# Greedy-Strategie

## **Definition** Paradigma Greedy

Der Greedy-Ansatz verwendet die Strategie

1. **Top-down** Auswahl: Bestimme in jedem Schritt eine lokal optimale Lösung, so dass man eine global optimale Lösung erhält.

## **Definition** Kompatible Veranstaltungen

$S = [n]$ heißt eine Menge von *Veranstaltungen*, falls jedes $i \in [n]$ eine Startzeiten $s_i$ und eine Endzeiten $f_i \geq s_i$ besitzt. Wir nennen $i, j \in S$ *kompatibel*, falls $[s_i, f_i)$ und $[s_j, f_j)$ nicht-überlappend sind. $J \subseteq S$ heißt *kompatibel*, falls je zwei verschiedene Elemente aus $J$ kompatibel sind.

## **Problem** Scheduling

**Gegeben:** S=[n] Menge von Veranstaltungen
**Gesucht:** $J \subseteq S$ kompatibel mit maximaler Größe $|J|$

# Greedy Lösung für das Scheduling-Problem

**Annahme:** Die Veranstaltungen seien aufsteigend nach $f_i$ sortiert.

## Algorithmus GREEDY-SCHEDULING

EINGABE: $s[1 \ldots n], f[1 \ldots n]$ mit $f[1] \leq \ldots \leq f[n]$

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: $J$

**Anmerkung:**

- **Korrektheit:** Wir zeigen zunächst, dass $J$ kompatibel ist.
- Invariante: $j$ ist ein Element maximaler Endzeit $f_j$ in $J$.
- Schritt 2.1: Falls $i$ zu $j$ kompatibel ist, dann auch zu allen anderen Elementen in $J$, da $f_j \geq f_k$ für alle $k \in J$.
- **Laufzeit:** $\mathcal{O}(n)$.

# Optimalität der Greedy-Lösung

**Satz** Optimalität von GREEDY-SCHEDULING

GREEDY-SCHEDULING liefert ein kompatibles $J$ maximaler Größe.

**Beweis:** Beweisstruktur

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

Der Satz folgt per Induktion über die Anzahl der gewählten Elemente.

- **ad 1:** Sei $J'$ eine optimale Lösung mit $1 \notin J'$ und $k$ minimal in $J'$.
- D.h. $f_k \geq f_1$ bzw. 1 ist kompatibel zu jedem Element in $J' \setminus \{k\}$.
- Damit ist $J = J' \setminus \{k\} \cup \{1\}$ eine optimale Lösung mit $1 \in J$.
- **ad 2:** Annahme: Sei $J$ optimal für $S$ mit $1 \in J$. Sei $J'$ eine bessere Lösung für $S_1$ als $J \setminus \{1\}$, d.h. $|J'| > |J| - 1$.
- $J' \cup \{1\}$ ist kompatibel, d.h. eine gültige Lösung für $S$.
- Damit folgt $|J' \cup \{1\}| = |J'| + 1 > |J|$.
  (Widerspruch zur Optimalität von $J$)

# Eigenschaften von Greedy-Problemen

**Struktur:** Optimale Lösungen mit Greedy-Ansatz erfordern

1. Optimalität der Greedy-Wahl
   - Es existiert ein optimale Lösung, die das lokale Optimimum enthält.
   - Gierige Wahl hängt *nicht* von der Lösung der Subprobleme ab.
   - D.h. die Wahl hängt nur von bisherigen Entscheidungen ab. (Top-down Ansatz)
2. Optimalität der Subprobleme
   - Optimale Lösung beinhaltet optimale Lösungen der Subprobleme.

**Anmerkungen:**

- Nicht jeder Greedy-Ansatz liefert eine optimale Lösung.
- Nicht erfolgreich beim Scheduling sind z.B. die gierige Wahl von kürzester Dauer und kleinster Überlappung. (Übungsaufgabe)
- Das Scheduling-Problem kann auch mittels Dynamischer Programmierung gelöst werden. (Übungsaufgabe)
- Diese Lösung liefert aber eine schlechtere Laufzeit.

# Greedy versus Dynamische Programmierung

## **Problem** Rucksack $\Pi_R$

**Gegeben:** $n$ Gegenstände mit Gewichten und Profiten $w_i, p_i \in \mathbb{N}$, Kapazitätsschranke $B$

**Gesucht:** $\alpha \in \{0,1\}^n$ mit Profit $\max_\alpha\{\sum_{i=1}^n \alpha_i p_i \mid \sum_{i=1}^n \alpha_i w_i \leq B\}$.

