Basic graph algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- Example problems
  Let G=(V,E) be a graph (directed or undirected).

  - Problem 1:
    - Input: Graph G in adjacency-list form.
    - Output: „yes“, if G contains a cycle; „no“ otherwise
  - Problem 2:
    - Input: Graph G in adjacency-list form.
    - Output: Number of graph-components.
  - Problem 3:
    - Input: Graph G in adjacency-list form.
    - A spannning tree for each component..

Basic graph algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

- Depth-First-Search (dfs)
  - **Input**: Graph G=(V,E) in form of a adjacency-list adj[v], i.e. an adjacency-list is given for each node.
    Output: edge set T (spanning forrest) and edge set B = V \ T.
  - Idea:
    - *Discover the graph node by node, starting from the last discovered one.*
    - If all adjacent nodes of the last discoved node v have been discovered before already, jump back to that node which was discovered just before v. Distinguish spanning tree nodes and cycle nodes.
    - If nodes stay undiscovered, start a new dfs on such a node.

Basic graph algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- Depth-First-Search, dfs

  – For orientation:
    - At the beginning:all nodes arecolored white.
    - Discovered nodes become grey.
    - Completely examined nodes become black.
    - There are two time stamps: d[v] and f[v] (bewteen1 and 2|V|)
    - d[v]: v is dicovered (discover-time)
    - f[v]: v is finished (finishing-time)

Basic graph algorithms

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)

„:=“: variable assignment
π stores predecessors (Vorgänger)

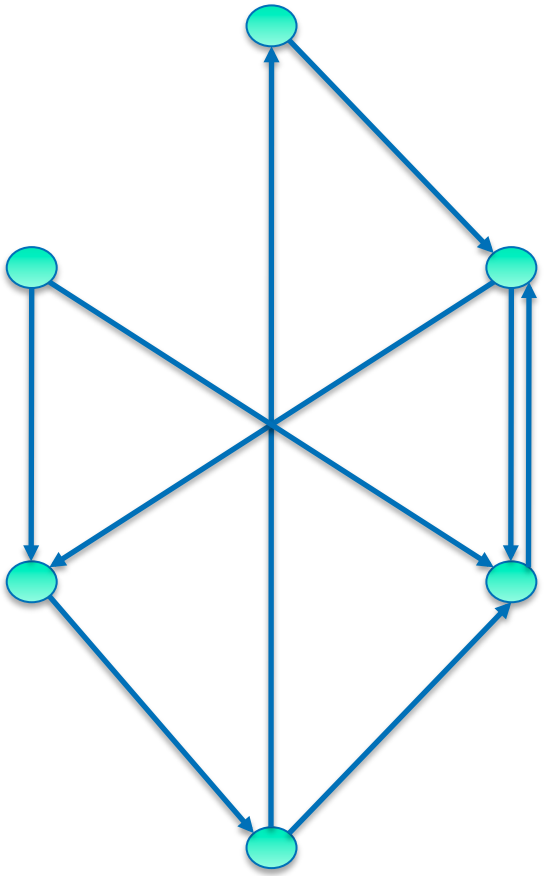
„==“: test on equality

Basic graph algorithms

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
```

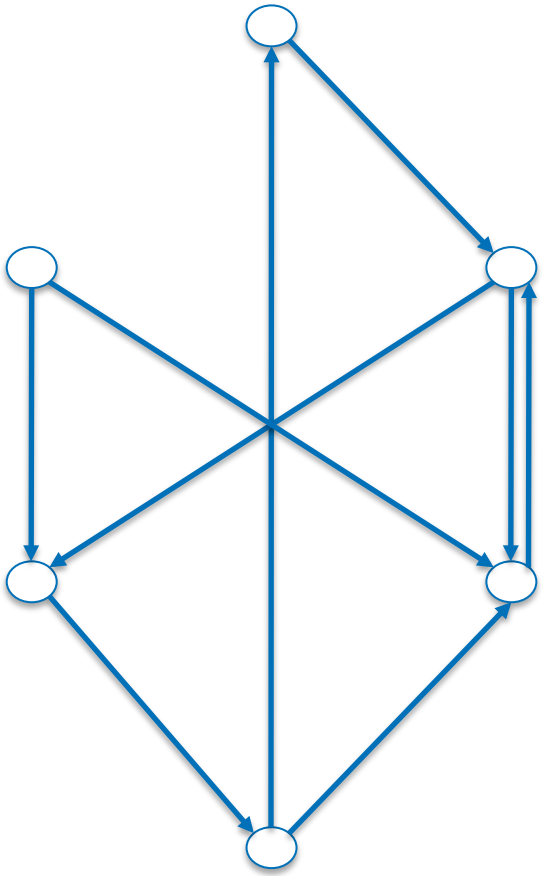


Basic graph algorithms

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
```



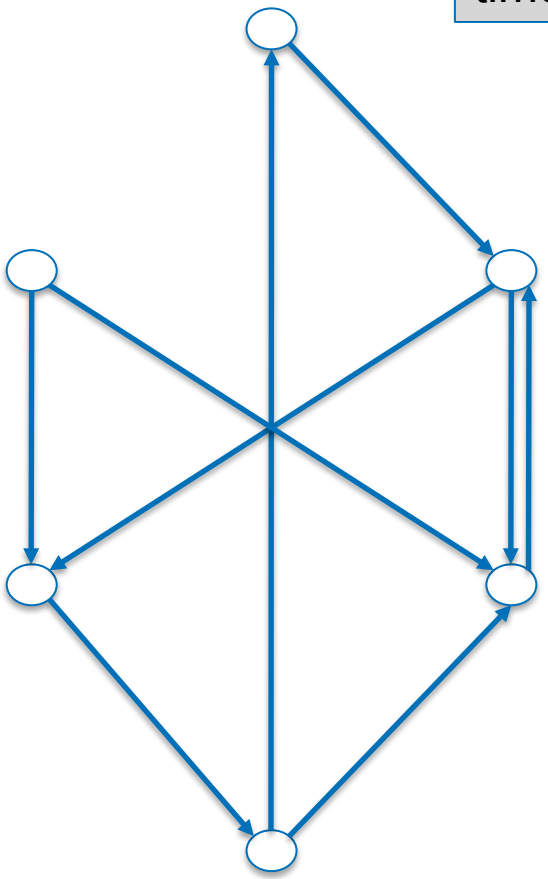
Basic graph algorithms

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



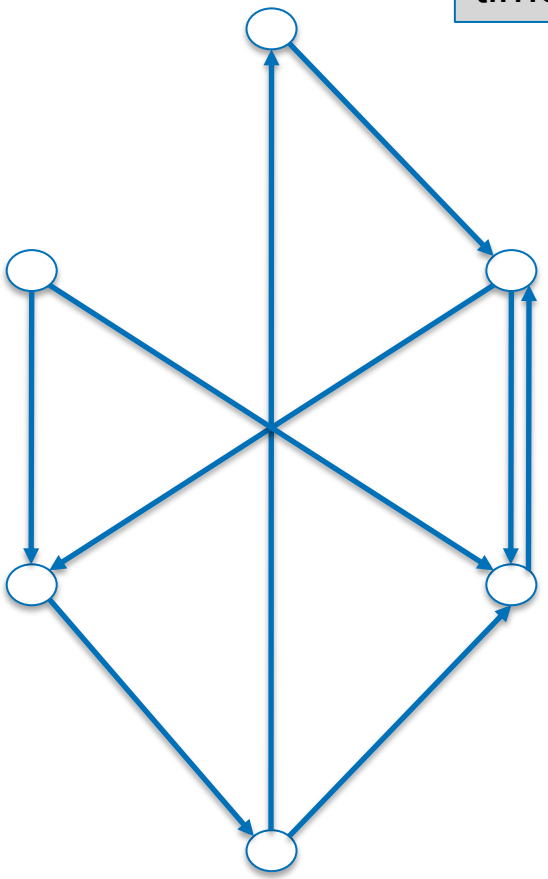
Basic graph algorithms

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

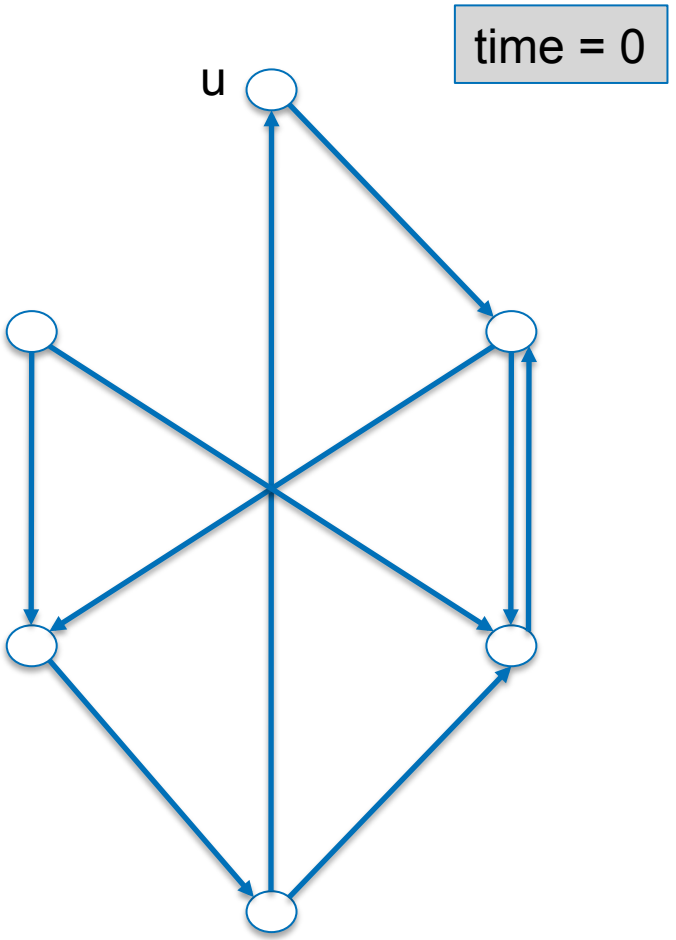


Basic graph algorithms

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
```

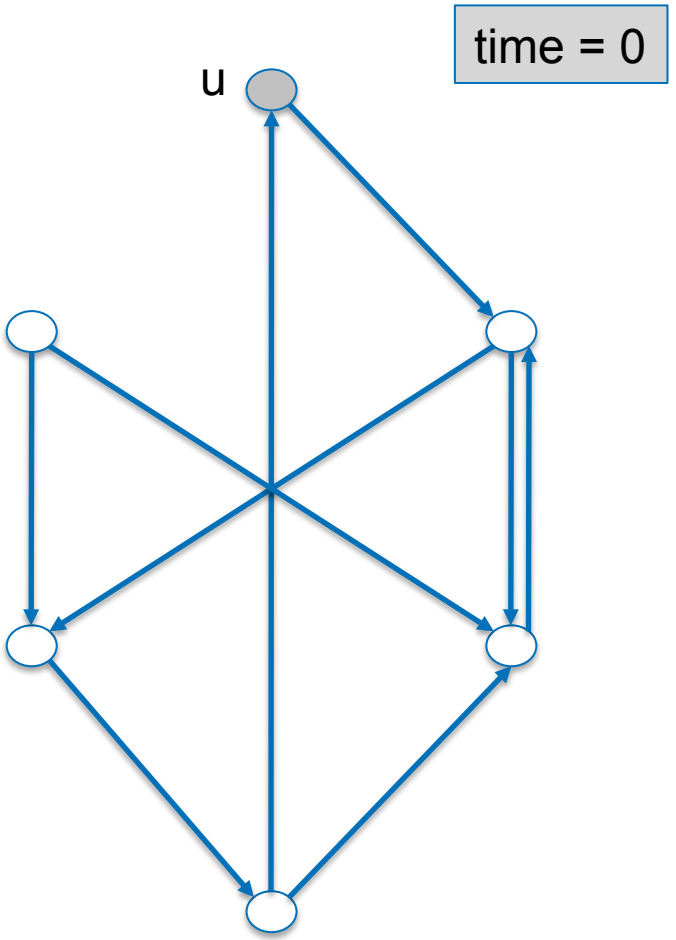


Basic graph algorithms

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
```

