# Kodierung der Eingabe

- Erinnerung: Zeitkomplexität $T_M(n)$ ist eine Funktion in $|w| = n$.
- Benötigen geeignete Kodierung der Eingabe $w$.
- Kodierung einer Zahl $n \in \mathbb{N}$
  - Verwenden Binärkodierung $\mathrm{bin(n)}$ mit Eingabelänge $\Theta(\log n)$.
- Kodierung eines Graphen $G = (V, E)$
  - Kodieren Knotenanzahl $n$ unär, d.h. $|V| = n$.
  - $m$ Kanten mit Adjazenzliste $|E| = m$ oder Adjazenzmatrix $|E| = \Theta(n^2)$.

**Bsp:**

- PFAD:= $\{(G, s, t) \mid G$ ist Graph mit Pfad von $s$ nach $t.\} \in \mathcal{P}$.
  - Starte Breitensuche in $s$.
  - Falls $t$ erreicht wird, akzeptiere. Sonst lehne ab.
  - Laufzeit $\mathcal{O}(|V| + |E|)$, d.h. linear in der Eingabelänge von $G$.
- TEILERFREMD:= $\{(x, y) \mid \gcd(x, y) = 1\} \in \mathcal{P}$.
  - Berechne mittels Euklidischem Algorithmus $d = \gcd(x, y)$.
  - Falls $d = 1$, akzeptiere. Sonst lehne ab.
  - $\mathcal{O}(\log^2(\max\{x, y\}))$, quadratisch in $|x| = \Theta(\log x)$, $|y| = \Theta(\log y)$.

# Optimierungsvariante vs Entscheidungsvariante

RUCKSACK$_{opt}$

- Gegeben: $n$ Gegenstände mit Gewichten $W = \{w_1, \ldots, w_n\} \subset \mathbb{N}$ und Profiten $P = \{p_1, \ldots, p_n\} \subset \mathbb{N}$, Kapazität $b \in \mathbb{N}$.
- Gesucht: $I \subseteq [n] : \sum_{i \in I} w_i \leq B$, so dass $\sum_{i \in I} p_i$ maximiert wird.

Sprache RUCKSACK:

$$\text{RUCKSACK} := \{(W, P, b, k) \mid \exists I \subseteq [n] : \textstyle\sum_{i \in I} w_i \leq B \text{ und } \sum_{i \in I} p_i \geq k\}.$$

**Naiver Algorithmus** zum Entscheiden von RUCKSACK

1. Für alle $I \subseteq [n]$:
   1. Falls $\sum_{i \in I} w_i \leq B$ und $\sum_{i \in I} p_i \geq k$, akzeptiere.
2. Lehne ab.

- Prüfung von $2^n$ vielen Untermengen in Schritt 1.
- Eingabegrößen: $\log w_i, \log p_i, n, \log b, \log k$.
- D.h. die Gesamtlaufzeit ist exponentiell im Eingabeparameter $n$.
- Prüfung *einzelner potentieller Lösungen* in Schritt 1.1 ist effizient.

# Polynomielle Verifizierer und NP

## **Definition** Polynomieller Verifizierer

Sei $L \subseteq \Sigma^*$ eine Sprache. Eine DTM $V$ heißt *Verifizierer für* $L$, falls $V$ für alle Eingaben $w \in \Sigma^*$ hält und folgendes gilt:

$$w \in L \Leftrightarrow \exists c \in \Sigma^* : V \text{ akzeptiert Eingabe } (w, c).$$

Das Wort $c$ nennt man einen *Zeugen oder Zertifikat für* $w$.

$V$ heißt *polynomieller Verifizierer für* $L$, falls $V$ für alle $w \in \Sigma^*$ in Laufzeit polynomiell in $|w|$ hält und folgendes gilt:

$$w \in L \Leftrightarrow \exists c \in \Sigma^*, |c| \leq |w|^k, k \in \mathbb{N} : V \text{ akzeptiert Eingabe } (w, c).$$

$L$ ist *polynomiell verifizierbar* $\Leftrightarrow \exists$ polynomieller Verifizierer für $L$.

