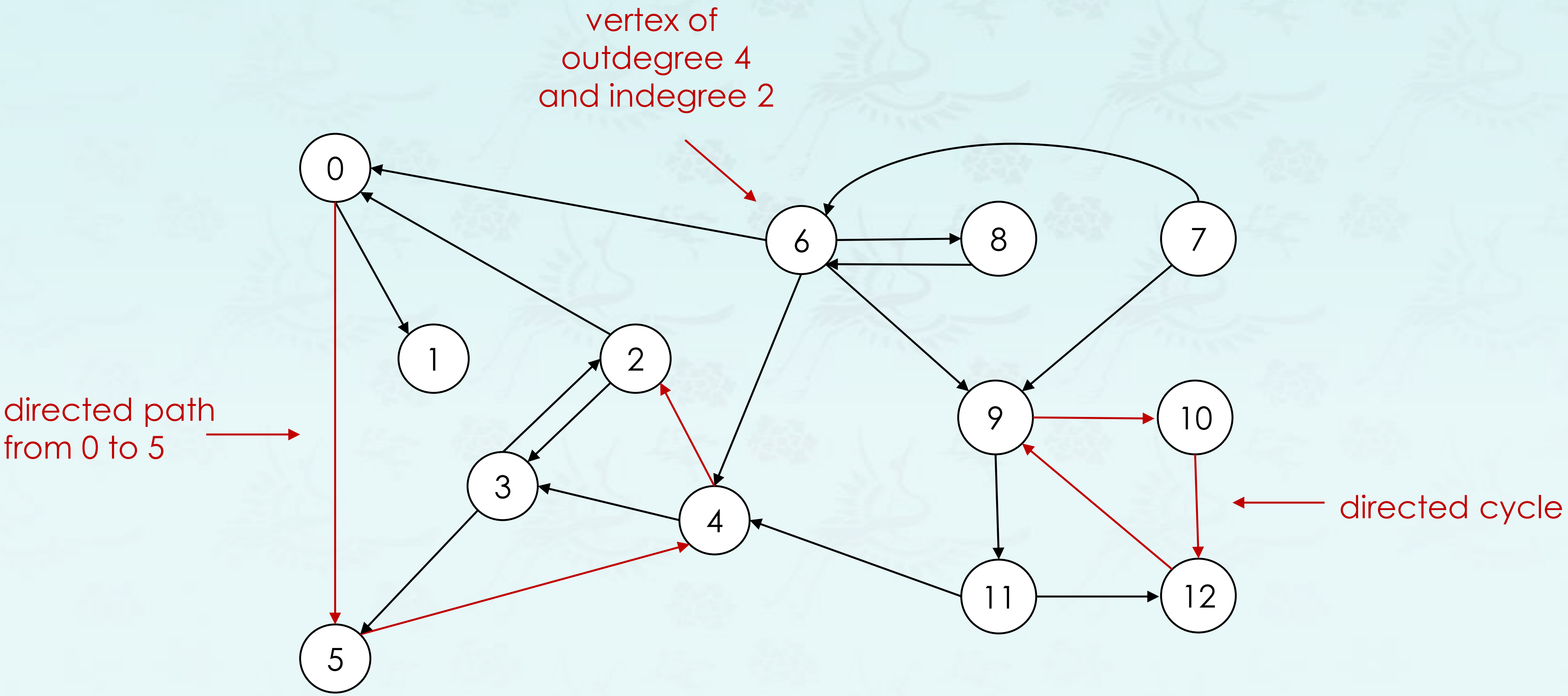
Data Structures
Chapter 7: Graph

1. Introduction
   - Terminology, Representation, ADT
2. Basic Operations
   - DFS, CC, BFS, Processing
3. **Digraph and Applications**
4. Minimum Spanning Tree(MST)

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

Directed graph

- **Digraph:** Set of vertices connected pairwise by directed edges.



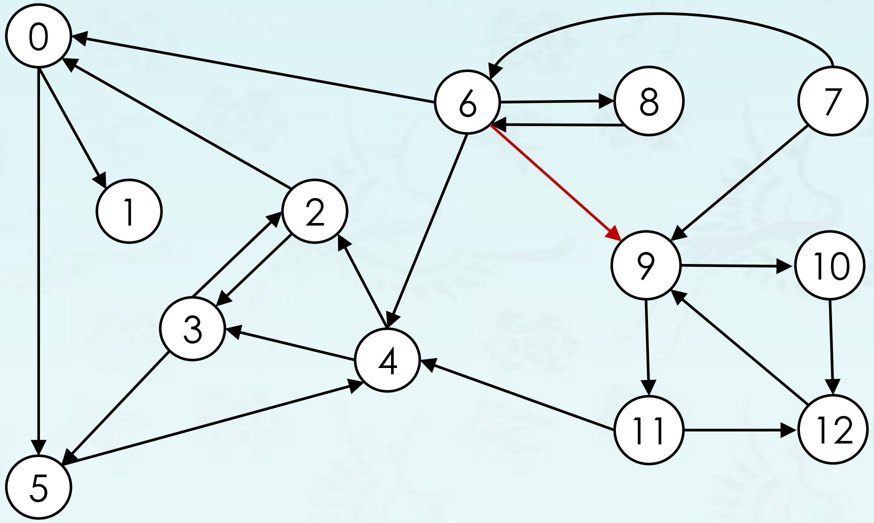
Directed graph – ADT

- **Digraph:** Set of vertices connected pairwise by directed edges.

	Directed Graph	
	`digraph(int V)`	create an empty digraph with V vertices
`void`	`addEdge(int v, int w)`	add a directed edge v → w
`vector<int>`	`adj(int v)`	vertices pointing from v
`int`	`V()`	member of vertices
`int`	`E()`	member of edges
`digraph`	`reverse()`	reverse of the digraph
`string`	`toString()`	string representation

Adjacency-lists digraph representation

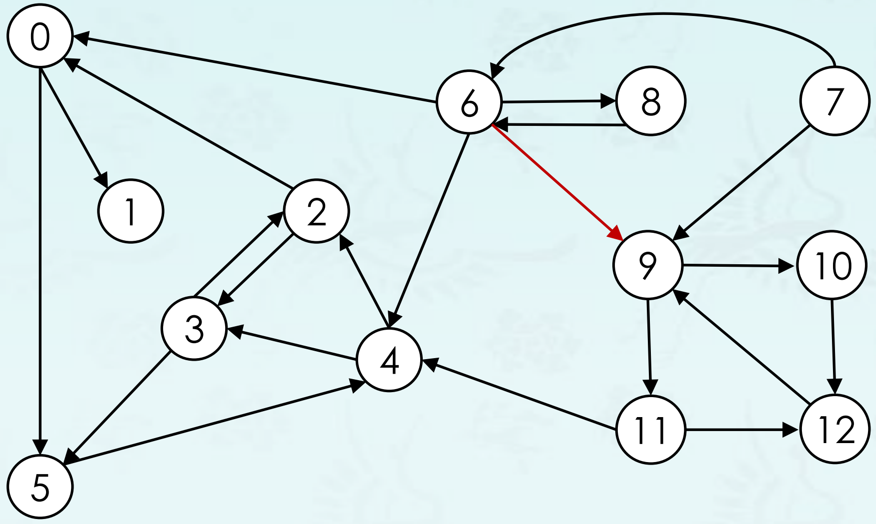
- **Maintain vertex-indexed array of lists.**



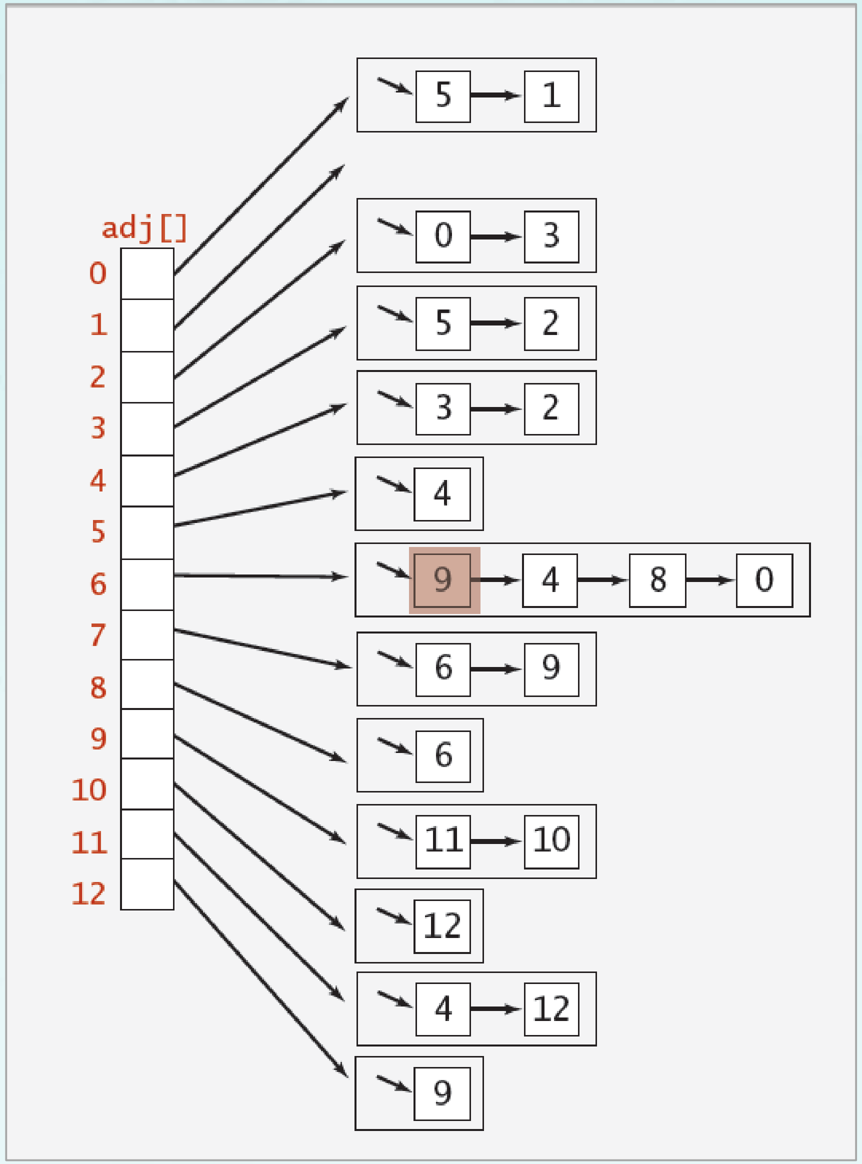

```
myG.txt
13      V
22      E
4 2
2 3
3 2
6 0
0 1
2 0
11 12
12 9
9 10
9 11
7 9
10 12
11 4
4 3
3 5
6 8
8 6
5 4
0 5
6 4
6 9
7 6
```

Adjacency-lists digraph representation

- **Maintain vertex-indexed array of lists.**




```
myG.txt
13   ← V
22   ← E
4 2
2 3
3 2
6 0
0 1
2 0
11 12
12 9
9 10
9 11
7 9
10 12
11 4
4 3
3 5
6 8
8 6
5 4
0 5
6 4
6 9
7 6
```



Adjacency-lists graph representation in Java

```
public class Graph {

  private final int V;
  private Bag<Integer>[] adj;

  public Graph(int V) {
    this.V = V;
    adj = (Bag<Integer>[]) new Bag[V];
    for (int v = 0; v < V; v++)
      adj[v] = new Bag<Integer>();
  }

  public void addEdge(int v, int w) {
    adj[v].add(w);
    adj[w].add(v);
  }

  public Iterable<Integer> adj(int v) {
    return adj[v];
  }
}
```

adjacency lists
(using Bag data type)

create empty graph
with V vertices

add edge v-w
(parallel edges and
self-loops allowed)

iterator for vertices
adjacent to v

Adjacency-lists digraph representation in Java

```
public class Digraph {

  private final int V;
  private Bag<Integer>[] adj;

  public Digraph(int V) {
    this.V = V;
    adj = (Bag<Integer>[]) new Bag[V];
    for (int v = 0; v < V; v++)
      adj[v] = new Bag<Integer>();
  }

  public void addEdge(int v, int w) {
    adj[v].add(w);

  }

  public Iterable<Integer> adj(int v) {
    return adj[v];
  }
}
```

adjacency lists
(using Bag data type)

create empty graph
with V vertices

add edge v -> w

iterator for vertices
pointing from v

Digraph representations

In practice: Use adjacency-lists representation.

- Algorithms based on iterating over vertices pointing from v.
- Real-world digraphs tend to be **sparse.**

huge number of vertices,
small average vertex degree

representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1	1	V
adjacency lists	$E + V$	1	$outdegree(v)$	$outdegree(v)$

Depth-first search in digraphs

Same methods as for undirected graphs:

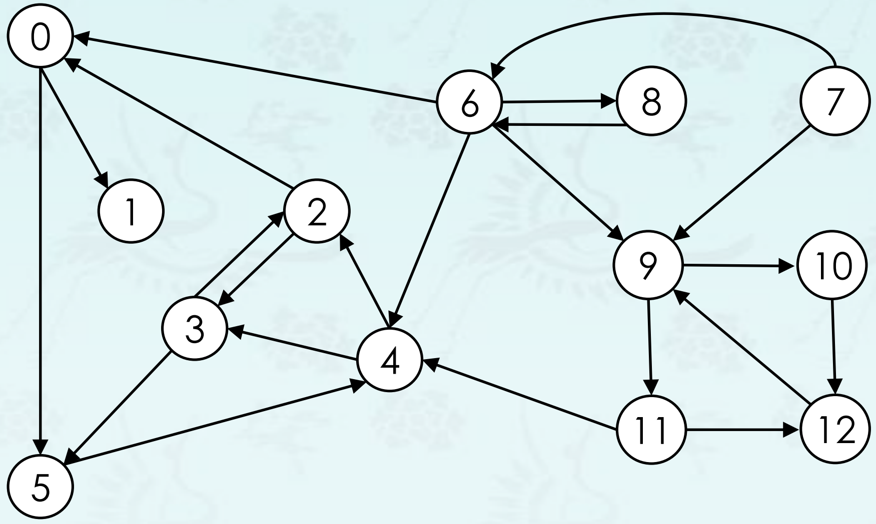
- Every undirected graph is digraph (with edges in both directions)
- DFS is a digraph algorithm.

DFS (to visit a vertex v)
• Mark v as visited. • Recursively visit all unmarked vertices w adjacent to v.

Depth-first search in digraphs

To visit a vertex v:

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v.



a directed graph

```
myG.txt
13    V
22    E
4  2
2  3
3  2
6  0
0  1
2  0
11  12
12  9
9  10
9  11
7  9
10  12
11  4
4  3
3  5
6  8
8  6
5  4
0  5
6  4
6  9
7  6
```

Digraph Quiz

To visit a vertex v:

- Suppose that a digraph G is represented using the adjacency-lists representation. What is the order of growth of the running time to find all vertices that **point to** a given vertex v or indegree of v?

