Laboratory 10: Graphs Traversals

CSC205A Data structures and Algorithms Laboratory B. Tech. 2015

Vaishali R Kulkarni

Department of Computer Science and Engineering
Faculty of Engineering and Technology
M. S. Ramaiah University of Applied Sciences
Email: vaishali.cs.et@msruas.ac.in
Tel: +91-80-4906-5555 (2212) WWW: www.msruas.ac.in

© Ramaiah University of Applied Sciences

Introduction and Purpose of Experiment

- Graph algorithms such Depth First Search (DFS) and Breadth First Search (BFS) are very important in many graph applications which are used to traverse the graphs, check for graph connectivity and to find spanning trees etc.
- This experiment introduces the applications DFS and BFS.

© Ramaiah University of Applied Sciences

Aim and objectives

Aim:

- To design and develop C program to traverse the given graph using Depth First Search(DFS) and Breadth First Search(BFS) techniques

Objectives:

At the end of this lab, the student will be able to

- Design and implement DFS traversal of graph
- Design and implement BFS traversal of graph
- Compare the efficiency of both
- Apply DFS and BFS for graph applications such as graph connectivity check

© Ramaiah University of Applied Sciences

Exploring a Labyrinth Without Getting Lost

- A **depth-first search (DFS)** in an undirected graph G is like wandering in a labyrinth with a string and a can of red paint without getting lost.
- We start at vertex ***s***, tying the end of our string to the point and painting ***s*** "visited". Next we label ***s*** as our current vertex called ***u***.
- Now we travel along an arbitrary edge (***u***, ***v***).
- If edge (***u***, ***v***) leads us to an already visited vertex ***v*** we return to ***u***.
- If vertex ***v*** is unvisited, we unroll our string and move to ***v***, paint ***v*** "visited", set ***v*** as our current vertex, and repeat the previous steps.

© Ramaiah University of Applied Sciences

Review: Graph Searching

- Given: a graph G = (V, E), directed or undirected
- Goal: methodically explore every vertex and every edge
- Ultimately: 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

© Ramaiah University of Applied Sciences

Review: 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.

