INT102
Algorithmic Foundations And Problem Solving
Graph Theory

Dr Yushi Li
Department of Computer Science

西交利物浦大学
Xi'an Jiaotong-Liverpool University

Learning outcomes

- Able to tell what is an undirected graph and what is a directed graph
  - Know how to represent a graph using matrix and list
- Understand what Euler path / circuit and able to determine whether such path / circuit exists in an undirected graph
- Able to apply BFS and DFS to traverse a graph
- Able to tell what a tree is

Learning outcomes

- **Able to tell what is an undirected graph and what is a directed graph**
  - **Know how to represent a graph using matrix and list**
- Understand what Euler path / circuit and able to determine whether such path / circuit exists in an undirected graph
- Able to apply BFS and DFS to traverse a graph
- Able to tell what a tree is

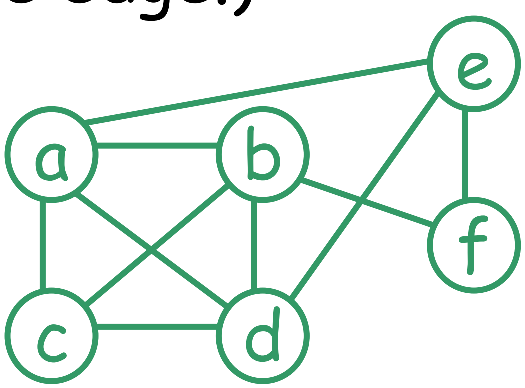
Graph …

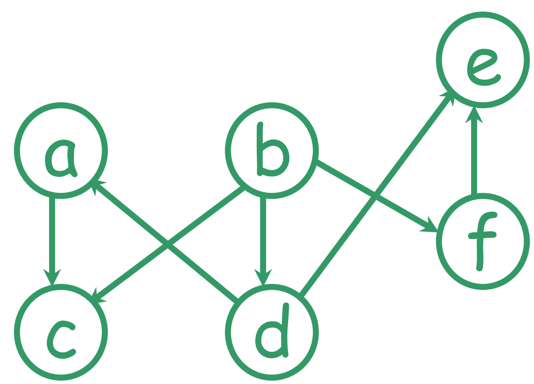
introduced in the 18th century

Graphs

Graph theory – an old subject with many modern applications.

An *undirected* graph G=(V,E) consists of a set of vertices V and a set of edges E. Each edge is an *unordered* pair of vertices. (E.g., {b,c} & {c,b} refer to the same edge.)





A *directed* graph G=(V,E) consists of … Each edge is an *ordered* pair of vertices. (E.g., (b,c) refer to an edge from b to c.)

Applications of graphs

In computer science, graphs are often used to model

- computer networks,
- precedence among processes,
- state space of playing chess (AI applications)
- resource conflicts, …

In other disciplines, graphs are also used to model the structure of objects. E.g.,

- biology - evolutionary relationship
- chemistry - structure of molecules

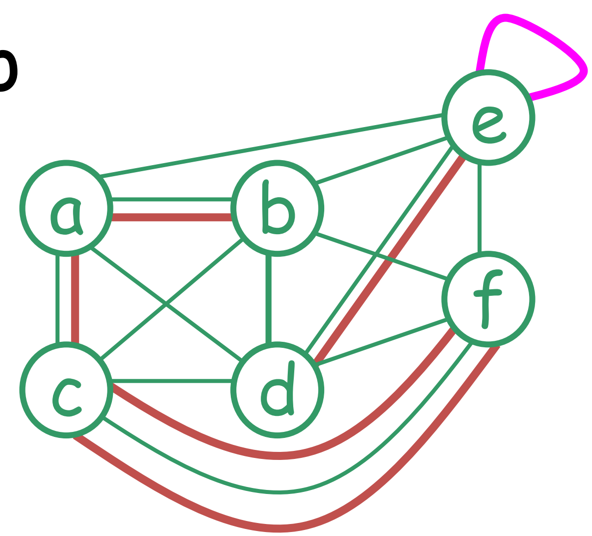
Undirected graph …

Undirected graphs

Undirected graphs:

- **simple graph:** at most one edge between two vertices, no self loop (i.e., an edge from a vertex to itself).
- **multigraph:** allows more than one edge between two vertices.
- **pseudograph:** allows a self loop

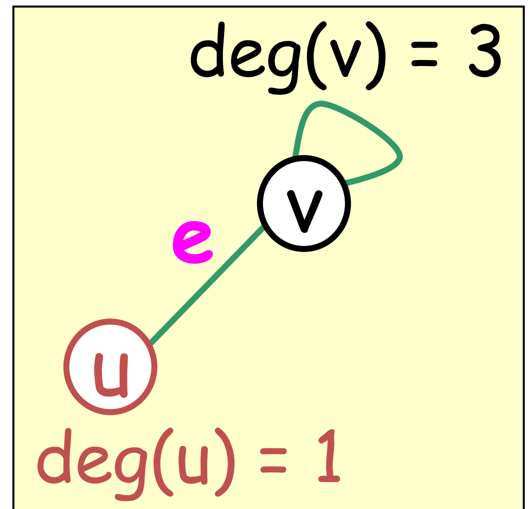
Reminder: An undirected graph G=(V,E) consists of a set of vertices V and a set of edges E. Each edge is an unordered pair of vertices.



Undirected graphs

In an undirected graph G, suppose that $e = \{u, v\}$ is an edge of G

- u and v are said to be ***adjacent*** and called ***neighbors*** of each other.
- u and v are called ***endpoints*** of e.
- e is said to be ***incident*** with u and v.
- e is said to ***connect*** u and v.



- The ***degree*** of a vertex v, denoted by **deg(v)**, is the number of edges incident with it (a loop contributes twice to the degree)

Representation (of undirected graphs)

An undirected graph can be represented by **adjacency matrix**, **adjacency list**, **incidence matrix** or **incidence list**.

Adjacency matrix and adjacency list record the relationship between **vertex adjacency**, i.e., a vertex is adjacent to which other vertices

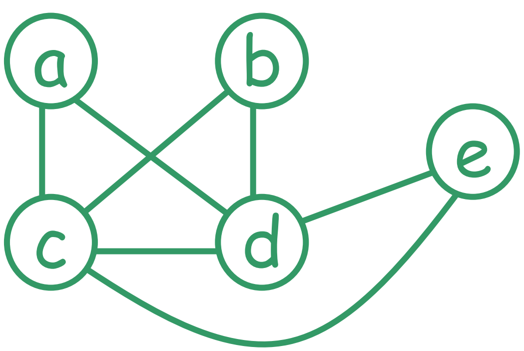
Incidence matrix and incidence list record the relationship between **edge incidence**, i.e., an edge is incident with which two vertices

Adjacency matrix / list

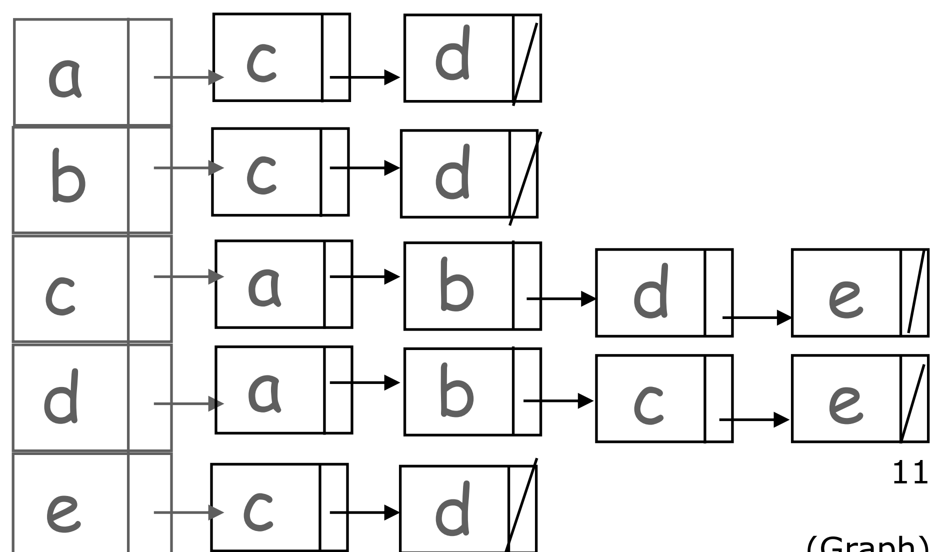
Adjacency matrix M for a simple **undirected** graph with n vertices:

- M is an nxn matrix
- M(i, j) = 1 if vertex i and vertex j are adjacent

Adjacency list: each vertex has a list of vertices to which it is adjacent



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array}\begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

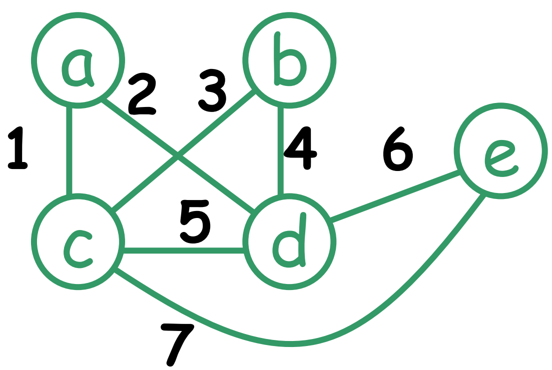


Incidence matrix / list

Incidence matrix M for a simple **undirected** graph with n vertices and m edges:

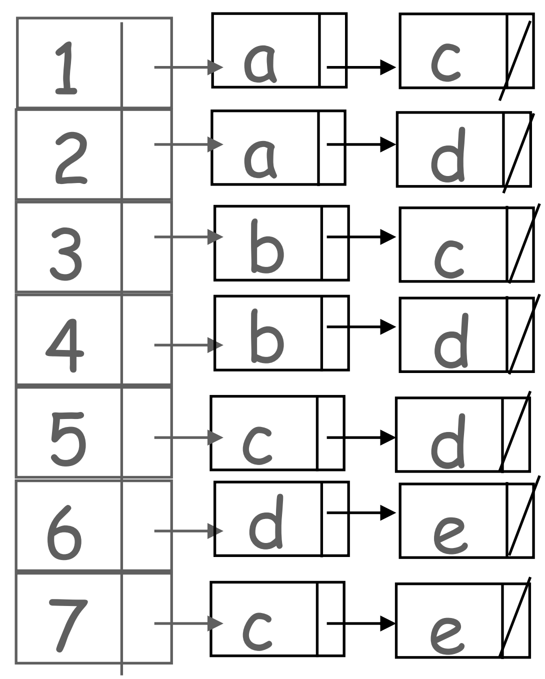
- M is an mxn matrix
- M(i, j) = 1 if edge i and vertex j are incidence

Incidence list: each edge has a list of vertices to which it is incident with



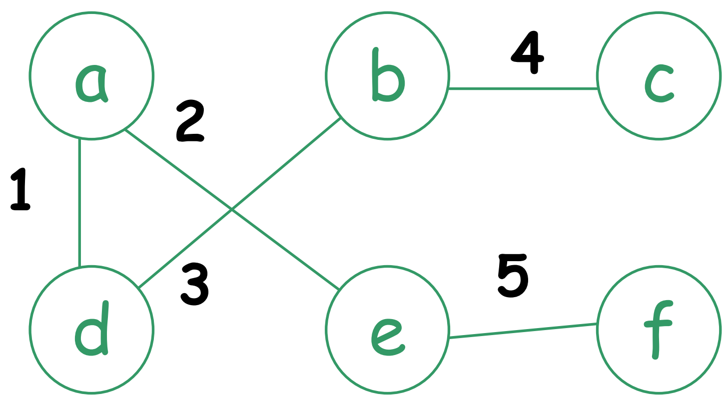
labels of edge are edge number

$$
\begin{array}{c}
 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7
\end{array}
\begin{array}{c}
\begin{array}{ccccc} a & b & c & d & e \end{array} \\
\begin{pmatrix}
1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1
\end{pmatrix}
\end{array}
$$



Exercise

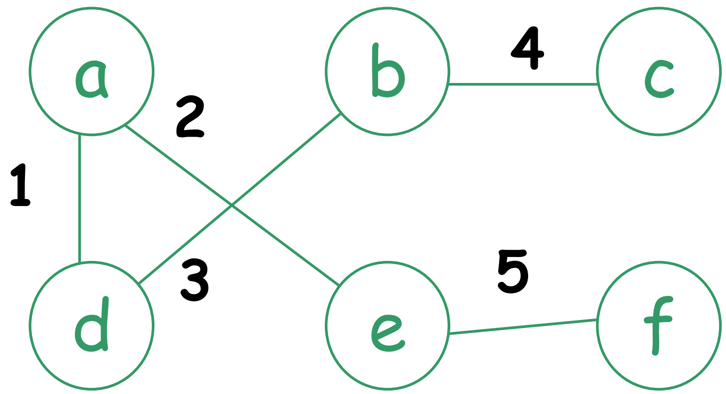
Give the adjacency matrix and incidence matrix of the following graph



labels of edge are edge number

Exercise

Give the adjacency matrix and incidence matrix of the following graph



labels of edge are edge number

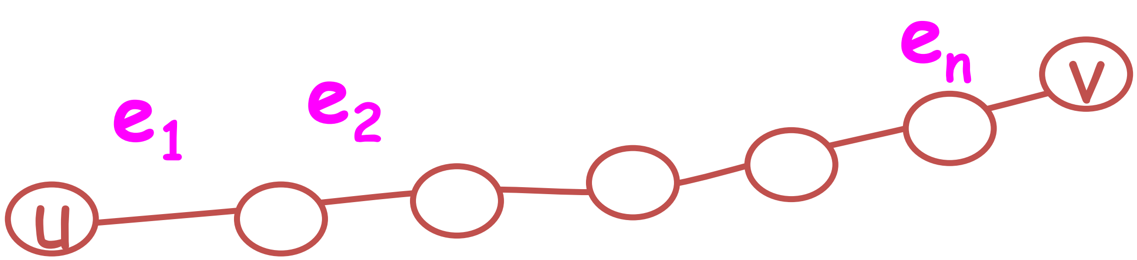
Learning outcomes

- ✓ Able to tell what is an undirected graph *and what is a directed graph*
- ✓ Know how to represent a graph using matrix and list
- ➢ **Understand what Euler path / circuit and able to determine whether such path / circuit exists in an undirected graph**
- ➢ Able to apply BFS and DFS to traverse a graph
- ➢ Able to tell what a tree is

Euler circuit / path …

Paths, circuits (in undirected graphs)

- In an undirected graph, a **path** from a vertex ***u*** to a vertex ***v*** is a sequence of edges $e_1 = \{u, x_1\}$, $e_2 = \{x_1, x_2\}$, …$e_n = \{x_{n-1}, v\}$, where **n≥1**.
- The **length** of this path is **n**.
- Note that a path from ***u*** to ***v*** implies a path from ***v*** to ***u***.
- If u = v, this path is called a **circuit** (cycle).

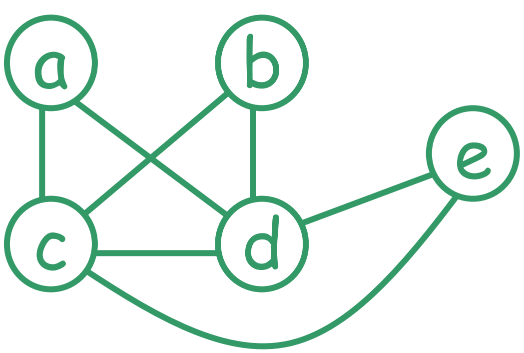


Euler circuit

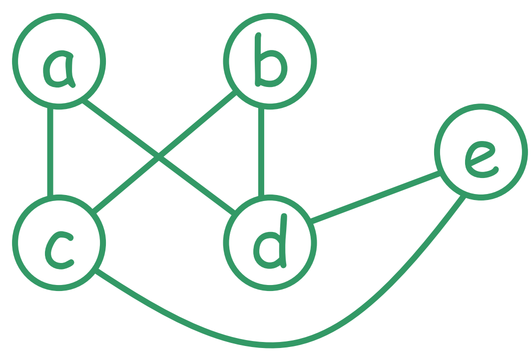
A simple circuit visits an edge **at most** once.

An **Euler** circuit in a graph G is a circuit visiting every edge of G **exactly** once.
(NB. A vertex can be repeated.)

Does every graph has an Euler circuit ?

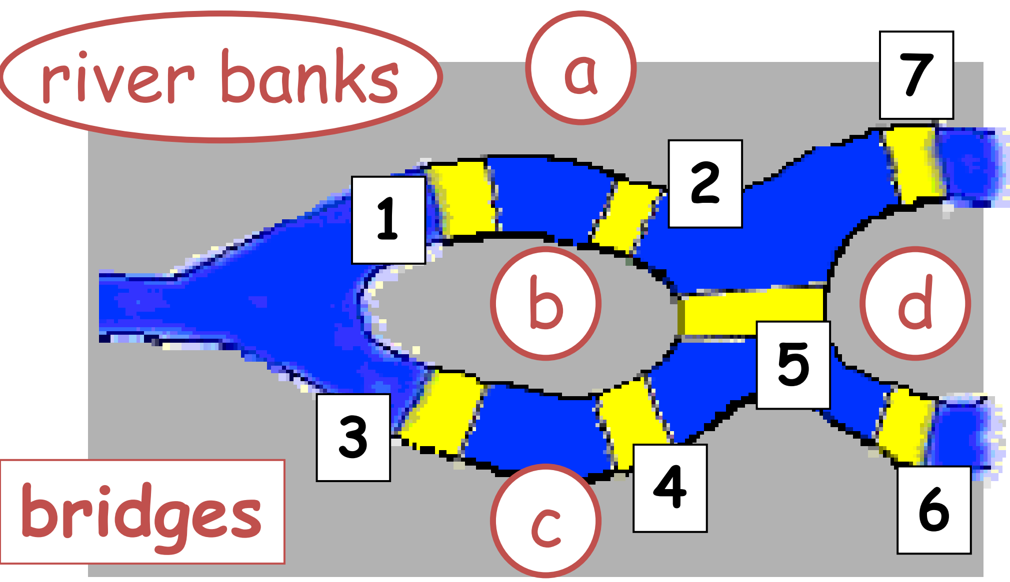


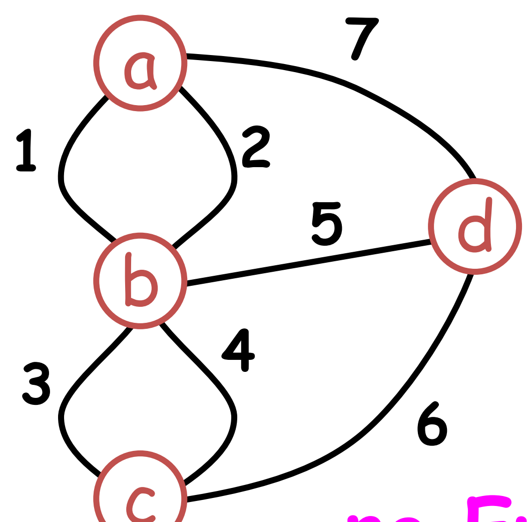
a c b d e c d a



no Euler circuit

History: In Konigsberg, Germany, a river ran through the city and seven bridges were built. The people wondered whether or not one could go around the city in a way that would involve crossing each bridge exactly once.