## **Definition** Klasse $\mathcal{NP}$

$$\mathcal{NP} := \{L \mid L \text{ ist polynomiell verifizierbar.}\}$$

# Polynomieller Verifizierer für RUCKSACK

**Satz**

RUCKSACK $\in \mathcal{NP}$.

**Beweis:**

**Algorithmus** Polynomieller Verifizierer für RUCKSACK

Eingabe: $(W, P, b, k, c)$ mit Zeuge $c = I \subseteq [n]$

1. Falls $\sum_{i \in I} w_i \leq b$ und $\sum_{i \in I} p_i \geq k$, akzeptiere.
2. Lehne ab.

**Laufzeit:**

- Eingabegrößen: $\log w_i, \log p_i, \log b, \log k, n$
- Laufzeit: $\mathcal{O}(n \cdot \log(\max_i\{w_i, p_i, b, k\}))$ auf RAM.
- D.h. die Laufzeit ist polynomiell in den Eingabegrößen.

# Optimaler Wert einer Lösung mittels Entscheidung

RUCKSACK$_{wert}$

- Gegeben: $W = \{w_1, \ldots, w_n\}$, $P = \{p_1, \ldots, p_n\}$ und $B$.
- Gesucht: $\max_{I \subseteq [n]} \{\sum_{i \in I} p_i \mid \sum_{i \in I} w_i \leq b\}$

Sei $M$ eine DTM, die RUCKSACK in Laufzeit $T(M)$ entscheide.

**Algorithmus** OPTIMUM

Eingabe: $W, P, B$

1. $\ell \leftarrow 0, r \leftarrow \sum_{i=1}^{n} p_i$
2. WHILE $(\ell \neq r)$
   1. Falls $M$ bei Eingabe $(W, P, b, \lceil \frac{\ell+r}{2} \rceil)$ akzeptiert, $\ell \leftarrow \lceil \frac{\ell+r}{2} \rceil$.
   2. Sonst $r \leftarrow \lceil \frac{\ell+r}{2} \rceil - 1$.

Ausgabe: $\ell$

- Korrektheit: Binäre Suche nach Optimum auf Intervall $[0, \sum_{i=1}^{n} p_i]$.
- Laufzeit: $\mathcal{O}(\log(\sum_{i=1}^{n} p_i)) \cdot T(M)$.
- Insbesondere: Laufzeit ist polynomiell, falls $T(M)$ polynomiell ist.

# Optimale Lösung mittels optimalem Wert

**Ziel:** Bestimme Lösung $I \subseteq [n]$ mit optimalem Wert.

## **Algorithmus** Optimale Lösung

Eingabe: $W, P, b$

1. $opt \leftarrow \textsc{Optimum}(W, P, b)$, $I \leftarrow \emptyset$
2. For $i \leftarrow 1$ to $n$
   1. Falls $(\textsc{Optimum}(W \setminus \{w_i\}, P \setminus \{p_i\}, b) = opt$, setze $W \leftarrow W \setminus \{w_i\}, P \leftarrow \{p_i\}$.
   2. Sonst $I \leftarrow I \cup \{i\}$.

Ausgabe: $I$

**Korrektheit:**

- Invariante vor $i$-tem Durchlauf: $\exists J \subseteq \{i, \ldots, n\}$: $I \cup J$ ist optimal.
- $i$ wird nur dann in $I$ aufgenommen, falls $I$ zu optimaler Teilmenge erweitert werden kann.
- **Laufzeit:** $\mathcal{O}(n \cdot T(\textsc{Optimum})) = \mathcal{O}(n \cdot \log(\sum_{i=1}^{n} p_i) \cdot T(M))$.
- D.h. Laufzeit ist polynomiell, falls $T(M)$ polynomiell ist.

# Sprache Zusammengesetzt

$\text{ZUSAMMENGESETZT} := \{N \in \mathbb{N} \mid N = pq \text{ mit } p, q \geq 2\}$

**Satz**

$\text{ZUSAMMENGESETZT} \in \mathcal{NP}$.

