TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Programmieraufgabe P2

1 Vorbereitung

Um diese Programmieraufgabe erfolgreich durchführen zu können, müssen folgende Schritte umgesetzt werden:

1. Laden Sie das Framework `P2.zip` aus TUWEL herunter.
2. Entpacken Sie `P2.zip` und öffnen Sie das entstehende Verzeichnis als Projekt in IntelliJ (nicht importieren, sondern öffnen).
3. Öffnen Sie die nachfolgend angeführte Datei im Projekt in IntelliJ. In dieser Datei sind sämtliche Programmiertätigkeiten durchzuführen. Ändern Sie keine anderen Dateien im Framework und fügen Sie auch keine neuen hinzu.
   `src/main/java/exercise/StudentSolutionImplementation.java`
4. Füllen Sie Vorname, Nachname und Matrikelnummer in der Methode `StudentInformation provideStudentInformation()` aus.

2 Hinweise

Einige Hinweise, die Sie während der Umsetzung dieser Aufgabe beachten müssen:

- Lösen Sie die Aufgaben selbst und nutzen Sie keine Bibliotheken, die diese Aufgaben abnehmen.
- Sie dürfen beliebig viele Hilfsmethoden schreiben und benutzen. Beachten Sie aber, dass Sie nur die oben geöffnete Datei abgeben und diese Datei mit dem zur Verfügung gestellten Framework lauffähig sein muss.

3 Übersicht

In dieser Programmieraufgabe wird das Directed Feedback Vertex Set Problem thematisiert. Dabei geht es darum, Knoten aus einem gerichteten Graphen zu entfernen, um alle Kreise zu eliminieren. Eine praktische Anwendung des Problems ist das Auflösen eines Deadlocks, indem (möglichst wenige) Prozesse gestoppt werden.

4 Theorie

Die notwendige Theorie kann in den Vorlesungsfolien „Graphen“ und „Greedy-Algorithmen“ gefunden werden.

5 Implementierung

Bei dieser Programmieraufgabe werden Sie mit der Klasse `Graph` arbeiten. `Graph` repräsentiert einfache gerichtete Graphen. `Graph g` bietet folgende Methoden an, die für Ihre Implementierung hilfreich sein könnten:

- `int numberOfVertices()`: Gibt die Anzahl an Knoten zurück.
  Mittels `int number = g.numberOfVertices()` können Sie die Anzahl der Knoten im Graph `g` auslesen.
- `int[] getVertices()`: Gibt die Knoten des Graphen zurück.
  Die Knoten des Graphen können mit `int[] vertices = g.getVertices()` abgefragt werden. Die Knoten werden als Array von Knoten-IDs repräsentiert.
- `int inDegree(int vertexId)`: Gibt den Eingangsknotengrad eines Knotens zurück.
  Der Eingangsknotengrad des Knoten mit der ID 1 kann mittels `int indeg = g.inDegree(1)` ermittelt werden. Bei der Übergabe einer ungültigen Knoten-ID wird `-1` zurückgegeben.
- `int outDegree(int vertexId)`: Gibt den Ausgangsknotengrad eines Knotens zurück.
  Der Ausgangsknotengrad des Knoten mit der ID 1 kann mittels `int outdeg = g.outDegree(1)` ermittelt werden. Bei der Übergabe einer ungültigen Knoten-ID wird `-1` zurückgegeben.

- `int[] getSuccessors(int vertexId)`: Gibt die Nachfolger eines Knotens zurück.
  Die Nachfolger des Knoten mit der ID 1 können mittels `int[] successors = g.getSuccessors(1)` ermittelt werden. Die Nachfolger werden als Array von Knoten-IDs repräsentiert. Bei der Übergabe einer ungültigen Knoten-ID wird `null` zurückgegeben.

- `int[] getPredecessors(int vertexId)`: Gibt die Vorgänger eines Knotens zurück.
  Die Vorgänger des Knoten mit der ID 1 können mittels `int[] predecessors = g.getPredecessors(1)` ermittelt werden. Die Vorgänger werden als Array von Knoten-IDs repräsentiert. Bei der Übergabe einer ungültigen Knoten-ID wird `null` zurückgegeben.

- `boolean removeVertex(int vertexId)`: Entfernt einen Knoten und alle zu dem Knoten adjazente Kanten aus dem Graph.
  Der Knoten mit ID 1 kann mittels `g.removeVertex(1)` aus dem Graph entfernt werden. Alle Kanten, die adjazent zum Knoten mit ID 1 sind werden ebenfalls entfernt. Falls das Entfernen des Knotens erfolgreich war wird `true` zurückgegeben. Andernfalls wird `false` zurückgegeben.

