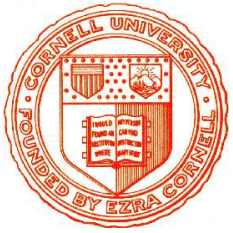
Theoretische Informatik II

Einheit 5.5



Grenzen überwinden

1. Pseudopolynomielle Algorithmen
2. Approximierende Algorithmen
3. Probabilistische Algorithmen

Wie kann man "unlösbare" Probleme angehen?

- **Viele Probleme sind nachweislich schlecht lösbar**
  - Terminierung, Korrektheit, Äquivalenz von Programmen (unentscheidbar)
  - Gültigkeit prädikatenlogischer Formeln (unentscheidbar)
  - Strategische Spiele / Marktanalysen ($PSPACE$-vollständig)
  - SAT Solver / Model-checking ($\mathcal{NP}$- oder $PSPACE$-vollständig)
  - Scheduling, Navigation, Verteilungsprobleme ($\mathcal{NP}$-vollständig)
  - Erzeugung kryptographischer Schlüssel ($\mathcal{NP}$ bzw. $\mathcal{O}(n^6)$)
- **Lösungen werden dennoch gebraucht**
  - Die Probleme tauchen in der Praxis auf
  - Aufgeben ist keine akzeptable Antwort
- **Versuche die Unlösbarkeiten zu umgehen**
  - Suche Alternativen zur perfekten Lösung, die es nicht geben kann
  - Entwickle Verfahren zur Konstruktion "suboptimaler Lösungen"
  - Untersuche die Qualität dieser Lösungen relativ zum Optimum

  Suche nach neuen Lösungsmöglichkeiten liefert tieferes Verständnis

Alternative Methoden für "unlösbare" Probleme

- **Heuristische Lösung unentscheidbarer Probleme**
  - Algorithmus "versucht" einen Standardlösungsweg und gibt auf, wenn dieser nicht zum Erfolg führt (Künstliche Intelligenz)
  - Verzicht auf Vollständigkeit zugunsten einer "Entscheidung"
  - Anwendung: Theorembeweisen, Programmverifikation und -synthese
- **Selbstorganisation statt vorformulierter Lösungen**
  - Lernverfahren, Neuronale Netze, genetische Algorithmen, ...
- **Approximationsverfahren**
  - Effiziente "Lösung" schwerer Optimierungsprobleme
  - Algorithmus bestimmt Näherungslösung anstelle des Optimums
  - Verzicht auf Optimalität zugunsten einer schnellen Antwort
- **Probabilistische Algorithmen**
  - Effiziente "Lösung" schwerer Entscheidungsprobleme
  - Algorithmus verwendet Zufallsvariablen bei Bestimung der Lösung
  - Antwort kann mit geringer Fehlerwahrscheinlichkeit auch falsch sein
  - Verzicht auf perfekte Korrektheit zugunsten einer schnellen Antwort

Nicht jedes $\mathcal{NP}$-Problem ist wirklich schwer

Gibt es "leichte" $\mathcal{NP}$-vollständige Probleme?

- **Was unterscheidet $CLIQUE$ von KP?**
  - Beide Probleme sind $\mathcal{NP}$-vollständig, aber
    - $3SAT \leq_p CLIQUE$ codiert Formel durch gleich großen Graph
    - $3SAT \leq_p KP$ benutzt exponentiell große Zahlen als Codierung
  - Ist KP nur wegen der großen Zahlen $\mathcal{NP}$-vollständig?
- **Es gibt "bessere" Lösungen für KP**

  $KP = \{ (g_1..g_n, a_1..a_n, G, A) \mid \exists J \subseteq \{1..n\}. \Sigma_{i \in J} g_i \leq G \wedge \Sigma_{i \in J} a_i \geq A \}$
  - Man muß nicht alle Kombinationen von $\{1..n\}$ einzeln auswerten
  - Man kann iterativ den optimalen Nutzen bestimmen, indem man die Anzahl der Gegenstände und das Gewicht erhöht
  - Sehr effizient, wenn das maximale Gewicht nicht zu groß wird

