ENTWICKLUNG EINES SOFTWARESYSTEMS

Mathematisch-technischer-Softwareentwickler

Abschlussarbeit

Optimierungsproblem

Lars Körte
Prüflingsnummer 05480

Inhalt

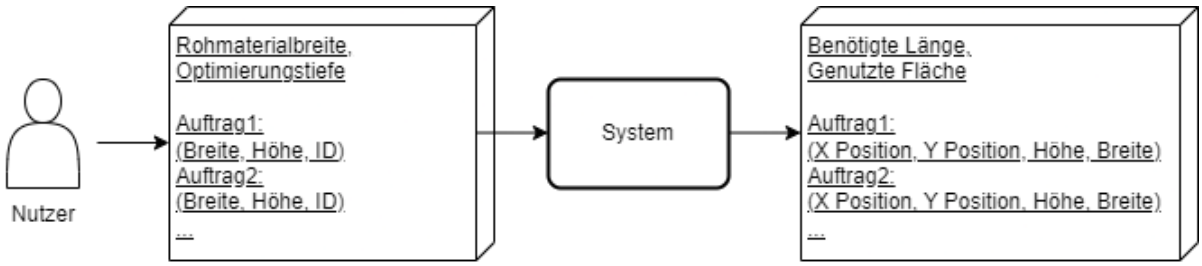
Problembeschreibung

Für die optimale Platzierung von diversen Aufträgen von großflächigen, rechteckigen Leiterplatinen auf einer Rolle aus Rohmaterial soll ein Softwaresystem erstellt werden.

Hierfür werden Andockpunkte, jeweils an der oberen linken und der unteren rechten Ecke einer platzierten Platine gebildet, an die weitere Platinen angedockt werden, wodurch der jeweilige verwendete Andockpunkt zu einem Ankerpunkt wird und nicht mehr zur Verfügung steht für weitere Platinen. Der erste Andockpunkt wird auf dem Rohmaterial an der Position (0,0) gesetzt. Eine Platine kann für die Optimierung sowohl horizontal als auch vertikal auf das Rohmaterial platziert werden.

Da sich hierbei mit größer werdenden Listen an Aufträgen schnell eine sehr große Anzahl an Kombinationsmöglichkeiten der jeweiligen Aufträge ergibt, wird bei der Initialisierung der Aufträge eine Optimierungstiefe mitgeliefert, welche die Liste an Aufträgen in kleinere Listen unterteilt. Die Ergebnisse der jeweiligen Teillisten werden zusammengezählt und ergeben ein Gesamtergebnis.

Eine Anordnung gilt dann als optimiert, wenn die verwendete Länge des Rohmaterials minimal, und somit die verwendete Fläche maximal, ist.



Grundanforderung

Eingabedaten

Als Eingabe werden Textdaten des Typs <Beispielname>.in verwendet.

Diese werden mit dem nachfolgenden Schema aufgebaut:

```
<Titelzeile>
<Materialbreite> <Optimierungstiefe>
<Breite> <Höhe> <ID> <Auftragsname>
<Breite> <Höhe> <ID> <Auftragsname>
<Breite> <Höhe> <ID> <Auftragsname>
...
```

Somit erhält die Textdatei einen Titel der Aufträge, Metadaten über die Rohmaterialrolle und die gewünschte Optimierungstiefe gefolgt von einer Aufzählung der Aufträge beinhalten der Breite, Höhe, einer Eindeutigen ID zur Identifizierung und einem Auftragsnamen.

Als Beispiel eine Liste aus der Aufgabenstellung:

```
// Auftrag IHK1
900 6
100, 200, 1, Auftrag A
297, 420, 2, Auftrag B
```

Ausgabedaten

Die Ausgabe soll sowohl als <Beispiel>.out Textdatei, als auch als <Beispiel>.gnu Skript erfolgen, sodass die angegeben Anordnung als Bild dargestellt werden kann.

Die Textdatei wird nach dem nachfolgenden Schema erstellt:

```
<Titelzeile>
Benötigte Länge: <Länge in CM>
Benötigte Fläche: <Prozent>

Positionierung der Kundenaufträge:
        <X><Y><Breite> <Höhe> - <ID> - <Auftragsname>
        <X><Y><Breite> <Höhe> - <ID> - <Auftragsname>
        <X><Y><Breite> <Höhe> - <ID> - <Auftragsname>
        ...

Verbleibende Andockpunkte:
        <X><Y>
        <X><Y>
        <X><Y>
        ...
```

Zusätzlich zu Gesamtlänge und prozentuale Auslastung der Fläche enthält sie also neben den Aufträgen ihre jeweilige Position des Ankerpunktes auf der Materialrolle auch die verbleibenden, noch ungenutzten Andockpunkte.

