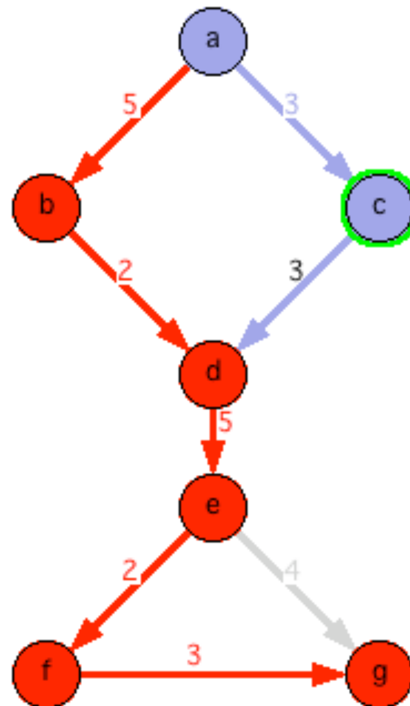
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich



Graphen (1)

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Bezeichnungen

V	**Menge der Knoten (vertices)**
|V|	**Anzahl der Knoten**
E	**Menge der Kanten (edges)**
|E|	**Anzahl der Kanten**
G	**Graph G = (V,E)**
(v,w)	**Kante von v nach w**

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Eigenschaften

gerichtet (directed) — **ungerichtet (undirected)**
gewichtet (weighted) — **ungewichtet (unweighted)**
zyklisch (cyclic) — **zyklenfrei (acyclic)**

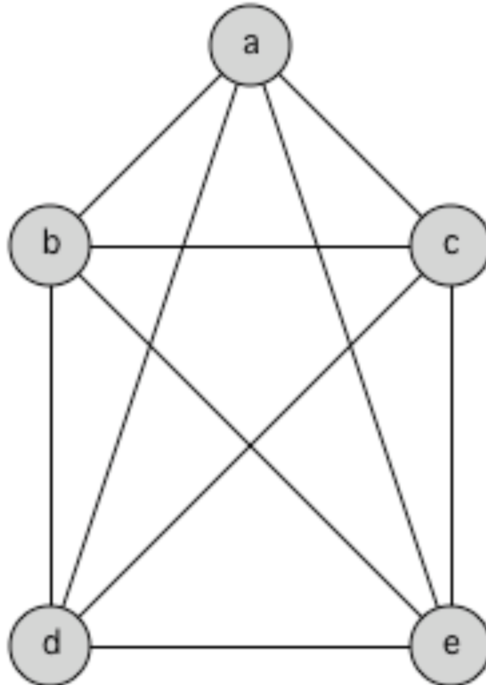
vollständig (complete)
bipartit (bipartite)

gerichtete Graphen:
- **schwach zusammenhängend (weakly connected)**
- **stark zusammenhängend (strongly connected)**

ungerichtete Graphen:
- **zusammenhängend (connected)**
- **zweifach zusammenhängend (biconnected)**

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

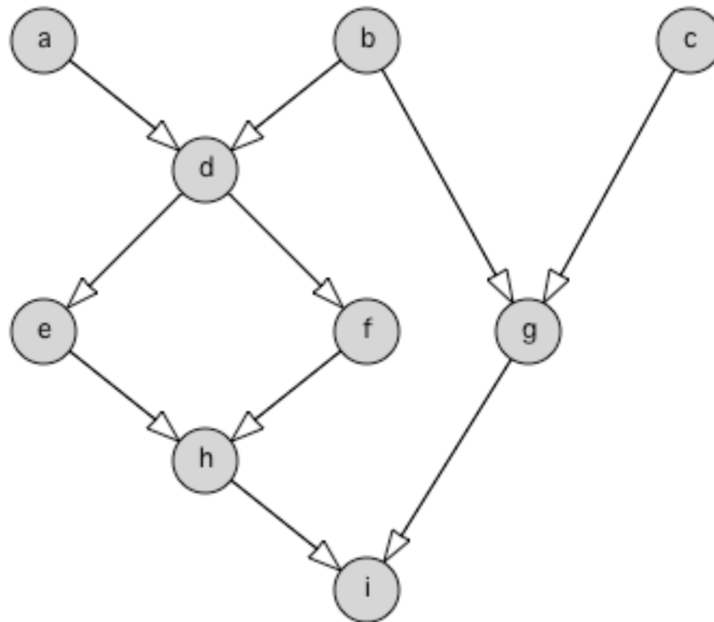
Beispiel:



- ungerichtet
- ungewichtet
- vollständig
- zweifach zusammenhängend
- zyklisch
- nicht planar

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

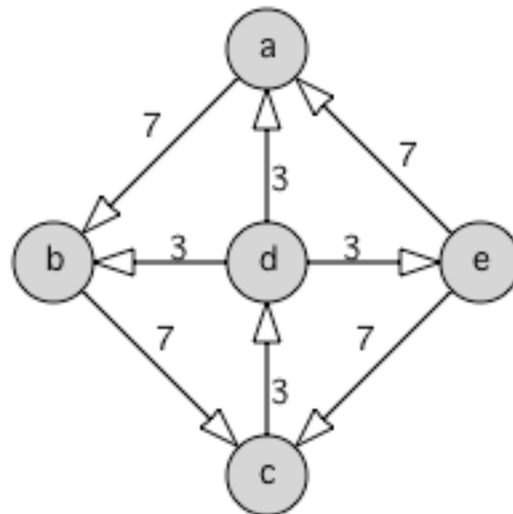
Beispiel:



- gerichtet
- ungewichtet
- schwach zusammenhängend
- zyklenfrei
- planar

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

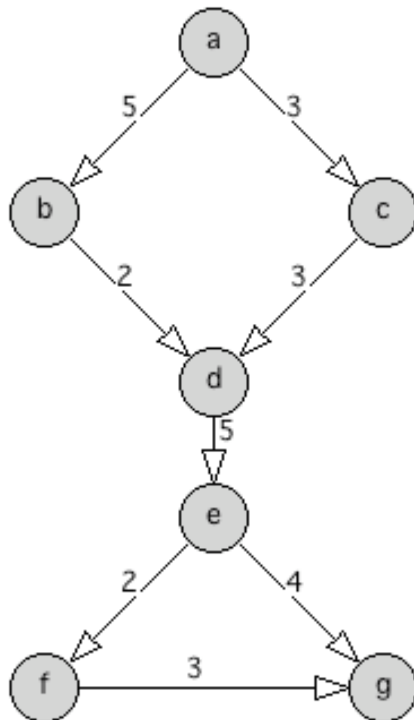
Beispiel:



- gerichtet
- gewichtet
- stark zusammenhängend
- zyklisch
- planar

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Adjazenzmatrix
Bsp: gerichteter und gewichteter Graph

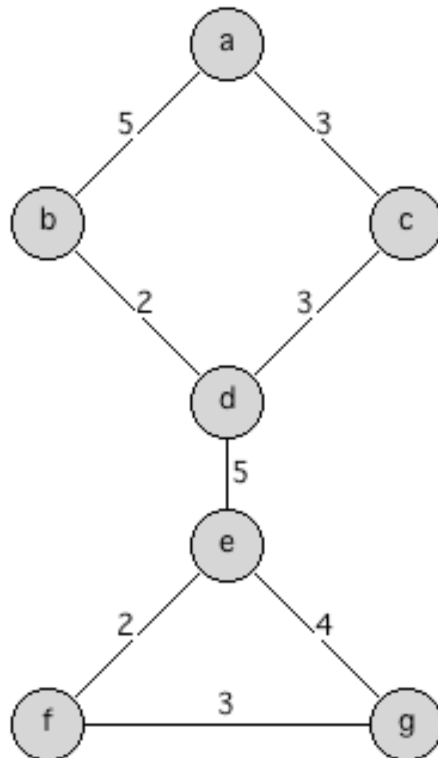


	a	b	c	d	e	f	g
a		5	3				
b				2			
c				3			
d					5		
e						2	4
f							3
g							

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Adjazenzmatrix

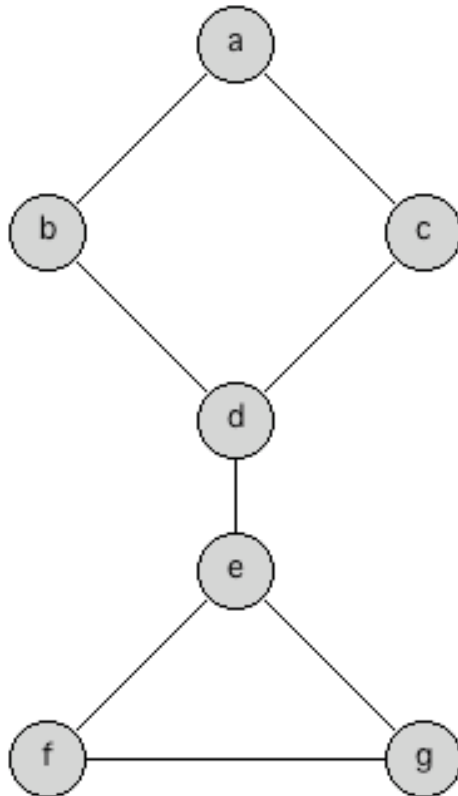
Bsp: ungerichteter gewichteter Graph



	a	b	c	d	e	f	g
a		5	3				
b	5			2			
c	3			3			
d		2	3		5		
e				5		2	4
f					2		3
g					4	3	

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

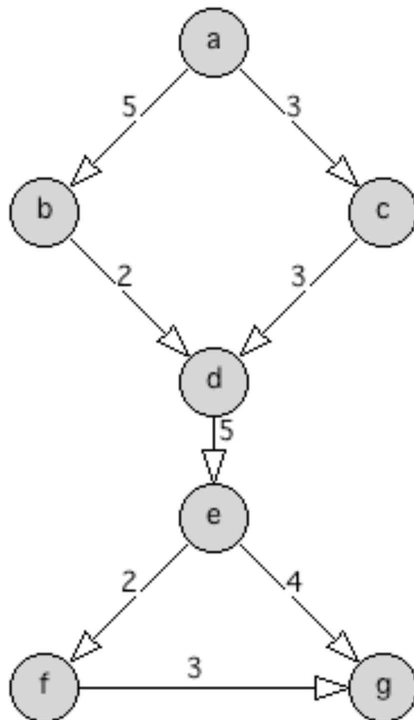
Adjazenzmatrix
Bsp: ungerichteter ungewichteter Graph

