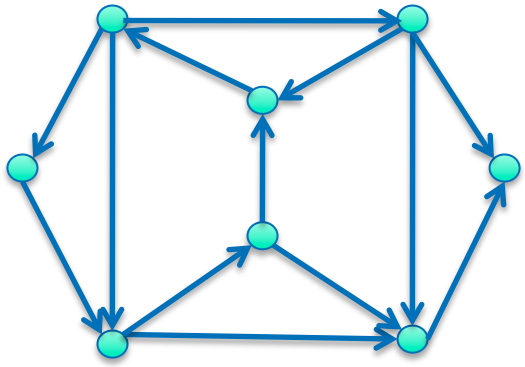
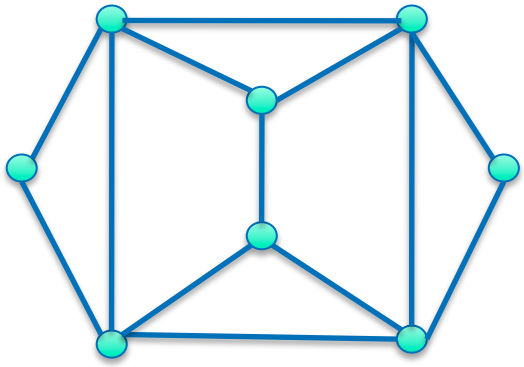
Graphen und klassische Graphprobleme

TECHNISCHE UNIVERSITÄT DARMSTADT

- Was ist ein Graph?
  - Ein ungerichteter **Graph** ist ein Paar G=(V,E), wobei $E \subseteq \{\{u,v\} \mid u,v \in V\}$.
    - Die Elemente von E sind ungeordnet.
    - Elemente aus V heißen **Knoten (engl.: node, vertex)**
      Elemente aus E heißen **Kanten (engl.: edge)**
  - Ein **gerichteter Graph** ist ebenfalls ein Paar G=(V,E), wobei aber dort Elemente von E geordnete Paare von Elementen aus V sind. Also $E \subseteq \{(u,v) \mid u,v \in V\}$.
    - Elemente aus V heißen Knoten
      Elemente aus E heißen **gerichtete Kanten** oder auch **Bögen**





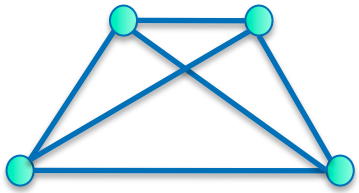
TECHNISCHE UNIVERSITÄT DARMSTADT

Nachbarschaftsbeziehungen

Inzidenz: Ein Knoten v heißt mit einer Kante e inzident, wenn gilt: $v \in e$.

Adjazenz: Zwei Knoten x und y heißen adjazent in G, wenn $\{x,y\} \in E$ ist.

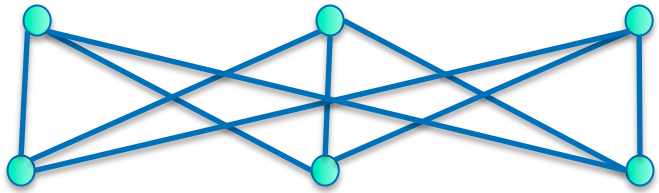
Grad: Der Grad eines Knotens v ist die Anzahl der mit v inzidenten Kanten.
Clique: Sind je 2 Knoten von G benachbart, heißt G vollständig. G wird auch Clique genannt.



TECHNISCHE UNIVERSITÄT DARMSTADT

Bipartite Graphen

Ein Graph heißt bipartit, wenn sich V in zwei disjunkte Teile V_1 und V_2 teilen läßt, so dass jede Kante in E einen Endknoten in V_1 und einen Endknoten in V_2 Besitzt.



Satz: Ein Graph ist genau dann bipartit, wenn er nur Kreise gerader Länge enthält.
Beweis: Übung

Graphen und klassische Graphprobleme

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Kodierungen von Graphen: Kantenlisten, Adjazenzmatrizen, Knoten-Kanten-Inzidenzmatrizen, Adjazenzlisten**

  – **Kantenliste**

  - Ist G=(V,E) ein Graph mit n=|V| Knoten und m=|E| Kanten, so sieht eine **Kantenliste** wie folgt aus:

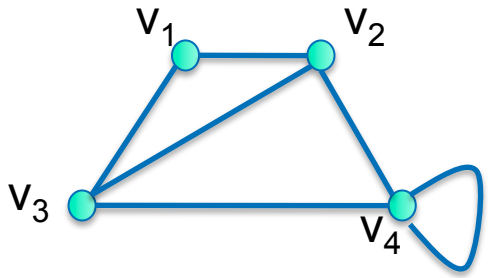
    $n,m,\{a_1,e_1\},\{a_2,e_2\},...,\{a_m,e_m\}$, mit $\{a_i,e_i\}$ sind Endknoten der Kante i

  - Die Reihenfolge der Kanten innerhalb der Liste ist beliebig
  - Bei Schleifen an einem Knoten v wird {v,v} eingefügt
  - Bei gerichteten Graphen ist zu beachten, dass immer zunächst der Anfangsknoten und dann der Endknoten repräsentiert wird

Graphen und klassische Graphprobleme

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiel:

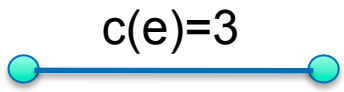


Kantenliste: $4,6,\{v_1,v_2\},\{v_2,v_4\},\{v_4,v_4\},\{v_3,v_4\},\{v_2,v_3\},\{v_3,v_1\}$

Graphen und klassische Graphprobleme

TECHNISCHE UNIVERSITÄT DARMSTADT

Kanten können auch ein Gewicht oder Kosten haben. Dann gibt es eine Kostenfunktion c: E → ℚ



Kodiert werden die Gewichte, indem man für jede Kante das Gewicht dazu notiert.

Eine Kantenliste erfordert 2m+2 Speicherstellen zur Speicherung,
Eine Kantenliste mit Gewichten 3m+2 Zellen

Graphen und klassische Graphprobleme

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Kodierungen von Graphen: Kantenlisten, Adjazenzmatrizen, Knoten-Kanten-Inzidenzmatrizen, Adjazenzlisten**

  – **Adjazenzmatrix**

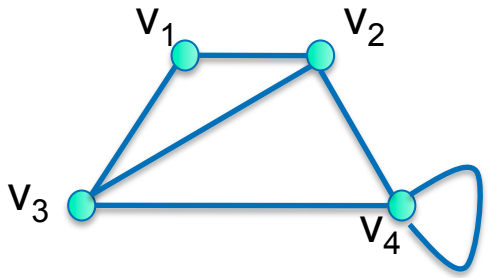
  - Sei G=(V,E) ein einfacher (d.h., er hat höchstens eine Kante je Knotenpaar) Graph mit den Knoten 1,...,n. Dann heißt die Matrix $A \in \mathbb{R}^{n \times n}$ mit
    - A_{ij} = 1, falls i und j durch eine Kante verbunden sind,
    - A_{ij} = 0 sonst

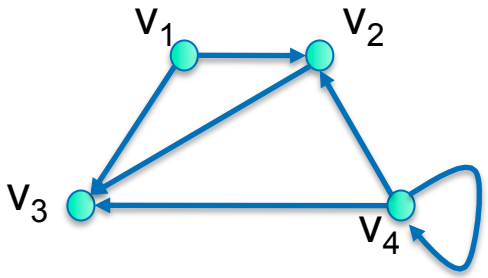
    die **Adjazenzmatrix** von G.
  - Falls G ungerichteter Graph ist, ist A symmetrisch und es reicht, die obere Dreiecksmatrix zu speichern.
  - Hat G Kantengewichte, so setzt man $A_{ij} = c((i,j))$, falls $(i,j) \in E$, und $A_{ij} = 0, +\infty$, oder $-\infty$ andernfalls, je nach Bedarf
  - Speicherbedarf beträgt $O(n^2)$ Zellen.

Graphen und klassische Graphprobleme

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiele:





Adjazenzmatrix:

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

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

Graphen und klassische Graphprobleme

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Kodierungen von Graphen: Kantenlisten, Adjazenzmatrizen, Knoten-Kanten-Inzidenzmatrizen, Adjazenzlisten**

  – **Knoten-Kanten-Inzidenzmatrix**

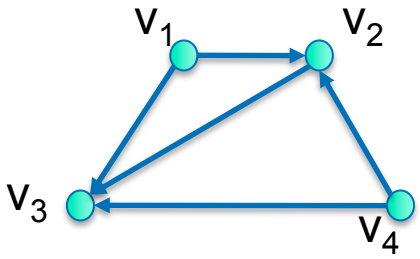
  - Sei G=(V,E) ein einfacher gerichteter Graph mit den Knoten 1,...,n. Dann heißt die Matrix $A \in \{-1,0,1\}^{|V| \times |E|}$ mit
    - A_{ij} = 1, falls Kante j verläßt Knoten i,
    - A_{ij} = -1, falls Kante j zeigt in Knoten i hinein,
    - A_{ij} = 0 sonst

    die **Knoten-Kanten-Inzidenzmatrix** von G.
  - Speicherbedarf beträgt $O(|V| \cdot |E|)$ Zellen.

Graphen und klassische Graphprobleme

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiele:



Knoten-Kanten-Inzidenzmatrix:

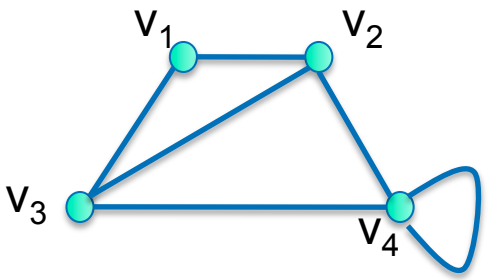
	v_1 v_2	v_1 v_3	v_2 v_3	v_4 v_2	v_4 v_3
v_1	1	1	0	0	0
v_2	-1	0	1	-1	0
v_3	0	-1	-1	0	-1
v_4	0	0	0	1	1

Graphen und klassische Graphprobleme

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Kodierungen von Graphen: Kantenlisten, Adjazenzmatrizen, Knoten-Kanten-Inzidenzmatrizen, Adjazenzlisten**

  – **Adjazenzliste**
  Speichert man für einen Graphen G=(V,E) die Anzahl von Knoten, und für jeden Knoten seinen Grad und seine Nachbarknoten, so nennt man solch eine Datenstruktur eine Adjazenzliste von G. Speicherbedarf: O(n+m)



#Knoten	Knoten-nummer	Grad	Nachbarn
1	1	2	2,3
	2	3	1,3,4
	3	3	1,2,4
	4	3	2,3,4

Allgemeine und spezielle Untergraphen

TECHNISCHE UNIVERSITÄT DARMSTADT

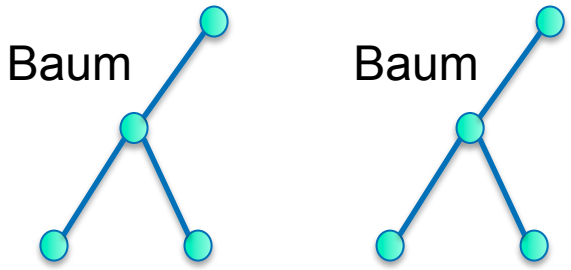
- Wege und Kreise
  - Ein **Weg** (oder **Pfad**) in einem Graphen (gerichtet oder ungerichtet) G = (V,E) ist eine Folge von Knoten $(v_0,v_1,...,v_k)$ aus V, so dass für alle $i \in \{1,...,k\}$ gilt, dass $(v_{i-1},v_i) \in E$ (bzw. $\{v_{i-1},v_i\} \in E$) eine Kante ist.
  - Ein Weg heißt **einfach**, falls in der Folge $(v_1,...,v_k)$ kein Knoten mehrfach auftritt.
  - Ein einfacher Weg heißt **Kreis**, falls $v_0 = v_k$.
  - Ein Graph heißt **azyklisch** (oder kreisfrei), falls es in ihm keinen Kreis gibt. (Gerichtete, azyklische Graphen werden im Englischen DAG genannt, für **directed acyclic graph**)

Vorsicht: Diese Definitionen sind so, wie sie meistens in der Literatur auftreten. Sie sind ein klein wenig anders, als in den Vorjahren der ADM-Vorlesungen, wo es eine Unterscheidung zwischen Weg und Pfad gab.

Allgemeine und spezielle Untergraphen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zusammenhangskomponenten, Bäume und Wälder
  - Ein Graph heißt **zusammenhängend**, falls es zu jedem Knotenpaar $v,w \in V$ einen Weg von v nach w gibt.
  - Die zusammenhängenden Teile von G heißen **Zusammenhangskomponenten**
  - Ein **Baum** ist ein zusammenhängender Graph, der keine Kreise enthält
  - Ein **Wald** ist ein Graph, der keine Kreise enthält, also eine Ansammlung von Bäumen.

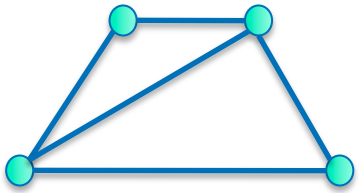


zusammen: ein Wald

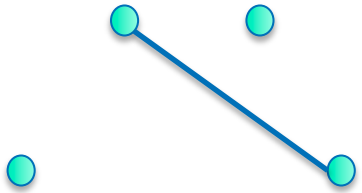
Allgemeine und spezielle Untergraphen

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Komplementärgraph**
  - Der zu G **komplementäre Graph** G‘ ist der Graph
    G‘=(V,E‘), mit (i,j)∈E‘ ⇔ (i,j)∉E



Graph



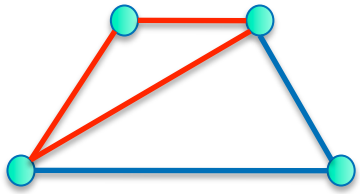
Komplement

  - Satz: Mindestens einer der Graphen G oder G‘ ist zusammenhängend.
    Beweis Übung.

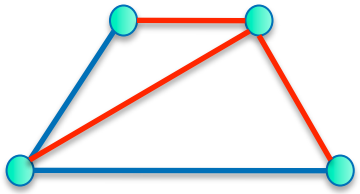
Allgemeine und spezielle Untergraphen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Untergraphen, aufspannende Untergraphen
  - G‘=(V‘,E‘) heißt **Untergraph** (oder **Teilgraph**) von G=(V,E), falls V‘⊆V und E‘⊆E ist.
  - G‘ ⊆ G heißt **aufspannender Untergraph** von G, falls zusätzlich gilt V‘=V



Graph mit Untergraph



aufspannender Baum

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Beispiele für Problemstellungen
  Sei G=(V,E) ein Graph (gerichtet oder ungerichtet).

  – Problem 1:
    - Eingabe: Graph G in Adjazenzlistenform.
    - Ausgabe: „ja“, wenn G einen Kreis enthält? Sonst „nein“
  – Problem 2:
    - Eingabe: Graph G in Adjazentlistenform.
    - Anzahl der Zusammenhangskomponenten von G?
  – Problem 3:
    - Eingabe: Graph G in Adjazentlistenform.
    - Für jede Zusammenhangskomponente einen aufspannenden Baum.

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Tiefensuche (engl. Depth-First-Search)

  – **Input**: Graph G=(V,E) in Form einer Adjazenzliste, d.h. für jeden Knoten v ist eine Adjazenzliste adj[v] gegeben.
  Output: Kantenmenge T (aufspannender Wald) und Kantenmenge B mit $B \cup T = E$ und $B \cap T = \varnothing$.

  – Idee:
    - entdecke neue Knoten ausgehend vom zuletzt besuchten
    - Sind alle adjazenten Knoten des zuletzt gefundenen Knotens v bereits entdeckt, springe zu dem Knoten zurück, von dem aus v entdeckt wurde. Rekursives Hinzufügen von Kanten zum Wald T oder zu kreisschließenden Kanten B
    - Wenn unentdeckte Knoten übrigbleiben, starte Tiefensuche von einem dieser Knoten neu.

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Tiefensuche (engl. Depth-First-Search)

  – Zur Orientierung im folgenden Algorithmus:
    - Zu Beginn:alle Knoten werden weiß gefärbt
    - Entdeckte Knoten werden grau
    - Fertig abgearbeitete Knoten werden schwarz gefärbt
    - Es gibt zwei Zeitstempel: d[v] und f[v] (zwischen 1 und 2|V|)
    - d[v]: v wird entdeckt (discover-time)
    - f[v]: v ist fertig bearbeitet (finishing-time)

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.  time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

Depth-**F**irst-**S**earch (dt.: Tiefensuche)
engl. für Knoten: node, vertex
„:=“: Variablenzuweisung
π speichert predecessor (Vorgänger)

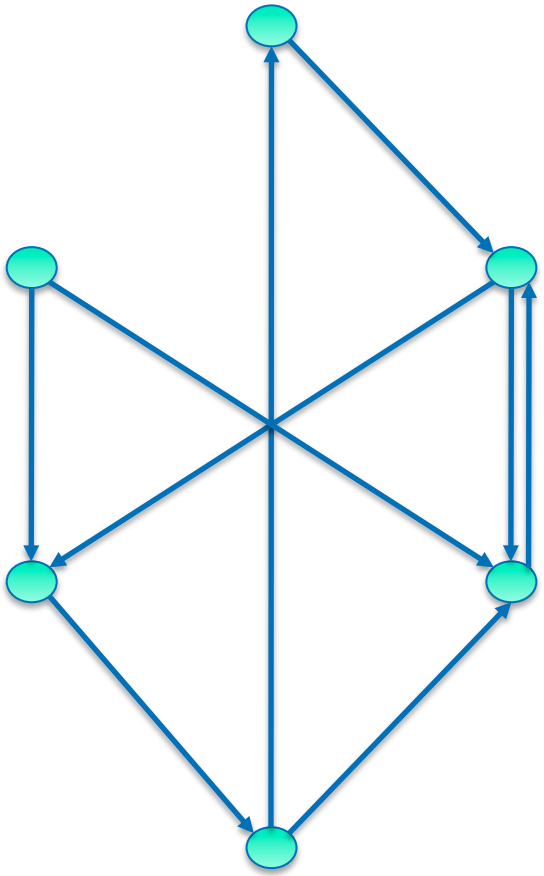
„==“: Vergleich

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.  time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

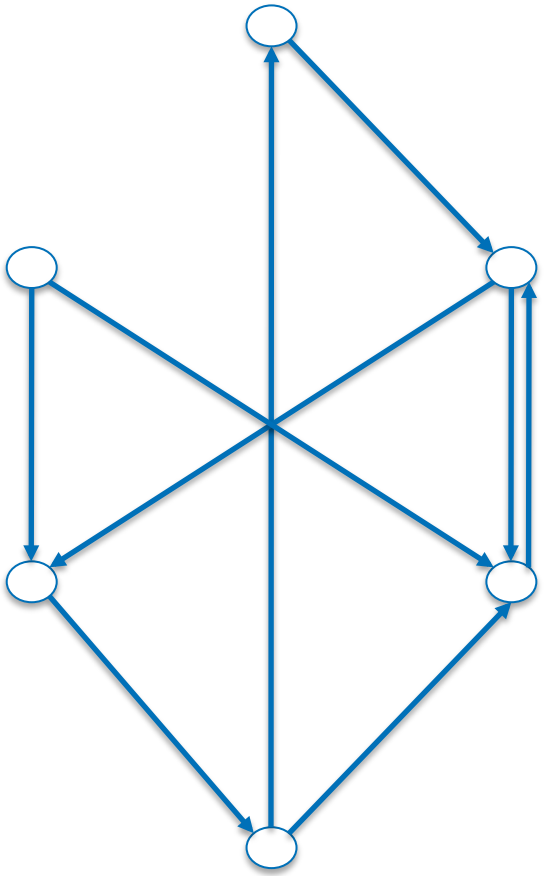


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.  time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```



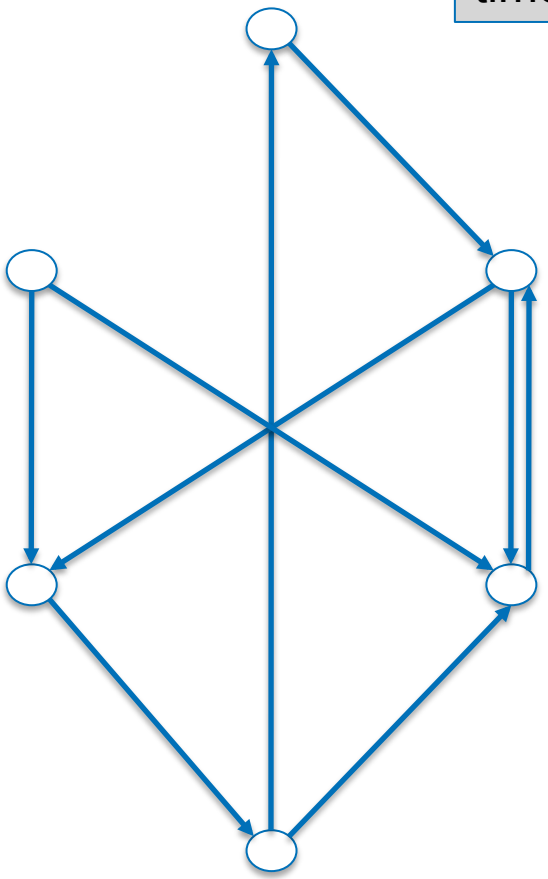
Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.  time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

time = 0

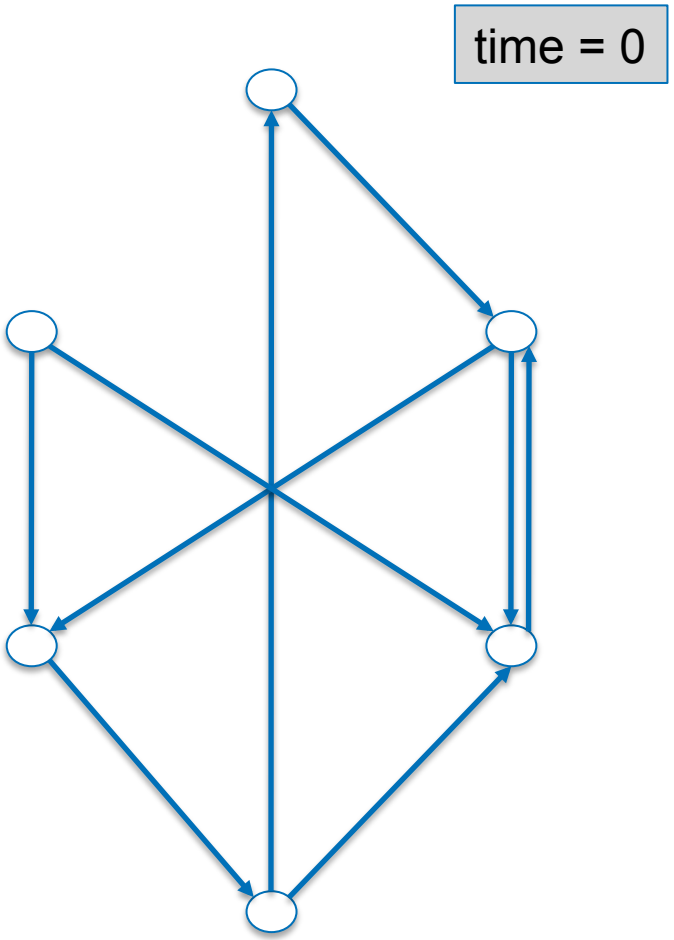


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

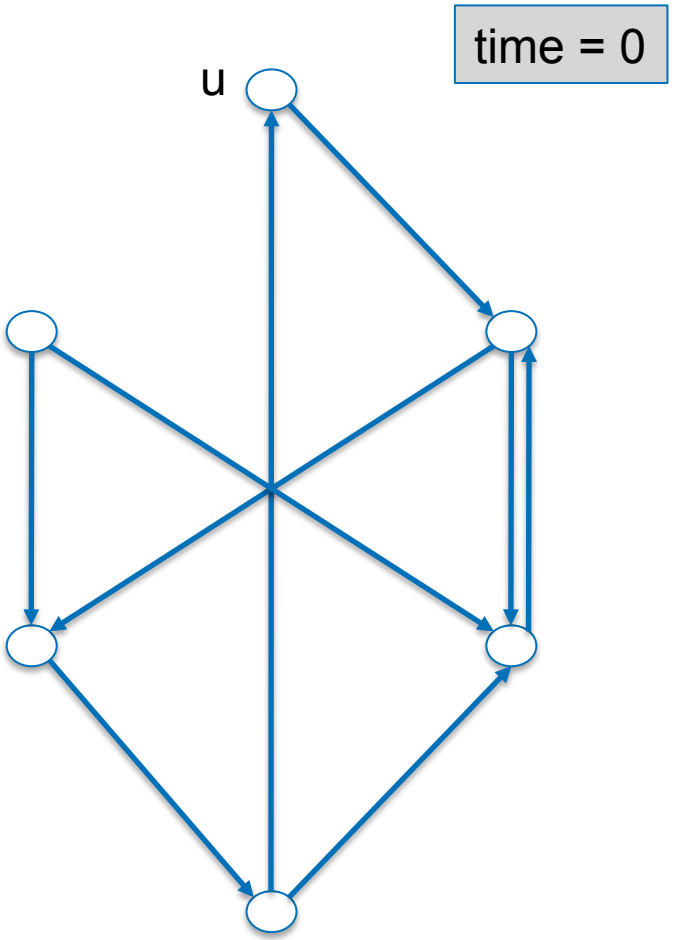


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

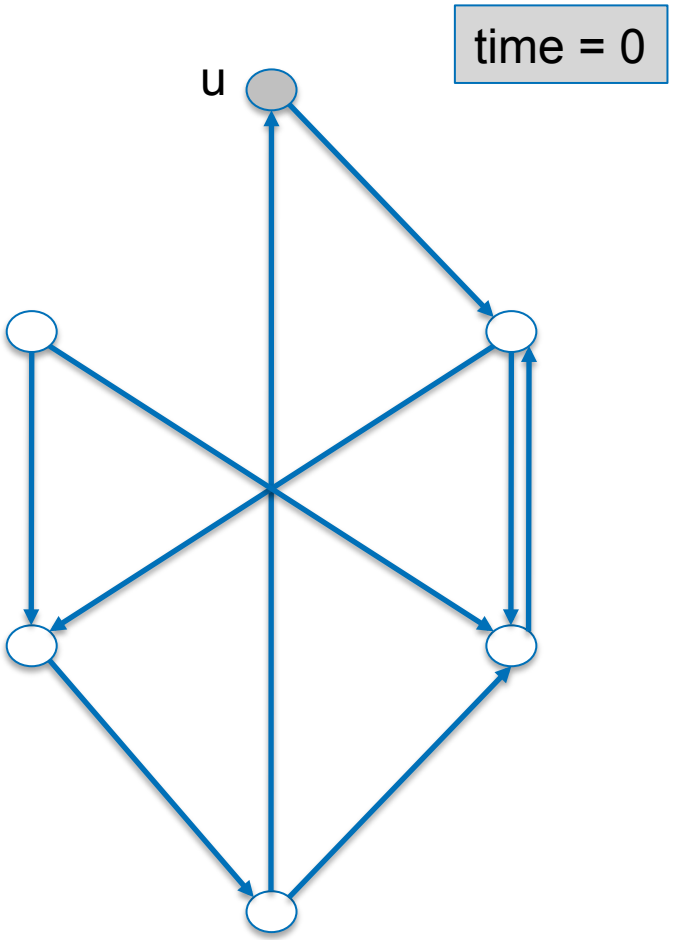


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

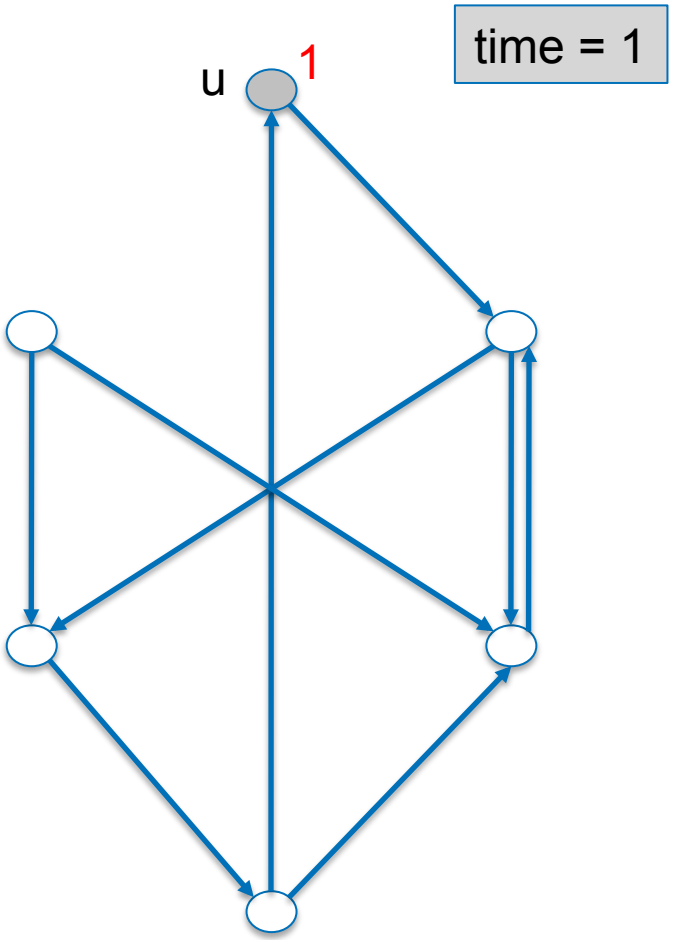


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

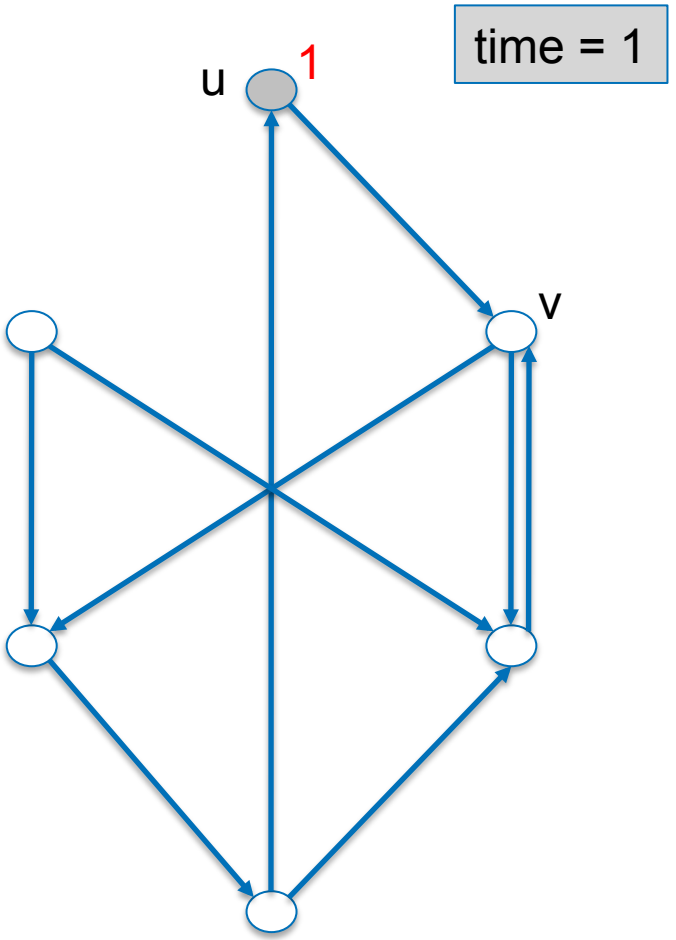


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1. for each node u∈V do
2.         color[u] := white;
3.         π[u] := nil;
4.  time := 0;
5. for each node u∈V do
6.         if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.         if color[v]==white then
5.                 π[v]:=u;
6.                 DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

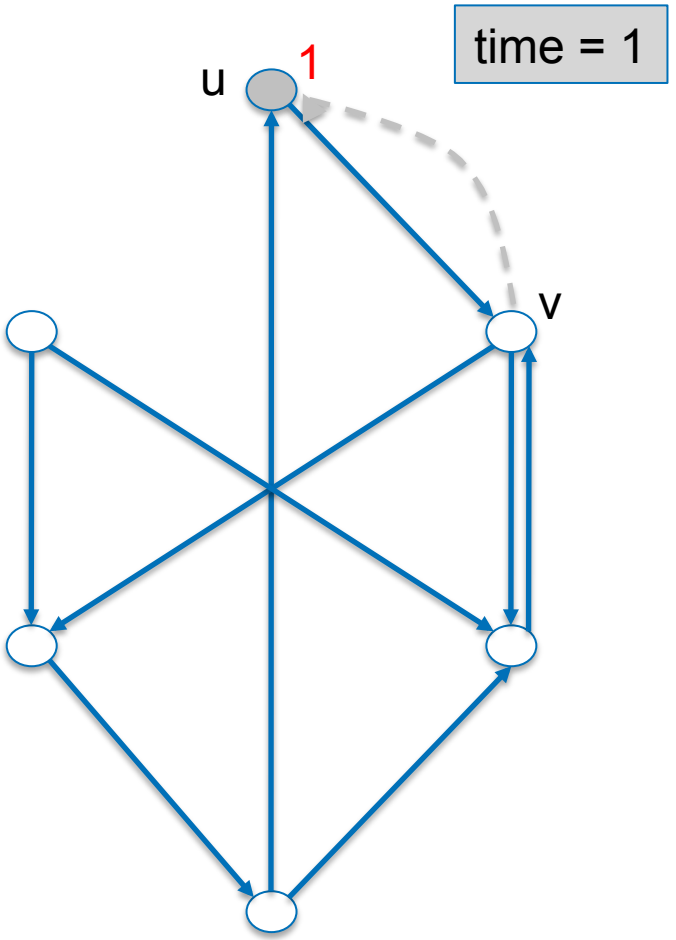


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1. for each node u∈V do
2.         color[u] := white;
3.         π[u] := nil;
4.  time := 0;
5. for each node u∈V do
6.         if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3. for each node v∈adj[u] do
4.         if color[v]==white then
5.                 π[v]:=u;
6.                 DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

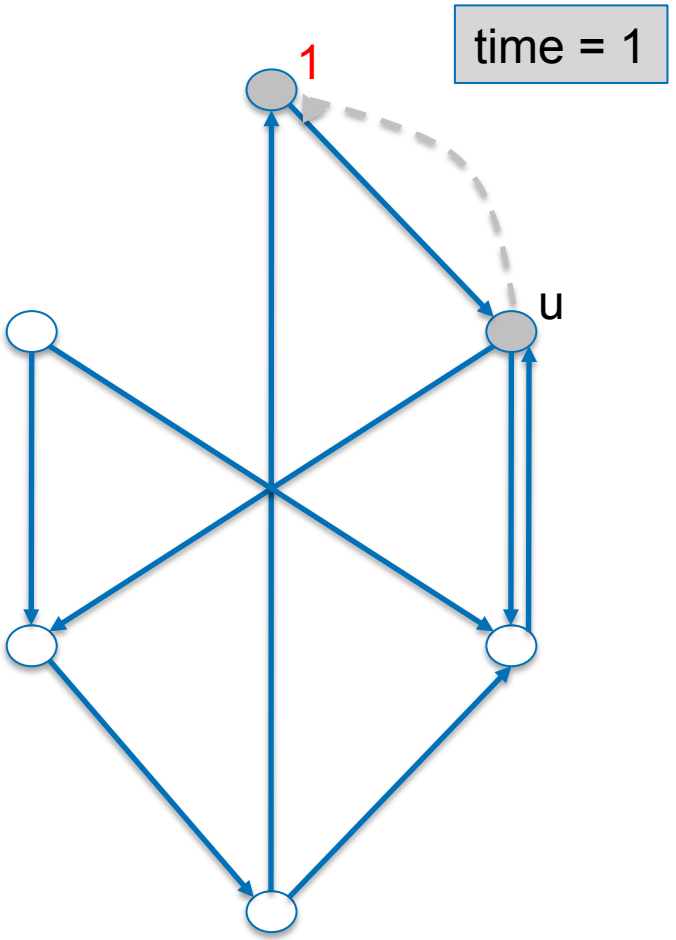


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

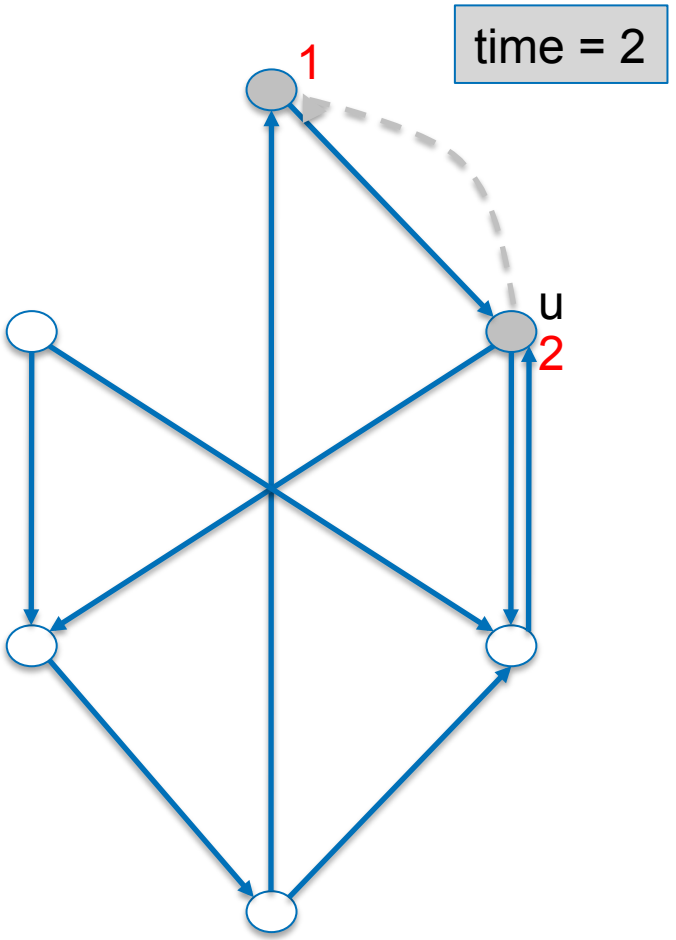


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1. for each node u∈V do
2.         color[u] := white;
3.         π[u] := nil;
4.  time := 0;
5. for each node u∈V do
6.         if color[u]==white then DFS-Visit(u);
```

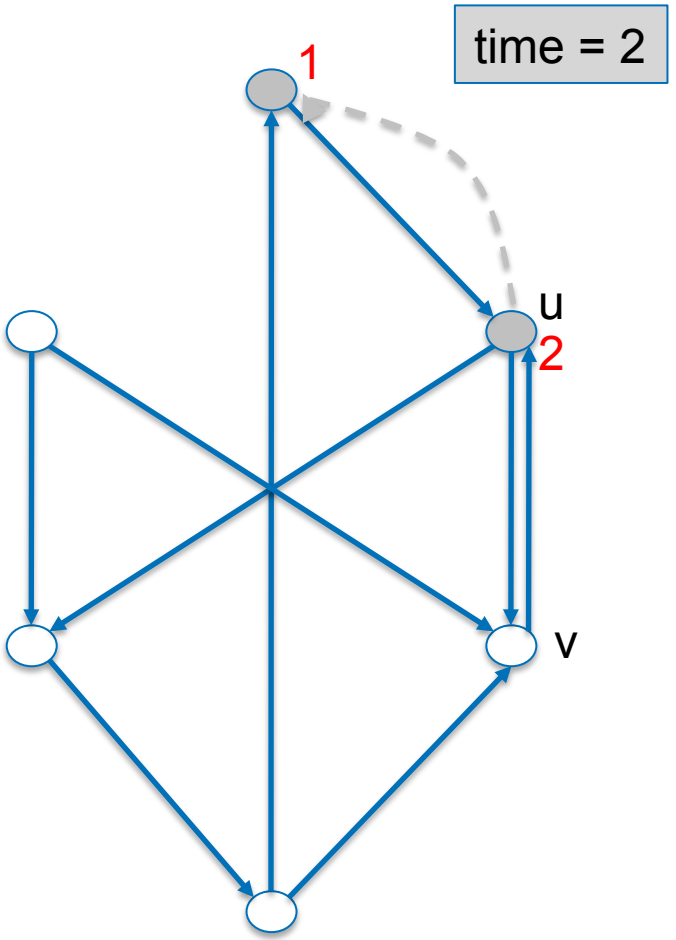
```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3. for each node v∈adj[u] do
4.         if color[v]==white then
5.                 π[v]:=u;
6.                 DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```



TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

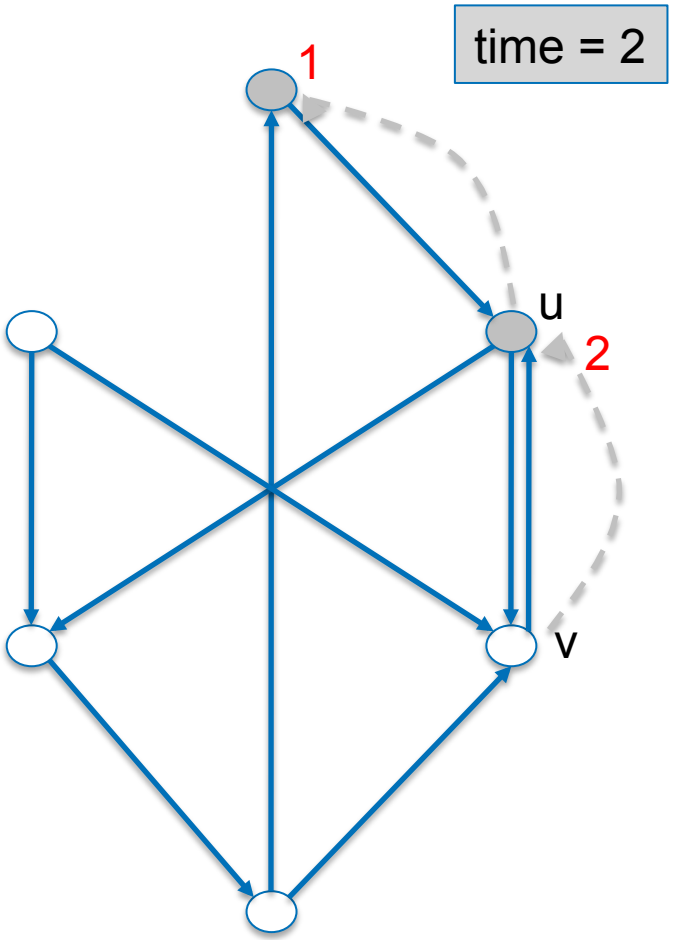


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1. for each node u∈V do
2.         color[u] := white;
3.         π[u] := nil;
4.  time := 0;
5. for each node u∈V do
6.         if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3. for each node v∈adj[u] do
4.         if color[v]==white then
5.                 π[v]:=u;
6.                 DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

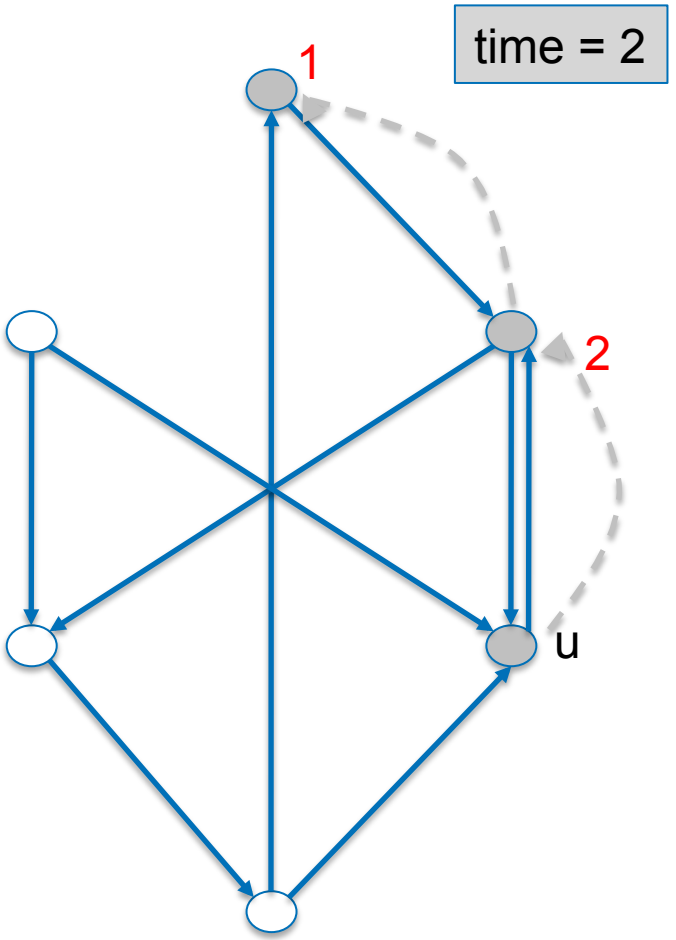


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

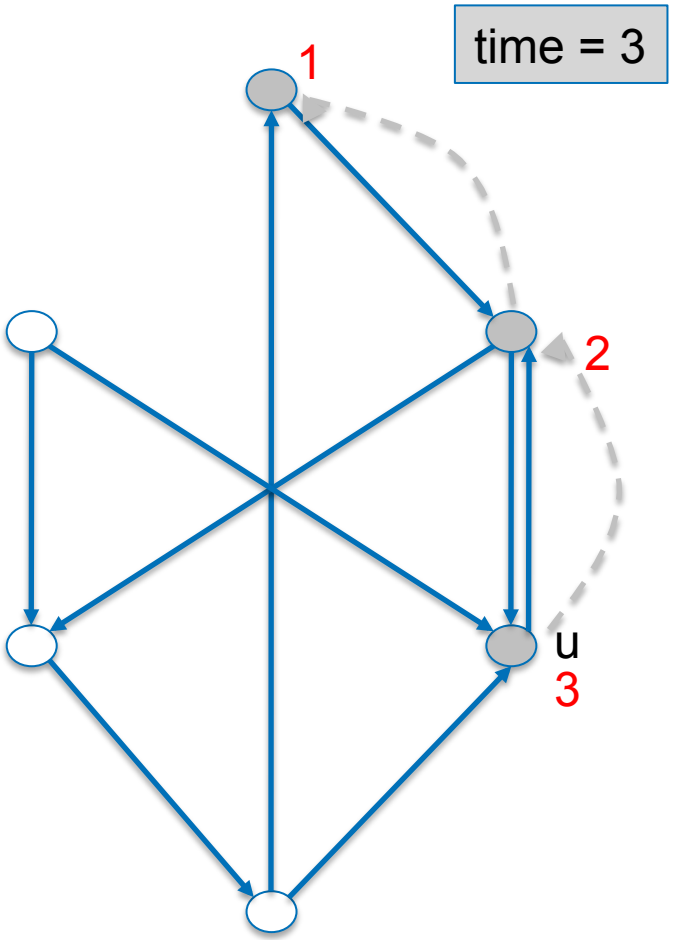


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

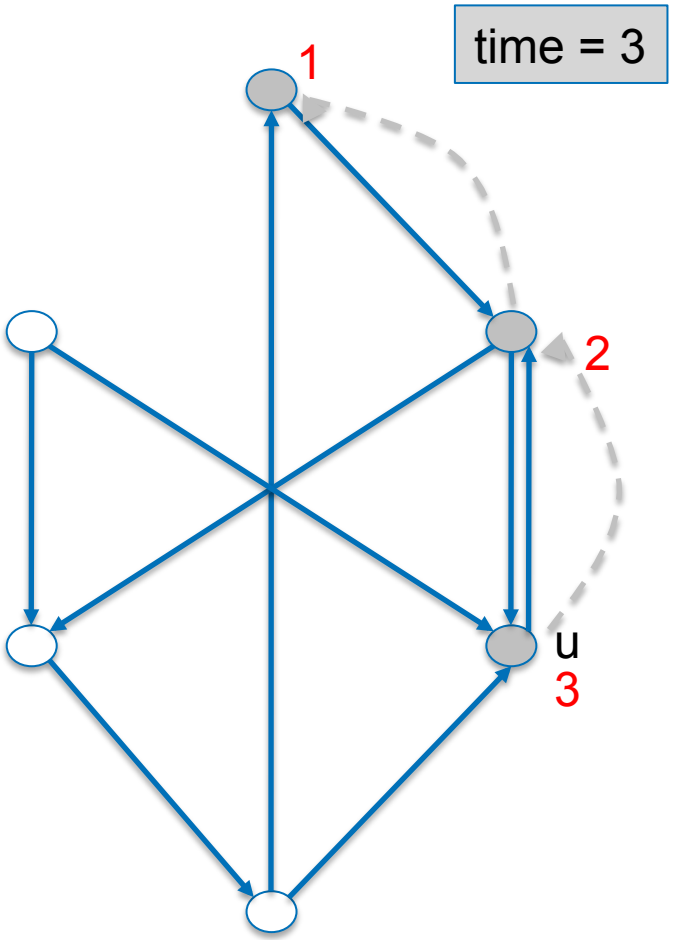


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

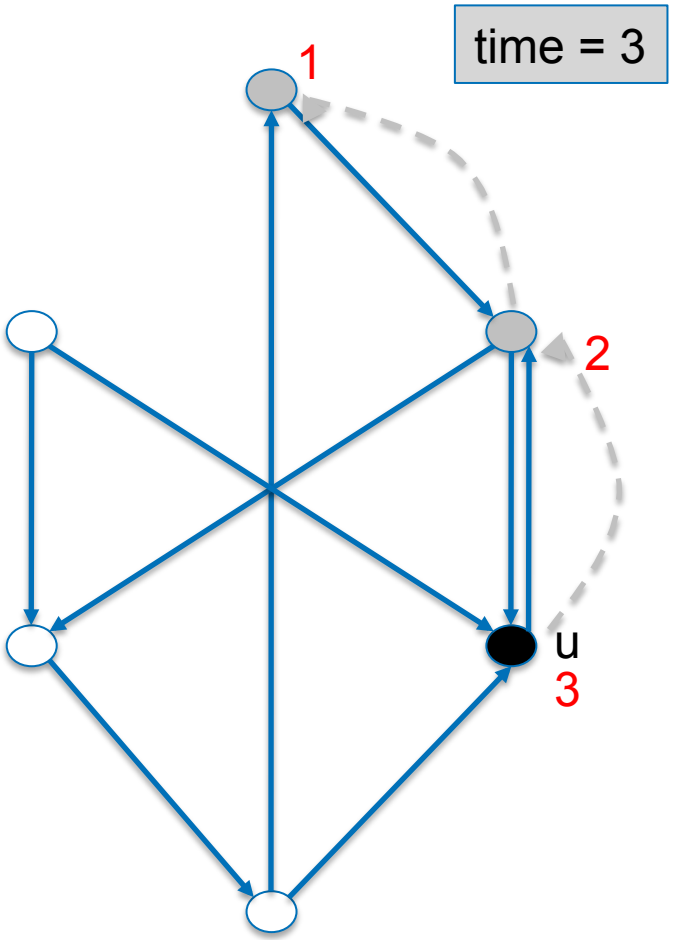


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

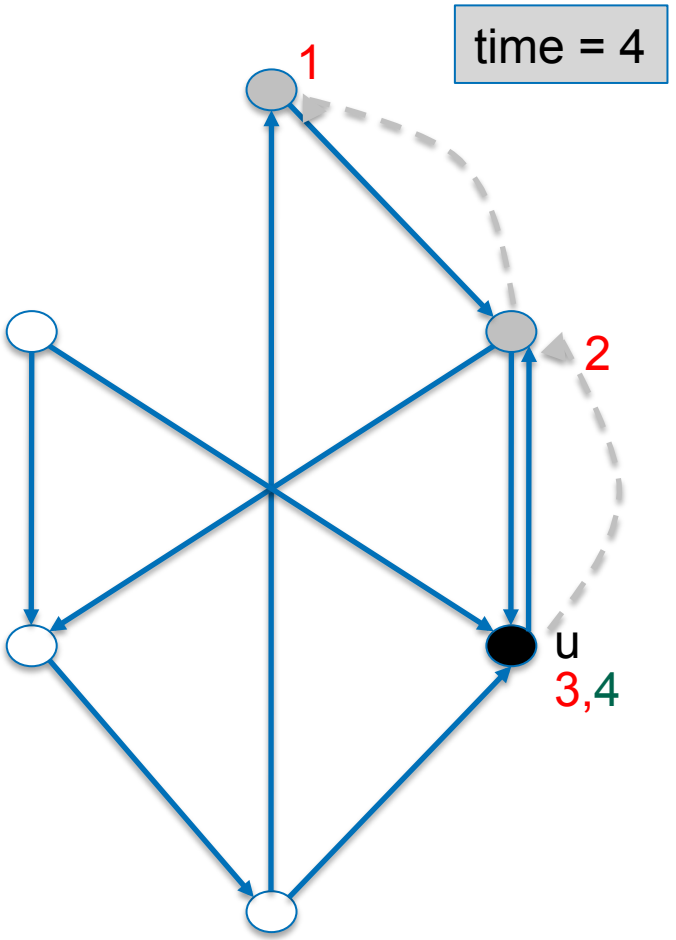


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

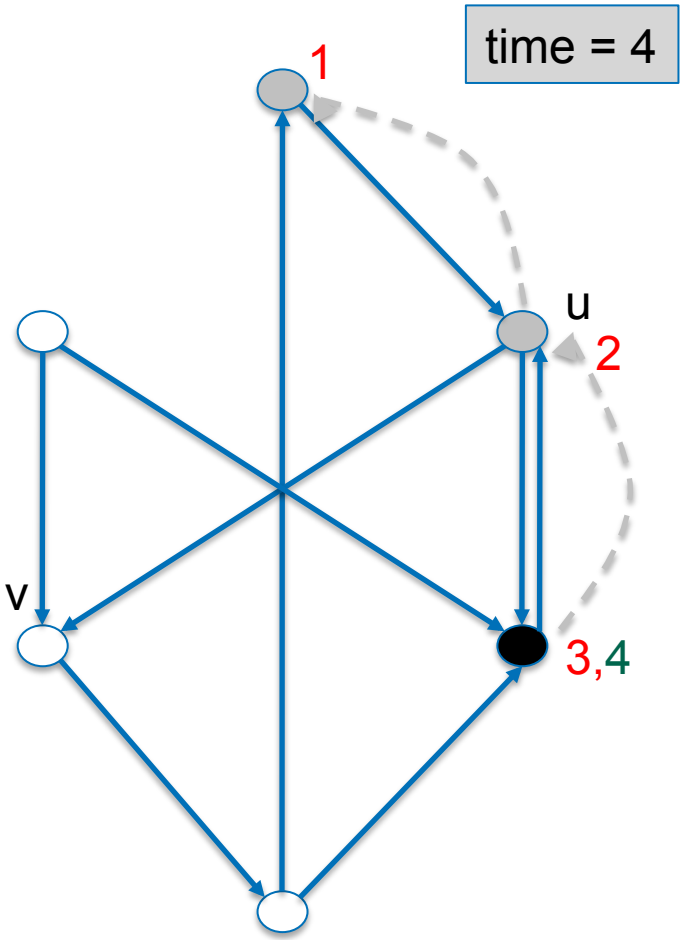


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

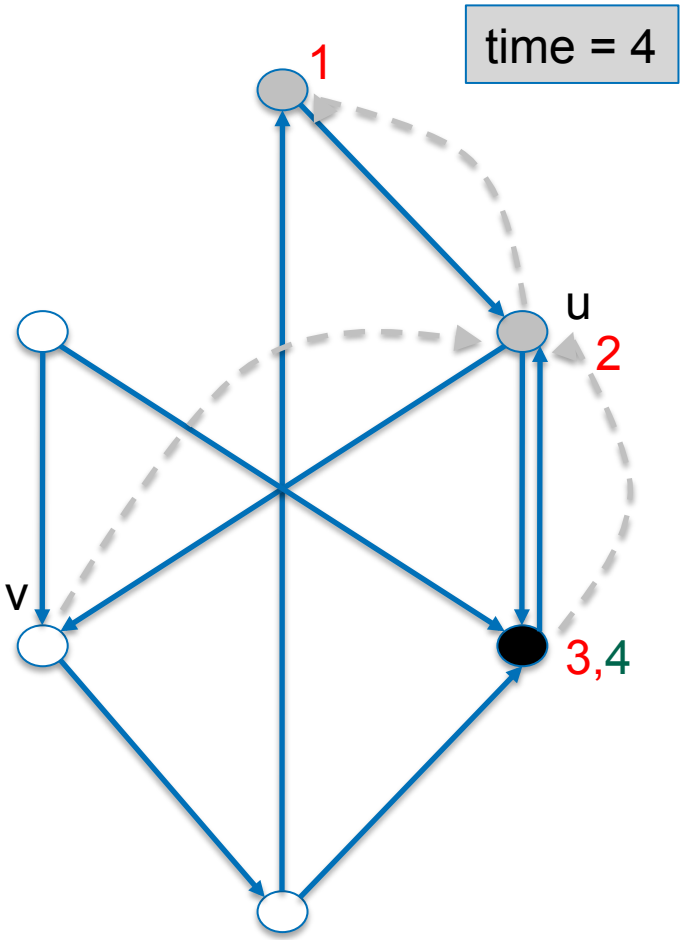


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

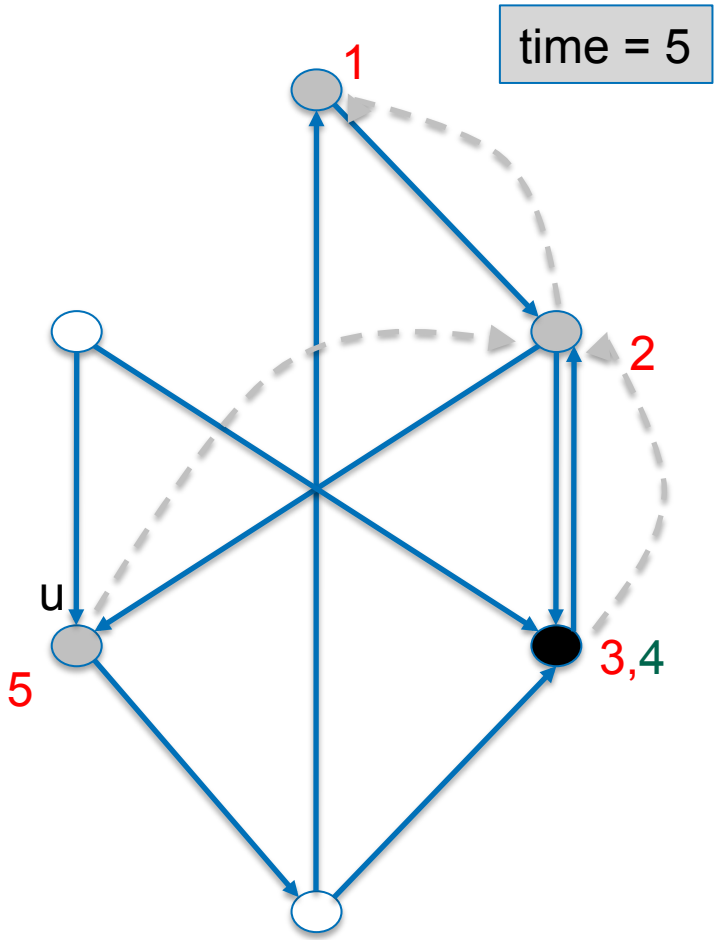


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

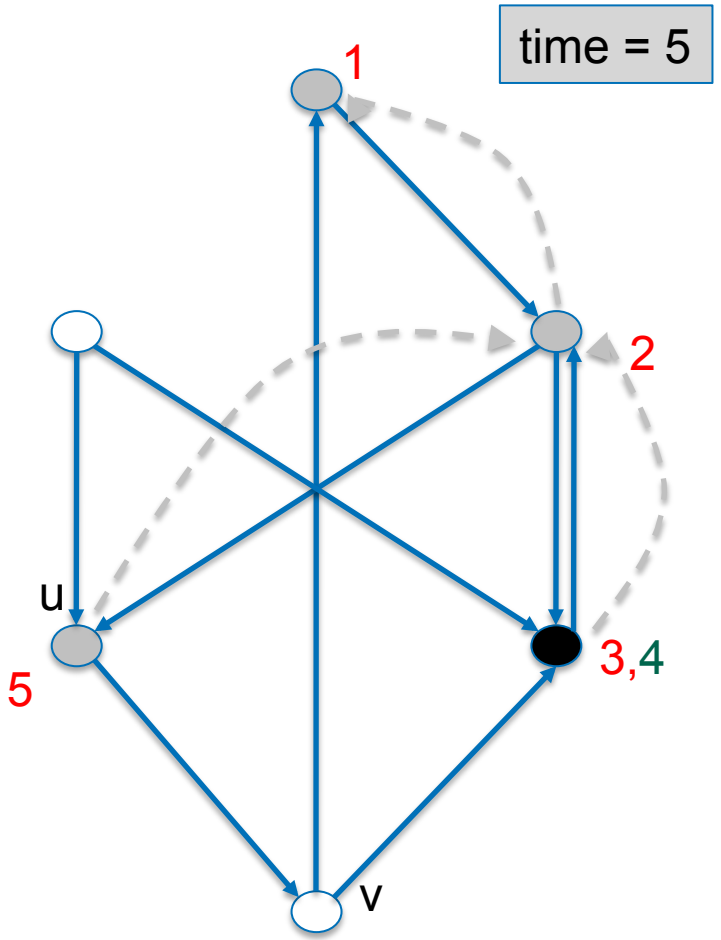


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

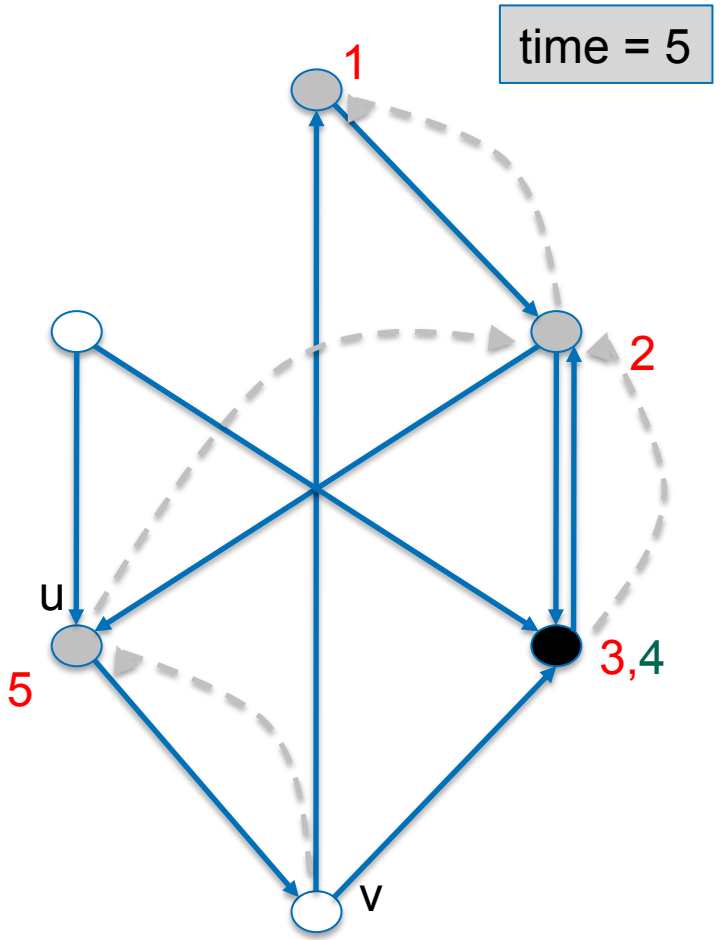


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

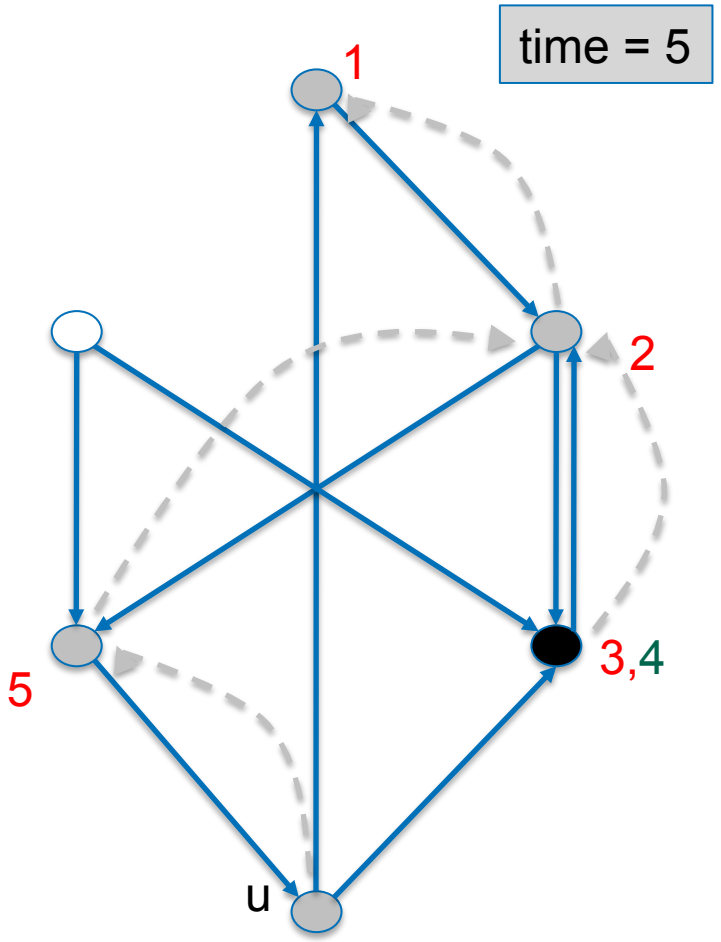


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

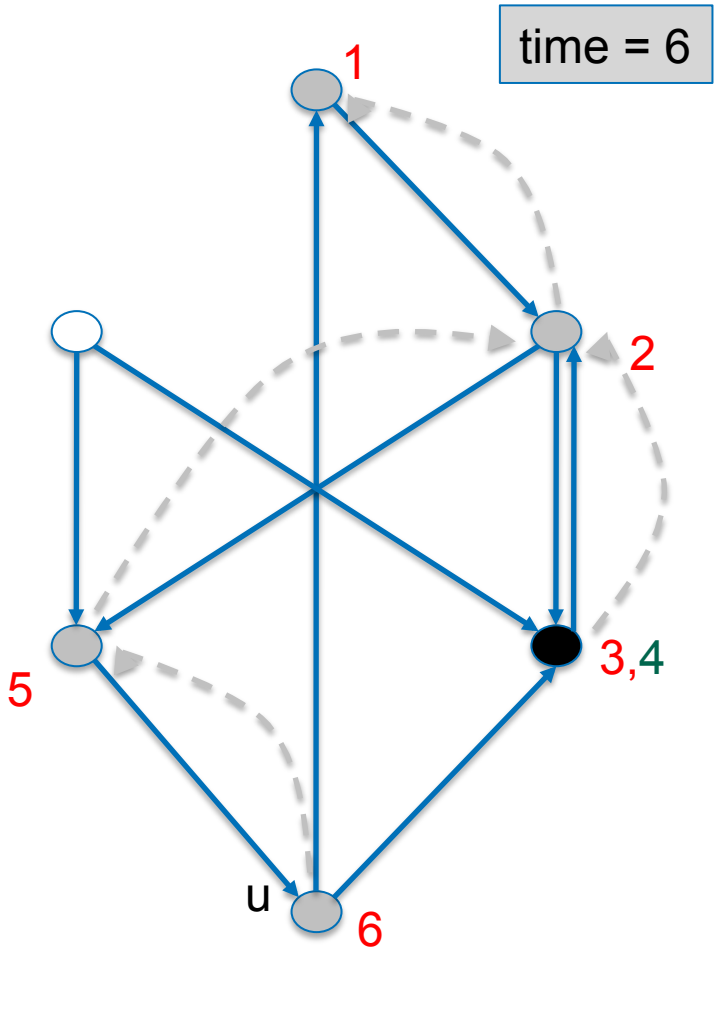


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

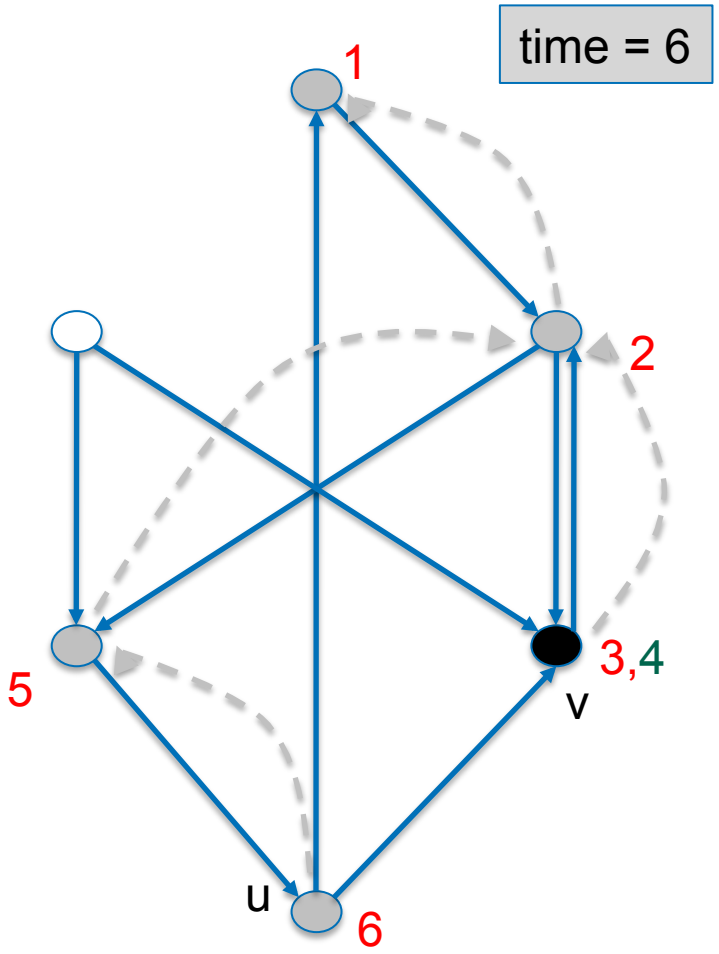


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1. for each node u∈V do
2.         color[u] := white;
3.         π[u] := nil;
4.  time := 0;
5. for each node u∈V do
6.         if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.         if color[v]==white then
5.                 π[v]:=u;
6.                 DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

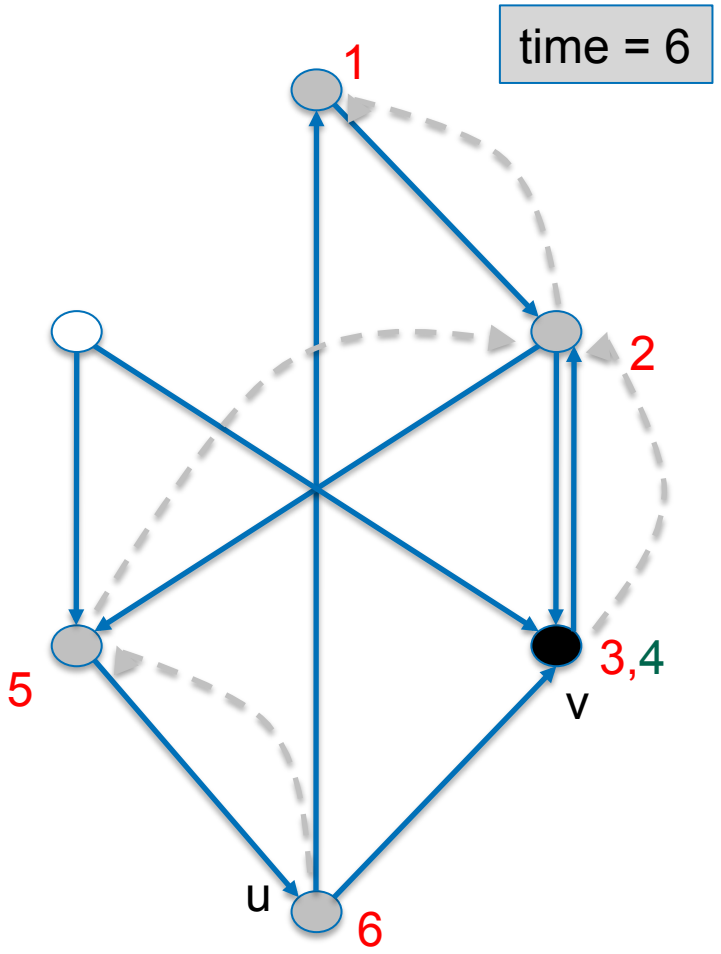


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

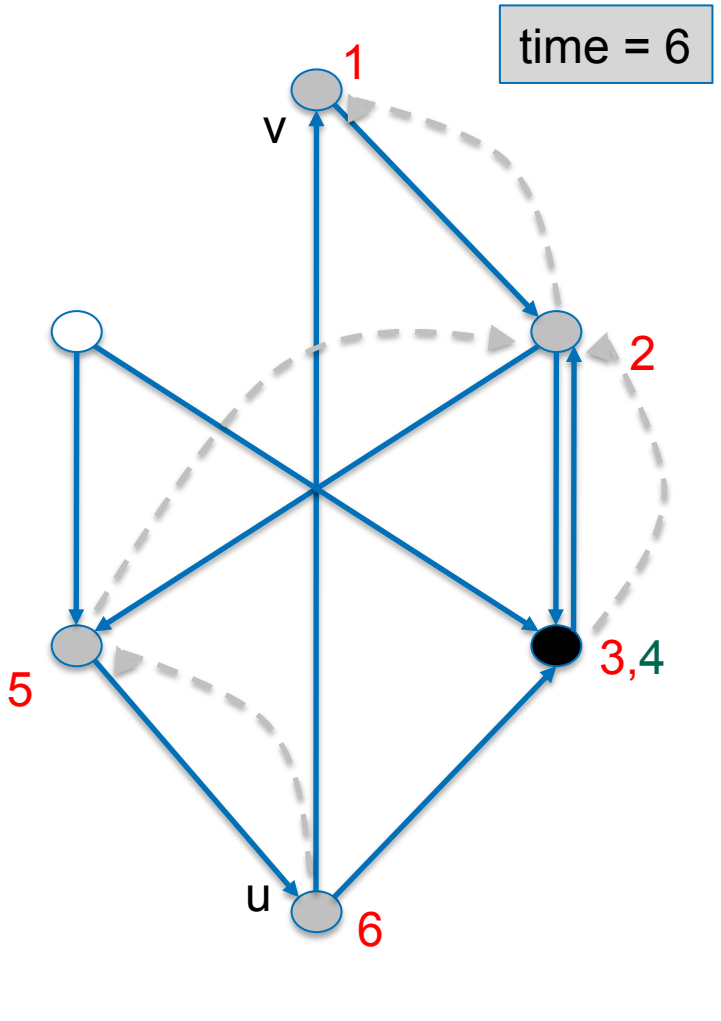


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

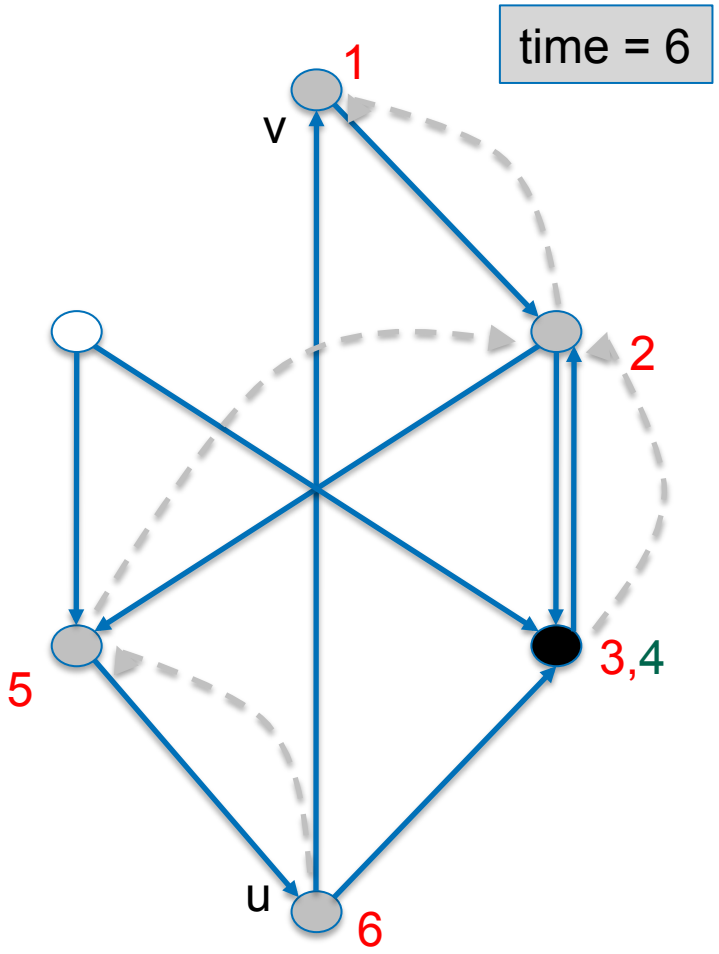


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1. for each node u∈V do
2.         color[u] := white;
3.         π[u] := nil;
4.  time := 0;
5. for each node u∈V do
6.         if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3. for each node v∈adj[u] do
4.         if color[v]==white then
5.                 π[v]:=u;
6.                 DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

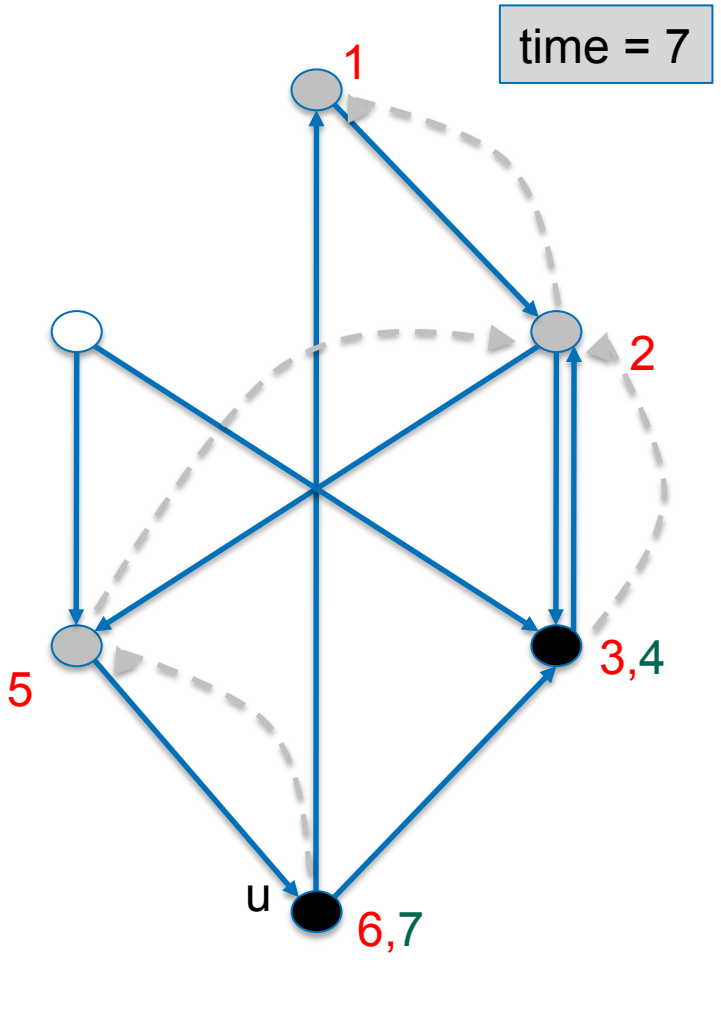


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

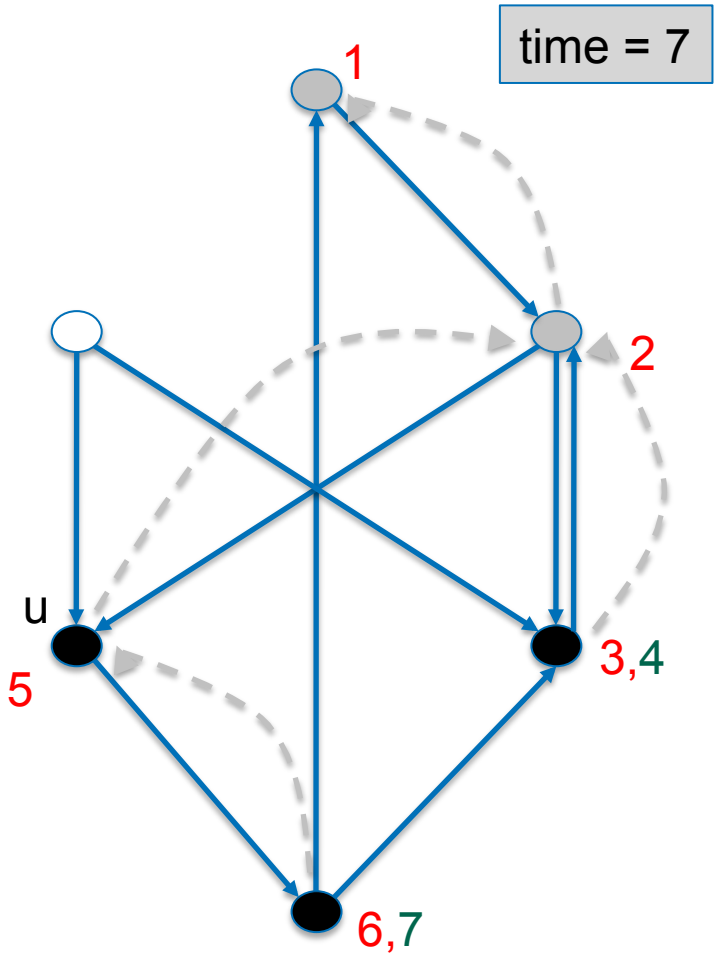


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

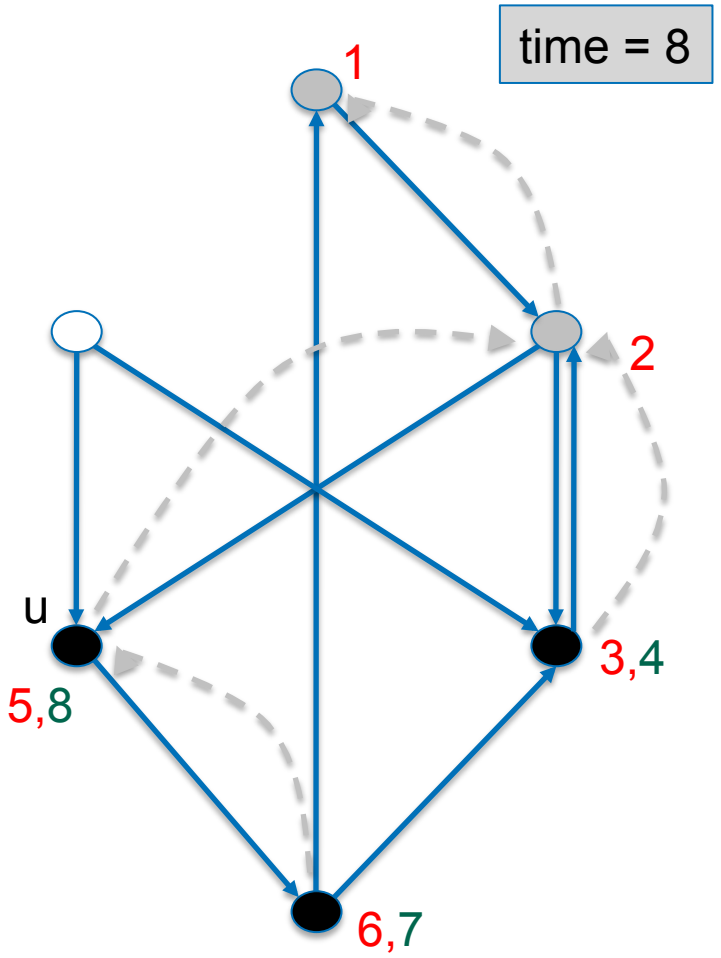


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

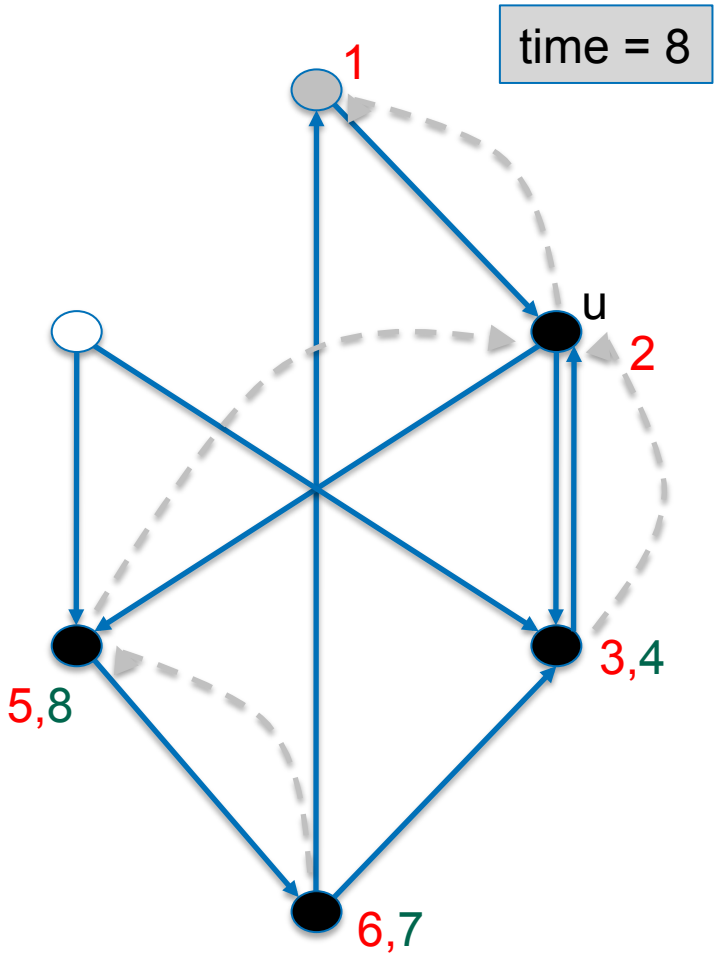


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

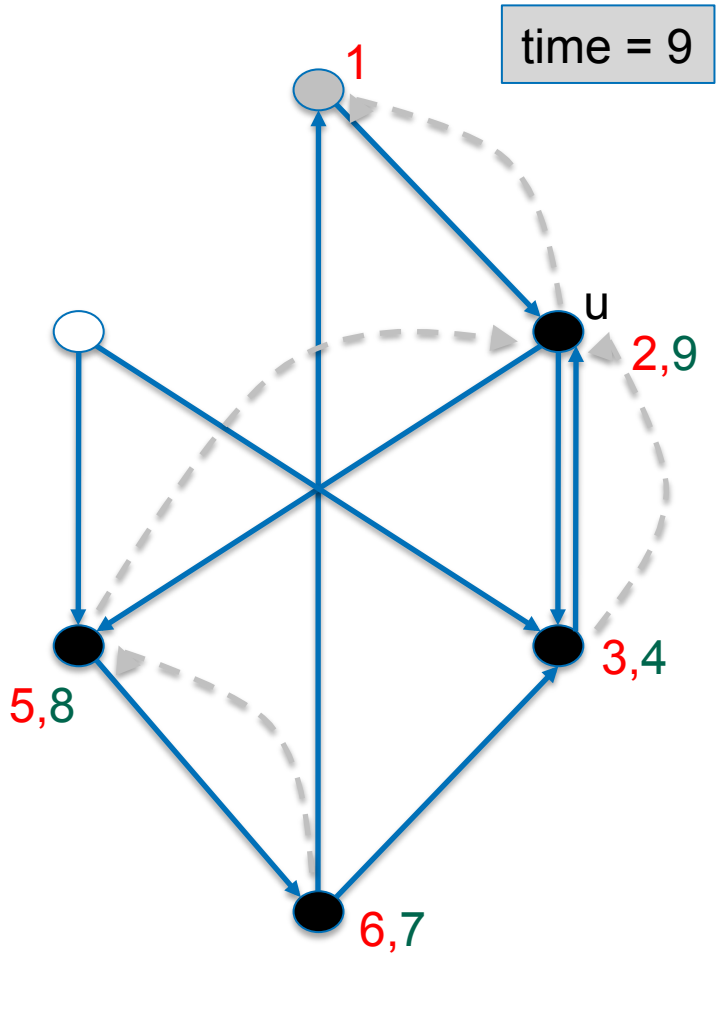


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

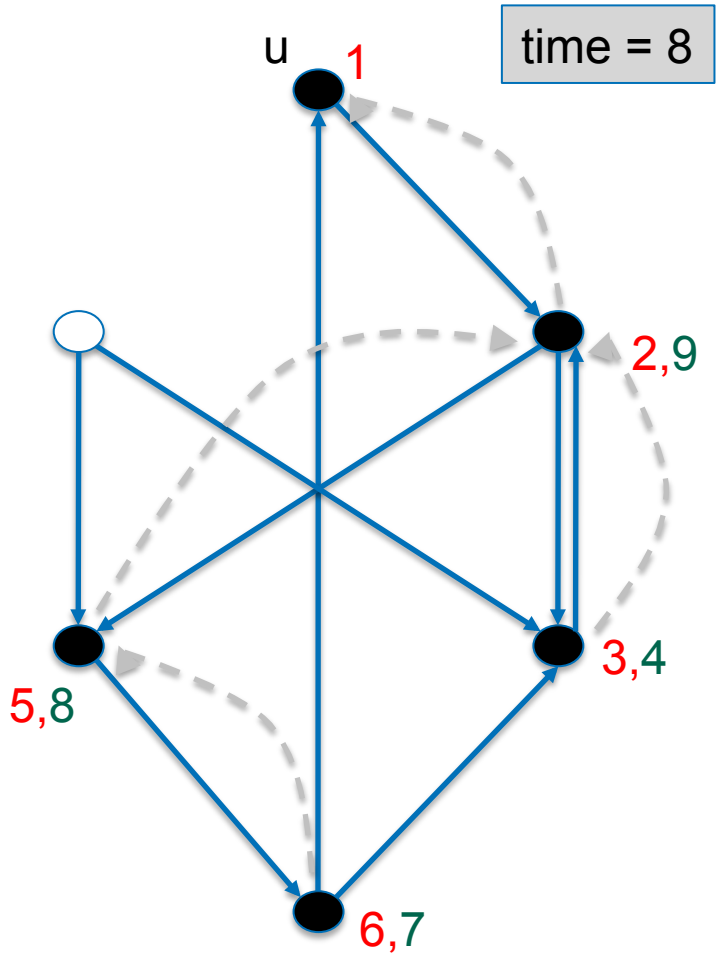


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

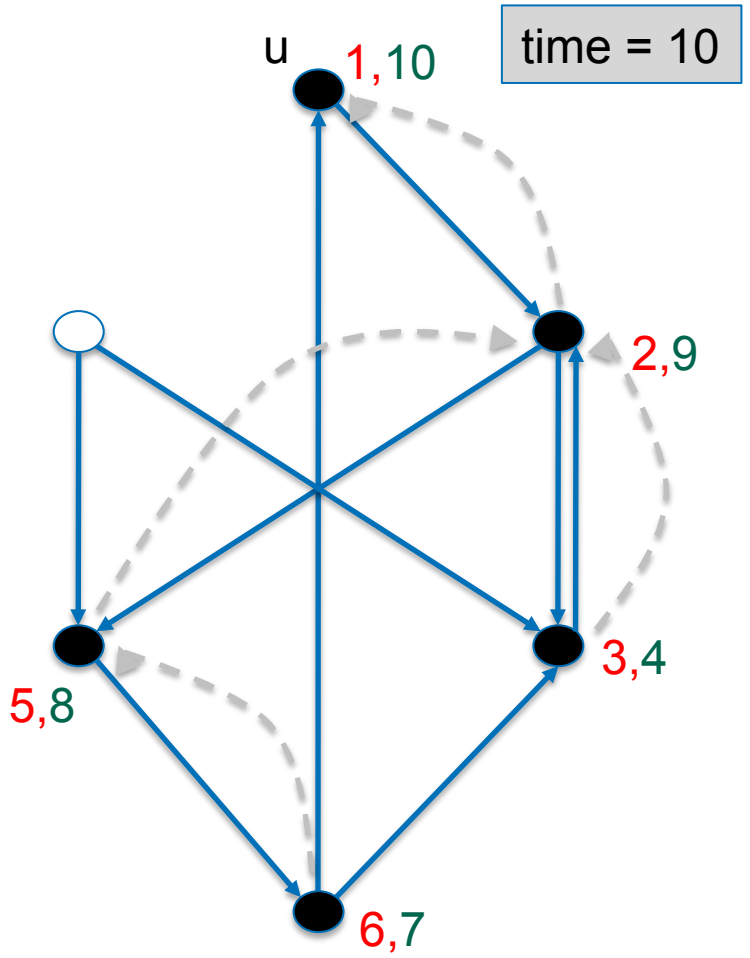


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

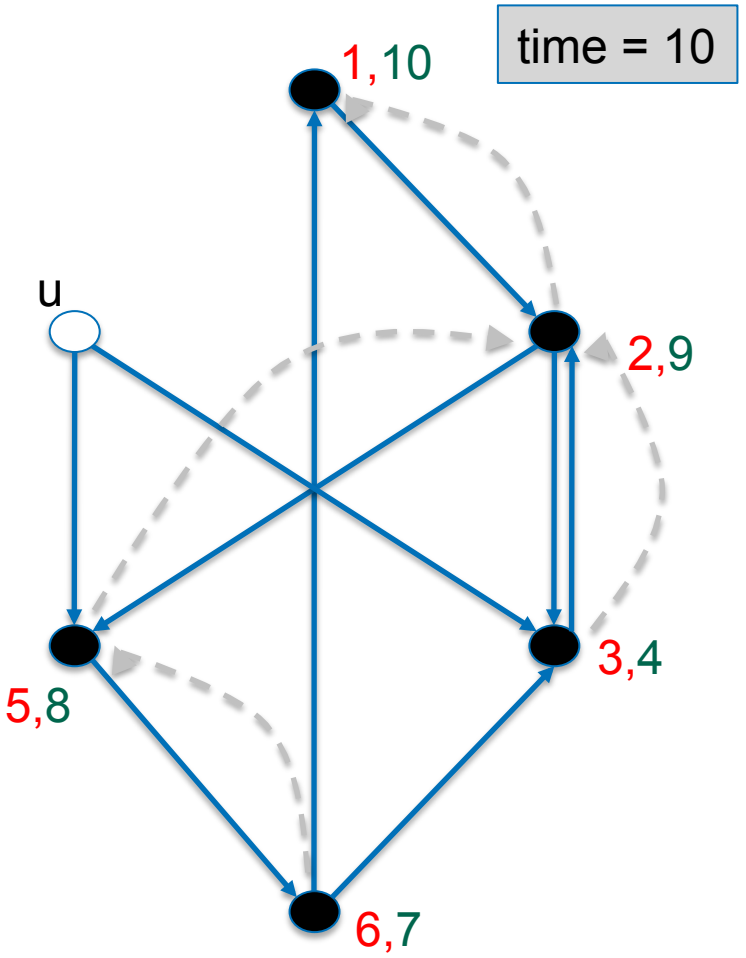


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```

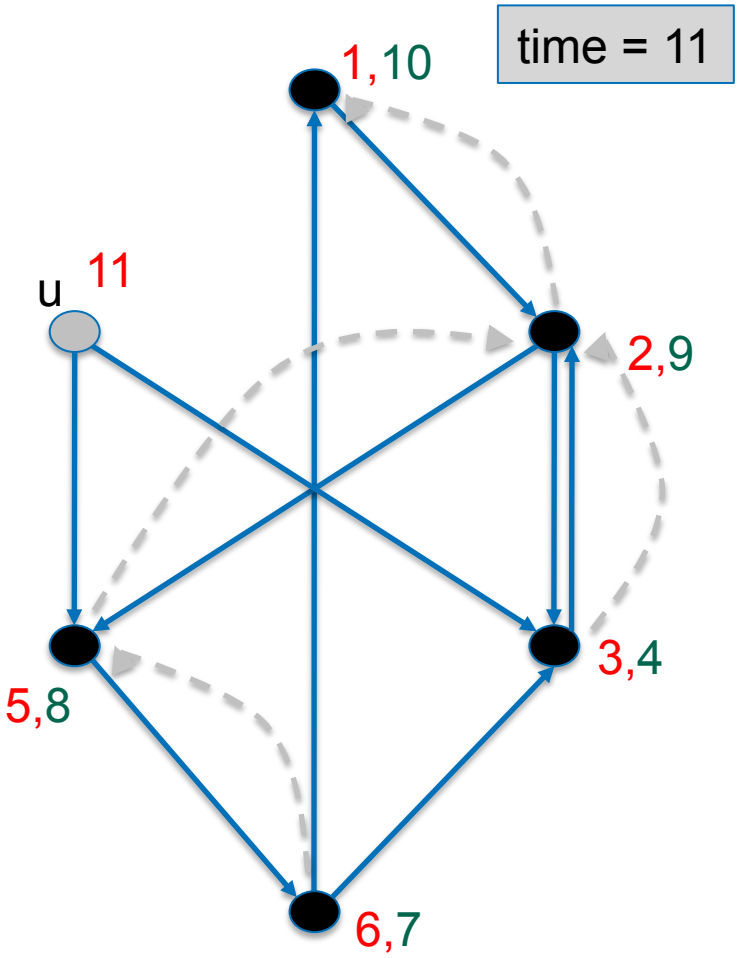


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.  color[u] := gray;
2.  time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.  color[u] := black
8.  time:=time+1;f[u]:=time
```

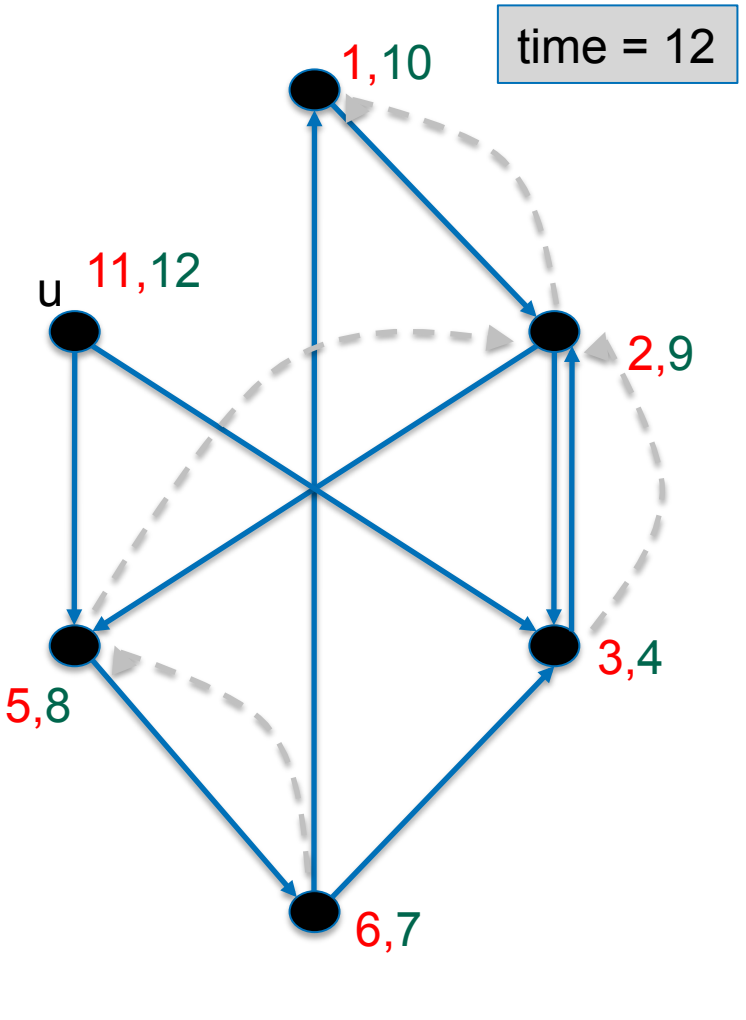


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.          π[u] := nil;
4.   time := 0;
5.  for each node u∈V do
6.          if color[u]==white then DFS-Visit(u);
```

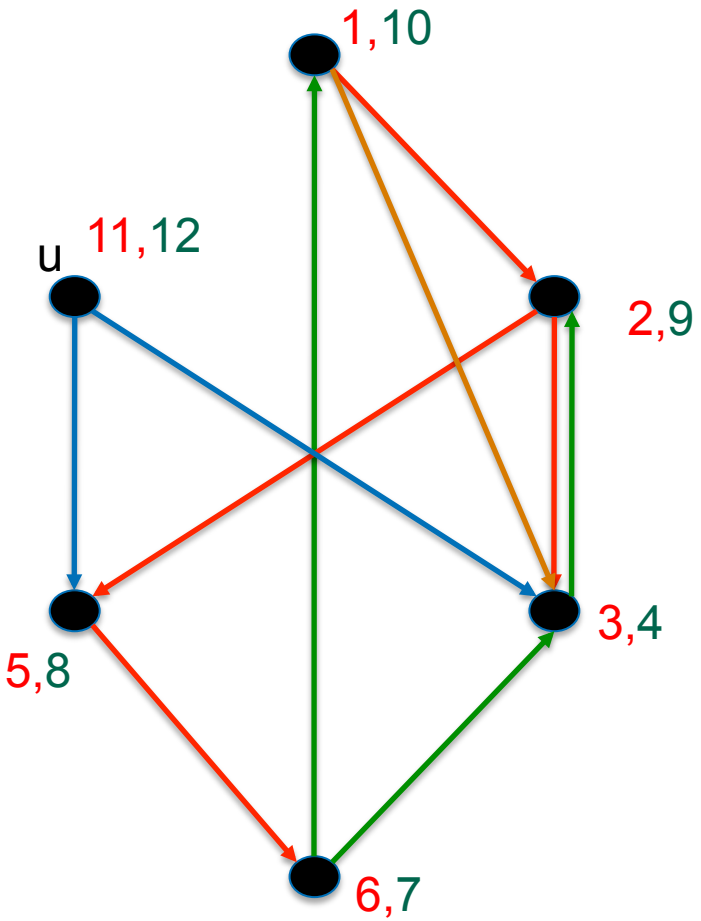
```
DFS-Visit(u)
1.   color[u] := gray;
2.   time := time+1; d[u] := time;
3.  for each node v∈adj[u] do
4.          if color[v]==white then
5.                  π[v]:=u;
6.                  DFS-Visit(v);
7.   color[u] := black
8.   time:=time+1;f[u]:=time
```



Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Tiefensuche (engl. Depth-First-Search)
  - DFS Algorithmus teilt die Kanten in vier Teilmengen
    - Baumkanten sind
      → Kanten des aufspannenden Waldes
    - Rückwärtskante sind
      → Kanten (u,v), die Knoten u mit Vorfahren von u im DFS-Baum verbinden
    - Vorwärtskanten sind
      → nicht-Baumkanten (u,v), die u mit einem Nachfolger v von u verbinden.
    - Kreuzungskanten sind alle anderen Kanten
  - Die Laufzeit der DFS ist O(|V|+|E|)

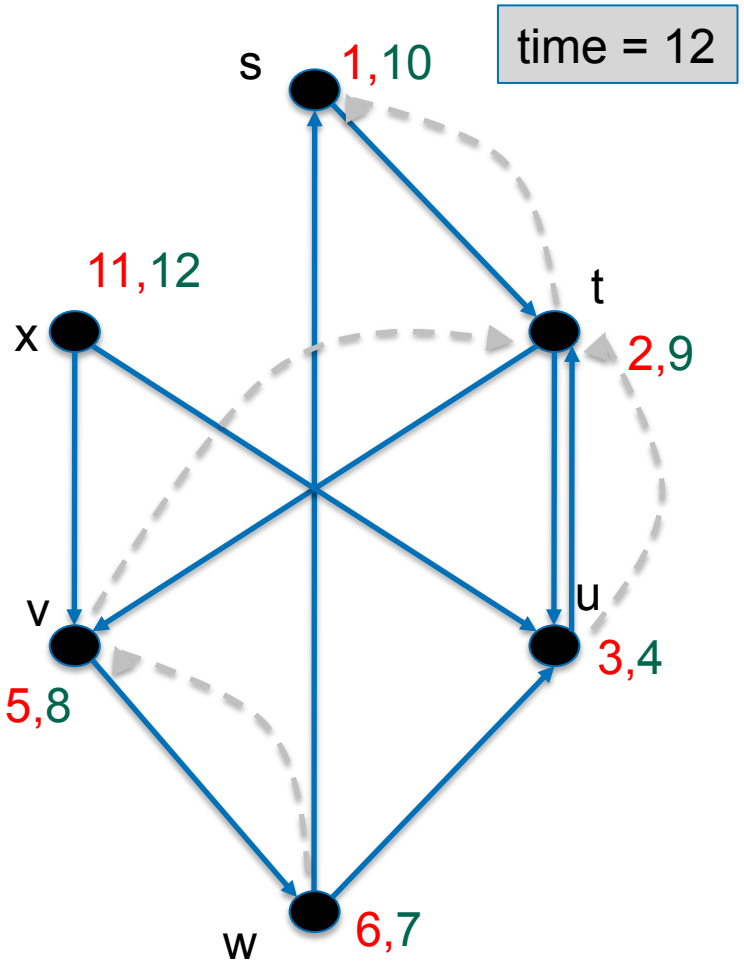


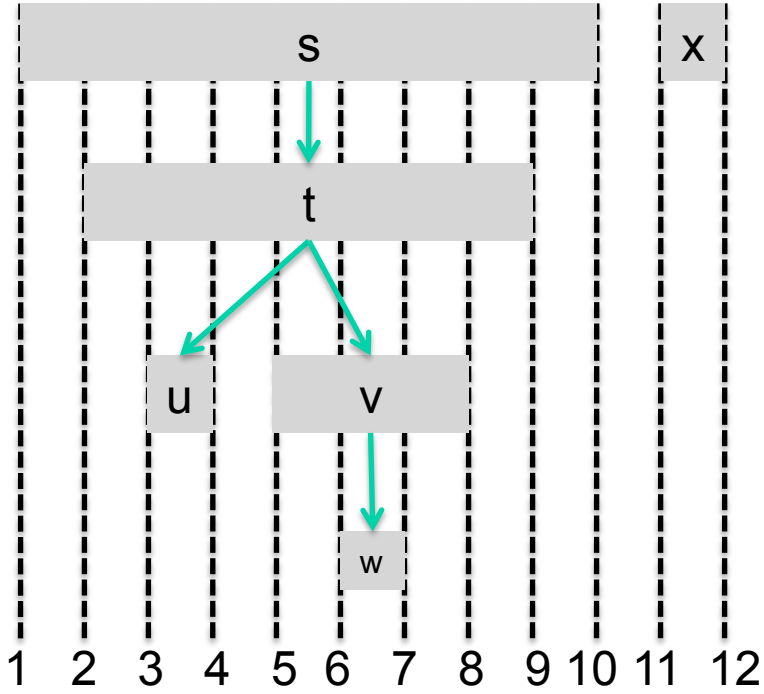
Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Klammerungstheorem
  - Seien u und v Knoten eines gerichteten oder ungerichteten Graphen G. Nach einer DFS-Suche auf G gilt eine der drei folgenden Aussagen
    - Die Intervalle [d[u],f[u]] und [d[v],f[v]] sind disjunkt
    - Das Intervall [d[u],f[u]] ist vollständig im Intervall [d[v],f[v]] enthalten, und u ist ein Nachfolger von v im DFS-Baum
    - Das Intervall [d[v],f[v]] ist vollständig im Intervall [d[u],f[u]] enthalten, und v ist ein Nachfolger von u im DFS-Baum

Elementare Graphalgorithmen





Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Klammerungstheorem, Beweis

  – Sei zunächst d[u]<d[v]. Dann gibt es zwei Unterfälle:

    1. d[v]<f[u]: v wurde entdeckt, als u noch grau war. v ist damit Nachfolger von u. Da v später als u entdeckt wurde, wurden alle ausgehende Kanten von v untersucht, bevor die Suche zu u zurückkehrte. Es wurde somit erst v, dann u beendet. Also f[v]<f[u]. Insgesamt: d[u]<d[v]<f[v]<f[u].
    2. f[u]<d[v]: v wurde erst entdeckt, als u schon beendet war. Da natürlich d[u]<f[u] und d[v]<f[v] gilt, sind die Intervalle [d[u],f[u]] und [d[v],f[v]] disjunkt.

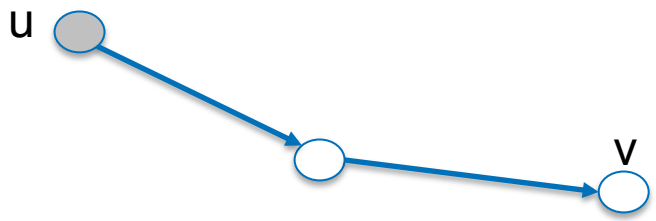
  – Der Fall d[v]<d[u] ist analog.

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der Satz vom weißen Weg

  – In einem DFS-Wald eines gerichteten oder ungerichteten Graphen G=(V,E) ist ein Knoten v genau dann ein Nachkomme eines Knotens u, wenn zur Zeit d[u] der Knoten v von u aus allein über weiße Knoten erreichbar ist.

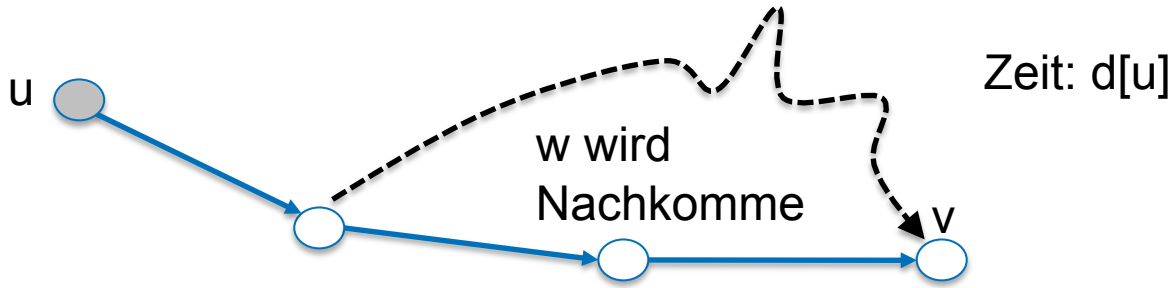


⇒: Sei v ein Nachkomme v von u. Sei also d[v]>d[u] und sei u noch nicht beendet. Dann war der Weg zwischen u und v zur Zeit d[u] offenbar weiß, denn DFS hat sich von u zu v mit Hilfe von Zeile 4 von DFS-Visit bewegt.

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der Satz vom weißen Weg



⇐: Es gebe nun einen weißen Weg von u nach v zur Zeit d[u]. O.b.d.A. werde jeder andere Knoten auf dem Weg ein Nachkomme von u. (falls das nicht so ist, machen wir folgende Überlegung für den ersten Knoten auf dem Weg, der nicht mehr Nachkomme von u wird.) Sei w der letzte Knoten auf dem weißen Weg, der noch Nachkomme von u wird. Offenbar gilt dann: f[w]<f[u].
Da d[v]>d[u] und d[v]<f[w] (v ist direkter Nachfolger von w) gilt: d[u]<(d[w]<)d[v]<f[w]<f[u]. Wegen des Klammerungstheorems muss f[v] < f[w] sein. Also ist v doch ein Nachkomme von u.

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Sei G=(V,E) ein **ungerichteter** Graph. Dann gilt:
  - Im DFS-Wald gibt es nur Baum- und Rückwärtskanten.
  - Die Menge aller Baumkanten bildet einen Wald, in dem jede Zusammenhangskomponente von G einen aufspannende Baum erzeugt.
  - G ist genau dann Kreisfrei, wenn es keine Rückwärtskanten gibt.

  Beweis: Übung

Elementare Graphalgorithmen

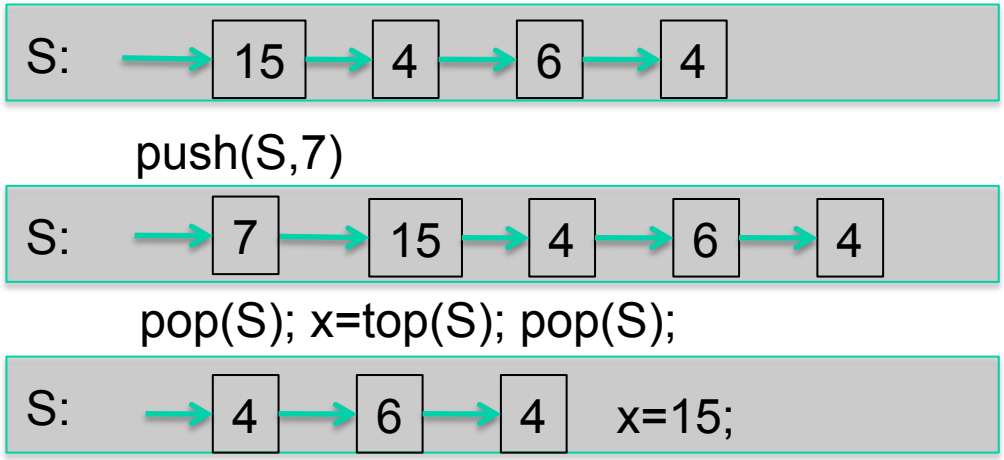
TECHNISCHE UNIVERSITÄT DARMSTADT

- **DFS nochmal anders**
  - Abstrakte Datentypen: Stack und Queue
    - Dynamische Datencontainer
    - Elemente können hinzugefügt oder weggenommen werden
    - Stack implementiert eine last-in-first-out (LIFO) Strategie
    - Queue implementiert eine first-in-first-out (FIFO) Strategie
    - Nur mittels vordefinierter Funktionen lassen sich Stack oder Queue manipulieren

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- DFS nochmal anders
  - Sei S ein Stack. Dann gibt es 4 Operationen mit Laufzeit O(1):
    - empty() Ist der Stack leer?
    - push(S,x) Füge das Element x zum Stack S hinzu
    - pop(S) Entferne das zuletzt zu S hinzugefügte Element wieder.
    - x=top(S) lese den Inhalt des zuletzt zu S hinzugefügten Elements

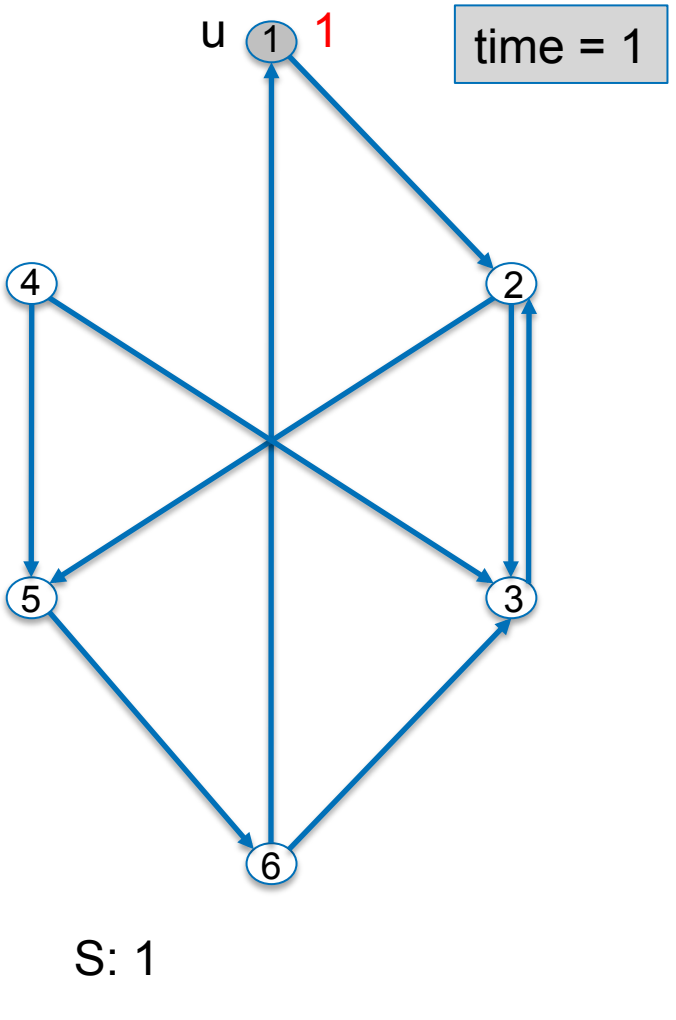


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; S:=∅;
4.  for each node u∈V do
5.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   push(S,u);
2.   while not empty(S) do
3.       u:=top(S);
4.       if color[u]==white then
5.           color[u] := gray; time:=time+1; d[u]:=time;
6.           for each node v∈adj[u] do
7.               if color[v]==white push(S,v);
8.       else if color[u]==gray then
9.           color[u] := black; time:=time+1; f[u]:=time;
10.          pop(S);
11.      else if color[v]==black pop(S);
```

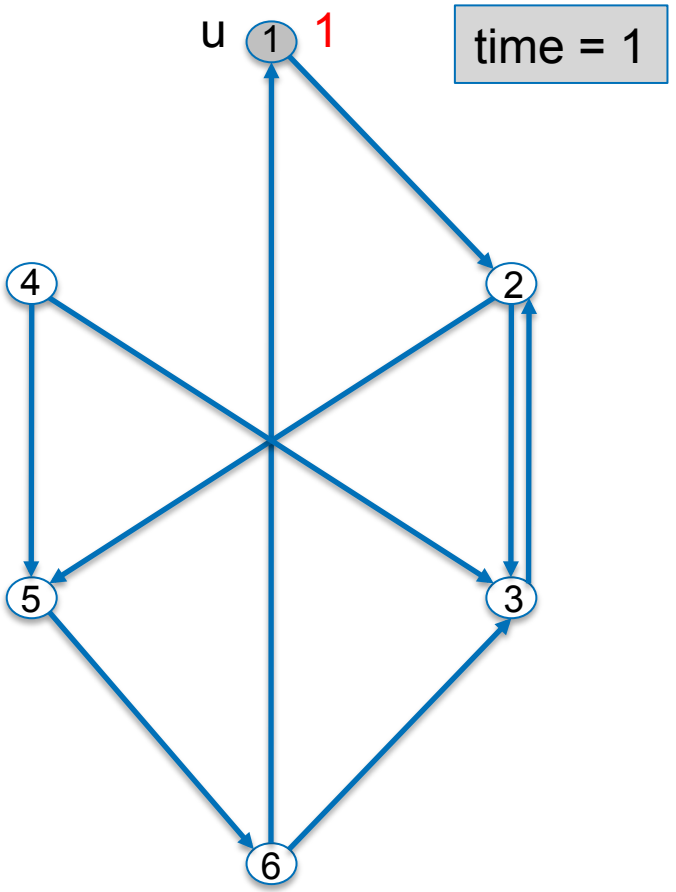


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; S:=∅;
4.  for each node u∈V do
5.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   push(S,u);
2.   while not empty(S) do
3.       u:=top(S);
4.       if color[u]==white then
5.           color[u] := gray; time:=time+1; d[u]:=time;
6.           for each node v∈adj[u] do
7.               if color[v]==white push(S,v);
8.       else if color[u]==gray then
9.           color[u] := black; time:=time+1; f[u]:=time;
10.          pop(S);
11.      else if color[v]==black pop(S);
```



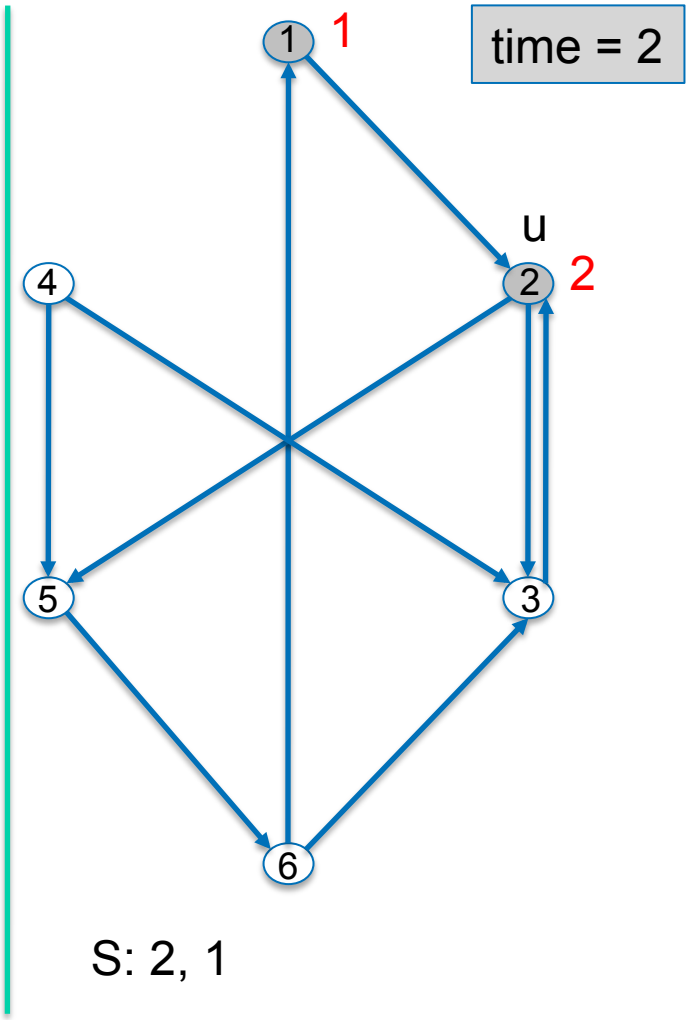
S: 2, 1

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; S:=∅;
4.  for each node u∈V do
5.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   push(S,u);
2.   while not empty(S) do
3.       u:=top(S);
4.       if color[u]==white then
5.           color[u] := gray; time:=time+1; d[u]:=time;
6.           for each node v∈adj[u] do
7.               if color[v]==white push(S,v);
8.       else if color[u]==gray then
9.           color[u] := black; time:=time+1; f[u]:=time;
10.          pop(S);
11.      else if color[v]==black pop(S);
```

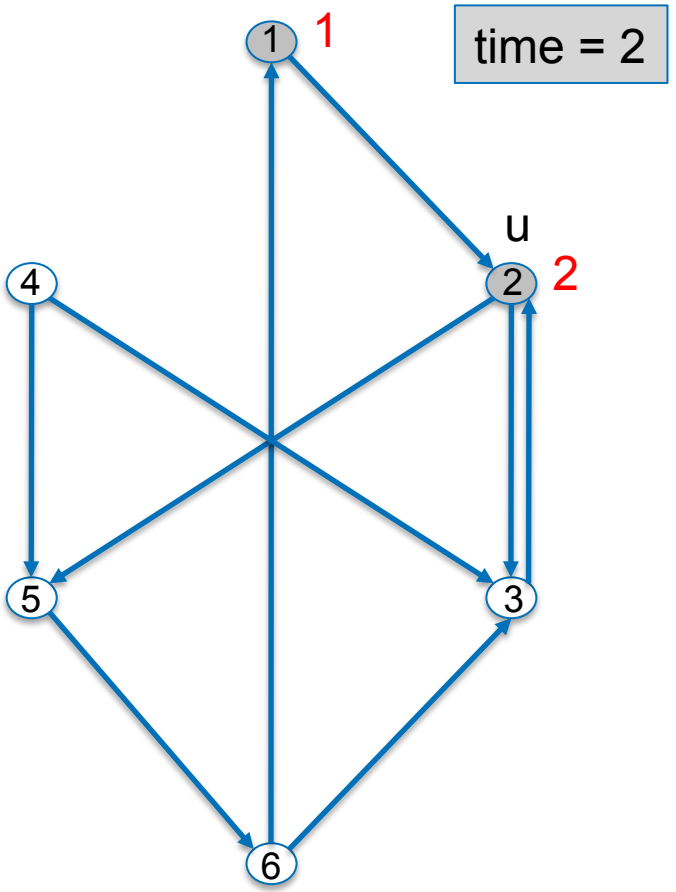


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; S:=∅;
4.  for each node u∈V do
5.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   push(S,u);
2.   while not empty(S) do
3.       u:=top(S);
4.       if color[u]==white then
5.           color[u] := gray; time:=time+1; d[u]:=time;
6.           for each node v∈adj[u] do
7.               if color[v]==white push(S,v);
8.       else if color[u]==gray then
9.           color[u] := black; time:=time+1; f[u]:=time;
10.          pop(S);
11.      else if color[v]==black pop(S);
```



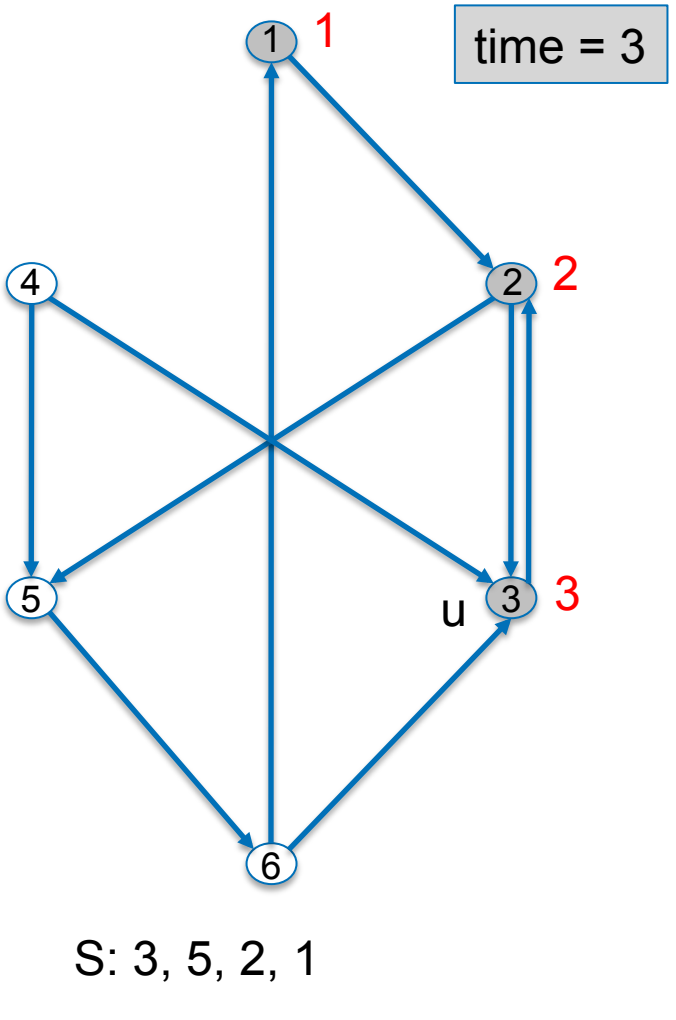
S: 3, 5, 2, 1

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; S:=∅;
4.  for each node u∈V do
5.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   push(S,u);
2.   while not empty(S) do
3.       u:=top(S);
4.       if color[u]==white then
5.           color[u] := gray; time:=time+1; d[u]:=time;
6.           for each node v∈adj[u] do
7.               if color[v]==white push(S,v);
8.       else if color[u]==gray then
9.           color[u] := black; time:=time+1; f[u]:=time;
10.          pop(S);
11.      else if color[v]==black pop(S);
```



S: 3, 5, 2, 1

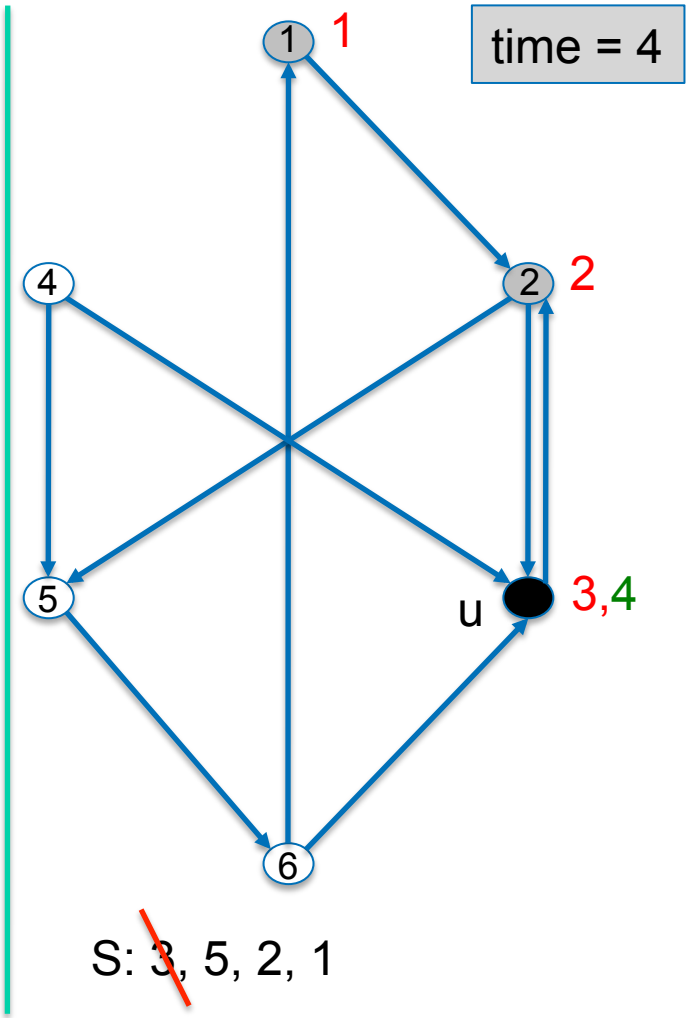
Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
DFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; S:=∅;
4.  for each node u∈V do
5.          if color[u]==white then DFS-Visit(u);
```

```
DFS-Visit(u)
1.   push(S,u);
2.   while not empty(S) do
3.       u:=top(S);
4.       if color[u]==white then
5.           color[u] := gray; time:=time+1; d[u]:=time;
6.           for each node v∈adj[u] do
7.               if color[v]==white push(S,v);
8.       else if color[u]==gray then
9.           color[u] := black; time:=time+1; f[u]:=time;
10.          pop(S);
11.      else if color[v]==black pop(S);
```

u.s.w.

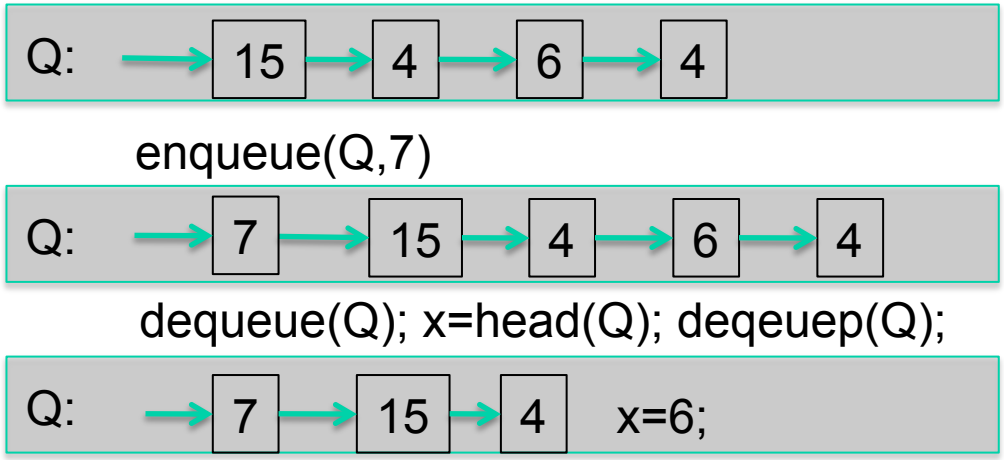


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- BFS (Breadth-First-Search, Breitensuche)

  – Sei Q ein Queue. Dann gibt es 4 Operationen mit Laufzeit O(1):
  - empty() Ist die Queue leer?
  - enqueue(S,x) Füge das Element x zur Queue Q hinzu
  - dequeue(S) Entferne das zuerst in Q eingefügte Element wieder.
  - x=head(S) lese den Inhalt des zuerst in Q eingefügten Elements

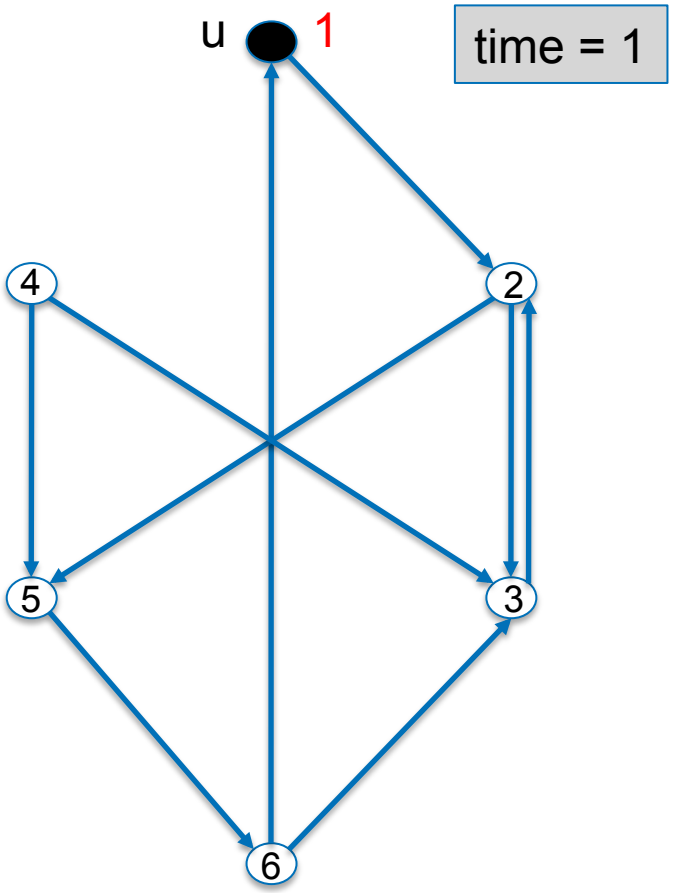


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.     u:=head(Q);
4.     if color[u] ≠ black then
5.       color[u] := black; time:=time+1; d[u]:=time;
6.       for each node v∈adj[u] do
7.         if color[v]==white enqueue(Q,v),color[v]:=gray
8.     else if color[u]==black then
9.        time:=time+1; f[u]:=time;
10.       dequeue(Q);
```



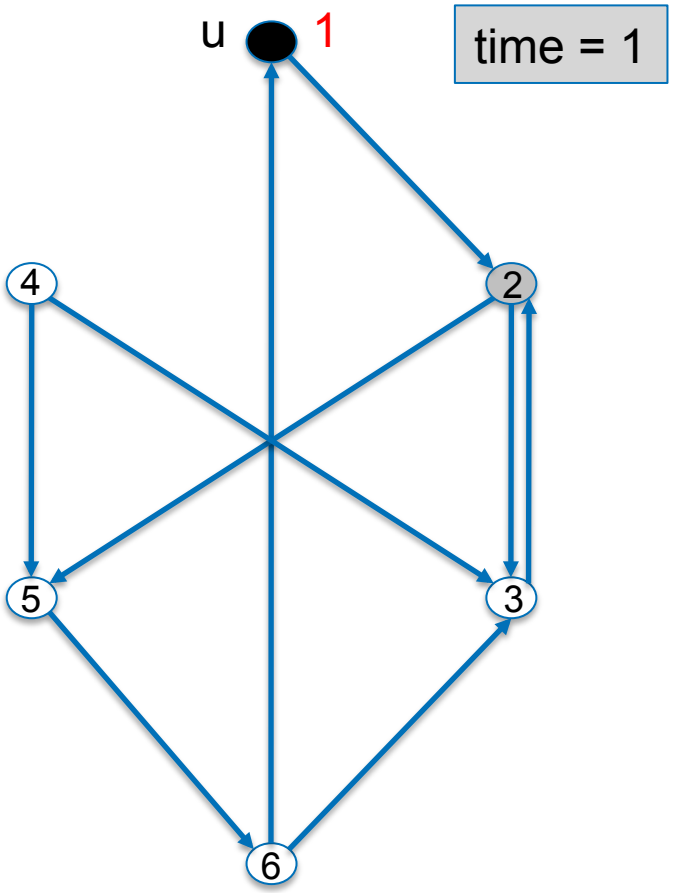
Q: 1

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.     u:=head(Q);
4.     if color[u] ≠ black then
5.       color[u] := black; time:=time+1; d[u]:=time;
6.       for each node v∈adj[u] do
7.         if color[v]==white enqueue(Q,v),color[v]:=gray
8.     else if color[u]==black then
9.         time:=time+1; f[u]:=time;
10.       dequeue(Q);
```



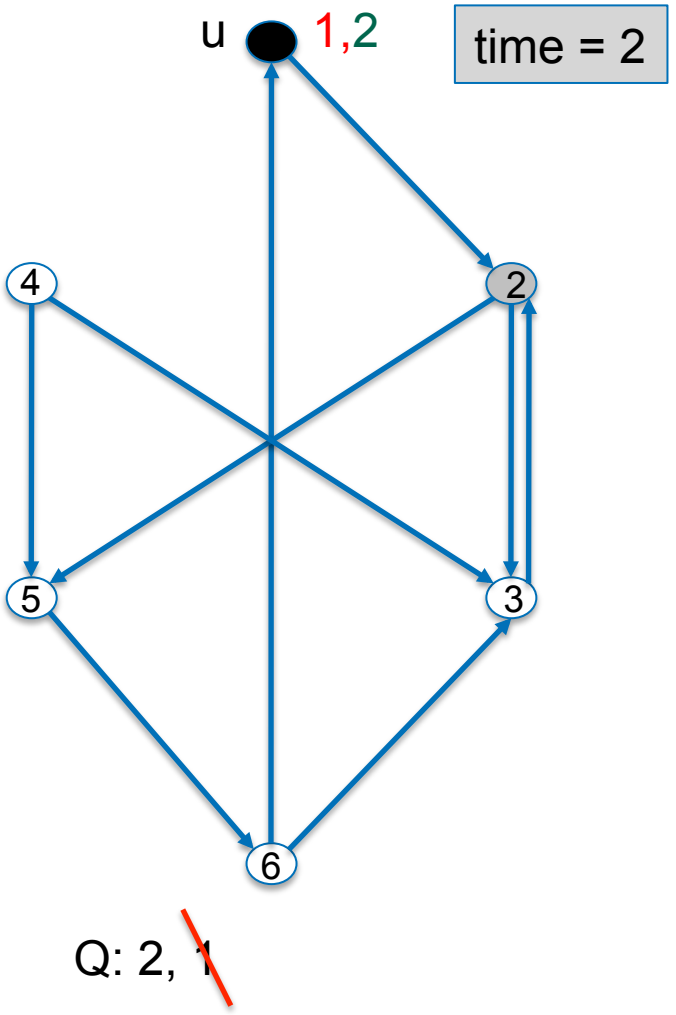
Q: 2, 1

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.    u:=head(Q);
4.    if color[u] ≠ black then
5.      color[u] := black; time:=time+1; d[u]:=time;
6.      for each node v∈adj[u] do
7.        if color[v]==white enqueue(Q,v),color[v]:=gray
8.    else if color[u]==black then
9.      time:=time+1; f[u]:=time;
10.     dequeue(Q);
```

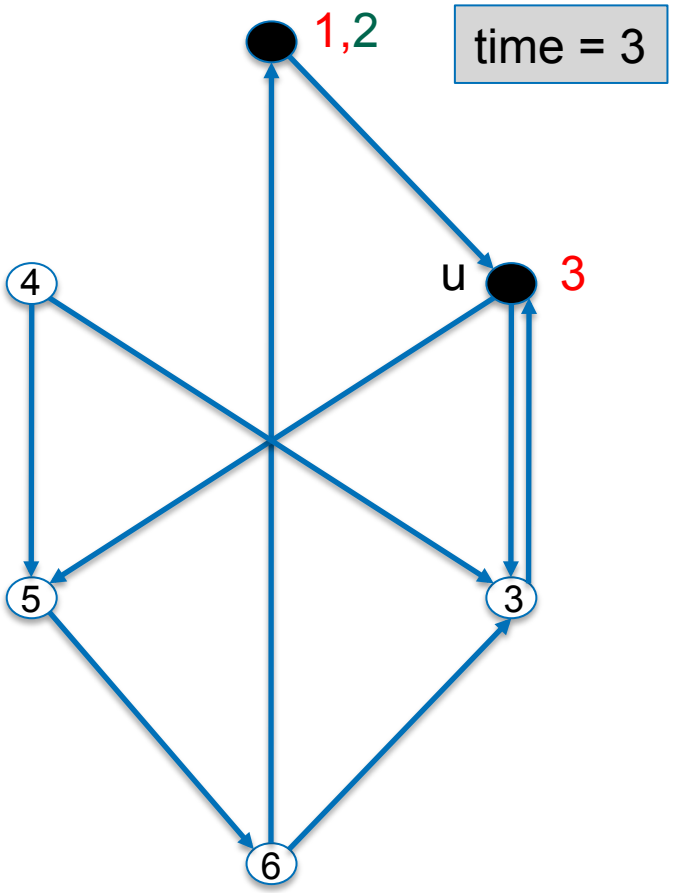


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.     u:=head(Q);
4.     if color[u] ≠ black then
5.       color[u] := black; time:=time+1; d[u]:=time;
6.       for each node v∈adj[u] do
7.          if color[v]==white enqueue(Q,v),color[v]:=gray
8.      else if color[u]==black then
9.         time:=time+1; f[u]:=time;
10.        dequeue(Q);
```



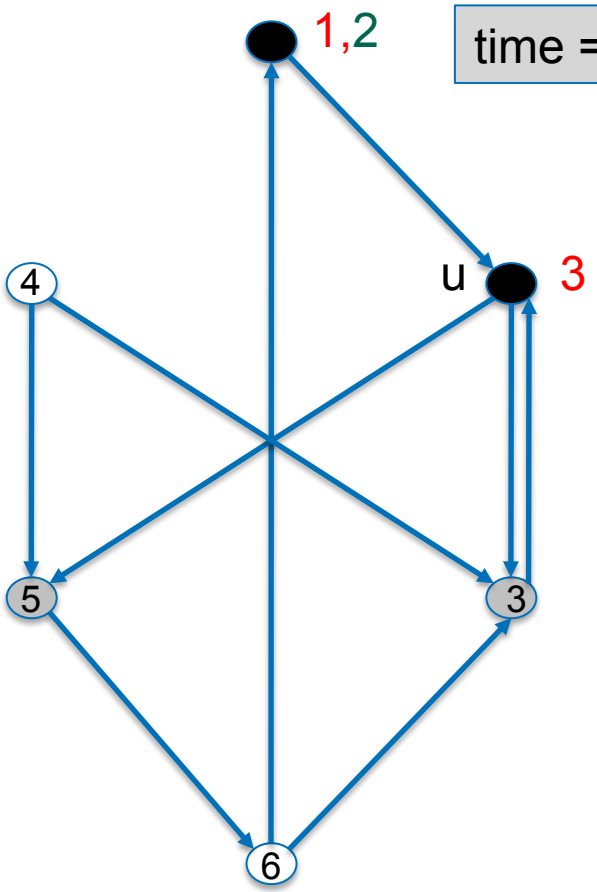
Q: 2

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.     u:=head(Q);
4.     if color[u] ≠ black then
5.       color[u] := black; time:=time+1; d[u]:=time;
6.       for each node v∈adj[u] do
7.         if color[v]==white enqueue(Q,v),color[v]:=gray
8.     else if color[u]==black then
9.         time:=time+1; f[u]:=time;
10.       dequeue(Q);
```



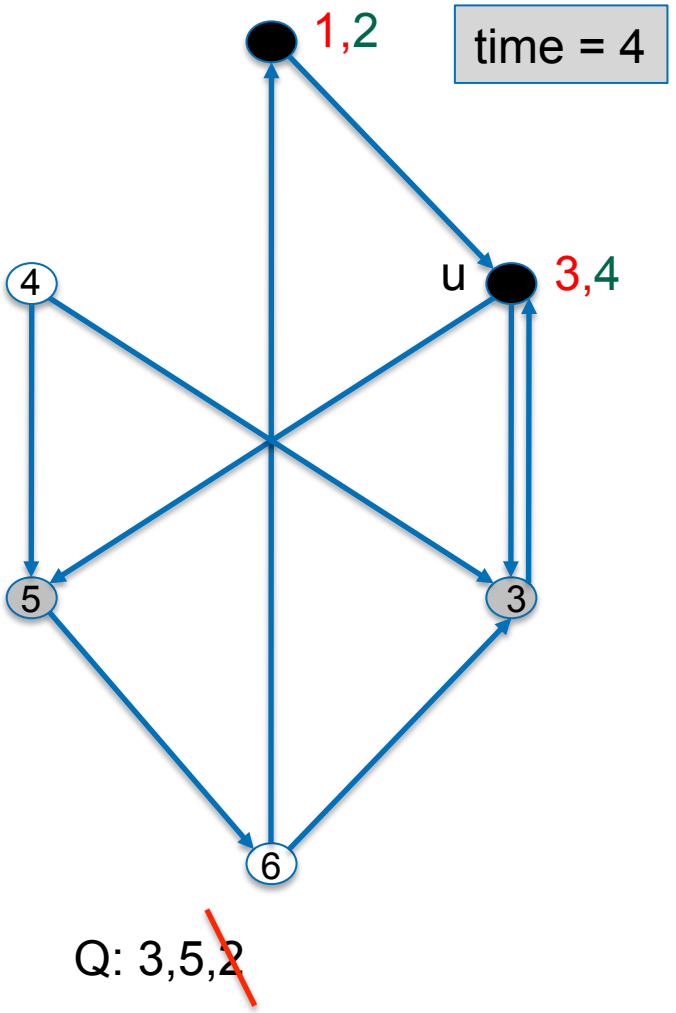
Q: 3,5,2

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.    u:=head(Q);
4.    if color[u] ≠ black then
5.      color[u] := black; time:=time+1; d[u]:=time;
6.      for each node v∈adj[u] do
7.        if color[v]==white enqueue(Q,v),color[v]:=gray
8.    else if color[u]==black then
9.      time:=time+1; f[u]:=time;
10.     dequeue(Q);
```

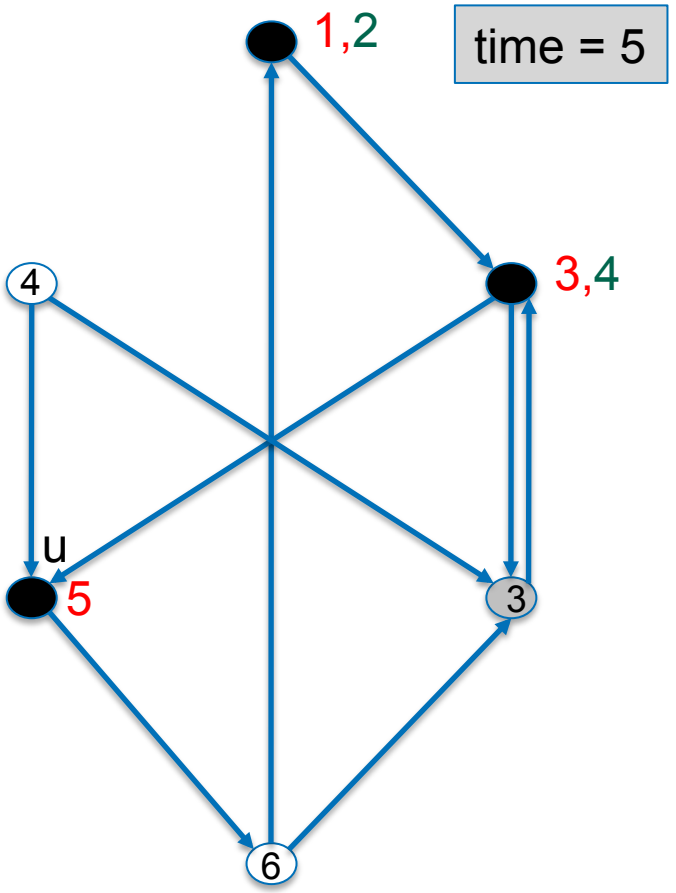


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.     u:=head(Q);
4.     if color[u] ≠ black then
5.       color[u] := black; time:=time+1; d[u]:=time;
6.       for each node v∈adj[u] do
7.          if color[v]==white enqueue(Q,v),color[v]:=gray
8.      else if color[u]==black then
9.         time:=time+1; f[u]:=time;
10.        dequeue(Q);
```



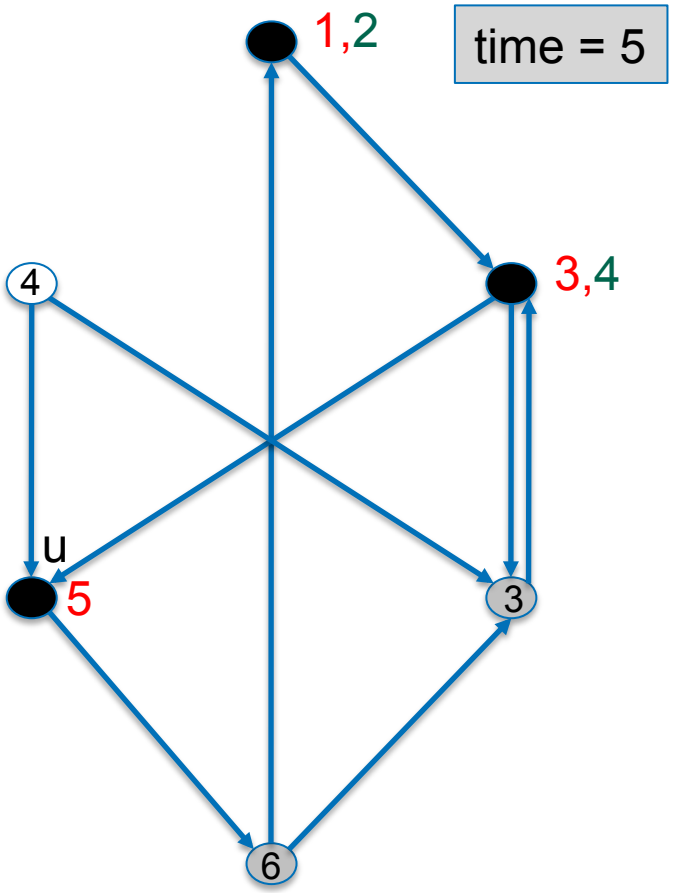
Q: 3,5

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.     u:=head(Q);
4.     if color[u] ≠ black then
5.       color[u] := black; time:=time+1; d[u]:=time;
6.       for each node v∈adj[u] do
7.         if color[v]==white enqueue(Q,v),color[v]:=gray
8.     else if color[u]==black then
9.        time:=time+1; f[u]:=time;
10.       dequeue(Q);
```



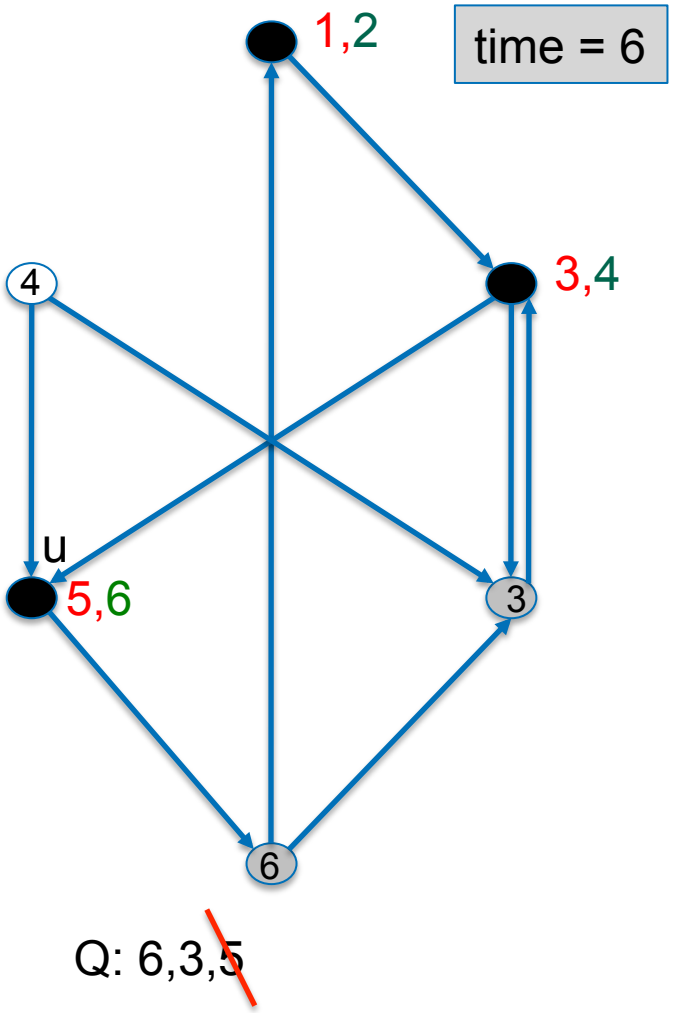
Q: 6,3,5

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.     u:=head(Q);
4.     if color[u] ≠ black then
5.       color[u] := black; time:=time+1; d[u]:=time;
6.       for each node v∈adj[u] do
7.         if color[v]==white enqueue(Q,v),color[v]:=gray
8.     else if color[u]==black then
9.       time:=time+1; f[u]:=time;
10.      dequeue(Q);
```

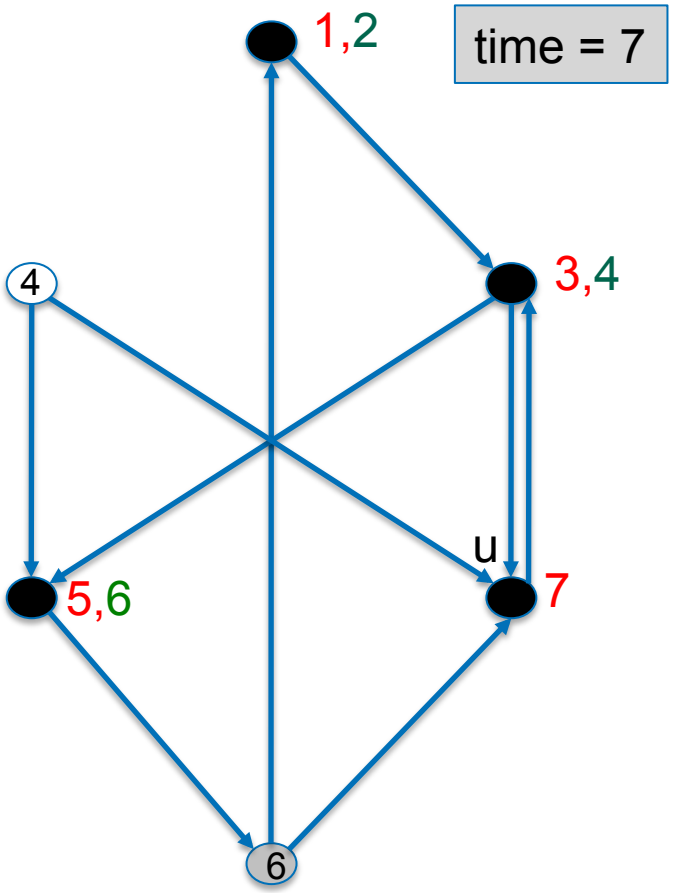


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.     u:=head(Q);
4.     if color[u] ≠ black then
5.       color[u] := black; time:=time+1; d[u]:=time;
6.       for each node v∈adj[u] do
7.          if color[v]==white enqueue(Q,v),color[v]:=gray
8.     else if color[u]==black then
9.        time:=time+1; f[u]:=time;
10.       dequeue(Q);
```



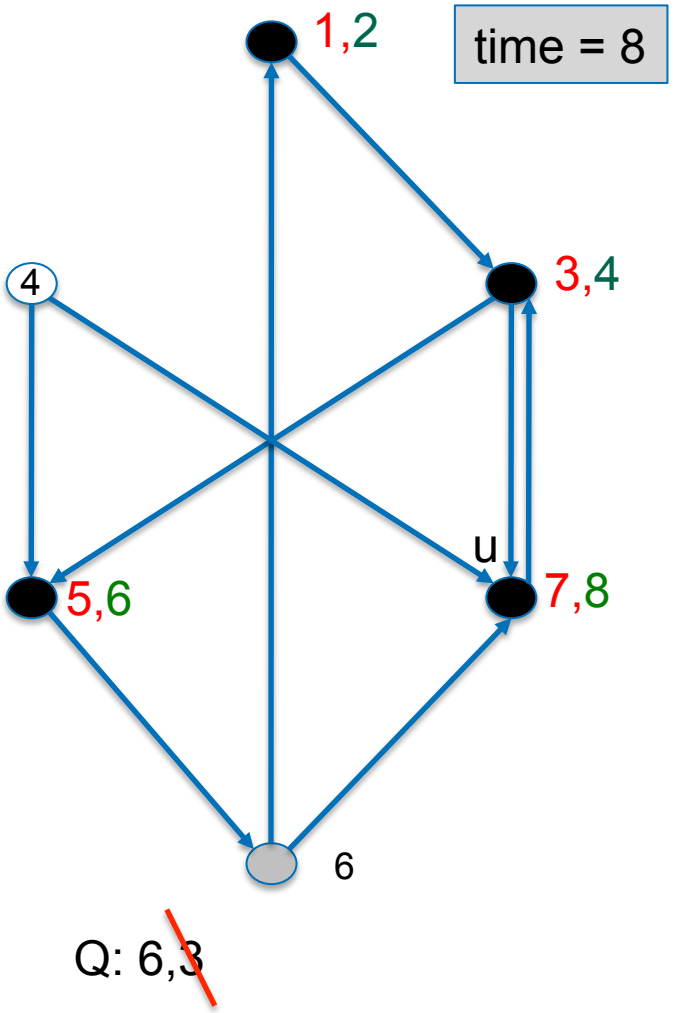
Q: 6,3

Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.     u:=head(Q);
4.     if color[u] ≠ black then
5.       color[u] := black; time:=time+1; d[u]:=time;
6.       for each node v∈adj[u] do
7.         if color[v]==white enqueue(Q,v),color[v]:=gray
8.     else if color[u]==black then
9.       time:=time+1; f[u]:=time;
10.      dequeue(Q);
```

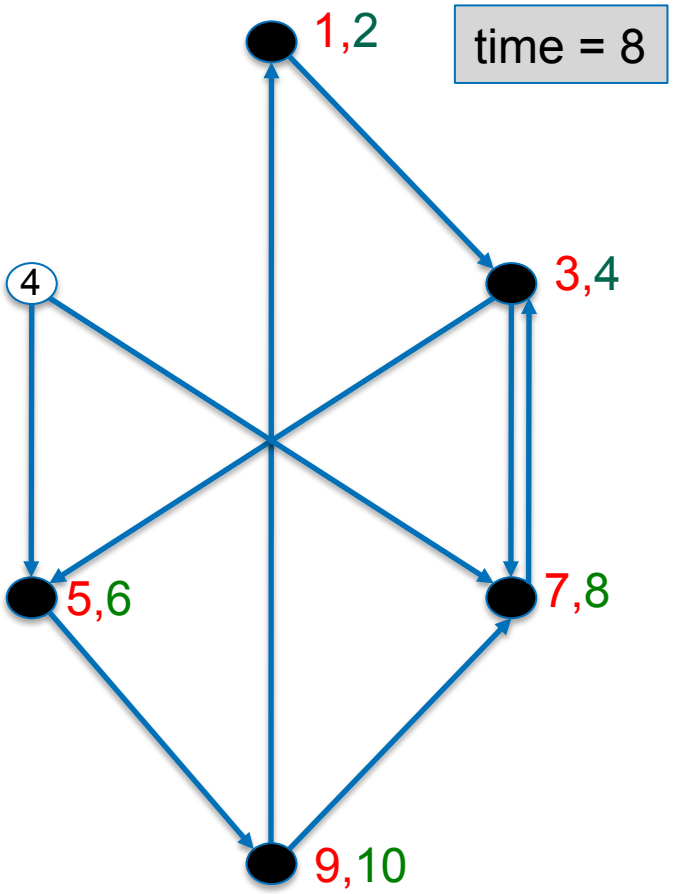


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white;
3.   time := 0; Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u)
1.   enqueue(Q,u);
2.  while not empty(Q) do
3.    u:=head(Q);
4.    if color[u] ≠ black then
5.      color[u] := black; time:=time+1; d[u]:=time;
6.      for each node v∈adj[u] do
7.        if color[v]==white enqueue(Q,v),color[v]:=gray
8.    else if color[u]==black then
9.      time:=time+1; f[u]:=time;
10.     dequeue(Q);
```

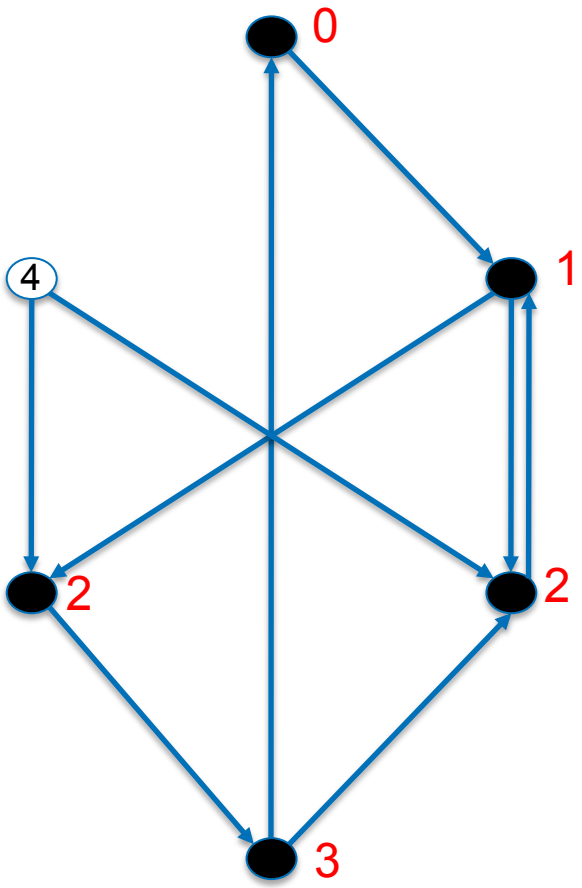


Elementare Graphalgorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
BFS(G)
1.  for each node u∈V do
2.          color[u] := white; dist[u] := ∞;
3.  Q:=∅;
4.  choose an arbitrary node u∈V and do
5.          BFS-Visit(u);
```

```
BFS-Visit(u), Variante
1.  enqueue(Q,u); dist(u):=0;
2.  while not empty(Q) do
3.      u:=head(Q);
4.      for each node v∈adj[u] do
5.          if color[v]==white
6.              color[v]:=gray;
7.              enqueue(Q,v);
8.              dist(v):=dist(u)+1;
9.              π[v]:=u;
10.     color[u] := black;
11.     dequeue(Q);
```



Q:

Breitensuche, Eigenschaften

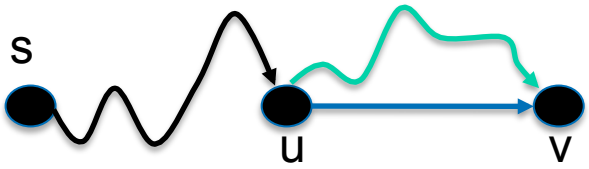
- BFS bildet Grundlage vieler Algorithmen
- Ziel kann es z.B. sein, bei gegebenen Graph G=(V,E) von einer Quelle $s \in V$ alle Knoten $v \in V$ zu finden, die von s aus erreichbar sind. Dabei ist ein Knoten v von s aus erreichbar, wenn es einen Weg von s nach v gibt.
- Die BFS errechnet für alle Knoten v den Abstand $\delta(s,v)$ von s zu v. Dabei ist der Abstand die minimale Anzahl von Kanten auf einem Weg von s zu v.
- Die BFS bestimmt alle Knoten mit Abstand <k vor den Kanten mit Abstand k. Daher der Name Breitensuche.
- Laufzeit: $O(|V| + |E|)$

Breitensuche, Eigenschaften

• Sei δ(s,v) die kürzeste-Wege Entfernung von s nach v, also die minimale Anzahl von Kanten, über die man laufen muss, um von s nach v zu kommen. Wenn es keinen Weg von s nach v gibt, sei δ(s,v) = ∞. BFS berechnet für alle Knoten v∈V die kürzeste Wegeentfernung von s nach v.

• Beweis:
  • Lemma bfs1: Sei G=(V,E) ein gerichteter oder ungerichteter Graph und sei s∈V sei ein beliebiger Knoten. Dann gilt für jede Kante (u,v)∈E:

$$\delta(s,v) \leq \delta(s,u)+1$$



Beweis: wenn u nicht von s erreichbar, ist δ(s,u)=∞. Damit klar.
Wenn u erreichbar ist, ist es auch v. Der Weg von s nach v kann nicht länger sein, als der Weg von s nach u plus die Kante (u,v). ✔

Elementare Graphalgorithmen

```
BFS-Visit(u), Variante
1.  enqueue(Q,u); dist(u):=0;
2.  while not empty(Q) do
3.      u:=head(Q);
4.      for each node v∈adj[u] do
5.          if color[v]==white
6.              color[v]:=gray;
7.              enqueue(Q,v);
8.              dist(v):=dist(u)+1;
9.              π[v]:=u;
10.     color[u] := black;
11.     dequeue(Q);
```

- Beweis (Forts.):
  - Lemma bfs2: Sei G=(V,E) ein Graph. Sei BFS auf G mit Startknoten s gelaufen. Dann gilt für jeden Knoten v:

$$dist[v| \geq \delta(s,v)$$

Beweis: Induktion über Anzahl enqueue-Aufrufe in BFS-Visit

Induktionsanfang: Sei s soeben in Q platziert worden. Dann wird dist[s]=0 gesetzt und auch nicht wieder verändert. Alle anderen Werte sind auf ∞. Auch gilt δ(s,s)=0. ✔

Induktionsvoraussetzung: Für alle v∈V, die bereits entdeckt wurden, gilt: dist[v| ≥ δ(s,v)

Induktionsschluss: Sei v von u aus entdeckt worden, Nach IV gilt: dist[u] ≥ δ(s,u). Wegen Zeile 8 (BFS-Visit), gilt dist[v] = dist[u]+1 ≥ δ(s,u) + 1 ≥ δ(s,v), wegen Lemma bfs1
Danach wird dist[v] nicht mehr verändert. ✔

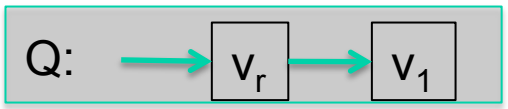
Elementare Graphalgorithmen

```
BFS-Visit(u), Variante
1.  enqueue(Q,u); dist(u):=0;
2.  while not empty(Q) do
3.      u:=head(Q);
4.      for each node v∈adj[u] do
5.          if color[v]==white
6.              color[v]:=gray;
7.              enqueue(Q,v);
8.              dist(v):=dist(u)+1;
9.              π[v]:=u;
10.     color[u] := black;
11.     dequeue(Q);
```

- Beweis (Forts.):
  - Lemma bfs3: Sei G=(V,E) ein Graph.
    Sei während BFS arbeitet: Q=<v_1,v_2,...,v_r>
    Sei v_1 head(Q). Dann gilt für i=1,2...,r-1:

    $dist[v_r] \leq dist[v_1]+1$ und $dist[v_i] \leq dist[v_{i+1}]$

    Beweis: Induktion über Anzahl Durchläufe der for-Schleife (insgesamt)
    Induktionsanfang: Nach dem ersten Schleifendurchlauf ist
    $v_1 = u = s$ und $v_r=v$. Damit $dist[v_1]=0$ und $dist[v_r]=1$. ✔



$dist[v_r] = dist[v_1]+1$

Elementare Graphalgorithmen

```
BFS-Visit(u), Variante
1.  enqueue(Q,u); dist(u):=0;
2.  while not empty(Q) do
3.      u:=head(Q);
4.      for each node v∈adj[u] do
5.          if color[v]==white
6.              color[v]:=gray;
7.              enqueue(Q,v);
8.              dist(v):=dist(u)+1;
9.              π[v]:=u;
10.     color[u] := black;
11.     dequeue(Q);
```

- Beweis (Forts.):
  - Lemma bfs3: Sei G=(V,E) ein Graph.
    Sei während BFS arbeitet: Q=<v_1,v_2,...,v_r>
    Sei v_1 head(Q). Dann gilt für i=1,2...,r-1:

    $dist[v_r] \leq dist[v_1]+1$ und $dist[v_i] \leq dist[v_{i+1}]$

    Beweis: ...

    Induktionsvoraussetzung: Für die ersten n Schleifendurchläufe gilt:
    $dist[v_{r(n)}] \leq dist[v_{1(n)}]+1$ und $dist[v_{i(n)}] \leq dist[v_{i(n)+1}]$
    nach dem jeweiligen Schleifendurchlauf.

Elementare Graphalgorithmen

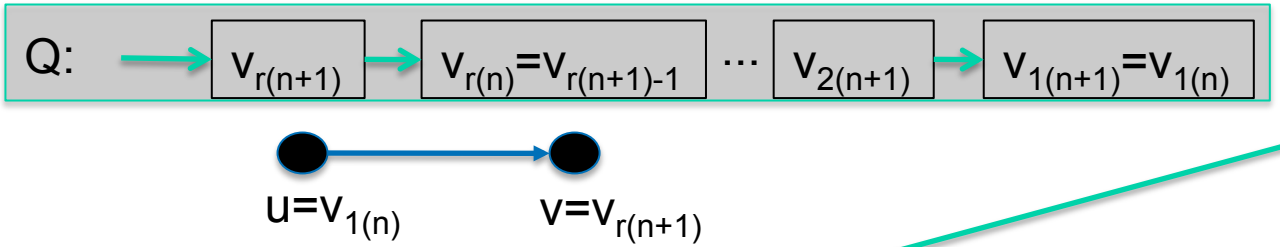
```
BFS-Visit(u), Variante
1.  enqueue(Q,u); dist(u):=0;
2.  while not empty(Q) do
3.      u:=head(Q);
4.      for each node v∈adj[u] do
5.          if color[v]==white
6.              color[v]:=gray;
7.              enqueue(Q,v);
8.              dist(v):=dist(u)+1;
9.              π[v]:=u;
10.     color[u] := black;
11.     dequeue(Q);
```

- Beweis (Forts.):
  - Lemma bfs3: Sei G=(V,E) ein Graph. Sei während BFS arbeitet: Q=<$v_1,v_2,\ldots,v_r$> Sei v_1 head(Q). Dann gilt für i=1,2...,r-1:

    $dist[v_r] \leq dist[v_1]+1$ und $dist[v_i] \leq dist[v_{i+1}]$

    Beweis: ...

    Induktionsschluss: Wenn $v_{r(n+1)}$ eingefügt wird, ist $v_{r(n+1)}$ Nachfolger von v_1.



Zwischendurchd dequeue? Nein -> klar. Ja? -> mit IV

Also: $dist[v_{r(n+1)}] \leq dist[v_{1(n+1)}]+1$. Nach IV ist aber auch

$dist[v_{r(n)}] \leq dist[v_{1(n)}]+1 \leq dist[v_{1(n+1)}]+1$ (= dist[u] +1 = dist[v]) = $dist[v_{r(n+1)}]$

IV:

Bei dequeue gilt $dist[v_{r(n+1)}] \leq dist[v_1] + 1 \leq d[v_2] + 1$ ✔

Elementare Graphalgorithmen

```
BFS-Visit(u), Variante
1.  enqueue(Q,u); dist(u):=0;
2.  while not empty(Q) do
3.      u:=head(Q);
4.      for each node v∈adj[u] do
5.          if color[v]==white
6.              color[v]:=gray;
7.              enqueue(Q,v);
8.              dist(v):=dist(u)+1;
9.              π[v]:=u;
10.     color[u] := black;
11.     dequeue(Q);
```

- Beweis (Forts.):
  - Lemma bfs3: Sei G=(V,E) ein Graph.
    Sei während BFS arbeitet: Q=<$v_1,v_2,...,v_r$>
    Sei v_1 head(Q). Dann gilt für i=1,2...,r-1:

    $dist[v_r| \leq dist[v_1]+1$ und $dist[v_i] \leq dist[v_{i+1}]$

    Beweis (Zusammenfassung):
    Induktion über Anzahl Durchläufe der for-Schleife (insgesamt)
    Induktionsanfang: Nach dem ersten Schleifendurchlauf ist
    $v_1 = u = s$ und $v_r=v$. Damit $dist[v_1]=0$ und $dist[v_r]=1$. ✔
    Induktionsvoraussetzung: Für die ersten n Schleifendurchläufe gilt:
    $dist[v_{r(n)}] \leq dist[v_{1(n)}]+1$ und $dist[v_{i(n)}] \leq dist[v_{i(n)+1}]$
    nach dem jeweiligen Schleifendurchlauf.
    Induktionsschluss: Wenn $v_{r(n+1)}$ eingefügt wird, ist $v_{r(n+1)}$ Nachfolger von v_1.
    Also: $dist[v_{r(n+1)}] = dist[v_{1(n)}]+1 \leq dist[v_{1(n+1)}]+1$. Nach IV ist aber auch
    $dist[v_{r(n)}] \leq dist[v_{1(n)}]+1 \leq dist[v_{1(n+1)}]+1$ (= dist[u] +1 = dist[v]) = $dist[v_{r(n+1)}]$
    ✔

Breitensuche, Eigenschaften

• Sei δ(s,v) die kürzeste-Wege Entfernung von s nach v, also die minimale Anzahl von Kanten, über die man laufen muss, um von s nach v zu kommen. Wenn es keinen Weg von s nach v gibt, sei δ(s,v) = ∞. BFS berechnet für alle Knoten $v \in V$ die kürzeste Wegeentfernung von s nach v.

• Beweis:
1) falls v nicht erreichbar ist, wird v nicht entdeckt und dist[v]=∞.
2) definiere $V_k := \{v \in V : \delta(s,v)=k\}$. Beweis über Induktion nach k.
   - Induktionsvoraussetzung: für alle Knoten mit $\delta(s,v) < k$ gilt die Behauptung.
   - IA: k=0: $V_0=\{s\}$ und dist[s]=0. ✔
   - IS: k-1→k: Sei $v \in V_k$. Wegen Lemma bfs3 gilt $dist[v_i] \leq dist[v_{i+1}]$, wenn v_i vor v_{i+1} in Q eingefügt wurde. Wegen Lemma bfs2 gilt $dist[v] \geq \delta(s,v)=k$.
     Deshalb muss v in Q eingefügt werden, nachdem alle $u \in V_{k-1}$ in Q eingefügt wurden.
     Da δ(s,v)=k, gibt es einen Weg der Länge k von s nach v und somit einen Knoten $u \in V_{k-1}$, so dass $(u,v) \in E$. nochmal IV anwenden ... ✔