Iterative Lösung für KP

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

- **Betrachte Subprobleme $\boldsymbol{KP(k,g)}$** (g und k fest)
  - Verwende Gegenstände $1, .., k$ und Maximalgewicht $g{\leq}G$
  - Definiere optimalen Nutzen $N(k,g)$
    - $N(k,0) = 0$ für alle k
    - $N(0,g) = 0$ für alle g
    - $N(k,g) = max\{N(k{-}1, g{-}g_k) + a_k,\ N(k{-}1,g)\}$
- **Löse Rucksackproblem $\boldsymbol{KP}$ iterativ**
  - Es gilt $(g_1..g_n,\ a_1..a_n,\ G,\ A) \in KP \Leftrightarrow N(n,G){\geq}A$
  - Gleichungen beschreiben rekursiven Algorithmus für $N(n,G)$
  - Tabellarischer Algorithmus bestimmt alle $N(k,g)$ mit $k{\leq}n$, $g{\leq}G$
  - Laufzeit ist $\mathcal{O}(n * G)$

$(g_1..g_n, a_1..a_n,\ G, A) \in KP$ ist in $\mathcal{O}(n{*}G)$ Schritten lösbar

Pseudopolynomielle Algorithmen

Liegt das Rucksackproblem KP etwa in $\mathcal{P}$?

- **Lösung für KP ist nicht wirklich polynomiell**
  - $n * G$ kann exponentiell wachsen relativ zur Größe der Eingabe
  - Größe von $(g_1..g_n,\ a_1..a_n,\ G,\ A)$ ist $\mathcal{O}(n * (logG + logA))$
- **KP ist ein Zahlproblem**
  - $L \subseteq \Sigma^*$ ist **Zahlproblem**, wenn $MAX(w)$, die größte in einer Eingabe w codierte Zahl, nicht durch ein Polynom beschränkt werden kann
  - Weitere Zahlprobleme: $PARTITION$, BPP, TSP, MSP, ...
  - Keine Zahlprobleme: $CLIQUE$, VC, IS, SGI, LCS, DHC, HC, GC,...
- **KP hat pseudopolynomielle Lösung**
  - Algorithmen für ein Zahlproblem $L \subseteq \Sigma^*$ sind **pseudopolynomiell**, wenn ihre Rechenzeit durch ein Polynom in $|w|$ und $MAX(w)$ beschränkt ist

Starke $\mathcal{NP}$-vollständigkeit

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**
  - Ist $L \subseteq \Sigma^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p
    $\boldsymbol{L_p} \equiv \{w \in L \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$
  - Die Restriktion von KP auf polynomiell große Gewichte liegt in $\mathcal{P}$
  - Hat jedes Zahlproblem eine pseudopolynomielle Lösung?
- **TSP ohne pseudopolynomielle Lösung** (falls $\mathcal{P} \neq \mathcal{NP}$)
  - Der Reduktionsbeweis $HC \leq_p TSP$ zeigt $HC \leq_p TSP_n$
  - Eine Restriktion von TSP auf kleine Zahlen bleibt $\mathcal{NP}$-vollständig
- **TSP ist stark $\mathcal{NP}$-vollständig**
  - $L \subseteq \Sigma^*$ **stark $\mathcal{NP}$-vollständig** $\equiv$ L_p $\mathcal{NP}$-vollständig für ein Polynom p
  - L stark $\mathcal{NP}$-vollständig $\Rightarrow$ L hat keine pseudopolynomielle Lösung

Einschränkung auf kleine Zahlen ist keine Lösung für das $\mathcal{P}-\mathcal{NP}$ Problem

Approximationsalgorithmen

- **Viele Probleme haben Optimierungsvariante**
  - $CLIQUE_{opt}$: bestimme die größte Clique im Graphen
  - TSP_{opt}: bestimme die kostengünstigste Rundreise
  - BPP_{opt}: bestimme die kleinste Anzahl der nötigen Behälter
  - KP_{opt}: bestimme das geringstmögliche Gewicht für einen festen Nutzen

  Alle Probleme sind $\mathcal{NP}$-hart
- **Approximation umgeht Komplexitätsproblematik**
  - Polynomielle Algorithmen können Näherungslösungen bestimmen
  - Die Näherung kann niemals optimal sein (wenn $\mathcal{P} \neq \mathcal{NP}$)
  - Ziel ist, so nahe wie möglich an das Optimum heranzukommen
- **Wie gut können Näherungslösungen sein?**
  - Vergleiche Approximation mit bestmöglicher Lösung
  - Wie gut ist das Ergebnis eines konkreten Approximationsalgorithmus?
  - Was ist das günstigste Ergebnis, das überhaupt erreichbar ist

