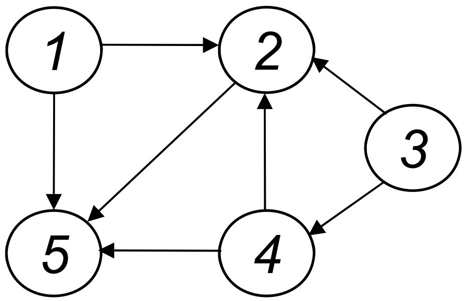
Lecture 4

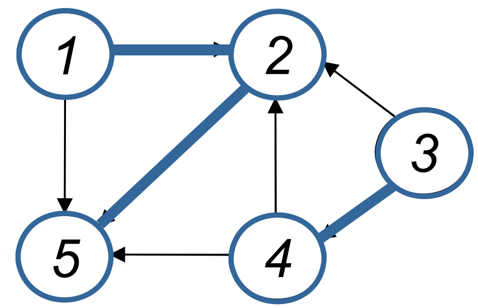
Amit Kumar Das
Lecturer
Department of CSE
East West University

Depth-First Search



- **Input:**
  - G = (V, E) (No source vertex given!)
- **Goal**:
  - Explore the edges of G to "discover" every vertex in V starting at the most current visited node
  - Search may be repeated from multiple sources
- **Output:**
  - 2 **timestamps** on each vertex:
    - d[v] = discovery time
    - f[v] = finishing time (done with examining v's adjacency list)
  - Depth-first forest

Depth-First Search

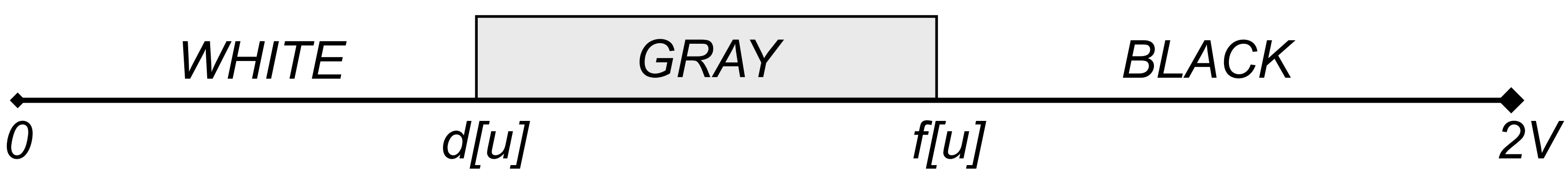


- Search "deeper" in the graph whenever possible
- Edges are explored out of the most recently discovered vertex v that still has unexplored edges
- *After all edges of v have been explored, the search "backtracks" from the parent of v*
- *The process continues until all vertices reachable from the original source have been discovered*
- *If undiscovered vertices remain, choose one of them as a new source and repeat the search from that vertex*
- *DFS creates a "depth-first forest"*

DFS Additional Data Structures

- Global variable: time-stamp
  - Incremented when nodes are discovered or finished
- color[u] – similar to BFS
  - White before discovery, gray while processing and black when finished processing
- prev[u] – predecessor of u
- d[u], f[u] – discovery and finish times

$$1 \le d[u] < f[u] \le 2\,|V|$$



Depth-First Search: The Code

```
Data: color[V], time,
   prev[V],d[V], f[V]
DFS(G) // where prog starts
{                                  Initialize
   for each vertex u ∈ V
   {
      color[u] = WHITE;
       prev[u]=NIL;
       f[u]=inf; d[u]=inf;
   }
   time = 0;
   for each vertex u ∈ V
     if (color[u] == WHITE)
         DFS_Visit(u);
}
```

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

Depth-First Search: The Code

```
Data: color[V], time,
   prev[V],d[V], f[V]
DFS(G) // where prog starts
{
   for each vertex u ∈ V
   {
      color[u] = WHITE;
       prev[u]=NIL;
       f[u]=inf; d[u]=inf;
   }
   time = 0;
   for each vertex u ∈ V
     if (color[u] == WHITE)
         DFS_Visit(u);
}
```

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

What does `u[d]` *represent?*

Depth-First Search: The Code

```
Data: color[V], time,
   prev[V],d[V], f[V]
DFS(G) // where prog starts
{
   for each vertex u ∈ V
   {
      color[u] = WHITE;
       prev[u]=NIL;
       f[u]=inf; d[u]=inf;
   }
   time = 0;
   for each vertex u ∈ V
     if (color[u] == WHITE)
         DFS_Visit(u);
}
```

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

What does `f[d]` *represent?*

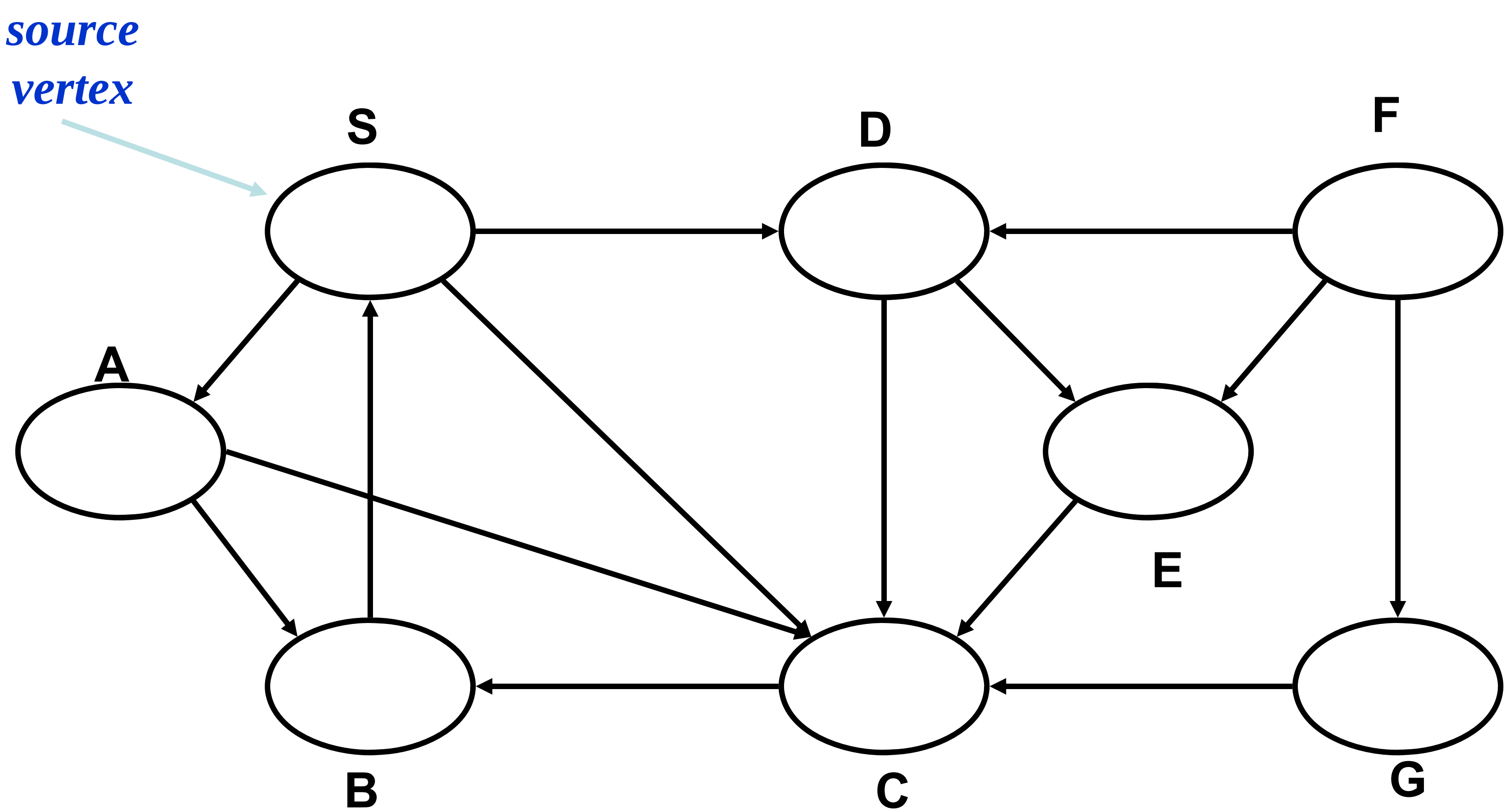
Depth-First Search: The Code

```
Data: color[V], time,
  prev[V],d[V], f[V]
DFS(G) // where prog starts
{
   for each vertex u ∈ V
   {
      color[u] = WHITE;
       prev[u]=NIL;
       f[u]=inf; d[u]=inf;
   }
   time = 0;
   for each vertex u ∈ V
     if (color[u] == WHITE)
         DFS_Visit(u);
}
```

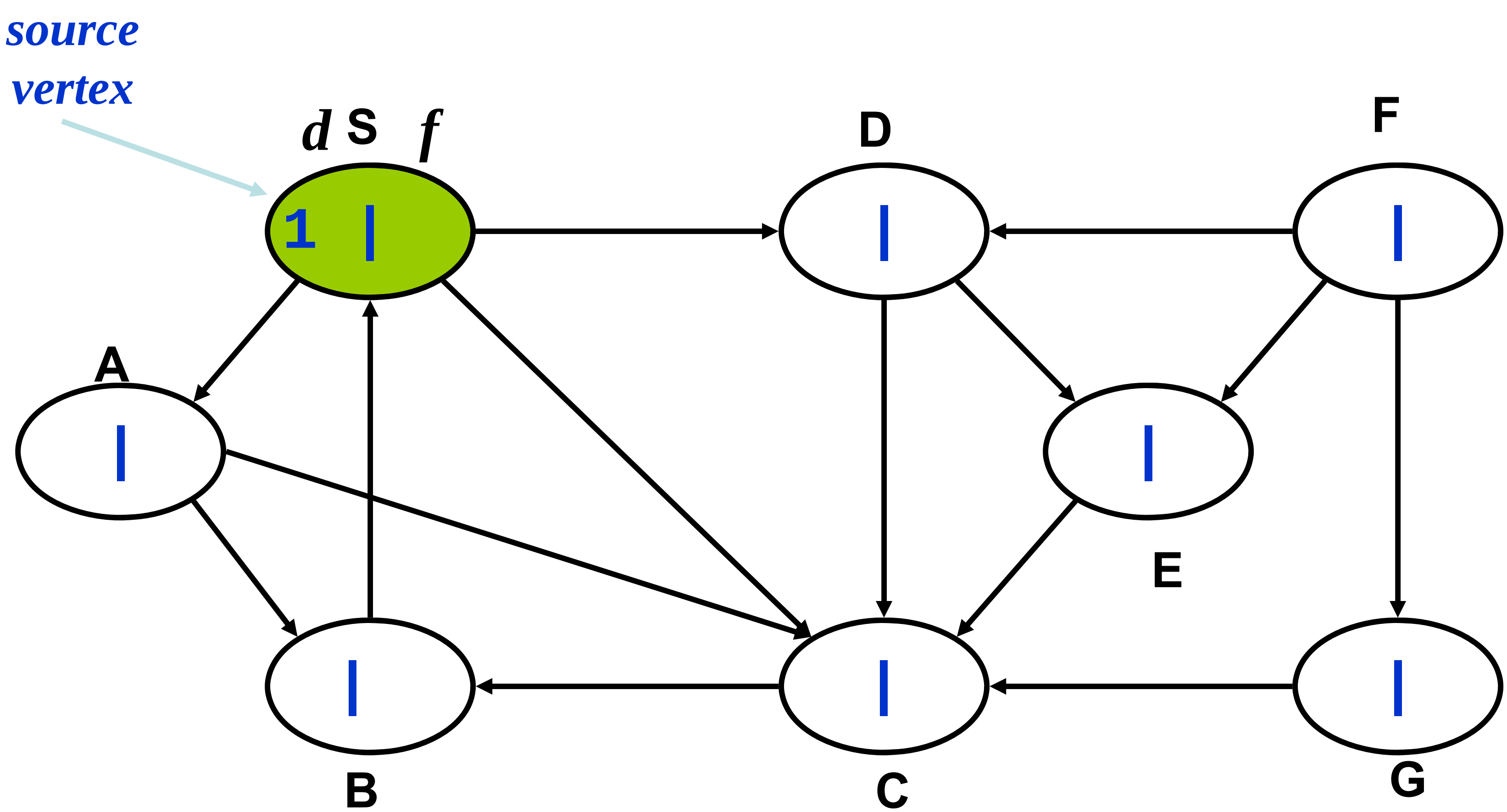
```
DFS_Visit(u)
{
   color[u] = GREY;
   time = time+1;
   d[u] = time;
   for each v ∈ Adj[u]
   {
      if(color[v] == WHITE){
         prev[v]=u;
          DFS_Visit(v);
   }}
   color[u] = BLACK;
   time = time+1;
   f[u] = time;
}
```

Will all vertices eventually be colored black?

