Algorithms & Data Structures

Graphs – Part II – Directed Graphs

Jacob Trier Frederiksen & Anders Kalhauge

Spring 2019

Program, Week 12

- Date for both the course evaluation, and the mid-term exam, has been agreed for the 23/4. We're nearing a fixation of that date, last chance to input on possible reasons to choose something apart from 23/4.
- Presentation and discussion of assignment #3 – 'Airport Queueing'. Teams 'NSqr' and 'NoGroup' are up as anchors.
- Review of exercises from last week: peer presentation and discussion.
- Walk-through material from chapter 4.2.
- A few pop quizzes along the way.

Peer review – Assignment #3, Airport Queueing

Team up are ‘NSquared’ and ‘NoGroup’

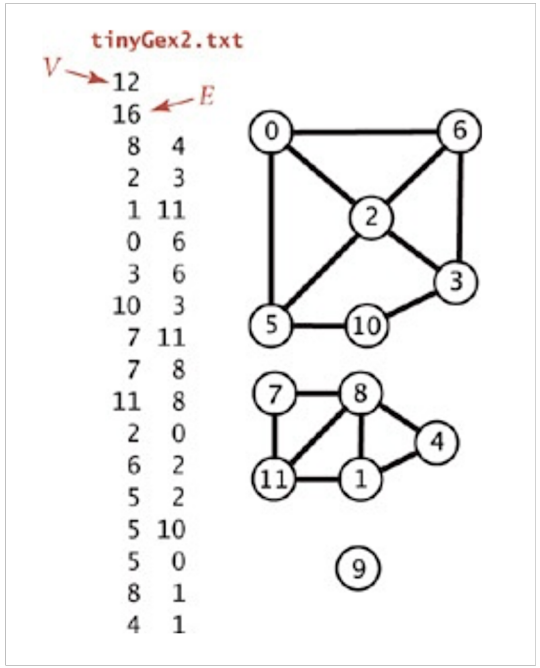
Exercise review – week 11

Did we do OK on the exercises? Or do wee need to try again...?

Exercise 1

- Draw the **array of adjacency lists** diagram for the built by the `Graph`'s input stream constructor for the graph, `tinyGex2.txt` to the right.

- Hint: you'll find all needed bits in Chapter 4.1, subsections *'Representation Alternatives'* and *'Adjacency-lists data structure'*.



Exercise 2

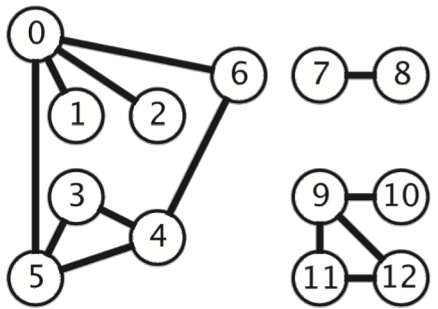
- Construct – rather synthesize – two un-directed graphs from scratch for two cases
  1. a sparse graph,
  2. a dense graph,

by any means that you choose. Java code would work, but so would Python, 'R' and many other scripting languages. Your choice.

- It should be saved to a text file, similar to the file `'tinyG.txt'` from the Algoritms data library.
- What to consider when making a synthetic graph?
- How to parametrize?

tinyG.txt

V → 13
E → 13
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3

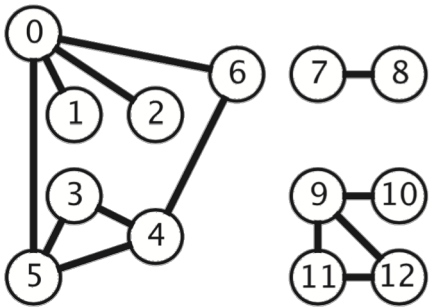


Exercise 3

- Produce the adjacency matrices for your two graphs.
- Save the matrix data (binary) to file and make an image representation of the two matrices respectively.
- For the two cases, do you have full connectivity, in a single connected component, or do you have more than one?
  - Can we even check that from an adjacency matrix? If so, howto?

tinyG.txt

V → 13
E ← 13
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3



Exercise 4

Out of time, use `mediumG.txt` from "Algorithms 4th Edition".
https://algs4.cs.princeton.edu/41graph/mediumG.txt

