Merkblatt Informatik-Leistungskurs

Zusammenfassung der Inhalte im Leistungskurs Informatik bezogen auf die Vorgaben für das Abitur in Nordrhein-Westfalen für das Jahr 2012[1]

Erstellt von

Patrick Robrecht

https://patrick-robrecht.de/

Inhaltsverzeichnis

1 Automaten **2**
- 1.1 Definition eines Automaten . . . 2
- 1.2 Klassifikation von Automaten . . . 2
- 1.3 Grammatiken und Sprachen . . . 3
- 1.4 Komplexitätsklassen . . . 5

2 Datenbanken **5**
- 2.1 Realisierung von Datenbank-Projekten . . . 5
- 2.2 Begriffe zur Modellierung mit dem ERM . . . 6
- 2.3 Übersetzung eines ER-Modells in ein Relationenschema . . . 6
- 2.4 Normalisierung von Datenbanken . . . 7
- 2.5 SQL-Befehle . . . 7
  - 2.5.1 Datenauswertung . . . 7
  - 2.5.2 Datenmanipulation (DML) . . . 8
  - 2.5.3 Datendefinition (DDL) . . . 9
  - 2.5.4 Datenzugriffskontrolle (DCL) . . . 9

3 Datenstrukturen **10**
- 3.1 Baumstrukturen . . . 10
- 3.2 Lineare Datenstrukturen . . . 10
- 3.3 Sortieralgorithmen . . . 10

4 Graphentheorie **11**
- 4.1 Grundbegriffe der Graphentheorie . . . 11
- 4.2 Graphalgorithmen . . . 11

[1] Keine Gewähr für Vollständigkeit, insbesondere nicht für spätere Jahrgänge

1 Automaten

Die **Automatentheorie** ist ein Teilgebiet der **Theoretischen Informatik**. Automaten sind einfache Modelle für informationsverarbeitende Maschinen.

1.1 Definition eines Automaten

Ein Automat wird über folgende Eigenschaften definiert:

1. ein Eingabealphabet E (eine Menge von Symbolen, auch mit Σ bezeichnet);
2. ggf. ein Ausgabealphabet A (eine Menge von Symbolen, auch mit Γ bezeichnet);
3. eine Zustandsmenge $Z = \{Z_0, , Z_n\}$ (auch mit Q bezeichnet);
4. einen Startzustand $Z_0 \in Z$ (auch Q_0);
5. eine Endzustandsmenge $Z_E \subseteq Z$ (Teilmenge, aber nicht unbedingt echte Teilmenge);
6. eine Zustandsüberführungsfunktion und ggf. eine Ausgabefunktion (siehe Klassifikation)

Zustandsübergänge werden durch einen **Zustandsübergangsgraphen** oder eine **Automatentafel** dargestellt.
Eine Automatentafel ist eine Tabelle mit dem Zeichen des Eingabealphabets horizontal und den Zuständen vertikal. In den Tabellenzellen wird dann jeweils der Folgezustand angegeben für den Fall, dass in einem Zustand ein bestimmtes Eingabezeichen eingegeben wird. Ggf. kann außerdem noch die Ausgabe beim Erreichen des neuen Zustands angegeben werden.
Außerdem kann man einen Automaten graphisch darstellen: Zustände als Kreise, Endzustände durch Doppelkreise gekennzeichnet, Übergänge durch Pfeile beschriftet mit Übergang.

1.2 Klassifikation von Automaten

Ein Akzeptor (erkennender Automat) definiert eine formale Sprache, in dem eine Folge von Eingaben (ein Eingabewort) nur als gültig akzeptiert wird, wenn ein gültiger Endzustand erreicht wird. Der Akzeptor gibt somit einen boolschen Wert zurück (zulässig oder abgelehnt).

Das Alphabet X der formalen Sprache beinhaltet alle Zeichen der Sprache. Ein Wort w ist eine endliche Menge von Zeichen aus X. Die Menge aller möglichen Wörter heißt $X*$ und beinhaltet auch das leere Wort (dargestellt als ϵ).