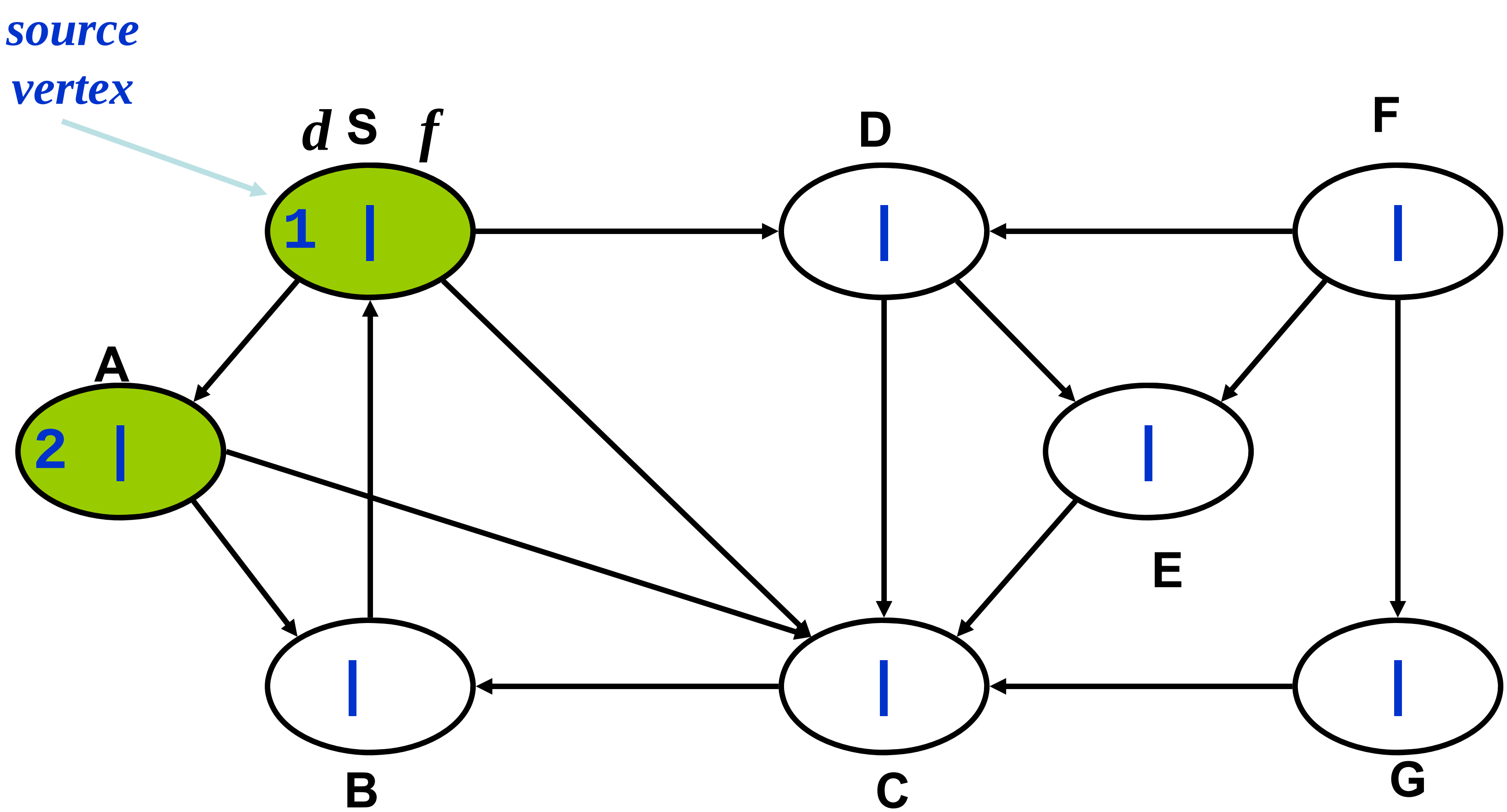
DFS Example



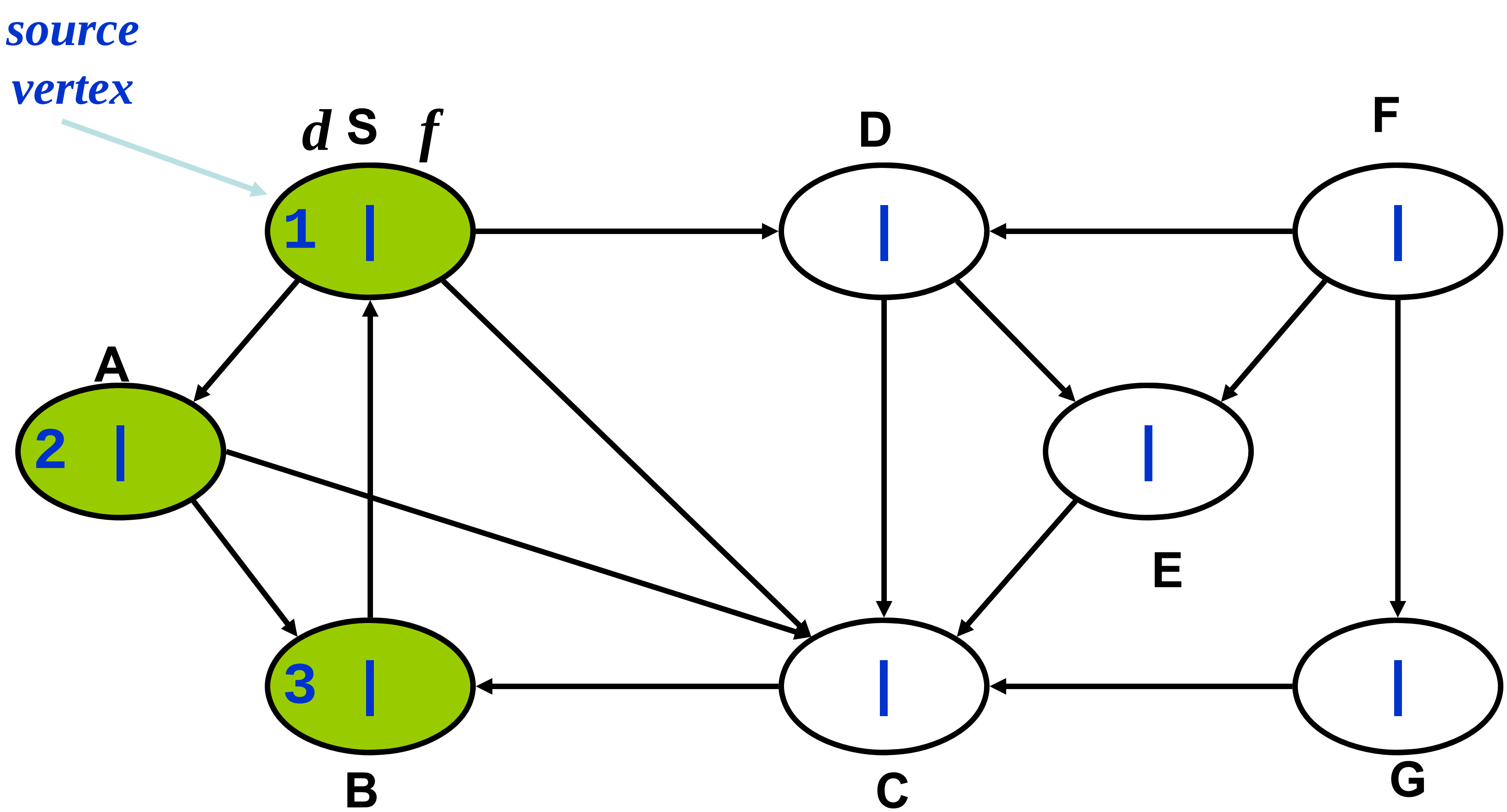
DFS Example



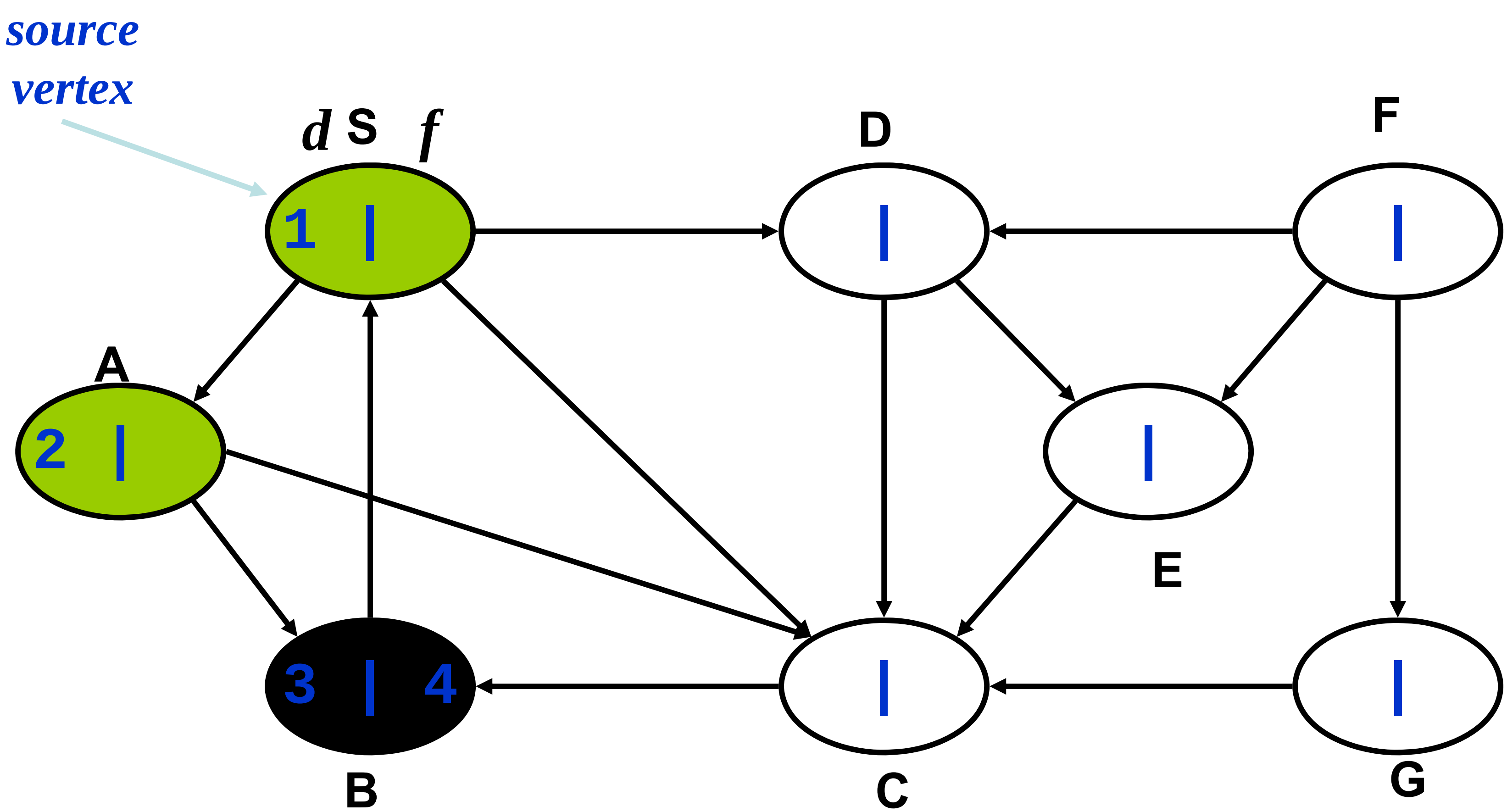
DFS Example



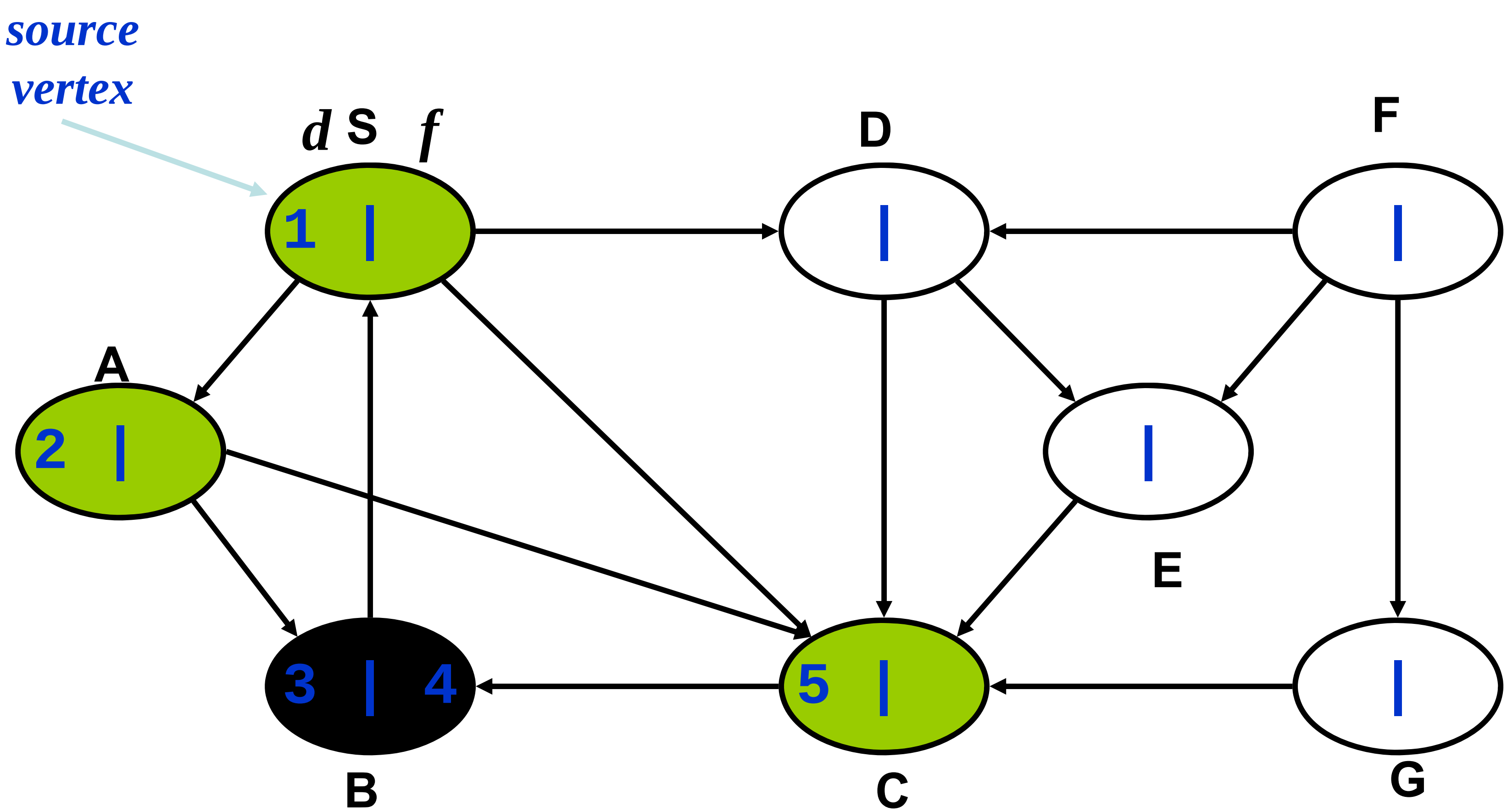
