Graph Traversals

Alessandro SAVINO
Politecnico di Torino (Italy)

alessandro.savino@polito.it

www.testgroup.polito.it

License Information

Disclaimer

- **We disclaim any warranties or representations as to the accuracy or completeness of this material.**
- **Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.**
- **Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.**

Goal

– **This lecture aims at presenting graphs visiting techniques.**

Prerequisites

- **Lectures:**
  - **11_7.x Pointers & Dynamic Memory**
  - **11_8.4 Graphs: Introduction & definitions**

Further readings

- **Students interested in a deeper look at the covered topics can refer, for instance, to the books listed at the end of the lecture.**

Outline

- **Introduction**
- **Breadth-First Search**
- **Depth-First Search**

Outline

- Introduction
- Breadth-First Search
- Depth-First Search

Graph Searching
(or Visiting or Traversing)

- **Given: a graph G = (V, E), directed or undirected**
- **Goal: explore every vertex and every edge from a source vertex.**
- **Result: build a tree on the graph**
  - **Pick a vertex as the root**
  - **Choose certain edges to produce a tree**
  - **Note: might also build a *forest* if graph is not connected**

Approaches

- **Two possible visits:**
  - ***Breadth first* visit**
  - ***Depth first* visit.**

Outline

- Introduction
- Breadth-First Search
- Depth-First Search

Breadth-First Search

- **"Explore" a graph, turning it into a tree**
  - **One vertex at a time**
  - **Expand frontier of explored vertices across the *breadth* of the frontier**
- **Builds a tree over the graph**
  - **Pick a *source vertex* to be the root**
  - **Find ("discover") its children, then their children, etc.**

Breadth-First Search

- **We will associate vertex "colors" to guide the algorithm**
  - **White nodes have not been discovered**
    - **All vertices start out white**
  - **Grey nodes are discovered but not fully explored**
    - **They may be adjacent to white vertices**
  - **Black nodes are discovered and fully explored**
    - **They are adjacent only to black and gray vertices**
- **Explore nodes by scanning adjacency list of grey nodes**

Pseudo-code

```
BFS(G, s)
 1   for ogni vertice u ∈ V[G] – {s}
 2         do  color[u] ← WHITE
 3             d[u] ← ∞
 4             π[u] ← NIL
 5   color[s] ← GRAY
 6   d[s] ← 0
 7   π[s] ← NIL
 8   Q ← {s}
 9   while Q ≠ ∅
10         do  u ← head[Q]
11             for ogni v ∈ Adj[u]
12                   do if color[v] = WHITE
13                         then  color[v] ← GRAY
14                               d[v] ← d[u] + 1
15                               π[v] ← u
16                               ENQUEUE(Q, v)
17             DEQUEUE(Q)
18             color[u] ← BLACK
```

Pseudo-code

```
BFS(G, s)
 1   for ogni vertice u ∈ V[G] – {s}
 2       do  color[u] ← WHITE
 3           d[u] ← ∞
 4           π[u] ← NIL
 5   color[s] ← GRAY
 6   d[s] ← 0
 7   π[s] ← NIL
 8   Q ← {s}
 9   while Q ≠ ∅
10       do  u ← head[Q]
11           for ogni v ∈ Adj[u]
12               do if color[v] = WHITE
13                     then  color[v] ← GRAY
14                           d[v] ← d[u] + 1
15                           π[v] ← u
16                           ENQUEUE(Q, v)
17           DEQUEUE(Q)
18           color[u] ← BLACK
```

Level

Parent

FIFO

- **Breadth-first visit is usually implemented using a FIFO:**
  - **Each time a vertex is visited is placed in the FIFO**
  - **At each step we get a vertex from the FIFO and its adjacent nodes are visited.**

BFS tree

- **The BFS produces a BFS tree, where**
  - **The root is the source node**
  - **The vertices are the same ones of the graph**
  - **The edges are a subset of the graph edges**

Breadth-First Search

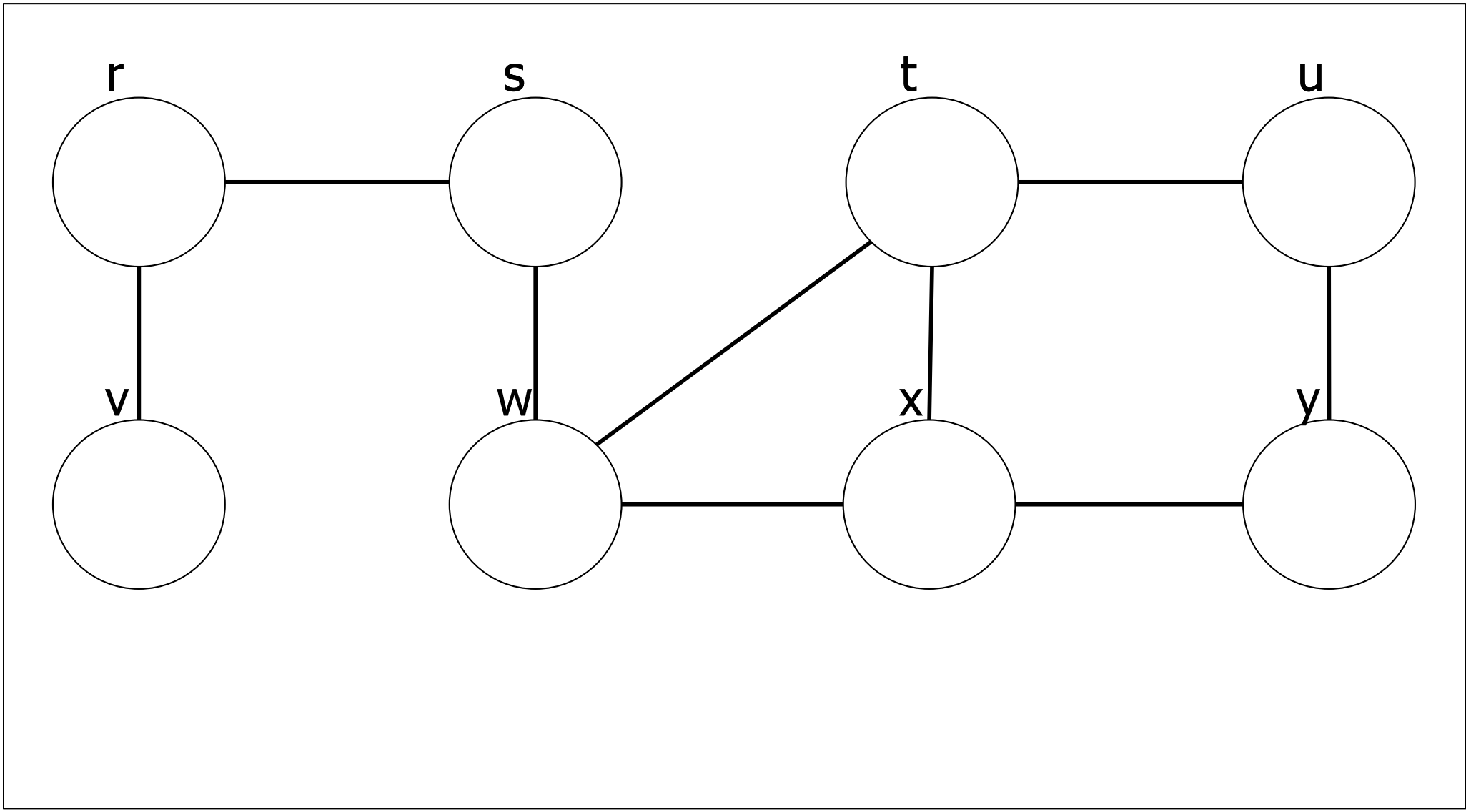
```
BFS(G, s) {
    initialize vertices;
    Q = {s};      // Q is a queue; initialize to s
    while (Q not empty) {
        u = RemoveTop(Q);
        for each v ∈ u->adj {
            if (v->color == WHITE)
                v->color = GREY;
                v->d = u->d + 1;
                v->p = u;
                Enqueue(Q, v);
        }
        u->color = BLACK;
    }
}
```

Note

- **Vertices coloring follows this criteria :**
  - **All vertices are initially white**
  - **A vertex becomes gray when is visited for the first time**
  - **A vertex becomes black when all its adjacent vertices which are not yet visited have been placed in the FIFO.**

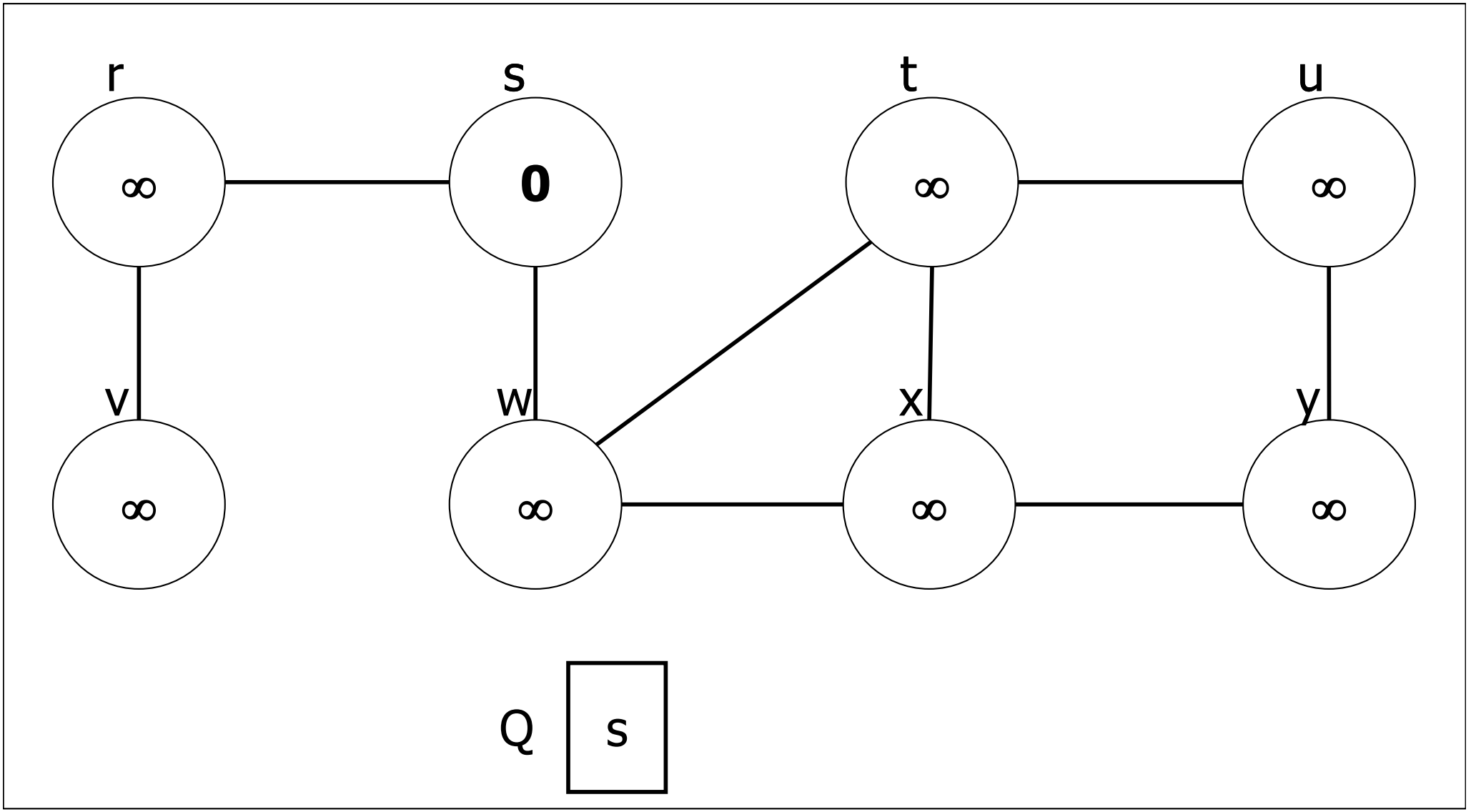
Example

- **source: s**



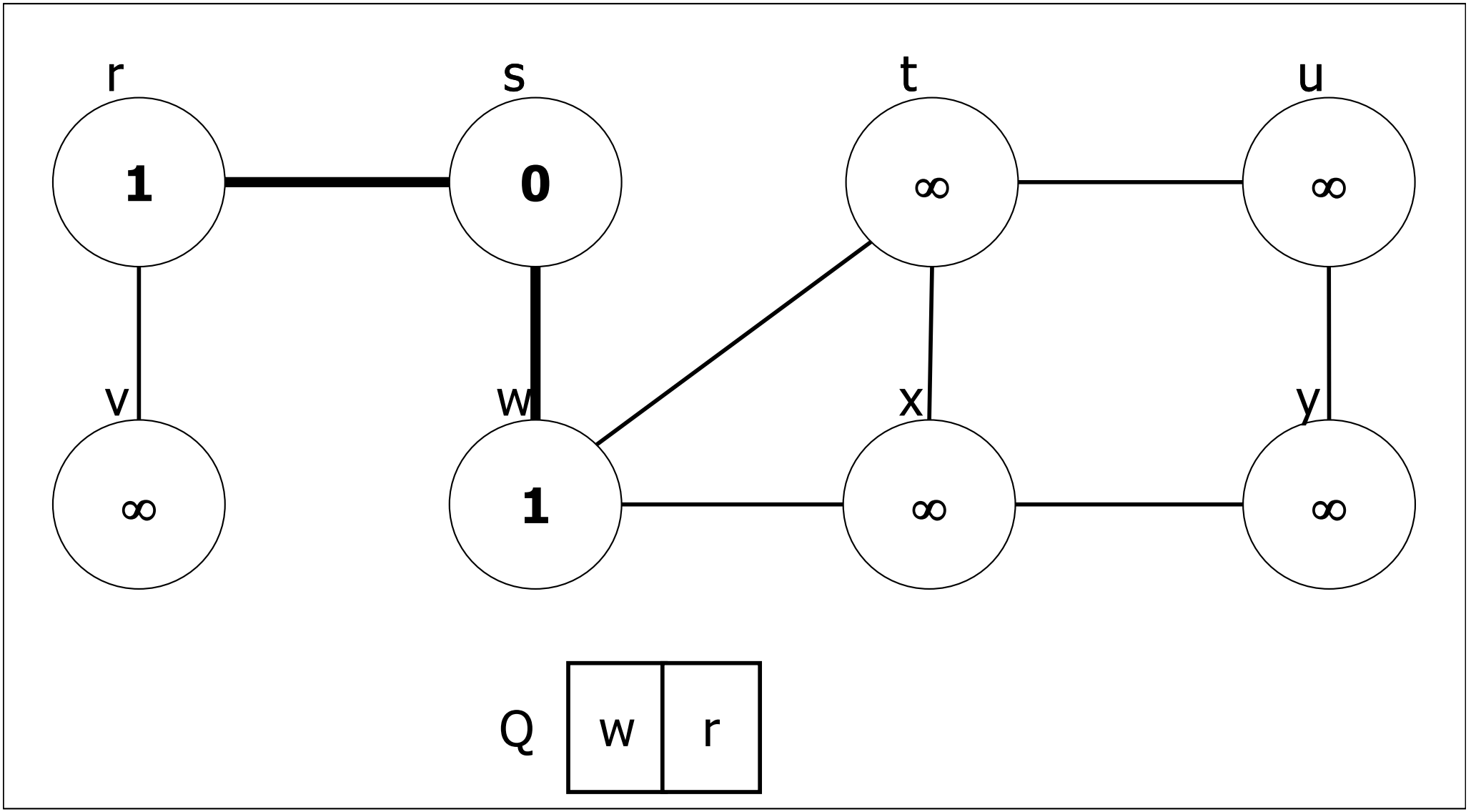
Example

- **Initialization**



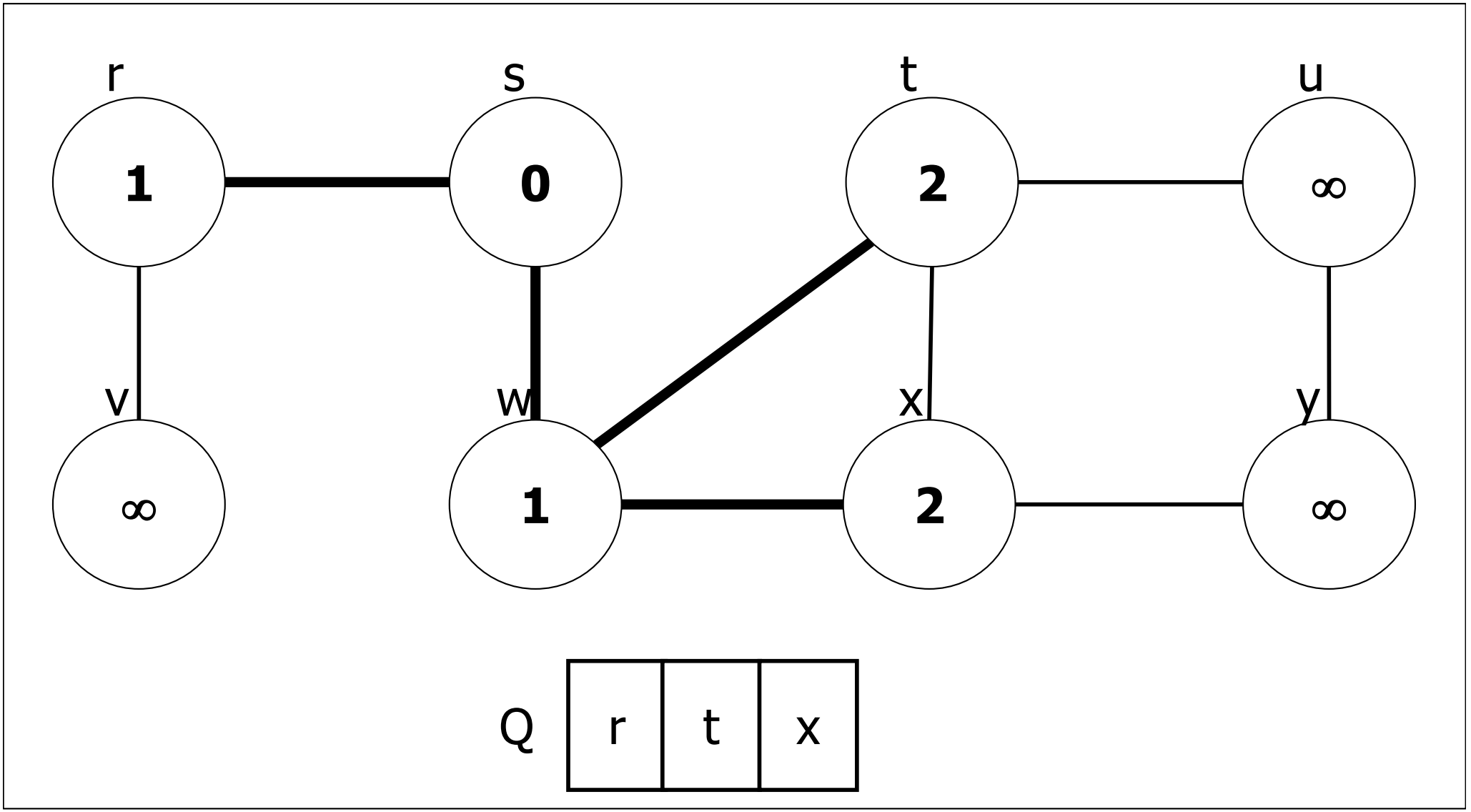
Example

- Extract s from Q
- Enque and gray color all adjacent vertices of s
- Color s in black

