Einführung in (wissenschaftliches) Schreiben

Jan Finis
finis@in.tum.de

Technische Universität München
Institute for Informatics - Chair III (I3)

April 17, 2013

Outline

1. Wissenschaftliches Schreiben
2. Inhalt
3. Sprache
4. Layout
5. LaTeX
6. Grafiken, Tabellen, Listings
7. Bibliographie und Zitieren

Outline

1 Wissenschaftliches Schreiben
2 Inhalt
3 Sprache
4 Layout
5 LaTeX
6 Grafiken, Tabellen, Listings
7 Bibliographie und Zitieren

Wissenschaftliches Schreiben

- Ziel: Kommunikation von Forschungsergebnissen
- Sachlich
- Vollständig, Präzise
  - Entweder verständlich und vollständig erklären oder weglassen!
  - Oft erlebter Horror: Komplexer Sachverhalt wird mit zwei Sätzen abgehandelt, “damit er noch erwähnt wurde”.
- Niemals: Schreiben um Seiten voll zu bekommen!
  $\Rightarrow$ Immer: Schreiben was es zu schreiben gibt, nicht mehr, nicht weniger!

“Self-containedness”

- Eine wissenschaftliche Arbeit muss vollständig self-contained sein.
  - D.h. vollständig verständlich ohne andere Literatur zu lesen
  - Allein auf Literatur zu verweisen reicht nicht!
  - Literaturverweis reicht nur, wenn Arbeit auch ohne das Lesen der Literatur verständlich ist
- Normaler Informatiker der in eurem Thema NICHT drin ist muss es verstehen
  - $\Rightarrow$ Komilitonen probelesen lassen!

“Self-containedness” und Grundwissen

- Grundwissen darf vorausgesetzt werden
  - Seminar, BA, MA: Informatikgrundwissen
  - Paper, Dissertation: Tieferes Grundwissen des Fachbereichs (z.B. abelsche Gruppe $\Leftrightarrow$ Gruppentheorie)
- Vorsicht: Nicht alles was man im Grundstudium lernt ist Grundwissen
  - Beispiel: Diskrete Strukturen $\Leftrightarrow$ chinesischer Restsatz

Vorwärtsreferenzen

- Eine wissenschaftliche Arbeit muss *von vorne nach hinten* lesbar sein.
- $\Rightarrow$ Vorwärtsreferenzen vermeiden!
  - Vorwärtsreferenz nur als Hinweis erlaubt,
    d.h. Text vollständig verständlich ohne der Referenz zu folgen
  - Beispiel (okay):

    We use XPath for referencing the nodes. The exact syntax of XPath expressions is not important here; it will be dicussed in Chapter 7.

  - Beispiel (nicht okay):

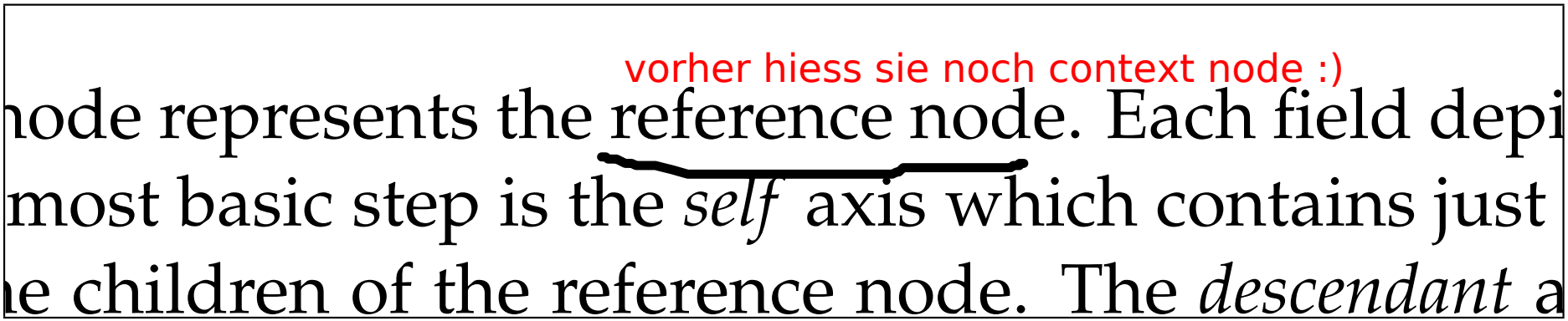
    The XPath expression `/a/b/c` *contains three axis steps using the child axis. Chapter 7 will explain XPath. The child axis is important for our considerations[. . .]*

Verwendung von Begriffen

- Fachterminologie verwenden
- Nicht-triviale Begriffe müssen vor Verwendung eingeführt werden.
  - Beispiel trivial: Depth-first search, kommutativität, amortisierte Laufzeit, NP-vollständig
  - Beispiel nicht-trivial: Levenshtein Distance, Kolmogorov-Axiome, abelsche Gruppe
  - Vom Fachbereich abhängig!
- Definition vor/bei erster Verwendung (!!!)
  - Keine forward Referenzen!
- Möglichkeiten der Definition
  - Mathematisch, mit `definition` environment
  - Im Fließtext

Verwendung von Begriffen

Ein Begriff für einen Sachverhalt! Nicht switchen!



Definition Environment

- Definition Environment wird bei wichtigen Definitionen verwendet
  - z.B. Formale Definitionen
  - Erlaubt Referenzierung mit `\ref`
  - Enthält nur eindeutige Definition
  - Keine Beispiele oder Erklärungen!

Definition (Recursive Kripke Structures)

A *Recursive Kripke Structure* $\mathcal{R}$ over a set of atomic propositions *AP* is a tuple $(\mathbb{S}, S^{\text{in}})$, where $\mathbb{S}$ is a set of sub-structures and $S^{\text{in}} \in \mathbb{S}$ is the *initial sub-structure*.

```
\begin{definition}[Recursive Kripke Structures]
A \emph{Recursive Kripke Structure}
\label{def:rks}
$\mathcal{R}$ over a set of atomic
propositions $AP$ is a tuple
$(\mathbb{S},S^{\mathrm{in}})$, where
$\mathbb{S}$ is a set of sub-structures
and $S^{\mathrm{in}} \in \mathbb{S}$
is the \emph{initial sub-structure}.
\end{definition}
```

In-Text Definition

- Bei weniger wichtigen oder weniger formellen Begriffen Definition im Fließtext
- Begriff wird bei Definition oder erster Verwendung (falls trivial) kursiv dargestellt (`\emph`)
- Definition muss nicht das Wort “definieren” benutzen.

Our approach requires the checking of all nodes in the client tree, so the tree must be traversed thoroughly. A *breath first traversal* is a tree traversal that starts at the tree root and visits all nodes of one level before descending into the next level. We use the breath first traversal for implementing our approach efficiently. Another approach would be to traverse the tree in *post-order*, that is, visiting all children of a node before the node itself is visited. Note that post-order is not the opposite of the well-known *pre-order*.

Outline

1 Wissenschaftliches Schreiben

2 Inhalt

3 Sprache

4 Layout

5 LaTeX

6 Grafiken, Tabellen, Listings

7 Bibliographie und Zitieren

Inhalt - Beispiel

- Introduction
- Related Work
- Preliminaries
- . . . Hauptarbeitsteil (u.U. mehrere Kapitel)
- Evaluation
- Conclusion
- Appendix, Bibliography, List of Figures, . . .

Inhalt - Introduction

- Funktion und Inhalt:
  - Führt ins Thema ein
  - Begründet, warum das Thema wichtig ist (Motivation)
  - Stellt heraus, was der Grundgedanke und der Inhalt der Arbeit ist (Contribution)
  - Stellt heraus, welche Aspekte NICHT Teil der Arbeit sind (Scope)
  - Gibt u.U. einen kurzen Überblick über die folgenden Kapitel (Outline)
- Gute Einleitung sehr wichtig, da die meisten nur Einleitung, Related Work und Conclusion lesen ;)
- Tolle Arbeit nützt nichts, wenn man den Leser nicht überzeugen kann, dass der Inhalt relevant ist!
- Muss immer vorhanden sein

Inhalt - Related Work

- Funktion und Inhalt:
  - Gibt einen Überblick über verwandte Arbeiten im Gebiet
  - Strukturiert und Gruppiert diese Arbeiten sinnvoll
  - Deckt möglichst alle relevanten Arbeiten ab
  - Erklärt kurz deren Inhalt und was sie von anderen Arbeiten (vor allem der Eigenen!) abheben
  - Positioniert die eigene Arbeit im Gebiet
- Muss kein eigenes Kapitel sein; kann auch in Einleitung oder nach dem Hauptteil sein
- Literaturrecherche ist ein wichtiger Teil jeder Arbeit der viel Zeit in Anspruch nimmt
  $\Rightarrow$ Schützt davor etwas zu machen was es schon gibt!
- Muss immer vorhanden sein

Inhalt - Preliminaries

- Funktion und Inhalt:
  - Führt ins Fachgebiet ein
  - Erklärt Grundlagen auf denen die Arbeit aufbaut
  - Definiert Begriffe die für das Verständnis der Arbeit von nöten sind
- Muss nicht vorhanden sein; muss kein eigenes Kapitel sein

Inhalt - Hauptteil

- Funktion und Inhalt:
  - Erklärt was gemacht wurde :)
  - Nicht zu tief in die Implementierung gehen; abstrakt beschreiben
  - Inhalt extrem von Ziellesern/Betreuer abhängig
- Muss vorhanden sein; sonst hat man nix gemacht ;)

Inhalt - Evaluation

- Funktion und Inhalt:
  - Reflektiert über die eigene Arbeit
  - Vergleicht diese mit Related Work
  - Beschreibt die Durchführung und Ergebnisse von Experimenten die die Tauglichkeit des Ansatzes messen
  - Zieht kurzes Fazit, z.B. “die Ergebnisse zeigen, dass unser Ansatz sinnvoll ist ;)”
- Sollte auch immer vorhanden sein; die Tauglichkeit des eigenen Ansatzes zu zeigen ist zentral für jegliche Forschung

Inhalt - Conclusion

- Funktion und Inhalt:
  - Fasst nochmal kurz den Inhalt und die Ergebnisse der Arbeit zusammen (Summary)
  - Erwähnt nochmal die wichtigsten Ergebnisse der Evaluierung
  - Zieht ein Fazit
  - Zeigt mögliche Themen für eine folgende Arbeit auf (Future Work)
- Muss immer vorhanden sein; wird oft als erstes gelesen

Inhalt Zusammenfassung

- Alles hier erwähnte sind nur generelle Richtlinien
- Vor allem die Kapitelnamen müssen nicht unbedingt so heissen
- Die genau Struktur der Arbeit sollte immer mit dem Betreuer abgesprochen werden
  $\Rightarrow$ Jeder will was anderes :)
- Abstimmung des Inhalts auf die Forschungscommunity
  $\Rightarrow$ Z.B. Datenbänkler wollen immer gemessene Zahlen, Algorithmiker wollen immer genaue theoretische Schranken
- Wie immer gilt: Der Meister darf die Regel brechen

Outline

1 Wissenschaftliches Schreiben

2 Inhalt

3 Sprache

4 Layout

5 LaTeX

6 Grafiken, Tabellen, Listings

7 Bibliographie und Zitieren

Sprache - Wissenschaftlich

- Ziel des Schreibens: Kommunikation von Forschungsergebnissen
  - Nicht den Leser ansprechen (Kein Lehrbuch!)