DFS Example



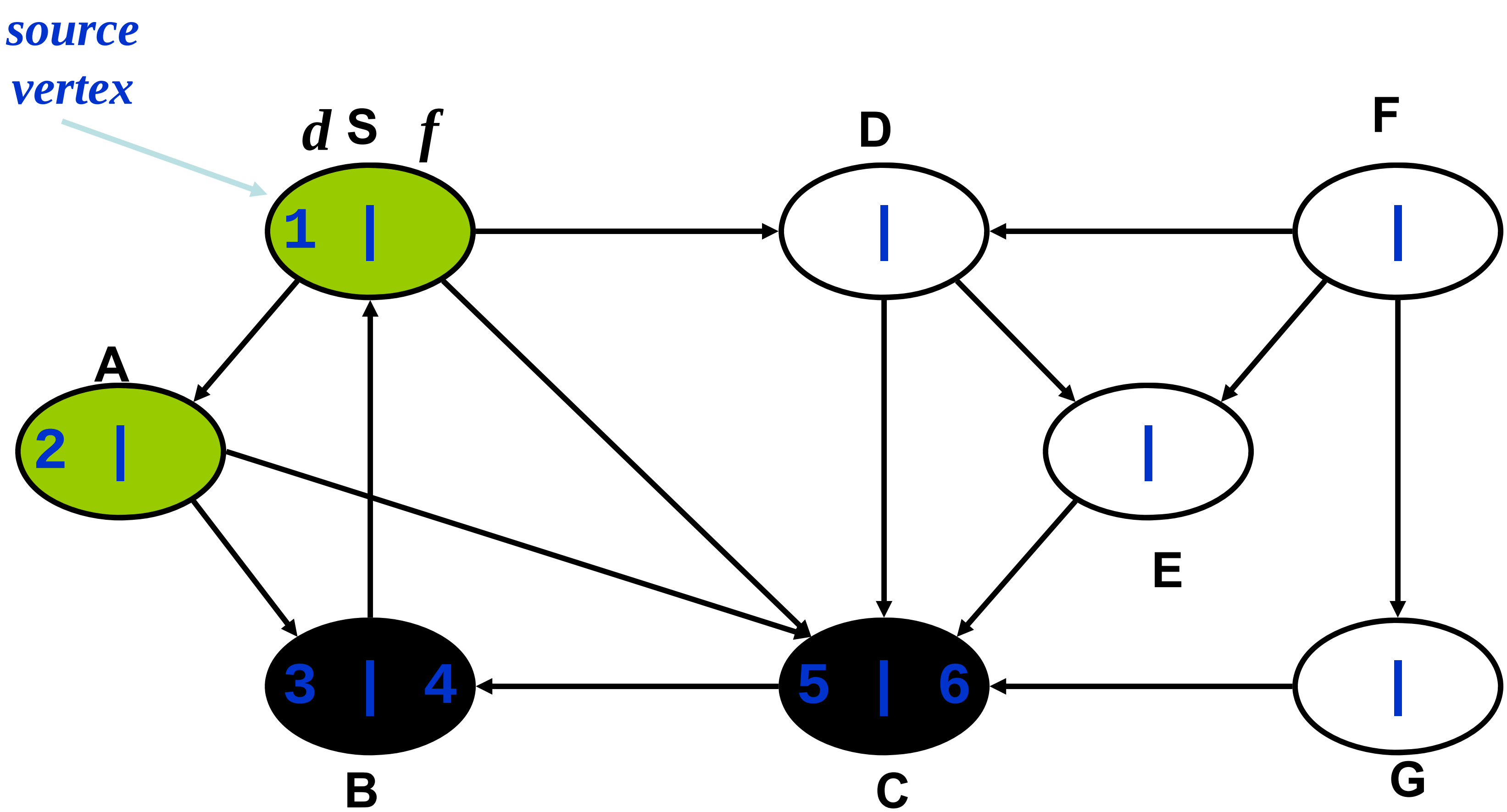
DFS Example



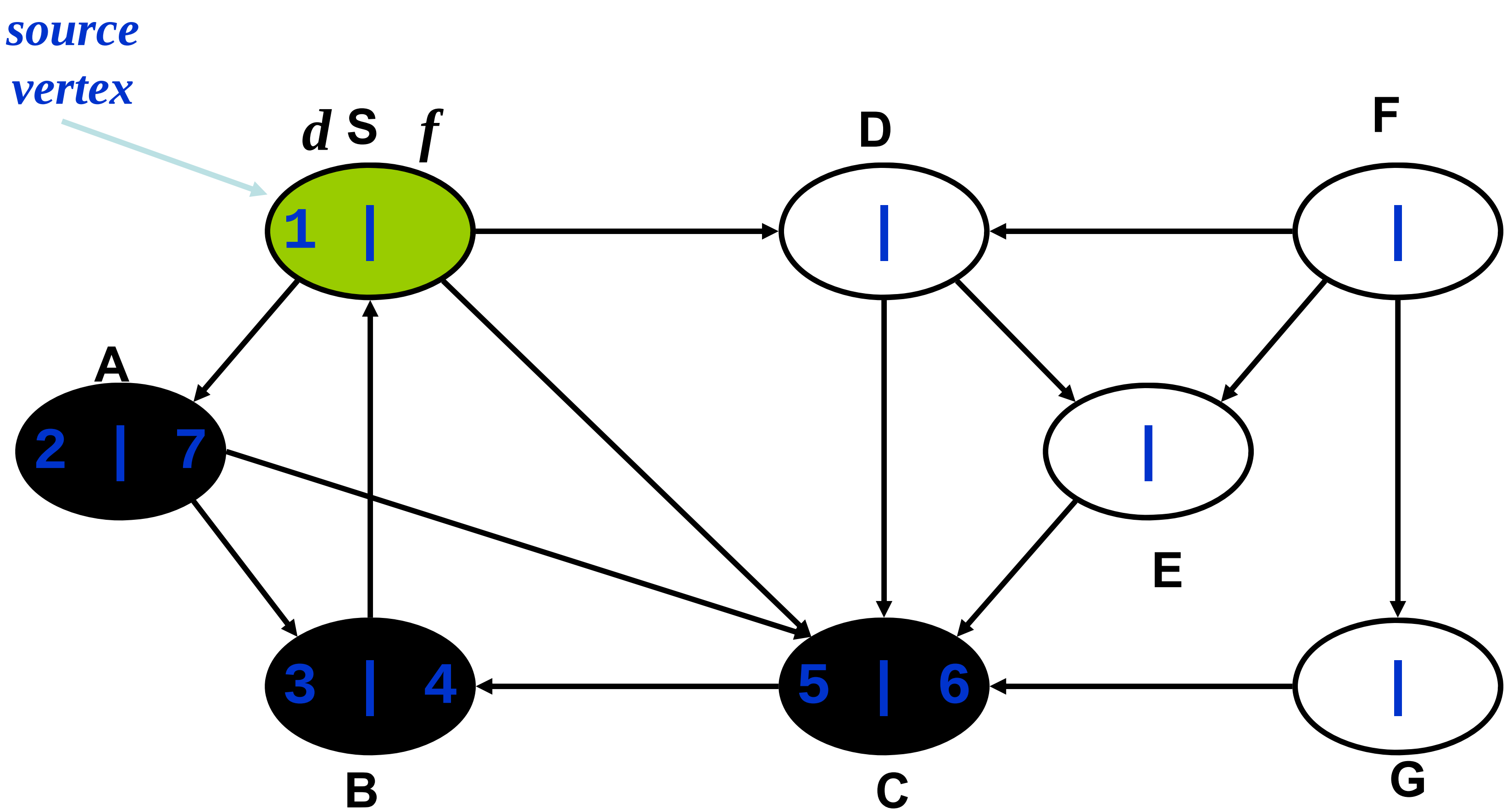
DFS Example



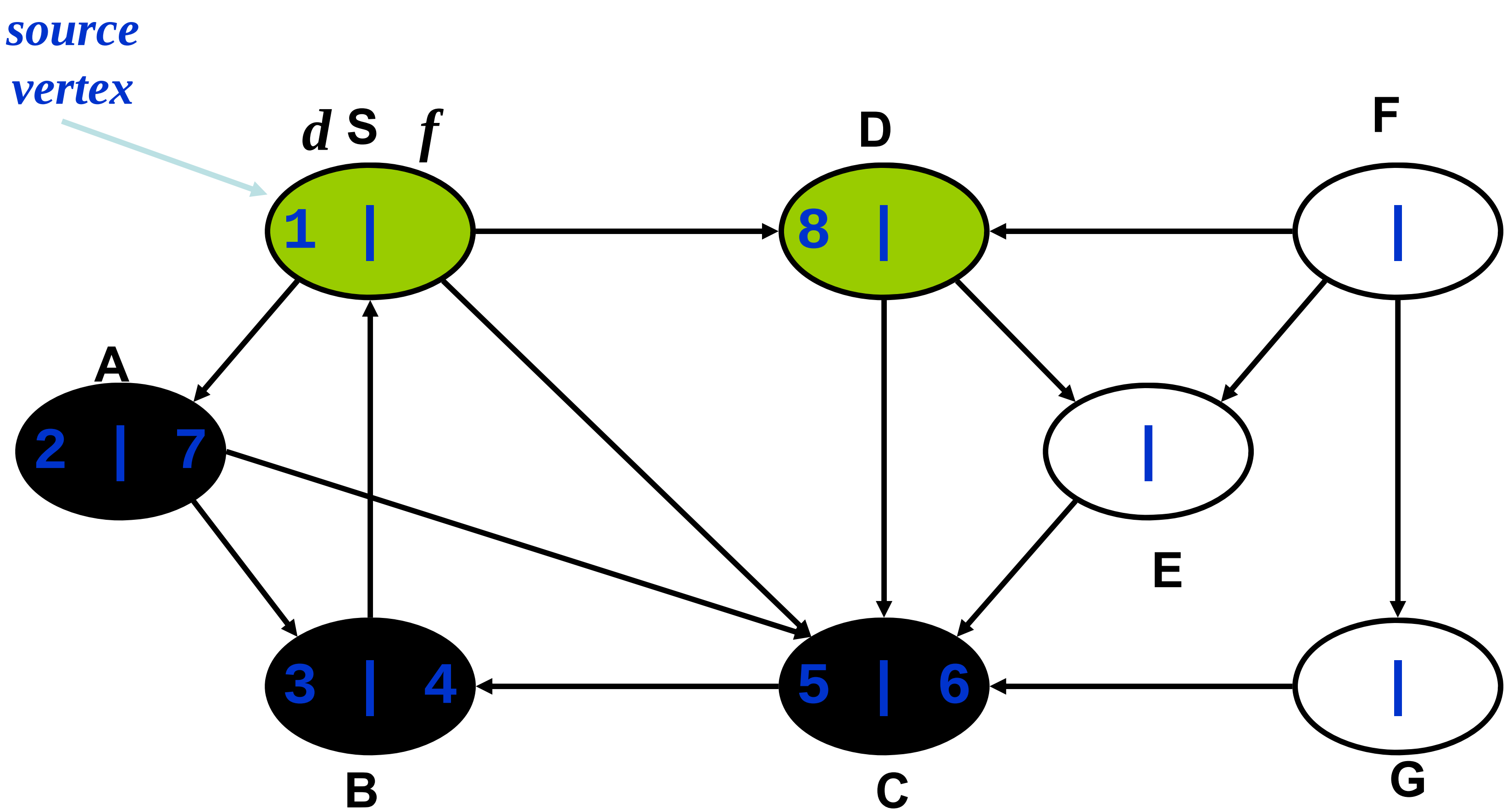
DFS Example



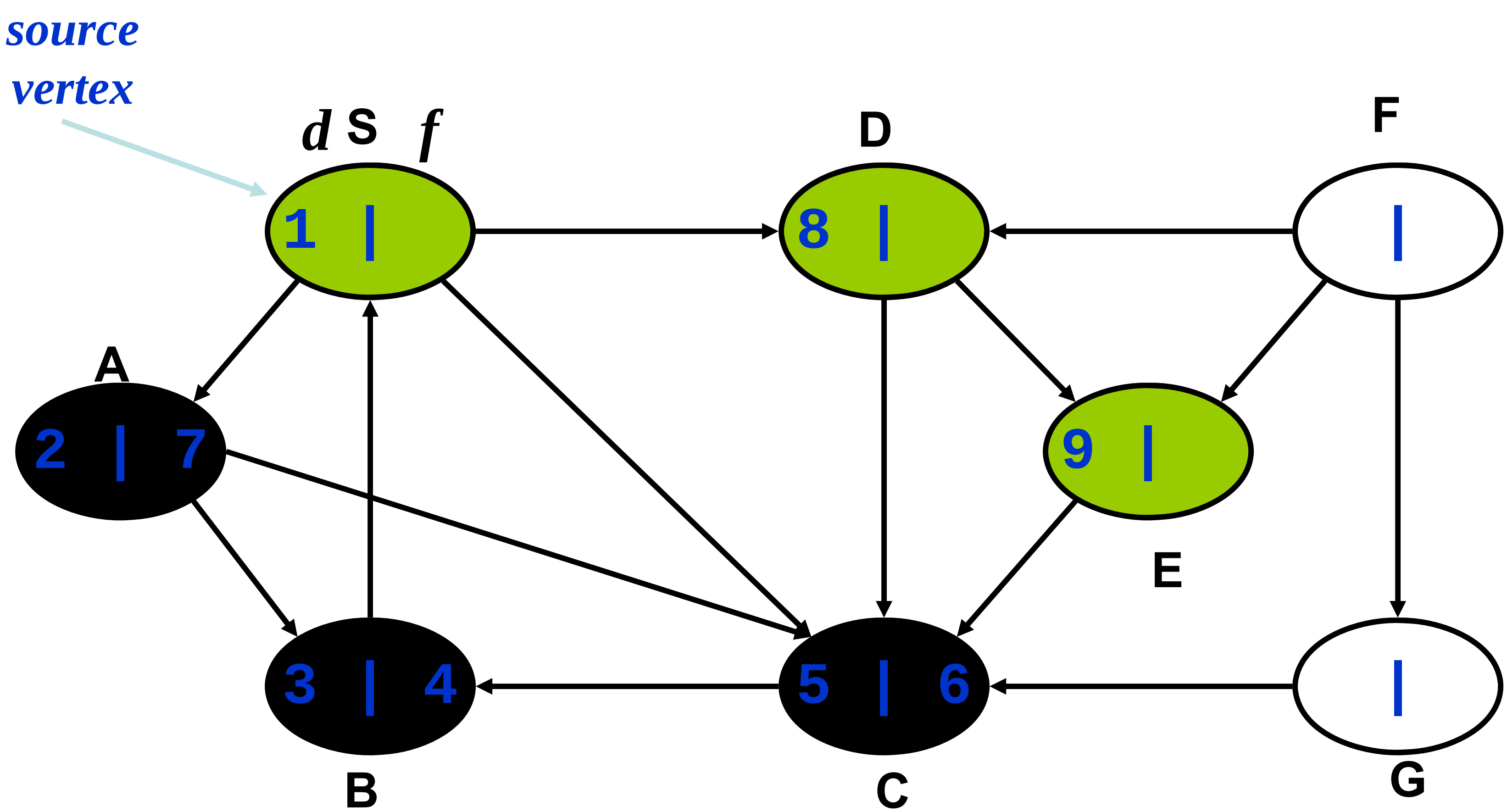
