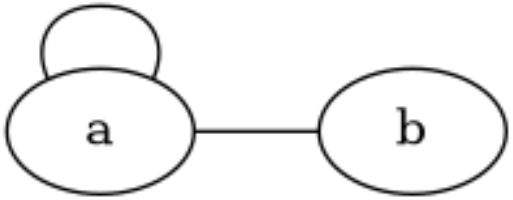
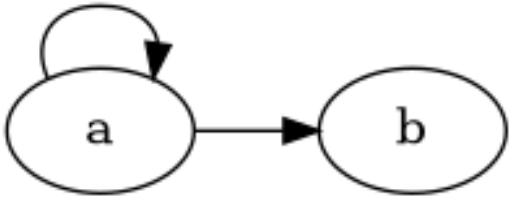
1 Week 1 empty

1.1 Basics

Das folgende ist **kein** Graph, weil jede Kante brauch einen Anfangs- und Endknoten.



Dies ist allerdings ein Graph (Ein Anfangskonten und Endknoten definiert durch die Pfeile).



1.2 Sprache

- Alphabet: normalerweise griechische Buchstaben (Σ)
- Die Menge aller Wörter: Σ^*
- Das leere Wort: ϵ
- Die Sprache ist eine Teilmenge aller Wörter: $L \subset \Sigma^*$

2 Week 2

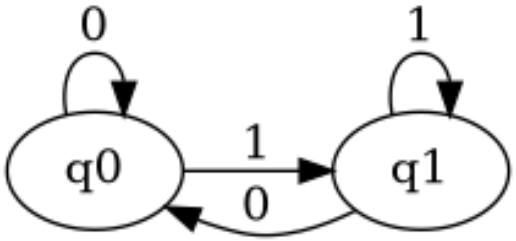
2.1 DEA (Deterministische endliche Automaten)

2.1.1 Definition

1. Q endliche Menge von Zuständen
2. Σ endliche Menge, das Alphabet
3. $\delta : Q \times \Sigma \to Q$ Übergangsfunktion
4. $q_0 \in Q$ Startzustand
5. $F \subset Q$ Menge der Akzeptiertzustände
   Von jedem Zustand, muss es für jedes Zeichen genau einen Übergang geben. Es ist also zu jeder Zeit klar, wo der Automat genau ist.

2.1.2 Beispiel

q	a	$\delta(q,a)$
q0	0	q0
q0	1	q1
q1	0	q0
q1	1	q1



2.1.3 Myhill-Nerode

Ist L eine requläre Sprache, dann wird L von dem endliche Automaten $A = (Q, \Sigma, \delta, q_0, F)$ akzeptiert mit

$$Q = \{L(w) | w \in \Sigma^*\}$$
$$q_0 = L$$
$$F = \{q \in Q | \epsilon \in q\}$$
$$\delta = Q \times \Sigma \to Q : (L(w), a) \mapsto L(wa)$$

Dieser Satz kann benutz werden, um zu entscheiden ob eine Sprache regulär ist: Wenn $\{L(w) | w \in \Sigma^*\}$ endlich ist, ist die Sprache regulär. Als Beispiel ist die Sprache $\{0^n 1^n | n \in \mathbb{N}\}$ über $\Sigma = \{0, 1\}$ nicht regulär, da man unendliche viele Zustände benötigt. Eine DEA hat **kein** Gedächnis.

2.1.4 Minimaler Automat

Automaten können reduziert werden zu einem minimalen Automaten. Wenn zwei verschiedene DEAs auf den selben minimalen DEA reduziert werden können, akzeptieren die beiden Automaten die selbe Sprache. Der minimale Automat kann mit dem Kreuzchen Algorithmus konstruiert werden.

3 Week 3

- Nicht deterministische endliche Automaten
  - "Könnte"-Automat
  - Transformation NEA -> DEA
- Mengenoperationen
  - Vereinigung
  - Durchschnitt (Produktautomat)
  - Differenz

3.1 Pumping Lema - Reguläre Sprachen

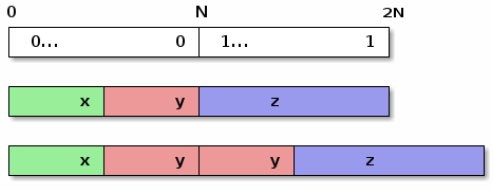
Mit dem Pumping Lema kann erkennt werden, ob eine Sprache nicht regulär ist. Wenn das Wort lange genug ist (Puming Length N), dann kann das Wort in drei Teile (x, y, z) aufgeteilt werden:

1. $|y| > 0$
2. $|xy| \leq \mathbb{N}$
3. $xy^k z \in L \forall \geq 0$

3.1.1 Beispiel

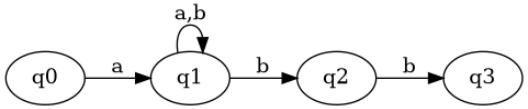
Beweise dass $L = \{0^n 1^n | n \in \mathbb{N}\}$ nicht regulär ist.