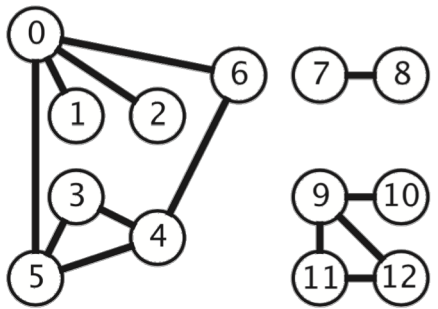
1. Check whether your graph is connected.
   Hint: `TestSearch`
2. Find DFS paths for your synthetic graphs
   Hint: `DepthFirstPaths`
3. Find BFS paths for your synthetic graphs
   Hint: `...`
4. Are your results for DFS and BFS different?
5. Find connected components for your graphs
   Hint: `...`
6. Check if your graphs have cycles
7. Check whether your graphs are bi-partite.

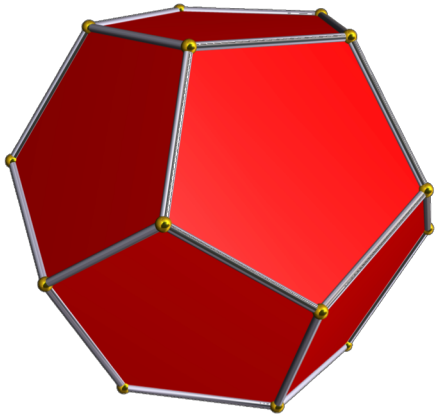
Exercise 5

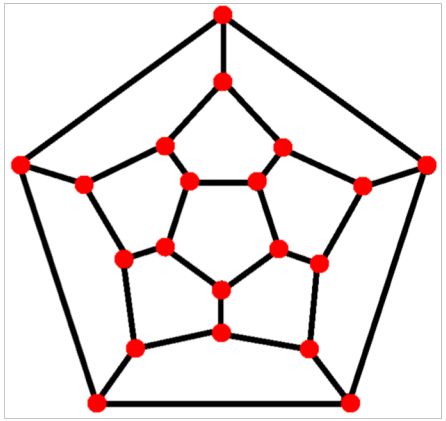
- For both of your synthesized graphs (sparse and dense) – which are a your previously generated text files (right?), find
  - by DFS whether 10 randomly chosen points are connected to an other chosen single point.
  - the shortest path between the same one chosen point and the randomly selected 10 other points, using BFS.

tinyG.txt

V → 13
13 ← *E*
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3







Q: Did you have a chance to visit arXiv.org?

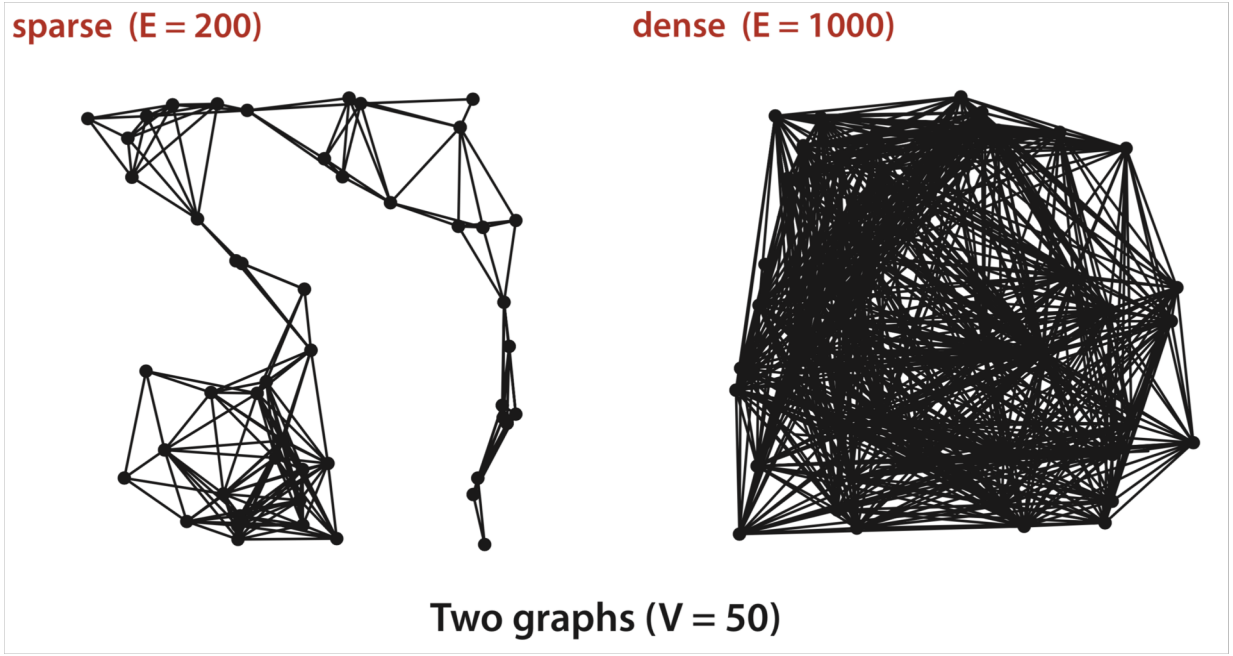
https://xxx.lanl.gov/ check it out, in particular Computer Science / Algs & data struct.s.

