Data structure [A14]

김종규, PhD

2017-06-05

Review

- `input.txt`
  - 0 at 875, 955, 1001 $\longrightarrow$ 875 에서 멈출 것

Outline

- Revisit Tree and graph
- Visiting graph with circle
  - Predecessor
  - BFS, DFS

Binary tree 의 print (PL-Python)


```
class Node:
    def __init__(self,val):
        self.val = val
        self.left = None
        self.right = None
def bst_print(tree, level):
    if (tree.right):
        bst_print(tree.right, level + 1)
    for i in range(level):
        print('    ', end = '')
    print(tree.val)
    if (tree.left):
        bst_print(tree.left, level + 1)
```

Example

```
def main():
    root = Node(2)
    a = Node(1)
    b = Node(3)
    root.left = a
    root.right = b
    print()
    bst_print(root,0)

    3
2
    1
```

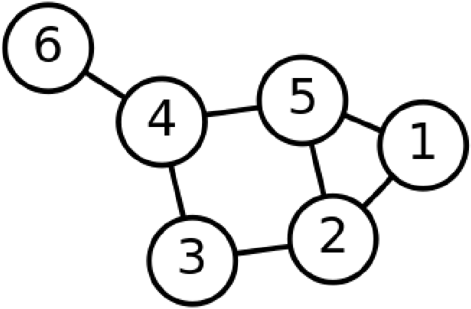
Example

```
def main():
    root = Node(2)
    a = Node(1)
    b = Node(3)
    root.left = a
    root.right = b
    a.left = root  # cycle
    print()
    bst_print(root,0)
```

```
    3
2
    1
            3
        2
            1
                    3
```

그래프 (Graph)

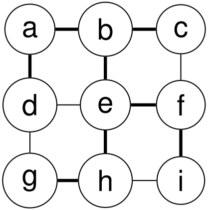
- $G = (V, E)$ where V vertex, E edge
- Subgraph $\hat{G} = (\hat{V}, \hat{E})$ where $\hat{G}$ is a graph, $\hat{V} \subseteq V$ and $\hat{E} \subseteq E$



Predecessor subgraph

- $G = (V, E)$, and $s \in V$
- $G_p \subseteq G$ where $V_p \subseteq V$ and $E_p \subseteq E$
- Construct G_p and $\pi : V \to V \cup \{\text{NIL}\}$
  - $V_p = \{s\}$, $E_p = \phi$, and $\pi(v) = NIL, \forall v \in V$
  - Pick vertex $u \in V_p$
  - Pick another vertext $v \in V$ where $v \notin V_p$ and $(u, v) \in E$
  - Set $\pi(v) = u$ and $V_p = V_p \cup \{v\}$
  - Add $(u, v) = (\pi(v), v)$ to E_p

Example



- a $\longrightarrow$ NIL
- b $\longrightarrow$ a
- c $\longrightarrow$ b
- d $\longrightarrow$ a
- e $\longrightarrow$ b
- f $\longrightarrow$ e
- g $\longrightarrow$ h
- h $\longrightarrow$ e
- i $\longrightarrow$ f

Predecessor subgraph

- Predessor subgraph is a tree
  - Every vertex is connected
  - We chose v when there is an edge (u, v)
  - There is no cycle
  - Suppose there are edges $(u, v), (v, u)$
  - Case 1: $v \in V_p$ We choose u and add (u, v) to E_p
    - We cannot choose (v, u) because $u \in V_p$ already
  - Case 2: $u \in V_p \cdots$

Building a precessor subgraph

- Visit: Add a vertex v to V_p
- BFS (Breadth first search)
  - Choose s randomly
  - Visit vertices which are connected to s
  - Do the same with the rest of the vertices
- DFS (Depth first search)
  - Choose s randomly
  - Visit a vertex v where $(s, v) \in E$
  - Do the same with v