1. Annahme L sei regulär
2. Da regulär und lange genug, gibt es eine Pumping Length N
3. Bilde ein Wort mit der Länge N (z.B. $0^N 1^N$)
4. Teile das Wort in xyz auf mit, $|xy| \leq N, |y| \geq 1$
5. Wortteil y aufpumpen
6. Widerspruch: Wort nicht mehr in Sprache



3.2 NEA (Nicht deterministische endliche Automaten

NEAs unterscheiden sich von DEAs nur geringfügig. Bei einem NEA ist es nicht notwendig, für jedes Zeichen einen Übergang zu haben. Zusätzlich können für das selbe Zeichen auch mehrere Übergänge existieren. Neben den bekannten Übergängen existieren im NEA auch ϵ-Übergänge. Diese Übergänge können zu jeder Zeit, ohne ein Inputzeichen zu verwenden, benutzt werden.
NEAs können aber nicht mehr Sprachen erkennen als die DEAs. Sie sind daher gleichwertig. Als Beispiel einen NEA, welcher Wörter erkennt, welche mit 2 b enden.



3.3 Thompson-NEA (Könnte-Automat)

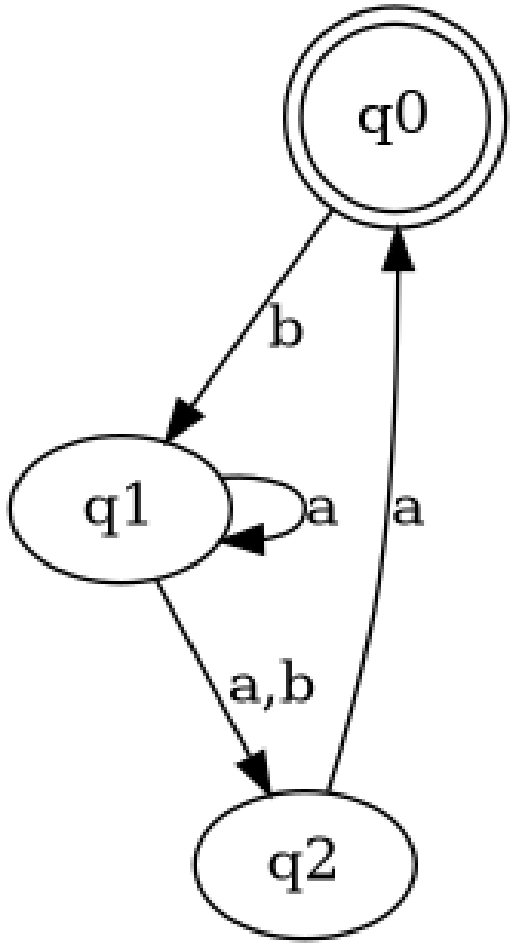
Bei diesem NEA wird sich immer gemerkt, in welchem Zustand der Automat sein könnte.

3.4 Transformation von NEA zu DEA

Bei der Transformaiton von einem NEA zu einem DEA geht es darum, die ϵ-Übergänge und die Mehrfachübergänge zu eliminieren. Das erreicht man, in dem der DEA Buch führt, in welchen Zuständen der NEA sein könnte. Das wird realisiert, in dem die Zustände des DEA die möglichen Zuständen des NEAs repräsentieren. Die Zustände Q des DEA ist die Potzenmenge der Menge Q des NEAs.