DFS Example



DFS Example

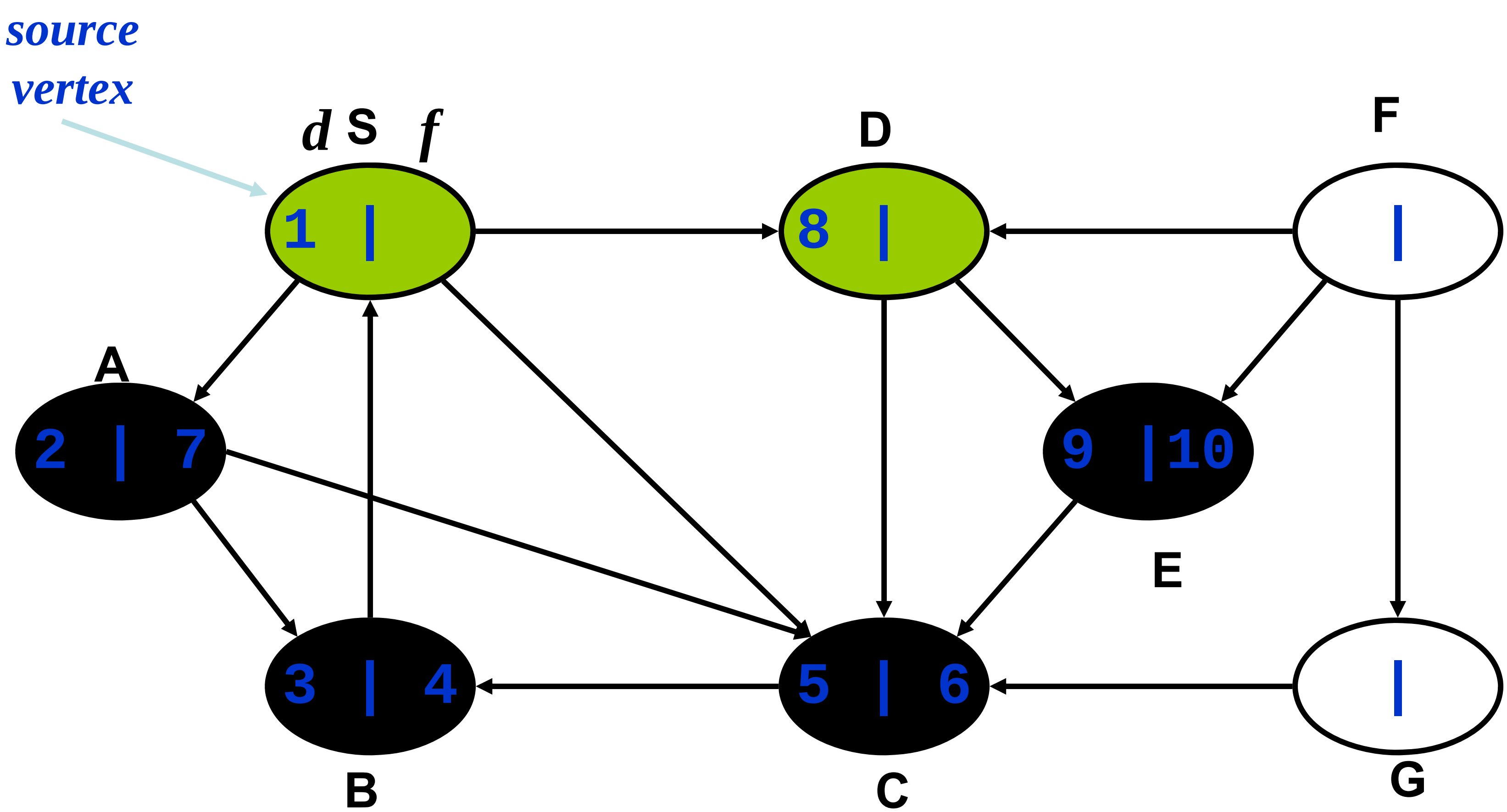


DFS Example

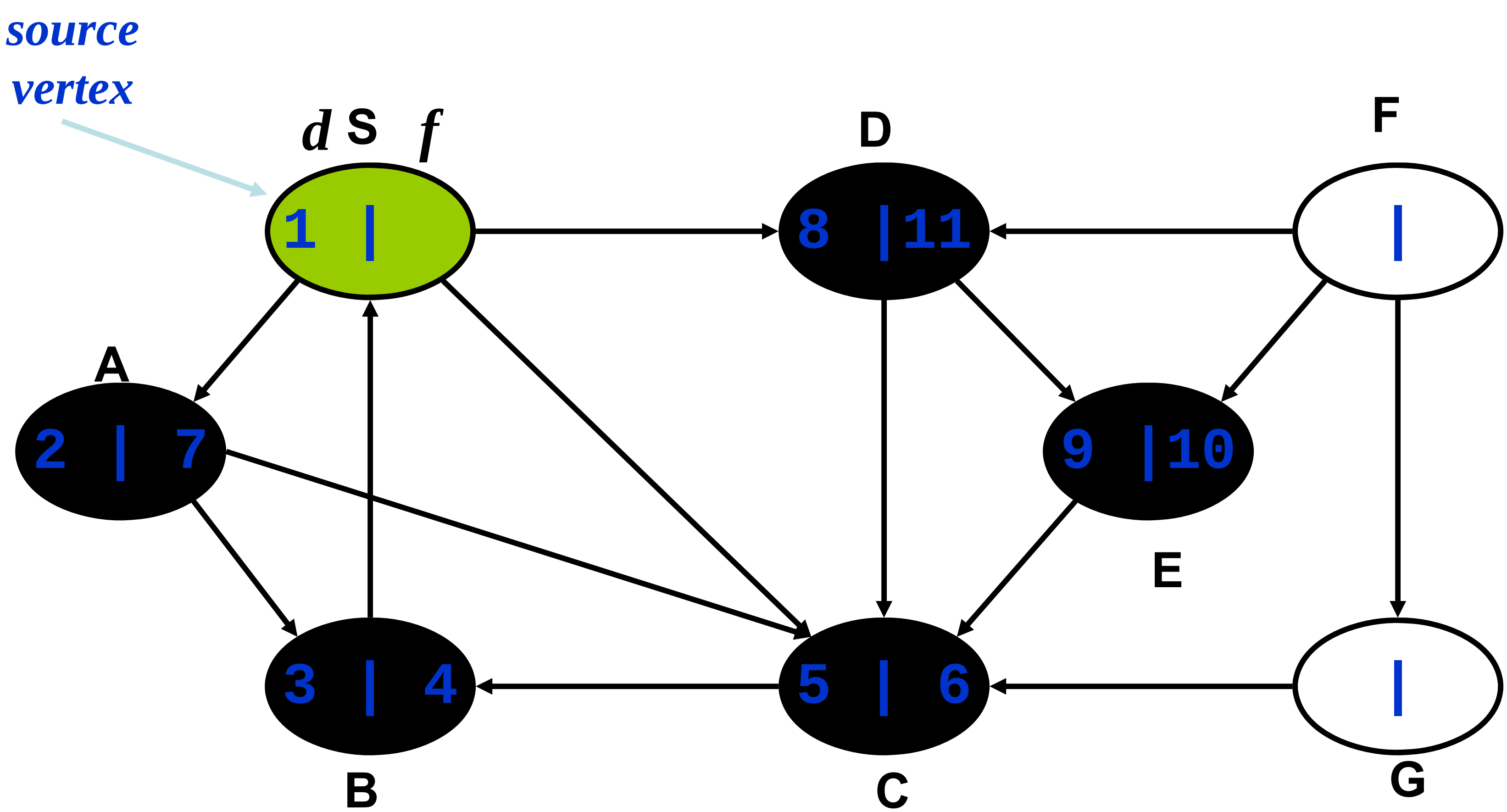


What is the structure of the grey vertices?
What do they represent?

