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

186.172 Algorithmen und Datenstrukturen 1 VL 4.0

Wintersemester 2008

Programmieraufgabe 2

abzugeben bis: 19. Jänner 2009, 15:00 Uhr

Organisatorisches

Im Rahmen der Lehrveranstaltung *Algorithmen und Datenstrukturen 1* gilt es, zwei voneinander unabhängige Programmieraufgaben selbständig zu lösen, die das Verständnis des im Vorlesungsteil vorgetragenen Stoffes vertiefen sollen.

Geben Sie bitte Ihr fertiges, gut getestetes und selbst geschriebenes Programm bis spätestens *Dienstag, 19. Jänner 2009, 15:00 Uhr*, mittels des eigens dafür eingerichteten Abgabesystems auf der Webseite zur Lehrveranstaltung ab. Der von Ihnen abgegebene Code wird vom System automatisch getestet und Sie erhalten per Mail eine entsprechende Abgabebestätigung.

In der Zeit von Mittwoch, den 21. Jänner 2009, bis Freitag, den 23. Jänner 2009, finden für alle LVA-Teilnehmer Abgabegespräche statt. Verwenden Sie bitte das Abgabesystem, um mit Ihrem Tutor beziehungsweise Ihrer Tutorin einen individuellen Gesprächstermin zu vereinbaren, bei dem Sie den von Ihnen eingereichten Programmcode erklären können müssen. Sobald Sie ein funktionstüchtiges Programm abgegeben haben, spätestens aber nach Ablauf der Abgabefrist am 19. Jänner, ist für Sie die Anmeldung zu einem Abgabegespräch freigeschaltet. Falls Sie Systemroutinen verwenden (z.B. das Auffinden eines Minimums), so sollten Sie auch über die Funktionsweise dieser Methoden *genau* Bescheid wissen. Je nach Funktionstüchtigkeit Ihres Programms, Innovation und Effizienz Ihrer Implementierung und dem Verständnis des benötigten Stoffes beziehungsweise der Qualität des Abgabegesprächs können Sie 0 bis 10 Punkte für diese Programmieraufgabe erhalten.

Um auftretende Verständnisfragen beziehungsweise Unklarheiten bei der Angabe klären zu können, stehen Ihnen in der Woche von Montag, den 12. Jänner 2009, bis Freitag, den 16. Jänner 2009, unsere Tutorinnen und Tutoren im Informatiklabor für persönliche Gespräche zur Verfügung. Die genauen Zeiten entnehmen Sie bitte der LVA-Webseite. Bedenken Sie allerdings, dass dies nur eine Hilfestellung ist und eine selbstständige Auseinandersetzung mit der Problemstellung nicht ersetzen soll und kann – die Tutoren sind nicht als „menschliche Debugger“ anzusehen.

Bitte beachten Sie, dass wir *ausnahmslos* keine verspäteten Abgaben akzeptieren. Sollten Sie kein Programm abgegeben haben oder das von Ihnen zuletzt abgegebene Programm ist nicht

in der Lage, alle Instanzen korrekt abzuarbeiten, haben Sie die Möglichkeit, beim Abgabegespräch eine korrekte Version Ihres Codes mitzubringen, können aber dann nur noch maximal 5 Punkte erreichen. Zusätzlich liegt es in Ihrem Verantwortungsbereich, dass der Sourcecode bei diesem Gespräch verfügbar und im Informatiklabor kompilierbar ist und somit die Lauffähigkeit Ihres Programms von Ihrem Tutor entsprechend getestet werden kann.

Um eine positive Note zu erhalten, müssen Sie bei jeder der zwei Programmieraufgaben mindestens einen Punkt erreichen. Ebenso müssen Sie Ihren Termin zum Abgabegespräch einhalten – anderenfalls ist es uns nicht möglich, ein positives Zeugnis auszustellen.

Gruppenabgaben bzw. mehrfache Abgaben des selben Programms unter verschiedenen Namen werden nicht akzeptiert und führen für alle Beteiligten (auch für den eigentlichen Entwickler) dazu, dass Sie die LVA nicht positiv abschließen können!

Aufgabenstellung

Nach der Meldung eines Überfalls will die Polizei mit möglichst wenig Personalaufwand das umliegende Straßennetz *vollständig* überwachen. Die Polizisten werden auf Kreuzungen positioniert, wodurch sie gleichzeitig alle angrenzenden Straßenabschnitte beobachten können. Um einen Straßenabschnitt zu überwachen, muss mindestens an einem der beiden Enden des Abschnitts ein Polizist positioniert werden.