© Ramaiah University of Applied Sciences

Review: Breadth-First Search

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

© Ramaiah University of Applied Sciences

Review: Breadth-First Search

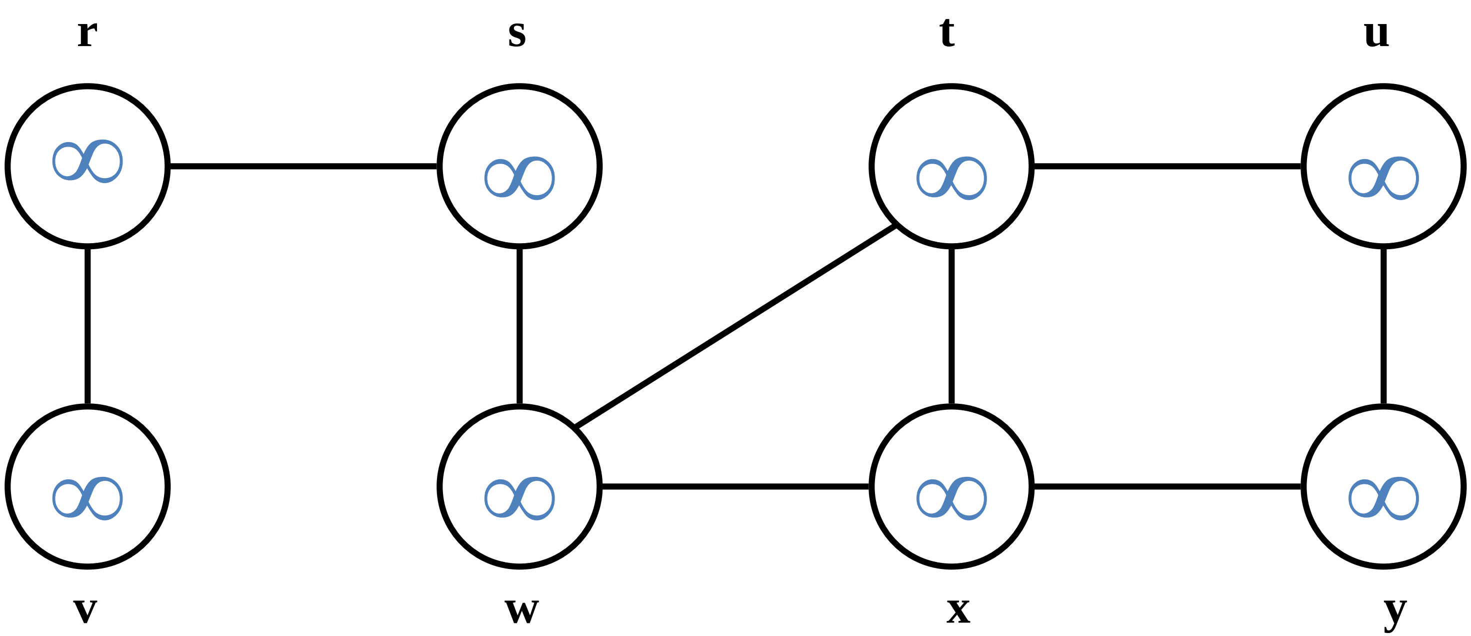
```
BFS(G, s) {
    initialize vertices;
    Q = {s};        // Q is a queue (duh); initialize to s
    while (Q not empty) {
        u = RemoveTop(Q);
        for each v ∈ u->adj {
            if (v->color == WHITE)
                v->color = GREY;
                v->d = u->d + 1;
                v->p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

What does `v->d` represent?

What does `v->p` represent?

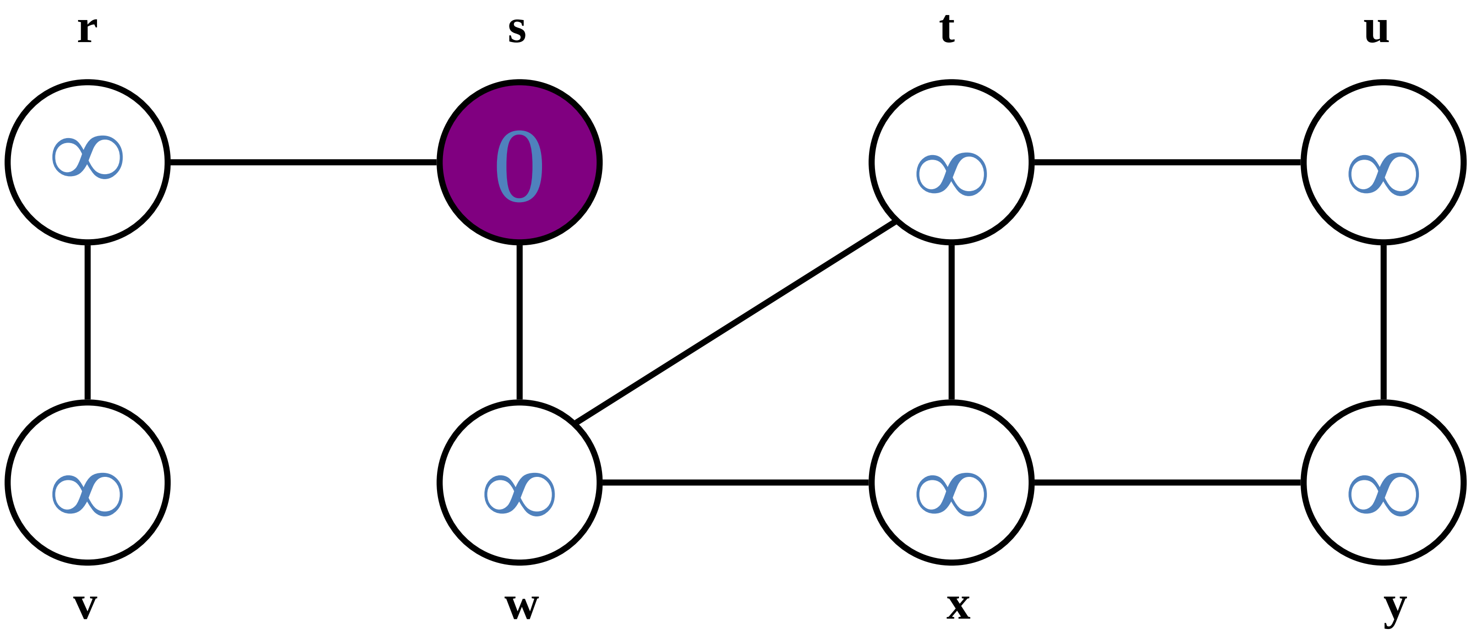
© Ramaiah University of Applied Sciences

Breadth-First Search: Example



© Ramaiah University of Applied Sciences

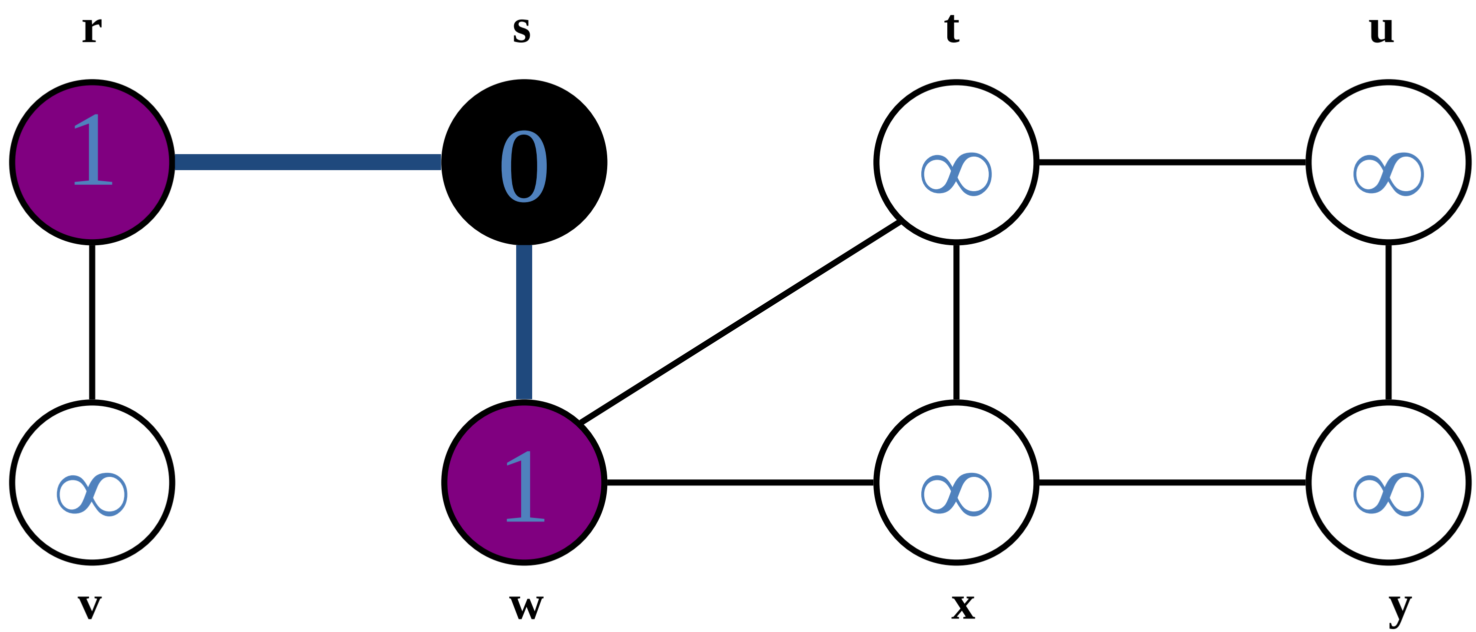
Breadth-First Search: Example



Q: $\boxed{s}$

© Ramaiah University of Applied Sciences

Breadth-First Search: Example

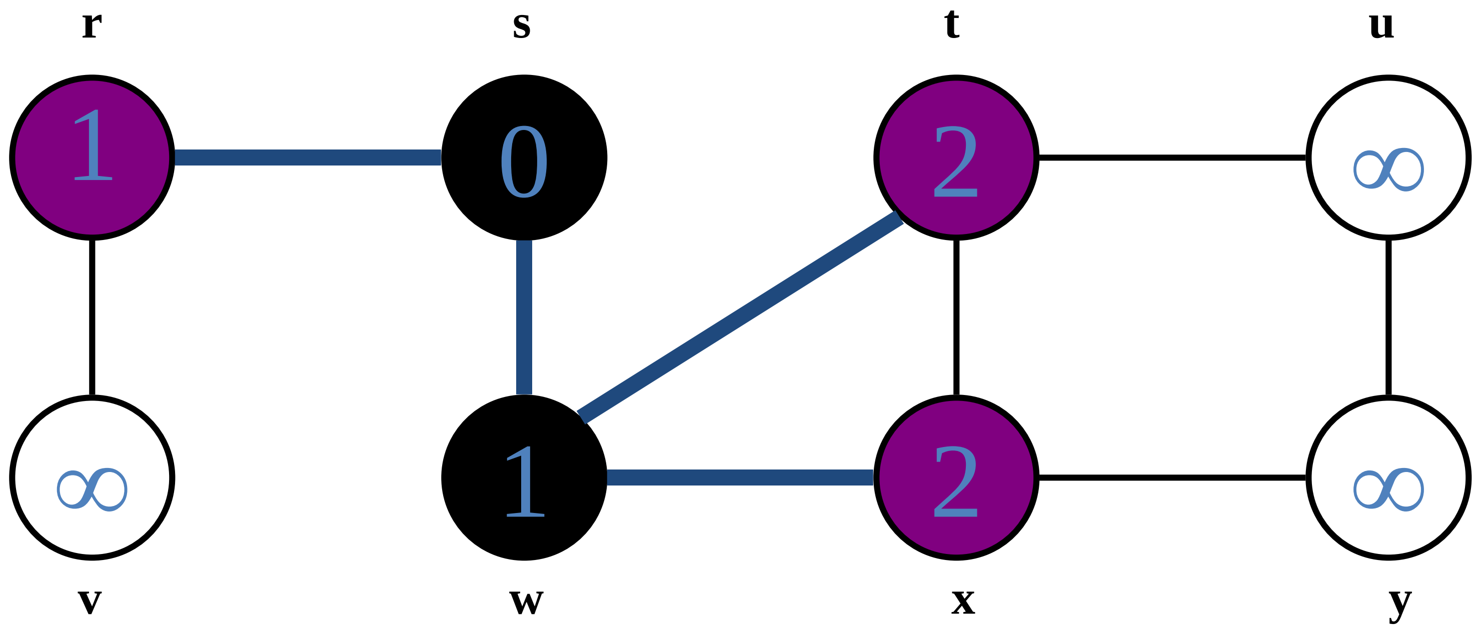


Q:

w	r

© Ramaiah University of Applied Sciences

Breadth-First Search: Example