    The reader may have a look at [. . .]

- “Unpersönliches” schreiben
  - Im Deutschen kein Ich oder Wir
  - Im Englischen ist “We” als “Die Autoren” oft okay

    Our approach yields the following benefits [. . .]

  - “We” als “der Leser und der Autor” ist nicht gut!

    Now we will have a look at [. . .]

- Umgangssprache vermeiden
- Im Englischen: Wörter mit lateinischer Herkunft (lang) statt Angelsächsischer Wörter (kurz)
  - let, set, get, put, test, need
  - assess, investigate, recommend, require

Sprache - Gute Sprache

- So schreiben wie man in Deutsch/Englisch Erörterungen gelernt hat!
- Gute Überleitungen verbinden Sätze und sind wichtig, e.g.:
  - Thus, Consequently, Next, Finally, usw.
- Englisch: Einfache Sätze! Keine komplizierten Schachtelsätze! (deutsche Unsitte)

Sprache - Using This Correctly

- Referenzierung mit “this” nur wenn Bezug absolut klar!
- Immer wieder checken! Oft baut man einen Satz dazwischen der Bezug zerreist.
- Häufiger Fehler!

Knotennamen noch nicht in der Überschrift nennen.

Select all descendants of node ”e”

Which? node

This query returns all descendants of e. In HanaDB this operation is called *subtree* and will be explained later. The aim of this example is to show how hierarchical queries are integrated into the query language of HanaDB.

Sprache - Using This Correctly

- Referenzierung mit “this” nur wenn Bezug absolut klar!
- Immer wieder checken! Oft baut man einen Satz dazwischen der Bezug zerreist.
- Häufiger Fehler!

object diagram contains three classes *Node*, *RootNode* and *Leaf*. These are the very basic types. Since we do not handle types in the example these are all classes to be defined. When modeling this, it is obvious that *Node* is the parent class, *RootNode* and *Leaf* are the descendants. The following listing shows how the basic structure of these classes looks

Was soll dieser Satz auss
? Unverständlich
what?
the

Sprache - Using This Correctly

- Referenzierung mit “this” nur wenn Bezug absolut klar!
- Immer wieder checken! Oft baut man einen Satz dazwischen der Bezug zerreist.
- Häufiger Fehler!

a new type called *NewNodeType*. The interesting part here is, that by adding methods to the hierarchy definition a natural way of querying is achieved. To show it the sibling query is displayed using the new method and the new hierarchy.

what?

where?
Referenzieren!

```
select s.id from n in NewExampleHierarchy, s in n.siblings() where n.id=1
```

Listing 3.9: Get siblings using OQL (basic hierarchy definition)

ting!

What?

This shows how hierarchies can be modeled in systems which support the ODMG standard. Similar to the *sibling* extension, methods can be implemented for all important func-

Hast du bereits auf der vorigen Seite gesagt!

Sprache - Englisch

- In der Regel wird Englisch geschrieben
- Informatik hat viele englische Wörter $\Rightarrow$ Deutsch hört sich meist komisch an.
- Englische Sätze sind kürzer $\Rightarrow$ einfacher zu verstehen
  - Kurz und prägnant formulieren! Keine Schachtelsätze wie im Deutschen.
- Auch mit nur durchschnittlichen Englischkenntnissen fällt schreiben auf Englisch schnell leichter als Deutsch!
- Tipp: Abschlussarbeiten auf Englisch schreiben :)
- Englisch erlaubt globale Veröffentlichung

Häufige Fehler im Englischen

- Bei Aufzählungen auch vor and und or ein Komma!
- Unterschied last und previous
- for example, i.e., e.g. (fast) immer in Kommas einschließen!
  - i.e. und e.g. nur in Klammern oder footnotes verwenden, sonst ausschreiben
  - i.e. = that is, e.g. = for example. Nicht vertauschen!
- Unterschied like (comparison) und such as (inclusion)
  - *I admire people like you*
  - *There are equivalent algorithms, such as bubble-sort, quick-sort,. . .*
  - *This complicates many tree structures, such as B-trees*
  - Oft benutzt: like, oft gemeint: such as

Sprache Diverses

- Jedes Wort Groß in Titel der Arbeit, Chapter- und Sectionüberschriften
  - Bis auf kleine Wörter wie Präpositionen und Artikel
  - Gilt nicht für Bild und Tabellen Unter- bzw. Überschriften
  - Beispiel:

    A Query Language for the Poor People and the Major of London

- Wenn eine Table, Figure, Chapter, Section mit Nummer genannt wird immer Groß, sonst normal klein
- Beispiel:

  As you can see in Figure 7.2 in Chapter 7, there are ten nodes. In addition, the figure in the mentioned chapter contains more content.

Outline

1 Wissenschaftliches Schreiben

2 Inhalt

3 Sprache

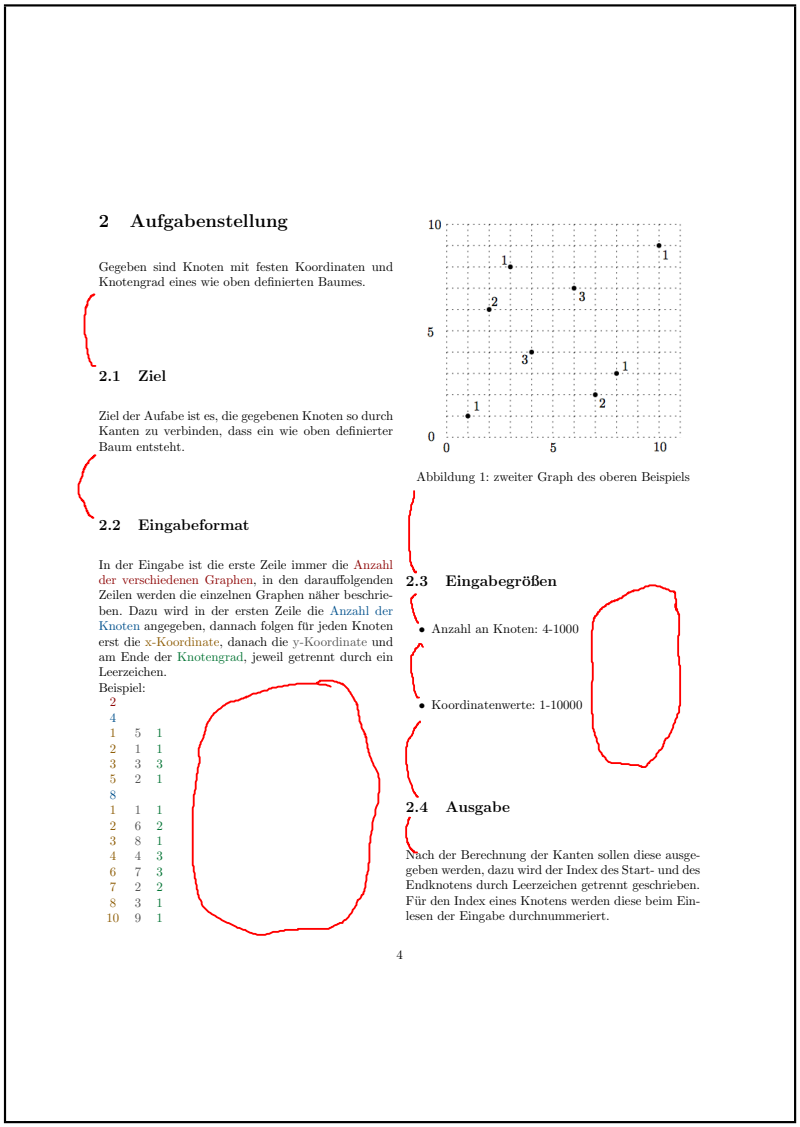
4 Layout

5 LaTeX

6 Grafiken, Tabellen, Listings

7 Bibliographie und Zitieren

Layout - Whitespace



2 Aufgabenstellung

Gegeben sind Knoten mit festen Koordinaten und Knotengrad eines wie oben definierten Baumes.

2.1 Ziel

Ziel der Aufabe ist es, die gegebenen Knoten so durch Kanten zu verbinden, dass ein wie oben definierter Baum entsteht.

