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

186.815 Algorithmen und Datenstrukturen 2 VU 3.0

Sommersemester 2017

Programmieraufgabe

abzugeben bis: Freitag, 16. Juni 2017, 12:00 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 verwendet.

Geben Sie bitte Ihr fertiges, gut getestetes und selbst geschriebenes Programm bis spätestens **Freitag, 16. Juni 2017, 12:00 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. Sie bekommen 11 Punkte, wenn Ihr Programm **alle Testinstanzen** korrekt abarbeitet. Die restlichen Punkte werden im Abgabegespräch vergeben.

Abgabefrist

Sie haben bis Freitag, 16. Juni 2017, 12:00 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

Die Abgabegespräche finden, nach erfolgreicher elektronischer Abgabe, in der Zeit von Montag, 19. Juni bis Freitag, 23. Juni 2017 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 zu diesem Abgabegespräch von 15. Juni 2017 bis 18. Juni 2017, 23:59 Uhr in TUWEL bei einer/m TutorIn anmelden. Melden Sie sich nur an, wenn Sie positiv abgegeben haben!

Falls Sie Aufrufe aus der Java-API 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

Sie haben vor Kurzem bei einem neu gegründeten Internetprovider angeheuert, der mit Ihrer Hilfe seine Infrastruktur aufbauen will. Der Provider hat es sich zur Aufgabe gemacht, Datenzentren an das Internet anzubinden. Vor Beginn des Netzbetriebes hat Ihr Auftraggeber bereits eine Menge an Kunden gefunden, die bereit sind, ihren Provider zu wechseln, sofern ihnen geforderte Mindestbandbreiten zugesichert werden und Sie ihnen ein günstiges Angebot vorlegen können.

Um die geforderten Bandbreiten gewährleisten zu können, müssen Sie jeden Kunden zu einem von mehreren möglichen (noch nicht gebauten) Knotenpunkten zuordnen, die diesen dann in Ihr Backbone einbinden. Dabei besitzt jeder Knotenpunkt eine maximal zur Verfügung stehende Bandbreite, die sich nach seiner Ausbaustufe $(1, 2, 3, \ldots)$ richtet. Jede Ausbaustufe erhöht dabei die Kosten des Knotenpunkts und vervielfacht die maximale Bandbreite des Knotenpunkts. Sie möchten die Kosten für das Errichten der Knotenpunkte und das Verlegen der Leitungen zwischen Knotenpunkt und Datenzentren minimieren.

Nach einigen Überlegungen erkennen Sie, dass es sich um eine Variante des bekannten Capacitated Facility Location Problems (CFLP) handelt.

Sie können Ihr Problem folgendermaßen definieren:

Gegeben sei eine Menge n möglicher Orte an denen Facilities (Knotenpunkte) in individuellen Ausbaustufen $f_i \in \mathbb{N}_0^+$, $i = 1, \ldots, n$, errichtet werden können. Die Errichtungskosten einer Facility der Ausbaustufe $k \in \mathbb{N}_0^+$ an Ort i sind dabei

$$c(k,i) = \begin{cases} 0 & \text{für } k = 0 \quad \text{(keine Facility am Standort } i\text{)} \\ a_i & \text{für } k = 1 \\ \lceil 1.5 a_i \rceil & \text{für } k = 2 \\ c(k-1,i) + c(k-2,i) + (4-k)a_i & \text{für } k \geq 3 \end{cases}$$

wobei $a_i > 0$ gegebene standortabhängige Basiserrichtungskosten sind.

Eine Facility der Ausbaustufe f_i besitzt eine maximale Gesamtbandbreite $f_i \cdot B_i$, wobei die Maximalbandbreite $B_i > 0$ pro Ausbaustufe ebenfalls für jeden Standort individuell gegeben ist.

Weiters sind die Bandbreitenanforderungen $b_j > 0$ von m anzubindenden Kunden $j = 1, \ldots, m$ gegeben. Die Entfernung zwischen jeder Facility i und jedem Kunden j sind durch $d_{ij} > 0$ gegeben, und die Herstellung einer entsprechenden Anbindung verursacht Kosten der Höhe $e \cdot d_{ij}$, wobei e gegebene, konstant angenommene Kosten pro Entfernungseinheit sind.

Jeder Kunde j muss genau an eine erbaute Facility i angebunden werden. Dabei darf die Summe der Bandbreitenanforderungen der Kunden, die der gleichen Facility i zugeordnet sind, $B_i \cdot f_i$ nicht übersteigen.

Ziel ist es, die Gesamtkosten, die sich aus Facility-Errichtungskosten und Anbindungskosten ergeben, zu minimieren.

Formal kann das Problem wie folgt dargestellt werden (äquivalent zum obigen Text):

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} c(f_i, i) + \sum_{i=1}^{n} \sum_{j=1}^{m} e \cdot d_{ij} \cdot x_{ij} & \\
& \sum_{i=1}^{n} x_{ij} = 1 & \forall j = 1, \ldots, m \\
& \sum_{j=1}^{m} b_j \cdot x_{ij} \leq B_i \cdot f_i & \forall i = 1, \ldots, n \\
& f_i \in \mathbb{N}_0^+ & \forall i = 1, \ldots, n \\
& x_{ij} \in \{0, 1\} & \forall i = 1, \ldots, n,\ j = 1, \ldots, m
\end{aligned}
$$

Hierbei gibt f_i an, ob und in welcher Ausbaustufe Facility i gebaut wird und x_{ij} ist genau dann und nur dann eins, wenn Kunde j an Facility i angebunden wird.

