Junior 2: Baywatch

Symbroson
Team-ID: 00165

24. November 2018

Inhaltsverzeichnis

1 Lösungsidee

Ziel ist es, jedes Rechteck möglichst optimal in ein größeres einzuordnen. Da jedes Rechteck möglichst wenig Platz zu seinem Nachbarn haben soll, kann man damit anfangen, jedes Reckteck der Reihe nach neben das Vorherige zu platzieren.

Im zweiten Schritt limitiert man die Breite des Rechtecks um einen Garten und sortiert so lange Gärten ein, bis die Breite des Rechtecks von einem Garten überschritten wird. Dann muss dieser Garten auf der Y-Achse verschoben und von vorn alle X-Werte ausprobiert werden, ohne dass sich der Garten nicht mit anderen Gärten überschneidet.

Nachdem alle Gärten eingeordnet sind, wird die Fläche des entstandenen Rechteckes berechnet und ggf. das Minimum überschrieben. Das wird solange wiederholt, bis das Reckteck so breit wie der breiteste Garten ist, denn schmaler kann das Rechteck nicht sein.

Dies allein reicht aber micht aus, um die optimale Anordnung der Gärten zu finden. Schließlich kann man die Gärten in jeder beliebigen Reihenfolge einsortieren, was zwangsläufig zu anderen Ergebnissen führt. Deswegen müssen alle Permutationen der Gärten ausprobiert und wie beschrieben in ein Rechteck einsortiert werden. Am Ende kann dann das Rechteck mit der geringsten Fläche ausgegeben werden.

2 Umsetzung

Jeder Garten besitzt Imformationen über Höhe und Breite, aber auch die X,Y Position und eine ID für die Farbgebung später bei der Ausgabe. Deshalb erfolgt die Abstrahierung wieder als Klasse mit den genannten Eigenschaften.

Um während des gesamten Einsortierungssprozesses Zeit zu sparen, wird bereits während des Einlesens der Gartenlisten der ggT von Höhe und Breite einer Gartenliste berechnet, um dann alle Höhen und Breiten durch dieses teilen zu können.

Bei der Permutation erfolgt der Tausch und erneute Permutation der verbleibenden Elemente nur, falls die Größe der Gärten sich unterscheidet. Dadurch kann mitunter eine erhebliche Menge an zu testenden Permutationen eingespart werden, je mehr Gärten und je mehr gleiche es gibt.

Die Prüfung der Gartenpermutationen erfolgt wie bereits in der Lösungsidee beschrieben. Die Gärten werden nacheinander in das Rechteck eingefügt und solange in X und Y-Richtung bewegt, bis sie weder überstehen noch mit einem anderen Garten kollidieren. Die Breite des Rechtecks ist hierbei für jeden Durchlauf fest definiert, die Höhe entspricht der höchsten Y-Ausdehnung eines Gartens nach vollständiger Einsortierung.

Wurde ein geringerer Flächeninhalt als zuvor gefunden werden die Garteneigenschaften in der aktuellen Reihenfolge in eine weitere Liste kopiert, um diese später ggf. ausgeben zu können.

Die Ausgabe erfolgt in der Konsole mit per Escape-Sequenz gefärbten Feldern. (nähere Informationen im README.txt) Es muss beachtet werden, dass die Größe weiterhin durch das ggT geteilt ist, wodurch das Seitenverhältnis möglicherweise nicht mehr stimmt. Die reellen Maße sowie der Flächeninhalt stehen jedoch zum Vergleich über der Ausgabe.

