19) Das RSA-Verfahren

Das RSA-Verfahren

Definition

Die Einheitengruppe des Ringes $\mathbb{Z}_m = \{\overline{0}, \overline{1}, \ldots, \overline{m-1}\}$ *ist definiert als die Menge aller Einheiten von* $\mathbb{Z}_m$.

$$\mathbb{Z}_m^* = \{\bar{a} \in \mathbb{Z}_m \mid \exists \bar{a}^{-1}\}.$$

Satz

$(\mathbb{Z}_m^*, \cdot)$ *ist eine Gruppe.*

Satz

Für $\bar{a} \in \mathbb{Z}_m$ *gilt:* $\bar{a} \in \mathbb{Z}_m^* \iff \mathrm{ggT}(a, m) = 1$.

Beweis: Eine Folgerung aus Abschnitt 17 lautete:
$ax \equiv b\ (m)$ ist genau dann lösbar, wenn $d = \mathrm{ggT}(a, m)$ ein Teiler von b ist.
Man setze $b = 1$, dann erhalten wir Lösbarkeit genau für $\mathrm{ggT}(a, m) = 1$. □

Das RSA-Verfahren

Definition

Die Eulersche φ-Funktion ist definiert durch $\varphi(m) = |\mathbb{Z}_m^*|$.

Satz

Sei $m = \prod_{i=1}^{r} p_i^{e_i}$ die Primfaktorzerlegung von m. Dann gilt

$$\varphi(m) = m \prod_{i=1}^{m} \left(1 - \frac{1}{p_i}\right).$$

Beweis: Für $i = 1, \ldots, r$ setze man $A_i = \{x : 1 \leq x \leq m,\ p_i \mid x\}$ und wende dann das Inklusions-Exklusions-Prinzip an. □

Satz (Kleiner Fermat, siehe Abschnitt 5)

Sei p eine Primzahl und $\mathrm{ggT}(a, p) = 1$. Dann gilt $a^{p-1} \equiv 1\ (p)$.

Allgemeiner gilt:

Satz (Euler-Fermat)

Sei $\mathrm{ggT}(a, m) = 1$. Dann gilt $a^{\varphi(m)} \equiv 1\ (m)$.

Das RSA-Verfahren

Satz

Seien p, q zwei verschiedene ungerade Primzahlen, $m = pq$ und $v = \mathrm{kgV}(p-1, q-1)$. Dann gilt für alle $a, k \in \mathbb{Z}$: $a^{kv+1} \equiv a\ (m)$.

Beweis: Es gilt $pq \mid (a^{kv+1} - a)$ genau dann, wenn sowohl $p \mid (a^{kv+1} - a)$ als auch $q \mid (a^{kv+1} - a)$.
Wir zeigen $p \mid (a^{kv+1} - a)$, der Beweis für q ist analog.

1. Fall: $p \mid a$: trivial.

2. Fall: $p \not| a$. Dann folgt $a^{p-1} \equiv 1\ (p)$. Folglich bekommen wir $a^{kv} \equiv 1\ (p)$ und daher $a^{kv+1} \equiv a\ (p)$. □

RSA-Verfahren

- $p, q \in \mathbb{P} \setminus \{2\}$, $m = pq$, $v = (p-1)(q-1)$, $\mathrm{ggT}(e, v) = 1$.
- Es gibt daher ein d mit $d \cdot e \equiv 1\ (v)$.
- Nachricht in Blöcke zerlegen: $a_1, a_2, \ldots,$; $0 \leq a_i \leq m-1$.
- Verschlüsselung: $E(a_j) = b_j := a_j^e \mod m$;
- Entschlüsselung: $D(b_j) = b_j^d \mod m$.

(m, e) öffentlicher Schlüssel, (m, d) privater Schlüssel.

