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

186.815 Algorithmen und Datenstrukturen 2 VU 3.0

Sommersemester 2014

Programmieraufgabe

abzugeben bis: Freitag, 13. Juni 2014, 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 6 verwendet.

Geben Sie bitte Ihr fertiges, gut getestetes und selbst geschriebenes Programm bis spätestens **Freitag, 13. Juni 2014, 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 einen Punkt erreichen** und Ihren Termin zum Abgabegespräch einhalten.

> 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 einen Punkt auf dieses Programmierbeispiel erhalten.

Abgabefrist

Sie haben bis Freitag, 13. Juni 2014, 15:00 Uhr die 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

Am Dienstag, 17. Juni 2014, Mittwoch, 18. Juni 2014, und Freitag, 20. Juni 2014, 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 13. Juni 2014 bis 16. Juni 2014, 23:59 Uhr in TUWEL bei einer/m TutorIn anmelden. Sollten in diesem Zeitraum keine Termine in TUWEL eingetragen oder bereits alle Termine vergeben sein, dann kontaktieren Sie bitte die Hotline `algodat2-ss14@ads.tuwien.ac.at`.
Melden Sie sich nur an, wenn Sie positiv abgegeben haben!

Falls Sie Systemroutinen in Ihrem Programm 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 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

Das Semester neigt sich dem Ende zu und Sie wollen mit Ihren Kollegen den erfolgreichen Abschluss gebührend feiern. Dafür planen Sie eine Gourmet-Tour durch Wien, bei der Sie Ihre Lieblingslokale besuchen und in jedem Lokal jeweils einen Gang zu sich nehmen, wobei Sie sich mit Ihren Kollegen einen Treffpunkt ausmachen, bei dem Sie einen Aperitif zu sich nehmen und zu dem Sie am Ende für einen letzten Schlummertrunk zurückkehren. Da Sie natürlich möglichst viel Zeit in den Lokalen und wenig Zeit mit Reisen verbringen möchten, fragen Sie sich, wie Sie die optimale Gourmet-Tour berechnen können.

Augenblicklich erinnern Sie sich an Ihre Algorithmen und Datenstrukturen 2 Vorlesung, wo Sie Bekanntschaft mit dem Traveling Salesman Problem (TSP) gemacht haben. Doch die Definition des TSP deckt sich nicht ganz mit Ihrem Problem, zumal für die Reihenfolge der Lokale und damit verbunden der Gänge zusätzliche Bedingungen herrschen. Sie wollen ja nicht die Sacher Torte vor dem Schnitzel essen.

Eine kurze Internetrecherche ergibt, dass auch andere Feinschmecker dieses Problem bereits hatten und es in der Literatur als *Traveling Salesman Problem with Precedence Constraints* (TSP-PC) bekannt ist.

Sie können Ihr Problem folgendermaßen formal definieren:
Gegeben ist ein vollständiger, ungerichteter, gewichteter Graph $G = (V, E)$ mit Knotenmenge (Lokale) V und Kantenmenge E. Jeder Kante $(i, j) \in E$ ist eine Länge $d_{ij} \geq 0$ (Distanz zwischen den Lokalen i und j) zugeordnet. Zusätzlich gibt es eine Liste mit Rangfolgen Bedingungen (Precedence Constraints). Jeder Eintrag $i \prec j, i \neq j, i, j \in V$, bedeutet, dass in einer gültigen Lösung Knoten i vor Knoten j besucht werden muss. Die Precedence Constraints sind transitiv, d.h., wenn $i \prec j$ und $j \prec k$ gilt, dann gilt auch $i \prec k, \forall i, j, k \in V$.

Gesucht ist eine kürzestmögliche Rundtour, die von einem von Ihrem Algorithmus gewählten Startknoten aus alle Knoten im Graphen genau einmal besucht und zum Startknoten zurückkehrt und die alle Precedence Constraints erfüllt.

Sie wissen aus Algorithmen und Datenstrukturen 2, dass das TSP $\mathcal{NP}$-schwer ist. Die Precedence Constraints vereinfachen das Problem im Allgemeinen nicht und daher ist auch das TSP-PC $\mathcal{NP}$-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. Optional besteht die Möglichkeit, eine lokale Suche zu implementieren um eine gefundene gültige Lösung und damit verbunden Ihre obere Schranke zusätzlich zu verbessern.

5. 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, 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.

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

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 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].

Sie können das Framework wie folgt kompilieren:

```
$ javac ads2/ss14/etsppc/*.java
```

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

Klassen & Methoden

`Location` speichert einen Punkt, den Sie in Ihrer Tour besuchen müssen.