Als Beispiel die Ausgabe für Aufgabenbeispiel 1:

```
// Auftrag IHK1
Benötigte Länge: 29.7 cm
Benutzte Flaeche: 54.15 %

Positionierung der Kundenaufträge:
    0 0 100 200 - 1 - Auftrag A
    100 0 420 297 - 2 - Auftrag B

Verbleibende Andockpunkte:
    0 200
    520 0
    100 297
```

Die Skriptausgabe erfolgt nach einem ähnlichen Schema mit einigen für Gnuplot relevanten Zeilen für die gewünschte Darstellung:

```
reset
set term png size <rollenbreite>,<ymax+100>
set output '<Beispiel>.png'
set xrange [0:<rollenbreite>]
set yrange [0:<ymax>]
set size ratio -1

set title "\
<titel>\n\
Benötigte Länge: <ymax>cm\n\
Genutzte Fläche: <anteil>%"

set style fill transperent solid 0.5 border
set key noautotitle

$data <<EOD
# x_LU y_LU x_RO y_RO Auftragsbeschreibung ID
<x_LU> <y_LU> <x_RO> <y_RO> <Auftragsbeschreibung> <ID>
<x_LU> <y_LU> <x_RO> <y_RO> <Auftragsbeschreibung> <ID>
...
EOD

$anchor <<EOD
# x y
<X> <Y>
<X> <Y>
<X> <Y>
...
EOD

plot \
'$data' using (($3-$1)/2+$1):(($4-$2)/2+$2):(($3-$1)/2):(($4-$2)/2):6 \
             with boxxy linecolor var, \
'$data' using (($3-$1)/2+$1):(($4-$2)/2+$2):5 \
             with labels font "arial,9", \
'$anchor' using 1:2 with circles lc rgb "red", \
'$data' using 1:2 with points lw 8 lc rgb "dark-green"
```

<ymax> beschreibt die maximale Länge der Rolle, <x_LU> und <y_LU> beschreiben den Ankerpunkt und <x_RO> und <y_RO> dem gegenüberliegenden Punkt einer jeden Fläche.

Datenstruktur des Programms

Das Programm folgt dem MVC-Muster, dargestellt durch die „Solver“ Klasse, die „DataManager“ Klasse und der Main Klasse und die dazugehörigen Datenobjekte.

Der Eintrittspunkt ist der Controller, die „Main“ Klasse, um über Argumente den Dateinamen der

Die View, in diesem Programm repräsentiert durch die „DataManager“ Klasse, speist Inputdateien in das System und schreibt die Output Dateien. Daten werden als Rohdaten, in diesem Fall Listen von Strings, an den Controller gegeben für die weitere Verarbeitung.

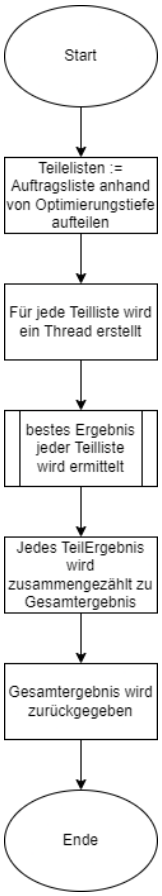
Der Controller kreiert das Model, hier die „Solver“ Klasse, und gibt die Rohdaten zur Umwandlung und Verarbeitung an dieses weiter.

Die Rohdaten werden in ein Auftragslistenobjekt umgewandelt, welches die Metadaten und eine Liste an Aufträgen beinhaltet. Aufträge bestehen aus Breiten, Längen, einer eindeutigen Identifikationsnummer und einer Beschreibung und können ihre Breite und Länge austauschen zur Änderung ihrer Orientierung.

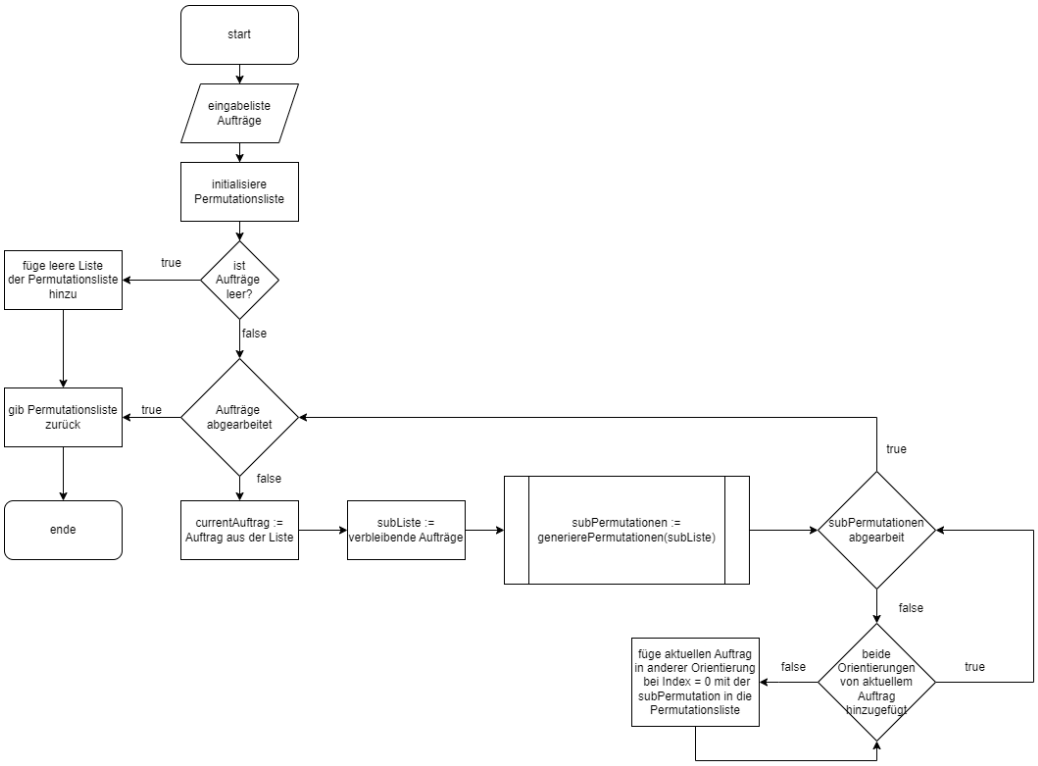
Die Lösung des Optimierungsproblems wird durch die Klasse Ergebnis dargestellt. Ein Ergebnis beinhaltet die Liste in der genauen Anordnung der Flächen, der Liste der verwendeten Anker und nichtverwendeten Andockpunkte sowie der sich aus den Flächen ergebenden Rollenhöhe und des genutzten Flächenanteils. Punkte beinhalten eine x und eine y Position, eine Unterscheidung, ob es sich um einen Andock- oder einen Ankerpunkt handelt, und eine Auftrags-ID um zu bestimmen welche Flächen erstellt werden. Flächen bestehen aus einem x und y Position sowie einem Auftrag, durch den sich die benötigte Fläche ableiten lässt.

