Aufgabe 1: Prim-Algorithmus

Es soll der Prim-Algorithmus in der Version aus der Vorlesung 6.c (mit der Verwaltung der Knoten aus 6.b) implementiert werden.

Dabei gehen wir davon aus, dass die Knoten im Graphen durchgehend beginnend mit 0 nummeriert seien. Das bedeutet: Ein Graph mit vier Knoten besitzt die Knoten $0, \ldots, 3$.

Im bereits (als Zip-Archiv in OPAL) vorgegebenen Quelltext existieren die folgenden Dateien:

- Verzeichnis `adsaufgabe2`
  Dies entspricht dem Paketnamen in Java – bitte **nicht** ändern.
- Datei `Graph.java`
  Diese bereits fertig ausprogrammierte Klasse realisiert die Speicherung des Graphen. Diese Datei darf **nicht** verändert werden. Im Hauptprogramm `TestProgramm.java` sehen Sie, wie sie mit dieser Klasse eigene Graphen für Testzwecke konfigurieren können.
- Datei `IGraph.java`
  Dieses Interface gibt an, welche Methoden zur Abfrage von Informationen über den Graphen der Implementation des Dijkstra-Algorithmus zur Verfügung stehen. Dies sind drei Methoden, welche die Anzahl der Knoten, die Existenz einer Kante im Graphen und das Gewicht an einer Kante liefern. Diese Datei darf **nicht** verändert werden.
- Datei `IPrim.java`
  Dieses Interface gibt die Signatur der Methoden vor, über die der Prim-Algorithmus gestartet werden soll und die Ergebnisse dann anschließend abgerufen werden. Ferner enthält diese Datei in den Kommentare konkrete Anforderungen hinsichtlich der zu implementierenden Methoden. Diese Datei darf **nicht** verändert werden.
- Datei `Prim_MEINNAME.java`
  Von dieser Datei legen Sie sich eine Kopie an, wobei Sie im Datei- und Klassennamen `MEINNAME` durch Ihren Vor- und Nachnamen oder Login-Namen ersetzen. In dieser Datei programmieren Sie dann die gewünschte Lösung. Achten Sie bitte darauf, dass die Klasse das `implements IPrim` enthält, der Klassenname zum Dateinamen angepasst wird und verzichten Sie zum Zeitpunkt der Abgabe auf jegliche Ausgaben z.B. per `System.out.print` in Ihrer Klasse. Konkret müssen die folgenden Methoden implementiert werden:
  - Konstruktor: Der Konstruktor kann leer sein, da die eigentliche Initialisierung der benötigten Attribute in der Methode `computeShortestPaths` erfolgen sollte.
  - `boolean computeMSB(IGraph graph, int sourceVertex)`:
    Die Berechnung wird beginnend mit dem angegebenen Startknoten durchgeführt und die Ergebnisse innerhalb des Objekts Ihrer Klasse gespeichert.
  - `int getProcessingOrder(int vertex)`: Zusätzlich zu den normalen Daten des Prim-Algorithmus soll diese Funktion für einen Knoten liefern, als wievielter Knoten er aus der Randmenge ausgewählt wurde. Der Startknoten habe die Nummer 1, die weiteren Knoten werden lückenlos durchnummeriert.
  - `int getPredecessor(int targetVertex)`: Der gespeicherte Vorgängerknoten im Minimalen-Spannbaum wird für einen Knoten erfragt.

- `int getSumOfWeights()`: Das Gesamtgewicht des errechneten minimalen Spannbaums wird erfragt.
- Datei `TestProgramm.java`
  Diese Klasse enthält ein Hauptprogramm zum Testen der eigenen Implementation. Ihre Version der Datei `Prim_MEINNAME.java` soll mit der unveränderten Version dieser Klasse laufen und keine Fehlermeldung liefern.

Bitte lesen Sie die Kommentare in den Dateien aufmerksam, bevor Sie mit der Bearbeitung beginnen!

Laden Sie Ihre Version der Datei `Prim_MEINNAME.java` in OPAL unter *Prog.Aufg. 2* hoch. Alle anderen Dateien sollen von Ihnen nicht geändert werden und sind daher auch nicht abzugeben. Achtung: Ich benötige in jedem Fall die Datei mit der Endung `.java` – Dateien mit der Endung `.class` werden nicht bei der Bewertung berücksichtigt.

Ich werde die Datei mit meinen erweiterten Testfällen übersetzen und entsprechend auswerten. Ferner prüfe ich auf Quelltext-Kopien (Plagiate).

Abgabedatum: **Mittwoch, 12. Juni 2024, 23:59 Uhr**

Hinweise:

1. Bitte beachten Sie, dass Ihre Klasse die Ergebnisse ordentlich kapseln soll. Wie in der Spezifikation beschrieben, sollen parallel in verschiedenen Objekten Ihrer Klasse unterschiedliche Kürzeste-Wege-Suchen speicher- und abrufbar sein. Auch soll ein Objekt nacheinander für mehrere unterschiedliche Kürzeste-Wege-Suchen benutzbar sein.
2. Bitte beachten Sie, dass die von mir zur Verfügung gestellten Testfälle nicht alle möglichen Situation und Sonderfälle abdecken. Sie sollten in jedem Fall mit eigenen Testfällen Ihre Abgabe prüfen, da verschiedene der fehlenden Situationen in meinen geheimen Testfällen enthalten sein werden.
3. Kopieren Sie keinen Quelltext – auch nicht in Teilen. Erarbeiten Sie immer zunächst ein eigenes Programm. Wenn dies nicht funktioniert: reden Sie in ihrer Gruppe über Fehlermeldungen, bitten Sie andere um Hilfe bei der Fehlersuche, aber kopieren Sie keine Lösung direkt. Wenn Ihr Programm läuft: tauschen Sie Testtreiber, d.h. Hauptprogramme mit anderen Testfällen, aus, und versuchen Sie möglichst "besondere" Testszenarien zu finden.
4. Punktvergabe: Bei der Umsetzung von Algorithmen oder Datenstrukturen in Quelltext gibt es letztendlich kein "fast richtig". Die algorithmischen Problemstellungen haben klare Spezifikationen und müssen diesen gerecht werden. Daher wird bei der Bewertung diejenige Implementation, die alle Testfälle meistert, deutlich besser bewertet als partiell korrekte Implementationen. Es gibt 20 Testfälle, die jeweils einen Punkt wert sind; für komplett korrekte Implementationen gibt es einen Bonus von 10 Punkten.
5. Alles, was mir Zusatzarbeit verursacht, wird mit einem Punkteabzug quittiert. Dazu zählen Umlaute und "ß" im Dateinamen, Ausgaben im Terminal und nicht fehlerfrei übersetzende Klassen.
6. Erscheinen mir Abgaben zu ähnlich, werde ich sie ggf. nicht werten oder die erreichten Punkte auf die Abgaben aufteilen.