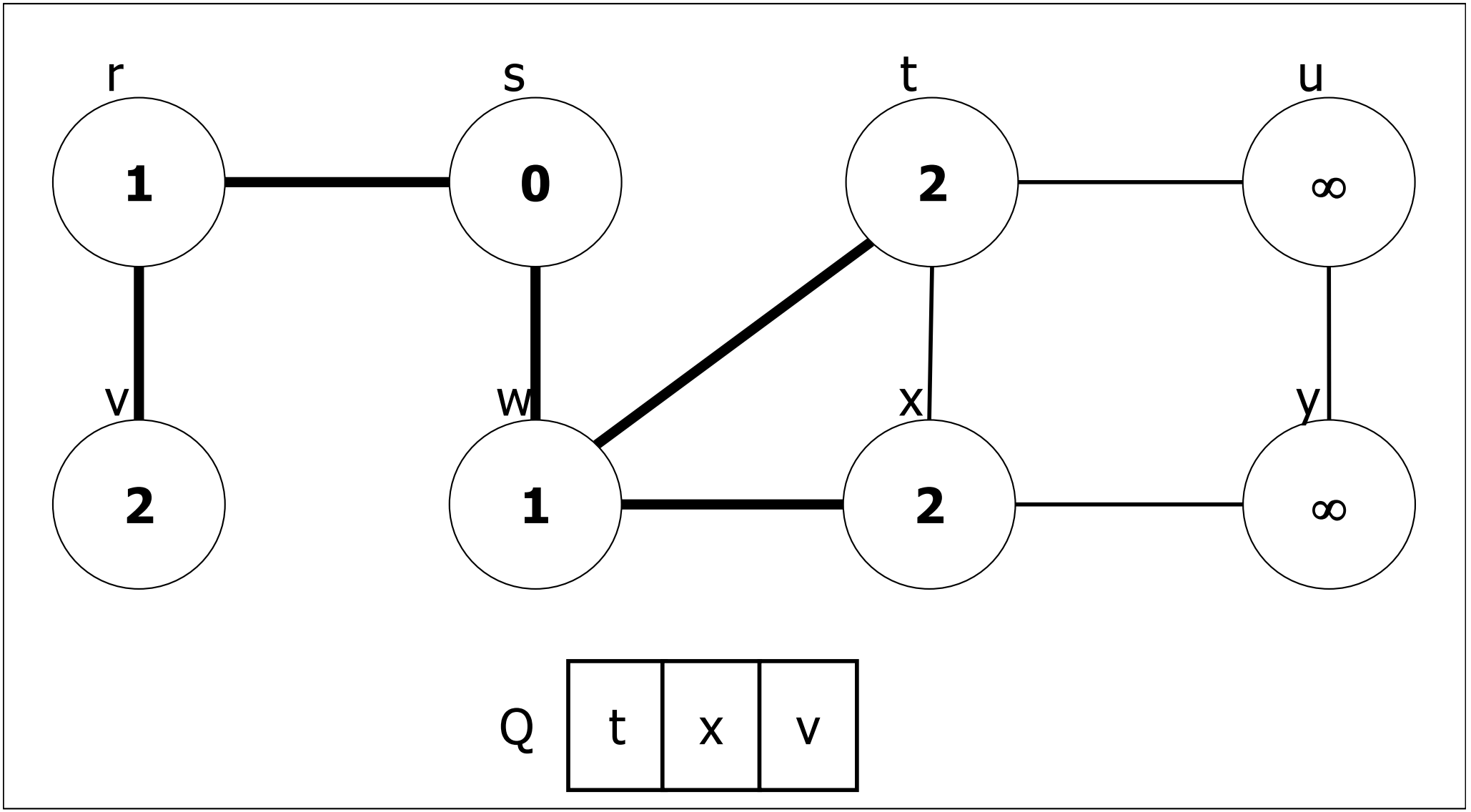


Example

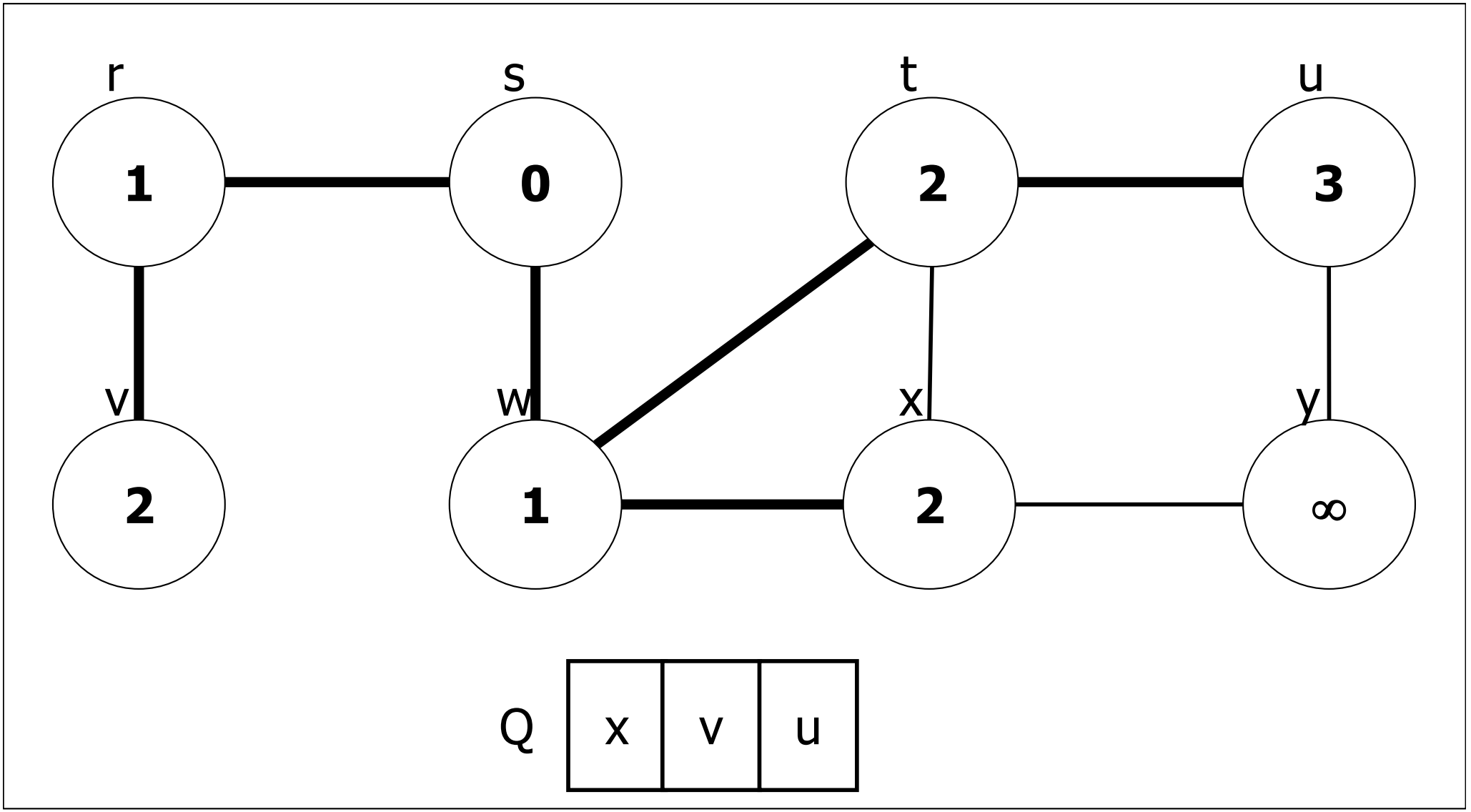
- Gets w from Q
- Enque and gray color all adjacent vertices of w
- Color w in black