Zustandsübergänge: Zustand $\times$ Zeichen $\rightarrow$ Folgezustand

Ein Transduktor ist ein Automat mit Ausgaben, die jeweils bei einem Zustandsübergang erfolgen.

Ein endlicher (finiter) Automat hat eine endliche Menge an definierten Zuständen.

Ein deterministischer Automat befindet sich immer in genau einem gespeicherten Zustand.

Zustandsübergänge: eine Funktion im mathematischen Sinne (alle Zustandübergänge sind eindeutig)

Ein nicht-deterministischer-Automat kann sich gleichzeitig in mehreren Zuständen befinden.

Zustandübergänge: keine Funktion im mathematischen Sinne, da nicht alle Übergänge eindeutig sind

Ein deterministischer endlicher Automat (DEA/DFA)

Eine **Moore-Maschine** ist ein DEA, der für jeden erreichten Zustand eine Ausgabe erzeugt.

Eine **Mealy-Maschine** ist ein DEA, der mit jeder Eingabe eine Ausgabe erzeugen kann.

Ein nicht-deterministischer endlicher Automat (NEA/NFA) kann in einen DEA umgewandelt werden.

Ein Kellerautomat hat zusätzlich noch einen Stack mit einem Startzeichen (Kellervorbelegungszeichen). Die Übergangsfunktion beinhaltet den jetzigen Zustand und das gelesene Zeichen (Eingabezeichen) als Eingaben; die Ausgaben sind der Folgestand, das geschriebene Zeichen (Kellerzeichen) und die Richtung (Kelleroperation). Für die Kelleroperation gibt es folgende Möglichkeiten:

- pop entfernt das oberste Kellerelement
- nop legt das oberste Kellerelement wieder auf den Keller
- push ergänzt ein Elemtent zum Keller und behält das vorherige bei

Zustandsübergänge: Zustand × Eingabezeichen × gelesenes Kellerzeichen → Kelleroperation und Folgezustand

Eine Turing-Maschine ist das vereinfachte Modell eines Computers und besitzt nicht einen Stack wie der Kellerautomat, sondern ein Band, auf dem sich der Zeiger in beide Richtungen bewegen kann.

Zustandsübergänge: Zustand × Eingabezeichen → geschriebenes Zeichen × Bewegungsrichtung × Folgezustand

1.3 Grammatiken und Sprachen

Eine Grammatik $G = \{T, N, S, P\}$ wird definiert durch

1. eine Menge von Terminalsymbolen T
2. eine Menge von Nichtterminalsymbolen N
3. ein Startzeichen S
4. Produktionsregeln P, welche mögliche Ersetzungen angeben

Sprache	Produktionsregeln der zugehörigen Grammatik	zugehöriger Automat, Beispiel
Typ 0, RE: rekursiv-aufzählbar	unbeschränkt: beliebige Folge von Terminal- und Nichtterminalsymbolen $\rightarrow$ beliebige Folge von Terminal- und Nichtterminalsymbolen	Turing-Maschine
Typ 1, CS: kontext-sensitiv	y + Nichtterminalsymbol + $z \rightarrow y$ + beliebige Folge von Terminal- und Nichtterminalsymbolen (außer ϵ) + z (y und z sind jeweils beliebige Folgen von Terminal- und Nichtterminalsymbolen)	linear gebundene, nichtdeterministische Turingmaschine (LBA) Sprache $L = a^n b^n c^n$ für $n \in \mathbb{N}$
Typ 2, CF: kontext-frei	Nichtterminalsymbol $\rightarrow$ beliebige Folge von Terminal- und Nichtterminalsymbolen	Kellerautomat Sprache $L = a^n b^n$ für $n \in \mathbb{N}$ mit der Grammatik $G = \{N, T, S, P\}$, $N = \{S\}$, $T = \{a, b\}$, $P = \{S \rightarrow \epsilon \mid aSb\}$ Palindrome (Wörter, die vor- und rückwärts gelesen dasselbe ergeben)
Typ 3, REG: regulär	$\rightarrow$ genau ein Terminalsymbol und maximal ein Nichtterminalsymbol (hier die Reihenfolge bei allen Regeln gleich: immer vor bzw. nach, entspricht links- bzw. rechtsregulär)	endlicher Automat Sprache $L = a^* b^*$ (Anzahlen egal)

