Algorithms & Data Structures

Graphs – Part I

Jacob Trier Frederiksen & Anders Kalhauge

Spring 2019

What's your favorite Mid-term Quiz?

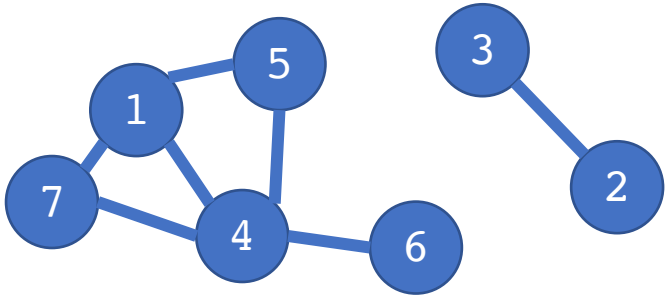
- What would you like to test yourself on for the mid-term quiz?
  - …
  - …
  - ..
  - ….

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

1. Can you name three generic queue topologies?
2. What is a Complete Binary Tree?
3. What is a max-oriented heap?
4. What are the basic steps in a heap sort (given random data)?
5. What is the complexity of heap sorting (worst case)?

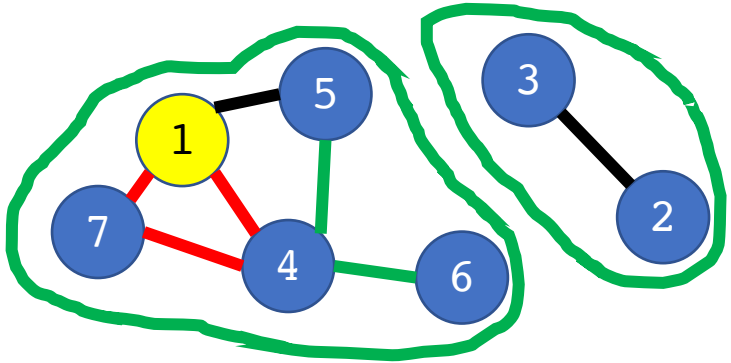
Graphs – Very simple struct.s.

- Set of vertices connected by edges.
- Edges connected in any possible way
- `G = (V,E)`



Graphs – Definitions

- **Path**: sequence of vertices connected by edges.
- **Cycle**: path whose first and last vertices are the same.
- **Connectivity**:
  - two vertices are connected if a path exists between them.
  - a graph is a set of connected components.
- Degree: a vertex with – say – three connections has degree 3.
- Acyclic (not shown): has no cycles.
- Directed (not shown): has directionality.



Copenhagen S-Trains

This is a graph.

