles[a].y){
                        rectangles[a].x = x_min;
                        rectangles[a].y = y_min;
                        return place(a+1);
                    } else {
                        return false;
                    }
                } else {
                    return false;
                }
            } else {
                return false;
            }
        } else {
            rectangles[a].x = 0;
            rectangles[a].y = 0;
            return place(a+1);
        }
    }

    public Boolean check (int x, int y, int b){
        for (int a = b; a<rectangles.length; a++){
            if (rectangles[a].x == x && rectangles[a].y == y){
                return false;
            }
        }
        return true;
    }
}
```