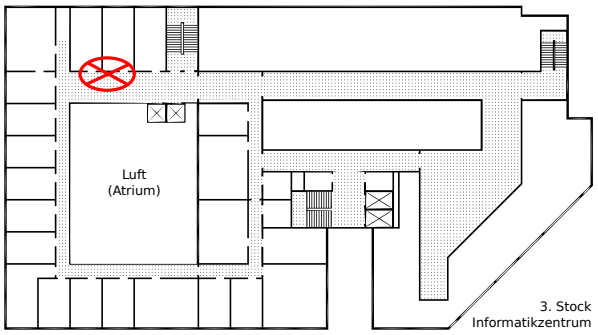
Abteilung Algorithmik **Sommer 2015**
Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig

Prof. Dr. Sándor P. Fekete
Dr. Christian Scheffer

Algorithmen und Datenstrukturen II
Übung 3 vom 20. 05. 2015

Abgabe der Lösungen bis zum Montag, den 08. 06. 2015 um 14:00 im Hausaufgabenrückgabeschrank.

Bitte die Blätter vorne deutlich mit eigenem Namen sowie Matrikel- und Gruppennummer versehen!



Auf diesem Blatt gibt es 40 Punkte, erreichbar (und abzugeben) sind aber lediglich (Teil-) Aufgaben im Gesamtwert von **maximal 30** Punkten. Mehr wird nicht gewertet!

Aufgabe 1 (Dynamische Programmierung): Die Firma *Lincoln Consultants* kann gemäß ihres aktuellen Mietvertrags in jedem Monat genau einen von zwei Büroräumen anmieten, die sich an weit voneinander entfernten Standorten, in Seattle und in Miami befinden.

Es soll nun ein optimaler Nutzungsplan für eine Folge von n Monaten im Voraus berechnet werden. Je nach geographischer Verteilung der im Monat i zu beratenden Kunden würden in diesem Monat bei Nutzung des Standorts Seattle Kosten von S_i Dollar anfallen, um alle Kunden zu betreuen, wohingegen bei Nutzung des Standorts Miami M_i Dollar anfallen würden. Der Umzug von einem Standort zu einem anderen Standort kann am Ende eines jeden Monats mit festen Kosten von U Dollar bewerkstelligt werden.

Die Aufgabe besteht nun darin, für eine vorgegebene Folge von Kostenalternativen die minimal notwendigen Gesamtkosten zu bestimmen. Zur Vereinfachung darf angenommen werden, dass es keinen Kostenunterschied für den ersten Monat macht, ob in Miami oder Seattle begonnen wird.

Beispiel: In der nachfolgend angegebenen Situation wäre für $U := 10$ der Wert einer optimalen Lösung $c = 1 + 3 + 10 + 2 + 4 = 20$, da die optimale Lösung in den ersten beiden Monaten das Büro in Seattle nutzt, dann einen Umzug vornimmt und dann das Büro in Miami nutzt.

	Monat 1	Monat 2	Monat 3	Monat 4
Seattle	1	3	20	30
Miami	50	20	2	4

a) Zeige durch Angabe eines Gegenbeispiels, dass der folgende Algorithmus nicht den Wert einer optimalen Lösung bestimmt.

```
1: Setze c := 0.                                   ▷ Initialisiere Kosten.
2: for i = 1 to n do
3:     if (S_i < M_i) then
4:         Setze c := c + S_i.                     ▷ Wähle Seattle.
5:     else
6:         Setze c := c + M_i.                     ▷ Wähle Miami.
7:     if (Umzug war notwendig) then
8:         Setze c := c + U.                       ▷ Addiere ggfs. Umzugskosten.
9:     return c.
```

b) Gib eine Situation an, in der jede optimale Lösung mindestens drei Umzüge erfordert. Begründe kurz Deine Antwort.

c) Entwirf und analysiere (bzgl. der Laufzeit) einen effizienten (d.h. polynomiellen) Algorithmus, der für gegebene Umzugskosten U und eine gegebene Folge von je n Werten M_i und S_i den Wert einer optimalen Lösung des obigen Problems bestimmt.

(3+2+5 Punkte)

Aufgabe 2 (Komplexität): Ein großer Onlineshop möchte seinen Dienst mit möglichst wenig Servern betreiben, um Kosten zu sparen. Damit alles läuft, müssen n Jobs mit einer Load von $0 < j_1, \ldots, j_n \leq 1$ dauerhaft ausgeführt werden. Jeder der baugleichen Server kann Jobs mit einer Gesamtload von maximal 1 ausführen, und Jobs können nicht auf mehrere Server verteilt werden.

Ideal wäre jetzt ein Algorithmus, der für jede Wahl von $j_1, \ldots, j_n$ bestimmt, wie viele Server gebraucht werden und wie die Jobs darauf verteilt werden müssen. Leider hat die IT noch keinen effizienten Algorithmus gefunden, sondern lediglich Approximationsalgorithmen. Betrachte im Kontext dieser Aufgabe Definition 1.25, die als Zusatzmaterial auf der Vorlesungsseite bereitgestellt wird.

a) Was genau garantiert ein ρ-Approximationsalgorithmus, $\rho > 1$, für dieses Problem?

b) Es ist bekannt, dass PARTITION (Problem 1.9 aus dem Vorlesungsskript) ein ,,schweres" Problem ist. Hilf der IT und zeige, dass man mit Hilfe eines ρ-Approximationsalgorithmus' für das Serverproblem auch jede Instanz von PARTITION lösen kann, wenn $1 \leq \rho < \frac{3}{2}$ ist.

c) Angenommen, es gibt keinen effizienten Algorithmus, der PARTITION löst, was bedeutet dann das Ergebnis von b) für das Serverproblem?

