Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.813 Algorithmen und Datenstrukturen 1 VU 6.0

Sommersemester 2016

Programmieraufgabe

abzugeben bis: Freitag, 20. Mai 2016, 15:00

Organisatorisches

Im Rahmen der Lehrveranstaltung **Algorithmen und Datenstrukturen 1** gilt es, eine Programmieraufgabe selbstständig zu lösen, die das Verständnis des im Vorlesungsteil vorgetragenen Stoffes vertiefen soll. Als Programmiersprache wird Java verwendet.

Geben Sie bitte Ihr fertiges, gut getestetes und selbst geschriebenes Programm bis spätestens **Freitag, 20. Mai 2016, 15:00** über das Abgabesystem in TUWEL ab. Der von Ihnen abgegebene Code wird vom System automatisch getestet, und Sie erhalten eine entsprechende Rückmeldung im Abgabesystem. Sie bekommen vier Punkte, wenn Ihr Programm **alle Testinstanzen** korrekt abarbeitet. Die restlichen Punkte werden im Abgabegespräch vergeben.

Abgabefrist

Sie haben bis Freitag, 20. Mai 2016, 15:00 die Möglichkeit, ein Programm abzugeben, das alle Testinstanzen korrekt abarbeitet. Danach werden keine weiteren Abgaben akzeptiert. Bitte achten Sie daher darauf, möglichst früh mit der Aufgabe zu beginnen und die Aufgabe auch möglichst früh nach dem Freischalten des Servers (ca. zwei Wochen vor der Deadline) abzugeben.

Abgabegespräche

Die Abgabegespräche finden, nach erfolgreicher elektronischer Abgabe, in der Zeit von Montag, 23. Mai bis Mittwoch, 25. Mai 2016 statt. Dazu vereinbaren Sie in TUWEL einen individuellen Gesprächstermin, bei dem Sie sich mit einer/m unserer TutorInnen im Informatiklabor treffen und den von Ihnen eingereichten Programmcode erklären. Sie können sich dazu von Donnerstag, 19. Mai, bis Sonntag, 22. Mai 2016, 23:59 in TUWEL bei einer/m TutorIn anmelden.

Falls Sie Datenstrukturen aus dem Java-API in Ihrem Programm verwenden, sollten Sie auch darüber **genau** Bescheid wissen. Je nach Funktionstüchtigkeit Ihres Programms, Innovation und Effizienz Ihrer Implementierung sowie der Qualität des Abgabegesprächs können Sie bis zu 14 Punkte für diese Programmieraufgabe erhalten. Voraussetzung für eine positive Absolvierung des Abgabegesprächs ist jedenfalls das Verständnis des zugrunde liegenden Stoffes und der eigenen Implementierung. Kann die Implementierung nicht erklärt werden, dann verfallen auch die 4 Punkte für eine korrekte Abgabe.

Aufgabenstellung

Sie sind Teil eines Teams, das eine Software für die Verwaltung von Daten eines Kommunikationsnetzwerks erstellen soll. Neben dem Sammeln und Aufbereiten von Daten aus externen Quellen müssen in diesem Programm Abfragen zur Netzwerkinfrastruktur implementiert werden. Dazu müssen die Daten in einer geeigneten Datenstruktur abgelegt werden und darauf die Abfragen effizient ermöglicht werden.

Beschreibung

Das Netzwerk wird vereinfacht als Sammlung von Netzwerkknoten und den Verbindungen zwischen diesen Netzwerkknoten gesehen. Dabei gilt:

- Es wird am Anfang jedes Tests einer Instanz die Anzahl n der Knoten festgelegt. Diese Anzahl bleibt immer gleich, d.h. es werden keine neuen Knoten hinzugefügt bzw. gelöscht. Es wird angenommen, dass $n > 0$ ist.
- Die Netzwerkknoten werden von 0 beginnend bis $n - 1$ durchnummeriert.
- Zwischen zwei Netzwerkknoten kann eine Verbindung bestehen, über die Daten ausgetauscht werden. Die Daten können in beide Richtungen fließen. Für die Lösung der Aufgaben ist die Richtung nicht von Bedeutung.
- Ein Knoten hat keine Verbindung zu sich selbst (Schleife).
- Die erwarteten Netzwerke sind dünn besetzt.