Breadth first search

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

그림: Breadth first search

Depth first visit


```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

그림: Depth first visit

Depth first search

DFS(G)

for each vertex $u \in G.V$
    $u.color = \text{WHITE}$
    $u.\pi = \text{NIL}$
$time = 0$
for each vertex $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT(G, u)

그림: Depth first search

Breadth first search

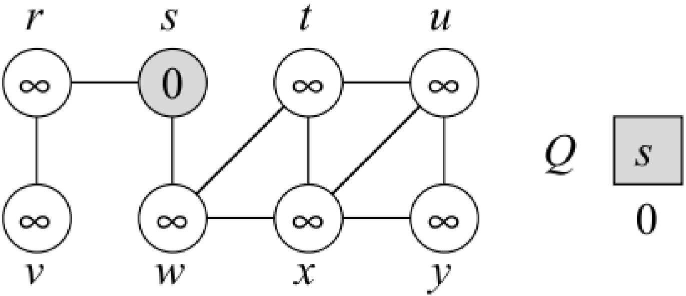


그림: Breadth first search (1/9)

Breadth first search

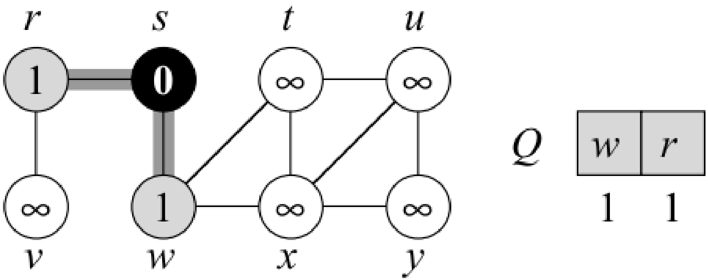


그림: Breadth first search (2/9)

Breadth first search

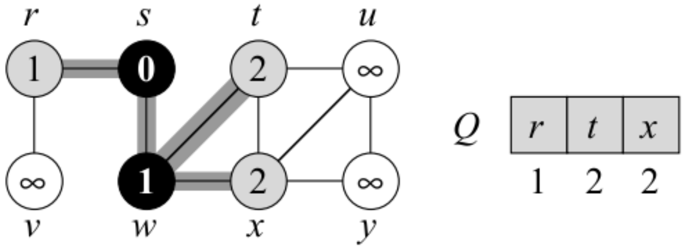


그림: Breadth first search (3/9)

Breadth first search

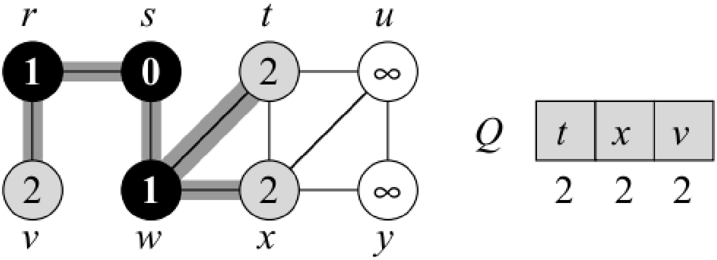


그림: Breadth first search (4/9)

Breadth first search

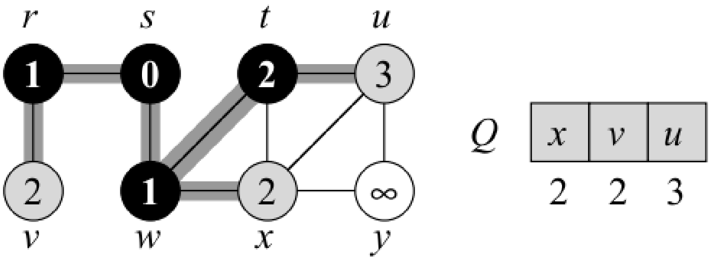


그림: Breadth first search (5/9)

Breadth first search

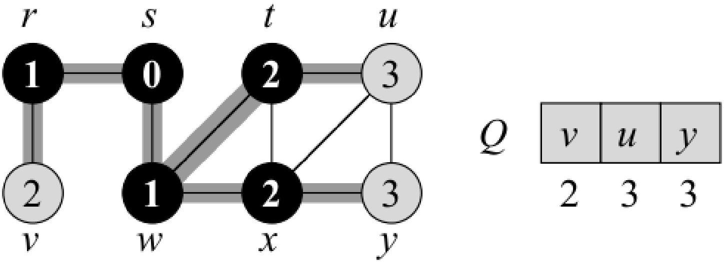


그림: Breadth first search (6/9)

Breadth first search