Von dieser Motivation leitet sich folgende Problemstellung ab: Das entsprechende Straßennetz liegt als *schlichter* ungerichteter Graph vor, d.h. es gibt keine Kanten, deren Endpunkte ein und derselbe Knoten ist (Schleifen), und keine Mehrfachkanten zwischen zwei Knoten.

Straßenkreuzungen werden in diesem Graph durch Knoten und Straßenabschnitte durch Kanten repräsentiert. Es soll nun eine *möglichst kleine Anzahl* an Kreuzungen bzw. Knoten ausgewählt werden, an denen ein Polizist positioniert werden muss. In jedem Fall müssen aber alle Straßenabschnitte von zumindest einem Beamten überwacht werden.

Hinweis zum graphentheoretischen Modell: Die Modellierung des Straßennetzes als Graph wird im Folgenden noch detaillierter beschrieben:

- Ein Straßenabschnitt wird als geradlinig angesehen, d.h. es ist möglich, dass ein Polizist an einem Ende den gesamten Abschnitt unabhängig von dessen Länge überwachen kann.
- Wenn Straßen (z.B. durch Baustellen) gesperrt sind, kann es vorkommen, dass eine „Kreuzung“ gar kein angrenzendes Straßenstück mehr hat (Knoten mit Grad 0). Hier gibt es nichts zu überwachen, es muss daher auch kein Polizist auf dieser Kreuzung platziert werden.
- Das Ende einer Sackgasse wird im Graph auch als Kreuzung dargestellt und kann deshalb auch als Posten für einen Beamten genutzt werden (Knoten mit Grad 1).
- Es können „Kreuzungen“ existieren, die nur von einer Straße durchquert werden, wobei es sich hier um zwei getrennte Straßenabschnitte handelt (Knoten mit Grad 2).

- Das Straßennetz muss nicht zwingend zusammenhängend sein (die Zusammenhangskomponenten könnten z.B. durch U-Bahn-Verbindungen oder nicht befestigte Straßen verbunden sein, die im Modell nicht erfasst werden), es muss jedoch das komplette Straßennetz überwacht werden!
- Der Graph muss nicht zwingend planar sein, d.h. es kann sein, dass sich der Graph in der Ebene nicht kreuzungsfrei zeichnen lässt. Das bedeutet aber nicht, dass sich in dem Kreuzungspunkt der beiden Kanten auch ein Knoten bzw. eine Kreuzung in unserem Sinn befindet. Als Entsprechung in der Realität könnte man sich Brücken, Tunnels oder ähnliches vorstellen.

Sie sollen nun eine *Greedy Heuristik* implementieren, die für beliebige schlichte Graphen eine möglichst gute und gültige Lösung für diese Problemstellung liefert.

Hinweis zur Komplexität: Dieses Problem ist *nicht* in polynomieller Zeit exakt lösbar, es handelt sich hier um ein $\mathcal{NP}$-schwieriges Problem. Aus diesem Grund sollen Sie auch eine *Heuristik* implementieren, d.h. eine Methode, die eine gültige Näherungslösung produziert. Sämtliche exakten Enumerationsverfahren würden bei der Abgabe das Zeitlimit erheblich überschreiten und somit auch zu einer negativen Abgabe führen. Ihr Programm sollte auch im Worst-Case eine Laufzeit von $O(n^2)$ nicht überschreiten ($n = |V|$, die Anzahl der Knoten im Graphen).

Beim Abgabegespräch müssen Sie Ihre Überlegungen zu folgenden Punkten präsentieren können:

- Warum liefert Ihre Heuristik eine gute Lösung?
- Wodurch stellen Sie sicher, dass Ihre Heuristik immer eine gültige Lösung liefert?
- Beschreiben Sie die Laufzeit Ihres Algorithmus in O-Notation in Abhängigkeit von der Anzahl der Knoten und Kanten im Instanzgraphen.
- Für welche Instanzen läuft Ihr Programm besonders schnell bzw. langsam (Best-Case, Worst-Case)?
- Für welche Instanzen liefert Ihr Programm eine sehr gute bzw. schlechte Lösung?

Codegerüst

Der Algorithmus ist in Java 6 zu implementieren. 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 zur Verfügung[1], welches alle notwendigen Methoden enthält, um auf den zugrundeliegenden Graphen zugreifen zu können. Dadurch sind allerdings auch Sie dazu angehalten, bestimmte Regeln zu befolgen,

[1] `http://www.ads.tuwien.ac.at/teaching/lva/186172.html#Programmieraufgaben`

um ein funktionierendes Programm (im Sinne der automatischen Überprüfung) zu schreiben. Neben diesen Einschränkungen nehmen wir für dieses Programmierbeispiel an, dass die Eingabedaten den Spezifikationen (siehe weiter unten) entsprechen.

Das Codegerüst besteht aus sechs Dateien, wobei Sie Ihren Code in die Datei `SecurityPlannerImpl.java` einfügen müssen. Um Debugging-Informationen auszugeben, steht die Methode `printDebug()` zur Verfügung. Die Debug-Ausgaben werden in `.dot`-Dateien gespeichert, welche Inputdateien für die GraphViz-Bibliothek darstellen[2]. Für genauere Informationen über die Debugging-Ausgaben lesen Sie bitte in der README des Codegerüstes nach. Vergessen Sie nicht, diese Aufrufe vor der Abgabe zu entfernen! Weitere Details, die das Codegerüst betreffen, entnehmen Sie bitte der LVA-Webseite[3]. Dort befindet sich auch der Link zur Javadoc des Codegerüstes.

Hinweis: Nicht jede Methode oder Membervariable, die vom Codegerüst zur Verfügung gestellt wird, ist zwingend für Ihre Implementierung notwendig.

Folgende Kriterien haben Einfluss auf die Bewertung Ihrer Abgabe:

- Effizienz des Algorithmus
- Speicherverbrauch
- Rechtfertigung des Lösungsweges
- Antworten auf die theoretischen Fragen der Aufgabenstellung

Testdaten

Auf der Webseite zur LVA sind 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 fehlerfrei behandelt. Testen Sie daher auch durchaus mit zusätzlichen, selbst erstellten Daten.

Die Datei einer Testinstanz ist folgendermaßen spezifiziert: Die erste Zeile besteht aus der oberen Schranke für die Anzahl der benötigten Polizisten, mit der Ihr Algorithmus noch eine positive Bewertung vom Abgabesystem erhält. Da es sich hier um ein Minimierungsproblem handelt, muss Ihre Heuristik somit eine gültige Lösung produzieren, die diesem oder einem geringerem Wert entspricht.

Die nachfolgenden Zeilen dienen der Codierung des Graphen: die zweite Zeile enthält die Anzahl der Knoten und die nachfolgenden Zeilen beschreiben jeweils eine Kante mit ihren beiden Endknoten.

[2] `http://www.graphviz.org/`, Bibliothek zur Visualisierung von Graphen (z.B. zum Debugging)
[3] `http://www.ads.tuwien.ac.at/teaching/lva/186172.html`

Abgabe

Die Abgabe erfolgt über das eigens eingerichtete Abgabesystem auf der Webseite zur LVA. Bedenken Sie bitte, dass Sie maximal 10 Mal abgeben können und dass 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. Verwenden Sie dazu neben den von uns zur Verfügung gestellten Testdaten (inklusive Lösung) auch selbst erstellte. Sollte die Abgabe dennoch nicht erfolgreich sein – Sie erhalten eine entsprechende Mail von uns – testen und debuggen Sie vor einer neuerlichen Abgabe bitte nochmals ausführlich.

Geben Sie alle von Ihnen geschriebenen Sourcedateien ab. Erstellen Sie eine ZIP-Datei (.zip, kein .gz, .tgz, tar.gz, etc.), die den gesamten von Ihnen geschriebenen Code beinhaltet – auch wenn Ihr gesamter Code in nur einer Datei enthalten ist. Alle von Ihnen geschriebenen Klassen müssen im Paket `ads1ws08.pa2` liegen. Nehmen Sie keine Änderungen an den Dateien `Main.java`, `Graph.java`, `SecurityPlanner.java`, `SecurityPlannerIO.java` und `Solution.java` vor und geben Sie diese daher auch *nicht* ab, da sie vom Abgabesystem ignoriert werden. Vergessen Sie nicht, dass Ihr abgegebenes Programm keine Debugging-Information ausgeben darf! Stellen Sie weiters bitte sicher, dass Ihr Programmarchiv nur `.java` Dateien enthält – alle anderen Dateien werden vom System automatisch gelöscht.