Tabelle 1: Chomsky-Hierarchie

Das **Pumping-Lemma** dient dazu, zu beweisen, dass eine Sprache nicht regulär ist. Um zu beweisen, dass eine Sprache regulär ist, muss ein endlicher Automat konstruiert werden, der die Sprache akzeptiert.

Eine Funktion ist dann **berechenbar**, wenn eine Turingmaschine konstruierbar ist, die der Funktion entspricht.
Es gibt keine Turing-Maschine, die entscheiden kann, ob eine beliebige Turing-Maschine nach endlich vielen Schritten anhält oder nicht (unlösbares **Halteproblem**).

1.4 Komplexitätsklassen

Klasse P alle mit polynomialen Aufwand von einer deterministischen Turingmaschine lösbaren Probleme.

Beispiele: rekursive Sortieralgorithmen: $O(n \log n)$, Suchen: $O(n)$, Gauss-Verfahren $O(n^3)$

Klasse NP alle mit polynomialen Aufwand von einer nichtdeterministischen Turingmaschine lösbaren Probleme

Klasse EXP alle mit exponentiellen Aufwand lösbaren Probleme. Solche Probleme sind nicht sinnvoll lösbar.

NP-vollständige Probleme sind Probleme der Klasse NP, die nicht in der Klasse P liegen. Der Beweis, ob P = NP gilt, ist als Milleniums-Problem mit einer Million Dollar notiert.

2 Datenbanken

2.1 Realisierung von Datenbank-Projekten

1. Die **Anforderungsdefinition** (Spezifikation) ist eine Beschreibung der Wünsche des Kunden bzw. des Auftraggebers. Sie legt fest, welche Prozesse abgebildet werden sollen, welche Funktionen die Benutzerschnittstelle beinhaltet und ob es Schnittstellen zu anderen Systemen geben soll (und ggf. wie diese aussehen sollen).

2. Das **konzeptionelle Modell** schafft ein semantisches Abbild der Realität. Hier werden die Daten, welche in der Datenbank gespeichert werden sollen, modelliert. Das Ergebnis des konzeptionellen Entwurfs wird in einem ER-Diagramm graphisch dargestellt.

3. Das **logische Modell** ist die normalisierte Form des Entwurfs in Tabellenform (Relationenmodell).

4. Das **physische Modell** umfasst die Implementierung, d. h. die Realisierung als Datenbank und die Schaffung einer Benutzerschnittstelle.

Erst im letzten Schritt kommt ein DBMS[2] ins Spiel, die ersten drei Schritte sind komplett DBMS-unabhängig.

[2]Data Base Management System, englisch für: Datenbank-Management-System. Beispiele: MySQL, Postgres, SQL Server

2.2 Begriffe zur Modellierung mit dem ERM

Entitäten (*entities*) sind Objekte aus der realen Welt (wie Personen oder bestimmte Dinge).

Entitätstypen werden im ER-Diagramm durch Rechtecke dargestellt.

Attribute beschreiben eine Entität (oder auch eine Beziehung) genauer und werden im ER-Diagramm als Ellipsen dargestellt. Zu jedem Attribut gehört ein Wertebereich, der die möglichen Werte dieses Attributs beschreibt.

Schlüssel ist ein Attribut, das einen Eintrag in der Datenbank eindeutig identifiziert. Der/die Schlüssel wird/werden in der Darstellung unterstrichen.

Beziehungstypen (*relationships*) verbinden zwei oder mehr Entitätstypen miteinander und werden im ER-Diagramm als Rauten dargestellt.

