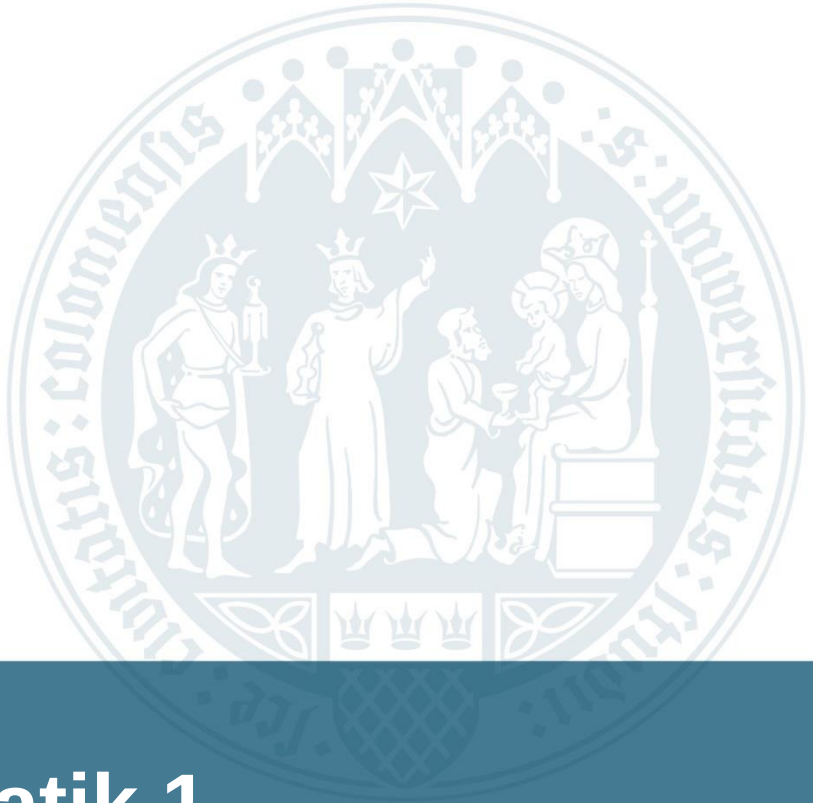
Universität zu Köln



Grundzüge der Informatik 1

Vorlesung 14 - flipped classroom

Gierige Algorithmen

Entwurfsprinzip „Gierige Algorithmen“

- Ziel: Lösung eines Optimierungsproblems
- Herangehensweise: Konstruiere Lösung Schritt für Schritt, indem immer ein einfaches „lokales“ Kriterium optimiert wird
- Vorteil: Typischerweise einfache, schnelle und leicht zu implementierende Algorithmen

Beobachtungen

- Gierige Algorithmen optimieren einfaches lokales Kriterium
- Dadurch werden nicht alle möglichen Lösungen betrachtet
- Dies macht die Algorithmen oft schnell
- Je nach Problem und Algorithmus kann die optimale Lösung übersehen werden

Gierige Algorithmen

Aufgabe 1

- Beim SetCover Problem besteht die Eingabe aus n Teilmengen $S_1,\ldots,S_n$ der Menge {1,…,m}. Dabei soll jedes Element aus {1,…,m} in mindestens einer Teilmenge vorkommen.
- Eine gültige Lösung für das SetCover Problem ist eine Menge I von Indizes, so dass $\bigcup_{i\in I} S_i$={1,…,m} ist
- Gesucht ist eine kleinstmögliche gültige Lösung, d.h. eine gültige Lösung, die |I| minimiert
- Was ist eine optimale Lösung für die folgenden Teilmengen?
- {1,2,4,7}, {2,3,5}, {6}, {3,5,6,8}, {1,2,3}
- Entwickeln Sie einen gierigen Algorithmus für das SetCover Problem.
- Zeigen Sie entweder, dass Ihr Algorithmus das Problem immer exakt löst oder geben Sie ein Gegenbeispiel an.

Gierige Algorithmen

Aufgabe 1

- Beim SetCover Problem besteht die Eingabe aus n Teilmengen $S_1,\ldots,S_n$ der Menge {1,…,m}. Dabei soll jedes Element aus {1,…,m} in mindestens einer Teilmenge vorkommen.
- Eine gültige Lösung für das SetCover Problem ist eine Menge I von Indizes, so dass $\bigcup_{i\in I} S_i$={1,…,m} ist
- Gesucht ist eine kleinstmögliche gültige Lösung, d.h. eine gültige Lösung, die |I| minimiert
- Was ist eine optimale Lösung für die folgenden Teilmengen?
- {1,2,4,7}, {2,3,5}, {6}, {3,5,6,8}, {1,2,3}
- Entwickeln Sie einen gierigen Algorithmus für das SetCover Problem.
- Zeigen Sie entweder, dass Ihr Algorithmus das Problem immer exakt löst oder geben Sie ein Gegenbeispiel an.

Gierige Algorithmen

GreedySetCover(S_1,…,S_n, m)

1. Sei T = {1,…,m}
2. **while** T≠∅ **do**
3. Sei i ein Index aus {1,…,n}, der $|T \cap S_i|$ maximiert
4. I = I ∪ {i}
5. T= T \ S_i
6. **return** I

Gierige Algorithmen

GreedySetCover(S_1,…,S_n, m)

1. Sei T = {1,…,m}
2. **while** T≠∅ **do**
3. Sei i ein Index aus {1,…,n}, der $|T \cap S_i|$ maximiert
4. I = I ∪ {i}
5. T= T \ S_i
6. **return** I

Beispiel

- {1,2,3,4}, {5}, {6}, {1,3,5}, {2,4,6}
- Jede Lösung von GreedySetCover besteht aus 3 Mengen
- Die optimale Lösung ist {1,3,5}, {2,4,6}

Gierige Algorithmen

Aufgabe 2

- Bei der Vorlesungsplanung müssen ALLE Vorlesungen auf Hörsäle abgebildet werden, so dass sich keine zwei Vorlesungen überschneiden. Wir gehen davon aus, dass jede Vorlesung in jedem Hörsaal abgehalten werden kann. Gesucht ist eine Zuordnung der Vorlesungen zu den Hörsälen, die die Anzahl der genutzten Hörsäle minimiert.

