Ü1

Aus der Vorlesung: Raumplanungsproblem

Gegeben: Intervalle $I_1, \dots, I_n$ mit
$I_i = (s_i, e_i)$ für alle $1 \leq i \leq n$

Gesucht: Auswahl S an Intervallen, die sich paarweise nicht überlappen, mit $|S|$ maximal
($|S|$: #Elemente in S)

Greedy-Algorithmus

1. Sortiere Intervalle nach deren Ende → Permutation π
2. $S \leftarrow \{I_{\pi(1)}\}$, $last \leftarrow \pi(1)$
3. For $i = 2$ to n do
   if $(e_{last} \leq s_{\pi(i)})$ then
   $S \leftarrow S \cup \{I_{\pi(i)}\}$
   $last \leftarrow \pi(i)$
4. Return S

Laufzeit: $O(n \log n)$

Sehr ähnlich zum Greedy-Algorithmus für Frac. Knapsack.

Diese Art von Problemen haben eine gewisse Struktur
↳ Matroid

Auf Matroiden lässt sich Greedy anwenden!

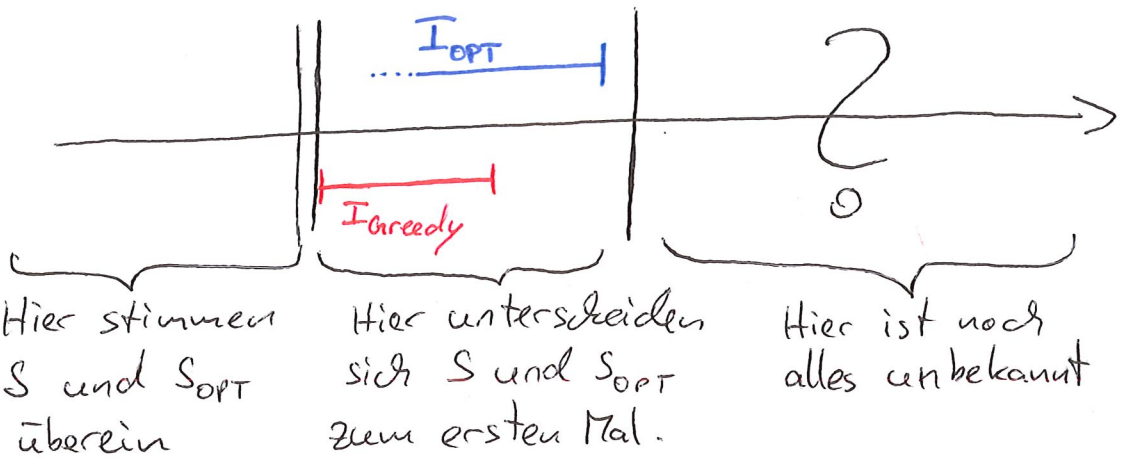
Beweis: selbst!

Korrektheit:

Angenommen, es gibt eine bessere Lösung S_{OPT},
also $|S_{OPT}| > |S|$.

Dann gibt es ein $I_{Greedy} \in S$, welches das erste Intervall ist, das nicht in S_{OPT} liegt. Sei andersherum $I_{OPT} \in S_{OPT}$ das erste

Intervall, das nicht in S liegt. Es muss gelten $e_{Greedy} < e_{OPT}$



Beobachtung: I_{Greedy} und I_{OPT} müssen sich überlappen!
Warum? Wir könnten sonst I_{Greedy} in S_{OPT} aufnehmen und S_{OPT} somit verbessern ↯

Wir setzen außerdem: Entferne I_{OPT} aus S_{OPT} und füge I_{Greedy} hinzu!

$(S_{OPT} \setminus I_{OPT}) \cup I_{Greedy}$ ist eine gülte Lösung und ist optimal.

Dieser Vorgang lässt sich wiederholen

$\Rightarrow S \subseteq S_{OPT}$ (Jedes Element aus S liegt auch in einer optimalen Lsg.)

Falls nun S_{OPT} noch weitere Intervalle enthält, dann überlappen die sich mit keinem Intervall aus S. Diese wären nach Algorithmus aufgenommen worden!

$\Rightarrow$ S ist optimal!

Weiteres Problem:
Teile die Intervalle in so wenig Räume wie möglich auf.

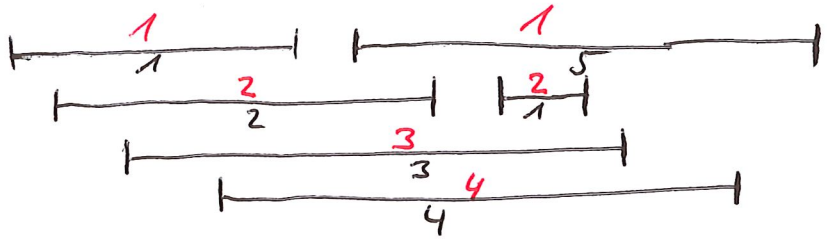
Formal:
Gegeben: Intervalle $I_1, \dots, I_n$ mit $I_i = (s_i, e_i)$ für alle $1 \le i \le n$
Gesucht: $R: \{I_1, \dots, I_n\} \to \mathbb{N} := \{1, 2, 3, \dots\}$, sodass

$R(I_i) \neq R(I_j)$ für $i \neq j$, falls I_i und I_j sich überlappen

und $\max_{1 \le i \le n} (R(I_i))$ minimal

Idee 1: Greedy-Algorithmus mehrfach ausführen und die Lösungen je einem Raum zuweisen

Ist das korrekt/optimal? Nein!