3.4.1 Beispiel

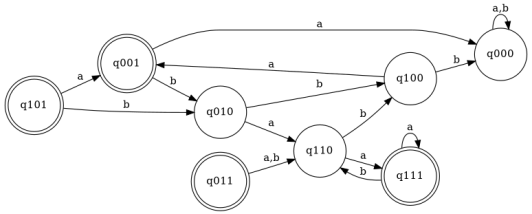
Folgender NEA soll in einen DEA überführt werden:



Dafür müssen die Potzenmenge der Zustände des NEAs gebildet werden:

- $q_{000} = \{\}$
- $q_{001} = \{q_0\}$
- $q_{010} = \{q_1\}$
- $q_{100} = \{q_2\}$
- $q_{110} = \{q_{2,q1}\}$
- $q_{101} = \{q_{2,q0}\}$
- $q_{011} = \{q_{1,q0}\}$
- $q_{111} = \{q_{2,q1,q0}\}$

Akzeptiertzustände sind alle welche den Status q_0 beinhalten => F = $\{q_{001}, q_{101}, q_{011}, q_{111}\}$. Startzustand ist q_{001}. Nun kann das Zustandsdiagramm gezeichnet werden:

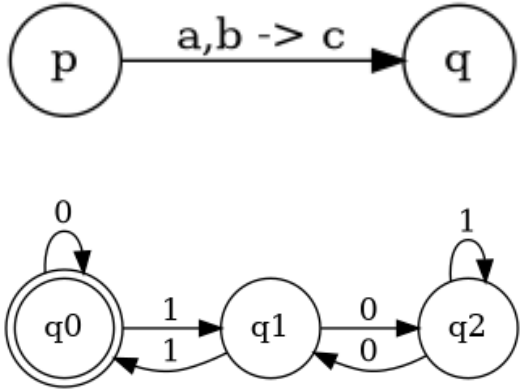


Im Bild ist ersichtlich, dass die Zustände q_{101} und q_{011} nie erreicht werden können und somit weggelassen werden können.

3.5 Mengenoperationen

Sprachen sind Menge von Wörter. Folglich sind auch deren Vereinigung, Durchschnitt, Differenz etc. auch Sprachen. Wenn die Sprachen reguläre sind, ist auch dessen Vereinigung etc. Dies lässt sich mit einem NEA einfach beweisen.

Um die Schnittmenge zu bilden wird ein **kartesischer Produktautomat** benötigt.



4 Week 4

- Reguläre Operationen
  - Alternative
  - Verkettung
  - *-Operation
- Umwandlung DEA <-> regulärer Ausdruck
- Reguläre Ausdrücke in der Praxis
  - Scanner-Generator flex
  - Performance von Regex-Engines

4.1 TODO Reguläre Ausdrücke (Regex)

Reguläre Ausdrücke sind eigentlich nichts anderes als eine kompakte Schreibweise für NEAs (folglich auch für DEAs).

Ausdruck	Bedeutung
a	das Zeichen $a \in \Sigma$
.	beliebiges Zeichen aus Σ
aei	ein Zeichen aus $a, e, i \subset \Sigma$
a-z	für kleine Buchstaben von a-z
ϵ	steht für das leere Wort
$\emptyset$	steht für die leere Sprach
$r_1 \vert r_2$	Alternative
r*	Mehrfaches vorkommen von r
r+	Mindestens einmal r
r?	0 oder 1 mal r

4.2 DEAs in Regex umwandeln

Jeder DEA kann mit Regex vereinfacht werden. Folgendes Vorgehen:

1. Neuer Akzeptiertzustand (q_{accept}), alle andere werden zu normalen Zuständen degradiert und mit q_{accept} verbunden
2. Neuer Startzustand (q_{start}), andere werden zu normalen Zuständen degradiert und mit q_{start} verbunden
3. Nach und nach alle Verbindungen raus nehmen und durch Regex reduzieren bis nur noch q_{start} -> q_{accept} vorig ist

5 Week 5

- Kapitel 4: Stackautomaten und kontextfreie Sprachen
  - Kontextfreie Grammatiken
  - Kontextfreie Sprachen
  - Beispiele
  - Reguläre Operationen für kontextfreie Grammatiken
  - Chomsky Normalform

5.1 Kontextfreie Sprachen

Eine kontextfreie Grammatik besteht aus:

- endliche Mengen von Variablen
- endliche Menge von Zeichen (Terminalsymbole)
- eine Menge von Regel
- Eine Startvariabel

Kontextfrei bedeutet in diesem Fall, dass es auf der linken Seite der Regel immer nur eine Variabel gibt. Kontextsensitive Regel:

$$aA \to AA, bA \to BB$$

Kontextfrei Regel:

$$A \to AA$$

Durch anwenden der Grammatik bis keine Variabel mehr übrigbleibt erzeugt eine **Kontextfreie Sprache**. Reguläre Sprachen sind eine Untermenge von kontextfreien Sprachen

5.2 Chomsky Normalform

Um zwei Grammatiken vergleichen zu können braucht es eine Normalform. Diese ist die Chomsky Normalform. Um in diese Form zu gelangen geht man wie folgt vor:

1. Die Startvariabel darf rechts nicht vorkommen. Notfalls eine neue Startvariabel S_0 erstellen mit der Regel $S_0 \to S$
2. Alle ϵ Regel entfernen. Die einzige Ausnahme ist $S_0 \to \epsilon$
3. Entfernen von *unit rules*: Regeln mit einer einzelnen Variabel auf der rechten Seite ($A \to B$)
4. Verkettungen auflösen: $A \to u_1 A_1$. u_1 wird durch U_1 ersetzte und $U_1 \to u_1$

6 Week 6

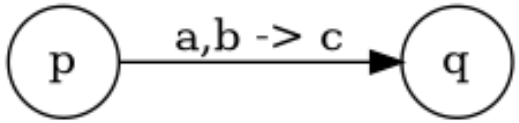
- deterministischer Parse Algorithmus (Cocke-Younger-Kasami)
- Stackautomaten
  - Beispiel zur Motivation: $0^n 1^n | \ n \geq 0$
  - Formale Definition
  - Stackautomat als gerichterer beschrifteter Graph
  - Stackautomat einer kontextfreien Grammatik
- Anwendung: Parser-Generator Bison

6.1 Cocke-Younger-Kasami

Um zu überprüfen, ob eine Wort zu einer kontextfreien Sprache gehört gibt es den **Cocke-Younger-Kasami**-Algorithmus. Dieser geht für das ganze Wort durch die Chomsky Normalform. Wenn es akzeptiert wird gehört es zu der Sprache ansonsten nicht.

6.2 Stackautomaten

Stackautomaten sind DEAs welche mit einem Speicher, einem Stack, ausgestattet sind. So kann die 0 auf den Stack gespeichert werden bei der Sprache $0^n 1^n | n \in \mathbb{N}$. Wenn der Stack am ende leer ist, wird das Wort akzeptiert. Bei Stackautomaten gibt es noch ein zusätzliches Alphabet, das Stack-Alphabet. Dieses wird verwendet um Symbole auf den Stack zu schreiben, bzw. von dort lesen.
Folgender Stackautomat, geht vom Zustand p nach q über, wenn ein a verarbeitet wird und gleichzeitig das Element c durch ein b ersetzt werden kann.

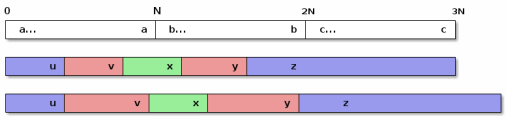


7 Week 7

- Pumping Lemma für kontextfreie Sprachen
- Beispiel: $\{ a^n b^n c^n \mid n \geq 0 \}$
- kontextfreie Grammatik eines Stackautomaten

7.1 Pumping Lemma für kontextfreie Sprachen

Ähnliche wie bei Regulären Sprachen gibt es auch ein Pumping Lemma für kontextfreie Sprachen.



1. Annahme: L ist kontextfrei
2. Es gibt eine Pumping Length N
3. Wort: $w = a^N b^N c^N$
4. Wort in u, v, x, y, z zerlegen $|vy| > 0$, $|vxy| \leq N$, $uv^k xy^k z \in L \forall k \in \mathbb{N}$

5. v und y aufpumpen
6. Widerspruch: L nicht kontextfrei

8 Week 8

- Kapitel 5: Turing Maschinen
  - Definition Turing Maschine, erkannte Sprache
  - Zustandsdiagramm
  - Varianten (Bandalphabet, Anzahl Spuren, Anzahl Schreib-/Leseköpfe)
  - Aufzähler
  - Nicht deterministische Turingmaschinen

  Jede nichtdeterministische Turingmaschine ist äquivalent zu einer deterministischen Turingmaschine.

9 Week 9

- Abzählbar unendlich und überabzählbar unendlich
- Die meisten Zahlen sind nicht berechenbar
- Das 10. Hilbertsche Problem
- Kapitel: 6 Entscheidbarkeit
  - Akzeptanzprobleme für reguläre Sprachen
  - Leerheitsproblem für reguläre Sprachen
  - Gleichheitsproblem für reguläre Sprachen
  - Akzeptanzproblem für kontextfreie Sprachen
  - Leerheitsproblem für kontextfreie Sprachen
  - Gleichheitsproblem für kontextfreie Sprachen
- Entscheidbarkeitsprobleme für kontextfreie Sprachen
  - Akzeptanzproblem für kontextfreie Sprachen
  - Leerheitsproblem für kontextfreie Sprachen
  - Gleichheitsproblem für kontextfreie Sprachen
- Halteproblem
  - Akzeptanzproblem für Turing Maschinen Reduktion
  - Allgemeines Halteproblem
  - Leerheitsproblem für TM
- Reduktion
- Satz von Rice

9.1 Satz von Rice

Sei P eine nicht triviale Eigenschaft von Turing-erkennbaren Sprachen, dann ist P nicht entscheidbar

Nicht trivial meint, dass gewisse Sprachen die Eigenschaft haben und andere nicht.

10 Week 10

- Kapitel 7: Komplexitätstheorie
- Laufzeitkomplexität
  - Definition der Laufzeit
  - Laufzeit für Varianten von Turingmaschinen
- Komplexitätsklassen P und NP
  - Beispiele von Sprachen in P
  - Verifizierer
  - Polynomielle Reduktion

10.1 Verifizierer

Wenn eine nichtdeterministische Turing Maschine ein Problem in polynomieller Zeit lösen kann, gibt es einen polynomieller Verifizieren. Ein polynomieller Verifizierer ist eine deterministische Turing Maschine, welche mit Hilfe eines Worts c (das Lösungszertifikat) überprüfen kann, ob das Wort w zu einer Sprache L gehört. Es gilt auch, wenn es einen Polynomieller Verifizierer gibt, dass eine nichtdeterministische TM in polynomieller Zeit entscheided.
Bsp: Eine nichtdetermistische Turing Maschine kann in polynomieller ein Sudoku lösen. Wenn man dem Verifiziere nun die Lösung des Sudokus gibt (die fehlenden Zahlen und deren Position), kann dieser in polynomieller Zeit die Lösung überprüfen / nachvollziehen.

10.1.1 Vorgehen

1. Ist das Problem Entscheidbar? Ja → Verifizierer konstruieren, Nein → fertig
2. Was ist das Lösungszertifikat?
3. Wie ist der Verifikationsalgorithmus?
4. Was ist die Laufzeit des Algorithmsu?
5. Schlussfolgerung

10.2 Klasse P

Die Klasse P besteht aus den Sprachen, die mit einem Entscheider mit polyno- mieller Laufzeit entschieden werden können.

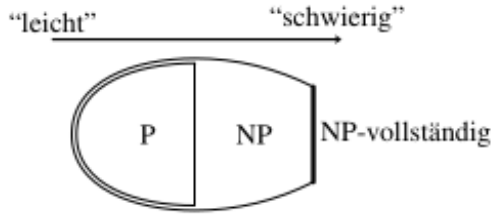
10.3 Klasse NP

Die Klasse der von einer nichtdeterministischen Turingmaschine in polynomi- eller Zeit entscheidbaren Sprachen heisst NP.

10.4 NP-Vollständig

NP-Vollständige Probleme sind die schwierigsten Probleme in NP. NP-Vollständige Probleme sind untereinander alle gleich schwierig zu lösen.

1. $B \in NP$
2. $A \leq_p B$ für alle $A \in NP$



10.5 Polynomielle Reduktion

Die Polynomielle Reduktion ist die Idee, ein gegebenes Problem auf ein bekanntes Problem zu reduzieren. Bsp: Stundenplan- und k-VERTEX-COLORING-Problem. Bei k-VERTEX-COLORING-Problem möchte man einen Graphen, mit k verschiedenen Farben so einfärben, dass kein Vertex die selbe Farbe hat, wie die Nachbarn. Die Reduktion vom Stundenplan auf k-VERTEX-COLORING sieht wie folgt aus:

- S -> k-VERTEX-COLORING
- Fach -> Vertex
- Zeitfenster -> Farbe
- Anzahl Zeitfenster -> k
- Student / Anmeldung -> Kante

Somit gilt: $S \leq_p k - VERTEX - COLORING$

11 Week 11

- SAT: Satz von Cook-Levin
- Weitere Beispiele: 3SAT, k-CLIQUE, HAMPATH, SUBSET-SUM

11.1 Satz von Cook-Levin

SAT ist NP-vollständig.

