Aufgabe 1: Flohmarkt in Langdorf

Teilnahme ID: 55627
Bearbeiter dieser Aufgabe: Elia Doumerc

April 2021

Inhaltsverzeichnis

---

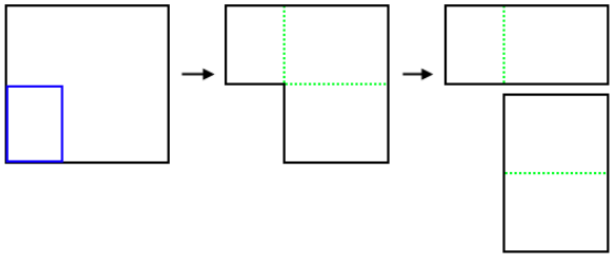
1 Lösungsidee

Die Anfragen lassen sich auch als Rechtecke der Länge $Standlänge(m)$ und Höhe $Mietende - Mietbeginn(h)$ betrachten, die in einen Rahmen mit der Länge $1000m$ und der Höhe $(18 - 8)h$ untergebracht werden sollen. Es handelt sich also um ein Rechteckverpackungsproblem mit den Besonderheiten, dass die Rechtecke gewählt werden müssen, nicht rotiert werden dürfen und ihre Position in der Y-Ebene (Mietbeginn) bereits bestimmt ist. Eine der vielen Herangehensweisen zu diesem Problem ist der Maximal Rectangles Algorithmus.

1.1 Maximal Rectangles Algorithmus

In diesem Algorithmus [1] wird der freie Raum mit einer Liste von maximalen freien Rechtecken F dargestellt. Das heißt, dass jedes Mal, wenn ein neues Rechteck R (Anfrage) in ein freies Rechteck F_i (ist am Anfang der Marktplatz), gesetzt wird, wird F_i an den im Bild gezeigten Achsen in neue Container geteilt. So hat man eine Datenstruktur, für die gilt: falls $R \subseteq \bigcup_{i=1}^{n} F_i$, dann gibt es ein Rechteck $F_i \in F$ mit der Eigenschaft $R \subseteq F_i$. Man ist sich also sicher, einen freien Platz für R zu finden, falls es noch möglich ist, es irgendwo einzusetzen. Da es bei weiterem Einsetzen von Rechtecken zu falschen freien Rechtecken in der Datenstruktur kommen kann, sind einige weitere Schritte nötig, um alle Rechtecke in F tatsächlich frei und maximal zu halten. Hier ist der gesamte Algorithmus zu sehen.

Abbildung 1: Maximal Rectangles-Schnitt



Algorithmus 1 : Maximal Rectangles

Data : $F = [(8, 18, 1000)]$

1 **for** $R(s, e, l)$ *in Anfragen* **do**
2 Wähle $F_i \in F$, in das R gesetzt werden soll
  `// Dazu gibt es meherere mögliche Heuristiken`
3 **if** R *passt in kein* F_i **then**
4 continue
5 Positioniere R in F_i.
6 $altMaxRects, neuMaxRects \leftarrow \{\}$
7 **for** $F_i in F$ **do**
8 **if** $(F_i \setminus R) > 0$ **then**
9 Baue die neuen maximalen Rechtecke.$N_1, ...N_4$
10 $altMaxRects \leftarrow altMaxRects \cup \{F_i\}$
11 $neuMaxRects \leftarrow neuMaxRects \cup \{N_1, ...N_4\}$
12 $F \leftarrow (F \setminus altMaxRects) \cup neuMaxRects$
13 **for** $F_i, F_j in F$ **do**
  `// Stellt sicher, dass alle freien Rechtecke maximal sind.`
14 **if** $F_i \subseteq F_j$ **then**
15 $F \leftarrow F \setminus \{F_i\}$
16

1.2 Heuristiken

In der Zeile 2 des Algorithmus wurden Heuristiken erwähnt. Sie sind nötig, da das Finden der perfekten Lösung bei großen Datensätzen wie denen, die vorliegen, rechnerisch zu aufwändig wäre. Sie entscheiden, welches Rechteck als nächstes wo eingesetzt werden muss nach einfachen Regeln,

[1] Ich habe mich für die Formulierung des Algorithmus auf der Arbeit „A Thousand Ways to Pack the Bin - A Practical Approach to Two-Dimensional Rectangle Bin Packing“ von Jukka Jylänki gestützt.

die (hoffentlich) zu einem besseren Ergebnis führen als beim noch einfacheren Einsetzen in das nächstbeste Feld.

