Shuffle off to. . . Milledgeville

Selected Fun Problems of the ACM Programming Contest (Proseminar)

Matthias Reitinger

Technische Universität München
Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme

3. Dezember 2007

Das Szenario

- Softwarefirma betreibt Bürogebäude in mehreren Städten

Das Szenario

- Softwarefirma betreibt Bürogebäude in mehreren Städten
- Büroräume werden aufgrund einer internen Umstrukturierung unter den Angestellten neu verteilt

Das Szenario

- Softwarefirma betreibt Bürogebäude in mehreren Städten
- Büroräume werden aufgrund einer internen Umstrukturierung unter den Angestellten neu verteilt
- Angestellte wechseln das Gebäude dabei allerdings nicht

Das Szenario

- Softwarefirma betreibt Bürogebäude in mehreren Städten
- Büroräume werden aufgrund einer internen Umstrukturierung unter den Angestellten neu verteilt
- Angestellte wechseln das Gebäude dabei allerdings nicht
- Problem:

Das Szenario

- Softwarefirma betreibt Bürogebäude in mehreren Städten
- Büroräume werden aufgrund einer internen Umstrukturierung unter den Angestellten neu verteilt
- Angestellte wechseln das Gebäude dabei allerdings nicht
- Problem:
  - Ein Angestellter kann sein neues Büro erst dann beziehen, wenn es bereits geräumt wurde.

Das Szenario

- Softwarefirma betreibt Bürogebäude in mehreren Städten
- Büroräume werden aufgrund einer internen Umstrukturierung unter den Angestellten neu verteilt
- Angestellte wechseln das Gebäude dabei allerdings nicht
- Problem:
  - Ein Angestellter kann sein neues Büro erst dann beziehen, wenn es bereits geräumt wurde.
  - Dadurch können Abhängigkeitsketten entstehen!

Das Szenario

- Softwarefirma betreibt Bürogebäude in mehreren Städten
- Büroräume werden aufgrund einer internen Umstrukturierung unter den Angestellten neu verteilt
- Angestellte wechseln das Gebäude dabei allerdings nicht
- Problem:
  - Ein Angestellter kann sein neues Büro erst dann beziehen, wenn es bereits geräumt wurde.
  - Dadurch können Abhängigkeitsketten entstehen!
- Zielsetzung: Erkennen solcher Abhängigkeitsketten

Ein- und Ausgabe

Eingabe

- n Büroräume

Ausgabe

Ein- und Ausgabe

Eingabe

- n Büroräume
- n Angestellte

Ausgabe

Ein- und Ausgabe

Eingabe

- n Büroräume
- n Angestellte
- Bürobelegung vor der Umstrukturierung

Ausgabe

Ein- und Ausgabe

Eingabe

- n Büroräume
- n Angestellte
- Bürobelegung vor der Umstrukturierung
- Bürobelegung nach der Umstrukturierung

Ausgabe

Ein- und Ausgabe

Eingabe

- n Büroräume
- n Angestellte
- Bürobelegung vor der Umstrukturierung
- Bürobelegung nach der Umstrukturierung

Ausgabe

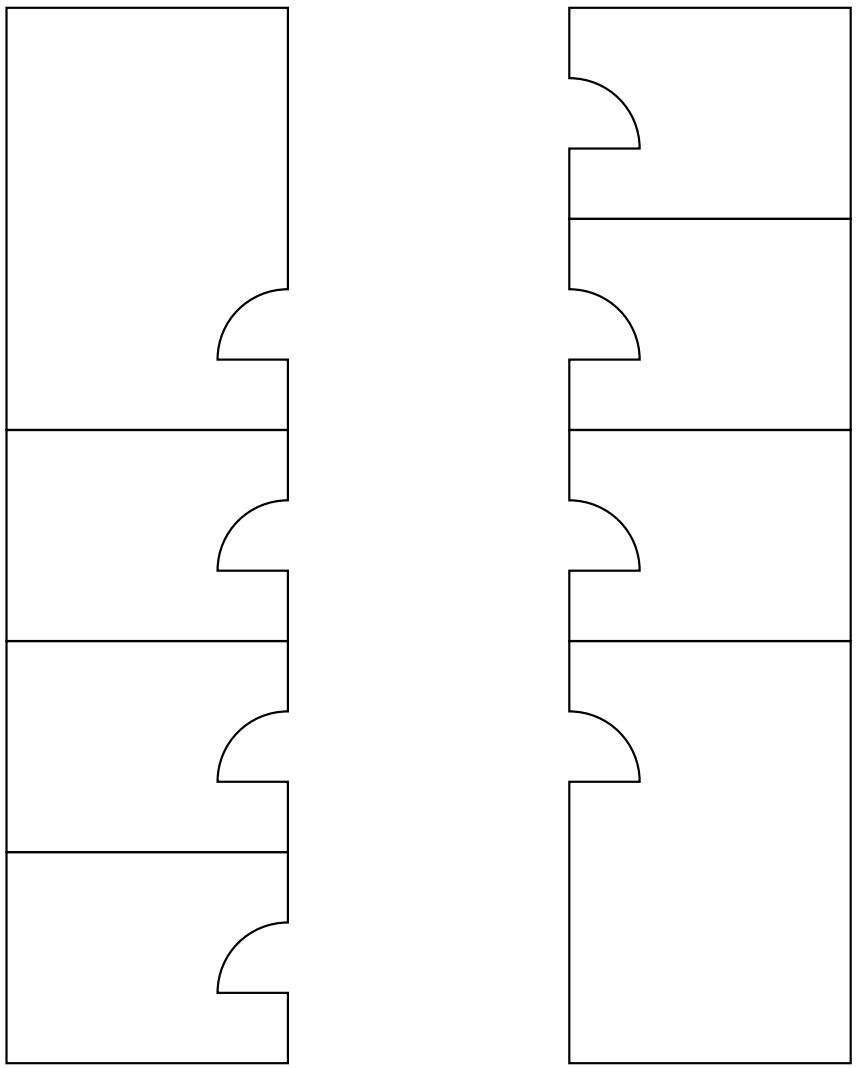
- Längste Abhängigkeitskette(n) ($A \rightarrow B \rightarrow C \rightarrow A$)

Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```

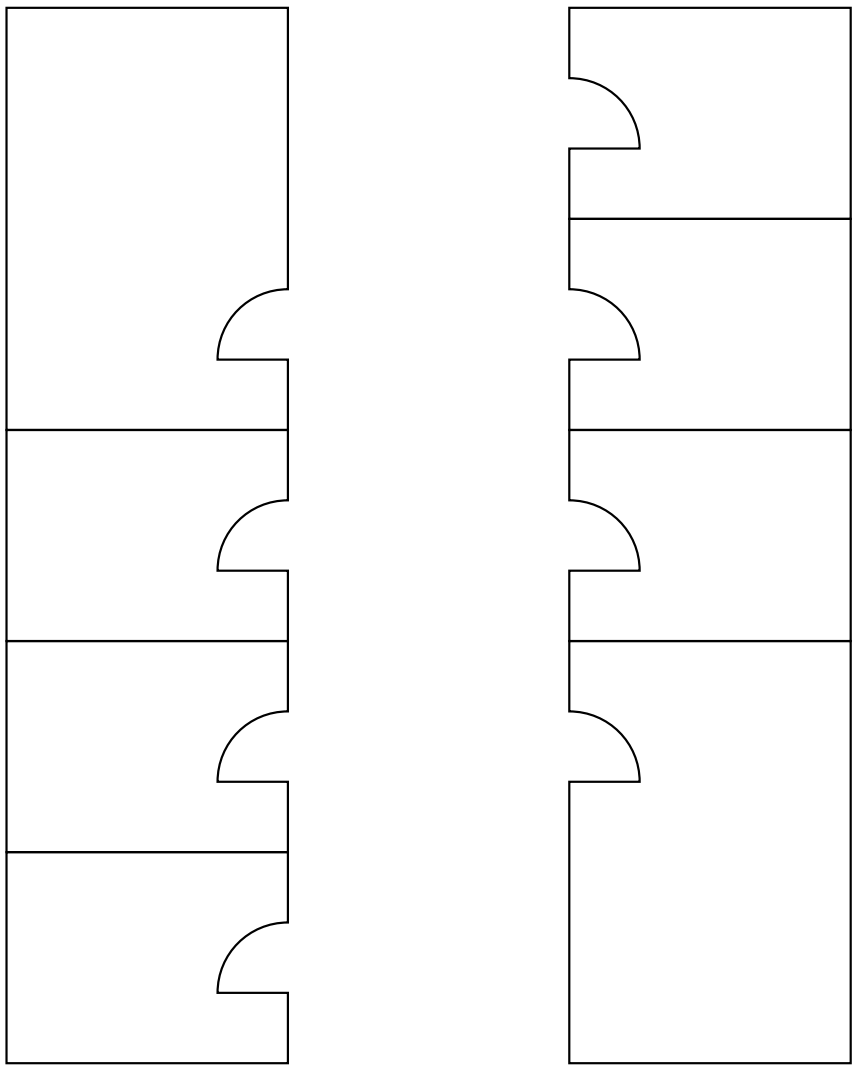
Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



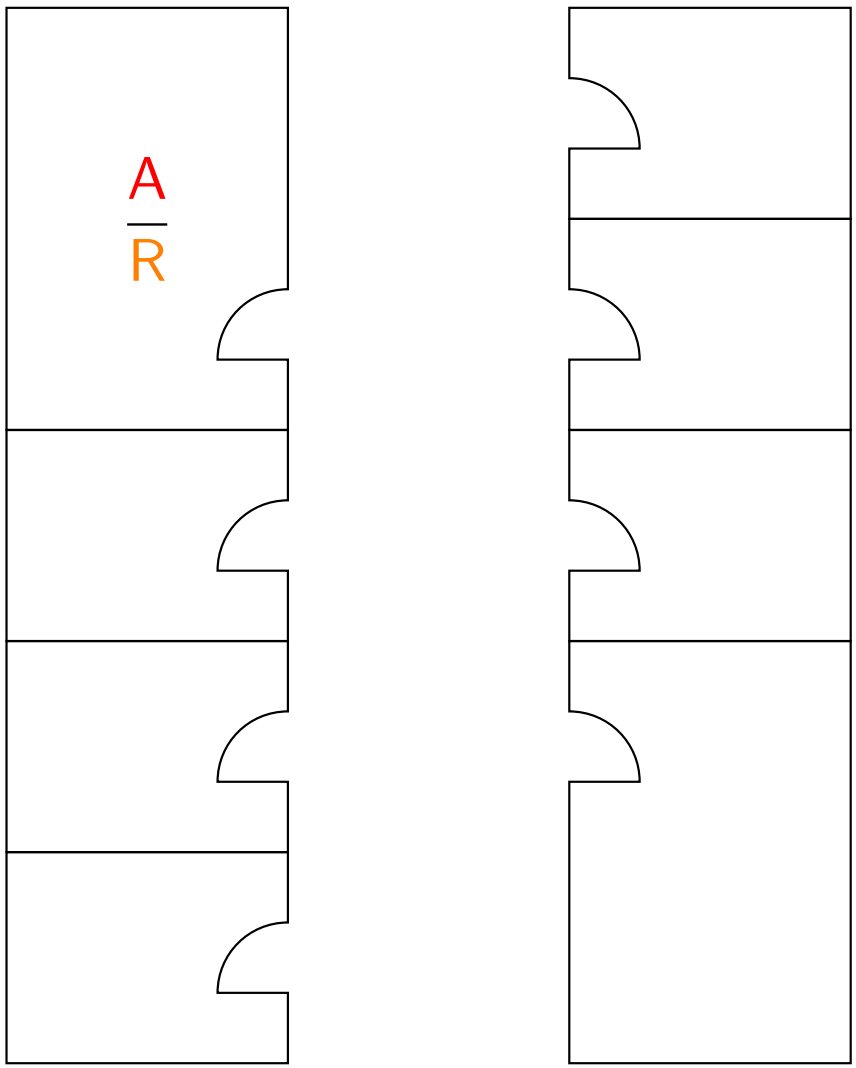
Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



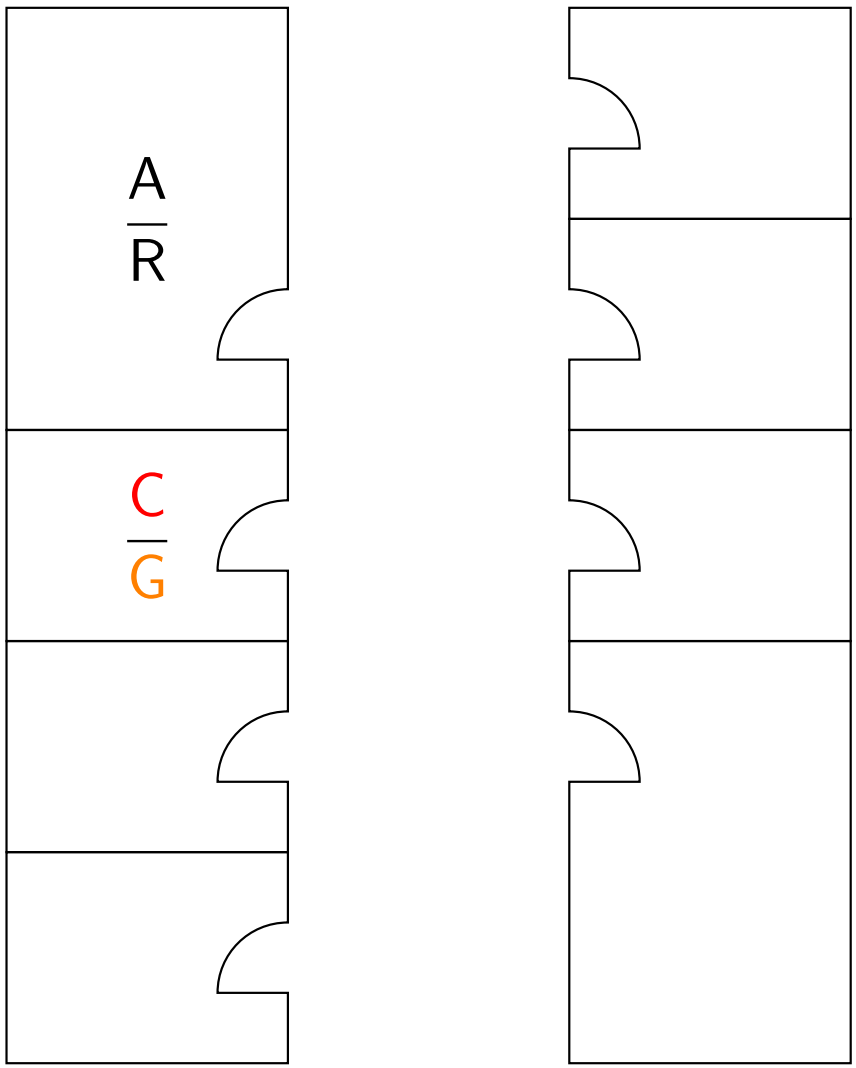
Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



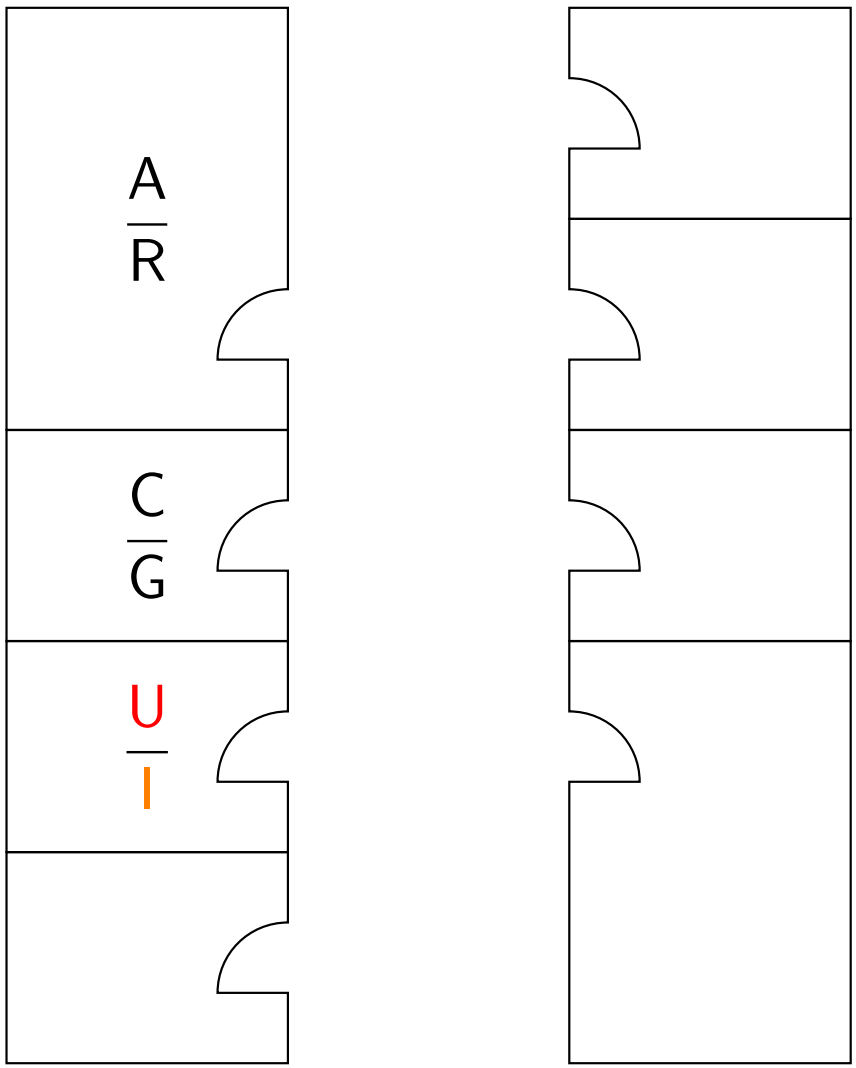
Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



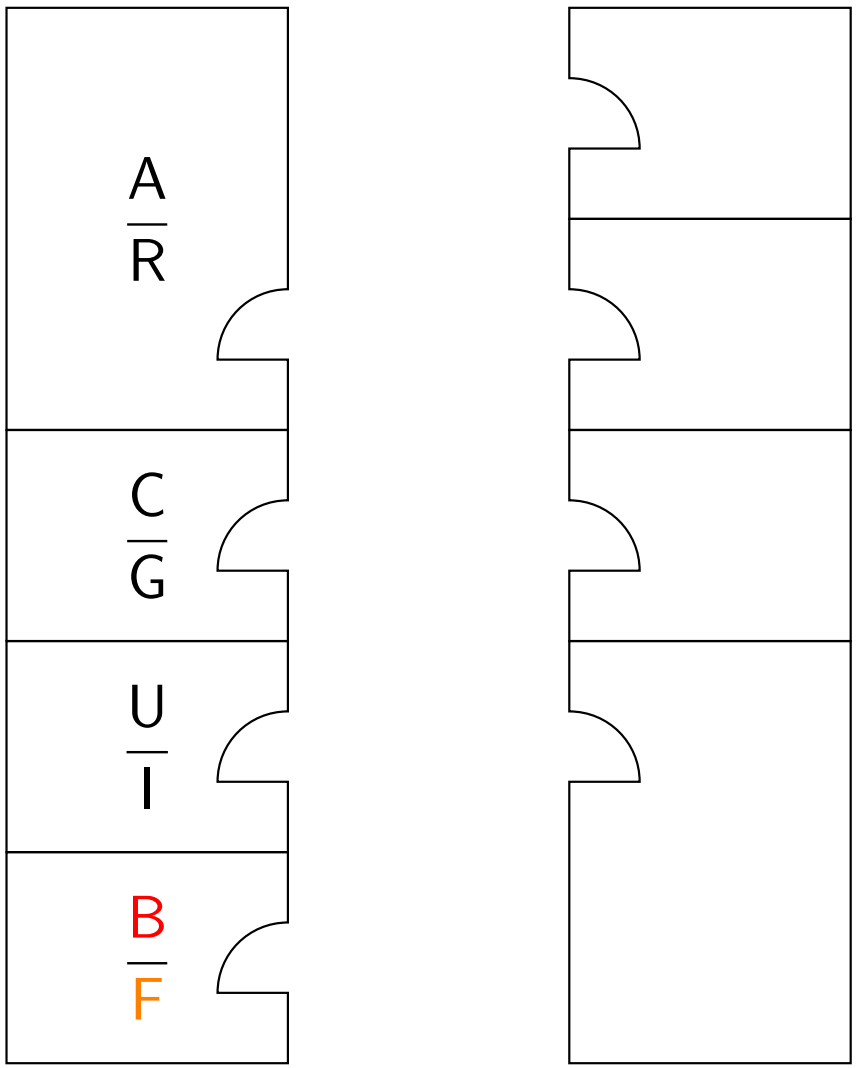
Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



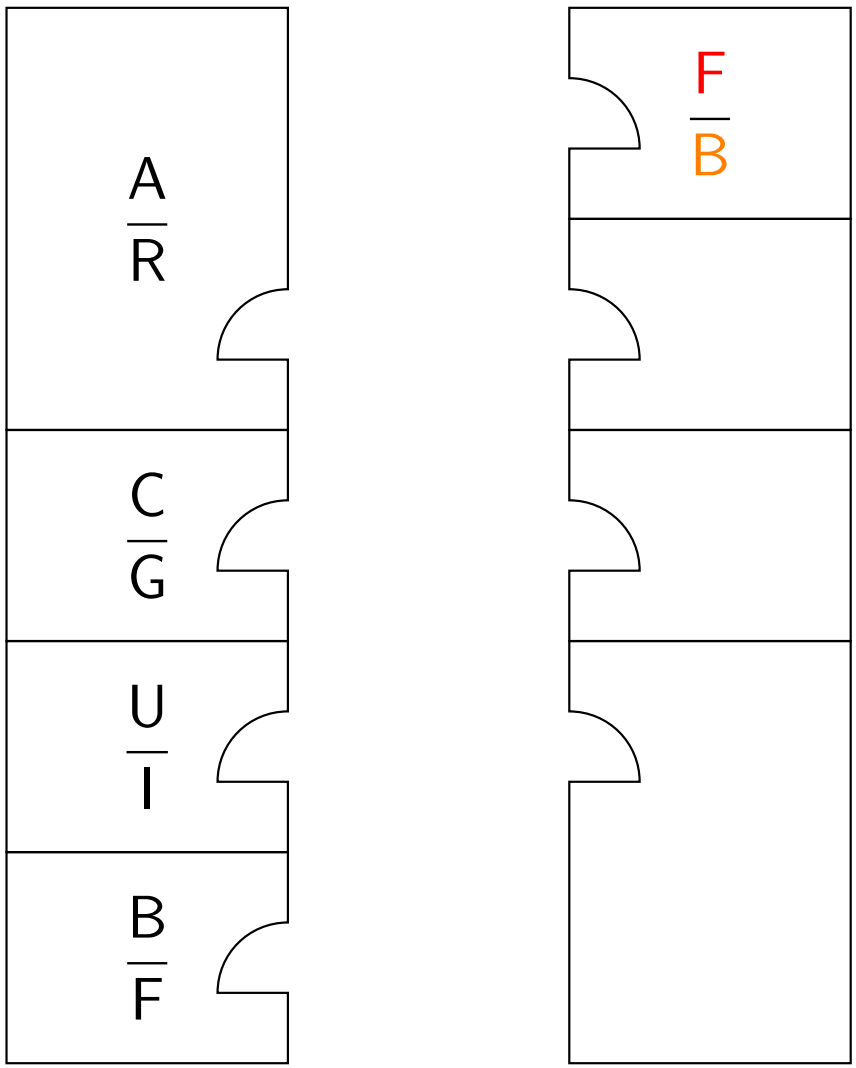
Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



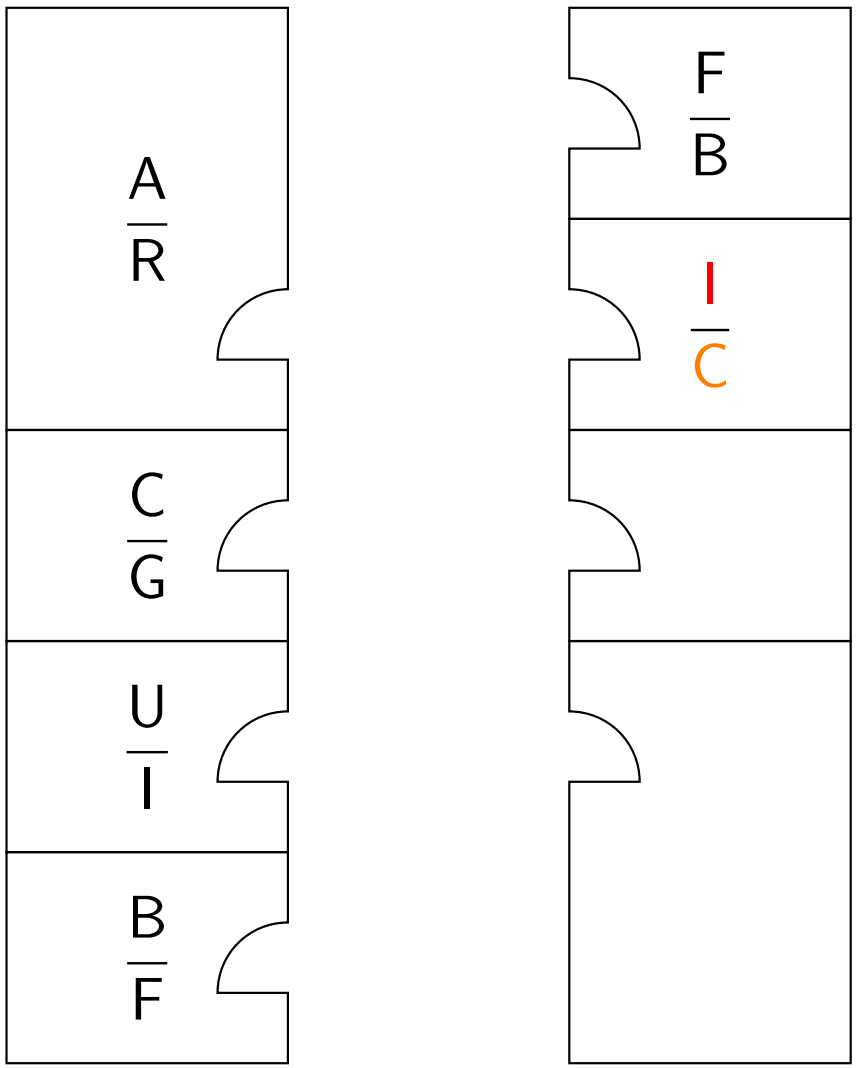
Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



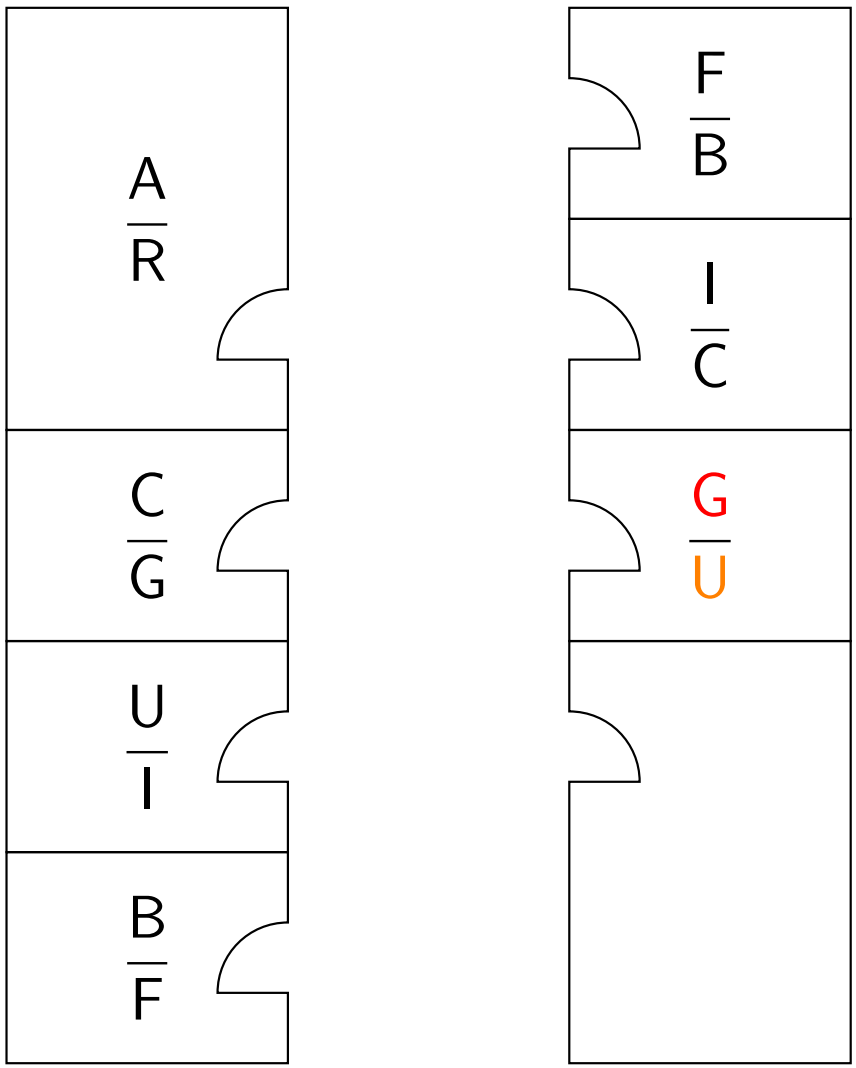
Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



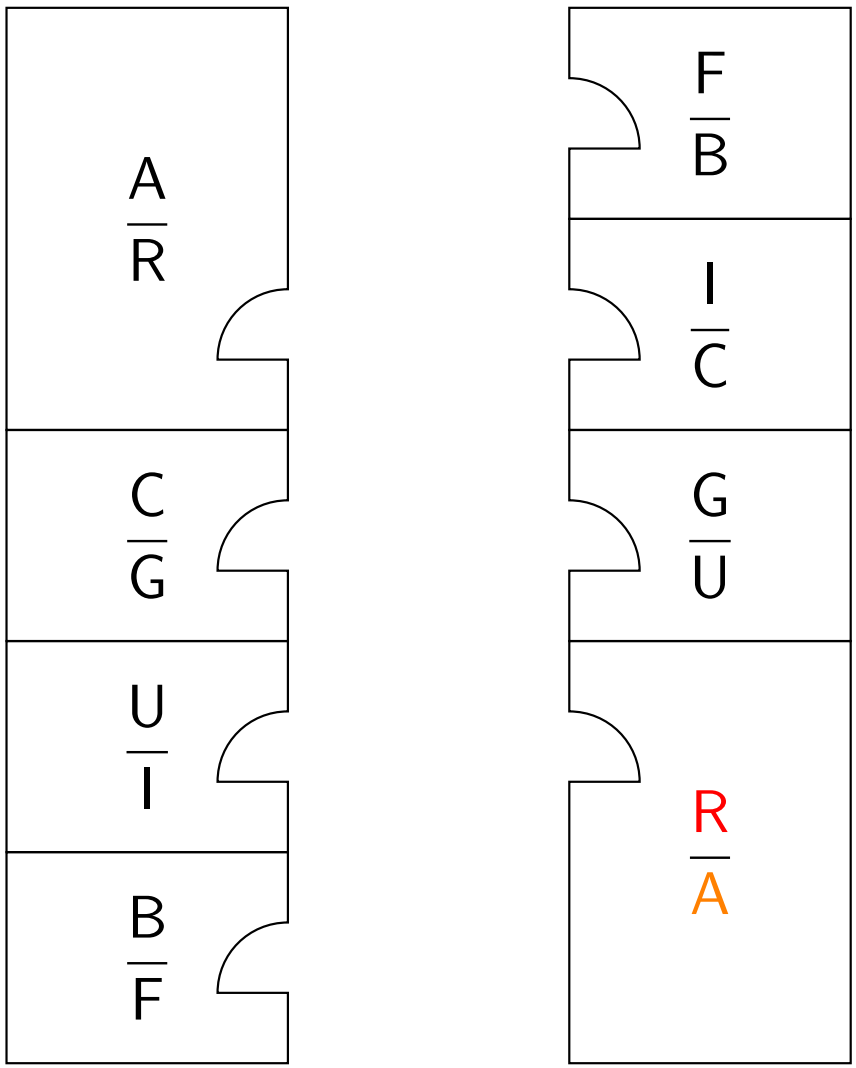
Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



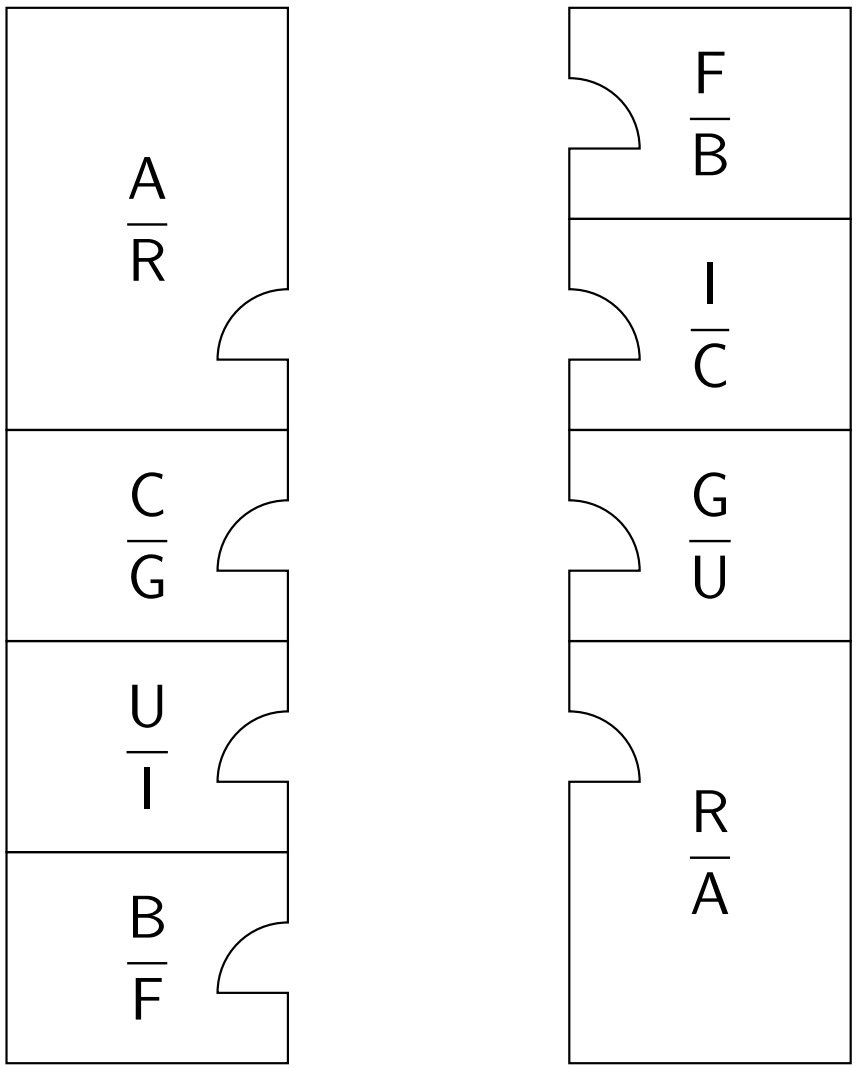
Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



Eingabeformat

```
8
Milledgeville
11 A R
21 C G
22 U I
23 B F
31 F B
32 I C
33 G U
41 R A
0
```



Ausgabeformat

```
Milledgeville
The longest dependency chain length is 4.
Chain 1: C I U G
```

Vorgaben

- Jede Kette mit dem Angestellten beginnen, der bei alphabetischer Sortierung als erster kommt

Ausgabeformat

```
Milledgeville
The longest dependency chain length is 4.
Chain 1: C I U G
```

Vorgaben

- Jede Kette mit dem Angestellten beginnen, der bei alphabetischer Sortierung als erster kommt
- Ketten untereinander alphabetisch sortiert ausgeben

Interpretation als graphentheoretisches Problem

Transformation

- Jeder Knoten repräsentiert ein Büro

Interpretation als graphentheoretisches Problem

Transformation

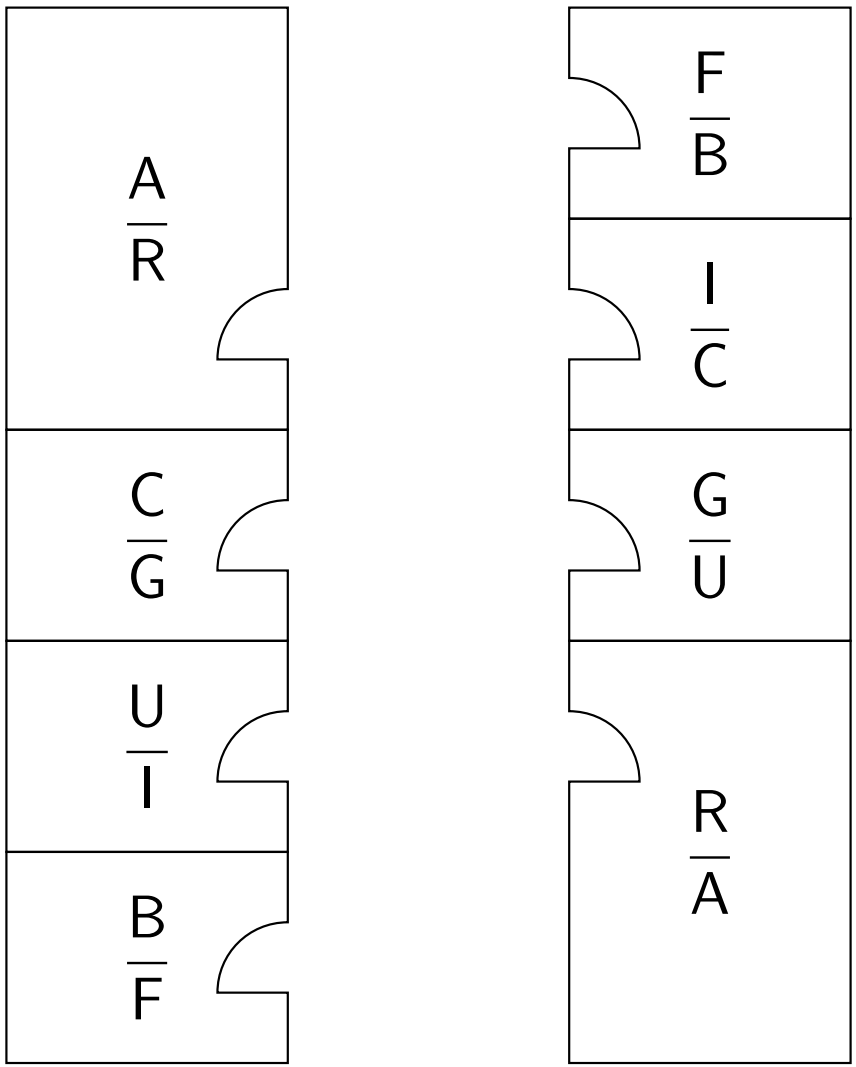
- Jeder Knoten repräsentiert ein Büro
- Jeder Knoten speichert Namen des Angestellten, der dem entsprechenden Büro zugeteilt war

Interpretation als graphentheoretisches Problem

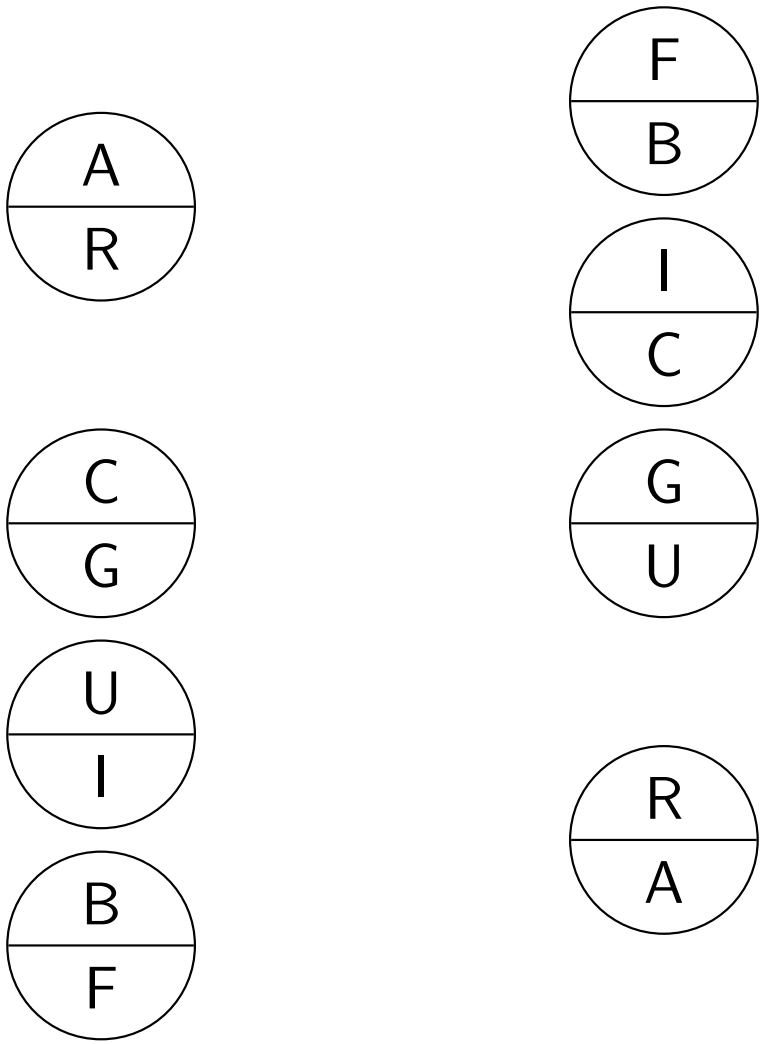
Transformation

- Jeder Knoten repräsentiert ein Büro
- Jeder Knoten speichert Namen des Angestellten, der dem entsprechenden Büro zugeteilt war
- Gerichtete Kante zwischen zwei Knoten: Angestellter zieht von Startknoten in Zielknoten um

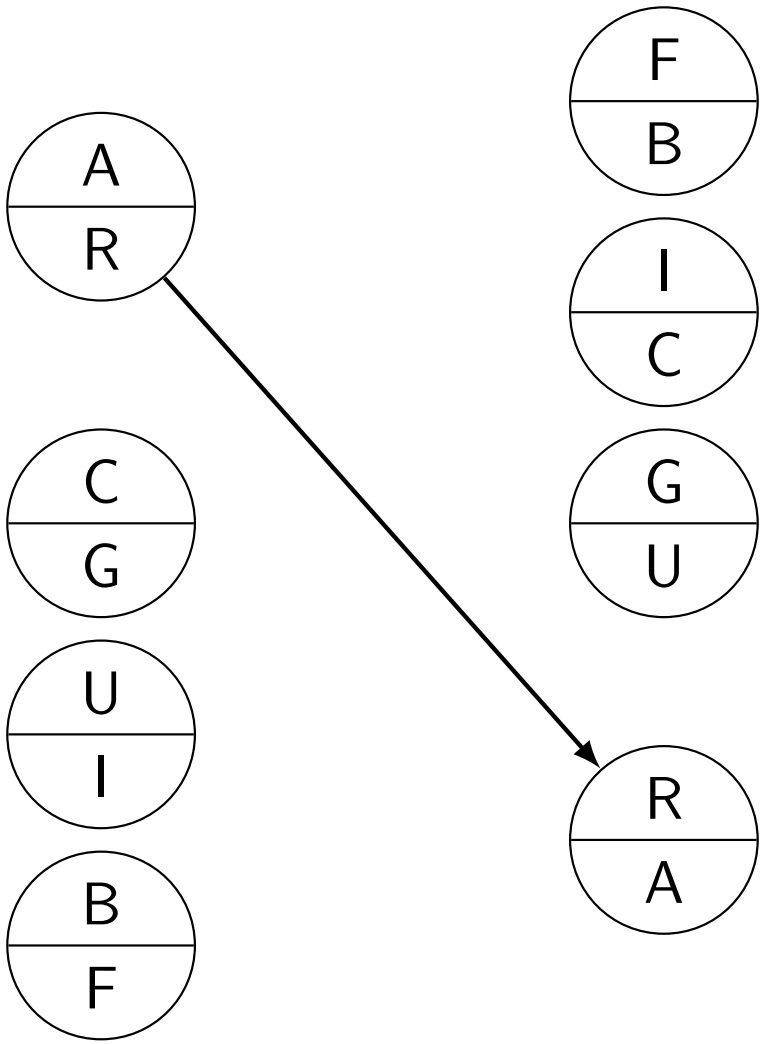
Beispiel: Aufbau des Graphen



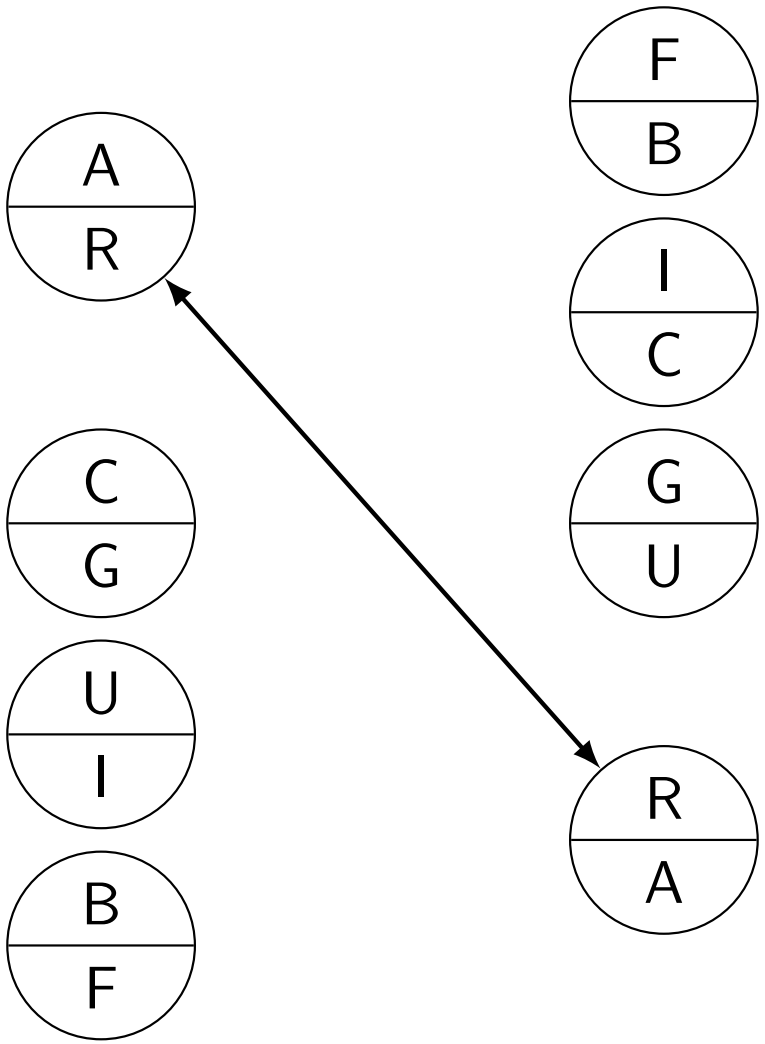
Beispiel: Aufbau des Graphen



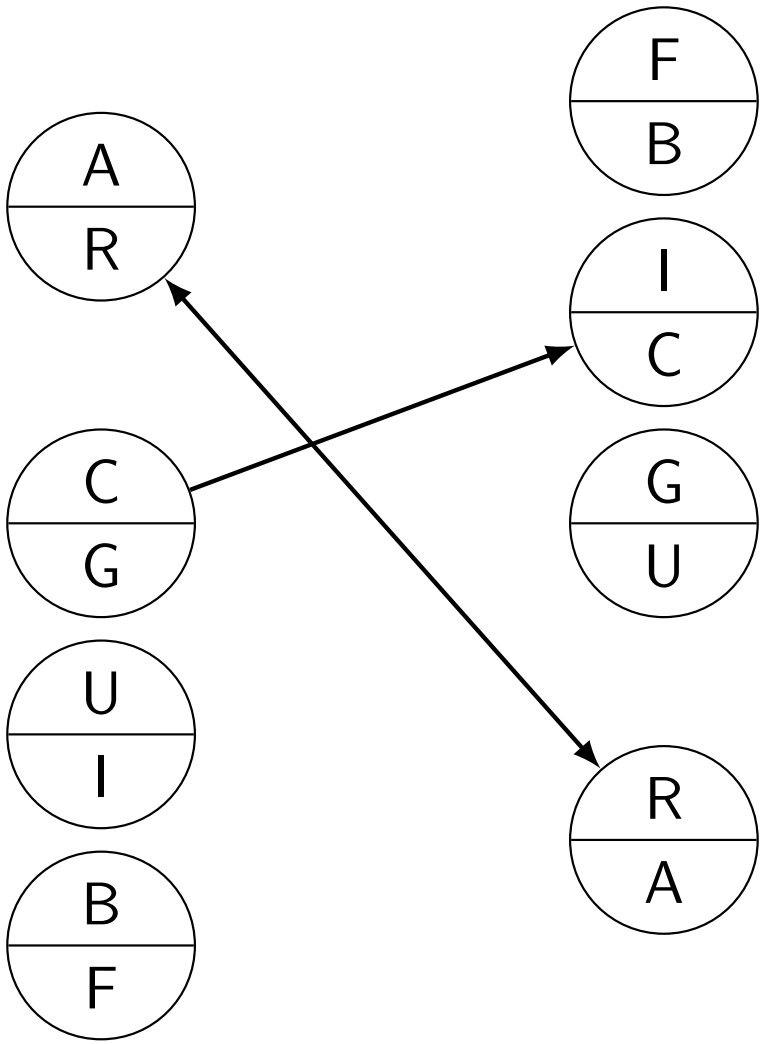
Beispiel: Aufbau des Graphen



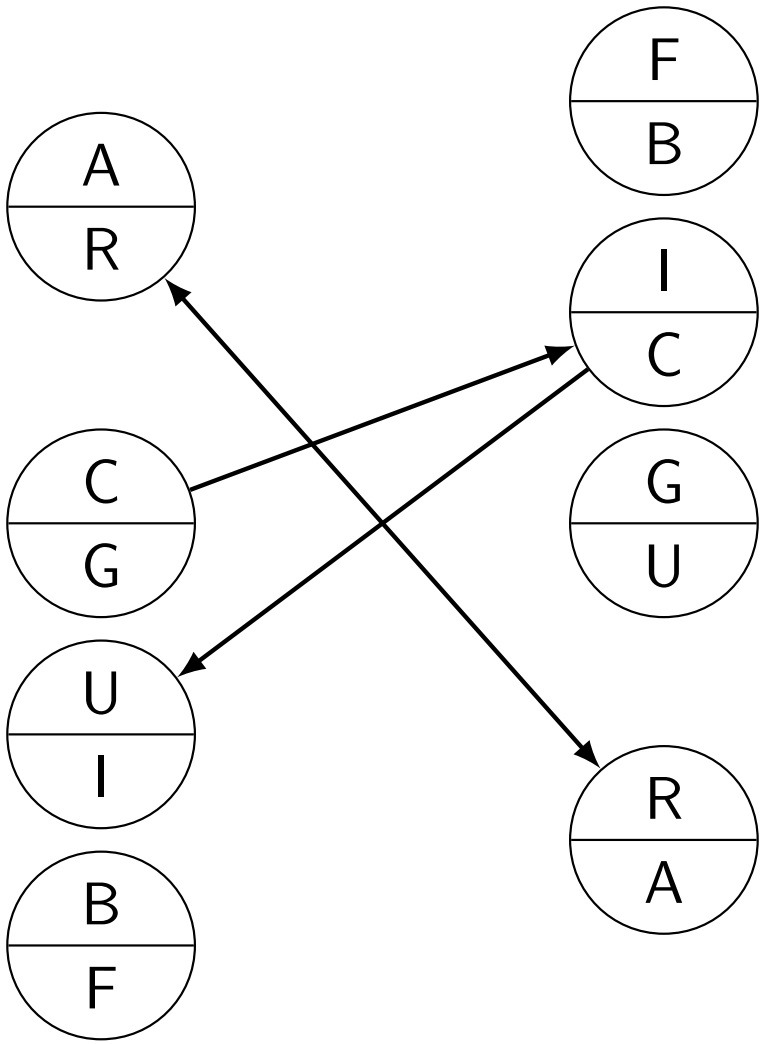
Beispiel: Aufbau des Graphen



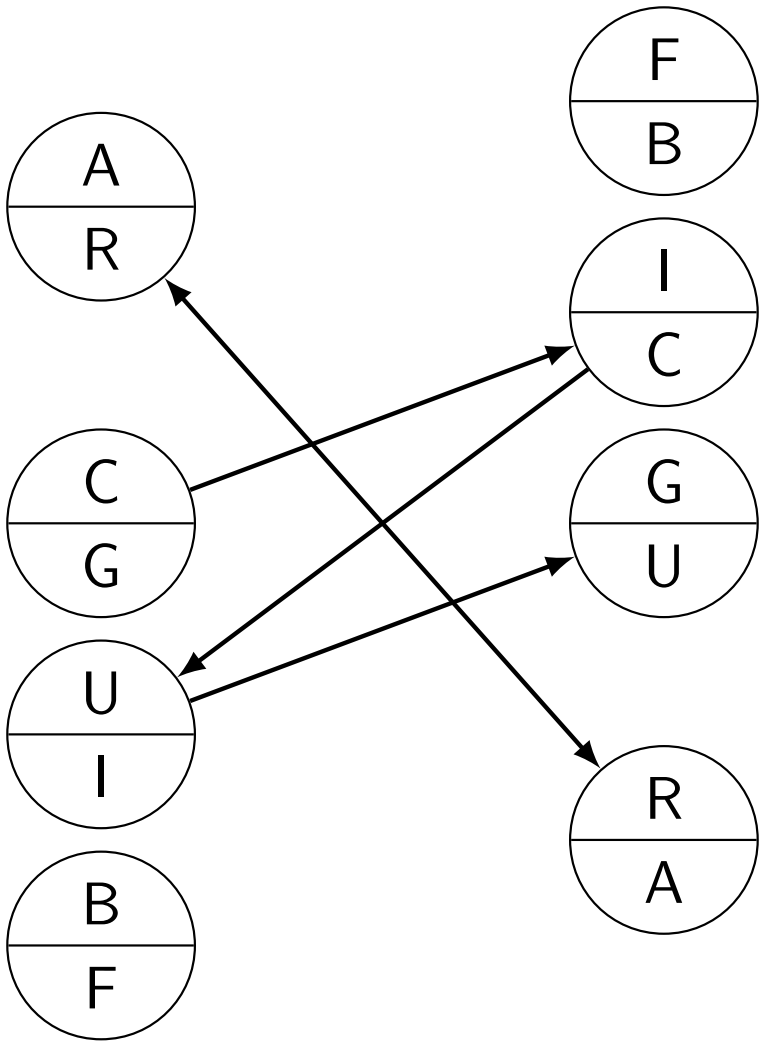
Beispiel: Aufbau des Graphen



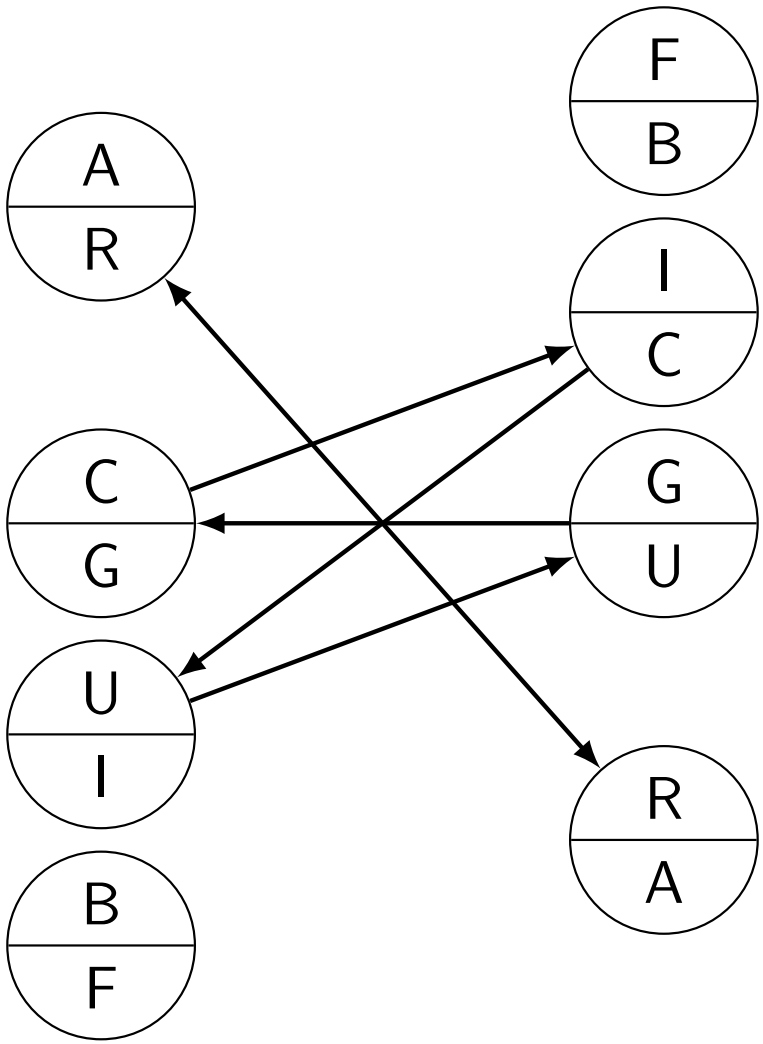
Beispiel: Aufbau des Graphen



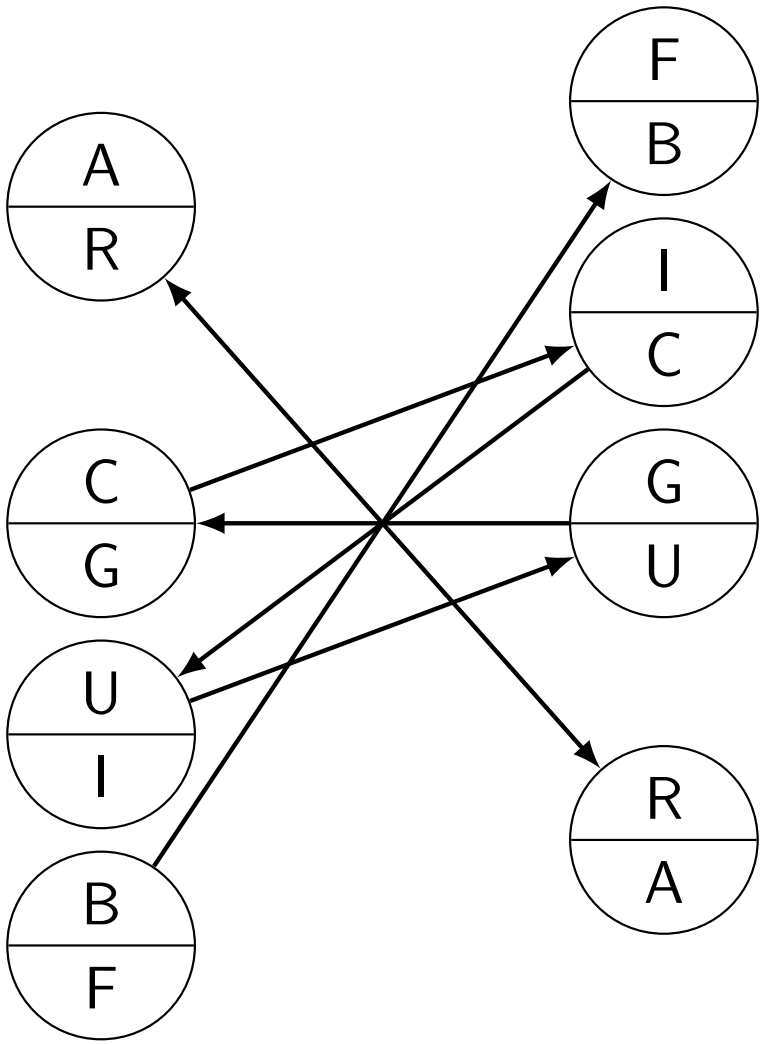
Beispiel: Aufbau des Graphen



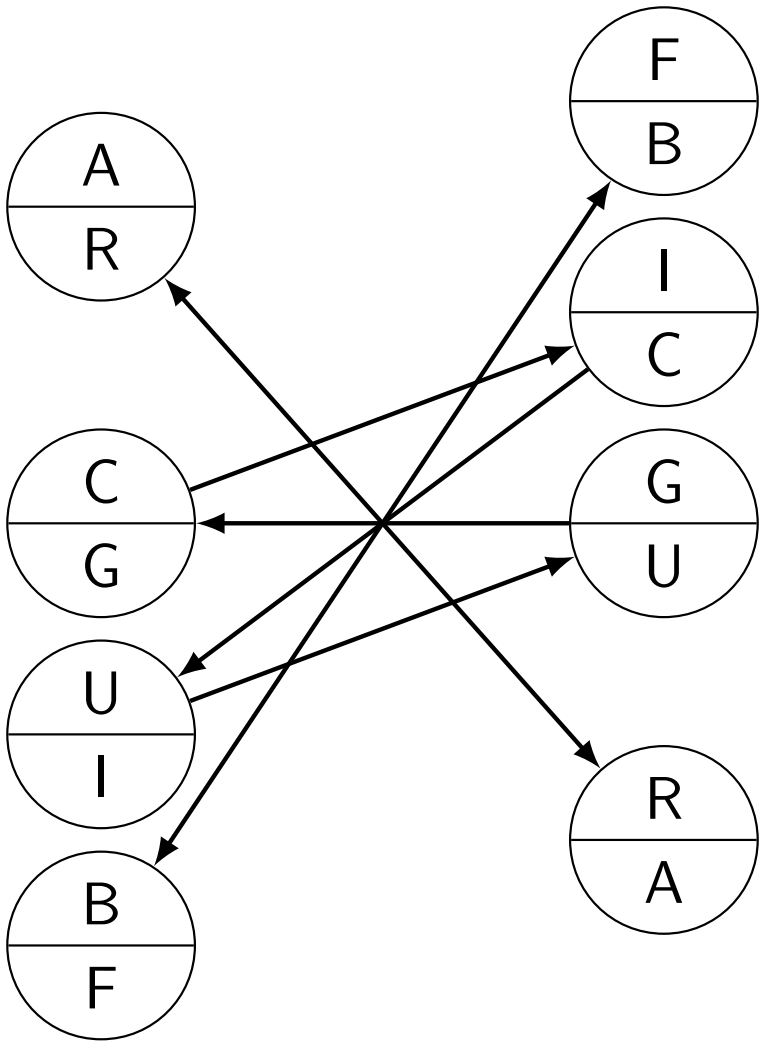
Beispiel: Aufbau des Graphen



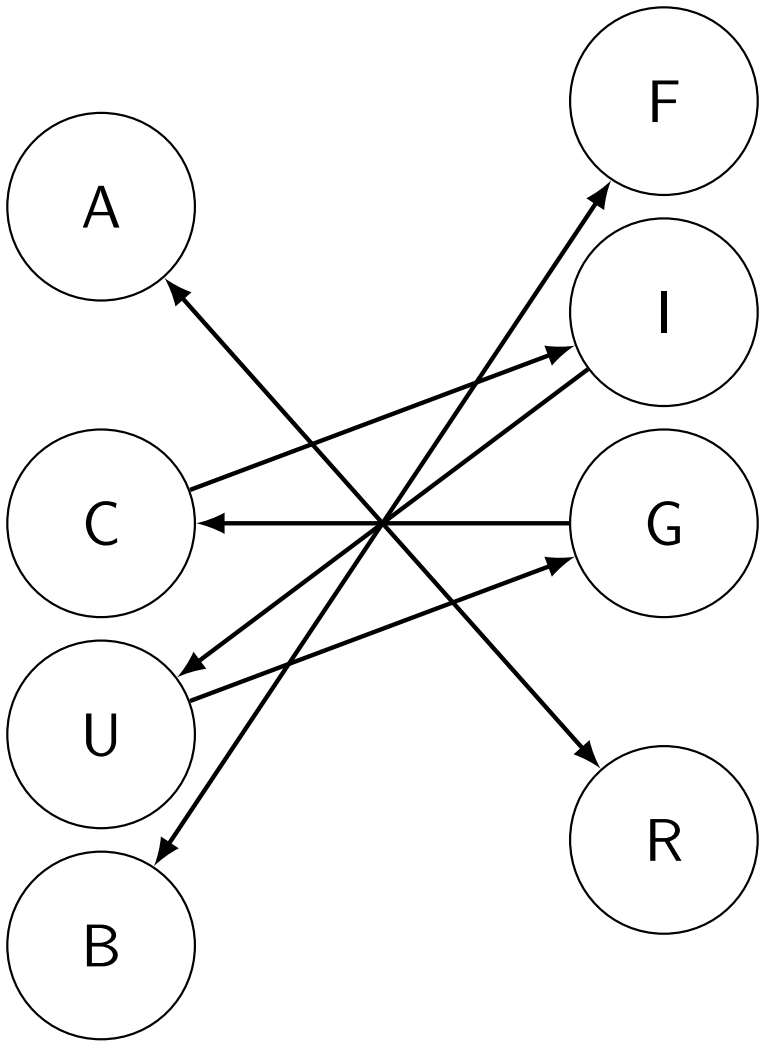
Beispiel: Aufbau des Graphen



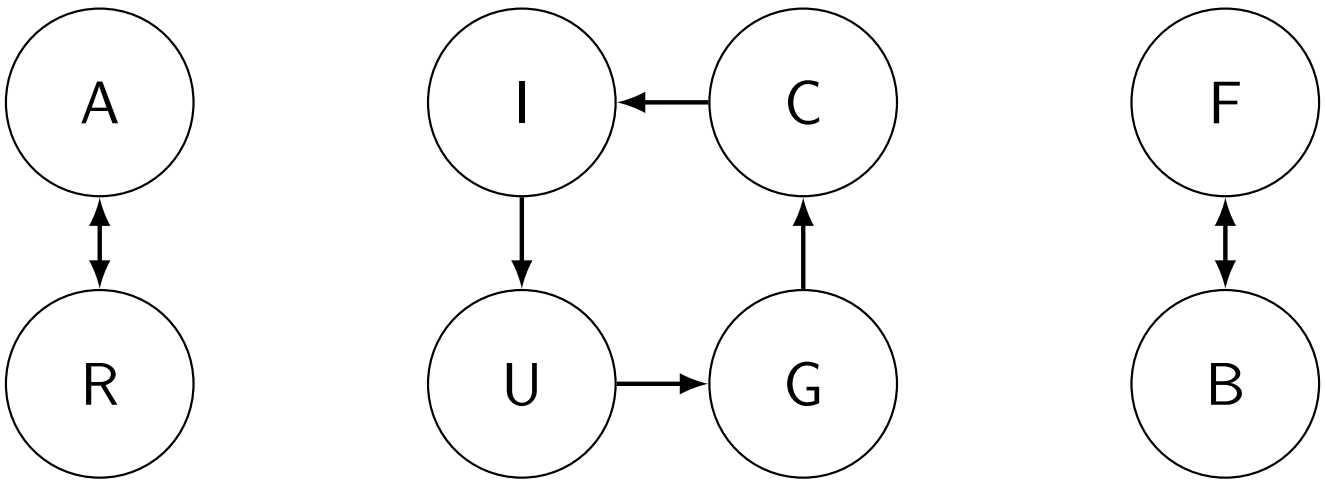
Beispiel: Aufbau des Graphen



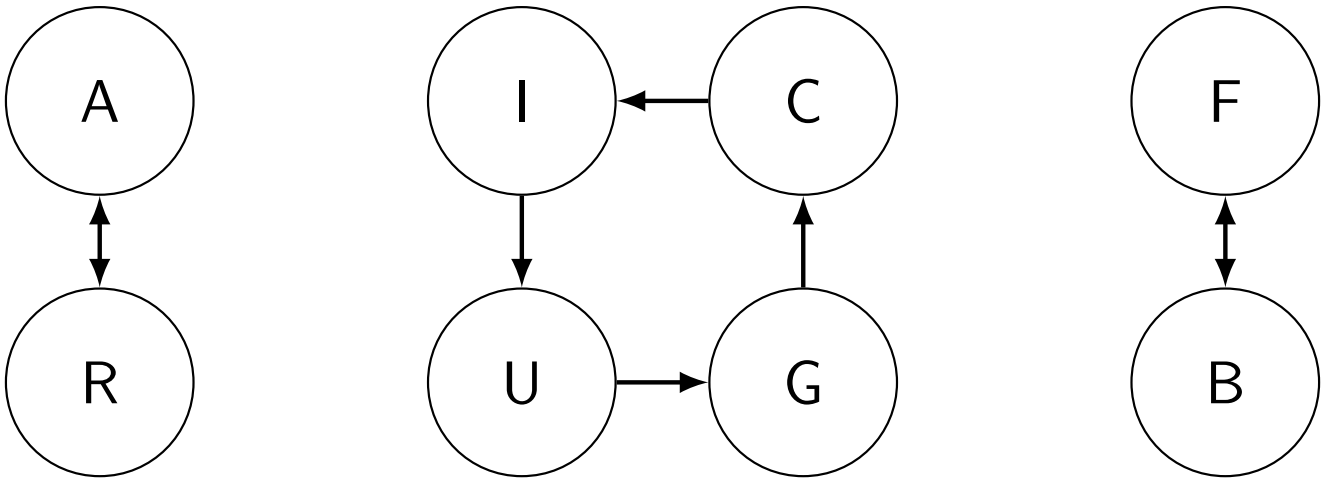
Beispiel: Aufbau des Graphen



Beispiel: Aufbau des Graphen

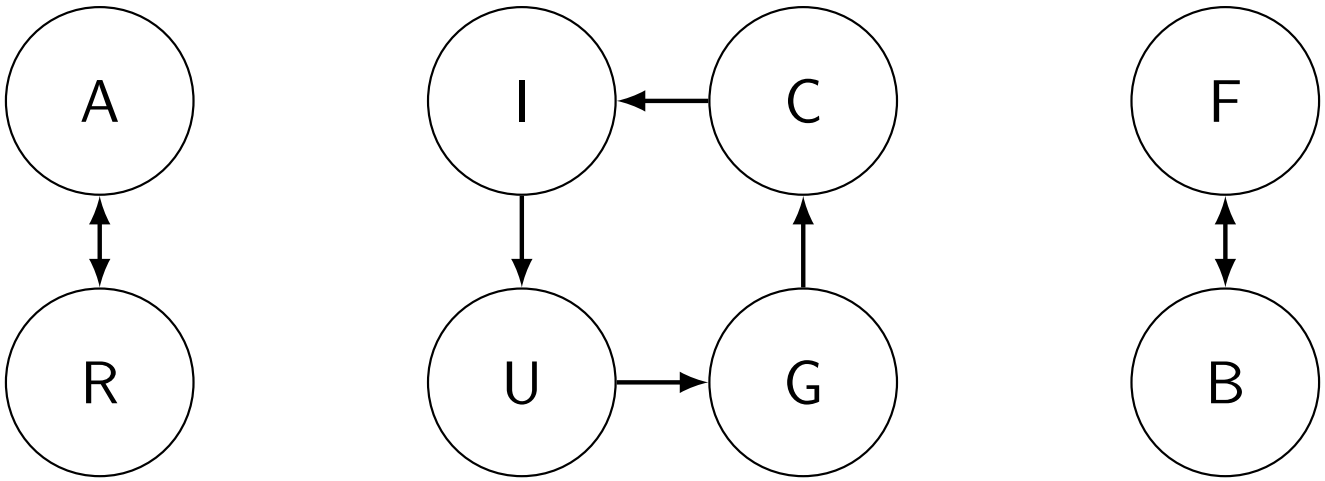


Besondere Eigenschaften des Graphen



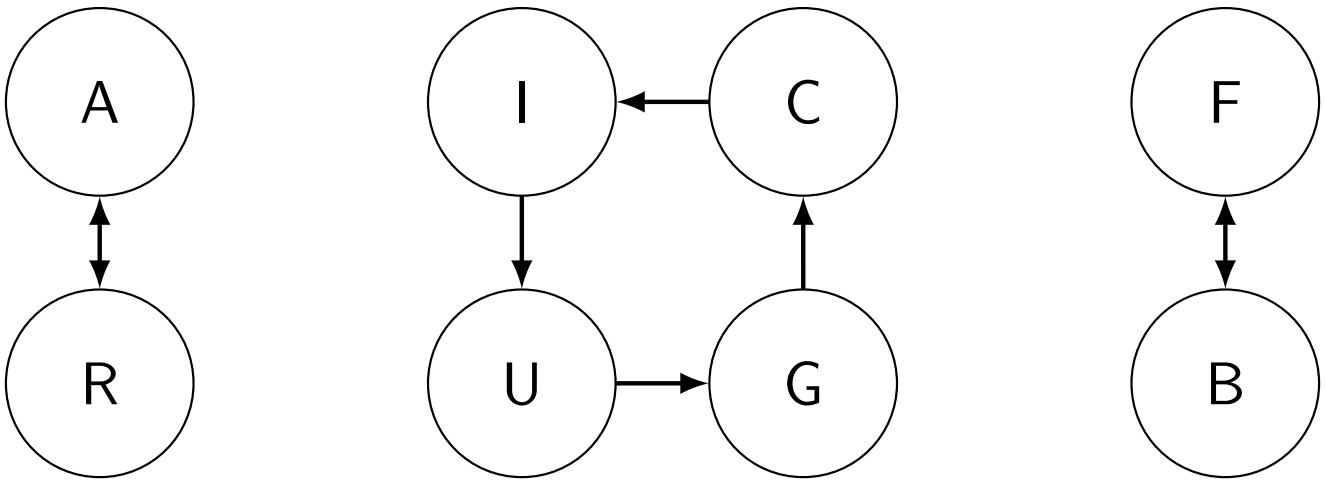
- Jeder Knoten besitzt genau eine eingehende und eine ausgehende Kante

Besondere Eigenschaften des Graphen



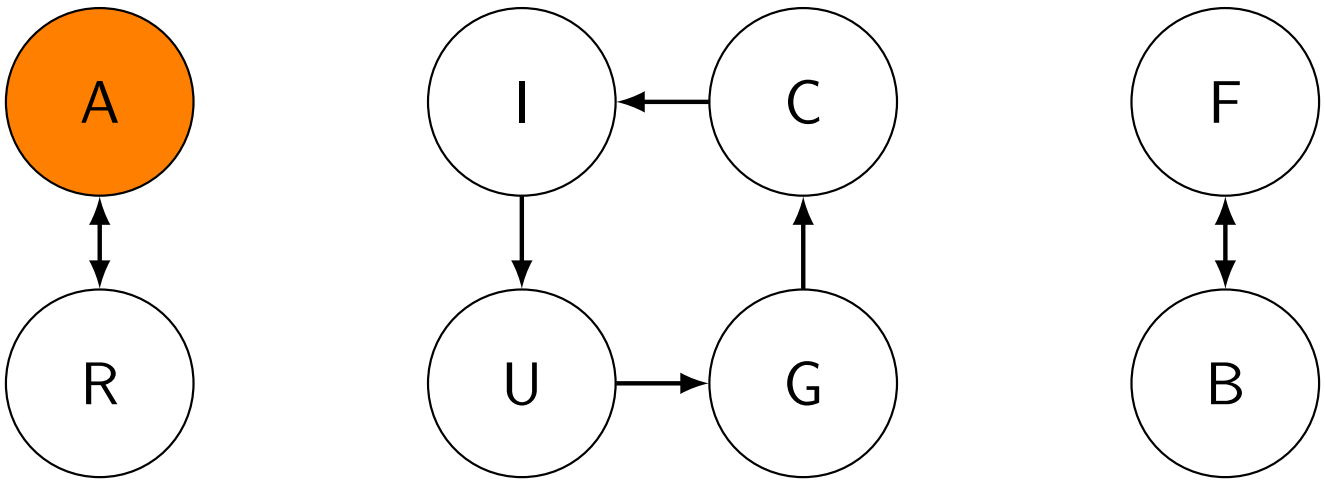
- Jeder Knoten besitzt genau eine eingehende und eine ausgehende Kante
- Konsequenz: Graph besteht nur aus Kreisen, was die Bestimmung aller Abhängigkeitsketten besonders einfach macht

Algorithmus



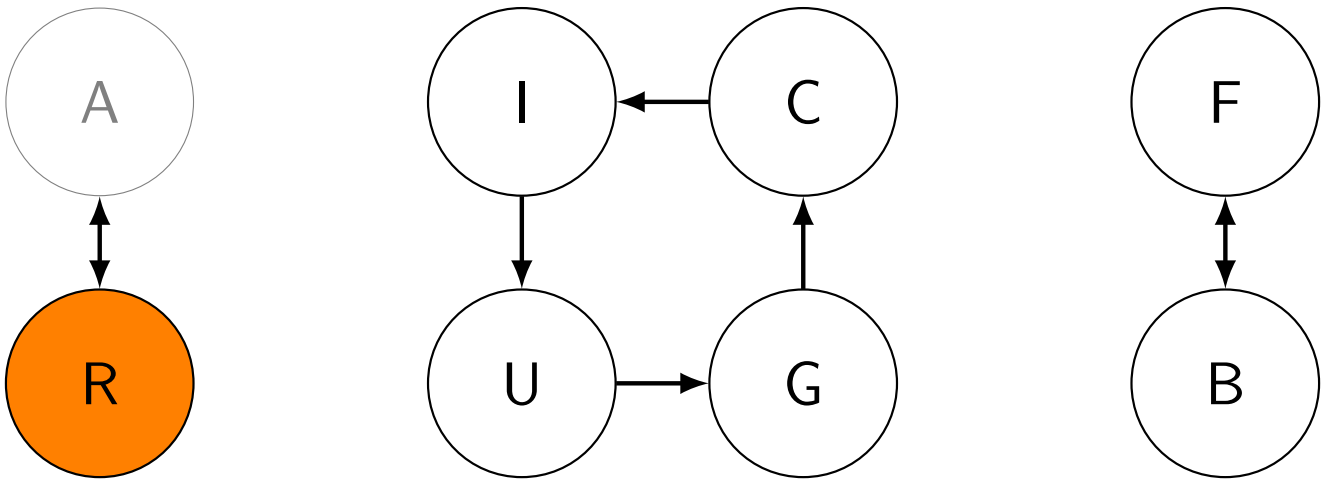
- Aktuelle Kette:
- Längste Kette(n):

Algorithmus



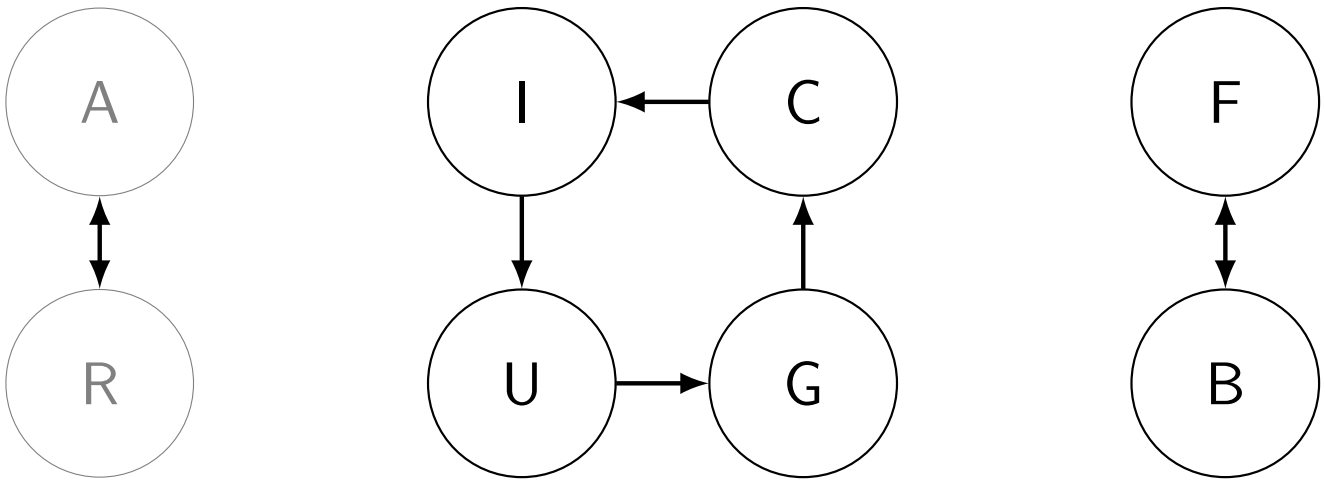
- Aktuelle Kette: $\langle A$
- Längste Kette(n):

Algorithmus



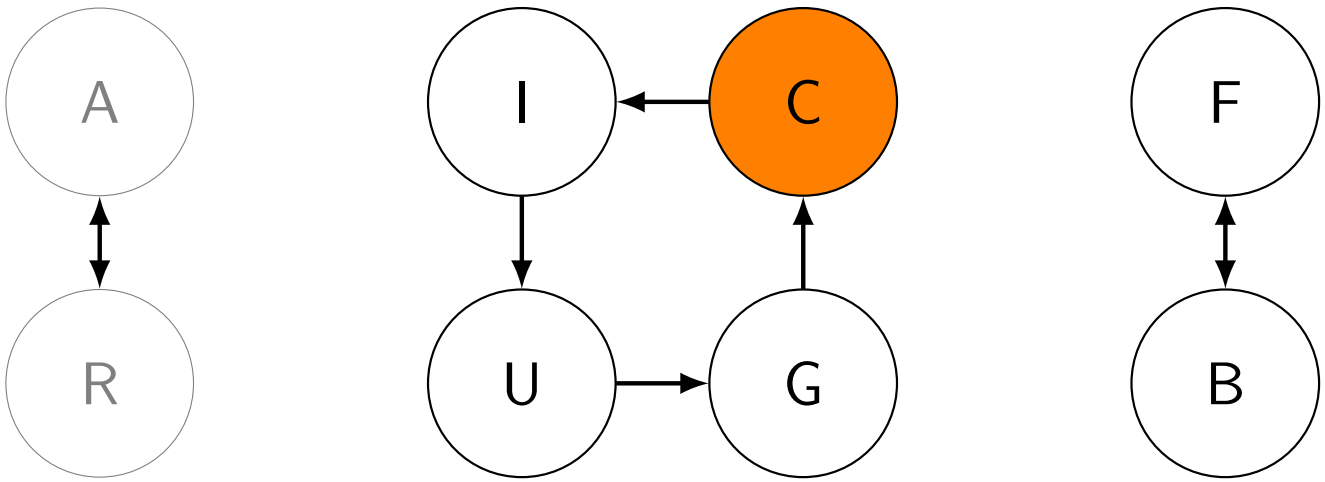
- Aktuelle Kette: $\langle A, R$
- Längste Kette(n):

Algorithmus



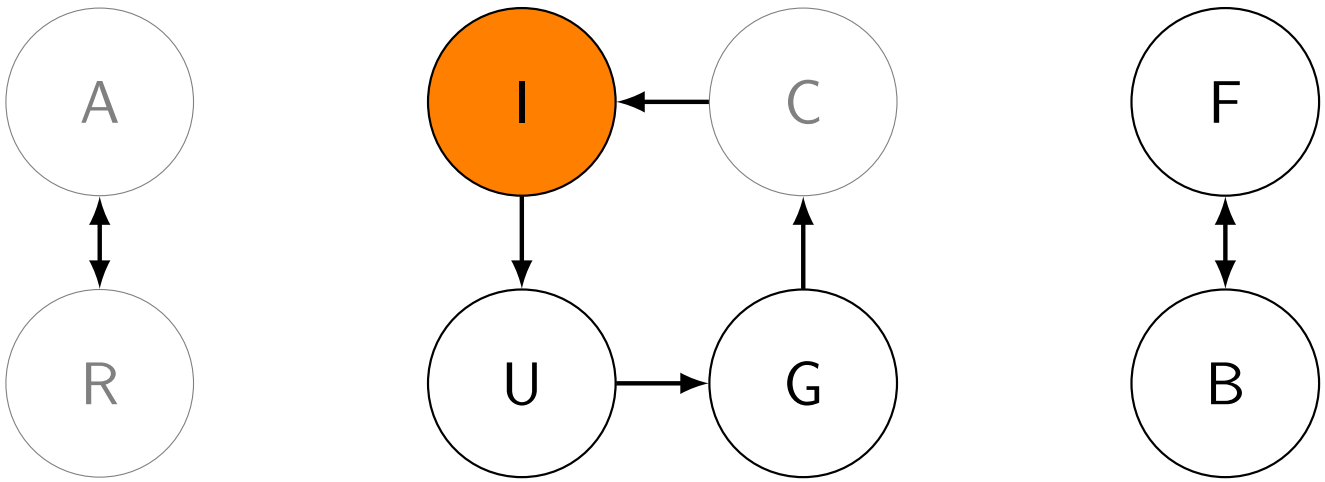
- Aktuelle Kette:
- Längste Kette(n): $\langle A, R \rangle$

Algorithmus



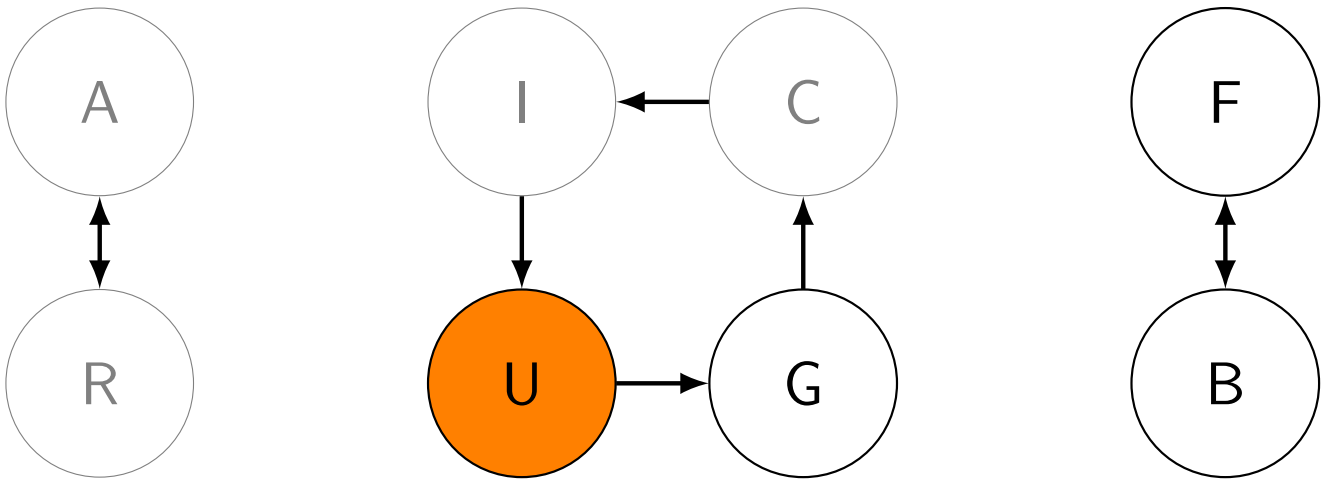
- Aktuelle Kette: $\langle C$
- Längste Kette(n): $\langle A, R \rangle$

Algorithmus



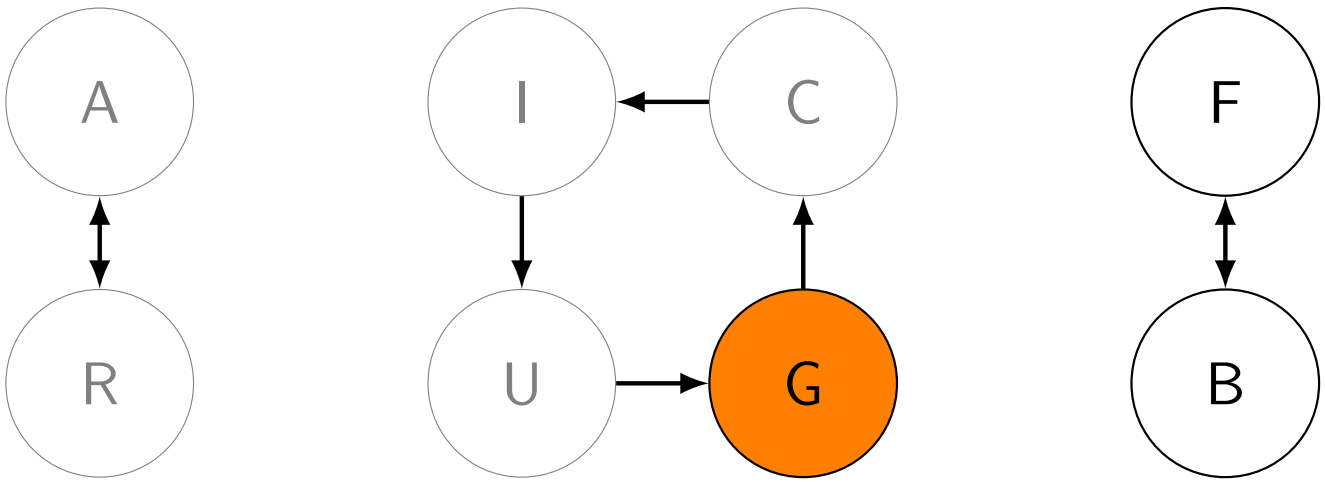
- Aktuelle Kette: $\langle C, I$
- Längste Kette(n): $\langle A, R \rangle$

Algorithmus



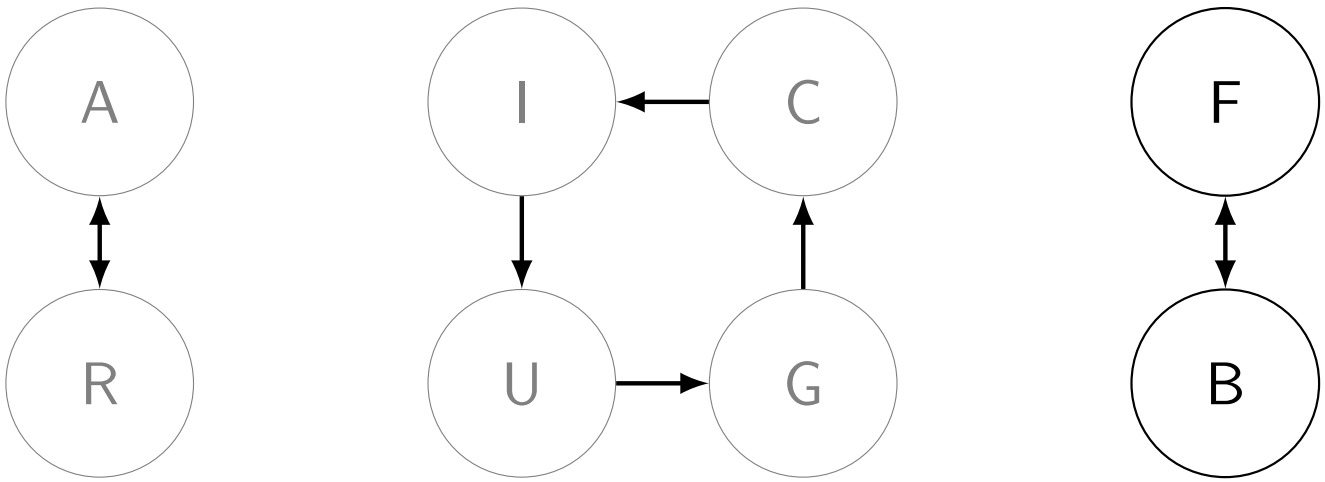
- Aktuelle Kette: $\langle C, I, U$
- Längste Kette(n): $\langle A, R \rangle$

Algorithmus



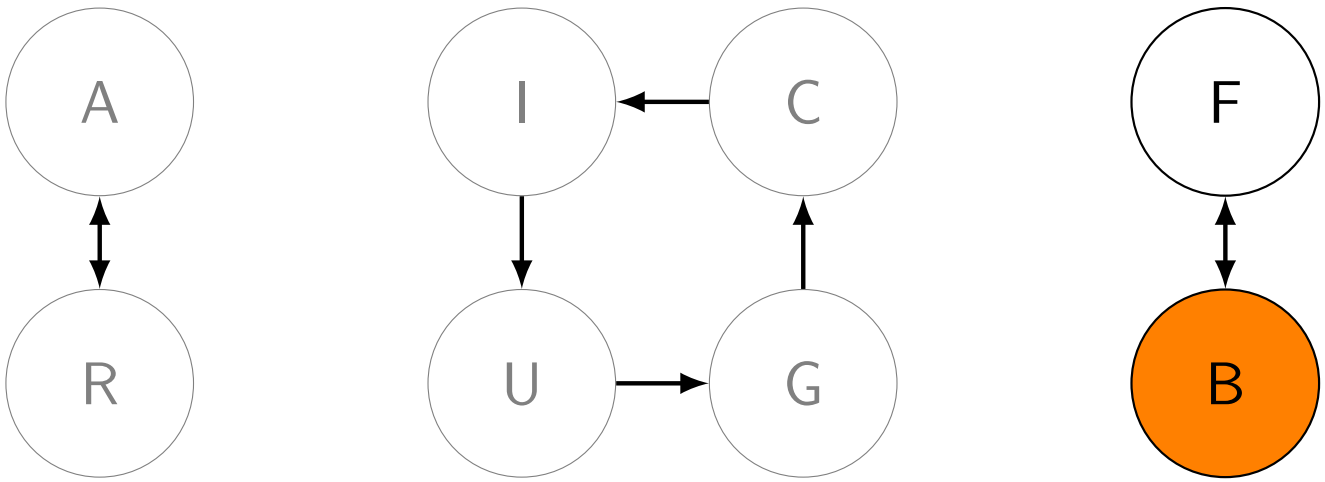
- Aktuelle Kette: $\langle C, I, U, G$
- Längste Kette(n): $\langle A, R \rangle$

Algorithmus



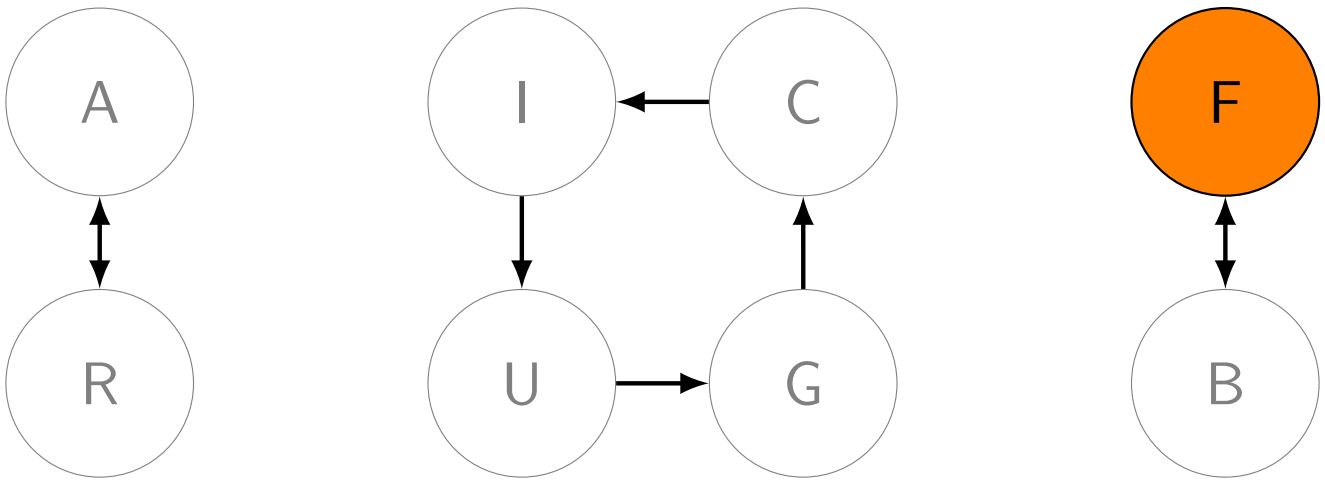
- Aktuelle Kette:
- Längste Kette(n): $\langle C, I, U, G \rangle$

Algorithmus



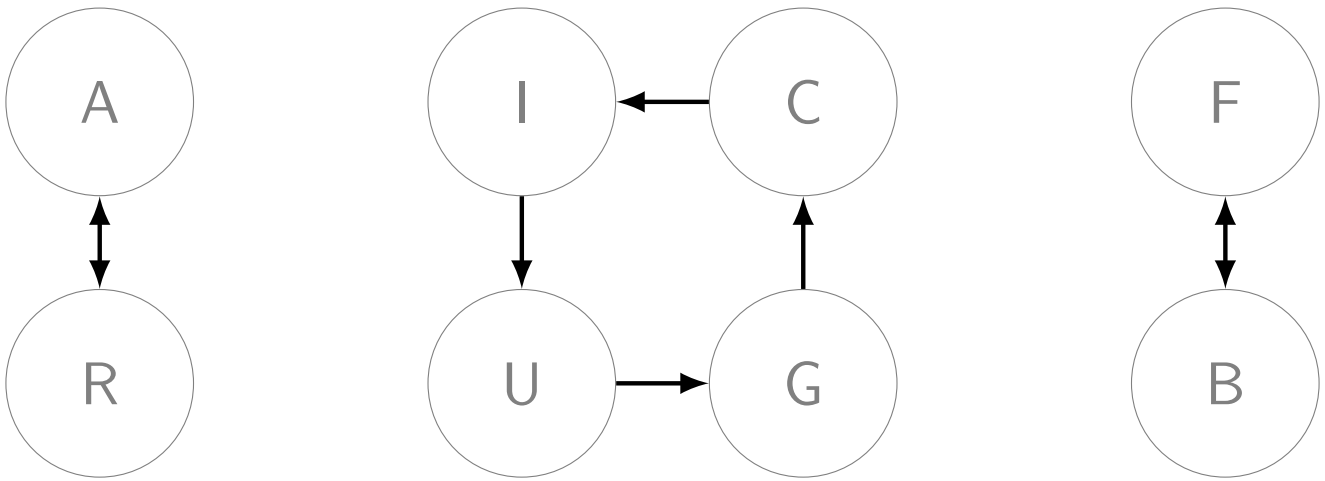
- Aktuelle Kette: $\langle B$
- Längste Kette(n): $\langle C, I, U, G \rangle$

Algorithmus



- Aktuelle Kette: $\langle B, F$
- Längste Kette(n): $\langle C, I, U, G \rangle$

Algorithmus



- Aktuelle Kette: $\langle B, F \rangle$
- Längste Kette(n): $\langle C, I, U, G \rangle$

Wahl der Programmiersprache

Warum Ruby?

Vorteile von Skriptsprachen

- Automatische Speicherverwaltung

Besondere Vorteile von Ruby

Wahl der Programmiersprache

Warum Ruby?

Vorteile von Skriptsprachen

- Automatische Speicherverwaltung
- Listen, Hashes oft integraler Sprachbestandteil

Besondere Vorteile von Ruby

Wahl der Programmiersprache

Warum Ruby?

Vorteile von Skriptsprachen

- Automatische Speicherverwaltung
- Listen, Hashes oft integraler Sprachbestandteil
- Ebenso reguläre Ausdrücke

Besondere Vorteile von Ruby

Wahl der Programmiersprache

Warum Ruby?

Vorteile von Skriptsprachen

- Automatische Speicherverwaltung
- Listen, Hashes oft integraler Sprachbestandteil
- Ebenso reguläre Ausdrücke

Besondere Vorteile von Ruby

- Funktionale Aspekte vereinfachen Operationen auf Listen (*map*, *inject*...)

Wahl der Programmiersprache

Warum Ruby?

Vorteile von Skriptsprachen

- Automatische Speicherverwaltung
- Listen, Hashes oft integraler Sprachbestandteil
- Ebenso reguläre Ausdrücke

Besondere Vorteile von Ruby

- Funktionale Aspekte vereinfachen Operationen auf Listen (*map*, *inject*...)
- Intuitive Syntax, daher besonders für Vorträge geeignet

Der algorithmische Kern

```
1  until unvisited_nodes.empty?
2    chain = []
3    node = unvisited_nodes.first
4
5    while unvisited_nodes.include?(node)
6      chain << node
7      unvisited_nodes.delete(node)
8      node = depend[node]
9    end
10
11   update_longest_chains(chain)
12 end
```

Demo

Fragen, Unklarheiten, Anmerkungen?

Fragen, Unklarheiten, Anmerkungen?

Danke für die Aufmerksamkeit!