Theoretische Informatik II

Einheit 8.3

Komplexität von Problemen

1. Untere Schranken für Komplexität
2. Nichtdeterministische Komplexität
3. Komplexitätsklassen

Komplexität von Problemen

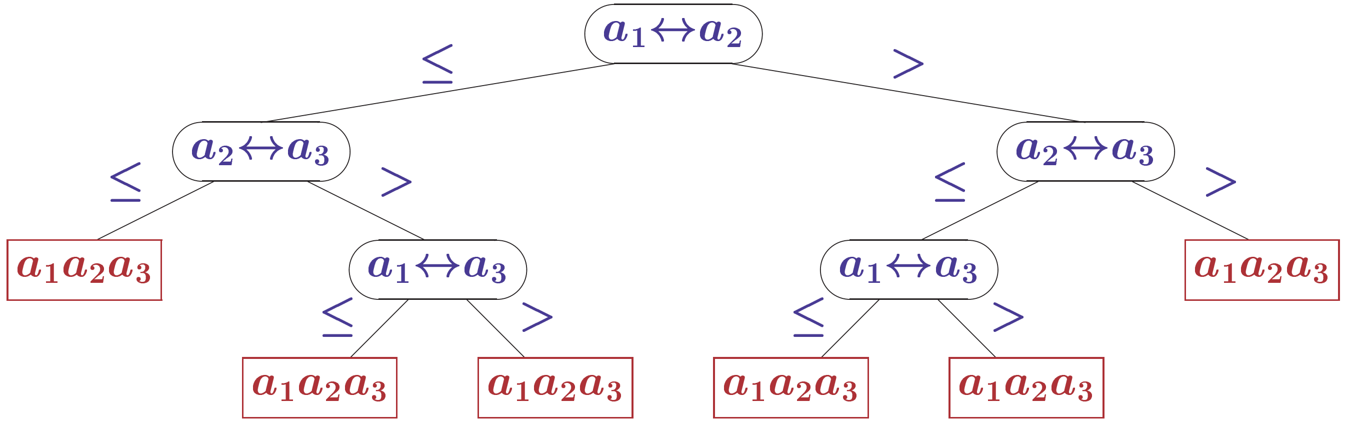
- **Probleme haben unterschiedlich gute Lösungen**
  – Suchen: Lineare Suche $\mathcal{O}(n)$ — Binärsuche $\mathcal{O}(\log_2 n)$
  – Sortieren: Bubblesort $\mathcal{O}(n^2)$ — Mergesort $\mathcal{O}(n * \log_2 n)$
- **Wie effizient kann ein Problem gelöst werden?**
  – Gibt es untere Schranken für die Komplexität von Lösungen
- **Wann ist eine Lösung gut genug?**
  – Ist ein Lösungsalgorithmus optimal bezüglich Zeit-/Platzbedarf?
- **Nachweis aufwendig**
  – Man muß über alle möglichen Algorithmen argumentieren

Komplexität von Sortierverfahren

Schneller als $\mathcal{O}(n * \log_2 n)$?

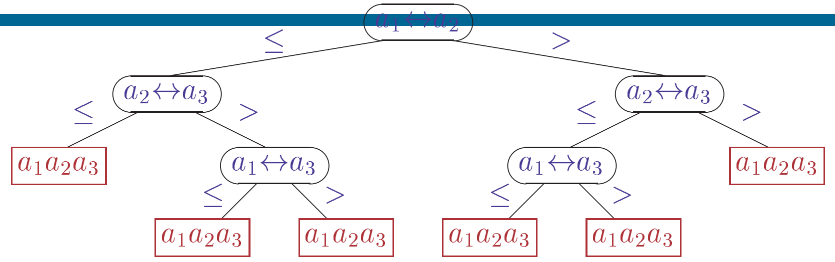
- **Sortierverfahren müssen Elemente vergleichen**
  - Sonst kann die Anordnung der Elemente nicht garantiert werden
  - Wieviel Vergleiche werden benötigt um $a_1..a_n$ zu ordnen?
  - Bestimme Anzahl der Vergleiche für den ungünstigsten Fall

- **Betrachte Entscheidungsbaum von Algorithmen**



  - Innere Knoten entsprechen den durchgeführten Vergleichen
  - Kanten markiert mit Vergleichergebnis ($\leq$,$>$)
  - Blätter sind resultierende Anordnung der Elemente

Komplexität von Sortierverfahren (II)