Dafür gibt es mehrere Möglichkeiten. Eine denkbare Art, das nächste freie Rechteck zu wählen ist, das zu nehmen, wessen kürzere übriggebliebene Seite am kleinsten ist, wo also $min(w_{F_i} - w_R, h_{F_i} - w_R)$ am kleinsten ist. Dasselbe lässt sich für die längere übriggebliebene Seite formulieren.

Zudem führt es zu allgemein kompakteren Verpackungen, wenn jedes Rechteck (wenn möglich) in eine Ecke von F_i (zum Beispiel links unten) gesetzt wird. Man könnte auch immer das kleinste freie Rechteck wählen, das R enthalten kann.

1.3 Verbesserung durch Sortieren der Anfragen

Wie die Ergebnisse in Abschnitt 3 zeigen, beeinflusst das Sortieren der Anfragen in Abhängigkeit von der verwendeten Heuristik und des Datensatzes stark die Ergebnisse. Es ist in keinem Fall hilfreich, die Anfragen in aufsteigender Größe zu sortieren, da sie größere Blöcke, die mehr Geld einbringen, oft auf unnötige Weise blockieren. Andersherum gibt es manchmal Blöcke, die durch ihre Größe bessere Kombinationen verhindern und so das Hauptproblem des Algorithmus zeigen: Man kann Rechtecke nachdem sie platziert wurden nicht mehr zurücknehmen oder verschieben. Allgemein führte aber das Sortieren der Anfragen nach aufsteigender Startstunde, dann absteigender Endstunde und schließlich absteigender Standlänge zu den besten Ergebnissen.

1.4 Globale beste Wahl

Bei dieser Variante des Algorithmus wird von allen nicht-gesetzten Rechtecken das ausgesucht, das das beste Ergebnis nach den Regeln der verwendeten Heuristik verursacht. Man iteriert also jedes Mal durch alle Elemente von F und alle noch nicht positionierten Anfragen. Das führt zur idealen Zuweisung, aber auch zur Erhöhung der Laufzeit von $O(n^x)$ auf $O(n^{x+1})$.

2 Implementierung

Ich habe mich für eine Implementierung in der Programmiersprache Python entschieden.

Die Eingabe wird mit Hilfe der Bibliothek click verwaltet (Für Details zur Nutzung, siehe den Abschnitt 3). Leider sind einige der generierten Ausgaben auf englisch, wodurch die Textsprache nicht homogen ist. Mit der Funktion `csv.DictReader` wird die Information der Datei in eine Liste von *Dictionaries* eingelesen, die alle noch ein Feld „ID“ erhalten [2], das bei der Ausgabe nützlich ist.

Falls der Benutzer es will, werden die Anfragen der Python-Funktion `sorted(liste, schlüssel)` übergeben, die eine Implementation des Timsort-Algorithmus ist. Die Sortierkriterien (2. Argument der Funktion) werden mit der Funktion `order_sorting(anfrage)` bestimmt.

Der weitere Verlauf des Programms hängt stark von den übergebenen Optionen an. Ohne der globalen Variante (ab jetzt `--global` genannt) wird für jede Anfrage die Funktion `maxrects()`

[2] Die IDs werden nach Reihenfolge in der Eingabedatei vergeben

ausgeführt. Mit `--global` wird `maxrects` so lange ausgeführt, bis ein Rechteck nicht eingesetzt werden kann.

Ohne `--global` führt `maxrects` zunächst nur `assign_free_rect()` aus, die die freien Rechtecke heraussucht, in die die Anfrage passt. Aus der entstehenden Liste freier Rechtecke werden dann mithilfe der Heuristik-Funktionen `bl`, `bssf` und `blsf` die besten freien Rechtecke gewählt. Dieses Auswahlverfahren läuft so lange, bis ein eindeutig bestes Rechteck gefunden wurde oder die Liste mit den vom Nutzer angegebenen Heuristiken vollständig durchgegangen wurde. Mit `--global` wird der oben beschriebene Prozess für jede noch ungelegte Anfrage durchgeführt. Dazu muss unter allen von der Funktion `assign_free_rect` zurückgegebenen Werten ebenfalls ein bester Wert gewählt werden.

