Abteilung Algorithmik
Institut für Betriebssysteme und Rechnerverbund **SoSe 08**
TU Braunschweig

Prof. Dr. Sándor Fekete
Dr. Tom Kamphans
Nils Schweer
Christopher Tessars

Praktikum-Algorithmik
Übung 1 vom 15.04.2008

Problemstellung

Das *Strip-Packing-Problem* ist eines der klassischen Packungsprobleme und kann wie folgt definiert werden:

Gegeben: Ein Streifen S der Breite 1 und unendlicher Höhe, sowie eine Menge von Recktecken $A_1, A_2, \ldots, A_n$. Rechteck A_i hat Breite w_i und Höhe h_i.

Gesucht: Eine Packung der Rechtecke in S, so dass die maximale Höhe möglichst klein ist.

Dabei ist zu beachten, dass die Rechtecke so gepackt werden müssen, dass ihre Kanten parallel zu den Seiten des Streifens sind und die Recktecke sich nicht gegenseitig überlappen. Ansonsten darf die Position eines Reckteckes innerhalb von S beliebig gewählt werden, jedoch nicht verändert werden, nachdem das Rechteck einmal platziert wurde. Es gibt verschiedene Varianten dieser allgemeinen Version des Problems. Die wichtigsten werden nun kurz vorgestellt:

(A) Ist die Menge der Rechtecke, die in S gepackt werden sollen bekannt, handelt es sich um die *offline* Version des Problems. Ist zu jedem Zeitpunkt nur das nächste zu platzierende Rechteck bekannt, d.h. zu Beginn ist nur A_1 bekannt, wenn dieses platziert ist, wird A_2 bekannt usw. spricht man von einem *online* Problem.

(B) Rotation der Rechtecke um $90\,^\circ$ ist erlaubt.

(C) Nach dem Platzieren müssen die Rechtecke nach unten durch ein anderes Rechteck oder den Boden von S begrenzt werden (Gravitation).

(D) Eine Bedingung, die nicht nur an das Spiel *Tetris* angelehnt ist, sondern auch durchaus praktische Bedeutung hat, ist die folgende: Das nächste zu platzierende Rechtecke A_i erscheint oberhalb aller bisher platzierten Rechtecke und muss nun auf einem Weg zu seiner endgültigen Position geführt werden. Dieser Weg muss mindestens die Breite w_i (bzw. h_i falls Rotation erlaubt ist) haben.

Ziel diese Praktikums ist es, ein Java-Applet zu programmieren, das es erlaubt, verschiedene Algorithmen für dieses Problem auszuwählen oder die Rechtecke selber zu steuern, sowie mit zufälligem Input zu arbeiten oder die Größen der Rechtecke vorzugeben. Außerdem soll es möglich sein, eine der verschiedenen Versionen des Problems auszuwählen.

Spezifikationen und Anforderungen

Die Algorithmen, die implementiert werden sollen, werden im Laufe des Semester in den jeweiligen Aufgaben kurz vorgestellt. Wichtig ist jedoch, dass zu Beginn einige grundlegende Dinge berücksichtig werden. Diese werden im Folgenden vorgestellt:

Wir betrachten ab jetzt nur noch den folgenden Spezialfall des Strip-Packing-Problem:

Gegeben: Ein Streifen S der Breite W und unendlicher Höhe, sowie eine Menge von Recktecken $A_1, A_2, \ldots$ wobei Rechteck A_i Breite w_i und Höhe h_i hat. Die Rechtecke erscheinen nacheinander (online).

Gesucht: Eine Packung der Rechtecke in S, so dass die maximale Höhe möglichst klein ist.

Das nächste Rechteck erscheint immer oberhalb aller bisher gepackten Rechtecke. Eine Rotation um $90\,^\circ$ darf nur zu Beginn der Platzierung erfolgen. Ob die Bedingungen (C) und (D) beim Packen erfüllt sein müssen, hängt nur von der gewählten Strategie/Gegenstrategie ab.

Darüber hinaus wollen wir das Problem diskretisieren, d.h. es wird angenommen, dass dem Streifen ein Gitter zugrunde liegt. Die Rechtecke können jeweils um ein Gitterfeld in die gewünschte Richtung bewegt werden. Die Breite W sowie die w_i und h_i beziehen sich dann auf die Anzahl der Gitterfelder.