Ihre Aufgabe besteht aus zwei Teilen. Einerseits müssen Sie eine passende Datenstruktur für das Speichern der Daten implementieren. Andererseits müssen Sie folgende Operationen auf dieser Datenstruktur unterstützen:

- `int numberOfNodes()`: Liefert die Anzahl der Knoten zurück.
- `int numberOfConnections()`: Liefert die Anzahl der Verbindungen zurück.
- `void addConnection(int v, int w)`: Fügt eine Verbindung im Netzwerk zwischen den Knoten `v` und `w` ein. Ist diese Verbindung schon vorhanden, dann passiert nichts, d.h. die Verbindung bleibt im Netzwerk erhalten.

- `void addAllConnections(int v)`: Fügt Verbindungen von einem bestimmten Knoten `v` zu allen anderen Knoten ein. Hatte der Knoten schon Verbindungen, dann bleiben diese erhalten.

- `void deleteConnection(int v, int w)`: Entfernt eine Verbindung zwischen den Knoten `v` und `w` aus dem Netzwerk. Ist die Verbindung nicht vorhanden, dann passiert nichts.

- `void deleteAllConnections(int v)`: Entfernt alle Verbindungen für einen bestimmten Knoten `v` aus dem Netzwerk. Hatte der Knoten noch keine Verbindungen, dann passiert nichts.

- `int numberOfComponents()`: Liefert die Anzahl der Zusammenhangskomponenten im Netzwerk zurück.

- `boolean hasCycle()`: Überprüft, ob das Netzwerk einen Kreis enthält. Wenn dies der Fall ist, wird `true` zurückgeliefert, ansonsten `false`.

- `int minimalNumberOfConnections(int start, int end)`: Liefert die kleinste Anzahl an Verbindungen, die durchlaufen werden muss, um von einem Startknoten `start` zu einem Endknoten `end` zu gelangen. Sind `start` und `end` gleich, dann soll 0 zurückgeliefert werden. Sind `start` und `end` nicht über einen Pfad miteinander verbunden, dann wird -1 zurückgeliefert.

- `List<Integer> criticalNodes()`: Liefert eine Liste jener Knoten zurück, die als kritisch eingestuft werden. Ein Knoten ist kritisch, wenn das Entfernen aller Verbindungen zu diesem Knoten nicht nur diesen Knoten isoliert, sondern auch seine ursprüngliche Zusammenhangskomponente in drei oder mehr Zusammenhangskomponenten zerfallen lässt.

Bitte beachten Sie noch folgende Hinweise zur Implementierung:

- Sie dürfen für die Datenhaltung (Speicherung der Knoten und Verbindungen) Java-Datenstrukturen verwenden.

- Alle oben angeführten Methoden müssen von Ihnen selbständig implementiert werden. Die Funktionalität darf **nicht** durch Aufruf von Methoden in Bibliotheken realisiert werden.