2.2 Eingabeformat

In der Eingabe ist die erste Zeile immer die Anzahl der verschiedenen Graphen, in den darauffolgenden Zeilen werden die einzelnen Graphen näher beschrieben. Dazu wird in der ersten Zeile die Anzahl der Knoten angegeben, dannach folgen für jeden Knoten erst die x-Koordinate, danach die y-Koordinate und am Ende der Knotengrad, jeweil getrennt durch ein Leerzeichen.
Beispiel:

```
2
4
1 5 1
2 1 1
3 3 3
5 2 1
8
1 1 1
2 6 2
3 8 1
4 4 3
6 7 3
7 2 2
8 3 1
10 9 1
```


Abbildung 1: zweiter Graph des oberen Beispiels

2.3 Eingabegrößen

- Anzahl an Knoten: 4-1000
- Koordinatenwerte: 1-10000

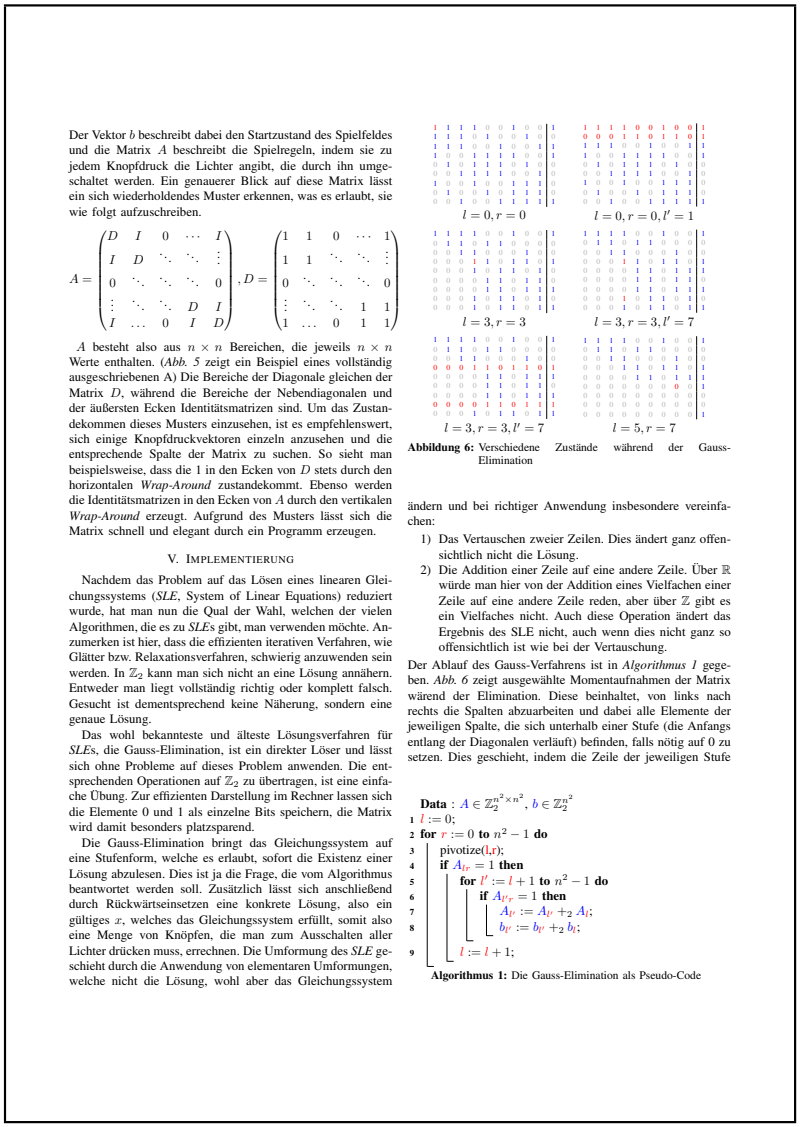
2.4 Ausgabe

Nach der Berechnung der Kanten sollen diese ausgegeben werden, dazu wird der Index des Start- und des Endknotens durch Leerzeichen getrennt geschrieben. Für den Index eines Knotens werden diese beim Einlesen der Eingabe durchnummeriert.

4

- Whitespace sieht generell schlecht aus
- $\rightarrow$ unbedingt vermeiden!
  - Genug Fließtext
  - Figures und co. umordnen
- Grundfrage: Würde so ein Fachbuch aussehen?

Layout - Whitespace



Der Vektor b beschreibt dabei den Startzustand des Spielfeldes und die Matrix A beschreibt die Spielregeln, indem sie zu jedem Knopfdruck die Lichter angibt, die durch ihn umgeschaltet werden. Ein genauerer Blick auf diese Matrix lässt ein sich wiederholdendes Muster erkennen, was es erlaubt, sie wie folgt aufzuschreiben.

$$A = \begin{pmatrix} D & I & 0 & \cdots & I \\ I & D & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & D & I \\ I & \dots & 0 & I & D \end{pmatrix}, D = \begin{pmatrix} 1 & 1 & 0 & \cdots & 1 \\ 1 & 1 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1 & 1 \\ 1 & \dots & 0 & 1 & 1 \end{pmatrix}$$

A besteht also aus $n \times n$ Bereichen, die jeweils $n \times n$ Werte enthalten. (*Abb. 5* zeigt ein Beispiel eines vollständig ausgeschriebenen A) Die Bereiche der Diagonale gleichen der Matrix D, während die Bereiche der Nebendiagonalen und der äußersten Ecken Identitätsmatrizen sind. Um das Zustandekommen dieses Musters einzusehen, ist es empfehlenswert, sich einige Knopfdruckvektoren einzeln anzusehen und die entsprechende Spalte der Matrix zu suchen. So sieht man beispielsweise, dass die 1 in den Ecken von D stets durch den horizontalen *Wrap-Around* zustandekommt. Ebenso werden die Identitätsmatrizen in den Ecken von A durch den vertikalen *Wrap-Around* erzeugt. Aufgrund des Musters lässt sich die Matrix schnell und elegant durch ein Programm erzeugen.

V. Implementierung

Nachdem das Problem auf das Lösen eines linearen Gleichungssystems (*SLE*, System of Linear Equations) reduziert wurde, hat man nun die Qual der Wahl, welchen der vielen Algorithmen, die es zu *SLE*s gibt, man verwenden möchte. Anzumerken ist hier, dass die effizienten iterativen Verfahren, wie Glätter bzw. Relaxationsverfahren, schwierig anzuwenden sein werden. In $\mathbb{Z}_2$ kann man sich nicht an eine Lösung annähern. Entweder man liegt vollständig richtig oder komplett falsch. Gesucht ist dementsprechend keine Näherung, sondern eine genaue Lösung.

Das wohl bekannteste und älteste Lösungsverfahren für *SLE*s, die Gauss-Elimination, ist ein direkter Löser und lässt sich ohne Probleme auf dieses Problem anwenden. Die entsprechenden Operationen auf $\mathbb{Z}_2$ zu übertragen, ist eine einfache Übung. Zur effizienten Darstellung im Rechner lassen sich die Elemente 0 und 1 als einzelne Bits speichern, die Matrix wird damit besonders platzsparend.

Die Gauss-Elimination bringt das Gleichungssystem auf eine Stufenform, welche es erlaubt, sofort die Existenz einer Lösung abzulesen. Dies ist ja die Frage, die vom Algorithmus beantwortet werden soll. Zusätzlich lässt sich anschließend durch Rückwärtseinsetzen eine konkrete Lösung, also ein gültiges x, welches das Gleichungssystem erfüllt, somit also eine Menge von Knöpfen, die man zum Ausschalten aller Lichter drücken muss, errechnen. Die Umformung des *SLE* geschieht durch die Anwendung von elementaren Umformungen, welche nicht die Lösung, wohl aber das Gleichungssystem

Abbildung 6: Verschiedene Zustände während der Gauss-Elimination

ändern und bei richtiger Anwendung insbesondere vereinfachen:

1) Das Vertauschen zweier Zeilen. Dies ändert ganz offensichtlich nicht die Lösung.
2) Die Addition einer Zeile auf eine andere Zeile. Über $\mathbb{R}$ würde man hier von der Addition eines Vielfachen einer Zeile auf eine andere Zeile reden, aber über $\mathbb{Z}$ gibt es ein Vielfaches nicht. Auch diese Operation ändert das Ergebnis des SLE nicht, auch wenn dies nicht ganz so offensichtlich ist wie bei der Vertauschung.

Der Ablauf des Gauss-Verfahrens ist in *Algorithmus 1* gegeben. *Abb. 6* zeigt ausgewählte Momentaufnahmen der Matrix wärend der Elimination. Diese beinhaltet, von links nach rechts die Spalten abzuarbeiten und dabei alle Elemente der jeweiligen Spalte, die sich unterhalb einer Stufe (die Anfangs entlang der Diagonalen verläuft) befinden, falls nötig auf 0 zu setzen. Dies geschieht, indem die Zeile der jeweiligen Stufe

Data : $A \in \mathbb{Z}_2^{n^2 \times n^2}$, $b \in \mathbb{Z}_2^{n^2}$
$l := 0$;
for $r := 0$ **to** $n^2 - 1$ **do**
  pivotize(l,r);
  if $A_{lr} = 1$ **then**
    for $l' := l + 1$ **to** $n^2 - 1$ **do**
      if $A_{l'r} = 1$ **then**
        $A_{l'} := A_{l'} +_2 A_l$;
        $b_{l'} := b_{l'} +_2 b_l$;
    $l := l + 1$;

Algorithmus 1: Die Gauss-Elimination als Pseudo-Code

- Whitespace sieht generell schlecht aus
- $\rightarrow$ unbedingt vermeiden!
  - Genug Fließtext
  - Figures und co. umordnen
- Grundfrage: Würde so ein Fachbuch aussehen?

Layout - “Textflow”

Der Vektor b beschreibt dabei den Startzustand des Spielfeldes und die Matrix A beschreibt die Spielregeln, indem sie zu jedem Knopfdruck die Lichter angibt, die durch ihn umgeschaltet werden. Ein genauerer Blick auf diese Matrix lässt ein sich wiederholdendes Muster erkennen, was es erlaubt, sie wie folgt aufzuschreiben.

$$A = \begin{pmatrix} D & I & 0 & \cdots & I \\ I & D & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & D & I \\ I & \dots & 0 & I & D \end{pmatrix}, D = \begin{pmatrix} 1 & 1 & 0 & \cdots & 1 \\ 1 & 1 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1 & 1 \\ 1 & \dots & 0 & 1 & 1 \end{pmatrix}$$

A besteht also aus $n \times n$ Bereichen, die jeweils $n \times n$ Werte enthalten. (*Abb. 5* zeigt ein Beispiel eines vollständig ausgeschriebenen A) Die Bereiche der Diagonale gleichen der Matrix D, während die Bereiche der Nebendiagonalen und der äußersten Ecken Identitätsmatrizen sind. Um das Zustandekommen dieses Musters einzusehen, ist es empfehlenswert, sich einige Knopfdruckvektoren einzeln anzusehen und die entsprechende Spalte der Matrix zu suchen. So sieht man beispielsweise, dass die 1 in den Ecken von D stets durch den horizontalen *Wrap-Around* zustandekommt. Ebenso werden die Identitätsmatrizen in den Ecken von A durch den vertikalen *Wrap-Around* erzeugt. Aufgrund des Musters lässt sich die Matrix schnell und elegant durch ein Programm erzeugen.

V. Implementierung

Nachdem das Problem auf das Lösen eines linearen Gleichungssystems (*SLE*, System of Linear Equations) reduziert wurde, hat man nun die Qual der Wahl, welchen der vielen Algorithmen, die es zu *SLE*s gibt, man verwenden möchte. Anzumerken ist hier, dass die effizienten iterativen Verfahren, wie Glätter bzw. Relaxationsverfahren, schwierig anzuwenden sein werden. In $\mathbb{Z}_2$ kann man sich nicht an eine Lösung annähern. Entweder man liegt vollständig richtig oder komplett falsch. Gesucht ist dementsprechend keine Näherung, sondern eine genaue Lösung.

Das wohl bekannteste und älteste Lösungsverfahren für *SLE*s, die Gauss-Elimination, ist ein direkter Löser und lässt sich ohne Probleme auf dieses Problem anwenden. Die entsprechenden Operationen auf $\mathbb{Z}_2$ zu übertragen, ist eine einfache Übung. Zur effizienten Darstellung im Rechner lassen sich die Elemente 0 und 1 als einzelne Bits speichern, die Matrix wird damit besonders platzsparend.

Die Gauss-Elimination bringt das Gleichungssystem auf eine Stufenform, welche es erlaubt, sofort die Existenz einer Lösung abzulesen. Dies ist ja die Frage, die vom Algorithmus beantwortet werden soll. Zusätzlich lässt sich anschließend durch Rückwärtseinsetzen eine konkrete Lösung, also ein gültiges x, welches das Gleichungssystem erfüllt, somit also eine Menge von Knöpfen, die man zum Ausschalten aller Lichter drücken muss, errechnen. Die Umformung des *SLE* geschieht durch die Anwendung von elementaren Umformungen, welche nicht die Lösung, wohl aber das Gleichungssystem

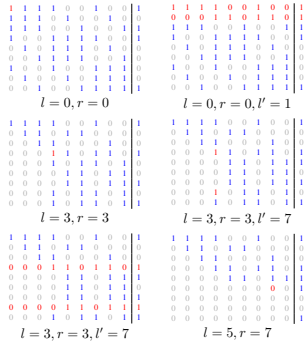


Abbildung 6: Verschiedene Zustände während der Gauss-Elimination

ändern und bei richtiger Anwendung insbesondere vereinfachen:

1) Das Vertauschen zweier Zeilen. Dies ändert ganz offensichtlich nicht die Lösung.
2) Die Addition einer Zeile auf eine andere Zeile. Über $\mathbb{R}$ würde man hier von der Addition eines Vielfachen einer Zeile auf eine andere Zeile reden, aber über $\mathbb{Z}$ gibt es ein Vielfaches nicht. Auch diese Operation ändert das Ergebnis des SLE nicht, auch wenn dies nicht ganz so offensichtlich ist wie bei der Vertauschung.

Der Ablauf des Gauss-Verfahrens ist in *Algorithmus 1* gegeben. *Abb. 6* zeigt ausgewählte Momentaufnahmen der Matrix während der Elimination. Diese beinhaltet, von links nach rechts die Spalten abzuarbeiten und dabei alle Elemente der jeweiligen Spalte, die sich unterhalb einer Stufe (die Anfangs entlang der Diagonalen verläuft) befinden, falls nötig auf 0 zu setzen. Dies geschieht, indem die Zeile der jeweiligen Stufe

Data : $A \in \mathbb{Z}_2^{n^2 \times n^2}$, $b \in \mathbb{Z}_2^{n^2}$
$l := 0$;
for $r := 0$ **to** $n^2 - 1$ **do**
  pivotize(l,r);
  if $A_{lr} = 1$ **then**
    for $l' := l + 1$ **to** $n^2 - 1$ **do**
      if $A_{l'r} = 1$ **then**
        $A_{l'} := A_{l'} +_2 A_l$;
        $b_{l'} := b_{l'} +_2 b_l$;
    $l := l + 1$;

Algorithmus 1: Die Gauss-Elimination als Pseudo-Code

- Text muss einen gewissen “Fluss” besitzen
- Arbeit angenehm zu lesen
- Hauptziel: Text nicht zu sehr zerstückeln
  - Genug Fließtext
  - Nicht zu viele Figures, Tables und Listen pro Seite!
  - Nicht zu viele Sections!
- Grundfrage: Würde so ein Fachbuch aussehen?

Layout - “Textflow”

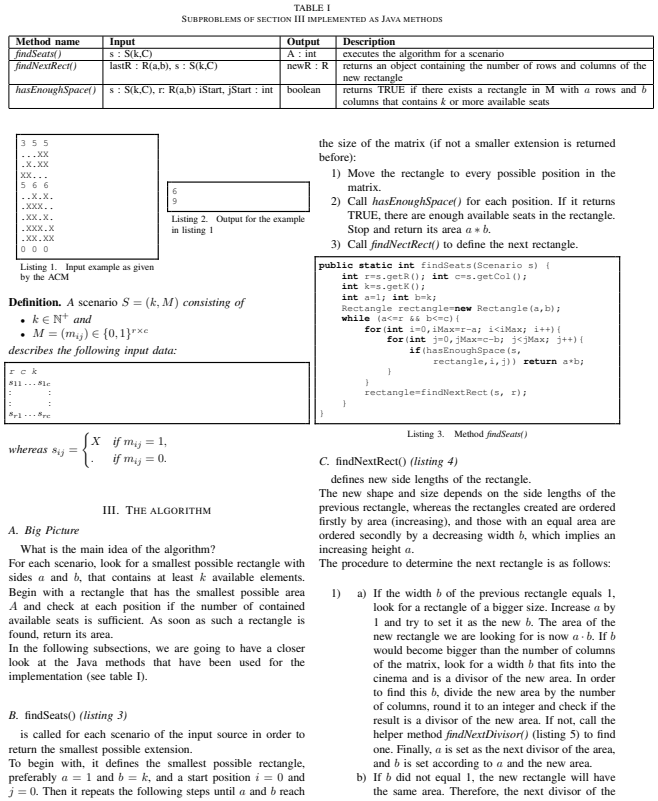


TABLE I
SUBPROBLEMS OF SECTION III IMPLEMENTED AS JAVA METHODS

Method name	Input	Output	Description
findSeats()	s : S(k,C)	A : int	executes the algorithm for a scenario
findNextRect()	lastR : R(a,b), s : S(k,C)	newR : R	returns an object containing the number of rows and columns of the new rectangle
hasEnoughSpace()	s : S(k,C), r: R(a,b) iStart, jStart : int	boolean	returns TRUE if there exists a rectangle in M with a rows and b columns that contains k or more available seats

```
3 5 5
...XX
.X.XX
XX...
5 6 6
..X.X.
.XXX..
.XX.X.
.XXX.X
.XX.XX
0 0 0
```

Listing 1. Input example as given by the ACM

```
6
9
```

Listing 2. Output for the example in listing 1

Definition. *A* scenario $S = (k, M)$ *consisting of*

- $k \in \mathbb{N}^+$ *and*
- $M = (m_{ij}) \in \{0,1\}^{r \times c}$

describes the following input data:

```
r c k
s11 ... s1c
 :       :
 :       :
sr1 ... src
```

whereas $s_{ij} = \begin{cases} X & \text{if } m_{ij} = 1, \\ . & \text{if } m_{ij} = 0. \end{cases}$

III. The algorithm

A. Big Picture

What is the main idea of the algorithm?
For each scenario, look for a smallest possible rectangle with sides a and b, that contains at least k available elements. Begin with a rectangle that has the smallest possible area A and check at each position if the number of contained available seats is sufficient. As soon as such a rectangle is found, return its area.
In the following subsections, we are going to have a closer look at the Java methods that have been used for the implementation (see table I).

B. findSeats() *(listing 3)*

is called for each scenario of the input source in order to return the smallest possible extension.
To begin with, it defines the smallest possible rectangle, preferably $a = 1$ and $b = k$, and a start position $i = 0$ and $j = 0$. Then it repeats the following steps until a and b reach the size of the matrix (if not a smaller extension is returned before):

1) Move the rectangle to every possible position in the matrix.
2) Call *hasEnoughSpace()* for each position. If it returns TRUE, there are enough available seats in the rectangle. Stop and return its area $a * b$.
3) Call *findNectRect()* to define the next rectangle.

```
public static int findSeats(Scenario s) {
    int r=s.getR(); int c=s.getCol();
    int k=s.getK();
    int a=1; int b=k;
    Rectangle rectangle=new Rectangle(a,b);
    while (a<=r && b<=c){
        for(int i=0,iMax=r-a; i<iMax; i++){
            for(int j=0,jMax=c-b; j<jMax; j++){
                if(hasEnoughSpace(s,
                    rectangle,i,j)) return a*b;
            }
        }
        rectangle=findNextRect(s, r);
    }
}
```

Listing 3. Method *findSeats()*

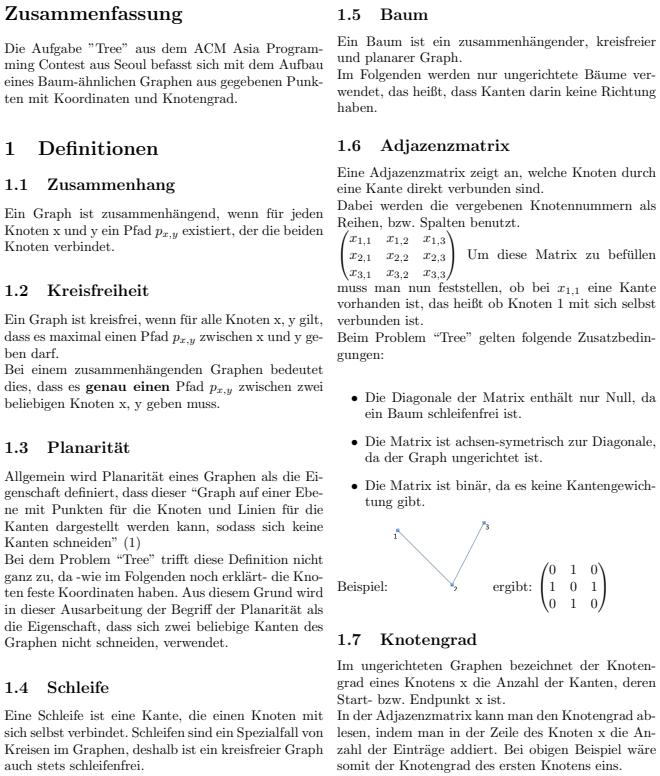
C. findNextRect() *(listing 4)*

defines new side lengths of the rectangle.
The new shape and size depends on the side lengths of the previous rectangle, whereas the rectangles created are ordered firstly by area (increasing), and those with an equal area are ordered secondly by a decreasing width b, which implies an increasing height a.
The procedure to determine the next rectangle is as follows:

1) a) If the width b of the previous rectangle equals 1, look for a rectangle of a bigger size. Increase a by 1 and try to set it as the new b. The area of the new rectangle we are looking for is now $a \cdot b$. If b would become bigger than the number of columns of the matrix, look for a width b that fits into the cinema and is a divisor of the new area. In order to find this b, divide the new area by the number of columns, round it to an integer and check if the result is a divisor of the new area. If not, call the helper method *findNextDivisor()* (listing 5) to find one. Finally, a is set as the next divisor of the area, and b is set according to a and the new area.
   b) If b did not equal 1, the new rectangle will have the same area. Therefore, the next divisor of the

- Text muss einen gewissen “Fluss” besitzen
- Arbeit angenehm zu lesen
- Hauptziel: Text nicht zu sehr zerstückeln
  - Genug Fließtext
  - Nicht zu viele Figures, Tables und Listen pro Seite!
  - Nicht zu viele Sections!
- Grundfrage: Würde so ein Fachbuch aussehen?

Layout - “Textflow”



Zusammenfassung

Die Aufgabe "Tree" aus dem ACM Asia Programming Contest aus Seoul befasst sich mit dem Aufbau eines Baum-ähnlichen Graphen aus gegebenen Punkten mit Koordinaten und Knotengrad.

1 Definitionen

1.1 Zusammenhang

Ein Graph ist zusammenhängend, wenn für jeden Knoten x und y ein Pfad $p_{x,y}$ existiert, der die beiden Knoten verbindet.

1.2 Kreisfreiheit

Ein Graph ist kreisfrei, wenn für alle Knoten x, y gilt, dass es maximal einen Pfad $p_{x,y}$ zwischen x und y geben darf.
Bei einem zusammenhängenden Graphen bedeutet dies, dass es **genau einen** Pfad $p_{x,y}$ zwischen zwei beliebigen Knoten x, y geben muss.

1.3 Planarität

Allgemein wird Planarität eines Graphen als die Eigenschaft definiert, dass dieser “Graph auf einer Ebene mit Punkten für die Knoten und Linien für die Kanten dargestellt werden kann, sodass sich keine Kanten schneiden” (1)
Bei dem Problem “Tree” trifft diese Definition nicht ganz zu, da -wie im Folgenden noch erklärt- die Knoten feste Koordinaten haben. Aus diesem Grund wird in dieser Ausarbeitung der Begriff der Planarität als die Eigenschaft, dass sich zwei beliebige Kanten des Graphen nicht schneiden, verwendet.

1.4 Schleife

Eine Schleife ist eine Kante, die einen Knoten mit sich selbst verbindet. Schleifen sind ein Spezialfall von Kreisen im Graphen, deshalb ist ein kreisfreier Graph auch stets schleifenfrei.

1.5 Baum

Ein Baum ist ein zusammenhängender, kreisfreier und planarer Graph.
Im Folgenden werden nur ungerichtete Bäume verwendet, das heißt, dass Kanten darin keine Richtung haben.

1.6 Adjazenzmatrix

Eine Adjazenzmatrix zeigt an, welche Knoten durch eine Kante direkt verbunden sind.
Dabei werden die vergebenen Knotennummern als Reihen, bzw. Spalten benutzt.
$\begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{pmatrix}$ Um diese Matrix zu befüllen muss man nun feststellen, ob bei $x_{1,1}$ eine Kante vorhanden ist, das heißt ob Knoten 1 mit sich selbst verbunden ist.
Beim Problem “Tree” gelten folgende Zusatzbedingungen:

- Die Diagonale der Matrix enthält nur Null, da ein Baum schleifenfrei ist.
- Die Matrix ist achsen-symetrisch zur Diagonale, da der Graph ungerichtet ist.
- Die Matrix ist binär, da es keine Kantengewichtung gibt.

Beispiel:

ergibt: $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

1.7 Knotengrad

Im ungerichteten Graphen bezeichnet der Knotengrad eines Knotens x die Anzahl der Kanten, deren Start- bzw. Endpunkt x ist.
In der Adjazenzmatrix kann man den Knotengrad ablesen, indem man in der Zeile des Knoten x die Anzahl der Einträge addiert. Bei obigen Beispiel wäre somit der Knotengrad des ersten Knotens eins.

3

- Text muss einen gewissen “Fluss” besitzen
- Arbeit angenehm zu lesen
- Hauptziel: Text nicht zu sehr zerstückeln
  - Genug Fließtext
  - Nicht zu viele Figures, Tables und Listen pro Seite!
  - Nicht zu viele Sections!
- Grundfrage: Würde so ein Fachbuch aussehen?

Layout - Diverses

Der Vektor b beschreibt dabei den Startzustand des Spielfeldes und die Matrix A beschreibt die Spielregeln, indem sie zu jedem Knopfdruck die Lichter angibt, die durch ihn umgeschaltet werden. Ein genauerer Blick auf diese Matrix lässt ein sich wiederholdendes Muster erkennen, was es erlaubt, sie wie folgt aufzuschreiben.

$$A = \begin{pmatrix} D & I & 0 & \cdots & I \\ I & D & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & D & I \\ I & \dots & 0 & I & D \end{pmatrix}, D = \begin{pmatrix} 1 & 1 & 0 & \cdots & 1 \\ 1 & 1 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1 & 1 \\ 1 & \dots & 0 & 1 & 1 \end{pmatrix}$$

A besteht also aus $n \times n$ Bereichen, die jeweils $n \times n$ Werte enthalten. (*Abb. 5* zeigt ein Beispiel eines vollständig ausgeschriebenen A) Die Bereiche der Diagonale gleichen der Matrix D, während die Bereiche der Nebendiagonalen und der äußersten Ecken Identitätsmatrizen sind. Um das Zustandekommen dieses Musters einzusehen, ist es empfehlenswert, sich einige Knopfdruckvektoren einzeln anzusehen und die entsprechende Spalte der Matrix zu suchen. So sieht man beispielsweise, dass die 1 in den Ecken von D stets durch den horizontalen *Wrap-Around* zustandekommt. Ebenso werden die Identitätsmatrizen in den Ecken von A durch den vertikalen *Wrap-Around* erzeugt. Aufgrund des Musters lässt sich die Matrix schnell und elegant durch ein Programm erzeugen.

V. Implementierung

Nachdem das Problem auf das Lösen eines linearen Gleichungssystems (*SLE*, System of Linear Equations) reduziert wurde, hat man nun die Qual der Wahl, welchen der vielen Algorithmen, die es zu *SLE*s gibt, man verwenden möchte. Anzumerken ist hier, dass die effizienten iterativen Verfahren, wie Glätter bzw. Relaxationsverfahren, schwierig anzuwenden sein werden. In $\mathbb{Z}_2$ kann man sich nicht an eine Lösung annähern. Entweder man liegt vollständig richtig oder komplett falsch. Gesucht ist dementsprechend keine Näherung, sondern eine genaue Lösung.

Das wohl bekannteste und älteste Lösungsverfahren für *SLE*s, die Gauss-Elimination, ist ein direkter Löser und lässt sich ohne Probleme auf dieses Problem anwenden. Die entsprechenden Operationen auf $\mathbb{Z}_2$ zu übertragen, ist eine einfache Übung. Zur effizienten Darstellung im Rechner lassen sich die Elemente 0 und 1 als einzelne Bits speichern, die Matrix wird damit besonders platzsparend.

Die Gauss-Elimination bringt das Gleichungssystem auf eine Stufenform, welche es erlaubt, sofort die Existenz einer Lösung abzulesen. Dies ist ja die Frage, die vom Algorithmus beantwortet werden soll. Zusätzlich lässt sich anschließend durch Rückwärtseinsetzen eine konkrete Lösung, also ein gültiges x, welches das Gleichungssystem erfüllt, somit also eine Menge von Knöpfen, die man zum Ausschalten aller Lichter drücken muss, errechnen. Die Umformung des *SLE* geschieht durch die Anwendung von elementaren Umformungen, welche nicht die Lösung, wohl aber das Gleichungssystem

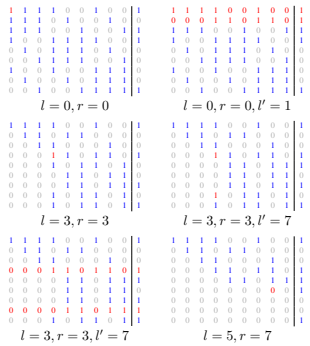


Abbildung 6: Verschiedene Zustände während der Gauss-Elimination

ändern und bei richtiger Anwendung insbesondere vereinfachen:

1) Das Vertauschen zweier Zeilen. Dies ändert ganz offensichtlich nicht die Lösung.
2) Die Addition einer Zeile auf eine andere Zeile. Über $\mathbb{R}$ würde man hier von der Addition eines Vielfachen einer Zeile auf eine andere Zeile reden, aber über $\mathbb{Z}$ gibt es ein Vielfaches nicht. Auch diese Operation ändert das Ergebnis des SLE nicht, auch wenn dies nicht ganz so offensichtlich ist wie bei der Vertauschung.

Der Ablauf des Gauss-Verfahrens ist in *Algorithmus 1* gegeben. *Abb. 6* zeigt ausgewählte Momentaufnahmen der Matrix während der Elimination. Diese beinhaltet, von links nach rechts die Spalten abzuarbeiten und dabei alle Elemente der jeweiligen Spalte, die sich unterhalb einer Stufe (die Anfangs entlang der Diagonalen verläuft) befinden, falls nötig auf 0 zu setzen. Dies geschieht, indem die Zeile der jeweiligen Stufe

Data : $A \in \mathbb{Z}_2^{n^2 \times n^2}$, $b \in \mathbb{Z}_2^{n^2}$
$l := 0$;
for $r := 0$ **to** $n^2 - 1$ **do**
  pivotize(l,r);
  if $A_{lr} = 1$ **then**
    for $l' := l + 1$ **to** $n^2 - 1$ **do**
      if $A_{l'r} = 1$ **then**
        $A_{l'} := A_{l'} +_2 A_l$;
        $b_{l'} := b_{l'} +_2 b_l$;
    $l := l + 1$;

Algorithmus 1: Die Gauss-Elimination als Pseudo-Code

- Text und Bilder dürfen nicht über Zeile hinausgehen
- Aufpassen, dass alle Referenzen und Citations existieren.
- LaTeX produziert bei beidem eine Warnung
  - $\Rightarrow$ Am Ende immer alle Warnungen checken!

Outline

1. Wissenschaftliches Schreiben
2. Inhalt
3. Sprache
4. Layout
5. LaTeX
6. Grafiken, Tabellen, Listings
7. Bibliographie und Zitieren

LaTeX

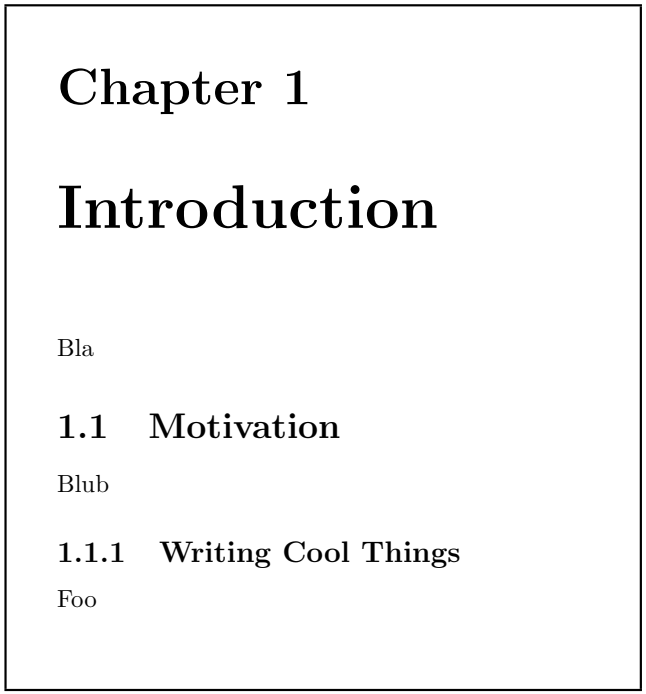
- Textsatzprogramm von Leslie Lamport (der mit den Uhren ;)
- Basiert auf TeX von Donald E. Knuth (ja, DER Knuth ;)
- Nicht WYSIWYG (wie Word), sondern WYSIWYM
  - Text wird programmiert
  - Man arbeitet direkt im Quelltext
  - `pdftex` compiliert den Quelltext nach `pdf`
- Vorteile
  - Sieht standardmäßig gut aus (wie ein Buch)
  - User hat volle Kontrolle über Layout
  - Trennung: Style $\Leftrightarrow$ logische Struktur
  - Extrem guter Formelsatz $\Rightarrow$ Mathematiker lieben es
  - Sonderzeichen einfach (z.B. griechisch α,ω oder rel. algebra $\bowtie$!)
- Standard in der Informatik
  - Manche schreiben noch in Word, das sieht man meistens sofort!
  - Leider noch nicht in vielen anderen Fachbereichen angekommen

LaTeX Basislayout

```
\documentclass{report}

\usepackage[utf8]{inputenc}

\begin{document}
\chapter{Introduction}
Bla
\section{Motivation}
Blub
\subsection{Writing Cool Things}
Foo
\end{document}
```



Genaueres: LaTeX Tutorial!

LaTeX Referenzierung

- Labels setzen mit `\label{NAME}`
- Auf Label verweisen mit `\ref{NAME}`
- Niemals Referenzen per Hand vergeben! Nummern können sich ändern.

```
\chapter{Introduction}
\label{chap:introduction}
\section{Motivation}
\label{sec:motivation}
\subsection{Writing Cool Things}

\begin{figure}[h]
\center
\includegraphics[scale=0.9]{img/Model.pdf}
\caption{This is a figure!}
\label{fig:model}
\end{figure}

We are currently in Chapter \ref{chap:introduction}
and Section \ref{sec:motivation}.
Our first figure is Figure \ref{fig:model}.
```

Chapter 1

Introduction

1.1 Motivation

1.1.1 Writing Cool Things

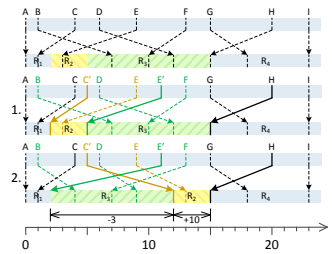


Figure 1.1: This is a figure!

We are currently in Chapter 1 and Section 1.1. Our first figure is Figure 1.1.

LaTeX Misc

- Verwende `$\mathcal{O}(n)$` für schönes Landau Symbol $\mathcal{O}(n)$.
- Verwende `~` für einen non-breaking-space bei `\ref` und `\cite` um Zeilenumbruch dazwischen zu verhindern:

```
As you can see in Figure~\ref{fig:foo},
nobody expects the spanish inquisition.
This fact has also been proven by
Python~\cite{python1970spanish},
so we can be quite sure about it.
```

LaTeX Makros

- Makros erlauben die Definition eigener Kommandos und Umgebungen
- Können extrem viel Schreibarbeit sparen
- Machen den Quelltext lesbarer
- Helfen Style von Inhalt zu trennen
- Makrodefinition relativ einfach, z.b.:
  - `\newcommand{\todo}[1]{\textbf{TODO: #1}}`
  - `\todo{Hier fehlt noch was}`
  - `\newcommand{setOfKs}{\mathcal{K}}`
  - `$\setOfKs = \{a,b\}$`
- Keine Angst vor Makros! Verwenden wenn sinnvoll!

Outline

1. Wissenschaftliches Schreiben
2. Inhalt
3. Sprache
4. Layout
5. LaTeX
6. Grafiken, Tabellen, Listings
7. Bibliographie und Zitieren

Float Environments

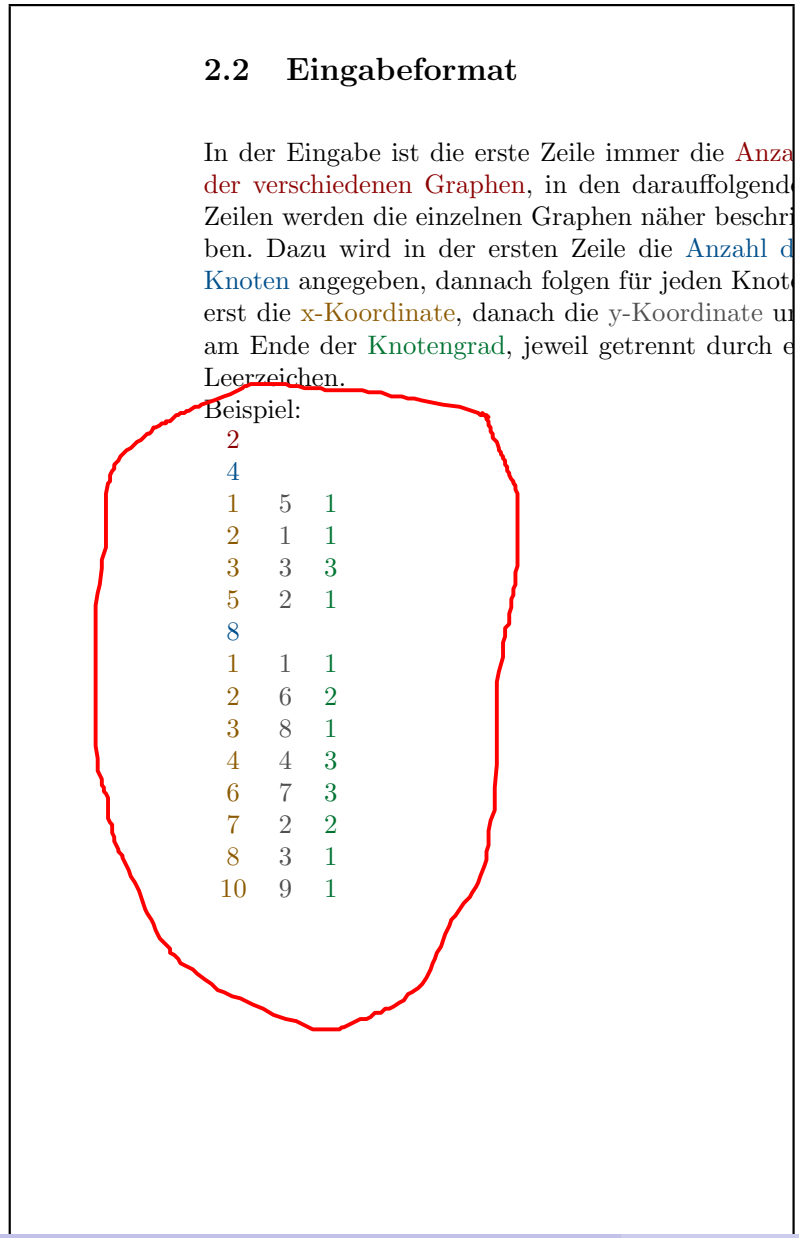
- Tabellen, Grafiken, Listings immer in entsprechende Float environments einbetten, z.B. `figure` und `table`
  - LaTeX übernimmt das Layout
  - Bilder werden da hingeschoben wo es “gut” aussieht
- Über- bzw. Unterschrift für Tabellen und Figures mit `\caption`
  - Keine Einführung wie “This figure show...”. Das weiss der Leser selbst. Nur Inhalt erklären!
- Im Text immer mit `\label` und `\ref` referenzieren
  - Niemals mit “the figure above” $\Rightarrow$ könnte verschoben werden!
  - Jede Tabelle und Figure muss im Text referenziert und vollständig erklärt werden. Unreferenzierte Figures sind ein absolutes No-Go!
  - “Hauptakteur” ist der Text. Der Leser ließt ihn und schaut Figures dann an wenn sie referenziert werden.

Float Environments Beispiel

```
\begin{figure}
  \center
  \includegraphics[scale=0.7]{img/InsertNode.pdf}
  \caption{Inserting a node into a graph}
  \label{fig:insert}
\end{figure}

Inserting a node into a graph is a difficult task.
An example insertion sequence is shown in
Figure~\ref{fig:insert}. First, the node is inserted.
Then,[...]
```

Float Environments



2.2 Eingabeformat

In der Eingabe ist die erste Zeile immer die Anza
der verschiedenen Graphen, in den darauffolgend
Zeilen werden die einzelnen Graphen näher beschri
ben. Dazu wird in der ersten Zeile die Anzahl d
Knoten angegeben, dannach folgen für jeden Knot
erst die x-Koordinate, danach die y-Koordinate u
am Ende der Knotengrad, jeweil getrennt durch e
Leerzeichen.
Beispiel:

```
2
4
1   5   1
2   1   1
3   3   3
5   2   1
8
1   1   1
2   6   2
3   8   1
4   4   3
6   7   3
7   2   2
8   3   1
10  9   1
```

- Alles in eurem File sollte entweder Fließtext, Aufzählungen, mathematische Formeln, oder Floats sein. Kein Zwischending!