There is still lots of research going on out there!

Today's Quiz – the white board "peleton".

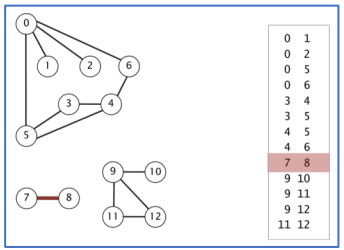
1. What is a sparse graph, and what is a dense graph?
2. What are three basic data structures for graph representation?
3. What is a bi-partite graph?
4. What is a weighted graph?
5. What is the complexity for array of adjacency lists for graph of V vertices and E edges:
    1. memory (space)?
    2. adding an edge?
    3. query if edge from vertex v to vertex w?
    4. iterste over vertices pointing from v?

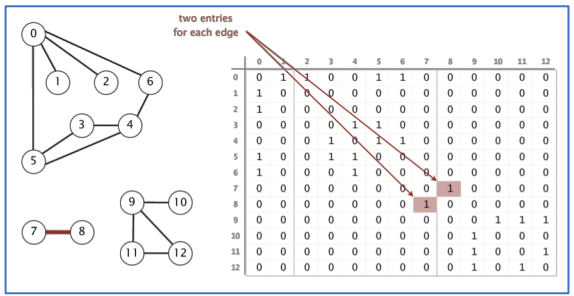
Quiz resolution: dense *vs.* sparse

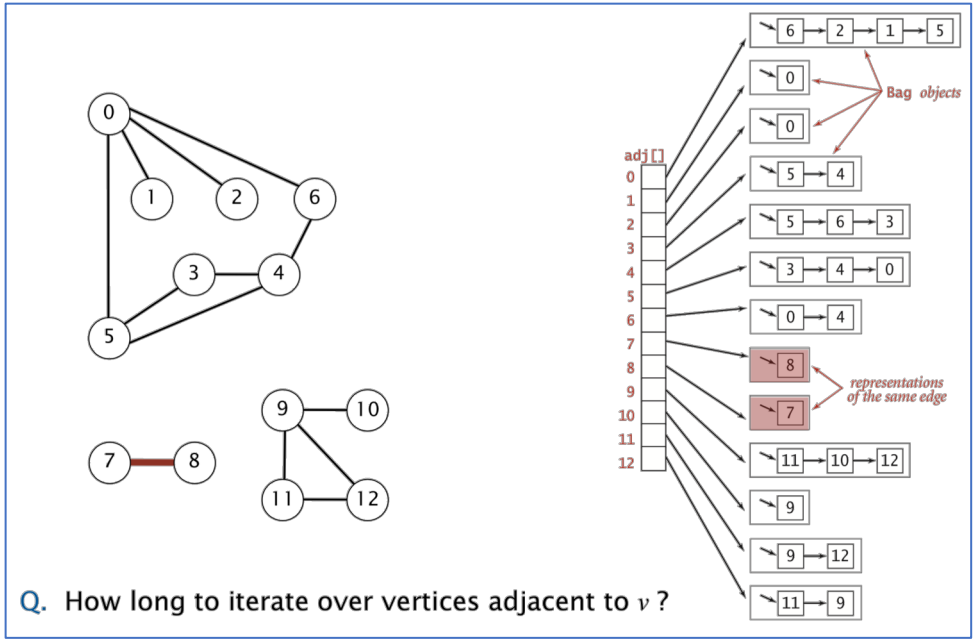


Quiz resolution: representations, data structures.

