Lehrstuhl II für Mathematik
Prof. Dr. Arie M.C.A. Koster
Timo Gersing, Christoph Grüne
optib@math2.rwth-aachen.de

WS 2019/20

OPTIMIERUNG B
Aufgabenblatt 10
Abgabe bis 27.01.2020

Für diese Abgabe wurde ein Framework in den Programmiersprachen C++ und Java angelegt. Alle Abgaben sind in einem der beiden Frameworks zu erledigen. Die Stellen, an denen Code einzufügen ist, sind durch TODO-Kommentare markiert. Die Funktionssignatur darf nicht verändert werden. Die Einlese- und Ausgabefunktionen sind dafür bereits implementiert.

Die Frameworks benutzen Bibliotheken, die Sie für die Implementierung verwenden sollen.
Für Java ist dies die Bibliothek JGraphT (https://jgrapht.org/, https://github.com/jgrapht/jgrapht). Dokumentationen dafür finden Sie unter den obigen Links.
Für C++ ist dies LEMON. Falls Sie C++ verwenden wollen, können Sie die mitgelieferte C-Make-Vorlage benutzen, um LEMON in eine Entwicklungsumgebung einzubinden. Eine mögliche Entwicklungsumgebung, für die diese C-Make funktioniert ist CLion (https://www.jetbrains.com/de-de/clion/), das kostenlos für Studierende erwerbbar ist. Ansonsten ist es auch möglich unter Linux mit g++ die Dateien zu kompilieren. Folgen Sie dafür der Anleitung unter http://lemon.cs.elte.hu/trac/lemon/wiki/Documentation. Dort finden Sie auch die gesamte Dokumentation für LEMON.

Bei allen Aufgaben ist gefordert, dass Sie die Algorithmen anwenden sollen. Liefern Sie hierfür alle Ausgaben, die das Programm erstellt ab.

Aufgabe 1: **6 Punkte**
Implementieren Sie einen Algorithmus der überprüft, ob ein gegebener ungerichteter Graph bipartit ist. Falls der Graph bipartit ist, soll der Algorithmus eine zulässige Bipartition ausgeben, falls der Graph nicht bipartit ist, soll ein Nachweis (ein sogenanntes Zertifikat) hierfür ausgegeben werden; implementieren Sie daher eine geeignete Datenstruktur in "NonBipartitionCertificate*". Wenden Sie den Algorithmus auf die Graphen in "graph*.lgf" an. Die Dateien, die hierfür verändert werden müssen heißen "BipartitionAlgorithm*".
Hinweis: Der Graph ist nicht zwangsläufig zusammenhängend.

Aufgabe 2: **4 Punkte**
Implementieren Sie einen Algorithmus der ein Matching maximalen Gewichts in einem bipartiten Graphen bestimmt. Nutzen Sie diesen um ein solches in allen bipartiten Graphen "graph*.lgf" aus Aufgabe 1. Die Dateien, die hierfür verändert werden müssen heißen "BipartitionMatchingAlgorithm*".

Aufgabe 3: **5 Punkte**
Der Lehrstuhl hat ein internes Bundesliga Tippspiel. Hierbei betrachten wir nur die Hinrunde, die an 17 Spieltagen aus allen möglichen Duellkonstellationen der 18 teilnehmenden Mannschaften besteht. Das Tippspiel hat die folgenden Regeln:

- An jedem Spieltag wählt man genau eine Mannschaft aus.
- Man darf jede Mannschaft nur an einem Spieltag auswählen.
- Wenn die augewählte Manschaft gewinnt, erhält man 3 Punkte, bei einem unentschieden einen Punkt und bei einer Niederlage keine Punkte.
- Der Spieler mit den meisten Punkten am Ende der Hinrunde gewinnt.

Nach dem Ende der Hinrunde, wenn die Ergebnisse bereits bekannt sind, wollen wir herausfinden, wie die optimalen Tipps ausgesehen hätten.

a) Modellieren Sie das Problem mit aus der Veranstaltung bekannten Mitteln.

b) Lösen Sie die Instanz in "Season12.txt"

Die Dateien, die hierfür verändert werden müssen heißen "FootballBettingGameAlgorithm*".

Aufgabe 4: **5 Punkte**

Sei $\mathcal{G} = \{(V, E, c) \mid (V, E)$ ist ungerichteter Graph, $c : E \to \mathbb{N}_0\}$ die Menge ungerichteter Graphen mit Kostenfunktion auf den Kanten. Wir definieren nun die Funktion $f : \mathcal{G} \to \mathcal{G}$, die einen ungerichteten Graphen wie folgt erweitert. Sei $G = (V, E, c) \in \mathcal{G}$. Dann ist $f(G) = (V, E', c')$ mit

$$E' = E \cup \{\{u, v\} \mid \text{ es existiert ein Pfad von } u \text{ nach } v \text{ in } G\}$$

und

$$c'(e) = \begin{cases} c(e), & \text{falls } e \in E \\ \min_{p \in \mathcal{P}_{(u,v)}} \sum_{e \in p} c(e), & \text{falls } e \in E' \setminus E \end{cases},$$

wobei $\mathcal{P}_{(u,v)}$ die Menge der Pfade von u nach v bezeichnet.

Implementieren Sie den Algorithmus von Christofides für das TSP auf den Graphen $\mathcal{G}_f = \{f(G) \mid G \in \mathcal{G}\}$. Der Algorithmus soll dabei eine TSP-Tour ausgeben, falls es eine gibt. Ansonsten soll er einen Nachweis liefern, warum es keine Tour geben kann. Wenden Sie den Algorithmus auf die Erweiterungen, definiert durch Funktion f, aller Graphen "graph*.lgf" an. Die Dateien, die hierfür verändert werden müssen heißen "ChristofidesAlgorithm*".