Algorithmen und
Berechnungskomplexität II
Sommersemester 2020

**Abgabe: 11.07.2020, 12.00 Uhr**
**Besprechung: KW 29**

UNIVERSITÄT BONN

Prof. Dr. Heiko Röglin
Anna Großwendt, David Kübel
*Institut für Informatik*

# Übungsblatt 12

## Aufgabe 12.1: Approximation Graphtraversierung

Wir betrachten das Problem der *Graphtraversierung*. Gegeben ist ein zusammenhängender Graph $G = (V, E)$ mit Kanten- und Knotengewichten, und ein Startknoten $v_s \in V$. Das Gewicht eines Knotens $v$ ist $w_v$ und das einer Kante $e$ ist $w_e$, alle Gewichte sind $\geq 1$. Alle Agenten starten auf dem Startknoten $v_s$, und diese können sich entlang der Kanten von $G$ bewegen. Eine Kante $e$ kann jeweils nur von mindestens $w_e$ Agenten überquert werden, und ein Knoten $v$ darf erstmals nur mit mindestens $w_v$ vielen Agenten betreten werden. Wird $v$ erstmals betreten so werden sofort $w_v$ Agenten auf $v$ platziert – diese dürfen $v$ nicht mehr verlassen. Das Ziel ist mit möglichst wenigen Agenten auf jedem Knoten $v$ mindestens $w_v$ Agenten zu platzieren.

Wir betrachten folgenden Algorithmus zum Traversieren eines Graphen $G$; der Algorithmus TraversiereBaumOptimal$(T, v_s)$ wird als Black-Box verwendet; sie berechnet eine Strategie mit der Baum $T$ von Knoten $v_s$ aus mit einer kleinstmöglichen Anzahl Agenten traversiert wird.

**GraphtraversierungMST**$(G = (V, E), v_s)$
1. Konstruiere einen minimalen Spannbaum $T = (V, E_T)$ von $G$
2. TraversierungsStrategie $S :=$ TraversiereBaumOptimal$(T, v_s)$
3. **return** $S$

Sei $N := \sum_{v \in V} w_v$ und $w_{max} = \max_{e \in E_T} w_e$. Zeigen Sie

(a) $S$ benötigt maximal $N + w_{max}$ Agenten zum Traversieren von $G$.

(b) Zum Traversieren von $G$ benötigt man mindestens $w_{max} = \max_{e \in E_T} w_e$ Agenten, wobei $E_T$ die Kantenmenge eines minimalen Spannbaums von $G$ ist.

(c) $S$ benötigt zum Traversieren von $G$ maximal doppelt so viele Agenten wie mindestens notwendig.

Es darf angenommen werden, dass alle nicht platzierten Agenten sich immer auf einem Knoten befinden bzw. sich in einem Schritt alle über die gleiche Kante bewegen.

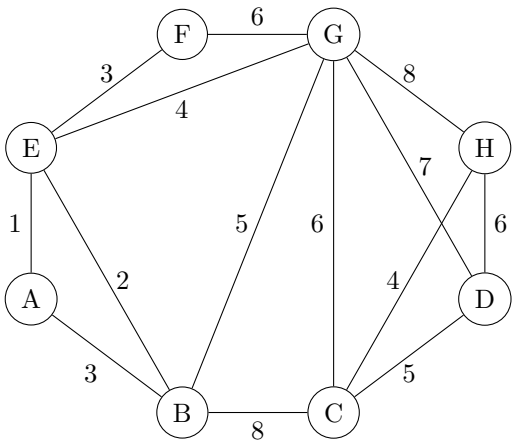
## Aufgabe 12.2: $1/2$-Approximation `Bin Covering`

Wir betrachten nun das Problem `Bin Covering`, eine Variante des Problems `Bin Packing`, welches in der Vorlesung vorgestellt wurde und ebenfalls NP-vollständig ist. Gegeben sei eine Menge $M$ von $n$ Objekten $\{w_1, \ldots, w_n\}$ mit Gewichten $w_i \in [0, 1]$ und beliebig viele Behälter unbeschränkter Kapazität. Ziel ist es eine Zuweisung der Elemente von $M$ auf die Behälter zu finden, die möglichst viele der Behälter füllt, d. h. die Summe der Gewichte der Elemente eines Behälters beträgt mindestens 1.

Finden Sie eine $\frac{1}{2}$-Approximation, d. h. geben Sie einen Algorithmus an, welcher mindestens halb so viele Behälter wie die optimale Lösung befüllt.

## Aufgabe 12.3: 2-Approximation metrisches TSP

Wenden Sie den in der Vorlesung vorgestellten Algorithmus zur Berechnung einer 2-Approximation auf die unten abgebildete Instanz des metrischen Traveling Salesperson Problems an. Dabei sei das Kantengewicht einer nicht eingezeichneten Kante $\{u, v\}$ gerade die Länge eines kürzesten Weges von $u$ nach $v$. Starten Sie die Suche nach einem Eulerkreis in Knoten A und besuchen Sie bei Wahlmöglichkeit Knoten mit im Alphabet vorangehenden Buchstaben zuerst. Zeichnen Sie die resultierende TSP-Tour und geben Sie die finalen Kantengewichte an!




## Aufgabe 12.4: `Vertex Cover` parametrisiert

Für die Entscheidungsvariante des Problems `VERTEX COVER` ist die Eingabe ein ungerichteter Graph $G = (V, E)$ und ein Parameter $k \in \mathbb{N}$. Es soll überprüft werden, ob $G$ ein `VERTEX COVER` der Kardinalität höchstens $k$ besitzt. Betrachten Sie folgenden Algorithmus:

`RECURSIVE-VC-CHECK`$(G, k)$

1. Falls $E = \emptyset$ und $k \geq 0$, **return ja**.
2. Falls $k \leq 0$, **return nein**.
3. Wähle beliebige Kante $(u, v)$ aus $E$.
4. Bestimme Graphen $G_u$ (bzw. $G_v$), durch Löschen von $v$ (bzw. $u$) und inzidenten Kanten in $G$.
5. **return** `RECURSIVE-VC-CHECK`$(G_u, k-1) \vee$ `RECURSIVE-VC-CHECK`$(G_v, k-1)$.

(a) Bestimmen Sie die Laufzeit von `RECURSIVE-VC-CHECK` in Abhängigkeit von (der Größe von) $G$ und $k$.

(b) Für welche Werte von $k$ ist die Laufzeit von `RECURSIVE-VC-CHECK` polynomiell in der Eingabegröße und für welche nicht?