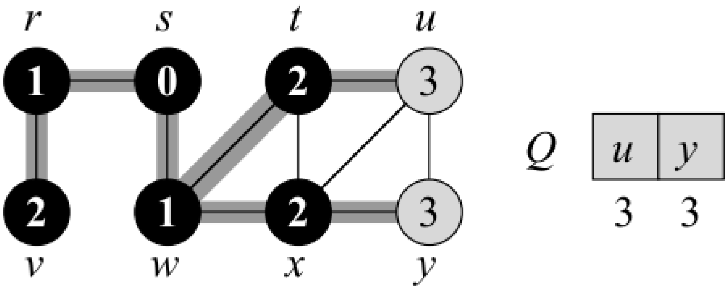


그림: Breadth first search (7/9)

Breadth first search

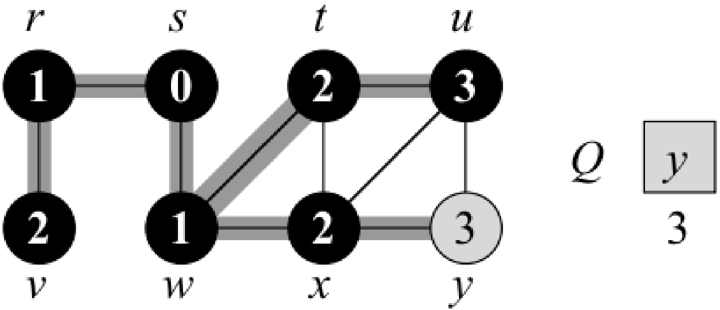


그림: Breadth first search (8/9)

Breadth first search

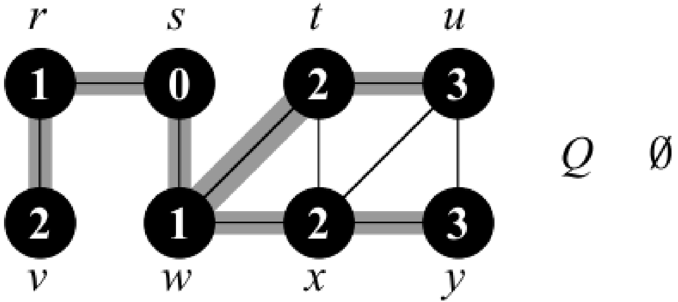


그림: Breadth first search (9/9)

Depth first search

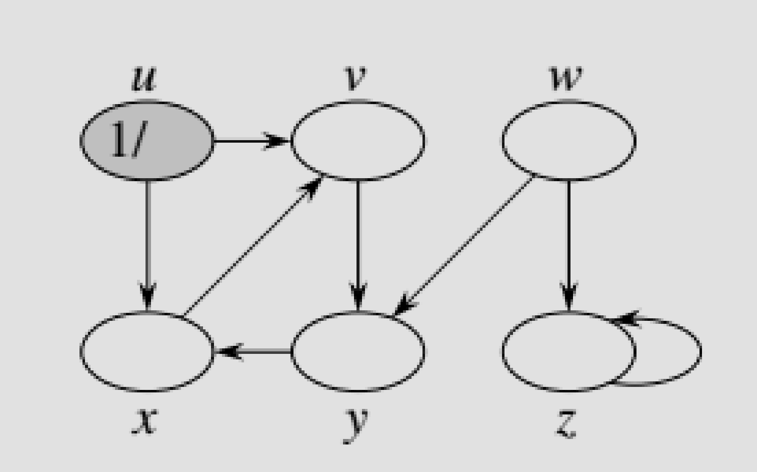


그림: Depth first search (1/16)

Depth first search

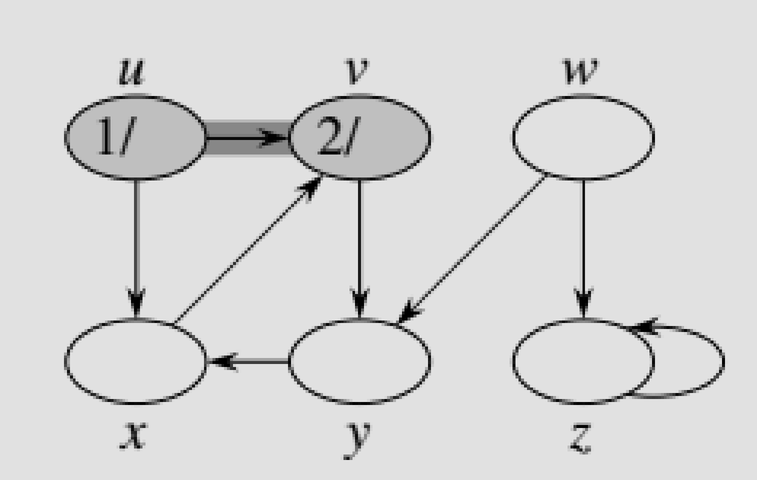


그림: Depth first search (2/16)

Depth first search

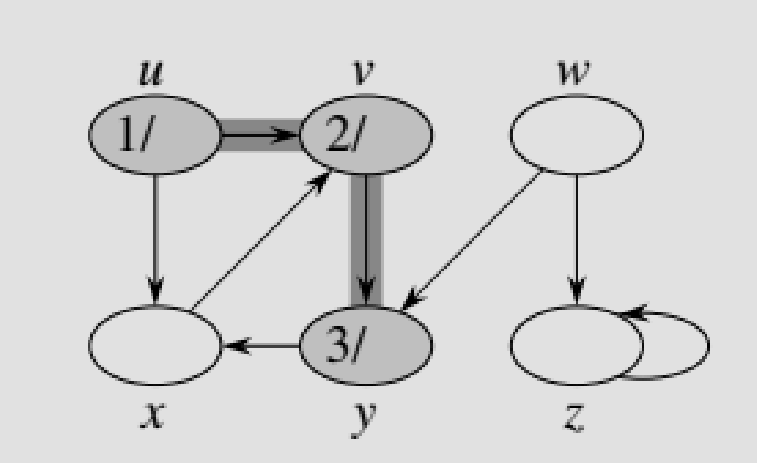