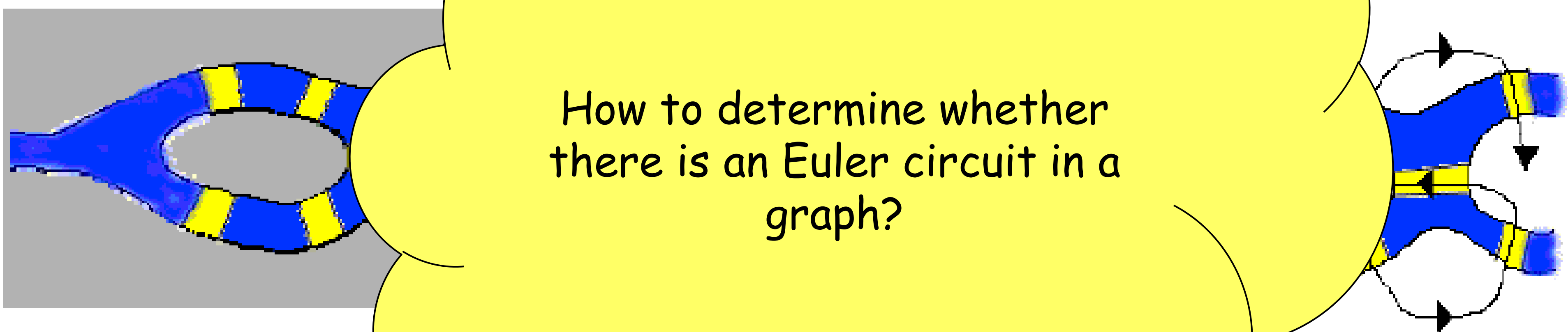


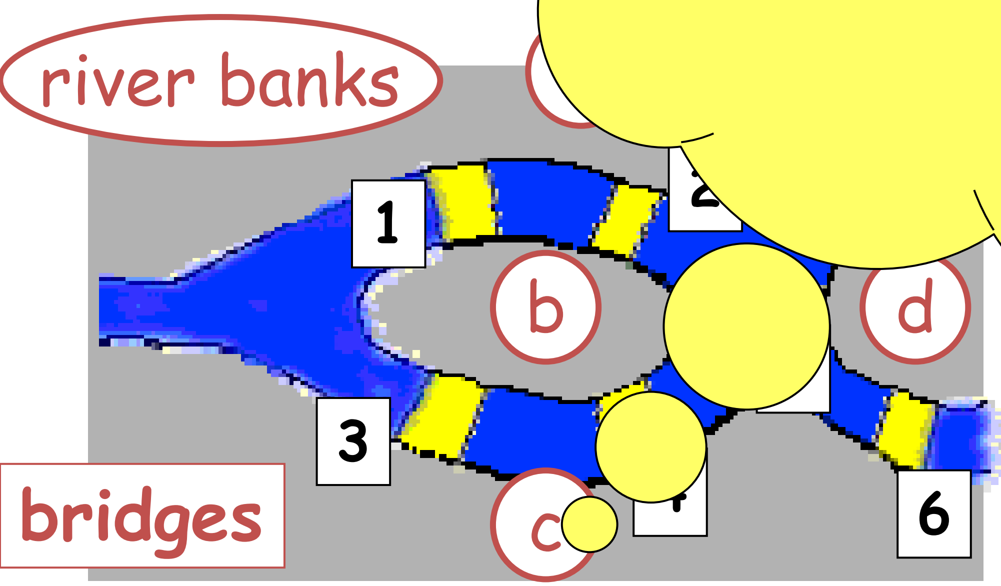
no Euler circuit

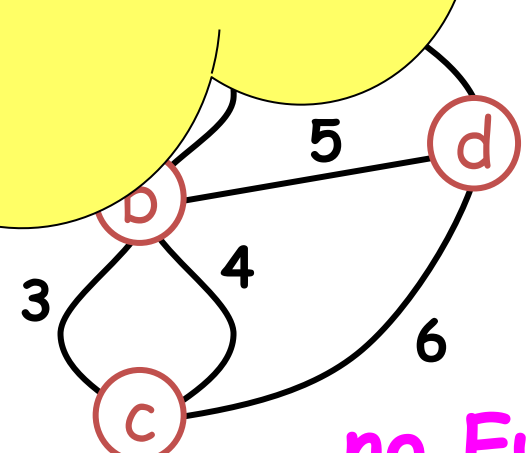
(Graph)

History: In Konigsberg, Germany, a river ran through the city and seven bridges were built. The people wondered whether or not one could go around the city in ... involve crossing eac...

How to determine whether there is an Euler circuit in a graph?



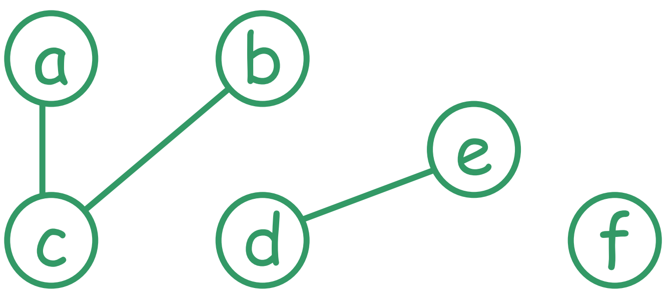




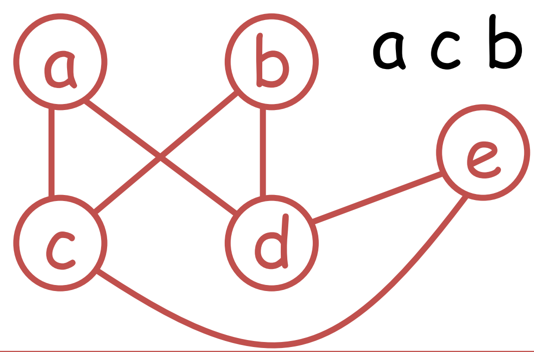
A trivial condition

An undirected graph G is said to be **connected** if there is a path between ***every pair*** of vertices.

If G is **not** connected, there is no single circuit to visit all edges or vertices.



Even if the graph is connected, there may be no Euler circuit either.

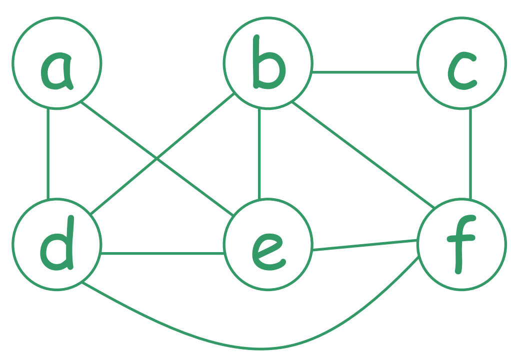


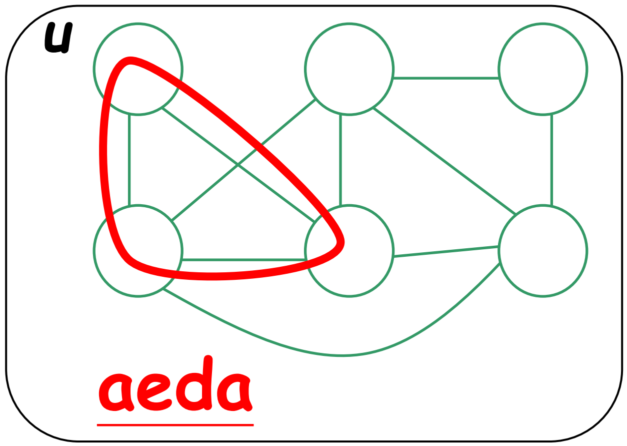
a c b d e c b d a

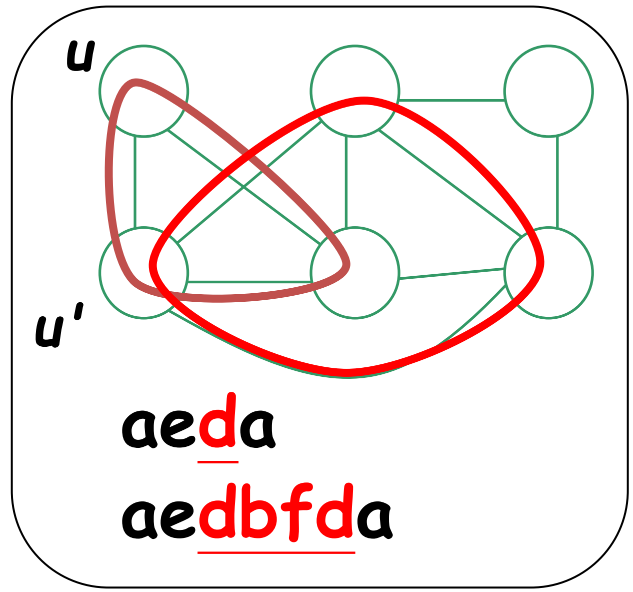
Necessary and sufficient condition

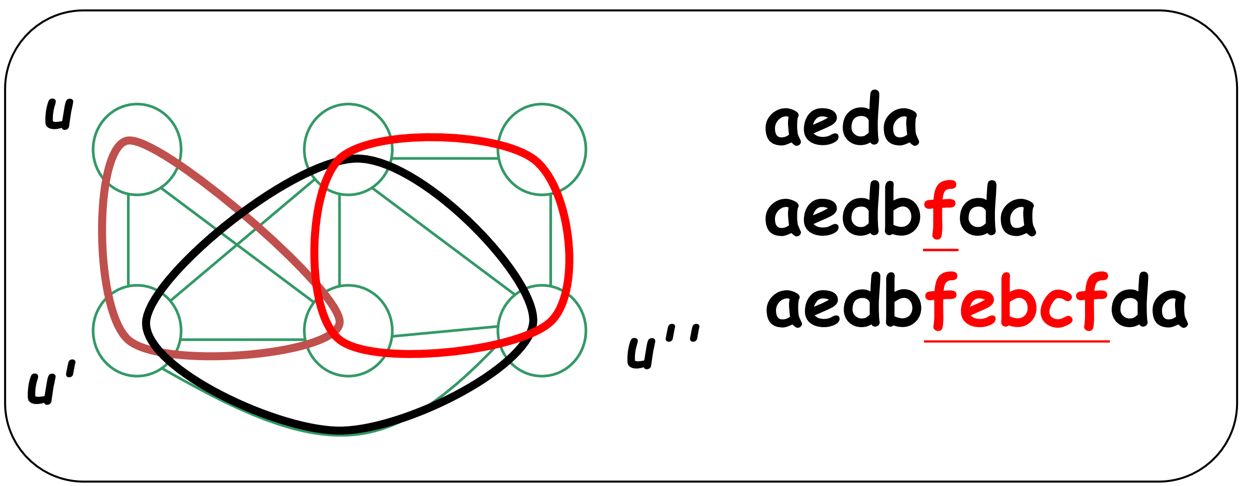
Let G be a connected graph.

Lemma: G contains an Euler circuit if and only if every vertex has **even** degree.









Euler path

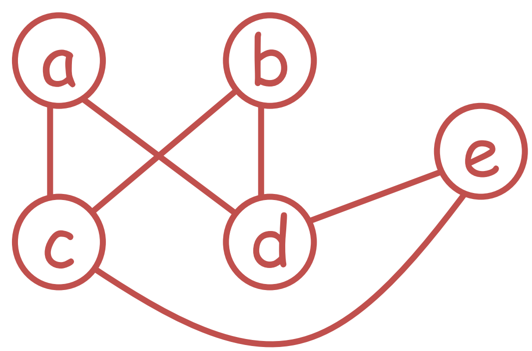
Let G be an undirected graph.

An **Euler path** is a path visiting every edge of G exactly once.

An undirected graph contains an Euler path if it is connected and contains exactly **two vertices of odd degree**.

This graph has no Euler circuit, but has an Euler path

c b d a c e d



Summary

Given a connected undirected graph

	Exist Euler circuit?	Exist Euler path?
all vertices have even degree	YES	YES
exactly two vertices have odd degree	NO	YES
more than 2 vertices have odd degree	NO	NO

Hamiltonian circuit / path

Let G be an undirected graph.

A **Hamiltonian circuit** (path) is a circuit (path) containing **every vertex** of G **exactly once**.

Note that a Hamiltonian circuit or path may NOT visit all edges.

Unlike the case of Euler circuits / paths, determining whether a graph contains a Hamiltonian circuit (path) is a very ***difficult*** problem. (NP-hard)

Learning outcomes

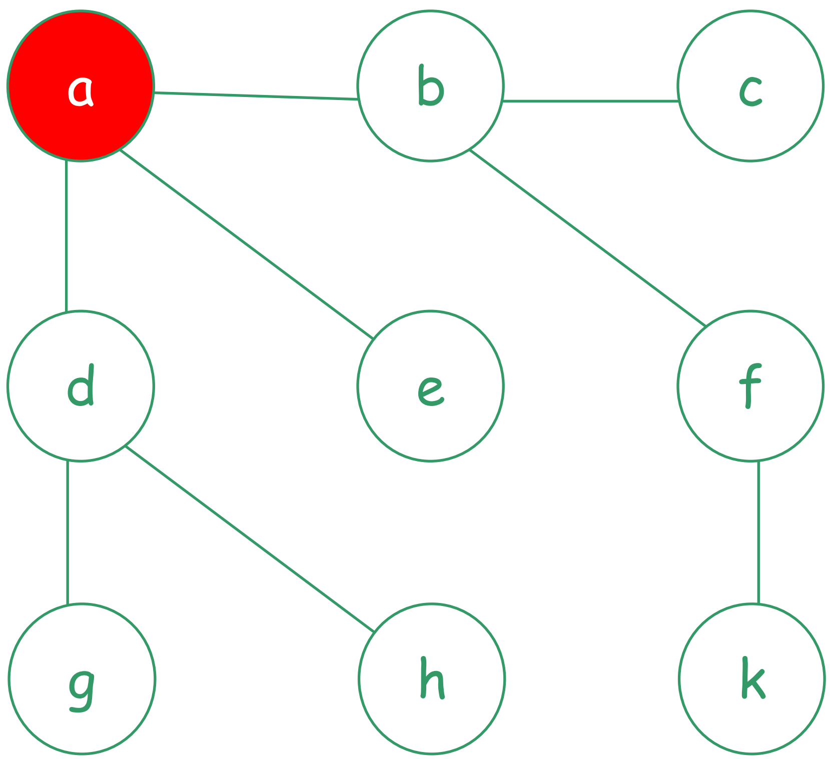
- ✓ Able to tell what is an undirected graph *and what is a directed graph*
  - ✓ Know how to represent a graph using matrix and list
- ✓ Understand what Euler path / circuit and able to determine whether such path / circuit exists in an undirected graph
- ➢ **Able to apply BFS and DFS to traverse a graph**
- ➢ Able to tell what a tree is

Breadth First Search BFS …

Breadth-first search (BFS)

All vertices at distance k from s are explored before any vertices at distance k+1.

The source is a.

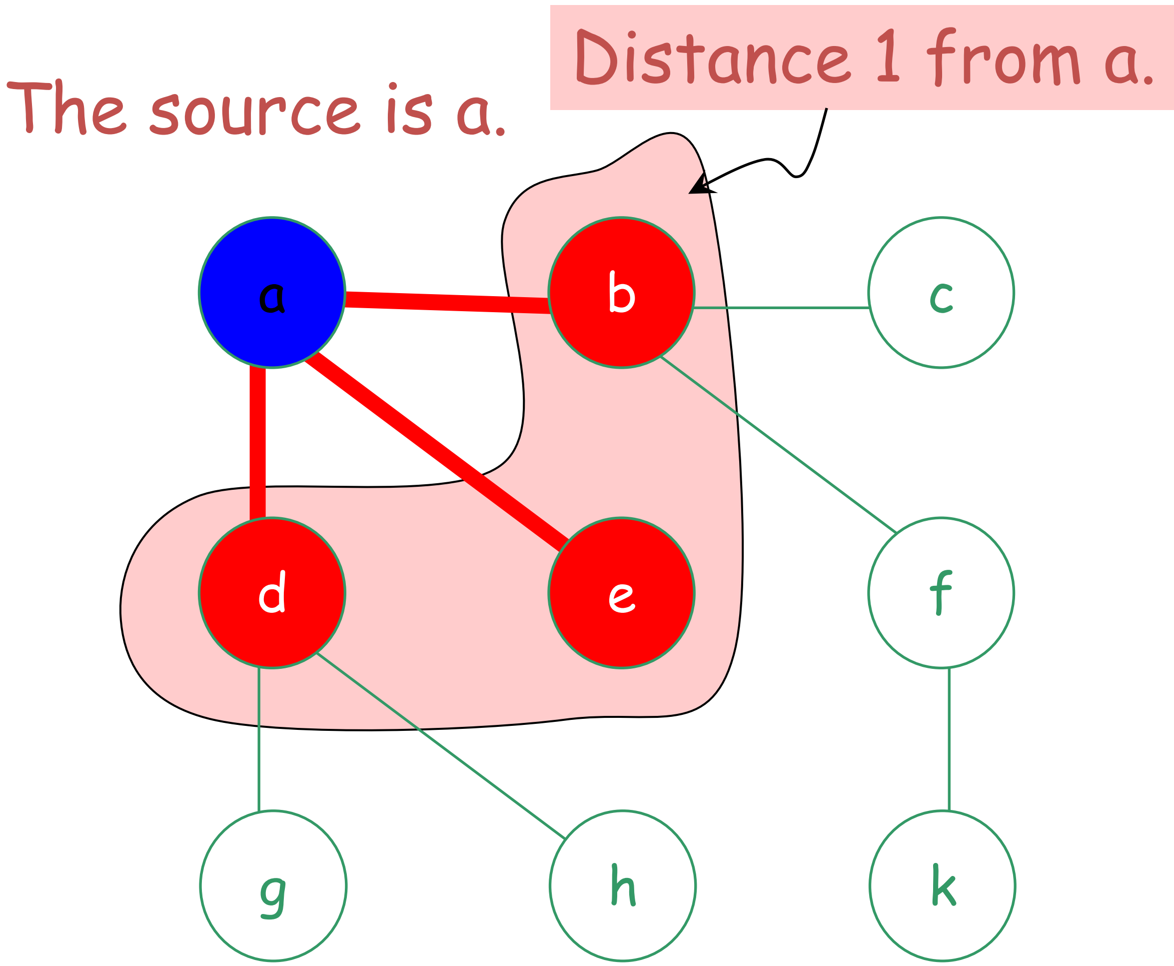


Order of exploration

a,

Breadth-first search (BFS)

All vertices at distance k from s are explored before any vertices at distance k+1.



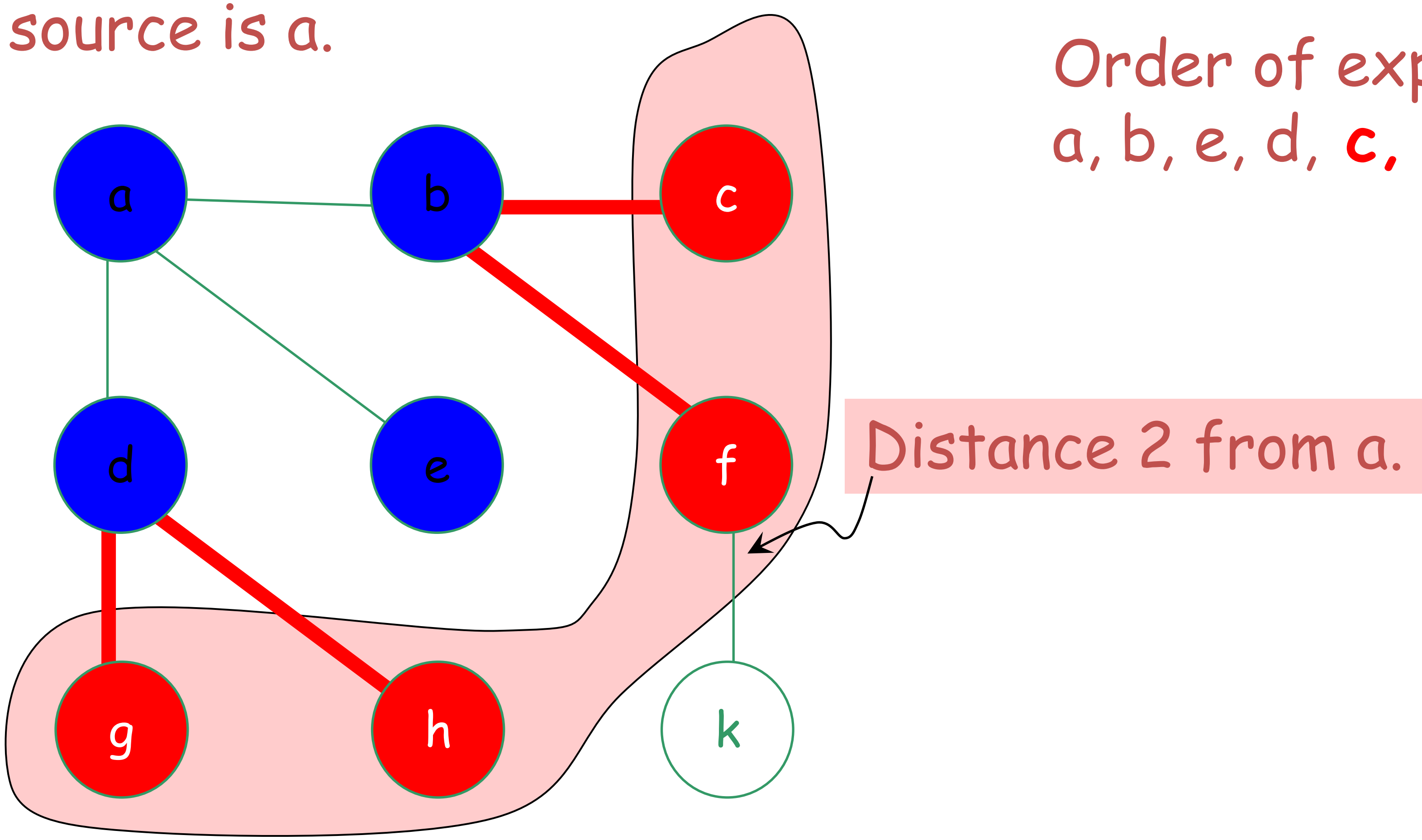
Order of exploration
a, **b, e, d**

Breadth-first search (BFS)

All vertices at distance k from s are explored before any vertices at distance k+1.

The source is a.

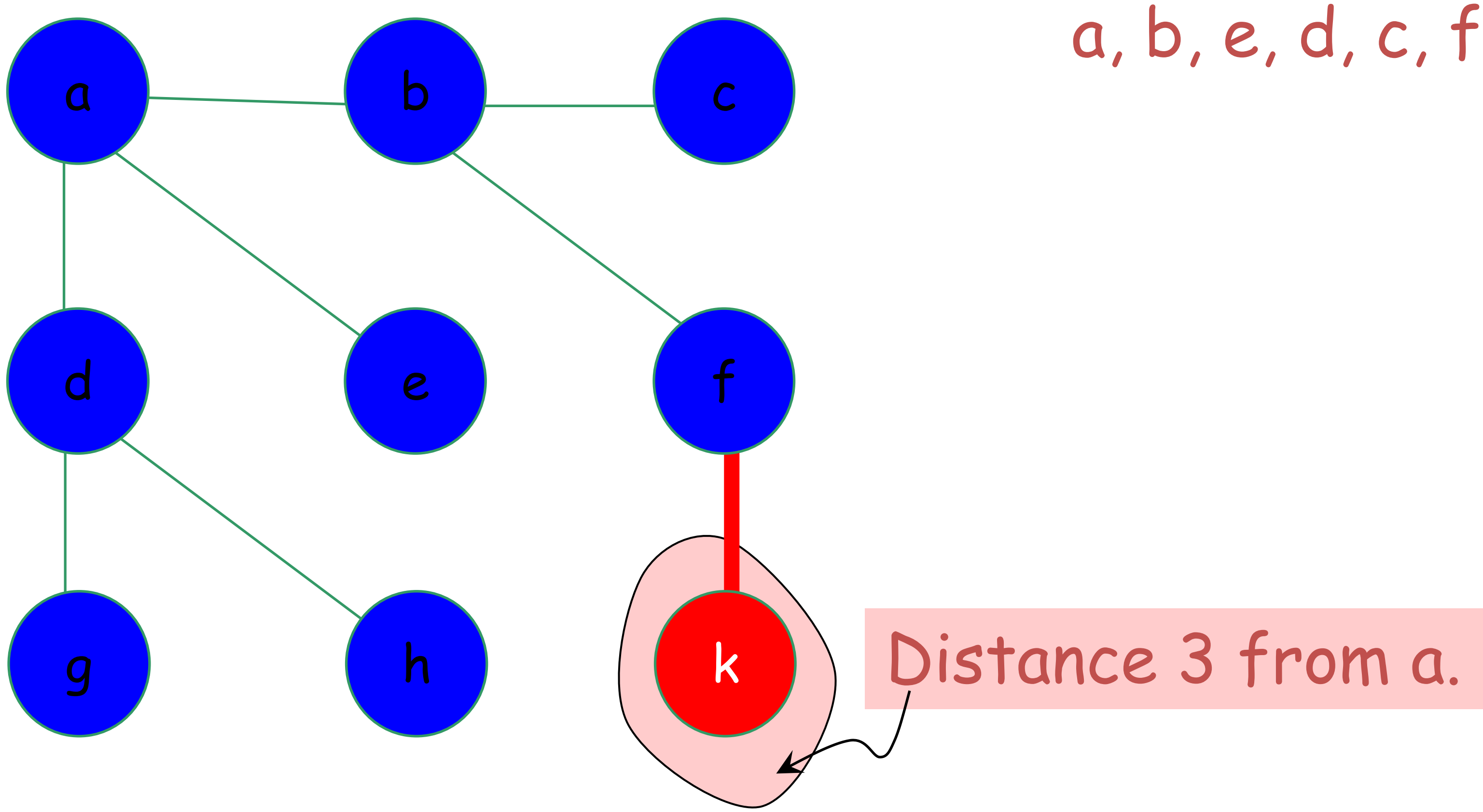
Order of exploration
a, b, e, d, **c, f, h, g**



Breadth-first search (BFS)

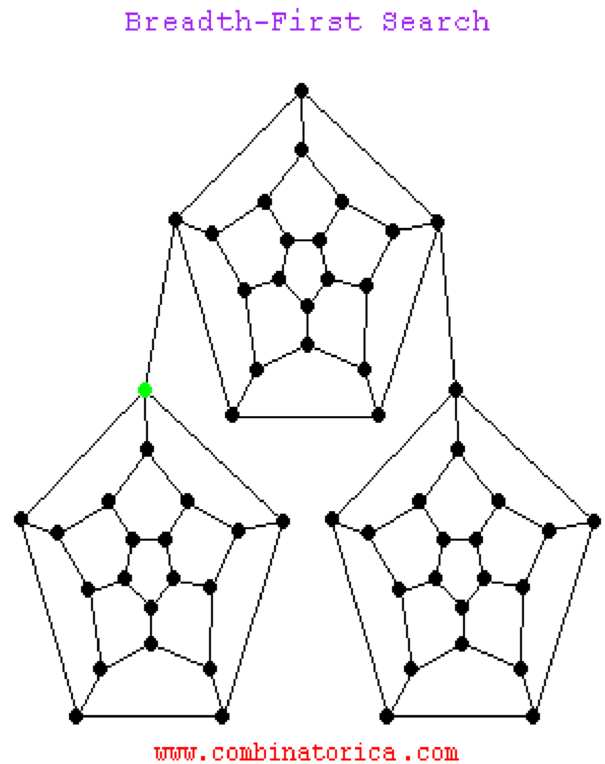
All vertices at distance k from s are explored before any vertices at distance k+1.

The source is a.



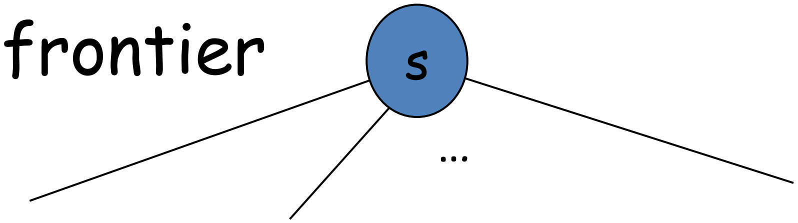
Order of exploration
a, b, e, d, c, f, h, g, **k**

In general (BFS)



In general (BFS)

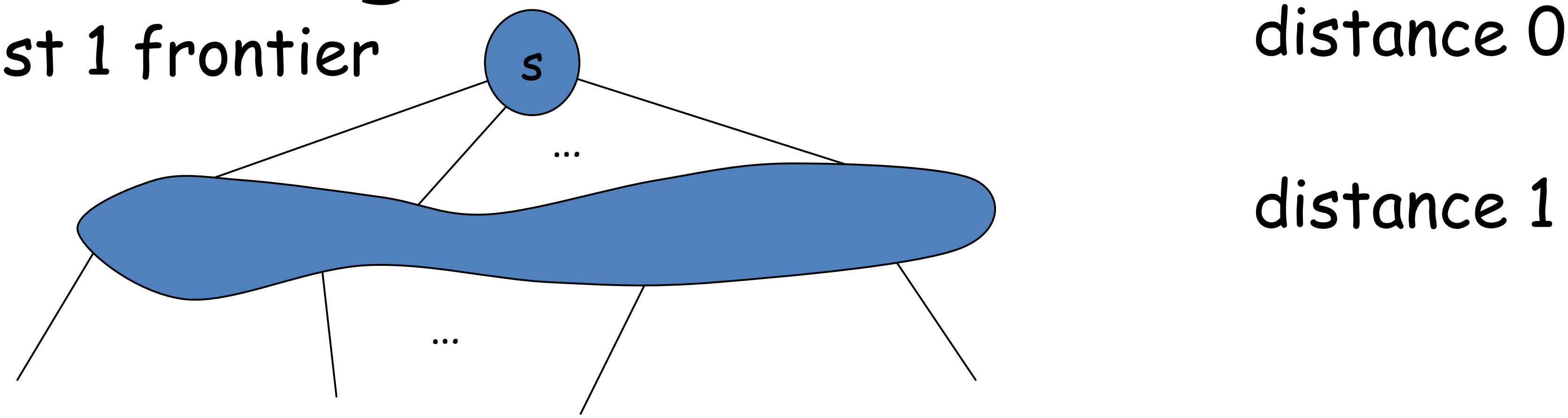
Explore dist 0 frontier



distance 0

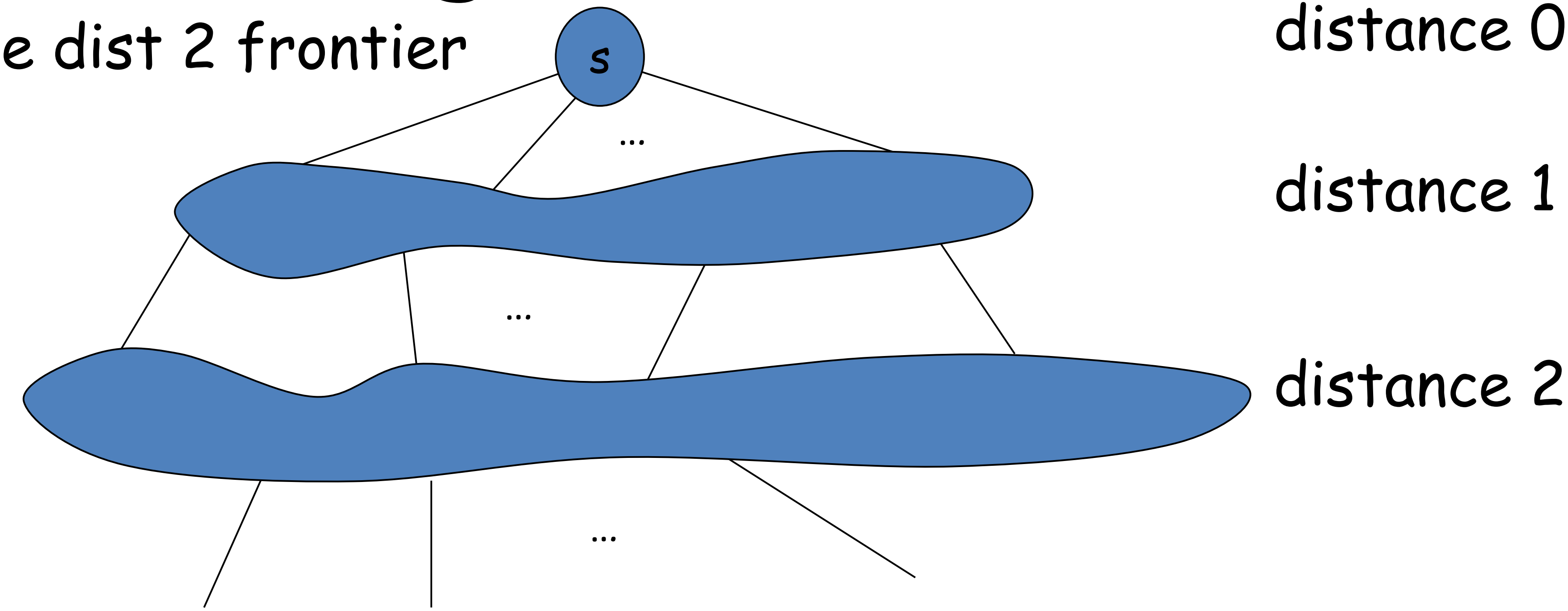
In general (BFS)

Explore dist 1 frontier



In general (BFS)

Explore dist 2 frontier



Breadth-first search (BFS)

A simple algorithm for searching a graph.

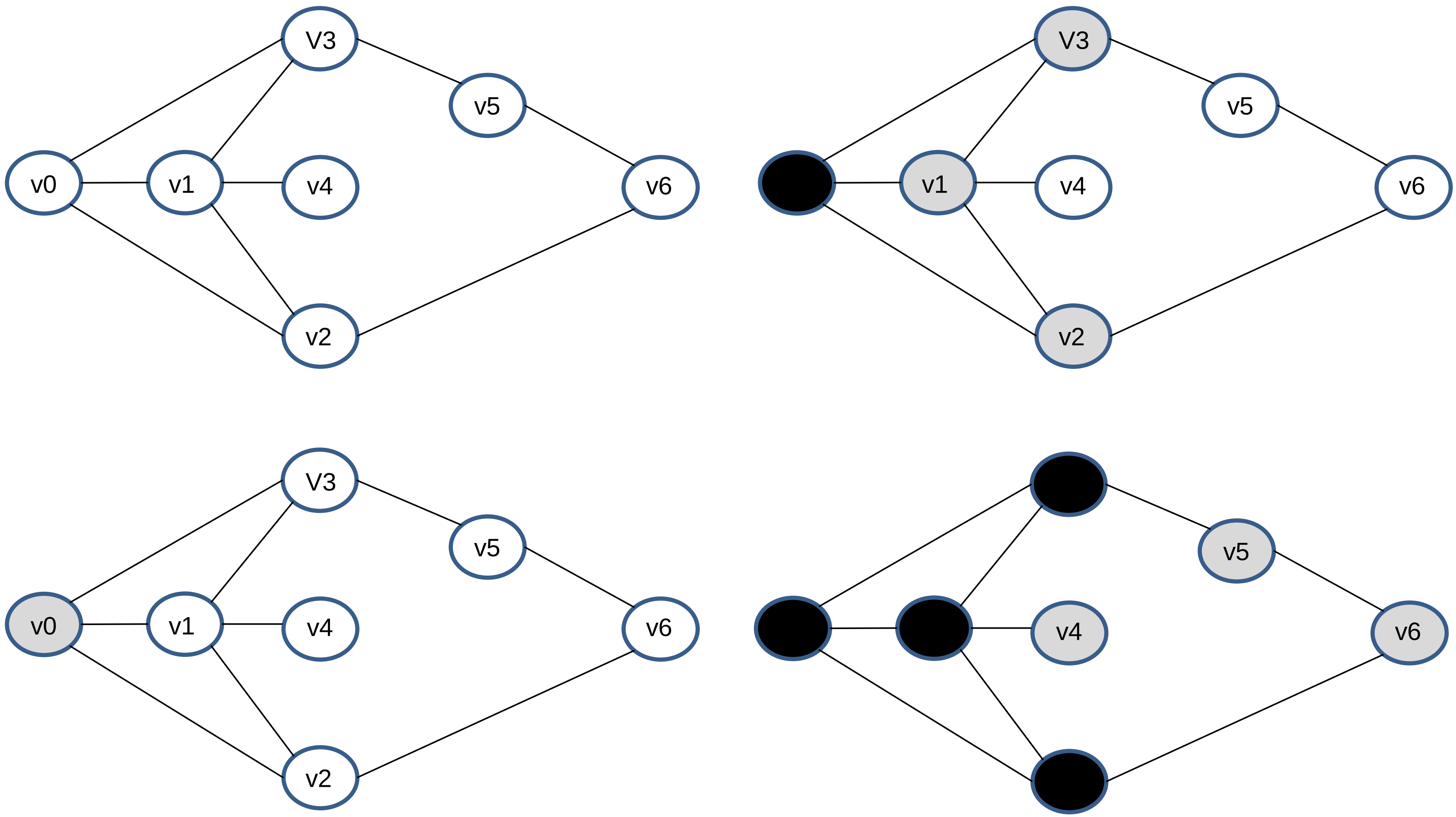
Given G=(V, E), and a distinguished source vertex **s**, BFS systematically explores the edges of G such that

- all vertices at **distance *k*** from s are explored **before** any vertices at **distance *k+1***.

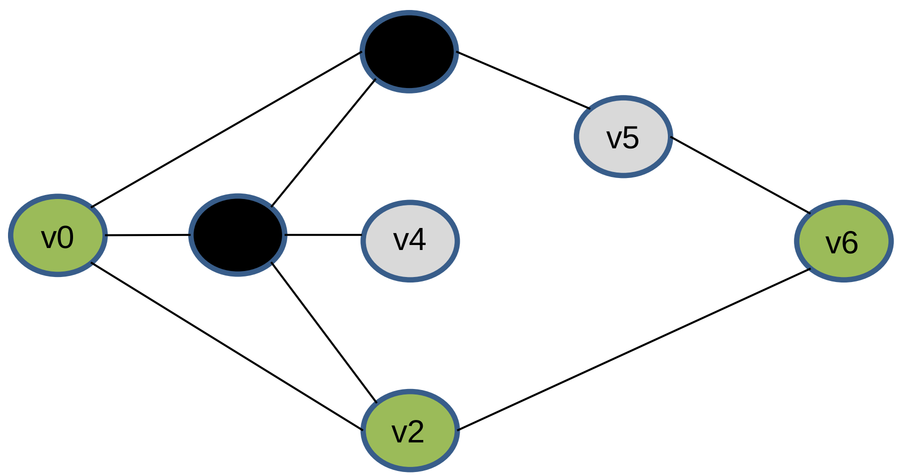
BFS – Pseudo code

```
unmark all vertices
choose some starting vertex s
mark s and insert s into tail of list L
while L is nonempty do
begin
   remove a vertex v from front of L
   visit v
   for each unmarked neighbor w of v do
         mark w and insert w into tail of list L
end
```

BFS – Pseudo Code (with data structure)



BFS – Pseudo Code (with data structure)

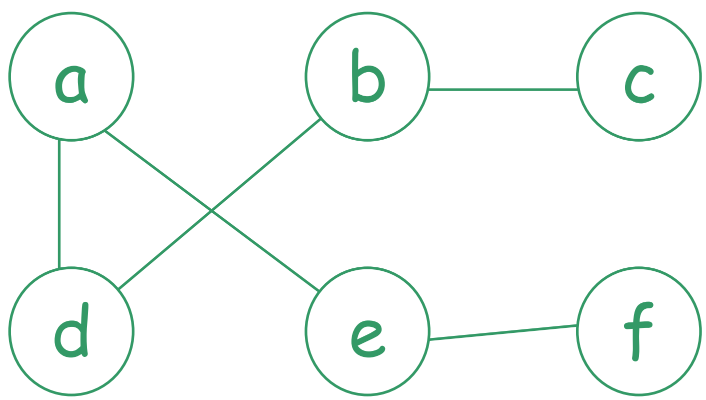


BFS – Pseudo Code (with data structure)

```
for each vertex u in V[G]-{s}
   do color[u] = white
Q=empty                                  //Q is a queue
enqueue(Q, s)
while Q is not empty
   do u = dequeue(Q)
       for each v in Adj(u)     //adjacency list of u
           do if color[v]=white then
                color[v]=gray
                enqueue(Q, v)
       color[u]=black
```

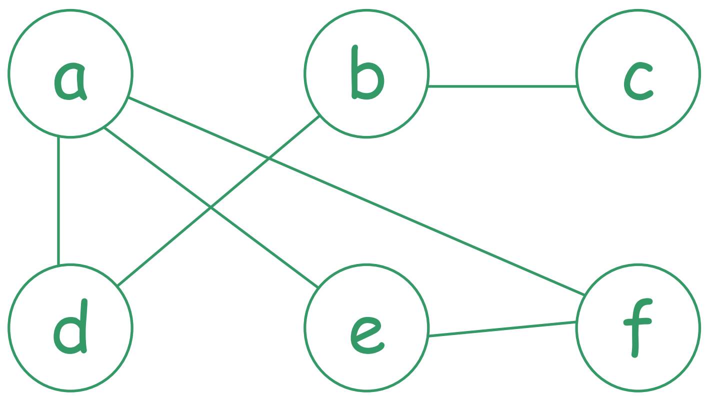
Exercise

Apply **BFS** to the following graph starting from vertex **a** and list the order of exploration



Exercise (2)

Apply **BFS** to the following graph starting from vertex **a** and list the order of exploration

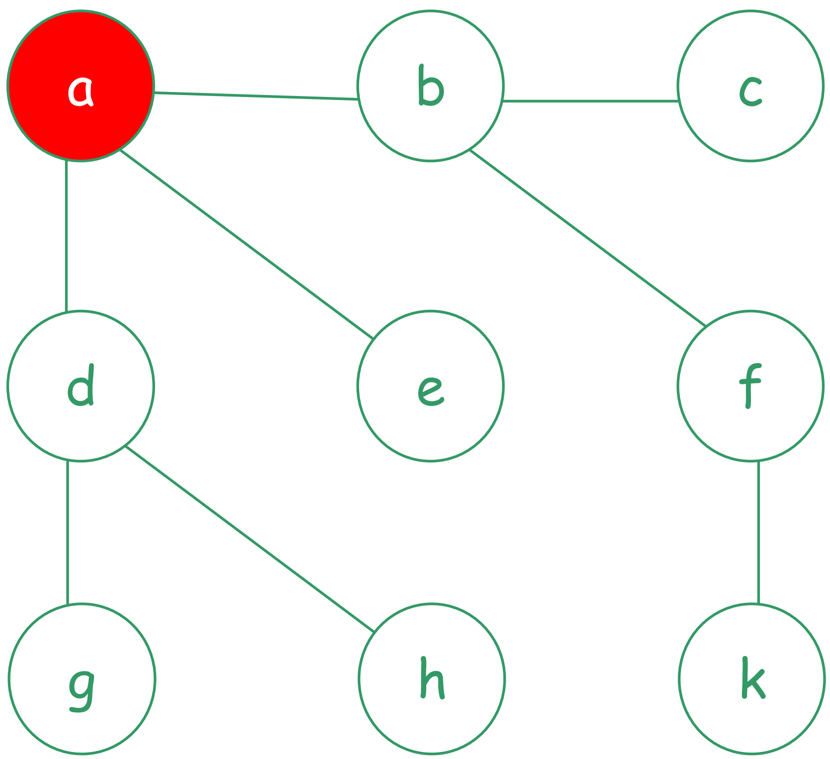


Depth First Search DFS …

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.