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-Verfahrens 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.

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, indem Sie Teilprobleme, zu denen es keine gültige Lösung geben kann, möglichst rasch ausschließen bzw. gar nicht erst erzeugen.
- Auch die Wahl des nächsten abzuarbeitenden Unterproblems hat starken Einfluss auf die Geschwindigkeit Ihres Algorithmus. Welche Unterprobleme sollten Sie wann betrachten um möglichst schnell gute gültige Lösungen und damit obere Schranken zu finden?
- Überlegen Sie, ob Sie ein ähnliches Problem (andere Kostenfunktion, einfachere Constraints, ...) und eine zugehörige untere Schranken in der Vorlesung oder Übung kennengelernt haben. Untere Schranken sind transitiv, d.h. jede untere Schranke einer unteren Schranke zu Ihrem Problem ist ebenfalls eine untere Schranke zum Ausgangsproblem (wenn auch möglicherweise keine Gute). Die Qualität Ihrer unteren Schrankenberechnung bestimmt maßgeblich die Geschwindigkeit Ihres Branch-and-Bound-Verfahrens.
- Es wird Ihnen ein Codegerüst zu Verfügung gestellt, das auch eine Berechnung der Zielfunktion für eine Gesamtlösung enthält. Dieser Code sollte aus gutem Grund jedoch nicht in Ihrem BnB-Verfahren verwendet werden. Wieso?
- Unter Umständen können sehr schleche Lösungen einen Integer-Overflow erzeugen. Die Berechnungsmethode im Framework wertet dies als Fehler. Sie können davon ausgehen, dass eine optimale Lösung nie einen Overflow erzeugt.

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. Bedenken Sie, dass Ihrem Programmablauf am Abgabeserver vermutlich weniger Ressourcen zu Verfügung gestellt werden als Ihnen Ihr Privatrechner bietet.

Codegerüst

Der Algorithmus ist in Java 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/ss17/cflp/*.java
```

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

Klassen & Methoden

`CFLPInstance` speichert alle Informationen zur aktuellen Probleminstanz.

- `int distanceCosts` entspricht den Anbindungskosten e pro Entfernungseinheit.
- `int getNumCustomers()` liefert die Anzahl der Kunden (m).
- `int getNumFacilities()` liefert die Anzahl der Facilities (n).
- `int baseOpeningCostsOf(int facilityIdx)` liefert Ihnen die Basiserrichtungskosten (a_i) für eine Facility an Ort i zurück.
- `int maxBandwidthOf(int facilityIdx)` liefert die Maximalbandbreite B_i pro Ausbaustufe für eine Facility an Ort i zurück.
- `int bandwidthOf(int customerIdx)` liefert Ihnen die Bandbreitenanforderungen b_j für den angegebenen Kunden zurück.
- `int distance(int facilityIdx, int customerIdx)` liefert Ihnen die Entfernung d_{ij} zwischen der angegebenen Facility und dem angegebenen Kunden zurück.

`CFLP` ist die Klasse, in der Sie Ihren Branch-and-Bound Algorithmus implementieren sollen, die Optimierung soll über `void run()` gestartet werden. Weiters können Sie den Konstruktor verwenden um Ihre Klasse zu initialisieren.

`AbstractCFLP` 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 `int 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 `AbstractCFLP` 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 auf 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 ueber 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.ss17.cflp.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 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 CFLP Instanz besitzt folgendes Format:

```
THRESHOLD: <Schwellwert der Instanz>
FACILITIES: <Anzahl an möglichen Orten für Facilities n>
CUSTOMERS: <Anzahl an Kunden m>

MAXBANDWIDTHS: <Maximalbandbreite B_i pro Ausbaustufe für jeden Ort i;
                durch Leerzeichen getrennt>
DISTANCECOSTS: <Anbindungskosten e pro Entfernungseinheit>
OPENINGCOSTS: <Liste der Basiserrichtungskosten a_i für das Eröffnen
               der i-ten Facility; durch Leerzeichen getrennt>

#BANDWIDTH; DISTANCE_TO_FACILITY_0 DISTANCE_TO_FACILITY_1 ...
<Liste im obigen Kommentar angegebenen Format für jeden Kunden;
 gibt Bandbreitenanforderungen b_j und Entfernung d_ij zu jeder Facility an;
 eine Zeile pro Kunde>
```

Alle Werte sind als Integerwerte anzugeben. Leerzeilen und Zeilen die mit # beginnen, werden ignoriert.

Abgabe

Die Abgabe Ihrer Implementierung der Klasse `CFLP` 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 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 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 Abgaben 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.ss17.cflp.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.

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 4 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	0	1	1	2	2	3	3	4	4	4	4	4

- Maximal 5 Punkte für die Implementierung bzw. Erklärungen für:
  - Branching bis zu **1 Punkt**
  - Bounding u. Dualheuristik (untere Schranke) bis zu **3 Punkte**
  - Primalheuristik (obere Schranke) bis zu **1 Punkt**

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 oder fehlerhaften Implementierungen können Punkte abgezogen werden.**