# Übung 5: Schwere Probleme

## Aufgabe 5-1:

Das Färben von Graphen gehört zu den schweren Problemen. Eine $k$-Färbung eines Graphen $G = (V, E)$ ist eine Zuordnung $f : V \to \{1, 2, \ldots, k\}$ von „Farben“ (eigentlich Zahlen) zu Knoten, sodass für jede Kante $e = \{u, v\} \in E$ gilt: $f(u) \neq f(v)$

(a) Begründen Sie, warum jeder Graph $G = (V, E)$ mit $\Delta(G)+1$ vielen Farben gefärbt werden kann, wobei $\Delta(G) = \max_{v \in V}\{\deg_G(v)\}$ der maximale Knotengrad im Graphen $G$ ist.

(b) Schreiben Sie einen Backtracking-Algorithmus, der zum gegebenen Graphen $G = (V, E)$ und der gegebenen Zahl $k \in \mathbb{N}$ prüft, ob der Graph $G$ mit $k$ Farben gefärbt werden kann.

(c) Implementieren Sie den Algorithmus aus Teil (b). Testen Sie Ihr Programm für einige Graphen, die im Moodle-Kurs zur Verfügung stehen. Dort steht auch eine Datenstruktur `Graph` bereit, um Graphen in C++-Programmen nutzen zu können.

(d) Der Algorithmus aus Teil (b) löst das sogenannte Entscheidungsproblem: Ist der Graph mit $k$ Farben färbbar? Wie kann der Algorithmus erweitert werden, damit das Optimierungsproblem gelöst wird: Bestimme das minimale $k$, sodass der Graph $k$-färbbar ist.

(e) Schreiben Sie einen Branch-and-Bound-Algorithmus, der eine optimale Färbung eines Graphen bestimmt. Vergleichen Sie die Laufzeiten der Algorithmen.

## Aufgabe 5-2:

In der Vorlesung wurde eine Lösung des 0/1-Rucksackproblems mittels dynamischer Programmierung vorgestellt, siehe Algorithmus DynamicKP auf Folie 121. Wir hatten die Laufzeit des Algorithmus mit $\mathcal{O}(n \cdot G)$ angegeben. Eine solche Laufzeit nennt man pseudo-polynomiell: Solange $G$ klein ist im Vergleich zur Anzahl der Objekte, z.B. $G \leq n^k$ für einen konstanten Wert $k$, dann ist die Laufzeit in $\mathcal{O}(n^{k+1})$ und damit polynomiell. Sind allerdings die Gewichtswerte groß im Vergleich zur Anzahl der Elemente, z.B. $G \approx 2^n$, dann ist die Laufzeit in $\mathcal{O}(n \cdot 2^n)$ und somit nicht polynomiell.

Wir wollen nun versuchen, einen Approximationsalgorithmus für das 0/1-Rucksackproblem zu entwickeln. Betrachten Sie zunächst den folgenden Algorithmus GreedyKP, der die Idee aufgreift, die wir beim Bruchteil-Rucksackproblem genutzt haben, das Sortieren der Objekte anhand Wert pro Gewicht.

```
1: function GreedyKP(w⃗, g⃗, G)
2:     sortiere die Objekte, sodass w_1/g_1 ≥ ... ≥ w_n/g_n
3:     M := ∅
4:     wght := 0
5:     for i := 1 to n do
6:         if wght + g_i ≤ G then
7:             M := M ∪ {i}
8:             wght := wght + g_i
9:     return M
```

Betrachten wir ein Maximierungsproblem wie das Rucksackproblem, dann definieren wir zu einem Algorithmus $A$ den relativen Fehler $R_A(I)$ des Algorithmus zu einer konkreten Eingabe $I$ als $R_A(I) = \mathrm{Opt}(I)/A(I)$. Dabei ist $\mathrm{Opt}(I)$ der Wert einer optimalen Lösung zur Eingabe $I$, wohingegen $A(I)$ den Wert der durch Algorithmus $A$ berechneten Lösung bezeichnet.

Betrachten Sie die folgende Eingabe $I = (\vec{w}, \vec{g}, G)$ mit

- $w_i = g_i = 1$ für $i \in \{1, \ldots, n-1\}$
- $w_n = G - 1$, $g_n = G$ mit $G = n \cdot k$ für ein beliebig großes $k$.

Wie groß ist der relative Fehler für Algorithmus GreedyKP für diese Eingabe $I$?

## Aufgabe 5-3:

Betrachten wir noch einmal den Algorithmus DynamicKP aus der Vorlesung. Dort haben wir folgenden Ansatz genutzt: Zu einem festen Gewicht $G$ suche eine Teilmenge $S \subseteq \{1, \ldots, n\}$ der Objekte, sodass der Gesamtwert $\sum_{k \in S} w_k$ maximal wird.