- **Algorithmen entsprechen Entscheidungsbäumen**
  – Abarbeitung für konkrete Eingaben entspricht einem Ast im Baum
  – Konkrete Laufzeit des Algorithmus entspricht Länge des Astes
  – Komplexität des Algorithmus entspricht Tiefe des Entscheidungsbaumes
  $\mapsto$ Komplexität von Sortieren $\equiv$ minimale Tiefe von Entscheidungsbäumen

- **Wie tief ist ein Entscheidungsbaum?**
  – Jeder Entscheidungsbaum für hat $a_1..a_n$ hat $n!$ Blätter
  – Ein binärer Baum der Tiefe k hat maximal 2^k Blätter
  – Jeder Entscheidungsbaum hat mindestens Tiefe $\log_2 n!$
  – $\log_2 n! = \log_2(\Pi_{i=1}^n i) = \Sigma_{i=1}^n \log_2 i \geq \Sigma_{i=n/2}^n \log_2(n/2) = n/2 * (\log_2 n - 1)$

$$\Downarrow$$

Sortieren ist in $\mathcal{O}(n * \log_2 n)$

Komplexität anderer Problemstellungen

- **Addition n-stelliger Zahlen** $\mathcal{O}(n)$
  - Einstellige Addition von rechts nach links mit Übertrag
- **Multiplikation n-stelliger Zahlen** $\mathcal{O}(n^2)$
  - Jede Stelle muß mit jeder Stelle multipliziert werden
- **Division n-stelliger Zahlen** $\mathcal{O}(n^2)$
  - Schriftliche Division bestimmt Ergebnis von links nach rechts
- **Berechnung von $n!$** $\mathcal{O}(n^2 * (\log_2 n)^2)$
  - Obergrenze: n-fache Multiplikation von n und $n!$: $n * \log_2 n * \log_2(n^n)$
  - Untergrenze: $n/2$-fach $n/2*(n/2)!$: $n/2 * \log_2(n/2) * n/4*(\log_2 n - 2)$
- **Primzahltest bei n-stelliger Zahlen** $\mathcal{O}(2^n)$
  - Teilbarkeit muß für alle kleineren Zahlen getestet werden
  - Untere Schranke $\mathcal{O}(2^n)$ nicht bewiesen $\mapsto$ NP-Vollständigkeit
  - Ergebnis gut für offene kryptographische Systeme (wähle $n > 200$)

Weitere Probleme mit exponentieller Komplexität

- **Travelling Salesman (TSP)**
  Gegeben n Städte, eine Kostentabelle von Kosten c_{ij} um von Stadt i nach j zu reisen und eine Kostenbeschränkung B. Gibt es eine Rundreise durch alle n Städte, deren Kosten unter B liegt?
- **Cliquen-Problem (CLIQUE)**
  Gegeben ein Graph $G = (V, E)$ der Größe n und eine Zahl $k \leq n$. Gibt es in G eine Clique (vollständig verbundener Teilgraph) der Größe k?
- **Erfüllbarkeitsproblem (SAT)**
  Ist eine aussagenlogische Formel in KNF der Größe n erfüllbar?
- **Multiprozessor-Scheduling**
  Verteile n Prozesse derart auf eine Menge von Prozessoren, daß die Ressourcen der Rechner optimal genutzt werden.
- **Binpacking**
  Minimiere Anzahl von Verpackungsbehältern, um n verschieden große Gegenstände zu transportieren.

Bisher nur durch Testen aller Möglichkeiten lösbar

Entscheidungs- und Optimierungsprobleme

Viele Probleme erscheinen in mehreren Varianten

- **Entscheidungsprobleme**
  - Teste ob eine Eingabe $x_1, .., x_n$ eine bestimmte Eigenschaft P erfüllt
  - Suchen in Listen, Primzahltests, Travelling Salesman, Clique ...
- **Berechnungsprobleme**
  - Bei Eingabe von $x_1, .., x_n$ berechne ein y, so daß $P(x_1, .., x_n, y)$ gilt
  - Sortieren, Primfaktorzerlegung, Matrixmultiplikation, ...
- **Optimierungsprobleme**
  - Bei Eingabe von $x_1, .., x_n$ berechne das beste y mit $P(x_1, .., x_n, y)$
  - Travelling Salesman, Clique, Binpacking, Multiprocessor Scheduling

Varianten des Cliquenproblems

Gegeben ein Graph $G = (V, E)$ und eine Zahl $k \leq |V|$. Eine k-Clique von G ist ein vollständig verbundener Teilgraph $C = (V_c, E_c)$ mit $|V_c| = k$