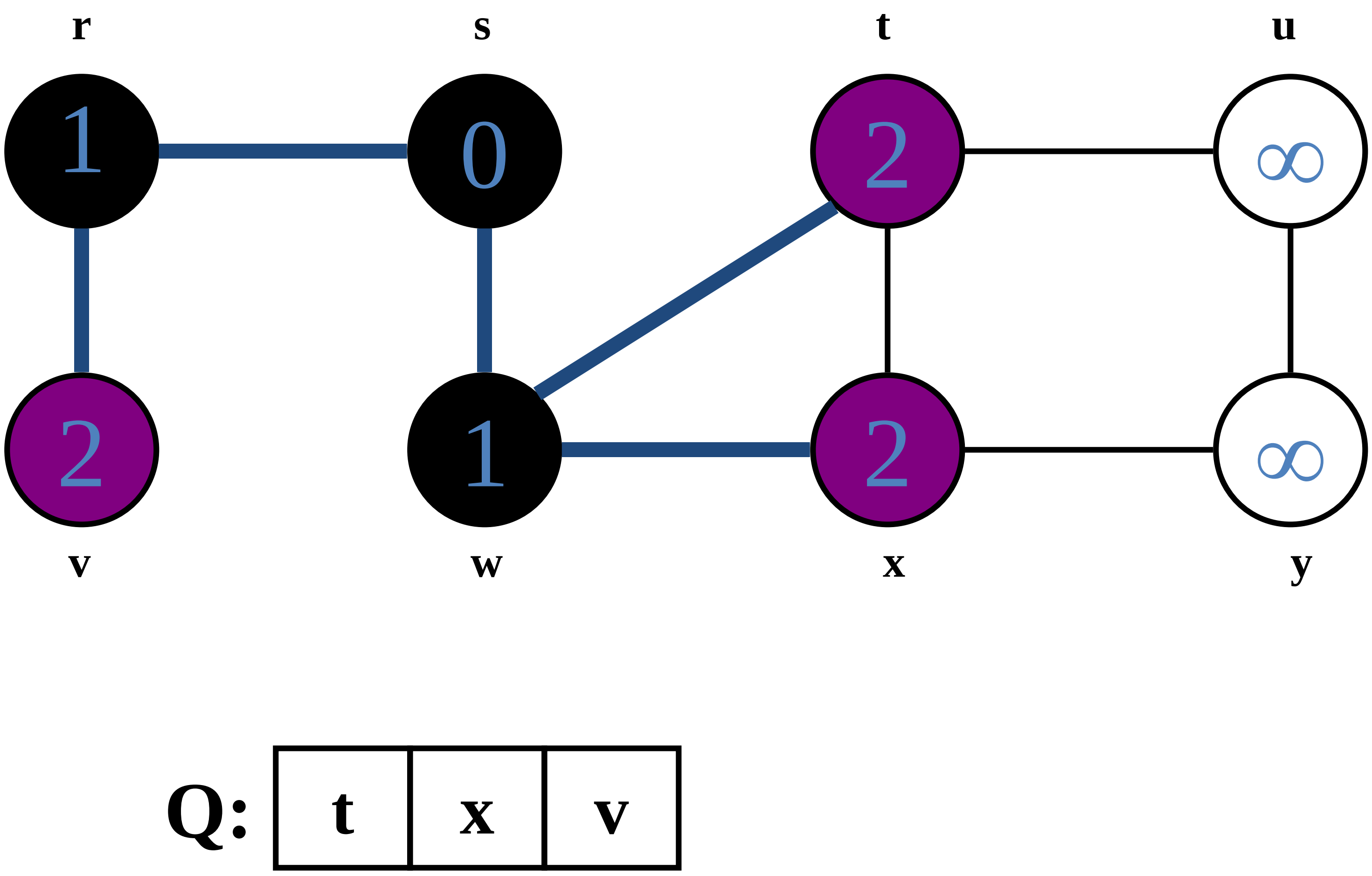


Q:

r	**t**	**x**

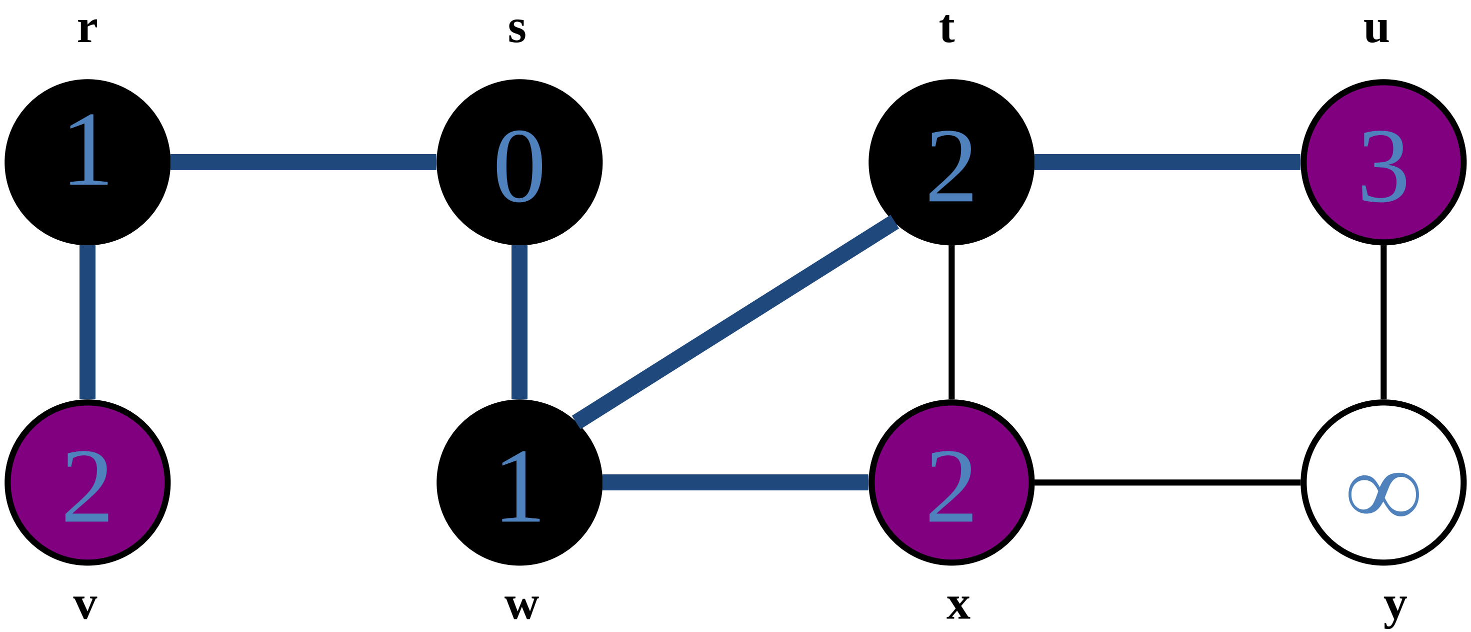
© Ramaiah University of Applied Sciences

Breadth-First Search: Example



© Ramaiah University of Applied Sciences

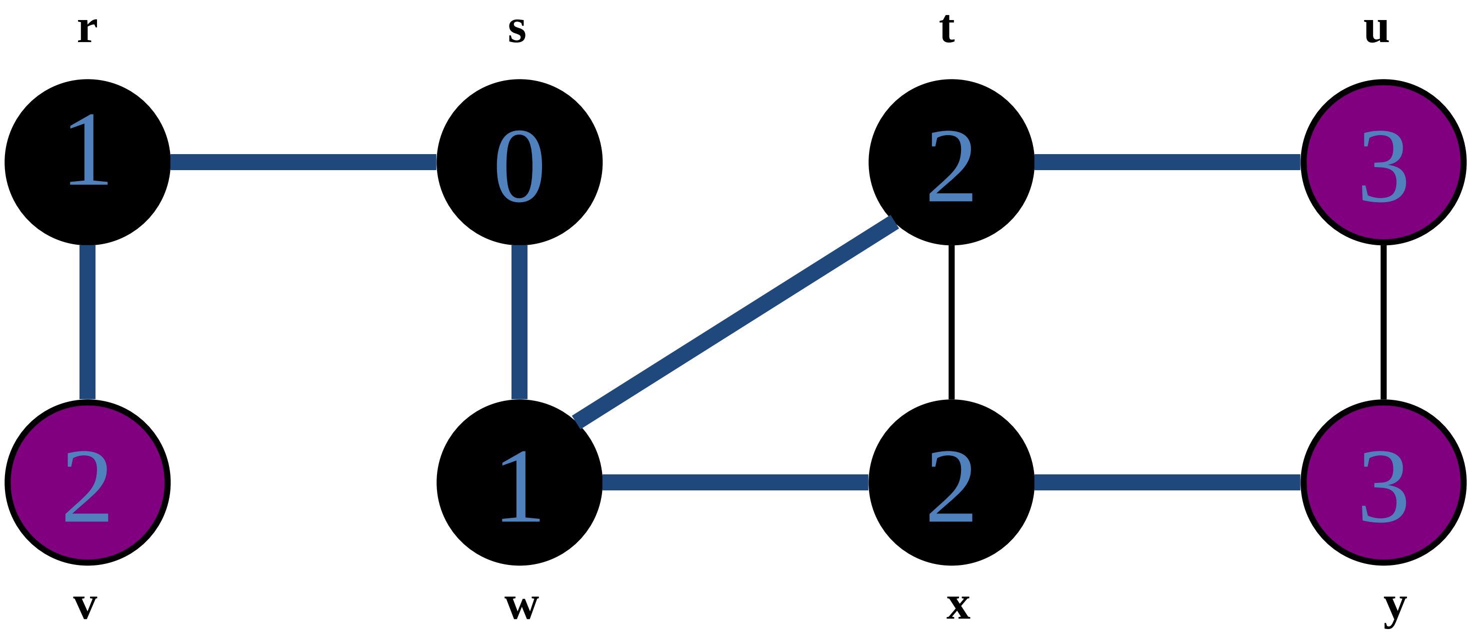
Breadth-First Search: Example



Q:	x	v	u

© Ramaiah University of Applied Sciences

Breadth-First Search: Example

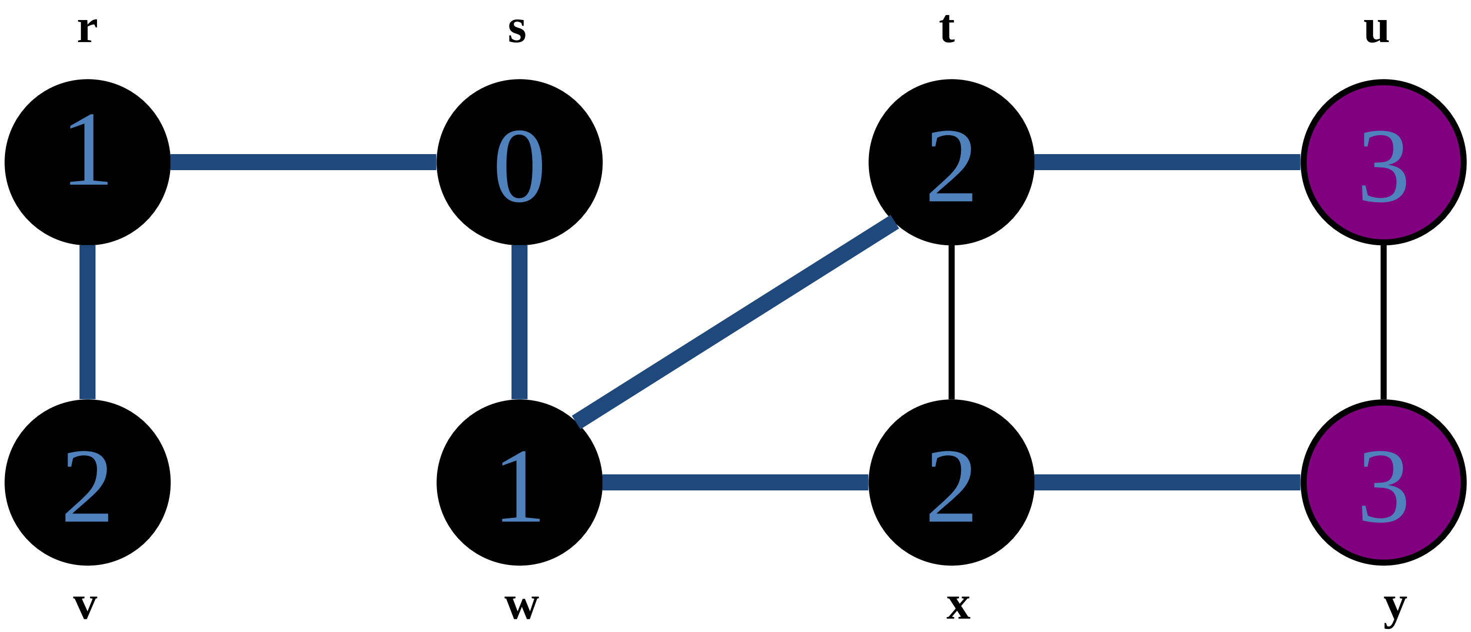


Q:

v	u	y

© Ramaiah University of Applied Sciences

Breadth-First Search: Example

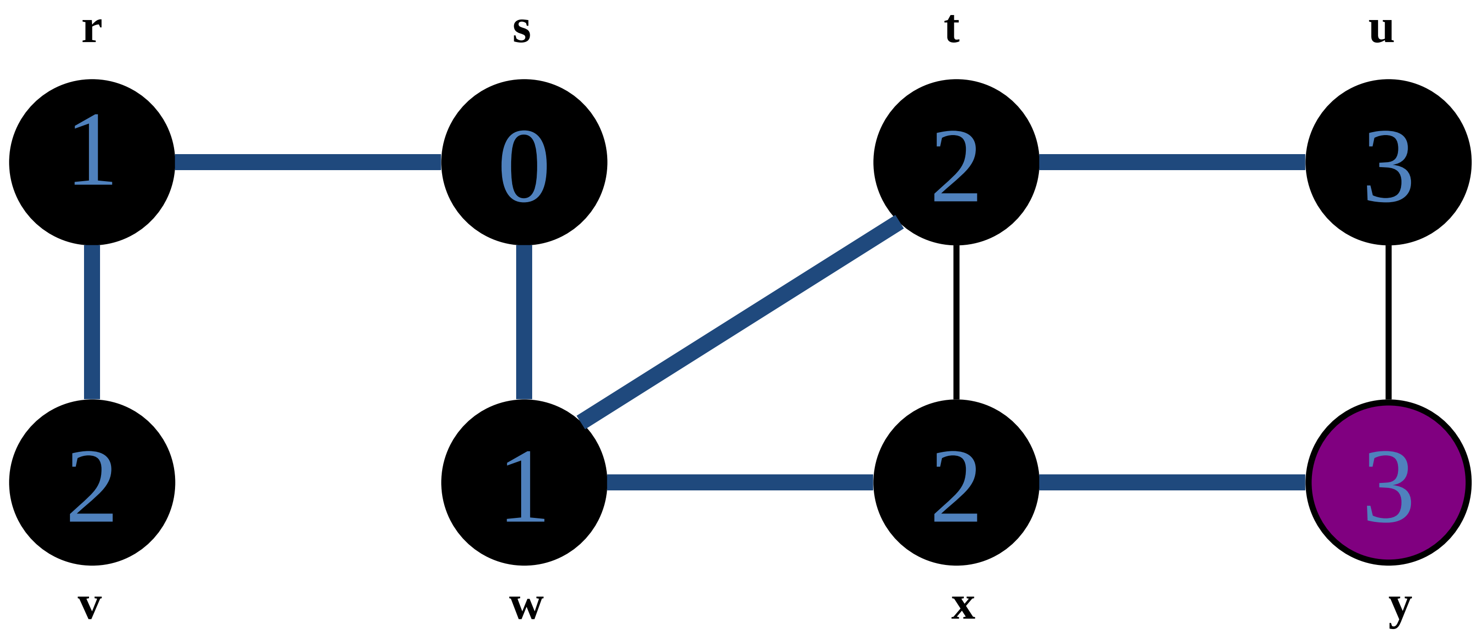


Q:

u	y

© Ramaiah University of Applied Sciences

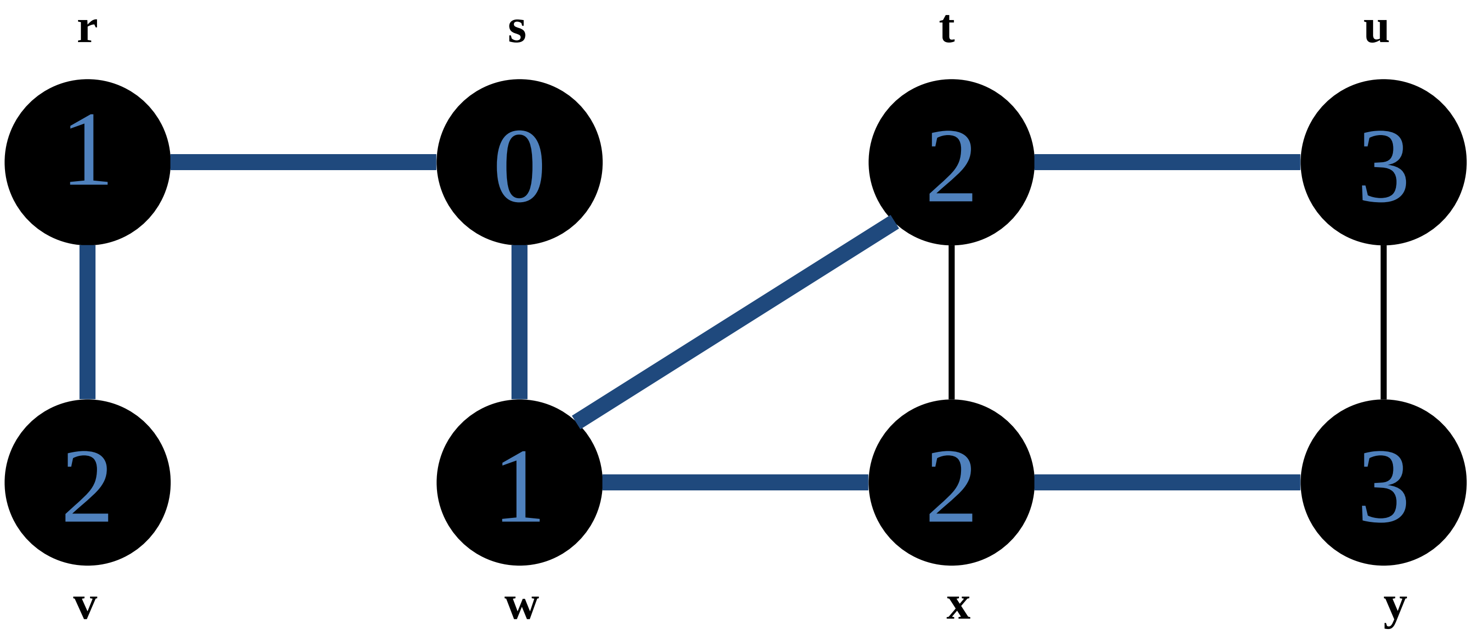
Breadth-First Search: Example



Q: y

© Ramaiah University of Applied Sciences

Breadth-First Search: Example



Q: Ø

© Ramaiah University of Applied Sciences

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

(Why?)

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

What will be the running time?

Total running time: O(V+E)

© Ramaiah University of Applied Sciences

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;
    }
}
```