Beschreibung des Verfahrens

Bei Start des Algorithmus wird zuerst die Liste an Aufträgen gemäß der Optimierungstiefe in Teillisten unterteilt. Dies ist sehr vorteilhaft, da das im späteren Verlauf verwendete Verfahren Permutationen der Teillisten verwendet, um die optimale Konstellation von Aufträgen zu bestimmen und diese mit einer Anzahl von n! Möglichkeiten sehr schnell ausarten.



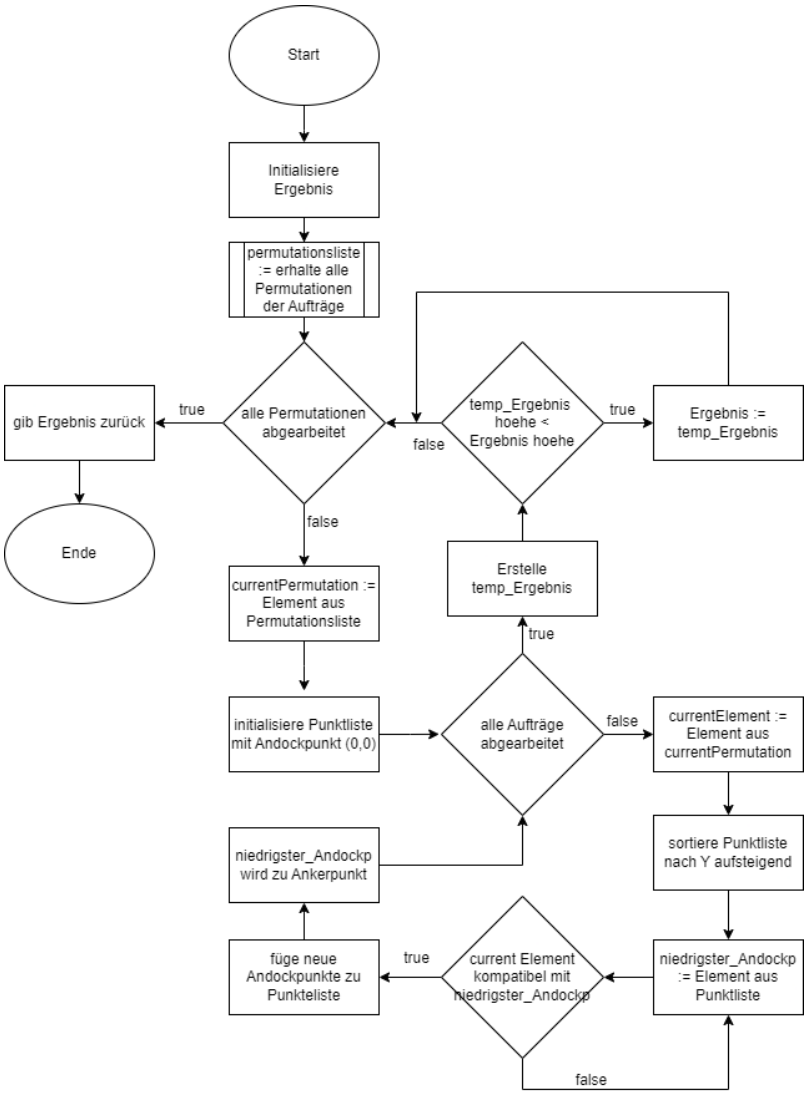
Das Ablauf startet für jede Teilliste einen eigenen Thread, in welchem für jede Teilliste das beste Teilergebnis berechnet wird. Hierbei wird für jede Teilliste ein Teilergebnis und anschließend die Permutationen der Teilliste erzeugt. Die Permutationen werden mit der selbst erstellten „PermutationUtil“ Klasse erstellt, um die aufgabenspezifische Orientierung jedes einzelnen Auftrags mit einzubeziehen.