## **Problem** Rationale Variante Rucksack $\Pi'_R$

**Gegeben:** $n$ Gegenstände mit Gewichten und Profiten $w_i, p_i \in \mathbb{N}$, Kapazitätsschranke $B$

**Gesucht:** $\alpha \in [0,1]^n$ mit Profit $\max_\alpha\{\sum_{i=1}^n \alpha_i p_i \mid \sum_{i=1}^n \alpha_i w_i \leq B\}$.

**Anmerkung:** Beide Probleme besitzen Optimalität der Subprobleme.

- $\Pi_R$: Entscheide, ob $i$ in optimaler Lösung $L$ ist, d.h. ob $\alpha_i = 1$.
  - Falls $i \notin L$: Bestimme optimale Lösung des Subproblems ohne $i$.
  - Falls $i \in L$: Bestimme optimale Lösung ohne $i$ mit Schranke $B - w_i$.
- $\Pi'_R$: Sei Bruchteil $\alpha_i$ des Gegenstands $i$ in optimaler Lösung.
  - Bestimme optimale Lösung ohne $i$ mit Schranke $B - \alpha_i w_i$.

# (Nicht-)Optimalität der Greedy-Wahl

**Algorithmus** GREEDY-RATIONALER-RUCKSACK

EINGABE: $n$, $p_1, \ldots, p_n$, $w_1, \ldots, w_n$, $B$

1. Sortiere die Elemente absteigend nach $\frac{p_i}{w_i}$.
2. For $j \in [n]$ in Reihenfolge absteigender Quotienten $\frac{p_i}{w_i}$
   1. Nimm von $j$ maximalen Bruchteil $\alpha_j$, der in den Rucksack passt.

AUSGABE: $\alpha \in [0,1]^n$

- **Korrektheit:** Algorithmus liefert eine optimale Lösung. (Übung)
- **Laufzeit:** $\mathcal{O}(n \log n)$.

**Anmerkung:**

- Greedy-Strategie funktioniert nicht für $\Pi_R$. Gegenbeispiel: $(w_1, p_1) = (1,3)$, $(w_2, p_2) = (2,4)$, $(w_3, p_3) = (4,4)$ und $B = 6$.
- Greedy wählt die Gegenstände 1 und 2. Optimal ist aber 2 und 3.

# Greedy-Algorithmus Minimaler Spannbaum (MST)

## **Definition** Minimaler Spannbaum

Sei $G = (V, E)$ ein ungerichteter Graph und $w : E \to \mathbb{N}$. Das *Gewicht* $w(T)$ eines Spannbaums $T = (V, E_T)$ ist $\sum_{e \in E_T} w(e)$. Ein *minimaler Spannbaum* (MST) ist ein Spannbaum minimalen Gewichts.

## **Algorithmus** KRUSKAL

EINGABE: $G = (V, E)$ mit $w : E \to \mathbb{N}$

1. $E_T \leftarrow \emptyset$
2. Sortiere die Kanten aufsteigend nach Gewicht
3. For $e \in E$ in Reihenfolge aufsteigenden Gewichts
   1. If $((V, E_T \cup \{e\})$ ist kreisfrei) then $E_T \leftarrow E_T \cup \{e\}$.

AUSGABE: MST $T = (V, E_T)$

- **Korrektheit:** $T$ ist ein Spannbaum. Minimalität s. nächste Folie.
- **Laufzeit:** $\mathcal{O}(|E| \log |E|)$.

# Optimalität von KRUSKAL

**Satz** MST-Eigenschaft von KRUSKAL

KRUSKAL berechnet einen minimalen Spannbaum von $G$.

**Beweis:**

- Seien $e_1, \ldots, e_m \in E$ aufsteigend nach Gewicht sortiert.
- **Zeigen:** Es existiert ein MST, der $e_1$ enthält.
- Sei $T$ ein MST, der $e_1$ nicht enthält. Wir fügen $e_1$ zu $T$ hinzu.
- $e_1$ schließt einen Kreis in $T$. Entfernen einer beliebigen Kreiskante $e$ liefert wiederum einen Spannbaum $T'$.
- Wegen $w(e_1) \leq w(e')$ folgt $w(T') \leq W(T)$.

