DIRECTED GRAPHS

- *introduction*
- *digraph API*
- *depth-first search*
- *breadth-first search*
- *topological sort*
- *strong components* ← see videos



http://ds.cs.Rutgers.edu

copyrighted content – Do not share

Some slides Adopted and modified from Sedgewick and Wayne

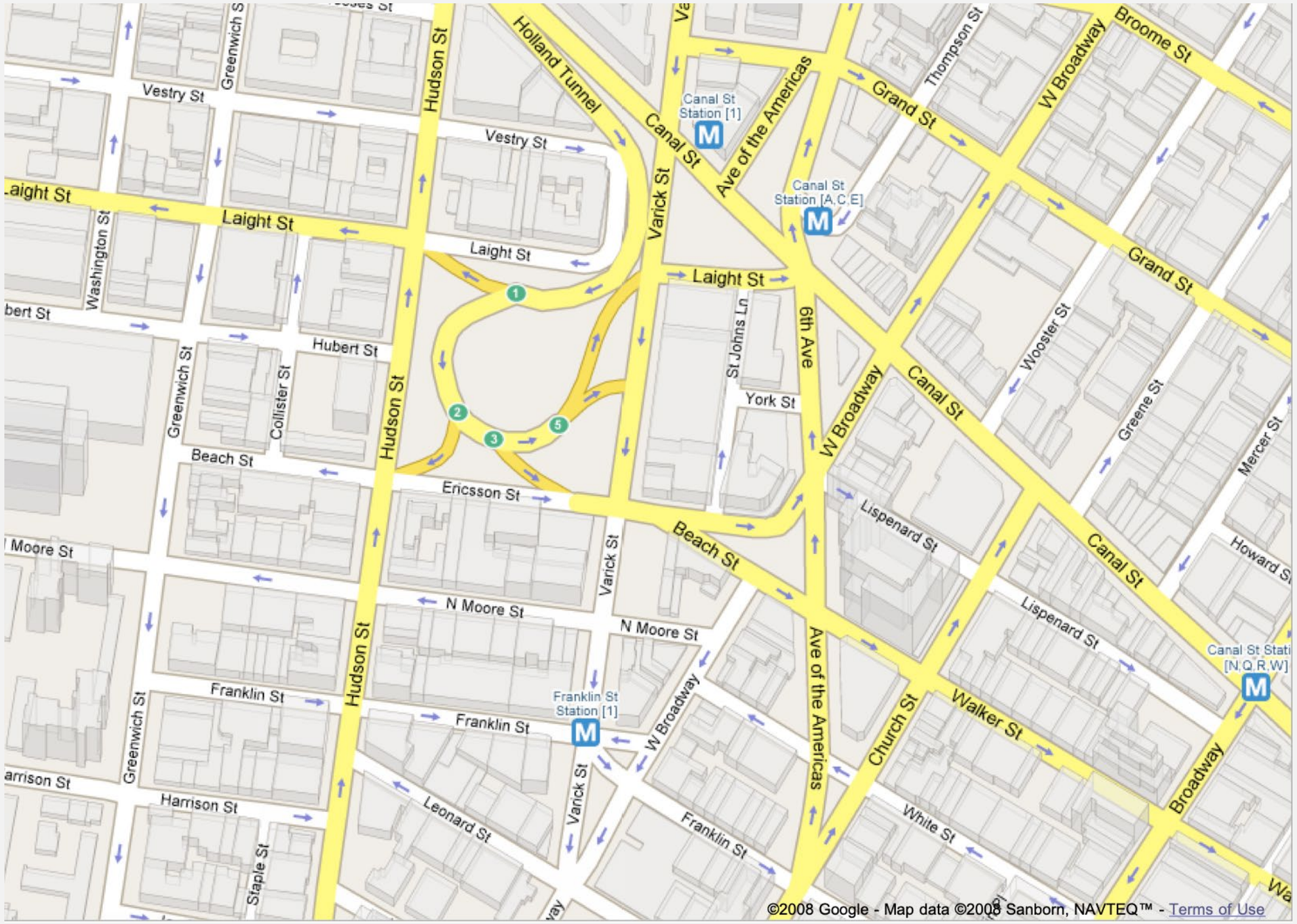
DIRECTED GRAPHS

- *introduction*
- *digraph API*
- *depth-first search*
- *breadth-first search*
- *topological sort*

copyrighted content – Do not share

Road networks

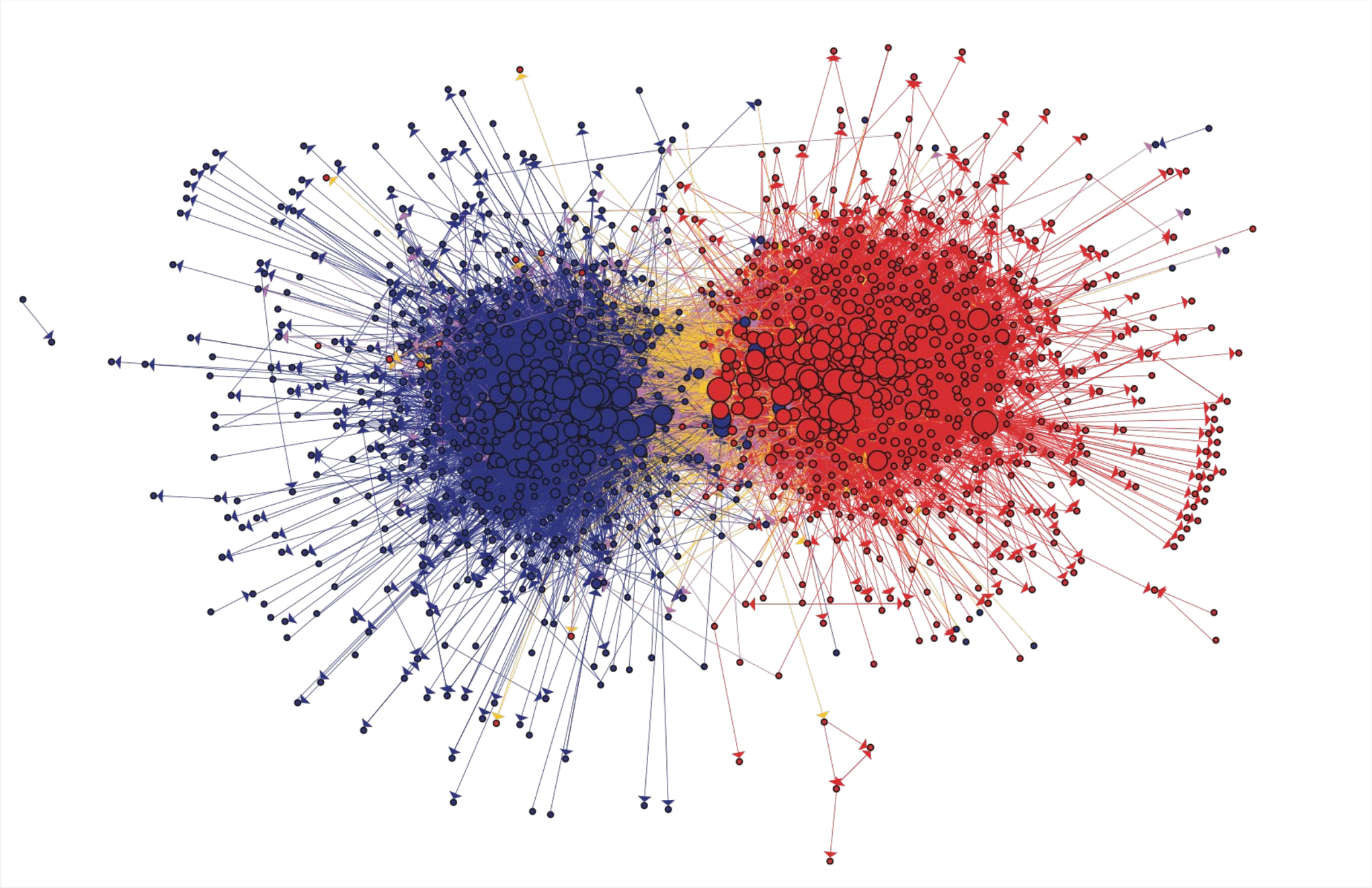
Vertex = intersection; edge = one-way street.



copyrighted content – Do not share

Political blogosphere links

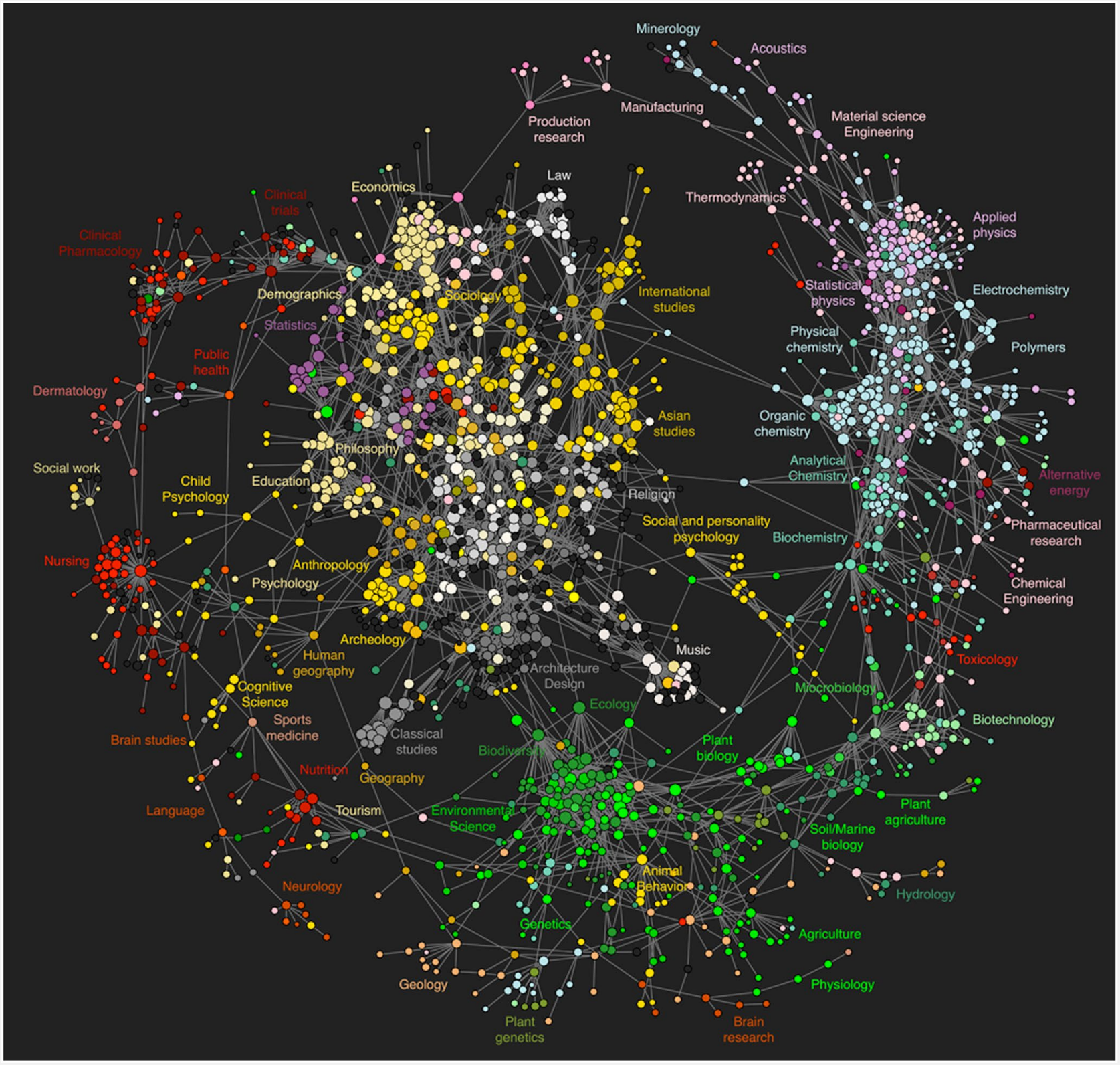
Vertex = political blog; edge = link.



The Political Blogosphere and the 2004 U.S. Election: Divided They Blog, Adamic and Glance, 2005

copyrighted content – Do not share

Science clickstreams

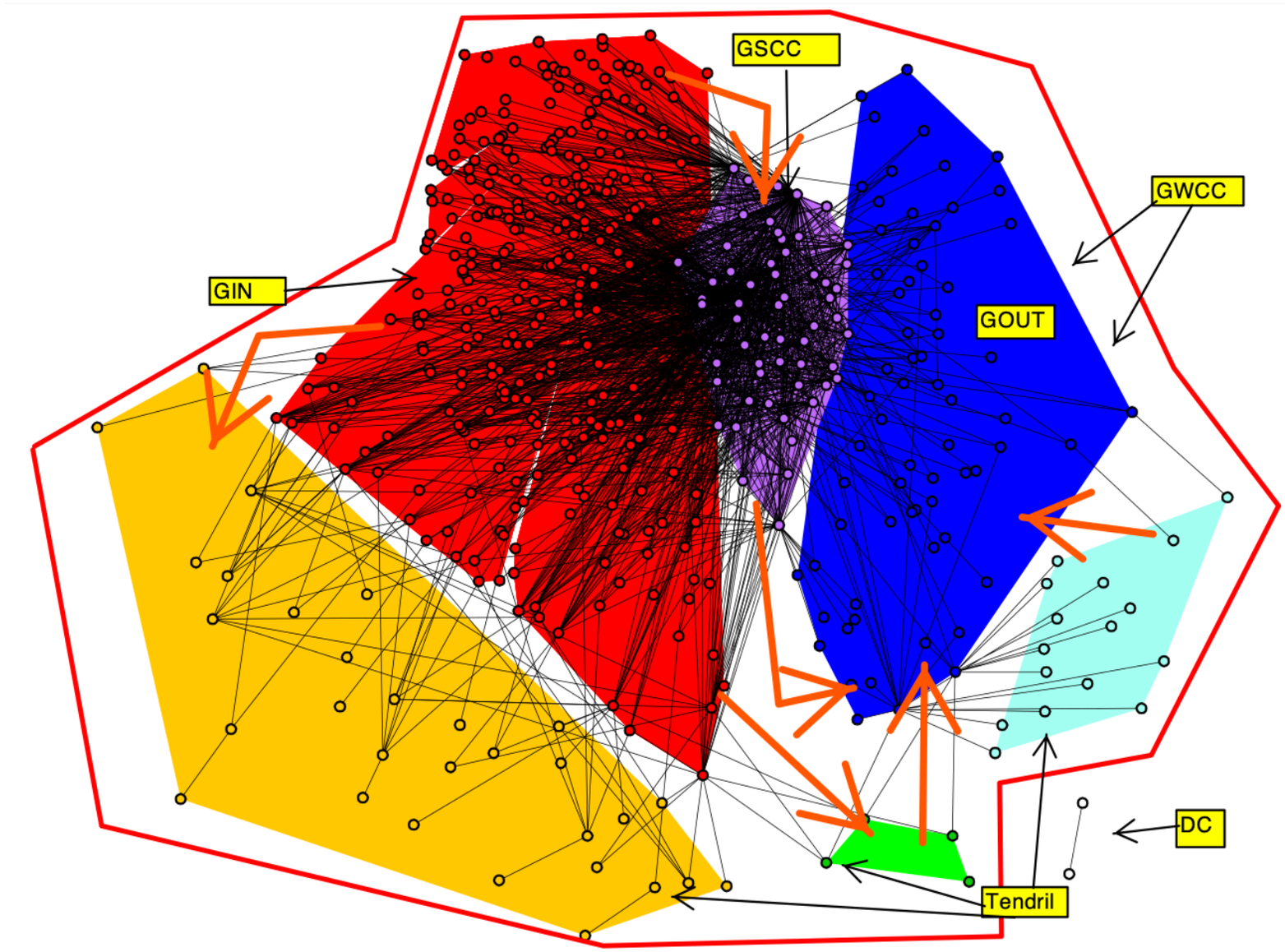


copyrighted content – Do not share

http://www.plosone.org/article/info:doi/10.1371/journal.pone.0004803

Overnight interbank loans

Vertex = bank; edge = overnight loan.