Der Ablauf ist hierbei rekursiv und beachtet die Orientierung der Aufträge. In jedem Ablauf wird dafür eine Kopie des jeweiligen Auftrags erstellt und gedreht, um nicht per Referenz auf das ursprüngliche Objekt zu verweisen.

Nach Erstellung der Permutationen der Teillisten an Aufträgen werden diese nun abgearbeitet. Für jede Permutation wird eine punkteliste mit dem Startpunkt (0,0) angelegt. Anschließend werden die Aufträge der Permutation abgearbeitet wobei zuerst die Punkte nach ihrem Y wert sortiert, und anschließend abgearbeitet werden. Sobald ein Punkt mit dem Auftrag kompatibel ist, also nicht über den Rollenrand hinaus oder sich mit einer anderen bisher gesetzten Fläche überschneidet, wird der Auftrag als abgearbeitet gewertet, der Andockpunkt zu einem Ankerpunkt umgeändert und die beiden Andockpunkte der neuen Fläche zu der punkteliste der Permutation hinzugefügt. Sollte eine Kombination auf diese weise nicht optimal für den jeweiligen Auftrag sein, wird dies durch eine andere Permutation abgedeckt, somit müssen wir nicht jeden Andockpunkt überprüfen.

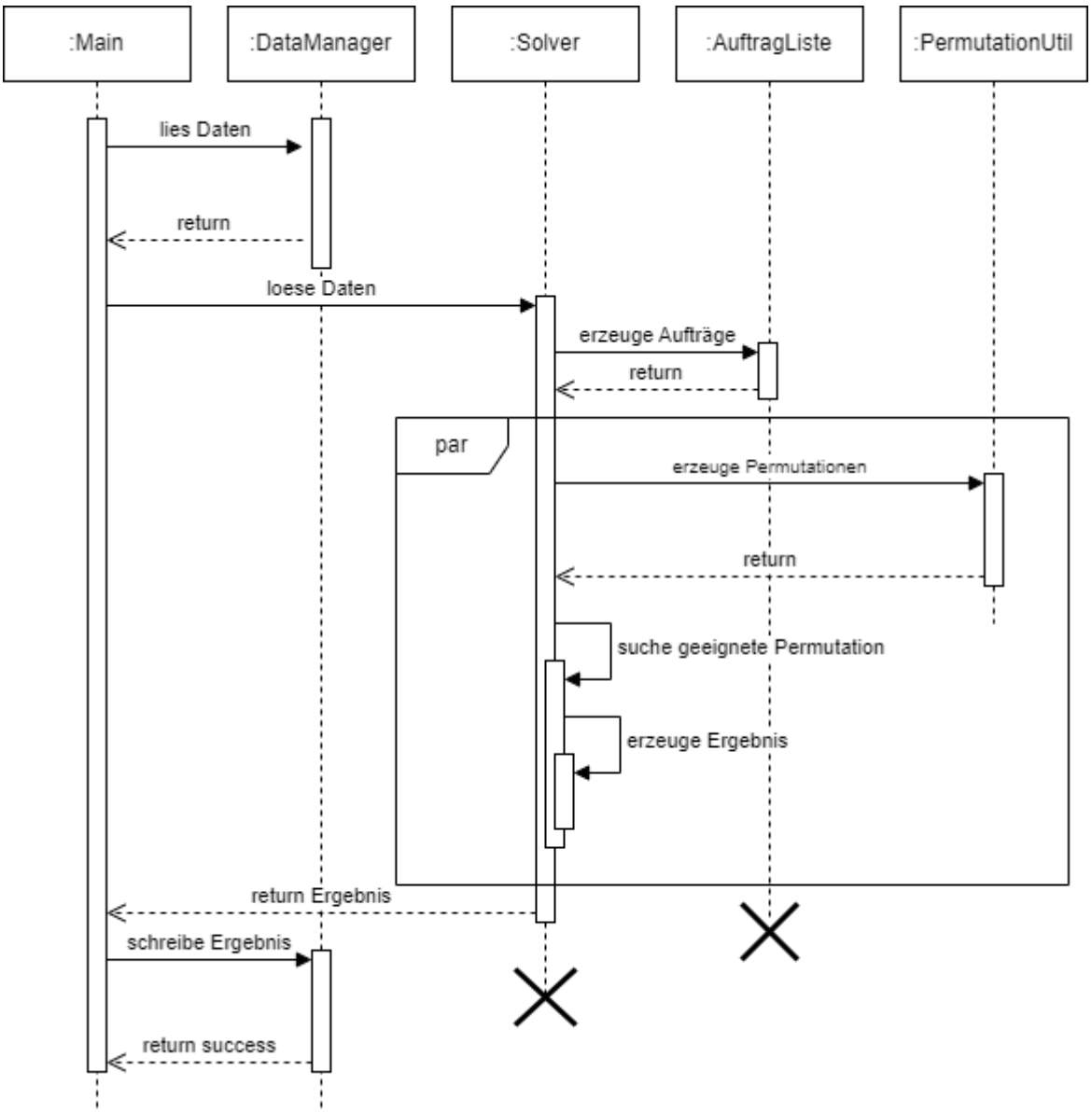
Sobald die Liste an Aufträgen abgearbeitet wurde und jeder Auftrag einer Fläche zugeordnet ist, wird ein Ergebnis erstellt und die benötigte Höhe dieser Lösung der jeweiligen Permutation berechnet. Sobald diese Höhe niedriger ist als die bisher gefundene Lösungshöhe, wird das Teilergebnis durch das Permutationsergebnis ersetzt. Sobald alle Permutationen abgearbeitet sind, wird dann das Teilergebnis aus dem Thread zurückgegeben.