- Array of edges, `E`? Nope.
- Adjacency matrix, `VxV`? Nope.
- Array of adjacency lists? Yep.







Graphs

- Undirected
- Directed
- Bi-partite
- Weighted and unweighted

White board exercise

(4 people)

Draw:

1. **Undirected cyclic graph**, with 3 connected components.
2. **Directed graph** with 1 acyclic and 2 cyclic connected components.
3. A **bi-partite graph**, single connected component
4. A **weighted graph** example.

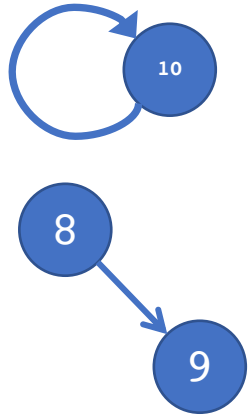
"Every undirected graph is a digraph (with edges in both directions)."

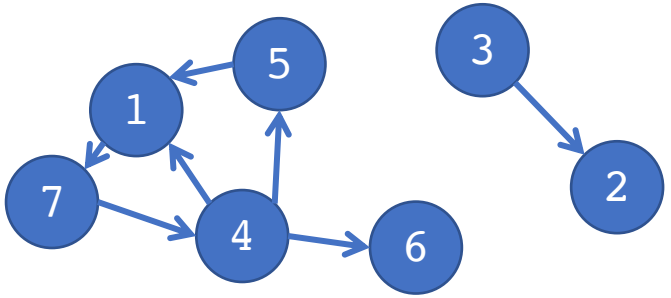
Digraphs

Week 12, extension of discussion on un-directed graphs

Digraphs – Very simple struct.s.

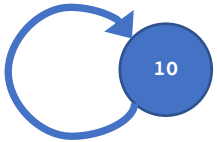
- Set of vertices connected by edges.
- Edges have direction, “from v, to w”.
- `G = (V,E)`

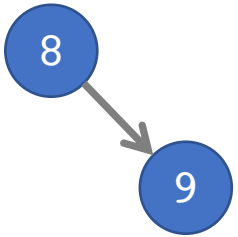


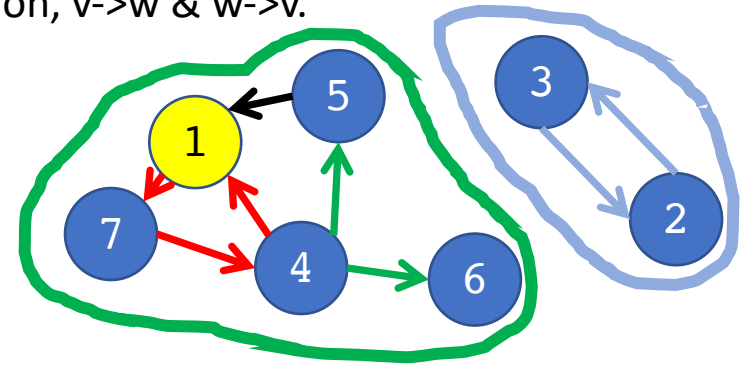


Digraphs – Definitions

- **Path**: sequence of vertices connected by directed edges.
- **Cycle**: path whose first and last vertices are the same.
- **Connectivity**:
  - one vertex is connected to another if a directed path exists between them.
  - a graph is a set of connected components.
- **Strong connectivity:**
  - reflexive, symmetric, transitive directed connection, v->w & w->v.
- **Outdegree/Indegree:**
  - a vertex with outdegree = 1, indegree = 2.
- **Directed Acyclic Graph (DAG)**
  - has no cycles.