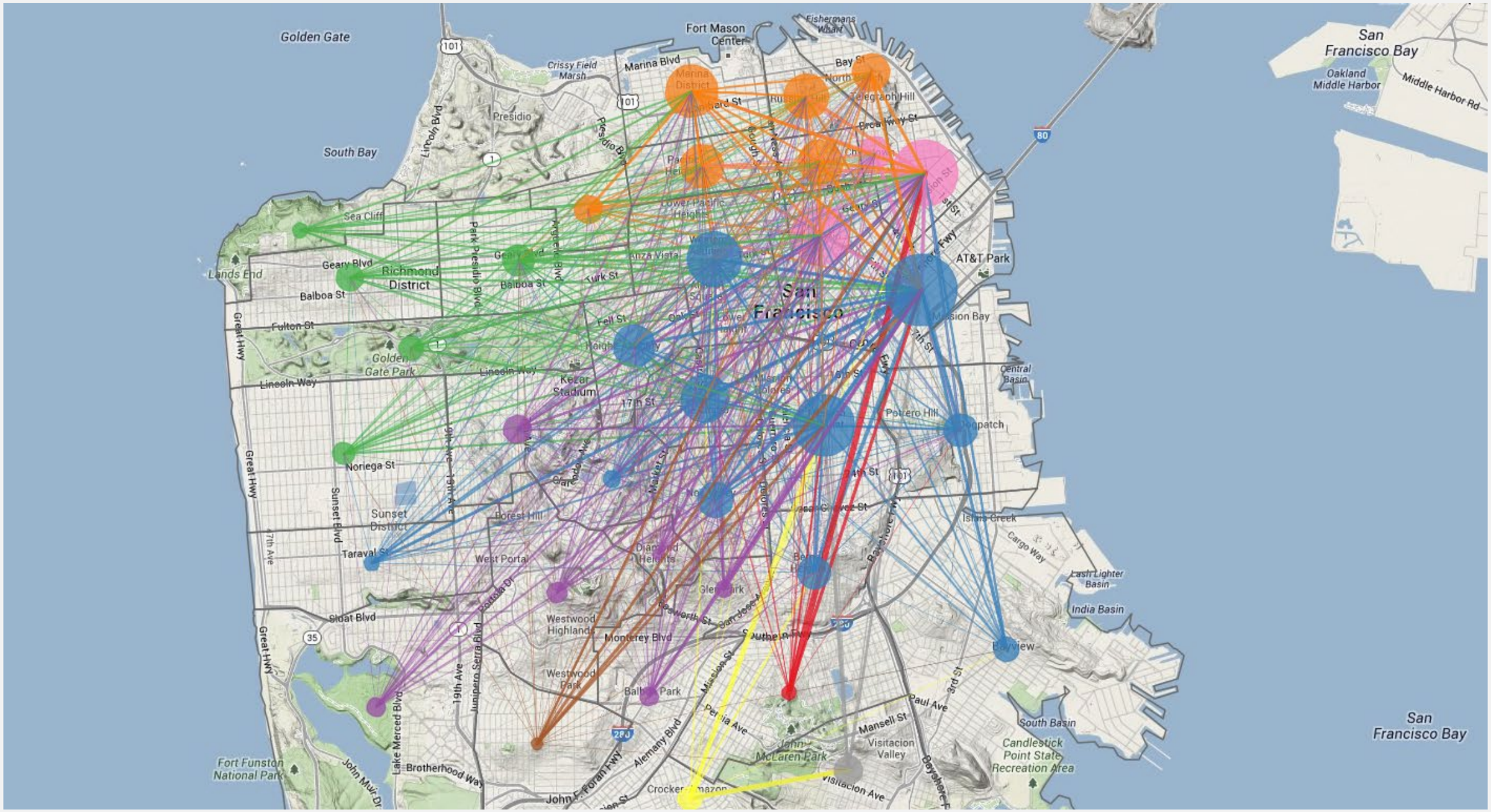


The Topology of the Federal Funds Market, Bech and Atalay, 2008

copyrighted content – Do not share

Uber rides

Vertex = taxi pickup; edge = taxi ride.


Golden Gate
Fort Mason Center
Fisherman's Wharf
San Francisco Bay
Oakland Middle Harbor
Middle Harbor Rd
Crissy Field Marsh
Marina Blvd
Marina District
Bay St
Presidio
South Bay
Lincoln Blvd
Presidio Blvd
Telegraph Hill
Broadway St
Sea Cliff
Lower Pacific Heights
Lands End
Geary Blvd
Richmond District
Balboa St
Park Presidio Blvd
Arguello Blvd
Divisadero St
Fillmore St
Van Ness Ave
Hayes St
Gough St
Franklin St
Octavia Blvd
Laguna St
Buchanan St
Webster St
Fillmore St
Steiner St
Pierce St
Scott St
Baker St
Lyon St
Presidio Ave
Cherry St
Maple St
Locust St
Walnut St
Spruce St
Mayfair Dr
Presidio Terrace
Anza Vista
Turk St
Balboa St
San Francisco
Fell St
Oak St
Haight Ashbury
Lower Haight
Hayes Valley
Civic Center
Mission Dolores
Duboce Ave
Market St
Octavia Blvd
Gough St
Franklin St
Van Ness Ave
Polk St
Larkin St
Hyde St
Leavenworth St
Jones St
Taylor St
Mason St
Powell St
Stockton St
Grant Ave
Kearny St
Montgomery St
Sansome St
Battery St
Front St
Drumm St
Embarcadero
Market St
Bush St
Pine St
California St
Sacramento St
Clay St
Washington St
Jackson St
Broadway
Pacific Ave
Columbus Ave
Green St
Vallejo St
Broadway
Union St
Filbert St
Greenwich St
Lombard St
Chestnut St
Francisco St
Bay St
Jefferson St
Fisherman's Wharf
Pier 39
Embarcadero
Battery St
Front St
Drumm St
Sansome St
Montgomery St
Kearny St
Grant Ave
Stockton St
Powell St
Mason St
Taylor St
Jones St
Leavenworth St
Hyde St
Larkin St
Polk St
Van Ness Ave
Franklin St
Gough St
Octavia Blvd
Laguna St
Buchanan St
Webster St
Fillmore St
Steiner St
Pierce St
Scott St
Baker St
Lyon St
Presidio Ave
Cherry St
Maple St
Locust St
Walnut St
Spruce St
Mayfair Dr
Presidio Terrace
Anza Vista
Turk St
Balboa St

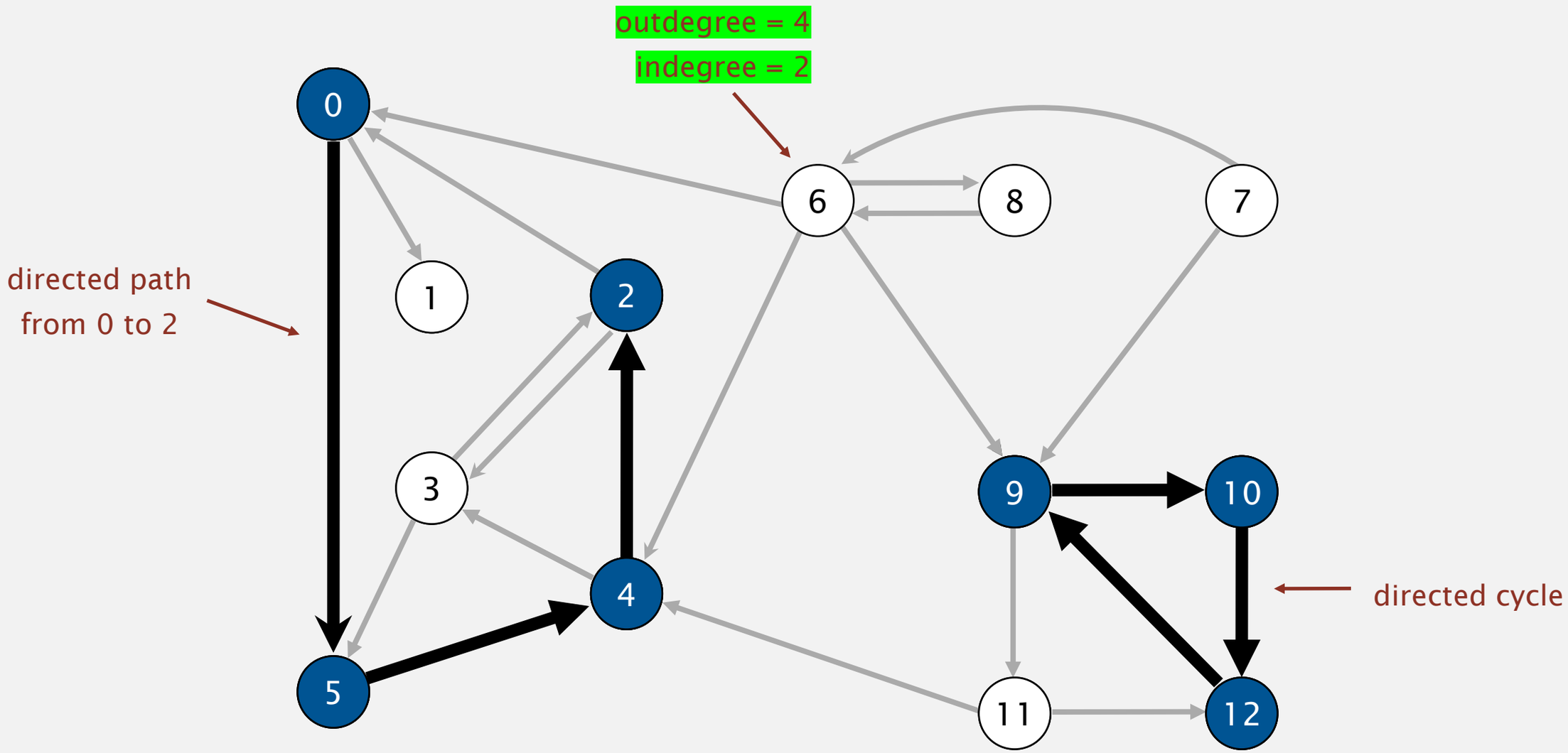
Digraph applications

digraph	vertex	directed edge
transportation	street intersection	one-way street
web	web page	hyperlink
food web	species	predator–prey relationship
WordNet	synset	hypernym
scheduling	task	precedence constraint
financial	bank	transaction
cell phone	person	placed call
infectious disease	person	infection
game	board position	legal move
citation	journal article	citation
object graph	object	pointer
inheritance hierarchy	class	inherits from
control flow	code block	jump

copyrighted content – Do not share

Directed graph terminology

Digraph. Set of vertices connected pairwise by directed edges.



copyrighted content – Do not share

Some digraph problems

problem	description
s-t path	*Is there a path from s to t ?*
shortest s-t path	*What is the shortest path from s to t ?*
directed cycle	*Is there a directed cycle in the graph ?*
topological sort	*Can the digraph be drawn so that all edges point upwards?*
strong connectivity	*Is there a directed path between every pairs of vertices ?*
transitive closure	*For which vertices v and w is there a directed path from v to w ?*
PageRank	*What is the importance of a web page ?*

copyrighted content – Do not share

DIRECTED GRAPHS

- *introduction*
- *digraph API*
- *depth-first search*
- *breadth-first search*
- *topological sort*

copyrighted content – Do not share

Digraph API

Almost identical to Graph API.

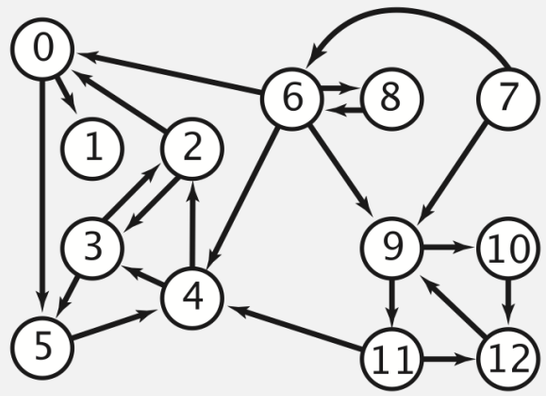
public class Digraph	//directed graph	
	Digraph(int V)	*create an empty digraph with V vertices*
	Digraph(In in)	*create a digraph from input stream*
void	addEdge(int v, int w)	*add a directed edge v→w*
Iterable<Integer>	adj(int v)	*vertices adjacent from v*
int	V()	*number of vertices*
int	E()	*number of edges*
Digraph	reverse()	*reverse of this digraph*
String	toString()	*string representation*

copyrighted content – Do not share

Digraph API

tinyDG.txt

```
V → 13
    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
     ⋮
```



% **java Digraph tinyDG.txt**

0->5 //vertex 0 is source, 5 is target

0->1

2->0

2->3

3->5

3->2

4->3

4->2

5->4

⋮

11->4

11->12

12->9

```
In in = new In(args[0]);
Digraph G = new Digraph(in);

for (int v = 0; v < G.V(); v++)
   for (int w : G.adj(v))
      StdOut.println(v + "->" + w);
```

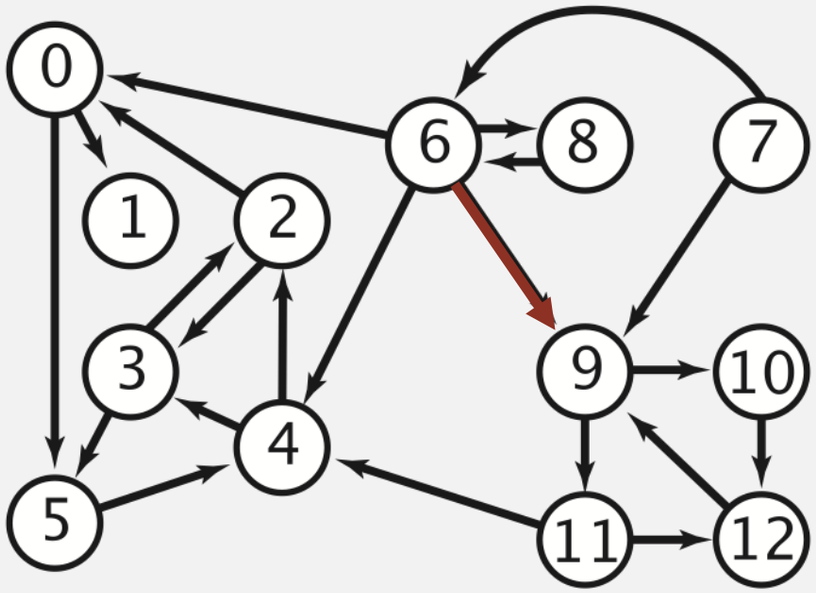
read digraph from input stream

print each edge (once) //edges have direction

copyrighted content – Do not share

Digraph representation: set of edges

Store a list of the edges (linked list or array).