- `boolean isAcyclic()`: Gibt `true` zurück, wenn der Graph Kreise enthält, und `false`, wenn er keine enthält.
  Ob `Graph g` Kreise enthält, kann mittels `g.isAcyclic()` festgestellt werden. Diese Methode greift auf Ihre Implementierung aus der Aufgabe von Kapitel 5.1 zurück und funktioniert daher erst, sobald Sie diese Aufgabe korrekt gelöst haben.

5.1 Überprüfung auf Kreise

Implementieren Sie die Methode `boolean isAcyclic(Graph g)`.

Diese Methode überprüft, ob der gegebene, gerichtete `Graph g` Kreise enthält. Falls ja muss `false` zurückgegeben werden, andernfalls `true`. `Graph g` darf in dieser Methode nicht verändert werden, d.h. `g.removeVertex(v)` darf *nicht* aufgerufen werden. Implementieren Sie die Methode so, dass sie auch funktioniert, wenn die Knoten-IDs nicht fortlaufend von 0 bis $n-1$ (n: Anzahl der Knoten) sind, sondern manche Knoten in der fortlaufenden Nummerierung fehlen.

5.2 Heuristische Bewertungen der Knoten

Implementieren Sie die Methoden

- `double heuristic1(Graph g, int vertex)`,
- `double heuristic2(Graph g, int vertex)` und
- `double heuristic3(Graph g, int vertex)`.

Alle drei Methoden sollen aus Eingangsknotengrad $\texttt{deg}^-$ und Ausgangsknotengrad $\texttt{deg}^+$ des Knoten mit der ID `vertex` den Rückgabewert wie folgt berechnen:

- `double heuristic1(Graph g, int vertex)`: Gibt $\texttt{deg}^- + \texttt{deg}^+$ zurück.
- `double heuristic2(Graph g, int vertex)`: Gibt $\texttt{deg}^- \cdot \texttt{deg}^+$ zurück.
- `double heuristic3(Graph g, int vertex)`: Gibt $\texttt{deg}^- + \texttt{deg}^+ - 0.3 \cdot |\texttt{deg}^- - \texttt{deg}^+|$ zurück.

Optional können Sie sich auch selbst eine Heuristik entwerfen und in `double heuristic4(Graph g, int vertex)` implementieren. Je eher ein Knoten in einem optimalen Feedback Vertex Set ist, desto höher sollte der zurückgegebene Wert sein, um sicher zu stellen, dass der Greedy-Algorithmus von der nächsten Aufgabe gute Ergebnisse liefert. Sollten Sie keine eigene Heuristik entwerfen, belassen Sie die Methode `double heuristic4(Graph g, int vertex)` wie sie ist.

5.3 Finden eines kleinen Directed Feedback Vertex Set

Implementieren Sie die Methode `void greedyFeedbackVertexSet(Graph g, Heuristic h, int[] feedbackVertexSet)`.

Die Knoten-IDs in `Graph g` sind bei Aufruf der Methode $\{1, \ldots, n\}$, wobei n die Anzahl der Knoten in `g` ist. Der Parameter `Heuristic h` ist ein Wrapper, der zur Berechnung der Heuristik dient. Je nach Testinstanz nutzt dieser Wrapper entweder `heuristic1`, `heuristic2` oder `heuristic3` aus Kapitel 5.2. `Heuristic h` hat nur eine Methode:

Index	0	1	2	3	4	5	6
Knoten-ID	7	3	5	2	-1	-1	-1

Tabelle 1: Gelöschte Knoten als Array

- `double eval(Graph g, int vertex)`: Gibt den Wert der Heuristik für einen Knoten zurück.
  Mittels `double heuristicValue = h.eval(g, 1)` wird die Heurisik für den Graphen `g` und seinen Knoten mit der ID 1 angewandt.

Die Methode `greedyFeedbackVertexSet` soll nacheinander Knoten löschen bis `Graph g` keine Kreise mehr enthält. Es soll immer der Knoten gelöscht werden, der die größte heuristische Bewertung hat, d.h. `h.eval(g, v)` soll für den Knoten, der gelöscht wird, den höchstmöglichen Wert liefern.

Legen Sie die IDs der gelöschten Knoten im Array `int[] feedbackVertexSet` ab. Das Array `feedbackVertexSet` hat einen Eintrag für jeden Knoten im Graph und die Einträge sind mit -1 initialisiert. Einen expliziten Rückgabewert gibt es nicht. Die IDs im Array müssen entsprechend der Reihenfolge, in der die Knoten gelöscht werden, abgelegt werden. Nicht benötigte Stellen am Ende des Arrays müssen Sie unverändert lassen.