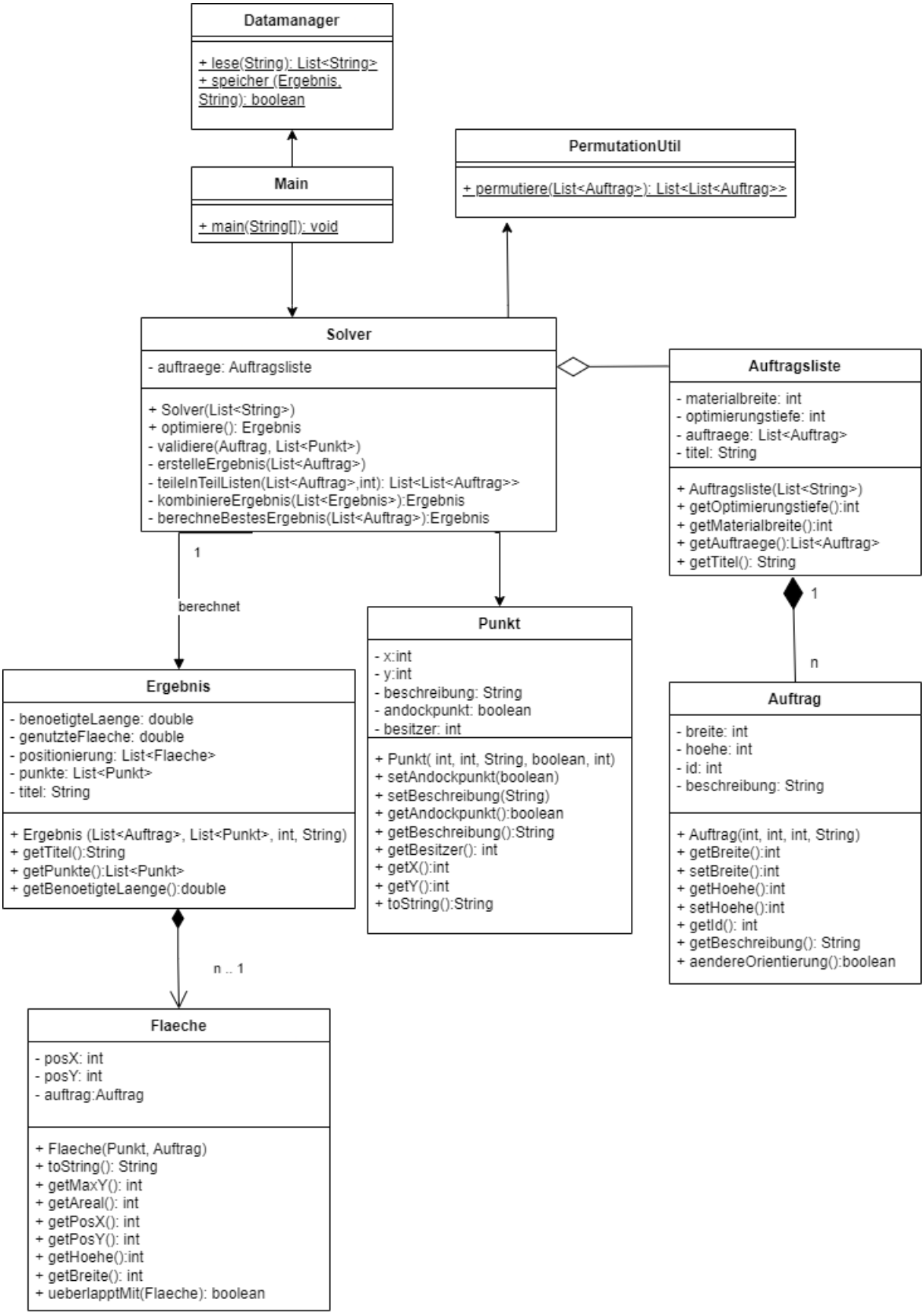
Die berechneten Teilergebnisse der einzelnen Threads werden daraufhin zusammengezählt, um ein Gesamtergebnis zu ergeben. Hierbei werden alle punkte aufeinanderfolgender Teilergebnisse um die Höhe des vorangegangenen Ergebnisses erweitert und dann als neues Ergebnisobjekt mit der Liste aller punkte zusammengeführt.