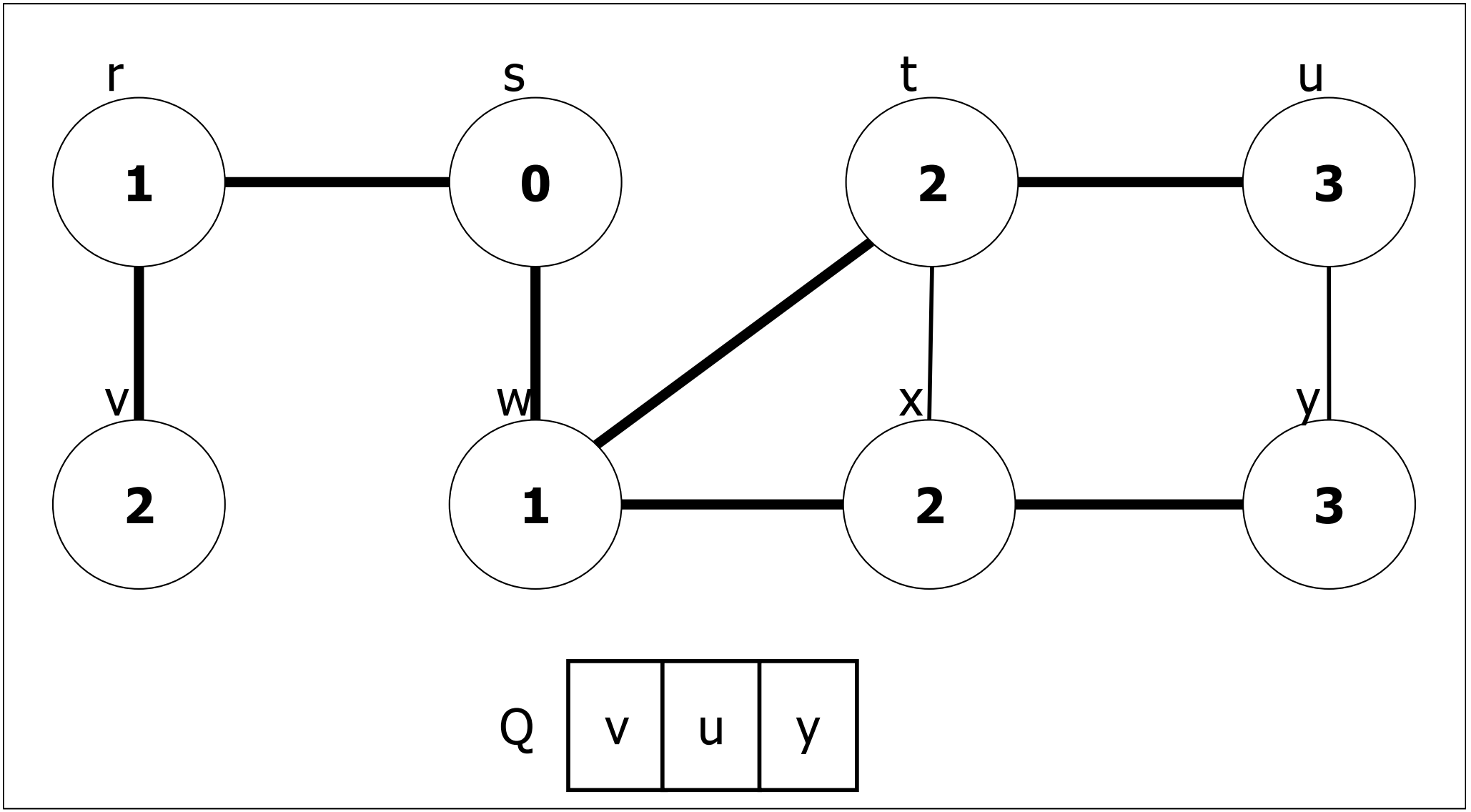
Example



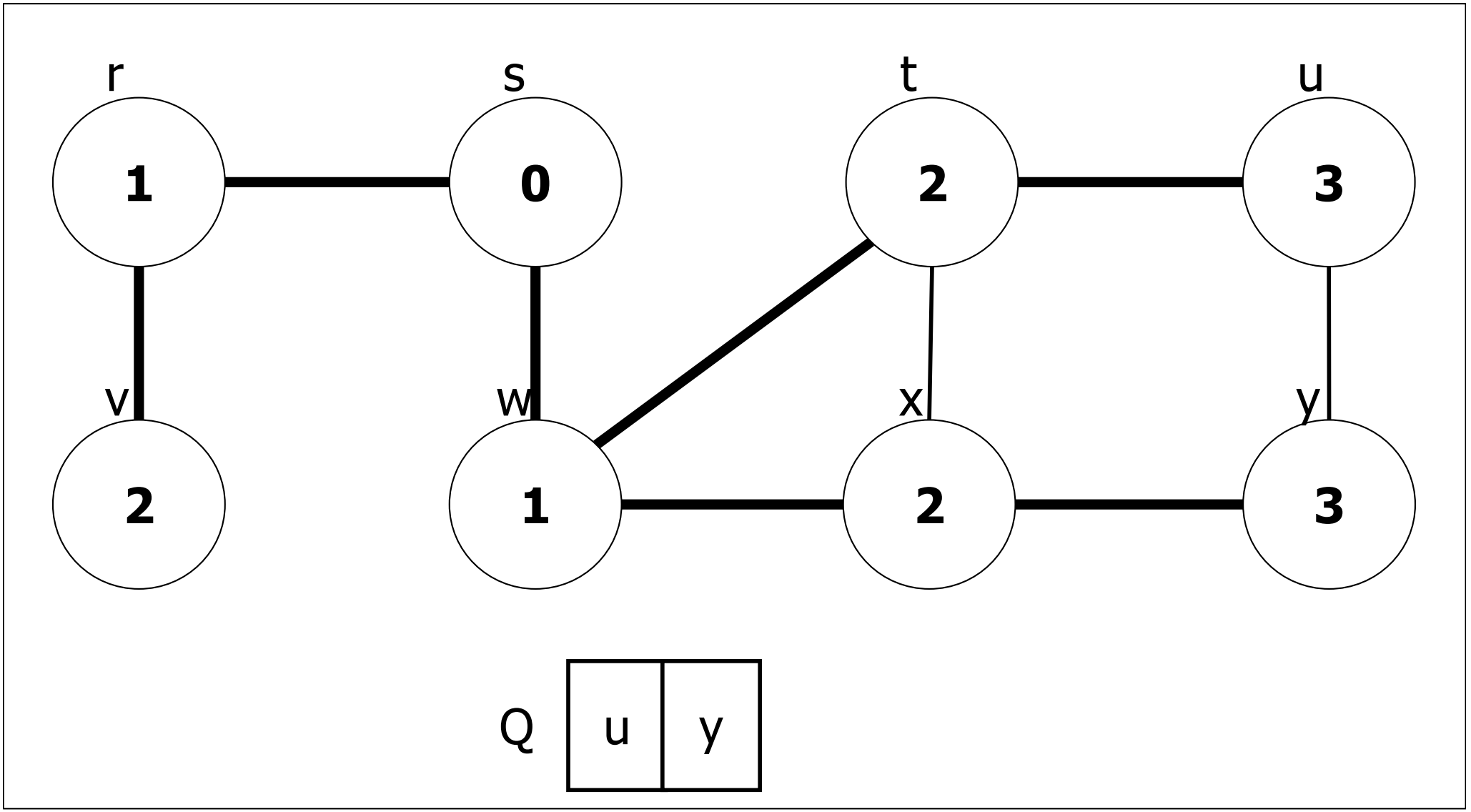
Example



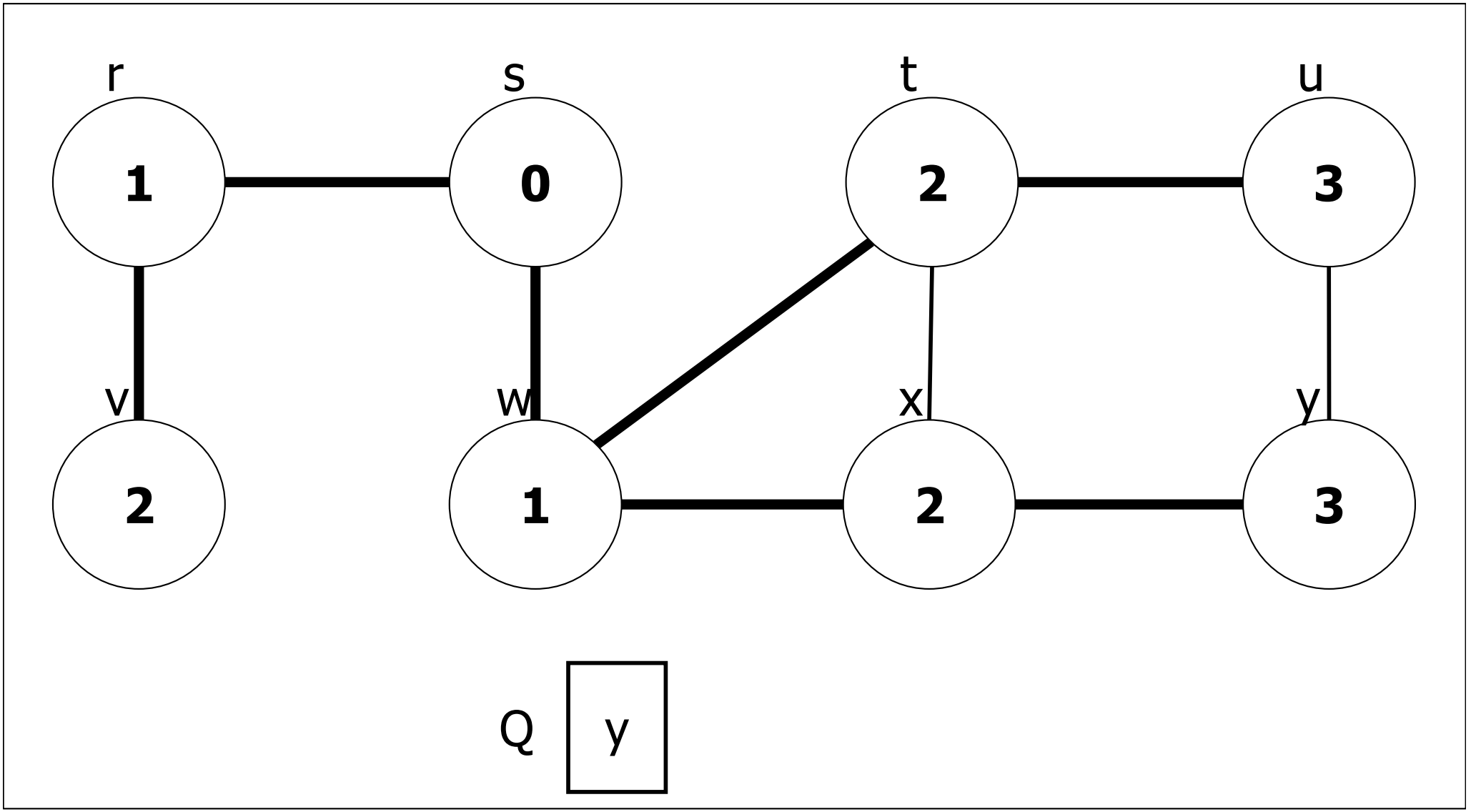
Example



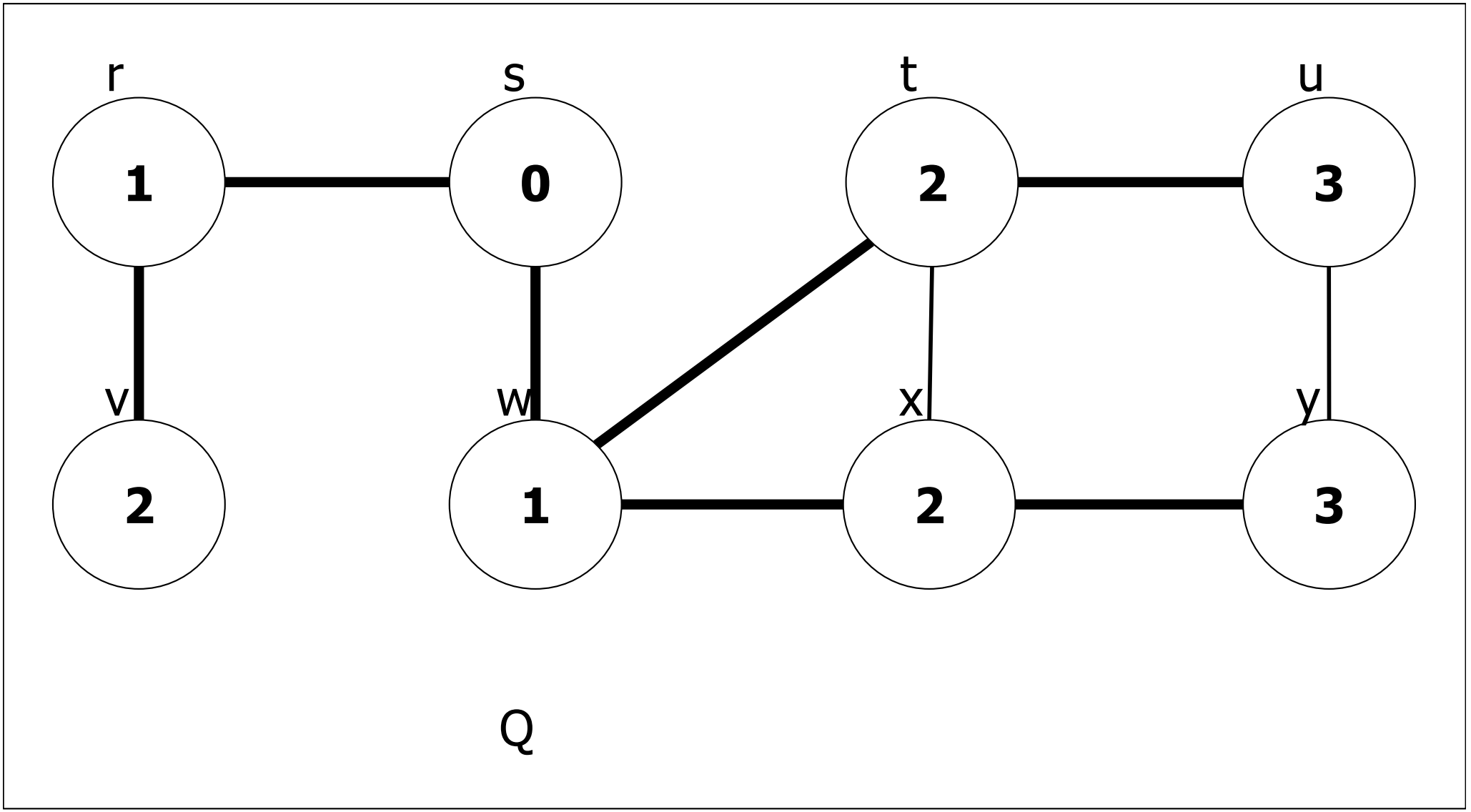
Example



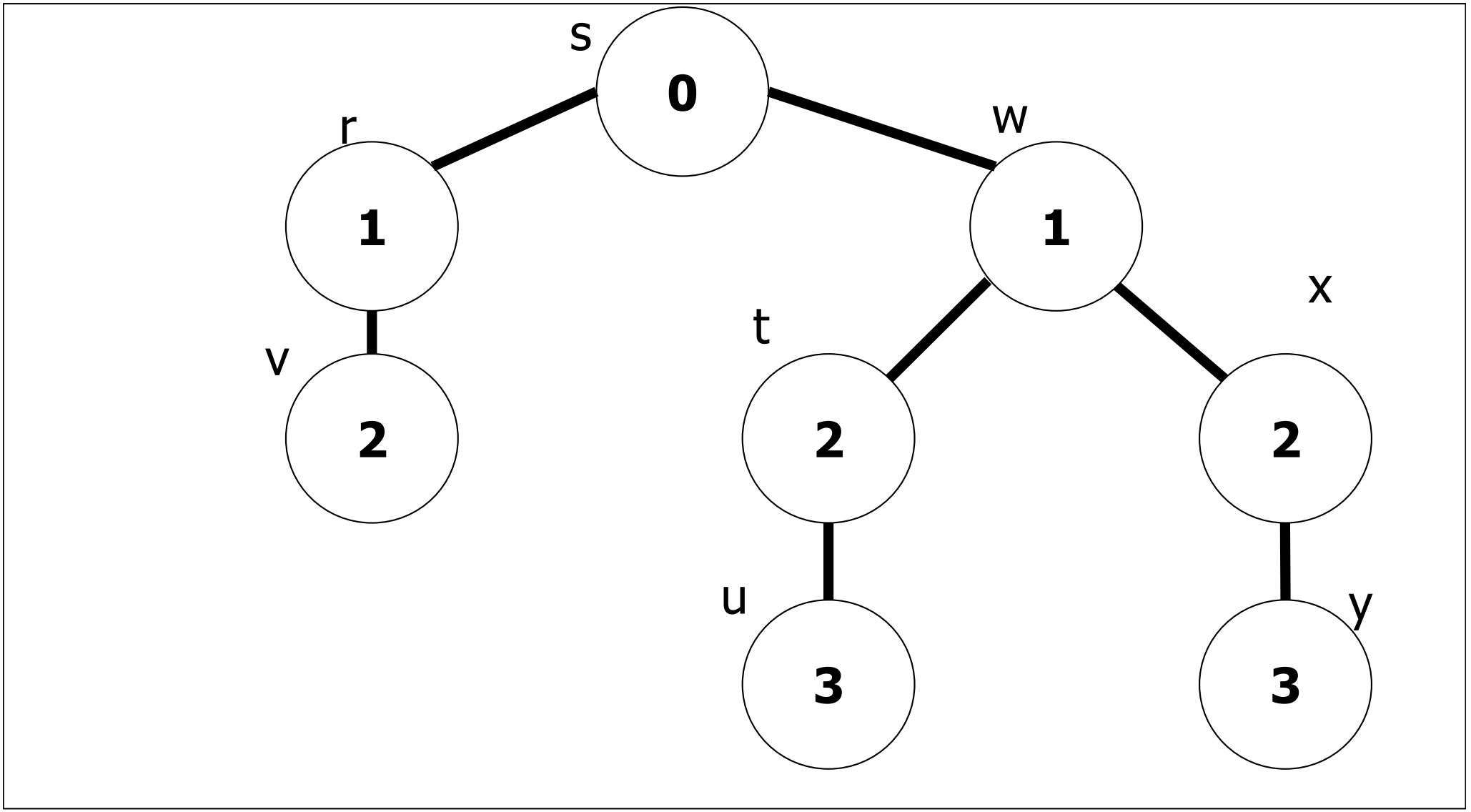
Example



Example



Example: Resulting Tree



BFS: The Code Again

```
BFS(G, s) {
    initialize vertices;
    Q = {s};
    while (Q not empty) {
        u = RemoveTop(Q);
        for each v ∈ u->adj {
            if (v->color == WHITE)
                v->color = GREY;
                v->d = u->d + 1;
                v->p = u;
                Enqueue(Q, v);
        }
        u->color = BLACK;
    }
}
```

Touch every vertex: O(V)

u = every vertex, but only once

So v = every vertex that appears in some other vert's adjacency list

Minimum Distance

- **Given two vertices s and v on an undirected graph, the minimum number of edges on a path from s to v is the distance on the minimum path.**
  - **The BFS computes this distance for each vertex of the graph from the source vertex.**

BFS: Properties

- **BFS calculates the *shortest-path distance* to the source node**
  - **Shortest-path distance $\delta(s,v)$ = minimum number of edges from s to v, or ∞ if v not reachable from s**

BFS: Properties

- **BFS builds *breadth-first tree*, in which paths to root represent shortest paths in G**
  - **Thus can use BFS to calculate shortest path from one vertex to another in O(V+E) time**

Complexity

- **BFS complexity is O(V+E).**

- **Q:**

  What will be the storage cost in addition to storing the tree?

- **A:**

  Total space used:
  O(max(degree(v))) = O(E)

Outline

- Introduction
- Breadth-First Search
- Depth-First Search

Depth First Visit

The Depth-First Search (DFS) follows a different approach.

- **At each step the algorithm visits a vertex adjacent to the last visited.**
- **When this is not possible, the algorithm goes back to the last visited node with adjacent vertex that still have to be visited.**

DFS is usually a recursive function.

- **Two parameters (time stamps) indicate the time when the node was visited:**
  - **d[u] first time visit**
  - **f[u] last time visit**

Pseudo-code (1)

```
DFS(G)
1   for each vertex u ∈ V[G]
2         do color[u] ← WHITE
3            π[u] ← NIL
4   time ← 0
5   for each vertex u ∈ V[G]
6         do if color[u] = WHITE
7               then DFS-VISIT(u)
```

Pseudo-code (2)

```
DFS-VISIT(u)
1  color[u] ← GRAY
2  time ← time + 1
3  d[u] ← time
4  for each v ∈ Adj[u]
5        do if color[v] = WHITE
6              then π[v] ← u
7                   DFS-VISIT(v)
8  color[u] ← BLACK
9  f[u] ← time ← time + 1
```

The white vertex u is visited

The edge (u,v) is traversed

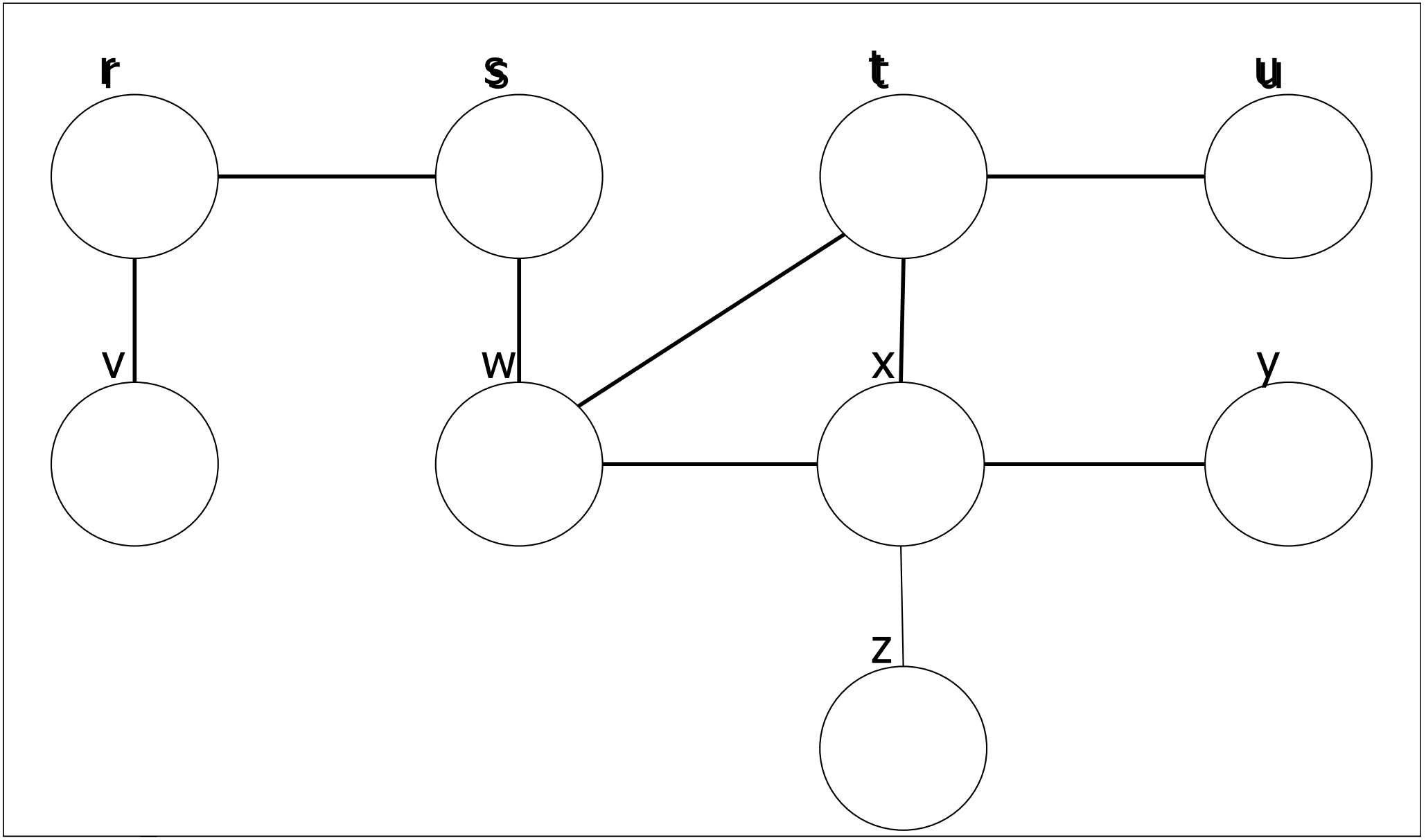
U becomes black

Note

- **The coloring of the vertices follows this criteria:**
  - **All vertices are white before beginning the visit.**
  - **A vertex becomes gray when is visited for the first time**
  - **A vertex becomes black when all its adjacent vertices have been visited.**

