Technische Universität Wien
Institut für Computergraphik und Algorithmen
Arbeitsbereich für Algorithmen und Komplexität

186.866 Algorithmen und Datenstrukturen VU

Programmieraufgabe E6

---

1 Übersicht

Ihre Aufgabe ist es den **Branch-and-Bound Algorithmus für Vertex Cover** zu implementieren. Das benötigte Framework ("AlgoDat_E6.zip") steht im TUWEL zur Verfügung. Implementieren Sie die Methode

- `Set<Integer> minVertexCover(ADGraph G)`

in der Datei "src/exercise/E6.java".

Das Framework erwartet, dass `minVertexCover` ein Set von Knoten zurück gibt, welches ein minimales Vertex Cover des ungerichteten Graphen G darstellt. Verwenden Sie sinnvolle Heuristiken für obere und untere Schranke, wie in der Vorlesung vorgestellt. Die für die Abgabe relevanten Instanzen sind zu groß, um mit trivialen Schranken gelöst zu werden.

2 Implementierung

Die Datei "E6.java" enthält die Methodensignatur für ein minimales Vertex Cover. Die Rückgabe ist ein Set mit Vertex Ids. Ein solches Set können Sie zum Beispiel mit

- `Set<Integer> var = new HashSet<>()`

anlegen. Diese Datenstruktur unterstützt Einfügen und Suchen in $O(1)$. Weitere Informationen zum "Set"-Interface finden Sie in der Dokumentation der Java-Standardbibliothek

- `https://docs.oracle.com/javase/8/docs/api/java/util/Set.html`.

Wahrscheinlich müssen Sie in Ihrer Implementierung eine Kopie einer Datenstruktur anlegen. Dies sollten Sie in der folgenden Form machen, ansonsten wird keine "richtige" Kopie angelegt:

- `Set<Integer> copy = new HashSet<>(oldSet)`

Beachten Sie: Die Klasse "ADGraph" beinhaltet keine Löschoperationen. Dies ist Absicht, da echtes Löschen und Kopieren im Graphen zu aufwendig ist. Speichern Sie daher die Zwischenlösungen in Sets oder Arrays. Es kann helfen, wenn Sie sich hierfür eine kleine Hilfsklasse schreiben, mit deren Instanzen Sie Zwischenlösungen repräsentieren. Sie haben aber volle Freiheit, wie Sie die Daten speichern.

Bedenken Sie dabei aber, dass Sie nur die eine Java-Datei "E6.java" abgeben können. Weiter Klassen müssen sich daher in dieser Datei befinden (Stichwort "inner class").

Der Graph wird als Objekt der Klasse "ADGraph" repräsentiert. Auf die Knoten/Kanten können Sie über die Knoten/Kanten-Ids zugreifen, welche als `int` gespeichert sind. Die Ids der Knoten/Kanten laufen von 0 (inklusive) bis `numVertices()` (exklusive) bzw. `numEdges()` (exklusive).

Die Klasse "ADGraph" beinhaltet die folgenden, für Sie relevanten Methoden. Dies bedeutet **nicht**, dass Sie alle diese Funktionen verwenden müssen, um die Aufgabe erfolgreich zu lösen!

- Die folgenden Methoden haben konstante Laufzeit:
  - `int degree(int v)`
    Gibt den Grad von v zurück.
  - `int source(int e)`
    Sei e eine Kante mit Knoten (u, v), diese Methode gibt u zurück.
  - `int target(int e)`
    Sei e eine Kante mit Knoten (u, v), diese Methode gibt v zurück.
  - `int numVertices()`
    Gibt die Anzahl Knoten im Graphen zurück.
  - `int numEdges()`
    Gibt die Anzahl Kanten im Graphen zurück.

- Diese Methoden haben schlimmstenfalls lineare Laufzeit in der Größe des Graphen:
  - `ArrayList<Integer> neighbors(int v)`
    Gibt die Nachbarn von v zurück.
  - `ArrayList<Integer> incidentEdges(int v)`
    Gibt alle zu v inzidenten Kanten zurück.
- Diese Methoden sind **ausschließlich zum Debuggen** gedacht:
  - `Set<Integer> getNonCoveredEdges(Set<Integer> vertices)`
    Gibt das Set von Kanten zurück, die nicht von den Knoten in `vertices` überdeckt werden.
  - `Set<String> edgesToString(Set<Integer> edges)`
    Gibt für jede Kante $e = (u, v)$ in `edges` eine String Repräsentation der Form "(u, v)" zurück.

3 Auswertung

Wenn Sie die Datei "AlgoDat_E6.java" ausführen, können Sie, wie im Tutorial beschrieben, in der Konsole eine Testinstanz auswählen. Es wird eine *.csv-Datei* im *solutions* Ordner erstellt, welche Sie mit der Datei "E6/Evaluation/plot.html" auswerten können. Weiter wird pro Graph eine SVG Datei erstellt, die den Graphen und das von Ihnen berechnete Vertex Cover visualisiert.

Diese Dateien werden auch geschrieben, wenn eine falsche Lösung berechnet wird. In diesem Fall sind im SVG natürlich entsprechend eine inkorrekte Auswahl an Knoten markiert.

Für die Abgabe lassen Sie bitte die Graphen aus der Datei "rome_ad_29.csv" durchlaufen. Dies sind Graphen aus einem Standardbenchmark mit bis zu 29 Knoten. Die anderen Dateien sind zum Debuggen gedacht. Die Datei "rome_ad_33.csv" beinhaltet Graphen mit bis zu 33 Knoten. Wenn es Ihnen gelingt eine Lösung für "rome_ad_33.csv" zu berechnen, können Sie alternativ auch den Plot für diese Graphen in Ihre Lösung kopieren. Die Graphen aus der Datei "test.csv" sind drei relativ kleine, zufällige Graphen, sowie ein Graph mit genau einer Kante und ein Dreieck.

Eine naive, dem Pseudocode der Vorlesung nachempfundene Lösung, benötigt wenige Sekunden für die Instanzen in der Datei "rome_ad_19.csv" und wenige Minute für die Instanzen in der Datei "rome_ad_29.csv". Für die Instanzen in "rome_ad_33.csv" können durchaus bis zu 10 Minuten anfallen.

Beantworten Sie für die Abgabe folgende Fragen:

- Wie haben Sie die Datenstruktur zum Verwalten der Instanzen implementiert?
- Erklären Sie grob Ihre Implementierung für das minimale Vertex Cover. Welche Probleme ergaben sich bei der Umsetzung von Pseudocode zu realem Javacode?
- Betrachten Sie den Plot. Passt das Wachstum zur theoretischen Laufzeitvorhersage? Begründen Sie Ihre Antwort.

Hinweis: Bevor Sie beginnen, lesen Sie sich unbedingt das Tutorial zu den Programmieraufgaben durch („Tutorial.pdf"). Dieses finden Sie im TUWEL. Sie bietet eine kurze Einführung in das verwendete Framework.