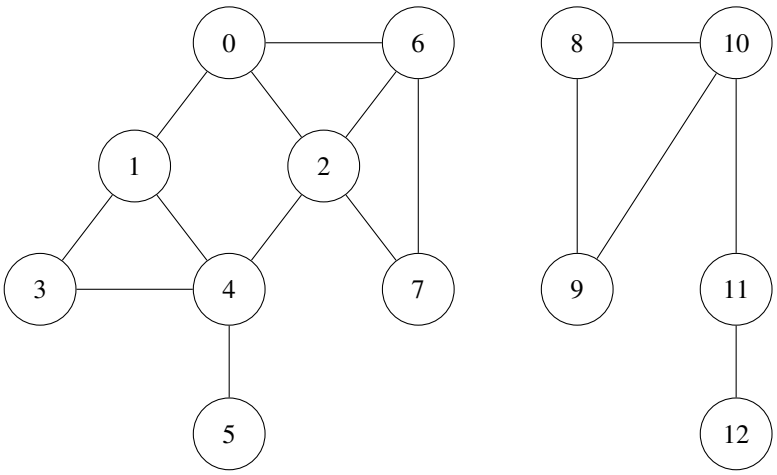
- Sie dürfen weitere Hilfsmethoden implementieren, der Testcode wird aber nur die oben genannten Methoden aufrufen und auf Korrektheit testen.

- Sie dürfen selbst entscheiden, ob Sie Ihre Methoden rekursiv oder iterativ implementieren.

- Achten Sie generell auf eine möglichst effiziente Implementierung (bezüglich Laufzeit und zusätzlich erforderlichen Speicherplatzes).

Beispiel

Es sei folgendes Netzwerk gegeben:



Es ergeben sich dann z.B. folgende Rückgabewerte:

- `numberOfNodes():13`
- `numberOfConnections():16`
- `numberOfComponents():2`
- `hasCycle():true`
- `minimalNumberOfConnections(0, 5):3`
- `minimalNumberOfConnections(0, 9):-1`
- `criticalNodes():[4, 11, 10]`

Codegerüst

Der Algorithmus ist in Java zu implementieren. Das Testsystem unterstützt Java 8. Um Ihnen einerseits das Lösen der Aufgabenstellung zu erleichtern und andererseits automatisches Testen mit entsprechender Rückmeldung an Sie zu ermöglichen, stellen wir Ihnen ein Codegerüst in TUWEL zur Verfügung.

Das Codegerüst besteht aus mehreren Klassen, wobei Sie Ihren Code in die Datei `Network.java` einfügen müssen.

Weiters beachten Sie bitte, dass alle Änderungen im Code außerhalb von `Network.java` nicht von unserem Abgabesystem berücksichtigt werden.

Hinweise
`Tester.printDebug(String msg)` kann verwendet werden um Debuginformationen auszugeben. Die Ausgabe erfolgt nur, wenn beim Aufrufen das Debugflag (`-d`) gesetzt wurde. Weitere Erläuterungen zur Ausgabe finden Sie im Abschnitt Testdaten. Sie müssen diese Ausgaben vor der Abgabe **nicht** entfernen, da das Testsystem beim Kontrollieren Ihrer Implementierung diese Flags nicht setzt. Das Framework nutzt Methoden, die das Ausführen von *sicherheitsbedenklichem* Code auf dem Abgabesystem verhindern soll. Werden trotzdem solche Programmteile abgegeben oder sollte versucht werden, das Sicherheitssystem zu umgehen, wird das als Betrugsversuch gewertet. Die minimale Konsequenz dafür ist ein negatives Zeugnis auf diese Lehrveranstaltung.

Rückgabe des Frameworks
Ist Ihre Lösung korrekt, gibt das Framework `OK` zurück. Andernfalls bekommen Sie eine Fehlermeldung mit einem Hinweis zur Ursache. Sollte beim Test auf dem Server keine Ursache angegeben werden, dann wurde der entsprechende Test abgebrochen. Ein Grund dafür kann ein zu großer Speicherverbrauch sein.

Testdaten

In TUWEL werden auch Testdaten veröffentlicht, die es Ihnen erleichtern sollen, Ihre Implementierung zu testen. Verarbeitet Ihr Programm diese Daten korrekt, heißt das aber nicht zwangsläufig, dass Ihr Programm alle Eingaben korrekt behandelt. Testen Sie daher auch mit zusätzlichen, selbst erstellten Daten.

