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

186.815 Algorithmen und Datenstrukturen 2 VU 3.0

Sommersemester 2016

Programmieraufgabe

abzugeben bis: Montag, 13. Juni 2016, 15:00 Uhr

Organisatorisches

Im Rahmen der Lehrveranstaltung **Algorithmen und Datenstrukturen 2** gilt es eine Programmieraufgabe selbstständig zu lösen, die das Verständnis des im Vorlesungsteil vorgetragenen Stoffes vertiefen soll. Als Programmiersprache wird Java 8 verwendet.

Geben Sie bitte Ihr fertiges, gut getestetes und **selbst geschriebenes Programm** bis spätestens **Montag, 13. Juni 2016, 15:00 Uhr** über das Abgabesystem in TUWEL ab. Der von Ihnen abgegebene Code wird vom System automatisch getestet, und Sie erhalten eine entsprechende Rückmeldung im Abgabesystem.

Um eine positive Note zu erhalten, müssen Sie bei der Programmieraufgabe **mindestens 11 Punkte erreichen** und Ihren Termin zum Abgabegespräch einhalten. Gruppenabgaben sind nicht erlaubt.

Abgabefrist

Sie haben bis Montag, 13. Juni 2016, 15:00 Uhrdie Möglichkeit ein Programm abzugeben, **das alle Testinstanzen korrekt abarbeitet**. Danach werden keine weiteren Abgaben akzeptiert, d.h., sollten Sie diese Frist versäumen beziehungsweise Ihr Programm nicht alle Testinstanzen korrekt abarbeiten, bekommen Sie keine Punkte auf die Programmieraufgabe. Beginnen Sie daher mit Ihrer Arbeit **rechtzeitig** und geben Sie nicht erst in den letzten Stunden ab!

Abgabegespräche

Zwischen Dienstag, 14. Juni 2016 und Donnerstag, 16. Juni 2016, finden für alle LVA-Teilnehmer Abgabegespräche 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 können müssen. Sie können sich zu diesem Abgabegespräch **von 10. Juni 2016 bis 13. Juni 2016, 23:59 Uhr in TUWEL** bei einer/m TutorIn anmelden. Bitte halten Sie diese Deadline ein! Sollten in diesem Zeitraum keine Termine in TUWEL eingetragen oder bereits alle Termine vergeben sein, dann

kontaktieren Sie bitte die Hotline `algodat2-ss16@ac.tuwien.ac.at`. Melden Sie sich nur an, wenn Sie positiv abgegeben haben!

Je nach Funktionstüchtigkeit Ihres Programms, Innovation und Effizienz Ihrer Implementierung sowie der Qualität des Abgabegesprächs können Sie bis zu 20 Punkte für diese Programmieraufgabe erhalten. Voraussetzung für eine positive Absolvierung des Abgabegesprächs ist jedenfalls das Verständnis des zugrunde liegenden Stoffes.

Aufgabenstellung

Sie haben vor Kurzem bei einem neu gegründeten Internetprovider zu arbeiten angefangen, der mit Ihrer Hilfe seine Infrastruktur aufbauen will. Der Provider hat es sich zur Aufgabe gemacht Datenzentren für sehr sensible Daten zu bauen und diese in verschiedenen Regionen der Welt zu errichten. Ihr Auftraggeber erhofft sich durch die geographische Verteilung und die unterschiedlichen Rechtslagen in den Ländern die Daten seiner Kunden besser schützen zu können.

Eine vorläufige Analyse hat ergeben, dass eine Verteilung auf k verschiedene Regionen eine ausreichende Sicherheit gewährleistet ohne zu hohe Kosten zu verursachen. Um die Kommunikation zwischen den Datenzentren besser kontrollieren zu können, hat sich ihr Auftraggeber dafür entschieden Standleitungen zwischen den Datenzentren zu errichten bzw. zu mieten. Dies ist jedoch mit erheblichen Kosten verbunden.

Nach kurzem Überlegen wird Ihnen klar, dass eine baumartige Vernetzung der Zentren die geringste Anzahl an Leitungen benötigt. Weiters wissen Sie, dass Ihr Auftraggeber die Kosten für den Bau eines Datenzentrums an jedem Standort als ident einschätzt. Aufgrund dieser Annahmen kann nur ein Baum die geringsten Kosten verursachen, da sonst immer eine Verbindung eingespart werden könnte.