2 edges with different direction

0	1
0	5
2	0
2	3
3	2
3	5
4	2
4	3
5	4
6	0
6	4
6	8
6	9
7	6
7	9
8	6
9	10
9	11
10	12
11	4
11	12
12	9

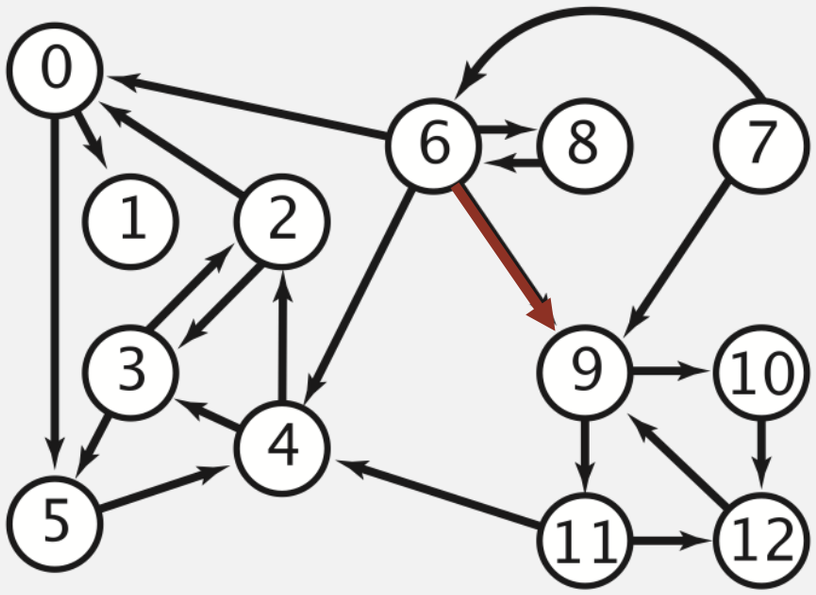
2 edges with different direction

copyrighted content – Do not share

Digraph representation: adjacency matrix

Maintain a two-dimensional V-by-V boolean array;

for each edge $v \rightarrow w$ in the digraph: adj[v][w] = true.



from \ to

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	1	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	1	0	0	0	0	0	0	0
4	0	0	1	1	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	1	1	0	0	0
7	0	0	0	0	0	0	1	0	0	1	0	0	0
8	0	0	0	0	0	0	1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	0
10	0	0	0	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	1	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	0	0

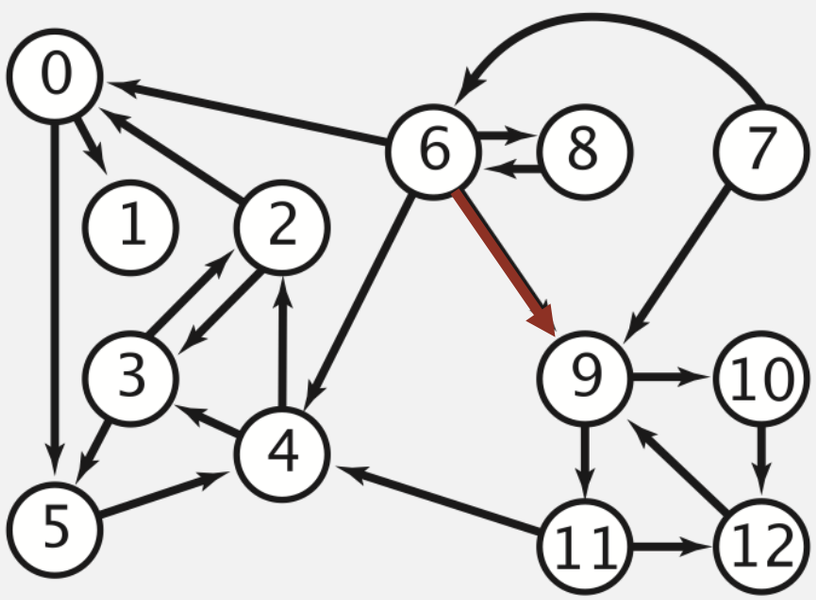
Space consumption: V^2

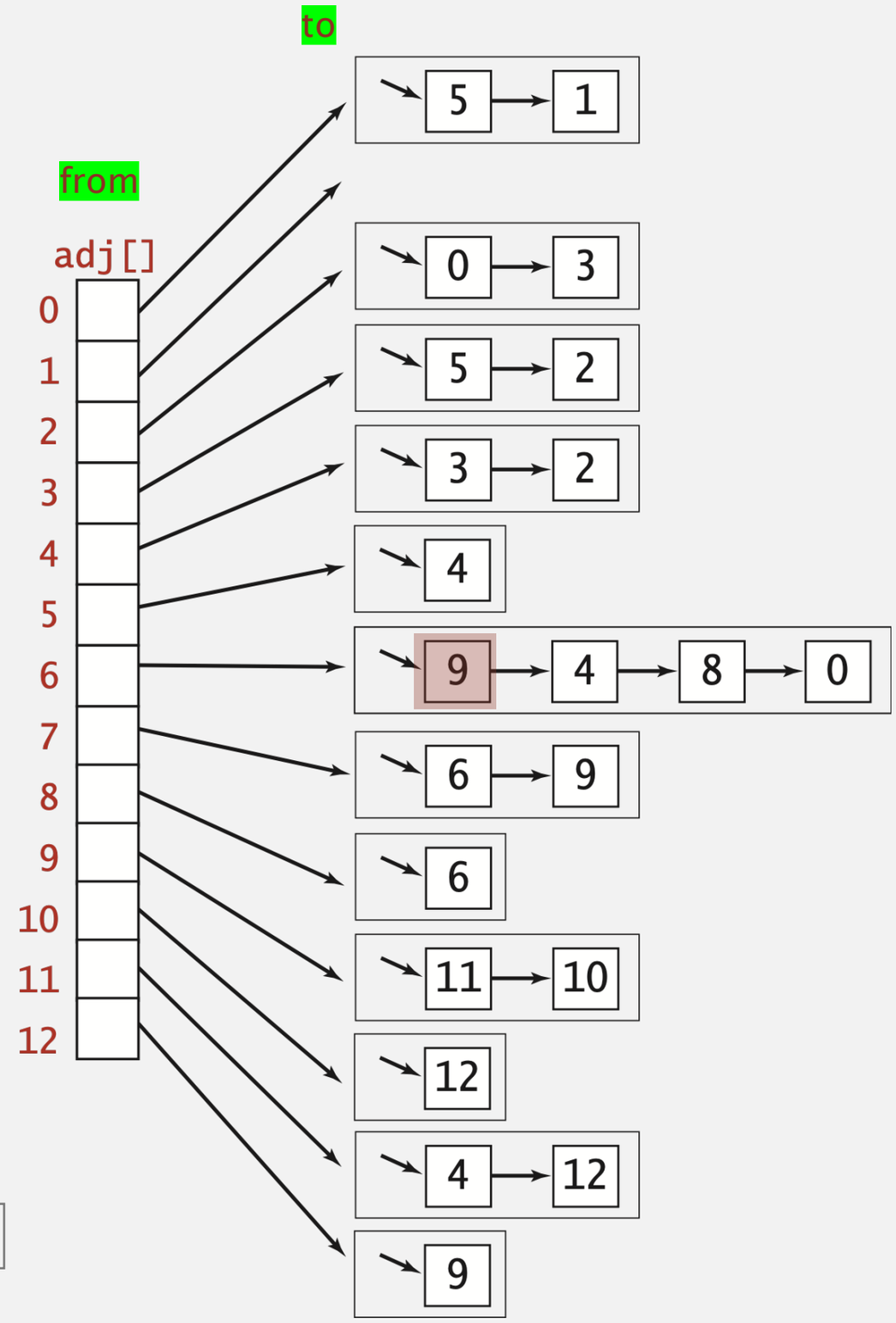
Note: parallel edges disallowed

copyrighted content – Do not share

Digraph representation: adjacency lists

Maintain vertex-indexed array of lists.





Assuming a linked list to store the adjacency list.

Space consumption: **V + 2E**

2 units per node: the vertex and the next pointer.

copyrighted content – Do not share

Directed graphs: quiz 1

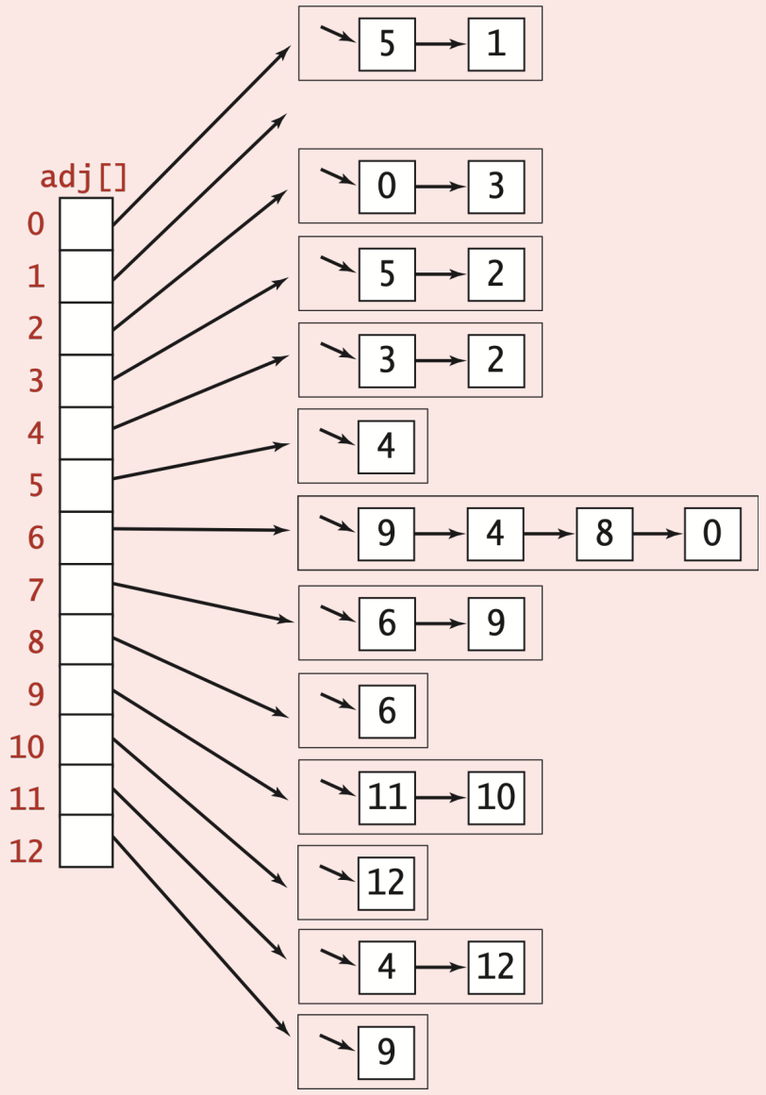
Which is order of growth of running time of removing an edge v→w from a digraph uses the adjacency-lists representation, where V is the number of vertices and E is the number of edges?

A. 1

B. $outdegree(v)$

C. $indegree(w)$

D. $outdegree(v) + indegree(w)$



copyrighted content – Do not share

Directed graphs: quiz 2

Which is order of growth of running time of the following code fragment if the digraph uses the adjacency-lists representation, where V is the number of vertices and E is the number of edges?

A. V

B. $E + V$

C. V^2

D. $V E$

```
for (int v = 0; v < G.V(); v++)
   for (int w : G.adj(v))
      StdOut.println(v + "->" + w);
```

prints each edge exactly once

copyrighted content – Do not share

Digraph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent from v.
- Real-world graphs tend to be sparse (not dense).

proportional to V edges

proportional to V^2 edges

representation	space	insert edgefrom v to w	edge from v to w?	iterate over vertices adjacent from v?
list of edges	E	1	E	E
adjacency matrix	V^2	1 †	1	V
adjacency lists	$E + V$	1	$outdegree(v)$	$outdegree(v)$

† disallows parallel edges

copyrighted content – Do not share

Adjacency-lists graph representation (review): Java implementation

```
public class Graph  //undirected 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

create empty graph with V vertices

add edge v-w

iterator for vertices adjacent to v

copyrighted content – Do not share

Adjacency-lists digraph representation: Java implementation

```
public class Digraph   //directed graph
{
        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);    //add one edge only

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

adjacency lists

create empty digraph with V vertices

add edge v→w

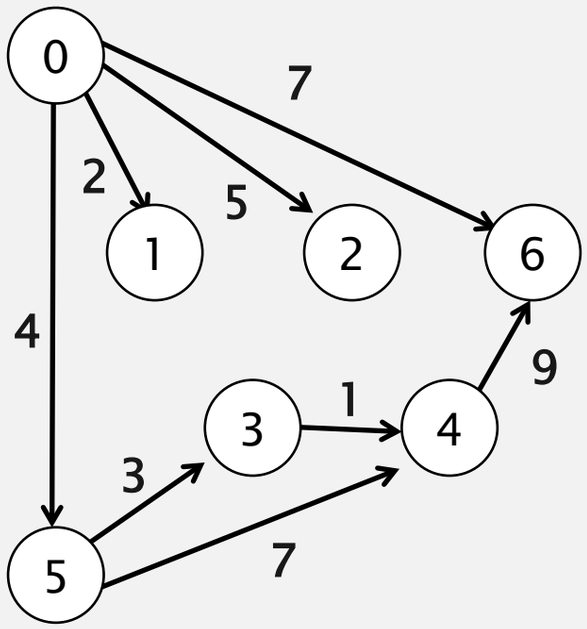
iterator for vertices adjacent from v

copyrighted content – Do not share

Weighted Digraph

Edges have a **cost** associated with them.

- For example, distance, dollar amount, traffic load, etc.



The cost of path 0, 6 is 7

The cost of path 0, 5, 4, 6 is 20

The cost of path 0, 5, 3, 4, 6 is 17

The **least cost path** from 0 to 6 is 7

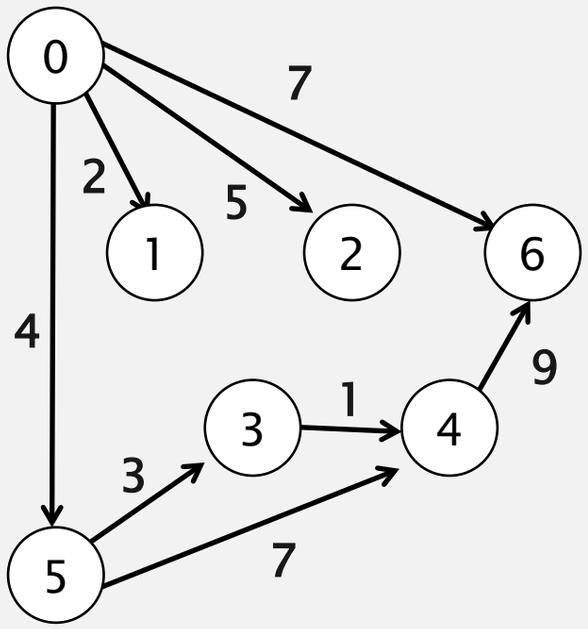
Least expensive path

copyrighted content – Do not share

Digraph representation: adjacency matrix

Maintain a two-dimensional V-by-V array;

for each edge $v \rightarrow w$ in graph: adj[v][w] = **edge cost or value**.



	0	1	2	3	4	5	6
0	0	2	5	0	0	4	7
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	1	0	0
4	0	0	0	0	0	0	9
5	0	0	0	3	7	0	0
6	0	0	0	0	0	0	0

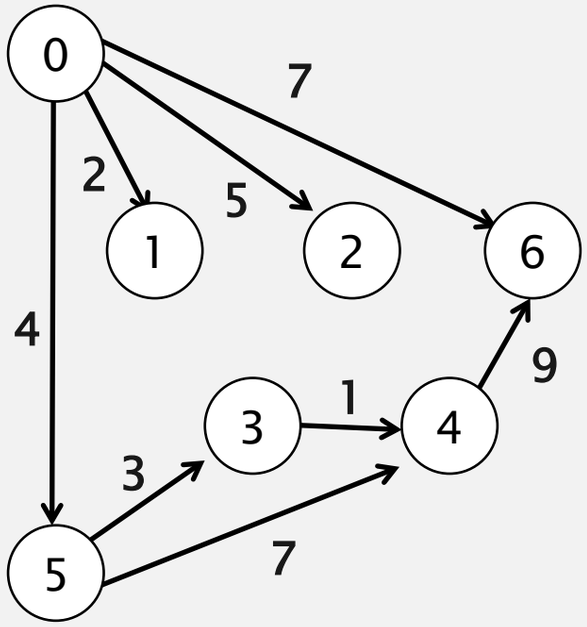
Space consumption: V^2

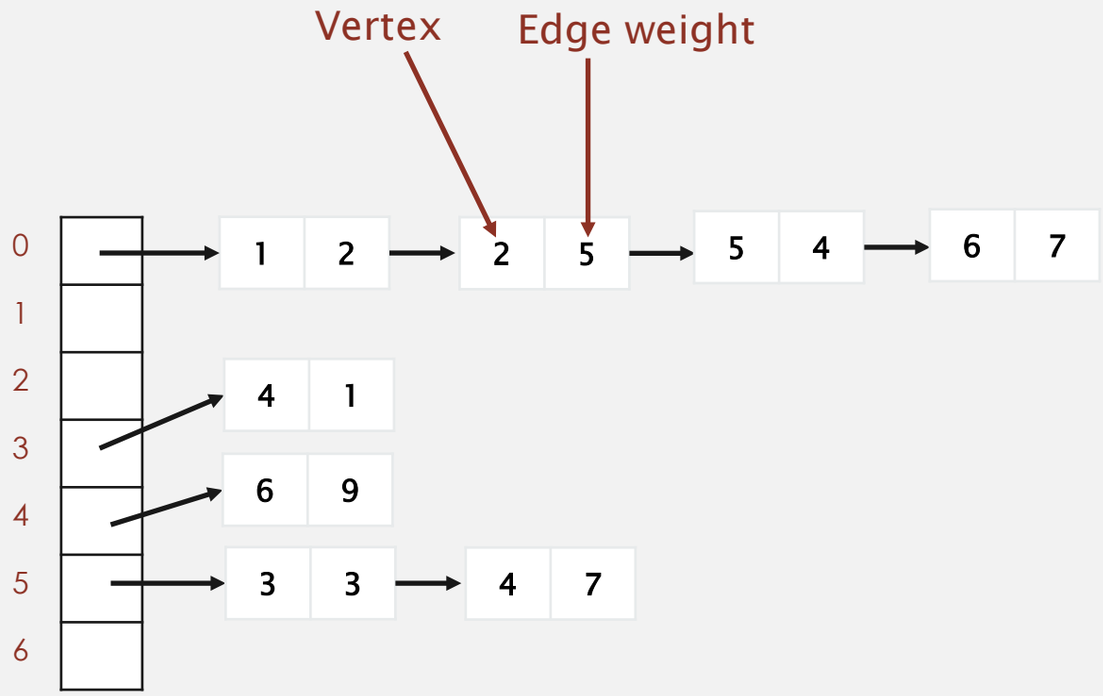
copyrighted content – Do not share

Graph representation: adjacency lists

Maintain vertex-indexed array of lists.