Qualität von Approximationslösungen

- **Beschreibung von Optimierungsproblemen**
  - Betrachte zugehöriges Entscheidungsproblem L als Menge aller akzeptablen Lösungen (x, y) für eine Eingabe x
    - z.B. $CLIQUE := \{(G,k) \mid G \text{ hat Clique der Größe } k\}$
      Bei Eingabe G sucht Optimierung größtes k mit $(G,k) \in CLIQUE$
  - Bestimme den Wert $\boldsymbol{W(x,y)}$ einer Lösung $(x,y) \in L$
  - $\boldsymbol{OPT_L(x)}$: Wert einer optimalen Lösung für Eingabe x
- **Güte von Approximationsalgorithmen**
  - Algorithmus A berechne für jede Eingabe x ein $y{=}A(x)$ mit $(x,y) \in L$
  - $\boldsymbol{R_A(x)}$: Güte des Algorithmus A bei Eingabe x ($R_A(x) > 1$ gilt immer)
    - $R_A(x) \equiv \max\{OPT_L(x)/W(x,A(x)),\ W(x,A(x))/OPT_L(x)\}$
- **Asymptotische Güte von Approximationen**
  - $\boldsymbol{R_A^\infty} = \inf\{r{\geq}1 \mid \forall^\infty x.R_A(x){\leq}r\}$: asymptotische worst-case Güte

Finde bestmögliche worst-case Güte von Problemen

Positive Ergebnisse

- **Knapsack: beliebig kleiner multiplikativer Fehler**
  - Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(x) \leq 1+\epsilon$ für alle x (ohne Beweis)
- **Binpacking: Asymptotische Güte 11/9 erreichbar**
  - First-Fit Decreasing: Sortiere Objekte in absteigender Reihenfolge und packe sie jeweils in erste freie Kiste, in der genügend Platz ist
  - Es gilt $W(x,\ FFD(x)) = 11/9 * OPT_{BPP}(x) + 4$ für alle x (ohne Beweis) also $R_A^\infty = 11/9$
- **TSP: $R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung**
  - Direkte Verbindung sind kürzer als Umwege: $\forall i, j, k.\ c_{i,j} \leq c_{i,k} + c_{k,j}$

Travelling Salesman mit Dreiecksungleichung

$$\boldsymbol{TSP_\Delta} = \{\, c_{12}, ..., c_{n-1,n},\ B \mid\ \forall i,j,k.\ c_{i,j} \leq c_{i,k} + c_{k,j}\ \wedge\ \exists \pi : \{1..n\} \to \{1..n\}.\ \pi \text{ bijektiv}\ \wedge\ \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B \,\}$$

- **Approximationsalgorithmus**
  - Zu $w = c_{12}, ..., c_{n-1,n}, B$ konstruiere vollständigen Graphen $G=(V, E)$ mit $V = \{v_1, .., v_n\}$ und Gewichten $c_{i,j}$ für $\{v_i, v_j\} \in E$
  - Konstruiere minimal spannenden Baum $T = (V, E_T)$
    Kruskal Algorithmus (Einheit 5.1) hat Laufzeit $\mathcal{O}(|V| + |E| \log |E|)$
  - Durchlaufe T so, daß jede Kante genau zweimal benutzt wird
  - Verkürze den entstandenen Rundweg so, daß von einem Knoten zum nächsten noch nicht angesteuerten Knoten gesprungen wird
- **Laufzeit des Algorithmus ist $O(n^2)$**
- **Güte des Algorithmus ist $R_A^\infty \leq 2$**
  - Mit lokalen Optimierungen kann man auf $R_A^\infty \leq 3/2$ kommen (aufwendige Analyse)

Negative Ergebnisse (falls $\mathcal{P} \neq \mathcal{NP}$)

- **Knapsack: konstanter additiver Fehler unmöglich**
  - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x
- **Binpacking: absolute Güte $R_A < 3/2$ unmöglich**
  - Es gibt keinen polynomiellen Algorithmus A_{BPP} mit $R_{A_{BPP}}(x) < 3/2$ für alle x
- **CLIQUE: Keine endliche absolute Güte möglich**
  - Es gibt ein $\epsilon > 0$, som daß es keinen polynomiellen Algorithmus A_{CL} geben kann mit $R_{A_{CL}}(x) < |x|^{1/2-\epsilon}$ für alle x (ohne Beweis)
- **TSP: Keine endliche worst-case Güte möglich**
  - Es gibt keinen polynomiellen Algorithmus A_{TSP} mit $R^{\infty}_{A_{TSP}} = r$ für ein $r \in \mathbb{N}$

Nachweise verwenden verschiedene Beweistechniken

Kein konstanter additiver Fehler für Knapsack

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J \subseteq \{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x