Als guter Algorithmiker erkennen Sie sofort, dass es sich um eine Generalisierung des Minimum Spanning Tree Problems (MST) handelt, welches nicht alle Knoten eines Graphen berücksichtigt, sondern nur eine zu treffende Auswahl von k Knoten. Dieses Problem ist als k-Minimum Spanning Tree Problem (k-MST) bekannt.

Sie können Ihr Problem folgendermaßen definieren:
Gegeben sei ein gewichteter Graph $G = (V, E)$ mit einer Kostenfunktion $c : E \to \mathbb{R}$ die jeder Kante Kosten zuordnet und ein zusätzlicher Parameter k. Es soll nun ein zusammenhängender Teilbaum minimalen Gewichts (Kantensumme) mit genau k Knoten aus V gefunden werden.

Sie wissen aus Algorithmen und Datenstrukturen 1, dass das MST Problem in polynomieller Zeit lösbar ist. Das k-MST Problem hingegen ist $\mathcal{NP}$-schwer. Da Sie vor Kurzem in Algorithmen und Datenstrukturen 2 das Branch-and-Bound Verfahren kennengelernt haben, entschließen Sie sich, das Problem auf diese Weise exakt zu lösen.

Konkret sieht Ihre Aufgabenstellung wie folgt aus:

1. Überlegen Sie, wie Sie das Branching ausführen, d.h., wie Sie ein (Unter-)Problem in weitere Unterprobleme zerteilen und mithilfe welcher Datenstrukturen Sie offene (Unter-)Probleme speichern.

2. Entwickeln Sie eine Dualheuristik, die für jedes (Unter-)Problem eine lokale untere Schranke liefert.

3. Entwickeln Sie darauf aufbauend einen heuristischen Algorithmus, um für ein (Unter-)Problem möglicherweise eine gültige Lösung zu erhalten, deren Wert eine globale obere Schranke darstellt.

4. Für die grundsätzliche Funktionsweise des Branch-and-Bound ist es egal, welches offene Unterproblem aus der Problemliste ausgewählt und als nächstes abgearbeitet wird. In der Praxis spielt diese Auswahlstrategie jedoch in Bezug auf die Laufzeit eine große Rolle. Wir schlagen hier vor, als erstes eine Depth-First-Strategie zu implementieren, bei der nach einem Branching immer eines der neu erzeugten Unterprobleme als unmittelbar nächstes abgearbeitet wird. In dieser Weise kann das Branch-and-Bound direkt als rekursiver Algorithmus ohne zusätzliche Datenstruktur zur expliziten Speicherung der Problemliste implementiert werden.

5. Implementieren Sie nun das vollständige Branch-and-Bound, das auf den oben genannten Punkten aufbaut.

Tipps

- Überlegen Sie sich, ob binäres Branchen im konkreten Fall die beste Lösung ist.
- Sie können das Verfahren mitunter erheblich beschleunigen, wenn Sie Unterprobleme, die keine gültige Lösung mehr erlauben möglichst früh erkennen und unmittelbar ausschließen.
- Auch die Wahl des nächsten abzuarbeitenden Unterproblems hat starken Einfluss auf die Geschwindigkeit Ihres Algorithmus. Welche Unterprobleme sollten Sie wann betrachten um schnell gute gültige Lösungen (obere Schranken) zu finden?
- Denken Sie nach, ob Sie einen bekannten Algorithmus für ein Problem verwenden können um untere Schranken zu finden. Die Qualität Ihrer unteren Schranke bestimmt maßgeblich die Geschwindigkeit Ihres Branch-and-Bound Verfahrens.

Hinweis zur Laufzeit

Pro Instanz stehen Ihrem Programm am Abgabeserver maximal 30 Sekunden CPU-Zeit zur Verfügung. Findet Ihr Algorithmus in dieser Zeit keine optimale Lösung, dann wird die beste bisher gefundene gültige Lösung zur Bewertung herangezogen.

Codegerüst

Der Algorithmus ist in Java 8 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.

Sie können das Framework wie folgt kompilieren:

```
$ javac ad2/ss16/pa/*.java
```

Das Codegerüst besteht aus mehreren Klassen, wobei Sie Ihren Code in die Datei `KMST.java` einfügen müssen. In dieser Datei können Sie mehrere Klassen definieren bzw. implementieren.