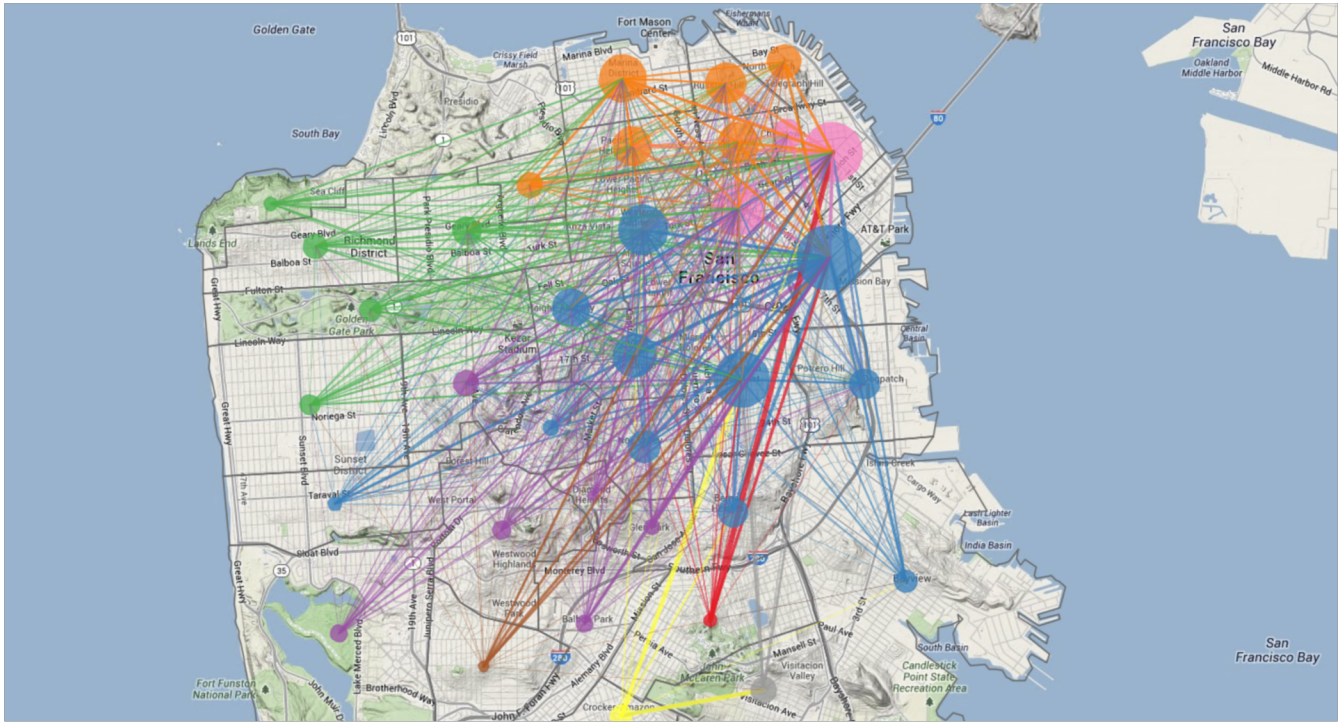
Q: what is the degree of Frederikssund?

Q: is Hundige connected with Egedal?

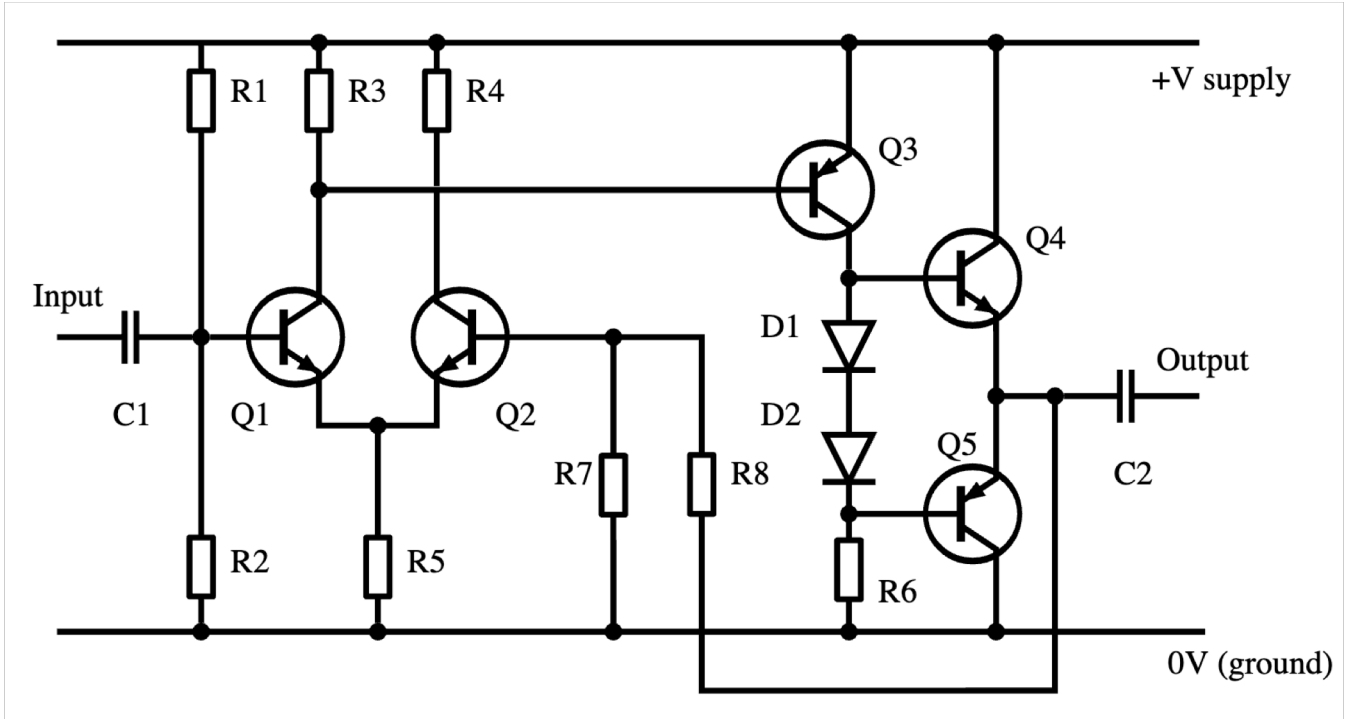
Q: are there any cycles on the S-Train net?