3 Beispiele - schrebergaerten.txt

3.1 Liste 0.

```
-- 0: ---------------------
min: 45m x 40m = 1800m²
 2 2 2 2 2 2 1 1 1
 2 2 2 2 2 2 1 1 1
 2 2 2 2 2 2 1 1 1
 3 3 3 3 0 0 1 1 1
 3 3 3 3 0 0 1 1 1
 3 3 3 3 4 4 4 4 4
 3 3 3 3 4 4 4 4 4
 3 3 3 3 4 4 4 4 4
```

3.2 Liste 1.

```
-- 1: ---------------------
min: 55m x 35m = 1925m²
 1 1 1 1 1 2 2 2 2 2 2
 1 1 1 1 1 2 2 2 2 2 2
 1 1 1 1 1 2 2 2 2 2 2
 3 3 3 3 3 4 4 4 4 4 0
 3 3 3 3 3 4 4 4 4 4 0
 3 3 3 3 3 4 4 4 4 4 0
 3 3 3 3 3 0 0 0 0 0 0
```

3.3 Liste 2.

```
-- 2: ---------------------
min: 9m x 7m = 63m²
 1 1 1 1 1 1 3 3 3
 1 1 1 1 1 1 2 2 0
 1 1 1 1 1 1 2 2 0
 4 4 4 4 5 5 5 5 0
 4 4 4 4 5 5 5 5 0
 4 4 4 4 5 5 5 5 0
 4 4 4 4 5 5 5 5 0
```

3.4 Liste 3.

```
-- 3: --------------------
min: 9m x 6m = 54m^2
 3 3 3 3 3 3 2 2 2
 4 4 4 4 4 0 2 2 2
 4 4 4 4 4 1 1 1 1
 5 5 5 5 5 1 1 1 1
 5 5 5 5 5 1 1 1 1
 5 5 5 5 5 1 1 1 1
```

3.5 Liste 4 - Sonderfall

Es kann durchaus sein, dass alle Gärten die gleiche Höhe oder Breite haben. Dadurch ist die optimale Anordnung in W-O bzw N-S Richtung. Das ist auch der Grund, warum man bei der maximalen Breite des Rechtecks keine Abstriche machen darf.

```
-- 4: --------------------
min: 26m x 1m = 26m^2
 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5
```