Add edge weight to node.





Assuming a linked list to store the adjacency list.

Space consumption: **V + 3E**

3 units per node: the vertex, the edge weight, and the next pointer.

copyrighted content – Do not share

Undirected graphs: quiz 4

What is the maximum number of edges on a Graph of V vertices?

A. $V * (V - 1) / 2$

B. $E + V$

C. V^2

D. $V E$

E. $V * (V - 1)$

copyrighted content – Do not share

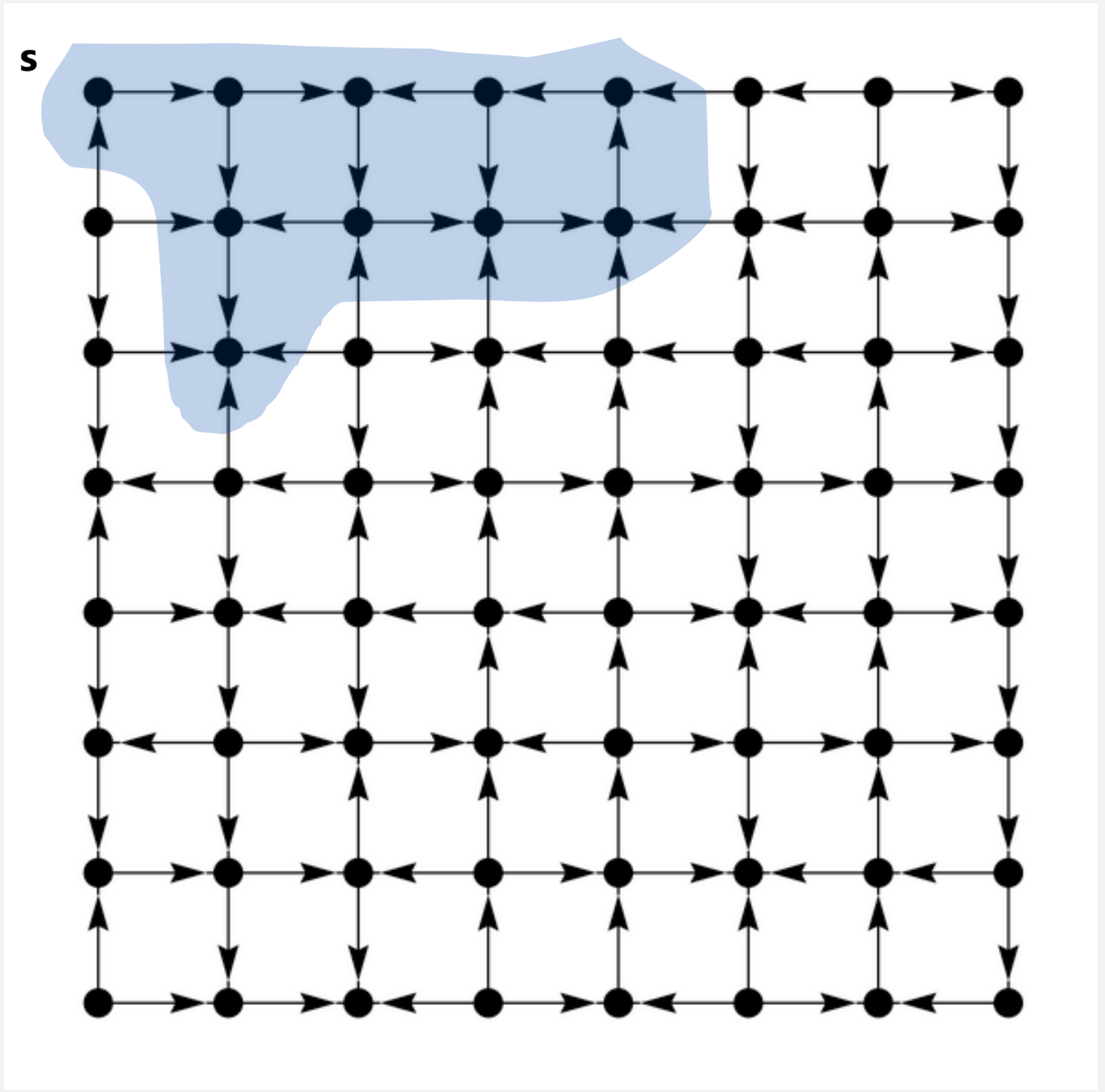
Directed Graphs

- *introduction*
- *digraph API*
- *depth-first search*
- *breadth-first search*
- *topological sort*

copyrighted content – Do not share

Reachability

Problem. Find all vertices reachable from s along a directed path.



copyrighted content – Do not share

Depth-first search in digraphs

Same method as for undirected graphs.

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

DFS (to visit a vertex v)

Mark vertex v.

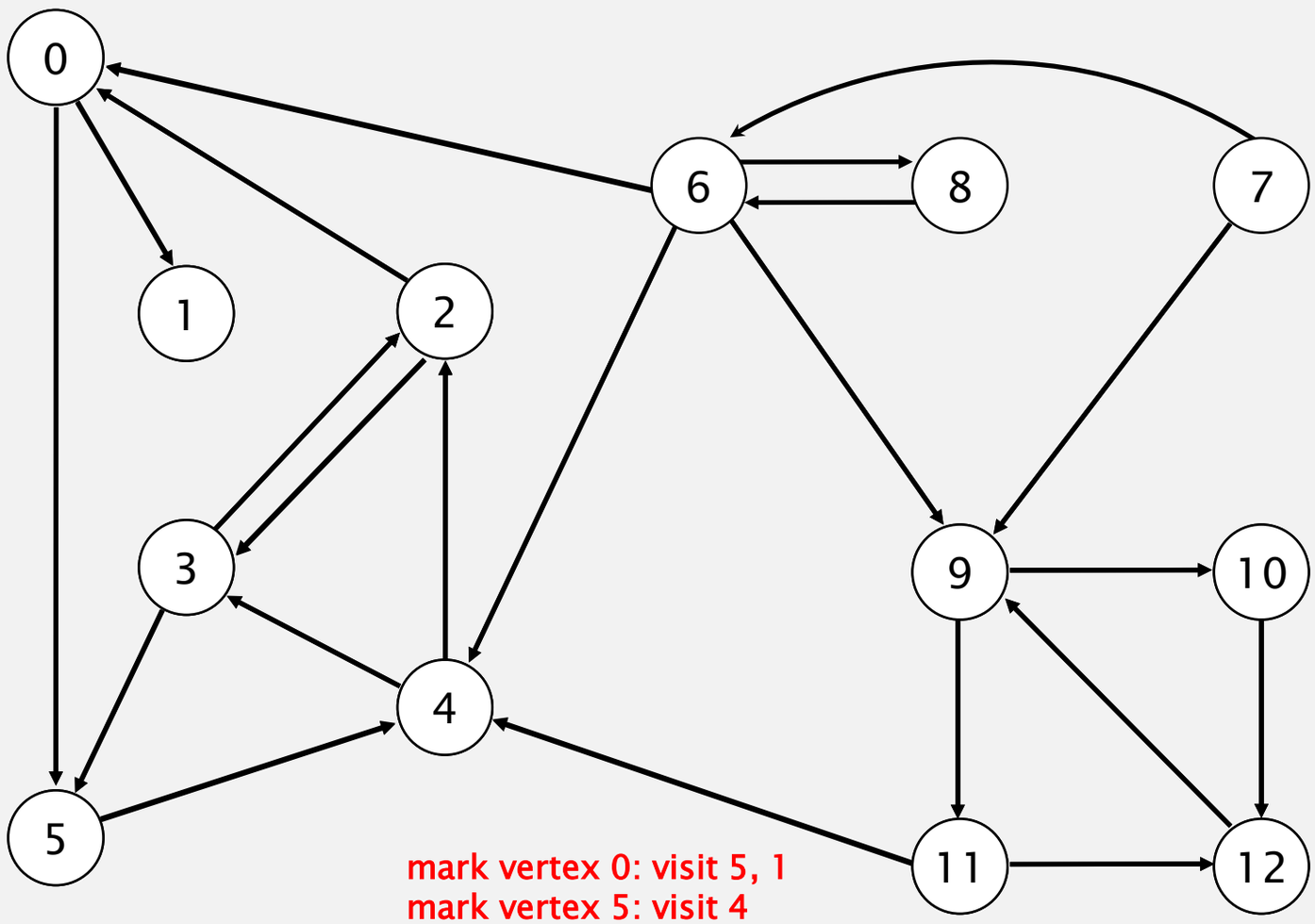
Recursively visit all unmarked vertices w adjacent from v.

copyrighted content – Do not share

Depth-first search demo

To visit a vertex v :

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



a directed graph

```
mark vertex 0: visit 5, 1
mark vertex 5: visit 4
mark vertex 4: visit 3, 2
mark vertex 3: visit 5, 2
mark vertex 2: visit 0, 3
done 2, 3, 4, 5
mark vertex 1: done
done 0
```

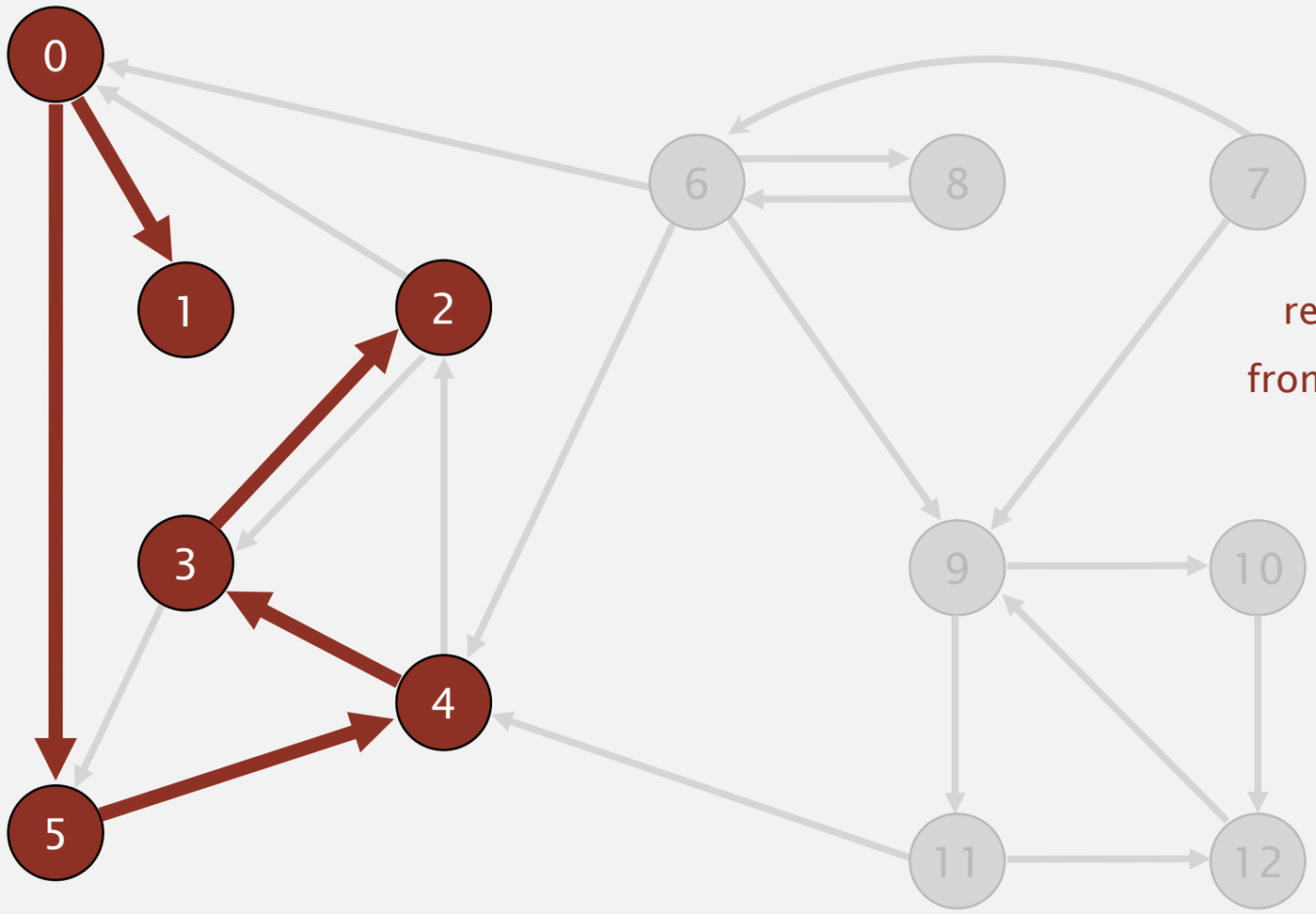
4→2
2→3
3→2
6→0
0→1
2→0
11→12
12→9
9→10
9→11
8→9
10→12
11→4
4→3
3→5
6→8
8→6
5→4
0→5
6→4

copyrighted content – Do not share

Depth-first search demo

To visit a vertex v :

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



reachable from vertex 0

v	marked[]	edgeTo[]
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	-

reachable from 0

copyrighted content – Do not share

Depth-first search (in undirected graphs)

Recall code for undirected graphs.

```
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 connected to s

constructor marks vertices connected to s

recursive DFS does the work

client can ask whether any vertex is connected to s

Depth-first search (in directed graphs)

Code for directed graphs identical to undirected one.

```
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 connected to s

constructor marks vertices connected to s

recursive DFS does the work

client can ask whether any vertex is connected to s

copyrighted content – Do not share

Reachability application: program control-flow analysis

LO 9B.11, 9B.13

Every program is a digraph.

