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

186.815 Algorithmen und Datenstrukturen 2 VU 3.0

Sommersemester 2015

Programmieraufgabe

abzugeben bis: Montag, 15. Juni 2015, 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 **Montag, 15. Juni 2015, 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 Montag, 15. Juni 2015, 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

Zwischen Dienstag, 16. Juni 2015 und Freitag, 19. Juni 2015, 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 12. Juni 2015 bis 15. Juni 2015, 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-ss15@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

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 (gleich für alle Knotenpunkte), die die zugeordneten Kunden nicht überschreiten dürfen, und eine maximale Anzahl an Kunden (unterschiedlich für verschiedene Knotenpunkte), die angeschlossen werden können. Sie möchten die Kosten für das Errichten der Knotenpunkte und das Verlegen der Leitungen zwischen Knotenpunkt und Datenzentren minimieren.

Als guter Algorithmiker erkennen Sie sofort, dass es sich um einen Spezialfall des Capacitated Facility Location Problems (CFLP) handelt, welches im konkreten Fall mehrere Capacity Constraints besitzt.

Sie können Ihr Problem folgendermaßen definieren:
Gegeben sei eine Menge an n möglichen Facilities i, $1 \leq i \leq n$ (Knotenpunkte) mit assoziierten Errichtungskosten $c_i > 0$, einer maximal zur Verfügung stehenden Bandbreite $B > 0$ und maximaler Kundenanzahl $k_i > 0$ pro Facility. Weiters ist eine Menge der m anzubindenden Kunden j, $1 \leq j \leq m$ (Datenzentren) mit Bandbreitenanforderungen $b_j > 0$ gegeben.

Die Distanz zwischen Facility i and Kunde j ist durch d_{ij} gegeben, wobei jede Distanzeinheit e Einheiten Geld kostet.

Jeder Kunde j muss exakt einer erbauten Facility i zugeordnet werden. Dabei darf die Summe der Bandbreiten der Kunden, die der gleichen Facility zugeordnet sind, B nicht übersteigen. Die Anzahl der Facility i zugeordnenten Kunden muss kleiner oder gleich k_i sein. Das Ziel ist es, die Gesamtkosten die sich aus Errichtungskosten und Leitungskosten ergeben, zu minimieren.

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

$$\min \sum_{i=1}^{n} c_i \cdot f_i + \sum_{i=1}^{n} \sum_{j=1}^{m} e \cdot d_{ij} \cdot x_{ij}$$

$$x_{ij} \leq f_i \qquad \forall i, j, 1 \leq i \leq n, 1 \leq j \leq m$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j, 1 \leq j \leq m$$

$$\sum_{j=1}^{m} b_j x_{ij} \leq B \qquad \forall i, 1 \leq i \leq n$$

$$\sum_{j=1}^{m} x_{ij} \leq k_i \qquad \forall i, 1 \leq i \leq n$$

$$f_i \in \{0, 1\} \qquad x_{ij} \in \{0, 1\}$$

wobei f_i angibt ob Facility i gebaut wird ($f_i = 1$) oder nicht gebaut wird ($f_i = 0$) und x_{ij} genau dann und nur dann eins ist, wenn Kunde j Facility i zugeordnet wird.

Sie wissen aus Algorithmen und Datenstrukturen 2, dass das Bin Packing Problem $\mathcal{NP}$-schwer ist. Da das CFLP eine Erweiterung des Bin Packing Problems (unterschiedliche Kosten pro Bin) darstellt und die zusätzlichen Constraints das Problem im Allgemeinen nicht leichter machen, ist auch das CFLP $\mathcal{NP}$-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.

Tipps

- Überlegen Sie sich, ob binäres Branchen im konkreten Fall die beste Lösung ist.
- Sie können das Verfahren mitunter erheblich beschleunigen, wenn Sie unvollständige aber bereits ungültige Lösungen im Branch-Tree sofort ausschließen und nicht weiter verfolgen.
- 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 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 Problem selbst (wenn auch keine Gute). 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 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.

Sie können das Framework wie folgt kompilieren:

```
$ javac ads2/ss15/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 über die aktuelle Probleminstanz.

- `int maxBandwidth` speichert die Bandbreitenbeschränkung, die für alle Facilities gültig ist.
- `int distanceCosts` speichert die Kosten für das Verlegen einer Längeneinheit an Leitungen.
- `int getNumCustomers()` liefert Ihnen die Anzahl an Kunden in der aktuellen Instanz.
- `int getNumFacilities()` liefert Ihnen die Anzahl an Facilities in der aktuellen Instanz.
- `int openingCostsFor(int facilityIdx)` liefert Ihnen die Eröffnungskosten für die angegebene Facility zurück. Für direkteren Zugriff steht auch das Array `int[] openingCosts` zur Verfügung.
- `int maxNumCustomersOf(int facilityIdx)` liefert Ihnen die maximale Anzahl an Kunden für die angegebene Facility zurück. Für direkteren Zugriff steht auch das Array `int[] maxCustomers` zur Verfügung.
- `int bandwidthOf(int customerIdx)` liefert Ihnen die Bandbreitenanforderungen für den angegebenen Kunden zurück. Für direkteren Zugriff steht auch das Array `int[] bandwidths` zur Verfügung.
- `int distance(int facilityIdx, int customerIdx)` liefert Ihnen die Enternung zwischen der angegebenen Facility und dem angegebenen Kunden zurück. Für direkteren Zugriff steht auch das Array `int[][] distances` zur Verfügung.
- `int calcObjectiveValue(int[] solution)` berechnet die Lösungsqualität für die angegebene Lösung. Eine Lösung ist ein Integer-Array der Größe m, das für jeden Kunden j (Index des Arrays, $0 \dots m-1$) die zugewiesene Facility i, $0 \dots n-1$ speichert. Wird für einen Kunden ein Wert kleiner 0 abgespeichert, dann wird die Zuordnung bei der Berechnung ignoriert und als noch nicht zugewiesen angesehen.
- `double getThreshold()` liefert Ihnen die obere Schranke zurück, die Sie mit Ihrer Lösung erreichen müssen.

Achtung! Beachten Sie, dass wenn sie auf `public` Arrays direkt zugreifen, dass diese auch beschrieben werden können und das dies möglicherweise zu ungewollten Fehlern führt. Die Instanz wird auf jeden Fall kopiert, bevor sie an Ihren Code übergeben wird, damit sich etwaige Fehler nicht auf die Überprüfung der Korrektheit auswirken können.

`CFLP` 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.

`AbstractCFLP` stellt Basismethoden für das Framework zur Verfügung.

- `boolean setSolution(int newUpperBound, int[] 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. `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 den 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 ads2.ss15.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 Facilities>
CUSTOMERS: <Anzahl an Kunden>

MAXBANDWIDTH: <maximale Bandbreite pro Facility>
MAXCUSTOMERS: <Liste der maximalen Anzahl an Kunden der i-ten Facility;
               durch Leerzeichen getrennt>

DISTANCECOSTS: <Kosten für eine Längeneinheit>
OPENINGCOSTS: <Liste der Kosten 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 und Distanz 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 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:

```
ads2.ss15.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

Algorithmus:

- Branching: bis zu **4 Punkte**
- Bounding u. Dualheuristik (untere Schranke): bis zu **4 Punkte**
- Primalheuristik (obere Schranke): bis zu **2 Punkte**

Ergebnisse:

Wurde eine Enumeration oder ein Branch and Bound Algorithmus umgesetzt, werden folgende zusätzliche Punkte für die Anzahl der optimal gelösten Instanzen vergeben.

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

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.