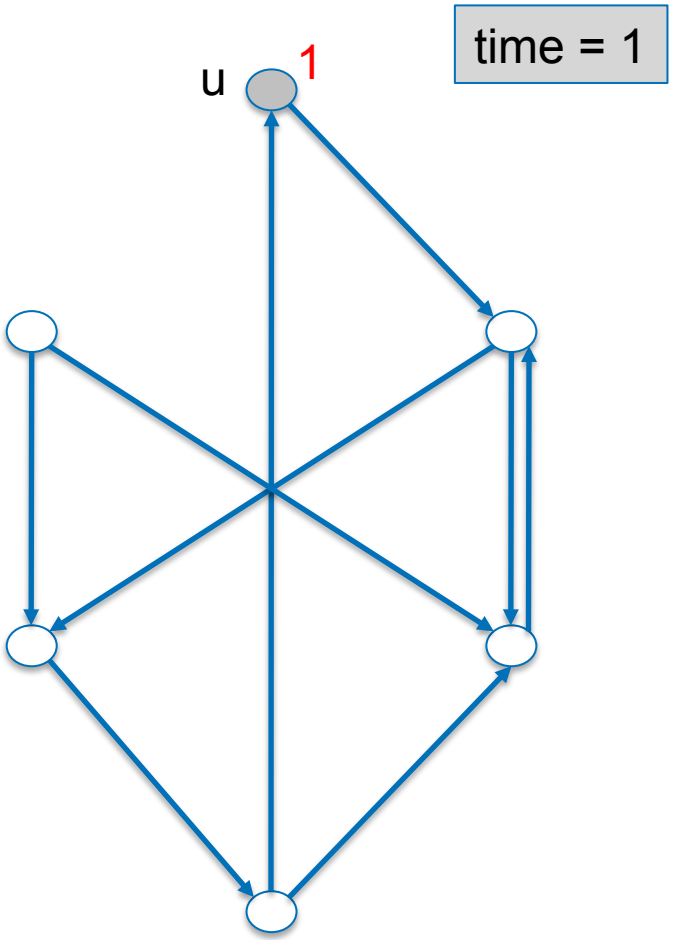


Basic graph algorithms

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
```

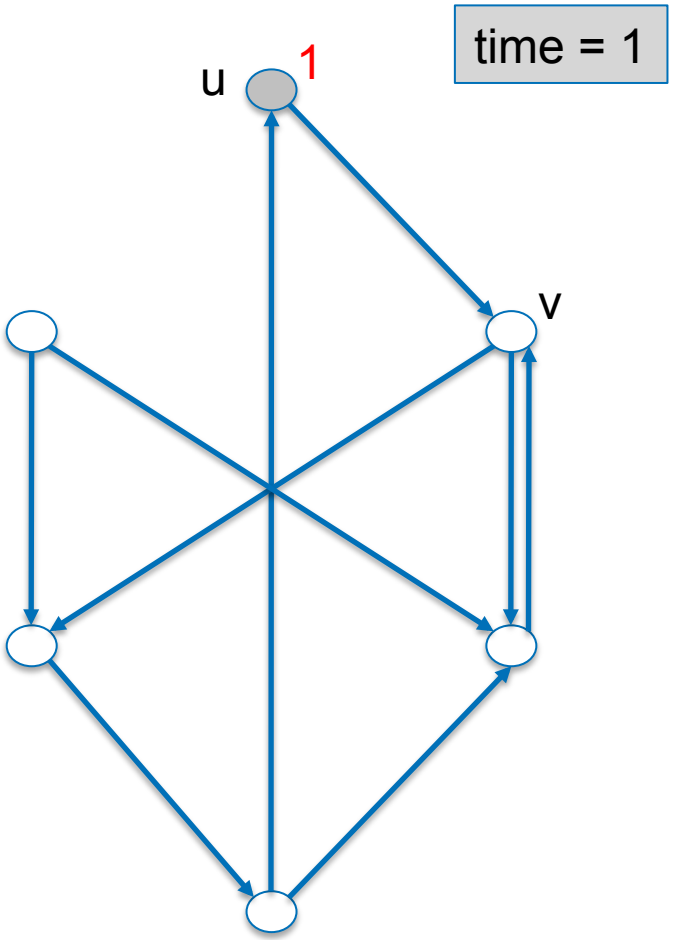


Basic graph algorithms

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
```

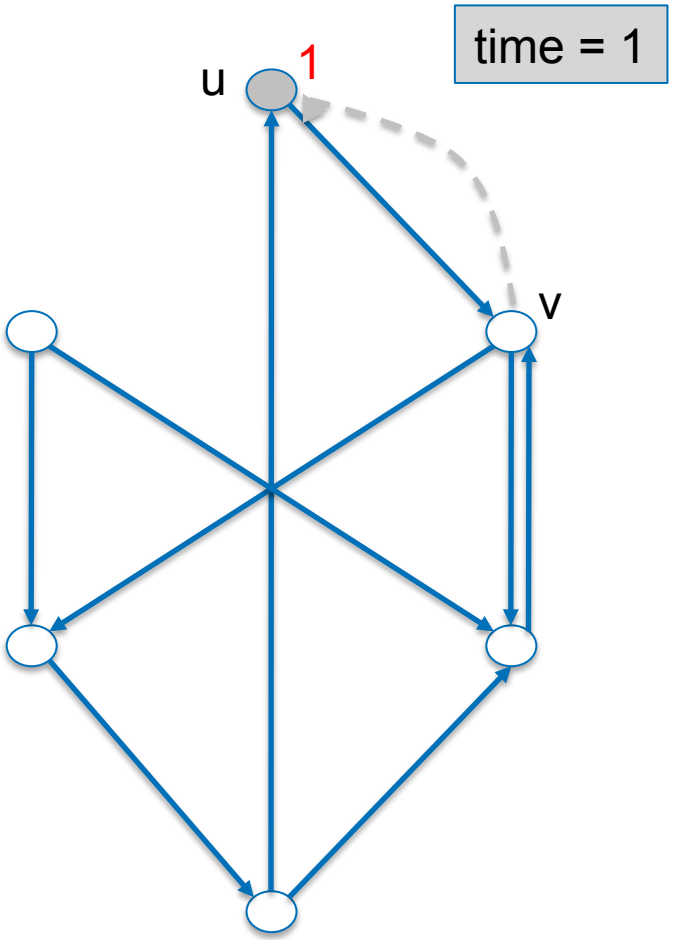


Basic graph algorithms

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
```

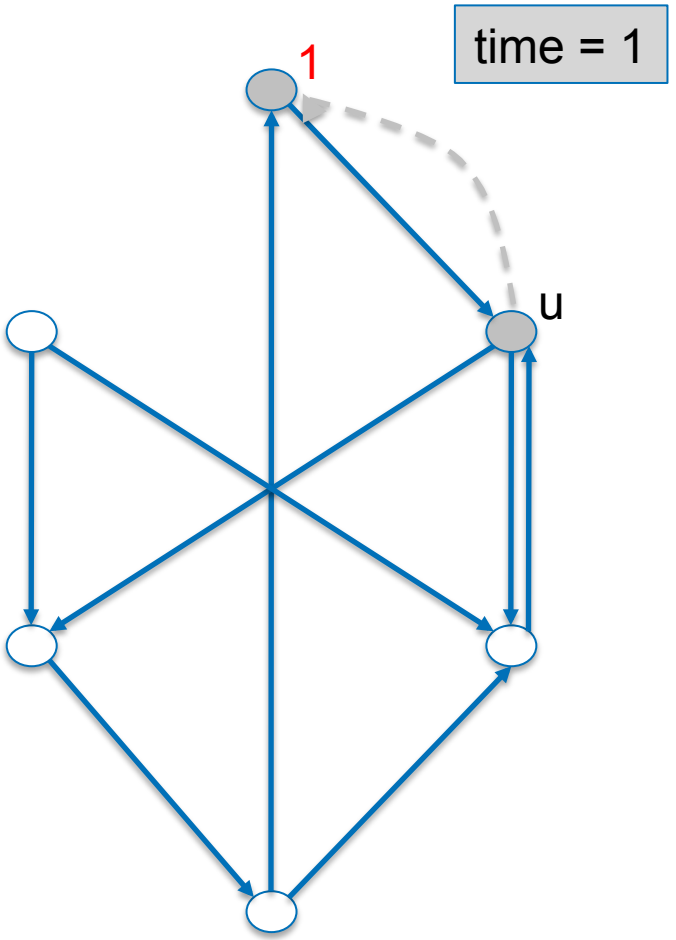


Basic graph algorithms

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
```

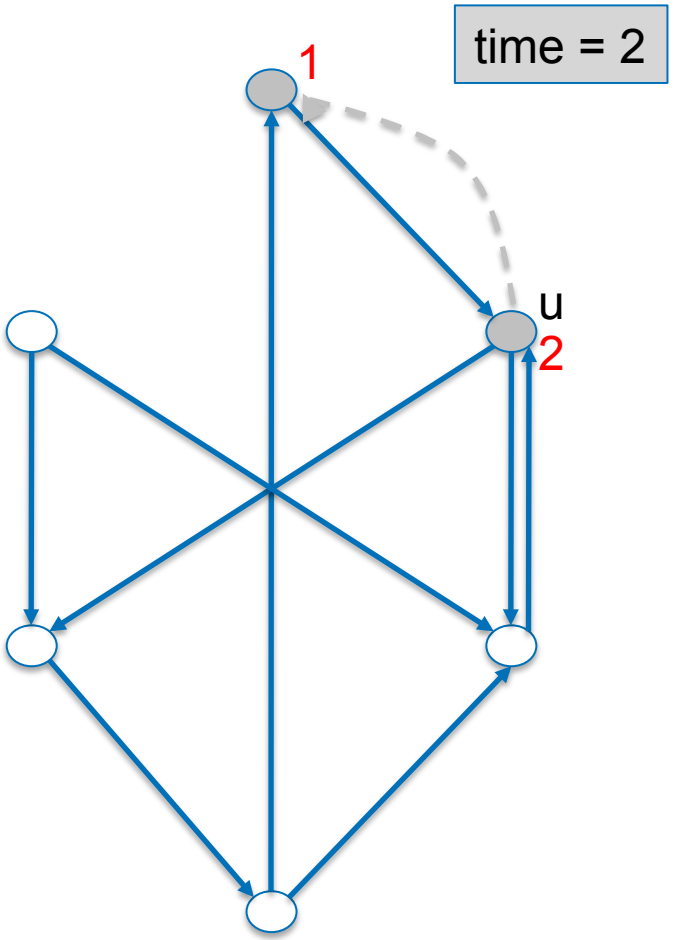


Basic graph algorithms

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
```

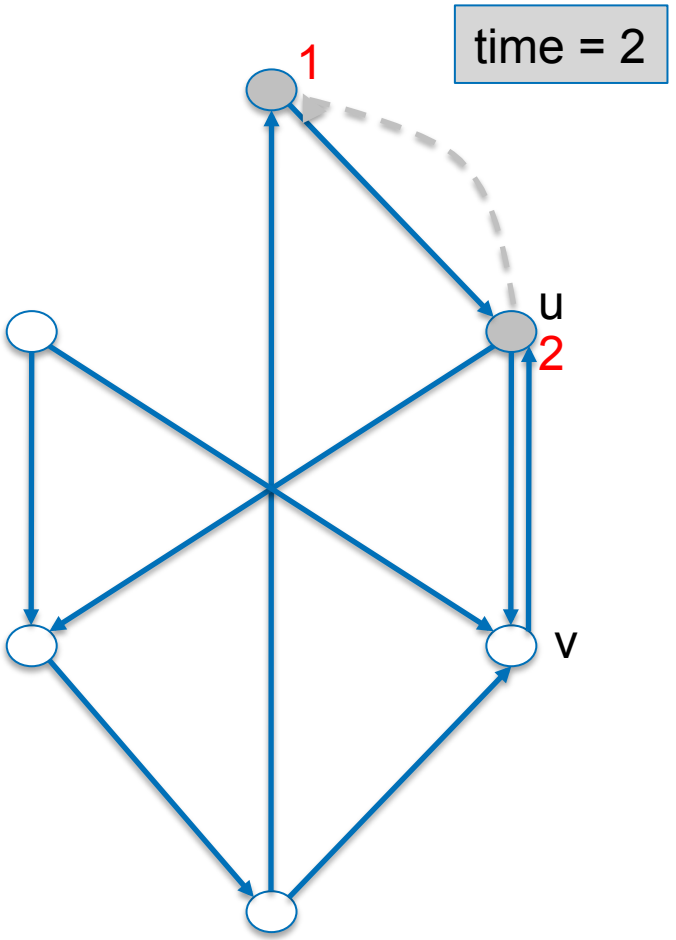


Basic graph algorithms

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
```

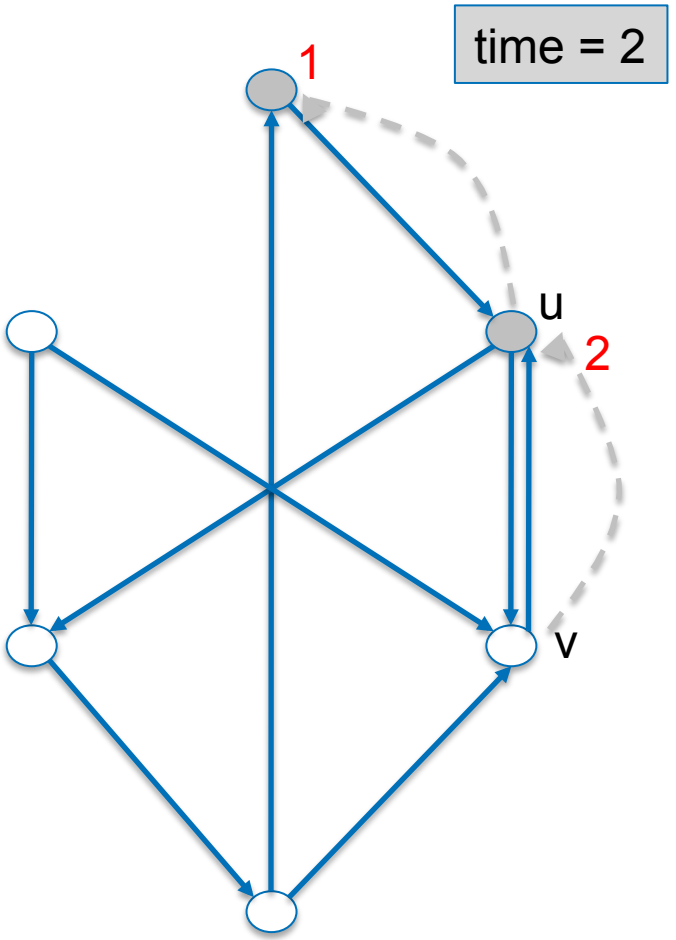


Basic graph algorithms

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
```

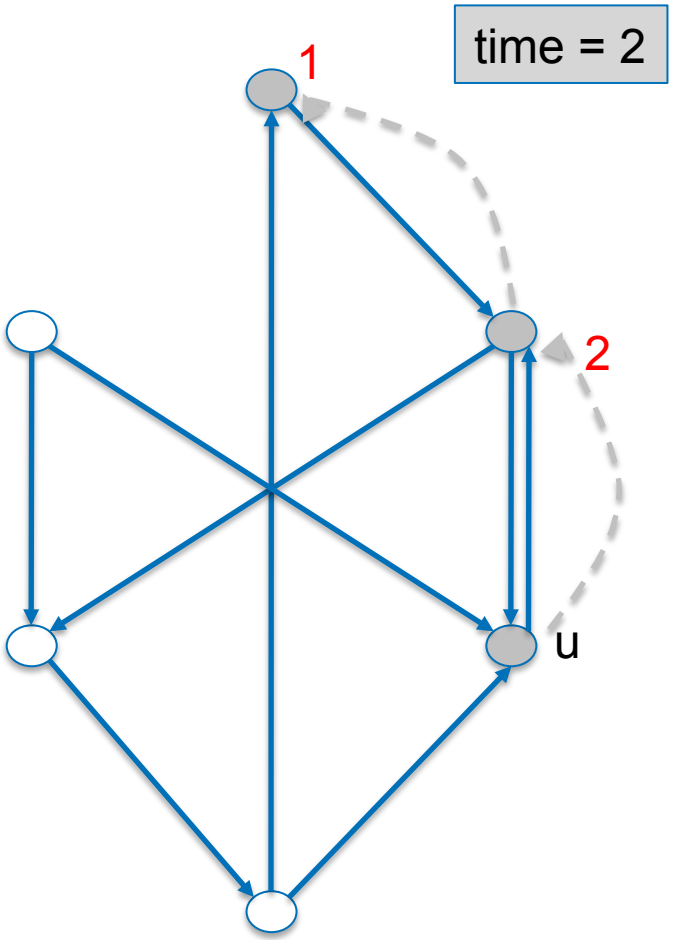


Basic graph algorithms

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
```

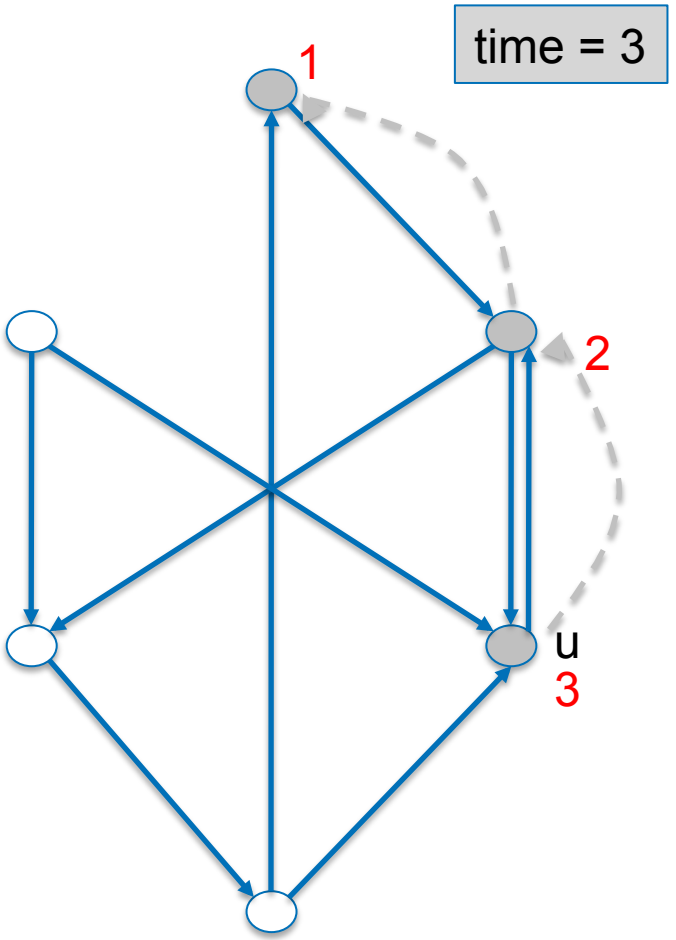


Basic graph algorithms

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
```

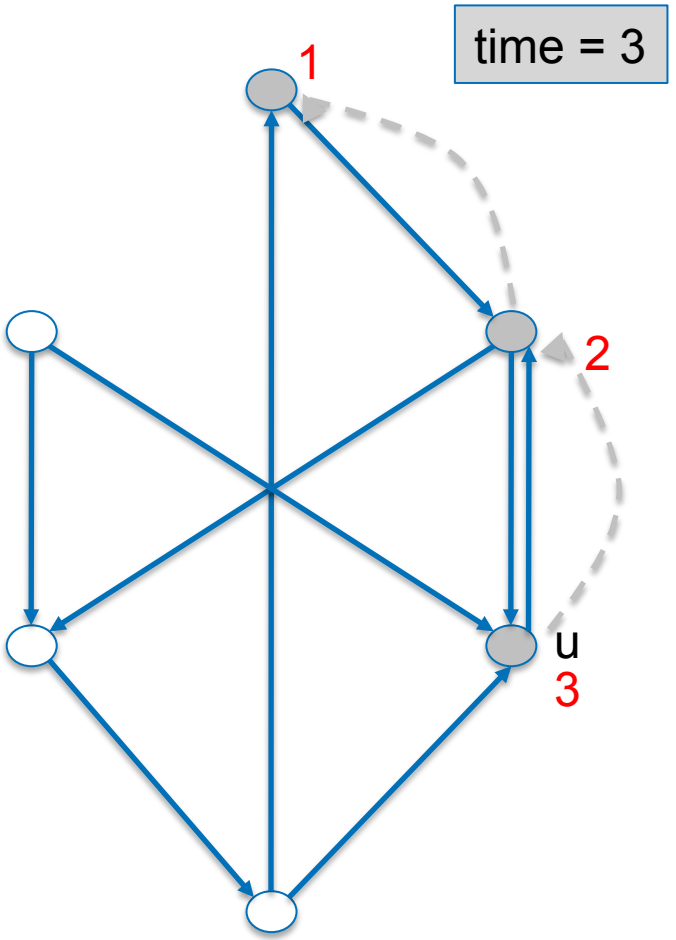


Basic graph algorithms

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
```

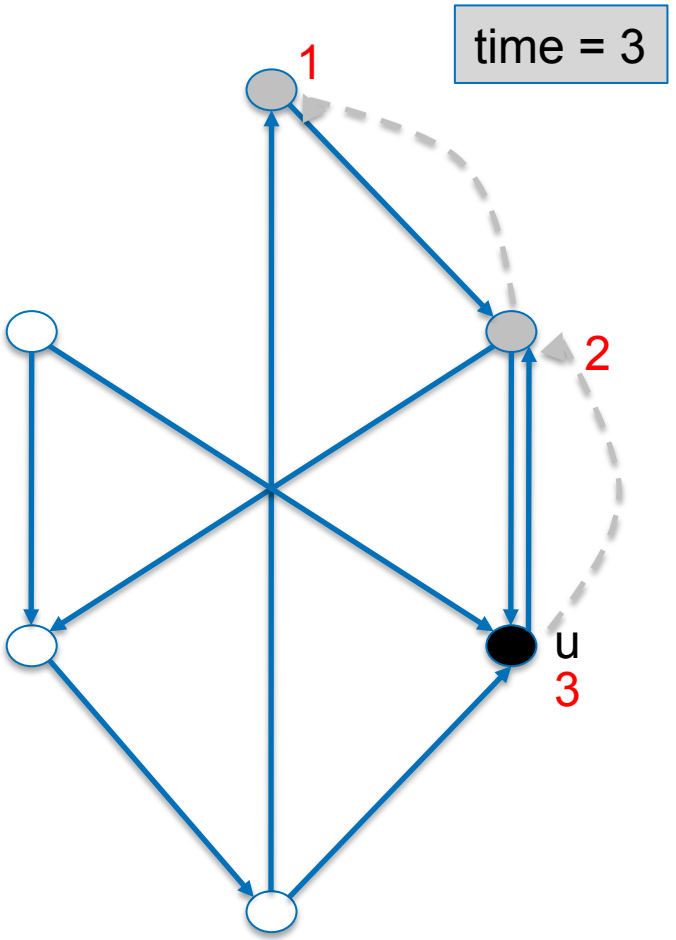


Basic graph algorithms

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
```

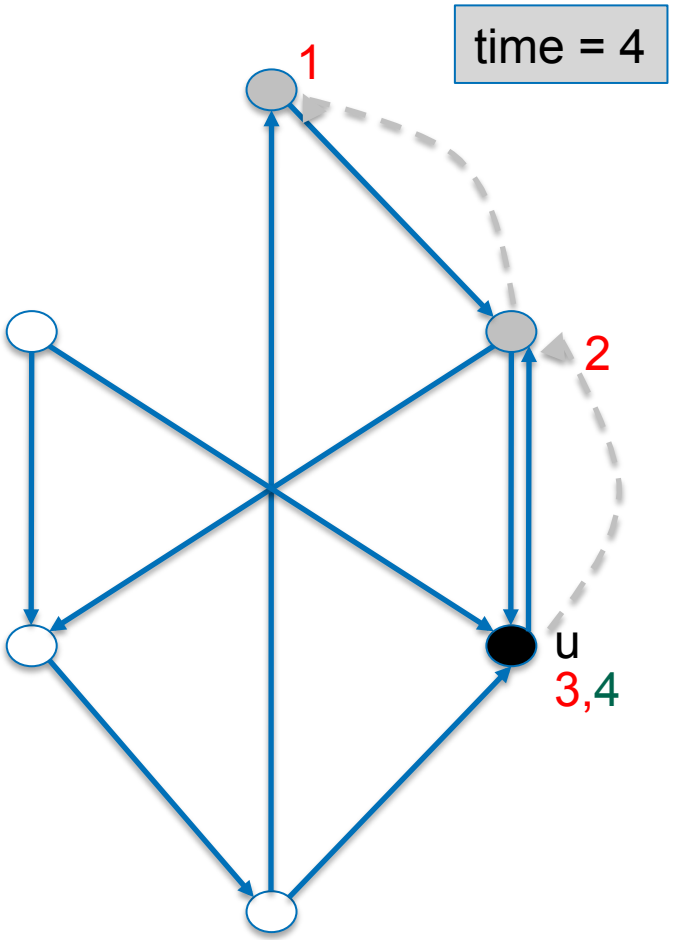


Basic graph algorithms

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
```

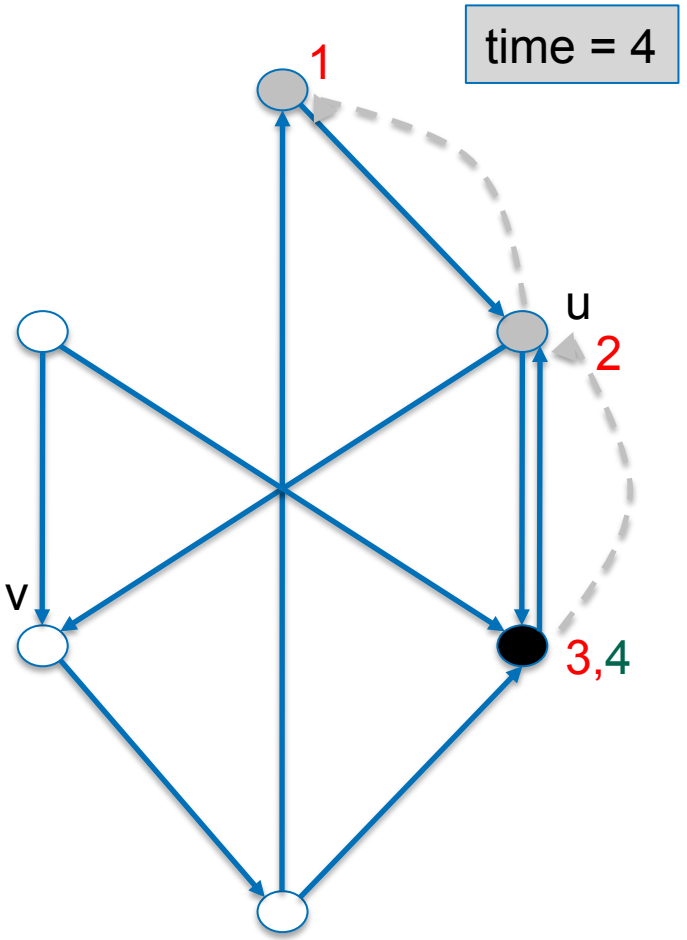


Basic graph algorithms

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
```

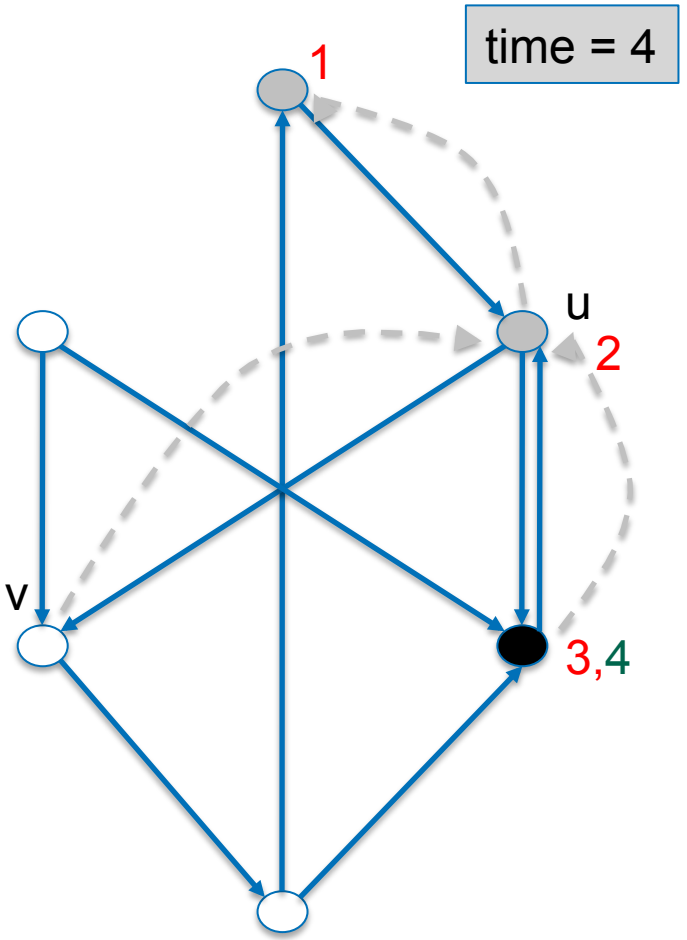


Basic graph algorithms

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
```

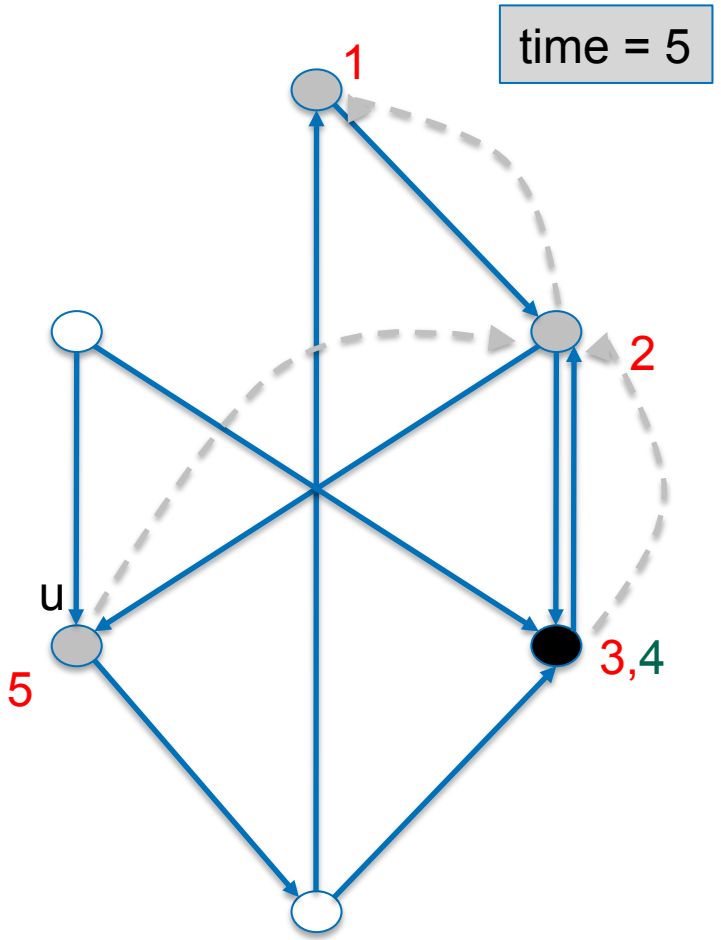


Basic graph algorithms

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
```

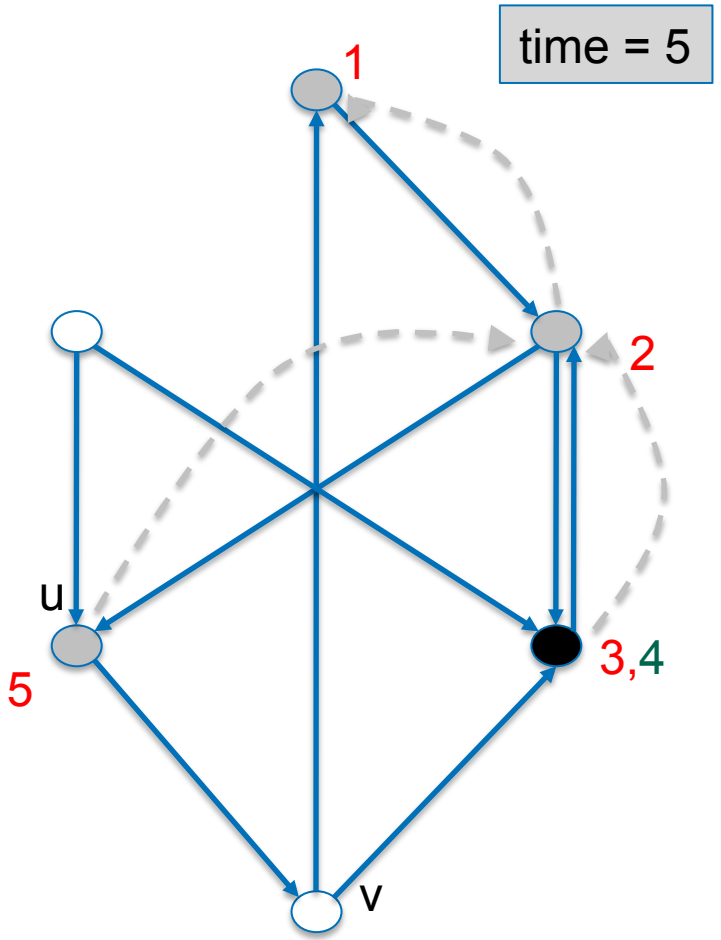


Basic graph algorithms

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
```

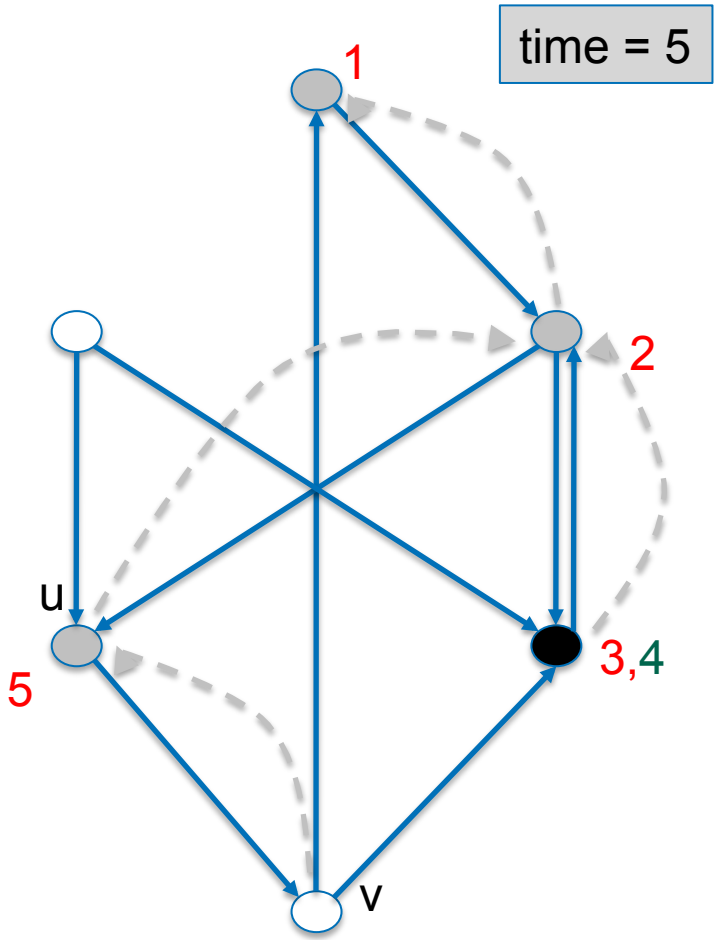


Basic graph algorithms

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
```

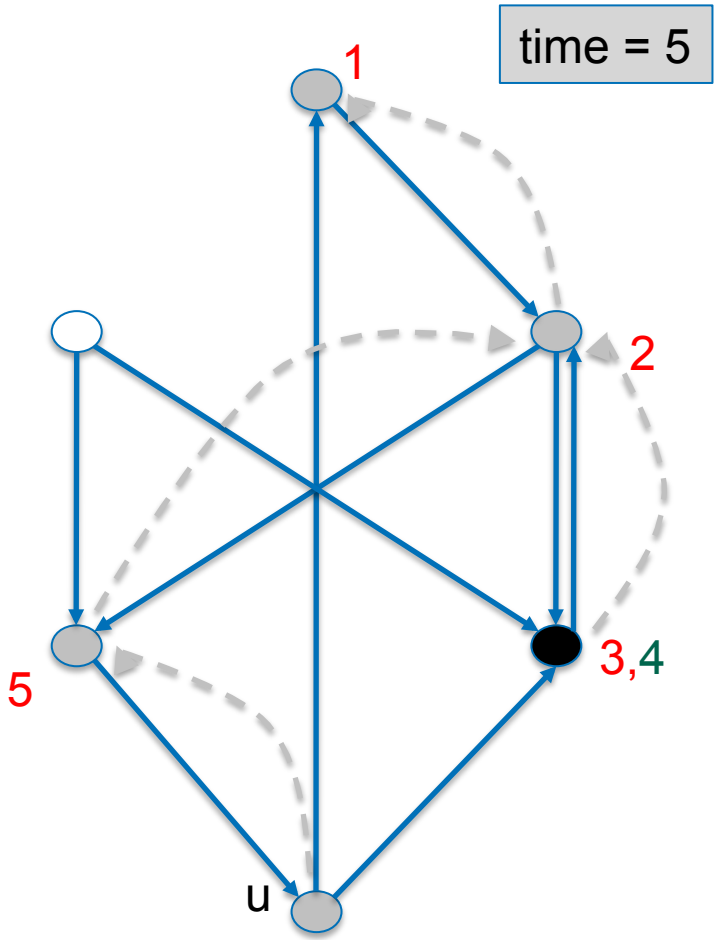


Basic graph algorithms

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
```

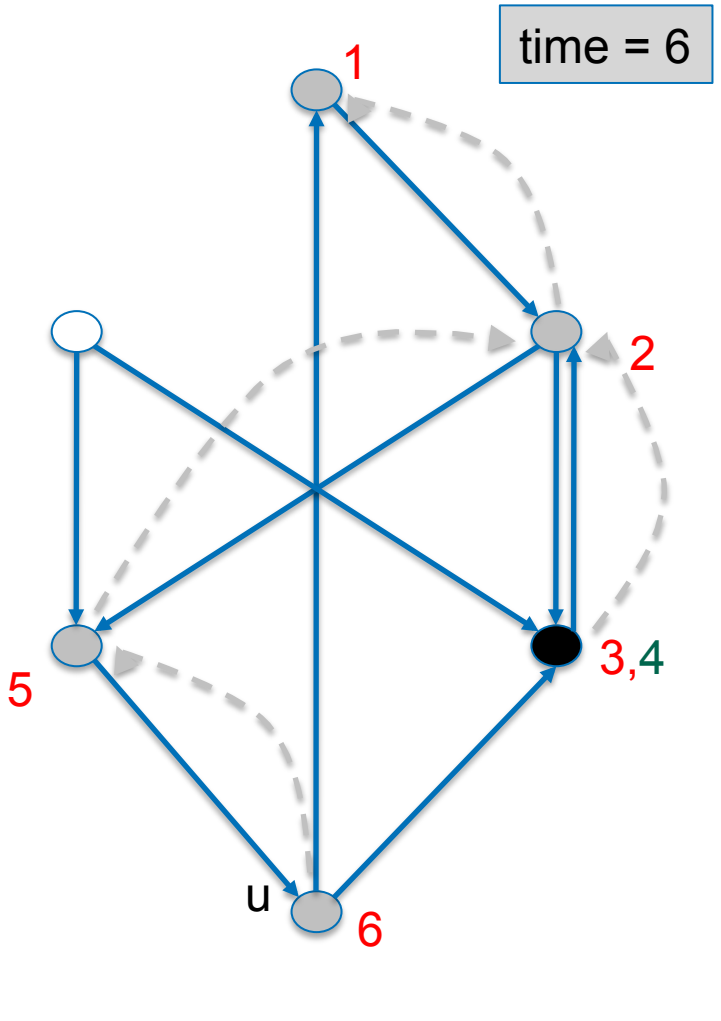


Basic graph algorithms

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
```

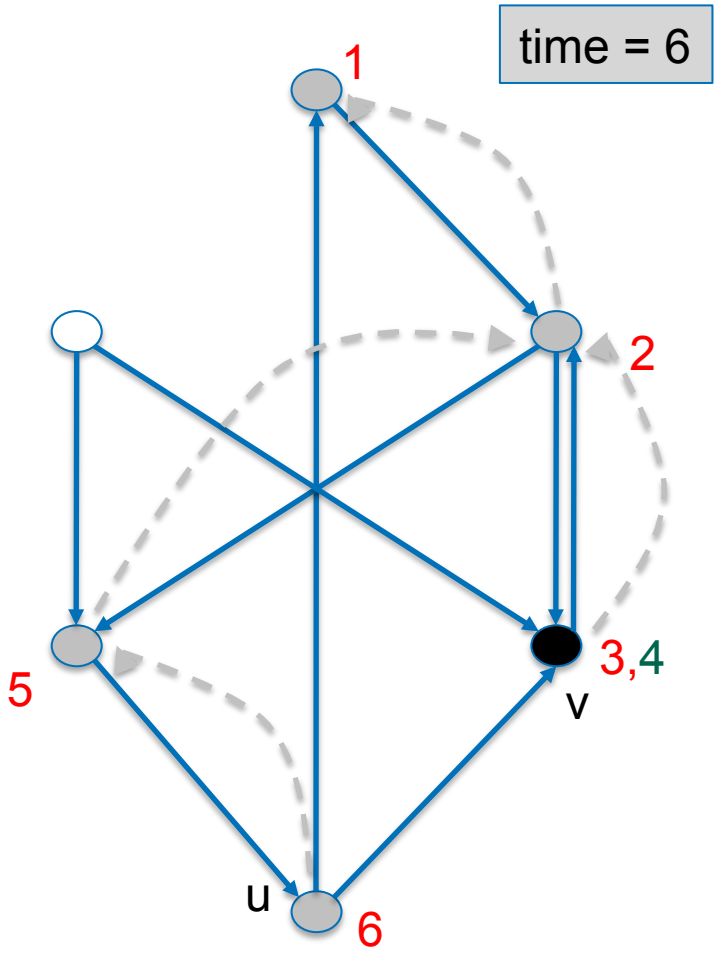


Basic graph algorithms

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
```

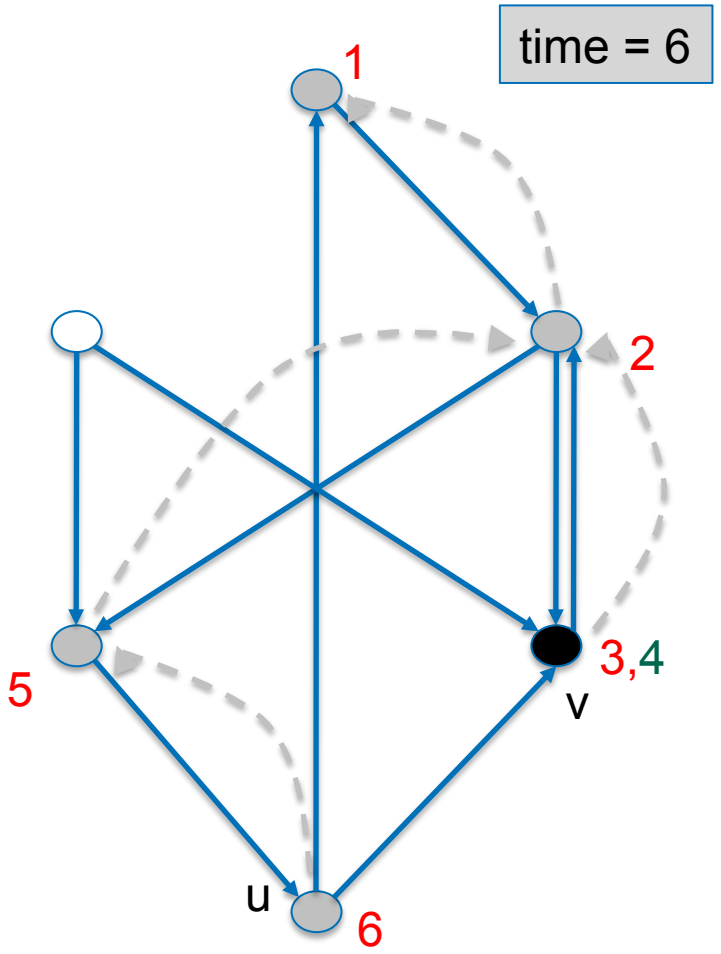


Basic graph algorithms

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
```

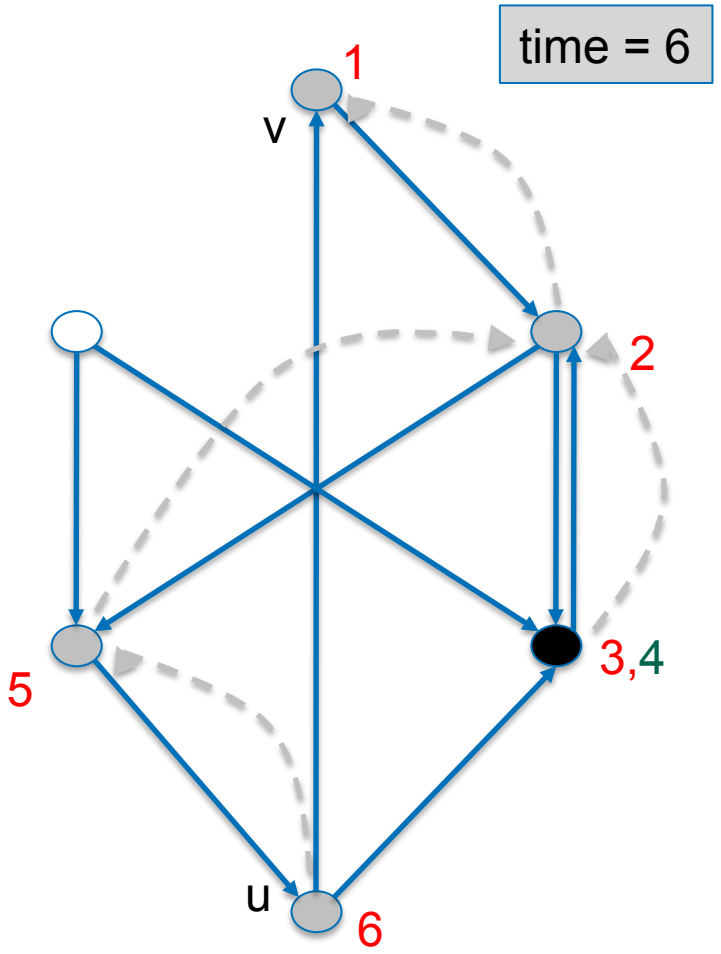


Basic graph algorithms

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
```

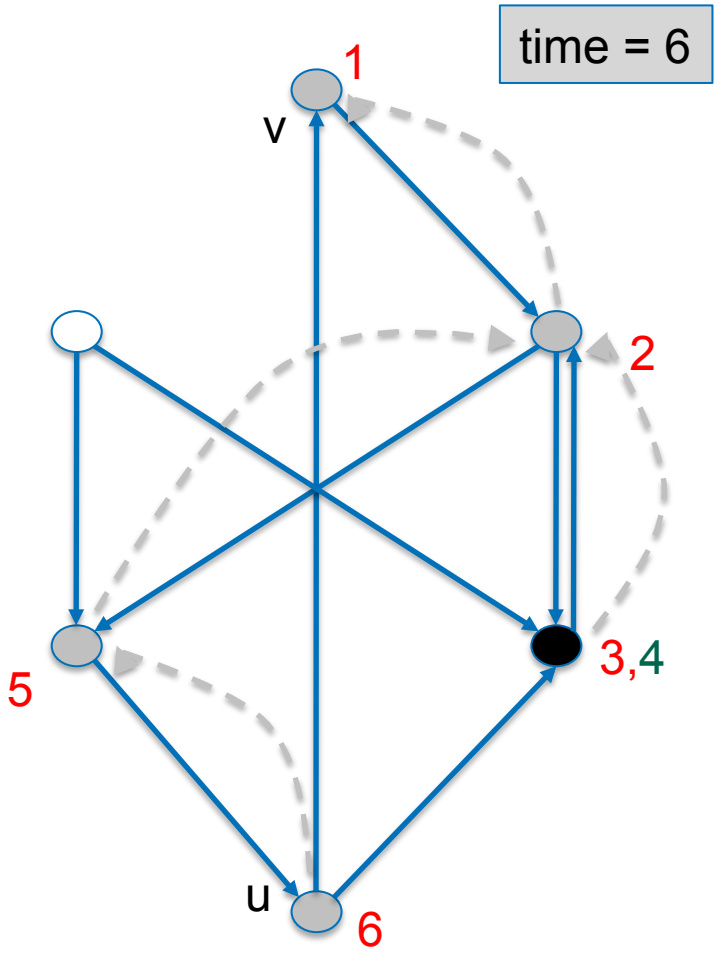


Basic graph algorithms

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
```

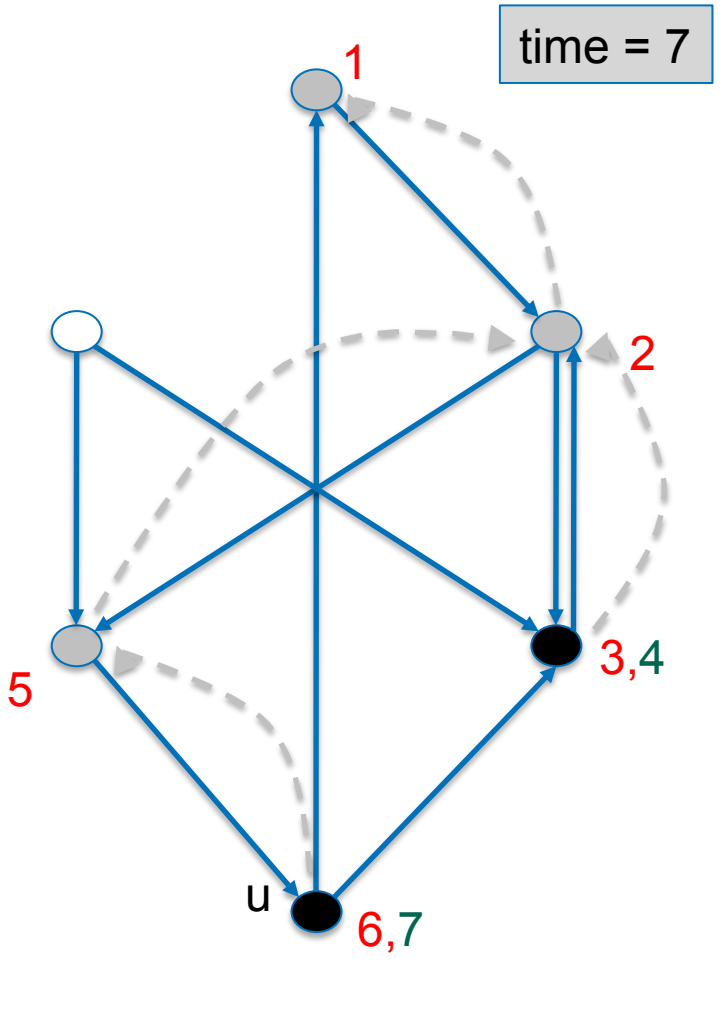


Basic graph algorithms

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
```

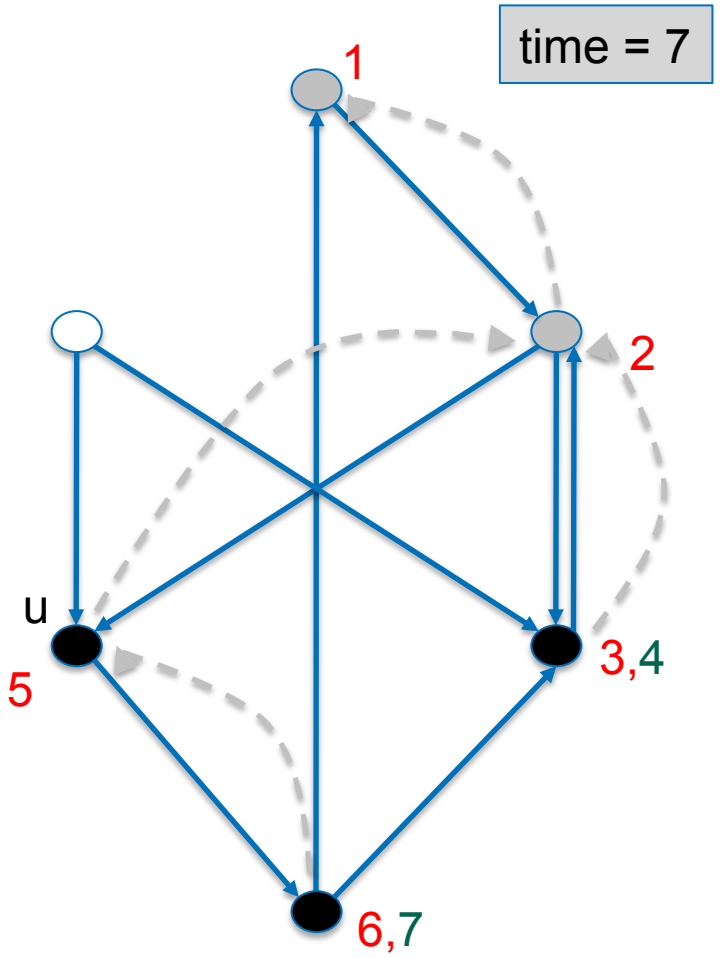


Basic graph algorithms

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
```

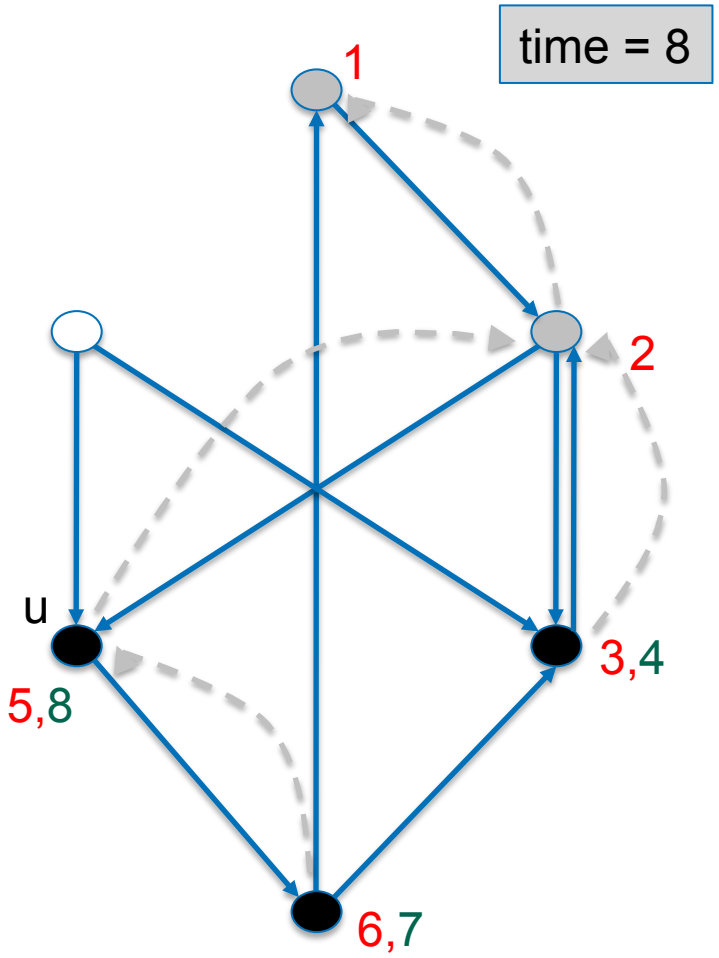


Basic graph algorithms

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
```

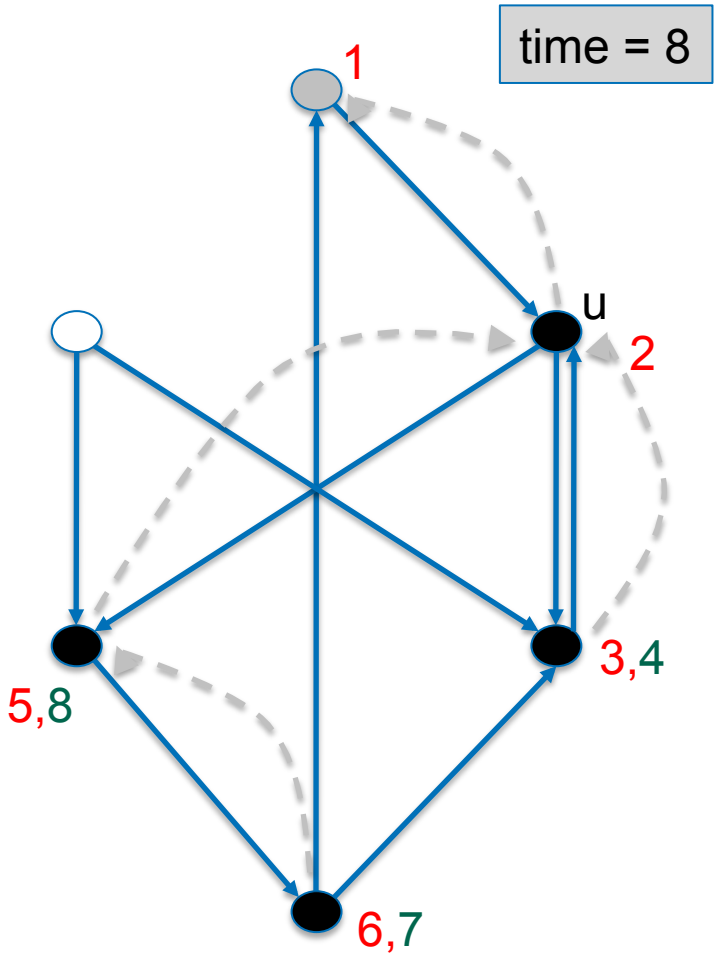


Basic graph algorithms

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
```

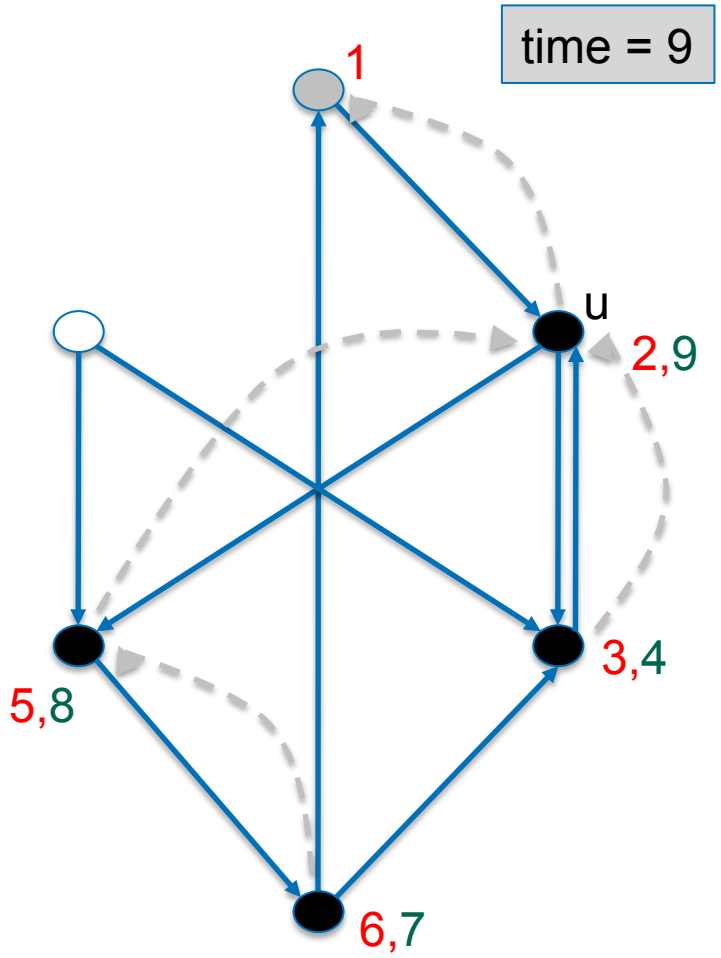


Basic graph algorithms

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
```

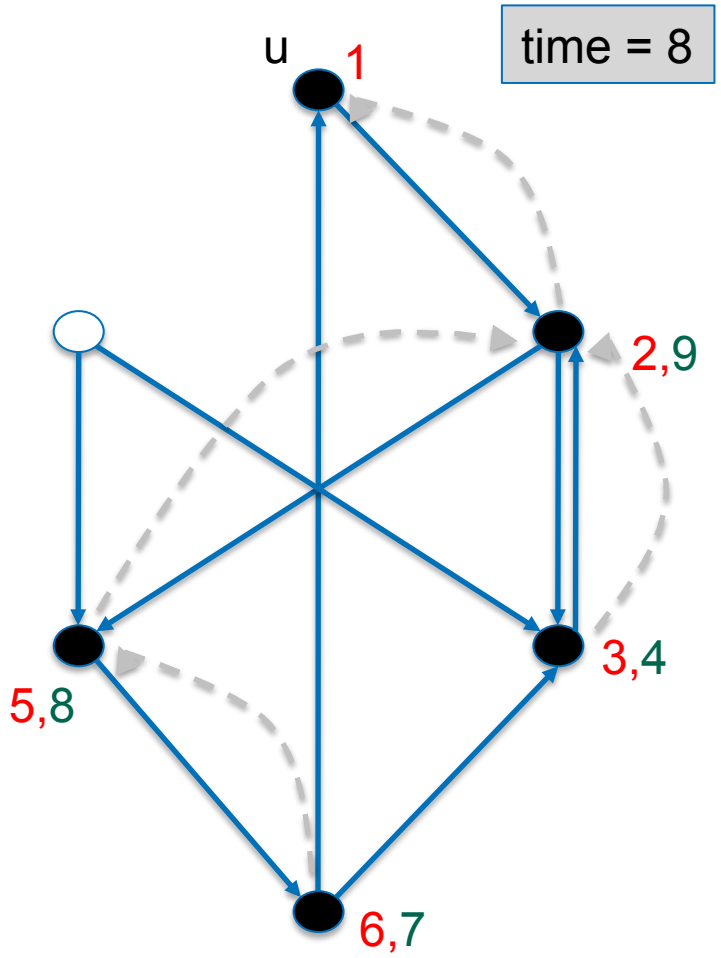


Basic graph algorithms

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
```

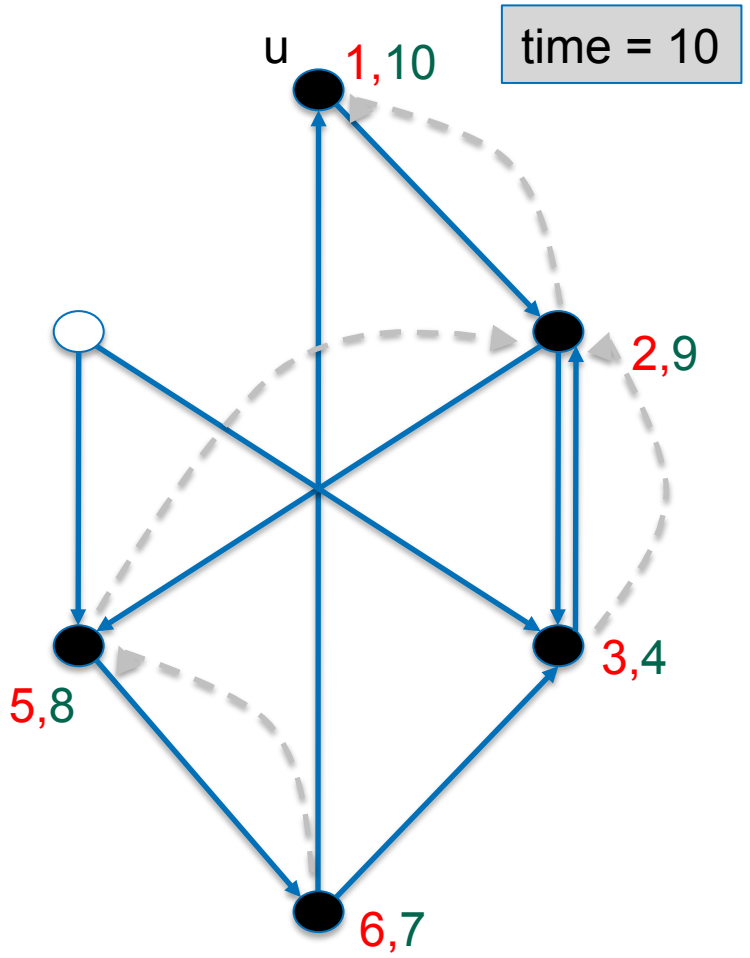


Basic graph algorithms

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
```

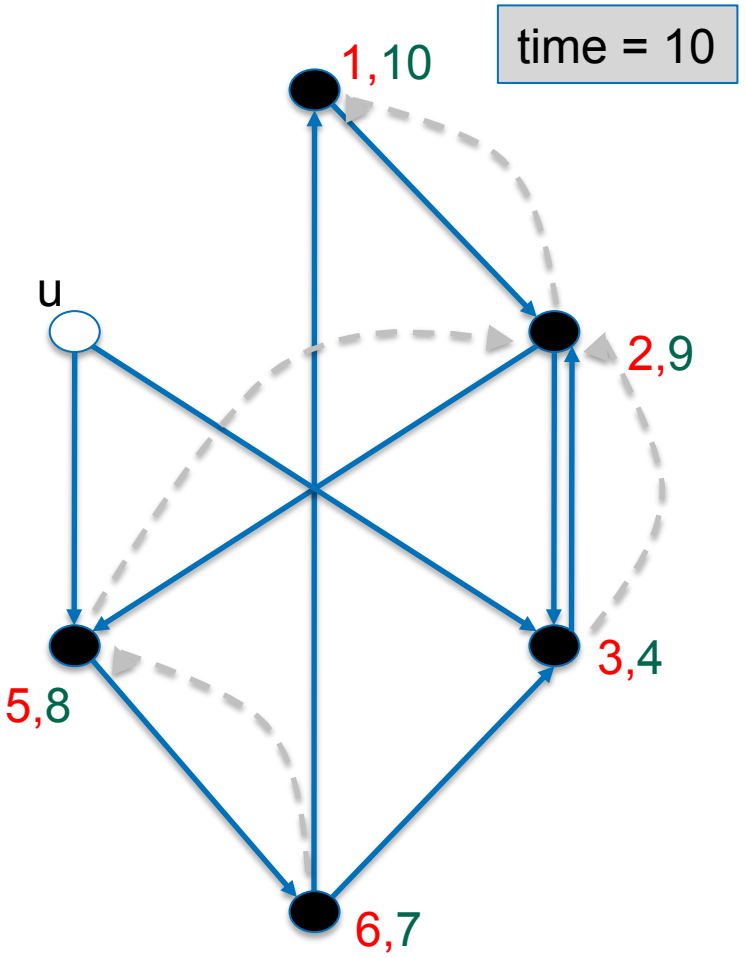


Basic graph algorithms

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
```

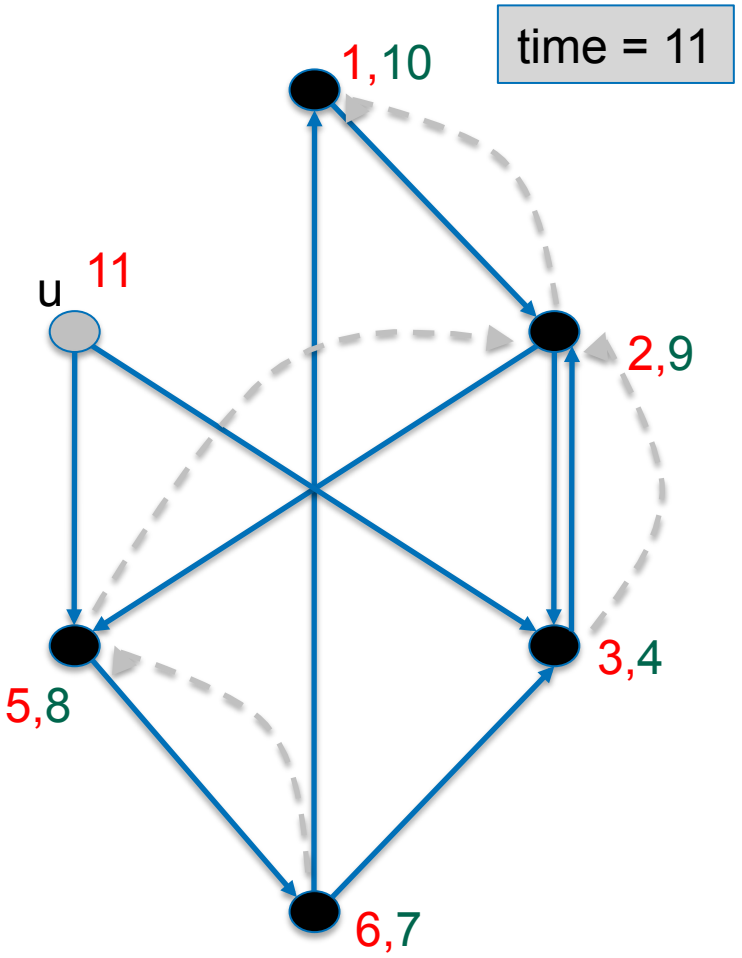


Basic graph algorithms

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
```

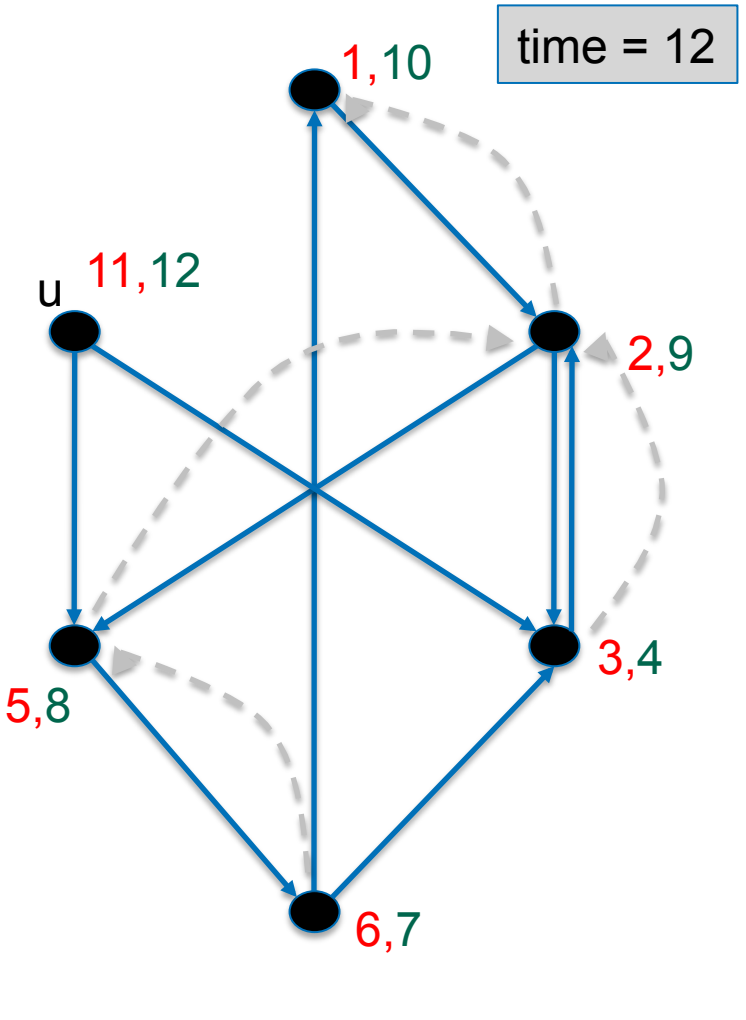


Basic graph algorithms

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
```



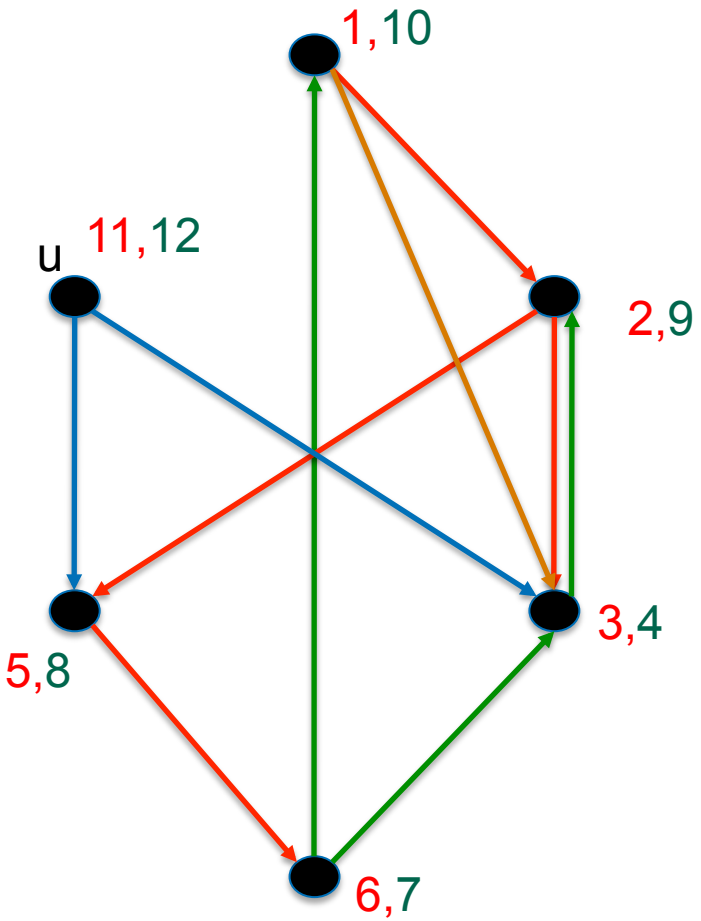
Basic graph algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- Depth-First-Search, dfs

  – DFS algorithmus partitions the edges into four subsets
    - tree edges are
      → edges of the spanning forest
    - back edges are
      → edges (u,v), connecting u with earlier discovered predecessors of v
    - forward edges are
      → non-tree edges (u,v), wich connect u with earlier discovered successors of v.
    - cross edges are all other edges.

  – Runtime of DFS is O(|V|+|E|)



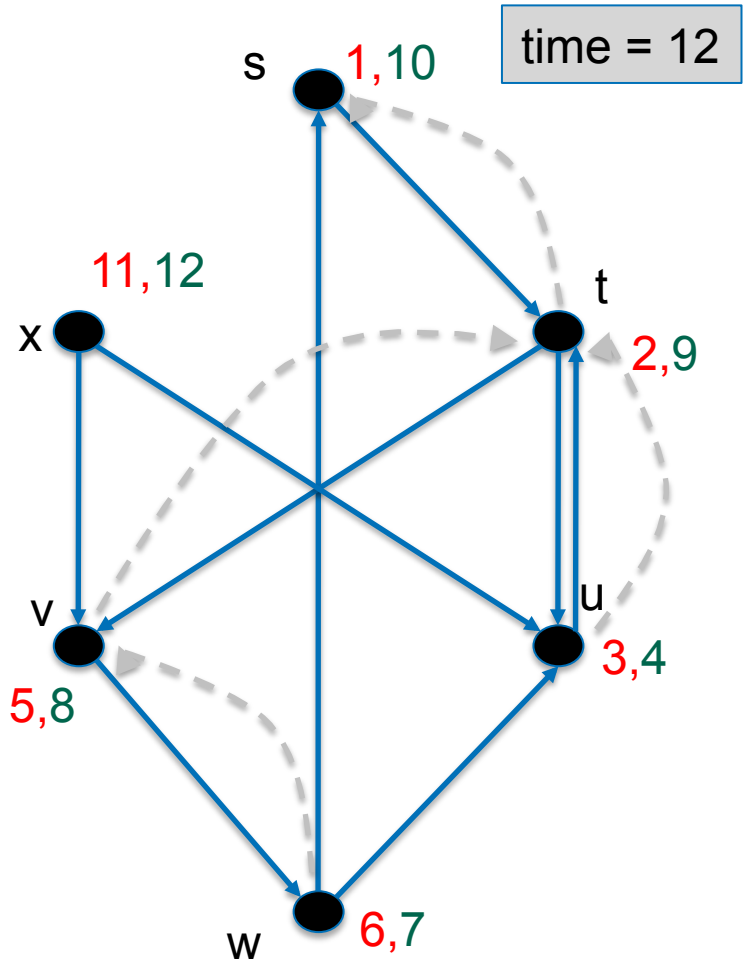
Basic graph algorithms

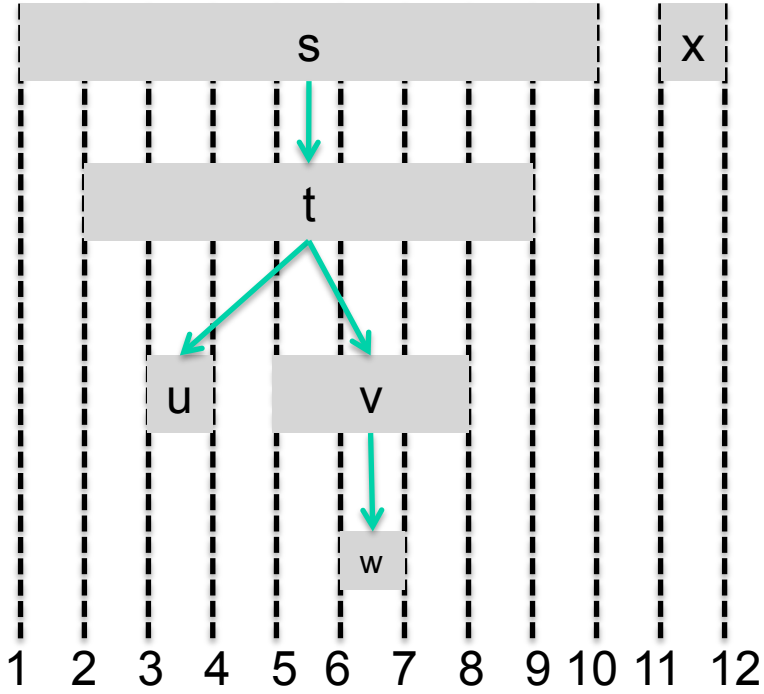
TECHNISCHE UNIVERSITÄT DARMSTADT

- Paranthesis theorem
  - Let u and v b nodes of a Graph G. When the DFS search has been finished, one of the following three statements is true:
    - The intervals [d[u],f[u]] and [d[v],f[v]] are disjoint
    - The interval [d[u],f[u]] is completely contained in [d[v],f[v]], and u is a successor of v in the DFS tree
    - The interval [d[v],f[v]] is completely contained in the interval [d[u],f[u]], and v is a successor of u in the DFS-tree.

Basic graph algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT





Basic graph algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- Paranthesis Theorem, Proof

  – Let d[u]<d[v]. Then, there are two cases:

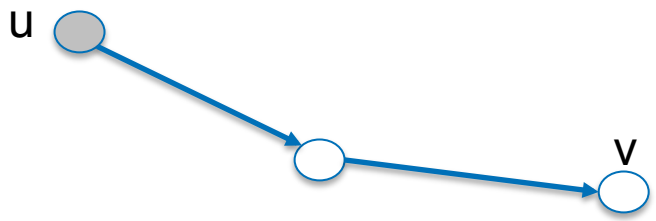
    1. d[v]<f[u]: v has been discovered when u was still grey. Thus, v is successor of u . Because v has been discovered later then u, it follows that all outgoing edges of v were examined before the search process returned to u. Therefore, firstly v and then u were finished. Therfore f[v]<f[u]. In summary : d[u]<d[v]<f[v]<f[u].
    2. f[u]<d[v]: v was discovered when u had been already finished. Of course are d[u]<f[u] and d[v]<f[v] , and thus, the intervals [d[u],f[u]] and [d[v],f[v]] are disjoint.

  – The case d[v]<d[u] is shown analogously.

Basic graph algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- White-Path Theorem
  - In a DFS-forest of a directed or undirected graph G=(V,E), a node v is successor of another node u if and only if node v can be reached from u via white nodes at time d[u].

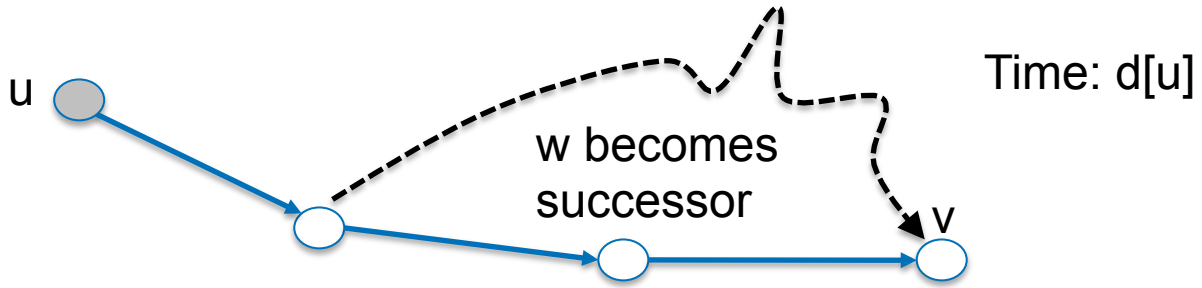


⇒: Let v be a successor of u, such that d[v]>d[u] . Moreover, let u not yet finished. Obviously, the path between u and v was white at time d[u], because DFS has moved from u to v with the help of line 4 of DFS-Visit.

Basic graph algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- White-Path Theorem



⇐: Assume, there is a white path from u to v at time d[u]. Wlog. we assume that v does not become a successor of u and every other node than v becomes a successor of u. (otherwise we do the following thoughts for the first non-successor of u) Let w be the last node on the white path from u to v becoming successor of u. Then obviously, it is valid: f[w]<f[u].
Because d[v]>d[u] and d[v]<f[w] (v is direct successor of w), it is valid: d[u]<(d[w]<)d[v]<f[w]<f[u]. With the help of the paranthesis follows f[v] < f[w]. After all, one of our assumptions does not hold; v must be a successor of u.

Basic graph algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- Let G=(V,E) an **undirected** graph. Then is valid:
  - There are only tree- and backward edges in the DFS-forest.
  - The set of all tree edges builds a forest. Each connecting component generates a spanning tree.
  - G is cycle-free if and only if the DFS-forest contains no backward edges.

  Proof: Exercise

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Computation of shortest paths in graphs

- A weighted graph G is a tupel (V,E) together with a weight-function f, where
  $E \subseteq V \times V$ und $f: E \rightarrow \mathbb{Q}$

- Let $u,v \in V$. A shortest path from u to v is a path with smallest possible weight from
  u to v- The weight of a path is the sum of the edge weights on this path.

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Computation of shortest paths in graphs

- *Single Source Shortest Path Problem*
  - given: weighted graph G and start node s
  - wanted: the distance δ(s,v) for each node v as well as the shortest path

- *Shortest paths between all pairs of nodes*

- Claim: Let G be a directed graph with non-negative edge weights. If a path w from s to t is a shortest path, and v is a node on that path, then also the paths from s to v and from v to t are shortest paths.

  Simple argument: If there was a shorter path w‘ from s to v, we could shorten the shortest path from s to t.

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra‘s Algorithm

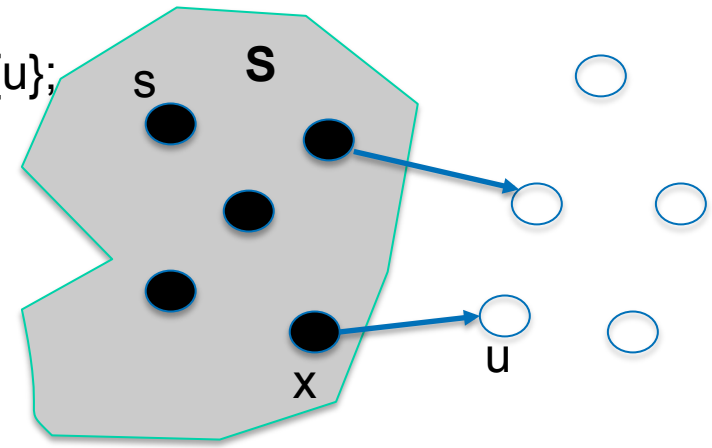
- Preliminaries
  - directed graph G=(V,E), all edge weights positiv
  - G is represented with the help of adjacency lists
  - start node s
  - S: set of nodes where the real distance to s is already known of
  - dist[v] distance estimation of v
  - π[v] predecessor node of v. After all, we can read the desired path from π.
  - set A: A := V \ S

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm

```
Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
S := ∅;
A := V;
while   A ≠ ∅  do
    u := argmin{ dist[a] | a∈A }; A := A \ {u};
    S := S ∪ {u};
    for each node v ∈ Adj[u] do
        if dist[v] > dist[u] + f(u,v) then
            dist[v] := dist[u] + f(u,v);
            π[v] := u;
```

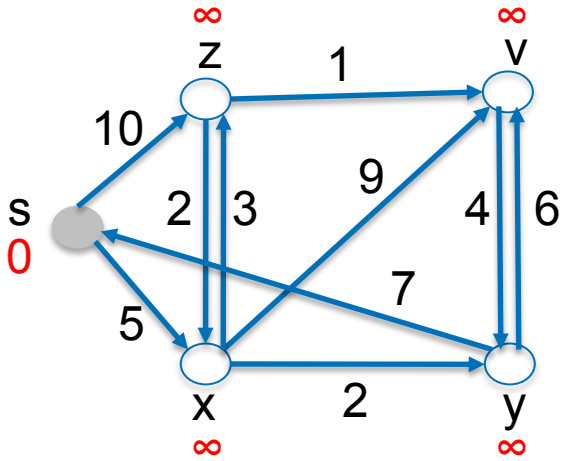


Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm

```
 1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
 2: S := ∅;
 3: A := V;
 4: while   A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A:= A \ {u}
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```



u concerning line 5

A

S

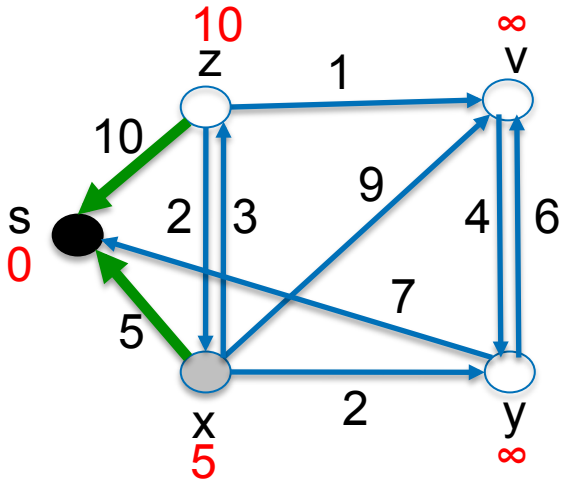
π[v]=u

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm

```
 1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
 2: S := ∅;
 3: A := V;
 4: while   A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A:= A \ {u}
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```



u concerning line 5

A

S

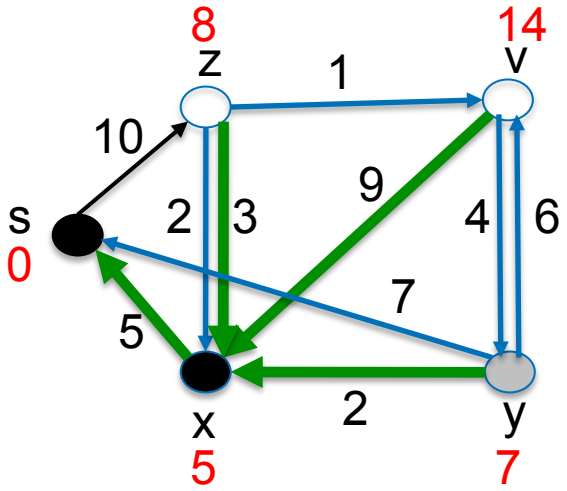
π[v]=u

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm

```
 1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
 2: S := ∅;
 3: A := V;
 4: while   A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A:= A \ {u}
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```



u concerning line 5

A

S

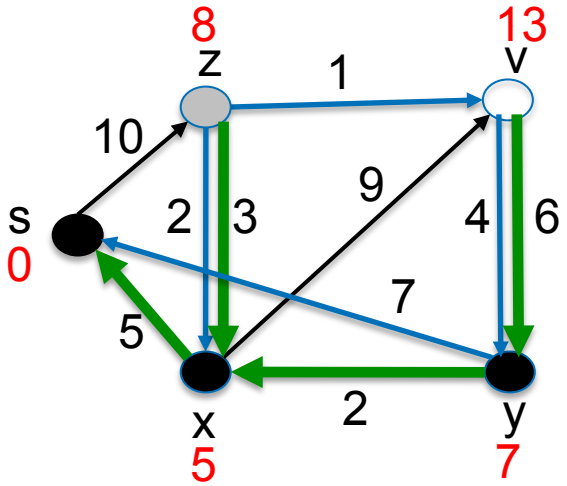
π[v]=u

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm

```
 1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
 2: S := ∅;
 3: A := V;
 4: while   A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A:= A \ {u}
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```



u concerning line 5

A

S

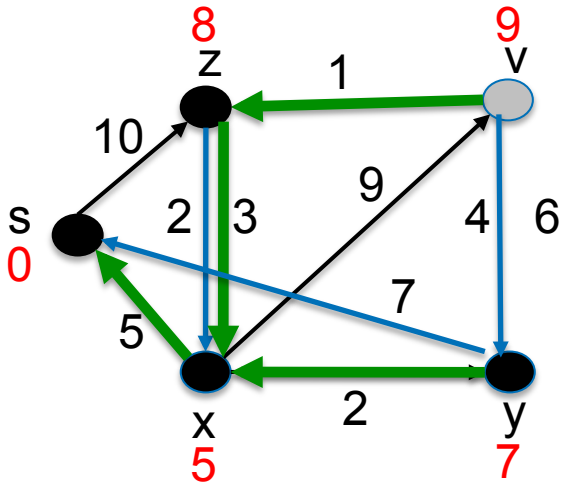
π[v]=u

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm

```
 1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
 2: S := ∅;
 3: A := V;
 4: while   A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A:= A \ {u}
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```



u concerning line 5

A

S

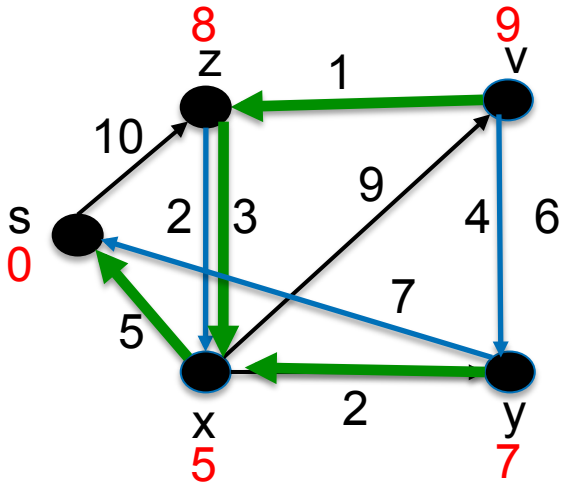
π[v]=u

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm

```
 1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
 2: S := ∅;
 3: A := V;
 4: while   A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A:= A \ {u}
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```



u concerning line 5

A

S

π[v]=u

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm

```
 1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
 2: S := ∅;
 3: A := V;
 4: while   A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A := A \ {u};
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```

Runtime: $O(|E| \cdot O(\text{Zeile 9}) + |V| \cdot O(\text{Zeile 5}))$
With simple idea: $O(|E| + |V| \cdot |V|) = O(|V|^2)$

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm, Correctness

Dijkstra's Algorithm

```
 1: Initialize(G,s)
 2: S := ∅;
 3: A := V;
 4: while  A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A:= A \ {u}
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```

- Lemma Dijk1: dist[v] ≥ δ(s,v)
  Prf. Using induction over the number of executions of lines 8 a. 9.
  - ISta: At beginning: dist[s] = 0 and dist[u] = ∞ for all other u.
  - IH: Claim is valid up to k-th execution of lines 8 and 9.
  - ISte: Let v be the first node for that the claim is wrong, generated by the (k+1)-st execution of lines 8 .a. 9.
    Then, after the (k+1)-st execution:
    dist[u]+f(u,v) = dist[v] < δ(s,v) [concerning assumption]. And it is
    δ(s,v) ≤ δ(s,u) + f(u,v) [shortest path property]
    Thus, in total dist[u]+f(u,v) < δ(s,u) + f(u,v). But, because dist[u] in (k+1)-st execution has not been changed at all, already before was dist[u]< δ(s,u).
    → Assumption dist[v] < δ(s,v) for v was wrong.

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra‘s Algorithm, Correctness

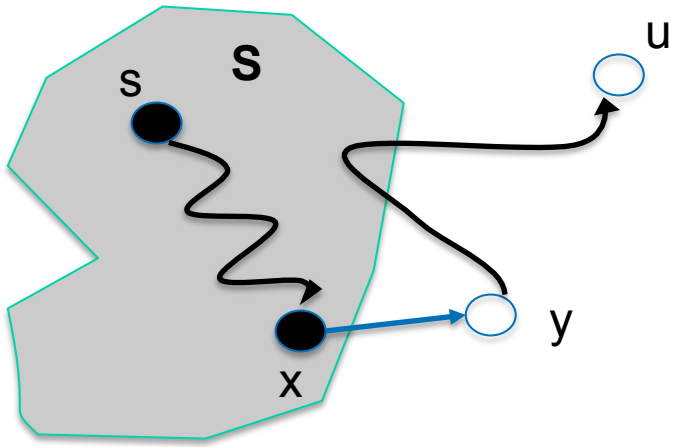
Claim DijkKor: Running Dijksta‘s Algorithm on a weighted directed graph with non-negativ weight function w and startnode s results in $u \in S$: dist[u] = δ(s,u) for all nodes, when the algorithm has finished its work.

Proof via contradiction: Let u be the first node for that it is dist[u] ≠ δ(s,u), when it is added to S. [We will be successful, if we can show that for this first u it is nonetheless true that dist[u] = δ(s,u).]

Now:
- u≠s, then dist[s] is correctly set and never again changed.
- because of u≠s, directly before adding u it is S≠∅
- additionally, there must exist a path from s to u (as otherwise: dist[u] = δ(s,u)=∞)
    - there is a shortest path p from s to u.

Dijkstra‘s Algorithm, Correctness

• p connects node s in S with another node u in A (=V \ S). Let y be the first node along p such that $y \in V \setminus S$ and let x be the predecessor of y. p can be devided into in p_1 and p_2, such that p_1 is completely contained in S.



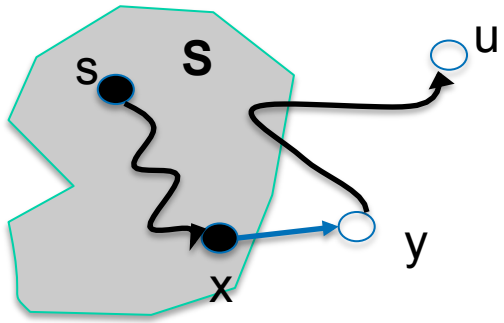
We concentrate our attention on the moment when u is added to S!
For y as well as for u it is valid: $y,u \in A$

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm, Correctness

- Lemma Dijk2: dist[y] ≤ δ(s,y)
  Proof:.
  x∈S and y is in the shortest path in front of u. This implies that y was set to its final value with the help of lines 8 a. 9, before (i.e. when x went to S) this happened to u.



Dijkstra's Algorithm

```
 1: Initialize(G,s)
 2: S := ∅;
 3: A := V;
 4: while  A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A:= A \ {u}
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```

Therefore: dist[y] ≤ dist[x] + f(x,y) [because of lines 8 a. 9]
= δ(s,x) + f(x,y) [because x∈S and assumption in DijkKor]
= δ(s,y), because s → x → y is shortest path. ✔

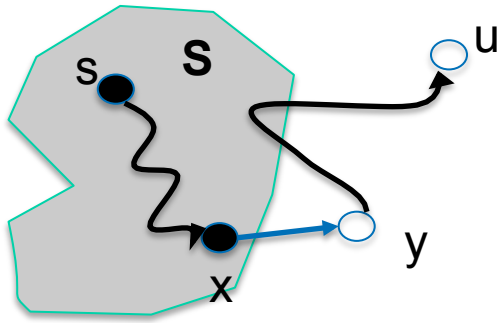
Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra‘s Algorithm, Correctness

- Lemma Dijk3: With lemmata Dijk1 and Dijk2 we know for the given situation:

  In the moment, when u goes into S, it is valid that dist[y] = δ(s,y).



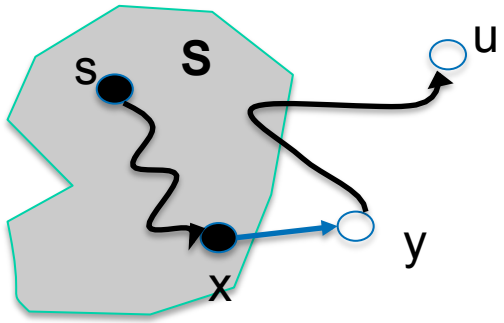
Dijkstra‘s Algorithm

```
 1: Initialize(G,s)
 2: S := ∅;
 3: A := V;
 4: while   A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A:= A \ {u}
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Dijkstra's Algorithm, Correctness



Dijkstra's Algorithm

```
 1: Initialize(G,s)
 2: S := ∅;
 3: A := V;
 4: while  A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }; A:= A \ {u}
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```

- We conclude:
  dist[y] = δ(s,y) ≤ δ(s,u) ≤ dist[u] and dist[u] ≤ dist[y] because of line 5 and y,u∈A

  shortest path Lemma Dijk1

  Thus, alltogether: dist[u] ≤ dist[y] ≤ δ(s,u) ≤ dist[u],
  also dist[u] = δ(s,u) → Contradiction to assupmtion the claim DijkKor

Shortest Path Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Improvement for Dijkstra's Algorithmu

```
 1: Initialize(G,s)  // für alle Knoten v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
 2: S := ∅;
 3: A := V;
 4: while   A ≠ ∅  do
 5:    u := argmin{ dist[a] | a∈A }
 6:    S := S ∪ {u};
 7:    for each node v ∈ Adj[u] do
 8:       if dist[v] > dist[u] + f(u,v) then
 9:          dist[v] := dist[u] + f(u,v);
10:          π[v] := u;
```

Runtime: $O(|E| \cdot O(\text{Zeile 9}) + |V| \cdot O(\text{Zeile 5})) = O(|V|^2)$
With the help of heaps (new abstract data type similar to queue or stack), it can be achieved : $O(|E| \cdot \log(|V|) + |V| \cdot \log(|V|))$ and even $O(|E| + |V| \cdot \log(|V|)$