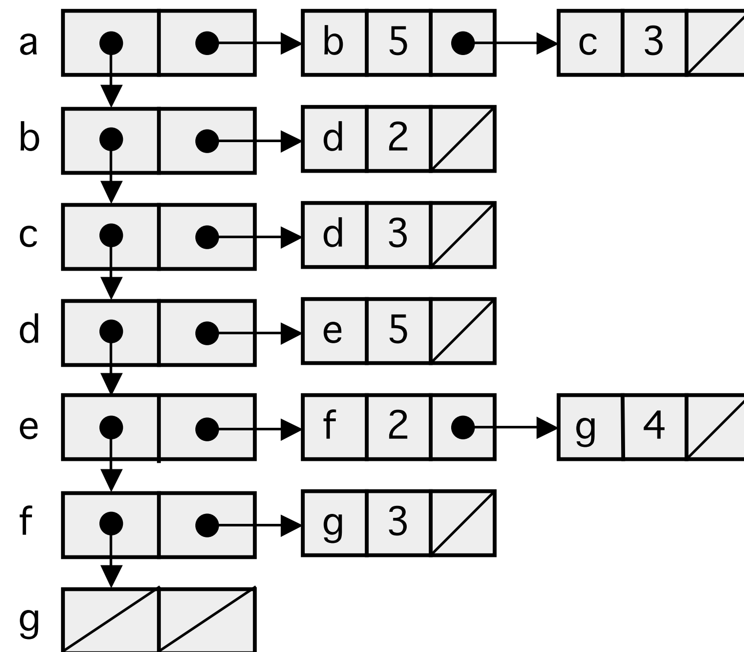


	a	b	c	d	e	f	g
a		1	1				
b	1			1			
c	1			1			
d		1	1		1		
e				1		1	1
f					1		1
g					1	1	

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

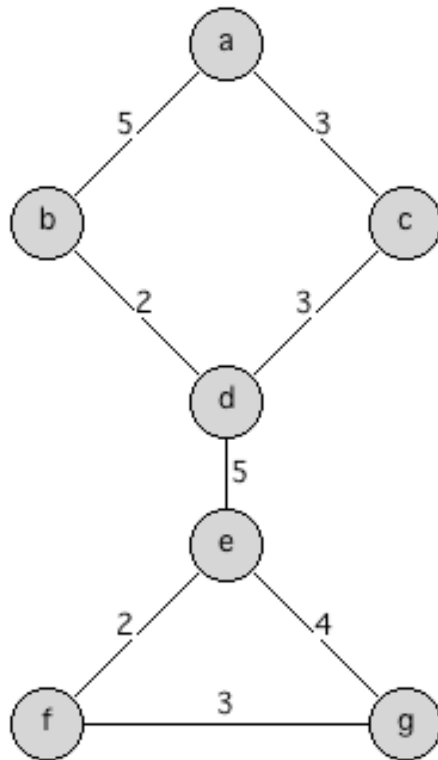
Adjazenzlisten
Bsp: gerichteter Graph

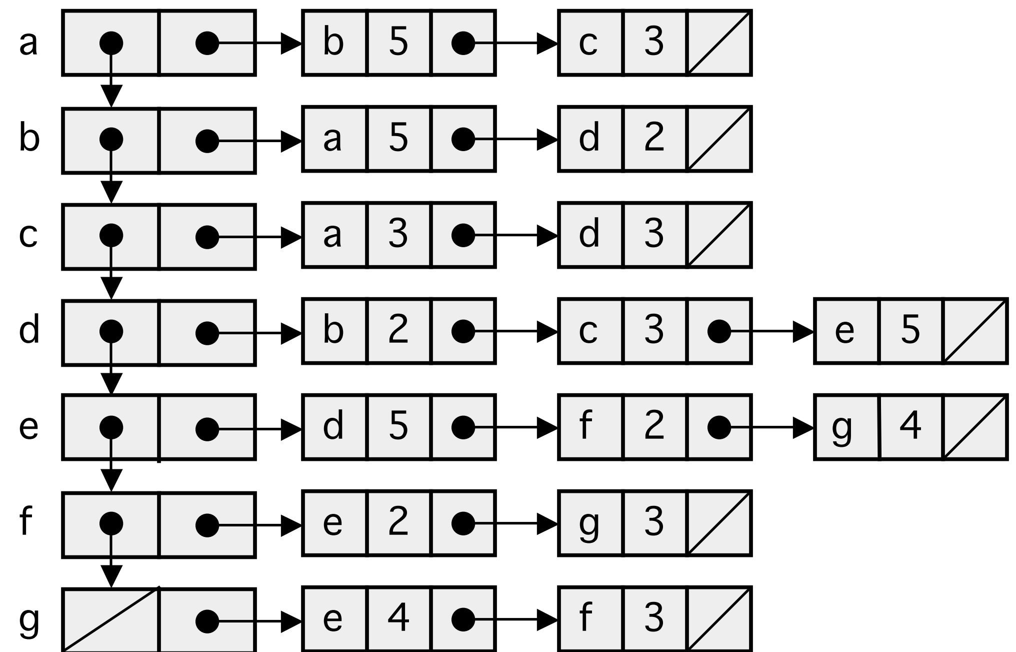




Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Adjazenzlisten
Bsp: ungerichteter Graph





Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Implementierung eines Graphen

```
class Graph {
    Vector nodes = new Vector();
    void clearNodes() {
        for (int i=0; i<nodes.size(); i++) {
            Node node = (Node)nodes.elementAt(i);
            node.setVisited(false);
        }
    }
    void depthFirst()...
    void breadthFirst()...
}
```

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Implementierung der Knoten

```
class Node {
	boolean visited = false;
	Vector edges = new Vector();

	boolean isVisited() {return visited;}
	void setVisited(boolean p) {visited = p;}
	void depthFirst()...
	void breadthFirst()...
}
```

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Implementierung der Kanten

```
class Edge {
    int weight;
    Node to;
    Edge(int weight, Node to) {
        this.weight = weight;
        this.to= to;
    }
}
```

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Tiefensuche (rekursiv)

```
class Graph {
...
    void depthFirst() {
        clearNodes();
        for (int i=0; i<nodes.size(); i++) {
            Node node = (Node)nodes.elementAt(i);
            if (!node.isVisited())
                node.depthFirst();
        }
    }
}
```

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

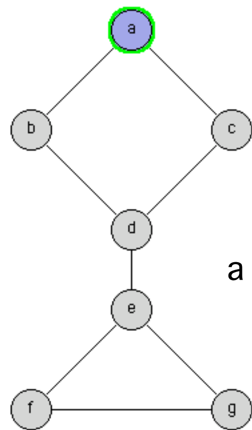
Tiefensuche (rekursiv)

$O(|V|+|E|)$

```
class Node {
...
    void depthFirst() {
        if (isVisited()) return;
        visit(this);
        setVisited(true);
        for (int i=0; i<edges.size(); i++) {
            Edge edge = (Edge)edges.elementAt(i);
            edge.to.depthFirst();
        }
    }
}
```

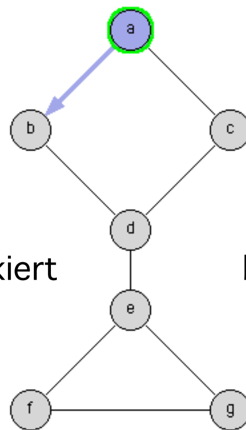
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Tiefensuche (1)



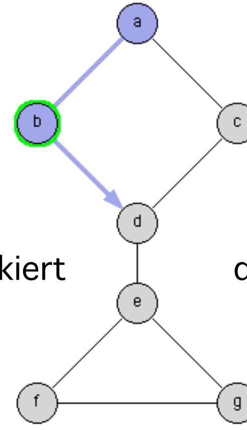
a wird markiert

1. Schritt:
Start mit a



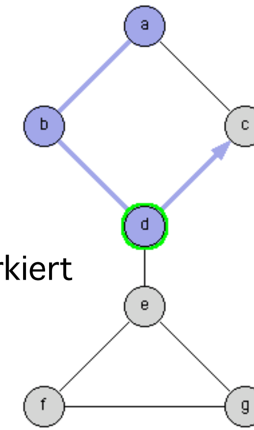
b wird markiert

2. Schritt: weiter
von a nach b



d wird markiert

3. Schritt: weiter
von b nach d

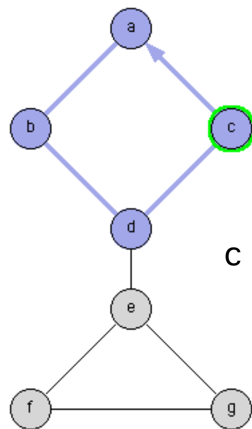


c wird markiert

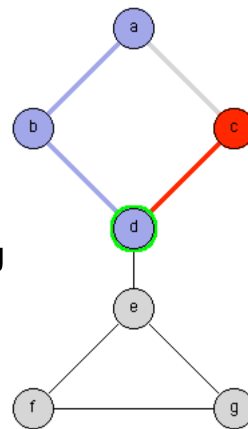
4. Schritt: weiter
von d nach c

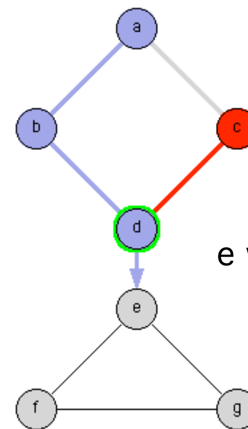
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Tiefensuche (2)

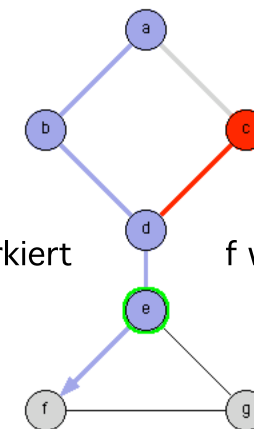


c ist fertig





e wird markiert



f wird markiert

5. Schritt: stop
a ist markiert!
Vorsicht Zyklus!

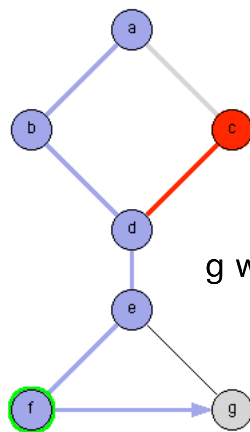
6. Schritt: zurück
von c nach d
(c,d) wird Baumkante