(a) Finden Sie für folgenden Vorlesungen eine Zuordnung auf möglichst wenige Hörsäle:

- 11-13, 12-14, 9-20 8-11, 9-17, 15-20, 18-19, 14-16

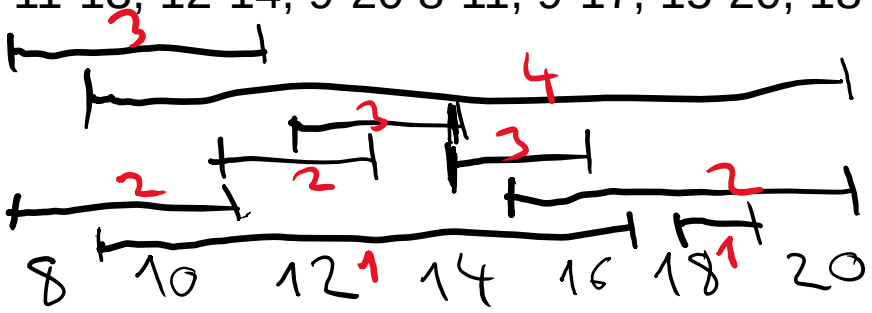
Gierige Algorithmen

Aufgabe 2

- Bei der Vorlesungsplanung müssen ALLE Vorlesungen auf Hörsäle abgebildet werden, so dass sich keine zwei Vorlesungen überschneiden. Wir gehen davon aus, dass jede Vorlesung in jedem Hörsaal abgehalten werden kann. Gesucht ist eine Zuordnung der Vorlesungen zu den Hörsälen, die die Anzahl der genutzten Hörsäle minimiert.

(a) Finden Sie für folgenden Vorlesungen eine Zuordnung auf möglichst wenige Hörsäle:

- 11-13, 12-14, 9-20 8-11, 9-17, 15-20, 18-19, 14-16



Gierige Algorithmen

Aufgabe 3

- Bei der Vorlesungsplanung müssen ALLE Vorlesungen auf Hörsäle abgebildet werden, so dass sich keine zwei Vorlesungen überschneiden. Wir gehen davon aus, dass jede Vorlesung in jedem Hörsaal abgehalten werden kann. Gesucht ist eine Zuordnung der Vorlesungen zu den Hörsälen, die die Anzahl der genutzten Hörsäle minimiert.

(a) Finden Sie für folgenden Vorlesungen eine Zuordnung auf möglichst wenige Hörsäle:

- 11-13, 12-14, 9-20 8-11, 9-17, 15-20, 18-19, 14-16

(b) Entwickeln Sie einen Algorithmus zur Bestimmung des maximalen Bedarfs. Die Start- und Endzeitpunkte sind dabei als Felder S und E gegeben, so dass S[i] den Startzeitpunkt von Vorlesung i enthält und E[i] den Endzeitpunkt. Die Anzahl Vorlesungen wird mit n bezeichnet. Können Sie Laufzeit O(n log n) erreichen?

Gierige Algorithmen

Beobachtung

- Für einen Zeitpunkt t bezeichne der *Bedarf* die Anzahl der Vorlesungen, die gleichzeitig zu Zeitpunkt t stattfinden sollen.
- Eine untere Schranke ergibt sich aus dem maximalen Bedarf über alle Zeitpunkte t.

Gierige Algorithmen

Verbund Data:
Zeitpunkt
StartEnd

BerechneBedarf(S,E,n)

1. Sei F[1..2n] ein Feld über Verbundtyp Data
2. **for** i=1 **to** n **do**
3. F[i] = (S[i], 0); F[n+i] = (E[i], 1)
4. Sortiere F nach Zeitpunkten
5. Bedarf=0
6. maxBedarf=0
7. **for** i=1 **to** 2n **do**
8. **if** F[i].StartEnd = 0 **then** Bedarf = Bedarf +1
9. **if** F[i].StartEnd = 1 **then** Bedarf = Bedarf -1
10. **if** Bedarf>maxBedarf **then** maxBedarf = Bedarf
11. **return** maxBedarf

Gierige Algorithmen

Aufgabe 4

- Bei der Vorlesungsplanung müssen ALLE Vorlesungen auf Hörsäle abgebildet werden, so dass sich keine zwei Vorlesungen überschneiden. Wir gehen davon aus, dass jede Vorlesung in jedem Hörsaal abgehalten werden kann. Gesucht ist eine Zuordnung der Vorlesungen zu den Hörsälen, die die Anzahl der genutzten Hörsäle minimiert.

(a) Entwickeln Sie einen gierigen Algorithmus zur Berechnung einer optimalen Zuordnung. Verwenden Sie den von Ihnen entwickelten Algorithmus zur Berechnung einer unteren Schranke

Gierige Algorithmen

Hörsäle(S,E,n)

1. new array Belegung[1..n]
2. maxBedarf = BerechneBedarf(S,E,n)
3. Sortiere S nach Startzeiten
4. **for** i=1 **to** n **do**
5. Wähle einen Hörsaal h aus {1,…,maxBedarf}, der zum Zeitpunkt S[i] nicht belegt ist
6. Belegung[i] = h
7. **return** Belegung