**Beweis:**

**Algorithmus** Polynomieller Verifizierer für ZUSAMMENGESETZT

Eingabe: $(N, c)$ mit $c = (p, q) \in \{2, \ldots, N-1\}^2$

1. Berechne $p \cdot q$. Falls $p \cdot q = N$, akzeptiere. Sonst lehne ab.

**Laufzeit:**

- Eingabelänge: $|N| = \Theta(\log N)$
- Laufzeit: $\mathcal{O}(\log^2 N)$, d.h. polynomiell in der Eingabelänge.

# $\mathcal{P}$ versus $\mathcal{NP}$

**Satz**

$\mathcal{P} \subseteq \mathcal{NP}$.

$L \in \mathcal{P}$ $\Rightarrow$ $\exists$ DTM $M$, die $L$ in polynomieller Laufzeit entscheidet.

$\Rightarrow$ $\exists$ DTM $M$, die stets hält und genau die Eingaben $w \in L$ in Laufzeit polynomiell in $|w|$ akzeptiert.

$\Rightarrow$ $\exists$ DTM $V$, die stets hält und genau die Eingaben $(w, c)$ mit $w \in L$, $c = \epsilon$ in Laufzeit polynomiell in $|w|$ akzeptiert. Dabei ignoriert $V$ die Eingabe $c$ und wendet $M$ auf $w$ an.

$\Rightarrow$ $L \in \mathcal{NP}$.

- **Großes offenes Problem:** Gilt $\mathcal{P} = \mathcal{NP}$ oder $\mathcal{P} \subset \mathcal{NP}$?

# Nichtdeterministische Turingmaschinen

Wir bezeichnen mit $\mathcal{P}(S)$ die Potenzmenge einer Menge $S$.

**Definition** Nichtderministische Turingmaschine

Eine *nicht-deterministische Turingmaschine (NTM)* ist ein Tupel $(Q, \Sigma, \Gamma, \delta)$, wobei

- $Q, \Sigma, \Gamma$ sind wie bei DTM definiert.
- $\delta : Q \setminus \{q_a, q_r\} \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$

- Bsp: $\delta(q, a) = \{(q_1, a_1, L), (q_2, a_2, R)\}$.
- NTM besitzt Wahlmöglichkeiten für den Zustandsübergang.
- Beschränken uns oBdA auf NTMs mit $\leq 2$ Wahlmöglichkeiten.

# Berechnungsbaum

- Seien die Konfigurationen einer NTM Knoten in einem Berechnungsbaum.
  - Die Startkonfiguration bildet den Wurzelknoten.
  - Mögliche Nachfolgekonfigurationen bilden Kinderknoten.
- Pfade heißen Berechnungspfade der NTM.
- Betrachten nur NTMs mit Berechnungspfaden endlicher Länge.
- Ein Berechnungpfad heißt akzeptierend, falls er in $q_a$ endet.

**Definition** Akzeptierte Sprache einer NTM

Sei $N$ eine NTM.

- $N$ akzeptiert Eingabe $w \Leftrightarrow \exists$ akzeptierenden Berechnungspfad im Berechnungsbaum von $N$ bei Eingabe $w$.
- Die von $N$ akzeptierte Sprache $L(N)$ ist definiert als

$$L(N) = \{w \in \Sigma^* \mid N \text{ akzeptiert die Eingabe } w.\}.$$

# Die Laufzeit einer NTM

## **Definition** Laufzeit einer NTM

Sei $N$ eine DTM mit Eingabe $w$.

- $T_N(w) :=$ **maximale** Anzahl Rechenschritte von $N$ auf $w$, d.h. $T_N(w)$ ist die Länge eines längsten Berechnungspfades.
- $T_N : \mathbb{N} \to \mathbb{N}, T_N(n) := \max\{T_N(w) \mid w \in \Sigma^{\leq n}\}$ heißt *Laufzeit* oder Zeitkomplexität von $N$.

- Wir definieren die Klasse NTIME für NTMs analog zur Klasse DTIME für DTMs.

## **Definition** NTIME

Sei $t : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion.

$$\text{NTIME}(t(n)) := \{L \mid L \text{ wird von NTM in Laufzeit } \mathcal{O}(t(n)) \text{ entschieden.}\}$$