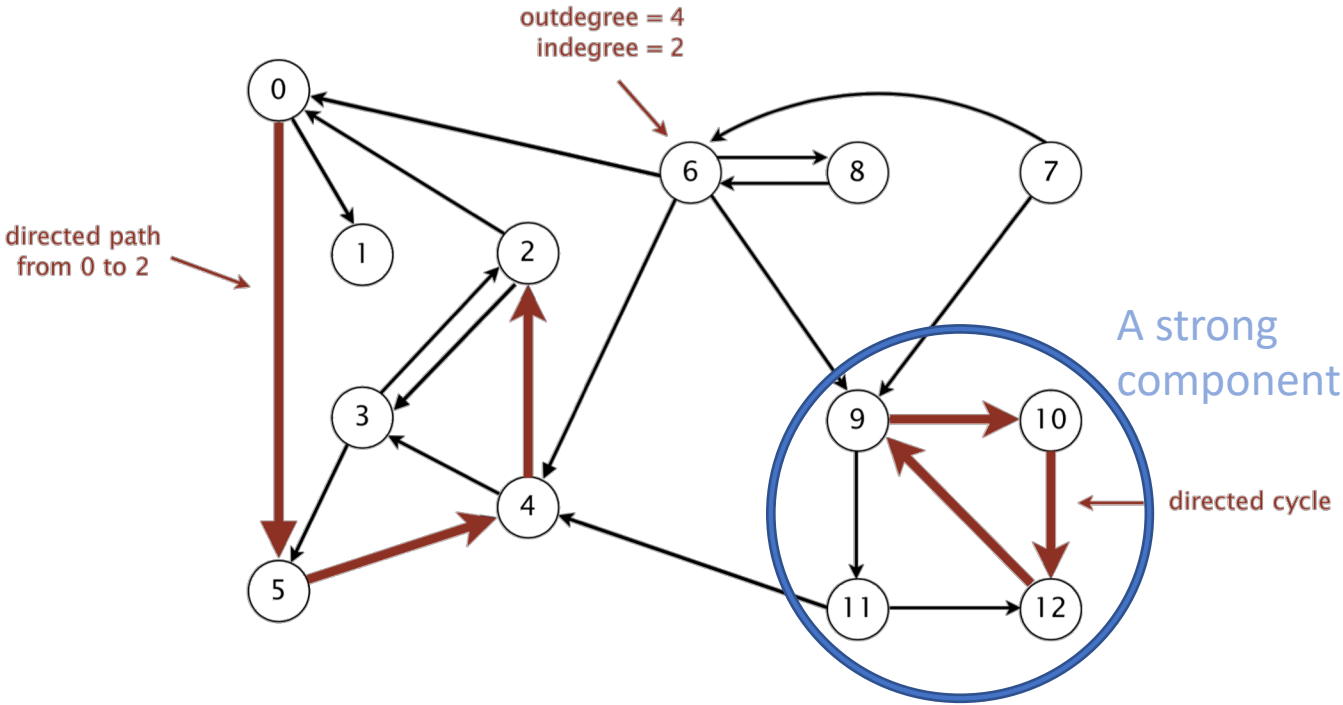


Pop 1 (NB: directed graphs, week 12):

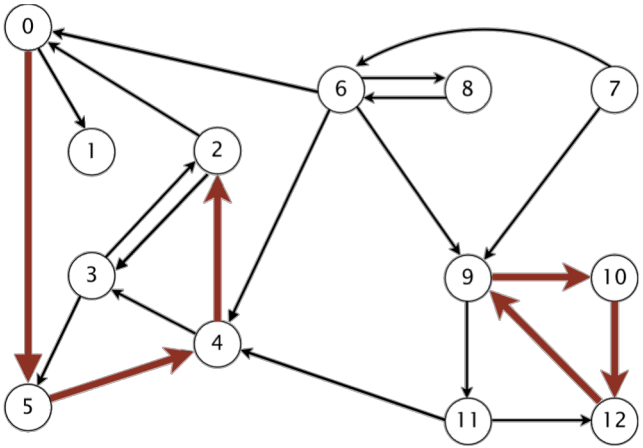
- How many different digraphs are there on V vertices?
  (allow for self-loops but do not allow parallel edges)

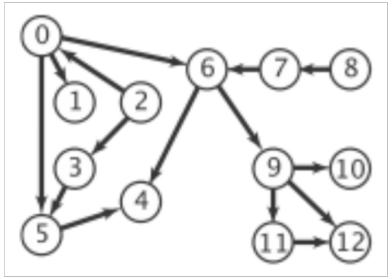
  - ❑ V
  - ❑ V^2
  - ❑ 2^V
  - ❑ $2^{(V^2)}$
  - ❑ $2^{(2^V)}$