Der weitere Verlauf von `maxrects` ist mit und ohne `--global` gleich: Das Positionsattribut der Anfrage wird auf den Positionswert des gewählten freien Rechteck gesetzt, man setzt die Anfrage also an dessen linke Seite. In Folge werden die neuen maximalen Rechtecke gebaut in `build_max_rects`. Dann werden die in Algorithmus 1 beschriebenen Schritte durch einfaches Vergleichen von Attributen durchgeführt: man stellt sicher, dass keines der anderen freien Rechtecke mit der frisch eingesetzten Anfrage überlappt und überprüft, dass kein freies Rechteck Teilmenge eines anderen maximalen Rechtecks (also nicht-maximal) ist.

Die Liste mit den angenommen Anfragen wird entweder im Terminal oder in einer Datei ausgegeben. Ich habe zudem mithilfe der Grafikbibliothek Pillow eine Option zur visuellen Ausgabe als Png-Datei der Anfragen implementiert.

2.1 Laufzeitbestimmung

Die Laufzeit beträgt für das Sortieren der Eingaben mit dem eingebauten Timsort-Algorithmus $O(n * log(n))$.

Der Maximal-Rectangles-Algorithmus wird n Mal ausgeführt und muss jedes Mal das nächste freie Rechteck finden, was F Schritte in Anspruch nimmt. Dann baut es die neuen maximalen Rechtecke in F(Zahl der freien Rechtecke) Schritten und eliminiert die nicht-maximalen Rechtecke in F^2 Schritten, was $O(F^4 * n)$ ergibt. Allerdings zeigen Messungen, dass F deutlich kleiner als n ausfällt, was mit der relativ geringen Höhe (Dauer des Marktes) des Rechtecks zusammenhängt. So wird F mit der Eingabedatei *flohmarkt6.txt* ($n > 700$) zum Beispiel nicht länger als 50.

Mit `--global` braucht das Programm $O(F^5 * n)$ Schritte.

3 Beispiele

Ich empfehle der Sicherheit halber die Verwendung einer virtuellen Umgebung. Benötigt werden die Pakete *Click* und *Pillow*. Die Programmoptionen lassen sich mit dem Befehl `--help` anzeigen.

Die Ausgabe ist in Form eines Python-Dictionaries und sieht bei der Datei `flohmarkt6.txt` folgenderweise aus:

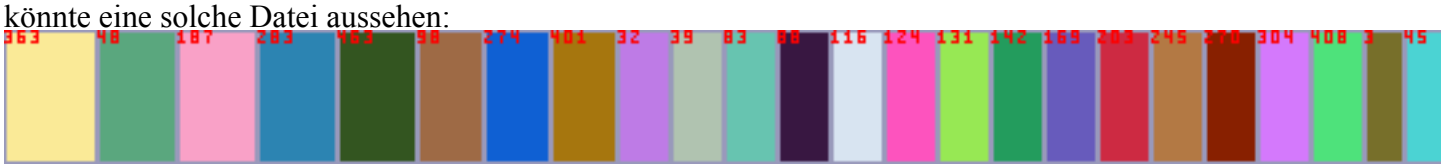
```
(venv) elia@elia-ubuntu:~/Desktop/bwinf392/Aufgabe1$ python3 Aufgabe1.py Beispieleingaben/flohmarkt6.txt -s s-e-l
Erträge: 100.0%
Angenommene Anfragen: 9 / 9
Maximale Zahl freier Rechtecke: 4
{'s': 8, 'e': 15, 'l': 249, 'id': 1, 'pos': 0}
{'s': 8, 'e': 12, 'l': 171, 'id': 8, 'pos': 249}
{'s': 8, 'e': 9, 'l': 503, 'id': 7, 'pos': 420}
{'s': 8, 'e': 9, 'l': 77, 'id': 6, 'pos': 923}
{'s': 9, 'e': 15, 'l': 477, 'id': 2, 'pos': 420}
{'s': 9, 'e': 15, 'l': 103, 'id': 9, 'pos': 897}
{'s': 12, 'e': 15, 'l': 171, 'id': 5, 'pos': 249}
{'s': 15, 'e': 18, 'l': 526, 'id': 3, 'pos': 0}
{'s': 15, 'e': 18, 'l': 474, 'id': 4, 'pos': 526}
```

Die erste Zeile zeigt den Anteil bedeckter Fläche, die zweite die Zahl der angenommen Anfragen und die dritte die maximale Zahl freier Rechtecke während des gesamten Ablaufs des Programms. Die Anfragen sind Dictionaries und die Namensgebung der Schlüssel ist wie folgt: 's', Startzeit; 'e', Endzeit; 'l', Länge; 'id', ID; 'pos', Position.