___ indgree(v)

___ outdegree(v)

___ V

___ E

___ V * E

___ V + E

Digraph Quiz

To visit a vertex v:

- Suppose that a digraph G is represented using the adjacency-lists representation. What is the order of growth of the running time to find all vertices that **point to** a given vertex v or indegree of v?

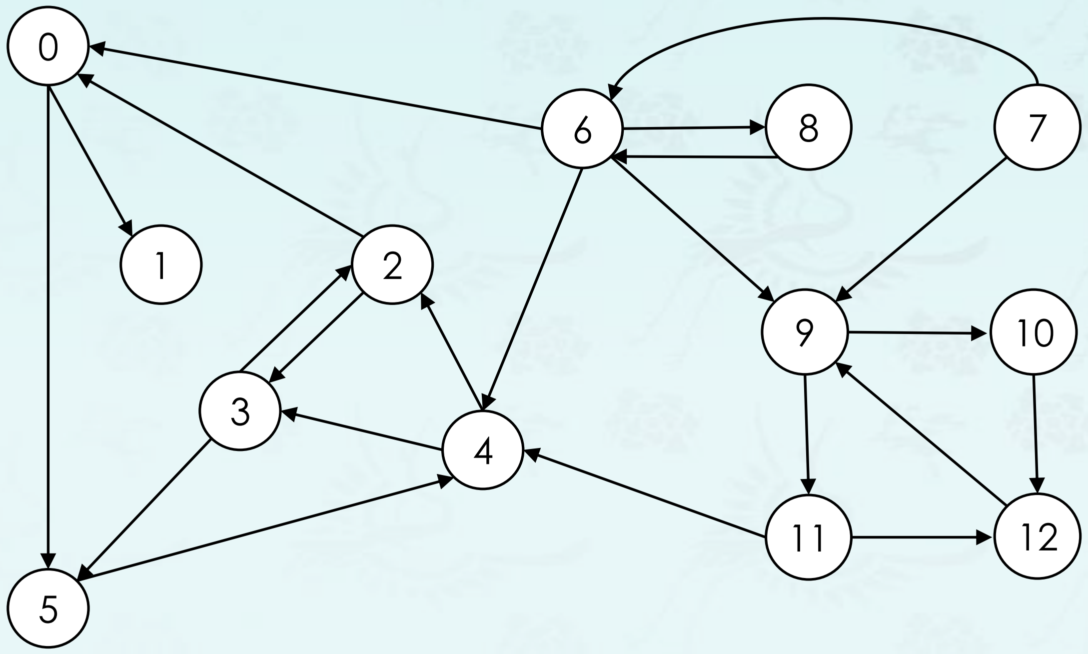
___ indgree(v)
___ outdegree(v)
___ V
___ E
___ V * E
___ V + E

Solution: You must scan through each of the V adjacency lists and each of the E edges. If this were a common operation in digraph-processing problems, you could associate two adjacency lists with each vertex—one containing all of the vertices pointing from v (as usual) and one containing all of the vertices pointing to v. (V + E)

Depth-first search in digraphs

To visit a vertex v:

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v.



a directed graph

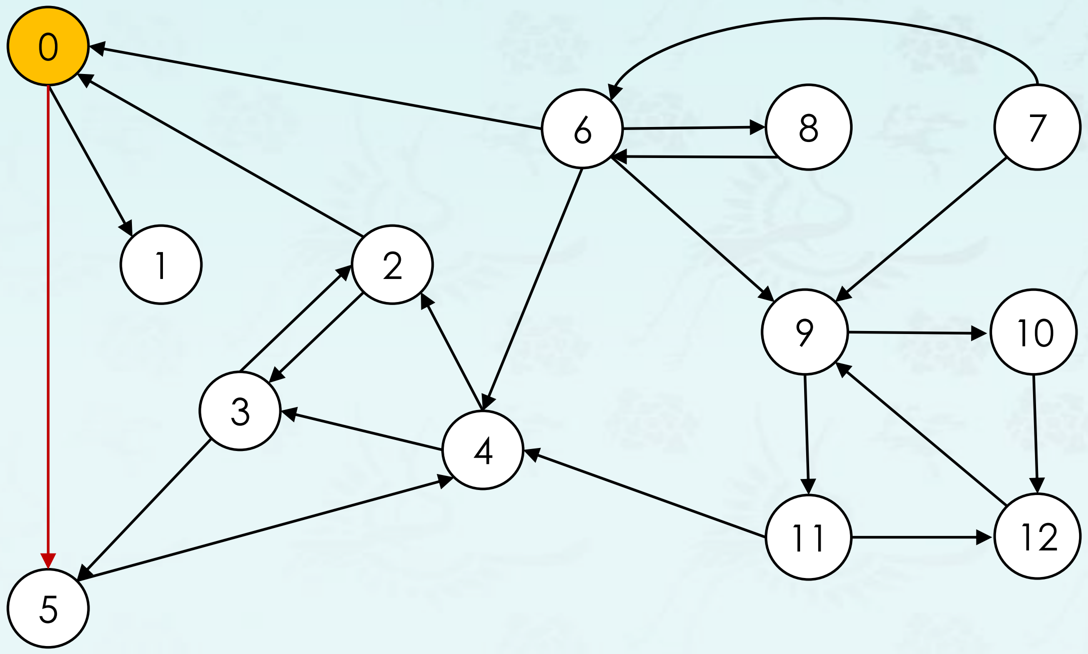
```
myG.txt
13  ←  V
22  ←  E
4 2
2 3
3 2
6 0
0 1
2 0
11 12
12 9
9 10
9 11
7 9
10 12
11 4
4 3
3 5
6 8
8 6
5 4
0 5
6 4
6 9
7 6
```

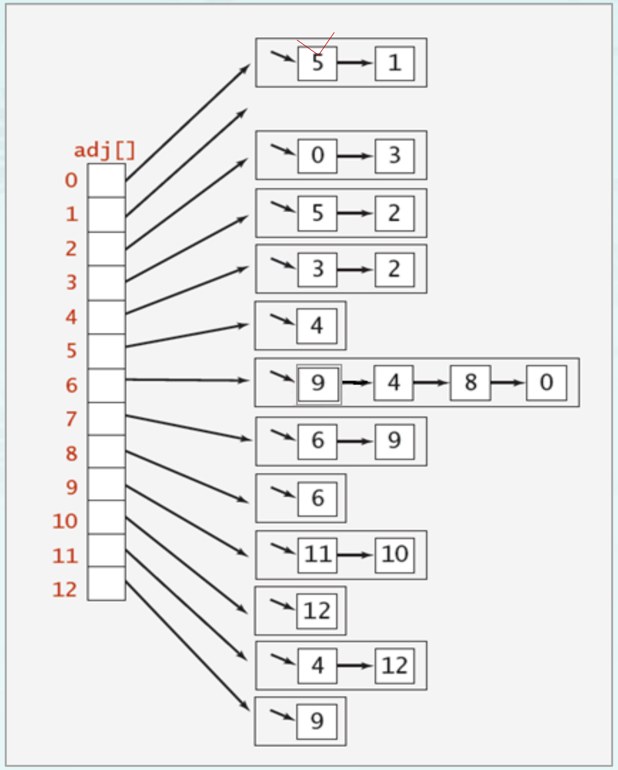
Depth-first search in digraphs

To visit a vertex v:

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v.

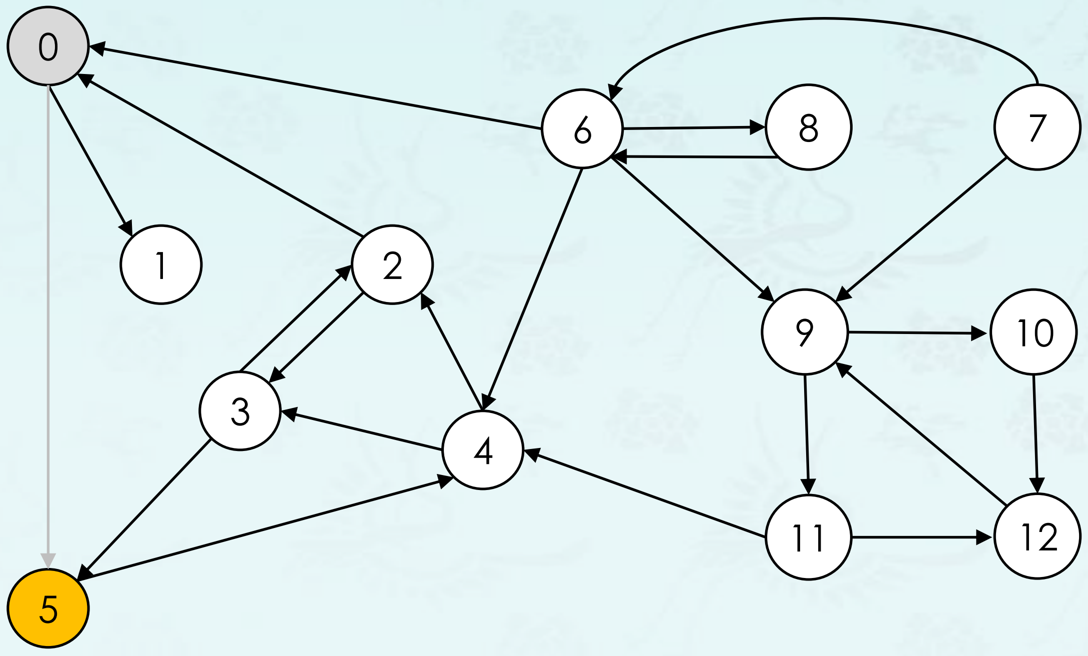
v	marked[]	parent[v]
0	T	–
1	F	–
2	F	–
3	F	–
4	F	–
5	F	–
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

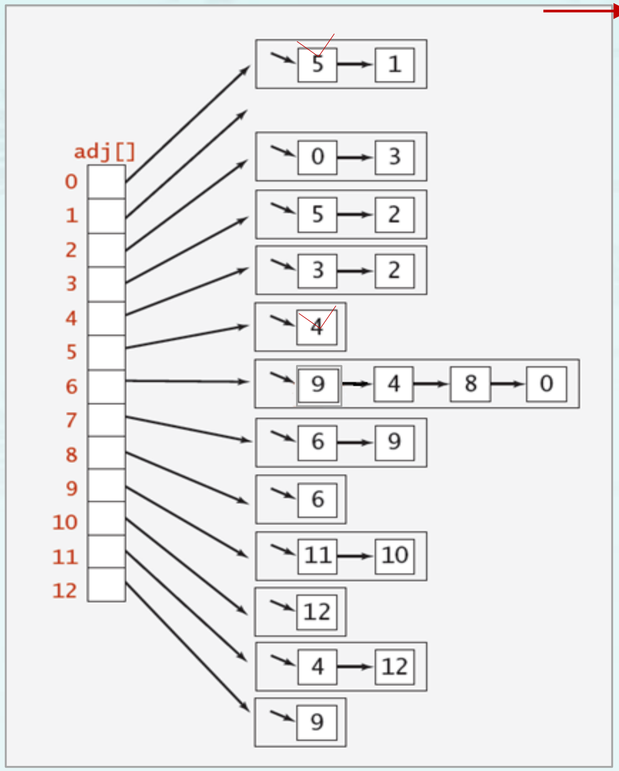




visit 0: check 5 and check 1

Depth-first search in digraphs

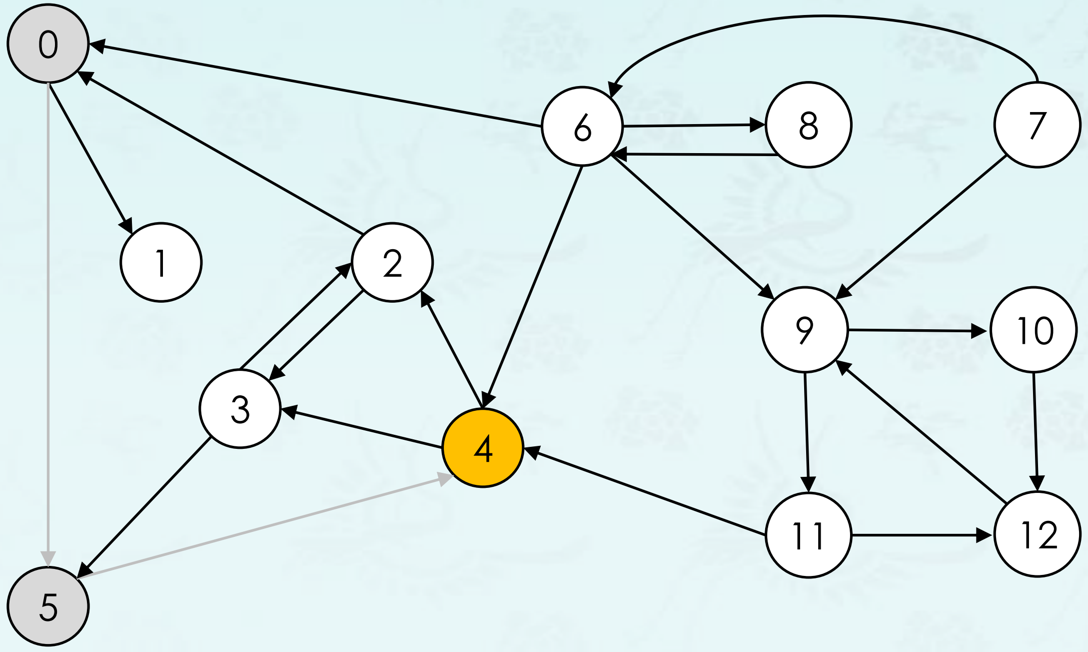


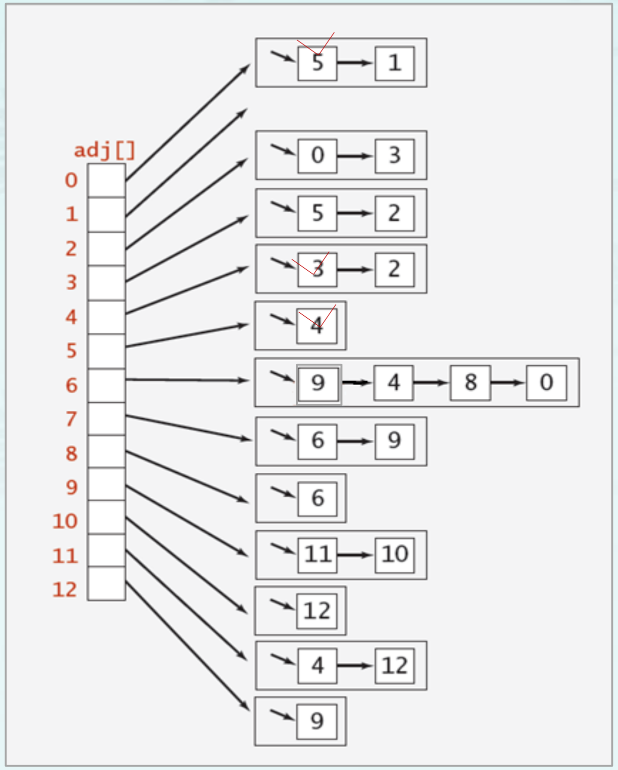


v	marked[]	parent[v]
0	T	–
1	F	–
2	F	–
3	F	–
4	F	–
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