7. Schritt: weiter
von d nach e

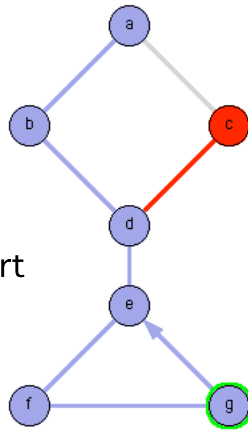
8. Schritt: weiter
von e nach f

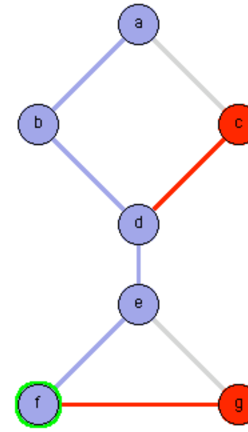
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

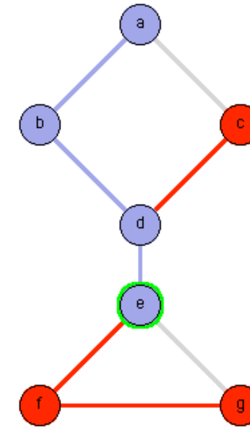
Tiefensuche (3)



g wird markiert







9. Schritt: weiter
von f nach g

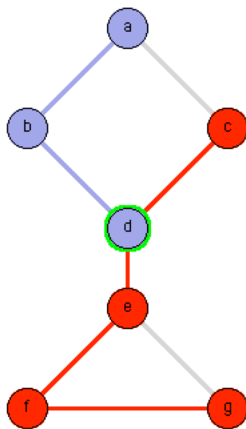
10. Schritt: stop
e ist markiert!
Vorsicht Zyklus!

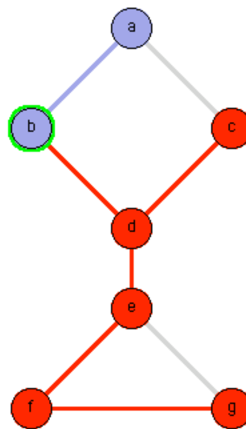
11. Schritt: zurück
von g nach f
(f,g) wird Baumkante

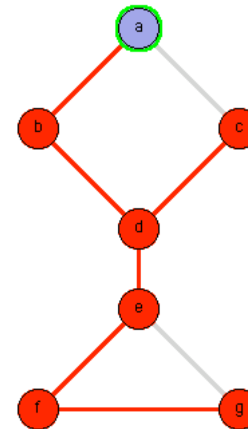
12. Schritt: zurück
von f nach e
(e,f) wird Baumkante

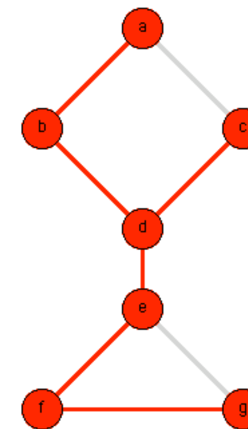
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Tiefensuche (4)









13. Schritt: zurück
von e nach d
(d,e) wird Baumkante

14. Schritt: zurück
von d nach b
(b,d) wird Baumkante

15. Schritt: zurück
von b nach a
(a,b) wird Baumkante

16. Schritt: fertig
als Ergebnis entsteht
ein spannender Baum!

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Tiefensuche (iterativ)

```
void depthFirst() {
	Stack stack = new Stack();
	clearNodes();
	for (int i=0; i<nodes.size(); i++) {
		Node start = (Node)nodes.elementAt(i);
		if (!start.isVisited()) {
			start.setVisited(true);
			stack.push(start);
			do {
				Node node = (Node)stack.pop();
				visit(node);
				for (int j=node.edges.size()-1; j>=0; j--) {
					Edge edge = (Edge)node.edges.elementAt(j);
					if (!edge.to.isVisited()) {
						edge.to.setVisited(true);
						stack.push(edge.to);
					}
				}
			} while (!stack.isEmpty())
		}
	}
}
```

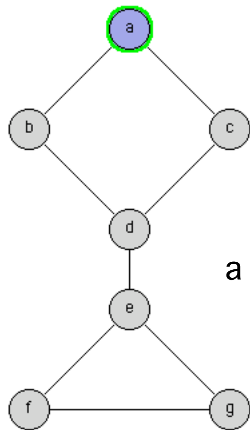
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Breitensuche (iterativ) O(|V|+|E|)

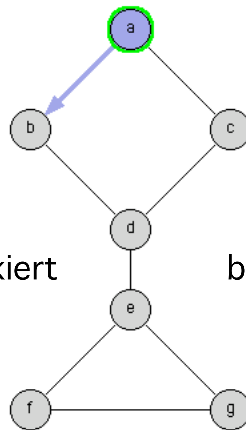
```
void breadthFirst() {
	Queue queue = new Queue();
	clearNodes();
	for (int i=0; i<nodes.size(); i++) {
		Node start = (Node)nodes.elementAt(i);
		if (!start.isVisited()) {
			start.setVisited(true);
			queue.put(start);
			do {
				Node node = (Node)queue.get();
				visit(node);
				for (int j=0; j<node.edges.size(); j++) {
					Edge edge = (Edge)node.edges.elementAt(j);
					if (!edge.to.isVisited()) {
						edge.to.setVisited(true);
						queue.put(edge.to);
					}
				}
			} while (!queue.isEmpty())
		}
	}
}
```

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

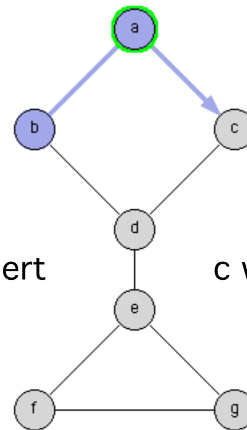
Breitensuche (1)



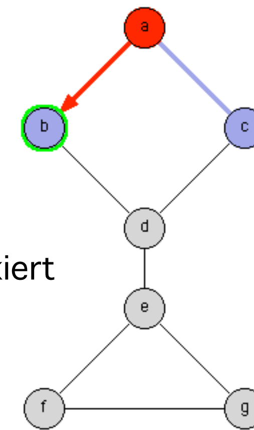
a wird markiert



b wird markiert



c wird markiert



1. Schritt:
Start mit a

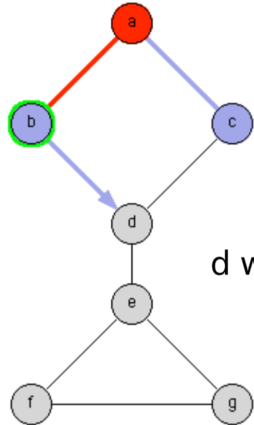
2. Schritt:
[] ⇦ (a,b)

3. Schritt:
[(a,b)] ⇦ (a,c)

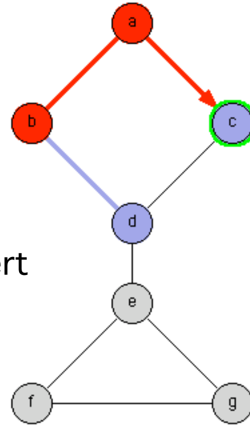
4. Schritt:
(a,b) ⇦ [(a,c)]
(a,b) wird Baumkante

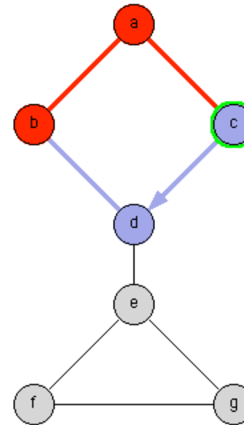
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

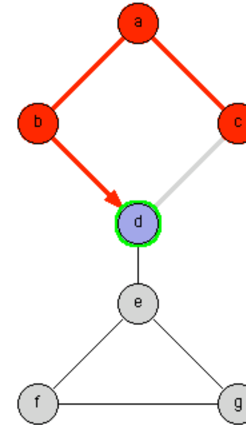
Breitensuche (2)



d wird markiert







5. Schritt:
[(a,c)] ⇦ (b,d)

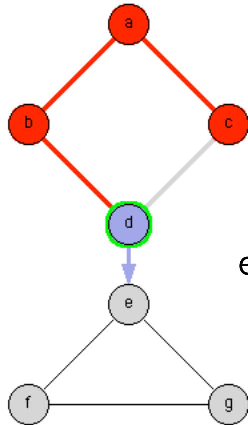
6. Schritt:
(a,c) ⇦ [(b,d)]
(a,c) wird Baumkante

7. Schritt:
d ist markiert
Vorsicht Zyklus!

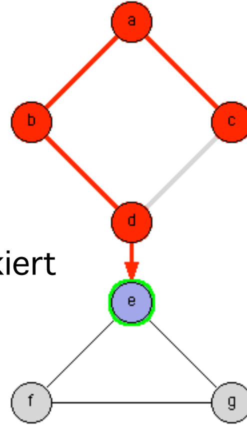
8. Schritt:
(b,d) ⇦ []
(b,d) wird Baumkante

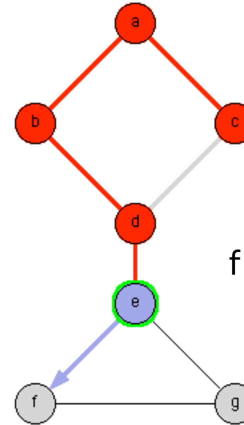
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Breitensuche (3)

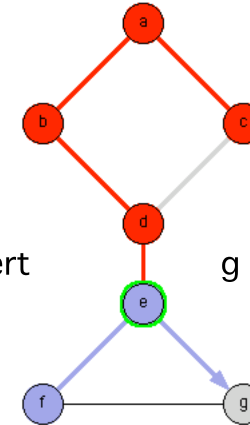


e wird markiert





f wird markiert



g wird markiert

9. Schritt:
[] ⇦ (d,e)

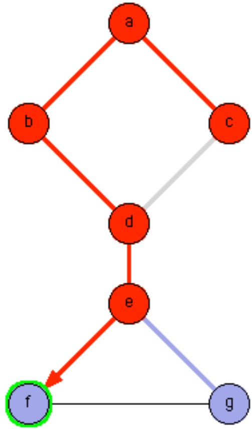
10. Schritt:
(d,e) ⇦ []
(d,e) wird Baumkante