Applications of graphs – Facebook

Uber traffic

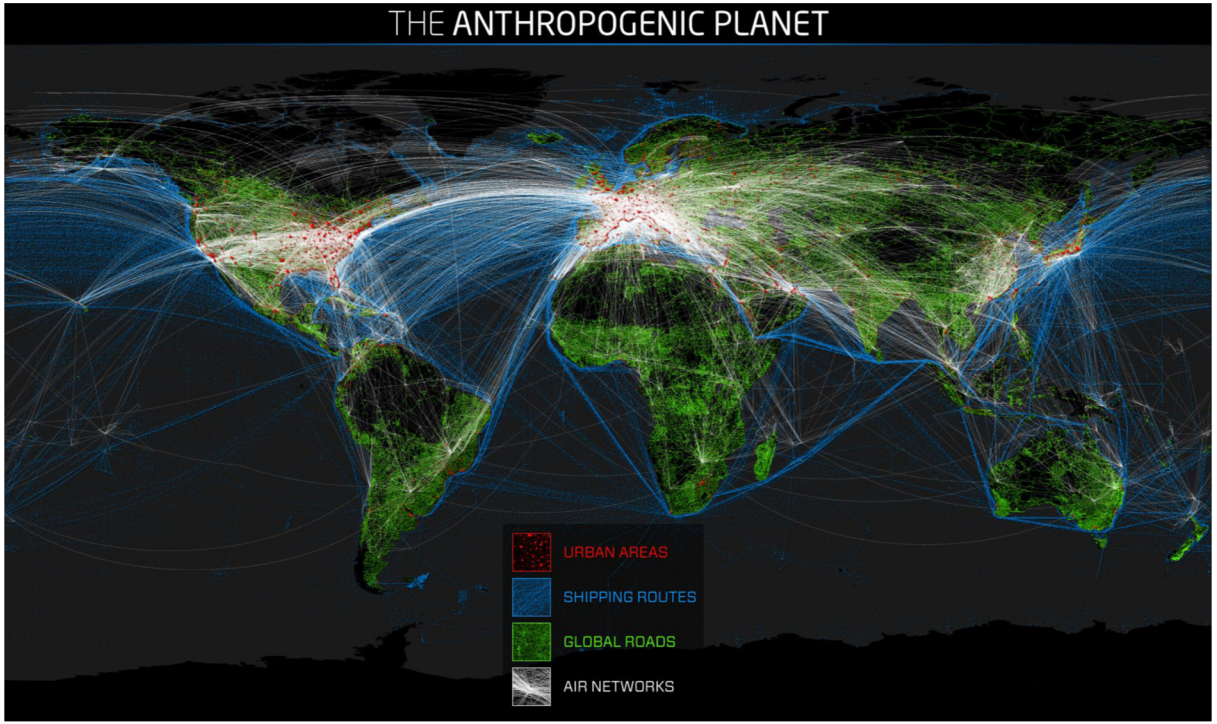


Electronic circuits



New York Street Map

The Whole Smack

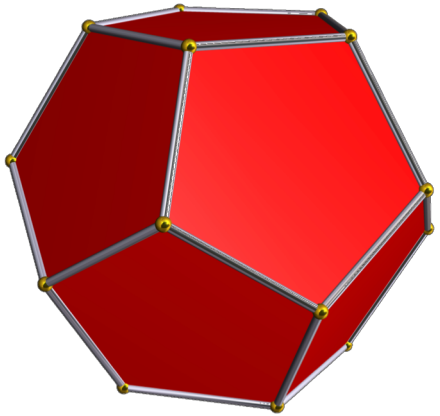


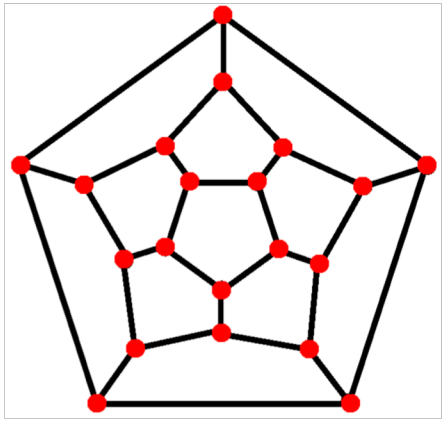
Applications of Graphs

- **Biology:** vertuces could represent regions where certain species exist and the edges represent migration path or movement between the regions, i.e. **neural networks**.

- **Maps**: Various locations are represented as vertices and the roads are represented as edges. Graph theory is used to **minimize path distance** between nodes.

- **E-commerce**: recommender engines use graph theory to **recommend** items of similar type to (previous) user choices.





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!

Typical graph theory questions

problem	description
s–t path	*Is there a path between s and t ?*
shortest s–t path	*What is the shortest path between s and t ?*
cycle	*Is there a cycle in the graph ?*
Euler cycle	*Is there a cycle that uses each edge exactly once ?*
Hamilton cycle	*Is there a cycle that uses each vertex exactly once ?*
connectivity	*Is there a way to connect all of the vertices ?*
biconnectivity	*Is there a vertex whose removal disconnects the graph ?*
planarity	*Can the graph be drawn in the plane with no crossing edges ?*
graph isomorphism	*Do two adjacency lists represent the same graph ?*

Graphs

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

White board exercise

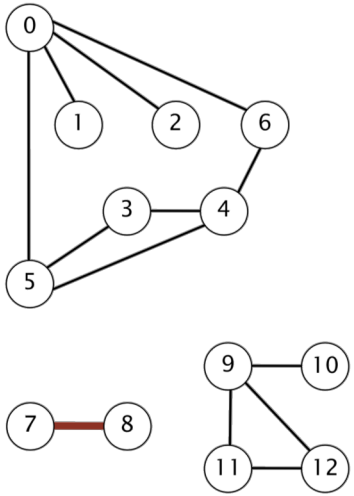
(4 people)

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.

Graphs – representations, data structures.

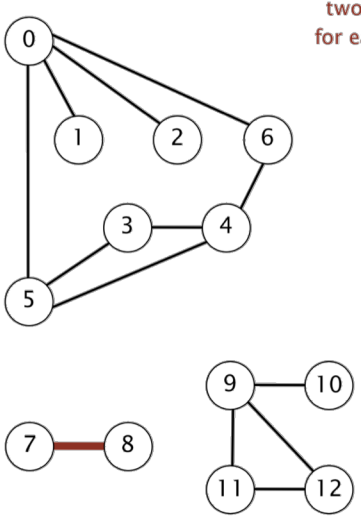
- Array of edges, `E`? Nope.



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

Graphs – representations, data structures.