visit 5: check 4

Depth-first search in digraphs

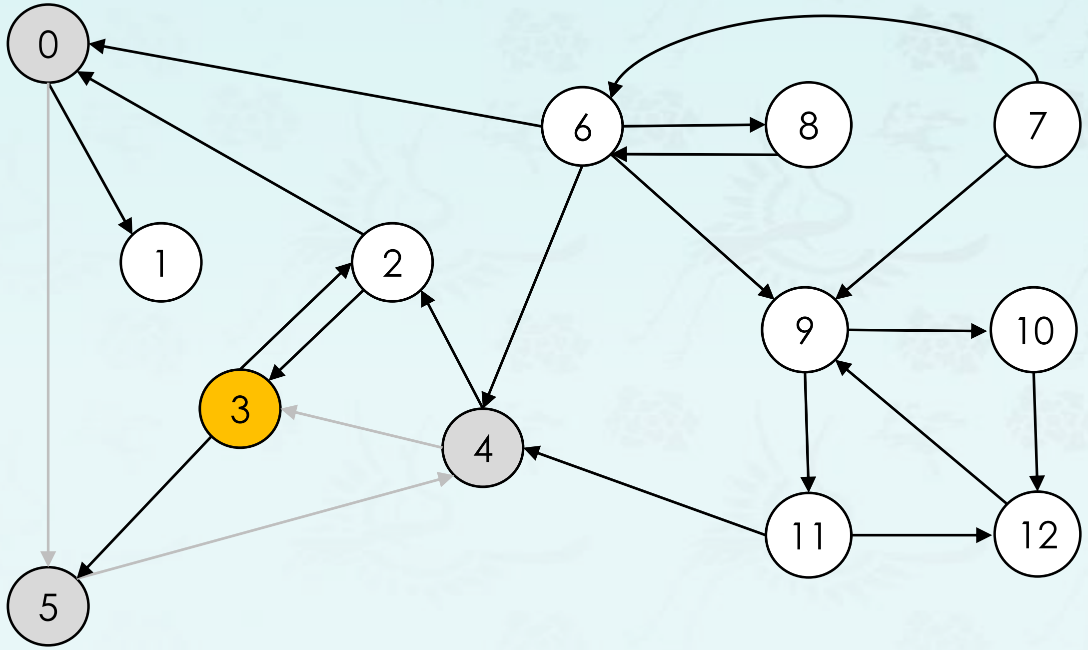


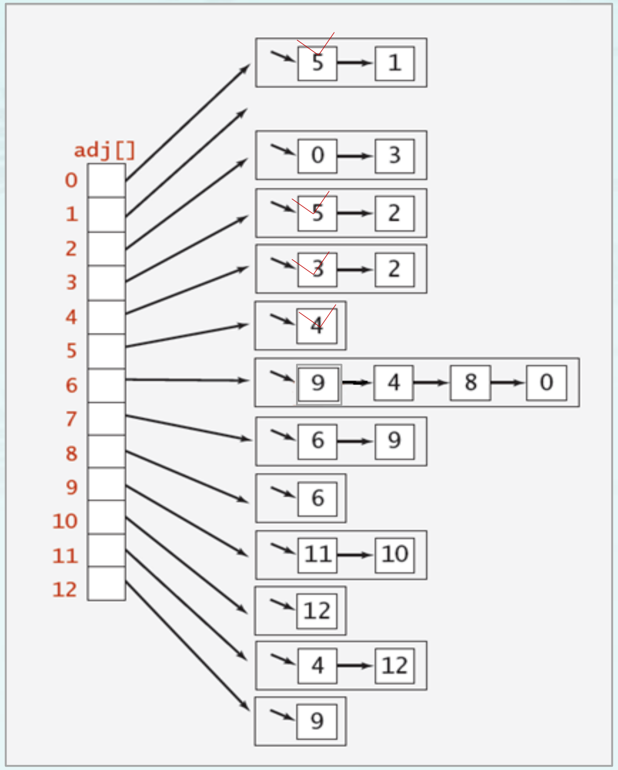


v	marked[]	parent[v]
0	T	–
1	F	–
2	F	–
3	F	–
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

visit 4: check 3 **and check 2**

Depth-first search in digraphs

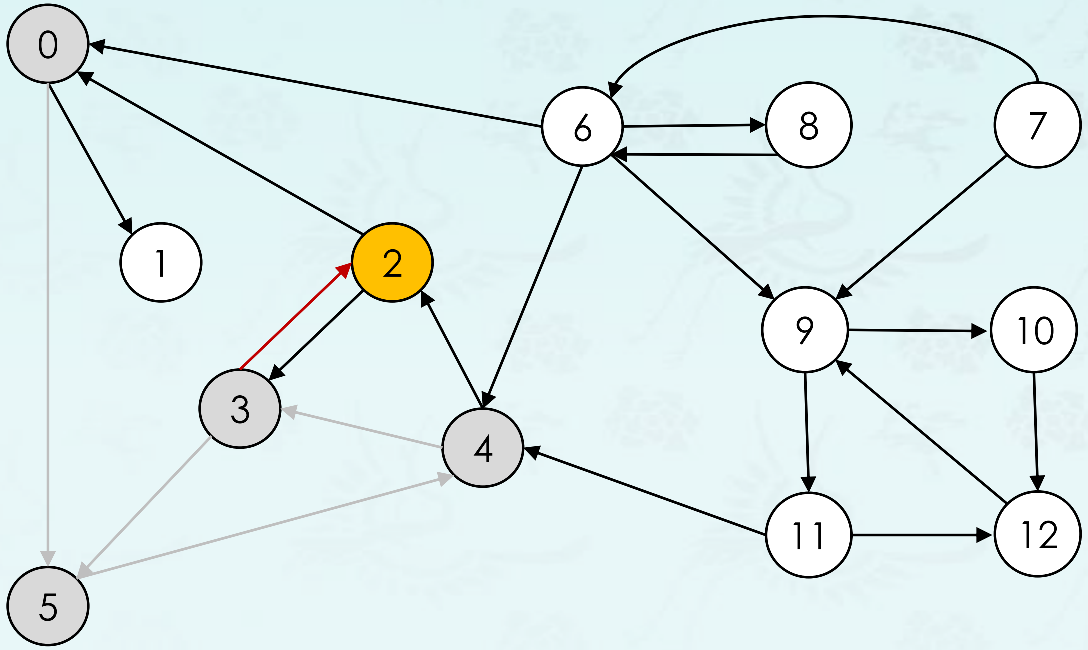


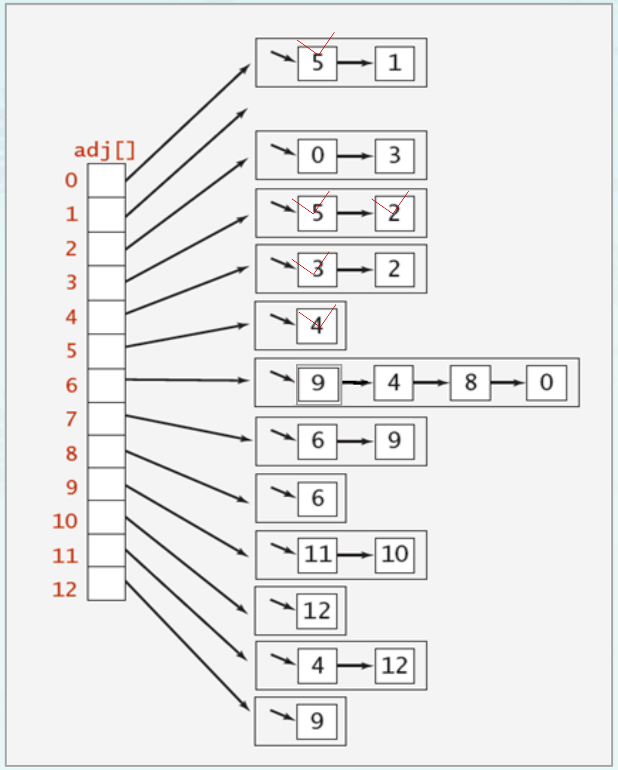


v	marked[]	parent[v]
0	T	–
1	F	–
2	F	–
3	T	4
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

visit 3: check 5 and check 2

Depth-first search in digraphs

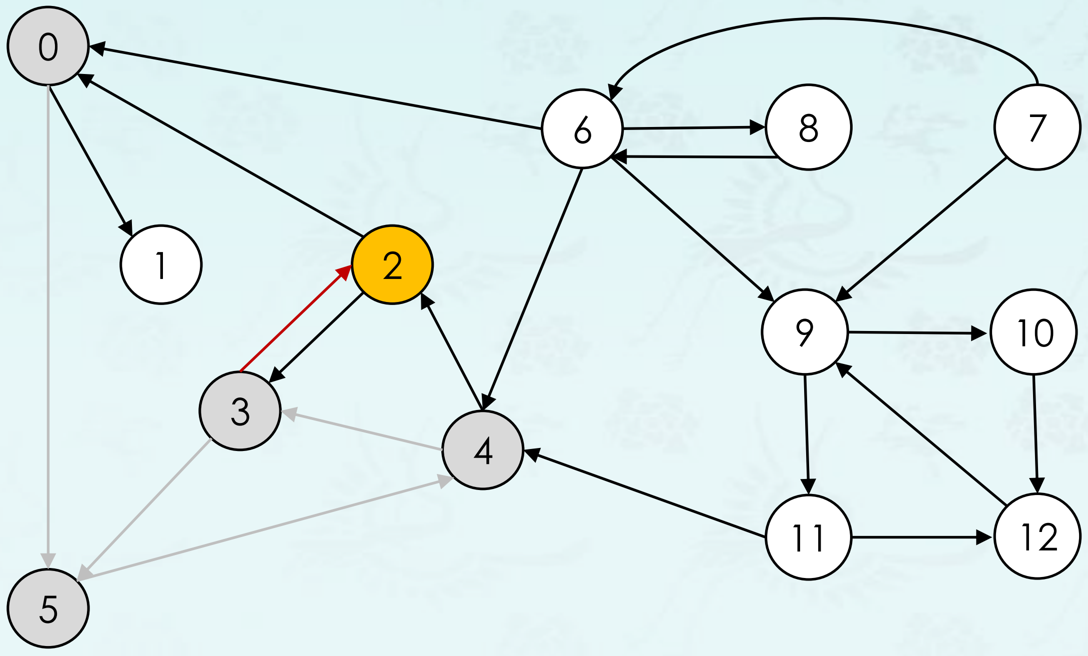


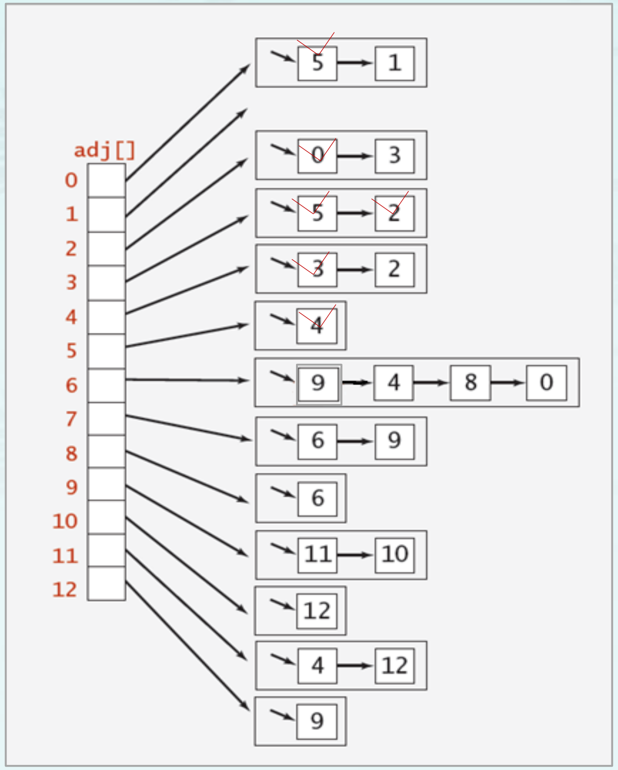


v	marked[]	parent[v]
0	T	–
1	F	–
2	F	–
3	T	4
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