Gäbe es A_{KP}, dann könnten wir KP wie folgt polynomiell entscheiden

– Transformiere $x = (g_1..g_n,\ a_1..a_n,\ G,\ A)$ in

$$x' = (g_1..g_n,\ a_1{*}(k{+}1)..a_n{*}(k{+}1),\ G,\ A{*}(k{+}1))$$

– Wegen $|OPT_{KP}(x') - W(x, A_{KP}(x'))| \leq k$ folgt

$$|OPT_{KP}(x) - \lfloor W(x', A_{KP}(x')/(k{+}1) \rfloor\ | \leq \lfloor k/(k{+}1) \rfloor = 0$$

– Also gilt $x \in KP \Leftrightarrow OPT_{KP}(x) = A$

$$\Leftrightarrow \lfloor W(x', A_{KP}(x'))/(k{+}1) \rfloor \geq A$$

Beweistechnik: Multiplikation des Problems, nachträgliche Division des Fehlers

Keine absolute Güte $R_A < 3/2$ für Binpacking

$$\boldsymbol{BPP} = \{\, a_1, \ldots a_n,\ b, k \mid \exists f{:}\{1..n\}\to\{1..k\}.\ \forall j{\leq}k.\ \textstyle\sum_{i \in \{i \mid f(i)=j\}} a_i {\leq} b \,\}$$

Es gibt keinen polynomiellen Algorithmus A_{BPP} mit $R_{A_{BPP}}(x) < 3/2$ für alle x

- Die Reduktion $PARTITION {\leq_p} BPP$ benutzt 2 Behälter der Größe $S := \sum_{i=1}^{n} a_i/2$, auf welche die Zahlen verteilt werden
- Für die Transformationsfunktion f gilt

  $$x \in PARTITION \Leftrightarrow OPT_{BPP}(f(x)) = 2$$

- Jeder Approximationsalgorithmus A mit $R_A(x) < 3/2$ liefert

  $$W(f(x),\ A(f(x))) = \begin{cases} 2 & \text{falls } x \in PARTITION \\ 3 & \text{sonst} \end{cases}$$

- Wegen $PARTITION \in \mathcal{NPC}$ kann A nicht polynomiell sein

Beweistechnik: Einbettung eines $\mathcal{NP}$-vollständigen Entscheidungsproblems

Keine endliche worst-case Güte für TSP

$$\boldsymbol{TSP} = \{\, c_{12}, .., c_{n-1,n},\ B \mid \exists \text{ Bijektion } \pi.\ \textstyle\sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B \,\}$$

Es gibt keinen polynomiellen Algorithmus A_{TSP} mit $R^{\infty}_{A_{TSP}} = r$ für ein $r \in \mathbb{N}$

Wenn es A geben würde, dann entscheiden wir HC polynomiell wie folgt

- Transformiere einen Graphen $G = (V, E)$ in $x = c_{12}, ..., c_{n-1,n},\ |V|$ mit $c_{ij} = 1$ falls $\{i, j\} \in E$ und $c_{ij} = r|V| + 1$ sonst
- Dann $G \in HC \Rightarrow OPT_{TSP}(x) = |V|$ und $G \notin HC \Rightarrow OPT_{TSP}(x) > (r+1) * |V|$
- Für große Graphen gilt dann: $W(x, A(x)) \leq r * OPT_{TSP}(x)$ also $G \in HC \Leftrightarrow W(x, A(x)) \leq r * |V|$

  Für kleine Graphen verwende den exponentiellen Entscheidungsalgorithmus

Beweistechnik: Reduktion auf $\mathcal{NP}$-vollständiges Problem mit Multiplikation des Kostenunterschieds zwischen positiver und negativer Antwort

Probabilistische Algorithmen

"Approximation" einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlgenerator**
  - Falsche Entscheidungen sind möglich aber unwahrscheinlich
  - Approximation $\hat{=}$ Verringerung der Fehlerwahrscheinlichkeit
  - Fehler unter 2^{-100} liegt unter Wahrscheinlichkeit von Hardwarefehlern
- **Viele sinnvolle Anwendungen**
  - Quicksort: schnellstes Sortierverfahren in der Praxis
  - Linearer Primzahltest (relativ zur Anzahl der Bits)
- **Wie weist man gute Eigenschaften nach?**
  - Einfaches Modell für probabilistische Algorithmen formulieren
  - Eigenschaften abstrakter probabilistischer Sprachklassen analysieren