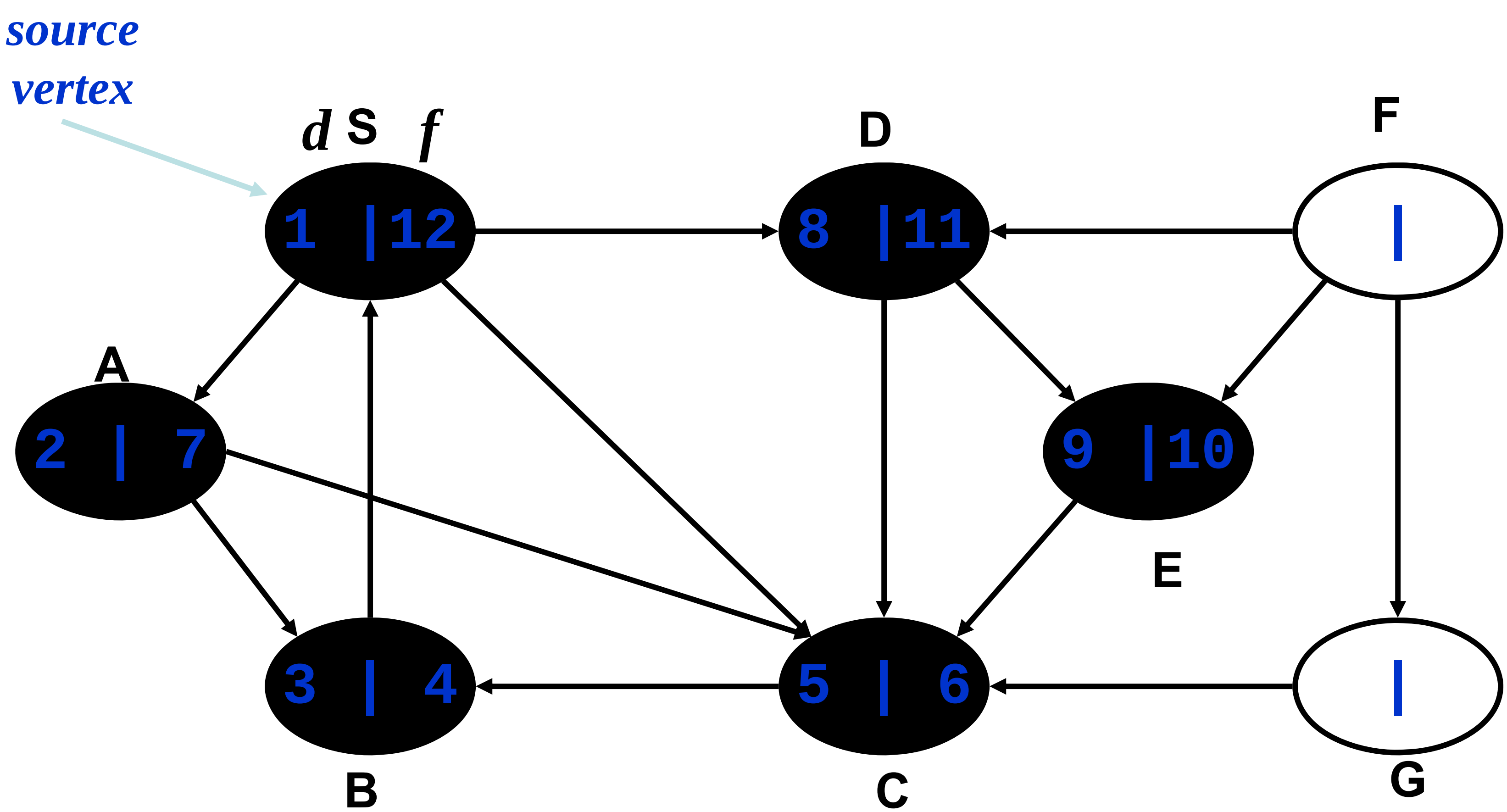
DFS Example



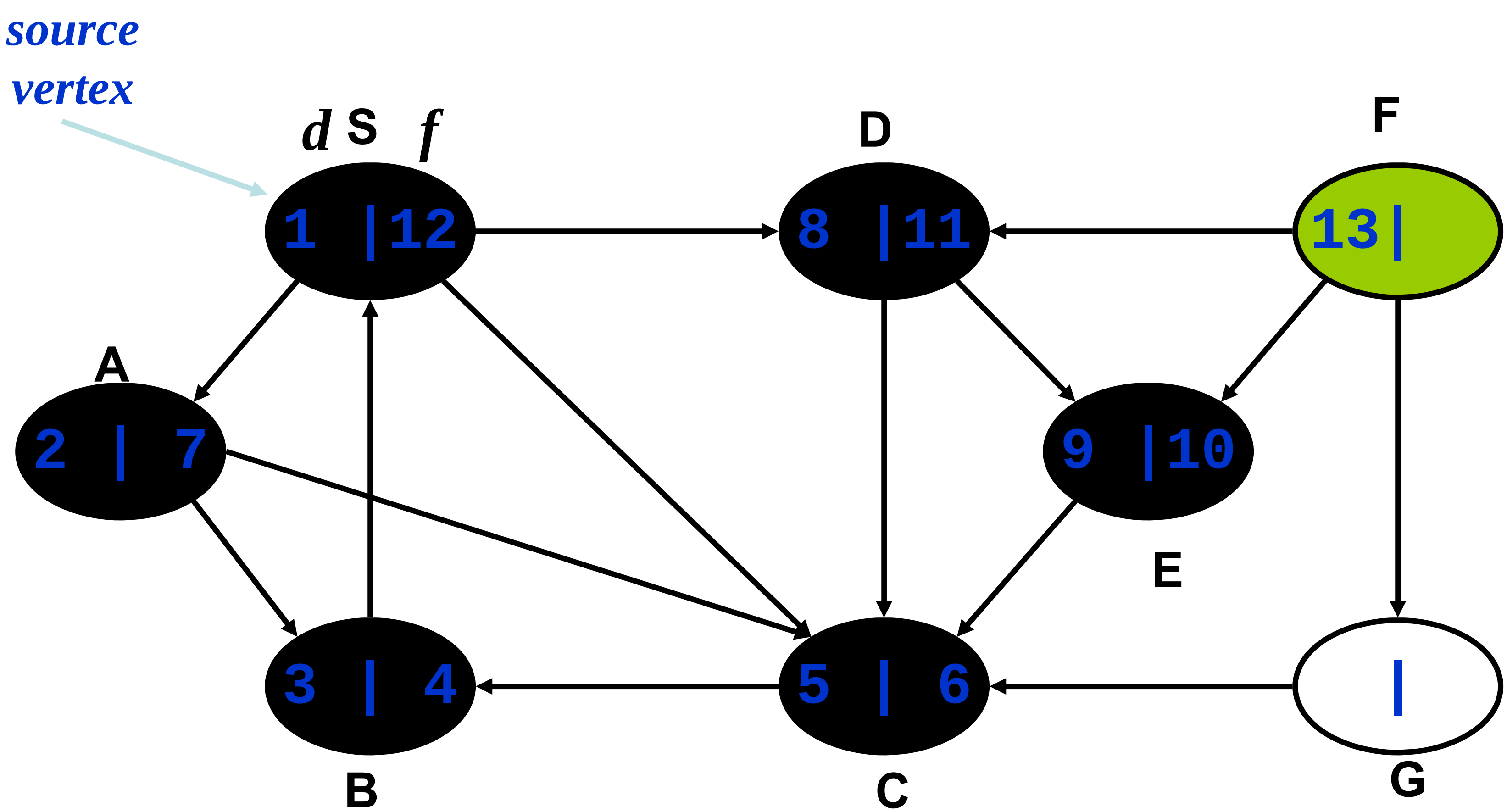
DFS Example



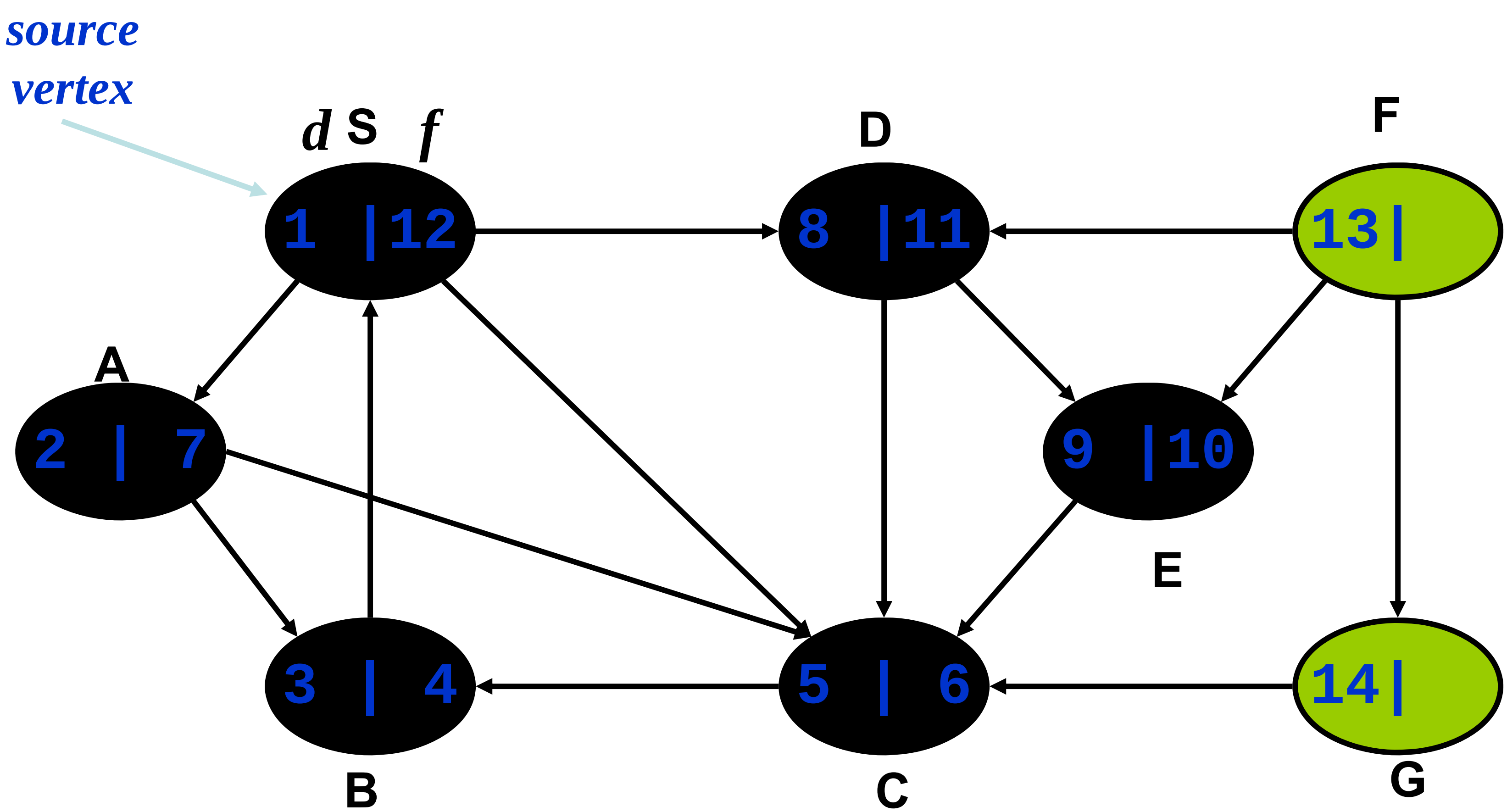
DFS Example



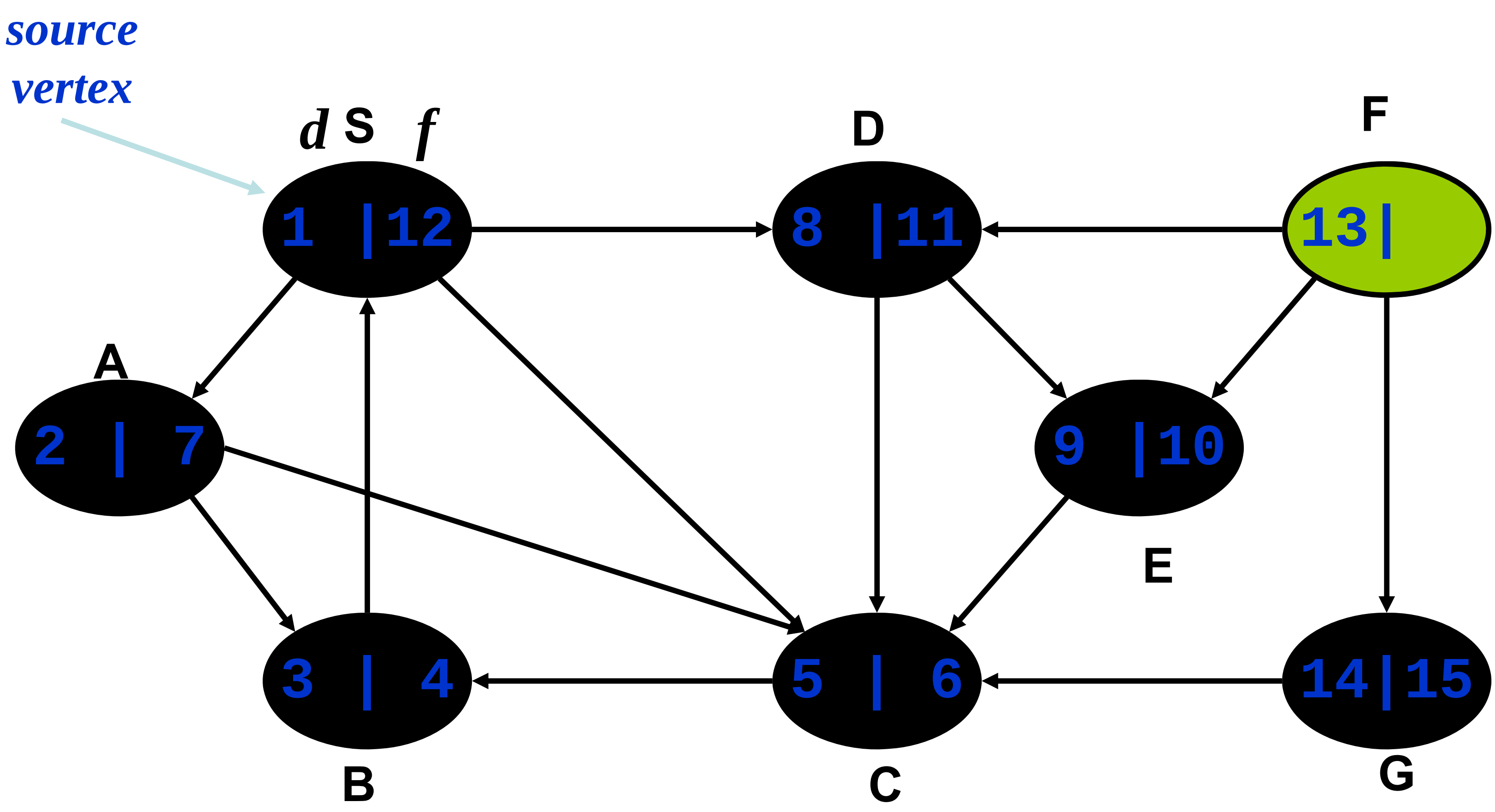
DFS Example



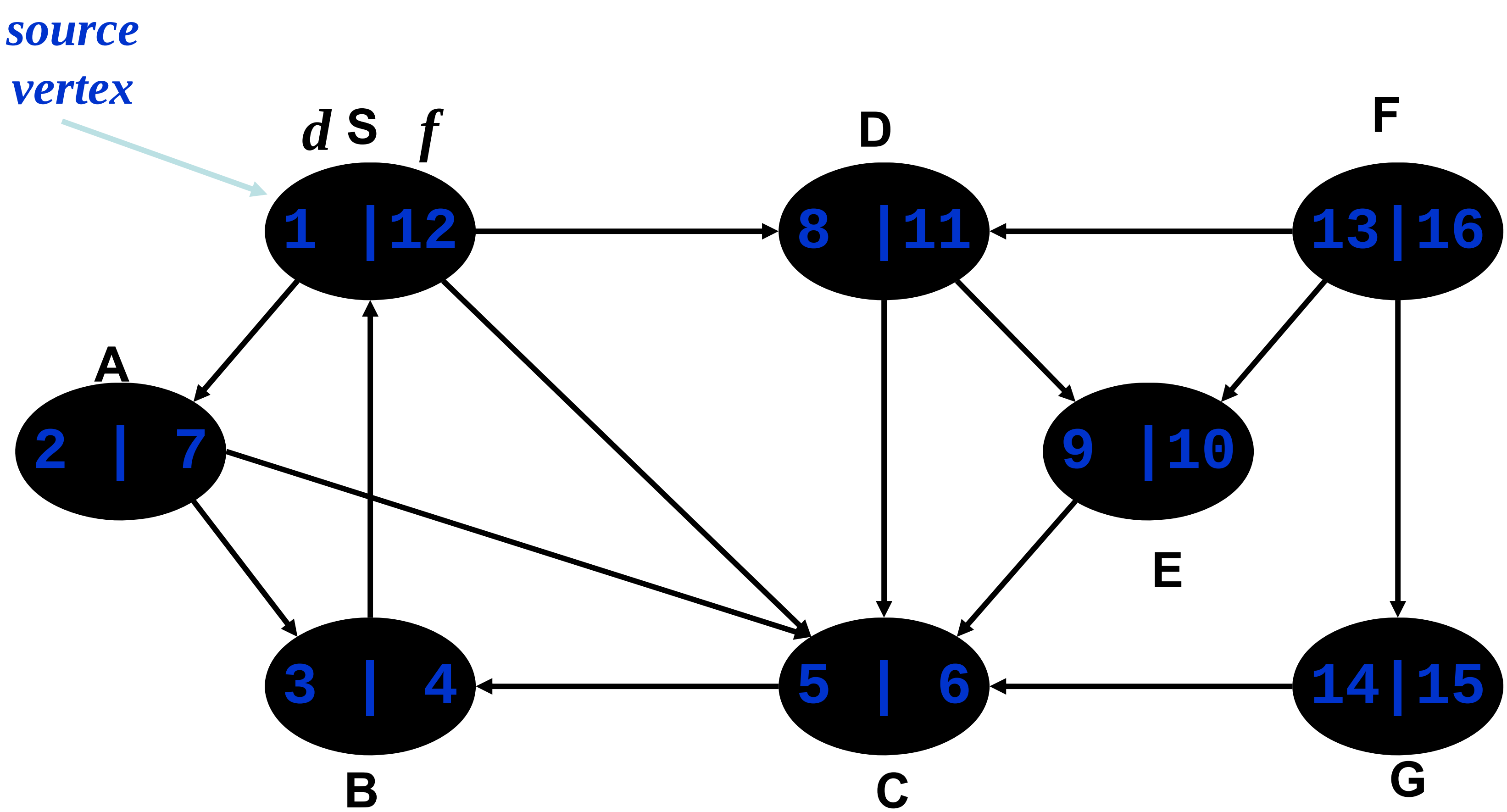
DFS Example



DFS Example



DFS Example



Depth-First Search: The Code

```
   Data: color[V], time,
     prev[V],d[V], f[V]
DFS(G) // where prog starts
{
   for each vertex u ∈ V
   {
      color[u] = WHITE;
       prev[u]=NIL;
       f[u]=inf; d[u]=inf;
   }
   time = 0;
   for each vertex u ∈ V
     if (color[u] == WHITE)
         DFS_Visit(u);
}
```

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

What will be the running time?

Depth-First Search: The Code

```
Data: color[V], time,
   prev[V],d[V], f[V]
DFS(G) // where prog starts
{
   for each vertex u ∈ V
   {
      color[u] = WHITE;
       prev[u]=NIL;
       f[u]=inf; d[u]=inf;
   }
   time = 0;
   for each vertex u ∈ V