visit 3: check 5 **and check 2**

Depth-first search in digraphs

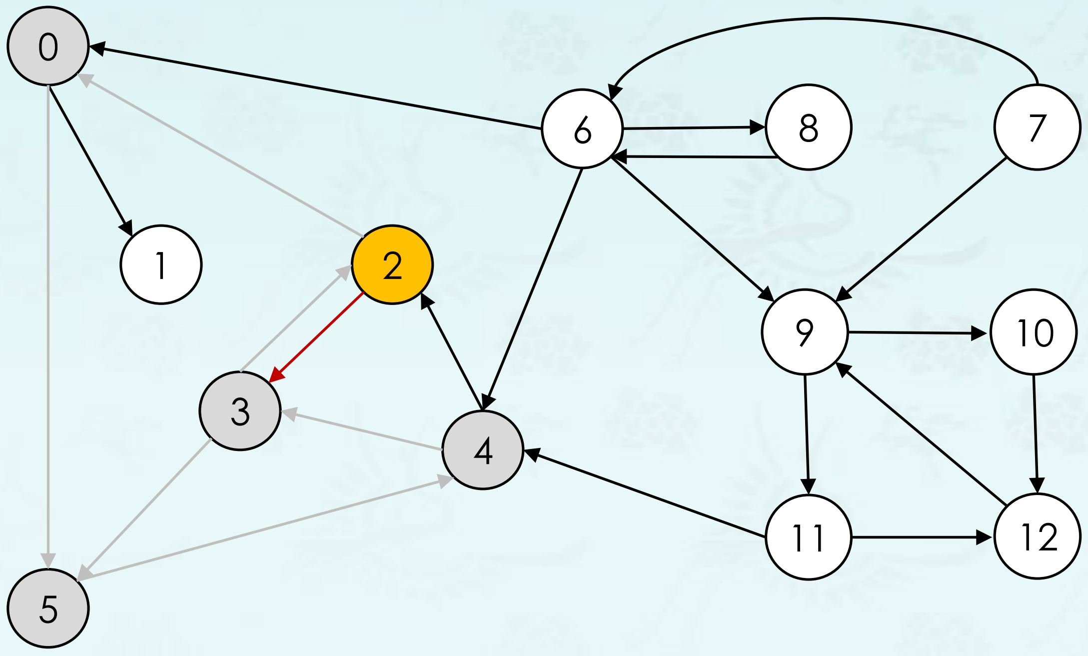


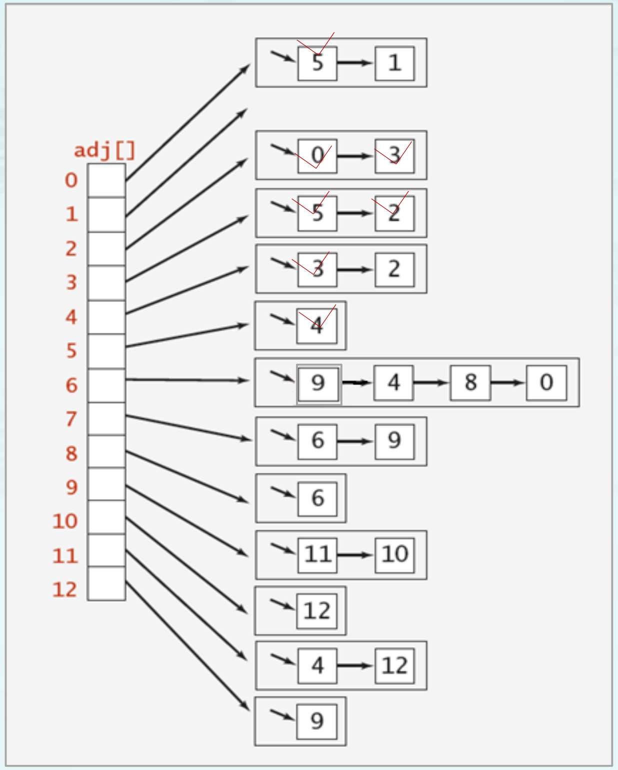


v	marked[]	parent[v]
0	T	–
1	F	–
2	T	3
3	T	4
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

visit 2: **check 0** **and check 3**

Depth-first search in digraphs



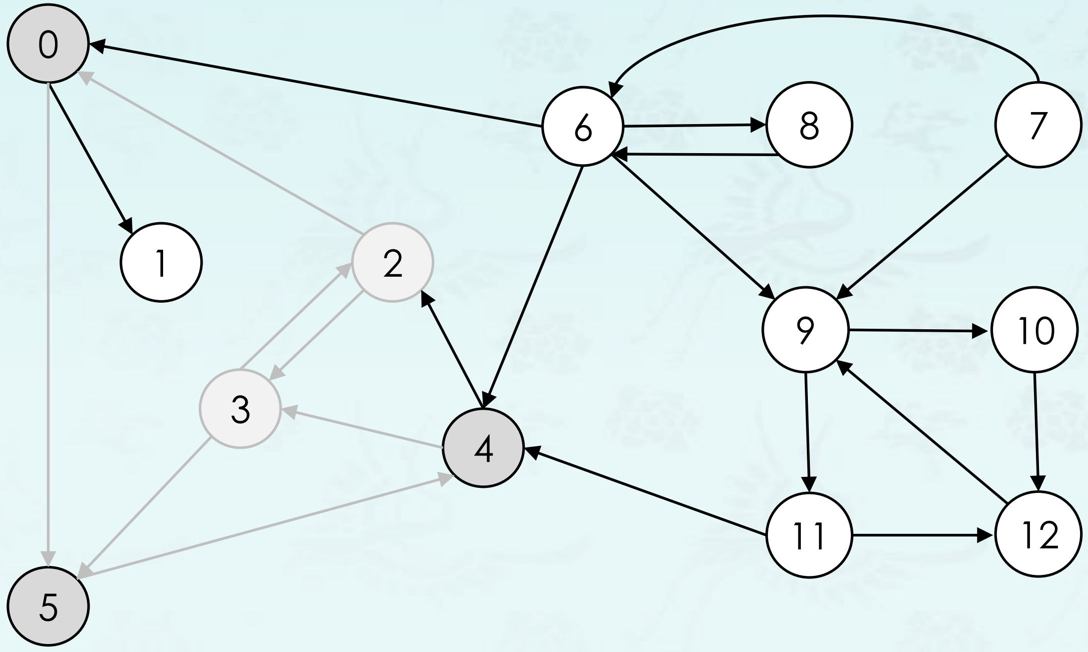


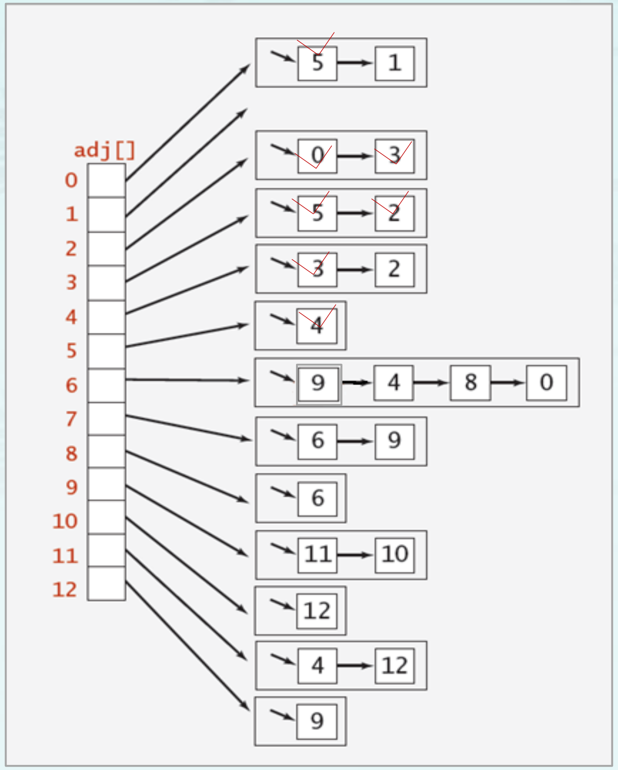
v	marked[]	parent[v]
0	T	–
1	F	–
2	T	3
3	T	4
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

visit 2: check 0 **and check 3**

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

Depth-first search in digraphs

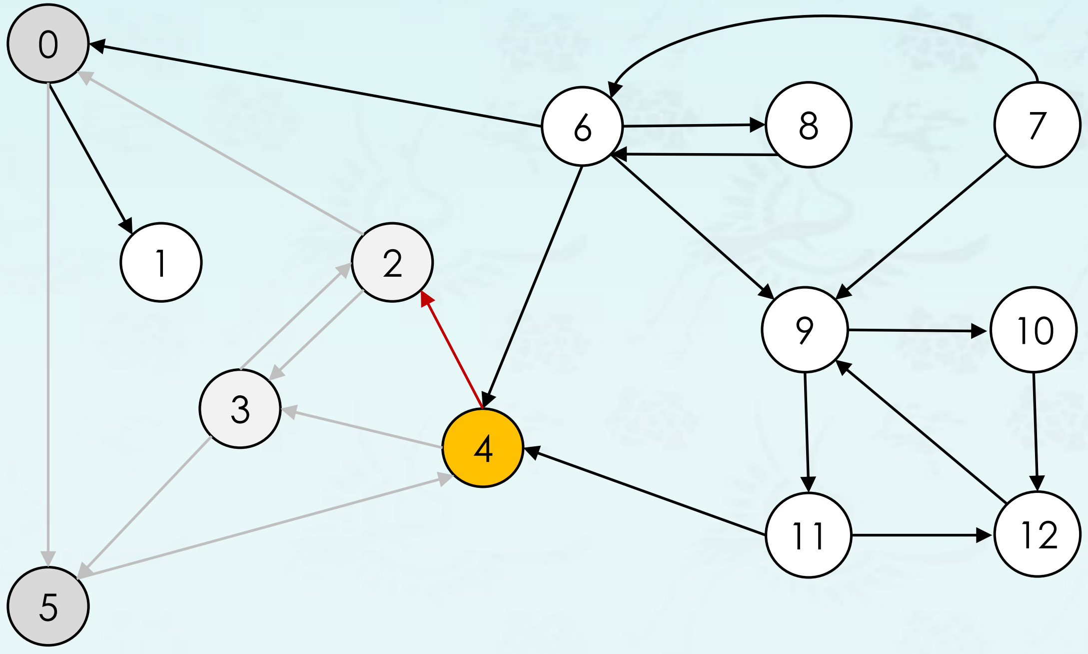


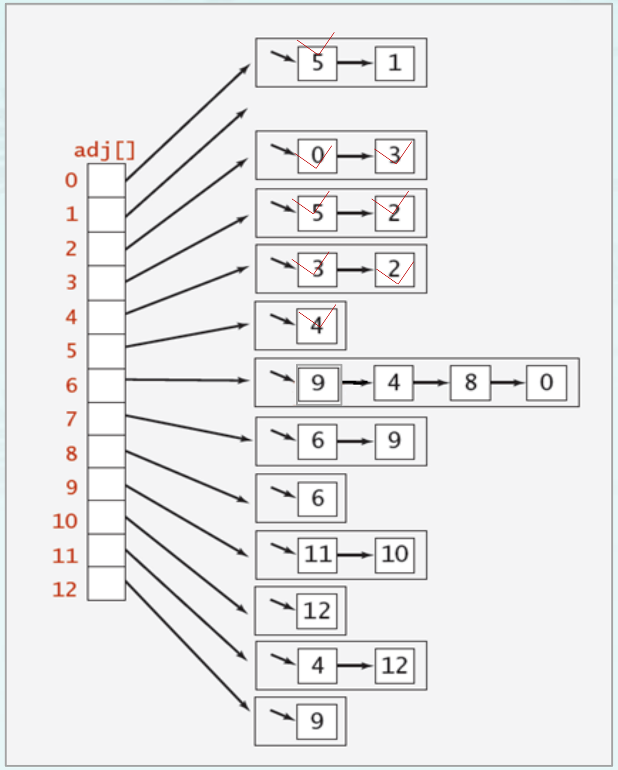


v	marked[]	parent[v]
0	T	–
1	F	–
2	T	3
3	T	4
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

done **3**

Depth-first search in digraphs

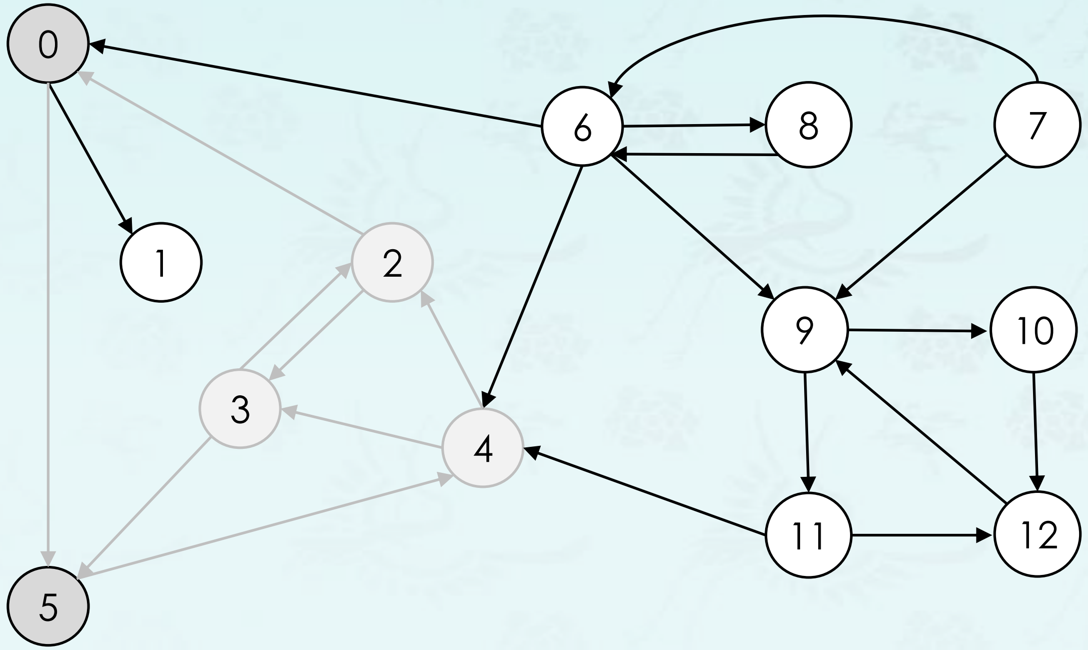


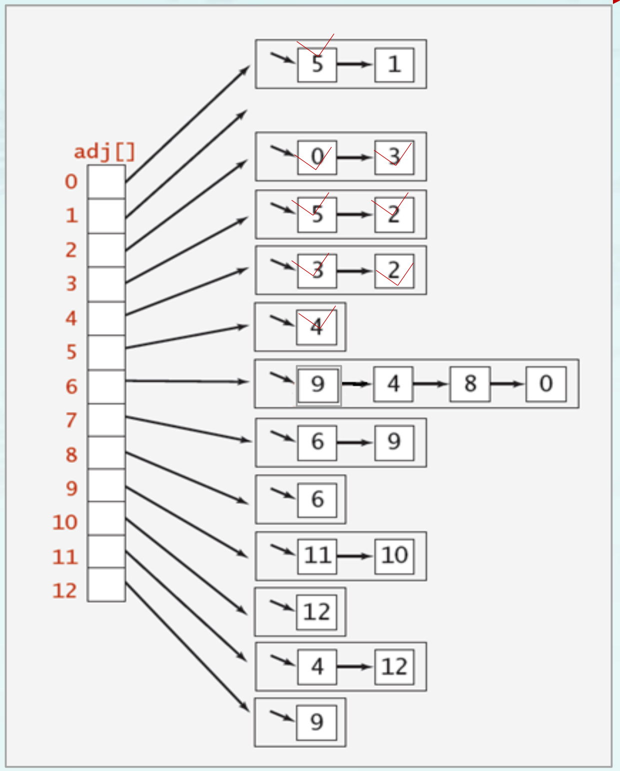


v	marked[]	parent[v]
0	T	–
1	F	–
2	T	3
3	T	4
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