12 Week 12 empty

- Katalog von Karp
- Minesweeper

13 Week 13

- Kapitel 8: Turing-Vollständigkeit
  - Definition
  - Universelle Turing-Maschine
  - LOOP

Eine Programmiersprache heisst Turing-vollständig, wenn sich jede berechenbare Abbildung in dieser Sprache formulieren lässt. Zu jeder berechenbare Abbildung $f : \sigma^* \to \sigma^*$ gibt es also ein Programm w so, dass c(w) die Funktion f berechnet.

13.1 LOOP

LOOP ist nicht Turing-Vollständig. Folgendes Listening wird x mal ausgeführt. LOOP liest 1x mal den Wert von x und führt die Operation P x-mal aus $\Longrightarrow$ terminiert immer. Es gibt aber Turing-Maschinen die nicht terminieren.

```
LOOP x DO P END
```

13.2 WHILE

Ähnlich wie LOOP. LOOP kann wie folgt durch WHILE gebildet werden:

```
LOOP x DO P END

y := x
WHILE y > 0 DO P; y:= y -1 END
```

13.3 GOTO

GOTO und WHILE sind äquivalent.

14 Katalog von Karp

14.0.1 SAT

Eine logische Formel welche wahr werden muss.

14.0.2 CLIQUE

Eine k-Clique in einem Graphen ist ein vollständiger (vollvermascht) Untergraph mit k Ecken in einem gegebenen Graphen. Im Cliquen-Problem müssen in einem gegebenen Graphen k Ecken gefunden werden, so dass jede mögliche Verbindung zwischen den Ecken auch im Graphen G besteht.

14.0.3 SET-PACKING:

Gegeben eine Familie $(\mathrm{S_i})_{\mathrm{i} \in \mathrm{I}}$ von Mengen und eine Zahl $k \in \mathbb{N}$. Gibt es eine k-elementige Teilfamilie $(S_i)_{i \in J}$ mit $J \subset I$, d.h $|J| = k$, derart, dass die Mengen der Teilfamlie paarweise diskjunkt sind, also

$$S_i \cap S_j = \emptyset \quad \forall i, j \in J \,\mathrm{mit}\, i \neq j$$

14.0.4 VERTEX-COVER:

Gegeben ein Graph G und eine Zahl k, gibt es eine Teilmenge von k Vertizes so, dass jede Kante des Graphen ein Ende in dieser Teilmenge hat?

14.0.5 FEEDBACK-NODE-SET:

Gegeben ein gerichteter Graph G und eine Zahl k, gibt es eine endliche Teilmenge von k Vertizes von G so, dass jeder Zyklus in G einen Vertex in der Teilmenge enthält?

14.0.6 FEEDBACK-ARC-SET:

Gegeben ein gerichteter Graph G und eine Zahl k, gibt es eine Teilmenge von k Kanten so, dass jeder Zyklus in G eine Kante aus der Teilmenge enthält?

14.0.7 HAMPATH

Ein Hamiltonscher Pfad in einem gerichteten Graphen ist ein Pfad, der jeden Vertex des Graphen genau einmal besucht.

14.0.8 UHAMPATH

Wie HAMPATH nur in einem ungerichteten Graphen.

14.0.9 SET-COVERING:

Gegeben eine endliche Familie endlicher Mengen $(\mathrm{S_j})_{1 \leq \mathrm{j} \leq \mathrm{n}}$ und eine Zahl k, gibt es eine Unterfamilie bestehend aus k Mengen, die die gleiche Vereinigung hat?

14.0.10 BIP

BIP ist “binary integer programming”, zu einer ganzzahligen Matrix C und einem ganzzahligen Vektor d ist ein binärer Vektor x zu finden mit Cx = d.

14.0.11 3SAT

Wie SAT nur in dreier Gruppen.

14.0.12 VERTEX-COLORING

Man sagt, die Vertizes eines Graphen G können mit k Farben eingefärbt werden, wenn sich für jeden Vertex eine der k Farben wählen lässt, so dass nie zwei durch eine Kante verbundene Vertizes die gleiche Farbe bekommen.

14.0.13 CLIQUE-COVER:

Gegeben ein Graph G und eine positive Zahl k, gibt es k Cliquen so, dass jede Ecke in genau einer der Cliquen ist?