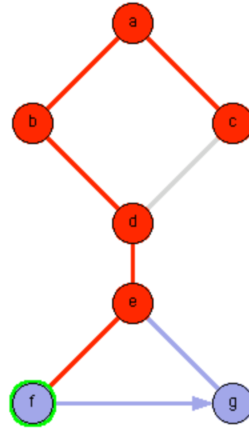
11. Schritt:
[] ⇦ (e,f)

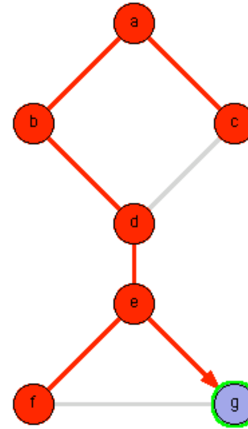
12. Schritt:
[(e,f)] ⇦ (e,g)

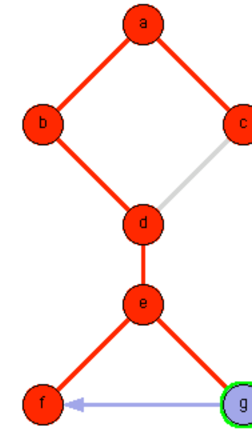
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Breitensuche (4)









13. Schritt:
(e,f) ⇦ [(e,g)]
(e,f) wird Baumkante

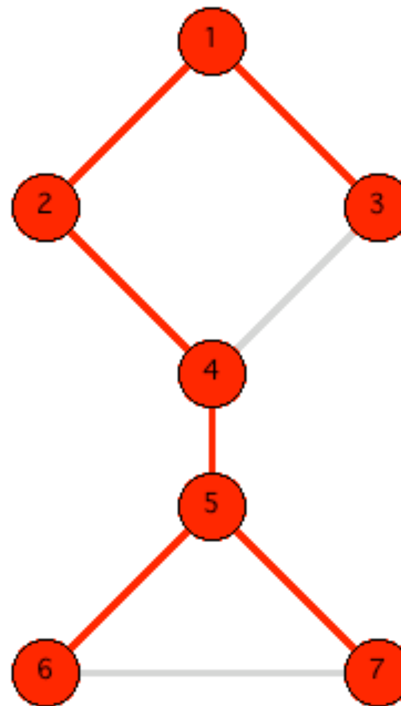
14. Schritt:
g ist markiert
Vorsicht Zyklus!

15. Schritt:
(e,g) ⇦ []
(e,g) wird Baumkante

16. Schritt:
f ist markiert
Vorsicht Zyklus!

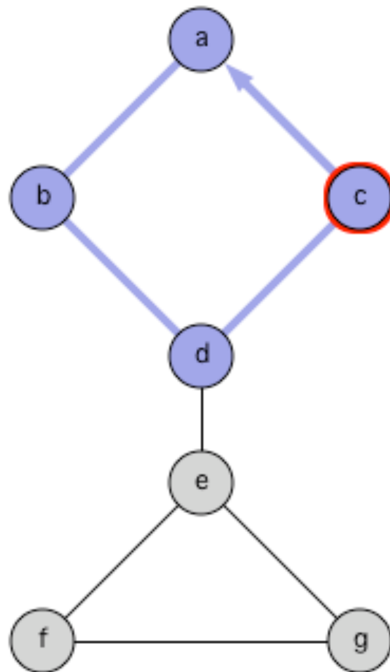
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Spannender Baum als Ergebnis der Breitensuche



Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

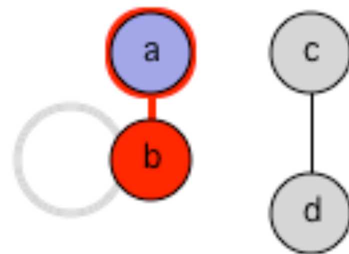
Traversieren eines ungerichteten Graphen:



Kanten die zu einem bereits markierten Knoten führen kennzeichnen einen Zyklus!

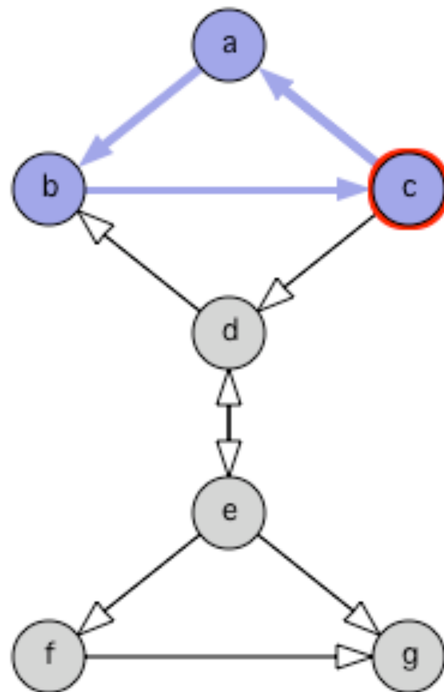
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Traversieren eines ungerichteten Graphen:



Alle von einem Startknoten aus erreichbaren Knoten gehören zur gleichen Komponente!

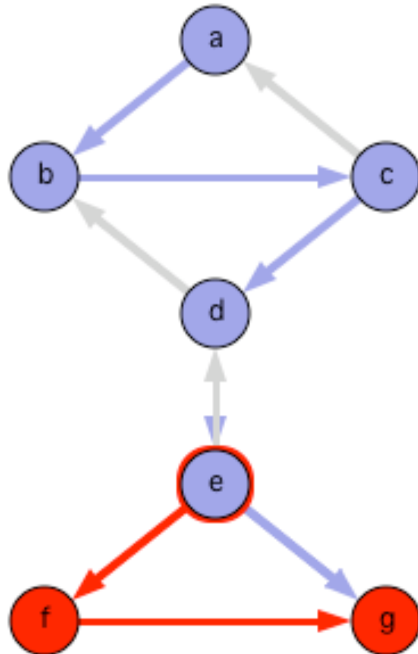
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich



Rekursive Tiefensuche in gerichteten Graphen:

Kanten die zu einem aktiven (blau) markierten Knoten führen kennzeichnen einen Zyklus!

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

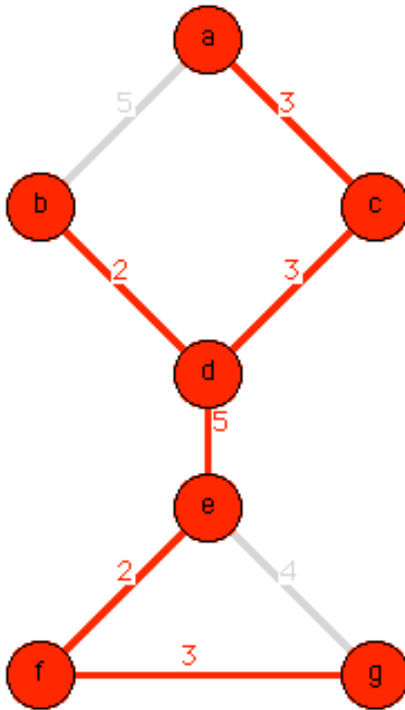


Rekursive Tiefensuche in gerichteten Graphen:

Kanten die zu einem (rot) markierten Baum-Knoten führen kennzeichnen keinen Zyklus!

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Minimum Spanning Tree (Dijkstra/Prim)



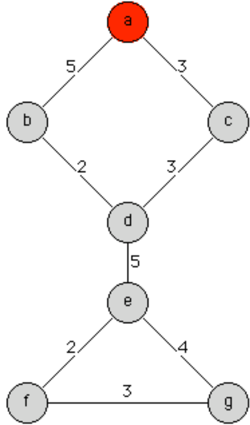
Ermittlung eines minimalen spannenden Baumes eines gewichteten Graphens nach Dijkstra/Prim

O(|E| log |V|)

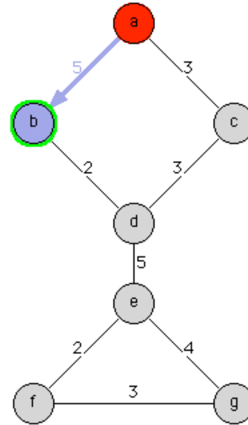
Breitensuche unter Verwendung einer PriorityQueue (geringes Gewicht hat hohe Priorität) liefert einen minimalen spannenden Baum!

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

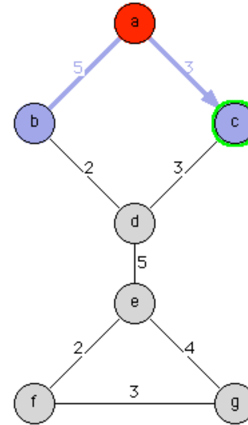
Dijkstra/Prim (1)



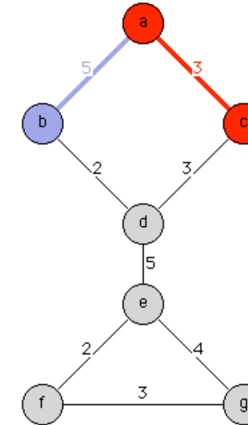
1. Schritt:
Start mit a



2. Schritt:
[] ⇦ (a,b,5)



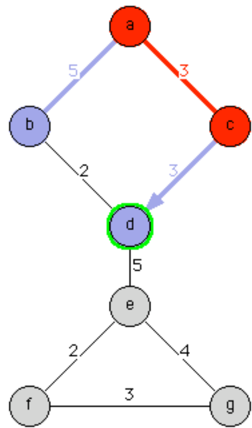
3. Schritt:
[(a,b,5)] ⇦ (a,c,3)



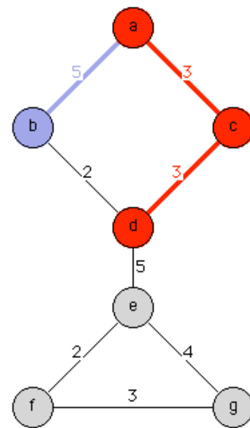
4. Schritt:
(a,c,3) ⇦ [(a,b,5)]

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

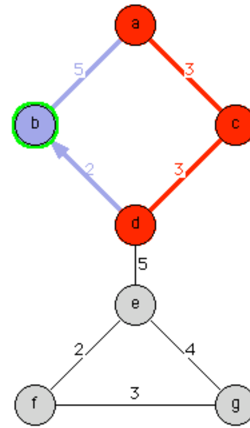
Dijkstra/Prim (2)