visit 4: **check 3 and check 2**

Depth-first search in digraphs

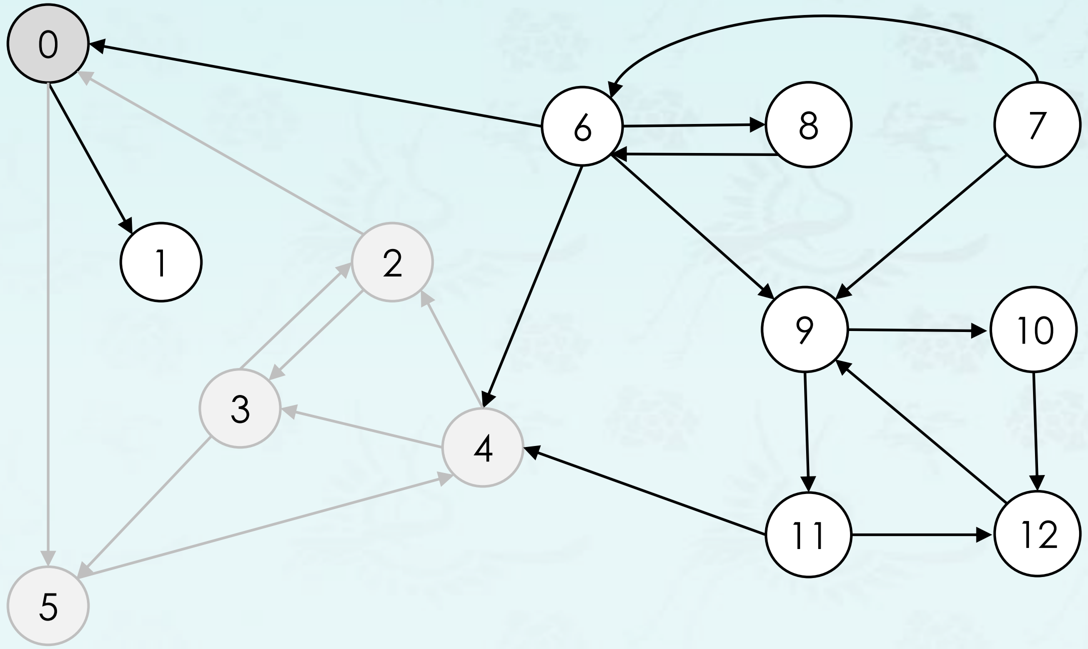


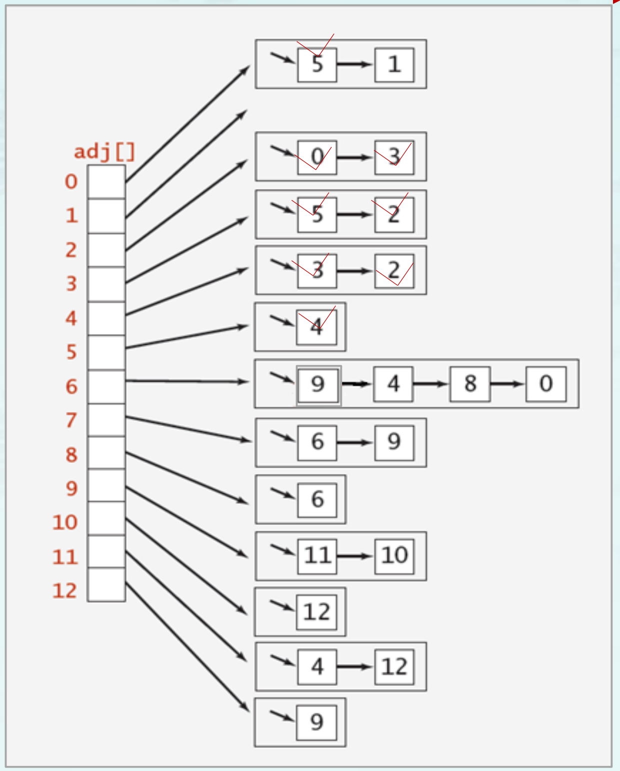


v	marked[]	parent[v]
0	T	–
1	F	–
2	T	3
3	T	4
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

Done 4

Depth-first search in digraphs

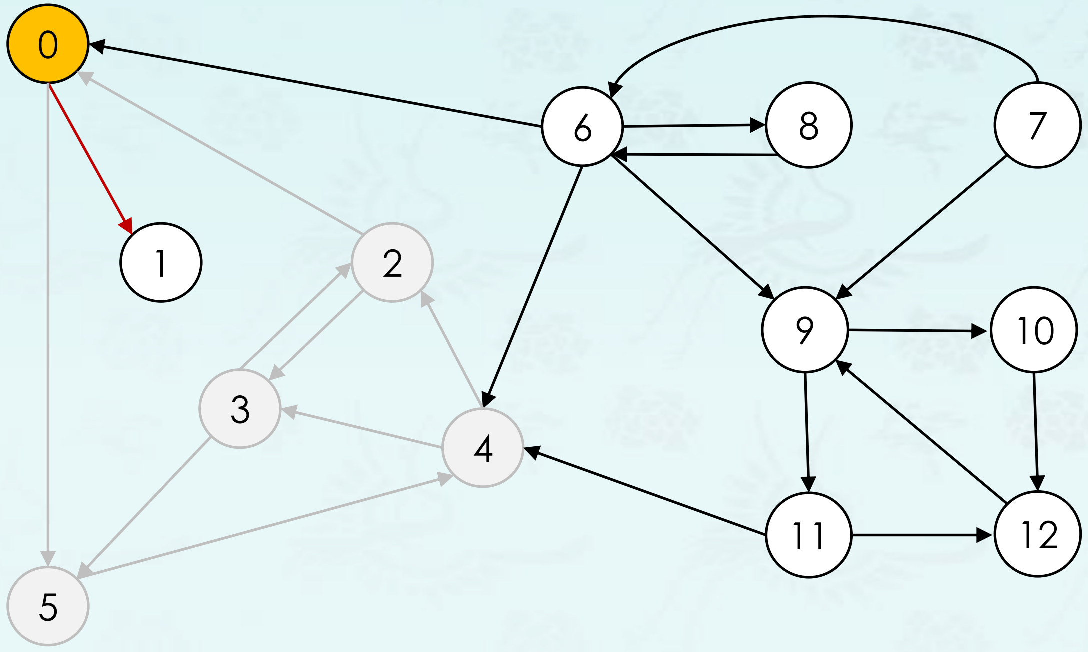


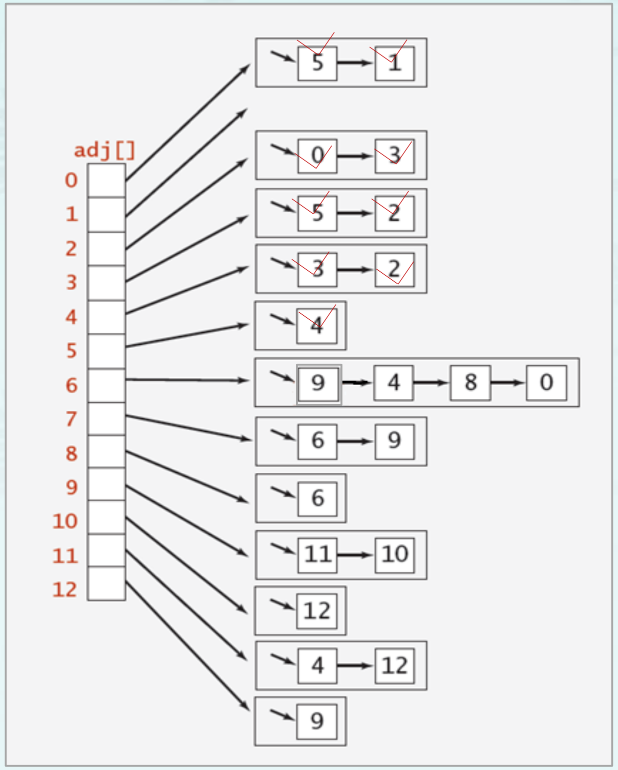


v	marked[]	parent[v]
0	T	–
1	F	–
2	T	3
3	T	4
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

Done 5

Depth-first search in digraphs

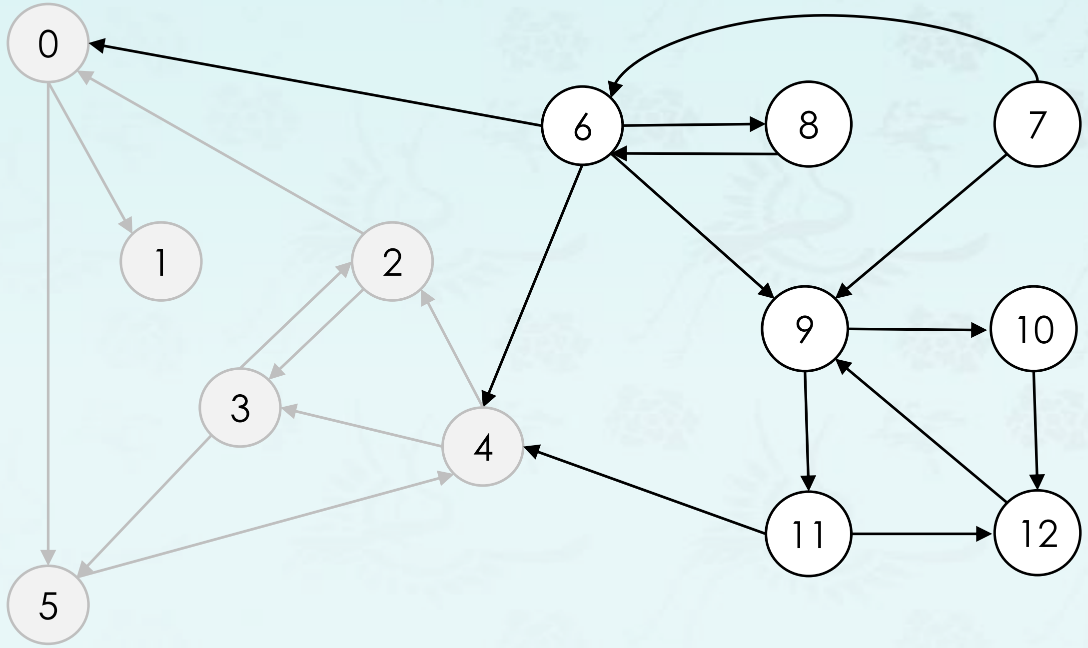


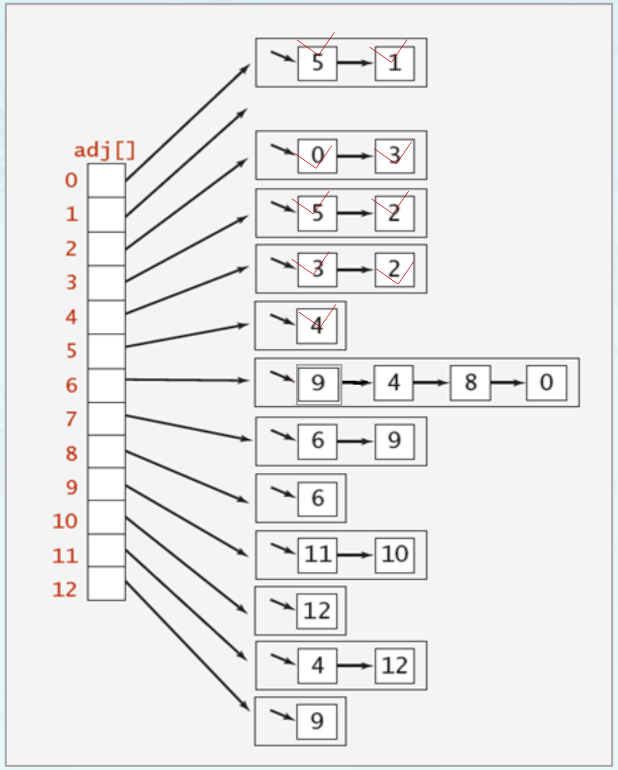


v	marked[]	parent[v]
0	T	–
1	T	0
2	T	3
3	T	4
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

visit 0: check 5 and **check 1**

Depth-first search in digraphs





v	marked[]	parent[v]
0	T	–
1	T	0
2	T	3
3	T	4
4	T	5
5	T	0
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

1 done

Depth-first search (in undirected graph) in Java

```
public class DepthFirstSearch {
  private boolean[] marked;

  public DepthFirstSearch(Graph G, int s)
  {
    marked = new Boolean[G.V()];
    dfs(G, s);
  }

  private void dfs(Graph G, int v)
  {
    marked[v] = true;
    for (int w : G.adj(v))
      if (!marked[w]) dfs(G, w);
  }

  public Boolean visited(int v)
  {  return marked[v];  }
}
```

true if path to s

constructor marks vertices connected to s

recursive DFS does the work

client can ask whether any vertex connected to s

Depth-first search (in directed graph) in Java

- Code for **directed** graphs identical to undirected one. [Substitute Digraph for Graph.]

```
public class DirectedDFS {
  private boolean[] marked;

  public DirectedDFS(Digraph G, int s)
  {
    marked = new Boolean[G.V()];
    dfs(G, s);
  }

  private void dfs(DiGraph G, int v)
  {
    marked[v] = true;
    for (int w : G.adj(v))
      if (!marked[w]) dfs(G, w);
  }

  public Boolean visited(int v)
  {  return marked[v];   }
}
```

true if path to s

constructor marks vertices connected to s

recursive DFS does the work

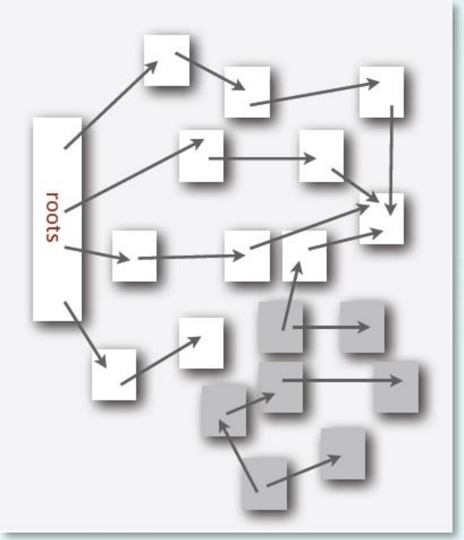
client can ask whether any vertex is **reachable from s**

Reachability application: mark-sweep garbage collector

Every data structure (in java) is a digraph.

- Vertex = object.
- Edge = reference.

- **Roots :** Objects known to be directly accessible by program (e.g., stack).
- **Reachable objects :** Objects indirectly accessible by program (starting at a root and following a chain of pointers).