Das RSA-Verfahren

Digitale Unterschrift:
Nachricht x mit $s = D(x)$ signieren und (x, s) senden.
Empfänger prüft, ob $x = E(s)$ erfüllt ist.
Gefahren:

- $E(x)$ kann viele Fixpunkte in $\mathbb{Z}_m$ haben.
  - $\Longrightarrow$ verschlüsselte Nachricht ist dann der Klartext.
  - $\rightsquigarrow$ gute Wahl von m und e. Anzahl der Fixpunkte ist $(1 + \mathrm{ggT}(e-1, p-1))(1 + \mathrm{ggT}(e-1, q-1))$.
- Maximale Ordnung $\lambda(v)$ eines Elements in $\mathbb{Z}_v^*$ kann niedrig sein.
  - $\longrightarrow$ Attacke durch Iteration: $E^i(x) = x^{e^i} \mod m$.
  - $\lambda(\lambda(m))$ muss möglichst groß sein.
    - optimal: $\lambda(\lambda(m)) = \frac{(p-3)(q-3)}{8}$,
    - genau dann, wenn dw $\frac{p-1}{2}, \frac{q-1}{2}, \frac{p-3}{4}, \frac{q-3}{4}$ Primzahlen sind.
    - Nachteil: Solche Primzahlen sind selten.
      Unbekannt, ob es unendlich viele sichere Primzahlen gibt.
      Vermutung: mindestens eine in $[n, 2n]$ für $n \geq 24$.

Bemerkung: $\lambda(n)$ heißte n-te Carmichaelzahl.

20) Dynamische Programmierung

Dynamische Programmierung

- Divide & Conquer:
  - Problem zerlegen in unabhängige Teilprobleme;
  - Rekursives Lösen der Teilprobleme;
  - Lösung aus den Lösungen der Teilprobleme konstruieren.
- Dynamische Programmierung:
  - Problem zerlegen in (überlappende) Teilprobleme;
  - Teilprobleme lösen, Ergebnisse speichern und wiederverwenden;
  - Lösung aus den Lösungen der Teilprobleme konstruieren.

Typische Anwendung: Optimierungsprobleme (viele Lösungen, jede hat einen *Wert*, suche optimalen Wert).
Entwurf von Algorithmen für dynamische Programmierung:

- Charakterisieren der Struktur einer optimalen Lösung (Wert);
- Definieren des Werts einer optimalen Lösung (rekursiv);
- Berechnen des Wertes einer optimalen Lösung (bottom-up);
- Konstruieren einer optimalen Lösung aus vorberechneten Informationen.

Beispiel: Stabzerlegungsproblem

Gegeben:

- Eine Stange der Länge n.
- Eine Preisliste $p = [p_1, p_2, \ldots, p_n]$.

Gesucht: Der maximale Erlös r_n.

Länge	1	2	3	4	5	6	7	8	9	10
Preis	1	5	8	9	10	17	17	20	24	30

Es gibt 2^{n-1} Möglichkeiten den Stab zu zerschneiden.