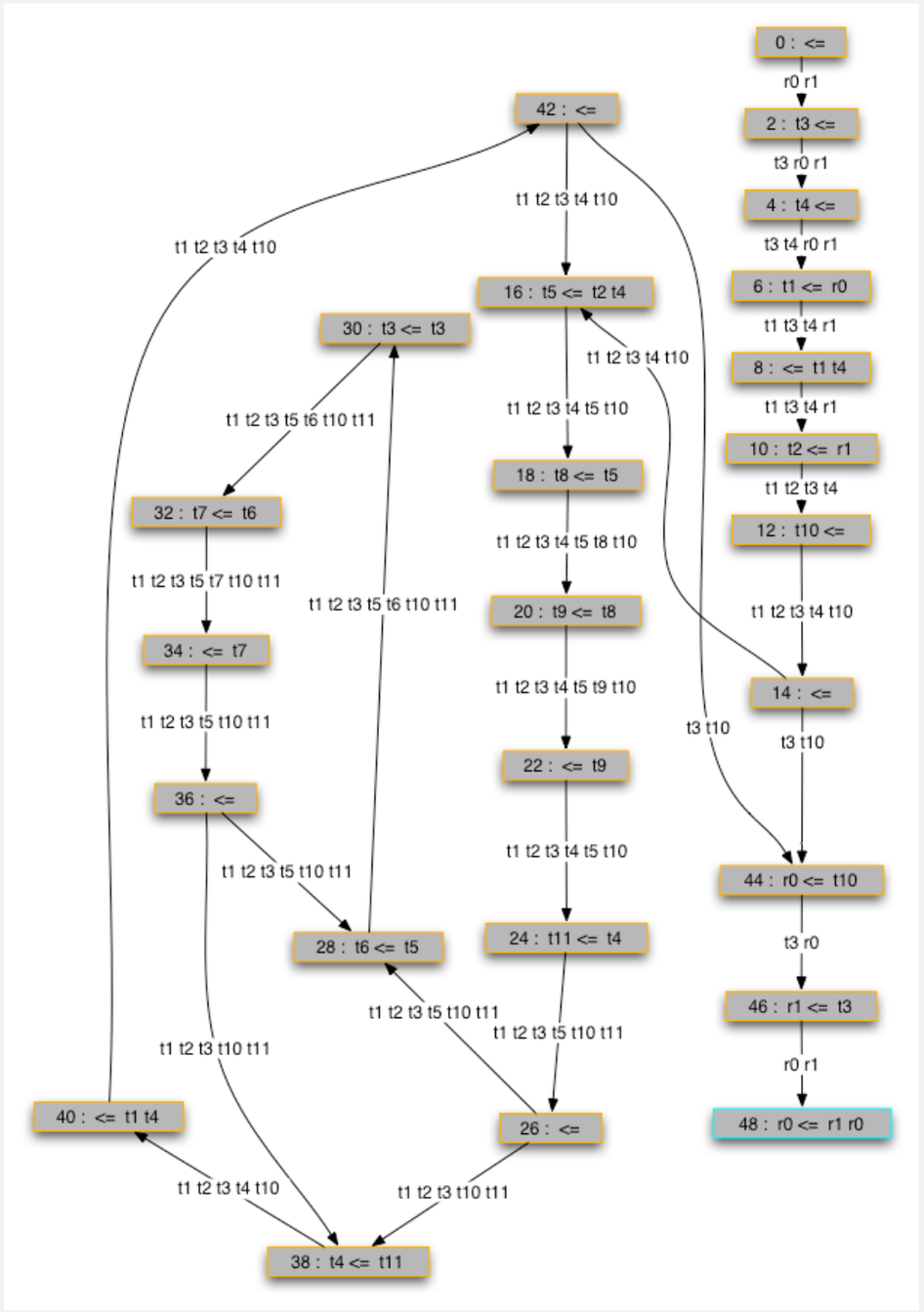
- Vertex = basic block of instructions (straight-line program).
- Edge = jump.

Dead-code elimination.

Find (and remove) unreachable code.

Infinite-loop detection.

Determine whether exit is unreachable.



copyrighted content – Do not share

Reachability application: mark–sweep garbage collector

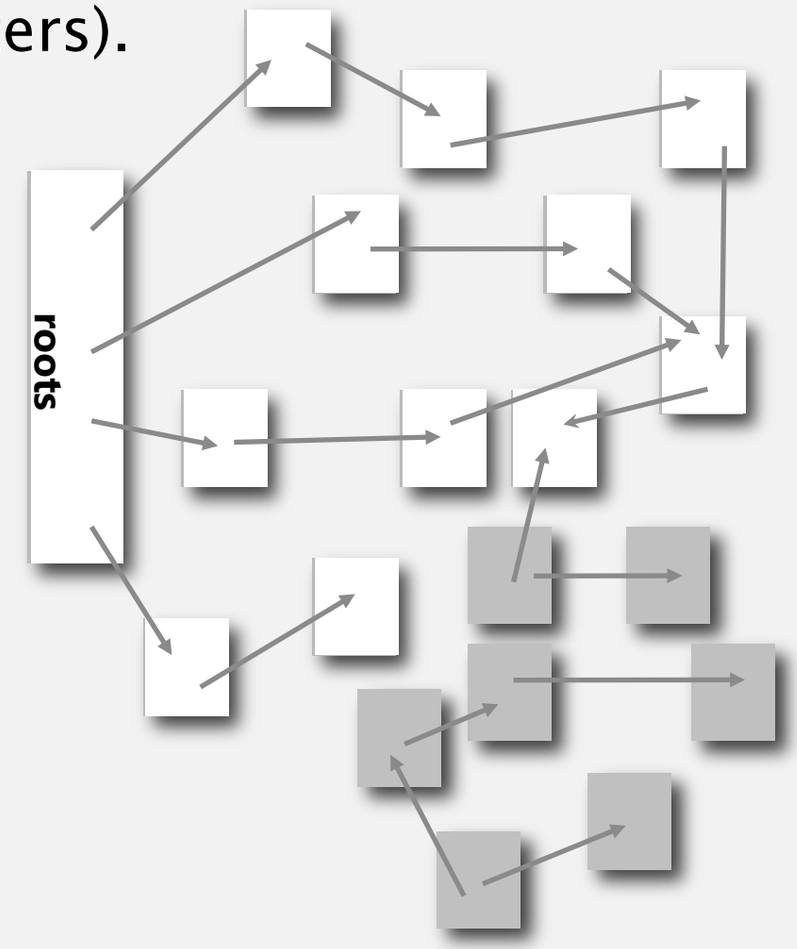
LO 9B.11, 9B.13

Every data structure 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).



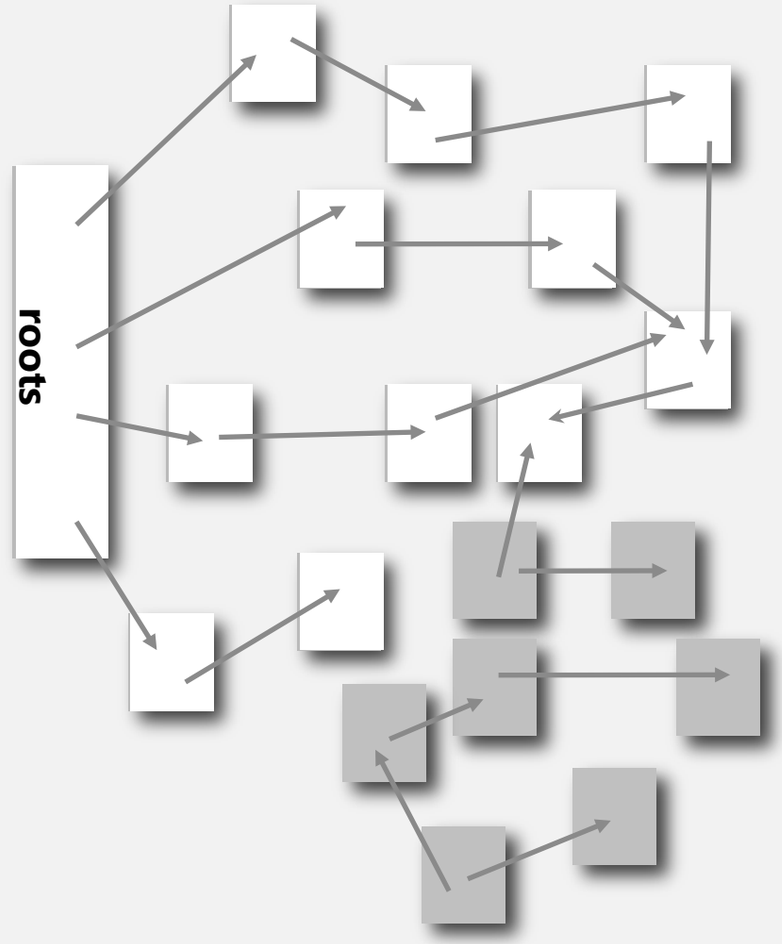
copyrighted content – Do not share

Reachability application: mark–sweep garbage collector

Mark–sweep algorithm. [McCarthy, 1960]

- Mark: mark all reachable objects.
- 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).



copyrighted content – Do not share

Depth-first search in digraphs summary

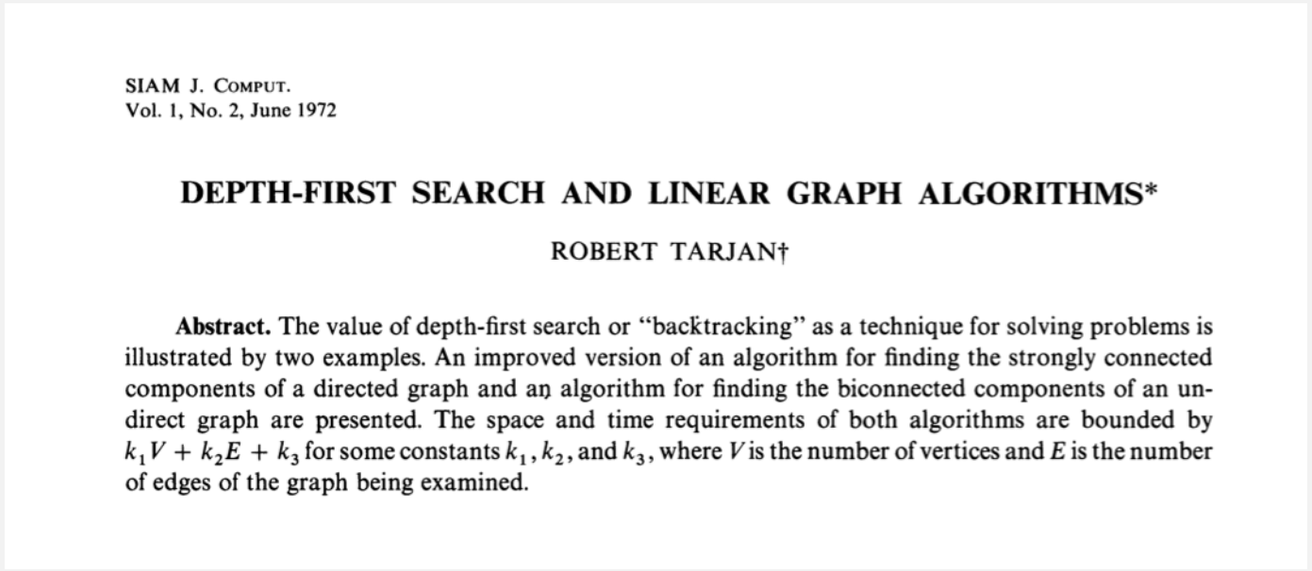
LO 9B.11, 9B.13

DFS enables direct solution of simple digraph problems.

- Reachability.
- Path finding.
- Topological sort.
- Directed cycle detection.

Basis for solving difficult digraph problems.

- 2-satisfiability.
- Directed Euler path.
- Strongly-connected components.



SIAM J. COMPUT.
Vol. 1, No. 2, June 1972

DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS*

ROBERT TARJAN†

Abstract. The value of depth-first search or "backtracking" as a technique for solving problems is illustrated by two examples. An improved version of an algorithm for finding the strongly connected components of a directed graph and an algorithm for finding the biconnected components of an undirect graph are presented. The space and time requirements of both algorithms are bounded by $k_1V + k_2E + k_3$ for some constants k_1, k_2, and k_3, where V is the number of vertices and E is the number of edges of the graph being examined.

copyrighted content – Do not share

Directed graphs: quiz 3

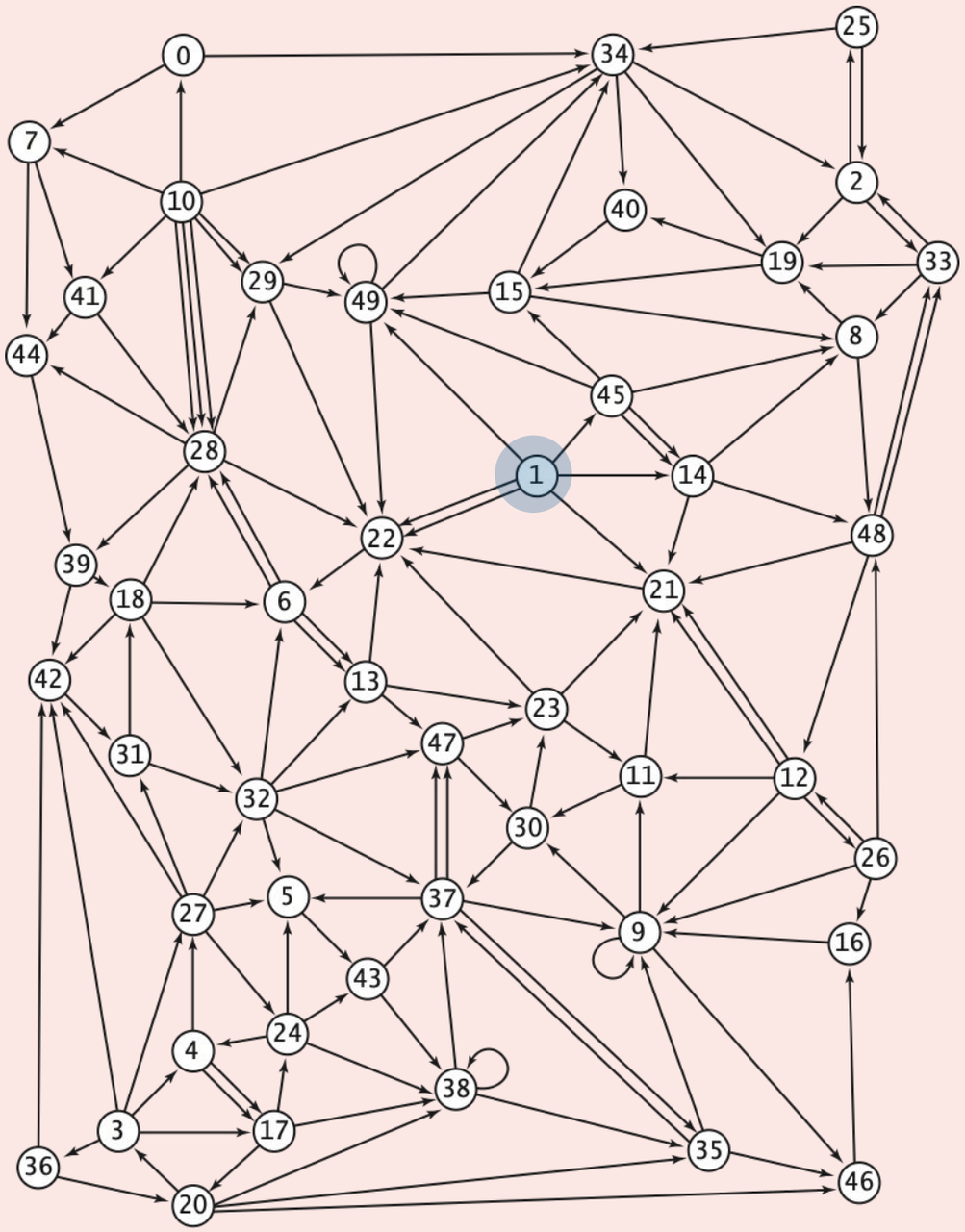
Suppose that you want to design a web crawler. Which graph search algorithm should you use?