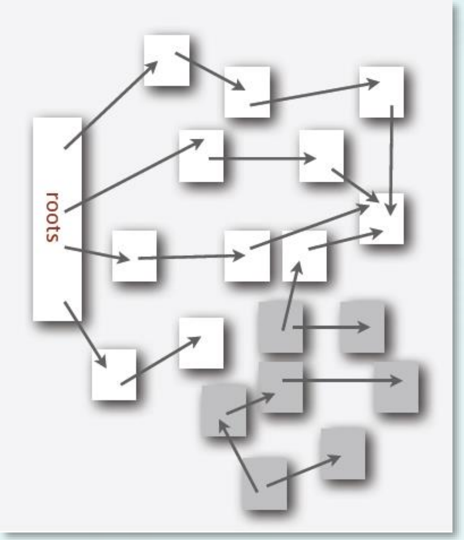


Reachability application: mark-sweep garbage collector

- **Mark-sweep algorithm (McCathy, 1960)**

1. Mark data objects in a program that cannot be accessed in the future.
2. Sweep: if object is unmarked, it is garbage (so add to free list).

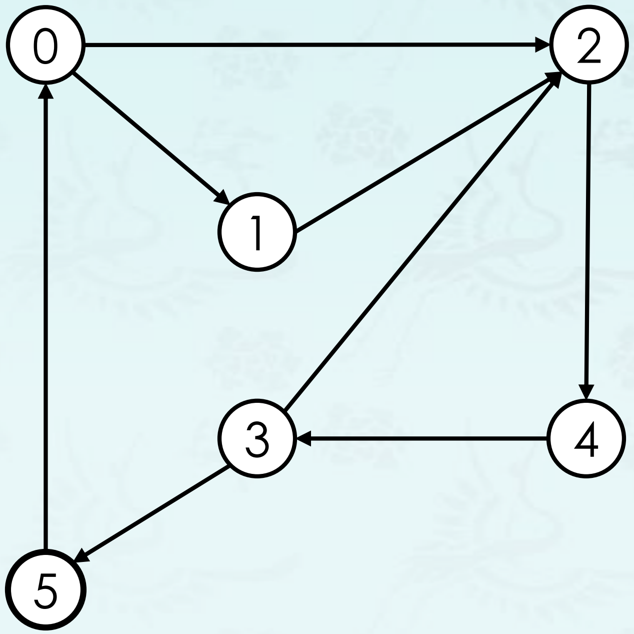
- **Memory cost :** Uses 1 extra mark bit per object (plus DFS stack).



Directed breadth-first search demo

Repeat until queue is empty.

- Remove vertex v from queue.
- Add to queue all unmarked vertices pointing from v and mark them.



Graph g:

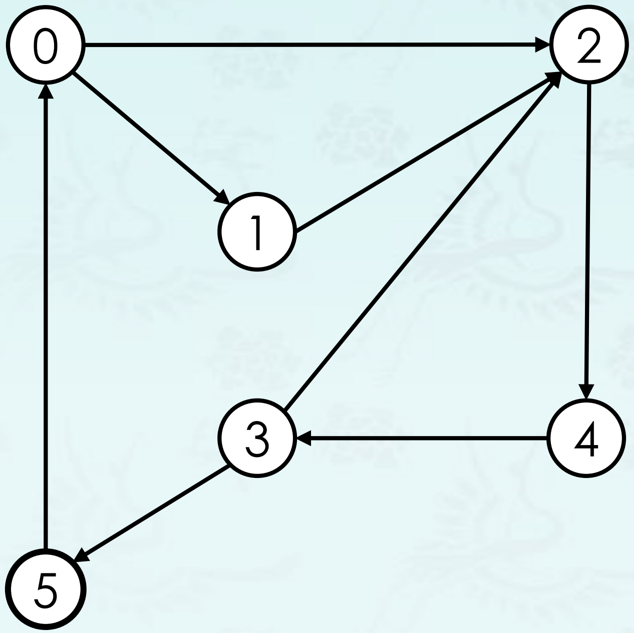
```
myDG.txt
6      V
8      E
5 0
2 4
3 2
1 2
0 1
4 3
3 5
0 2
```

Challenge: build adjacency lists – Job done

Directed breadth-first search demo

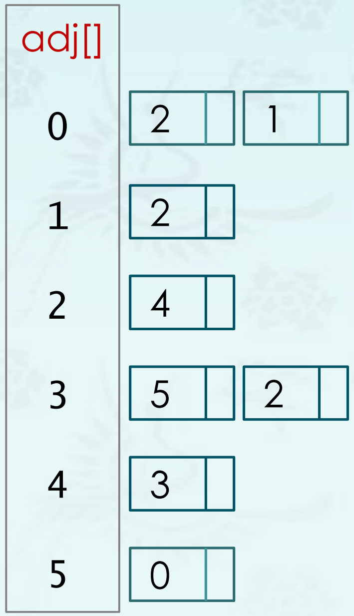
Repeat until queue is empty.

- Remove vertex v from queue.
- Add to queue all unmarked vertices pointing from v and mark them.



Graph g:

Adjacency lists

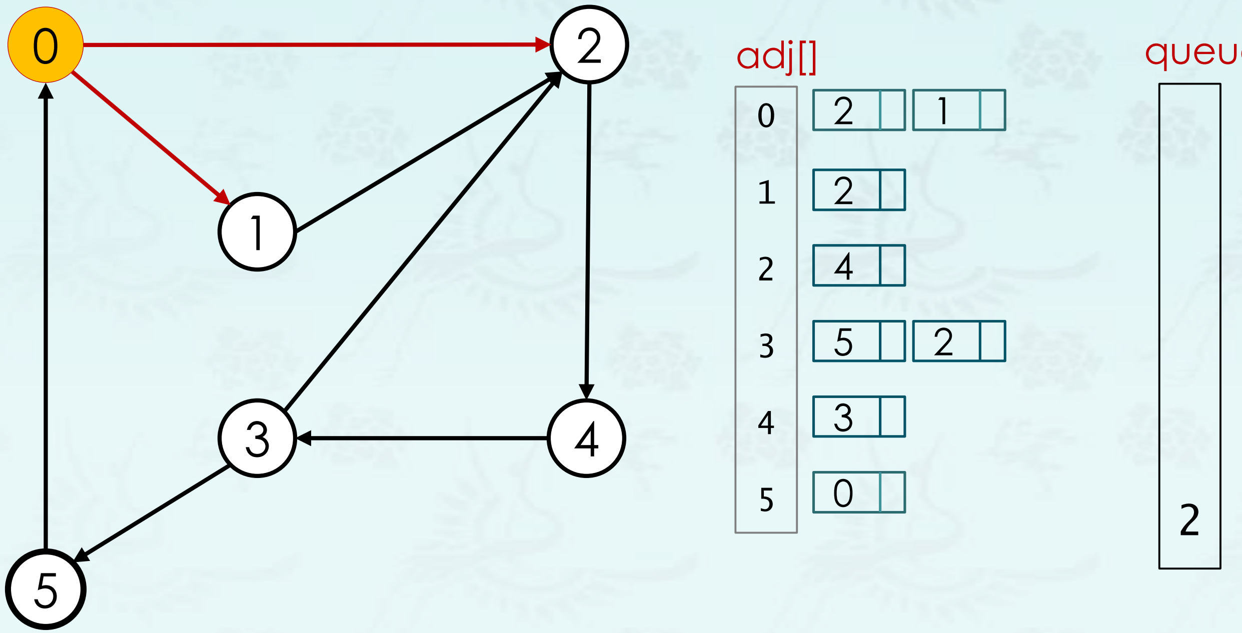



```
myDG.txt
6
8
5 0
2 4
3 2
1 2
0 1
4 3
3 5
0 2
```

V
E

Challenge: build adjacency lists – Job done

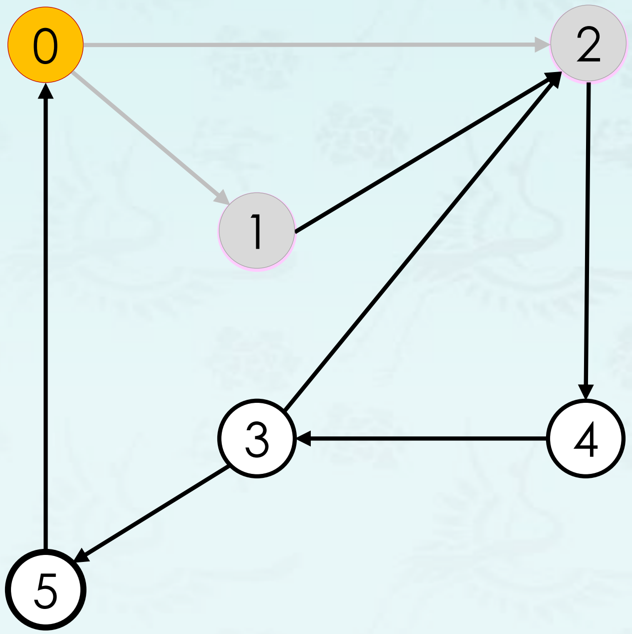
Directed breadth-first search demo

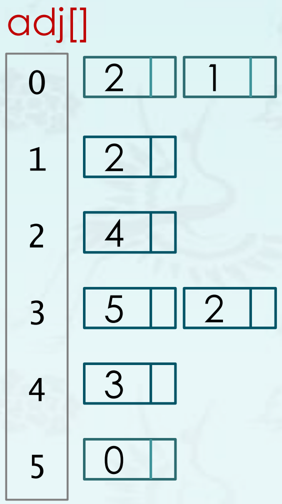


v	parent[]	distTo[]
0	–	0
1	–	–
2	0	1
3	–	–
4	–	–
5	–	–

dequeue 0: check 2 and check 1

Directed breadth-first search demo



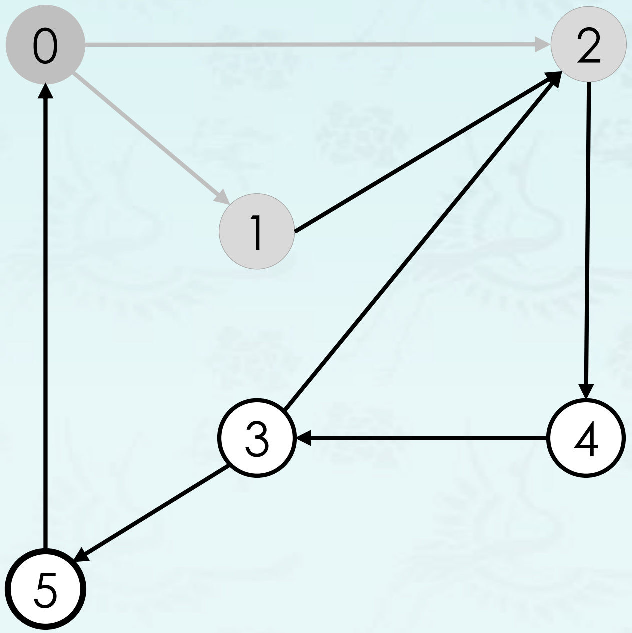


queue: 1, 2

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	-	-
4	-	-
5	-	-

dequeue 0: check 2 and check 1

Directed breadth-first search demo



adj[]

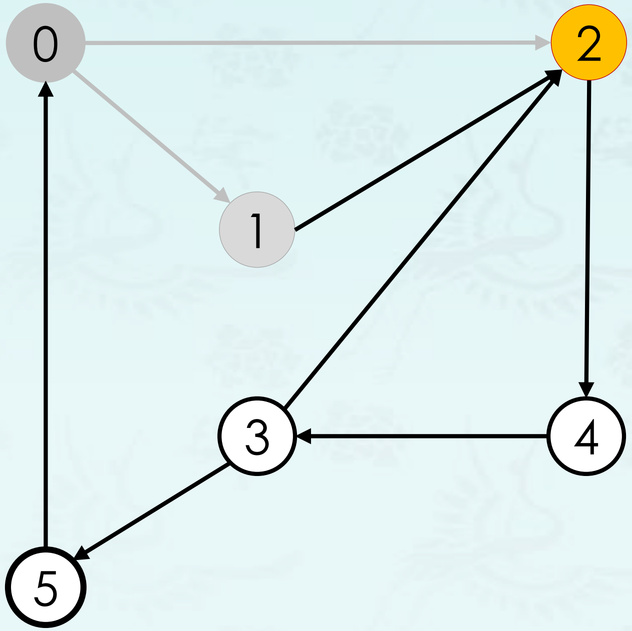
0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

1
2

0	–	0
1	0	1
2	0	1
3	–	–
4	–	–
5	–	–

0 done

Directed breadth-first search demo



adj[]

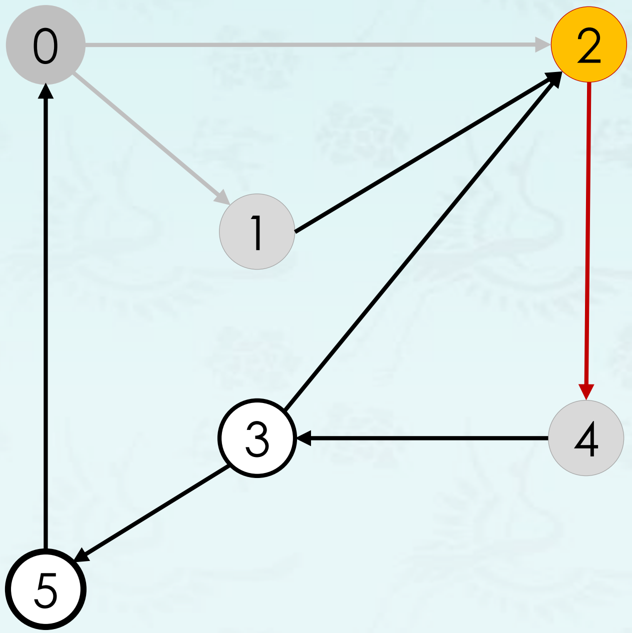
0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

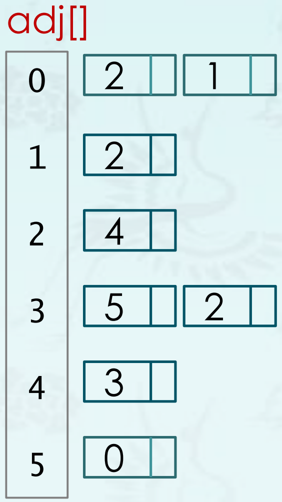
1

0	–	0
1	0	1
2	0	1
3	–	–
4	–	–
5	–	–

dequeue 2

Directed breadth-first search demo





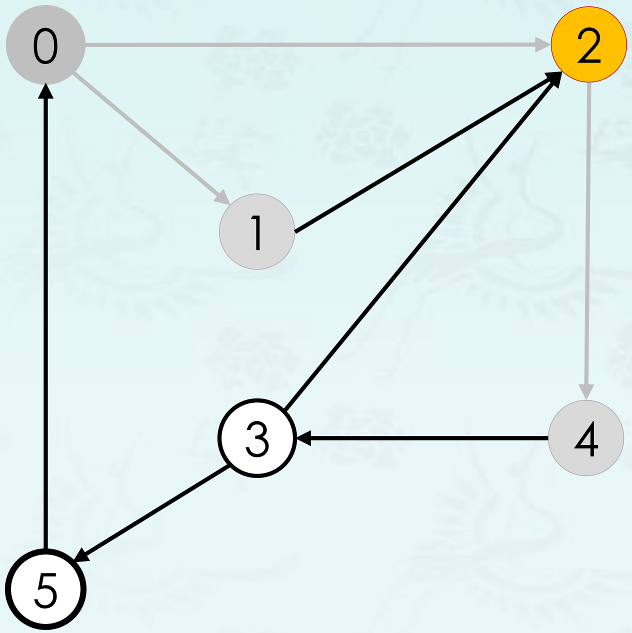
queue v parent[] distTo[]

queue
4
1

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	-	-
4	2	2
5	-	-

dequeue 2 : check 4

Directed breadth-first search demo



adj[]