Float Environments

III. Aufgabestellung

Die Aufgabestellung betrachtet n verschiedene Küchenschränke, die jeweils als zweidimensionales Koordinatensystem behandelt werden. Die untere linke Ecke (0,0) und die obere rechte Ecke (1000,1000) geben uns die genaue Größe des Raumes. Damit können wir leicht die Anzahl m an Elementen, die wir höchstens einfügen können bzw. mindestens einfügen müssen, berechnen, und zwar auf m mit $0 < m < 100$. Jedes Element wird mit zwei Koordinaten beschrieben, nämlich der unteren linken und oberen rechten Ecke. Das erleichtet uns die Suche nach bestimmten Elementen von unserem Küchenschrank. Laut unserem Beispiel (*siehe Fig.*1) haben die Elemente die folgenden Koordinaten:

- A: (5,0) (8,3);
- B: (2,3) (6,7);
- C: (7,3) (10,6);
- D: (5,7) (12,9);
- E: (4,7) (5,10);

Spezifikation der Eingabe:

m
$n_1 ... n_m$
n_j
$m = 0$

Spezifikation der Ausgabe:

s

m	Anzahl an Elementen im Küchenschrank
$n_1 n_2 ... n_m$	Koordinaten von allen Elementen n
n_j	Koordinaten des gewünschten Elementes ($n_j \ \epsilon(n_1, ..., n_m)$)
$m = 0$	Ein leerer Küchenschrank beendet die Eingabe
s	Anzahl an Elementen, die entfernt werden müssen

Beispiel 1 (*siehe Fig.*1)*:*

8
5 0 8 3 2 3 6 7 7 3 10 6 5 7 12 9 4 7 5 10
5 0 8 3
0

Ausgabe zu Beispiel 1:

4

IV. Algorithmus und Implementierung

Die Grundidee des Algorithmuses besteht darin, so schnell und effizient wie möglich durch alle Kombinationen von Elementen durchzugehen und zu zählen, wie viele davon über unseren gewünschten Element liegen. Um die Anzahl an Überpruefungen möglichst gering zu halten, nehmen wir an, dass sich ein Element nur dann über einem anderen befinden kann, wenn es sich an einer höhere Position in dem Koordinatensystem befindet. Sei A das ausgewählte Element und B das eventuell darüberliegende Element. Daraus folgen drei Fälle, in denen B genau über A liegen kann:

1 Fall: B liegt rechts von A (*siehe Fig.*2)
2 Fall: B liegt links von A (*siehe Fig.*3)
3 Fall: B beginnt links von A oder über A und endet rechts von A oder über A (*siehe Fig.*4)

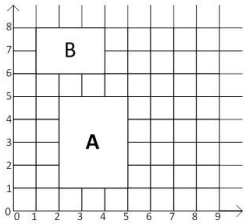


Fig. 2. Fall 1

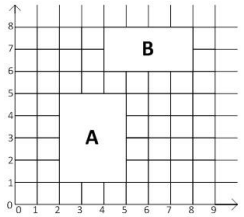


Fig. 3. Fall 2

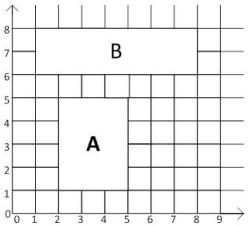
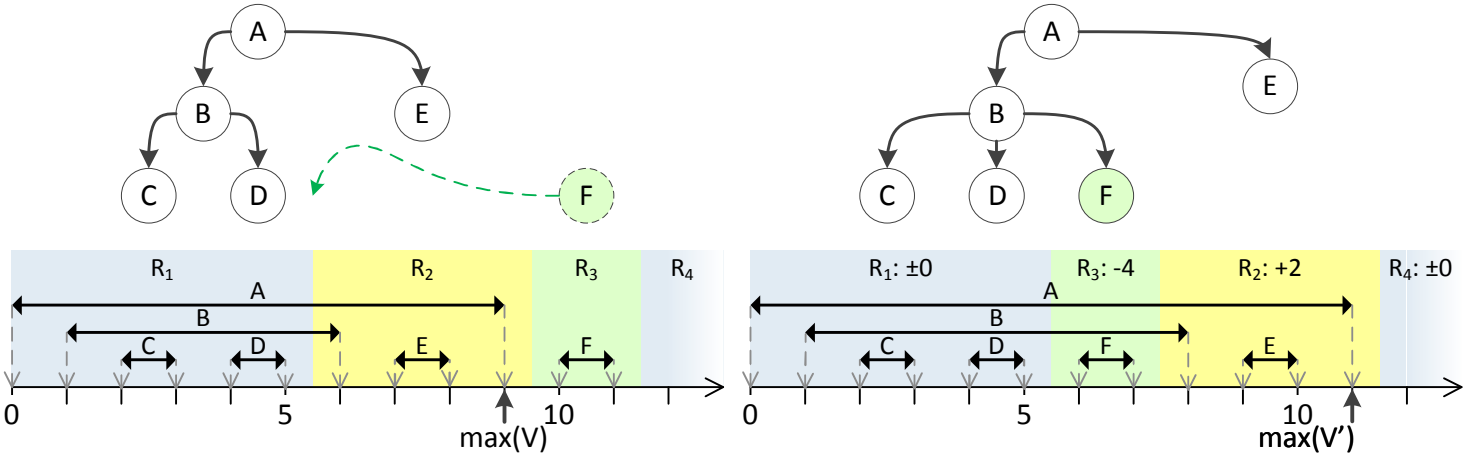


Fig. 4. Fall 3

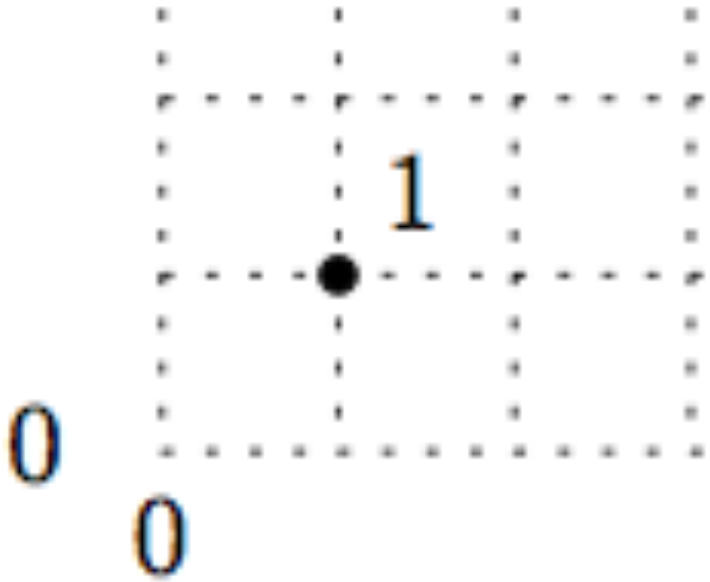
- Alles in eurem File sollte entweder Fließtext, Aufzählungen, mathematische Formeln, oder Floats sein. Kein Zwischending!

Grafiken

- Gute (!) Grafiken sind wichtiger Bestandteil jeder Arbeit
- Menschen verstehen Bilder schneller als Text
- “Ein Bild sagt mehr als tausend Worte”
  - . . . aber nur wenn es gut ist!
- Bilder müssen im Text erklärt werden



Grafiken - Format



- Vektorgrafiken statt Bitmaps
  - Scharf bei Zoom und Drucken
  - Speicherschonend
- Dateiformat normalerweise `pdf`
- Niemals `jpg`, `png`, `bmp` (Bitmaps!)

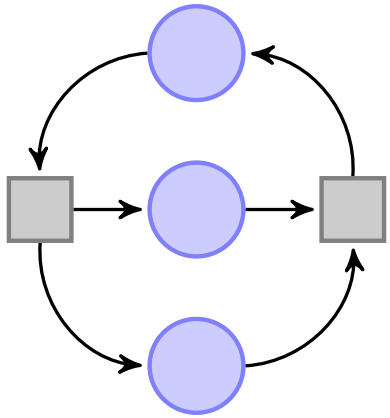
Abbildung 1: zwe

Grafiken - Programme

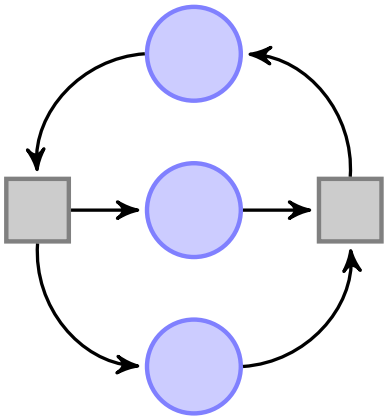
- TikZ
  - Sehr gutes Ergebnis
  - Hoher Aufwand
  - Lohnt sich bei manchen Formen
  - Tutorial siehe Website
- Visio
  - Gutes Ergebnis
  - Gratis Visio 2013 Professional auf dem RBG Dreamspark Server
  - Intuitiv, geringer Aufwand
  - Raster
- Andere Vektorzeichenprogramme
  - Wer's kann...
- Powerpoint und co.
  - Das lassen wir lieber!