     if (color[u] == WHITE)
        DFS_Visit(u);
}
```

O(V)

O(V)

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

O(V)

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

Depth-First Search: The Code

```
Data: color[V], time,
  prev[V],d[V], f[V]
DFS(G) // where prog starts
{
   for each vertex u ∈ V
   {
      color[u] = WHITE;
      prev[u]=NIL;
      f[u]=inf; d[u]=inf;
   }
   time = 0;
   for each vertex u ∈ V
     if (color[u] == WHITE)
         DFS_Visit(u);
}
```

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

BUT, there is actually a tighter bound.

How many times will DFS_Visit() actually be called?

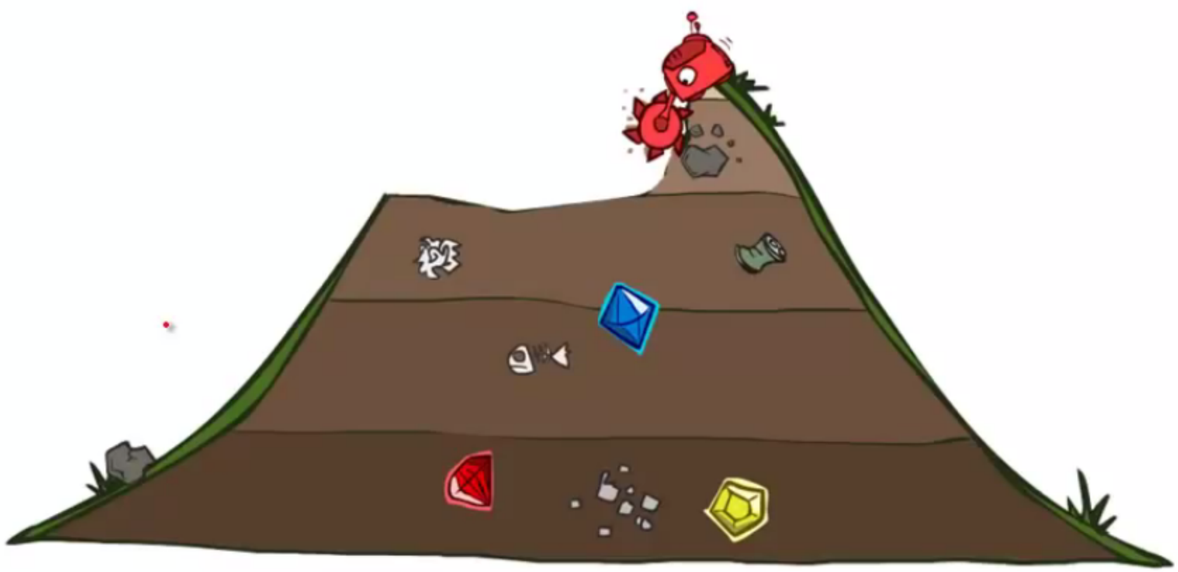
Depth-First Search: The Code

```
Data: color[V], time,
   prev[V],d[V], f[V]
DFS(G) // where prog starts
{
   for each vertex u ∈ V
   {
      color[u] = WHITE;
       prev[u]=NIL;
       f[u]=inf; d[u]=inf;
   }
   time = 0;
   for each vertex u ∈ V
     if (color[u] == WHITE)
         DFS_Visit(u);
}
```