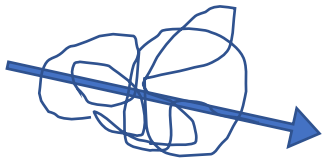
Digraphs – strongly connected components

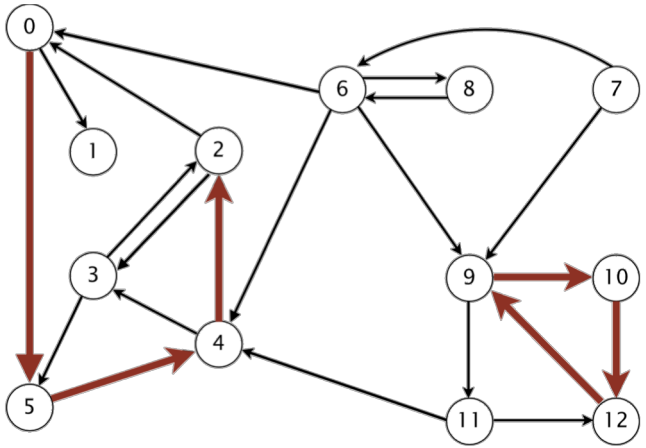


Digraphs, white board exercise, two teams: transform into a ‘DAG’.









Applications of digraphs

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

Graphs – representations, data structures.

- Array of edges, `E`? Nope.
- Adjacency matrix, `VxV`? Nope.
- Array of adjacency lists? Yep.

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	$degree(v)$	$degree(v)$

Typical digraph theory questions

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 all 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 ?*

Digraph API *vs.* un-directed graph API – a comparison

Un-directed graph

```
public class Graph
{
   private final int V;
   private final 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];  }
}
```