Funktionalitäten/Kardinalität beschreiben die Art der Beziehung genauer, indem sie angeben, mit wie vielen anderen Entitäten eine Entität eine Beziehung eingeht. Mögliche Funktionalitäten sind 1:1, 1:N, N:1, N:M.

Optionalität beschreibt, ob eine Verbindung unbedingt erforderlich ist (*Muss-Verbindung*) oder ob sie optional ist (*Kann-Verbindung*).

Rollen beschreiben, in welcher Weise Entitäten an einer Beziehung teilnehmen. Sie können zum Beispiel zur Beschreibung rekursiver Strukturen benutzt werden.

2.3 Übersetzung eines ER-Modells in ein Relationenschema

1. Übersetzung der Entitäten
   - Jede Entität wird eine Relation.
   - Der Schlüssel der Entität wird **Primärschlüssel** der Relation.
   - Weitere Attribute der Entität werden zu Attributen der Relation.
2. Übersetzung der Beziehungen
   - Jede Beziehung wird eine Relation.
   - Attribute dieser Relation sind die Schlüssel der teilnehmenden Relationen (bei N:M Beziehungen beide). Bei 1:N und N:1 Beziehungen wird die N-Seite der Schüssel. Bei 1:1 Beziehungen wird einer der beiden Schlüssel gewählt.
   - Attribute der Beziehung selbst werden ebenfalls zu Attributen der neuen Relation.
3. Vereinfachung des Schemas

- Wenn es mehrere Relationen mit demselben Schlüssel gibt, werden diese zu einer Relation zusammengefasst.

Fremdschlüssel sind die Schlüssel in Relationen, in denen diese keine Primärschlüssel sind, sondern nur auf eine andere Relation verweisen.

2.4 Normalisierung von Datenbanken

Normalformen helfen, Anomalien und Redundanzen zu vermeiden. Eine Relation ist jeweils in der 1., 2. oder 3. Normalform, wenn folgende Bedingungen erfüllt sind.

1. **Normalform** Alle Attribute weisen nur einfache Attributwerte auf (sind atomar), d. h. Werte dürfen keine Mengen oder Aufzählungen enthalten.

2. **Normalform** In einer Relation der 1. Normalform wird kein Nicht-Schlüssel-Attribut von einem Teil des Schlüssels identifiziert (keine funktionale Abhängigkeiten).

3. **Normalform** In einer Relation der 2. Normalform folgt aus keinem Nicht-Schlüssel-Attribut ein anderes Nicht-Schlüssel-Attribut (keine transitive Abhängigkeiten).

2.5 SQL-Befehle

Im Folgenden ist nur eine begrenzte Auswahl an MySQL-Befehlen aufgelistet. Alle SQL-Befehle sind hier vollständig in Großbuchstaben geschrieben.

2.5.1 Datenauswertung

Bei der **Selektion** fragt man nach Datensätzen und bei der **Projektion** nach Spalten.

```
SELECT [DISTINCT] * [,rechenausdruck [AS ergebnis]]
FROM tabellen_name [AS tabellen_alias][, tabellen_name2, ...]
[WHERE attribut operator wert(eliste)]
[GROUP BY attr [HAVING attribut operator wert(eliste)]]
[ORDER BY attribut|ergebnis|Spaltenposition [DESC|ASC]]
[LIMIT a[,b]]
```

`DISTINCT` sorgt dafür, dass evtl. vorhandene Dopplungen aussortiert werden.
Statt `*` können auch nur einzelne Attribute (Spalten) abgefragt werden.
Ein Rechenausdruck kann neben den Grundrechenarten folgende Funktionen enthalten:

- `SUM(attr)` für die Summe, `COUNT(attr)` für die Anzahl der von `NULL` verschiedenen Einträge

- `MIN(attr)` für den Minimalwert, `MAX(attr)` für den Maximalwert, `AVG(attr)` für den Durchschnittswert eines Attributs

Der Teil `operator value` in Bedingungen kann wie gefolgt aussehen:

- Vergleichsoperator `<`, `<=`, `=`, `>`, `>=` oder `<>` mit einem Wert (nur Strings werden dabei in Anführungszeichen gesetzt)
- `BETWEEN wert1 AND wert2` (Bereichsauswahl)
- `LIKE '%beispiel_'`, wobei in der angegebenen Zeichenkette `%` für eine beliebige Zeichenfolge und `_` für ein beliebiges Zeichen steht.
- `[NOT] IN ('wert1','wert2'[, ...])`
- `IS [NOT] wert`

Mehrere Bedingungen können mit `AND` und `OR` verbunden werden.
Mit `GROUP BY` können die Ergebnis-Datensätze nach bestimmten Attributen gruppiert werden.
Mit `ORDER BY` können die Ergebnis-Datensätze nach Attributen sortiert werden, dabei bedeutet DESC eine absteigende Reihenfolge und ASC eine aufsteigende (dabei können mehrere Attribute angegeben werden, die dann nacheinander für die Sortierung genutzt werden).
Mit `LIMIT a` wird die Ausgabe auf die ersten `a` Datensätze beschränkt, bei `LIMIT a,b` auf `b` Datensätze ab dem `a+1`. Datensätze.
`SELECT`-Befehle können verschachtelt werden, indem bei einer Bedinung als Wert(eliste) in Klammern (...) eine weitere `SELECT` steht.
Tabellen können mit `INNER JOIN` verknüpft werden:

```
SELECT * FROM tabellen_name
INNER JOIN tabellen_name2
ON tabellen_name.attr = tabellen_name2.attr
```

Beim `OUTER JOIN` werden alle Datensätze ausgewählt, auch wenn die Bedingung nicht stimmt.
Mit `LEFT|RIGHT OUTER JOIN` werden die Datensätze ausgewählt, zu denen es keinen Partner in der anderen Tabelle gibt.

2.5.2 Datenmanipulation (DML)

Neue Datensätze werden mit dem `INSERT`-Befehl eingefügt; nicht bekannte Werte können auf `NULL` gesetzt werden, sofern das Datenbankschema für das entsprechende Attribut `NULL` zulässt.

```
INSERT INTO tabellen_name (attr1, attr2, attr3)
VALUES ('wert1','wert2','wert3')
```

Werte können mit dem `UPDATE`-Befehl aktualisiert werden.

```
UPDATE tabellen_name
SET attr1 = 'wert1', attr2 = 'wert2'
WHERE Bedingung
```

Werte können auch relativ zum alten Wert überschrieben werden:

```
UPDATE tabellen_name
SET attr = attr * 2
```

Datensätze werden mit `DELETE` gelöscht:

```
DELETE FROM tabellen_name
WHERE Bedingung
```

2.5.3 Datendefinition (DDL)

Eine neue Datenbank wird mit `CREATE DATBASE` erstellt.

```
CREATE DATABASE datenbank_name
```

Eine Datenbank wird mit `DROP DATABASE` gelöscht.

```
DROP DATABASE datenbank_name
```

Eine neue Tabelle wird mit `CREATE TABLE` erstellt.

```
CREATE TABLE tabellen_name (
attr1 data_type,
[attr2 data_type,
...]
)
```

Eine Tabelle wird mit `DROP TABLE` gelöscht.

```
DROP TABLE tabellen_name
```

Die Daten in einer Tabelle werden mit `TRUNCATE TABLE` gelöscht.

```
TRUNCATE TABLE tabellen_name
```

2.5.4 Datenzugriffskontrolle (DCL)

Die DCL spielt für uns erst einmal keine Rolle. Sie umfasst Befehle wie `GRANT`, `REVOKE`, `LOCK`, mit denen die Zugriffsrechte einzelner Benutzer eines DBMS festgelegt werden können.

