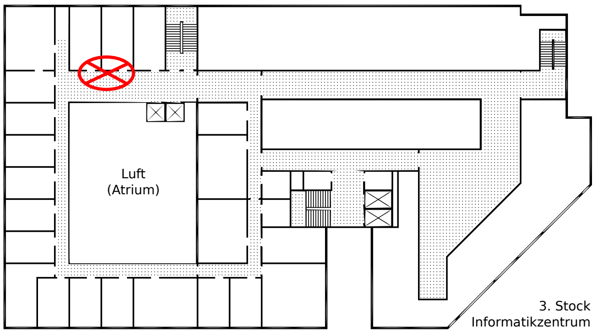
Algorithmen und Datenstrukturen 2 **SoSe 2023**
Prof. Dr. Sándor Fekete **Abgabe:** 28.06.2023
Dr. Phillip Keldenich **Rückgabe:** ab 04.07.2023

Hausaufgabenblatt 4

Abgabe der Lösungen bis zum 28.06.2023 um 13:30 Uhr im Hausaufgabenschrank bei Raum IZ 337 (siehe Skizze rechts). Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet. **Bitte die Blätter zusammenheften und vorne deutlich mit Namen, Matrikel- Übungs- und Gruppennummer versehen!**



Hausaufgabe 1 (Modellierung): **(8 Punkte)**
Eine wichtige Technik zum Lösen schwerer Probleme in der Praxis ist die sogenannte *Modellierung*. Dabei transformiert man eine Instanz I eines schweren Problems $\mathcal{A}$, an deren Lösung man interessiert ist, effizient in eine Instanz I' eines anderen schweren Problems $\mathcal{S}$. Die Instanz I' soll dabei zur Instanz I äquivalent sein in dem Sinne, dass sie eine Lösung hat genau dann, wenn I eine Lösung hat, und sich eine Lösung für I' effizient in eine Lösung für I transformieren lässt.

In der Regel verwendet man dabei Probleme $\mathcal{S}$, für die es bereits Programme gibt, die Instanzen des Problems in der Praxis gut und mit akzeptabler Geschwindigkeit lösen. Eine mögliche Option für $\mathcal{S}$ ist das Problem SAT aus der großen Übung.

Wir wollen uns in dieser Aufgabe mit dem verallgemeinerten Sudoku-Problem beschäftigen. Dabei besteht die Eingabe aus einer Zahl $k \geq 2$ und einem zweidimensionalen Feld $\mathcal{F}$ der Größe $k^2 \times k^2$ (normales Sudoku hat $k = 3$). Jede Zelle $z_{x,y}$ des gegebenen Feldes enthält einen Wert aus $\{\perp\} \cup \{1, \ldots, k^2\}$.

Das Feld ist unterteilt in eine Menge $\mathcal{B}$ von Blöcken aus je $k \times k$ Zellen, die analog zu den 3×3-Blöcken von Sudoku das gesamte Feld überdecken. Jede Zelle $z_{x,y}$ des Feldes ist Teil genau eines Blocks $B(x, y) \in \mathcal{B}$. Weiterhin ist jede Zelle $z_{x,y}$ Teil genau einer Zeile R_y und einer Spalte C_x. Zu einem gegebenen k und einem gegebenen Feld $\mathcal{F}$ wollen wir wissen, ob es eine Möglichkeit gibt, das Feld korrekt auszufüllen, also im gegebenen Feld $\mathcal{F}$ alle Einträge $\perp$ durch Zahlen aus $\{1, \ldots, k^2\}$ zu ersetzen, sodass in jedem Block, in jeder Spalte und in jeder Zeile alle Werte aus $\{1, \ldots, k^2\}$ genau einmal vorkommen, ohne dabei die gegebenen Zahlen zu ändern.

Gib eine Transformation an, die aus einer verallgemeinerten Sudoku-Instanz eine SAT-Instanz macht, die lösbar ist genau dann, wenn die verallgemeinerte Sudoku-Instanz lösbar ist. Die Transformation sollte effizient (mit einer polynomiell von k abhängigen Laufzeit) berechenbar sein (kurze Begründung reicht).
Hinweis: SAT erlaubt nur wahr/falsch-Variablen. Eine mögliche Lösung verwendet für ein reguläres Sudoku-Feld $81 \cdot 9 = 729$ Variablen.

Hausaufgabe 2 (Wiederholung: Dynamic Programming): (6 Punkte)
Wende das Dynamic Program für MAXIMUM KNAPSACK auf folgende Instanz an.

i	1	2	3	4
z_i	3	4	2	7
p_i	1	4	1	4

und $Z = 12$.

Hausaufgabe 3 (INTEGER KNAPSACK): (3+3 Punkte)
Betrachte die KNAPSACK-Variante INTEGER KNAPSACK, bei der wir jedes Objekt i beliebig (ganzzahlig) oft mitnehmen dürfen. Wir suchen also Werte $b_i \in \mathbb{Z}_{\geq 0}$, sodass die Gewichtsbedingung $\sum_i b_i z_i \leq Z$ eingehalten wird und der Gewinn $\sum_i b_i p_i$ maximiert wird. Wir wollen in dieser Aufgabe einen Branch-and-Bound-Algorithmus für dieses Problem entwickeln.

Die fraktionale Variante des Problems, bei der wir alle Objekte beliebig (nicht-negativ) oft mitnehmen dürfen (d.h. $b_i \in \mathbb{R}_{\geq 0}$), kann durch einen Greedy-Algorithmus in $O(n)$ Zeit gelöst werden (wir nehmen einfach soviele Einheiten des Objekts mit dem besten Preis pro Gewicht mit, wie wir können).

a) Beschreibe kurz einen Greedy-Algorithmus, der in $O(n \log n)$ Zeit eine ganzzahlige, zulässige Lösung zurückgibt, die nicht erweitert werden kann, d.h. das Hinzufügen eines beliebigen Elements macht die Lösung unzulässig.

b) Wandle den Pseudocode des Branch-and-Bound-Algorithmus für MAXIMUM KNAPSACK so ab, dass er für INTEGER KNAPSACK funktioniert.