Formale Methoden 2 LVA 703019, 703020

(http://cl-informatik.uibk.ac.at/teaching/ss06/fmII/)

Georg Moser (VO)[1] Martin Korp (UE)[2]
Friedrich Neurauter (UE)[3] Christian Vogt (UE)[4]

[1] georg.moser@uibk.ac.at	Sprechstunden:	Mittwoch, 13:00–15:00 (3M09)
[2] csac9615@uibk.ac.at	Sprechstunden:	Freitag, 12:00–14:00 (3M03)
[3] csad2836@uibk.ac.at	Sprechstunden:	Montag, 11:00–13:00 (3M03)
[4] christian.vogt@uibk.ac.at	Sprechstunden:	Donnerstag, 10:00–12:00 (3M12)

Sommersemester 2006

Einführung in die Berechenbarkeitstheorie

- ➡ Ist jedes Problem algorithmisch lösbar?
- ➡ Nein!

Ein einfaches Programm:

```
main()
{
      printf(‘‘hello, world\n’’);
}
```


Ein untypisches “hello-world” Programm

```
main()
{
      int n, summe, x, y, z;
      scanf(‘‘%d’’, &n);
      summe = 3;
      while (1) {
          for (x=1; x <= summe-2; x++)
              for (y=1; y<=summe-x-1; y++) {
                  z = summe - x - y;
                  if (exp(x,n) + exp(y,n) == exp(z,n))
                     printf(‘‘hello, world\n’’);
              }
          summe++;
      }
}
```


Noch ein untypisches “hello-world” Programm

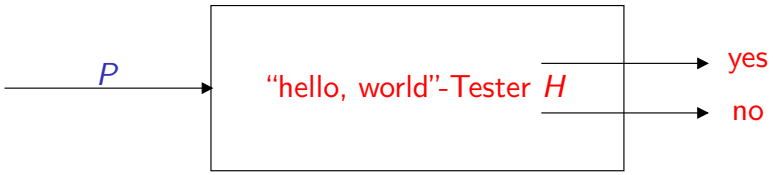
```
main()
{
      int n, summe, x, y, z;
      summe = 4;
      while (1) {
          for (x=2; x <= summe; x++) {
                  y = summe - x;
                  if (primes(x) && primes(y))
                     printf(‘‘hello, world\n’’);
              }
          summe=summe+2;
      }
}
```

- ➡ Druckt dieses beziehungsweise das vorherige Programm “hello, world” ?

Es kann keinen "hello, world"-Tester geben

- Die angegebenen Beispiele zeigen, dass es nicht trivial ist festzustellen, ob ein bestimmtes Programm als erste Ausgabe "hello, world" schreibt.

Nun betrachten wir ein hypothetisches Programm H, das wir "hello, world" Tester nennen.



- Man kann zeigen, dass ein "hello, world"-Tester nicht existieren kann; es ist unentscheidbar, ob ein beliebiges Programm ein "hello, world" Programm ist.

Unvollständige Liste unentscheidbarer Probleme

Die folgenden Probleme sind unentscheidbar:

- Das Problem, ob ein beliebiges Programm auf seiner Eingabe hält (Halteproblem)
- Postsches Korrespondenzproblem (PCP): Gegeben zwei Listen von Strings der gleichen Länge $w_1, w_2, \ldots, w_n$ und $x_1, x_2, \ldots, x_n$. Gesucht sind Indizes $i_1, i_2, \ldots, i_m$, sodass

$$w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$$

- Das Problem, ob eine KFG eindeutig ist.
- Das Problem, ob für eine KFG G, gilt $\mathrm{L}(G) = \Sigma^*$.
- ...

Pioniere der theoretischen Informatik

Vollständigkeit der erststufigen Logik; Unvollständigkeit der Arithmetik

Turing Maschinen; Unentscheidbarkeit des Halteproblems

Einführung in die Komplexitätstheorie

- Komplexitätstheorie: Analysiert Algorithmen und Probleme, zentrale Frage:

  Welche Resourcen benötigt ein bestimmter Algorithmus oder ein Problem?

- Resourcen sind etwa Speicherplatz oder Rechenzeit
- Ein Problem ist eine allgemeine, zu beantwortende Frage
- Ein Algorithmus ist ein Verfahren zur Beantwortung so einer Frage
- Komplexität eines Algorithmus: Die notwendigerweise aufgewandten Resourcen (als Funktion der Eingabe)
- Komplexität eines Problems: Komplexität des effizientesten Algorithmus der das Problem löst.