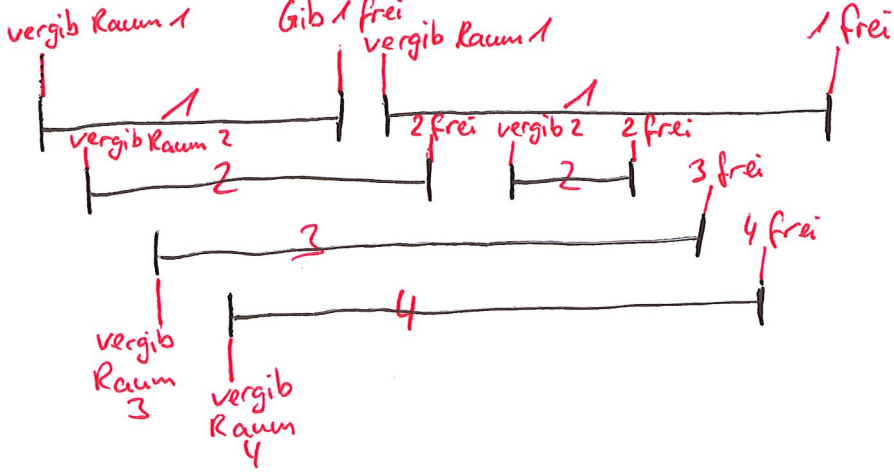


Idee 1: 5

OPT: 4

Warum scheitert das? Wichtig ist nicht nur das Ende, sondern auch der Anfang.
⇒ Arbeite jedes Intervall zweimal ab: wenn es startet, vergib den kleinsten Raum, wenn es endet, gib den Raum frei. (Idee 2)

Das Beispiel von oben:



Idee 2: 4

Als Pseudocode:

1. Sortiere Intervalle nach deren Anfang
   → Permutation $\pi_s(1), \dots, \pi_s(n)$
2. Sortiere Intervalle nach deren Ende
   → Permutation $\pi_e(1), \dots, \pi_e(n)$
3. Array $R[1..n]$ mit 0 initialisiert (enthält Raumnummer für jedes Intervall)

   Array $Avail[1..n]$ mit true initialisiert (enthält Boolean, ob Raum frei ist)
4. $i, j := 1$
5. while $(i \leq n)$ do ← Solange noch ein Intervall mit Startpunkt existiert.
6. ⎸ if $(s_{\pi_s(i)} < e_{\pi_e(j)})$ then ← Als nächstes Beginnt ein Intervall
7. ⎸ ⎸ $k := 1$
8. ⎸ ⎸ while $(Avail[k] = false)$ do } Suche freien Raum mit kleinster Nummer
9. ⎸ ⎸ ⎸ $k := k+1$
10. ⎸ ⎸ $R[\pi_s(i)] := k$, $Avail[k] := false$, $i := i+1$
11. ⎸ else ← Als nächstes Endet ein Intervall
12. ⎸ ⎸ $Avail[R[\pi_e(j)]] := true$, $j := j+1$ Gib Raum frei
13. Return R

Dieser Algorithmus ist optimal.

Beweis:
Zunächst suchen wir eine untere Schranke für das Optimum.
Eine untere Schranke ist die Anzahl an Intervallen, die sich zu einem Zeitpunkt paarweise überlappen. Sei x das Maximum.
Offensichtlich gilt $OPT \geq x$
Wir zeigen, dass unser Algorithmus mit x Räumen auskommt und somit optimal ist.

Für ein Intervall benötigen wir nur einen Raum und $\chi = 1$. Das passt also.

Angenommen, unser Algorithmus benötigt maximal χ Räume für die ersten $i-1$ Intervalle.

Betrachte das Intervall $I_{\pi_s(i)}$, welches den Raum k bekommt.

Fall 1: k wurde schon einmal vergeben. Nach Voraussetzung ist $k \leq \chi$.

Fall 2: k wurde noch nicht vergeben.

Zum Zeitpunkt $s_{\pi_s(i)}$ müssen sich k Intervalle gegenseitig überlappen (andernfalls wäre zuvor ein kleinerer Raum freigegeben worden). Das bedeutet widerum, dass $\chi \geq k$ sein muss.

$\Rightarrow$ In beiden Fällen haben wir auch für das i-te Intervall maximal χ Räume benutzt. □

Laufzeit (ohne Beweis)

Der Algorithmus besitzt eine Laufzeit von

$$O(n \log n + m),$$

Hier steckt die Suche nach dem freien Raum drin.

wobei m die Anzahl an Überlappungen ist.