0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

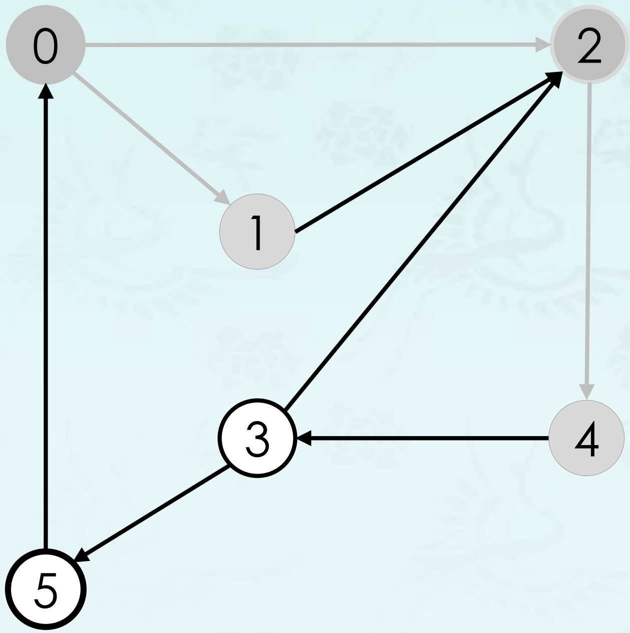
queue

4
1

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	-	-
4	2	2
5	-	-

dequeue 2 : check 4

Directed breadth-first search demo



adj[]

0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

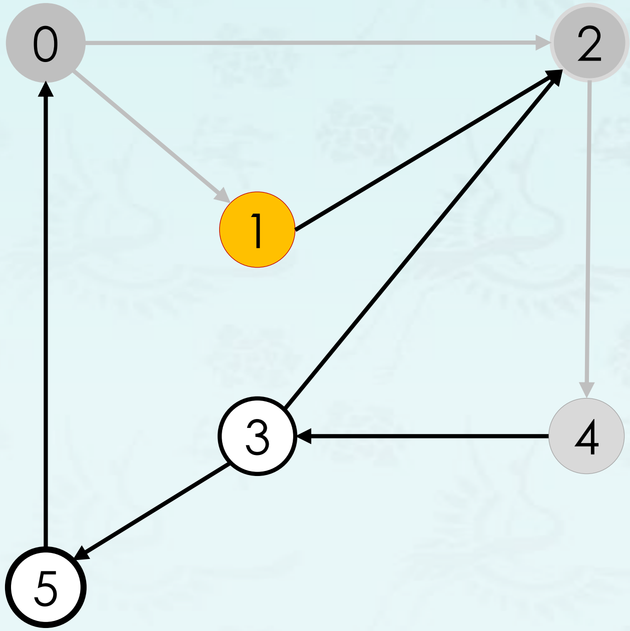
queue

4
1

v	parent[]	distTo[]
0	–	0
1	0	1
2	0	1
3	–	–
4	2	2
5	–	–

2 done

Directed breadth-first search demo



adj[]

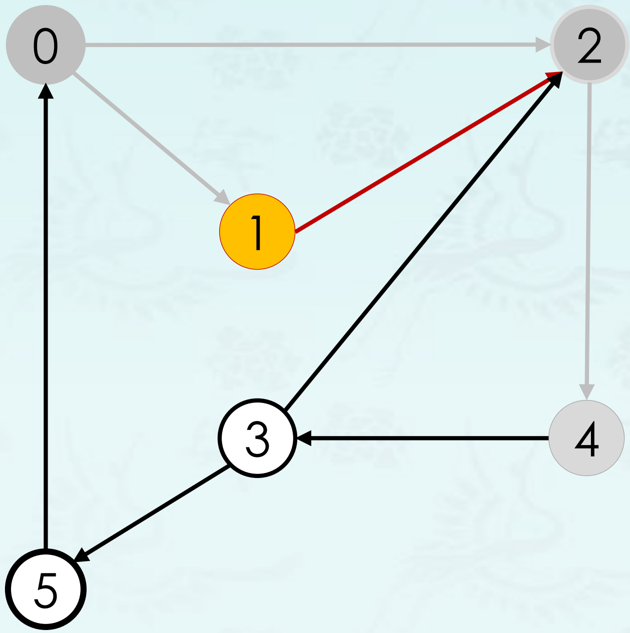
0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

4

0	–	0
1	0	1
2	0	1
3	–	–
4	2	2
5	–	–

dequeue 1

Directed breadth-first search demo



adj[]

0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

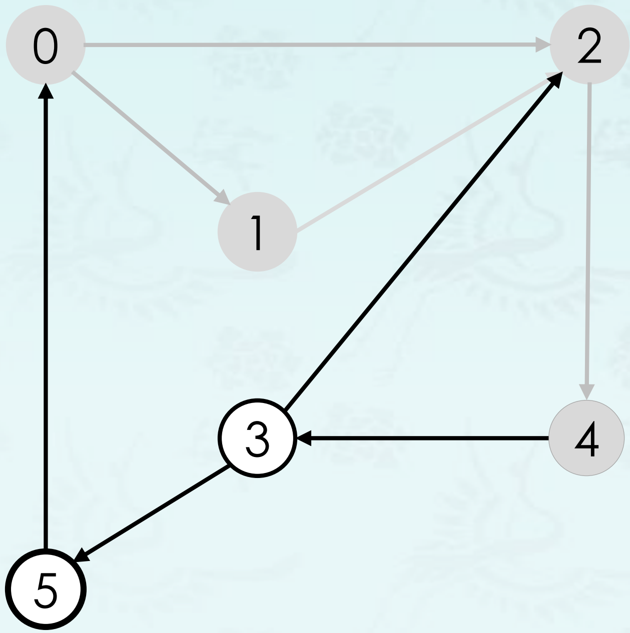
queue

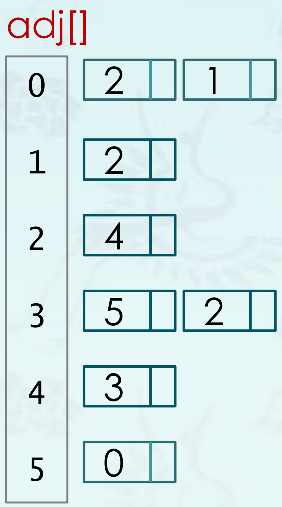
4

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	-	-
4	2	2
5	-	-

dequeue 1 : check 2

Directed breadth-first search demo



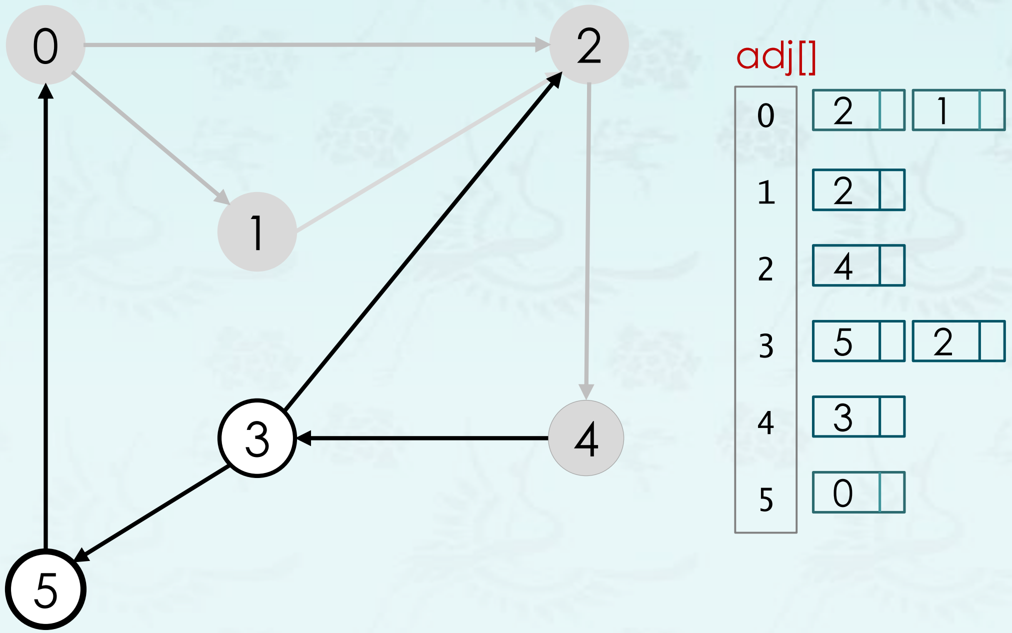


queue: 4

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	-	-
4	2	2
5	-	-

1 done

Directed breadth-first search demo

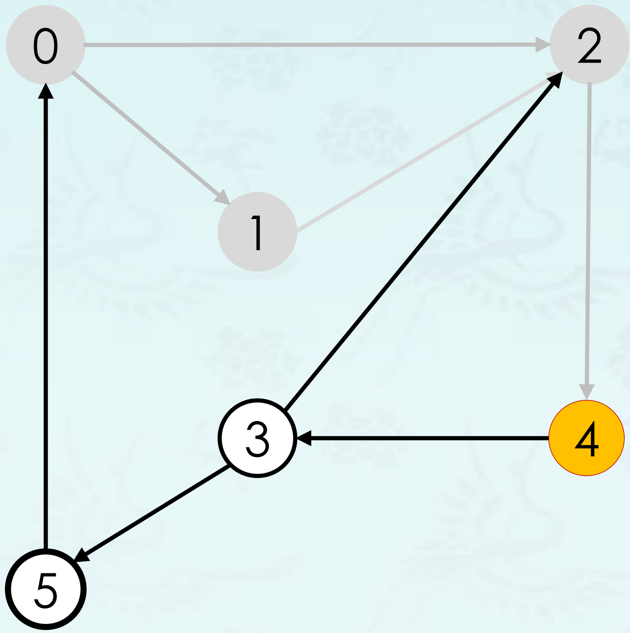


queue: 4

v	parent[]	distTo[]
0	–	0
1	0	1
2	0	1
3	–	–
4	2	2
5	–	–

dequeue 4

Directed breadth-first search demo



adj[]

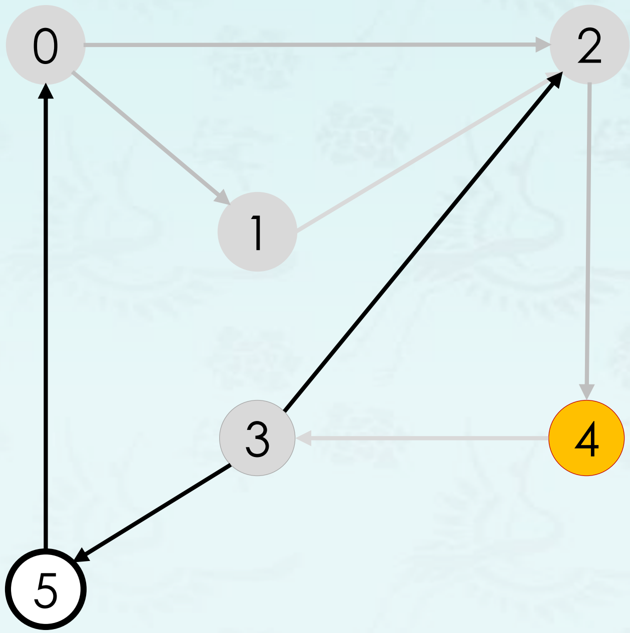
0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

queue

v	parent[]	distTo[]
0	–	0
1	0	1
2	0	1
3	–	–
4	2	2
5	–	–

dequeue 4

Directed breadth-first search demo



adj[]

0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

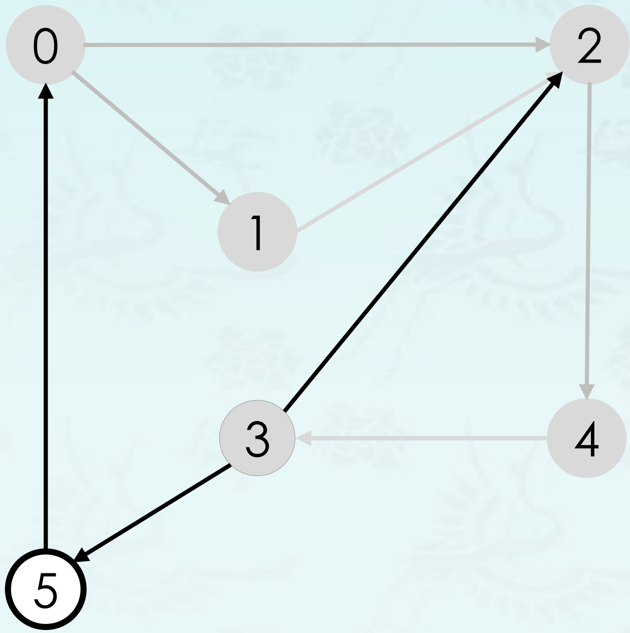
queue

3

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	4	3
4	2	2
5	-	-

dequeue 4 : check 3

Directed breadth-first search demo



adj[]