그림: Depth first search (3/16)

Depth first search

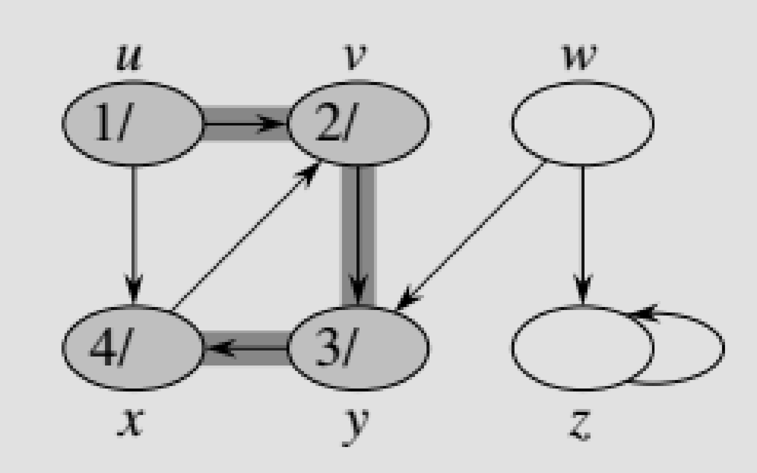


그림: Depth first search (4/16)

Depth first search

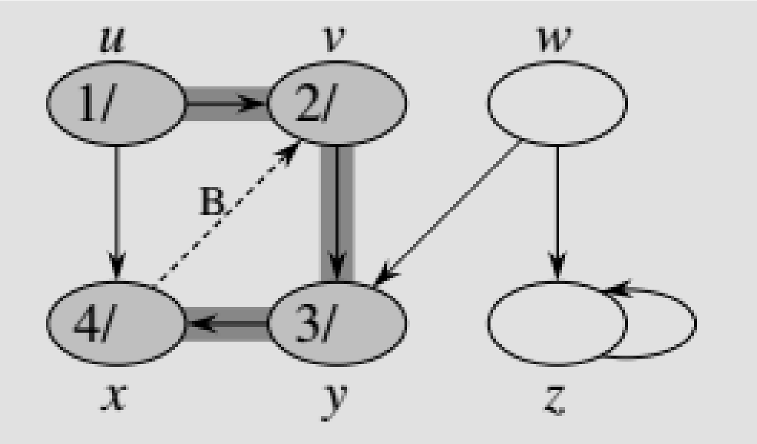


그림: Depth first search (5/16)

Depth first search

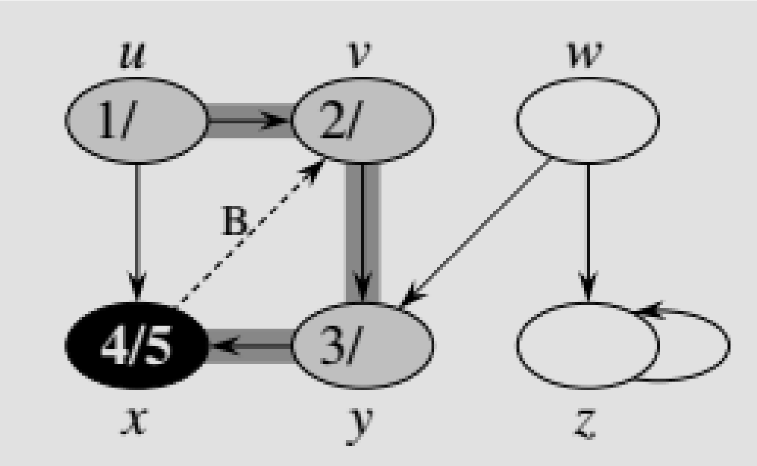


그림: Depth first search (6/16)

Depth first search

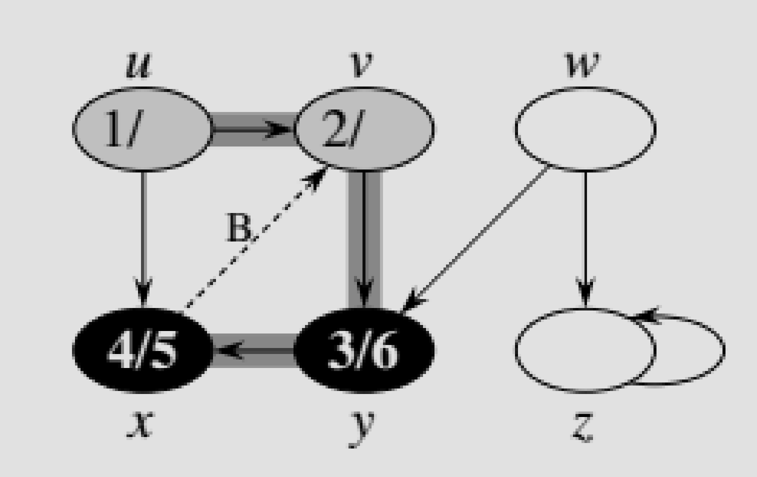


그림: Depth first search (7/16)

Depth first search

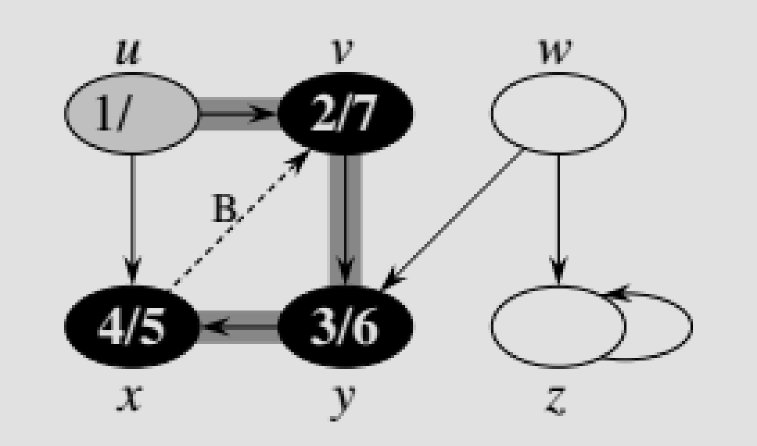


그림: Depth first search (8/16)

Depth first search