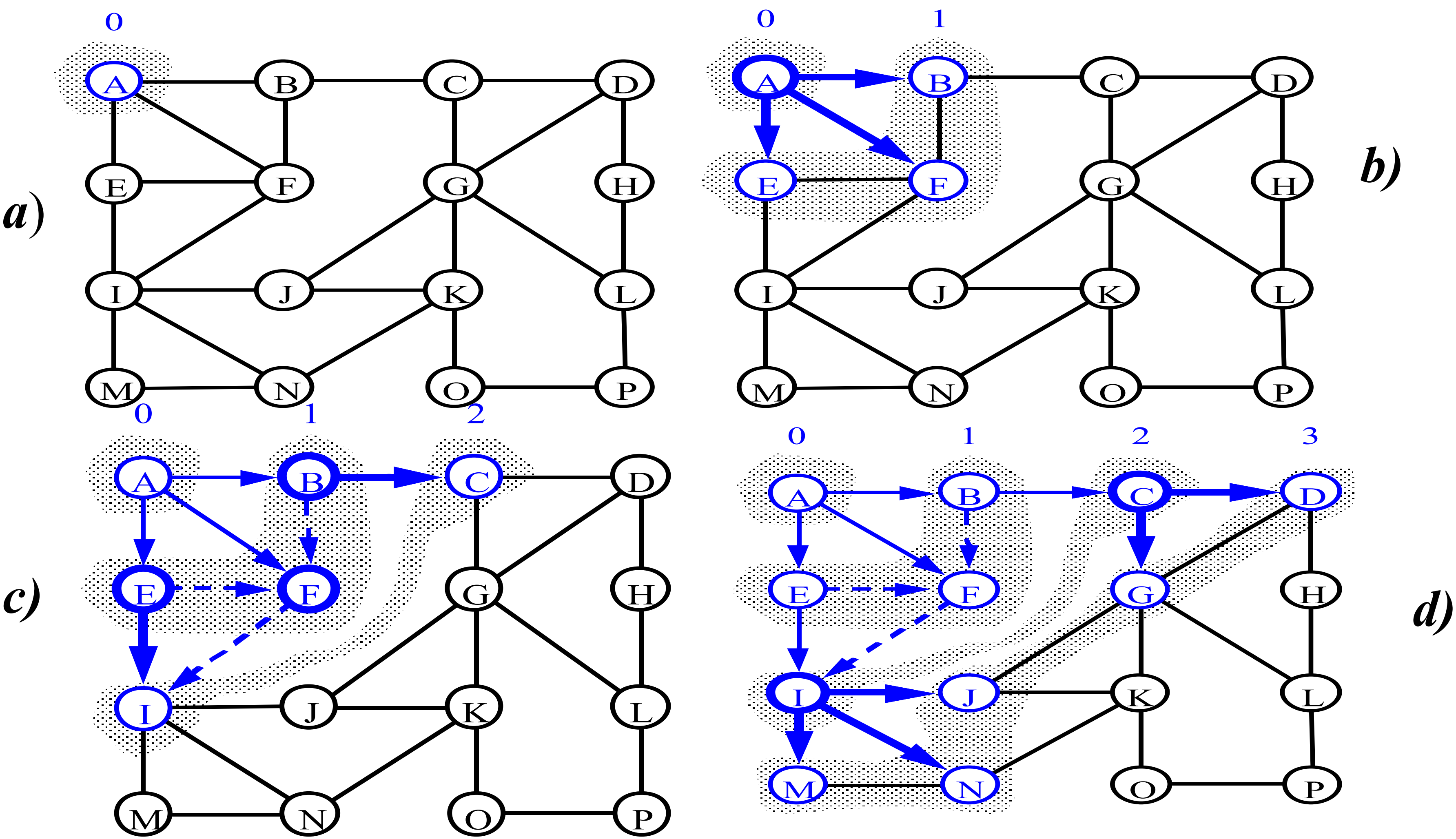
What will be the storage cost in addition to storing the graph?

Total space used:

O(max(degree(v))) = O(E)

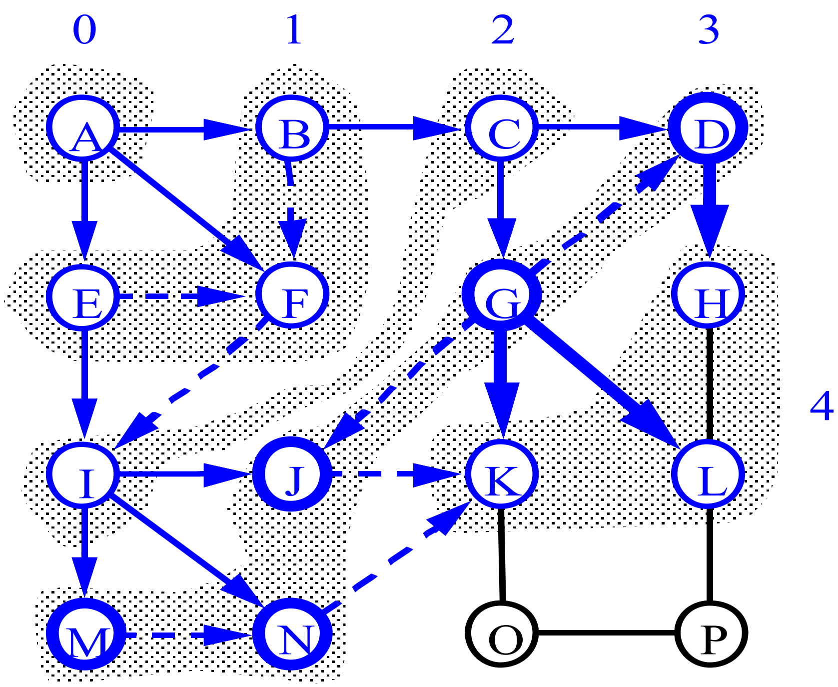
© Ramaiah University of Applied Sciences

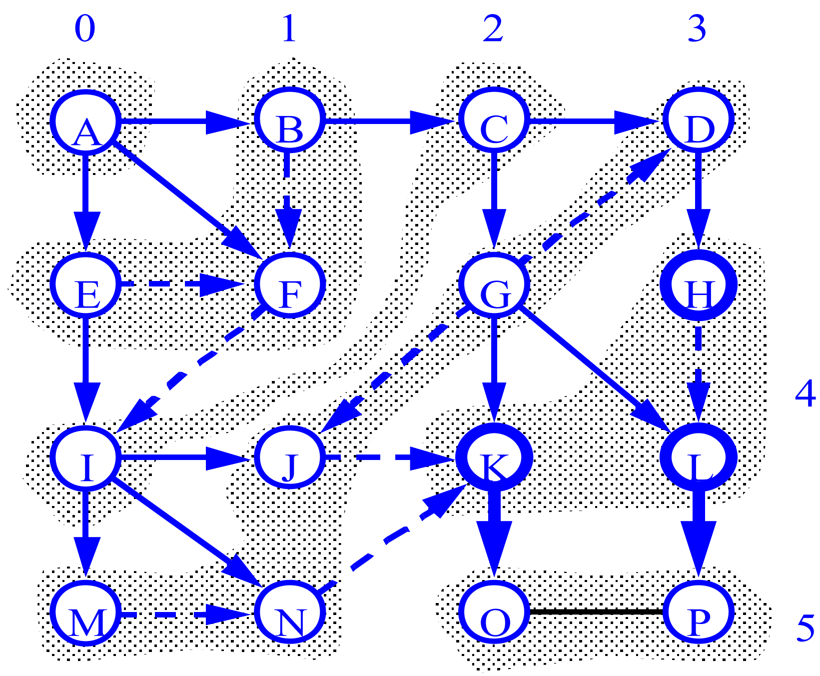
BFS - A Graphical Representation



© Ramaiah University of Applied Sciences

More BFS





© Ramaiah University of Applied Sciences

Breadth-First Search: 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 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

© Ramaiah University of Applied Sciences

Depth-First Search

- *Depth-first search* is another strategy for exploring a graph
  - Explore "deeper" in the graph whenever possible
  - Edges are explored out of the most recently discovered vertex v that still has unexplored edges
  - When all of v's edges have been explored, backtrack to the vertex from which v was discovered

© Ramaiah University of Applied Sciences

Depth-First Search

- Vertices initially colored white
- Then colored gray when discovered
- Then black when finished

© Ramaiah University of Applied Sciences

Depth-First Search: The Code

```
DFS(G)
{
    for each vertex u ∈ G->V
    {
        u->color = WHITE;
    }
    time = 0;
    for each vertex u ∈ G->V
    {
        if (u->color == WHITE)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    u->color = GREY;
    time = time+1;
    u->d = time;
    for each v ∈ u->Adj[]
    {
        if (v->color == WHITE)
            DFS_Visit(v);
    }
    u->color = BLACK;
    time = time+1;
    u->f = time;
}
```


© Ramaiah University of Applied Sciences

Depth-First Search: The Code

```
DFS(G)
{
    for each vertex u ∈ G->V
    {
        u->color = WHITE;
    }
    time = 0;
    for each vertex u ∈ G->V
    {
        if (u->color == WHITE)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    u->color = GREY;
    time = time+1;
    u->d = time;
    for each v ∈ u->Adj[]
    {
        if (v->color == WHITE)
            DFS_Visit(v);
    }
    u->color = BLACK;
    time = time+1;
    u->f = time;
}
```

What does u->d represent?

© Ramaiah University of Applied Sciences

Depth-First Search: The Code

```
DFS(G)
{
    for each vertex u ∈ G->V
    {
        u->color = WHITE;
    }
    time = 0;
    for each vertex u ∈ G->V
    {
        if (u->color == WHITE)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    u->color = GREY;
    time = time+1;
    u->d = time;
    for each v ∈ u->Adj[]
    {
        if (v->color == WHITE)
            DFS_Visit(v);
    }
    u->color = BLACK;
    time = time+1;
    u->f = time;
}
```

What does u->f represent?

© Ramaiah University of Applied Sciences

Depth-First Search: The Code

```
DFS(G)
{
    for each vertex u ∈ G->V
    {
        u->color = WHITE;
    }
    time = 0;
    for each vertex u ∈ G->V
    {
        if (u->color == WHITE)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    u->color = GREY;
    time = time+1;
    u->d = time;
    for each v ∈ u->Adj[]
    {
        if (v->color == WHITE)
            DFS_Visit(v);
    }
    u->color = BLACK;
    time = time+1;
    u->f = time;
}
```

Will all vertices eventually be colored black?

© Ramaiah University of Applied Sciences

Depth-First Search: The Code

```
DFS(G)
{
    for each vertex u ∈ G->V
    {
        u->color = WHITE;
    }
    time = 0;
    for each vertex u ∈ G->V
    {
        if (u->color == WHITE)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    u->color = GREY;
    time = time+1;
    u->d = time;
    for each v ∈ u->Adj[]
    {
        if (v->color == WHITE)
            DFS_Visit(v);
    }
    u->color = BLACK;
    time = time+1;
    u->f = time;
}
```

What will be the running time?

© Ramaiah University of Applied Sciences

Depth-First Search: The Code

```
DFS(G)
{
    for each vertex u ∈ G->V
    {
        u->color = WHITE;
    }
    time = 0;
    for each vertex u ∈ G->V
    {
        if (u->color == WHITE)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    u->color = GREY;
    time = time+1;
    u->d = time;
    for each v ∈ u->Adj[]
    {
        if (v->color == WHITE)
            DFS_Visit(v);
    }
    u->color = BLACK;
    time = time+1;
    u->f = time;
}
```

Running time: O(n^2) because call DFS_Visit on each vertex, and the loop over Adj[] can run as many as |V| times

© Ramaiah University of Applied Sciences

Depth-First Search: The Code

```
DFS(G)
{
    for each vertex u ∈ G->V
    {
        u->color = WHITE;
    }
    time = 0;
    for each vertex u ∈ G->V
    {
        if (u->color == WHITE)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    u->color = GREY;
    time = time+1;
    u->d = time;
    for each v ∈ u->Adj[]
    {
        if (v->color == WHITE)
            DFS_Visit(v);
    }
    u->color = BLACK;
    time = time+1;
    u->f = time;
}
```

BUT, there is actually a tighter bound.

How many times will DFS_Visit() actually be called?

© Ramaiah University of Applied Sciences

Depth-First Search: The Code

```
DFS(G)
{
    for each vertex u ∈ G->V
    {
        u->color = WHITE;
    }
    time = 0;
    for each vertex u ∈ G->V
    {
        if (u->color == WHITE)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    u->color = GREY;
    time = time+1;
    u->d = time;
    for each v ∈ u->Adj[]
    {
        if (v->color == WHITE)
            DFS_Visit(v);
    }
    u->color = BLACK;
    time = time+1;
    u->f = time;
}
```

So, running time of DFS = O(V+E)

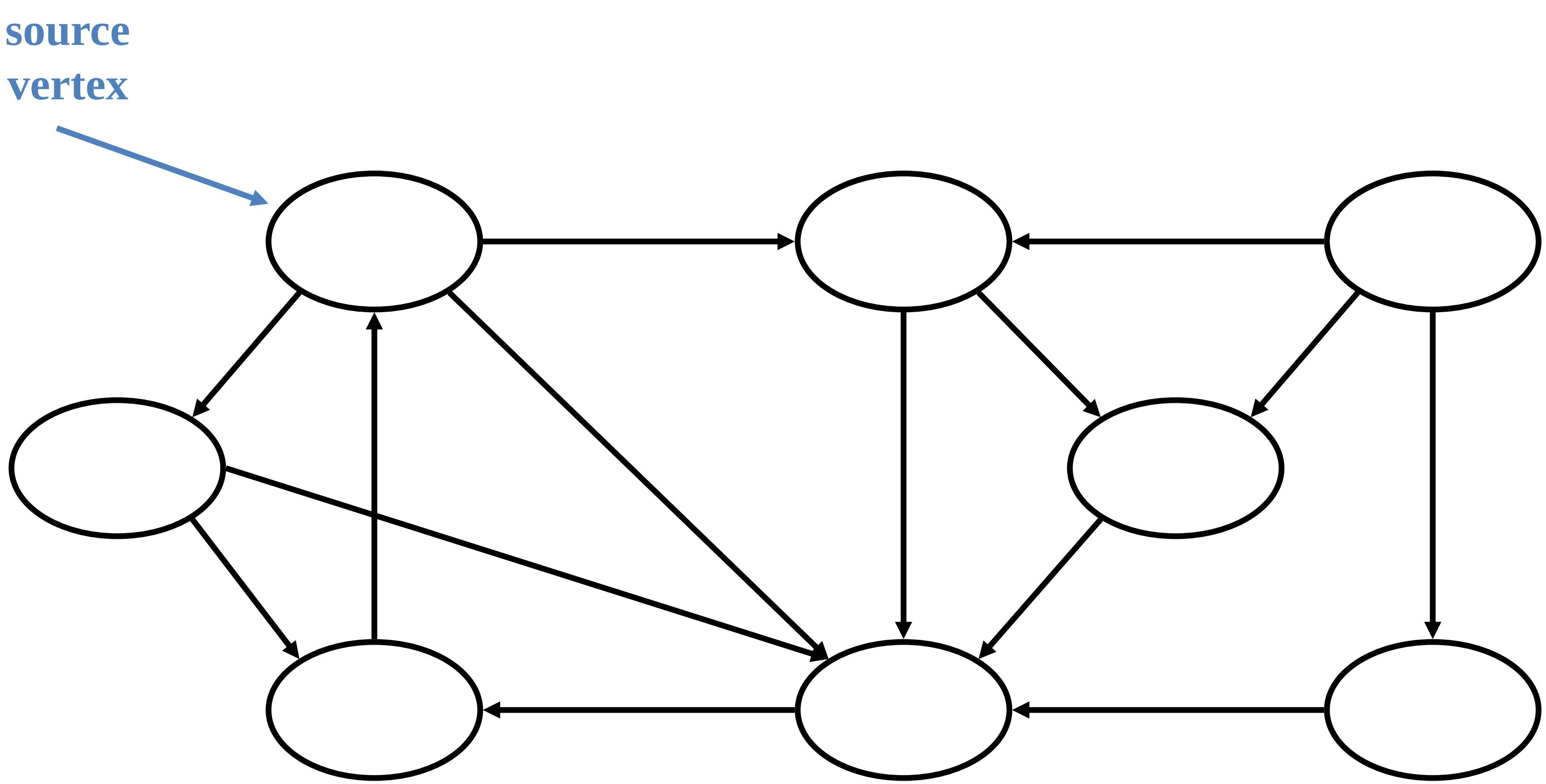
© Ramaiah University of Applied Sciences

Depth-First Sort Analysis

- This running time argument is an informal example of *amortized analysis*
  - "Charge" the exploration of edge to the edge:
    - Each loop in DFS_Visit can be attributed to an edge in the graph
    - Runs once/edge if directed graph, twice if undirected
    - Thus loop will run in O(E) time, algorithm O(V+E)
      - Considered linear for graph, b/c adj list requires O(V+E) storage
  - Important to be comfortable with this kind of reasoning and analysis

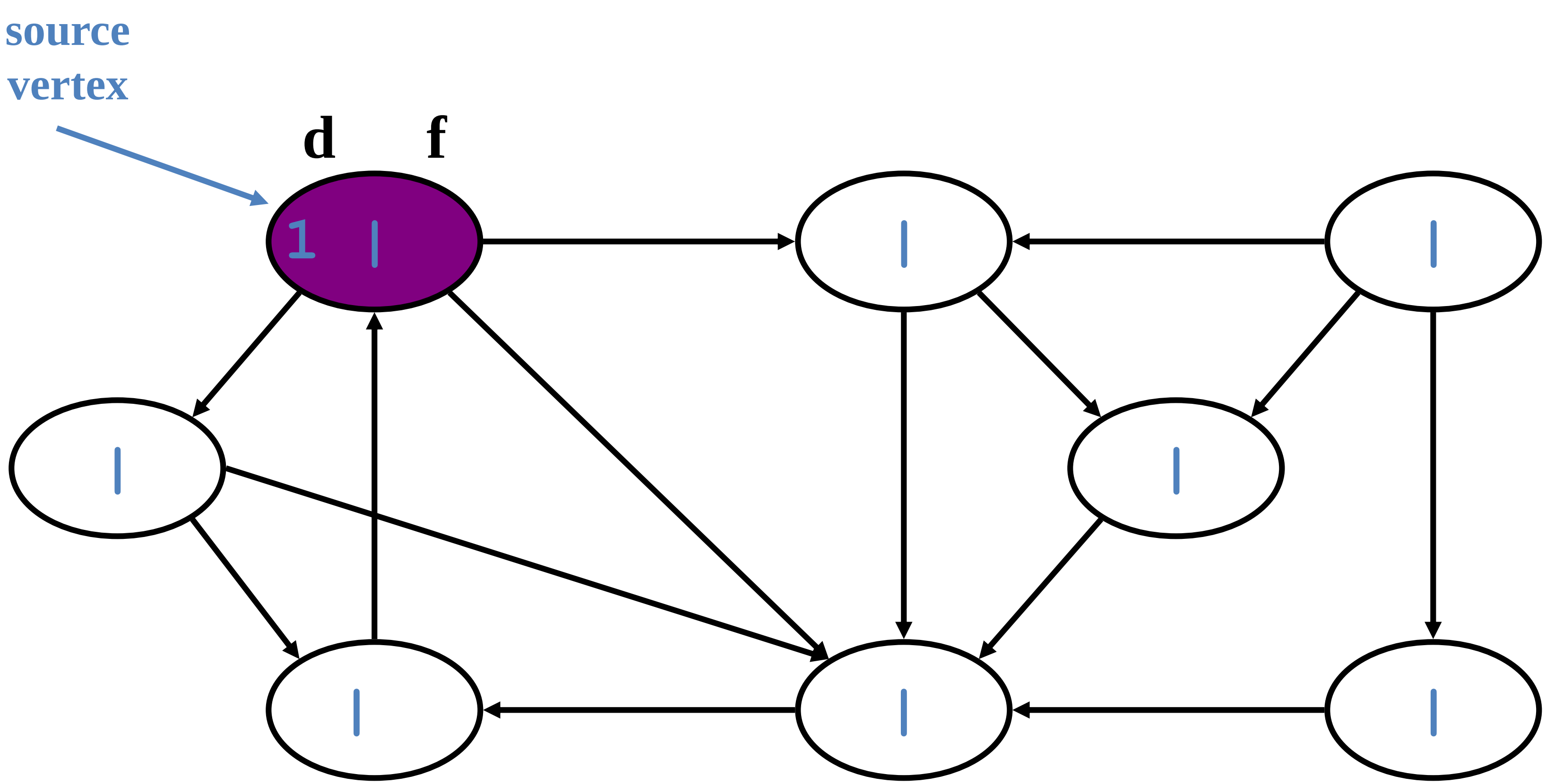
© Ramaiah University of Applied Sciences

DFS Example



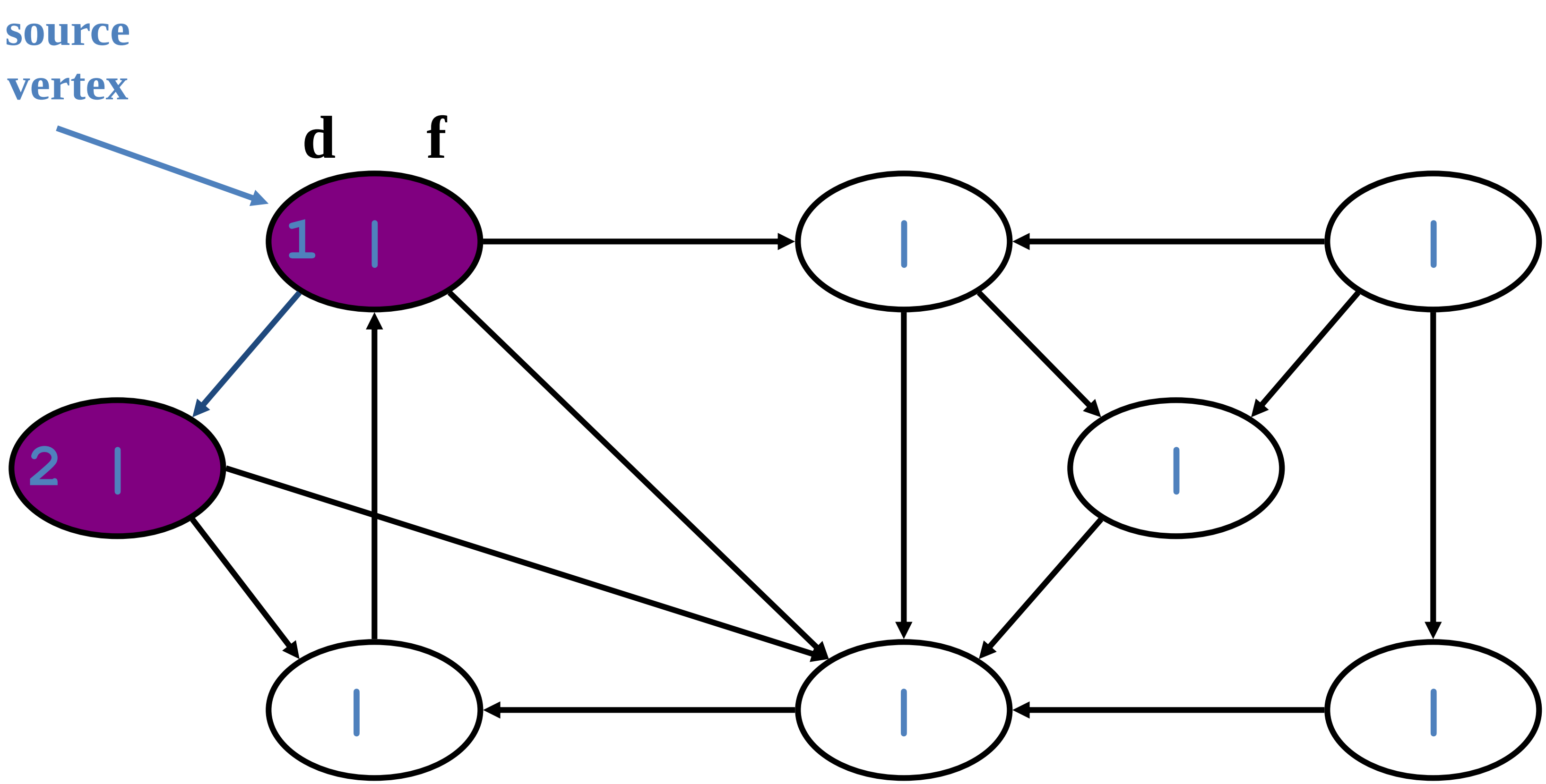
© Ramaiah University of Applied Sciences

DFS Example

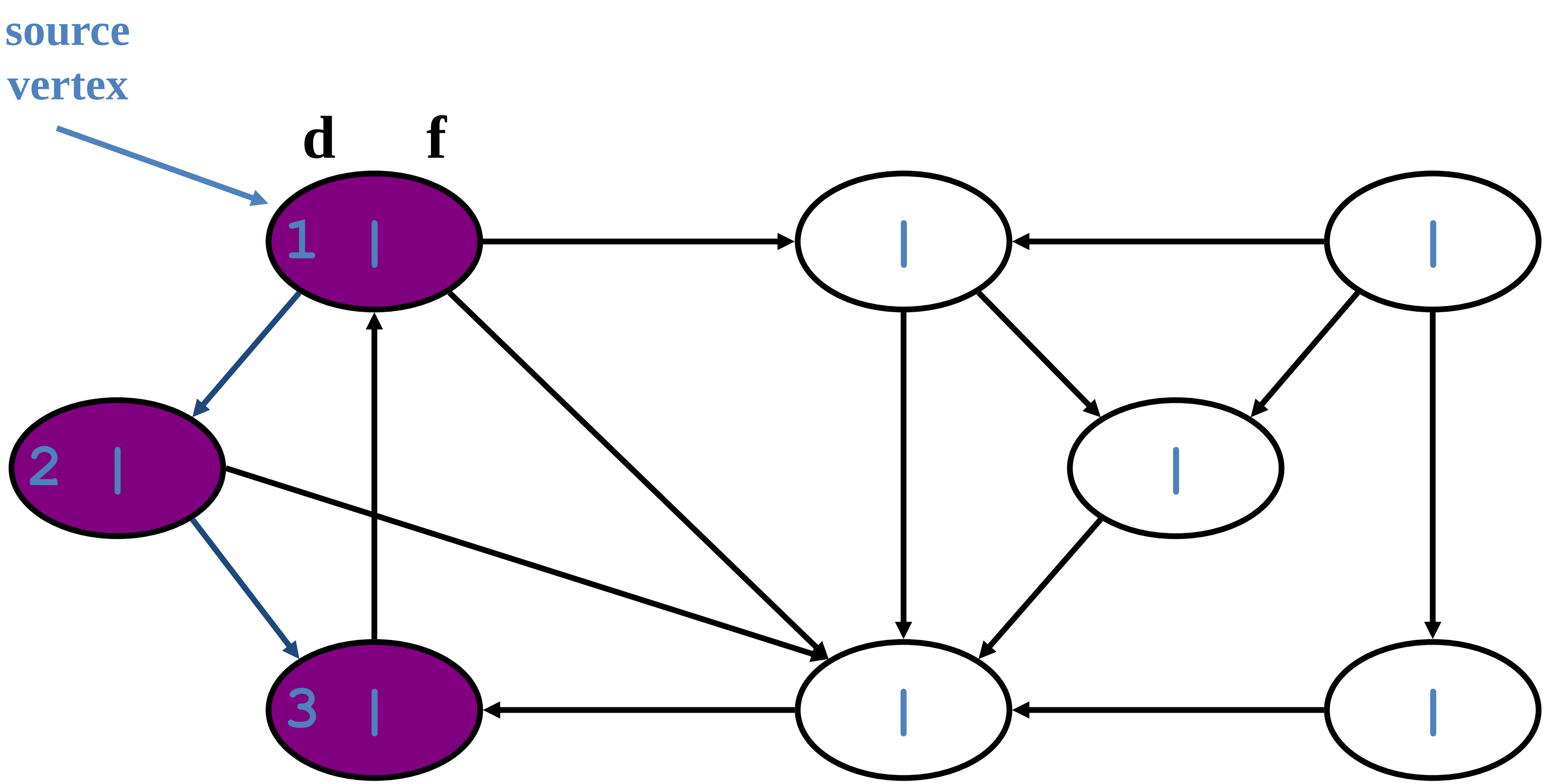


© Ramaiah University of Applied Sciences

DFS Example

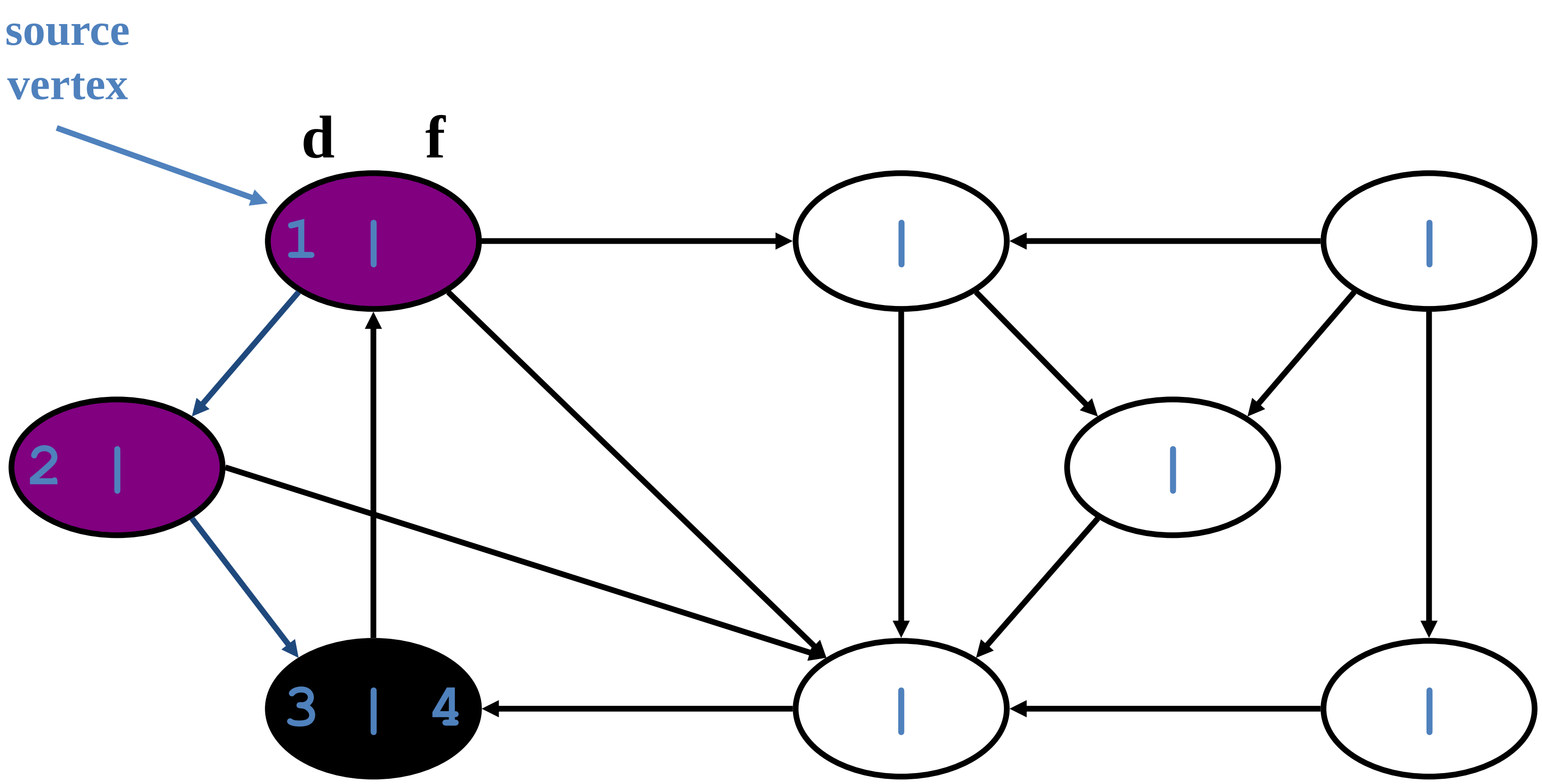


DFS Example

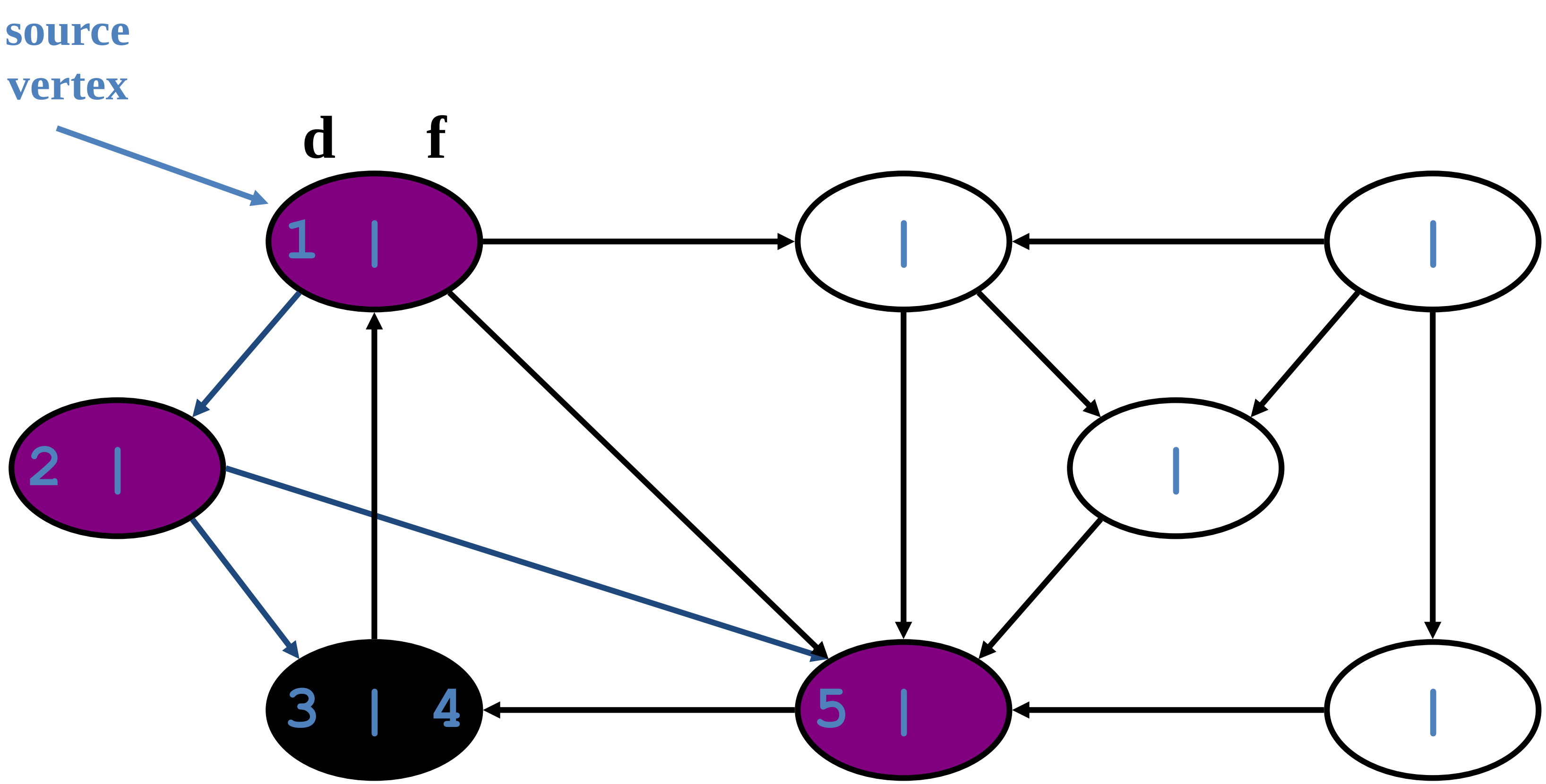


© Ramaiah University of Applied Sciences

DFS Example

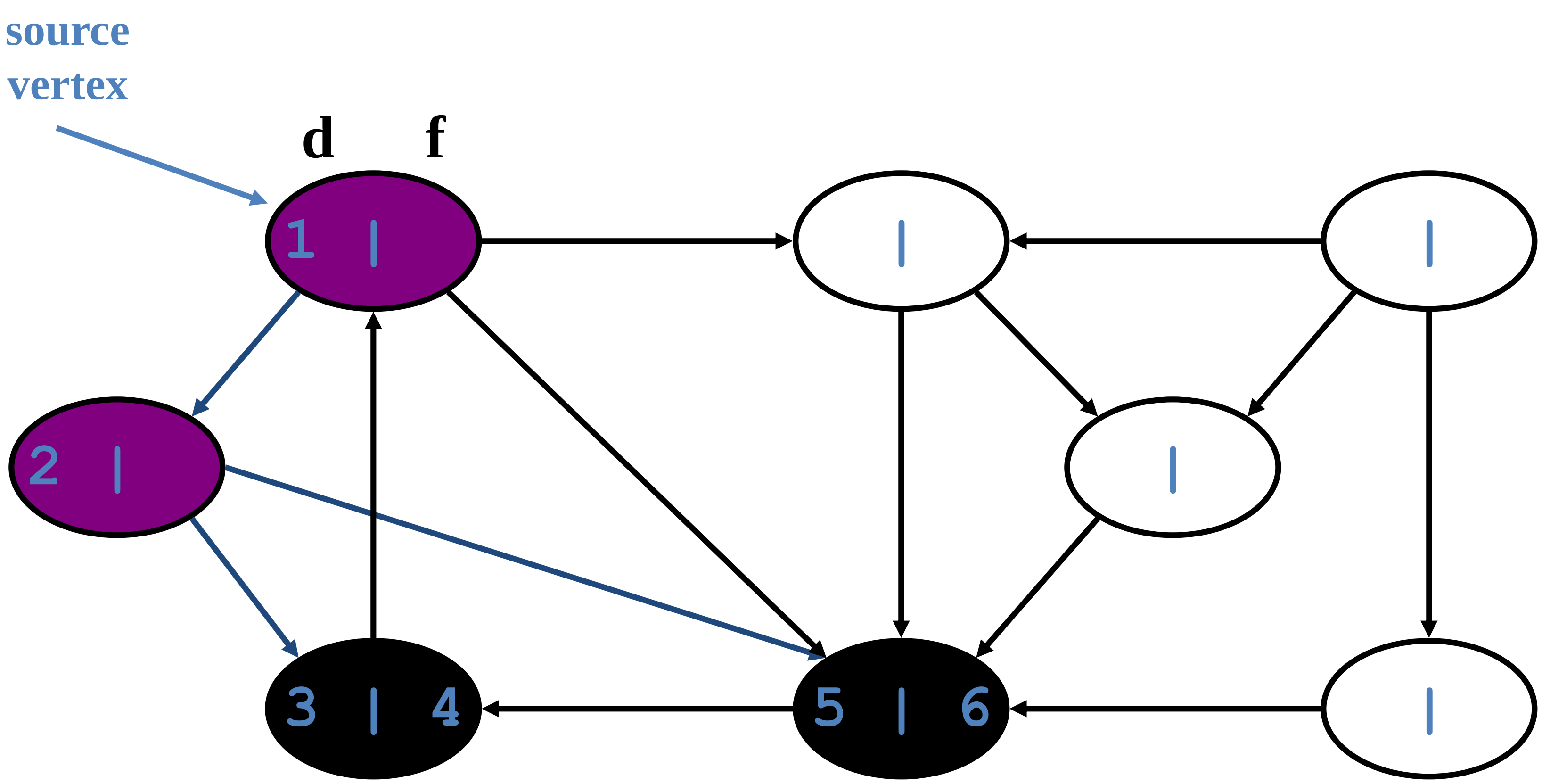


DFS Example



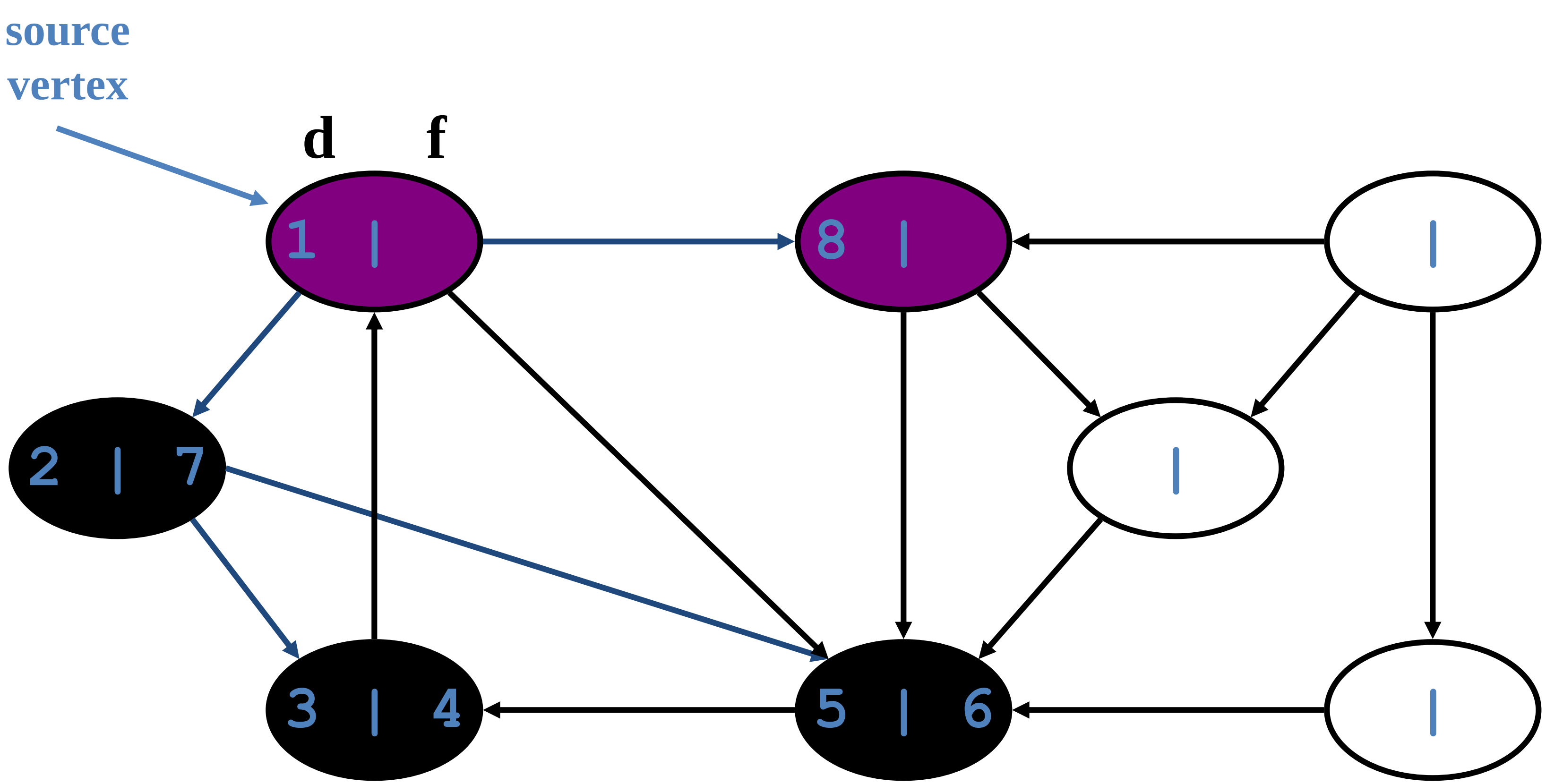
© Ramaiah University of Applied Sciences

DFS Example



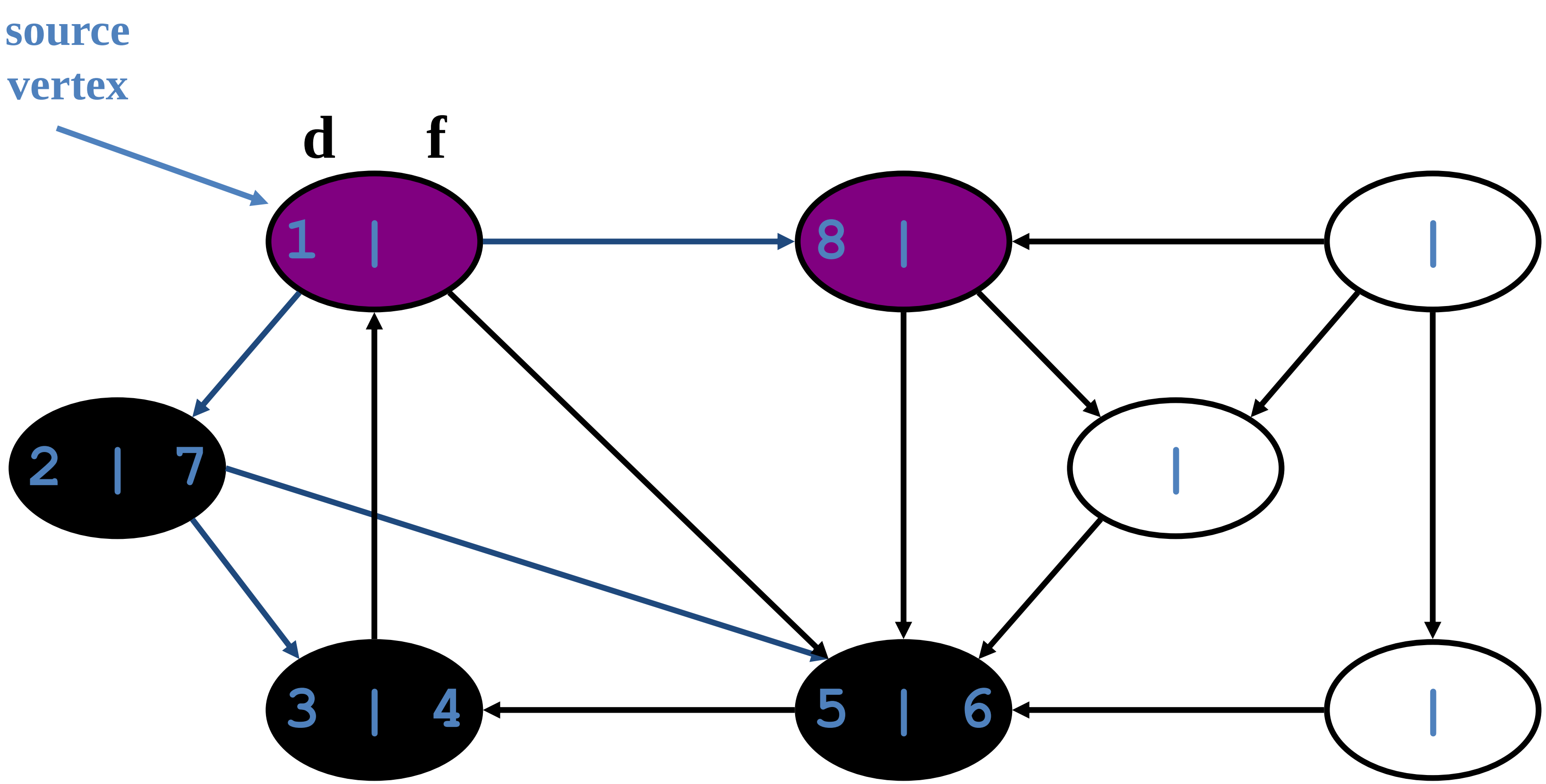
© Ramaiah University of Applied Sciences

DFS Example



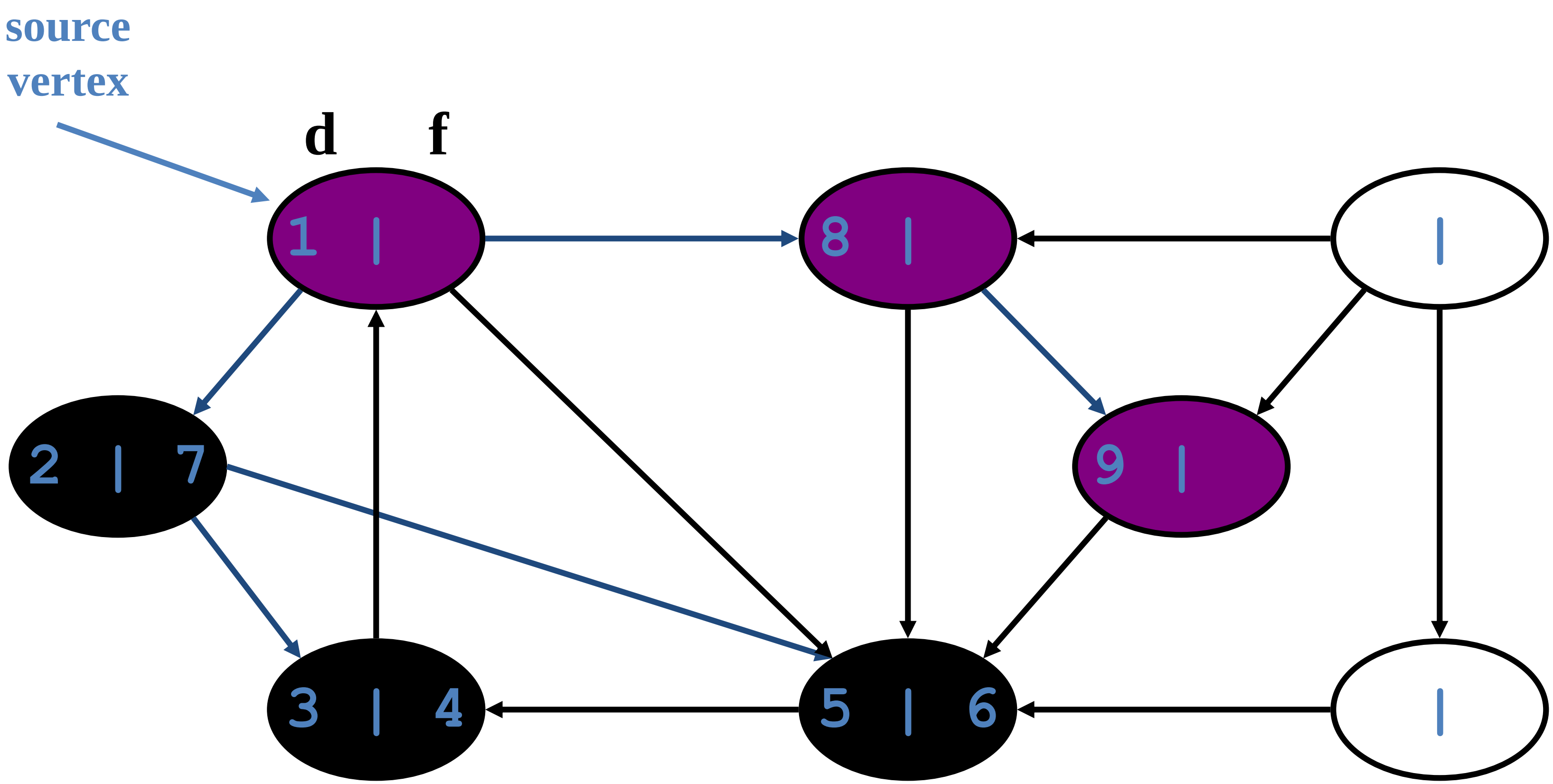
© Ramaiah University of Applied Sciences

DFS Example



© Ramaiah University of Applied Sciences

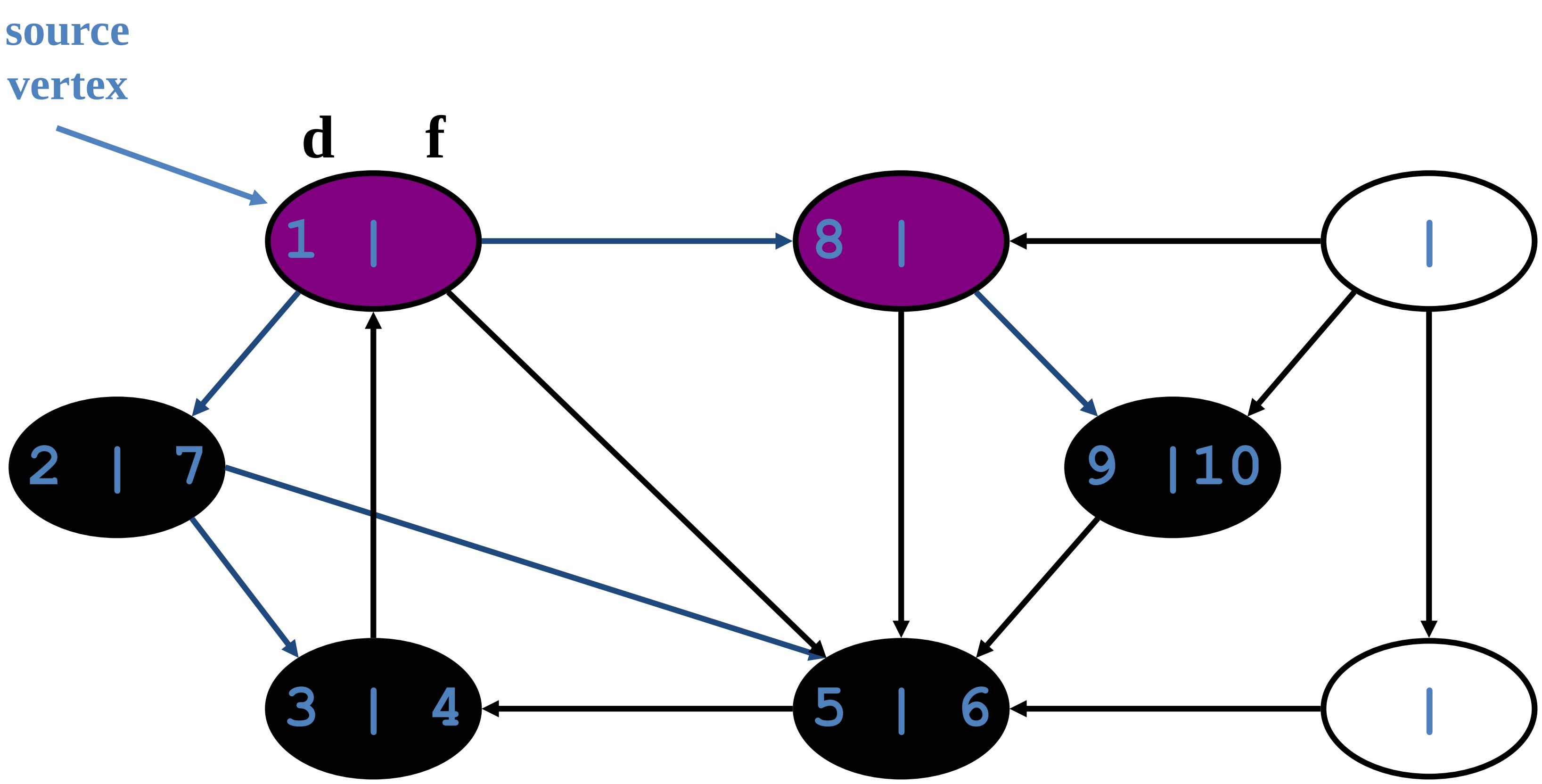
DFS Example



What is the structure of the grey vertices?
What do they represent?

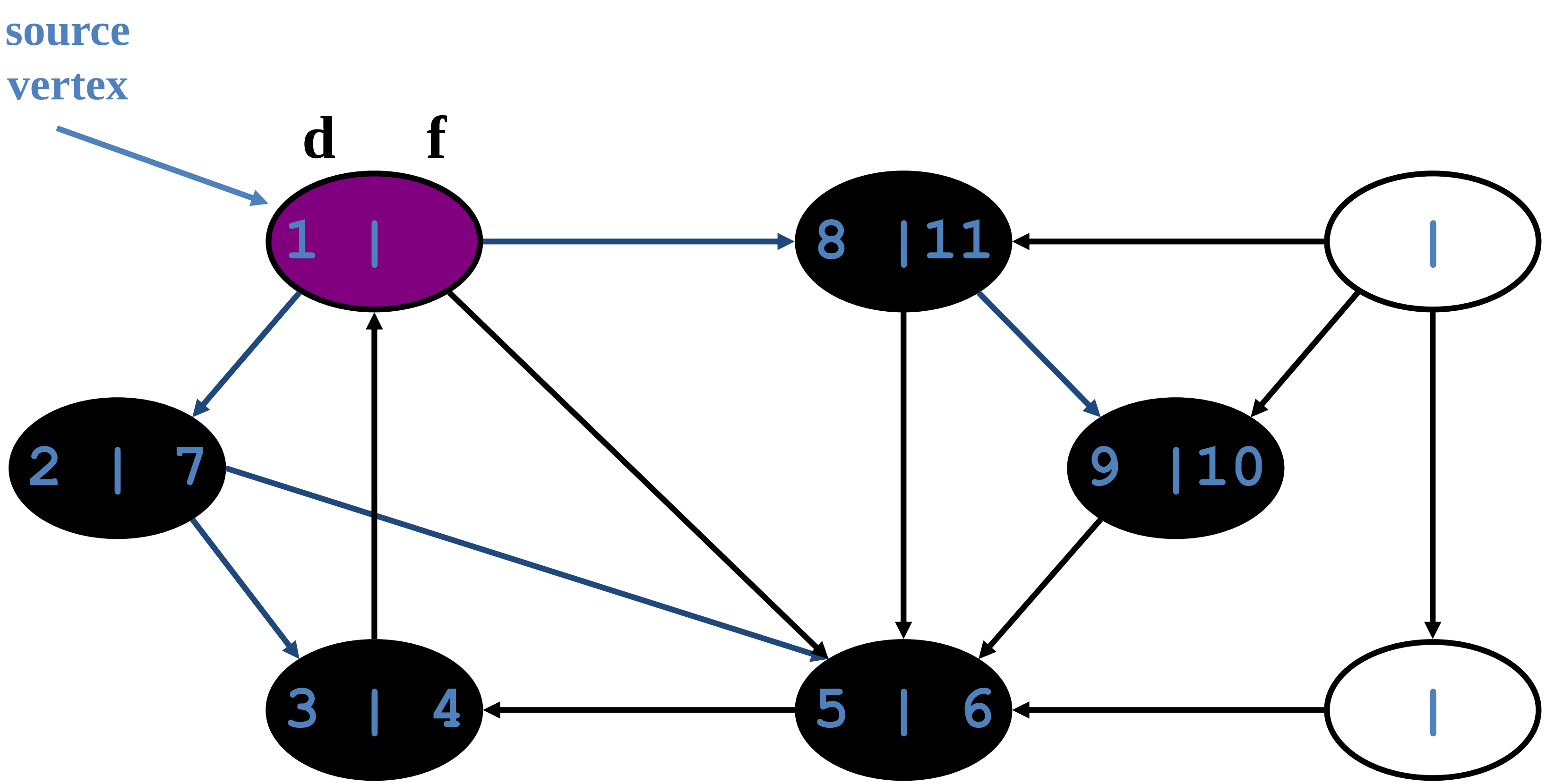
© Ramaiah University of Applied Sciences

DFS Example



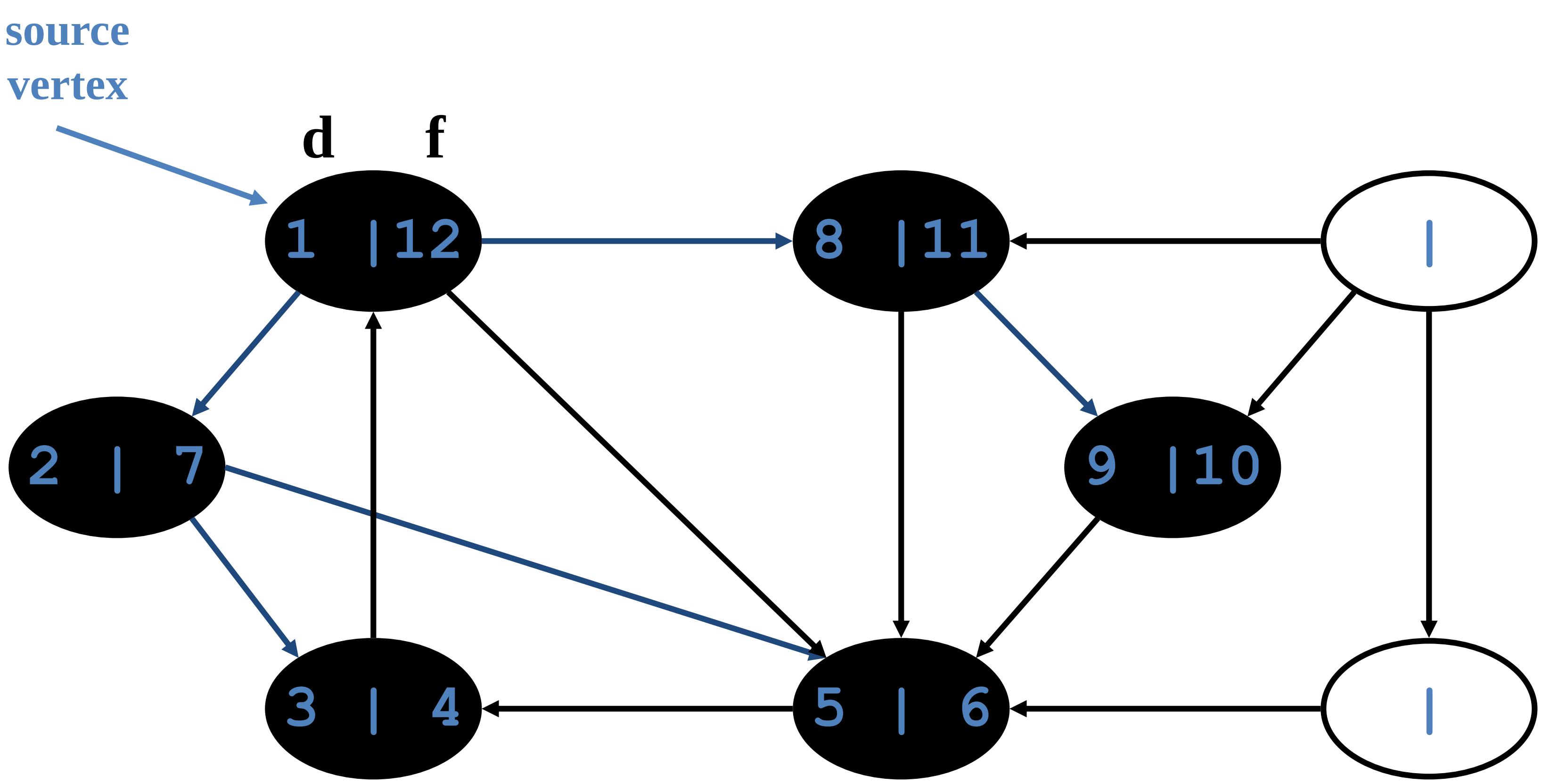
© Ramaiah University of Applied Sciences

DFS Example



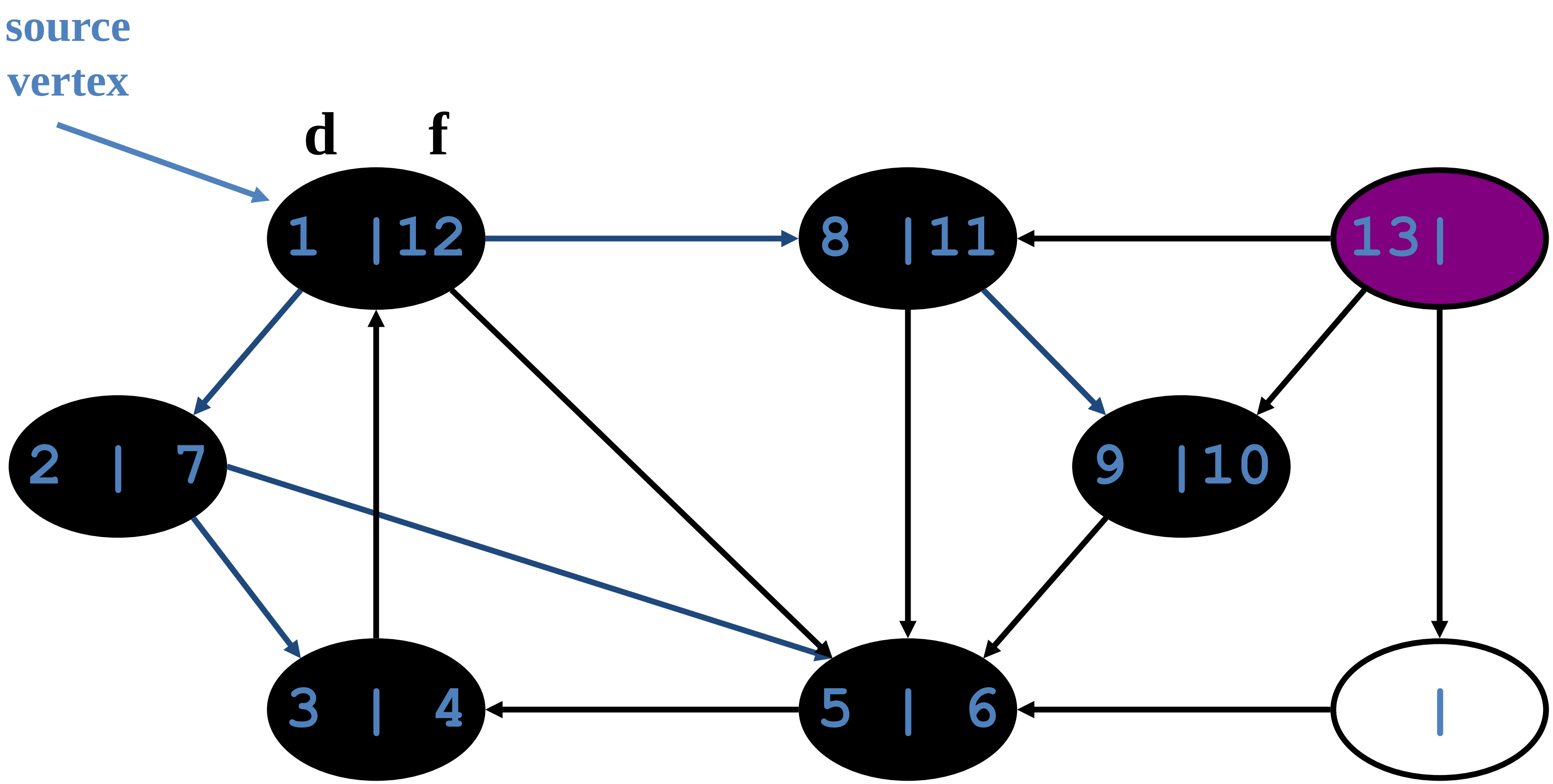
© Ramaiah University of Applied Sciences