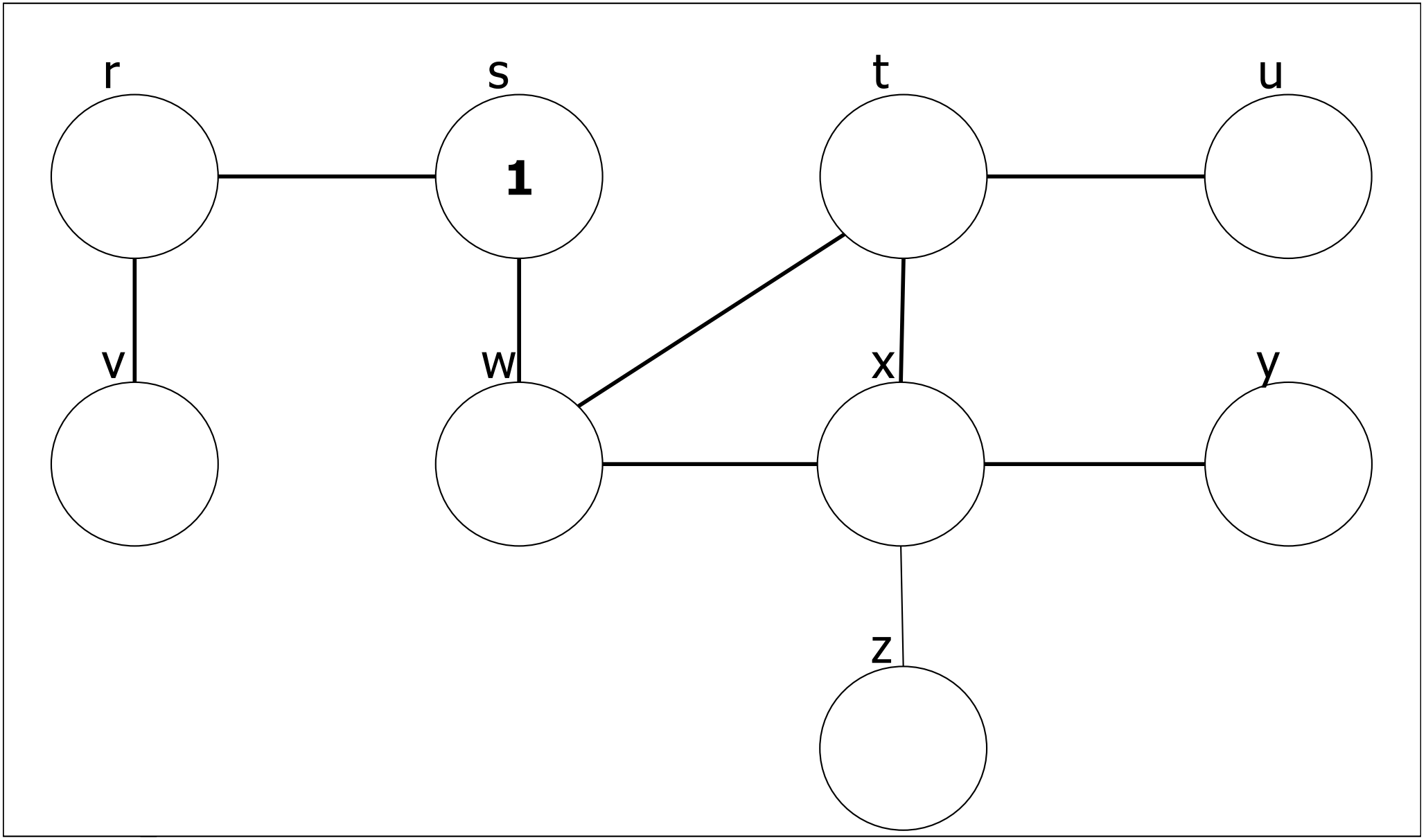
Example

source: s



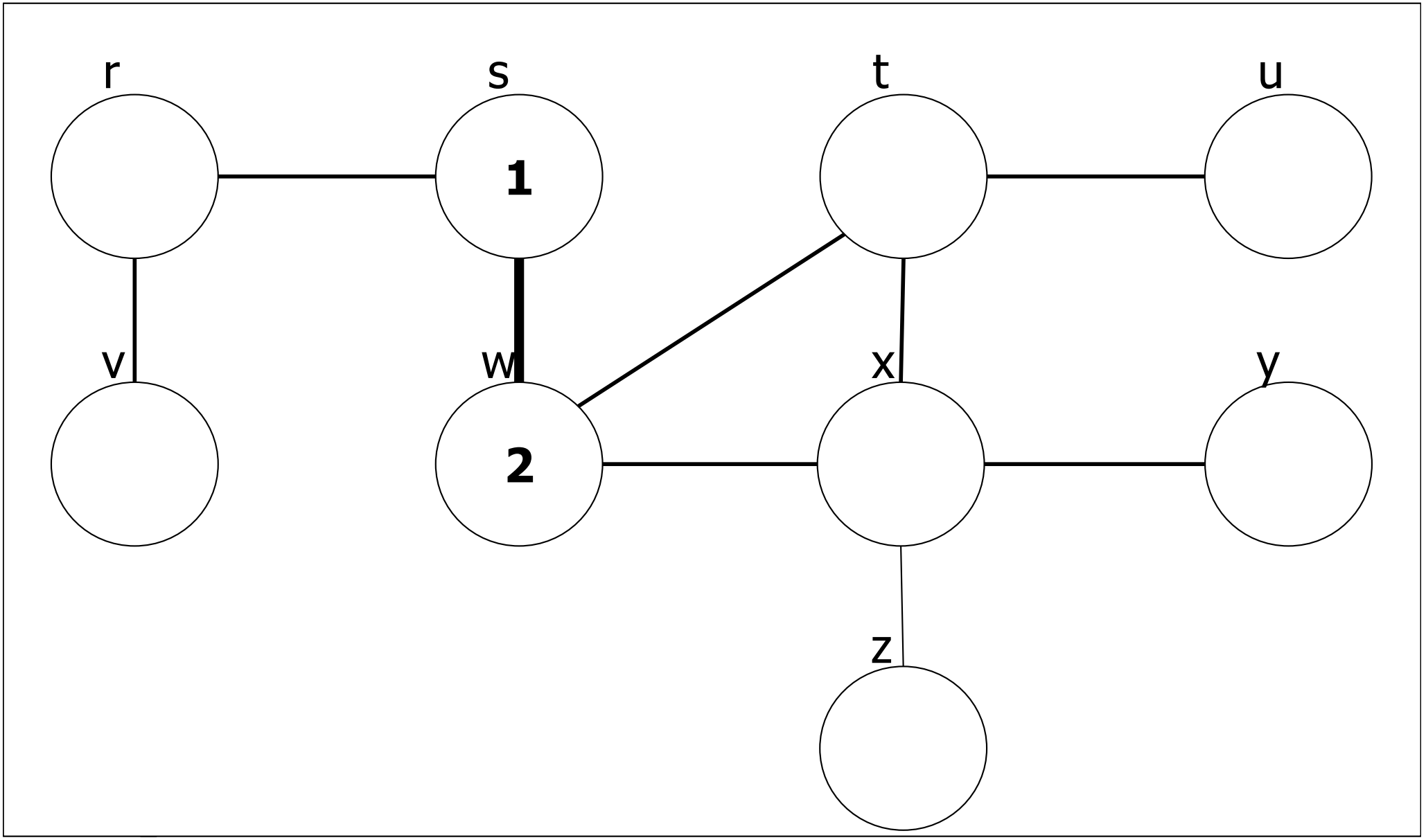
Example

Visit s



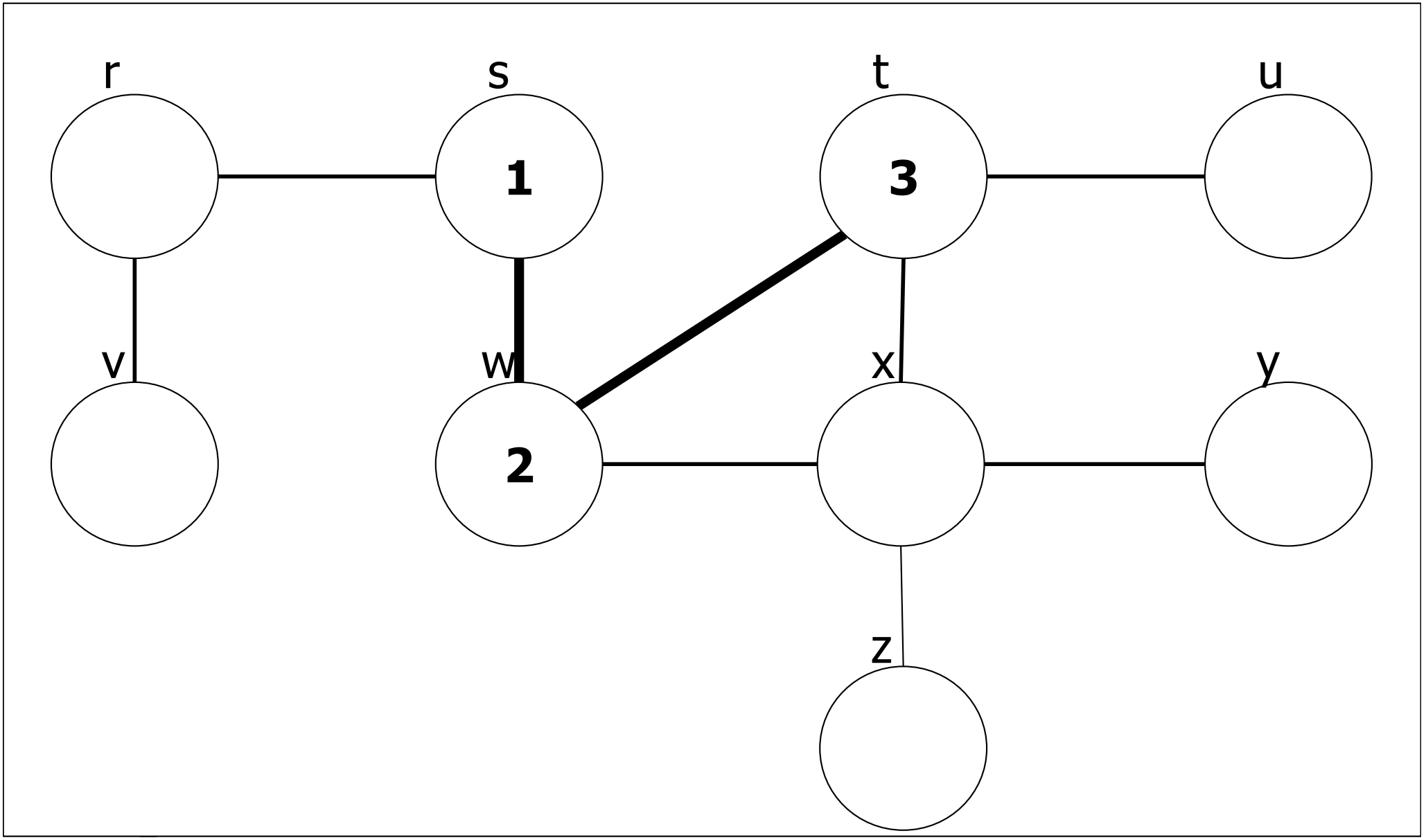
Example

Visit w



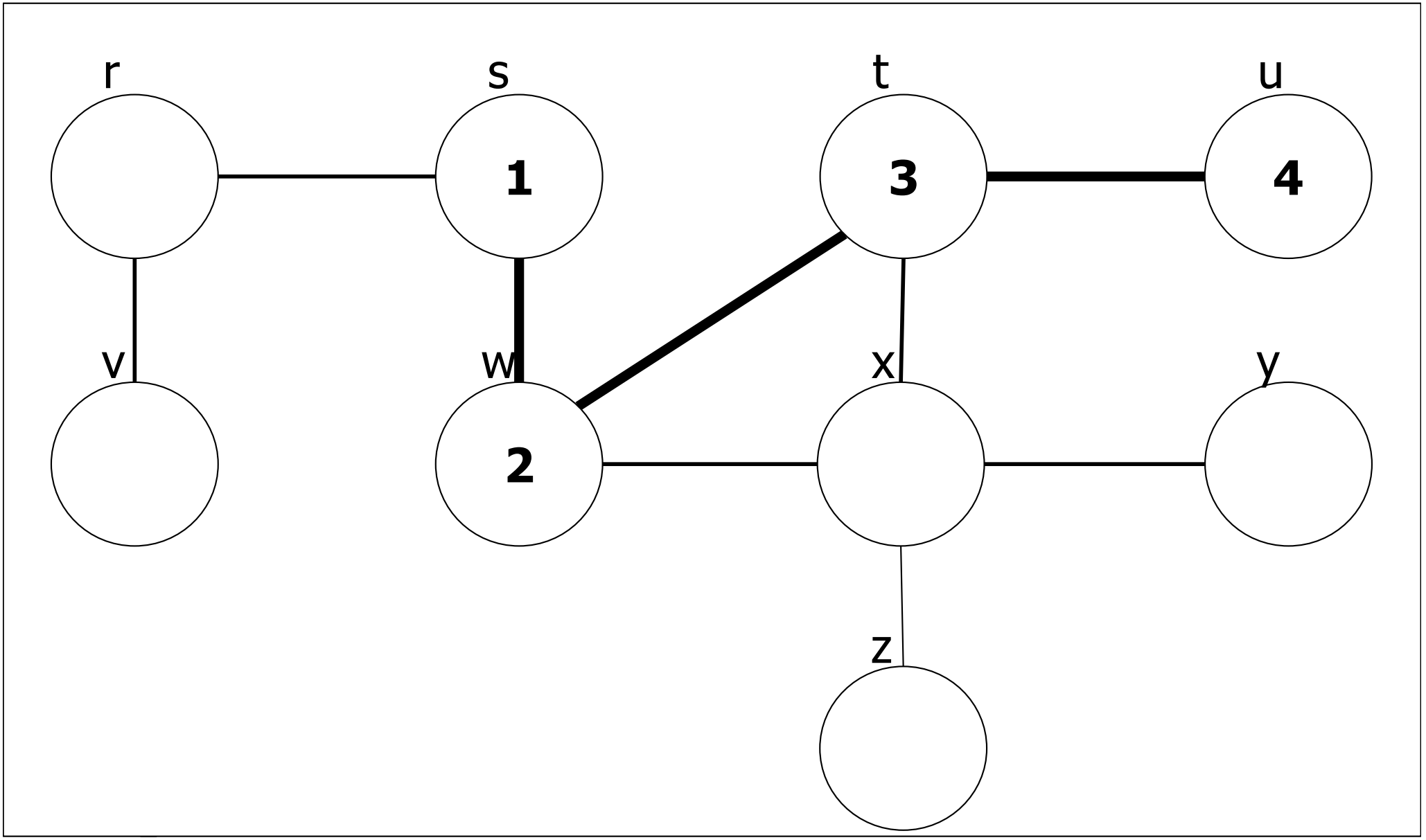
Example

Visit t



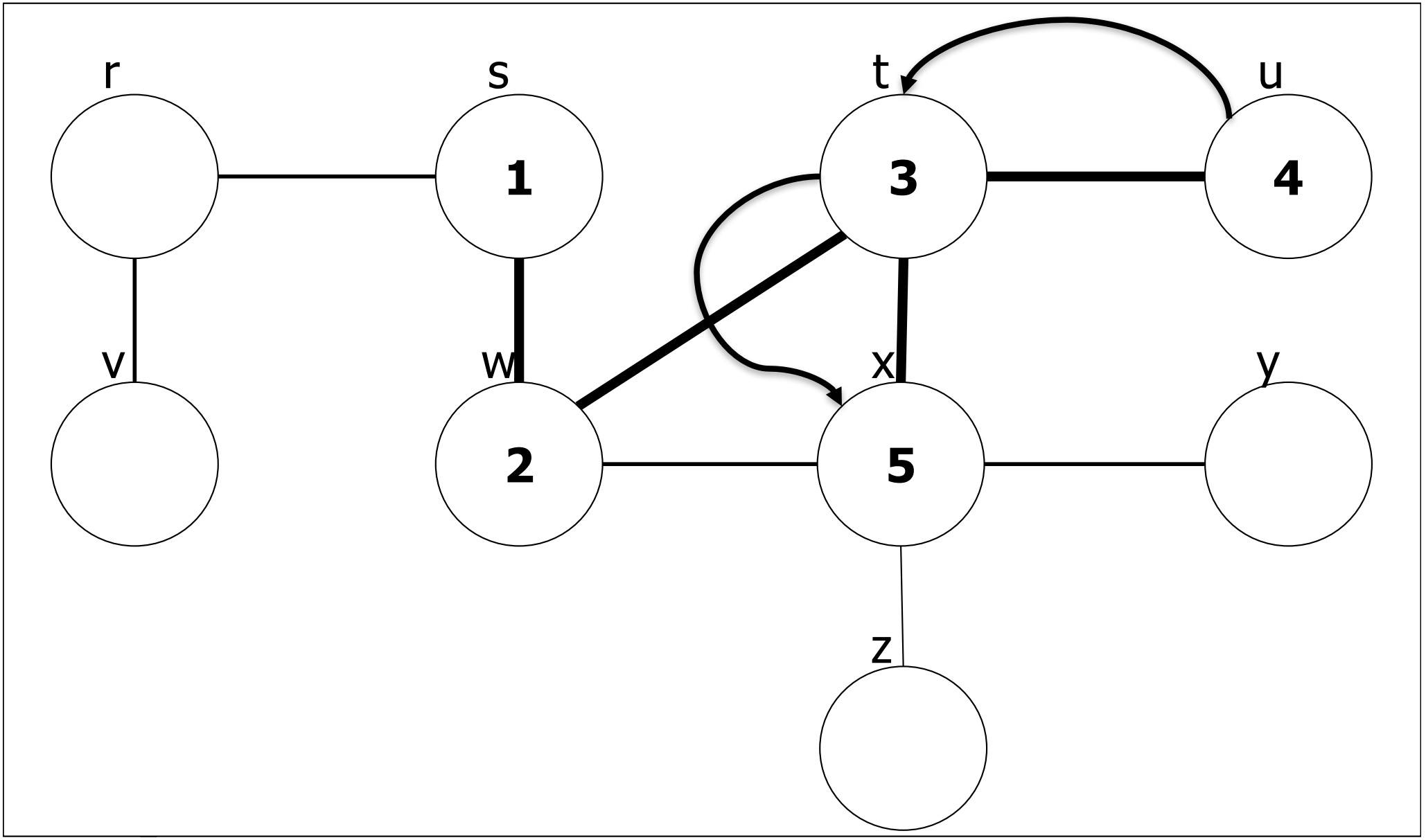
Example

Visit u



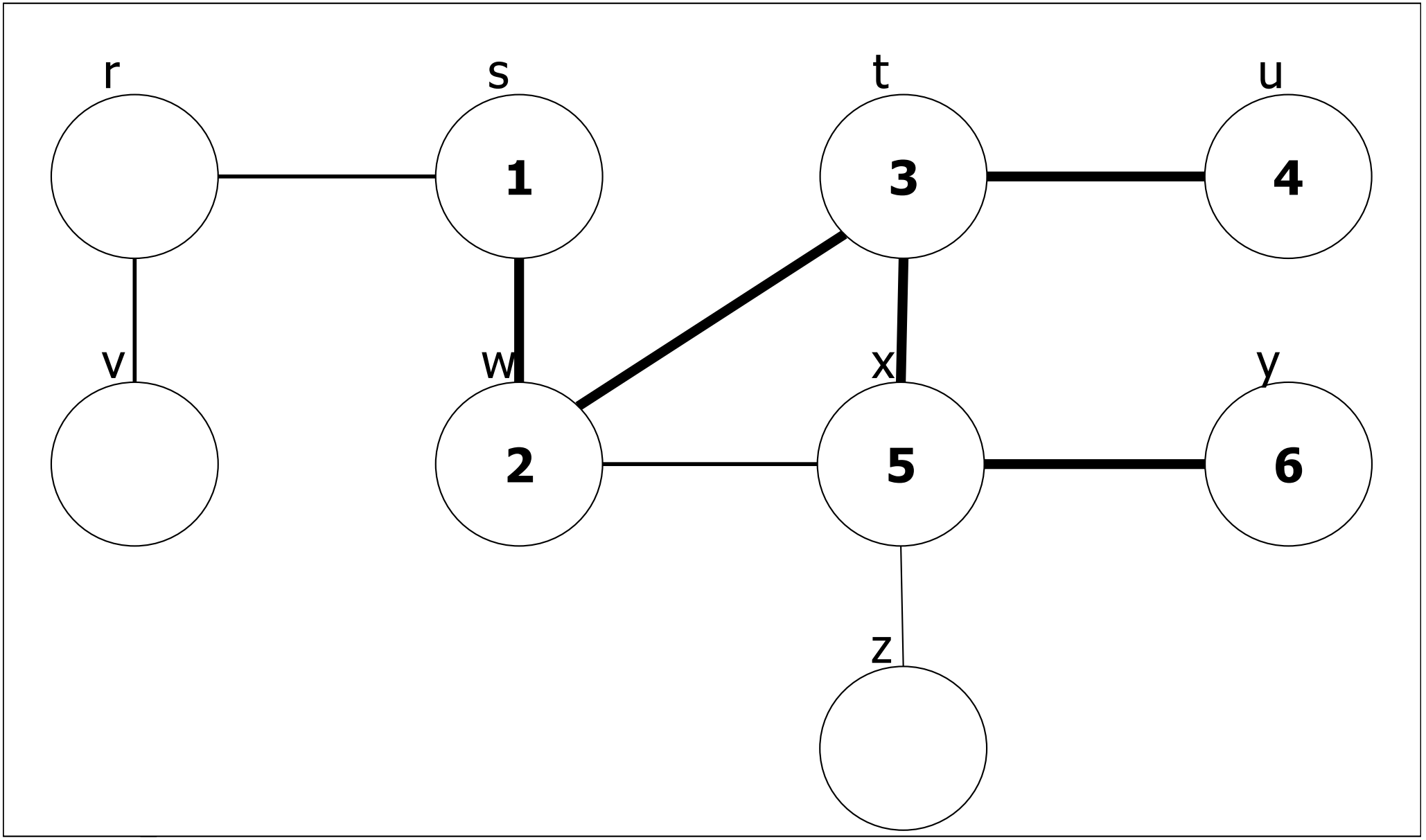
Example

Visit x



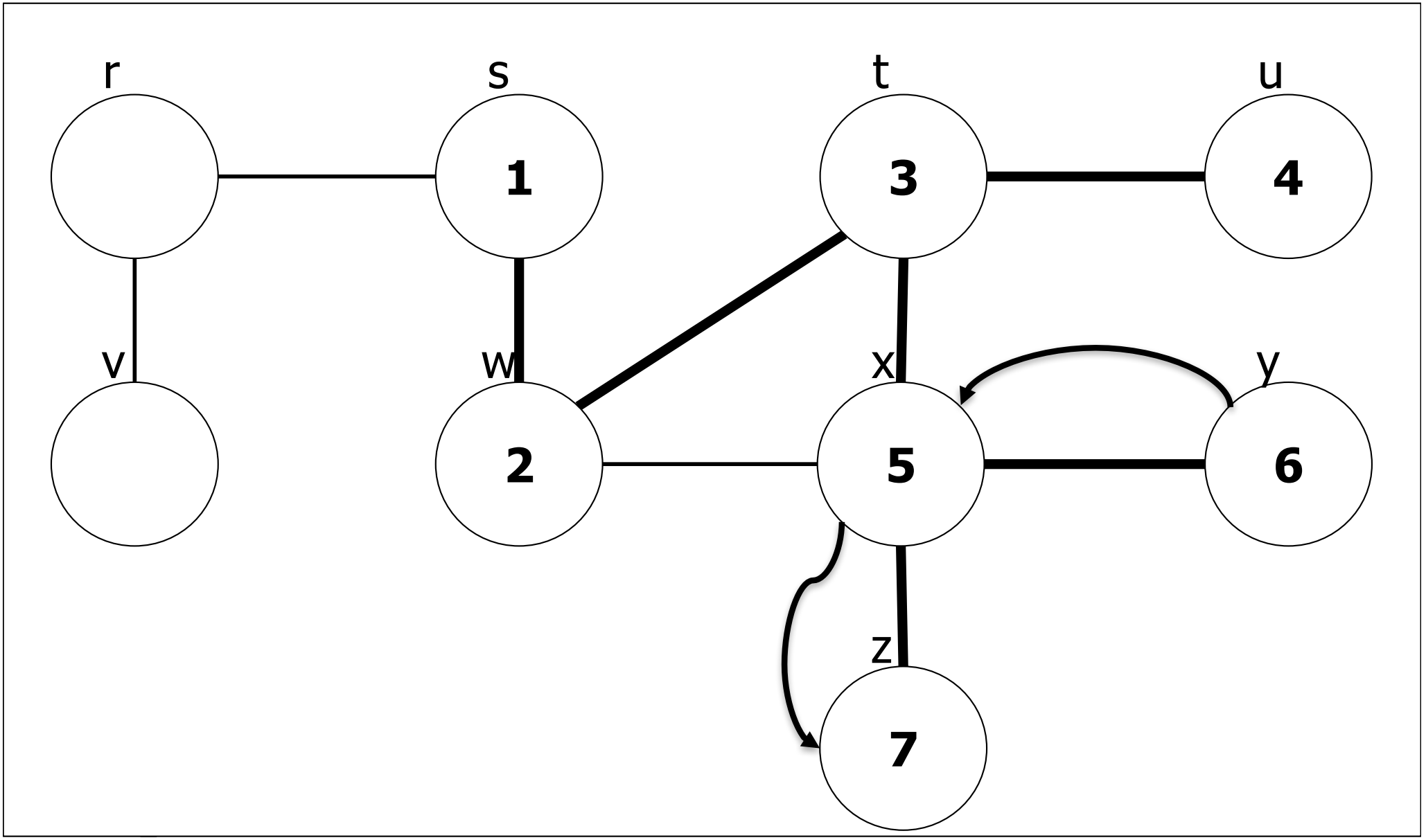
Example

Visit y



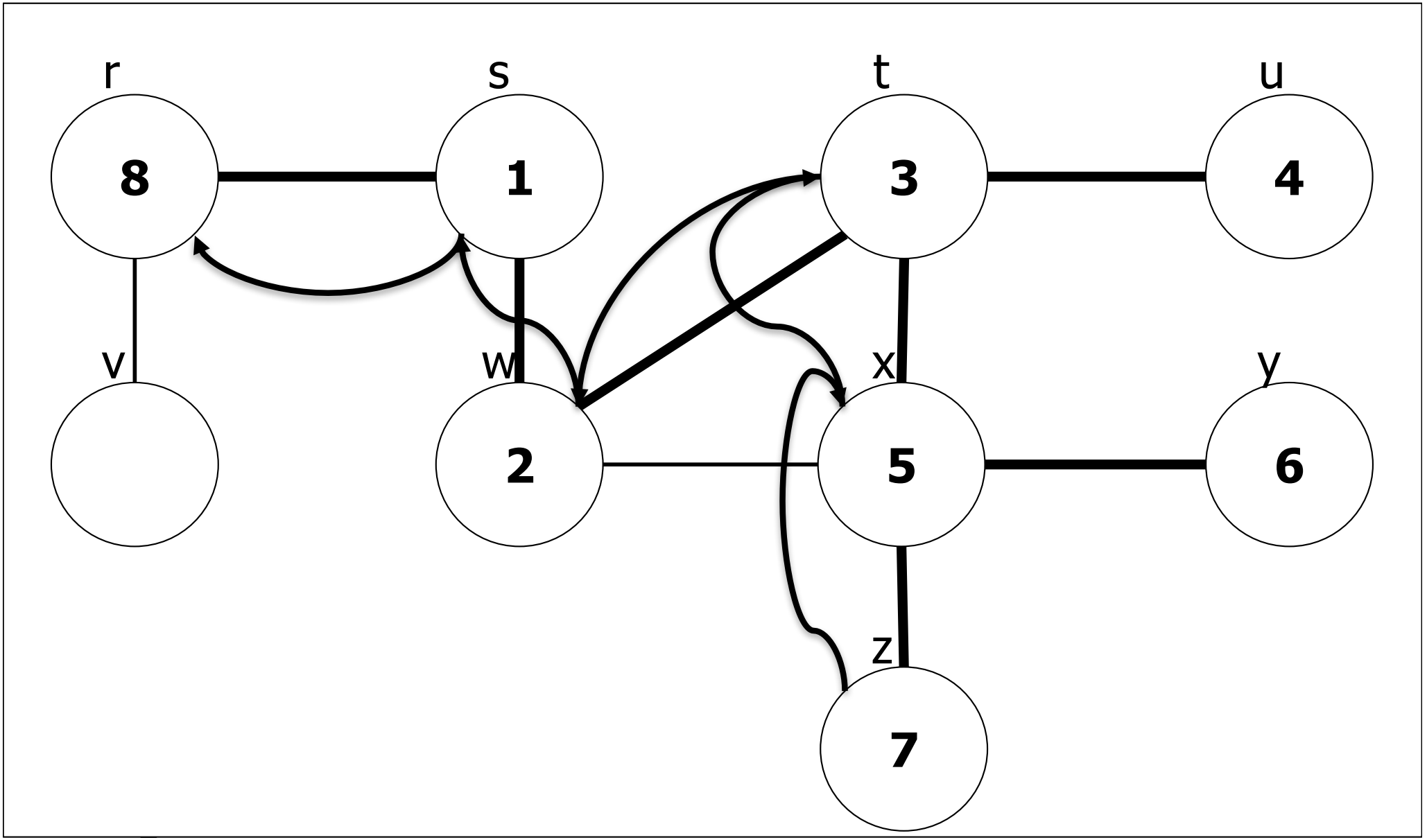
Example

Visit z



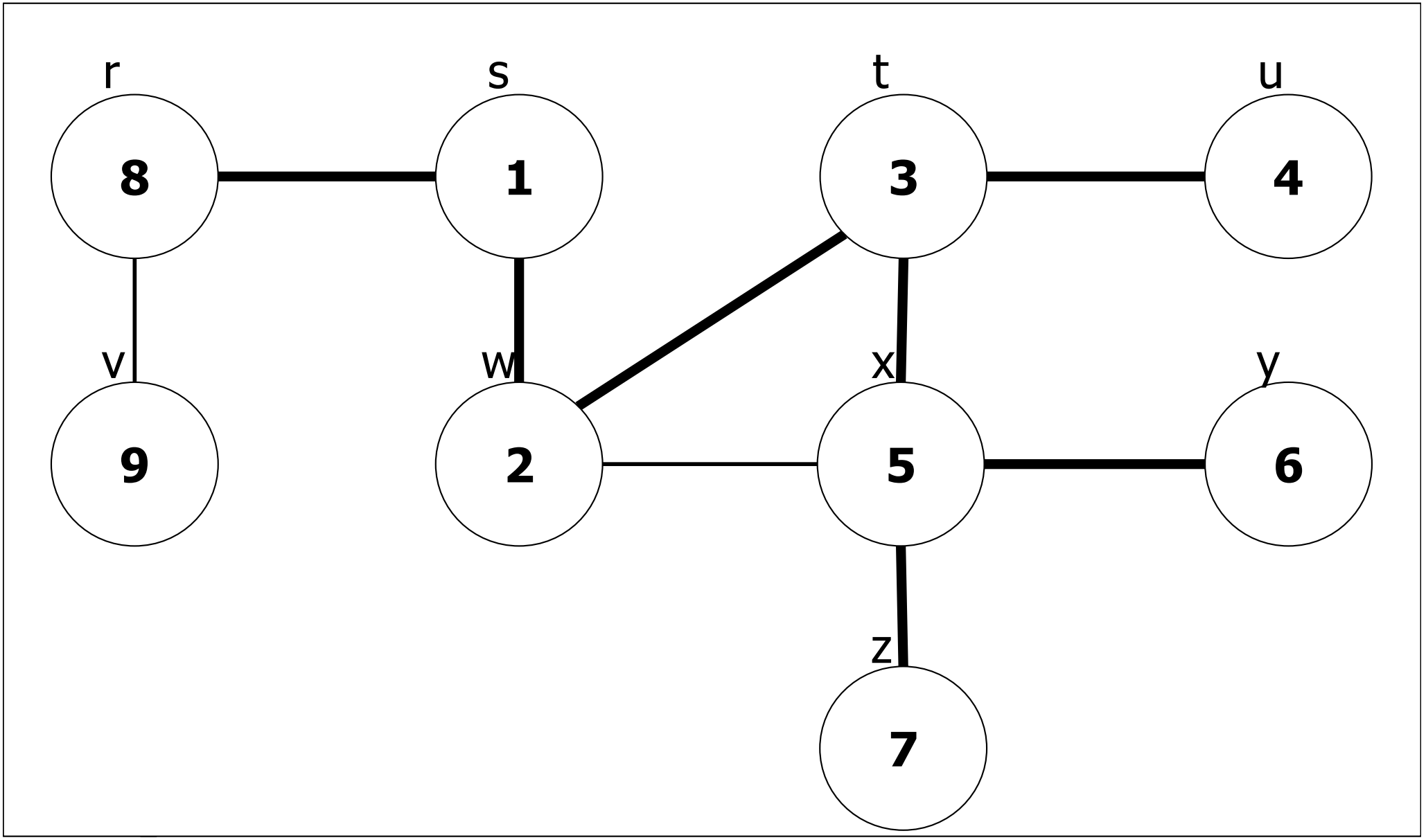
Example

Visit r



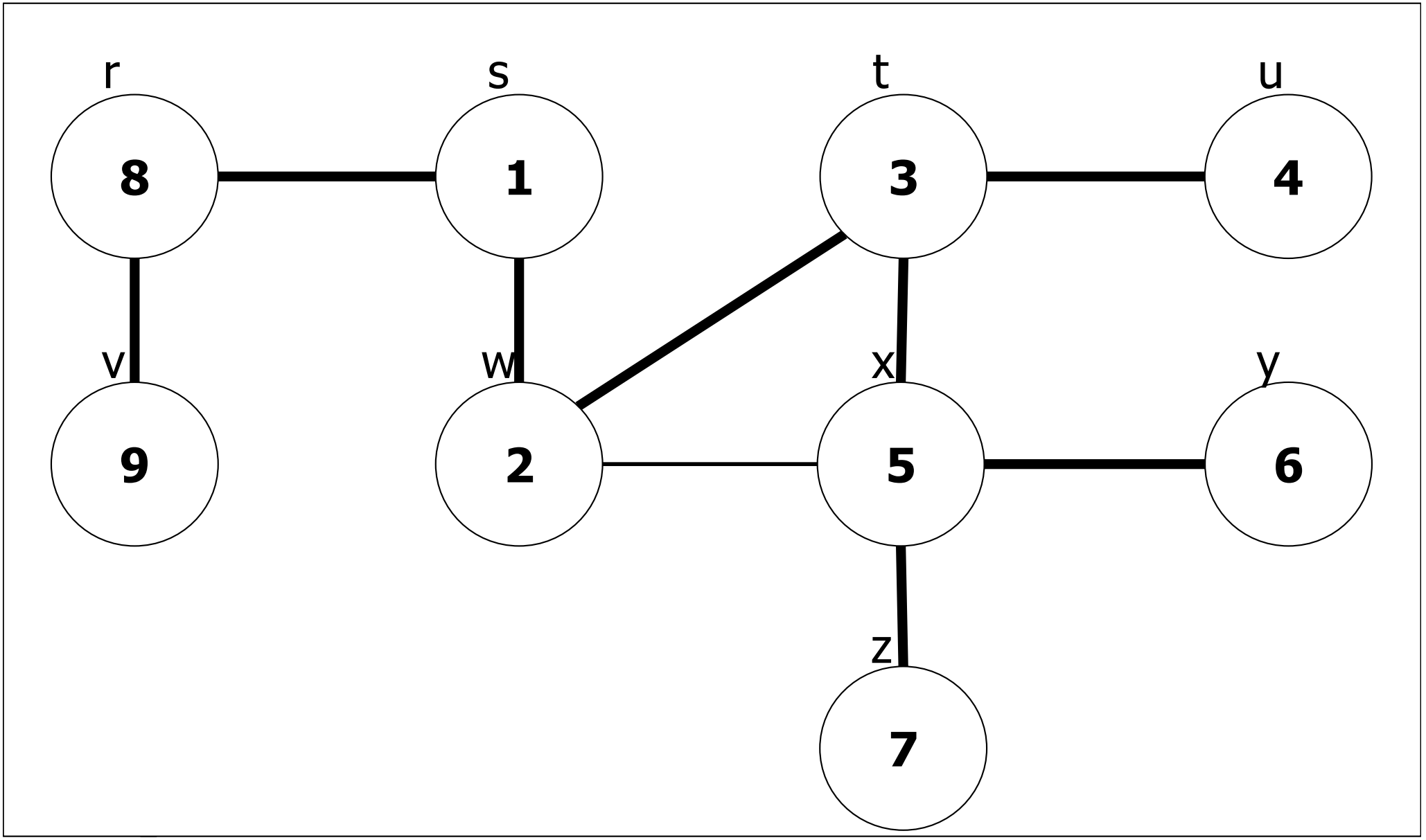
Example

Visit v



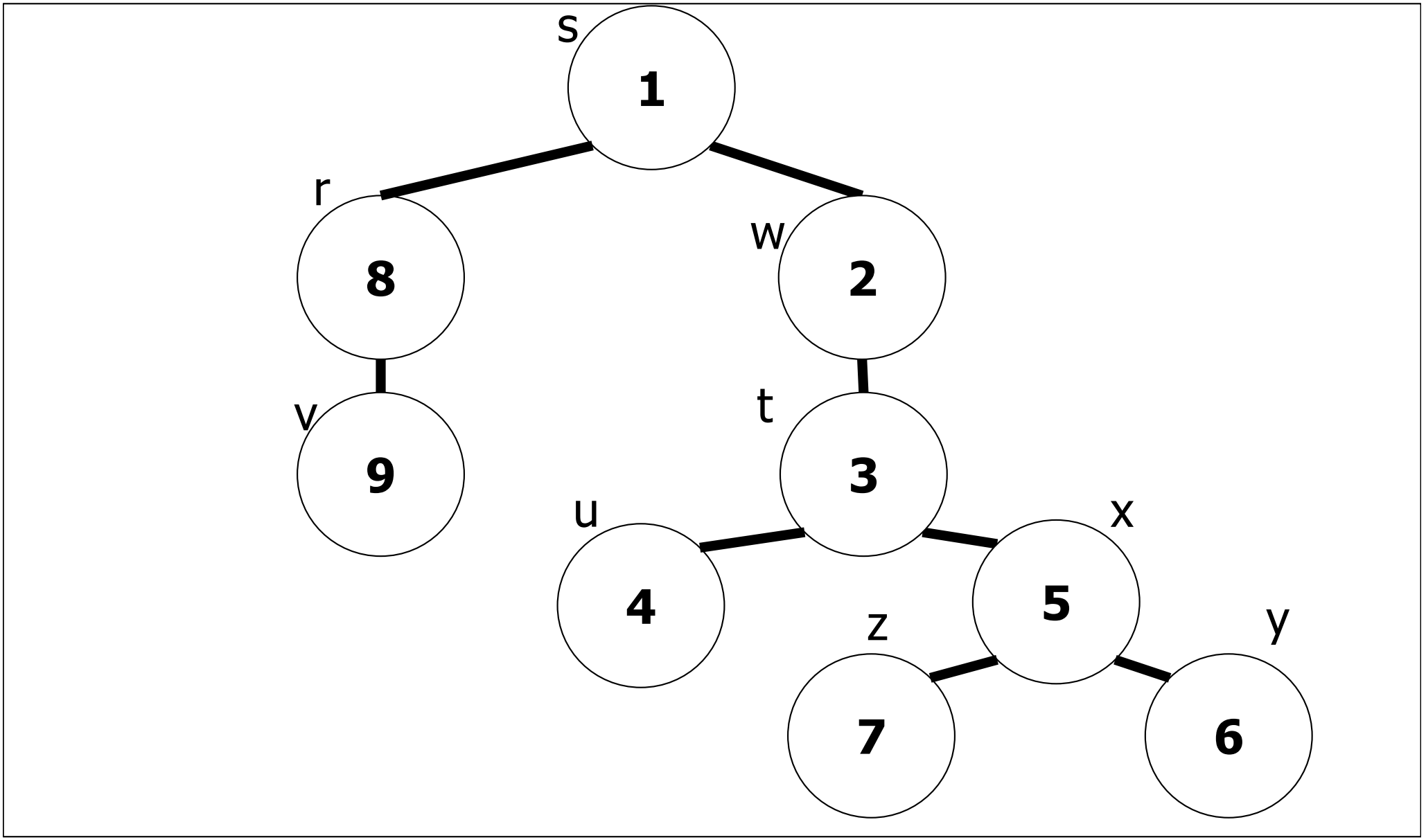
Example

Concludes the visit



Example

Resulting Tree



Complexity

- **The complexity of the DFS visit is Θ(V+E).**

DFS Forest

- **The DFS builds a DFS forest, composed of one or more DFS trees.**
- **The edges of the forest are called tree edges.**

Edge classification

- **In a directed graph, edges can fall in one of 4 categories:**
  - ***Tree edges*** **(T)**
  - ***Backward edges*** **(B): they are not T edges, and they connect a vertex with one of its parents**
  - ***Forward edges*** **(F): they are not T or B edges and they connect a vertex with one of its children**
  - ***Cross edges*** **(C): the remaining edges.**

