COMP20230: Data Structures & Algorithms **2018-19 Semester 2**

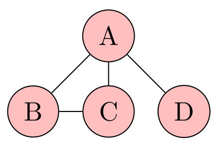
Lecture 16: Graphs

Lecturer: Dr. Andrew Hines *Scribes: Philip McGrath, Daniel Raftery*

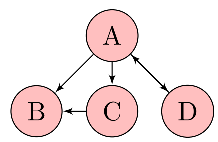
Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

16.1 Outline

A graph is a non-linear abstract data type (ADT) which consists of a finite set of vertices (nodes) joined via edges, similar to a tree. However, unlike in a tree, cycles and non-connected components are permitted in a graph. Graphs do not have a definite beginning or end, and can be either directed or undirected.



undirected graph



directed graph

An example of a graph structure is a list of websites produced by a search engine. After the search is executed, a tree starting from a root parents website leading to children sites is produced. These children sites may possibly link back to their parent size, thus producing a graph structure.

16.2 Terminology

- Each element within a graph is called a **node**.
- Each node is connected to another node bide an **edge**.
- A **neighbour** to a node is another node such that there exists an edge between them.
- A **cycle** may consist of different components and occurs when
  1. A node is connected to itself.
  2. A node is connected to itself via edges between other nodes.

16.3 Graph ADT

A graph `G` with vertices `x, y` has ADT operations defined by

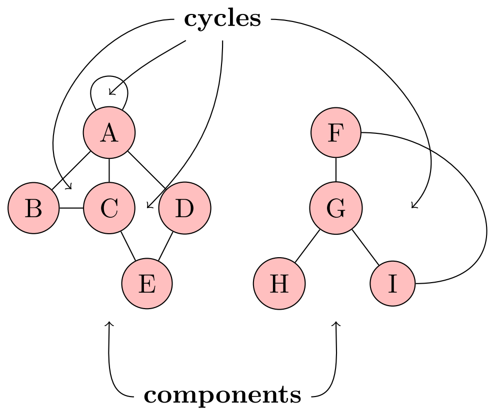
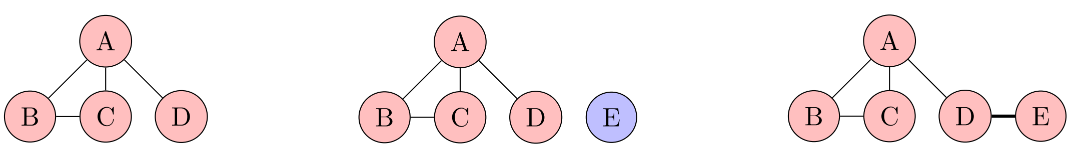


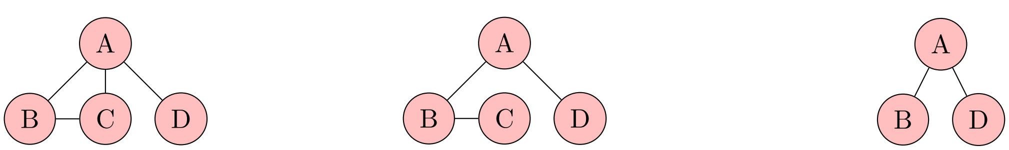
Figure 16.1: Illustration of a graph with cycles and components labelled

- `adjacent(G, x, y)`
  Tests if there exists an edge from vertex `x` to vertex `y`.
- `neighbours(G, x)`
  Lists all vertices `y` such that there exists an edge from vertex `x` to vertex `y`.
- `add_vertex(G, x)`
  Adds the vertex `x` to `G` if it does not already exist.
- `remove_vertex(G, x)`
  Removes the vertex `x` from `G` if it exists.
- `add_edge(G, x, y)`
  Adds an edge from vertex `x` to vertex `y` if one does not already exist.
- `remove_edge(G, x, y)`
  Removes the edge from vertex `x` to vertex `y` if it exists.



Original Graph `add_vertex(G, E)` `add_edge(G, D, E)`

Figure 16.2: Demonstration of `add_vertex()` and `add_edge()`



Original Graph `remove_edge(G, A, C)` `remove_vertex(G, C)`

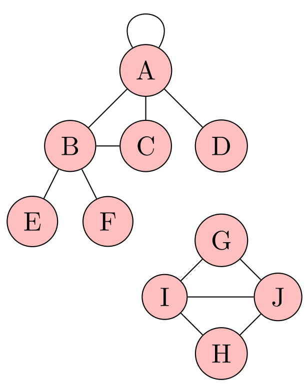
Figure 16.3: Demonstration of `remove_edge()` and `remove_vertex()`

16.4 Graph Representation

A graph G can be computationally represented as a function of the vertices V and edges E, as $G(V, E)$. There are two distinct methods to represent the graph depending on the data, using an *adjacency-list* or an *adjacency-matrix*.

16.4.1 Adjacency-list

An adjacency-list is more suitable for data that produces a sparse graph, where $|E| \ll |V|^2$. As an example, consider the following graph.



(a) Graph

A	A, B, C, D
B	A, C, E, F
C	A, B,
D	A
E	B
F	B
G	I, J
H	I, J
I	G, H, J
J	G, H, I

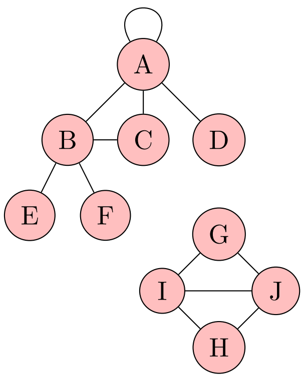
(b) Adjacency List

Figure 16.4: Example of a graph and the corresponding adjacency-list

An adjacency-list can be described as an array of lists. In this array, an entry `array[i]` is a list of all vertices adjacent to the vertex `i`. The beneficial aspect of using a list is that addition of vertices is easy and adjacency-lists tend to consume less space [GG].

16.4.2 Adjacency-matrix

In cases where the data would instead produce a dense graph, an adjacency-matrix is a better alternative to an adjacency-list. This is when $|E| \approx |V|^2$, when the number of edges is approximately equal to the number of vertices squared. An adjacency-matrix is a 2-dimensional array, and if the graph is undirected is it square symmetric, having equal number of rows as columns. In this representation, an edge exists between vertices `i` and`j` if in the adjacency-matrix (denoted `adm`), `adm[i][j] = 1` [GG]. The benefit of using an adjacency-matrix is that removing edges and performing queries to determine the existence of an edge between two vertices is much easier to implement than with an adjacency-list. However, an adjacency-list is faster when adding vertices and also consumes less space than an adjacency-matrix. [GG]



(a) Graph

	A	B	C	D	E	F	G	H	I	J
A	1	1	1	1	0	0	0	0	0	0
B	1	0	1	0	1	1	0	0	0	0
C	1	1	0	0	0	0	0	0	0	0
D	1	0	0	0	0	0	0	0	0	0
E	0	1	0	0	0	0	0	0	0	0
F	0	1	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	1	1
H	0	0	0	0	0	0	0	0	1	1
I	0	0	0	0	0	0	1	1	0	1
J	0	0	0	0	0	0	1	1	1	0

(b) Adjacency Matrix

Figure 16.5: Example of a graph and the corresponding adjacency-matrix

16.5 Traversing a Graph/Searching

16.5.1 Depth First Search

Algorithm 1: DFS (non-recursive)

<u>DFS</u> (g, n);
Input : A Graph g and node n
Output: The procedure explores every node station from n
to_visit ← empty stack
add n to to_visit
while to_visit is not empty do
    current ← pop to_visit
    push all neighbours of current (not in visited) to to_visit
    do something
endfor

Algorithm 2: DFS (recursive)

```
DFS (g, n);
Input  : a Graph g and node n
Output: the function explores every node from n
flag n as visited
for each neighbour n_c of n which is not visited do
    dfs(n_c)
endfor
```

16.5.2 Breadth First Search

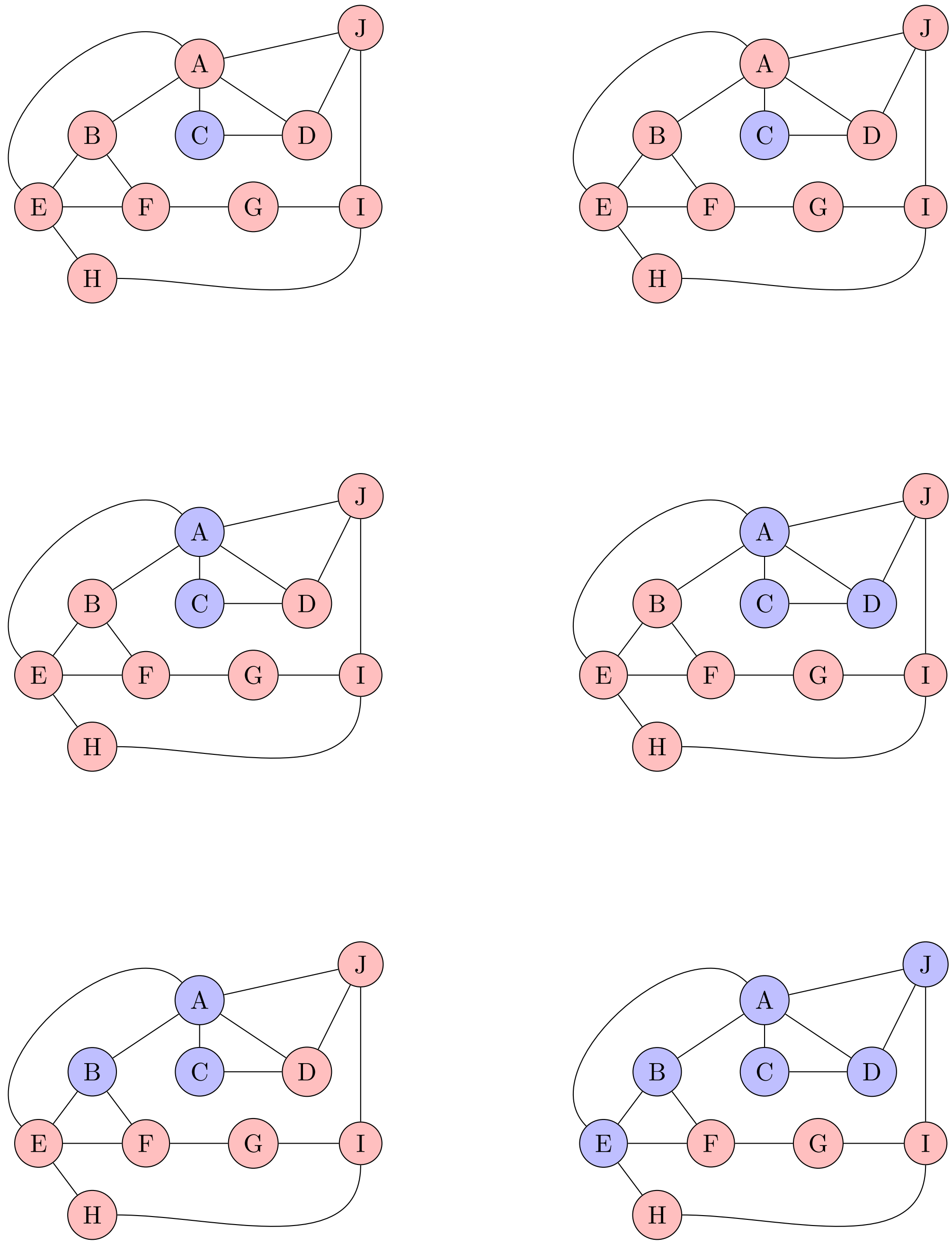
Algorithm 3: BFS (recursive)

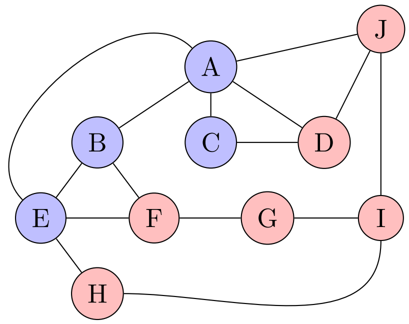
```
BFS(g, q);
Input  : a Graph g and a queue q (originally having a starting node)
Output: the functions explores every node from n
if queue is empty then # base case
    do something
else
    current ← dequeue q
    flag current as visited
    for each neighbour n_c of current not visited do
        enqueue n_c
    endfor
    do something
    bfs(q)
endif
```

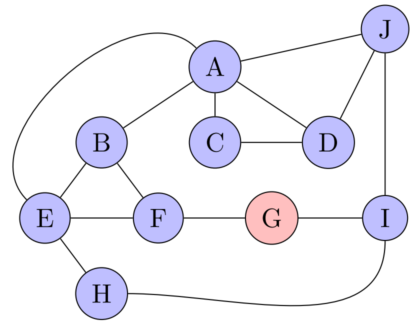
```
BFS(g, n);
Input  : a Graph g and a node n
Output: this procedure explores every node of g from n
to_visit is a queue
enqueue n
visited is a sequence
```

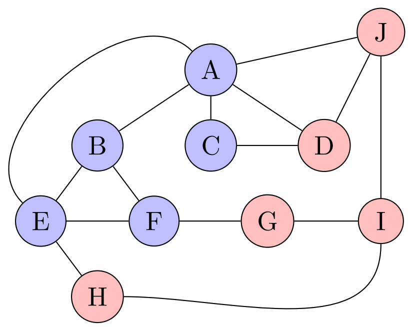
16.5.3 Illustrated Comparison

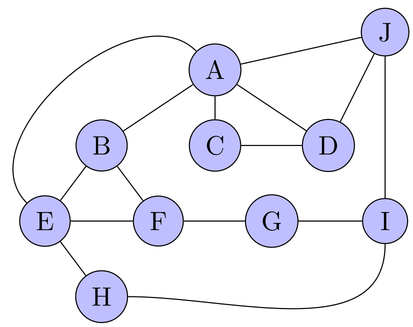
For the purposes of this illustration DFS will be contained within the column on the left-hand side, whereas BFS will be contained within the column on the right-hand side (all the way down).

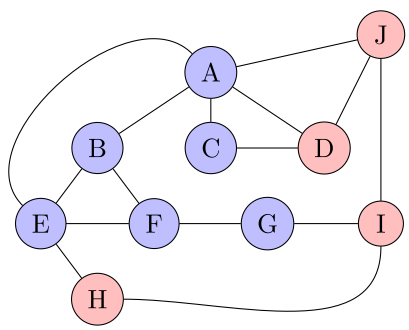


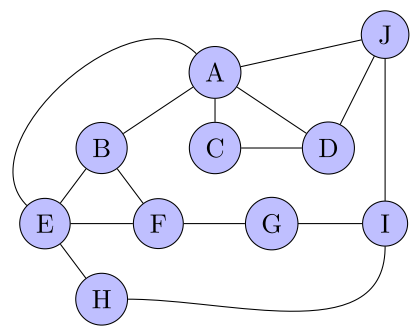

A
B
C
D
E
F
G
H
I
J

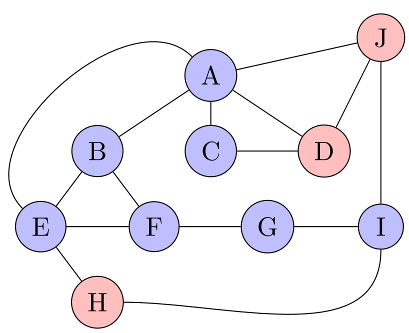

A
B
C
D
E
F
G
H
I
J

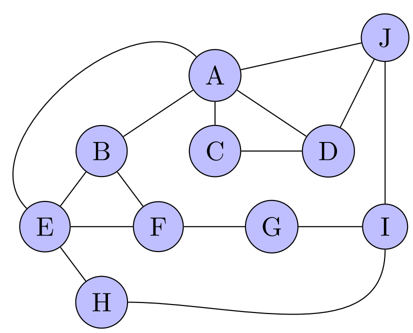

A
B
C
D
E
F
G
H
I
J

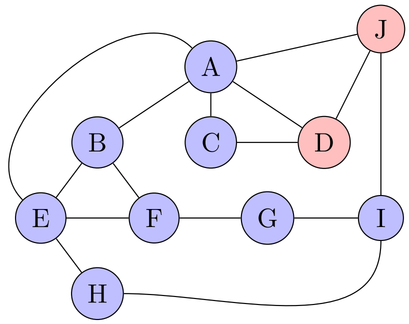

A
B
C
D
E
F
G
H
I
J

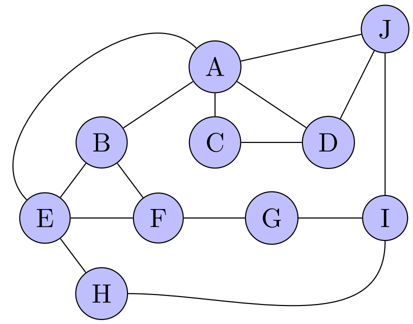

A
B
C
D
E
F
G
H
I
J

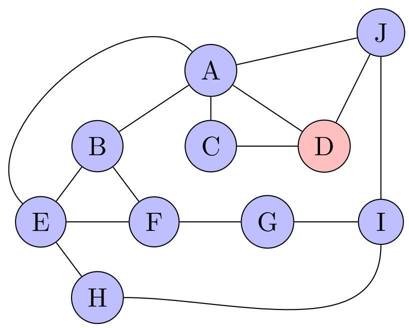

A
B
C
D
E
F
G
H
I
J

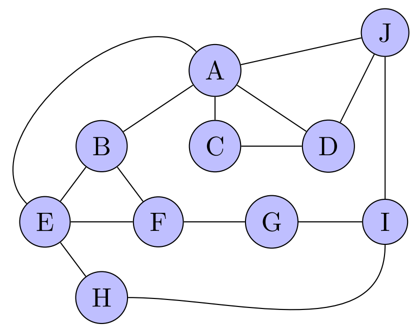

A
B
C
D
E
F
G
H
I
J

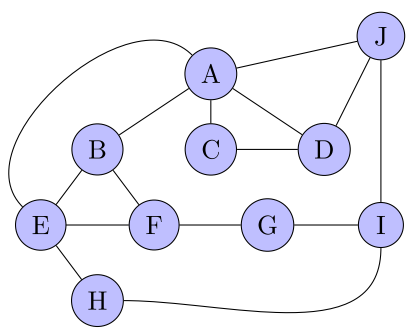

A
B
C
D
E
F
G
H
I
J

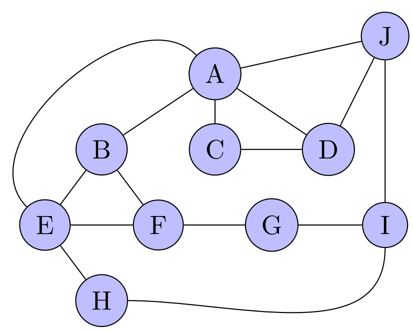












References

[1] Graph and its representations. Accessed 15/04/19.
`https://www.geeksforgeeks.org/graph-and-its-representations/`