Klassen & Methoden

`KMST` ist die Klasse, in der Sie Ihren Branch-and-Bound Algorithmus implementieren und in `void run()` starten. Weiters können Sie den Konstruktur zur Initialisierung Ihrer Daten verwenden.

Dem Konstruktor wird die Anzahl der Knoten und Kanten, die Kantenmenge, sowie der Parameter k der Instanz übergeben.

`Edge` repräsentiert eine Kante des Graphen. Die Klasse beinhaltet die Attribute `node1` und `node2` die die verbundenen Knoten beschreiben, sowie das Attribut `weight`, das das Gewicht der Kante angibt.

`AbstractKMST` stellt Basismethoden für das Framework zur Verfügung.

- `boolean setSolution(int newUpperBound, int[] newSolution)` müssen Sie verwenden, um neu gefundene Lösungen dem Framework mitzuteilen (siehe Hinweise weiter unten).
- `BnBSolution getBestSolution()` liefert die bisher beste gefundene Lösung zurück. Verwenden Sie `double getUpperBound()` bzw. `int[] getBestSolution()` um die entsprechenden Werte aus der zurückgelieferten Instanz zu extrahieren.

Achtung! Sie sind dafür verantwortlich, dass die Methoden mit sinnvollen Werten aufgerufen werden. Sollte das nicht passieren wird Ihr Programm in den meisten Fällen eine Runtime Exception werfen.

Hinweise

`Main.printDebug(String msg)` kann verwendet werden, um Debuginformationen auszugeben. Die Ausgabe erfolgt nur, wenn beim Aufrufen das Debugflag (`-d`) gesetzt wurde. Sie müssen diese Ausgaben vor der Abgabe **nicht** entfernen, da das Testsystem beim Kontrollieren Ihrer Implementierung dieses Flag nicht setzt.

Sie müssen die Methode `boolean setSolution(int newUpperBound, int[] newSolution)` der Klasse `AbstractKMST` verwenden, um dem Framework eine neue (beste) Lösung bekannt zu geben. Die Lösung wird nur übernommen, wenn Ihr Wert eine verbesserte (d.h. neue niedrigste) obere Schanke darstellt. Die Methode liefert `true` zurück,

wenn die Lösung übernommen wurde.
Achtung: Die Korrektheit der Lösung wird von der Methode nicht überprüft. Sie müssen sicherstellen, dass es sich um eine korrekte Lösung handelt.

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 für diese Lehrveranstaltung.

Rückgabe des Frameworks

Nach dem Aufruf Ihres Branch-and-Bound Verfahrens wird die zurückgelieferte Lösung auf Korrektheit geprüft und die Kostenfunktion berechnet. Für eine positive Abgabe muss Ihr Verfahren für jede Instanz eine Lösung mit einer Lösungsqualität liefern, die einen vorgegebenen Schwellwert nicht überschreitet.

Wenn Ihre Lösung in diesem Sinne ausreichend gut ist, werden vom Framework eine entsprechende Meldung sowie der Lösungswert zurückgegeben:

```
Schwellwert = 543. Ihr Ergebnis ist OK mit 318
```

Ist das Ergebnis Ihres Verfahrens nicht gut genug, werden Sie Meldungen wie die folgende sehen:

```
ERR zu schlechte Loesung: Ihr Ergebnis 765 liegt über dem
Schwellwert (567)
```

Liefert Ihr Verfahren `null` oder ein ungültiges Ergebnis zurück, sehen Sie eine Fehlermeldung wie diese:

```
ERR Keine gueltige Loesung!
```

Kommandozeilenoptionen Um Ihnen die Arbeit ein wenig zu erleichtern, gibt es Kommandozeilenoptionen, die das Framework veranlassen, mehr Informationen auszugeben.

Wenn Sie beim Aufrufen von Main `-t` angeben, wird die Laufzeit Ihrer Implementierung gemessen. Diese ist natürlich computerabhängig, kann aber trotzdem beim Finden von ineffizienten Algorithmen helfen.

```
$ java ad2.ss16.pa.Main -t tests/input/angabe

tests/input/angabe: Schwellwert = 543. Ihr Ergebnis ist OK mit
318, Zeit: 15 ms
```

Um zu verhindern, dass das Framework Ihren Algorithmus nach 30 Sekunden beendet, können Sie beim Aufrufen von Main `-s` angeben. Dies ist vor allem zum Debuggen nützlich, da sonst nach dem Ablaufen der Zeit kein weiteres schrittweises Abarbeiten mehr möglich ist.