DFS Example



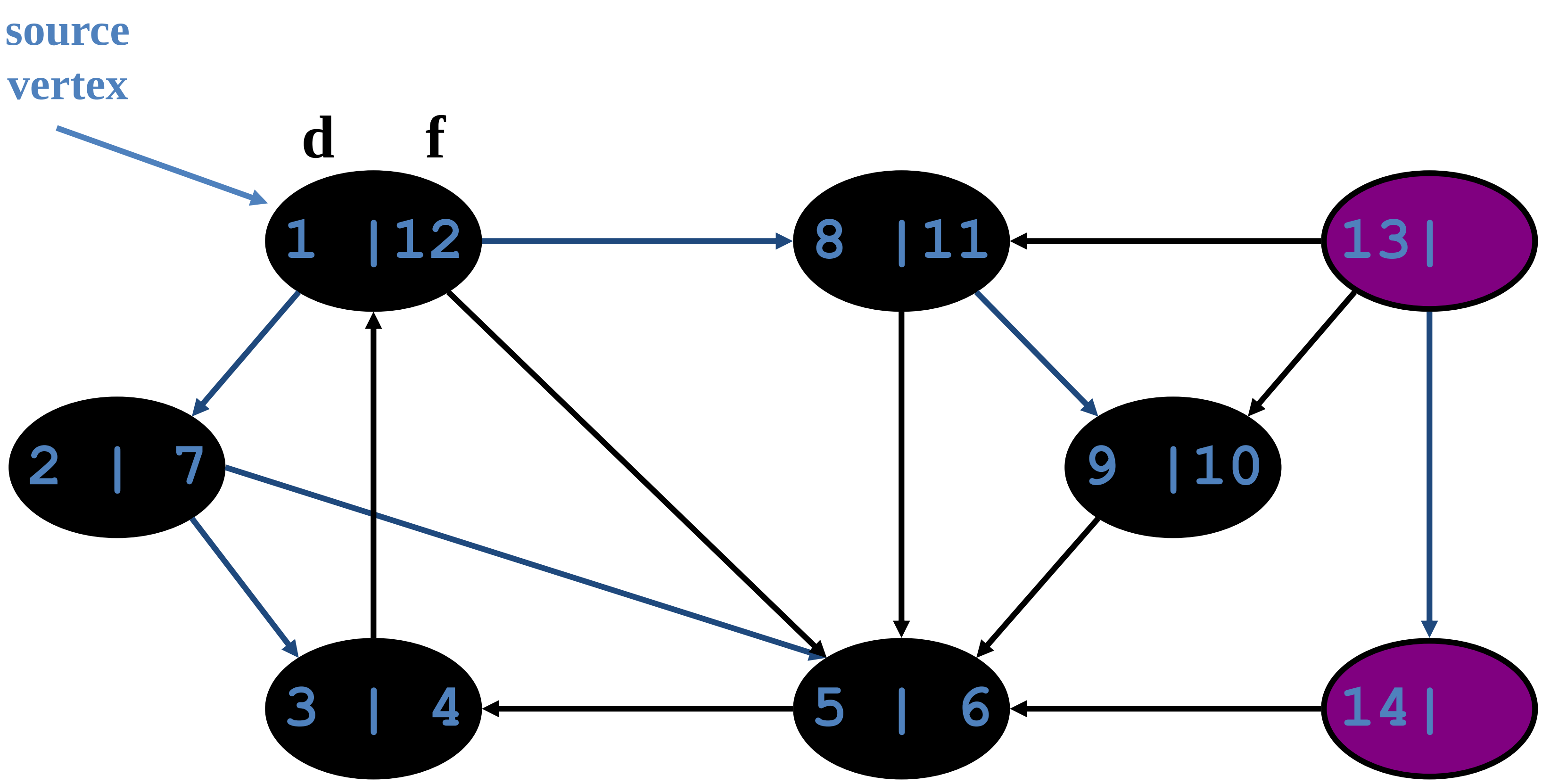
© Ramaiah University of Applied Sciences

DFS Example



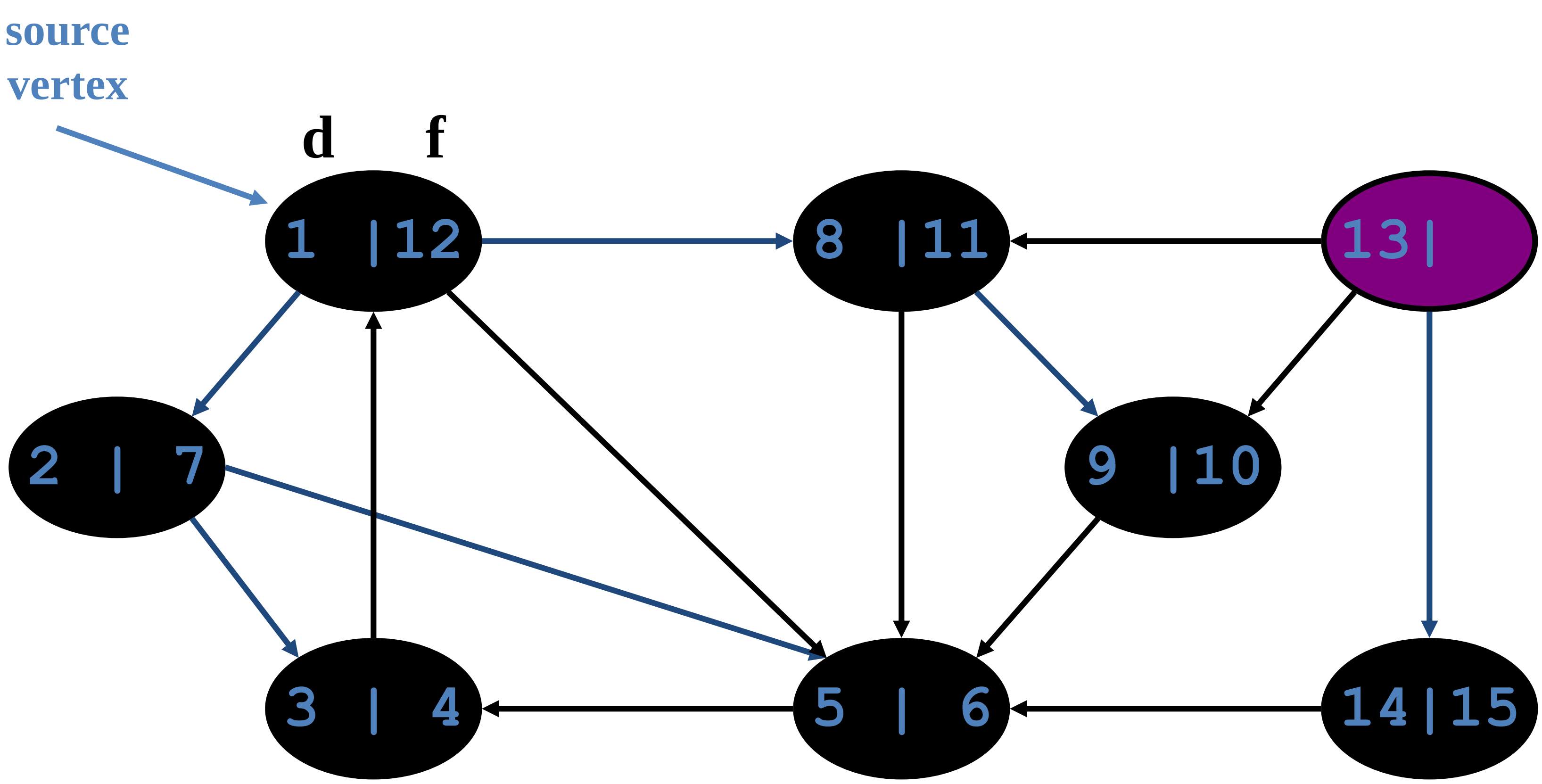
© Ramaiah University of Applied Sciences

DFS Example



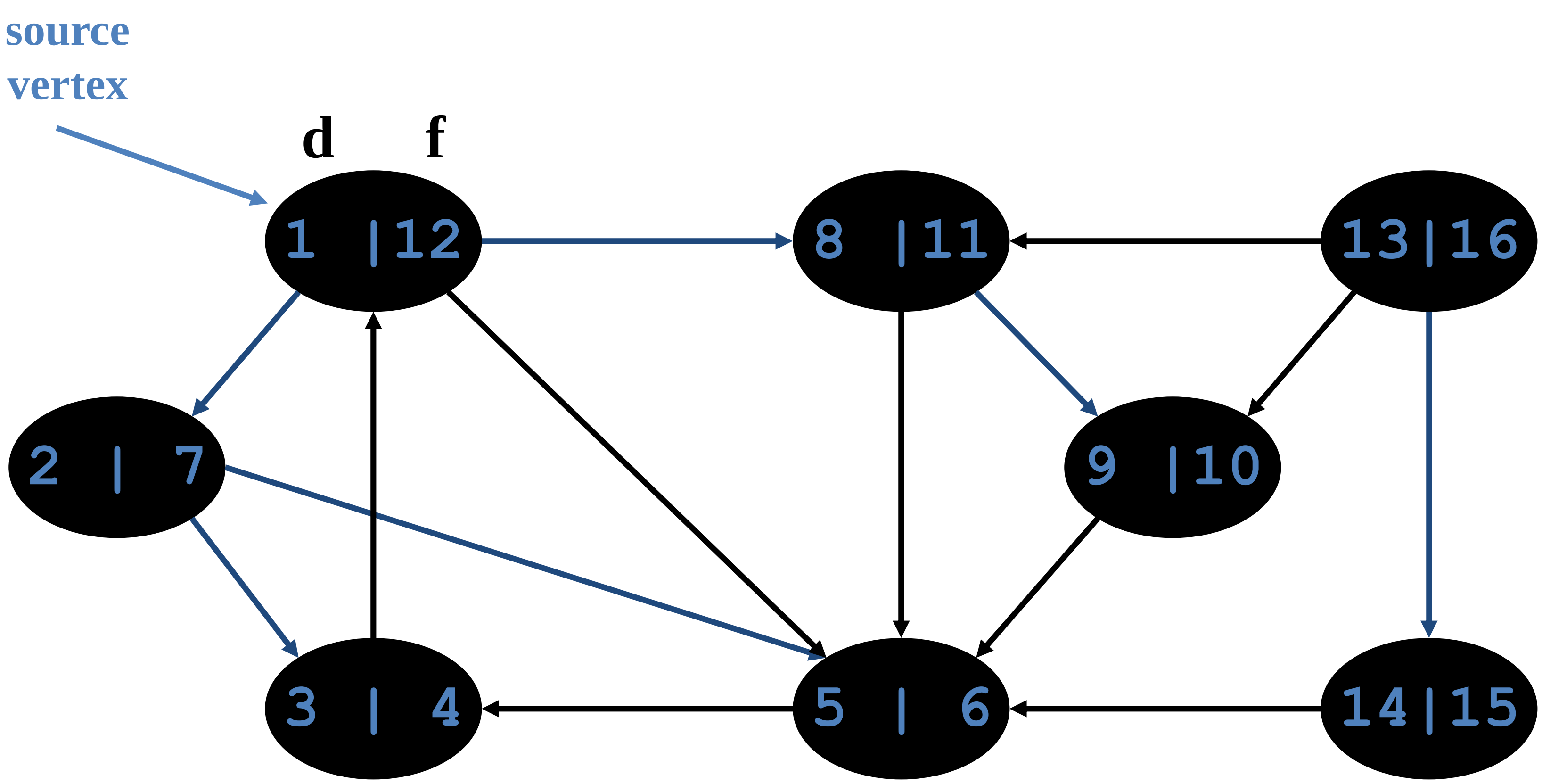
© Ramaiah University of Applied Sciences

DFS Example



© Ramaiah University of Applied Sciences

DFS Example



© Ramaiah University of Applied Sciences

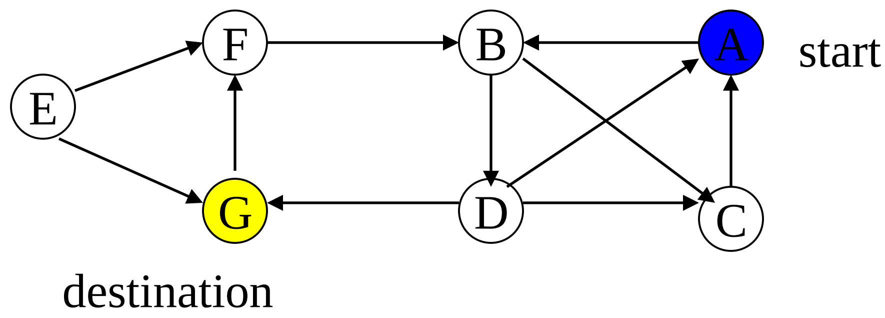
Applications: Finding a Path

- Find path from source vertex s to destination vertex d
- Use graph search starting at s and terminating as soon as we reach d
  - Need to remember edges traversed
- Use depth – first search ?
- Use breath – first search?
- Check whether a given graphs are connected or not

© Ramaiah University of Applied Sciences

DFS vs. BFS

DFS Process



A

DFS on A

B
A

DFS on B

C
B
A

DFS on C

D
B
A

Call DFS on D

Return to call on B

G
D
B
A

Call DFS on G

found destination - done!

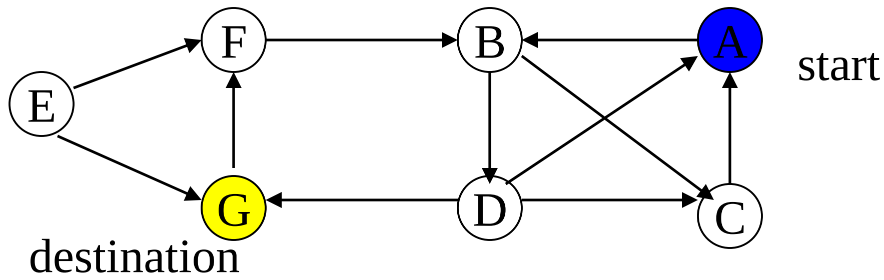
Path is implicitly stored in DFS recursion

Path is: A, B, D, G

© Ramaiah University of Applied Sciences

DFS vs. BFS

BFS Process



rear	front
	A

Initial call to BFS on A
Add A to queue

rear	front
	B

Dequeue A
Add B

rear	front
	D C

Dequeue B
Add C, D

rear	front
	D

Dequeue C
Nothing to add

rear	front
	G

Dequeue D
Add G

found destination - done!
Path must be stored separately

© Ramaiah University of Applied Sciences

Experimental Procedure

- Analyse the problem statement
- Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- Implement the algorithm in C language
- Compile the C program
- Design test cases and test the implemented program
- Document the Results
- Analyse and discuss the outcomes of your experiment

© Ramaiah University of Applied Sciences

Exercise

- Design and develop algorithms to check whether given graph is connected or not using DFS and BFS. Tabulate the output for various inputs and verify against expected values. Analyse the efficiency of both the algorithms. Describe your learning along with the limitations of both, if any. Suggest how these can be overcome.

© Ramaiah University of Applied Sciences

Key factors and discussion

- Implement DFS for traversing a given graph
- Implement BFS for traversing a given graph
- Check if a given graphs are connected or not

© Ramaiah University of Applied Sciences

Results and Presentations

- Calculations/Computations/Algorithms

  The calculations/computations/algorithms involved in each program has to be presented

- Presentation of Results

  The results for all the valid and invalid cases have to be presented

- Analysis and Discussions

  how the data is manipulated or transformed, what are the key operations involved. Errors encounters and how they are resolved.

- Conclusions

  Summary

© Ramaiah University of Applied Sciences

Comments

- Limitations of Experiments

  Outline the loopholes in the program, data structures or solution approach.

- Limitations of Results

  Present the test cases; justify if the program is tested correctly considering all the outcomes. Mention what is not tested, if any.

- Learning happened

  What is the overall learning happened

- Conclusions

  Summary

© Ramaiah University of Applied Sciences

References

- Gilberg, R. F., and Forouzan, B. A. (2007): A Pseudocode Approach With C, 2nd edn. Cengage Learning

© Ramaiah University of Applied Sciences