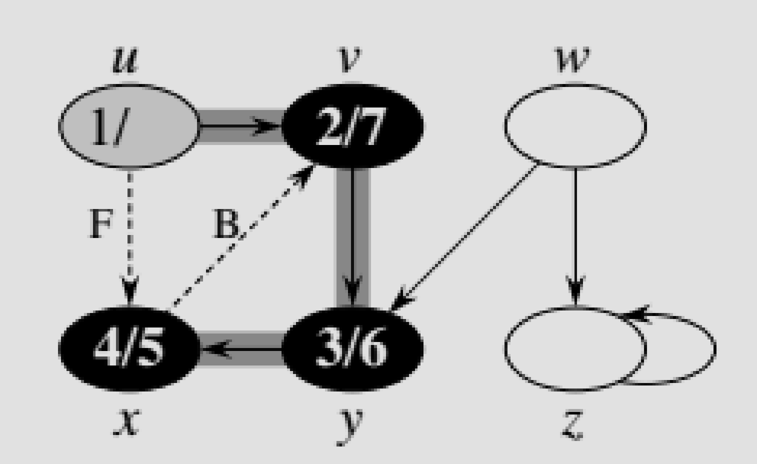


그림: Depth first search (9/16)

Depth first search

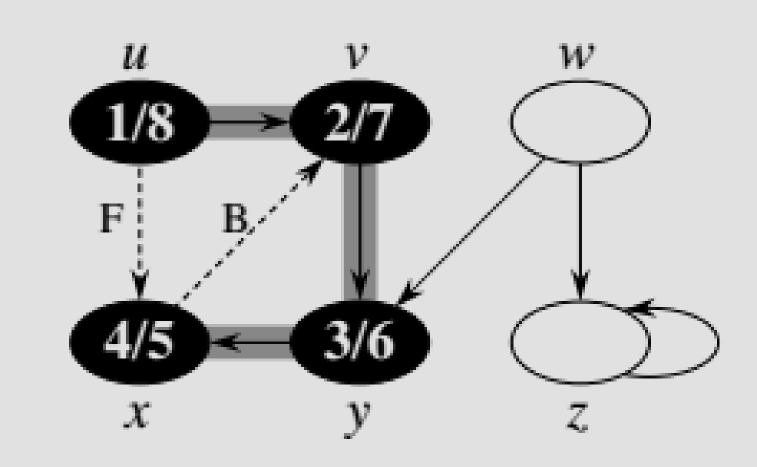


그림: Depth first search (10/16)

Depth first search

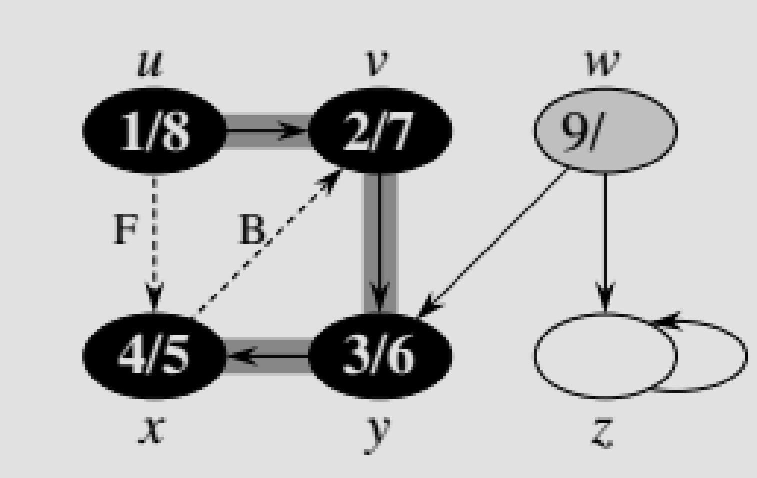


그림: Depth first search (11/16)

Depth first search

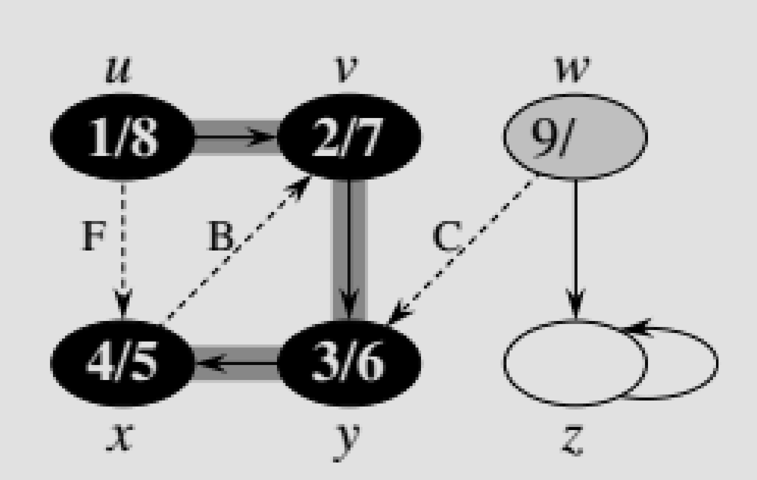


그림: Depth first search (12/16)

Depth first search

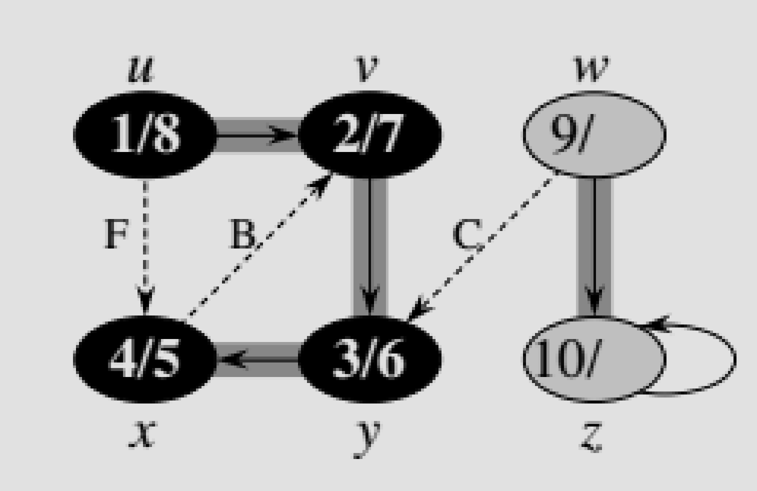


그림: Depth first search (13/16)

Depth first search

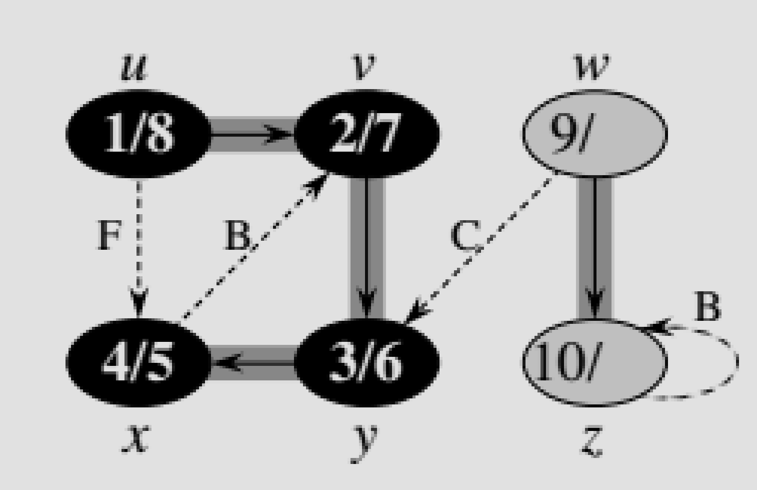