r_n von der Form

$$r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_s}.$$

Es gilt

$$r_n = \max\{p_n, \max_{1 \leq k < n} (r_k + r_{n-k})\}.$$

Optimale Teilstruktur Eigenschaft

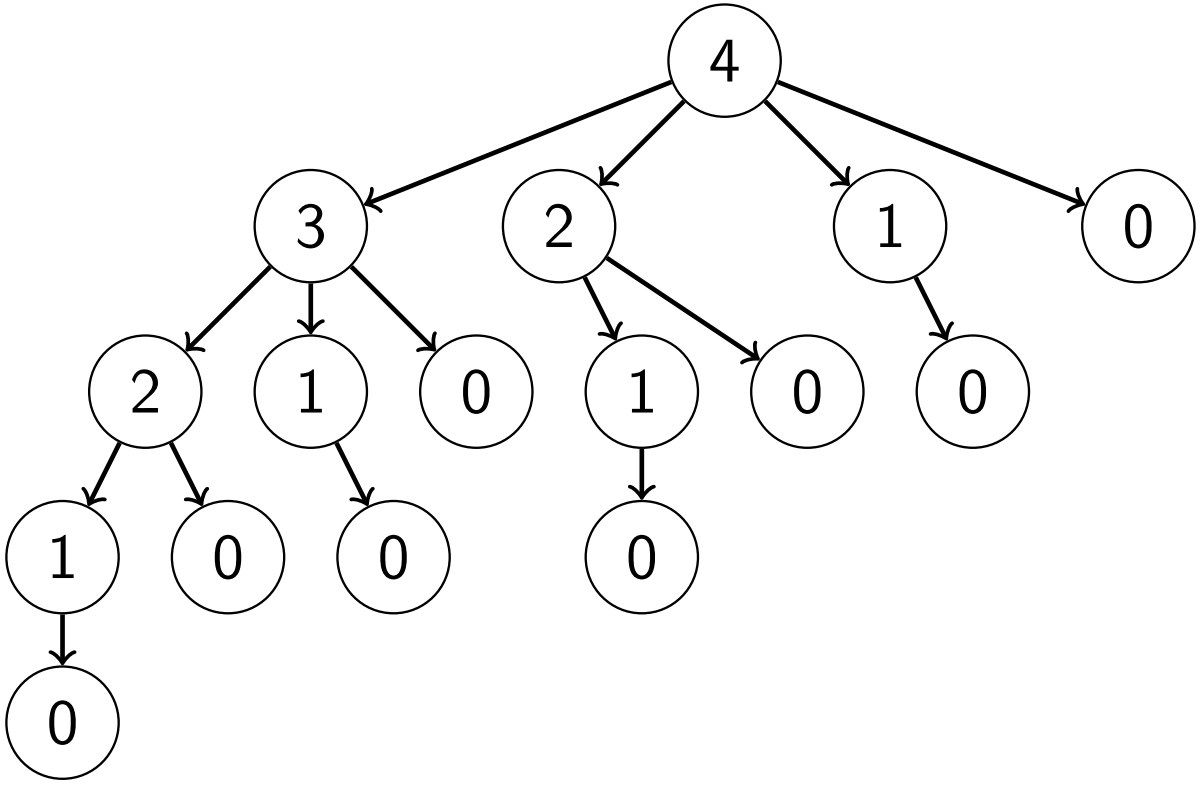
$$r_n = \max_{1 \le k \le n} \{p_k + r_{n-k}\}, \quad r_0 = 0.$$

Algorithm CUT-ROD(p, n)

```
1: if n = 0 then
2:     return 0
3: end if
4: q := −∞
5: for i = 1 to n do
6:     q := max{q, p[i] + CUT-ROD(p, n − i)}
7: end for
8: return q
```

Laufzeit: $T(n) = 1 + \sum_{i=0}^{n-1} T(i)$ und daher
$T(n) - T(n-1) = T(n-1)$. Daraus folgt
$T(n) = 2T(n-1), \quad T(1) = 1$ und schließlich $T(n) = 2^n$.

Rekursionbaum von CUT-ROD für $n = 4$:



Top-Down Ansatz mit Memoisation

Algorithm MCR(p, n)

```
1: Sei r[0..n] ein neues Feld
2: for i = 0 to n do
3:     r[i] := −∞
4: end for
5: return MCR-AUX(p, n, r)
```

Algorithm MCR-AUX(p, n, r)

```
1: if r[n] ≥ 0 then
2:     return r[n]
3: end if
4: if n = 0 then
5:     q := 0
6: else
7:     q := −∞
8:     for i = 1 to n do
9:         q := max{q, p[i]+MCR-AUX(p, n−i, r)}
10:    end for
11:    r[n] := q
12: end if
13: return q
```

Laufzeit:

$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right).$$

Bottom-up Ansatz

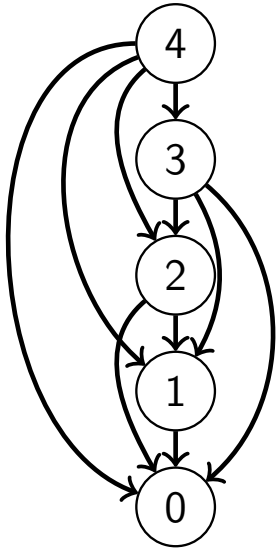
Algorithm BOTTOM-UP-CUT-ROD(p, n)

```
1:  Sei r[0..n] ein neues Feld
2:  r[0] := 0
3:  for j = 1 to n do
4:      q := −∞
5:      for i = 1 to j do
6:          q = max{q, p[i] + r[j − i]}
7:      end for
8:      r[j] := q
9:  end for
10: return r[n]
```

Laufzeit: $\Theta\left(n^2\right)$ (kleinere Konstante als bei MCR)

Teilproblem-Graphen

- Gerichteter Graph;
- Die Knoten entsprechen verschiedenen Teilproblemen;
- Eine Kante vom Knoten x zum Knoten y bedeutet, dass, um eine optimale Lösung für Teilproblem x zu finden, wir zuerst eine optimale Lösung für das Teilproblem y bestimmen müssen.



Bottum-up entspricht umgekehrter topologischer Sortierung

Laufzeit: Jedes Teilproblem einmal, typischerweise $T_v = \mathcal{O}(d^+(v))$, insgesamt $\mathcal{O}((|V| + |E|))$.

Top-Down entspricht einer Tiefensuche.

Algorithm EXTENDED-BOTTOM-UP-CUT-ROD(p, n)

```
 1: Seien r[0..n] und s[0..n] neue Felder
 2: r[0] := 0
 3: s[0] := 0
 4: for j = 1 to n do
 5:     q := −∞
 6:     for i = 1 to j do
 7:         if q < p[i] + r[j − i] then
 8:             q := p[i] + r[j − i]
 9:             s[j] := i
10:         end if
11:     end for
12:     r[j] := q
13: end for
14: return r und s
```

Beispiel: Stabzerlegungsproblem

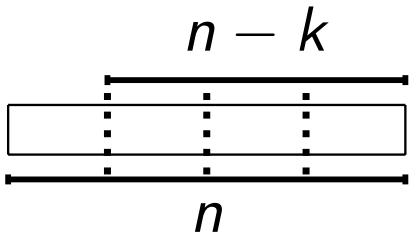
Länge	0	1	2	3	4	5	6	7	8	9	10
Preis	0	1	5	8	9	10	17	17	20	24	30
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

Eigenschaften der dynamischen Programmierung

- optimale Teilstruktur;
- überlappende Teilprobleme.

Optimale Teilstruktur:

Optimale Lösung enthält optimale Lösungen aller Teilprobleme.



Allgemeines Muster:

- Zeigen, dass eine Lösung darin besteht, eine Entscheidung zu treffen, wie das Problem aufgeteilt wird.
- Annahme, dass eine Entscheidung existiert, die zu einer optimalen Lösung führt.
- Bestimmen, welche Teilprobleme sich dadurch ergeben.
- Zeigen: Lösungen für Teilprobleme selbst optimal. (Cut & paste-Meth.)

Dynamische Programmierung

Achtung: Es gibt Probleme ohne optimale Teilstruktur

Sei $G = (V, E)$, $u, v \in V$. Wir betrachten zwei Probleme:

- Kürzester Weg (ungewichtet):
  - Suche Weg $u \rightsquigarrow v$ mit minimaler Kantenanzahl.
  - Optimale Teilstruktur.
  - Denn wenn $u \rightsquigarrow w \rightsquigarrow v$ kürzester Weg, dann sind auch die Teilwege $u \rightsquigarrow w$ und $w \rightsquigarrow v$ kürzeste Wege.
- Längster Weg (ungewichtet):
  - Suche Weg $u \rightsquigarrow v$ mit maximaler Kantenanzahl.
  - Keine optimale Teilstruktur.
  - Beispiel: