Karlsruher Institut für Technologie

9. Übung – Algorithmen I

INSTITUT FÜR THEORETISCHE INFORMATIK

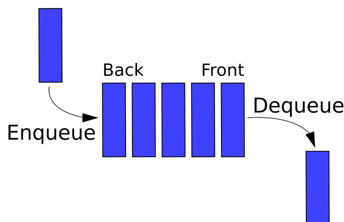
Musterlösung der Probeklausur

... online ab Dienstag, 17.6.

Breiten- und Tiefensuche in Graphen

Wiederholung: Breitensuche

```
Procedure bfs(G=(V, E) : Graph, s: node)
    Q=[ ] : Queue
    Q.enqueue(s)
    label s as visited
    while Q is not empty do
        v = Q.dequeue()
        process(v)
        foreach w : ∃{v, w} ∈ E do
            if w not visited then
                Q.enqueue(w)
                label w as visited
```



Wiederholung: Tiefensuche rekursiv

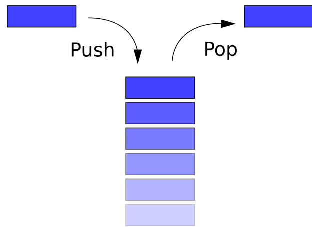
Karlsruher Institut für Technologie

```
Procedure dfs(G=(V, E) : Graph, v: node)
    label v as visited
    process(v)
    foreach w : ∃{v, w} ∈ E do
        if w not visited then
            dfs(G,w)
```

Tiefensuche iterativ

KIT
Karlsruher Institut für Technologie

```
Procedure dfs(G=(V, E) : Graph, s: node)
    S=[ ] : Stack
    S.push(s)
    label s as visited
    while S is not empty do
        v = S.pop()
        if v is not visited then
            label v as visited
            process(v)
            foreach w : ∃{v, w} ∈ E do
                S.push(w)
```



DFS iterativ oder rekursiv: Was ist besser?

Karlsruher Institut für Technologie

DFS iterativ oder rekursiv: Was ist besser?

Karlsruher Institut für Technologie

Performancefrage nicht pauschal zu beantworten - hängt von komplizierten Einflüssen ab, z.B.

- Compileroptimierungen
- Cache-Effekten
- Tiefe der Suche
- ...

DFS iterativ oder rekursiv: Was ist besser?

Karlsruher Institut für Technologie

Performancefrage nicht pauschal zu beantworten - hängt von komplizierten Einflüssen ab, z.B.

- Compileroptimierungen
- Cache-Effekten
- Tiefe der Suche
- ...

Also: Profiling und Experimente mit echtem Code ($\rightarrow$ *algorithm engineering*)

Breitensuche (BFS) animiert

`https://www.cs.usfca.edu/~galles/visualization/BFS.html`

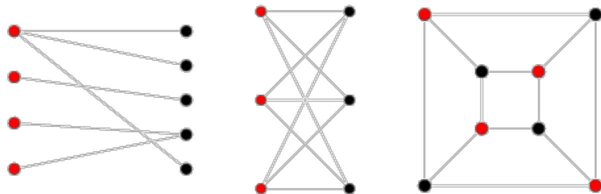
Tiefensuche (DFS) animiert

`http://www.cs.usfca.edu/~galles/visualization/DFS.html`

Beispielanwendung Breitensuche

Karlsruher Institut für Technologie

Problem: Ist ein (zusammenhängender) Graph bipartit?



Beispielanwendung Breitensuche

Karlsruher Institut für Technologie

Lösung in $O(m)$: Knoten schwarz oder rot einfärben während der Breitensuche.

```
Procedure isBipartite(G=(V, E) : Graph, s: node)
    Q=[ ] : Queue
    set color to red
    Q.enqueue(s); label s as visited
    color s
    while Q is not empty do
        v = Q.dequeue()
        switch color
        foreach w : ∃{v, w} ∈ E do
            if w not visited then
                Q.enqueue(w); label w as visited
                color w
            else
                if w has same color as v return false
    return true
```

Karlsruher Institut für Technologie

Breitensuche in DAGs

Wiederholung: DAG

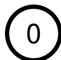
Directed Acyclic Graph

Karlsruher Institut für Technologie

0

Wiederholung: DAG

Directed Acyclic Graph



JA (wenn gerichtet)
Ein Knoten ist Wurzel und Blatt

Wiederholung: DAG

Directed Acyclic Graph

Wiederholung: DAG

Directed Acyclic Graph

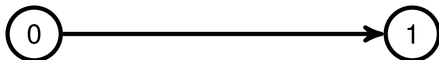
NEIN
Schleife ist auch ein Kreis!

Wiederholung: DAG

Directed Acyclic Graph

Wiederholung: DAG

Directed Acyclic Graph

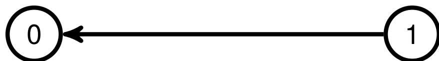


JA

Eine Wurzel, ein Blatt

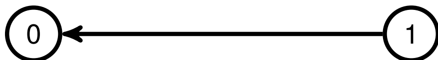
Wiederholung: DAG

Directed Acyclic Graph



Wiederholung: DAG

Directed Acyclic Graph

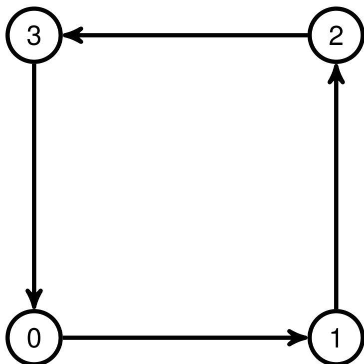


JA

Benennung der Knoten ist irrelevant

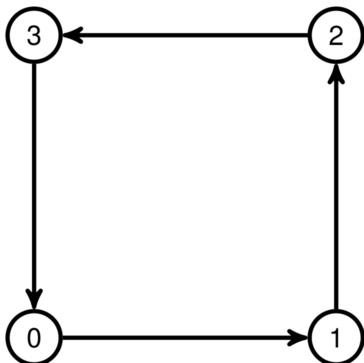
Wiederholung: DAG

Directed Acyclic Graph



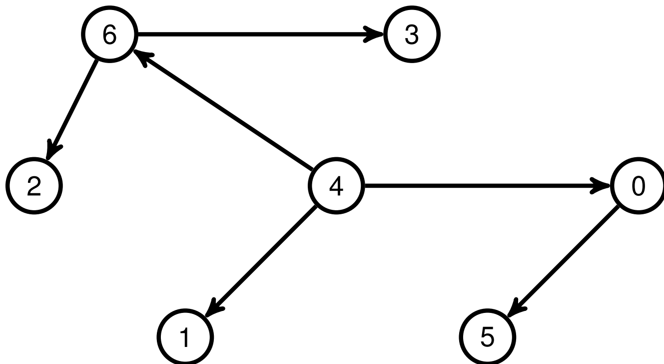
Wiederholung: DAG

Directed Acyclic Graph



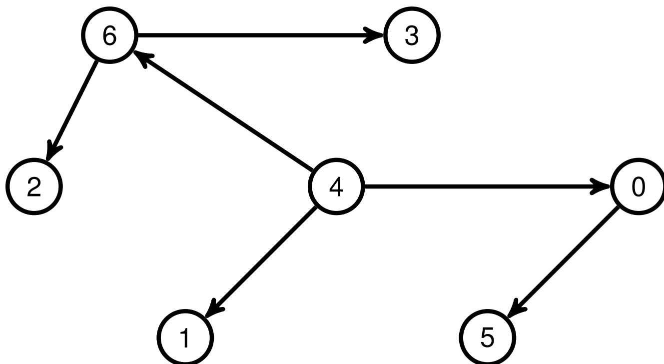
Wiederholung: DAG

Directed Acyclic Graph



Wiederholung: DAG

Directed Acyclic Graph

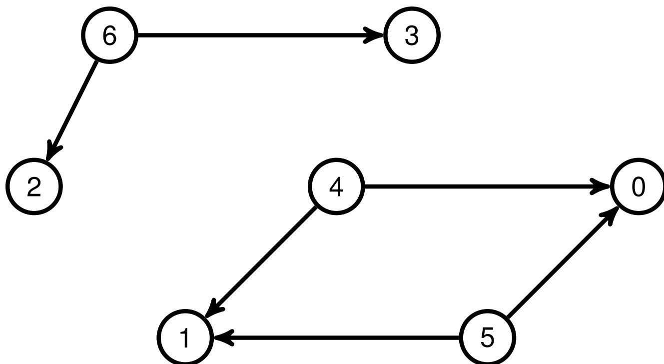


JA

Sogar ein gewurzelter Baum: ein Knoten Eingangsgrad 0, alle anderen Eingangsgrad 1

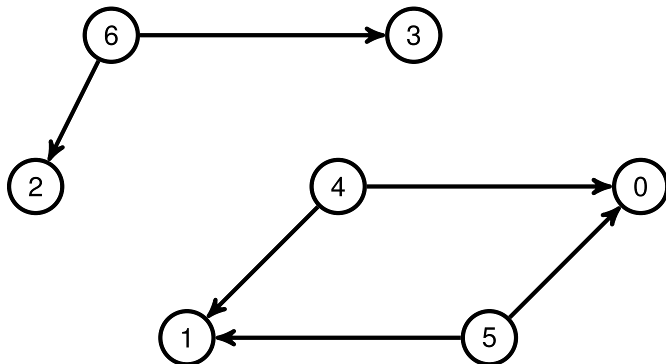
Wiederholung: DAG

Directed Acyclic Graph



Wiederholung: DAG

Directed Acyclic Graph



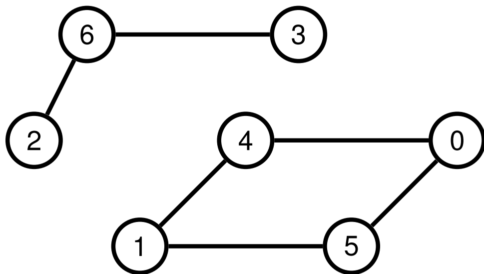
JA

Zwei Zusammenhangskomponenten, drei Wurzeln
Jeder DAG hat mind. eine Wurzel!

Breitensuche..

für nicht zusammenhängende, ungerichtete Graphen

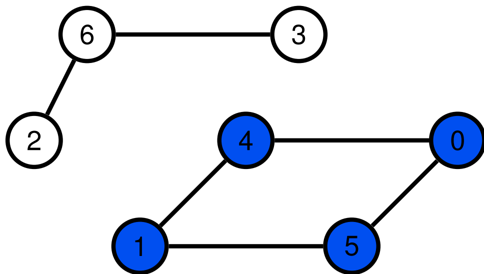
Procedure doBFS($G = (V, E)$)
  foreach $v \in V$ **do**
    if v is not marked **then**
      bfs(v)



Breitensuche..

für nicht zusammenhängende, ungerichtete Graphen

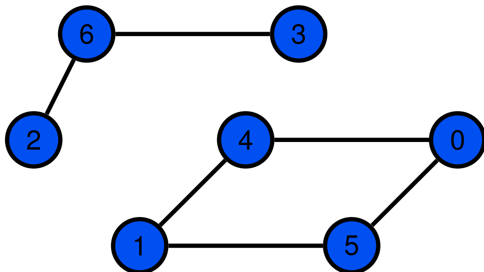
Procedure doBFS($G = (V, E)$)
  foreach $v \in V$ **do**
    if v is not marked **then**
      bfs(v)



Breitensuche..

für nicht zusammenhängende, ungerichtete Graphen

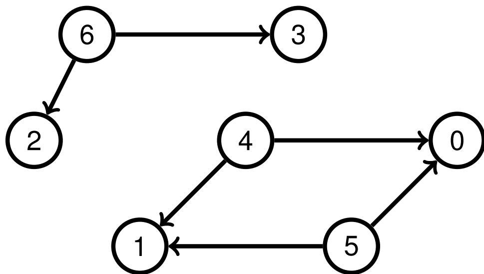
Procedure doBFS($G = (V, E)$)
  foreach $v \in V$ **do**
    if v is not marked **then**
      bfs(v)



Breitensuche..

für nicht zusammenhängende, gerichtete Graphen

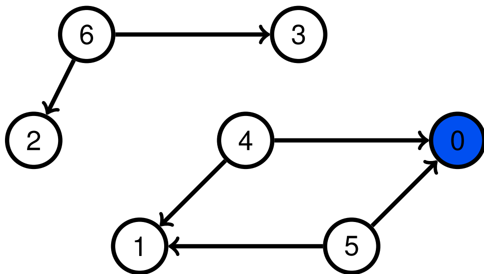
Procedure doBFS($G = (V, E)$)
  foreach $v \in V$ **do**
    if v is not marked **then**
      bfs(v)



Breitensuche..

für nicht zusammenhängende, gerichtete Graphen

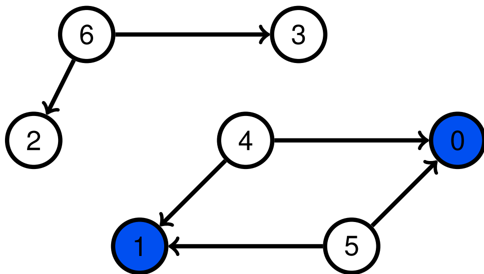
Procedure doBFS($G = (V, E)$)
  foreach $v \in V$ **do**
    if v is not marked **then**
      bfs(v)



Breitensuche..

für nicht zusammenhängende, gerichtete Graphen

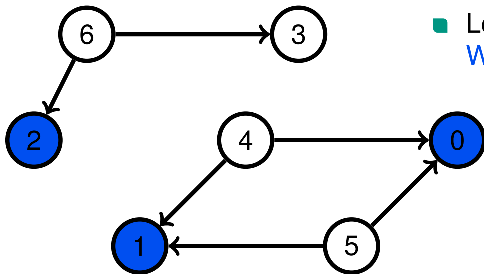
Procedure doBFS($G = (V, E)$)
  foreach $v \in V$ **do**
    if v is not marked **then**
      bfs(v)



Breitensuche..

für nicht zusammenhängende, gerichtete Graphen

Procedure doBFS($G = (V, E)$)
  foreach $v \in V$ **do**
    if v is not marked **then**
      bfs(v)



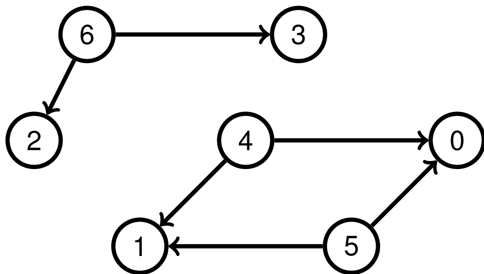
- Reihenfolge der Knoten spielt Rolle
- Zwar werden alle Knoten besucht
- u.U. nur wenig Infos über Struktur des DAG
- Lösung: finde zunächst alle Wurzeln

Breitensuche..

für nicht zusammenhängende, gerichtete Graphen

Function doBFS$(G = (V, E))$
  isRoot=True : **Array of** $\{0, 1\}$
  foreach $(u, v) \in E$ **do**
    $b[v] := \mathrm{False}$

- Finde zunächst alle Wurzeln
  - (jeder DAG hat mind. eine!)

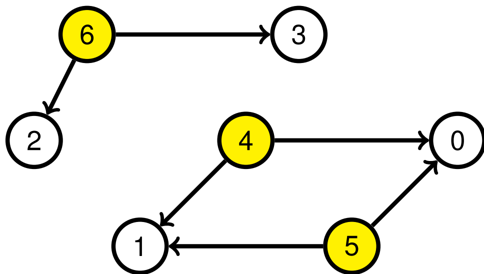


Breitensuche..

für nicht zusammenhängende, gerichtete Graphen

Function doBFS($G = (V, E)$)
    isRoot=True : **Array of** $\{0, 1\}$
    foreach $(u, v) \in E$ **do**
        $b[v] := \mathrm{False}$

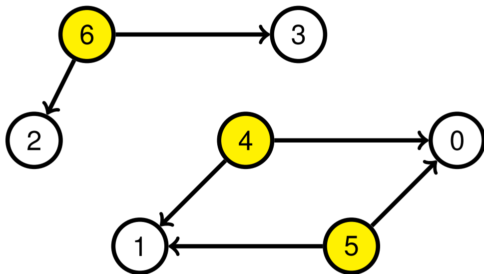
- Finde zunächst alle Wurzeln
  - (jeder DAG hat mind. eine!)



Breitensuche..

für nicht zusammenhängende, gerichtete Graphen

Function doBFS($G = (V, E)$)
  isRoot=True : **Array of** $\{0, 1\}$
  foreach $(u, v) \in E$ **do**
    $b[v] := \mathrm{False}$
  foreach $v \in V$, $b[v] = \mathrm{True}$ **do**
    bfs(v)

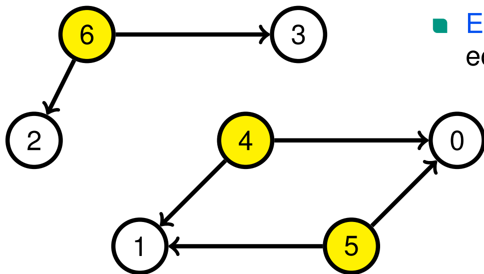


- Finde zunächst alle Wurzeln
  - (jeder DAG hat mind. eine!)
- Dann: Entweder starte BFS von jeder Wurzel einzeln

Breitensuche..

für nicht zusammenhängende, gerichtete Graphen

Function doBFS$(G = (V, E))$
  isRoot$=\mathrm{True}$: **Array of** $\{0, 1\}$
  foreach $(u, v) \in E$ **do**
    $b[v] := \mathrm{False}$
  $Q = \langle v \in V, b[v] = \mathrm{True} \rangle$:

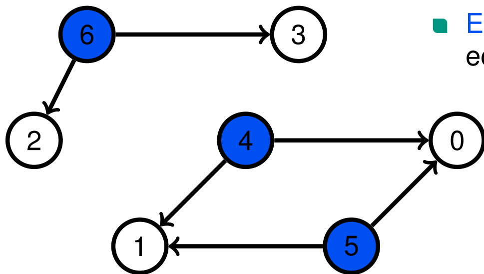


- Finde zunächst alle Wurzeln
  - (jeder DAG hat mind. eine!)
- Oder: Setze erstes Level := Menge d. Wurzeln
- Eine imaginäre Wurzel zu allen echten Wurzeln

Breitensuche..

für nicht zusammenhängende, gerichtete Graphen

Function doBFS($G = (V, E)$)
  isRoot=True : **Array of** $\{0, 1\}$
  foreach $(u, v) \in E$ **do**
    $b[v] :=$ False
  $Q = \langle v \in V, b[v] = \text{True} \rangle$:

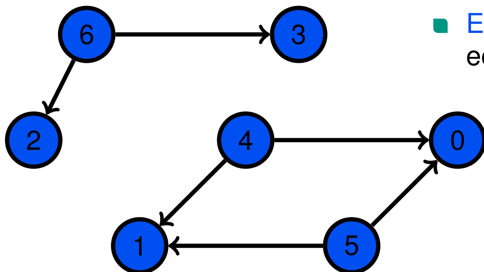


- Finde zunächst alle Wurzeln
  - (jeder DAG hat mind. eine!)
- Oder: Setze erstes Level := Menge d. Wurzeln
- Eine imaginäre Wurzel zu allen echten Wurzeln

Breitensuche..

für nicht zusammenhängende, gerichtete Graphen

Function doBFS($G = (V, E)$)
  isRoot=True : **Array of** $\{0, 1\}$
  foreach $(u, v) \in E$ **do**
    $b[v] := \mathrm{False}$
  $Q = \langle v \in V, b[v] = \mathrm{True} \rangle$:



- Finde zunächst alle Wurzeln
  - (jeder DAG hat mind. eine!)
- Oder: Setze erstes Level := Menge d. Wurzeln
- Eine imaginäre Wurzel zu allen echten Wurzeln