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

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

Uniform Cost Search



Iterative deepening search

Function Iterative_Deepening_Search(*problem*) return *solution* or *failure*

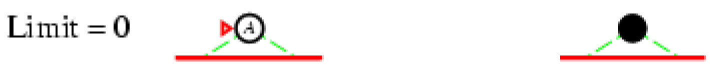
Inputs: *problem*, a problem

For *depth* ← 0 to ∞ do

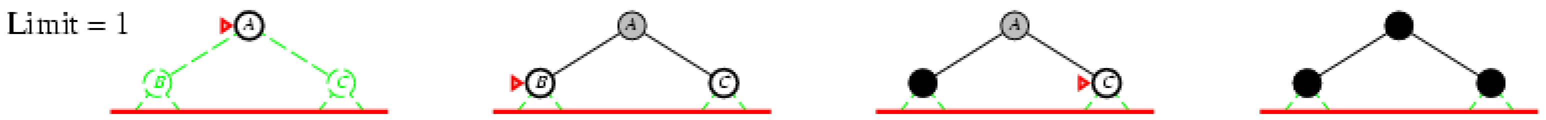
  result ← Depth_Limited_Search (*problem, depth*)

  if *result* ≠ cutoff then return *result*

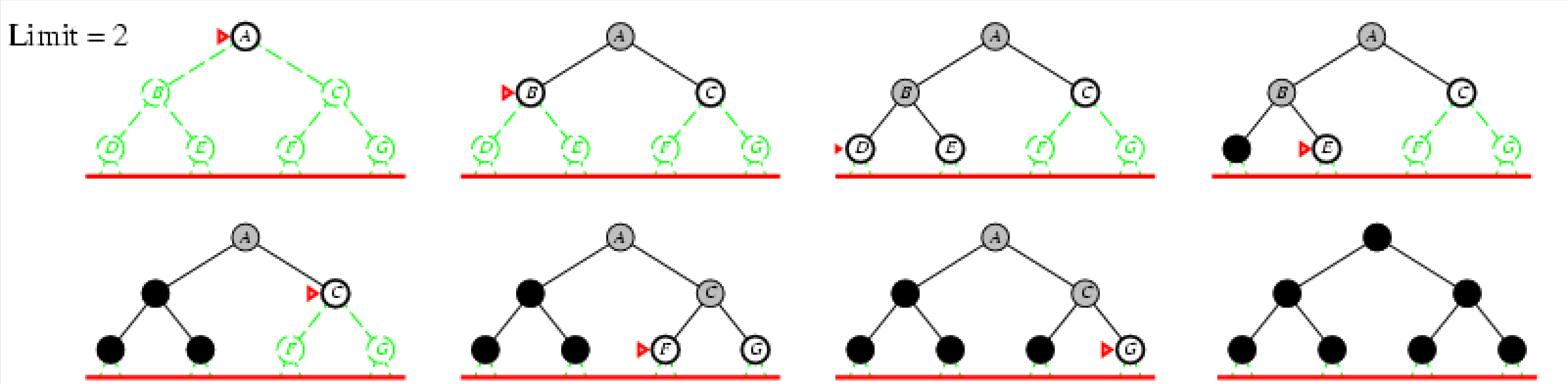
Iterative deepening search $l = 0$



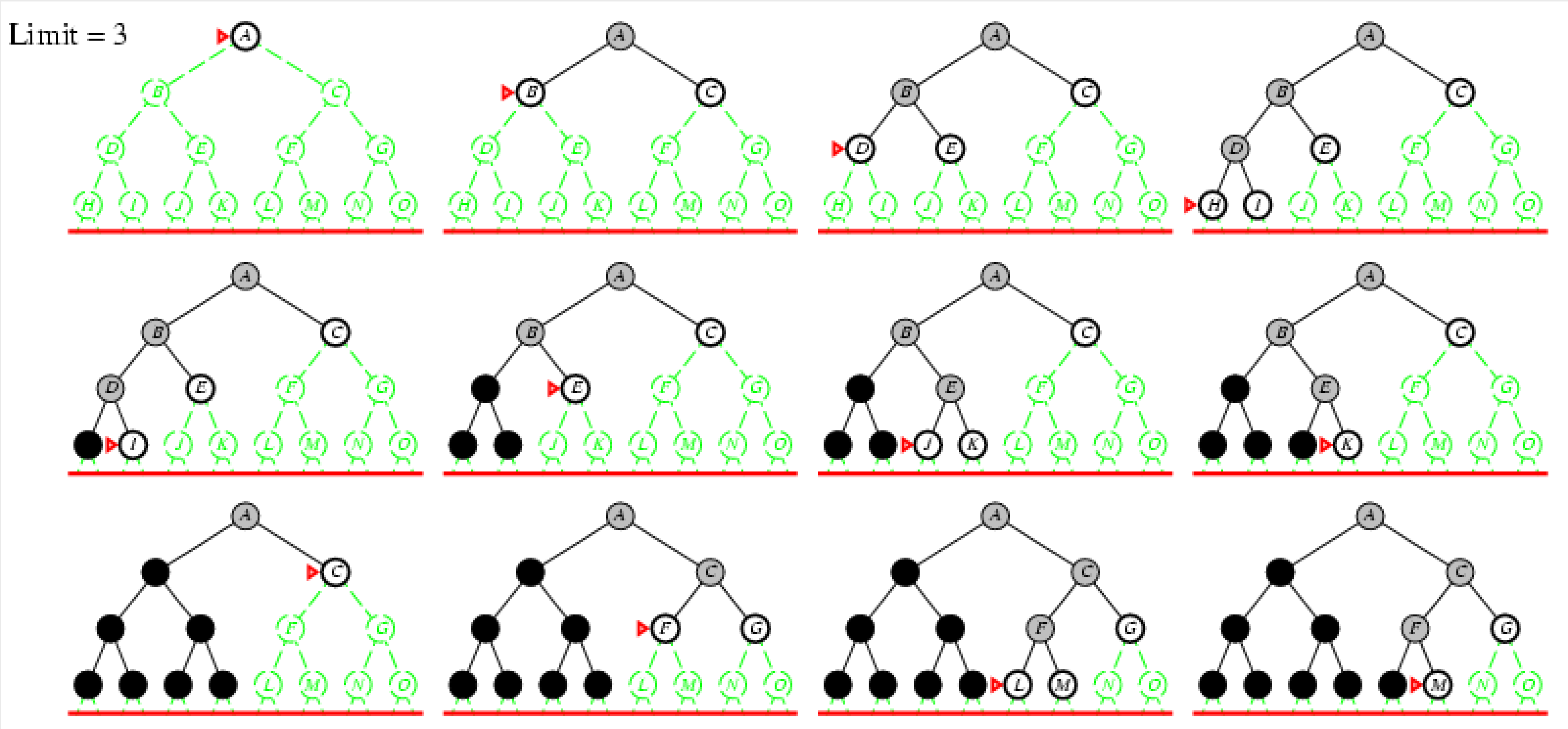
Iterative deepening search l =1



Iterative deepening search *l* =2



Iterative deepening search *l* =3



Properties of iterative deepening search

- **Complete?**

  Yes

- **Optimal?**

  Yes, if step cost = 1

- **Time?**

  $(d+1)b^0 + d\, b^1 + (d-1)b^2 + \ldots + b^d$

- **Space?**

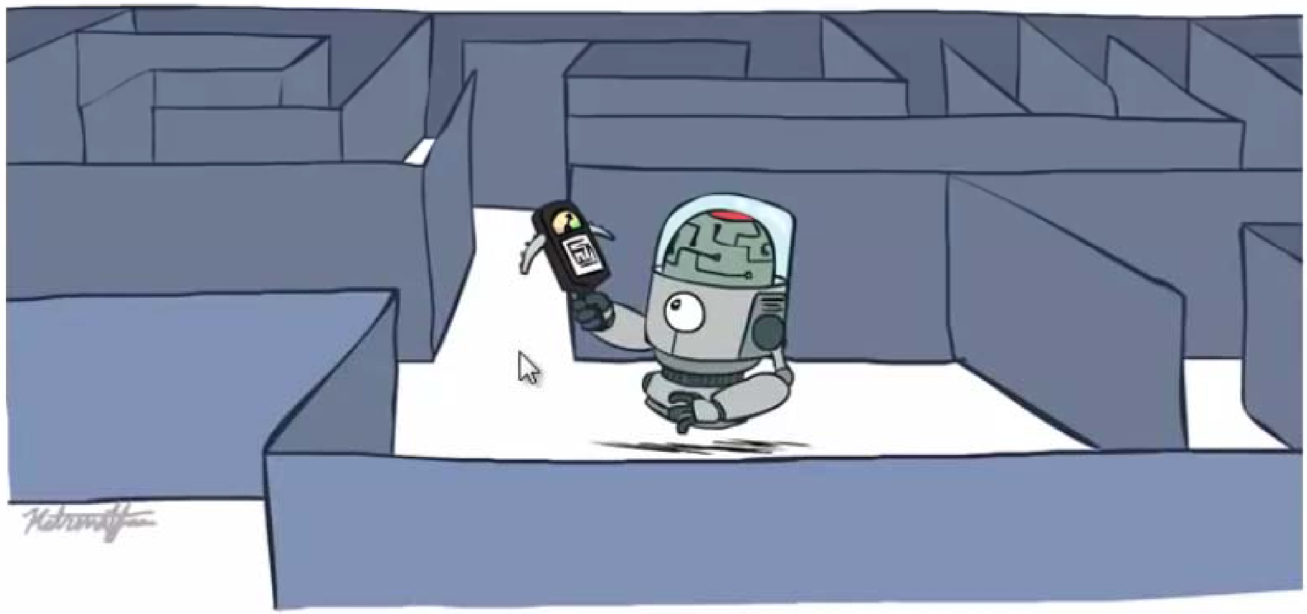
  $O(bd)$

Review: Uninformed search strategies

Algorithm	Complete?	Optimal?	Time complexity	Space complexity
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	If all step costs are equal	$O(b^d)$	$O(bd)$
UCS	Yes	Yes	Number of nodes with $g(n) \leq C^*$	

b: maximum branching factor of the search tree
d: depth of the optimal solution
m: maximum length of any path in the state space
C*: cost of optimal solution
g(n): cost of path from start state to node n

Informed Search



Today

- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search

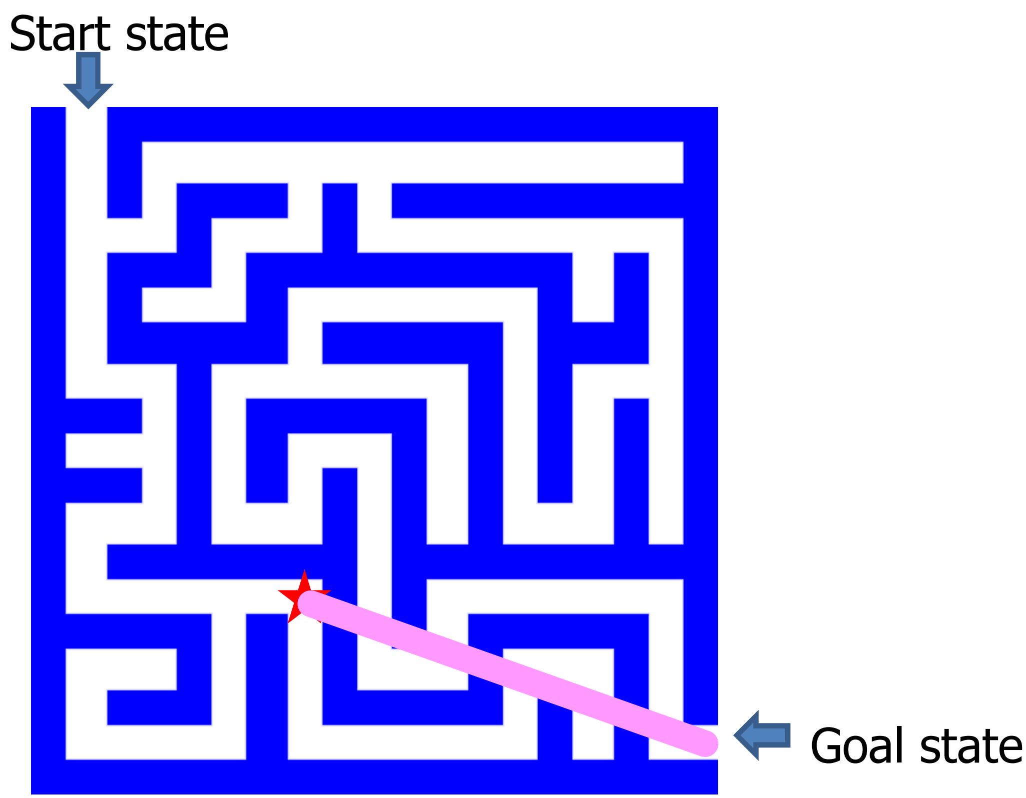
- Graph Search

Informed Search Strategies

- Informed search algorithm have some idea of where to look for solutions.
- This uses problem specific knowledge and can find solutions more efficiently than uninformed search.
- These strategies often depend on the use of heuristic information (heuristic search function).
- Heuristic search function ***h(n),*** is estimated cost of the cheapest path from node n to goal node.
- If n is goal then h(n)=0.