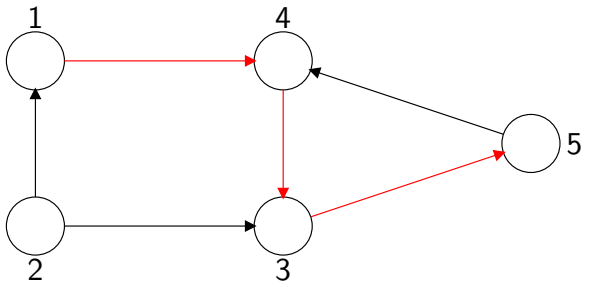
Das Problem REACHABILITY

Ein Graph $G = (V, E)$ besteht aus einer endlichen Menge von Knoten V und einer Menge von Kanten E, geordnete Paare von Knoten.

- Gegeben ein Graph G und Knoten $1, n \in V$. Gibt es einen (gerichteten) Pfad zwischen 1 und n?

Dieses Problem nennt man REACHABILITY.

Beispiel



Dijsktra Algorithmus

- $S := \{1\}$; Markiere Knoten 1.
- Wähle $i \in S$, lösche i aus S
  - Wenn $(i, j) \in E$ und j unmarkiert, markiere j, füge j zu S hinzu
- Iteriere bis S leer ist. Antwort "Ja" wenn n markiert, sonst "Nein"

Bemerkung: Je nach Wahl der Auswahlfunktion kann die Suche depth-first oder breadth-first sein.

Beispiel

- Setze $S := \{1\}$; Markiere Knoten 1.
- Wähle $1 \in S$, und lösche 1, also $S = \emptyset$
- Wähle die Kante $(1, 4)$, 4 ist unmarkiert, also markiere 4 und setze $S = \{4\}$
- Lösche 4 aus S,
- Wähle die Kante $(4, 3)$; 3 ist unmarkiert, markiere 3, setze $S = \{3\}$
- Lösche 3 aus S
- Wähle die Kante $(3, 5)$; markiere 5, setze $S = \{5\}$
- Wähle $5 \in S$ und lösche 5 aus S, also $S = \emptyset$
- Nun kann keine Kante $(5, j)$ gefunden werden, sodass j unmarkiert.
- 5 wurde markiert, also antworte mit "Ja"

Aufgewandte Resourcen

Zeitkomplexität

- Die Zeitkomplexität des Suchalgorithmus ist $\mathcal{O}(n^2)$, wobei n die Anzahl der Knoten darstellt.
- Begründung: Jede Kante muss maximal einmal behandelt werden.
- Das heißt die verbrauchte Zeit hängt polynomiell von der Anzahl der Knoten ab.
- Wenn wir als Eingabe des Algorithmus den Graphen verstehen, dann können wir das verallgemeinern: Die Zeitkomplexität ist polynomiell in der Länge der Eingabe

Platzkomplexität

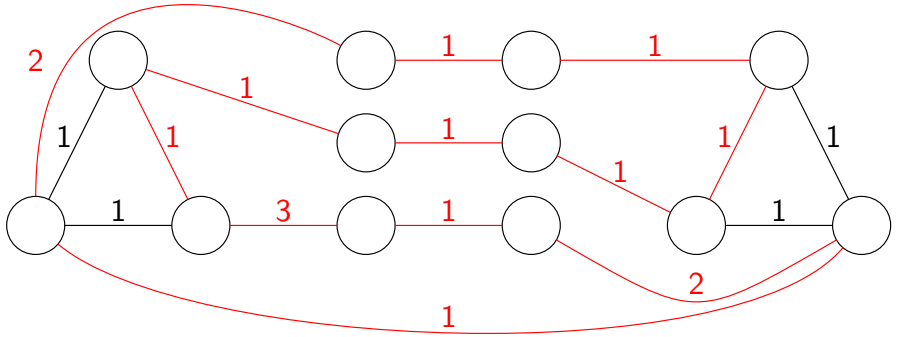
- Der Platzbedarf des Suchalgorithmus ist $\mathcal{O}(n)$, aber eine Verbesserung auf $\mathcal{O}(\log^2 n)$ ist möglich.

worst-case analysis

- In der Komplexitätstheorie betrachtet man hauptsächlich den schlimmsten denkbaren Fall (worst-case analysis)
- Außerdem interessiert man sich nur für die Wachstumsraten der Schranken, etwa ob diese polynomiell oder exponentiell sind.
- Wir betrachten Probleme, deren Komplexität polynomiell wächst als (einfach) lösbar, Problem, deren Komplexität aber exponentiell wächst als unlösbar.
- Average case analysis wäre besser, aber
  - Was ist die richtige Eingabeverteilung des Problems?
  - Unsere Analyse muss generell sein, den schlimmsten Fall also ebenfalls einschließen

Das Problem TSP

Gegeben n Städte, mit Distanz $d_{ij} > 0$, sodass $d_{ij} = d_{ji}$.