Entwickler Dokumentation

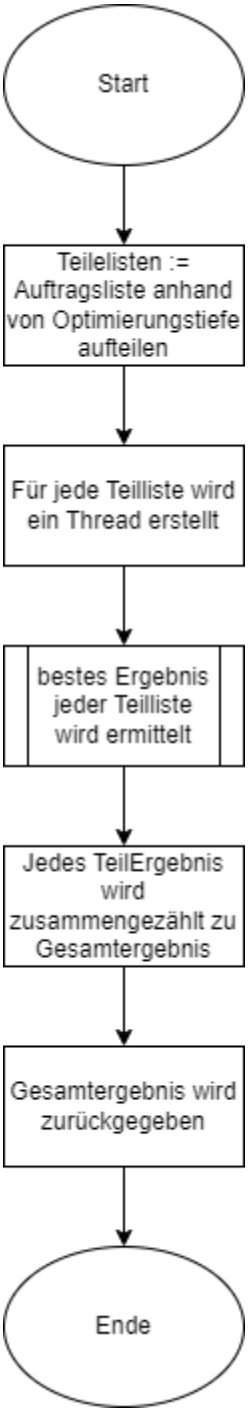
Sequenzdiagramm



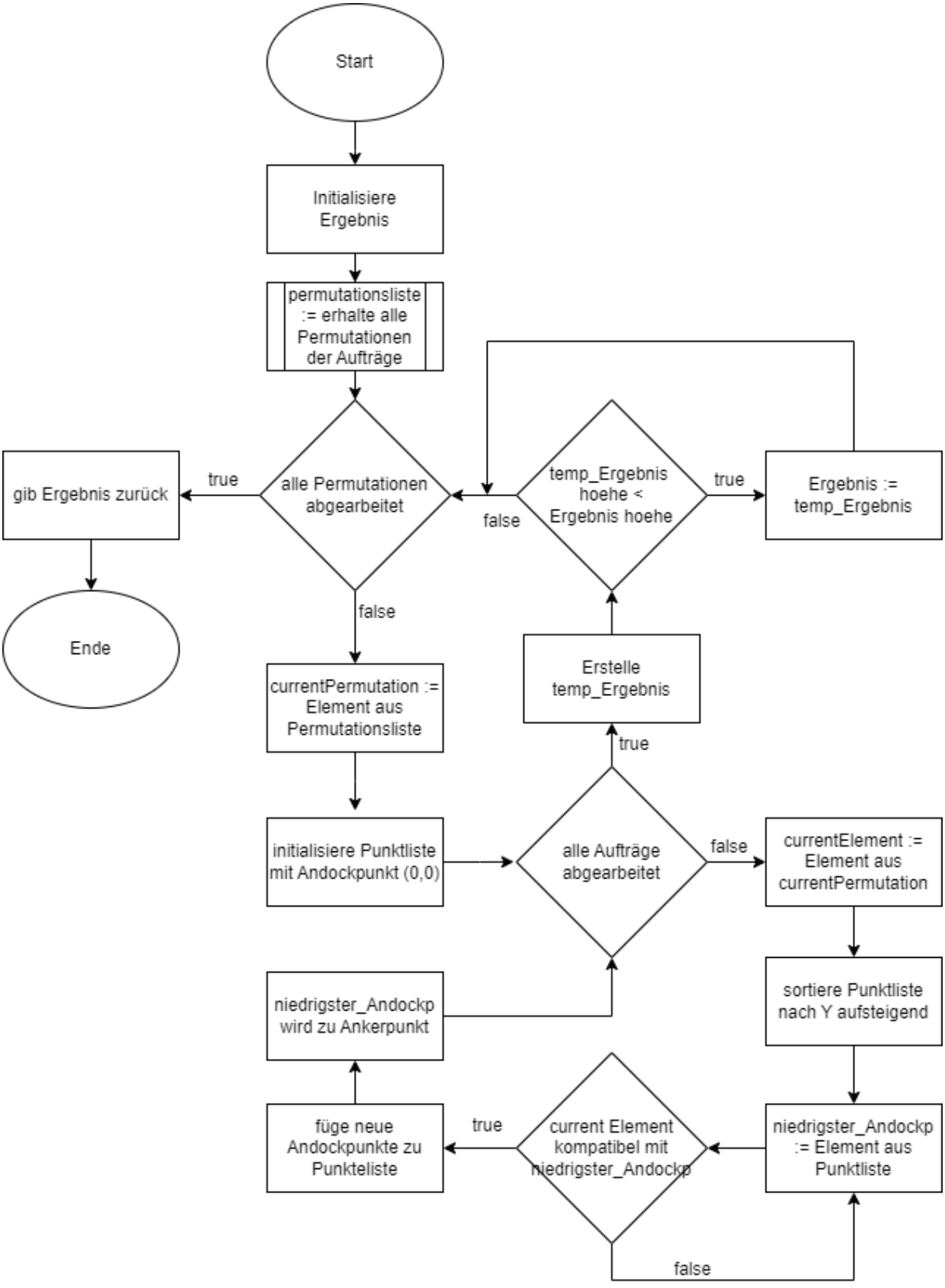
Klassendiagramm



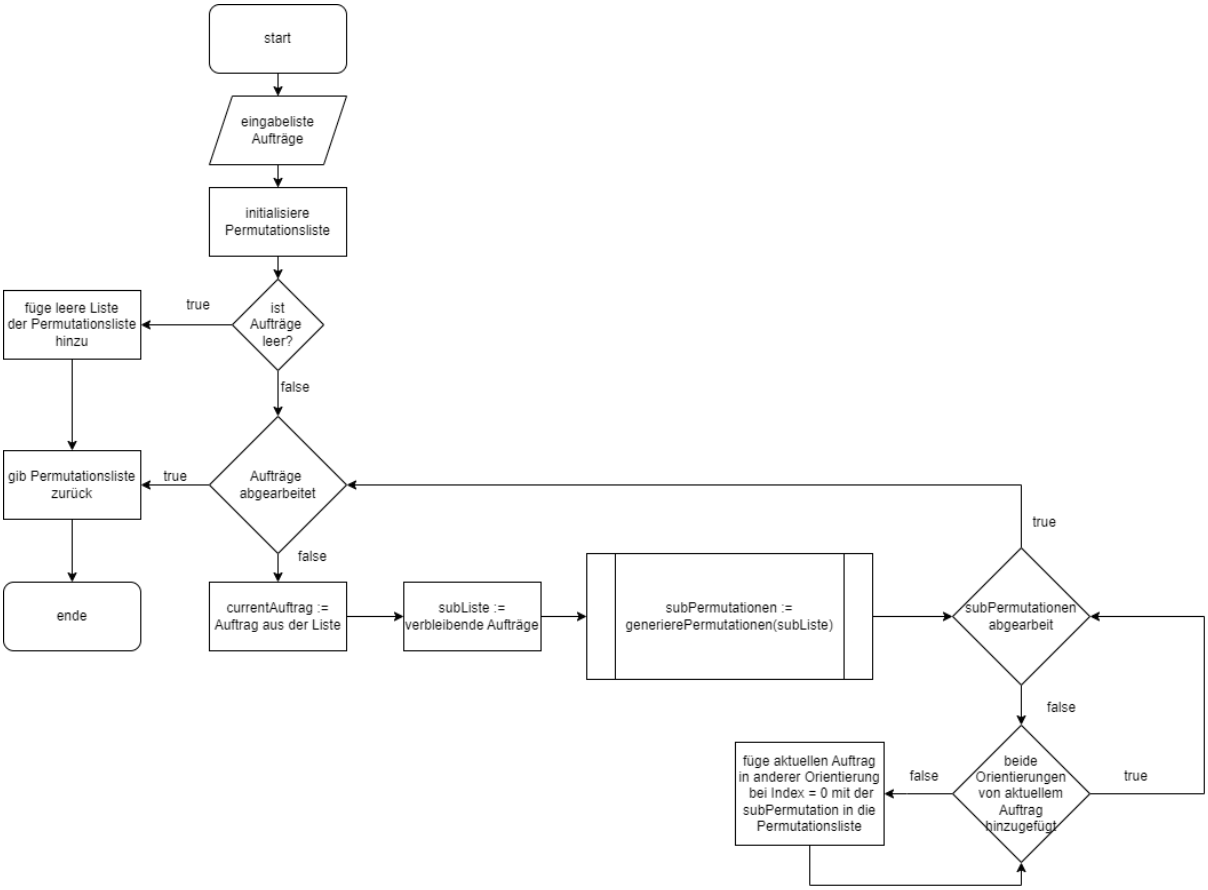
Ablaufdiagramm – optimiere()



Ablaufdiagramm – berechneBestesErgebnis()



Ablaufdiagramm – permutiere()



Testdiskussion

Die folgenden Dateien befinden sich im Ordner als Positivbeispiele:

Beispiel1.in	Beispiel aus Aufgabenstellung
Beispiel2.in	Beispiel aus Aufgabenstellung
Beispiel3.in	Beispiel aus Aufgabenstellung
Beispiel4-verlaengert.in	Abwandlung Beispiel3 mit mehr Einträgen
Beispiel5-verlaengert-tiefe-maximal.in	Abwandlung Beispiel4 mit maximal bearbeitbarer Optimierungstiefe

Zur Überprüfung des korrekten Verhaltens wurden die Beispiel Eingaben verwendet.

Außerdem wurde eine Abwandlung von Beispiel3 hinzugefügt mit deutlich mehr Einträgen, um die benötigte Dauer mit der Parallelisierung zu testen.

Des Weiteren wurde diese Datei auch noch umgewandelt mit der maximal für die Liste verwendbaren Optimierungstiefe.

Zum Testen der Grenz- und Fehlerfälle wurden des Weiteren folgende weitere testeingaben hinzugefügt:

Beispiel6-ungueltige-tiefe.in	Optimierungstiefe übersteigt Maximalwert
Beispiel7-unmoegliche-elemente.in	Elemente sind größer als das Rohmaterial
Beispiel8-tiefe-negativ.in	Die Optimierungstiefe ist negativ
Beispiel9-leer.in	Die Datei ist leer.