Digraph

```
public class Digraph
{
   private final int V;
   private final 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

create empty graph with V vertices

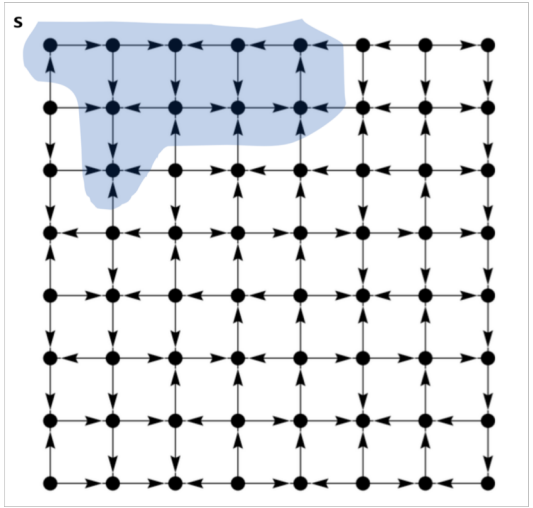
add edge v–w

iterator for vertices adjacent to v

Digraph search – DFS

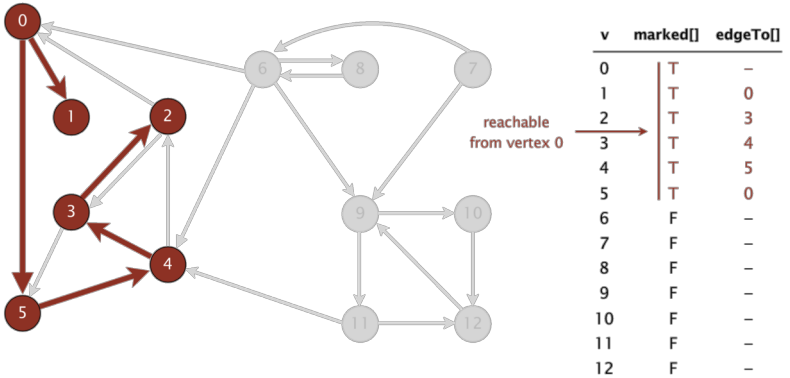
Find all vertices reachable from ‘s’

along a directed paths



DFS on digraphs.

Nb: identical to DFS for un-directed graphs



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	–

Pop 2 (NB: directed graphs, week 12)

During a DFS in a digraph G, `dfs(v)` is called after `dfs(w)` is called but before `dfs(w)` returns. What *must* be true about the graph G?

- There exists a directed path `v -> w`.
- There exists a directed path `w -> v`.
- There does not exist a directed path `v -> w`.
- There exists a directed cycle containing `v -> w`.

Pop 3 (NB: directed graphs, week 12)

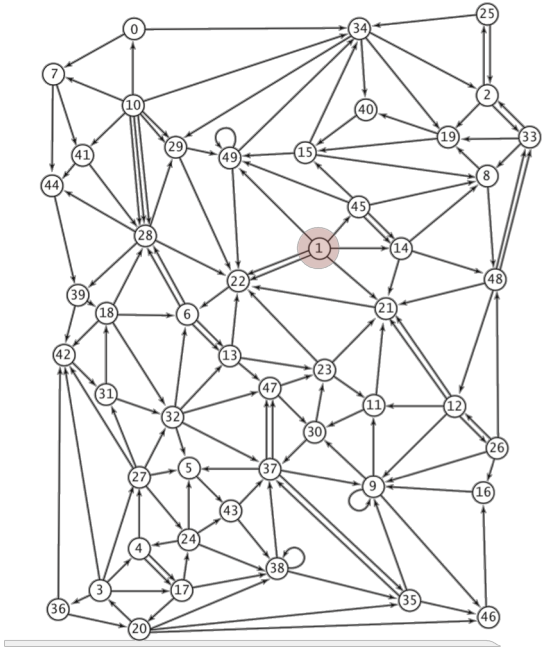
A DFS i done on a DAG which contains edge `v -> w`.

Which of the following is impossible at the time of calling `dfs(v)`?

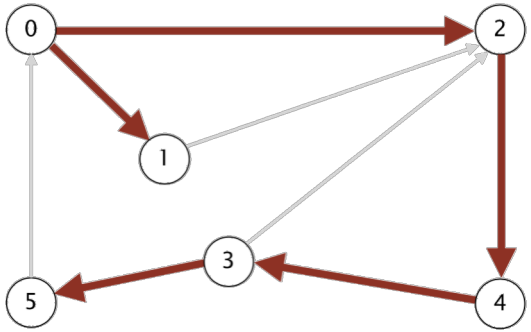
- `dfs(w)` has not yet been called.
- `dfs(w)` will be the next recursive call after `dfs(v)`.
- `dfs(w)` has already been called but not yet returned.
- `dfs(w)` has already been called and returned.

Digraph search – BFS

Crawling the WWW – an example where BFS is viable.



BFS finds shortest distance from vertex ‘s’ to all other vertices ‘v’ (if any exists)