- **Entscheidungsproblem**
  - "Gibt es eine Lösung mit einem bestimmten Wert"
    - CLIQUE: Gibt es in G eine Clique der Größe k?
- **Berechnungsproblem**
  - Bestimme eine konkrete Lösung mit einem bestimmten Wert
    - CLIQUE_2: Bestimme eine Clique $C \subseteq G$ der Größe k
- **Optimierungsprobleme**
  - Bestimme den Wert einer optimalen Lösung
    - CLIQUE_{opt}: Bestimme das größte k, so daß G eine k-Clique enthält
  - Berechne die optimale Lösung
    - CLIQUE_{opt_2}: Bestimme eine Clique $C \subseteq G$ mit maximaler Größe k

Problemvarianten von Clique sind gleich schwer

- **Löse CLIQUE$_{opt}$ mit CLIQUE**
  - Beginne mit $k := |V|$ und teste ob es in G eine k-Clique gibt
  - Reduziere k bis der Test erfolgreich ist und gebe k aus
  - Zusatzaufwand ist linear in $|V|$
- **Löse CLIQUE$_{opt_2}$ mit CLIQUE$_{opt}$**
  - Bestimme k_{opt} für G und beginne mit $E_c := E$
  - Wähle Kante $e \in E$ und teste ob es in $(V, E_c - \{e\})$ eine k_{opt}-Clique gibt
  - Ist dies der Fall, so setze $E_c := E_c - \{e\}$
  - Wiederhole dies iterativ für alle Kanten aus E
  - Die resultierende Menge E_c und die zugehörigen Ecken bilden die k_{opt}-Clique
  - Zusatzaufwand ist linear in $|E|$

Löse analog CLIQUE$_2$ mit CLIQUE

Die Umkehrungen sind trivial

$$\Downarrow$$

$$\boxed{\textbf{Es reicht Entscheidungsprobleme zu analysieren}}$$

Viele schwere Probleme haben leichte Erfolgstests

- **Travelling Salesman:** Für eine gegebene Rundreise $i_1..i_n$ können die Kosten $c_{i_1 i_2} + \ldots c_{i_n i_1}$ in linearer Zeit berechnet und mit der Kostenbeschränkung B verglichen werden
- **Cliquen-Problem:** Ein gegebener Teilgraph der Größe k kann in polynomieller Zeit auf Vollständigkeit überprüft werden
- **Erfüllbarkeitsproblem:** Man kann in polynomieller Zeit testen, ob eine gegebene Belegung der Variablen eine Formel erfüllt
- **Binpacking:** Man kann in polynomieller Zeit testen, ob eine gegebene Verteilung der Gegenstände in k Verpackungsbehälter paßt
- **Zusammengesetztheitstest:** Man kann in quadratischer Zeit testen, ob eine gegebene Zahl Teiler von x (also x keine Primzahl) ist

Nichtdeterministische Maschinen liefern polynomielle Lösung

Nichtdeterministische Lösbarkeit

Asteroth/Baier §2.3

- **Nichtdeterministische Turingmaschine τ**
  - Komponenten S, X, Γ, s_0, b wie bei normaler Turingmaschine
  - Zustandsüberführungsfunktion $\delta{:}S{\times}\Gamma \to 2^{S\times\Gamma\times\{r,l,h\}}$
  - Nachfolgekonfigurationsfunktion $\hat{\delta}{:}K_\tau{\to}2^{K_\tau}$
  - Semantik $h_\tau{:}X^*{\to}2^{\Gamma^*}$

    $h_\tau(w) = \perp$, wenn $\hat{\delta}$ unendliche Konfigurationsfolgen ermöglicht
  - **Keine Erweiterung des Berechenbarkeitsbegriffs**

- **Nichtdeterministische Entscheidbarkeit**
  - **τ entscheidet $M{\subseteq}X^*$**, falls $w \in M \;\Leftrightarrow\; 1 \in h_\tau(w)$

    "Es gibt eine akzeptierende Berechnung für w"
  - Rechenzeit $\boldsymbol{nt_\tau(w)} = \begin{cases} \text{minimale Länge einer akzep–} & \text{falls } w \in M \\ \text{tierenden Berechnung für } w & \\ 0 & \text{sonst} \end{cases}$
  - Zeitkomplexität $\boldsymbol{ntime_\tau(n)} = \max\{nt_\tau(w) \mid |w|{=}n\}$
  - Platzkomplexität $\boldsymbol{nspace_\tau(n)}$ analog definiert