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

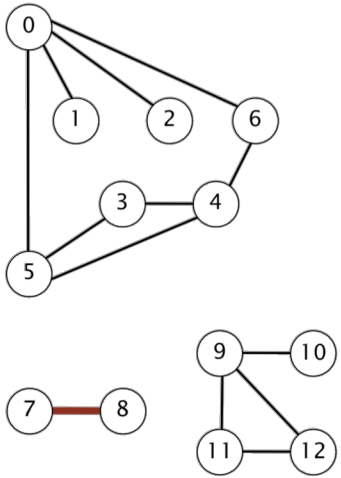


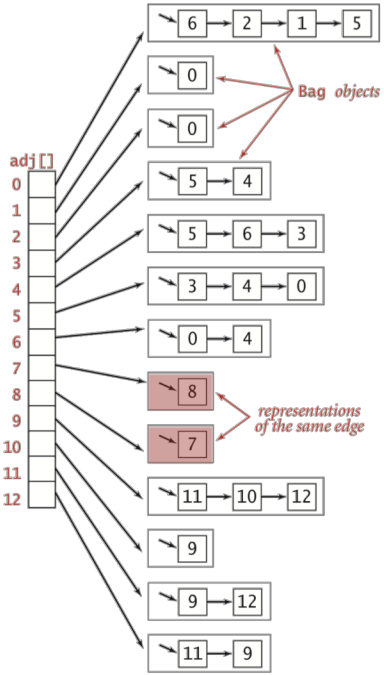
two entries for each edge

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

Graphs – representations, data structures.

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





Graphs – representations, data structures.

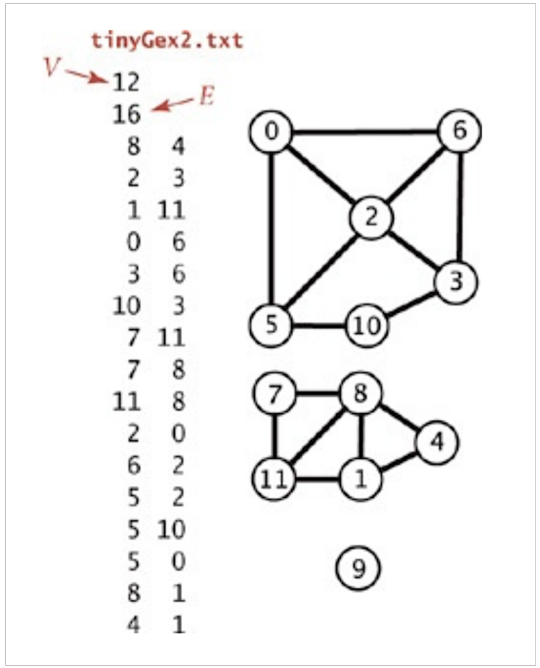
- Array of edges, `E`? Nope.
- Adjacency matrix, `VxV`? Nope.
- Array of adjacency lists? Yep.