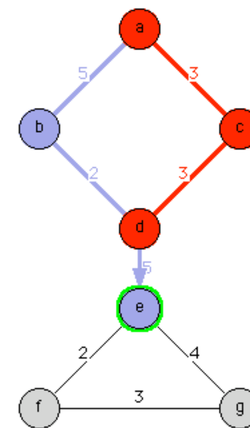
5. Schritt:
[(a,b,5)] ⇦ (c,d,3)



6. Schritt:
(c,d,3) ⇦ [(a,b,5)]



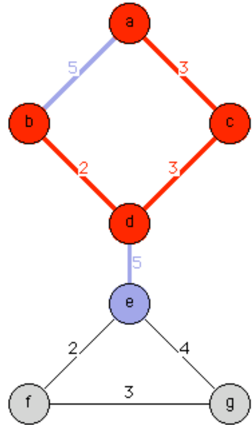
7. Schritt:
[(a,b,5)] ⇦ (d,b,2)
(d,b,2) ersetzt (a,b,5)



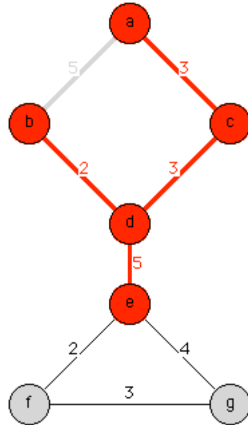
8. Schritt:
[(d,b,2)] ⇦ (d,e,5)

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

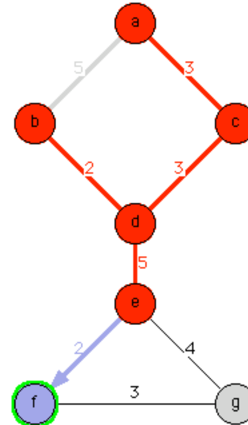
Dijkstra/Prim (3)



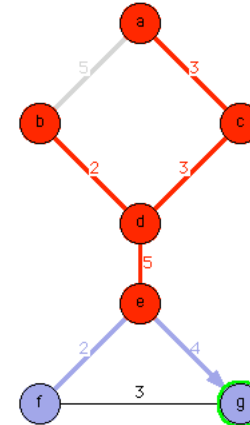
9. Schritt:
(d,b,2) ⇦ [(d,e,5)]



10. Schritt:
(d,e,5) ⇦ []



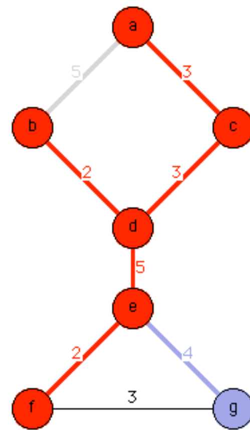
11. Schritt:
[] ⇦ (e,f,2)

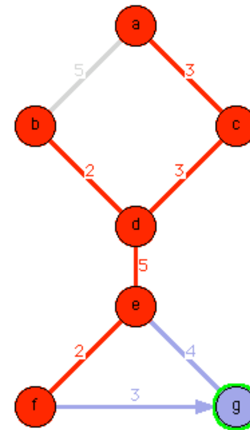


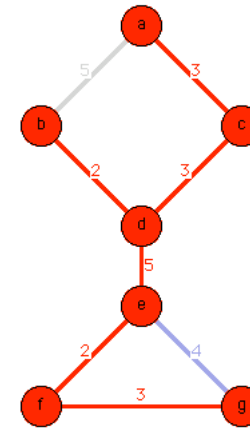
12. Schritt:
[(e,f,2)] ⇦ (e,g,4)

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Dijkstra/Prim (4)







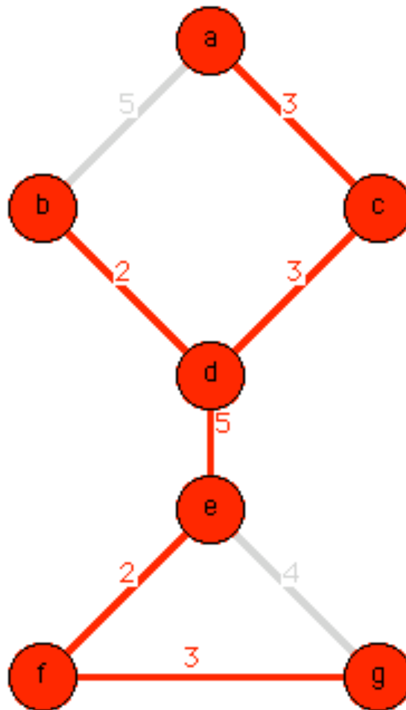
13. Schritt:
(e,f,2) ⇦ [(e,g,4)]

14. Schritt:
[(e,g,4)] ⇦ (f,g,3)
(f,g,3) ersetzt (e,g,4)

15. Schritt:
(f,g,3) ⇦ []
fertig

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Minimum Spanning Tree (Kruskal)



Ermittlung eines minimalen spannenden Baumes eines gewichteten Graphens nach Kruskal

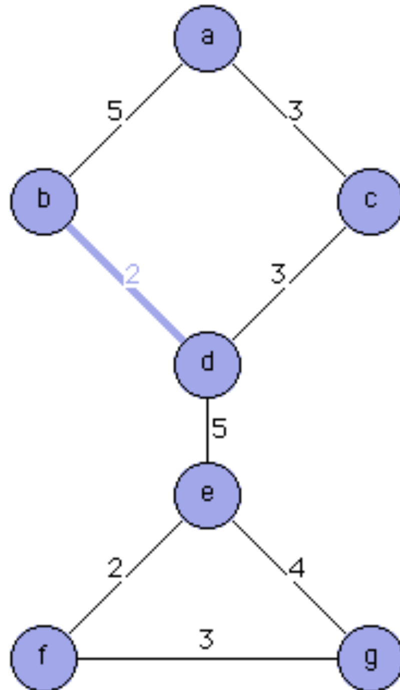
$O(|E| \log|E|)$

Entfernen sämtlicher Kanten (führt zu einem Graphen mit |V| isolierten Knoten).

Hinzufügen jener Kanten in nach Gewichten steigend sortierter Reihenfolge die die Anzahl der Komponenten verringern.

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

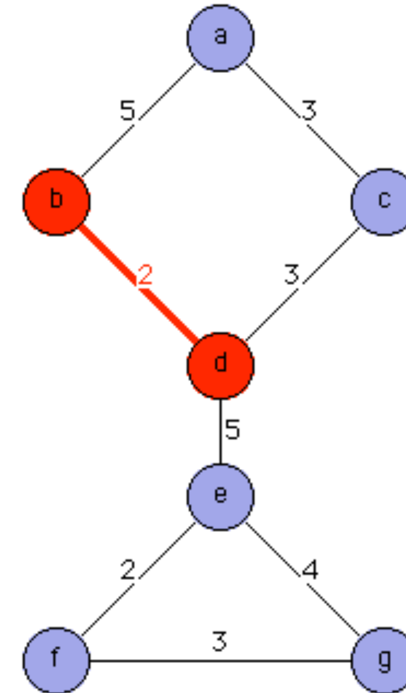
Kruskal (1)



b≠d ⇨ union(b,d)

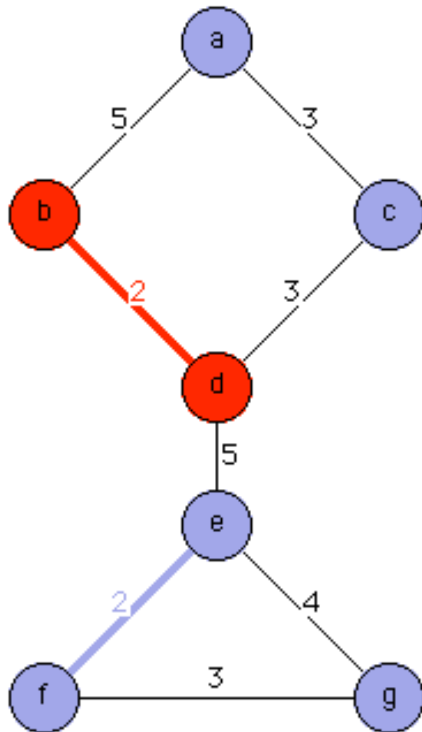
Die Knoten b und d befinden sich in unterschiedlichen Komponenten.

Die Kante (b,d) verbindet die beiden isolierten Knoten b und d



Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

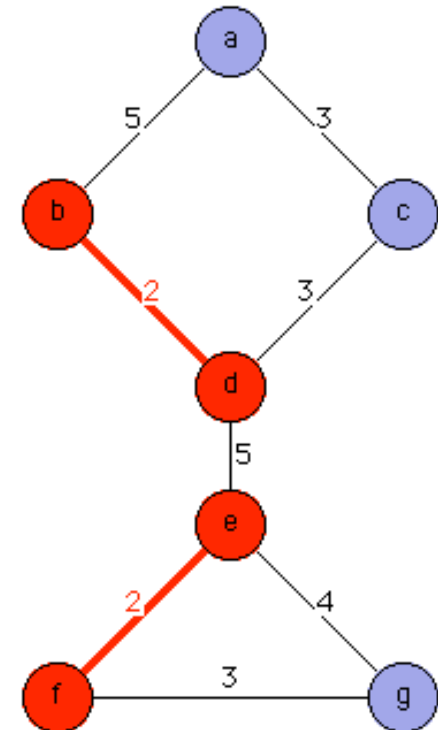
Kruskal (2)



e≠f ⇨ union(e,f)

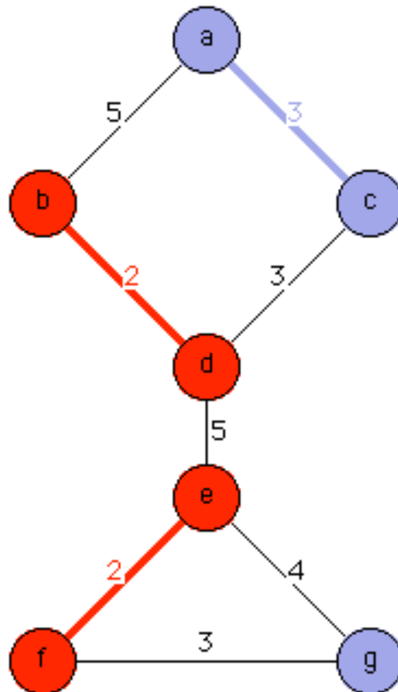
Die Knoten e und f befinden sich in unterschiedlichen Komponenten.