Wir können auch einen gegensätzlichen Ansatz wählen: Zu einem festen Wert $W$ suche eine Teilmenge $S \subseteq \{1, \ldots, n\}$ der Objekte, sodass das Gesamtgewicht $\sum_{k \in S} g_k$ minimal wird. Auch mit diesem Ansatz können wir das Rucksackproblem lösen. Dazu sei

$$\text{wght}(w, i) := \min \left\{ \sum_{k \in S} g_k \mid S \subseteq \{1, \ldots, i\},\ \sum_{k \in S} w_k \geq w \right\}$$

das minimale Gewicht einer Belegung aus den ersten $i$ Objekten mit Wert mindestens $w$. Für $i = 0$ dürfen keine Objekte gewählt werden. Die Idee hierbei ist, dass $\text{wght}(w, i)$ angibt, wie schwer der Rucksack mindestens sein muss, damit bei einer Auswahl aus den ersten $i$ Objekten der Gesamtwert $w$ beträgt.

Eine Lösung des 0/1-Rucksackproblems ist also der maximale Wert $w$, sodass $\text{wght}(w, n) \leq G$ ist. Geben Sie einen Algorithmus DynamicKP-2 an, der diese Idee nutzt und mittels dynamischer Programmierung arbeitet. Welche Laufzeit hat der Algorithmus?

## Aufgabe 5-4:

Die Laufzeit des Algorithmus aus Aufgabe 5-3 kann reduziert werden, wenn die Werte verkleinert werden. Sei $c$ eine „ausreichend große“ natürliche Zahl.

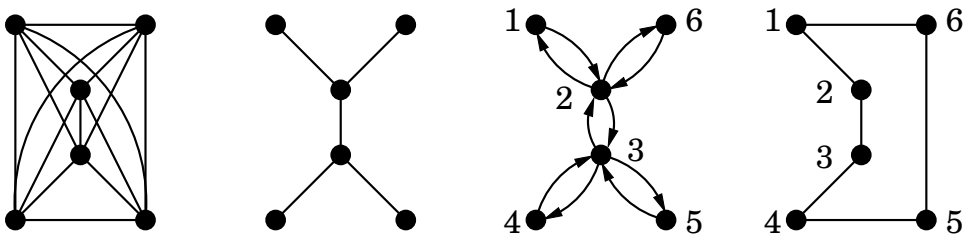
**function** ApproxKP($\vec{w}$, $\vec{g}$, $G$)
  **for** $i := 1$ to $n$ **do**
    $w'_i := \lfloor w_i / c \rfloor$
  DynamicKP-2($\vec{w'}$, $\vec{g}$, $G$)

Wie groß kann der relative Fehler bei dem Algorithmus ApproxKP werden? Ist der relative Fehler beschränkt?

## Aufgabe 5-5:

Wir wollen einen Approximationsalgorithmus für das metrische Traveling-Salesperson-Problem entwickeln. Sei $G = (V, E, c)$ ein vollständiger, gewichteter Graph mit einer symmetrischen Kostenfunktion $c : E \to \mathbb{R}^+$, wobei zusätzlich für alle $u, v, w \in V$ die Dreiecksungleichung $c(\{u, v\}) + c(\{v, w\}) \geq c(\{u, w\})$ gilt.

Zunächst berechnen wir einen minimalen Spannbaum $T$ für $G$, siehe dazu auch die folgende Abbildung. Anschließend führen wir auf dem Baum $T$ eine Tiefensuche aus und sei $v_1, \ldots, v_n$ die Preorder-Nummerierung der Knoten.




Die Folge $v_1, \ldots, v_n, v_1$ ist eine Rundreise. Wie groß kann der relative Fehler bei diesem Vorgehen werden? Ist der relative Fehler beschränkt? Betrachten Sie dazu die Folge, in der die Knoten bei der Tiefensuche auf $T$ besucht werden, im Beispiel also $(1, 2, 3, 4, 3, 5, 3, 2, 6, 2, 1)$. Wie lang ist diese Rundreise im Vergleich zu einer optimalen Rundreise?

Entfernen Sie dann sukzessiv mehrfach vorkommende Knoten aus der Folge. Im Beispiel tritt der Knoten 3 als erstes zweimal auf. Entfernen wir das zweite Vorkommen des Knotens 3, so erhalten wir als neue Folge $(1, 2, 3, 4, 5, 3, 2, 6, 2, 1)$. Vergleichen Sie die Länge der alten mit der neuen Rundreise.

Um zu sehen, dass der obige relative Fehler auch tatsächlich erreicht werden kann, wenden Sie den Approximationsalgorithmus auf den folgenden Graphen an. Der vollständige Graph $K_{2n} = (V, E, c)$ mit $V = \{1, \ldots, 2n\}$ habe z.B. die folgenden Kosten:

$$\begin{aligned} c(\{i, i+1\}) &= 1 \text{ für } i \in \{1, \ldots, n-1\} \\ c(\{i, i+n\}) &= 1 \text{ für } i \in \{1, \ldots, n\} \end{aligned}$$

Die Distanzen seien durch die Manhatten-Distanz der Punkte gegeben. Daher gilt also z.B. auch $c(\{1, n\}) = n - 1$ oder $c(\{n, n+1\}) = n$.