Heuristic function

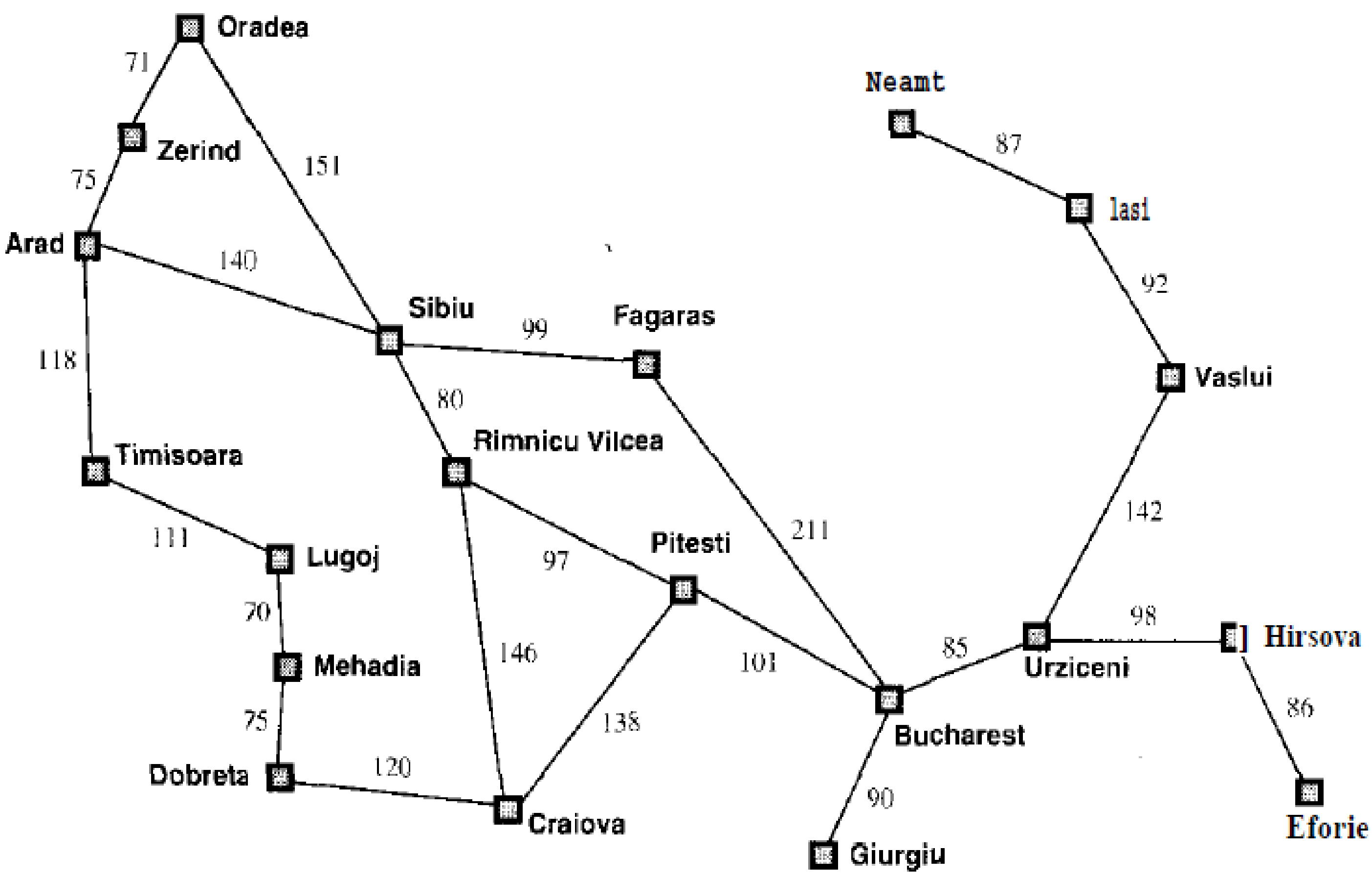
- **Heuristic function** *h*(*n*) estimates the cost of reaching goal from node *n*
- Example:



Heuristic Information

- Information about the problem:
  - The nature of the states
  - The cost of transforming from one state to another
  - The promise of taking certain path
  - The characteristics of the goals
- This information can often be expressed in the form of heuristic evaluation function *f(n,g)*, a function of the node n and/or the goal g.

Romania with step costs in km



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

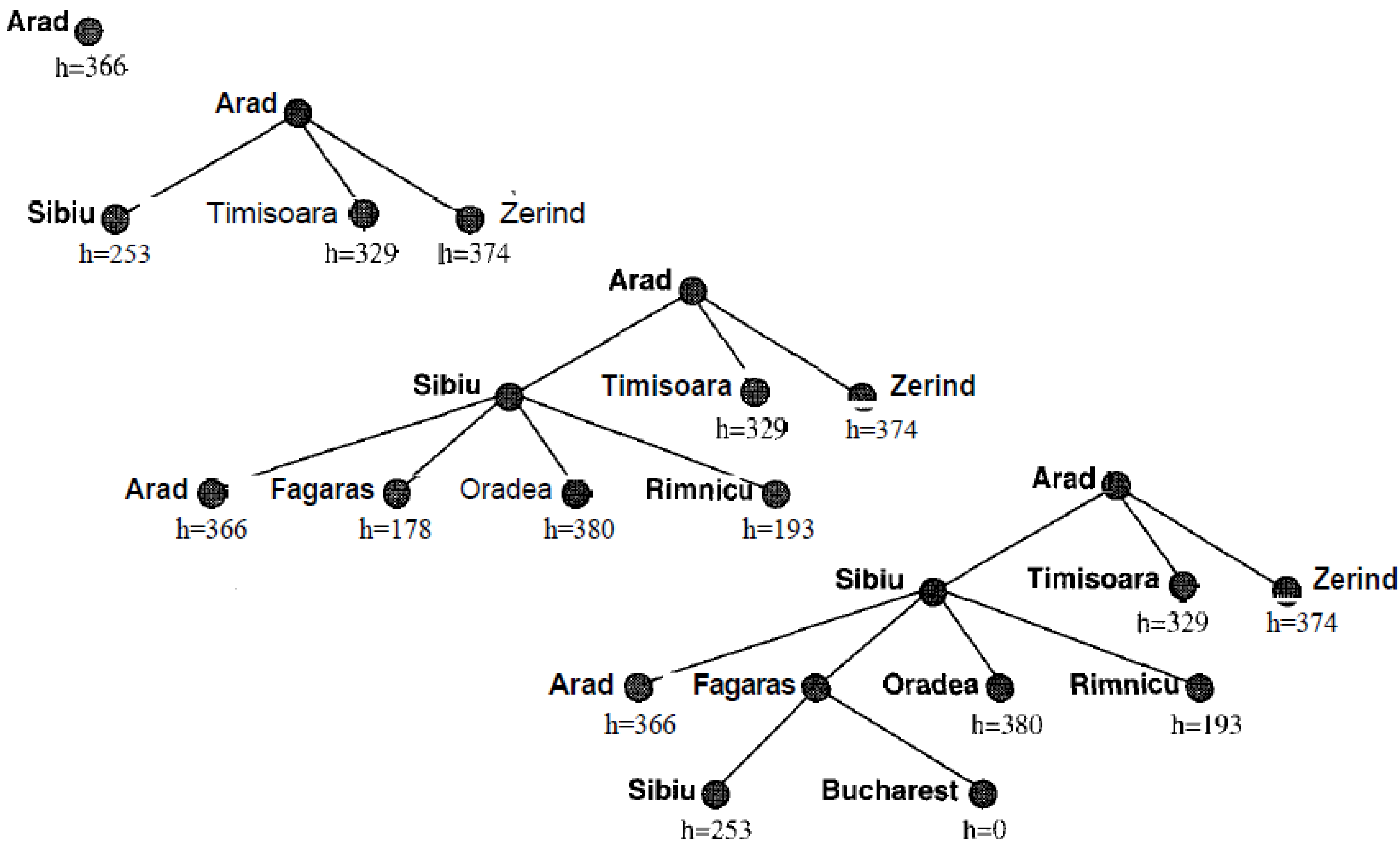
- Evaluation function $f(n) = h(n)$ (heuristic)

  = estimate of cost from n to *goal*

- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that appears to

  be closest to goal.

Greedy best-first search example



Greedy best-first search example

- Not Optimal. But performs quite well.
- The path it found via Sibiu and Fagaras to Bucharest is 32 miles longer than the path through Rimnicu Vilcea and

  Pitesti.
- Incomplete : start down an infinite path and never return to try other possibilities.
- Susceptible to false start. Try to go from Iasi to Fagaras.
  - » Oscillate between Iasi and Neamt.
  - » Leads to dead end.
  - » Should avoid repeated states

A* Search

A* Search

UCS

Greedy

A* Search

UCS

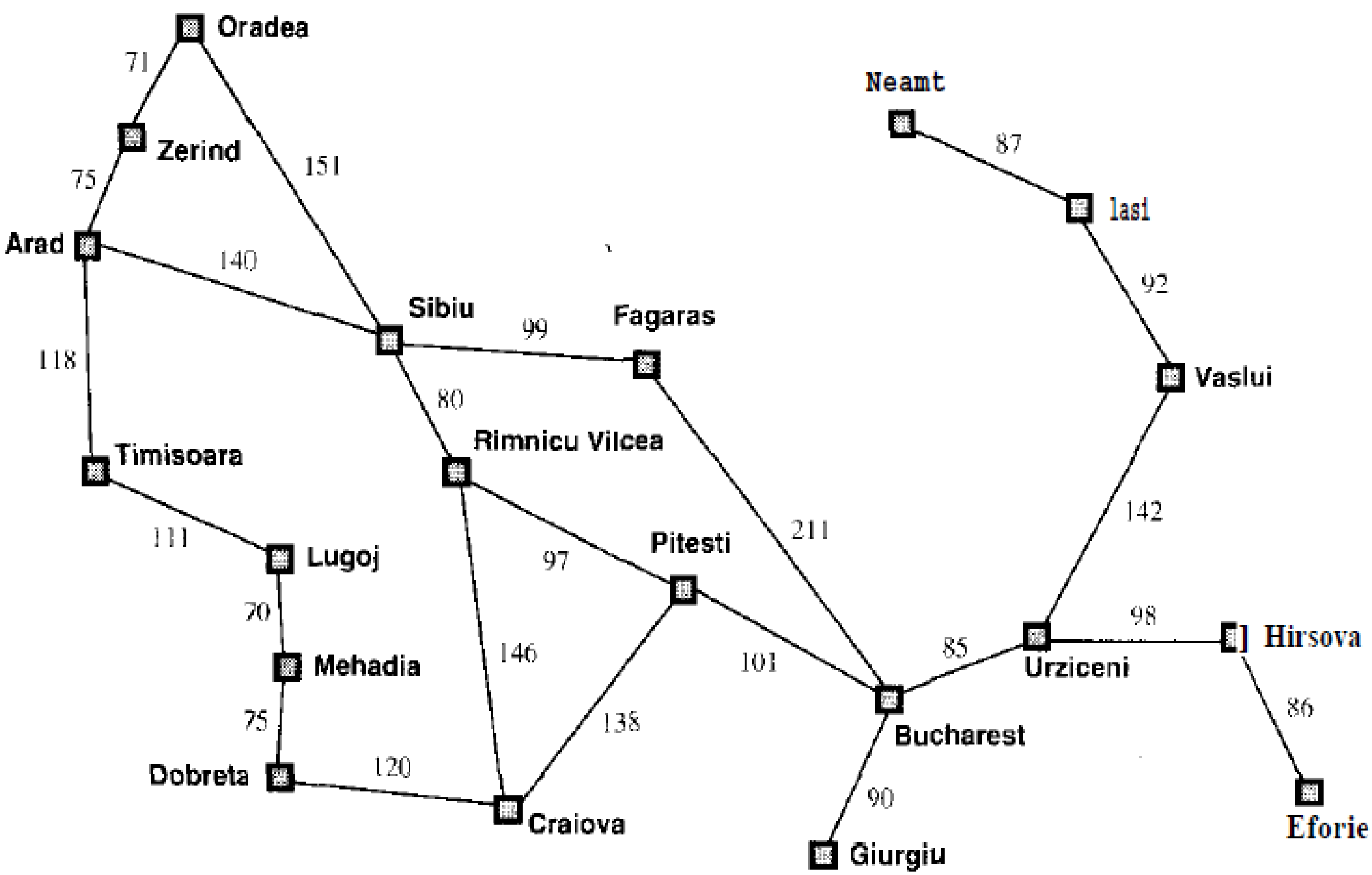
Greedy

A*

A^* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n
- Best First search has $f(n)=h(n)$
- Uniform Cost search has $f(n)=g(n)$

Romania with step costs in km



Straight-line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search

- Idea: avoid expanding paths that are already expensive
- The **evaluation function** $f(n)$ is the estimated total cost of the path through node n to the goal:

$$f(n) = g(n) + h(n)$$

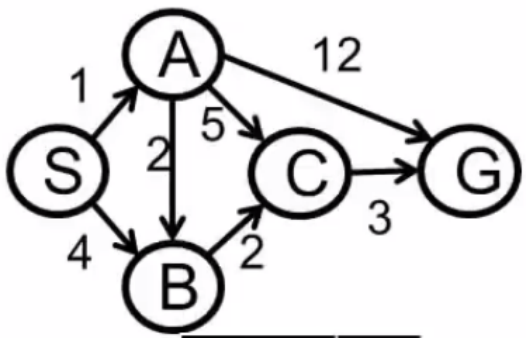
$g(n)$: cost so far to reach n (path cost)

$h(n)$: estimated cost from n to goal (heuristic)

A* Tree Search

Search Tree Visualization

State-Space Graph



State	H
S	7
A	6
B	2
C	1
G	0

Thank You