A. depth-first search

B. breadth-first search

C. either A or B

D. neither A nor B



copyrighted content – Do not share

Web crawler output (an example)

BFS crawl

http://www.rutgers.edu
http://www.w3.org
http://ogp.me
http://giving.Rutgers.edu
http://www.princetonartmuseum.org
http://www.goprincetontigers.com
http://library. Rutgers.edu
http://helpdesk.princeton.edu
http://tigernet. Rutgers.edu
http://alumni.princeton.edu
http://gradschool.princeton.edu
http://vimeo.com
http://princetonusg.com
http://artmuseum.princeton.edu
http://jobs.princeton.edu
http://odoc.princeton.edu
http://blogs.princeton.edu
http://www.facebook.com
http://twitter.com
http://www.youtube.com
http://deimos.apple.com
http://qeprize.org
http://en.wikipedia.org

DFS crawl

http://www.rutgers.edu
http://deimos.apple.com
http://www.youtube.com
http://www.google.com
http://news.google.com
http://csi.gstatic.com
http://googlenewsblog.blogspot.com
http://labs.google.com
http://groups.google.com
http://img1.blogblog.com
http://feeds.feedburner.com
http:/buttons.googlesyndication.com
http://fusion.google.com
http://insidesearch.blogspot.com
http://agoogleaday.com
http://static.googleusercontent.com
http://searchresearch1.blogspot.com
http://feedburner.google.com
http://www.dot.ca.gov
http://www.TahoeRoads.com
http://www.LakeTahoeTransit.com
http://www.laketahoe.com
http://ethel.tahoeguide.com

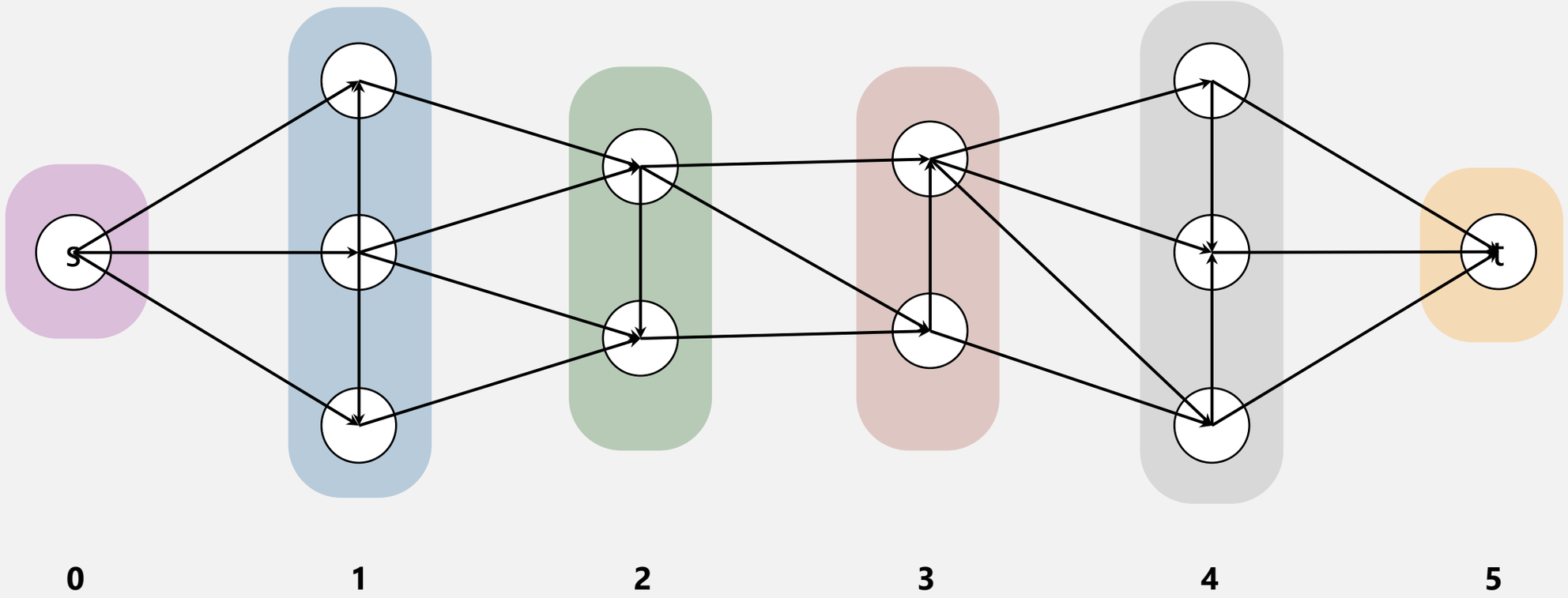
Directed Graphs

- *introduction*
- *digraph API*
- *depth-first search*
- *breadth-first search*
- *topological sort*

copyrighted content – Do not share

Shortest directed paths

Problem. Find directed path from s to each vertex that uses fewest edges.



copyrighted content – Do not share

Breadth-first search in digraphs

Same method as for undirected graphs.

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

BFS (from source vertex s)

Put s onto a FIFO queue, and mark s as visited.

Repeat until the queue is empty:

- **remove the least recently added vertex v**
- **for each unmarked vertex adjacent from v:**

add to queue and mark as visited.

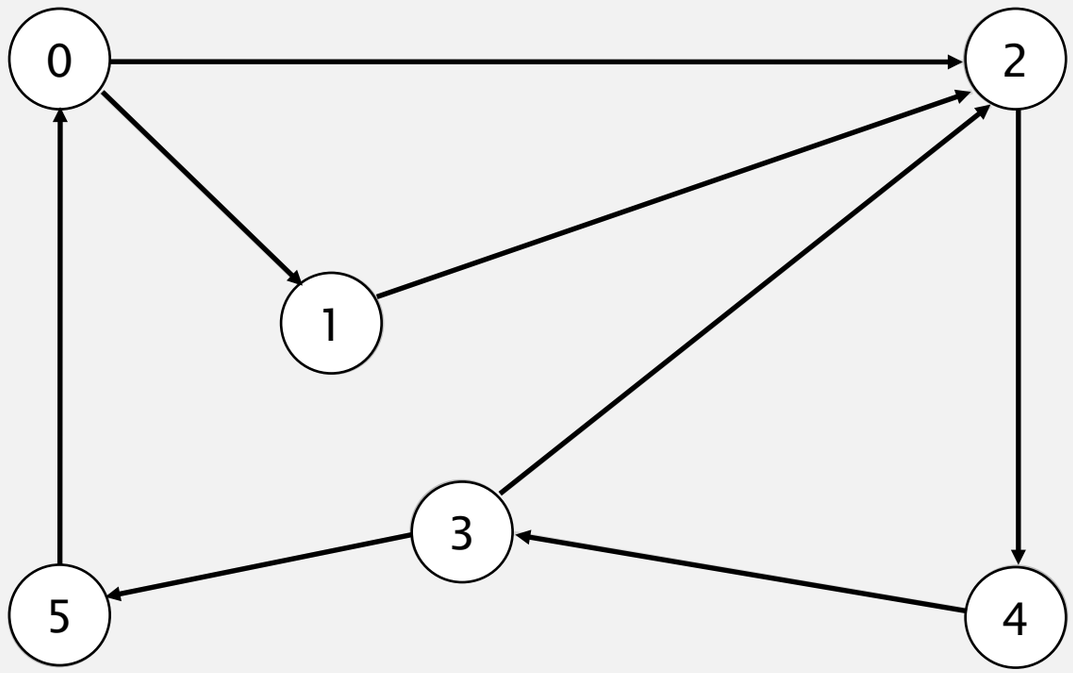
Proposition. BFS computes directed path with fewest edges from s to each vertex in time proportional to $E + V$.

copyrighted content – Do not share

Directed breadth-first search demo

Repeat until queue is empty:

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



graph G

tinyDG2.txt

```
V
6        E
8
5 0
2 4
3 2
1 2
0 1
4 3
3 5
0 2
```

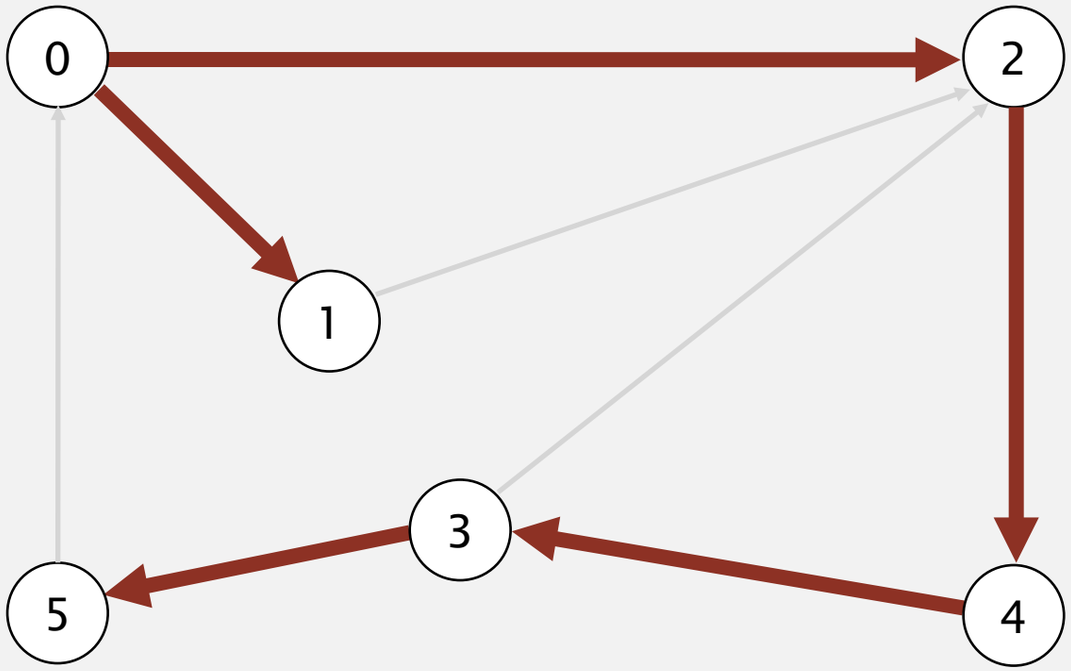
visit 0,
dequeue 0, enqueue 2, 1
dequeue 2, enqueue 4
dequeue 1,
dequeue 4, enqueue 3
dequeue 3, enqueue 5
dequeue 5, done

copyrighted content – Do not share

Directed breadth-first search demo

Repeat until queue is empty:

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



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

done

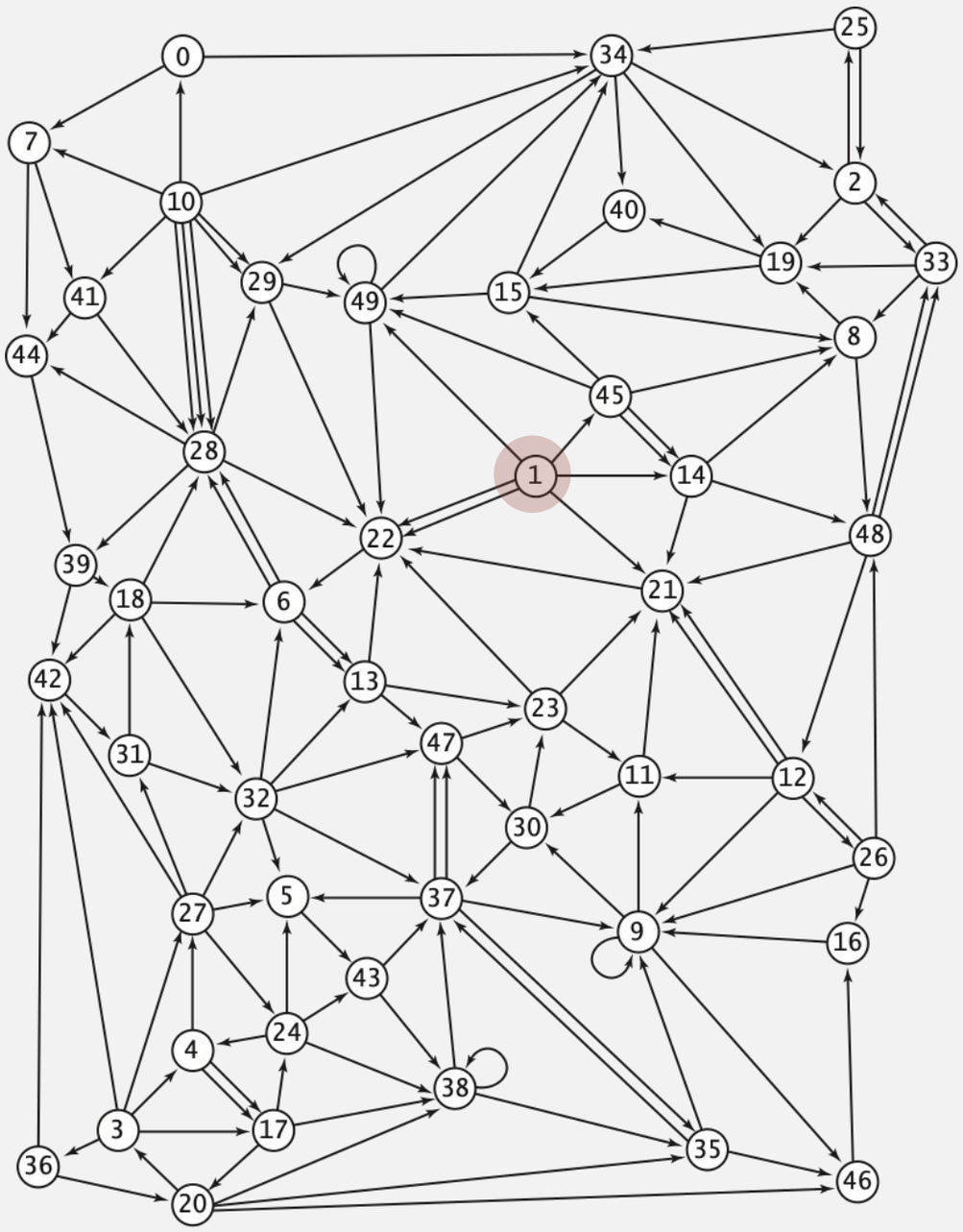
copyrighted content – Do not share

Breadth-first search in digraphs application: web crawler

Goal. Crawl web, starting from some root web page, say www.rutgerts.edu.

Solution. [BFS with implicit digraph]

- Choose root web page as source s.
- Maintain a Queue of websites to explore.
- Maintain a SET of marked websites.
- Dequeue the next website and enqueue any unmarked websites to which it links.



copyrighted content – Do not share

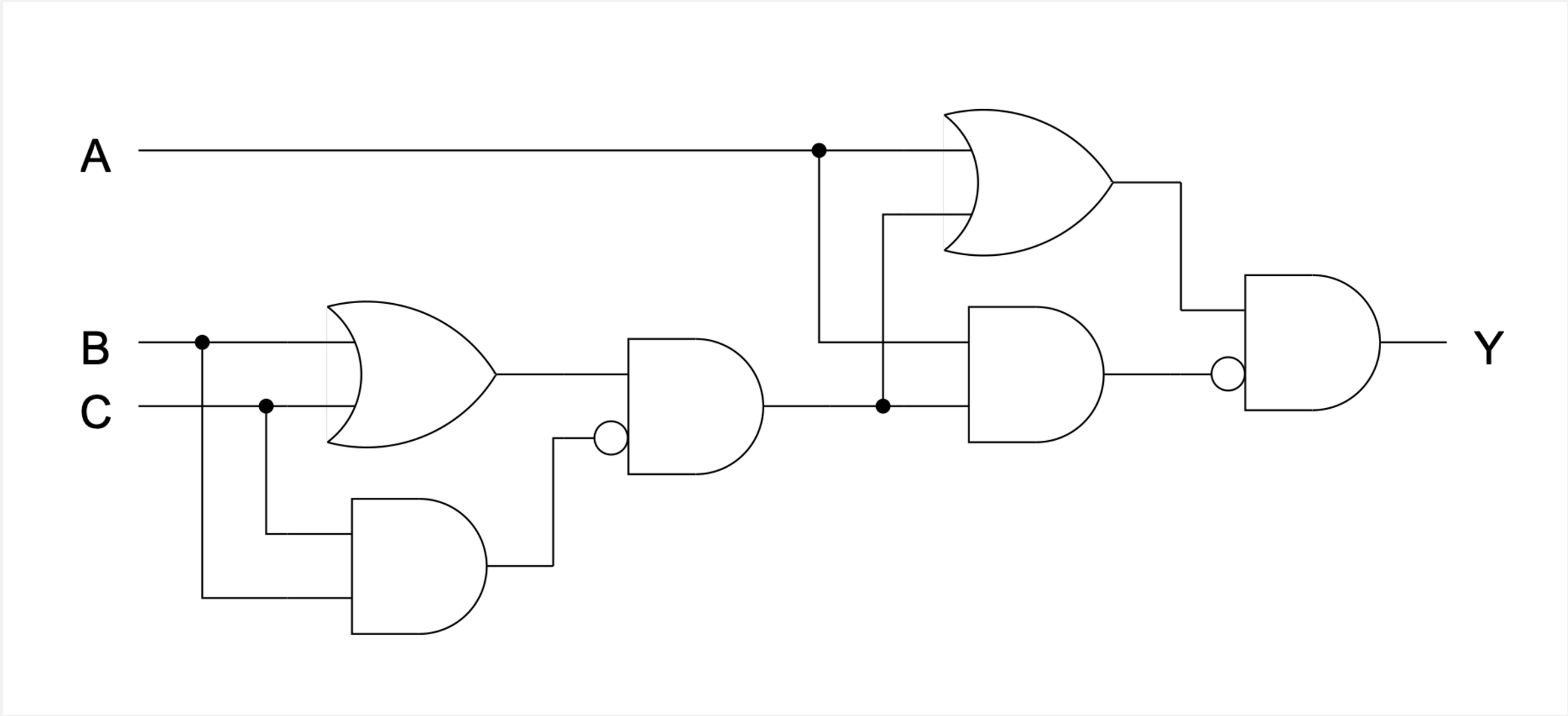
DIRECTED GRAPHS

- *introduction*
- *digraph API*
- *depth-first search*
- *breadth-first search*
- *topological sort*

copyrighted content – Do not share

Combinational circuit (optional)