Im Fehlerfall werden .log Dateien in dem Angegebenen Output Ordner anstatt den Lösungen erstellt, welche den Grund des jeweiligen Fehlers dokumentieren.

Zu Beispiel 6:

Dadurch, dass die Permutation rekursiv erstellt wird, kommt es bei zu hohen Optimierungstiefen sowie dementsprechenden Auftragslisten zur Überreizung des Speichers bei Generierung der Listen. Um dies vorzubeugen werden Eingaben mit einer höheren Optimierungstiefe und Auftragsanzahl als 8 als ungültig angesehen und eine .log Datei erstellt.

Zu Beispiel 7:

Wenn Elemente in beiden Orientierungen, sowohl horizontal als auch vertikal, größer sind als die breite der Rohmaterialrolle, dann gibt es keine Lösung für diese Auftragsliste. In diesem Fall erzeugt das Programm ebenfalls eine .log Datei.

Zu Beispiel 8:

Eine Optimierungstiefe kleiner 1 ist unlogisch und führt zu Problemen bei der Aufteilung der Teillisten, weshalb in diesem Fall ebenfalls eine .log Datei erstellt wird.

Zu Beispiel 9:

Leere Listen können nicht bearbeitet werden und eine .log Datei wird erstellt.

Benutzeranleitung

Ordnerstruktur

Die Abgabe enthält die folgenden Inhalte:

Name	**Inhalt**
Tests	- Testeingabedaten - Skripte zum starten jeglicher .in Dateien - results-demo Ordner mit Demoausgaben der Eingabedaten
src	Enthält den Quelltext der Anwendung
Dokumentation.pdf	Diese Dokumentation
Optimierungsproblem.jar	Das Programm zur Berechnung optimaler Platzierung von rechteckigen Aufträgen auf eine begrenzte Rolle von Rohmaterial

Ausführen des Programms

Das Programm wird gestartet über eine Shell mit einem einfachen Java Command:

```
java -j Optimierungsproblem.jar <pfad/zu/dateiohneendung> <optional/ausgabe/pfad>
```

Da die Aufgabe explizit .in Dateien verlangt, reicht der Dateiname ohne Endung um die Eingabedatei zu finden. Standardmäßig werden die Ausgaben in dem Ordner der Eingabedatei erzeugt, dies kann aber durch optionale Angabe eines Ausgabepfads noch spezifiziert werden.
Am Ende der Berechnung des Programms wird die benötigte Dauer des Optimierungsverfahrens in der Konsole in Sekunden angezeigt.

Ausführen der Tests

Im Ordner „tests“ befindet sich ein .bat und ein .sh Skript um das Programm mit den beigelegten Testdateien zu starten. Die Skripte suchen hierfür jegliche .in Dateien in dem Ordner, in dem sie sich befinden und starten diese mit der Optimierungsproblem.jar welche sich neben oder eine Ebene über dem Skript befinden kann.

Über die Konsole kann man diese starten über:

<absoluter/Pfad/>skript.bat

oder mit bash:

<absoluter/Pfad/>skript.sh

oder wenn man sich bereits in dem Testordner befindet:

./skript.sh

Die Resultate werden dann in einem Ordner „results“ neben den Skripten gespeichert.

Version

Das Programm wurde in Java 21 geschrieben und funktioniert sichergestellt mit Version

java 21.0.5 2024-10-15 LTS

Ausblick

Da die Permutationsklasse auf einen rekursiven Algorithmus zugreift ist die Komplexität beziehungsweise Lesbarkeit des Codes in einem passablen Rahmen, führt aber zu Stack Overflow Fehlern, wenn die Optimierungstiefe einen bestimmten Wert erreicht.

In diesem Fall wäre ein iterativer Ansatz besser, aber da eine höhere Optimierungstiefe sehr schnell sehr viel mehr Zeit benötigt zur Berechnung wurde hierauf verzichtet. Nichtsdestotrotz wäre eine Implementierung gut für die Zukunft.

Abgesehen davon ist das Programm so geschrieben das die View ausgetauscht werden könnte, zum Beispiel gegen eine GUI, um die Ergebnisse direkt anzuzeigen. Alles, was hierfür extra geschehen müsste wäre eine genaue Schnittstelle zu definieren

Quellen

https://www.baeldung.com/java-executor-service-tutorial

https://www.geeksforgeeks.org/thread-pools-java/

https://www.tutorialspoint.com/batch_script/index.htm

https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/

https://docs.oracle.com/en/java/javase/21/docs/api/index.html

Eigenständigkeitserklärung

Ich versichere durch meine Unterschrift, dass ich das Prüfungsprodukt und die eingereichten Dokumentationen selbstständig angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche ausgewiesen sind. Die Arbeit hat in dieser Form keiner anderen Prüfungsinstitution vorgelegen.

_____ _____

Datum, Ort Unterschrift Prüfungsteilnehmer/-in