14.0.14 EXACT-COVER:

Gegeben eine Familie $(\mathrm{S_j})_{1 \leq \mathrm{j} \leq \mathrm{n}}$ von Teilmengen einer Menge U, gibt es eine Unterfamilie von Mengen, die disjunkt sind, aber die gleiche Vereinigung haben? Die Unterfamilie $(\mathrm{S_{ji}})_{1 \leq \mathrm{i} \leq \mathrm{m}}$ muss also $\mathrm{S_{ji}} \cap \mathrm{S_{jk}} = \emptyset$ und

$$\bigcup_{j=1}^{n} S_j = \bigcup_{i=1}^{m} S_{ji}$$

erfüllen.

14.0.15 3D-MATCHING:

Sei T eine endliche Menge und U eine Menge von Tripeln aus $T : U \subset T \times T \times T$. Gibt es eine Teilmenge $W \subset U$ so, dass $|\mathrm{W}| = |\mathrm{T}|$ und keine zwei Elemente von W stimmen in irgendeiner Koordinate überein?

14.0.16 HITTING-SET:

Gegeben eine Menge von Teilmengen $S_i \subset S$, gibt es eine Menge H, die jede Menge in genau einem Punkt trifft, also $|H \cap S_i| = 1 \, \forall i$?

14.0.17 STEINER-TREE:

Gegeben ein Graph G, eine Teilmenge R von Vertizes, und eine Gewichtsfunktion $w : E \to Z$ und eine postive Zahl k, gibt es einen Baum mit Gewicht $\leq$ k, dessen Knoten in R enthalten sind? Das Gewicht des Baumes ist die Summe der Gewichte $w(u, v)$ über alle Kanten u, v im Baum.

14.0.18 SEQUENCING:

Gegeben sei ein Vektor $(t_1, \ldots, t_p) \in \mathbb{Z}^p$ von Laufzeiten von p Jobs, ein Vektor von spätesten Ausführungszeiten $(d_1, \ldots, d_p) \in \mathbb{Z}^p$, einem Strafenvektor $(s_1, \ldots, s_p) \in \mathbb{Z}^p$ und eine positive ganze Zahl k. Gibt es eine Permutation π der Zahlen $1, \ldots, p$, so dass die Gesamtstrafe für verspätete Ausführung bei der Ausführung der Jobs in der Reihenfolge $\pi(1), \ldots, \pi(p)$ nicht grösser ist als k? Formal lautet die Bedingung

$$\sum_{j=1}^{p} \theta(t_{\pi(1)} + \cdots + t_{\pi(j)} - d_{\pi(j)}) s_{\pi}(j) \leq k$$

darin ist die Stufenfunktion definiert durch

$$\theta(x) = 1 \quad x \geq 00 \quad x < 0$$

14.0.19 PARTITION:

Gegeben eine Folge von s ganzen Zahlen $c_1, c_2, \ldots, c_s$, kann man die Indizes $1, 2, \ldots, s$ in zwei Teilmengen I und $\overline{I}$ teilen, so dass die Summe der zugehörigen $\mathrm{c_i}$ identisch ist:

$$\sum_{i \in I} c_i = \sum_{i \notin I} c_i$$

14.0.20 MAX-CUT:

Gegeben ein Graph G mit einer Gewichtsfunktion $w : E \to Z$ und eine ganze Zahl W. Gibt es eine Teilmenge S der Vertizes, so dass das Gesamtgewicht der Kanten, die S mit seinem Komplement verbinden, mindestens so gross ist wie W:

$$\sum_{\{u,v\} \in E \wedge u \in S \wedge v \notin S} w(\{u, v\}) \geq W$$

14.0.21 SUBSET-SUM

Das Problem SUBSET-SUM ist auch bekannt als das Rucksack-Problem. Gegeben ist eine Menge S von ganzen Zahlen, kann man darin eine Teilmenge finden, die als Summe einen bestimmten Wert t hat? Als Sprache formuliert:

$$\mathit{SUBSET\text{-}SUM} = \left\{ \langle S, t \rangle \;\middle|\; \begin{array}{l} S \text{ eine Liste von ganzen Zahlen, in der es eine} \\ \text{Teilliste } T \subset S \text{ gibt mit} \\ \\ \displaystyle\sum_{x \in T} x = t. \end{array} \right\}$$