Edge classification

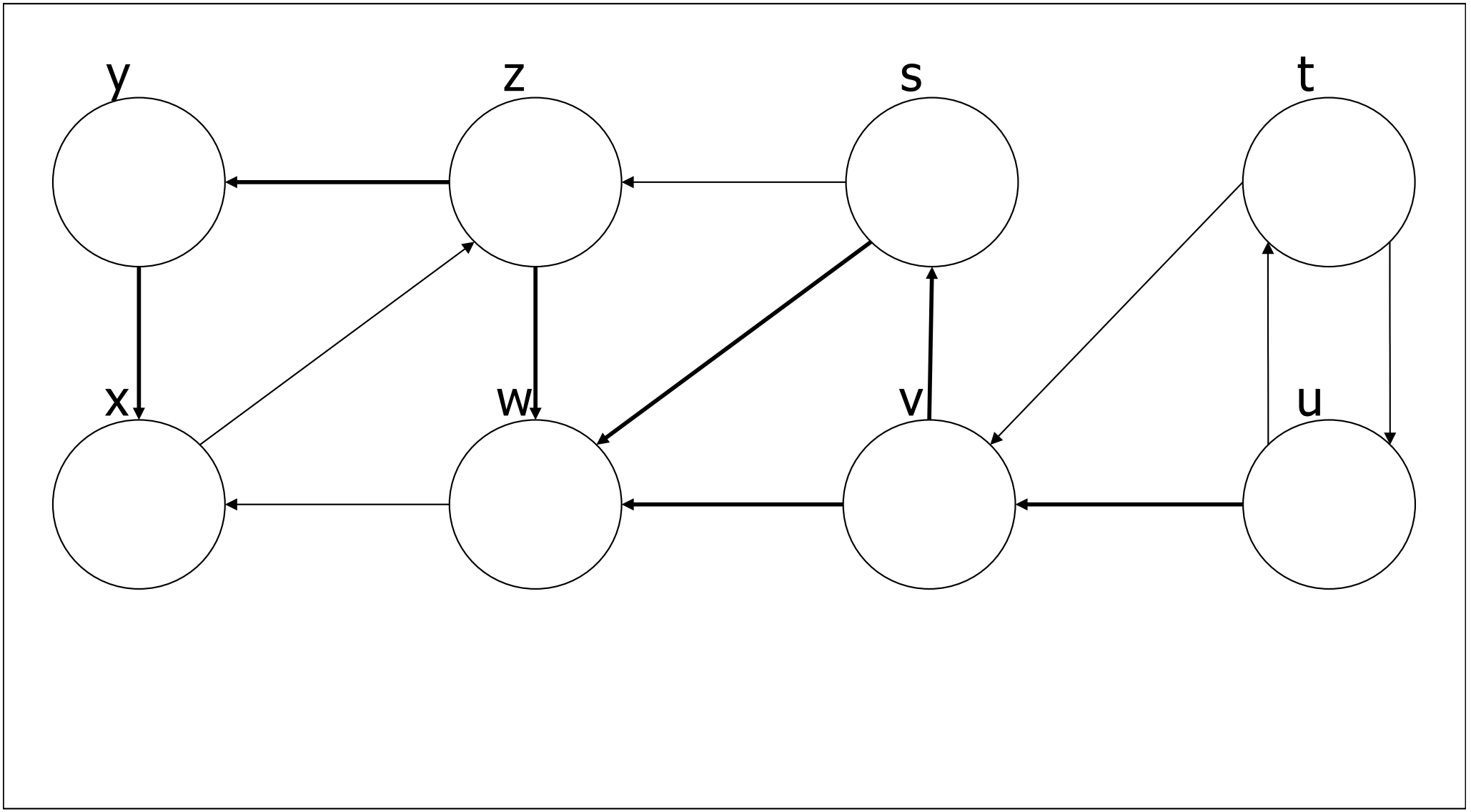
- **DFS can be easily modified to classify edges.**
- **Every time the algorithm traverses an edge (u,v) it checks the color of the v vertex:**
  - **If it is white, the edge is a T edge**
  - **If it is grey, the edge is a B edge**
  - **If it is black, the edge is an F edge (if d[u]<d[v]) or a C edge (if d[u]>d[v]), where d is the discovery time stamp.**

Visiting time stamps

- **Each vertex has two time stamps:**
  - **The first time stamp (or discovery time stamp) records when a vertex is first discovered**
  - **The second time stamp records when the search finishes examining adjacency list of vertex.**

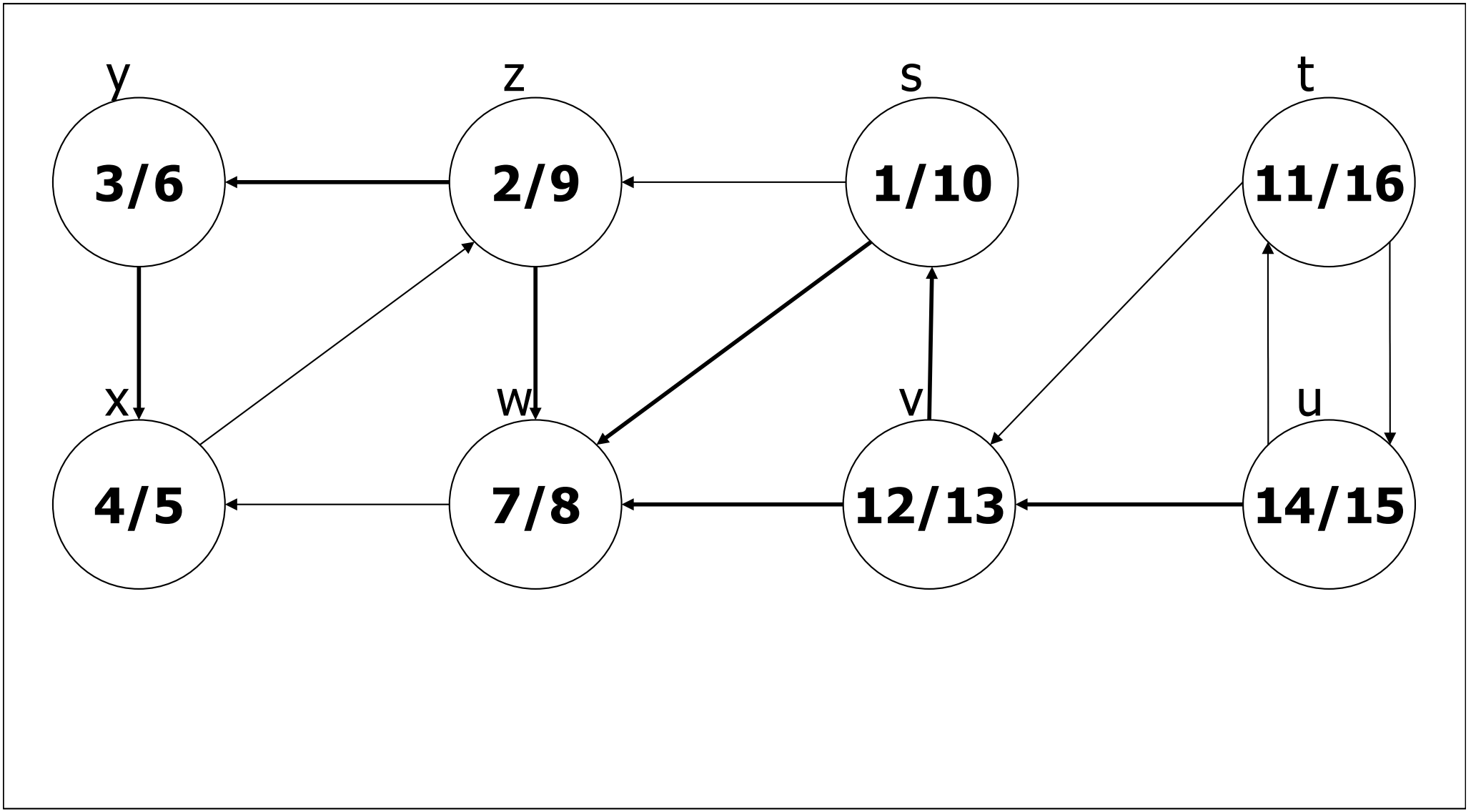
Example

source: s



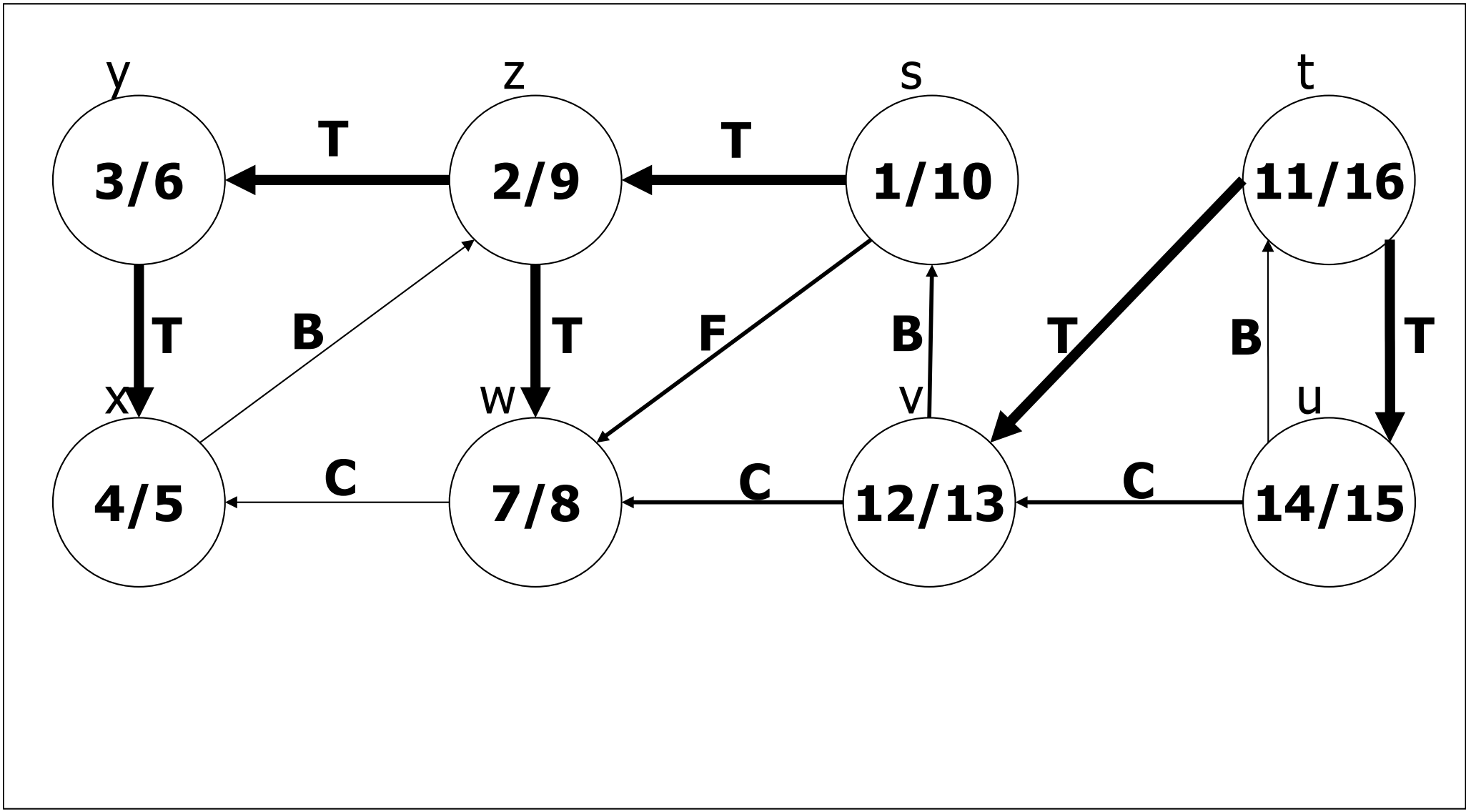
Example

DFS and intervals computation



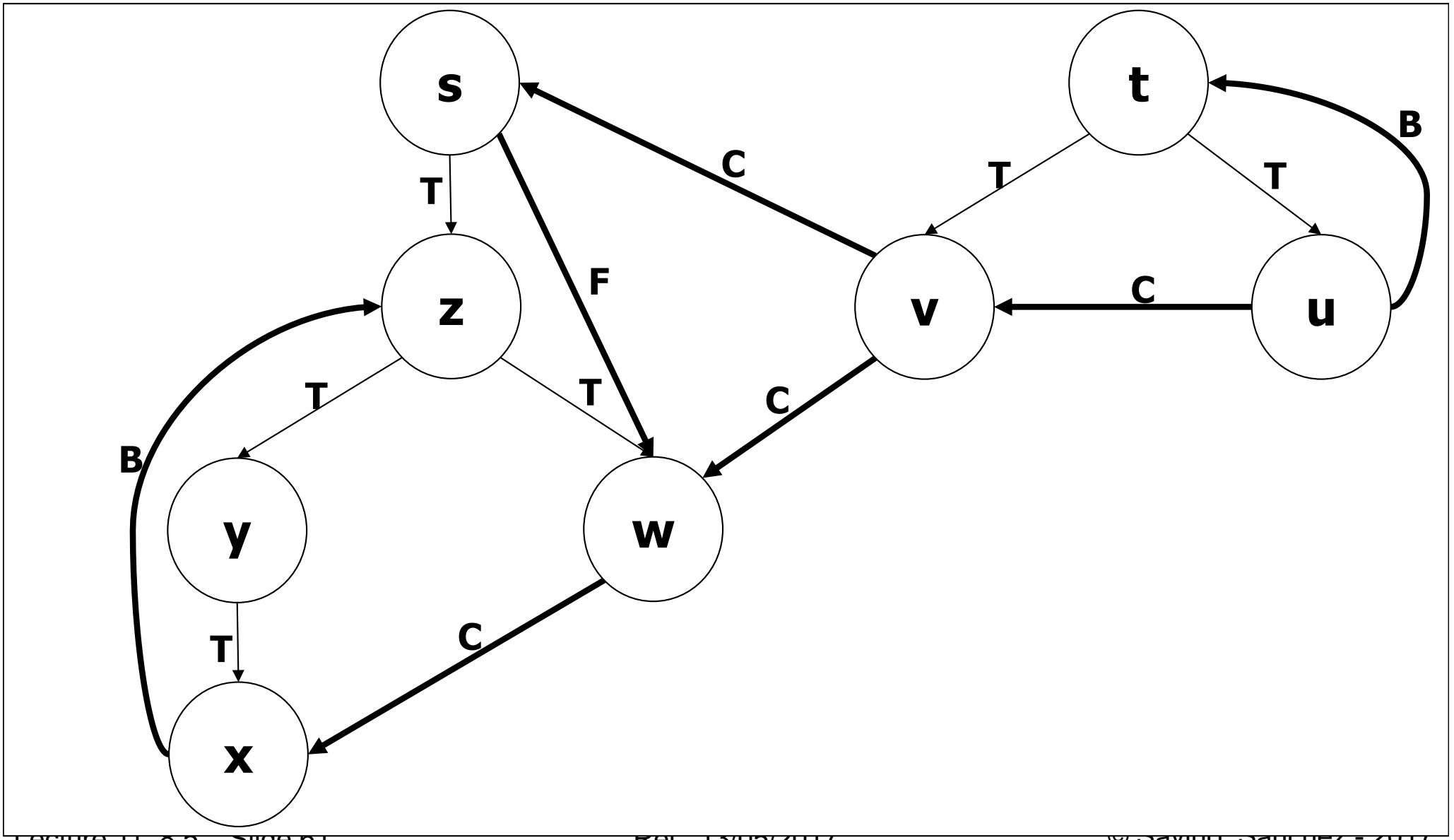
Example

Edges Classification



Example (

Edges classification



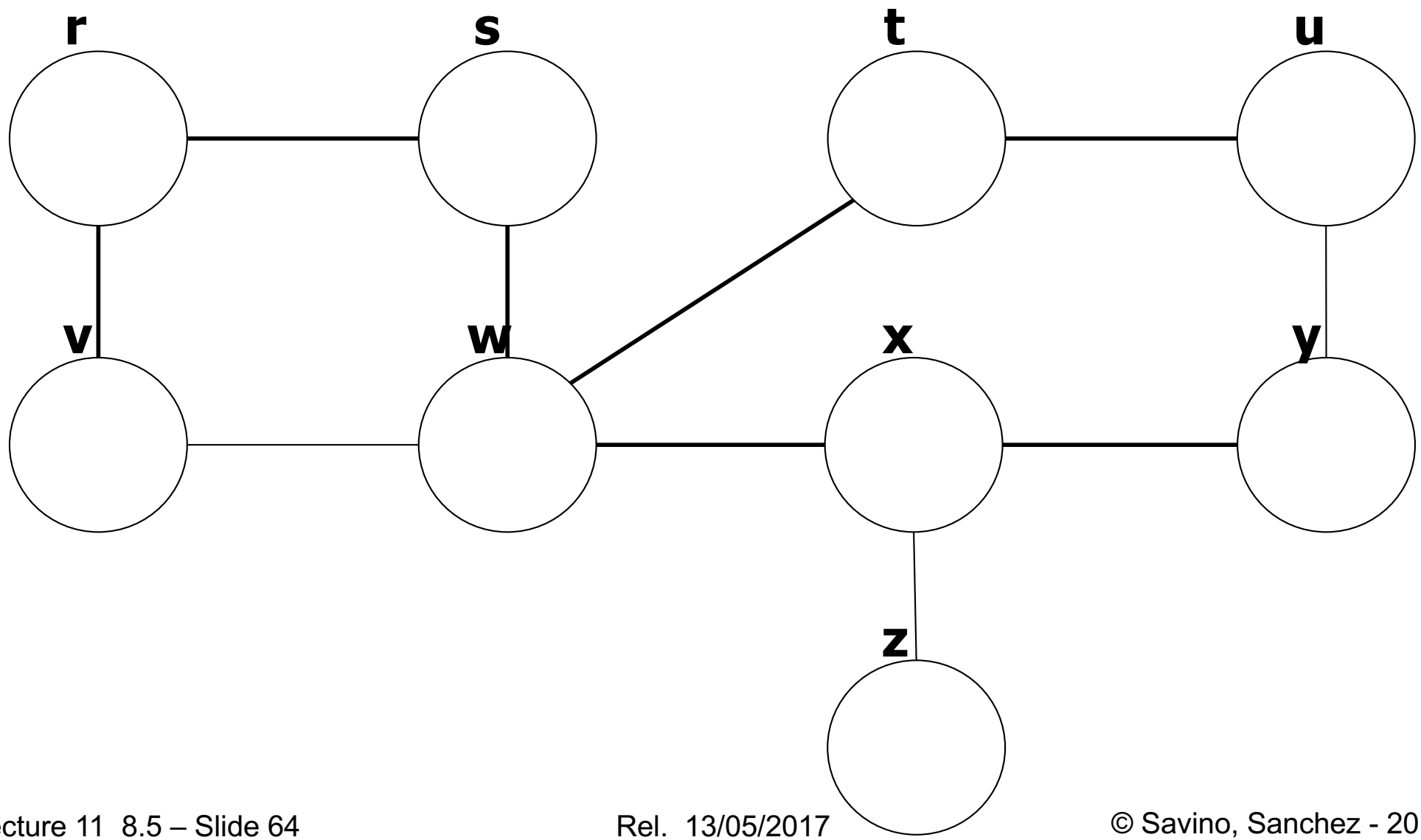
Edges classification – undirected graphs