Die Pack-Strategie soll intern durch geeignete Datenstrukturen, Klassen, Funktionen etc. abgebildet werden; ausserdem soll eine grafische Benutzeroberfläche erstellt werden, die folgende Anforderungen erfüllt:

- Es soll der Streifen S dargestellt werden.
- Das Gitter soll ebenfalls dargestellt werden.
- Die Breite W des Streifens S und die Breite der Gitterzellen soll durch den Benutzer zu Begin auswählbar sein.
- Platzierte Rechtecke sollen unterschiedlich gefärbt sein.
- Es soll Buttons für
  - die Auswahl der Strategie (s.u.)
  - die Auswahl der Gegenstrategie (s.u.)
  - Start und Stop, Step (ein Schritt), Restart
  - die Ausgabe der Oberfläche in eine Postscript-Datei (dafür stellt Java Funktionen bereit)

  geben.
- Es soll einstellbar sein, in welchen zeitlichen Abständen die neuen Rechtecke erscheinen und wie langsam / schnell das Bewegungen der Rechtecke ausgeführt wird.

- Es soll die Möglichkeit bestehen auszuwählen, dass nur die endgültige Position des Rechteckes angezeigt wird (ohne den Weg dorthin).
- Die aktuelle Höhe M soll angezeigt werden.
- Wird Rechteck A_k platziert, soll $V = \frac{1}{W}\sum_{i=1}^{k} h_i \cdot w_i$ ausgegeben werden. Außerdem soll $\frac{M}{V}$ ausgegeben werden.

Weiterhin soll es später komfortabel möglich sein, das Applet um neue Strategien (und Gegenstrategien) zu erweitern. Dies soll bereits im Entwurf durch geeignete objektorientierte Massnahmen berücksichtigt werden.

Nun betrachten wird ein Beispiel für eine Stratgie / einen Algorithmus der in diesem Semster implementiert werden soll. Wie oben schon erwähnt werden im Laufe des Semester immer wieder neue hinzukommen:

```
1 for i = 1, 2, ... do
2 |   Wähle die beste zulässige Position für A_i (falls es mehrere gibt wähle die, die am
  |   weitesten links ist)
3 end
```

Dabei ist mit "beste Position" eine Position gemeint, die den Abstand zur Grundfläche von S minimiert. An dieser Strategie kann man sich orientieren, wenn (wie in Aufgabe 1 gefordert) eine geeignete Datenstruktur für die Umsetzung gewählt werden soll.
Eine weitere Möglichkeit ist die Steuerung der Rechtecke per Hand, d.h. mit Hilfe von Pfeiltasten sollen die Rechtecke in die gewünschte Richtung bewegt werden können. Auch diese Möglichkeit soll durch den Benutzer wählbar sein.

Unter einer Gegenstrategie versteht man vereinfacht gesagt die Erzeugung von Input, die der Strategie das "Leben" unterschiedlich schwer machen kann. Wir wollen folgende drei Möglichkeiten zur Erzeugung des Input vorsehen:

- Der Input wird zufällig generiert.
- Das nächste Rechtecke kann durch den Benutzer spezifiziert werden.
- Der Input wir aus einer Datei mit dem Format

  Breite von S

  Gitterbreite

  w_1 h_1

  w_2 h_2

  ...

  w_n h_n

  eingelesen.

Um die Ein- und Ausgaben nachvollziehen zu können, sollen in einer Log-Datei die wesentlichen Schritte gespeichert werden. Die Datei soll so gestaltet werden, dass sie als Input-Datei auch wieder eingelesen werden kann. Der Name der Log-Datei soll optional über die grafische Oberfläche wählbar sein.

Dokumentation

Begleitend zur Implementierungsphase soll bereits die Dokumentation des Projektes erfolgen. Hierzu bietet sich das Werkzeug *Javadoc* an, dass aus Kommentaren in den Quelltexten, die in einem bestimmten Format angelegt sind, eine Dokumentation der einzelnen Klassen und Methoden erstellt. Das Javadoc-Tool selbst generiert die Dokumentation in HTML, andere Tools (z. B. Doxygen) benutzen das gleiche Format um andere Ausgabeformate zu erstellen.
Beispiel (Klasse):

```
/**
 * This class implements a smart strategy for packing squares
 *
 * @version 0.1 Feb. 31, 2008
 * @author Homer Simpson, Ned Flanders
 */
```

Beispiel (Methode):

```
/**
 * Does somethings with the input parameters and calculates some output
 *
 * @param input1: some input
 * @param input2: another input
 * @return some fancy result
 */

 int CleverMethod( int input1, float input2 )
```

Eine Dokumentation zu Javadoc findet sich z. B. unter
`http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/javadoc.html`.

Aufgabe 1 (Entwurfsphase):
Im ersten Schritt soll ein Entwurf für das Applet erstellt werden.

- In welche Klassen läßt sich das Problem zerlegen?
- Welche Schnittstellen werden zwischen den Klassen benötigt?
- Wie können die Benutzerinteraktion und die graphische Benutzeroberfläche aussehen?
- Welche Datenstruktur(en) bieten sich zur Speicherung der aktuellen Konfiguration an?

Euer Entwurf soll am **Fr., 25.4.2008, 13.00** in Raum 262A vorgestellt und diskutiert werden.