# Junior 2: Baywatch

Symbroson
Team-ID: 00165

13. September 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. Man muss nun auch darauf achten, dass sich der Garten nicht mit anderen Gärten überschneidet.

Sind alle Gärten eingeordnet, wird jeweils die Fläche des entstandenen Rechteckes berechnet und ggf. das Minimum überschrieben.

Wiederholt wird das Ganze solange, 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.

Es werden also alle Permutationen der Gärten ausprobiert und wie beschrieben in ein Rechteck einsortiert. 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 Darstellung wieder als Klasse mit genannten Eigenschaften.

Um während des gesamten Einsortierungssprozesses Zeit zu sparen, wird bereits während dem Einlesen der Gartenlisten der ggT von Höhe und Breite einer Gartenliste berechnet (da das Assoziativgesetz beim ggT gilt, einfach umzusetzen.) um dann alle Höhen und Breiten durch diese 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 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 Höhe des Rechtecks entspricht der höchsten Y-Koordinate (+Höhe) eines Gartens, da ja die Breite für jeden Durchlauf fest definiert ist.

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

Falls der debug-Modus eingeschaltet ist, werden alle möglichen Gartenkonstellationen ausgegeben. Dabei muss beachtet werden, dass diese ebenfalls durch die ggT's geschrumpft wurden und die Seitenverhältnisse möglicherweise nicht mehr stimmen, falls sich die ggT's unterscheiden.

# 3 Beispiele - schrebergaerten.txt

## 3.1 Sonderfall 1 - Liste 4.

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 Quellcode

```
// Includes
/* ... */

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

    /* ... */
};

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

// 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;


uint ggt(uint x, uint y) {
    /* ...*/
}

void freeSchrebergaerten() {
    /* ... */
}

bool initSchrebergaerten(FILE *fp) {
    uint w, h, l, c, n, dx, dy, minw;
    char line[1024], *lp;
    bool read = false;

    // Erwartet Daten im Format "[n].\n[w1] x [h1], ..., [wn] x [hn]."
    while (fgets(line, 1024, fp) != NULL) {
        // lese Gartenliste wenn zuvor Listennummer gefunden ("[n].\n")
        if (read) {
            lp = line;
            n = minw = dx = dy = 0;
            gardenList.push_back({});

            // scannt Zeile nach [w] x [h] Paar
```

```
            while (*lp && (l = sscanf(lp, " %u x %u", &w, &h)) != (uint)EOF) {
                // Speichert Garten
                if (l == 2) {
                    gardenList.back().push_back(new Rect(0, 0, w, h, n++));
                    // berechnet ggT (Assoziativgesetz gilt)
                    dx = dx ? ggt(dx, w) : w;
                    dy = dy ? ggt(dy, h) : h;
                }

                // gehe zu nächstem Garten oder Zeilenende
                while (*lp)
                    if (*lp++ == ',') break;
            }

            // speichere zusätzliche Gartenlisteninformationen
            // (ggt(w), ggt(h), minw, maxh, n)
            gardenList.back().push_back(new Rect(dx, dy, minw, c, n));
            dx = dy = 0;
            read    = false;

            // prüfe Zeile auf "[n].\n" ([n] -> index)
        } else {
            c  = 0;
            lp = line;
            while (isdigit(*lp)) c = 10 * c + *lp++ - '0';
            read = lp[0] == '.' && lp[1] == '\n';
        }
    }

    return false;
}

// 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;

    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;

    for (y = 0; y < h; y++) {
        for (x = 0; x < w; x++) {
            if (out[y][x]) {
                printf("\033[30;%um%2u", out[y][x] % 9 + 99, out[y][x]);
            } else
                printf("\033[0;90m 0");
        }
        printf("\033[0;37m\n");
    }
    printf("\n");
}

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

    while (--n) {
        maxw = 0;

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

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

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

            do {
                coll = false;

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

            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)) {
            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);
    }
}


inline void swap(Rect **list, uint a, uint b) {
    /* ... */
}

// 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, end);
                permut(r, end - 1);
                swap(r, i, end);
            }
        }
    }
}


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

```

```
    // Argumente einlesen
    /* ... */

    // Default Datei öffnen falls nötig
    if (fp == NULL && tryOpen("res/schrebergaerten.txt", fp)) goto error;

    // Datei Einlesen & Parsen
    if (initSchrebergaerten(fp)) {
        error("initialization failed");
        goto error;
    }
    fclose(fp);

    // 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: %u x %u = %u\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;
}
```