Nichtdeterminismus als Raten und Verifizieren

- **Deterministische Turingmaschine mit Orakel**
  - Eingabe des Wortes w auf erstem Arbeitsband
  - Phase 1: Orakel generiert Wort w' auf zweitem Arbeitsband
  - Phase 2: τ verarbeitet w und w' deterministisch
  - τ entscheidet $M \subseteq X^*$, falls $\forall w, w'. h_\tau(w, w') \neq \perp$
    und $w \in M \;\Leftrightarrow\; \exists w'.\, h_\tau(w, w') = 1$
  - Rechenzeit $ot_\tau(w) = \min\{t_\tau(w, w') \mid h_\tau(w, w') = 1\}$ $\quad$ (0 falls $w \notin M$)
    Orakel benötigt keine Rechenzeit zum Schreiben
  - Zeitkomplexität $otime_\tau(n) = \max\{ot_\tau(w) \mid |w| = n\}$
- **NTM's und OTM's sind äquivalent**
  - OTM kann Berechnungsfolge der NTM raten und deterministisch ausführen
  - NTM kann alle Berechnungsfolgen der OTM “parallel” ausführen
  - **Rechenzeit ist identisch**

Komplexität von (Entscheidungs-)Problemen

- **Zeitkomplexität**
  - Eine Menge M hat Zeitkomplexität $\mathcal{O}(f)$, falls es eine DTM τ mit $time_\tau \in \mathcal{O}(f)$ gibt, die M entscheidet.
  - **DTIME(f)** $= \{M \mid M$ hat Zeitkomplexität $\mathcal{O}(f)\ \}$
  - Eine Menge M hat nichtdeterministische Zeitkomplexität $\mathcal{O}(f)$, falls es eine NTM τ mit $ntime_\tau \in \mathcal{O}(f)$ gibt, die M entscheidet.
  - **NTIME(f)** $= \{M \mid M$ hat nichtdeterministische Zeitkomplexität $\mathcal{O}(f)\}$
- **Platzkomplexität**
  - Eine Menge M hat Platzkomplexität $\mathcal{O}(f)$, falls es eine DTM τ mit $space_\tau \in \mathcal{O}(f)$ gibt, die M entscheidet.
  - **DSPACE(f)** $= \{M \mid M$ hat Platzkomplexität $\mathcal{O}(f)\ \}$
  - Eine Menge M hat nichtdeterministische Platzkomplexität $\mathcal{O}(f)$, falls es eine NTM τ mit $nspace_\tau \in \mathcal{O}(f)$ gibt, die M entscheidet.
  - **NSPACE(f)** $= \{M \mid M$ hat nichtdeterministische Platzkomplexität $\mathcal{O}(f)\}$

Konkrete Analysen werden mit abstrakten deterministischen oder nichtdeterministischen Algorithmen durchgeführt

Wichtige Komplexitätsklassen

- $\mathcal{P} = \bigcup_k \; DTIME(n^k)$: **Effiziente Lösbarkeit**
  - Menge der in polynomieller Zeit lösbaren Probleme
- $\mathcal{NP} = \bigcup_k \; NTIME(n^k)$:
  - Menge der nichtdeterministisch in polynomieller Zeit lösbaren Probleme
- **Weitere Zeitkomplexitätsklassen**
  - $EXPTIME = \bigcup_k \; DTIME(2^{n^k})$
  - $NEXPTIME = \bigcup_k \; NTIME(2^{n^k})$
  - $LOGTIME = DTIME(\log_2 n)$
  - $NLOGTIME = NTIME(\log_2 n)$
- **Platzkomplexitätsklassen**
  - $LOGSPACE = DSPACE(\log_2 n)$
  - $NLOGSPACE = NSPACE(\log_2 n)$
  - $PSPACE = \bigcup_k \; DSPACE(n^k)$
  - $NPSPACE = \bigcup_k \; NSPACE(n^k)$
  - $EXPSPACE = \bigcup_k \; DSPACE(2^{n^k})$
  - $EXPSPACE = \bigcup_k \; NSPACE(2^{n^k})$

Beispiele für Komplexitätsklassen

- $\mathcal{P}$:
  - Arithmetische Operationen, Sortieren, Matrixmultiplikation, ...
- $\mathcal{NP}$:
  - Travelling Salesman, Cliquen-Problem, Erfüllbarkeitsproblem
  - Multiprozessor-Scheduling, Binpacking, Zusammengesetztheitstest
- $\boldsymbol{EXPTIME}$:
  - Travelling Salesman, Cliquen-Problem, Erfüllbarkeitsproblem
  - Multiprozessor-Scheduling, Binpacking, Primzahltest
- **Komplexitätsklassenhierarchie**

$$LOGTIME \subseteq NLOGTIME \subseteq LOGSPACE \subseteq NLOGSPACE$$
$$\subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE \subseteq NPSPACE$$
$$\subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \ldots$$

  - Es wird vermutet, daß alle Inklusionen echt sind