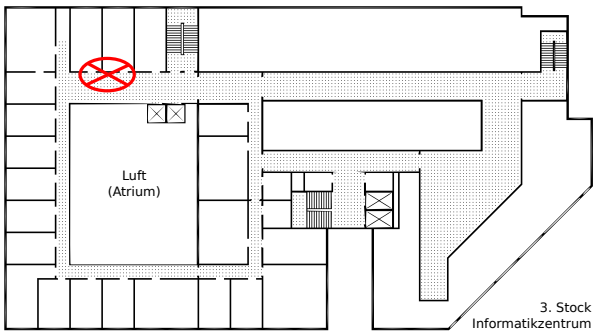
Abteilung Algorithmik **Winter 2019/20**
Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig

Dr. Linda Kleist
Phillip Keldenich
Dominik Krupke

Mathematische Methoden der Algorithmik
Übungsblatt 3 vom 27. 11. 2018

Die Abgabe eurer Lösungen zu diesem Blatt sind bis Mittwoch, den 11. 12. 2018 um 14:50 im Hausaufgabenschrank der Algorithmik möglich (siehe Skizze) oder direkt in der kleinen Übung.

Bitte die Blätter vorne deutlich mit eigenem Namen sowie Matrikelnummer versehen und zusammenheften!



Aufgabe 1 (Naiver LP-Solver mit Fundamentalsatz): Implementiere mit Python und NumPy einen Algorithmus der Lineare Programme in der Standardform $\min c^T x, Ax = b, x \geq 0$ optimal lösen kann indem er alle Basen ausprobiert. Warum ist die Lösung optimal?

Siehe hierzu auch das angehängte Beispiel. **(20 Punkte)**

Aufgabe 2 (Integer Programming: Bottleneck-MST): Das Minimum Spanning Tree Problem, das in einem gewichteten Graph $G = (V, E)$ nach einem Baum minimalen Gewichts sucht, kann in folgender Form als Integer Program angegeben werden.

$$\min \sum_{e \in E} w_e * \mathbf{x_e} \tag{1}$$

$$\sum_{e \in E} x_e = |V| - 1 \tag{2}$$

$$\sum_{e \in E(S, V \setminus S)} \mathbf{x_e} \geq 1 \qquad \forall S \subsetneq V, S \neq \emptyset \tag{3}$$

$$\mathbf{x_e} \in \mathbb{B} \qquad \forall e \in E \tag{4}$$

$x_e = 1$ bedeutet hierbei, dass die Kante e im Baum enthalten ist. Wir minimieren die Summe des Gewichts w_e der verwendeten Kanten. Die beiden Bedingungen geben an, dass eine Lösung mit $|V| - 1$ Kanten suchen die zusätzlich zusammenhängend ist (was einen Baum beschreibt).

Euch ist vielleicht aufgefallen, dass es exponentiell viele Ungleichungen in diesem Integer Program gibt. Tatsächlich ist dies gewöhnlich kein Problem, solange effizient bestimmt

werden kann ob eine beliebige Lösung (fraktional) gültig ist oder ansonsten eine verletzte Bedingungen genannt werden kann. In der Praxis würde man die zusätzlichen Bedingungen dann nach und nach hinzufügen bis wir eine gültige Lösung erhalten.

Ändere die Formulierung nun so ab, dass das Gewicht der größten Kante in dem Baum minimiert wird. **(10 Punkte)**

Aufgabe 3 (Mixed Integer Programming): Gib ein Mixed Integer Program an, das einen MST findet, bei dem das Gewicht jedes Pfades im MST zu einer bestimmten Wurzel v_r maximal l sein darf.
Also: Finde einen MST mit beschränkter Tiefe in dem der kürzeste Weg eines jeden Knotens zur Wurzel v_r maximal l ist.

Hinweise: 1. Nutze eine zusätzliche fraktionale Variable für jeden Knoten. 2. Nutze gerichtete Kanten. 3. Informiere dich über die Big-M-Methode. **(15 Punkte)**

Aufgabe 4 (Lösungsmengen von LPs): Betrachte ein LP in der Form

$$(P)\begin{cases} \max & c^T x & & \\ \text{s.t.} & Ax & \leq & b \\ & x & \in & \mathbb{R}^n \end{cases}$$

mit $A \in \mathbb{R}^{m \times n}$. Dabei sei $L = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ die Menge der zulässigen Lösungen von (P). Beweise oder widerlege folgende Aussagen:

(a) Es gilt immer $L \neq \emptyset$.

(b) Wenn $L \neq \emptyset$, dann existiert auch $\max\{c^T x \mid x \in L\}$.

(c) Wenn $\max\{c^T x \mid x \in L, x \text{ Basislösung}\}$ existiert, dann existiert keine weitere Basislösung mit dem selben Optimalwert.

(1+2+2 Punkte)

Aufgabe 5 (Konvexe Räume): Beweise, dass die Schnittmenge von konvexen Mengen wieder eine konvexe Menge ist. **(10 Punkte)**

numpy_examples

November 25, 2019

1 Python and NumPy Examples

Also checkout https://docs.scipy.org/doc/numpy/user/quickstart.html And: Google and StackOverflow are your friends. You can find a copy&paste ready example for nearly everything.

[1]:
```
# Import some libraries. NumPy needs to be installed
import numpy as np # all numpy function are now available via np.*
import itertools # for nice loops
```

[2]:
```
# Create a matrix A
A = np.array([[1, 0, 2, 1],
              [0, 2, 0, 4]])
```

[8]:
```
x = np.array([1,1,0,0]).reshape((1,-1)) # Create column vector x
A@x.T # @ does a matrix multiplication and T transforms a matrix
```

[8]:
```
array([[1],
       [2]])
```

[18]:
```
def is_basis(B: np.array):
    """
    Is B a basis (square and independent)?
    """
    try:
        return np.linalg.det(B)!=0 # a square matrix is independent if the␣
↪determinant is not zero
    except np.linalg.LinAlgError as lae: # if B is for example not square.
        return False
```

[22]:
```
column_indices = list(range(A.shape[1]))
for basis_indices in itertools.combinations(column_indices, A.shape[0]):
    B = A[:, basis_indices]
    if is_basis(B):
        print("Basis:\n", B)
```

```
Basis:
 [[1 0]
 [0 2]]
```

```
Basis:
 [[1 1]
 [0 4]]
Basis:
 [[0 2]
 [2 0]]
Basis:
 [[0 1]
 [2 4]]
Basis:
 [[2 1]
 [0 4]]
```