Eine Testdatei muss bzw. kann folgende Befehlszeilen beinhalten:

- In der ersten Zeile muss mit `nodes n` die Anzahl n der Knoten im Netzwerk spezifiziert werden.
- Mit `add` wird dem Testprogramm mitgeteilt, dass alle nachfolgenden Zeilen (bis zu einem weiteren Befehl), die nur Zahlen beinhalten, hinzuzufügende Verbindungen spezifizieren.
- Mit `delete` wird dem Testprogramm mitgeteilt, dass alle nachfolgenden Zeilen bis zu einem weiteren Befehl, die nur Zahlen beinhalten, zu löschende Verbindungen spezifizieren.
- Enthält eine Zeile nur Zahlen, dann wird sie folgendermaßen verarbeitet:
  - Enthält die Zeile nur eine Zahl, dann wird bei `add` von diesem Knoten zu allen anderen Knoten eine Verbindung hergestellt. Bei `delete` werden alle vorhandenen Verbindungen gelöscht und der Knoten ist somit isoliert (eine eigene Zusammenhangskomponente).
  - Enthält eine Zeile mehr Zahlen, dann gibt die erste Zahl den Startknoten an und bei `add` wird von diesem Startknoten zu allen weiter angeführten Knoten jeweils eine Verbindung hergestellt. Bei `delete` werden die entsprechenden Verbindungen gelöscht.

- `structure` veranlasst das Testprogramm einen Strukturtest durchzuführen. Dazu müssen durch Leerzeichen getrennt drei Werte angegeben werden:
  - Die erwartete Anzahl aller Verbindungen zwischen den Knoten.
  - Die erwartete Anzahl an Zusammenhangskomponenten.
  - Das erwartete Ergebnis für den Test, ob das Netzwerk zumindest einen Kreis enthält.
- `distance` veranlasst den Tester die Implementierung für die Distanzbestimmung zwischen zwei Knoten zu testen. Dazu können durch Leerzeichen getrennt Tripel der Form `(x,y,z)` angegeben werden. Dabei wird überprüft, ob die Implementierung zwischen x und y die Distanz z ermittelt.
- `critical` veranlasst das Testprogramm, die Implementierung von `criticalNodes` zu überprüfen. Dazu können durch Leerzeichen getrennt, die erwarteten kritischen Knoten angegeben werden. Wird eine leere Liste erwartet, dann wird nur der Befehl angegeben.

Bitte beachten Sie, dass das Testprogramm nur ein einfaches Matching durchführt und die Parameter direkt umwandelt. Fehler im Aufbau bzw. beim Format führen zu Ausnahmen und zum Beenden des Testprogramms. Sie können davon ausgehen, dass **alle** von uns erstellten Testinstanzen dieser Spezifikation entsprechen, und müssen selbst keine Prüfung auf Syntaxfehler durchführen.

Abgabe

Die Abgabe Ihrer Implementierung der Klasse `Network` erfolgt über TUWEL. Bedenken Sie bitte, dass Sie maximal 30-mal abgeben können und ausschließlich die jeweils **letzte Abgabe** bewertet wird. Prinzipiell sollte es nicht nötig sein, mehr als eine Abgabe zu tätigen, wenn Sie Ihr Programm entsprechend getestet haben.

Hinweis

Verwenden Sie bei Ihrer Implementierung unter keinen Umständen Sonderzeichen, wie zum Beispiel Umlaute (ä, ö, ü, Ä, Ö, Ü) oder das Zeichen „ß“. Dies kann sonst – bei unterschiedlichen Zeichensätzen am Entwicklungs- und Abgabesystem – zu unvorhersehbaren Problemen und Fehlern führen, was schlussendlich auch als fehlerhafter Abgabeversuch gewertet wird.