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

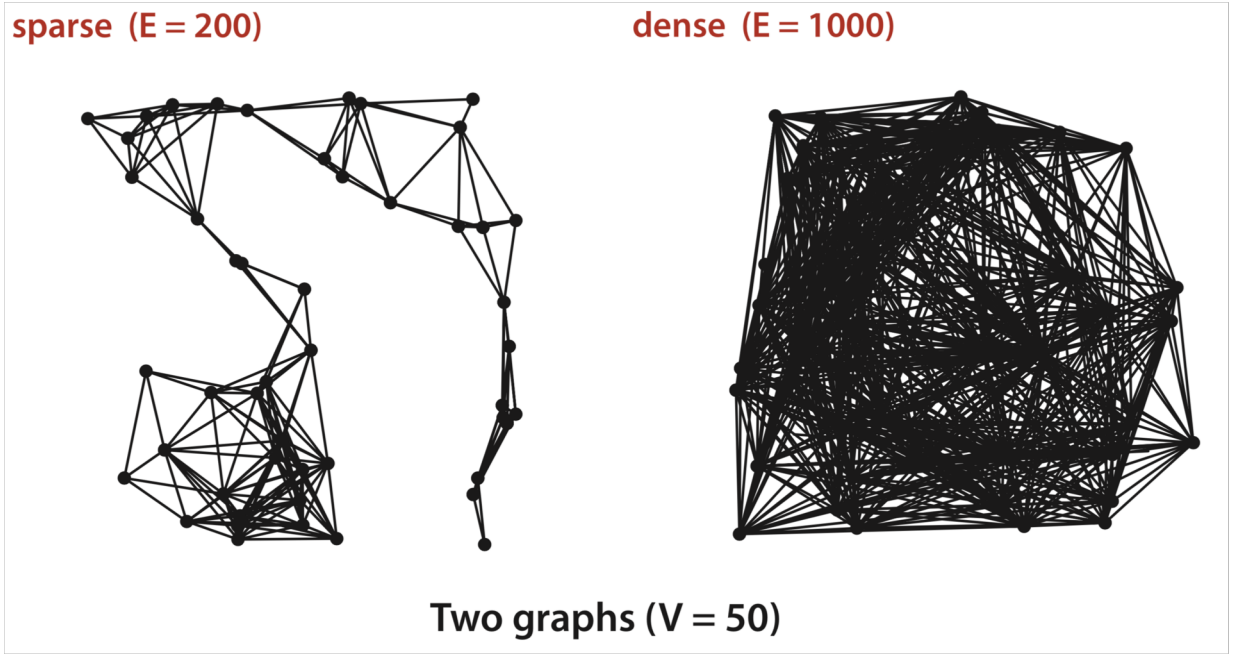
Exercise 1

- 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'*.



Graphs – dense *vs.* sparse



Exercise 2

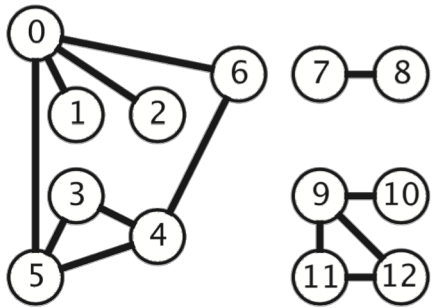
- Construct – rather synthesize – two un-directed graphs from scratch for two cases
  1. a sparse graph,
  2. a dense graph,