Während der Eingabe kann man mit der Option `-s` die Kriterien für die Anfragensortierung angeben. Mit dem Wert `s-e-l` legt man beispielsweise fest, dass die Anfragen nach aufsteigender Startzeit, dann nach absteigender Endzeit und schließlich nach absteigender Länge sortiert werden.

Des Weiteren ermöglicht die wiederholte Verwendung der Option `-h` die Festlegung der Reihenfolge für die Heuristiken. `-h bssf -h blsf` legt zum Beispiel fest, dass zuerst die Best-short-side-fit-Heuristik und dann die Best-long-side-fit-Heuristik verwendet wird.

Die Option `-i dateiname.png` fügt die Ausgabe als Bilddatei dem Programmablauf hinzu. So könnte eine solche Datei aussehen:



Freie Rechtecke werden hier gestrichelt angezeigt und die Zahlen sind die IDs der Anfragen.

Kommen wir nun zu den eigentlichen Ergebnissen. Hier ist nur die Kurzform, die Ausgabe der Anfragen wurde in Dateien verschoben. Das Skript ist aber neben dem Quelltext enthalten. Damit es ohne Anpassung funktioniert, müssen die Beispieleingaben im Ordner „Beispieleingaben“ sein und der Ordner „Ausgabe“ muss existieren.

```
(venv) elia@elia-ubuntu:~/Desktop/bwinf392/Aufgabe1$ python3 test.py
python3 Aufgabe1.py Beispieleingaben/flohmarkt1.txt -o Ausgabe/out1.txt -s s-e-l -h bssf
Erträge: 80.28%
Angenommene Anfragen: 490 / 490
Maximale Zahl freier Rechtecke: 41
python3 Aufgabe1.py Beispieleingaben/flohmarkt2.txt -o Ausgabe/out2.txt -s s-e-l -h bssf
Erträge: 90.56%
Angenommene Anfragen: 495 / 603
Maximale Zahl freier Rechtecke: 32
python3 Aufgabe1.py Beispieleingaben/flohmarkt3.txt -o Ausgabe/out3.txt -s s-e-l -h bssf
Erträge: 87.78%
Angenommene Anfragen: 550 / 735
Maximale Zahl freier Rechtecke: 51
python3 Aufgabe1.py Beispieleingaben/flohmarkt4.txt -o Ausgabe/out4.txt -s s-e-l -h bssf
Erträge: 73.7%
Angenommene Anfragen: 5 / 7
Maximale Zahl freier Rechtecke: 4
python3 Aufgabe1.py Beispieleingaben/flohmarkt5.txt -o Ausgabe/out5.txt -s s-e-l -h bssf
Erträge: 79.62%
Angenommene Anfragen: 6 / 25
Maximale Zahl freier Rechtecke: 6
python3 Aufgabe1.py Beispieleingaben/flohmarkt6.txt -o Ausgabe/out6.txt -s s-e-l -h bssf
Erträge: 100.0%
Angenommene Anfragen: 9 / 9
Maximale Zahl freier Rechtecke: 4
python3 Aufgabe1.py Beispieleingaben/flohmarkt7.txt -o Ausgabe/out7.txt -s s-e-l -h bssf
Erträge: 99.35000000000001%
Angenommene Anfragen: 562 / 566
Maximale Zahl freier Rechtecke: 30
```