# Beweis der Optimalität

**Beweis:** Fortsetzung

- Sei nun $T$ ein optimaler Spannbaum mit Kante $e_1$.
- Entfernen von $e_1$ liefert zwei ZHKs $G[V_1]$ und $G[V_2]$.
- $T$ induziert Spannbäume $T_1$, $T_2$ für $G[V_1]$ und $G[V_2]$.
- Annahme: Sei $T'$ ein MST für $G[V_1]$ mit $w(T') < w(T)$.
- Wir ersetzen in $T$ die Kanten von $T_1$ durch die Kanten von $T'$.
- Dies liefert einen Spannbaum von $G$ mit Gewicht kleiner als $w(T)$. (Widerspruch zur Minimalität von $T$)

# Lösen von Rekursionsgleichungen

**Ziel:** Lösen von Rekursionsgleichungen der Form

1. $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
2. $x_n = a_1 x_{n-1} + \ldots + a_k x_{n-k} + b_k$

Beide Gleichungstypen treten häufig in der Laufzeitanalyse auf.

# Das Master-Theorem

## **Satz** Master-Theorem

Sei $T(n) = aT(\frac{n}{b}) + f(n)$ mit $a, b \geq 1$. Dann gilt:

1. **Fall 1:** $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$, $\epsilon > 0$ :
$$T(n) = \Theta(n^{\log_b a})$$
2. **Fall 2:** $f(n) = \Theta(n^{\log_b a})$ :
$$T(n) = \Theta(n^{\log_b a} \log n)$$
3. **Fall 3:** $f(n) = \Omega(n^{\log_b a + \epsilon})$ und $af(\frac{n}{b}) \leq cf(n)$ für ein $c < 1$:
$$T(n) = \Theta(f(n)).$$

**Anmerkung:**

- Man kann zeigen, dass $\frac{n}{b}$ sowohl als $\lfloor \frac{n}{b} \rfloor$ als auch als $\lceil \frac{n}{b} \rceil$ interpretiert werden kann, ohne die Analyse zu beeinflussen.

# Beispiele bereits bekannter Anwendungen

**Mergesort und FFT:**

- Rekursionsgleichung $T(n) = 2T(\frac{n}{2}) + \Theta(n)$.
- D.h. wir erhalten die Parameter $a = b = 2$ und $f(n) = \Theta(n)$.
- Damit gilt $n^{\log_b a} = n$, d.h. wir sind in Fall 2: $f(n) = \Theta(n^{\log_b a})$.
- Es folgt mit Master-Theorem $T(n) = \Theta(n \log n)$.

**Karatsuba:**

- Rekursionsgleichung $T(n) = 3T(\frac{n}{2}) + \Theta(n)$.
- D.h. wir erhalten die Parameter $a = 3$, $b = 2$ und $f(n) = \Theta(n)$.
- Es gilt $n^{\log_b a} = n^{\log_2 3} \approx n^{1.58}$.
- Damit Fall 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ für $0 < \epsilon < \log_2 3 - 1$.
- Master-Theorem liefert $T(n) = \Theta(n^{\log_2 3})$.

# Weitere Beispiele

**Binäre Suche:**

- Rekursionsgleichung $T(n) = T(\frac{n}{2}) + \Theta(1)$.
- Wir erhalten Parameter $a = 1$, $b = 2$ und $f(n) = \Theta(1)$.
- Wegen $n^{\log_b a} = 1$ gilt Fall 2: $f(n) = \Theta(n^0) = \Theta(n^{\log_b a})$.
- Das Master-Theorem liefert $T(n) = \Theta(n^{\log_b a} \log n) = \Theta(\log n)$.

**Beispiel für Fall 3:**

- Verwende Rekursionsgleichung $T(n) = 3T(\frac{n}{4}) + n \log n$.
- Es gilt $n^{\log_b a} = n^{\log_4 3} \approx n^{0.79}$.
- D.h. wir sind in Fall 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ für $0 < \epsilon \leq 1 - \log_4 3$.
- Ferner gilt $af(\frac{n}{b}) = 3\frac{n}{4} \log(\frac{n}{4}) \leq \frac{3}{4} n \log n = cf(n)$ für $c = \frac{3}{4} < 1$.
- Das Master-Theorem liefert $T(n) = \Theta(f(n)) = \Theta(n \log n)$.