Die Kante (e,f) verbindet die beiden isolierten Knoten e und f



Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

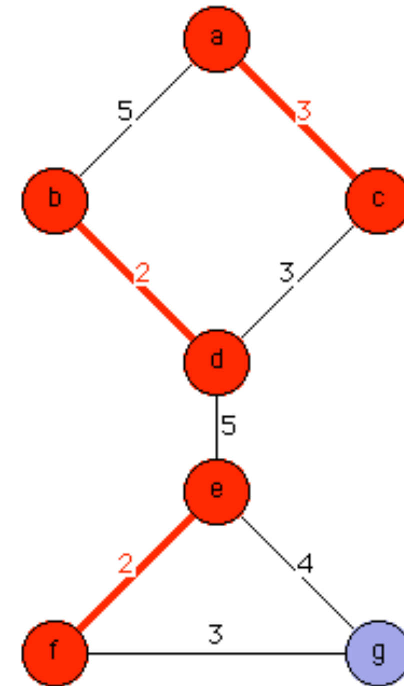
Kruskal (3)



a≠c ⇨ union(a,c)

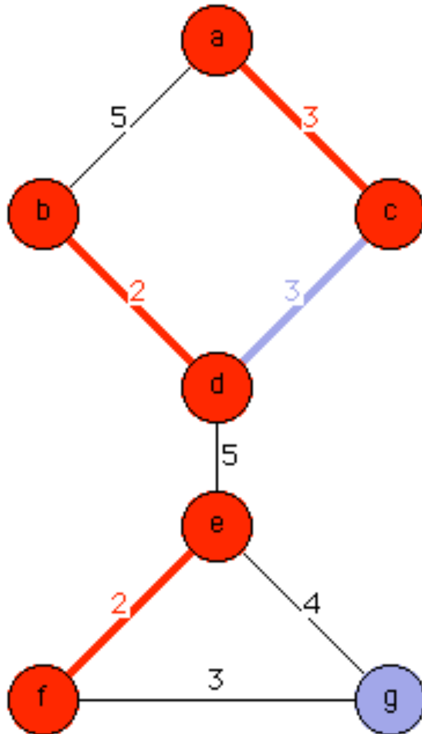
Die Knoten a und c befinden sich in unterschiedlichen Komponenten.

Die Kante (a,c) verbindet die beiden isolierten Knoten a und c



Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

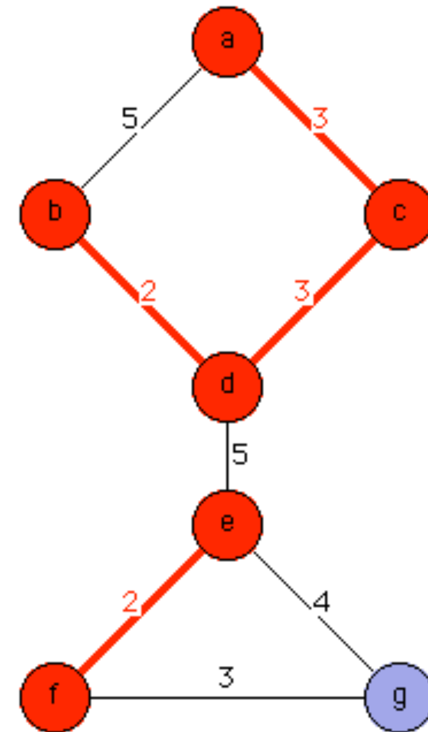
Kruskal (4)



c≠d ⇨ union(c,d)

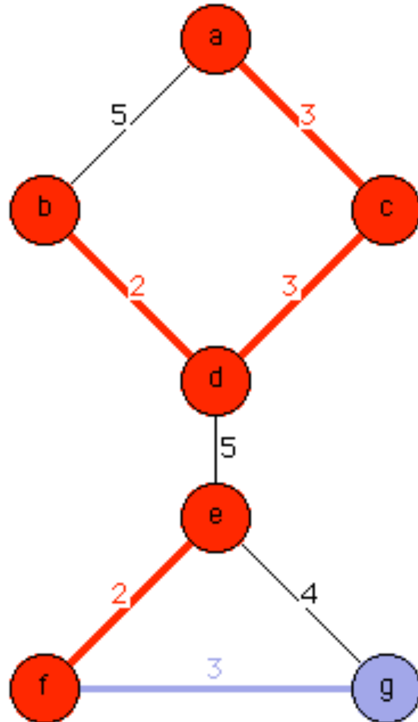
Die Knoten c und d befinden sich in unterschiedlichen Komponenten.

Die Kante (c,d) verbindet die beiden Komponenten ac und bd



Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

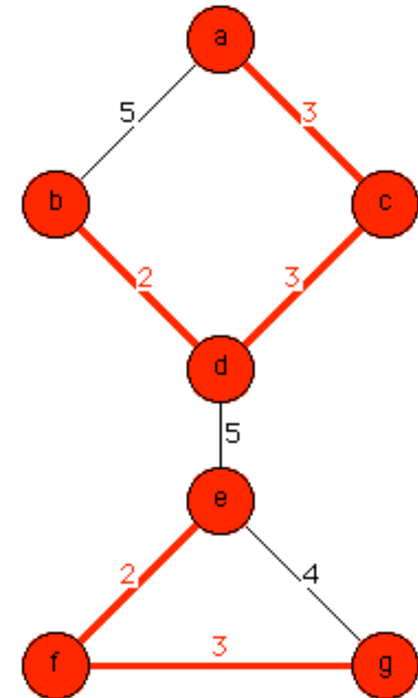
Kruskal (5)



f≠g ⇨ union(f,g)

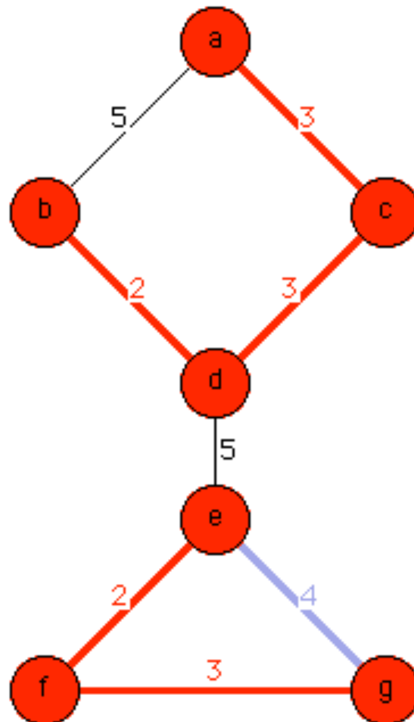
Die Knoten f und g befinden sich in unterschiedlichen Komponenten.

Die Kante (f,g) verbindet die Komponente ef und den isolierten Knoten g



Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Kruskal (6)



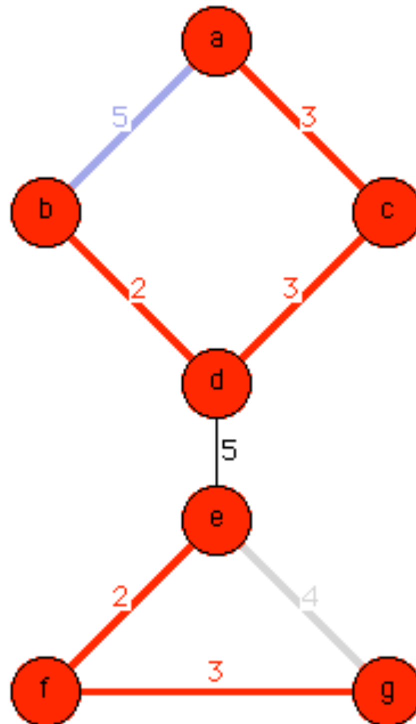
e=g

Die Knoten e und g befinden sich in derselben Komponente efg.

Die Kante (e,g) ist nicht Bestandteil des minimalen spannenden Baumes!

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Kruskal (7)



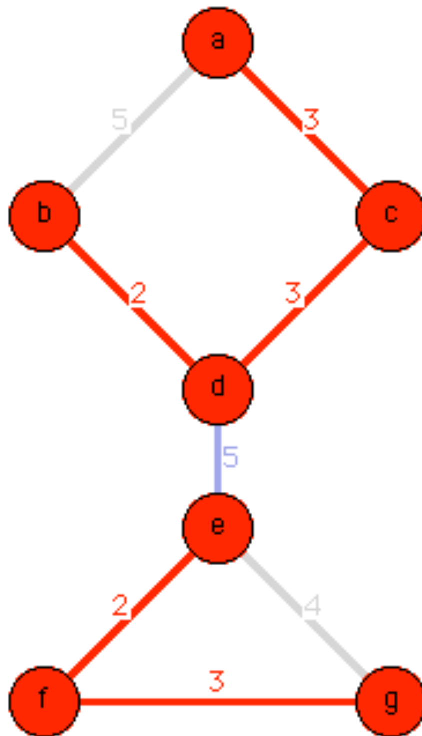
a=b

Die Knoten a und b befinden sich in derselben Komponente abcd.

Die Kante (a,b) ist nicht Bestandteil des minimalen spannenden Baumes!

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

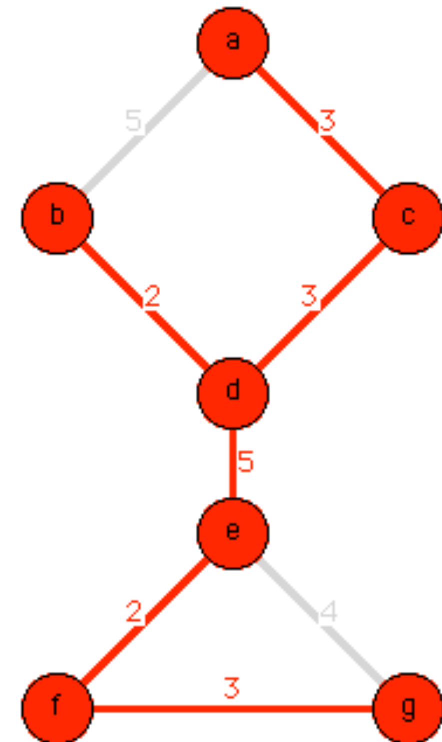
Kruskal (8)



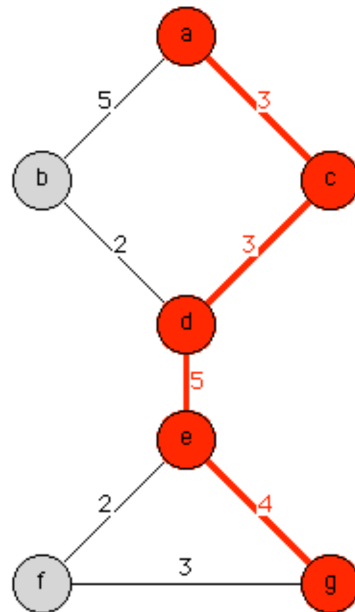
d≠e ⇨ union(d,e)

Die Knoten d und e befinden sich in unterschiedlichen Komponenten.

Die Kante (d,e) verbindet die Komponente abcd und efg



Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich



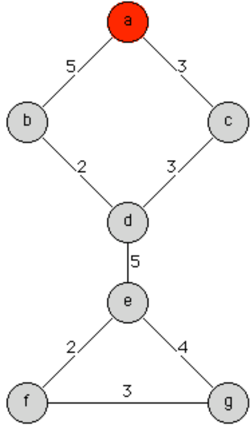
Ermittlung aller von einem Startknoten ausgehenden kürzesten Wege eines gewichteten Graphens nach Dijkstra

O(|E| log|V|)

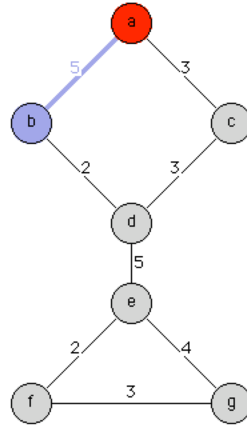
Breitensuche unter Verwendung einer PriorityQueue (geringe Entfernung vom Startknoten hat hohe Priorität) ordnet jedem Knoten den kürzesten Weg vom Startknoten zu!

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

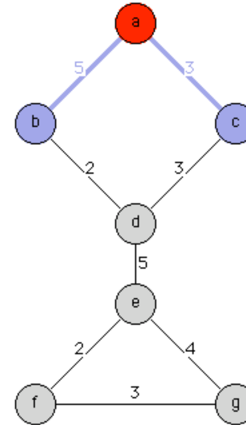
Dijkstra (1)



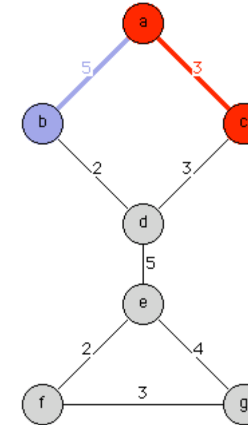
1. Schritt:
Start mit a



2. Schritt:
[] ⇦ (a,b,5)



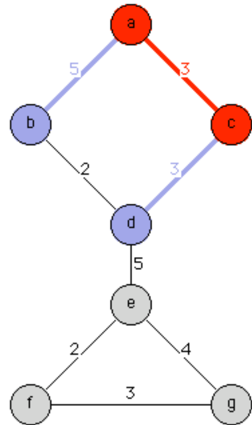
3. Schritt:
[(a,b,5)] ⇦ (a,c,3)

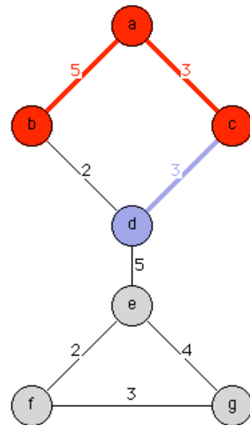


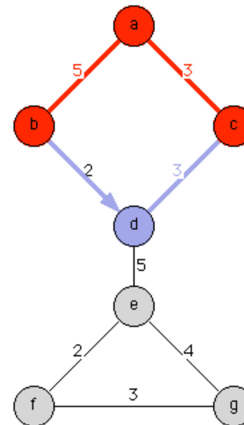
4. Schritt:
(a,c,3) ⇦ [(a,b,5)]

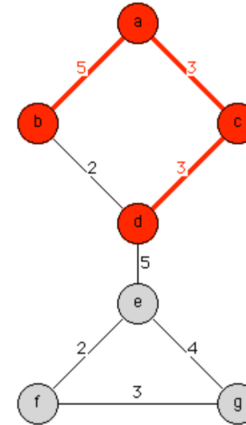
Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Dijkstra (2)









5. Schritt:
[(a,b,5)] ⇦ (c,d,6)

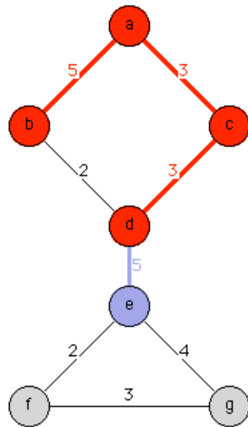
6. Schritt:
(a,b,5) ⇦ [(c,d,6)]

7. Schritt:
[(c,d,6)]
(b,d,7) > (c,d,6)

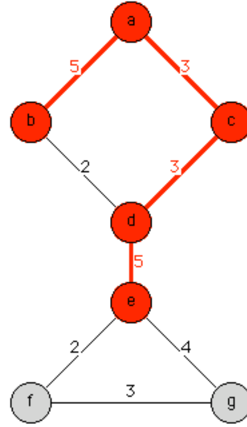
8. Schritt:
(c,d,6) ⇦ []

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

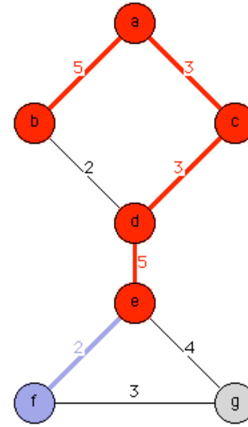
Dijkstra (3)



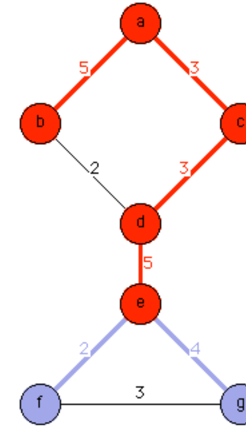
9. Schritt:
[] ⇦ (d,e,11)



10. Schritt:
(d,e,11) ⇦ []



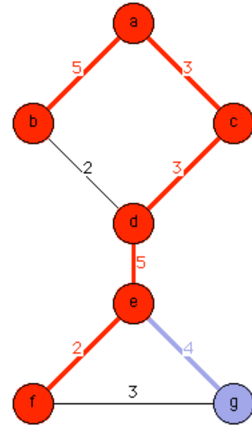
11. Schritt:
[] ⇦ (e,f,13)



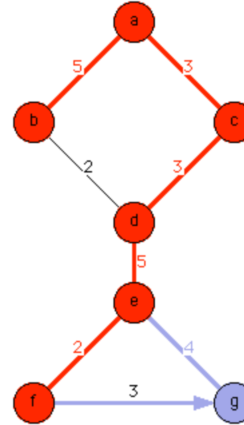
12. Schritt:
[(e,f,13)] ⇦ (e,g,15)

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

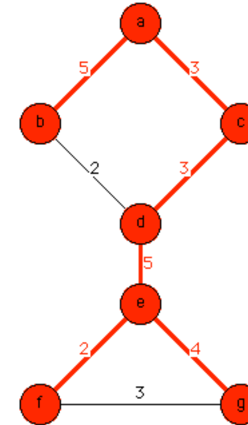
Dijkstra (4)



13. Schritt:
(e,f,13) ⇦ [(e,g,15)]



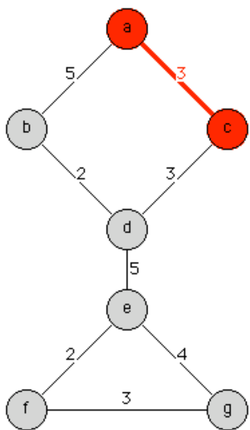
14. Schritt:
[(e,g,15)]
(f,g,16) > (e,g,15)



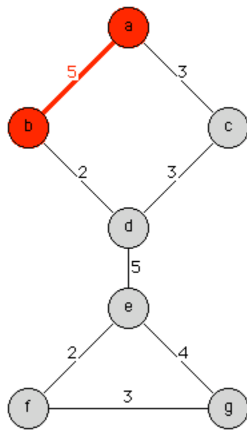
15. Schritt:
(e,g,15) ⇦ []
fertig

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

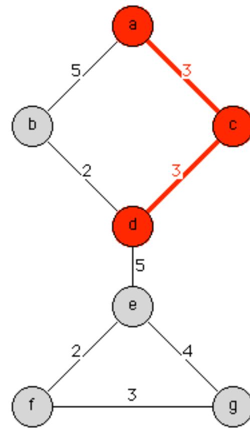
Dijkstra (5)
Alle kürzesten Wege von a:



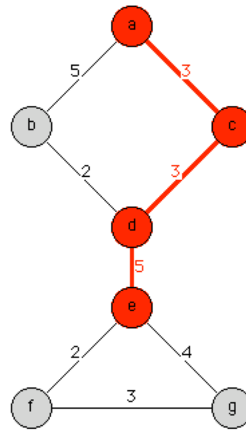
a-c 3



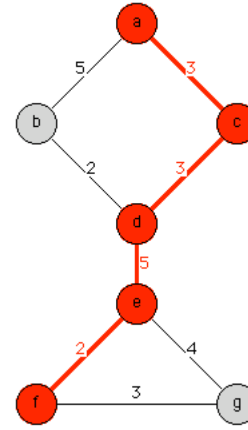
a-b 5



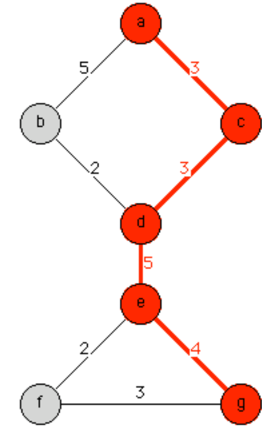
a-c-d 6



a-c-d-e 11



a-c-d-e-f 13



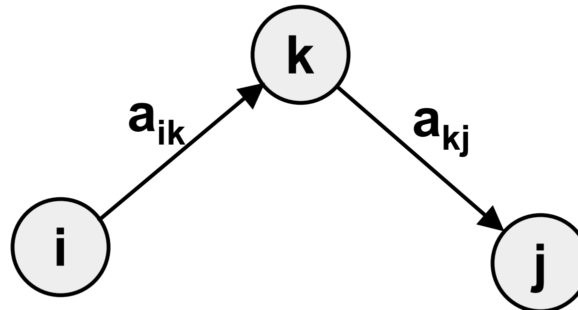
a-c-d-e-g 15

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Transitive Hülle

Adjazenzmatrix a: **boolean a[][] = new boolean[n][n];**

Transitive Hülle: **All i,j,k: 0≤i,j,k<n: a[i][k] and a[k][j] ⇒ a[i][j]**



Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Transitive Hülle
Algorithmus von Warshall
$O(|V|^3)$

```
for (int k=0; k<n; k++)
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            if (a[i][k] && a[k][j]) a[i][j] = true;

            oder

            a[i][j] = a[i][j] || (a[i][k] && a[k][j]);
```