Dies sind die allgemein besten Ergebnisse, sie wurden mit der Heuristik Best-short-side-fit erzielt. Merkwürdigerweise verbesserte die Option `--global` die Ergebnisse nicht:

```
python3 Aufgabe1.py Beispieleingaben/flohmarkt1.txt -o Ausgabe/out1.txt -g -h bssf
Erträge: 80.27%
Angenommene Anfragen: 489 / 490
Maximale Zahl freier Rechtecke: 0
python3 Aufgabe1.py Beispieleingaben/flohmarkt2.txt -o Ausgabe/out2.txt -g -h bssf
Erträge: 90.47%
Angenommene Anfragen: 522 / 603
Maximale Zahl freier Rechtecke: 0
python3 Aufgabe1.py Beispieleingaben/flohmarkt3.txt -o Ausgabe/out3.txt -g -h bssf
Erträge: 87.58%
Angenommene Anfragen: 574 / 735
Maximale Zahl freier Rechtecke: 0
python3 Aufgabe1.py Beispieleingaben/flohmarkt4.txt -o Ausgabe/out4.txt -g -h bssf
Erträge: 66.86%
Angenommene Anfragen: 4 / 7
Maximale Zahl freier Rechtecke: 0
python3 Aufgabe1.py Beispieleingaben/flohmarkt5.txt -o Ausgabe/out5.txt -g -h bssf
Erträge: 81.19%
Angenommene Anfragen: 6 / 25
Maximale Zahl freier Rechtecke: 0
python3 Aufgabe1.py Beispieleingaben/flohmarkt6.txt -o Ausgabe/out6.txt -g -h bssf
Erträge: 93.82000000000001%
Angenommene Anfragen: 8 / 9
Maximale Zahl freier Rechtecke: 0
python3 Aufgabe1.py Beispieleingaben/flohmarkt7.txt -o Ausgabe/out7.txt -g -h bssf
Erträge: 98.42%
Angenommene Anfragen: 555 / 566
Maximale Zahl freier Rechtecke: 0
```

4 Quelltext

Hier ist der wichtigste Teil des Quelltextes, der mit dem Maximal-Rectangles-Algorithmus zu tun hat.