Hinweis: Verwenden Sie bei ihrer Implementation unter keinen Umständen Umlaute (ä,ö,ü,Ä,Ö,Ü) sowie das Zeichen „ß“ außerhalb von Kommentaren. Dies kann sonst – bei unterschiedlichen Zeichensätzen am Entwicklungs- und Abgabesystem – zu unvorhersehbaren Problemen und Fehlern führen, was schlussendlich auch als fehlerhafter Abgabeversuch gewertet wird. Sollten Sie verspätet abgeben und ihre Abgabedatei im Informatiklabor aus den oben genannten Gründen nicht kompilierbar sein, so erhalten Sie automatisch keine Punkte auf das Programmierbeispiel.

An alle Benutzer von Betriebssystemen aus dem Hause Apple: Bitte beachten Sie, dass die systeminterne `zip`-Funktion keine herkömmlichen Zip-Dateien erzeugt. Es werden standardmäßig zusätzlich zu den zu packenden Dateien weitere in das Archiv hinzugefügt, deren Dateinamen nur aus Sonderzeichen – aber mit der Endung `.java` – bestehen. Stellen Sie bitte sicher, dass diese Dateien nicht in Ihrem Zip-Archiv enthalten sind. Anderfalls kommt es zu einer negativen Bewertung Ihrer Programmieraufgabe, da diese zusätzlichen Dateien vom Abgabesystem nicht automatisch als ungültig erkannt werden können, aber zu Compilerfehlern führen.

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

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

veröffentlichte Testdaten Bei diesem Schritt wird Ihr Programm mit den auf der Webseite veröffentlichten Daten getestet.

unveröffentlichte Testdaten Abschließend wird Ihre Lösung noch mit Ihnen nicht bekannten, aber den Spezifikationen entsprechenden Eingabedaten ausgeführt. Die hier verwendeten Testdaten werden nach der Deadline zur Programmabgabe auf der Webseite zur LVA veröffentlicht.

Nach Beendigung der Tests, die direkt nach Ihrer Abgabe gestartet werden, erhalten Sie von uns eine e-Mail an die uns bekannte Adresse (siehe dazu auch Webseite) mit entsprechenden kurzen Erfolgs- bzw. Fehlermeldungen. Aufgrund der großen Hörerzahl kann es zu Verzögerungen beim Zusenden von Nachrichten kommen. Haben Sie daher bitte ein wenig Geduld beziehungsweise geben Sie 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 den von Ihren Kollegen abgegebenen Programmen automatisch vergleichen werden, um Plagiate zu erkennen. Geben Sie daher keine nicht selbst implementierte Lösung ab!

Testumgebung

Unser Testsystem ruft Ihr Programm mit folgendem Kommandozeilenbefehl auf:

```
java ads1ws08.pa2.Main < input > output
```

wobei `input` eine Eingabedatei darstellt. Die Ausgabe wird in die Datei `output` gespeichert. Um die Funktionsweise zu testen, wird die von Ihrem Code erzeugte Ausgabedatei mittels des Unix-Kommandos `diff` mit den Musterlösungen auf Gleichheit verglichen. Bedenken Sie bitte daher, sämtliche Debuggingausgaben vor der Programmabgabe zu entfernen.

Zusätzliche Informationen

Lesen Sie bitte auch die auf der Webseite zu dieser LVA veröffentlichten Hinweise. Sie finden im Skriptum zur Vorlesung eine Einführung zu Graphen und Optimierung.

Es besteht selbstverständlich auch die Möglichkeit, mit anderen Studenten über die Aufgabenstellung zu diskutieren beziehungsweise sich gegenseitig zu helfen. Bedenken Sie allerdings, dass eine Abgabe des Programmcodes eines Kollegen als nicht ausreichende Leistung erachtet wird. Auf alle Fälle sollten Sie das von Ihnen abgegebene Programm sehr gut erklären können.

Im Falle von Fragen oder weiteren Problemen, helfen Ihnen auch Ihr Tutor bzw. Ihre Tutorin, die Studienassistenten und die Mitarbeiter unseres Instituts – in dieser Reihenfolge – gerne weiter. Sie können sich auch an die AlgoDat1-Hotline unter `algodat1-ws08@ads.tuwien.ac.at` wenden.