Angenommen es werden die Knoten 7, 3, 5 und 2 in dieser Reihenfolge gelöscht. Der korrekte Inhalt des Arrays nach Ausführen der Methode ist in Tabelle 1 dargestellt.

6 Testen

Führen Sie zunächst die `main`-Methode in der Datei `src/main/java/framework/Exercise.java` aus.

Anschließend wird Ihnen in der Konsole eine Auswahl an Testinstanzen angeboten, darunter befindet sich zumindest `abgabe.csv`:

```
Select an instance set or exit:
[1] abgabe.csv
[0] Exit
```

Durch die Eingabe der entsprechenden Ziffer kann entweder eine Testinstanz ausgewählt werden, oder das Programm (mittels der Eingabe

von `0`) verlassen werden. Wird eine Testinstanz gewählt, dann wird der von Ihnen implementierte Programmcode ausgeführt. Kommt es dabei zu einem Fehler, wird ein Hinweis in der Konsole ausgegeben.

Relevant für die Abgabe ist das Ausführen der Testinstanz `abgabe.csv`.

Die weiteren Testinstanzen `isAcyclic.csv`, `heuristic1.csv`, `heuristic2.csv`, `heuristic3.csv` und `greedyFeedbackVertexSet.csv` sind nur zum jeweiligen Testen der einzelnen Unteraufgaben gedacht.

7 Evaluierung

Wenn der von Ihnen implementierte Programmcode mit der Testinstanz `abgabe.csv` ohne Fehler ausgeführt werden kann, dann wird nach dem Beenden des Programms im Ordner `results` eine Ergebnis-Datei mit dem Namen `solution-abgabe.csv` erzeugt.

Die Datei `solution-abgabe.csv` beinhaltet Ergebnisse der Ausführung der Testinstanz `abgabe.csv`, welche in einem Web-Browser visualisiert werden können. (Auch Ergebnis-Dateien anderer Testinstanzen können zu Testzwecken visualisiert werden.) Öffnen Sie dazu die Datei `visualization.html` in Ihrem Web-Browser und klicken Sie rechts oben auf den Knopf *Ergebnis-Datei auswählen*, um `solution-abgabe.csv` auszuwählen.

Beantworten Sie basierend auf der Visualisierung die Fragestellungen aus dem folgenden Abschnitt.

8 Fragestellungen

Öffnen Sie `solution-abgabe.csv` und bearbeiten Sie folgende Aufgaben- und Fragestellungen.

1. Bei einer einfachen Implementierung der Greedy-Methode werden jedes Mal, bevor ein Knoten entfernt wird, alle verbleibenden Knoten neu evaluiert. Das bedeutet einen linearen Aufwand für jeden entfernten Knoten. Gibt es für lichte Graphen (auch dünn besetzte Graphen genannt) einen schnelleren Weg, falls eine der von Ihnen implementierten Heuristiken verwendet wird? Wie kann man dann effizient den nächsten Knoten, der entfernt werden soll, bestimmen?

2. Bei einer einfachen Implementierung wird jedes Mal, nachdem ein Knoten entfernt wird, überprüft, ob der Graph noch Kreise enthält. Angenommen die Reihenfolge, in der die Knoten mit der Heuristik gelöscht werden, ist schon bestimmt worden. Die Greedy-Methode würde dann die ersten k Knoten von dieser Reihenfolge zurückgeben, wobei der Graph genau nach Entfernen des k-ten Knoten azyklisch wird. Wenn man die Folge der Graphen betrachtet, die durch das Entfernen der Knoten in dieser Reihenfolge entsteht, bedeutet das, dass die ersten k Graphen Zyklen enthalten und alle danach kommenden Graphen azyklisch sind.

   - Wie kann k effizient bestimmt werden?
     Hinweis: Es müssen nur $\mathcal{O}(log(n))$ Graphen auf Zyklen überprüft werden, um k zu bestimmen.
   - Was für eine asymptotische Gesamtlaufzeit ergibt sich, wenn zum Bestimmen der Reihenfolge die Maßnahme aus Punkt 1 verwendet wird und es sich wieder um lichte Graphen handelt?

3. Liefert der Greedy-Algorithmus für eine der heuristischen Bewertungen immer die Optimallösung? Begründen Sie Ihre Antwort kurz.