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

Kürzeste Wege
Algorithmus von Floyd
$O(|V|^3)$

Adjazenzmatrix a: `double a[][] = new double[n][n];`

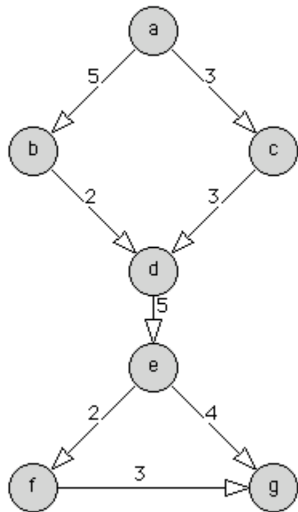
a_{ij} ... Entfernung von i nach j

$a_{ij} = \infty$ falls kein Weg von i nach j

```
for (int k=0; k<n; k++)
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            a[i][j] = min(a[i][j], a[i][k] + a[k][j]);
```

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

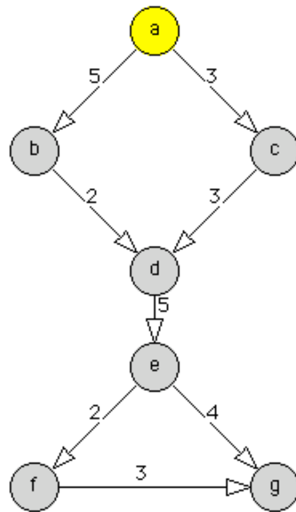
Floyd (1)



shortest paths:

	a	b	c	d	e	f	g
a		5	3				
b				2			
c				3			
d					5		
e						2	4
f							3
g							

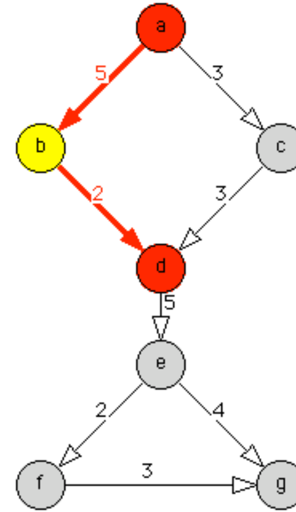
Adjazenzmatrix



shortest paths:

	a	b	c	d	e	f	g
a		5	3				
b				2			
c				3			
d					5		
e						2	4
f							3
g							

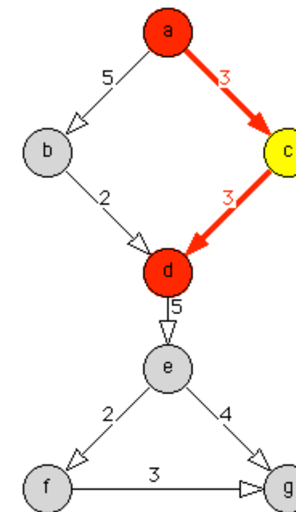
es führt kein Weg übera



shortest paths:

	a	b	c	d	e	f	g
a		5	3	7			
b				2			
c				3			
d					5		
e						2	4
f							3
g							

a - b -d (7)



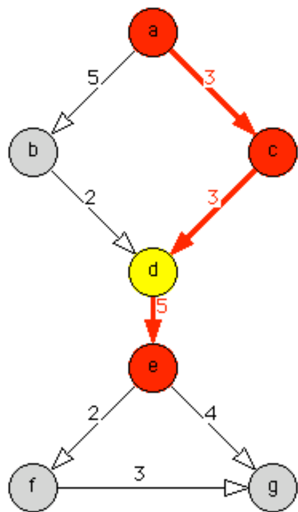
shortest paths:

	a	b	c	d	e	f	g
a		5	3	6			
b				2			
c				3			
d					5		
e						2	4
f							3
g							

a - c -d (6)< a - b -d (7)

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

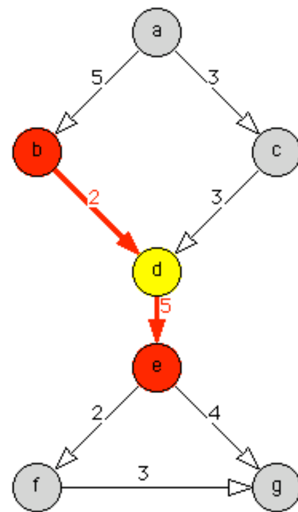
Floyd (2)



shortest paths:

	a	b	c	d	e	f	g
a		5	3	6	11		
b				2			
c				3			
d					5		
e						2	4
f							3
g							

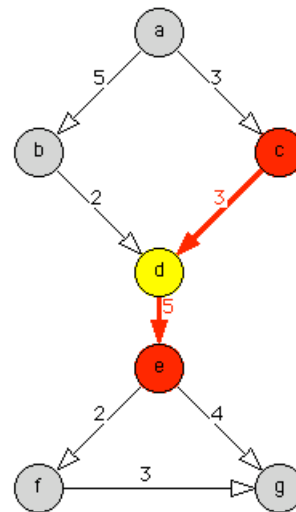
a -c -d -e (11)



shortest paths:

	a	b	c	d	e	f	g
a		5	3	6	11		
b				2	7		
c				3			
d					5		
e						2	4
f							3
g							

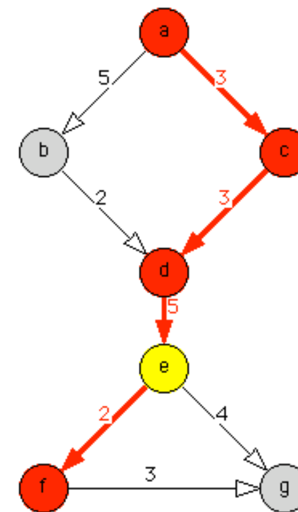
b -d -e (7)



shortest paths:

	a	b	c	d	e	f	g
a		5	3	6	11		
b				2	7		
c				3	8		
d					5		
e						2	4
f							3
g							

c -d -e (8)



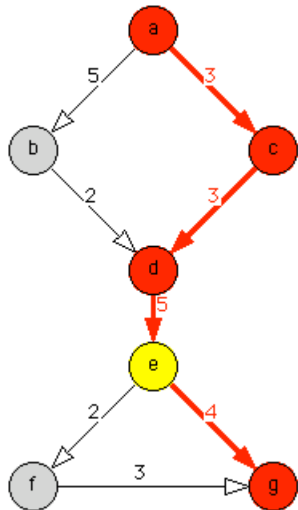
shortest paths:

	a	b	c	d	e	f	g
a		5	3	6	11	13	
b				2	7		
c				3	8		
d					5		
e						2	4
f							3
g							

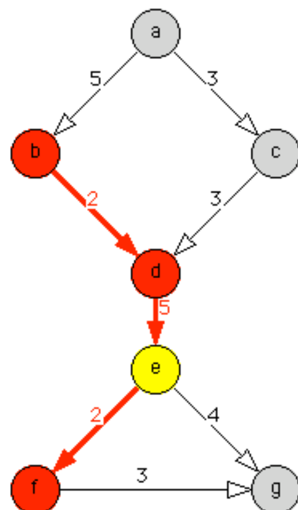
a - c -d -e -f (13)

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

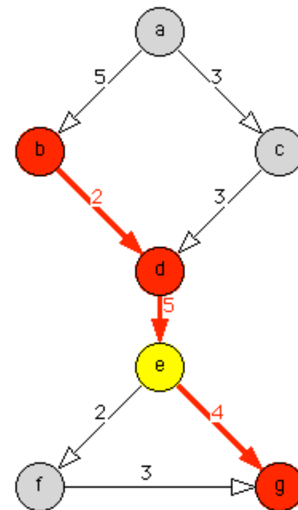
Floyd (3)



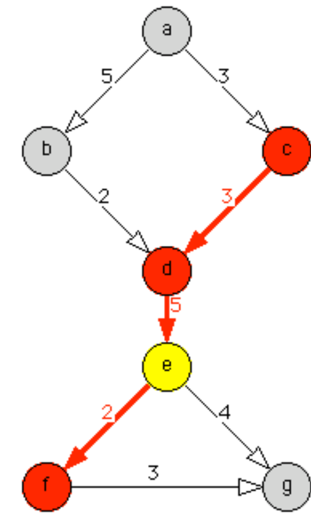
a - c -d -e -g (15)



b -d -e -f (9)



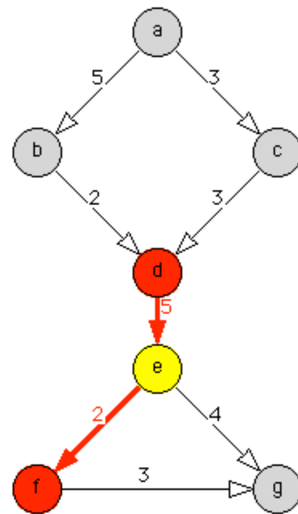
b -d -e -g (11)



c -d -e -f (10)

Helmut Schauer
Educational Engineering Lab
Department for Information Technology
University of Zurich

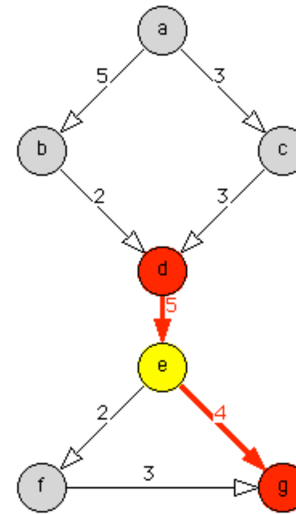
Floyd (4)



shortest paths:

	a	b	c	d	e	f	g
a		5	3	6	11	13	15
b				2	7	9	11
c				3	8	10	12
d					5	7	
e						2	4
f							3
g							

d -e -f (7)



shortest paths:

	a	b	c	d	e	f	g
a		5	3	6	11	13	15
b				2	7	9	11
c				3	8	10	12
d					5	7	9
e						2	4
f							3
g							

d -e -g (9)