- **In a not directed graph there are no forward or traversing edges.**

Cycles

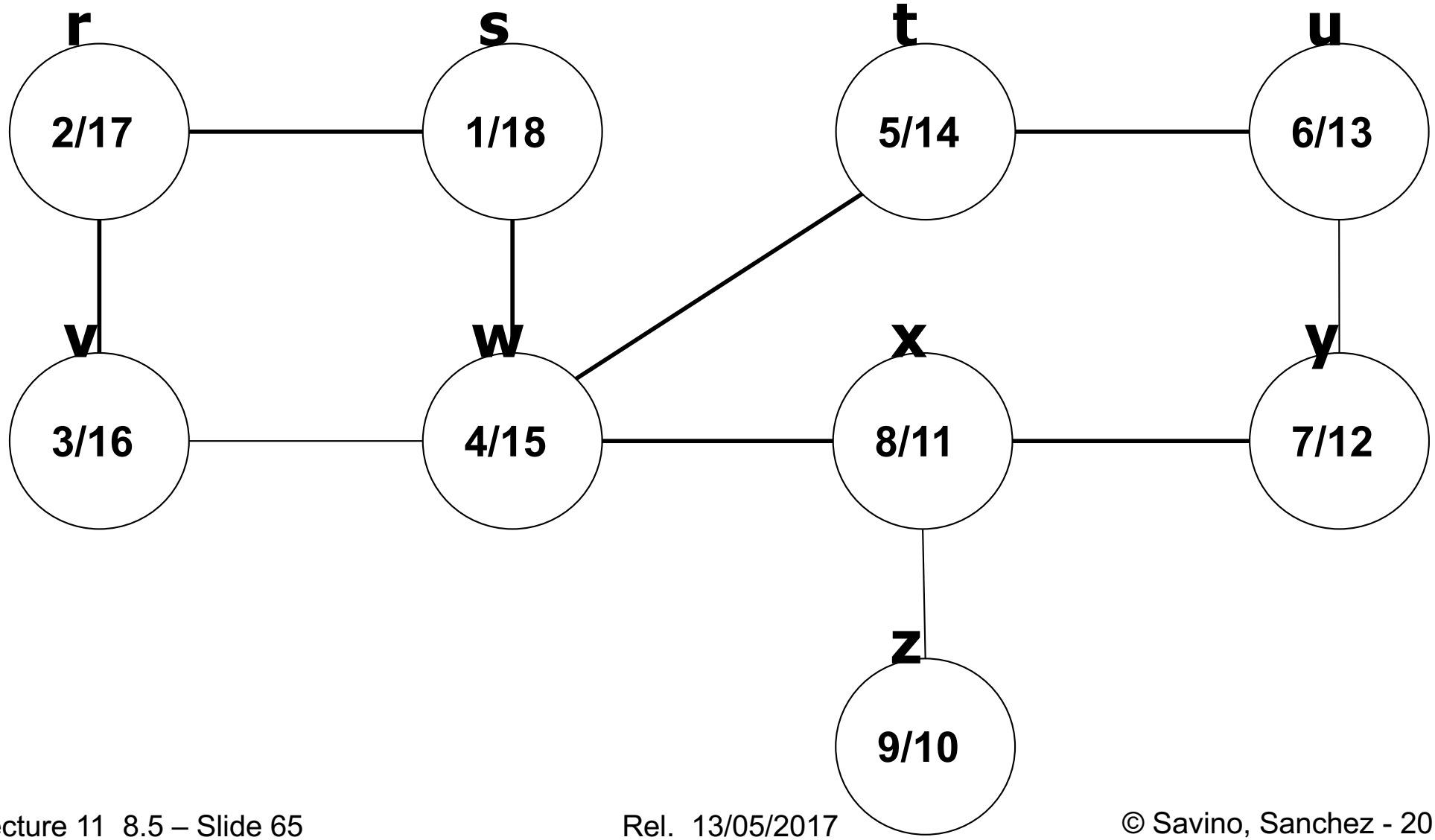
- **A directed graph DOES NOT HAVE CYCLES iff a DFS does not produce backward edges.**

Example



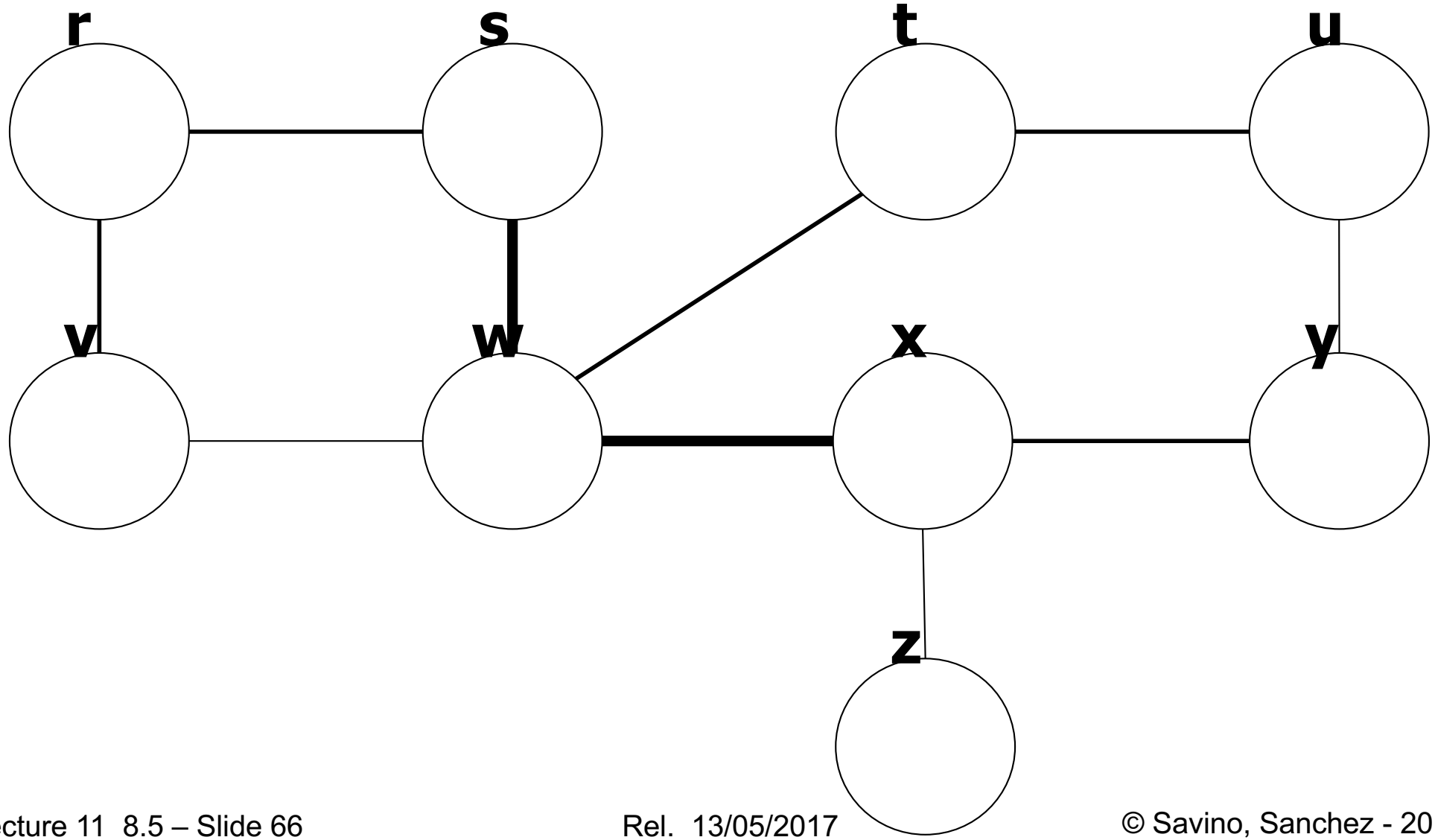
Exa

DFS and time stamps



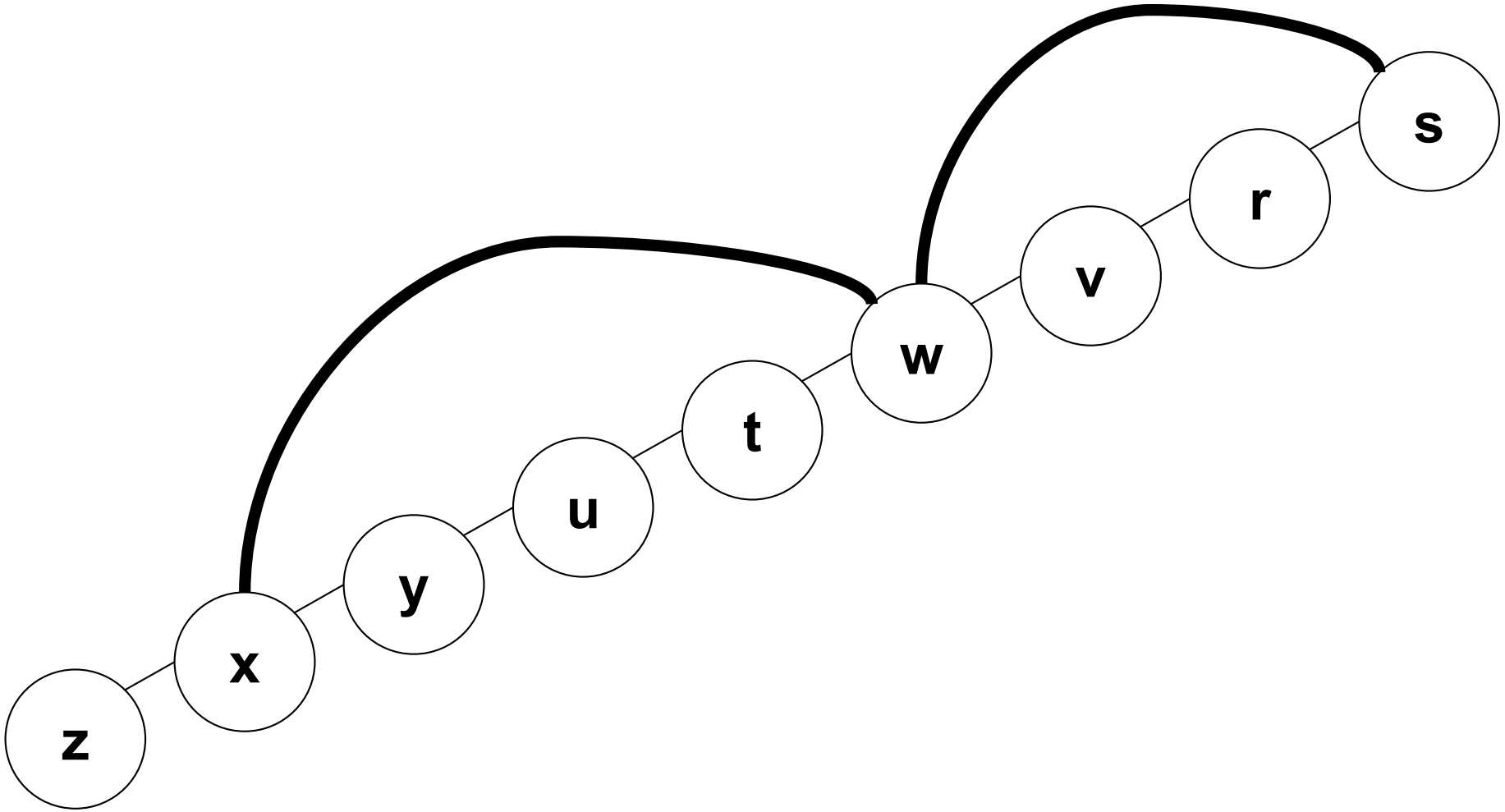
Exa

Edges classification



Exa

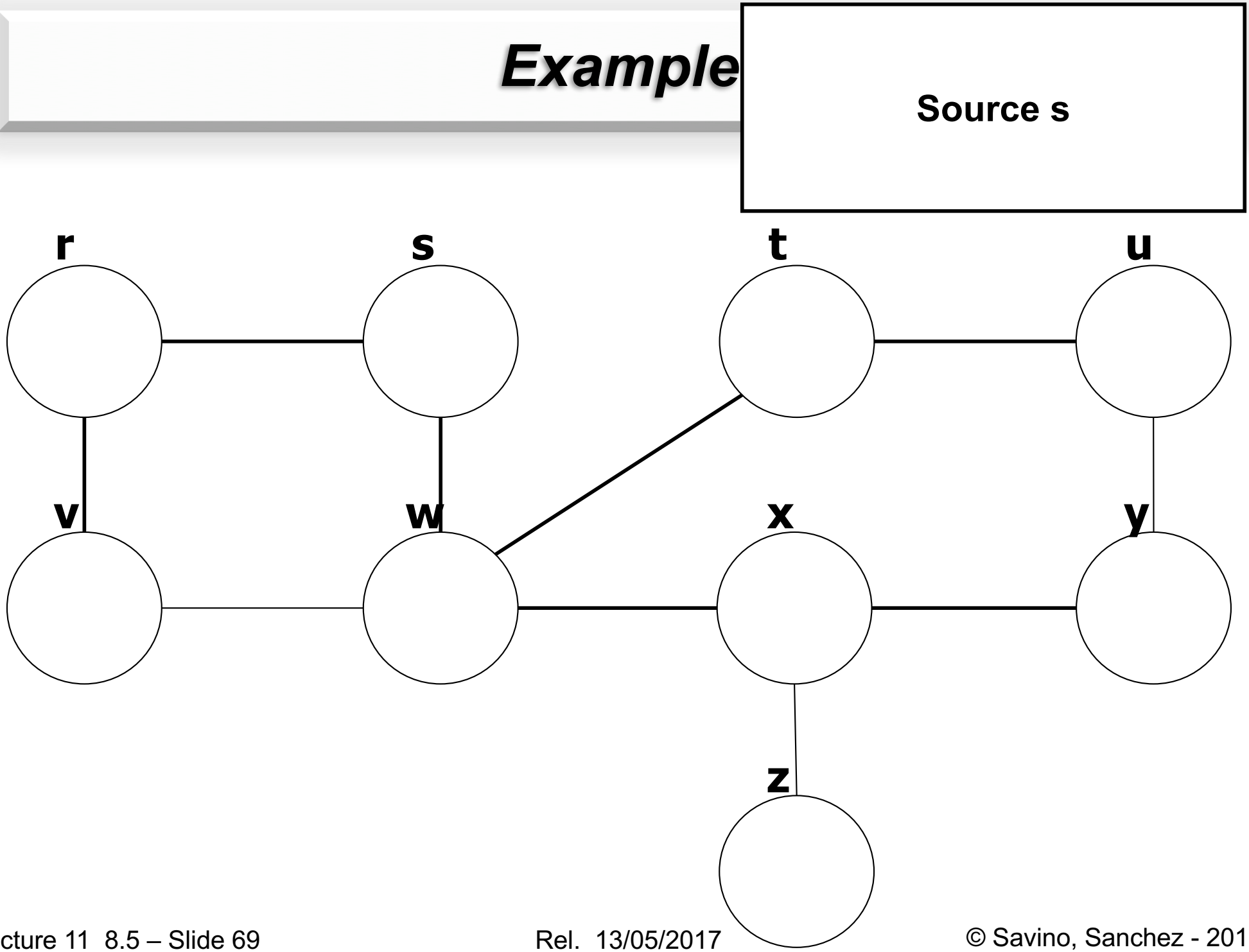
If there are B edges, the graph has cycles



Finding the cut vertices

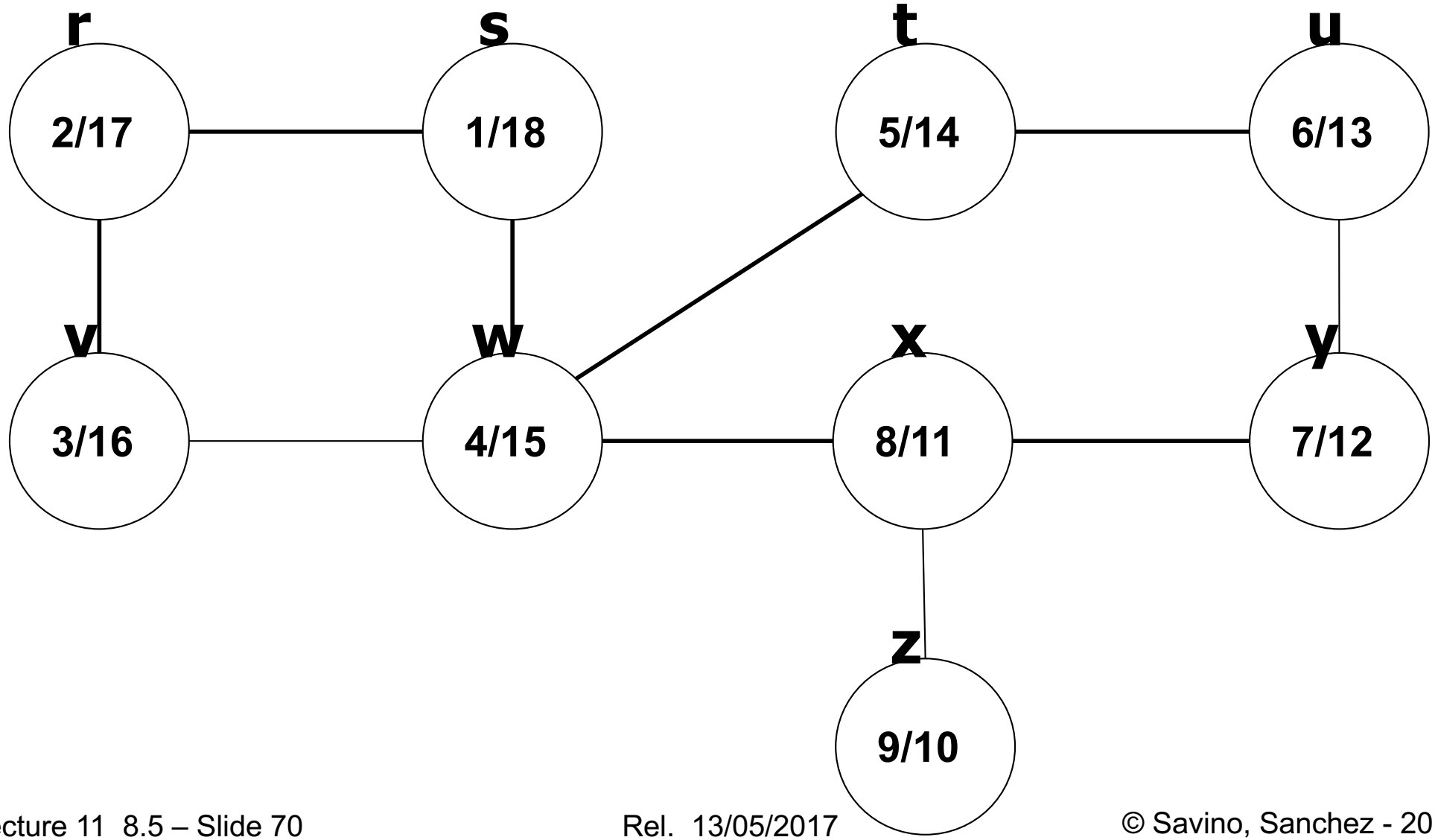
- **A DFS is used to classify each edge.**
- **A vertex v is a cut vertex IFF v has a child s so that there are no B edges from s or from any of its descendants to a predecessor of v.**