4. Öffnen Sie in der Visualisierung den Tab „abgabe.csv“. Dargestellt wird eine Tabelle, die die Lösungsqualität in Abhängigkeit von der Instanzgröße zeigt. Für jede Instanz ist die Heuristik, die am besten abgeschnitten hat, in hellgrün hinterlegt. Ist die von der Heuristik gefundene Lösung gleich gut wie die bekannte obere Schranke, so ist die entsprechende Zelle in dunkelgrün hinterlegt. Schneidet eine der Heuristiken besser bzw. schlechter ab? Wie schneidet der Greedy-Algorithmus im Vergleich zur bekannten oberen Schranke ab?

   Fertigen Sie im Anschluss einen Screenshot der Tabelle an.

5. Vergleichen Sie die Ausührungen des Greedy-Algorithmus mit den verschiedenen Heuristiken für die Instanz `cyc_0020-02`. Um den Vergleich zu vereinfachen, können Sie auch `solution-abgabe.csv` mehrfach in einem Browser-Fenster öffnen, oder `visualization.html` in zwei separaten Browser-Fenstern öffnen. Wählen Sie dazu im Dropdown unten links die Instanz aus und scrollen Sie nach unten. Nach Auswahl der Instanz erscheint der jeweilige Graph. Mithilfe der Knöpfe über dem Graphen können Sie nun die Ausführung des Greedy-Algorithmus Schritt für Schritt verfolgen. Nacheinander werden die von der Heuristik gewählten Knoten aus dem Graphen entfernt.

Wie unterscheiden sich die Lösungen, die vom Greedy-Algorithmus mit den verschiedenen Heuristiken gefunden werden?

Springen Sie anschließend in den Visualisierungen zum letzten Schritt. Mit der Maus können Sie die Knoten verschieben. Zeigen Sie, dass es sich tatsächlich um azyklische Graphen handelt, indem Sie Sie die Knoten so anordnen, dass alle Kanten weiter rechts enden als sie starten. Machen Sie von den so geordneten Graphen jeweils einen Screenshot.

6. Da die Graphen nach dem letzten Schritt keine Kreise mehr enthalten, haben sie eine topologische Sortierung. Geben Sie für die drei Graphen, die für die Instanz `cyc_0020-02` nach Ausführen des Greedy-Algorithmus entstehen, eine solche topologische Sortierung an. Ist diese Sortierung eindeutig? Warum bzw. warum nicht?

Fügen Sie Ihre Antworten in einem Bericht gemeinsam mit den erstellten Screenshots der Visualisierungen der Testinstanz `abgabe.csv` zusammen.

9 Abgabe

Laden Sie die Datei `src/main/java/exercise/StudentSolutionImplementation.java` in der TUWEL-Aktivität *Hochladen Source-Code P2* hoch. Fassen Sie diesen Bericht mit den anderen für das zugehörige Abgabegespräch relevanten Berichten in einem PDF zusammen und geben Sie dieses in der TUWEL-Aktivität *Hochladen Bericht Abgabegespräch 2* ab.

10 Nachwort

Wie Sie in der Vorlesung gelernt haben sind gerichtete azyklische Graphen (Directed Acyclic Graphs, DAGs) eine ganz besondere Klasse von Graphen, da sie es eben erlauben eine topologische Sortierung der Knoten zu finden. Repräsentieren die Kanten im Graphen Reihenfolgebeschränkungen zur Abarbeitung der Knoten, so liefert die topologische Sortierung eben eine gültige Knotenfolge. In verschiedenen praktischen Anwendungen kommt es nun vor, dass der gegebene Graph kein DAG ist, d.h., eben zyklische Abhängigkeiten beinhaltet, aber mit möglichst wenigen Eingriffen – in

Form einer Entfernung von Knoten – eine topologische Sortierung gefunden werden soll.

Ein konkretes Beispiel ist das Zeichnen von Graphen in der euklidischen Ebene, sodass sich keine bzw. möglichst wenige Kanten kreuzen. Für DAGs gibt es spezielle Algorithmen, die auf eine topologische Sortierung bzw. konkreter die Einteilung der Knoten in Layers basieren, sodass Kanten jeweils nur von Knoten eines Layers zu Knoten tieferer Layer gezeichnet werden müssen. Die Zeichnung kann so effizient Layer für Layer aufgebaut werden. Diese Zeichenalgorithmen können nun auch für Graphen eingesetzt werden, die keine DAGs sind aber nicht allzu viele Kreise beinhalten. Hierzu wird das Directed Feedback Vertex Set Problem gelöst, der erhaltene DAG gezeichnet, und abschließend werden die entfernten Knoten mit ihren Kanten mit einem vergleichsweise einfachen heuristischen greedy Algorithmus hinzugefügt.

Aktuell gibt es unter `https://pacechallenge.org/2022/` auch einen internationalen Wettbewerb zu Lösungsalgorithmen für das Directed Feedback Vertex Set Problem.