UNIVERSITÄT BERN

Übungsserie 10
Datenstrukturen & Algorithmen

Universität Bern

Frühling 2018

Administratives

- **Donnerstag, 10.5. Auffahrt!**
- **Abgabe** Übungsserie 10 am Mittwoch, 9.5. in der Vorlesung
  - Oder Scan per E-Mail an Hilfsassistierende
- Korrigierte Übungsserie 9 wird am 9.5. in der Vorlesung zurückgegeben
- Keine Vorbesprechung zu Übungsserie 11
  - Nutzt Forum auf Ilias

Übungsserie 10

UNIVERSITÄT BERN

- **Greedy Algorithmen**
- 3 theoretische Aufgaben
- 3 praktische Aufgaben
  - Huffman-Kodierung
  - Letzte Serie mit praktischen Aufgaben

- Poolstunde 7. Mai 1700 – 1800

Aktivitäten-Auswahl-Problem

- **Aufgabe 1** Zeige für 3 Greedy Ansätze, dass diese nicht zu optimaler Lösung führen
  - Finde für jeden Ansatz ein Beispiel
  - Skizziere Beispiele grafisch

- **Tipp für Ansatz 2** *«Wähle immer die Aktivität, die mit den wenigsten anderen überlappt»*
  - Minimales Beispiel besteht aus 9 Aktivitäten
  - Durch den Ansatz werden drei statt vier kompatible Aktivitäten gefunden

Wechselgeld-Problem

- **Aufgabe 3** Wechselgeld für `n` Rappen zusammenstellen mit möglichst wenig Münzen
  - Menge von Münzwerten: `{1,5,10,25}`
  - Greedy Algorithmus entwerfen

- **Beispiel** `n = 50`, Lösung: `{25,25}` (2 Münzen)
  - Falsch wäre: `{25,10,10,5}` (4 Münzen)

- **Tipps**
  - Wie gehst du vor, wenn du möglichst schnell Rückgeld gibst?
  - Denke nicht zu weit!

Wechselgeld-Problem

- **Aufgabe 3 a)** Greedy Algorithmus entwerfen
  - Beschreibung reicht, Pseudocode optional
  - Funktioniert für gegebene Münzmenge (`{1,5,10,25}`)
  - Funktioniert nicht für beliebige Mengen!

- Zeige, dass Algorithmus zu optimaler Lösung führt
  - Zeige **optimale Teilstruktur**
  - Zeige **gierige-Auswahl-Eigenschaft**

Wechselgeld-Problem

- **Optimale Teilstruktur** Optimale Lösung besteht aus optimalen Lösungen von Teilproblemen

- Zeige
  - **Annahme** $z(n) = \{a_1,...,a_k\}$ ist eine optimale Zerlegung von n
  - **Dann gilt** $z(n) \setminus a_j$ ist eine optimale Zerlegung von $n - a_j$

Wechselgeld-Problem

- **Gierige Auswahl** In jedem Schritt wird «gierig» ein Element zur Lösung hinzugefügt
- **Gierige Auswahl Eigenschaft** Gierige Auswahl führt zu optimaler Lösung
  - **Genauer** Erweitert man gierig *eine Teillösung*, muss die Erweiterung immer noch Teil einer optimalen Lösung sein
  - **Beispiel** `n = 75`, **Lösung** `{25,25,25}`

Gierige Auswahl: 25

Teillösung	`{25}`	→	`{25,25}`
Rest	`50`		`25`

→ `{25,25}` muss Teillösung sein!

Wechselgeld-Problem

- Beweis «Gierige Auswahl Eigenschaft» via Fallunterscheidung

- **Annahme** `L` ist optimale Teillösung, Rest `< 5`
- **Zeige** Greedy Auswahl führt zu optimaler Lösung
  - Denn Auswahl führt zu neuer Teillösung die in optimaler Lösung enthalten ist
  - Dasselbe für
    - `5 ≤` Rest `< 10`
    - `10 ≤` Rest `< 25`
    - `25 ≤` Rest

Wechselgeld-Problem

- **Aufgabe 3 b)** Gib eine Münzmenge und ein Beispiel `n` an, wo dein Greedy-Algorithmus versagt
  - Münzmenge muss für beliebige `n` eine Lösung haben

- **Aufgabe 3 c)** Finde mit Dynamischer Programmierung einen Algorithmus, der das Problem für beliebige Münzmengen und beliebige `n` löst!
  - Ausführlicher Hinweis in Aufgabenstellung

Praktische Aufgabe

- Huffmann-Code implementieren
  - Kein Basiscode!
- Implementiere Methode `prefixCode(String s)`
  - Aufbau Präfix-Code-Baum
    - Berechnet per-Zeichen-Häufigkeit
- Implementiere Methode `printCode(String s)`
  - Übersetzt `Sting s` in Huffman Code
  - Gibt 0-1-Folge in Konsole aus

Praktische Aufgabe

- **Class** `HuffmanCode`
  - `prefixCode(String s)`
  - `printCode(String s)`
  - Klassenvariablen für Baum etc.
- **Class** `Node`
  - Implementiert `Comparable` Interface
    - für `java.util.PriorityQueue`
  - **Node** `left, right, parent`
  - **int** `frequency`
  - **char** `payload` (für Blätter)
  - **…**

Praktische Aufgabe

- **Tipp** Berechnen und Speichern von Häufigkeiten
  - Speichern in `int[256]` oder `Node[256]`
    - ASCII characters können als Integer `∈ [0, 255]` interpretiert werden
      - Array mit char als Index ansprechen:

```
Node array[];
array['a'] = ...
```

  - `Java.Util.Hashtable<Character,Node>`

- **Tipp** Aufbau des Baumes
  - `Java.Util.PriorityQueue<Node>`
    - Siehe Pseudocode im Buch Kapitel 16 S. 434

Praktische Aufgabe

UNIVERSITÄT BERN

- **Tipp** Code-Rekonstruktion
- Finde Code für ein Zeichen, gegeben ein Präfix-Code-Baum
- Traversiere den Baum von unten nach oben!
  - Finde erst das entsprechende Blatt (Startpunkt)
  - Gehe zur Wurzel, Fallunterscheidung
    - Linkes Kind: **pre**pend 0
    - Rechte Kind: **pre**pend 1
    - **Beispiel** code(**c**) = **1 0 0**
  - Wie Startpunkt finden?
    - Speicher Blätter zusätzlich in Array `Node[256]`