Testdaten

Im TUWEL Kurs der LVA sind 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 gültigen Eingaben korrekt behandelt. Testen Sie daher auch mit zusätzlichen, selbst erstellten Daten. Das Abgabesystem wird Ihr Programm mit den öffentlichen und zusätzlich mit nicht veröffentlichten Daten testen.

Eine KMST Instanz besitzt folgendes Format:

Die ersten vier Zeilen der Testdaten enthalten die Anzahl der Knoten, die Anzahl der Kanten, den Wert k und den Schwellwert, der jedenfalls erreicht werden muss (in dieser Reihenfolge). Die weiteren Zeilen enthalten die Kanteninformationen, d.h. für jede Kante folgende durch Leerzeichen getrennte Werte: `kantennummer`, `startknotennummer`, `endknotennummer` und `kantengewicht`.

Abgabe

Die Abgabe Ihrer Implementierung der Klasse `KMST` 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:

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 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 über das Abgabesystem in TUWEL. Da es bei dieser Aufgabe mehrere gültige Lösungen geben kann, werden diese je nach Qualität in Kategorien eingeteilt. Ein

grünes Zeichen im Abgabesystem bedeutet, dass Ihre Lösung für die jeweilige Instanz sehr gut ist, ein gelbes Zeichen zeigt Ihnen, dass Ihre Lösung ausreichend ist, um positiv bewertet zu werden. Falls Ihre Lösung für eine Instanz ungültig ist oder der Lösungswert oberhalb des vorgegebenen Schwellwerts liegt, sehen Sie ein rotes Zeichen. In diesem Fall müssen Sie Ihren Algorithmus noch verbessern, um eine positive Bewertung zu erhalten.

Aufgrund der großen Studierendenzahl 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 den von Ihren Kollegen **abgegebenen Programmen automatisch vergleichen werden, um Plagiate zu erkennen**. Geben Sie daher nur selbst implementierte Lösungen ab!

Theoriefragen

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

- Welche Vorteile/Nachteile hat binäres Branchen im konkreten Fall. Welche Alternativen gibt es?
- Sind die durch Ihre Verfahren gefundenen unteren und oberen Schranken jeweils global gültig oder nur für das entsprechende Teilproblem? Warum ist dies so?
- Wann weiß man beim Branch-and-Bound, dass die beweisbar beste Lösung gefunden wurde?
- Wie führen Sie das Branching aus und wie sieht Ihr Suchbaum aus?

Testumgebung

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

```
ad2.ss16.pa.Main input > output
```

wobei `input` eine Eingabedatei darstellt. Die Ausgabe wird in der Datei `output` gespeichert. Pro Testinstanz darf eine maximale Ausführungszeit von 30 Sekunden nicht überschritten werden, anderenfalls wird die Ausführung abgebrochen.

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:

- Korrektheit des Algorithmus, d.h., alle Instanzen erfolgreich gelöst (gelb oder grün)
- Erzielte Lösungsqualität
- Laufzeiteffizienz
- Speicherverbrauch
- Rechtfertigung des Lösungsweges
- Antworten auf die theoretischen Fragen der Aufgabenstellung

Voraussetzung für eine positive Absolvierung des Abgabegesprächs ist jedenfalls auch das grundsätzliche Verständnis der Problemstellung.

Punktevergabe

Ergebnisse:

Sobald alle Instanzen gelb gelöst wurden, gibt es dafür 11 Punkte sofern Sie Ihr Programm beim Abgabegespräch erklären können. Es werden folgende zusätzliche Punkte vergeben:

- Maximal 3 Punkte für die Anzahl der optimal (grün) gelösten Instanzen

Instanzen auf grün	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Punkte	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3

- Maximal 6 Punkte für die Implementierung:
  - Branching bis zu **2 Punkte**
  - Bounding u. Dualheuristik (untere Schranke) bis zu **2 Punkte**
  - Primalheuristik (obere Schranke) bis zu **2 Punkte**

Anmerkungen:

- Für eine **positive** Abgabe müssen **zumindest** das Branching und das Bounding implementiert werden und es darf **keine** Instanz **rot** im Abgabesystem markiert sein.
- Die angegebenen Punkte bilden eine Basis. **Bei schlechter Erklärung können Punkte abgezogen werden.**