Die Überprüfung Ihres abgegebenen Codes erfolgt automatisch, wobei in drei Schritten getestet wird:

1. Kompilation: Es wird der **Bytecode** erzeugt, der beim anschließenden Testen verwendet wird.

2. Veröffentlichte Testdaten: Bei diesem Schritt wird Ihr Programm mit den auf der Webseite veröffentlichten Daten getestet.
3. Unveröffentlichte Testdaten: Abschließend wird Ihr Programm noch mit Ihnen nicht bekannten aber den Spezifikationen entsprechenden Eingabedaten ausgeführt.

Nach Beendigung der Tests, die direkt nach Ihrer Abgabe gestartet werden, erhalten Sie eine Rückmeldung in TUWEL mit entsprechenden kurzen Erfolgs- bzw. Fehlermeldungen.

Abgabe

Aufgrund der großen Hörerzahl kann es zu Verzögerungen beim Verarbeiten Ihrer Abgaben kommen. Geben Sie daher Ihre Lösung nicht erst in den letzten Stunden ab, sondern versuchen Sie, **rechtzeitig** die Aufgabenstellung zu lösen.

Beachten Sie bitte auch, dass wir Ihren Code mit anderen abgegebenen Programmen automatisch vergleichen werden, um Plagiate zu erkennen. **Geben Sie daher nur selbst implementierte Lösungen ab!** Gruppenabgaben bzw. mehrfache Abgaben desselben Programms unter verschiedenen Namen werden nicht akzeptiert. **Wenn Sie das abgegebene Programm nicht selbst programmiert haben, erhalten Sie ein negatives Zeugnis.** Auch der eigentliche Entwickler kann nur mehr maximal vier Punkte auf dieses Programmierbeispiel erhalten.

Abgabegespräch

Beim Abgabegespräch müssen Sie folgende Fragen beantworten können:

- Welche Datenstruktur wurde gewählt und warum?
- Wie wurden die einzelnen Operationen implementiert? Sie sollten auch ad-hoc für kleine Beispiele den Ablauf einer Operation **genau** erklären können. Das gilt insbesondere für Implementierungen, die von den Beschreibungen in den Folien abweichen bzw. auf Algorithmen aus anderen Quellen aufbauen oder komplett auf Basis eigener Ideen erstellt wurden.
- Welche Laufzeit haben die Operationen, die von Ihnen implementiert wurden (mit entsprechender Komplexitätsabschätzung)?

Abgabesystem

Unser Abgabesystem mit Intel Xeon X5650 CPU ruft Ihr Programm mit folgendem Kommandozeilenbefehl auf:

```
java ad1.ss16.pa.Tester input > output
```

wobei `input` eine Eingabedatei darstellt. Die Ausgabe wird in die Datei `output` gespeichert. Pro Testinstanz darf eine maximale Ausführungszeit von 20 Sekunden nicht überschritten werden, anderenfalls wird die Ausführung abgebrochen.

Bitte beachten Sie, dass unser Abgabesystem **kein** Debugging-Tool ist!

Bewertung

Abschließend seien nochmals die wesentlichen Punkte zusammengefasst, die in die Bewertung Ihrer Abgabe einfließen und schließlich über die Anzahl der Punkte für diese Programmieraufgabe entscheiden:

- **Alle** Instanzen korrekt gelöst
- Laufzeiteffizienz
- Speicherverbrauch
- Antworten beim Abgabegespräch

Werden nicht alle Instanzen korrekt gelöst (z.B. wegen Fehler bzw. Timeout) oder kann die Implementierung nicht erklärt werden, dann gibt es 0 Punkte auf die Programmieraufgabe und damit ein negatives Zeugnis auf diesen Kurs.

Zusätzliche Informationen

Lesen Sie bitte auch die in TUWEL veröffentlichten Hinweise. Wenn Sie Fragen oder Probleme haben, wenden Sie sich rechtzeitig an die AlgoDat1-Hotline unter `algodat1-ss16@ac.tuwien.ac.at` oder posten Sie Ihre Frage im TUWEL-Diskussionsforum.