Was ist die billigste Tour durch die Städte, das heißt das Minimum der Summe:

$$\sum_{i=1}^{n} d_{\pi(i),\pi(i+1)} \quad \pi : \{1,\ldots,n\} \to \{1,\ldots,n\}, \pi(n+1) = \pi(1) \; .$$

Naiver Algorithmus

- Nummeriere alle möglichen Lösungen.
- Berechne die Kosten.
- Wähle die Tour mit den geringsten Kosten.

Komplexität

- Zeitschranke: $\mathcal{O}(n!)$
- Platzschranke: $\mathcal{O}(n)$.

Auch für klügere Algorithmen läßt sich die Zeitschranke nur leicht verbessern, sie bleibt jedenfalls exponentiell!

- Wenn ein solcher Algorithmus existiert, dann gilt **P** = **NP**. Die Frage ob **P** = **NP** ist eines der 7 Milleniumprobleme des Clay Institutes

Das **P** ≠ **NP** Problem

Was ist **P**

- Informelle Antwort: Die Klasse **P** ist eine Klasse von Entscheidungsproblemen (Probleme, die mit "Ja" oder "Nein" beantwortbar sind.) die in polynomieller Zeit berechenbar, also algorithmisch lösbar sind.
- P heißt Entscheidungsprozedur für eine Sprache $\mathrm{L} \subseteq \Sigma^*$, wenn
  - P für jede Eingabe hält,
  - P Eingabe x akzeptiert, wenn $x \in \mathrm{L}$ und
  - P Eingabe x verwirft, wenn $x \notin \mathrm{L}$.
- Wenn P die Eingabe in Zeit $\mathcal{O}(n^k)$ entscheidet, ist P ein polytime-Algorithmus; n bezeichnet die Länge der Eingabe.

P = $\{\mathrm{L} \mid$ Es gibt eine polytime-Entscheidungsprozedur P für $\mathrm{L}\}$

Was ist **NP**?

- Ein Verifikator einer Sprache $\mathrm{L} \subseteq \Sigma^*$ ist ein Algorithmus P, sodass

  $\mathrm{L} = \{w \mid \text{es existiert ein String } c, \text{ sodass P akzeptiert } \langle w, c\rangle\}$

- Ein polynomieller Verifikator ist ein Verifikator, der in Zeit $\mathcal{O}(n^k)$ läuft, wobei $n = |w|$
- Der String c wird Zertifikat genannt; dieser String entspricht einer Sequenz von Entscheidungen einer nicht-deterministischen Maschine.

$$\mathbf{NP} = \{\mathrm{L} \mid \mathrm{L} \text{ hat einen polynomiellen Verifikator}\}$$

TSP $\in$ **NP**

Zunächst formulieren wir das Entscheidungsproblem zu TSP, bezeichnet mit TSP(D)

- Gegeben die Eingabe zu TSP und eine natürliche Zahl B
- Gilt dann

$$\sum_{i=1}^{n} d_{\pi(i),\pi(i+1)} \leq B \quad \text{wenn } \langle \pi(1), \ldots, \pi(n), \pi(1)\rangle \text{ eine Tour?}$$

Pseudo-code für den Verifier:

- Eingabe: $\langle$Instanz von TSP(D), bestimmte Tour$\rangle$.
- Prüfe, ob die Kosten der Tour ($= \langle \pi(1), \ldots, \pi(n), \pi(n+1)\rangle$) $\leq B$.
- Test kostet maximal $\mathcal{O}(n)$.
- Antworte "Ja", wenn Bedingung erfüllt, sonst "Nein".

Zusammenfassung

- Einführung in die Berechenbarkeitstheorie
- Unentscheidbarkeit
- Einführung in die Komplexitätstheorie
- Die Klassen von Sprachen **P** und **NP**