Wiederholung

- Jeder vergleichbasierte Sortierer benötigt $\Omega(n \log n)$

Dynamische Programmierung

- Subprobleme hängen voneinander ab (Speicherung)
  - Beginne bei trivialen Subproblemen
  - Kombiniere Lösungen von Subproblemen

- Beispiele für Dynamische Programmierung
  - Berechnung von Binomialkoeffizienten
  - Minimierungsproblem Matrizenmultiplikation
  - Maximierungsproblem Rucksack

Das Schedulingproblem

Strategie Dynamische Programmierung

- Bottom up:
  - Optimiere in jedem Schritt, abhängig von der Lösung der Subprobleme.

Strategie Greedy

- Top down:
  - Bestimmen in jedem Schritt (gierig) eine lokal optimale Lösung.
  - Hoffnung: Führt zu globalem Optimum.

Scheduling Problem

- Gegeben: Menge S={1,…,n} von Veranstaltungen mit
  Startzeiten s_i und Endzeiten f_i, $1 \leq i \leq n$.
  - Veranstaltungen i, j kompatibel $\Leftrightarrow$ $[s_i,f_i)$, $[s_j,f_j)$ nicht überlappend
    $\Leftrightarrow s_j \geq f_i$ oder $s_i \geq f_j$
- Gesucht: $J \subseteq S$ paarweise kompatibler Veranstaltungen, $|J|$ maximal.

Greedy-Lösung von Scheduling

Annahme: f_i seien aufsteigend sortiert.

Algorithmus Greedy-Scheduling

Eingabe: s[1..n], f[1..n] mit $f[1] \leq f[2] \leq \ldots \leq f[n]$

1. $J \leftarrow \{1\}$; $j \leftarrow 1$;
2. for $i \leftarrow 2$ to n
   1. if $(s_i \geq f_j)$ then $J \leftarrow \{i\}$; $j \leftarrow i$;

Ausgabe: J

- Invariante: j ist maximales Element in J.
  - Schritt 2.1: Falls i zu j kompatibel ist, dann auch zu allen anderen Element in J, denn $f_j \geq f_k$ für alle $k \in J$.
- Laufzeit: $\mathcal{O}(n)$

Optimalität Greedy-Scheduling

Satz: Greedy-Scheduling liefert eine optimale Lösung des Scheduling-Problems.

Zeigen:

1) Es gibt eine optimale Lösung J mit $1 \in J$.
2) $J \setminus \{1\}$ ist eine optimale Lösung für Subproblem $S_1 = \{i \in S \mid s_i \geq f_1\}$

Satz folgt per Induktion über die Anzahl der gewählten Elemente.

ad 1) Annahme: Sei J‘ optimale Lösung mit $1 \notin J$‘.

- Sei $k \neq 1$ minimales Element in J‘.
- Es gilt $f_k \geq f_1$, d.h. 1 ist kompatibel zu jedem Element in J‘.
- Definiere $J := J' \setminus \{k\} \cup \{1\}$.
- Wegen $|J| = |J'|$ ist J optimal.

ad 2) Annahme: Sei J‘ eine bessere Lösung für S_1 als $J \setminus \{1\}$, d.h. $|J'| > |J \setminus \{1\}|$

- 1 ist kompatibel zu allen Elementen in S_1, d.h. $J' \cup \{1\}$ ist gültige Lösung für S.
  $\Rightarrow |J' \cup \{1\}| > |J|$ (Widerspruch zur Optimalität von J)

Struktur von Greedy-Problemen

Optimale Lösungen mit Greedy erfordern

- **Optimalität der Greedy-Wahl**
  - z.B. es gibt optimale Lösung, die 1 enthält.
  - Unterschied zur Dyn. Programmierung:
    Wahl hängt bei Greedy **nicht** von der Lösung der Subprobleme ab.
  - Insbesondere: Wahl kann nur von bisherigen Entscheidungen abhängen.
    Top-down Ansatz im Gegensatz zur Bottom-up Lösung

- **Optimalität der Subprobleme**
  - z.B. J‘ = J\{1} ist optimal für S‘.
  - Jede optimale Lösung beinhaltet optimale Lösungen der Subprobleme.

Bemerkungen

- Nicht jeder Greedy-Ansatz liefert Erfolg.
- z.B. nicht erfolgreich: Wähle kompatible Veranstaltung mit
  - kürzester Dauer
  - kleinster Überlappung (Übungsaufgabe)

- Scheduling-Problem mittels Dynamischer Programmierung
  - schlechtere Laufzeit (Übungsaufgabe)

Greedy vs. Dynamische Programmierung

Maximierungsproblem Rucksackproblem Π_R:

Gegeben:

- n Gegenstände mit Gewichten $w_i \in \mathbb{N}$ und Profiten $p_i \in \mathbb{N}$, i=1,…,n.
- Kapazitätsschranke B

Gesucht:

- $\max_{J \subseteq [n]} \{\sum_{j \in J} p_j \mid \sum_{j \in J} w_j \leq B\}$ optimale Belegung des Rucksacks

Rationale Variante Π'_R:

- Man darf Bruchteile der Gegenstände in den Rucksack packen.

Beide Probleme besitzen Optimalität der Subprobleme:

- Π_R: Entscheide, ob i in optimaler Lösung L ist.
  - $i \notin L$: Optimale Lösung des Subproblems ohne i.
  - $i \in L$: Optimale Lösung des Subproblems ohne i mit Schranke $B-w_i$.
- Π'_R: Nimm Gewicht $w \leq w_i$ des Gegenstandes i.
  - Optimale Lösung des Subproblems ohne i mit Schranke W-w.

Optimalität der gierigen Wahl

Π'_R optimal mit Greedy-Strategie lösbar:

- Sortiere $a_i := p_i/w_i$ absteigend.
- for i $\leftarrow$ 1 to n
  - Nimm von i denjenigen Bruchteil, der noch in den Rucksack passt.

- Korrektheit: Algorithmus liefert optimale Lösung. (Übungsaufgabe)
- Laufzeit: $\mathcal{O}$(n logn)

Strategie funktioniert nicht für Π_R:

- Gegenbeispiel: $(w_1,p_1)=(1,3)$, $(w_2,p_2)=(2,4)$, $(w_3,p_3)=(4,4)$ und B = 6.
- Greedy wählt Gegenstand 1 und 2. Optimal ist 2 und 3.

Minimale Spannbäume MST

Problem minimaler Spannbaum (MST=minimum spanning tree):

Gegeben:

- G=(V,E), zusammenhängend, ungerichtet
- Gewichtsfunktion w: $E \rightarrow \mathbb{R}$

Gesucht :

- Spannbaum $T=(V,E_T)$ mit minimalem Gewicht $w(T) = \sum_{e \in E_T} w(e)$.

Greedy-Strategie:

- Sortiere die Kanten aufsteigend nach Gewicht.
- Wähle nächste Kante, die keinen Kreis schließt.