그림: Depth first search (14/16)

Depth first search

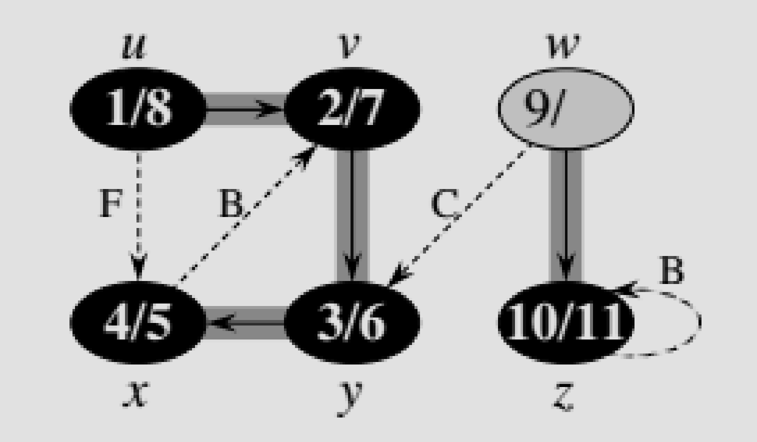


그림: Depth first search (15/16)

Depth first search

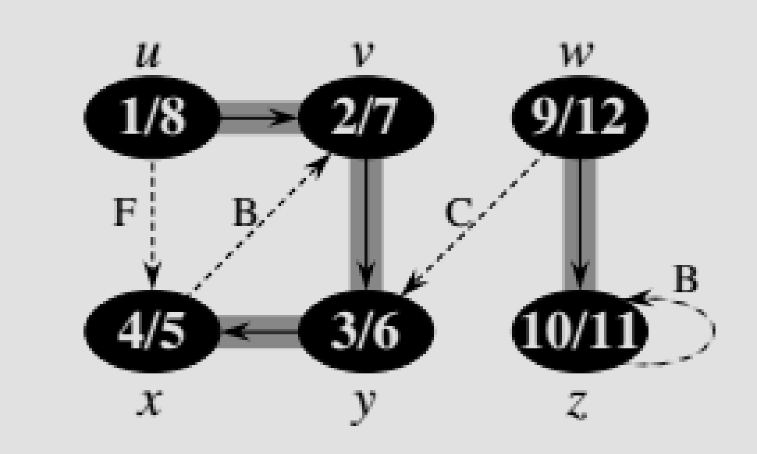


그림: Depth first search (16/16)

Comparison

- BFS
  - Uses Queue to complete the algorithm
  - Update the attribute $u.d$
  - $u.d$ is the distance from the vertex s
  - Following precessor gives the shortest path
- DFS
  - Use stack to complete the algorithm
  - Update the attributes $u.d$ and $u.f$
  - $u.d$ and $u.f$ gives starting and finishing time of a task
  - if $v.d > u.d$ and $v.f < u.f$ then v is a descendent of u in predecessor subgraph

Wrap-up

- Tree is a connected and acyclic graph
- To understand the cyclic nature of a graph, we learned predecessor subgraph concept
- Among many predessor subgraphs, BFS and DFS are especially useful
- We learned the relation between shortest path and BFS
- We learned the relation between tree structure and DFS
- We will learn more sophisticated algorithms regarding graph