Example



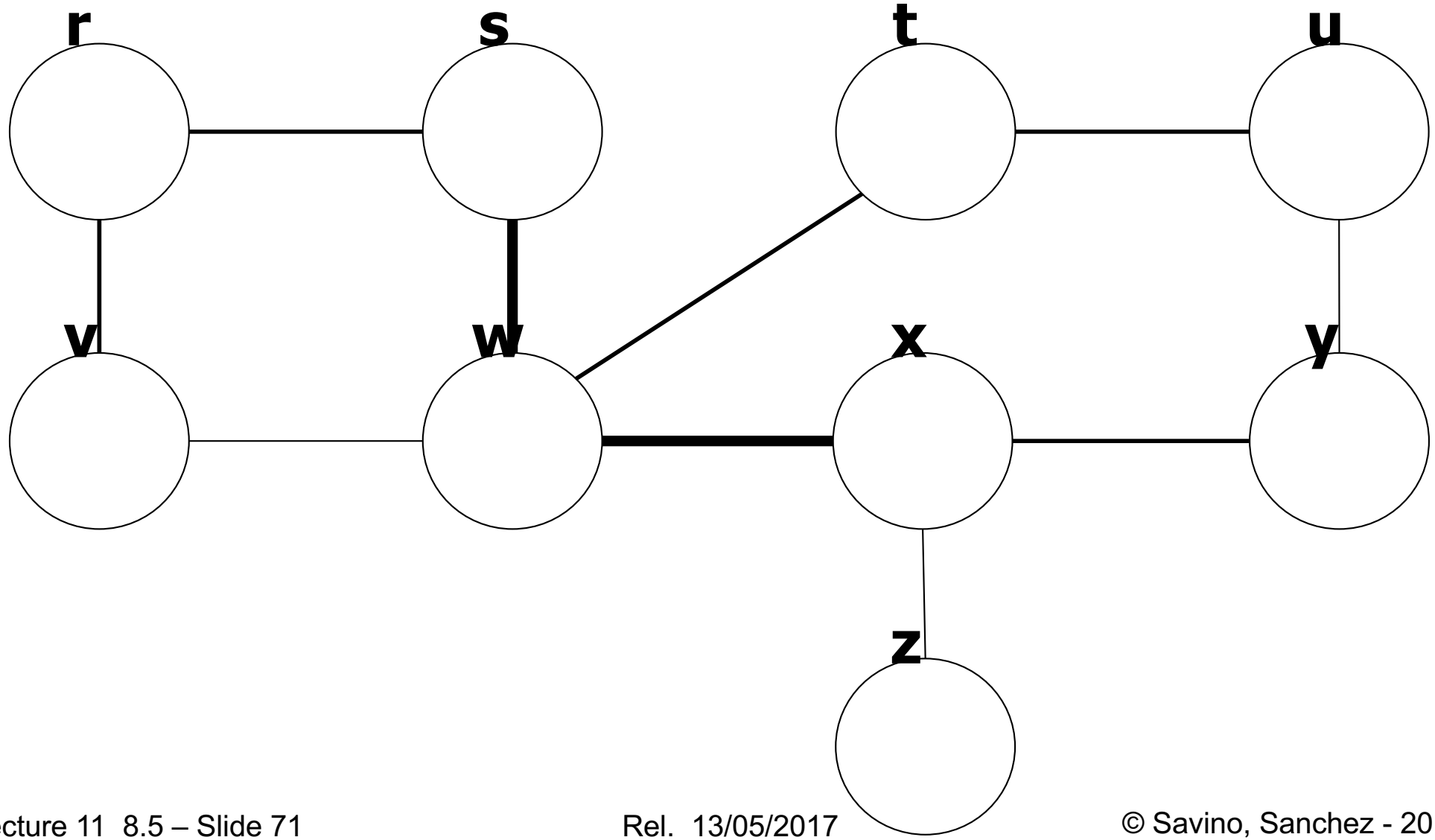
Exa

DFS and time stamps



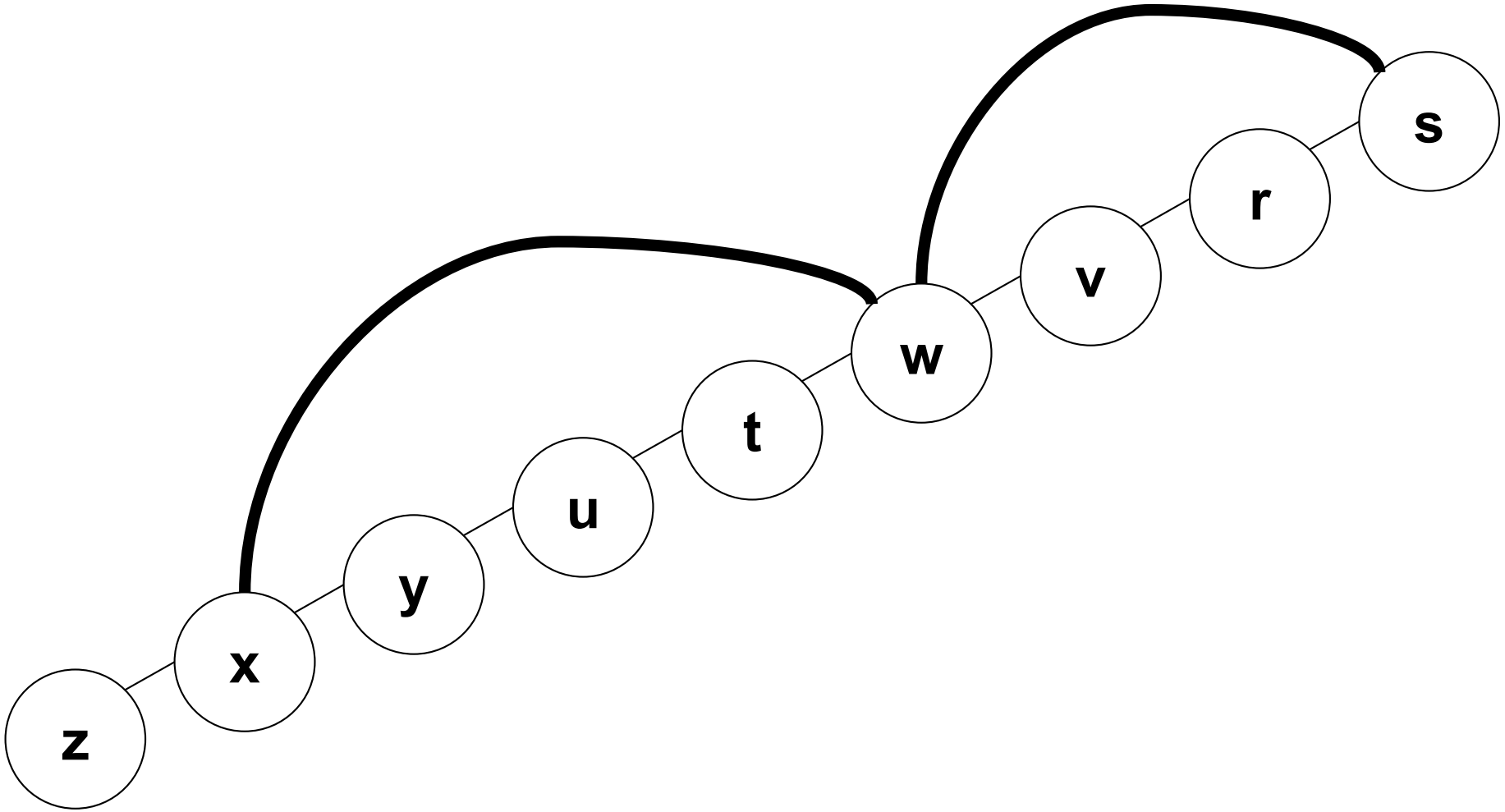
Exa

Edges classification



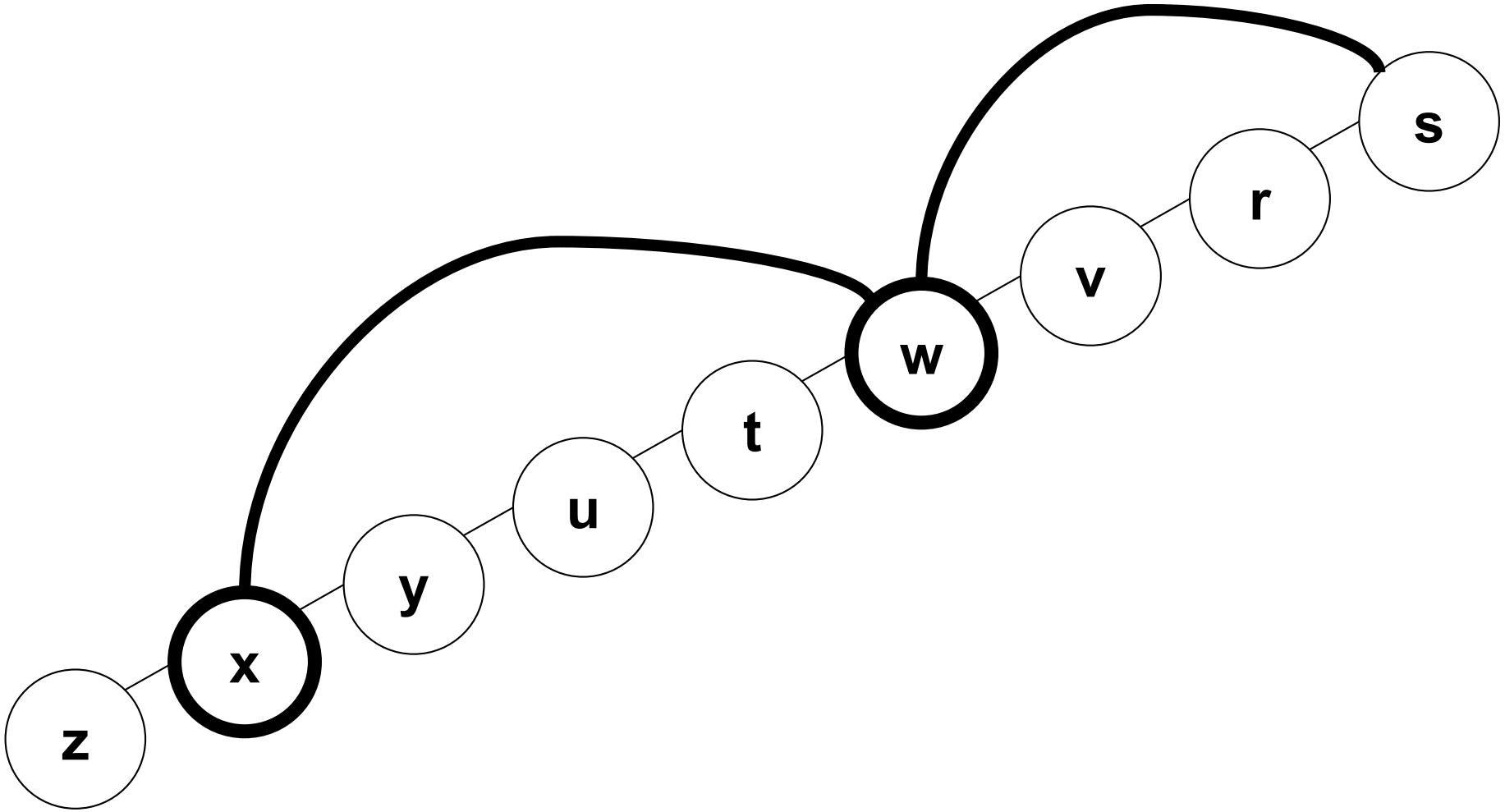
Exa

Search tree



Exa

Cut vertices identification

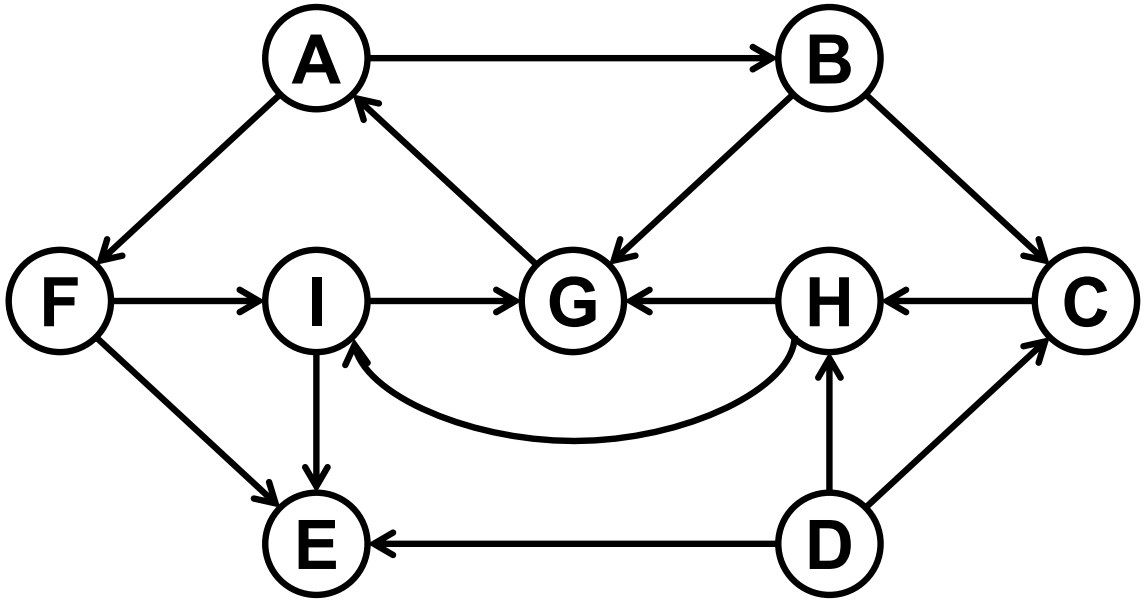


Note

- **The source vertex is a cut vertex IFF at least one of its children does not belong to any of the subtrees starting from its other adjacent nodes.**

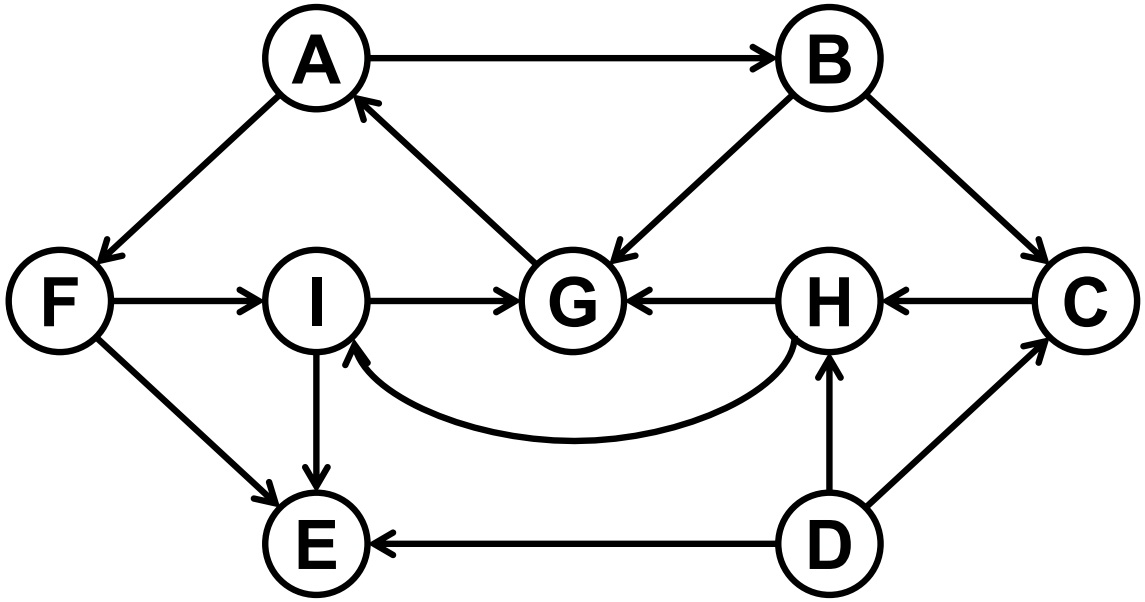
BFS and DFS exercise

- **Perform BFS starting from A:**
  ???
- **Perform DFS starting from A:**
  ???



BFS and DFS solution

- **Perform BFS starting from A:**
  A – B – F – C – G – E – I – H
- **Perform DFS starting from A:**
  A – B – C – H – G – I – E – F // D



References

- **A.V. Aho, J.E. Hopcroft, J.D. Ullman: "Data Structures and Algorithms," Addison Wesley, Reading MA (USA), 1983 pp. 198-252**
- **G. Ausiello, A. Marchetti-Spaccamela, M. Protasi: "Teoria e Progetto di Algoritmi Fondamentali," Ed. Franco Angeli, Milano, 1985, pp. 265-364**
- **E. Horowitz, S. Sahni: "Fundamentals of Computer Algorithms," Pittman, London (UK), 1978, pp. 272-325**

References

- **C.L. Liu:**
  "Introduction to Combinatorial Mathematics,"
  McGraw-Hill Book Company, New York (USA),
  pp. 167-297
- **R. Sedgewick:**
  "Algorithms in C,"
  Addison Wesley, Reading MA (USA), 1990
  pp. 415-508
- **C.J. Van Wyk:**
  "Data Structures and C Programs," Addison Wesley, Reading MA (USA), 1988
  pp. 313-341

References

- **R.J. Wilson:**
  “Introduzione alla teoria dei grafi,”
  Cremonese, Roma 1978, pp. 1-155


Малые Автюхи, Калинковичский район, Республики Беларусь