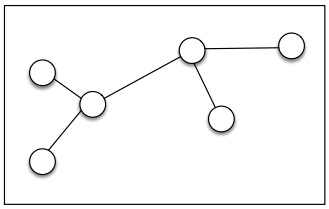
(2+6+2 Punkte)

Aufgabe 3 (TSP-Approximation): Betrachte eine n-elementige Punktmenge S in der Ebene. Ein *geometrischer Graph für S* ist ein Graph, dessen Knoten mit den Punkten in S identifiziert werden und in dem alle Kanten mit der euklidischen Distanz der mit ihren Endpunkten identifizierten Punkte aus S gewichtet sind. Ein *minimaler aufspannender Baum für S* ist ein ungerichteter, azyklischer geometrischer Graph für S, der

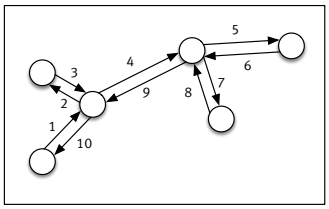
minimales Gesamtgewicht hat. Das *Problem des Handlungsreisenden* besteht darin, für eine gegebene Menge S wie oben einen geometrischen Graph minimalen Gesamtgewichts zu konstruieren, der aus genau einem Zyklus besteht, der jeden Knoten enthält. Dieses Problem ist $\mathcal{NP}$-schwer.

Der folgende, umgangssprachlich gegebene Algorithmus (eine exakte Beschreibung ist für die Bearbeitung dieser Aufgabe nicht notwendig!) bestimmt eine Approximationslösung für dieses Problem, d.h. einen Zyklus wie gefordert, dessen Gesamtgewicht nur um einen konstanten Faktor schlechter ist als die optimale Lösung.

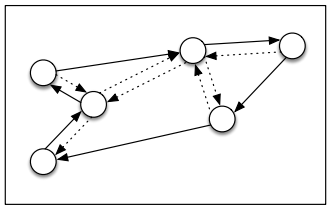
1: Erstelle einen minimalen Spannbaum T für S, wobei die Kantengewichte entsprechend der Distanzen der Endpunkte gegeben seien [dieser Algorithmus darf als gegeben angenommen werden].
2: Ersetze jede Kante $e = (v, w)$ in $E(T)$ durch zwei gerichtete Kanten (v, w) und (w, v) mit dem gleichen Gewicht.
3: Traversiere den so entstehenden Zyklus von einem beliebigen Startknoten aus und überspringe dabei alle Knoten, die bereits besucht wurden.



1. MST



2. Bsp. des Zyklus



3. Ausgabe

Beweise, dass der ausgegeben Zyklus maximal um den Faktor 2 länger ist als die optimale Lösung. **(10 Punkte)**

Aufgabe 4 (Implementierung Server Scheduling): Betrachte das Server-Scheduling-Problem aus Aufgabe 2.

a) Implementiere *First Fit (FF)*. Das heißt, jeder neue Job wird dem ersten Server zugewiesen, der ihn noch bewältigen kann. Falls der Job auf keinen Server mehr passt, wird ein neuer angefangen.

b) Implementiere *First Fit Decreasing (FFD)*. Das funktioniert wie FF, nur dass die Tasks vorher absteigend nach Load sortiert werden.

c) Implementiere *Best Fit (BF)*. Dabei wird jeder Job demjenigen Server mit minimalem freien Load zugeordnet, der die Task noch bewältigen kann. Passt der Job auf keinen Server, wird ein neuer eröffnet.

d) Implementiere eine untere Schranke, zum Beispiel $\lceil \sum_{i=1}^{n} j_i \rceil$. Bessere untere Schranken sind natürlich erlaubt, sollten aber kurz dokumentiert werden!

e) Halte in einer Tabelle fest, für welche Testinstanz FF, FFD und BF welche Anzahl von Servern benötigt, sowie die untere Schranke. Welche Heuristik funktioniert am besten?

Einziger Unterschied zwischen der Theorie in Aufgabe 2 und der Implementierung ist, dass die Server nicht die maximale Load 1 haben, sondern einen größeren Integer; auch die

einzelnen Jobs haben eine ganzzahlige Load. So umgehen wir Rundungsungenauigkeiten in der Gleitkommaberechnung. Die Testinstanzen stehen auf der Vorlesungshomepage[1] zur Verfügung.

Wir testen deine Software mit

```
javac ServerScheduling1234567.java && java ServerScheduling1234567 \
    < instance_xyz
```

Zur Abgabe: Ersetze `1234567` durch deine Matrikelnummer und gib die Javadatei per Mail an `scheffer@ibr.cs.tu-bs.de` ab. Nenne in der Mail Name, Matrikel- und Gruppennummer. Es gilt dieselbe Frist wie für die anderen Aufgaben. **(3+1+3+2+1 Punkte)**

[1] `http://www.ibr.cs.tu-bs.de/courses/ss15/aud2/`