3 Datenstrukturen

3.1 Baumstrukturen

Der **Binärbaum** ist eine Graphenstruktur, bei der jeder Knoten einen linken und einen rechten Kindknoten besitzt. Diese können wiederum selbst Binärbäume sein.
Mögliche **Traversierungen** einer Baumstruktur sind (Reihenfolge der Angabe der linken und rechten Kinder sowie der Wurzel):

In-Order Links-Wurzel-Rechts

Pre-Oreder Wurzel-Links-Rechts

Post-Order Links-Rechts-Wurzel

Level-Order einzelne Ebenen werden nacheinander ausgelesen

Der **binärere Suchbaum** ist ein Binärbaum, dessen Einträge in einer sinnvollen Reihenfolge sortiert sind. In optimierter Form reduziert er den Aufwand beim Suchen auf ein Minimum.

3.2 Lineare Datenstrukturen

Die **Queue** (Schlange) ist eine dynamische Datenstruktur nach dem FiFo-Prinzip (*First in, first out*), d.h. das zuerst eingefügte Element wird auch als erstes wieder entnommen.
Der **Stack** (Stapel oder Keller) ist eine dynamische Datenstruktur nach dem LiFo-Prinzip (*Last-In-First-Out*), d.h. das zuletzt eingefügte Objekt wird als erstes wieder entnommen.
Die Liste ist eine dynamische Datenstruktur zum Speichern von Objekten. Bei der **einfach verketteten Liste** besitzt jedes Listenelement einen Verweis auf seinen Nachfolger. Bei der **doppelt verketteten Liste** besitzt jedes Listenelment einen Verweis auf Vorgänger und Nachfolger. Dies bewirkt ein effizienteres Suchen, erhöht allerdings ein wenig auch den Speicherbedarf.
Generische Datentypen, in Java auch Generics genannt, fügen einen Typparameter hinzu. So können in eine `Liste<T>` nur Elemente des Typs `T` eingefügt werden.

3.3 Sortieralgorithmen

Bei **Bubblesort** wird das Array jeweils von links nach rechts durchlaufen, ggf. werden dabei nebeneinanderliegende Einträge vertauscht. **Insertionsort** bringt nacheinander alle Elemente an die richtige Stelle. **Selectionsort** sucht nacheinander jeweils den kleinsten Wert in einem Array und baut so das sortierte Array auf. Der Aufwand dieser einfachen Sortierverfahren Bubblesort, Insertionsort und Selectionsort liegt bei $O(n^2)$.
Mergesort halbiert die zu sortierende Liste solange, bis einzelne Elemente übrig bleiben. So wird schrittweise sortiert. Bei **Quicksort** erfolgt die Halbierung beim sog. Pivotelement. Der Aufwand rekursiver Sortierverfahren wie Mergesort und Quicksort liegt bei $O(n \log n)$.

4 Graphentheorie

Ein Graph ist eine Datenstruktur aus der Graphentheorie, die aus einer Menge von Punkten, zwischen denen Linien (Verbindungen) verlaufen, besteht.

4.1 Grundbegriffe der Graphentheorie

Graph Datenstruktur mit einer Menge von Knoten, die über Kanten verbunden sind

Knoten (*node*) ein Punkt im Graphen

Kante (*edge*) Verbindung zwischen zwei Knoten

gerichtete Kante (*directed edge*) Verbindung zwischen zwei Knoten, die nur in eine Richtung besteht

ungerichtete Kante Verbindung zwischen zwei Knoten, die in beide Richtungen besteht

gerichteter Graph (*directed graph*) Graph mit mindestens einer gerichteten Kante

ungerichteter Graph Graph ohne gerichtete Kanten, d.h. nur ungerichtete Kanten

gewichtete Kante Kante, die ein Gewicht zugewiesen bekommen hat

gewichteter Graph Graph mit mindestens einer gewichteten Kante

Richtung/Orientierung eine Kante kann in eine Richtung oder in beide Richtung weisen.

Gewicht (*weight*) Bewertung einer Kante

Schlinge Kante, die einen Knoten mit sich selbst verbindet

zyklischer Graph Graph, in dem Knoten in Kreisstruktur miteinander verbunden sind

Eulerweg Weg über Kanten durch den Graphen von einem Anfangsknoten bis zu einem Endknoten, bei dem alle Knoten genau einmal berührt werden.