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
    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
```

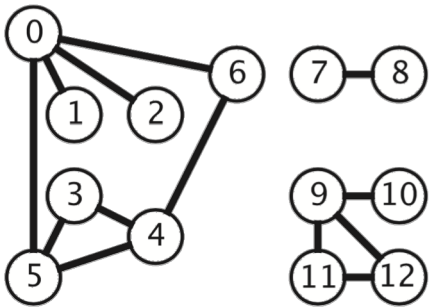


Exercise 3

- 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



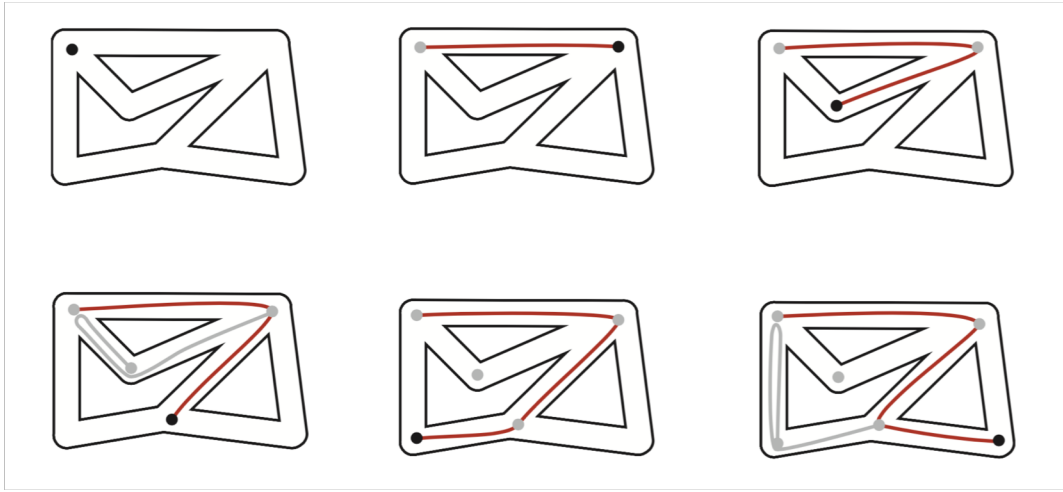
Depth-first Search, DFS

Applications:

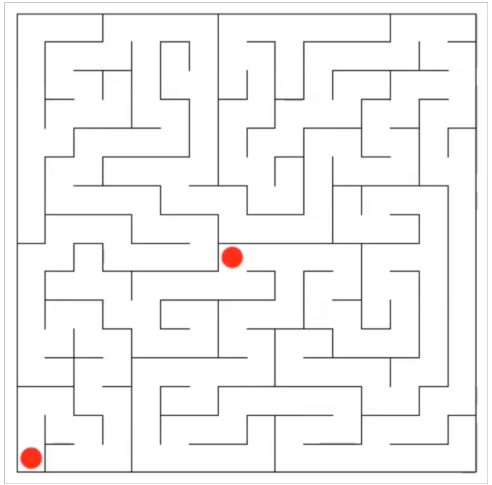
- Finding connected components.
- Finding strongly connected components.
- Topological sorting.
- Finding the bridges of a graph.
- Planarity testing.
- Solve mazes.
- Maze generation, use a randomized depth-first search.
- Finding biconnectivity in graphs.

Tremaux Maze – depth-first search.

- Or the Theseus ... or just common sense.



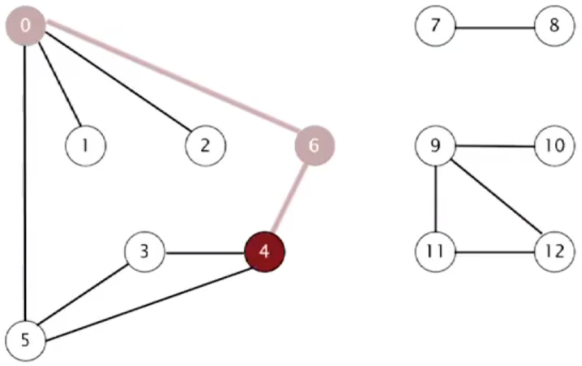
How to find your way?



Depth-first Search, DFS

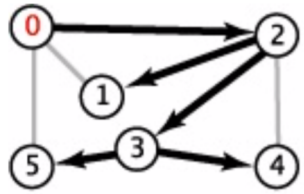
- `pathTo[v]:` DFS uses a **stack** to store the `edgeTo[v]` values.

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



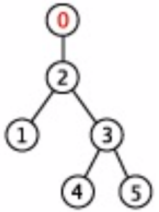
v	marked[]	edgeTo[v]
0	T	–
1	F	–
2	F	–
3	F	–
4	T	6
5	F	–
6	T	0
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