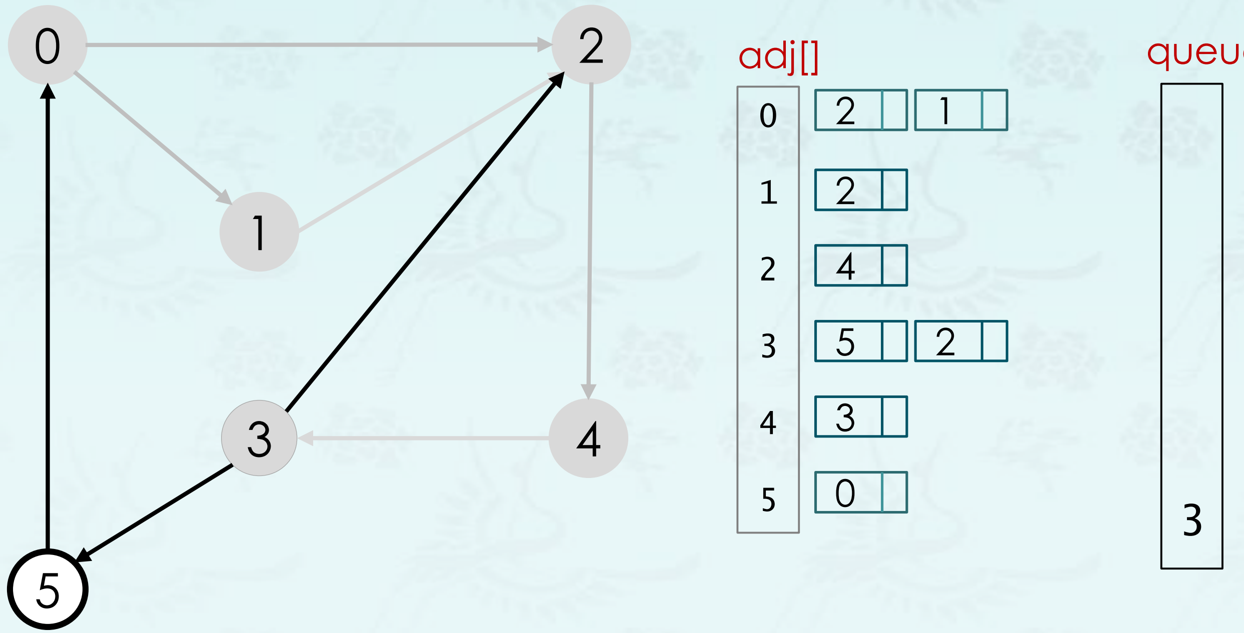
0	2, 1
1	2
2	4
3	5, 2
4	3
5	0

queue: 3

v	parent[]	distTo[]
0	–	0
1	0	1
2	0	1
3	4	3
4	2	2
5	–	–

4 done

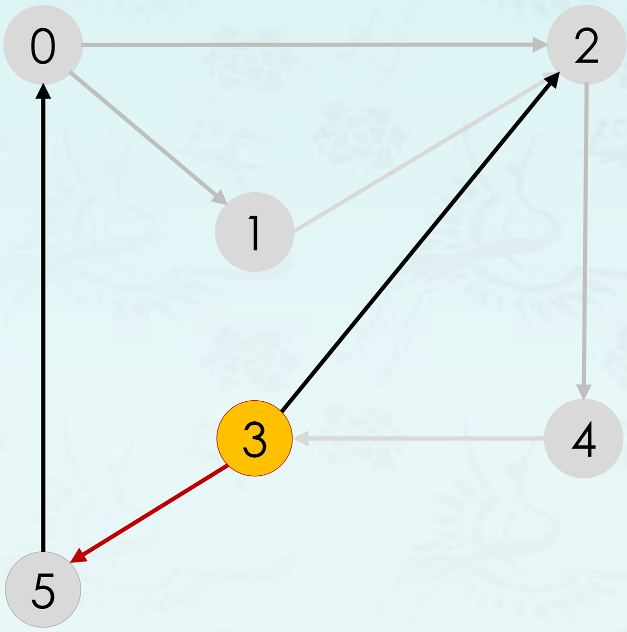
Directed breadth-first search demo



v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	4	3
4	2	2
5	-	-

dequeue 3

Directed breadth-first search demo



adj[]

0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

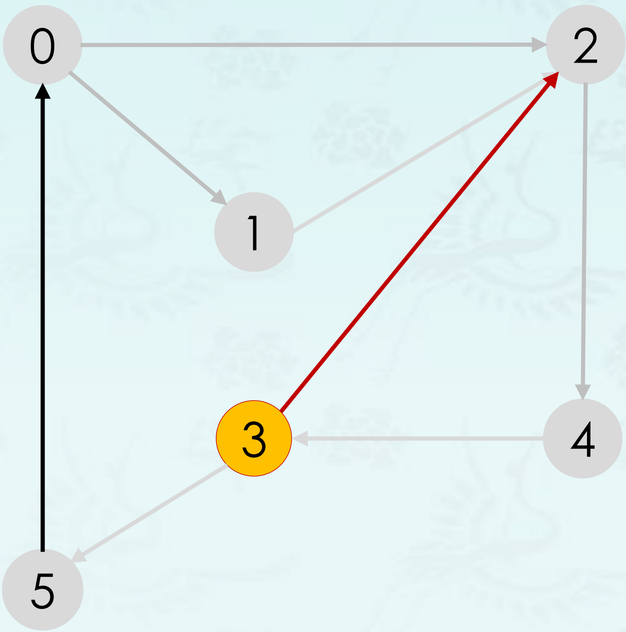
queue

5

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	4	3
4	2	2
5	3	4

dequeue 3 : check 5 and check 2

Directed breadth-first search demo



adj[]

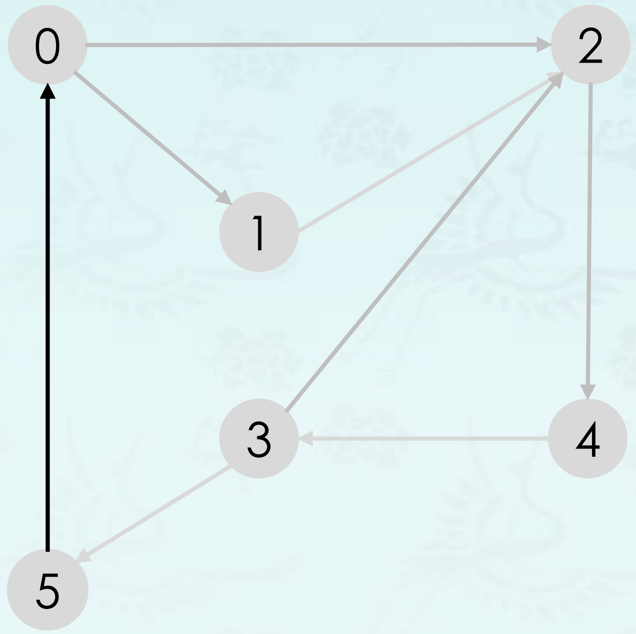
0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

queue: 5

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	4	3
4	2	2
5	3	4

dequeue 3 : check 5 and check 2

Directed breadth-first search demo



adj[]

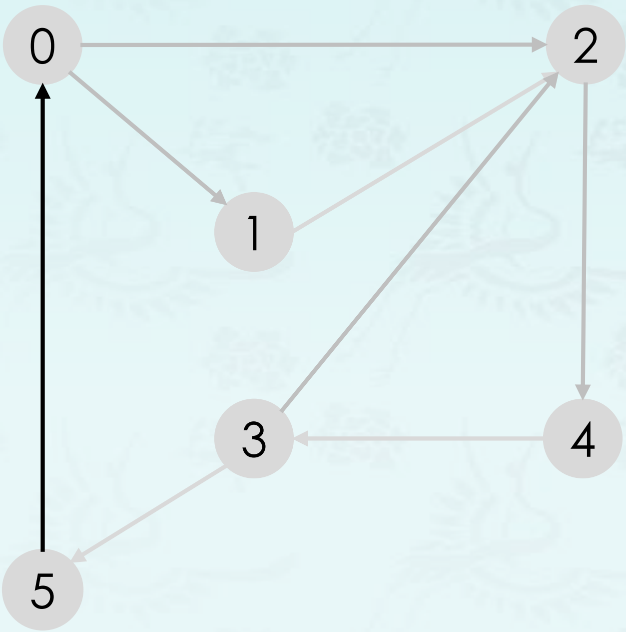
0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

queue: 5

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	4	3
4	2	2
5	3	4

dequeue 3 : check 5 and check 2

Directed breadth-first search demo



adj[]

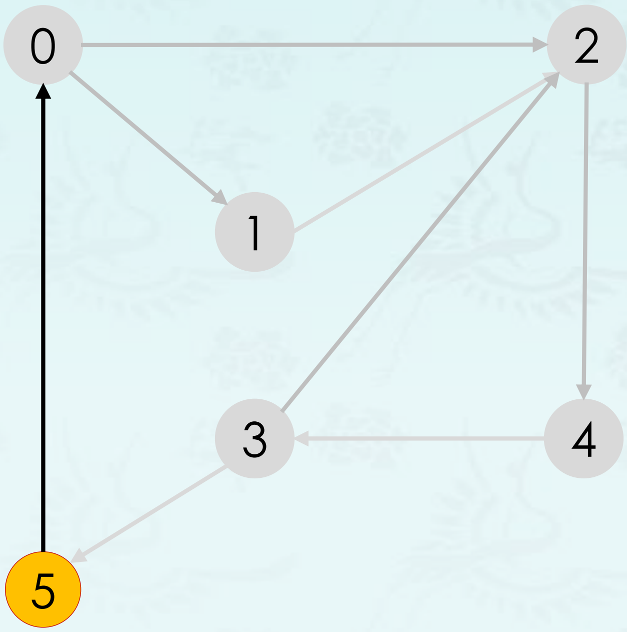
0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

queue: 5

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	4	3
4	2	2
5	3	4

dequeue 5 :

Directed breadth-first search demo



adj[]

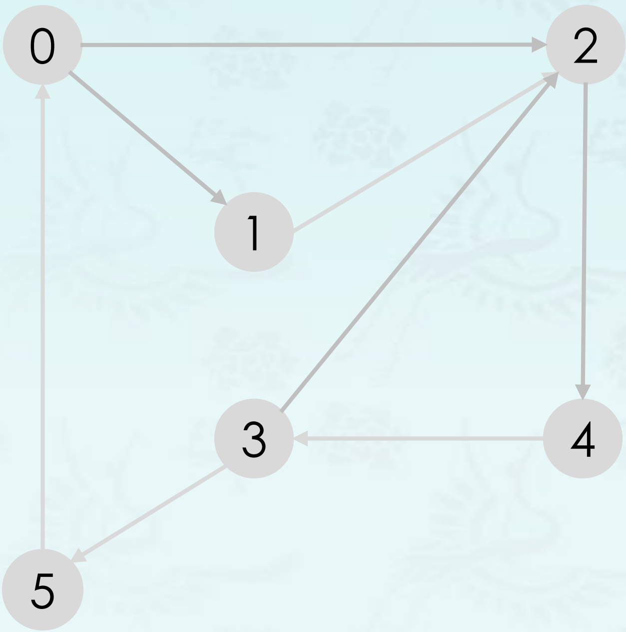
0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

queue

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	4	3
4	2	2
5	3	4

dequeue 5 : check 0

Directed breadth-first search demo



adj[]

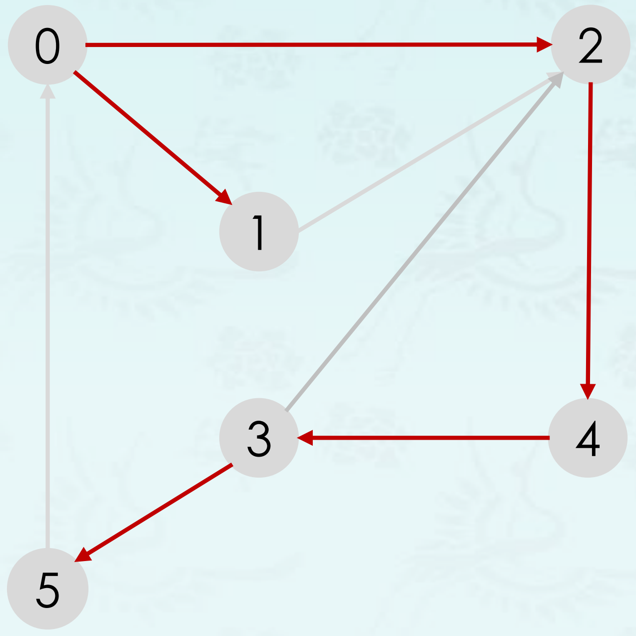
0	2	1
1	2	
2	4	
3	5	2
4	3	
5	0	

queue

v	parent[]	distTo[]
0	-	0
1	0	1
2	0	1
3	4	3
4	2	2
5	3	4

5 done

Directed breadth-first search demo



v	parent[]	distTo[]
0	–	0
1	0	1
2	0	1
3	4	3
4	2	2
5	3	4

queue

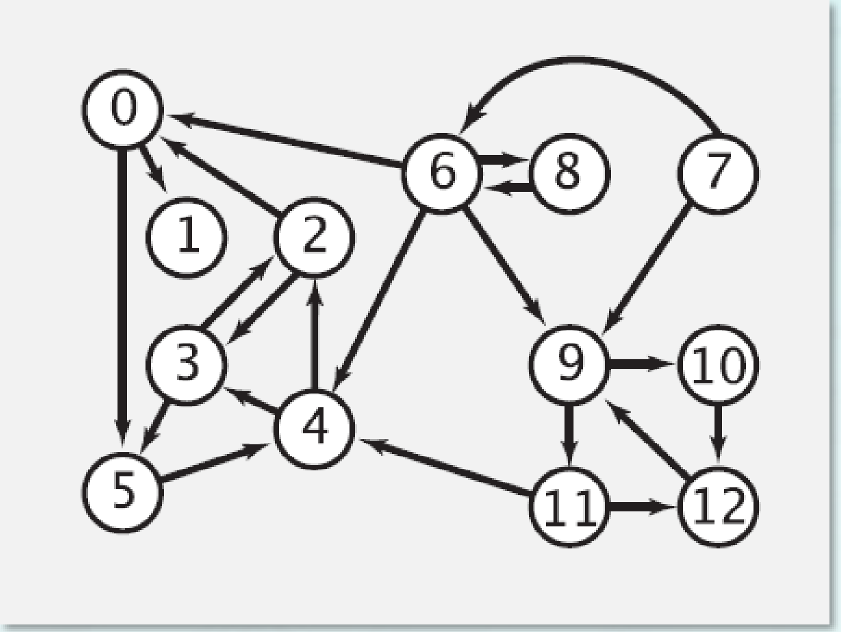
done

Multiple-source shortest paths

- **Multiple-source shortest paths:** Given a digraph and a set of source vertices, find shortest path from any vertex in the set to each other vertex.

Ex: S = { 7, 10 }, D = { 4, 5, 12 }

- Shortest path to 4 is 7 → 6 → 4.
- Shortest path to 5 is 7 → 6 → 0 → 5.
- Shortest path to 12 is 10 → 12.



- **Q:** How to implement multi-source shortest paths algorithm?
- **A:** Use BFS, but initialize by enqueuing all source vertices.

Data Structures
Chapter 7: Graph

1. Introduction
   - Terminology, Representation, ADT
2. Basic Operations
   - DFS, CC, BFS, Processing
3. **Digraph and** Applications
4. Minimum Spanning Tree(MST)

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University