4 Quellcode

```
/* ... */


struct Rect {
    uint x, y, w, h, i;

    Rect(uint X, uint Y, uint W, uint H, uint I):
        x(X), y(Y), w(W), h(H), i(I) {}

    void assign(Rect *v) {
        x = v->x; y = v->y; w = v->w; h = v->h; i = v->i;
    }
};


Rect *opt,   // enthält {ggT(w1, .., wn), ggT(h1, .., hn), minw, i, n} einer Gartenliste
    **order; // enthält Zeiger auf Gärten in optimaler Reihenfolge

// Rechteck:
uint minA, // Minimalfläche
    minw,  // Minimalbreite
    maxW,  // Breite
    maxH;  // Höhe

bool debug = false;

// das erste Element einer Gartenliste sind die 'opt' Parameter
list<list<Rect *>> gardenList;


/* ... */


// Ausgabe
void printGardens(Rect **list, uint w, uint h) {
    uint x, y, i, out[h][w];
    for (y = 0; y < h; y++)
        for (x = 0; x < w; x++) out[y][x] = 0;
```

```
40
    for (i = 0; i < opt->i; i++)
        for (y = 0; y < list[i]->h; y++)
            for (x = 0; x < list[i]->w; x++)
                out[list[i]->y + y][list[i]->x + x] = list[i]->i + 1;
45
    for (y = 0; y < h; y++) {
        for (x = 0; x < w; x++) {
            if (out[y][x]) {
                printf("\033[30;%um%2u", out[y][x] % 8 + 100, out[y][x]);
50          } else
                printf("\033[0;90m 0");
        }
        printf("\033[0;37m\n");
    }
55  printf("\n");
}

// Sortiert Gartenpermutation in Rechteck
void testGardens(Rect **gds) {
60  uint i, j,          // Zähler
        n = 1 + opt->i, // ersten n Rechtecke ergeben Maximalbreite
        maxw,           // Macimalbreite des Rechtecks
        maxy = 0;       // höchster Y-Wert

65  while (--n) {
        maxw = 0;

        //  maxw berechnen
        for (i = 0; i < n; i++) maxw += gds[i]->w;
70      if (maxw < minw) maxw = minw;

        // Positionen resetten
        for (i = 0; i < opt->i; i++) {
            gds[i]->x = 0;
75          gds[i]->y = 0;
        }

        // Alle Gärten einsortieren
        for (i = 0; i < opt->i; i++) {
80          bool coll;

            do {
                coll = false;

85              // Überschneidung mit anderem Garten?
                for (j = 0; j < i; j++) {
                    if ((gds[i]->x < gds[j]->x + gds[j]->w) &&
                        (gds[i]->y < gds[j]->y + gds[j]->h) &&
                        (gds[i]->y + gds[i]->h > gds[j]->y) &&
90                      (gds[i]->x + gds[i]->w > gds[j]->x)) {
                        // akt. Garten hinter gefundenen Garten bewegen
                        gds[i]->x = gds[j]->x + gds[j]->w;

                        // Rechteckbreite überschritten
95                      if (gds[i]->x + gds[i]->w > maxw) {
                            gds[i]->x = 0;
                            gds[i]->y++;
                        }
                        coll = true;
100                     break;
                    }
                }
            } while (coll);

105         if (gds[i]->y + gds[i]->h > maxy) maxy = gds[i]->y + gds[i]->h;
        }

        // neues minimum der Rechteckfläche gefunden
        if (!minA || (maxy * maxw != 0 && maxy * maxw < minA)) {
110         minA = maxy * maxw;
            maxW = maxw;
            maxH = maxy;
```

```
            // alle Garteneigenschaften für Ausgabe kopieren
            for (i = 0; i < opt->i; i++) order[i]->assign(gds[i]);
        }

        if (debug) printGardens(gds, maxw, maxy);
    }
}


// Permutation
void permut(Rect **r, uint end) {
    if (end == 0) {
        testGardens(r);
    } else {
        permut(r, end - 1);
        uint i;

        for (i = 0; i < end; i++) {
            // nicht tauschen und permutieren wenn Größe übereinstimmt
            if (r[i]->w != r[end]->w || r[i]->h != r[end]->h) {
                swap(r[i], r[end]);
                permut(r, end - 1);
                swap(r[i], r[end]);
            }
        }
    }
}


int main(int argc, const char *argv[]) {
    FILE *fp = NULL;
    uint i;

    // Argumente und Eingabedatei einlesen
    /* ... */

    // jede Gartenliste aus Eingabedatei behandeln
    for (auto &gardens: gardenList) {
        minw = i = 0;

        // lese Gartenlisteninformationen
        opt = gardens.back();
        gardens.pop_back();

        printf("\n-- %i: ----------------------\n", opt->h);

        // wende ggT an
        opt->w /= opt->x;

        // besetze Speicher
        Rect *garden[opt->i], // random-access Gartenliste
            *_order[opt->i];  // für Kopie der besten Reihenfolge
        order = _order;

        // resette minimale Fläche
        minA = 0;

        // Vorberechnungen
        for (Rect *&rect: gardens) {
            // Speichere Zeiger auf Gärten in Array
            garden[i++] = rect;

            // wende ggT an
            rect->w /= opt->x;
            rect->h /= opt->y;

            // berechne minimale Rechteck-Breite
            if (minw < rect->w) minw = rect->w;

            // erstelle Zeiger auf Garten in order-Liste
            *order++ = new Rect(0, 0, 0, 0, i);
```

```
        }

        // teste alle Garten-Permutationen
        order = _order;
        permut(garden, opt->i - 1);

        // Ausgabe
        printf(
            "min: %um x %um = %um²\n", maxW * opt->x, maxH * opt->y,
            minA * opt->x * opt->y);

        printGardens(order, maxW, maxH);

        for (i = 0; i < opt->i; i++) delete order[i];
    }

    freeSchrebergaerten();
    return 0;

error:
    fclose(fp);
    return 1;
}
```