visit 4: check 5, check 6 and check 3



edgeTo[]

0	
1	2
2	0
3	2
4	3
5	3



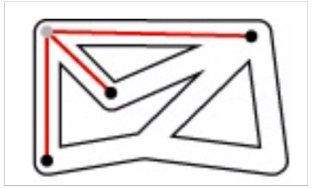
x	path
5	5
3	3 5
2	2 3 5
0	0 2 3 5

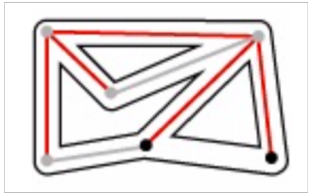
Breadth-first Search, BFS

Applications include:

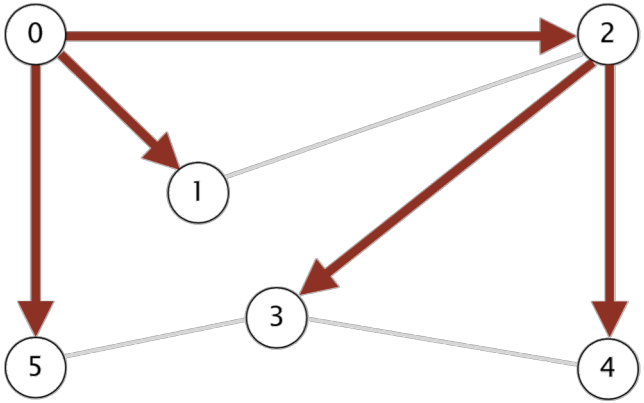
- Finding the shortest path between two nodes u and v, with path length measured by number of edges (an advantage over DFS)
- Testing bipartiteness of a graph.
- (Reverse) Cuthill–McKee mesh numbering
- Ford–Fulkerson method for computing the maximum flow in a flow network
- Serialization/Deserialization of a binary tree vs serialization in sorted order, allows the tree to be re-constructed in an efficient manner.
- Copying garbage collection, Cheney's algorithm

Breadth-first Search, BFS





Uses a **queue** to keep track of vertices visited but whose adjacent vertices not yet visited.



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

Exercise 4

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

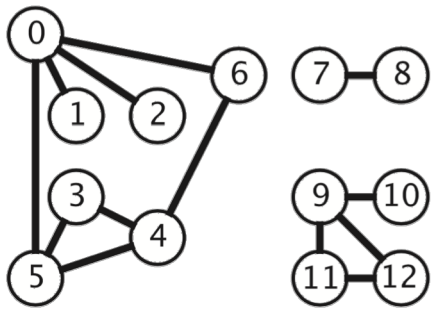
1. Check whether your graph is connected.