Quicksort als zufallsabhängiger Algorithmus

- **Divide & Conquer Ansatz für Sortierung**
  - Wähle Pivotelement a_i aus Liste $a_1, ..., a_n$
  - Zerlege Liste in Elemente, die größer oder kleiner als a_i sind
  - Sortiere Teillisten und hänge Ergebnisse aneinander
- **Laufzeit abhängig vom Pivotelement**
  - $\mathcal{O}(n * \log_2 n)$, wenn Teillisten in etwa gleich groß sind
  - Pivotelement muß nahe am Mittelwert sein
  - Deterministische Bestimmung des Mittelwertes zu zeitaufwendig
  - Wahl eines festen Pivotelements erhöht Laufzeit von Quicksort für bestimmte Eingabelisten auf $\mathcal{O}(n^2)$
- **Gute Pivotelemente sind in der Mehrzahl**
  - Zufällige Wahl führt für jede Eingabe zum Erwartungswert $\mathcal{O}(n * \log_2 n)$
  - Im Durchschnitt schneller als deterministische $\mathcal{O}(n * \log_2 n)$-Verfahren

Probabilistische Berechnungsmodelle

- **Probabilistische Turingmaschine**
  - Struktur: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
  - Zustandsüberführungsfunktion: $\delta : Q \times \Gamma \to (Q \times \Gamma \times \{L, R\})^2$
    Jede Alternative wird mit Wahrscheinlichkeit $1/2$ gewählt
  - Rechenzeit: maximale Rechenzeit aller möglichen Rechenwege
  - ***PTM***: polynomiell zeitbeschränkte probabilistische Turingmaschine
- ***Prob*$(M \downarrow w)$: Wahrscheinlichkeit der Akzeptanz**
  - Wahrscheinlichkeit aller akzeptierenden Konfigurationsfolgen
    relativ zu allen möglichen Konfigurationsfolgen bei Eingabe w
- **Probabilistische Algorithmen**
  - Abstrakteres Modell: Programme mit zufälligen Entscheidungen
  - Komplexitätsbestimmung durch asymptotische Analyse wie bisher

Was kann man mit polynomiell zeitbeschränkten probabilistischen Algorithmen erreichen?

Klassen probabilistischer Sprachen

- ***PP*: Probabilistic Polynomial** Monte-Carlo-Algorithmen
  - Wahrscheinlichkeit für korrekte Antwort ist mindestens $1/2$
    $w \in L \Rightarrow Prob(M \downarrow w) \geq 1/2$ und $w \notin L \Rightarrow Prob(M \not\downarrow w) > 1/2$
  - Laufzeit polynomiell
- ***BPP*: Bounded error Probabilistic Polynomial**
  - Wahrscheinlichkeit für korrekte Antwort ist mindestens $1/2+\epsilon$
  - Laufzeit polynomiell
- ***RP*: Random Polynomial**
  - Wahrscheinlichkeit für Akzeptanz von $w \in L$ ist mindestens $1/2$
  - Eingaben $w \notin L$ werden mit Sicherheit nicht akzeptiert
  - Laufzeit polynomiell
- ***ZPP*: Zero error PP** Las-Vegas-Algorithmen
  - M gibt immer eine korrekte Antwort
  - Erwartungswert der Laufzeit ist polynomiell

  Alternative Definitionen verwenden Erkenntnis $ZPP = RP \cap co{-}RP$ (Folie 22)

Beispiel probabilistischer Algorithmen

Prüfe ob ein Graph $G=(V,E)$ eine 3-Clique enthält

- **Teste zufällige Dreieckskandidaten**
  - Zufällige Auswahl von $v_1 \in V$ und $\{v_2, v_3\} \in E$ mit $v_1 \notin \{v_2, v_3\}$
  - Teste deterministisch , ob $\{v_1, v_3\} \in E$ und $\{v_1, v_2\} \in E$ gilt
  - Akzeptiere, wenn Dreieck in k Iterationen gefunden, sonst verwerfe
- **Wahrscheinlichkeit korrekter Akzeptanz $> 1/2$**
  - Wahrscheinlichkeit für Auswahl einer Dreieckskante $> \frac{3}{|E|}$
  - Wahrscheinlichkeit für Auswahl des dritten Knotens $> \frac{1}{|V|-2}$
  - Akzeptanzwahrscheinlichkeit $> 1 - (1 - \frac{3}{|E|\cdot(|V|-2)})^k \doteq 1 - e^{-k\frac{3}{|E|\cdot(|V|-2)}}$
  - $k := \frac{|E|\cdot(|V|-2)}{3}$ erhöht Akzeptanzwahrscheinlichkeit auf mehr als $1/2$
- **Keine falschen Positive**
  - Es wird nur akzeptiert, wenn wirklich ein Dreieck gefunden wird
- **Laufzeit polynomiell (*RP* Algorithmus)**
  - Individueller Test ist linear in $|G|$, Anzahl der Iterationen quadratisch