- `int getId()` liefert eine eindeutige ID für die `Location` zurück.
- `double getX()` und `double getY()` liefern die X und Y Koordinaten der `Location` zurück.
- `double distanceTo(Location other)` berechnet die Distanz zu einer anderen `Location`.

`PrecedenceConstraint` repräsentiert ein Precedence Constraint, dass Ihre Tour erfüllen muss.

- `int getFirst()` liefert die ID der `Location` zurück, die vor der zweiten `Location` in der Tour besucht werden muss. Die zweite `Location` ID erhalten Sie mittels `int getSecond()`.
  Beachten Sie, dass einige Precedence Constraints redundant sein können.

`ETSPPCInstance` speichert alle Informationen über die aktuelle Probleminstanz.

- `List<PrecedenceConstraint> getConstraints()` liefert Ihnen alle Precedence Constraints der Instanz zurück.
- `Map<Integer, Location> getAllLocations()` liefert Ihnen ein Mapping zwischen IDs und zugehörigen `Locations` zurück.
- `double getThreshold()` liefert Ihnen die obere Schranke zurück, die Sie mit Ihrer Lösung erreichen müssen.

`ETSPPC` ist die Klasse, in der Sie Ihren Branch-and-Bound Algorithmus implementieren und in `void run()` starten. Weiters können Sie den Konstruktor verwenden um Ihre Klasse zu initialisieren.

[1] `http://www.ads.tuwien.ac.at/teaching/lva/186815.html`

`AbstractETSPPC` stellt Basismethoden für das Framework zur Verfügung.

- `boolean setSolution(double newUpperBound, List<Location> newSolution)` müssen Sie verweden 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. `List<Location> getBestSolution()` um die entsprechenden Werte aus der zurückgelieferten Instanz zu extrahieren.

Eine Lösung wird als `List<Location>` repräsentiert, die alle `Locations` in der Reihenfolge beinhaltet, in der sie besucht werden. Die Upper Bound der Lösung können Sie mit `double Main.calcObjectiveValue(List<Location> solution)` berechnen. Beachten Sie, dass Sie die erste `Location` am Ende der Liste nicht noch einmal einfügen müssen, aber können. Das Framework berechnet in beiden Fällen die richtige Tourlänge.

Weitere Details, die das Codegerüst betreffen, entnehmen Sie bitte der LVA-Webseite[2]. Dort befindet sich auch der Link zur Dokumentation (Javadoc) des Codegerüstes.

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, List<Location> newSolution)` der Klasse `AbstractETSPPC` 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 Tour 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 auf diese Lehrveranstaltung.

Rückgabe des Frameworks

Nach dem Aufruf Ihres Branch-and-Bound Verfahrens wird die zurückgelieferte Tour auf Korrektheit geprüft und die Tourlänge wird berechnet. Für eine positive Abgabe muss Ihr

[2] `http://www.ads.tuwien.ac.at/teaching/lva/186815.html`

Verfahren für jede Instanz eine Lösung mit einer Tourlänge liefern, die einen vorgegebenen Schwellwert nicht überschreitet.

Wenn Ihre Lösung in diesem Sinne ausreichend gut ist, werden vom Framework eine entsprechende Meldung sowie die Tourlänge 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.ss14.etsppc.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

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 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 TSP Instanz mit Precedence Constraints besitzt folgendes Format:

```
DIMENSION: <Anzahl an Locations>
THRESHOLD: <Schwellwert der Instanz>
NODE_COORD_SECTION
ID1 x1 y1
ID2 x2 y2
...
IDn xn yn
PRECEDENCE_SECTION
IDi IDj
IDk IDl
...
EOF
```

`IDs` und die `<Anzahl an Locations>` sind als Integerwerte anzugeben. Der `<Schwellwert der Instanz>` und die `x` bzw. `y` Koordinaten sind als Doublewerte anzugeben.

Abgabe

Die Abgabe Ihrer Implementierung der Klasse `ETSPPC` 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 überhalb 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 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 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:

- Ist jedes Lokal als Startknoten geeignet? Welchen Einfluss hat die Wahl des Startknotens auf die Tourlänge?
- 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.ss14.etsppc.Main input > output
```

wobei `input` eine Eingabedatei darstellt. Die Ausgabe wird in die 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
- 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.

Zusätzliche Informationen

Lesen Sie bitte auch die auf der Webseite zu dieser LVA veröffentlichten Hinweise. Wenn Sie Fragen oder Probleme haben, wenden Sie sich rechtzeitig an die AlgoDat2-Hotline unter `algodat2-ss14@ads.tuwien.ac.at`.