   Hint: `TestSearch`
2. Find DFS paths for your synthetic graphs
   Hint: `DepthFirstPaths`
3. Find BFS paths for your synthetic graphs
   Hint: `...`
4. Are your results for DFS and BFS different?
5. Find connected components for your graphs
   Hint: `...`
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. Record running time for the 10 cases.
  - the shortest path between the same one chosen point and the randomly selected 10 other points, using BFS. Record running time for the 10 cases.

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



A note on **Symbol Graphs** and symbol tables

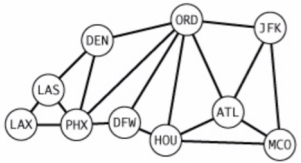
- Not so different....

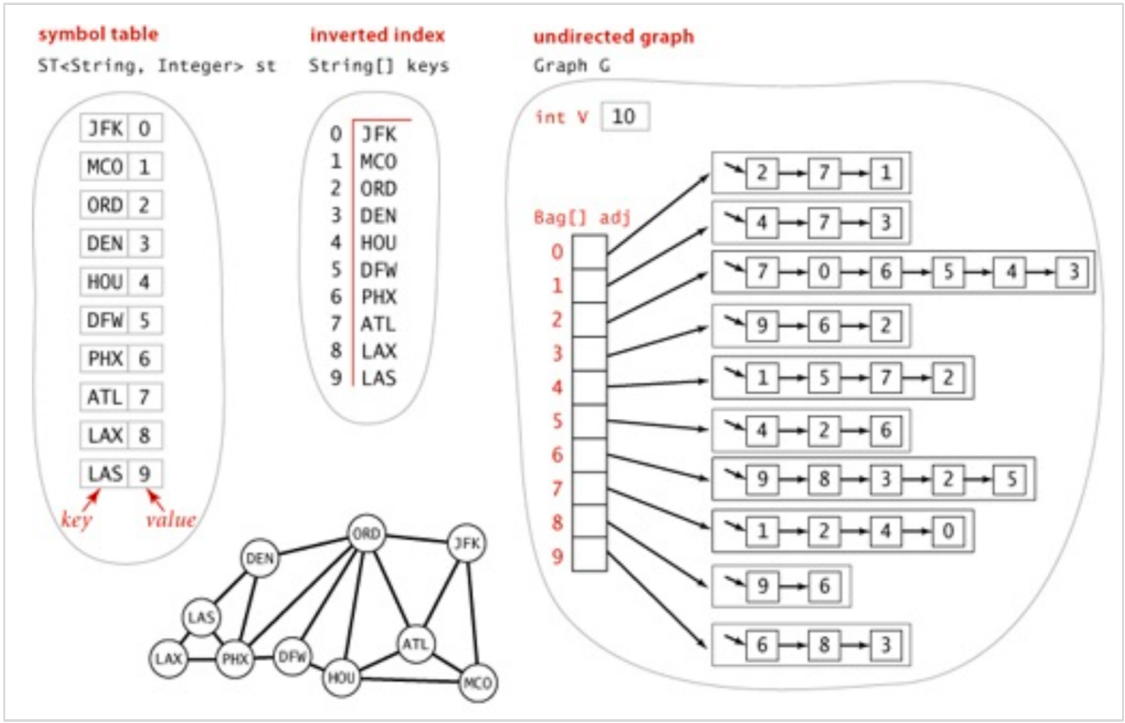
routes.txt

```
JFK MCO
ORD DEN
ORD HOU
DFW PHX
JFK ATL
ORD DFW
ORD PHX
ATL HOU
DEN PHX
PHX LAX
JFK ORD
DEN LAS
DFW HOU
ORD ATL
LAS LAX
ATL MCO
HOU MCO
LAS PHX
```

V and E not explicitly specified

" " delimiter





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

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.

Acknowledgement

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