Standard

```
€
def maxrects(heuristics, anfrage=None):
    """
    Implementierung des Maximal Rectangles-Algorithmus, nach
    Jukka Jylänki (http://pds25.egloos.com/pds/201504/21/98/RectangleBinPack.pdf
    )
    """
    global free_rects, anfragen, angenommen

    # Falls keine Anfrage übergeben wurde, führe die globale Option aus.
    if anfrage is None:

        min_score = 1000
        options = []

        for a in anfragen:
            min_score, f_r = assign_free_rect(heuristics, a)
            if f_r is None:
                continue
            options.append(f_r)  # erhält (anfrage, free_rect)

        i = 0  # Current heuristic iterator position
        while len(options) > 1 and i != len(heuristics):
            min_score, options = untier(options, heuristics[i])
            i += 1

        if min_score == 1000:
            return False  # Es konnte keine weiteres Rechteck eingesetzt werden.
        anfrage = options[0][0]
        fi = options[0][1]

    else:
        # wähle das nächste freie Rechteck, in das anfrage gesetzt werden soll.
        x, fi = assign_free_rect(heuristics, anfrage)
        if not fi:
            return  # Falls das Rechteck nicht passt, nächstes Rechteck
        fi = fi[1]

    # Setze die Anfrage in das freie Rechteck links ein
    anfrage['pos'] = fi['pos']

    anfragen.remove(anfrage)  # Entferne die angenommene (beste) Anfrage.
    angenommen.append(anfrage)
```

```
    # Erstelle die neuen maximalen Rechtecke
    temp_corrupted_rects = []
    temp_free_rects = []

    # Gucke, ob anfrage auch in einem der anderen free_rects (teilweise) enthalten ist und bilde ggf. neue maximale Rechtecke
    for f in free_rects:
        if anfrage['s'] >= f['e'] or anfrage['e'] <= f['s'] or (anfrage['pos'] + anfrage['l']) < f['pos'] or anfrage['pos'] > (f['pos'] + f['l']):
            continue
        else:
            temp_corrupted_rects.append(f)
            temp_free_rects += build_max_rects(anfrage, f)

    # Lösche alle alten maximalen Rechtecke
    for f in temp_corrupted_rects:
        free_rects.remove(f)
    free_rects += temp_free_rects  # Füge die neuen Rechtecke hinzu.

    # Prüft, ob alle Rechtecke tatsächlich maximale Rechtecke sind, indem für jedes Rechteck geguckt wird, ob dessen Fläche
    # eine Teilmenge der Fläche des anderen Rechtecks ist.
    temp = []
    for f1 in free_rects:
        for f2 in free_rects:
            if f1 == f2: continue
            if f1['s'] < f2['s'] or f1['e'] > f2['e'] or (f1['pos'] + f1['l']) > (f2['pos'] + f2['l']) or f1['pos'] < f2['pos']:
                continue
            else:
                temp.append(f1)
    for i in temp:  # TODO use sets for better performance.
        try:
            free_rects.remove(i)
        except:
            pass

    return True


# Tied: [(anfrage, f)]
def untier(tied, heuristic):
    """Findet die eindeutig beste Anfrage --> Rechteck Zuordnung"""
    min_val = 1000
    min_rects = []
    for paar in tied:
        tmp_val = eval(heuristic)(paar[0], paar[1])
        if tmp_val == min_val:
            min_rects.append(paar)
```

```
        elif tmp_val < min_val:
            min_rects = [paar]
            min_val = tmp_val

    return (min_val, min_rects)


def assign_free_rect(heuristics, anfrage):
    """
    Findet das beste freie Rechteck im Sinne der übergebenen Heuristik.
    i = current heuristic iterator position
    """
    global free_rects
    tmp_best = (1000, [])

    # Filtert die Rechtecke, in die die Anfrage passt.
    for f in free_rects:
        if anfrage['e'] <= f['e'] and anfrage['l'] <= f['l'] and anfrage['s'] >= f['s']:
            tmp_best[1].append((anfrage, f))

    # Solange es mehrere beste Paare gibt und es heuristische Regeln zum probieren gibt.
    i = 0  # Current heuristic iterator position
    while len(tmp_best[1]) > 1 and i != len(heuristics):
        tmp_best = untier(tmp_best[1], heuristics[i])
        i += 1

    if len(tmp_best[1]) > 0:
        return tmp_best[0], tmp_best[1][0]  # Erste Anfrage in tmp_best
    else:
        return tmp_best[0], None


def bssf(anfrage, f):
    """
    Gibt die Länge der kürzeren übriggeblieben Seite zurück.
    """
    f_hoehe = (f['e'] - f['s']) - (anfrage['e'] - anfrage['s'])
    f_breite = f['l'] - anfrage['l']
    return min(f_breite, f_hoehe)


def blsf(anfrage, f):
    """
    Gibt die Länge der kürzeren übriggeblieben Seite zurück.
    """
    f_hoehe = (f['e'] - f['s']) - (anfrage['e'] - anfrage['s'])
    f_breite = f['l'] - anfrage['l']
```

```
136     return max(f_breite, f_hoehe)


138
def bl(anfrage, f):
140     """
        Gibt die Differenz zwischen Start von Anfrage und Start von f zurück (0,
        falls sich Anfrage in bl von f platzieren lässt)
142     """
        return anfrage['s'] - f['s']
144


146 def build_max_rects(anfrage, fi):
        """
148     Baut die maximalen Rechtecke um die Anfrage herum und speichert sie in
        free_rects.
        Es wird angenommen, dass sich das neue Rechteck am linken Rand on fi
        befindet.
150     """
        global free_rects
152     temp_free_rects = []
        if anfrage['s'] > fi['s']:  # freies Rechteck unter anfrage
154         temp_free_rects.append({'s': fi['s'], 'e': anfrage['s'], 'l': fi['l'], '
        pos': fi['pos']})

156     if anfrage['pos'] > fi['pos']:  # freies Rechteck links von Anfrage
            temp_free_rects.append({'s': fi['s'], 'e': fi['e'], 'l': anfrage['pos']
        - fi['pos'], 'pos': fi['pos']})
158
        if (anfrage['pos'] + anfrage['l']) < (fi['pos'] + fi['l']):  # freies
        Rechteck rechts von anfrage
160         temp_free_rects.append({'s': fi['s'], 'e': fi['e'], 'l': (fi['pos'] + fi
        ['l']) - (anfrage['pos'] + anfrage['l']), 'pos': anfrage['pos'] + anfrage['l'
        ]})

162     if anfrage['e'] < fi['e']:  # freies Rechteck über anfrage
            temp_free_rects.append({'s': anfrage['e'], 'e': fi['e'], 'l': fi['l'], '
        pos': fi['pos']})
164
        return temp_free_rects
```