Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Thomas Tschager
Antonis Thomas

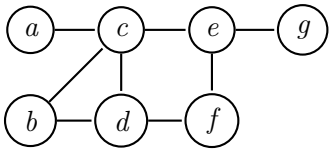
11. Mai 2016

Datenstrukturen & Algorithmen — Blatt 11 — FS 16

Aufgabe 11.1 *Branch-and-Bound.*

In dieser Aufgabe soll ein klassisches Optimierungsproblem mit *Branch-and-Bound* gelöst werden. Beim Problem *Minimum Dominating Set* ist ein (ungerichteter) Graph $G = (V, E)$ mit $n = |V|$ Knoten und $m = |E|$ Kanten gegeben. Gesucht ist eine kleinstmögliche Menge $D \subseteq V$, für die jeder Knoten in $V \backslash D$ einen Nachbarn in D hat (solch eine Menge nennt man *dominierend*).

Wir betrachten den rechts dargestellten Graphen. In diesem Graphen ist $D = \{a, b, f, g\}$ eine dominierende Menge. Keine echte Teilmenge von D ist dominierend, und dennoch ist D nicht kleinstmöglich. Ihre Aufgabe ist es nun, eine kleinstmögliche dominierende Menge für den gezeigten Graphen zu finden, indem Sie die Branch-and-Bound-Methode von Hand anwenden. Gehen Sie dabei wie folgt vor:



a) Überlegen Sie sich zunächst eine untere Schranke (Bound) für die Anzahl benötigter Knoten, wenn wir von einer gegebenen Teillösung ausgehen. Eine Teillösung ist dabei durch zwei Mengen In und Out beschrieben. Die Menge In enthält die Knoten, die Teil der dominierenden Menge sein sollen, während die Menge Out Knoten enthält, die *nicht* Teil der dominierenden Menge sein sollen.

b) Legen Sie eine Regel fest, die angibt, welcher Knoten als nächstes betrachtet wird (Branch).

c) Führen Sie Branch-and-Bound mit Hilfe Ihrer Antworten zu a) und b) durch. Verzichten Sie auf den Einsatz des Lernmoduls. Zeichnen Sie einen vollständigen Entscheidungsbaum und geben Sie die Reihenfolge der Entscheidungen sowie die jeweiligen unteren Schranken an. Machen Sie die endgültige Lösung kenntlich.

Aufgabe 11.2 *Paging-Problem.*

Wir betrachten eine Speicherhierarchie bestehend aus einem langsamen, grossen Hauptspeicher und einem schneller, kleinen Cache-Speicher. Der Speicher ist in Speicherseiten fixer Grösse unterteilt. Der Cache-Speicher kann k und der Hauptspeicher m Seiten speichern, wobei wir annehmen, dass m viel grösser als k ist. Eine Anfrage auf eine Speicherseite kann beantwortet werden, wenn sich die Seite im schnellen Cache-Speicher befindet. Andernfalls tritt ein *Seitenfehler* auf und die angefragte Seite muss zuerst in den Cache-Speicher geladen werden (indem sie eine andere Seite im Cache ersetzt).

Das *Paging-Problem* ist ein *Online-Minimierungsproblem*, d.h. die Eingabe ist eine Folge von n Anfragen auf Speicherseiten $x_1, \ldots, x_n$, wobei die i-te Anfrage ohne Berücksichtigung der folgenden Anfragen beantwortet werden muss, und die Anzahl der Seitenfehler minimiert werden soll. Ein Algorithmus für das Paging-Problem gibt an, welche Seite im Cache bei einem Seitenfehlers ersetzt werden soll. Wir nehmen an, dass vor der ersten Anfrage die ersten k Speicherseiten im Cache gespeichert sind.

a) Zeigen Sie, dass folgende Strategie nicht kompetitiv ist:

Last In – First Out (LIFO): Es wird diejenige Seite aus dem Cache ersetzt, die zuletzt aufgenommen wurde. Beim ersten Seitenfehler wird die erste Seite im Cache verdrängt.

b) Zeigen Sie, dass die folgende Strategie k-kompetitiv ist:

First In – First Out (FIFO): Bei einem Seitenfehler wird diejenige Seite aus dem Cache ersetzt, die bereits am längsten im Cache ist. Die ersten k Seiten werden in aufsteigender Reihenfolge verdrängt.

Hinweis: Unterteilen Sie die Eingabe in aufeinanderfolgende, disjunkte Phasen. Die erste Phase endet nach dem ersten Seitenfehler, der bei FIFO auftritt. Jede weitere Phase endet nach genau k Seitenfehler bei FIFO, bzw. mit der letzten Anfrage. Zeigen Sie, dass ein optimaler Algorithmus – abgesehen von der letzten Phase – in jeder Phase mindestens einen Seitenfehler verursacht.

Aufgabe 11.3 *Rucksackproblem (**Programmieraufgabe**).*

In dieser Aufgabe soll ein Algorithmus der dynamischen Programmierung zur Lösung des *Rucksackproblems* implementiert werden. Für dieses Problem ist neben einer Menge $S = \{1, \ldots, n\}$ von n Objekten mit den Werten $v_1, \ldots, v_n \in \mathbb{N}$ und den Gewichten $w_1, \ldots, w_n \in \mathbb{N}$ noch eine Gewichtsschranke $W \in \mathbb{N}$ gegeben. Eine *Bepackung* ist eine Menge $I \subseteq S$ mit $\sum_{i \in I} w_i \leq W$, und sie besitzt den Wert $\sum_{i \in I} v_i$. Das Ziel ist es, eine Bepackung I mit maximalem Wert zu finden.

Eingabe Die erste Zeile der Eingabe enthält lediglich die Anzahl t der Testinstanzen. Danach folgen genau drei Zeilen pro Testinstanz. Die erste Zeile enthält (in dieser Reihenfolge) die Werte n und W mit $n, W \in \mathbb{N}$, $1 \leq n \leq 30$ and $1 \leq W \leq 200$. Die zweite Zeile enthält die Werte $v_1, \ldots, v_n$, und die dritte Zeile enthält die Gewichte $w_1, \ldots, w_n$. Für jedes Objekt $i \in \{1, \ldots, n\}$ sind $1 \leq v_i \leq 1000$ und $1 \leq w_i \leq 20$.

Ausgabe Für jede Testinstanz soll lediglich eine Zeile ausgegeben werden. Sie enthält den *Wert* einer optimalen Bepackung und die *Indizes* von den Objekten einer optimalen Bepackung. Um Mehrdeutigkeiten zu vermeiden, sollen diese Indizes aufsteigend sortiert ausgegeben werden.

Beispiel

Eingabe:

```
2
2 3
1 1
2 5
3 5
1 3 2
2 2 1
```

Ausgabe:

```
1 1
6 1 2 3
```

Für dieser Aufgabe, es gibt nur ein Test für 100 Punkten.

Abgabe: Am Mittwoch, den 18. Mai 2016 in Ihrer Übungsgruppe.