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

Topological Sort on DAGs

General problem of solving order of precedence for any system.

Must have a Directed Acyclic Graph (DAG), otherwise there is ambivalence.

Redraw all edges so they point in same direction.

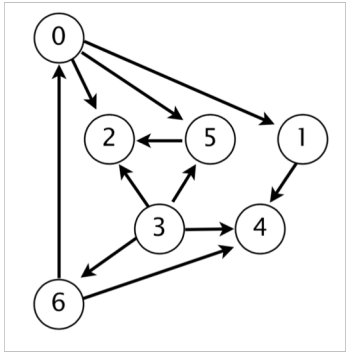
Theorem: a reverse DFS postorder stack of a DAG is a topological order.

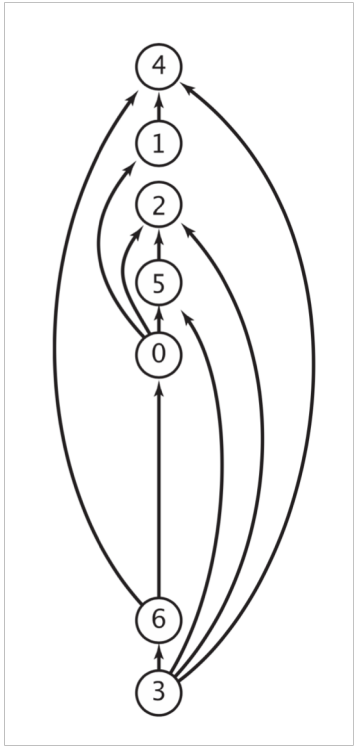
EXERCISE

- **show by demo'ing.....on the white board.**
- **does the order of visits matter?**

Example of use of Top.Sort. on DAGs is order of precedence scheduling.

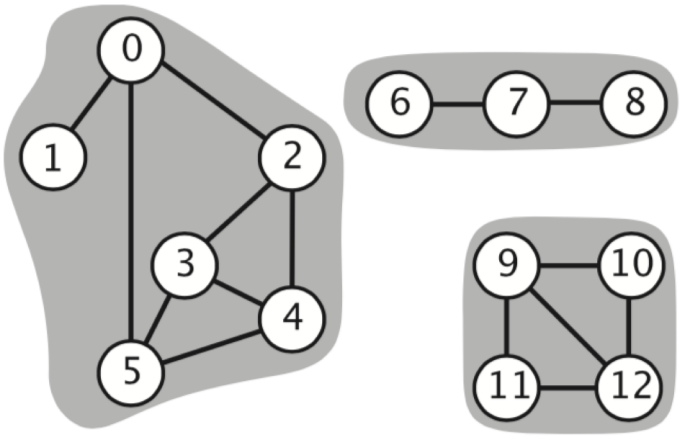
0. Algorithms
1. Complexity Theory
2. Artificial Intelligence
3. Intro to CS
4. Cryptography
5. Scientific Computing
6. Advanced Programming





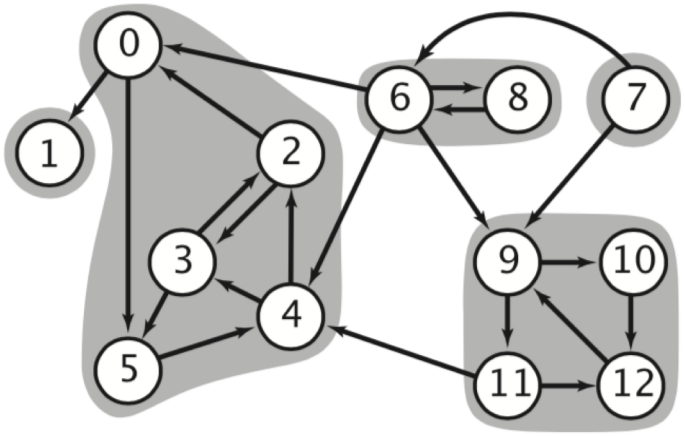
Strong connectivity in digraphs

Un-directed graphs



3 connected components

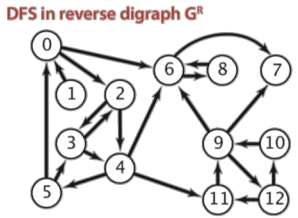
Digraphs



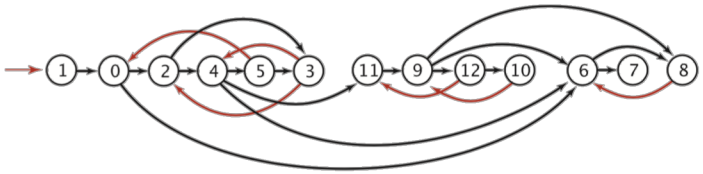
5 strongly-connected components

Kosaraju-Sharir algorithm: compute strong components using twice DFS.

Step 1

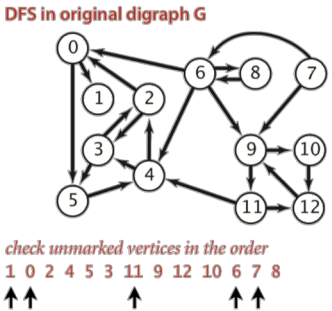


check unmarked vertices in the order
0 1 2 3 4 5 6 7 8 9 10 11 12

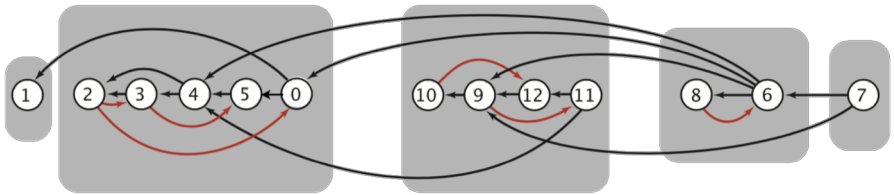


reverse postorder for use in second `dfs()`
1 0 2 4 5 3 11 9 12 10 6 7 8

Step 2



check unmarked vertices in the order
1 0 2 4 5 3 11 9 12 10 6 7 8



Acknowledgement

- Some material has been taken from the Princeton course ‘Algorithms 4th Edition’, by Sedgewick & Wayne.