Primzahltest für $n \geq 3$ (Solovay/Strassen)

Essentiell für offene kryptographische Systeme

1. Ist n gerade dann halte ohne zu akzeptieren
2. Ansonsten wähle $a \in \{1 \ldots n-1\}$ zufällig
3. Ist $ggT(n,a) \neq 1$ dann halte ohne zu akzeptieren
4. Ansonsten teste $a^{(n-1)/2} \equiv J(a,n) \ (mod\ n)$ *(Jacobi Symbol)*
   Ist dies der Fall, dann akzeptiere n, sonst verwerfe n

- RP-Algorithmus für Erkennung von Zusammengesetztheit
  - Für jede Primzahl gilt $a^{(n-1)/2} \equiv J(a,n) \ (mod\ n)$ (Ausgabe korrekt)
  - Für zusammengesetztes n gilt dies für maximal die Hälfte aller $a<n$
    (Wahrscheinlichkeit für korrektes Verwerfen ist mindestens $1/2$)
- Iterative Berechnung von $J(a,n)$ möglich

$a=1$	1
$a<n$ gerade:	$J(a/2,\ n)$, falls $(n^2-1)/8$ gerade, sonst $-J(a/2,\ n)$
$a<n$ ungerade, $ggT(a,n)=1$:	$J(n,a)$, falls $(a-1)(n-1)/4$ gerade, sonst $-J(n,a)$
$a>n$:	$J(a \ mod\ n,\ n)$

- Rechenzeit maximal $6 * (\log n)^2$

Iteration probabilistischer Algorithmen

Iteration kann Fehler extrem klein machen

- **k-fache Iteration von RP Algorithmen reduziert Fehlerwahrscheinlichkeit auf 2^{-k}**
  - Ist M die k-fache statistisch unabhängige Iteration einer PTM M_L für $L \in RP$, so gilt $w \in L \Rightarrow Prob(M \downarrow w) \geq 1-2^{-k}$ und $w \notin L \Rightarrow Prob(M \not\downarrow w) = 1$
  - Einfaches wahrscheinlichkeitstheoretisches Argument
- **$(2t+1)$-fache Iteration eines BPP Algorithmus für $t > \frac{k}{-\log(1-4\epsilon^2)}$ liefert Fehlerwahrscheinlichkeit $< 2^{-k}$**
  - Sei M^t die $(2t+1)$-fache statistisch unabhängige Iteration einer PTM M für $L \in BPP$, die genau dann akzeptiert, wenn M mindestens $t+1$-mal akzeptiert, so gilt für $t > \frac{k-1}{-\log(1-4\epsilon^2)}$ $w \in L \Rightarrow Prob(M^t \downarrow w) > 1-2^{-k}$ und $w \notin L \Rightarrow Prob(M^t \not\downarrow w) > 1-2^{-k}$
  - Aufwendige Analyse (siehe Wegener 75–77 für Details)
- **Keine Aussagen für PP Algorithmen möglich**

Zusammenhänge zwischen den Sprachklassen

- $\mathcal{P} \subseteq ZPP \subseteq RP \subseteq BPP \subseteq PP$
  - ZPP ist wie $\mathcal{P}$, aber Laufzeit ist nur im Erwartungswert polynomiell
  - $ZPP \subseteq RP$ und $BPP \subseteq PP$ folgt direkt aus den Definitionen
  - $RP \subseteq BPP$ folgt aus dem Iterationssatz für RP
- $ZPP = RP \cap co\text{-}RP$ HMU, Satz 11.17
  - Beweis durch gegenseitige Simulation
- $RP \subseteq \mathcal{NP}$ HMU, Satz 11.19
  - Das Verhalten einer PTM kann durch eine NTM M simuliert werden
  - Da die PTM kein Wort $w \notin L$ akzeptiert, akzeptiert M ebenfalls nicht
- $\mathcal{NP} \cup co\text{-}\mathcal{NP} \subseteq PP$
  - NTM akzeptiert, wenn Wahrscheinlichkeit der Akzeptanz nicht Null
  - Aufwendige Simulation durch PP Algorithmen möglich

Sprachklassenhierarchie