Multigraph Graph, in dem ein Knotenpaar durch mehrere Kanten verbunden werden kann

Adjazenzmatrix Repräsentation eines Nicht-Multigraphen

4.2 Graphalgorithmen

Ein **minimaler Spannbaum** (*Minimum Spanning Tree* [MST]) ist ein Baum mit allen Knoten des Graphen, der die minimale Verbindung zwischen den Knoten angibt.

Djikstra berechnet die kürzeste Entfernung zwischen zwei Knoten.

1. Weise allen Knoten die beiden Eigenschaften „Distanz“ und „Vorgänger“ zu. Initialisiere die Distanz im Startknoten mit 0 und in allen anderen Knoten mit ∞.
2. Solange es noch unbesuchte Knoten gibt, wähle darunter denjenigen mit minimaler Distanz aus und
   (a) Markiere den Knoten.
   (b) Berechne für alle noch unbesuchten Nachbarknoten die Summe des jeweiligen Kantengewichtes und der Distanz im aktuellen Knoten.
   (c) Ist dieser Wert für einen Knoten kleiner als die dort gespeicherte Distanz, aktualisiere sie und setze den aktuellen Knoten als Vorgänger.

Kruskal berechnet den minimalen Spannbaum eines Graphen.

1. Führe so oft wie möglich aus: Wähle unter den noch nicht ausgewählten Kanten des Graphen die kürzeste Kante, die mit den schon gewählten Kanten keinen Kreis bildet.

Prim berechnet den minimalen Spannbaum eines Graphen.

1. Wähle einen beliebigen Knoten als Startgraph T (dieser wird zum minimalen Spannbaum erweitert).
2. Solange T noch nicht alle Knoten enthält:
   (a) Wähle eine Kante e minimalen Gewichts aus, die einen noch nicht in T enthaltenen Knoten v mit T verbindet.
   (b) Füge e und v dem Graphen T hinzu.

Tiefensuche besucht rekursiv alle Knoten, die über Kanten erreichbar sind (kann zum Beispiel die Anzahl der Knoten zählen).

1. Besuche ersten vom Anfangsknoten aus erreichbaren Knoten und markiere diesen.
2. Besuche den ersten von diesem Knoten aus erreichbaren und markiere diesen, und so weiter.
3. Wenn kein Knoten mehr erreichbar ist, kehre zum letzten Punkt zurück, an dem es noch weitere Kanten gab.

Breitensuche durchläuft alle Knoten (kann zum Beispiel den kürzesten Weg zwischen zwei Knoten berechnen).

1. Bestimme einen Startknoten und speichere ihn in einer Warteschlange ab.
2. Entnimm einen Knoten vom Beginn der Warteschlange und markiere ihn.
3. Falls das gesuchte Element gefunden wurde, brich die Suche ab und liefere „gefunden“ zurück.

4. Anderenfalls hänge alle bisher unmarkierten Nachfolger dieses Knotens, die sich noch nicht in der Warteschlange befinden, ans Ende der Warteschlange an.
5. Wenn die Warteschlange leer ist, dann wurde jeder Knoten bereits untersucht. Beende die Suche und liefere „nicht gefunden“ zurück.
6. Wiederhole Schritt 2.

Zyklensuche durchsucht den Graphen nach einem Zyklus, d. h. überprüft ob ein kreisförmiger Weg durch den Graphen möglich ist. Eine Implementierung gibt entweder als boolschen Wert zurück, ob ein Zyklus gefunden wurde, oder eine Liste der im Zyklus enthaltenen Knoten.

Die Algorithmen von Djikstra, Prim und Kruskal sind Beispiele für **Greedy-Algorithmen**, d. h. jeder Schritt führt weiter zur Lösung des Problems. Es gibt somit kein Backtracking. Der Aufwand ist fast linear.