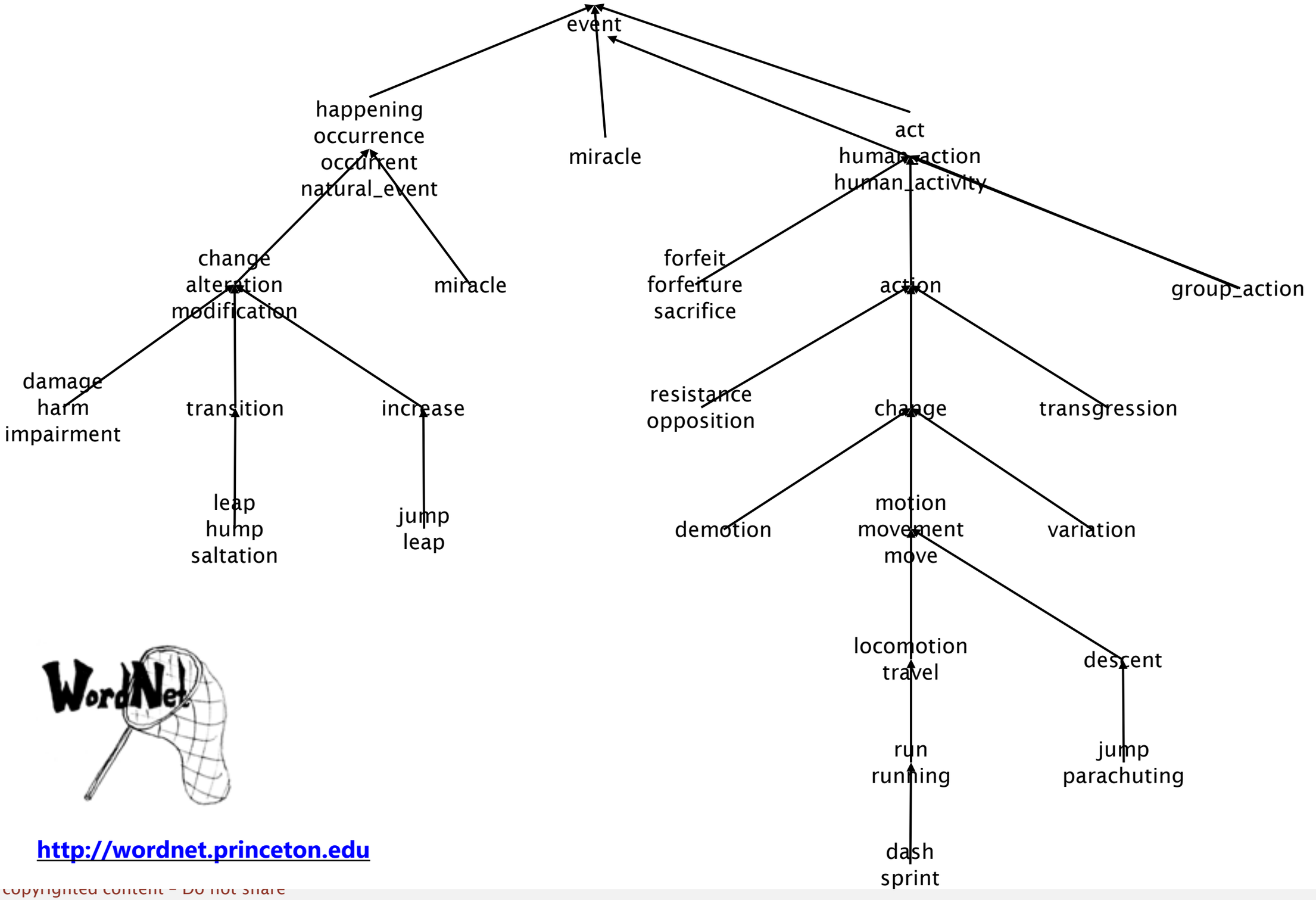
Vertex = logical gate; edge = wire.



copyrighted content – Do not share

WordNet digraph

Vertex = synset; edge = hypernym relationship.



http://wordnet.princeton.edu

copyrighted content – Do not share

Candy land

Vertex = game state; edge = next state.



copyrighted content – Do not share

Git digraph

Vertex = revision of repository; edge = revision relationship.

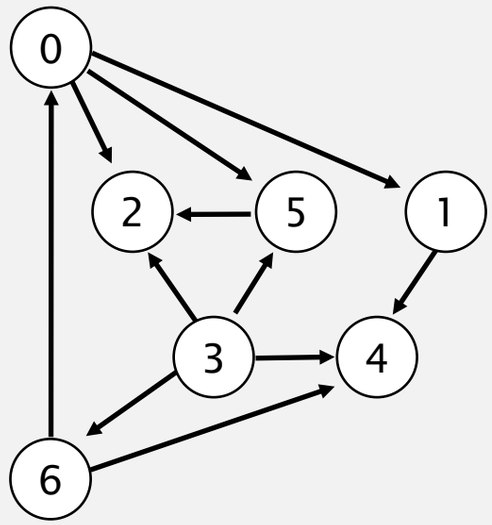
Precedence scheduling

Goal. Given a set of tasks to be completed with precedence constraints, in which order should we schedule the tasks?

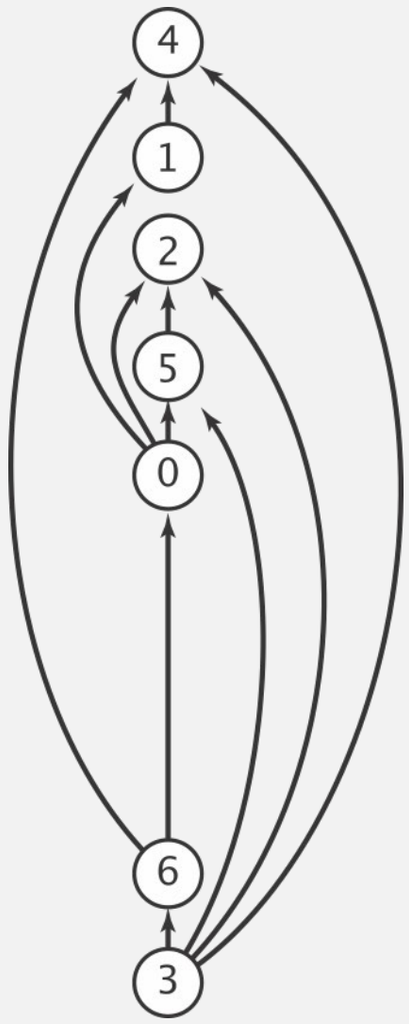
Digraph model. vertex = task; edge = precedence constraint.

0. Algorithms
1. Complexity Theory
2. Artificial Intelligence
3. Intro to CS
4. Cryptography
5. Scientific Computing
6. Advanced Programming

tasks



precedence constraint graph



feasible schedule

copyrighted content – Do not share

Topological sort

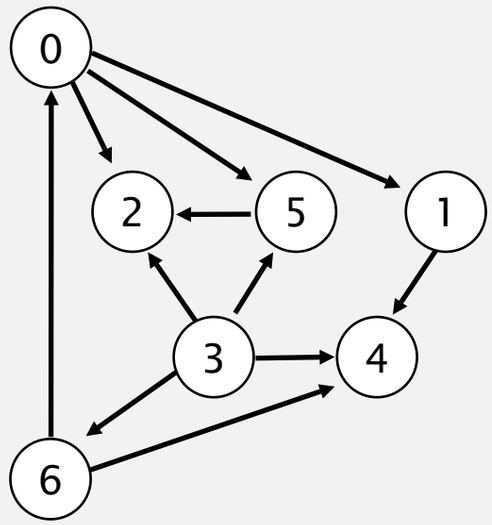
DAG. Directed acyclic graph.

Topological sort. Redraw DAG so all edges point upwards.

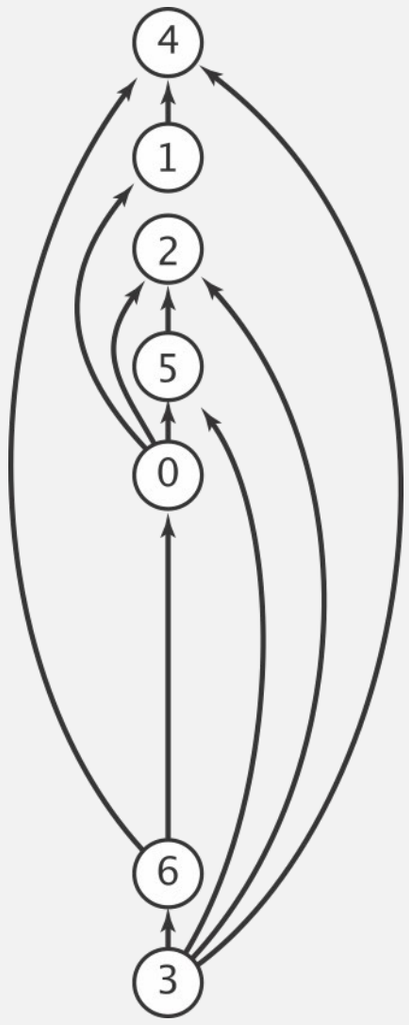
Network Topology

0→5
0→1
3→5
5→2
6→0
1→4
0→2
3→6
3→4
6→4
3→2

directed edges



DAG



topological order

copyrighted content – Do not share

Directed graphs: quiz 3

Suppose that you want to find a topological order of a DAG.

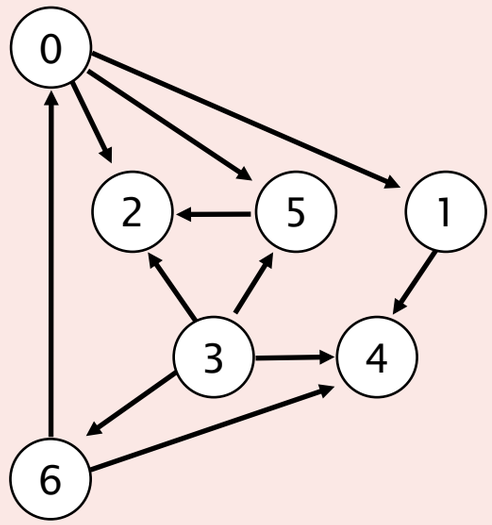
Which graph search algorithm should you use?

A. depth-first search

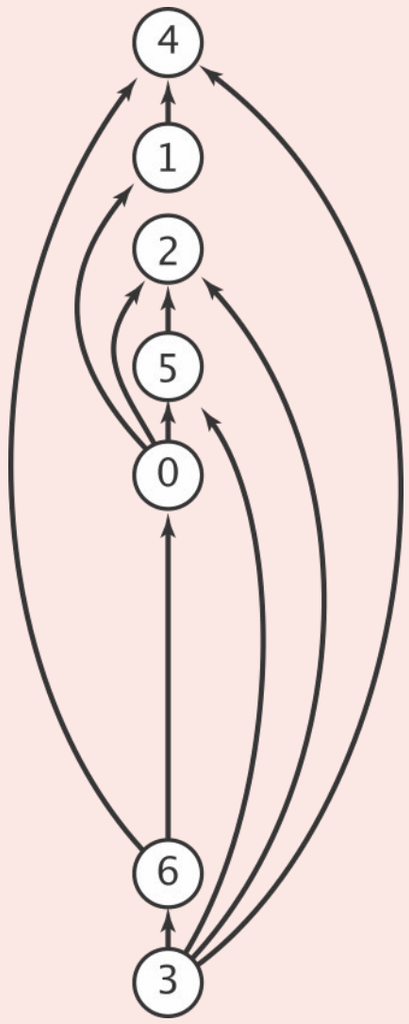
B. breadth-first search

C. either A or B

D. neither A nor B



DAG

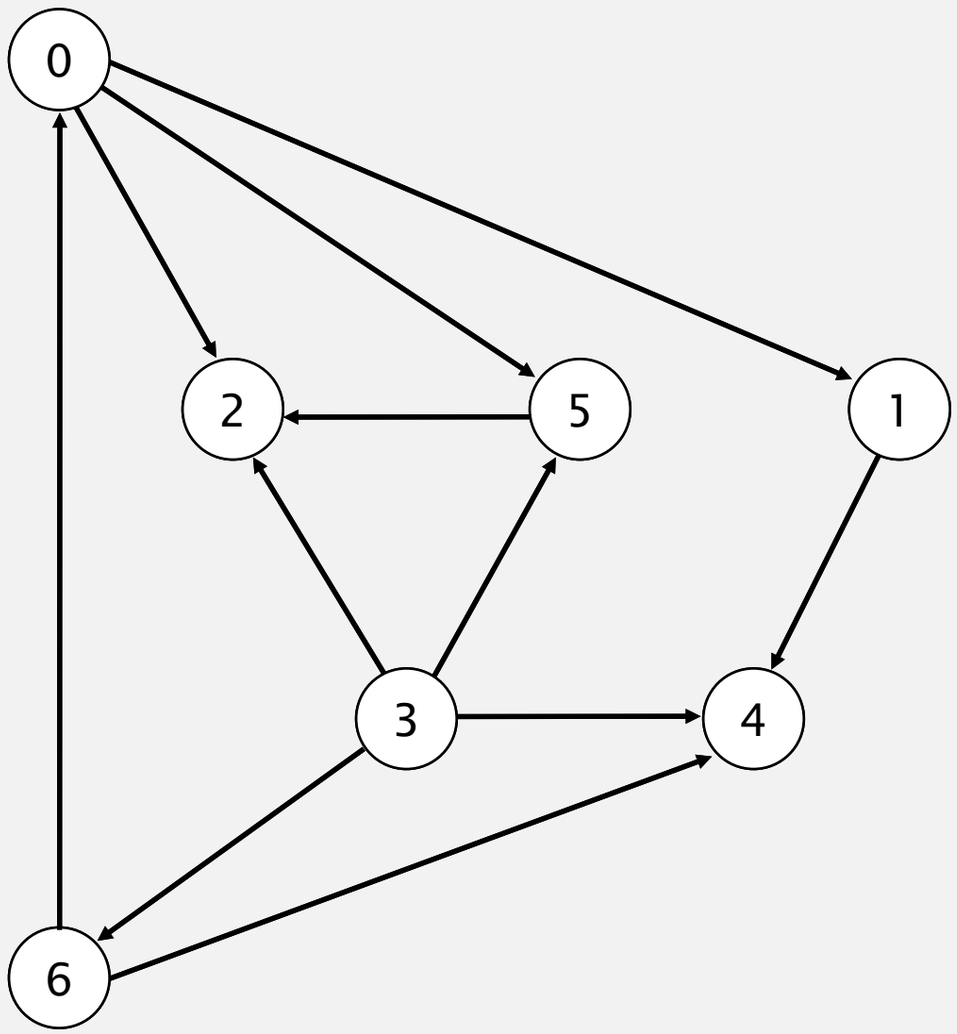


topological order

copyrighted content – Do not share

Topological sort demo

- Run depth-first search.
- Return vertices in reverse postorder.