TikZ

- Bilder werden im LaTeX code “programmiert”
- Langwierig, aber extrem genau
- Automatisierbarkeit durch Macros
  - Positiv wenn man mehrere ähnliche Bilder macht
  - z.B. Automaten, Petrinetze und co.
- Logische Positionierung möglich

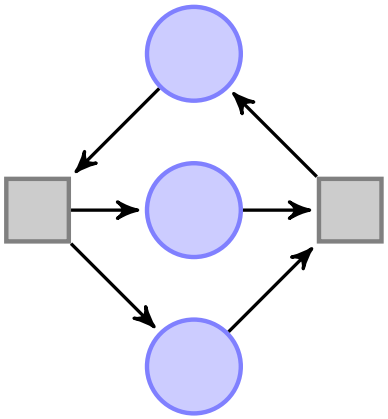


TikZ



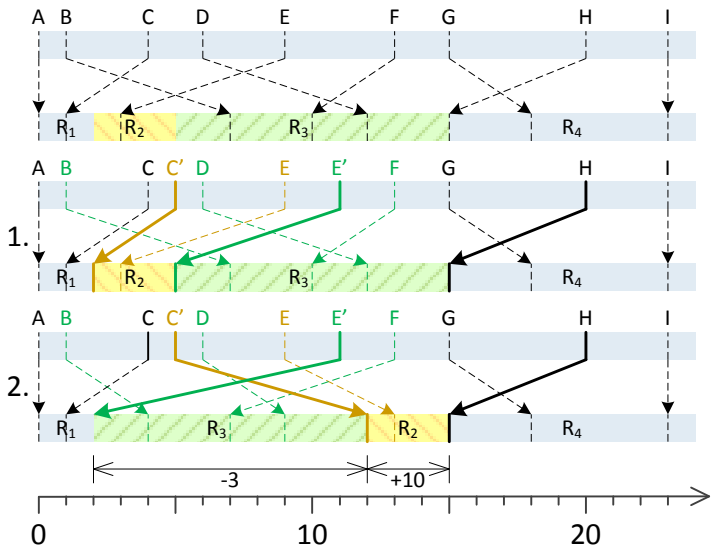
```
\tikzstyle{place}=[circle,draw=blue!50,fill=blue!20,thick,
inner sep=0pt,minimum size=6mm]
\tikzstyle{transition}=[rectangle,draw=black!50,fill=black!20,thick,
inner sep=0pt,minimum size=4mm]
\tikzstyle{pre}=[<-,shorten <=1pt,>=stealth',semithick]
\tikzstyle{post}=[->,shorten >=1pt,>=stealth',semithick]
\begin{tikzpicture}[bend angle=45]
\node[place] (waiting) {};
\node[place] (critical) [below of=waiting] {};
\node[place] (semaphore) [below of=critical] {};
\node[transition] (leave critical) [right of=critical] {}
edge [pre] (critical)
edge [post,bend right] node[auto,swap] {} (waiting)
edge [pre, bend left] (semaphore);
\node[transition] (enter critical) [left of=critical] {}
edge [post] (critical)
edge [pre, bend left] (waiting)
edge [post,bend right] (semaphore);
\end{tikzpicture}
```

TikZ



```
\tikzstyle{place}=[circle,draw=blue!50,fill=blue!20,thick,
inner sep=0pt,minimum size=6mm]
\tikzstyle{transition}=[rectangle,draw=black!50,fill=black!20,thick,
inner sep=0pt,minimum size=4mm]
\tikzstyle{pre}=[<-,shorten <=1pt,>=stealth',semithick]
\tikzstyle{post}=[->,shorten >=1pt,>=stealth',semithick]
\begin{tikzpicture}[bend angle=0]
\node[place] (waiting) {};
\node[place] (critical) [below of=waiting] {};
\node[place] (semaphore) [below of=critical] {};
\node[transition] (leave critical) [right of=critical] {}
edge [pre] (critical)
edge [post,bend right] node[auto,swap] {} (waiting)
edge [pre, bend left] (semaphore);
\node[transition] (enter critical) [left of=critical] {}
edge [post] (critical)
edge [pre, bend left] (waiting)
edge [post,bend right] (semaphore);
\end{tikzpicture}
```

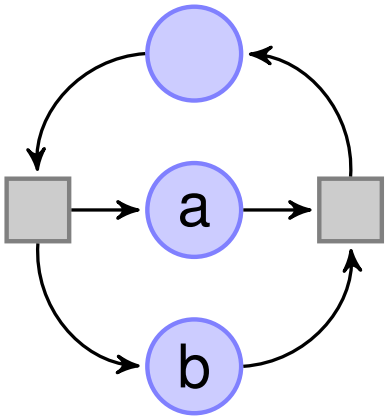
Visio



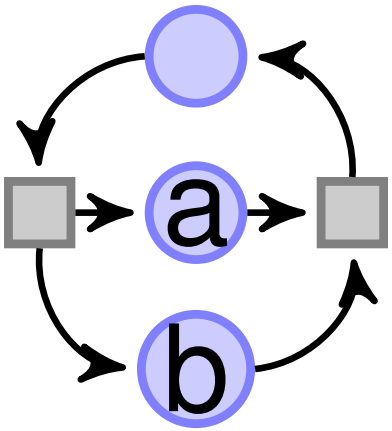
- Bilder werden gemalt
- Gutes Alignment dank Grid und Connection Points
- Bilder als PDF exportieren
- Tipps
  - Grid fest einstellen
  - Einige snaps ausschalten
  - Copy und paste special (Enhanced Metafile) in PP
  - DEMO!

Grafiken - Inhalt

Richtig gemalt



Zu klein gemalt + skaliert



- Schriftgröße und Art sollte nicht zu stark vom Text abweichen
  - Einfach bei TikZ
  - Kleinere Schrift und serifenlos u.U. okay
- Am besten: Bilder nicht skalieren; gleich in richtiger Größe malen
  - Sonst stimmt Strichstärke und Schriftgröße nicht
  - Unstimmiges Gesamtbild
- Keine zu dicken Strichstärken, nicht grundlos variieren

Grafiken - Inhalt

- Grafik sollte gut gelayoutet und durchdacht sein
  - Einige Stunden pro Grafik einrechnen!
- Grafik muss Verständlich ein oder mehrere Dinge vermitteln
  - klingt trivial, wird aber oft nicht bedacht
  - Beispiel: Grafik wird “einfach so” eingebaut um mehr Figures zu haben. . .
- Grafiken zeigen oft Beispielausprägungen
  - Gute Beispiele wählen!
  - Nicht den trivialen Fall!
  - Kein uninteressanter Randfall!
- Gute Bilder machen oft den Unterschied! (auch in der Note ;)

Outline

1 Wissenschaftliches Schreiben

2 Inhalt

3 Sprache

4 Layout

5 LaTeX

6 Grafiken, Tabellen, Listings

7 Bibliographie und Zitieren

Zitate

- Zitate zeigen an, dass man Inhalt aus anderen Dokumenten übernommen hat
- Zitate müssen immer durch [xyz] (`\cite`) kenntlich gemacht werden
- Gute Quellen und korrektes Zitieren ist A und O in der Wissenschaft
  - Fließt stark in Bewertung ein
  - Unsauberes Zitieren kann extrem problematisch werden!
    $\Rightarrow$ siehe Guttenberg, Schavan und co.
  - Gute Quellen: Bücher, Paper, usw.
  - Schlechte Quellen: Webseiten, vor allem Wikipedia!
- Bibliographie am Ende des Dokuments listet alle Quellen auf
- Achtung: Alle Quellen in der Bibliographie müssen auch irgendwo Zitiert werden!
  $\Rightarrow$ Niemals `\nocite` verwenden!

Zitiermöglichkeiten

- Wörtlich
  - Text einrücken oder Anführungszeichen
  - Eher unüblich, selten
- Sinngemäß
  - Inhalt in eigenen Worten ausdrücken
  - `\cite` am Ende von Satz oder Absatz
- Bilder
  - Bild Sinngemäß nachzeichnen
  - `\cite` am Ende der Bildunterschrift

Nutzen von Zitaten

- In der Related Work Section sind Zitate absolut zentral
  - Saubere related work deckt alle Arbeiten im Gebiet der Arbeit ab
  - Stellt Großteil der Bibliographie
    $\Rightarrow$ Bibliographie ist “Aushängeschild” der Arbeit
- In Einleitung/Motivation kann durch Zitate die eigene Argumentation unterstützt werden
- Überall sonst wo Gedanken übernommen werden
  - Seltener im Hauptteil bei Informatik, öfter bei anderen Fachrichtungen
  - Vorsicht: Nur Gedanken übernehmen alleine reicht nicht!
    $\Rightarrow$ Eigene Arbeit muss erkennbar sein, sonst negativ

BibTeX

- Bib-Datei enthält Einträge die über einen eindeutigen Key (frei wählbar) referenziert werden können:

```
@article{chien2001xml,
  title={{XML} document versioning},
  author={Chien, S.Y. and Tsotras, V. J. and Zaniolo, C.},
  journal={SIGMOD Record},
  volume={30},
  number={3},
  year={2001}
}
```

- Fast alle Onlinearchieve (z.B. DBLP) und Suchmaschinen (z.B. Google Scholar) erlauben Export
  $\Rightarrow$ Selten selbst schreiben!
- DEMO!
  - Aber: Oft Nachbearbeitung nötig
- LaTeX muss öfter ausgeführt werden bis neue Einträge übernommen werden!

Zitieren mit LaTeX und BibTeX

- Bibliographie wird in LaTeX normal mit BibTeX verwaltet
- BibTeX Bibliographie in `xyz.bib` File
- Einbinden und anzeigen mit `\bibliography{xyz}`
  - Nur Einträge die zitiert werden werden angezeigt
  - Eine Bib-Datei kann für mehrere Dokumente verwendet werden
- Zitieren mit `\cite{KEY}`. Wird z.B. zu `[1]`
- Mehrere Quellen mit `\cite{KEY1,KEY2,KEY3}`. Wird z.B. zu `[1,5,17]`

BibTeX Nachbearbeiten

- Bibliographie ist “Aushängeschild” der Arbeit
- Sauberes Quellenverzeichnis ist sehr wichtig!
- Fehler: Jede Referenz überprüfen! Oft enthalten generierte Einträge Fehler
  $\Rightarrow$ Fehlender Autor, falscher Autor (z.B. `C. Science`), fehlende Felder
  - Konferenz oder Journal o.ä. muss vorhanden sein!
  - Jahr muss vorhanden sein
  - Bei Journal auch Volume und Issue wichtig
  - Verleger und weitere Details werden oft weggelassen

BibTeX Nachbearbeiten

Groß- und Kleinschreibung

- Standardmäßig schreibt BibTeX alles klein bis auf Satzanfang
- Das ist falsch bei Abkürzungen wie z.B. XML
- Umschließen mit {XML} zwingt BibTeX die Schreibweise nicht zu verändern

BibTeX Nachbearbeiten

Konferenz und Journalnamen

- Es gibt viele Möglichkeiten eine Konferenz zu beschreiben, z.B:
  - Proceedings of the 23th International Conference on Management of Data (SIGMOD)
  - Proceedings of the ACM Conference on Management of Data (ACM SIGMOD)
  - Proc. Int. Conf. Management of Data (SIGMOD 2007)
  - International Conference on Managament of Data
  - SIGMOD
  - ACM SIGMOD
  - SIGMOD'07
- Nur einer dieser Styles sollte für alle Referenzen verwendet werden!
  - Macht Arbeit, aber lohnt sich :)