tinyDAG7.txt

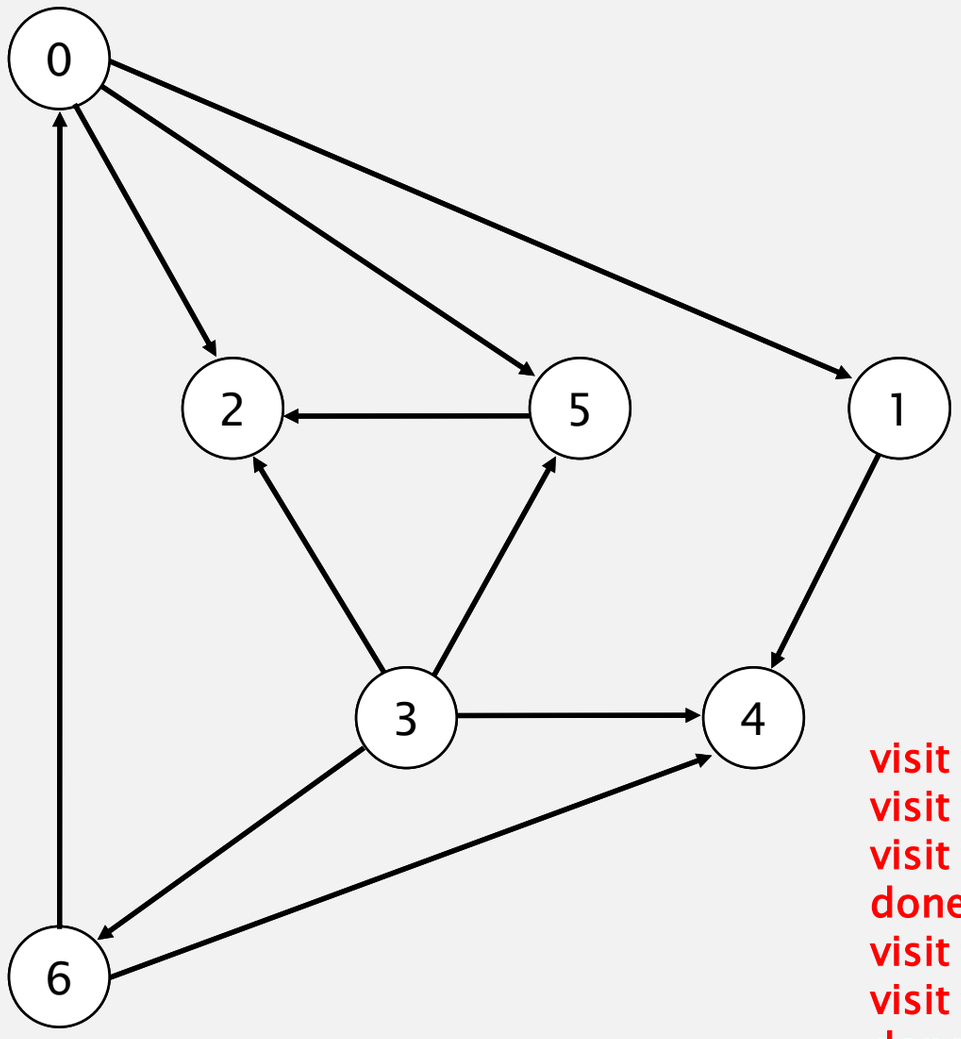
```
7
11
 0  5
 0  2
 0  1
 3  6
 3  5
 3  4
 5  2
 6  4
 6  0
 3  2
```

a directed acyclic graph

copyrighted content – Do not share

Topological sort demo

- Run depth-first search, starting from index 0 to index V - 1
- Return vertices in reverse postorder



postorder
(in order of returns from dfs(G, v).

4 1 2 5 0 6 3

add to list when "done"

topological order

3 6 0 5 2 1 4

visit vertex 0: check 1, 2, 5
visit vertex 1: check 4
visit vertex 4: done
done 1
visit vertex 2: done
visit vertex 5: done
done 0
visit vertex 3: check 6
visit vertex 6: done
done 3

done

copyrighted content – Do not share

Depth-first search order


```
public class DepthFirstOrder
{
   private boolean[] marked;
   private Stack<Integer> reversePostorder;
   public DepthFirstOrder(Digraph G)
   {
      reversePostorder = new Stack<Integer>();
      marked = new boolean[G.V()];
      for (int v = 0; v < G.V(); v++) //check every vertex
         if (!marked[v]) dfs(G, v);
   }

   private void dfs(Digraph G, int v)
   {
      marked[v] = true;
      for (int w : G.adj(v))
         if (!marked[w]) dfs(G, w);
      reversePostorder.push(v);  //done with the G.adj(v)
   }

 public Iterable<Integer> reversePostorder()
 {  return reversePostorder;  }

}
```

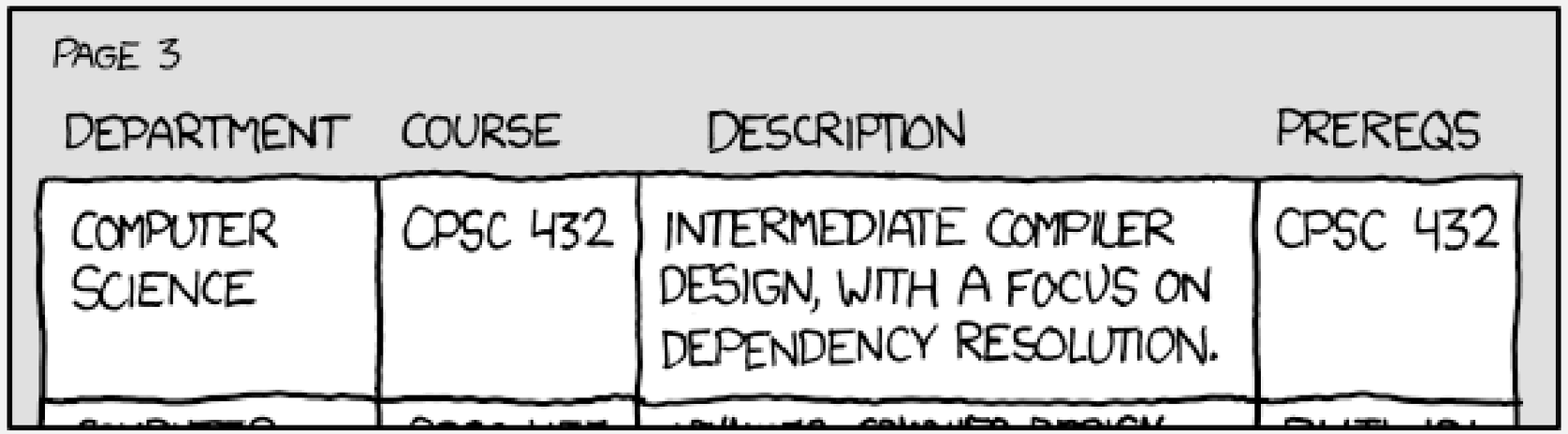
Three **vertex orderings** are of interest in typical applications:

- *Preorder*: put the vertex on a queue before the recursive calls.
- *Postorder*: put the vertex on a queue after the recursive calls.
- *Reverse postorder*: put the vertex on a stack after the recursive calls.

returns all vertices in "reverse DFS postorder"

Directed cycle detection application: precedence scheduling

Scheduling. Given a set of tasks to be completed with precedence constraints, in what order should we schedule the tasks?



http://xkcd.com/754

Remark. A directed cycle implies scheduling problem is infeasible.

copyrighted content – Do not share

Directed cycle detection application: cyclic inheritance

The Java compiler does cycle detection.

```
public class A extends B
{
   ...
}
```

```
public class B extends C
{
   ...
}
```

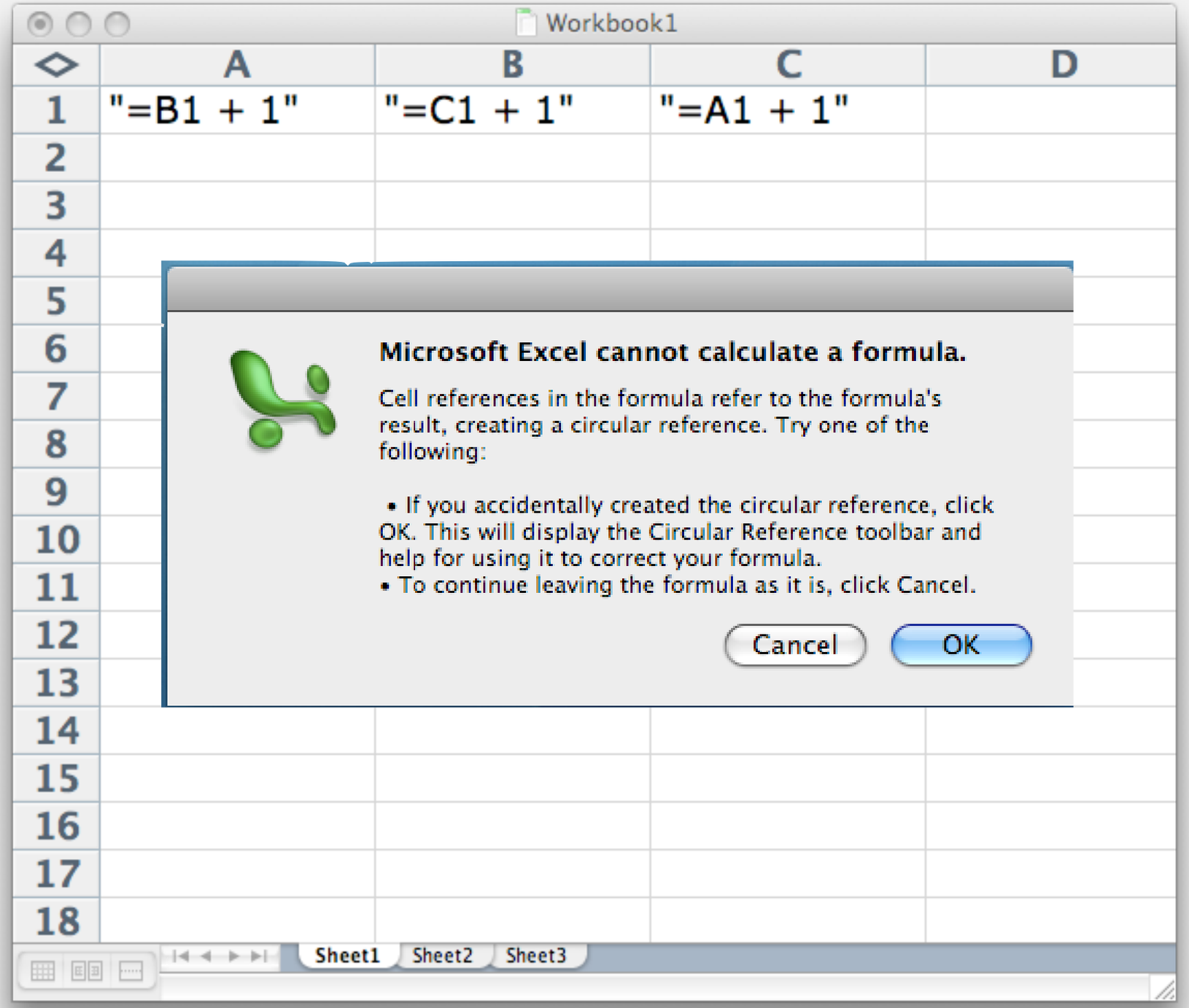
```
public class C extends A
{
   ...
}
```

```
% javac A.java
A.java:1: cyclic inheritance involving A
public class A extends B { }
       ^
1 error
```

copyrighted content – Do not share

Directed cycle detection application: spreadsheet recalculation

Microsoft Excel does cycle detection.



copyrighted content – Do not share

Directed cycle detection applications

- Causalities.
- Email loops.
- Compilation units.
- Class inheritance.
- Course prerequisites.
- Deadlocking detection.
- Precedence scheduling.
- Temporal dependencies.
- Pipeline of computing jobs.
- Check for symbolic link loop.
- Evaluate formula in spreadsheet.

copyrighted content – Do not share

Depth-first search orders

Observation. DFS visits each vertex exactly once. The order in which it does so can be important.

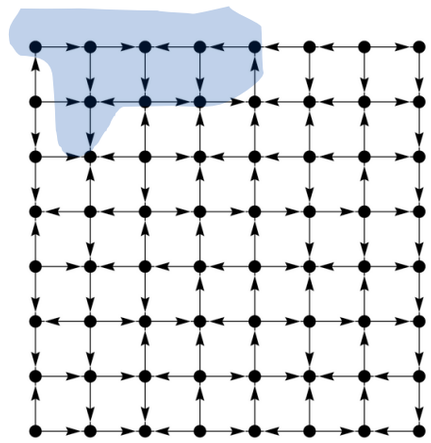
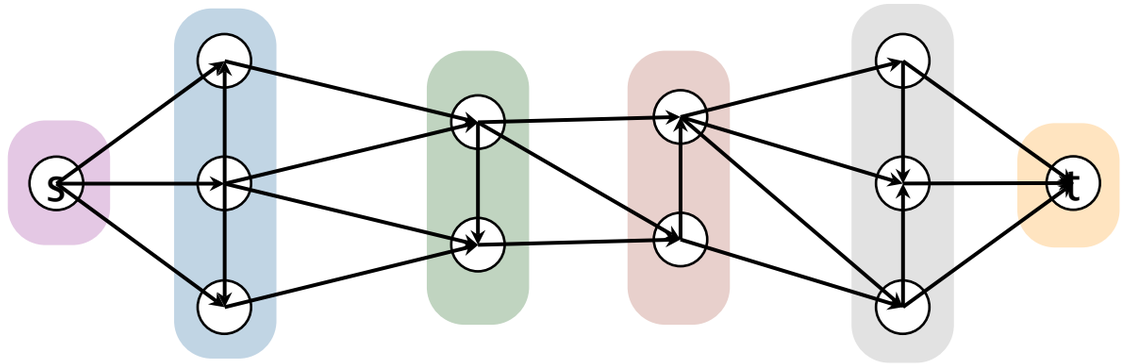
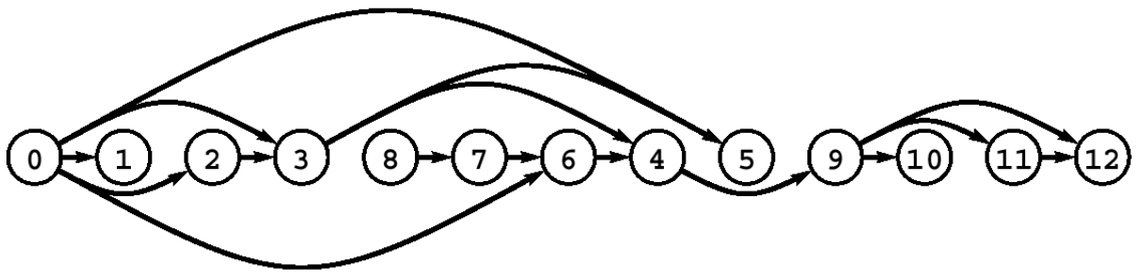
Orderings.

- Preorder: order in which dfs() is called.
- Postorder: order in which dfs() returns.
- Reverse postorder: reverse order in which dfs() returns.

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

copyrighted content – Do not share

Digraph-processing summary: algorithms of the day

single-source reachability in a digraph		DFS/BFS
shortest path in a digraph		BFS
topological sort in a DAG		DFS

copyrighted content – Do not share

DIRECTED GRAPHS

- *introduction*
- *digraph API*
- *depth-first search*
- *breadth-first search*
- *topological sort*
- *strong components* ← see videos



http://ds.cs.Rutgers.edu

copyrighted content – Do not share

Last updated on 2/11/18 1:29 PM

Some slides Adopted and modified from Sedgewick and Wayne