Order of exploration

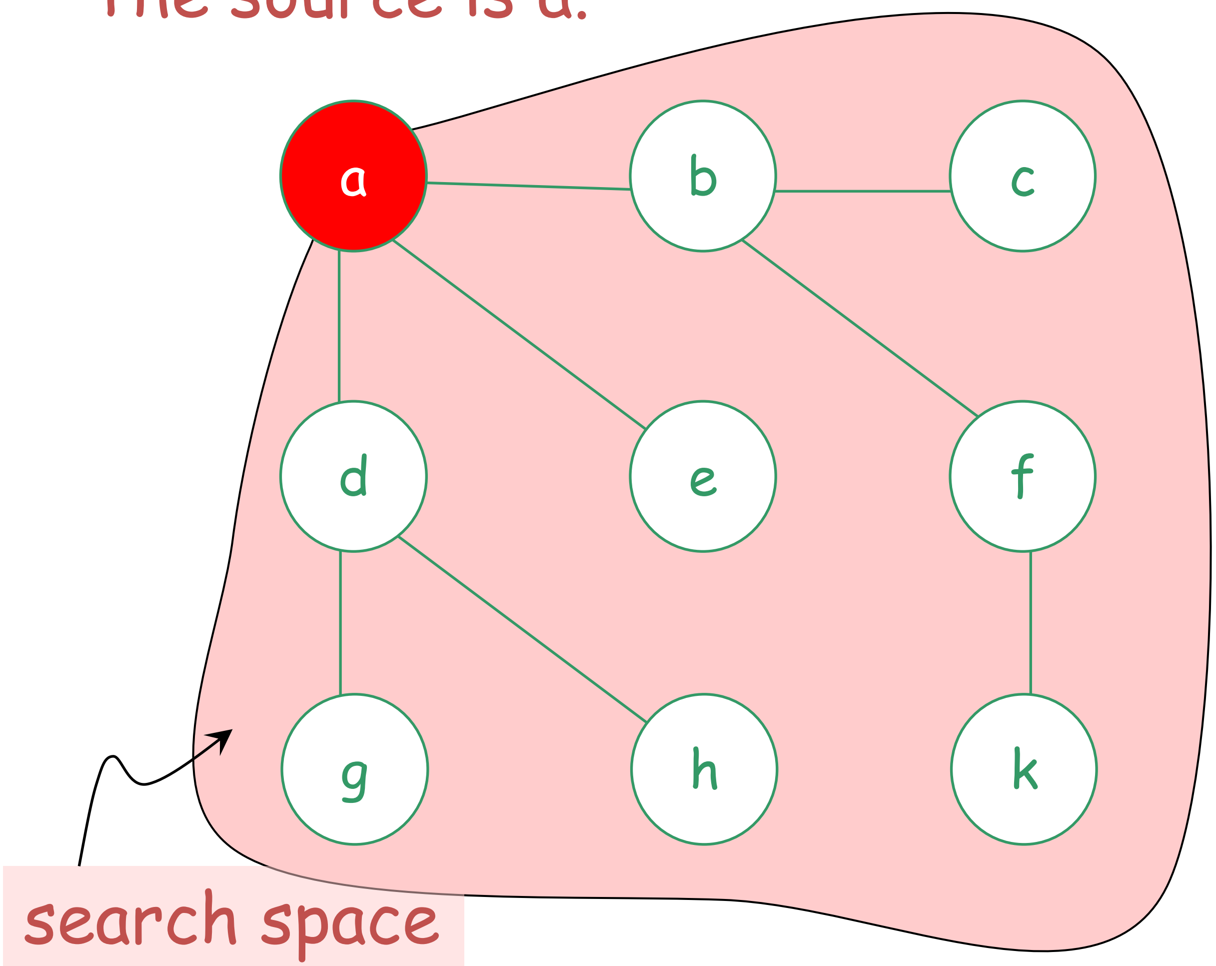
a,

DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.



Order of exploration

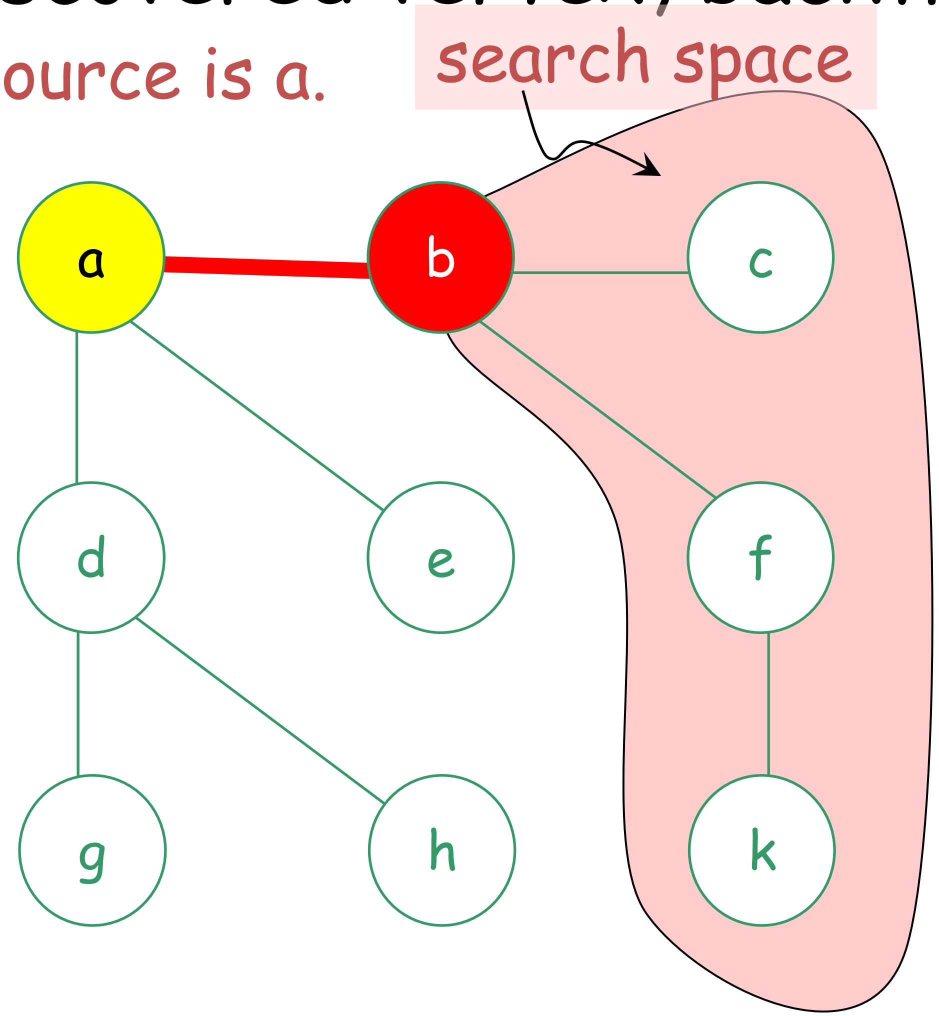
a,

DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.



Order of exploration
a, **b**

DFS searches "deeper" in the graph whenever possible

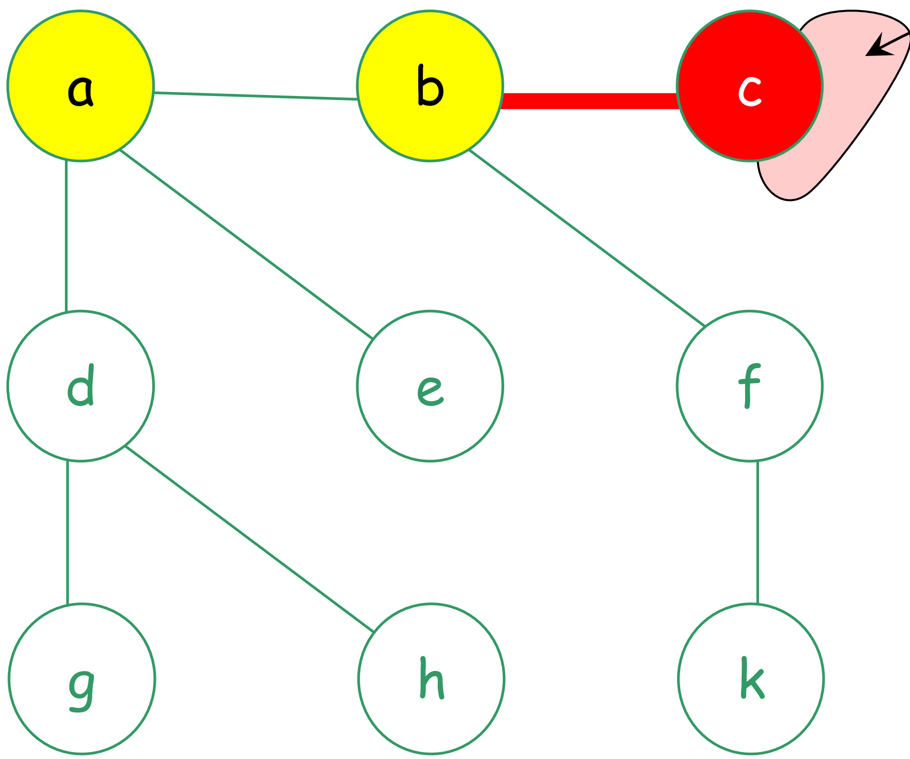
Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

search space is empty

Order of exploration
a, b, **c**



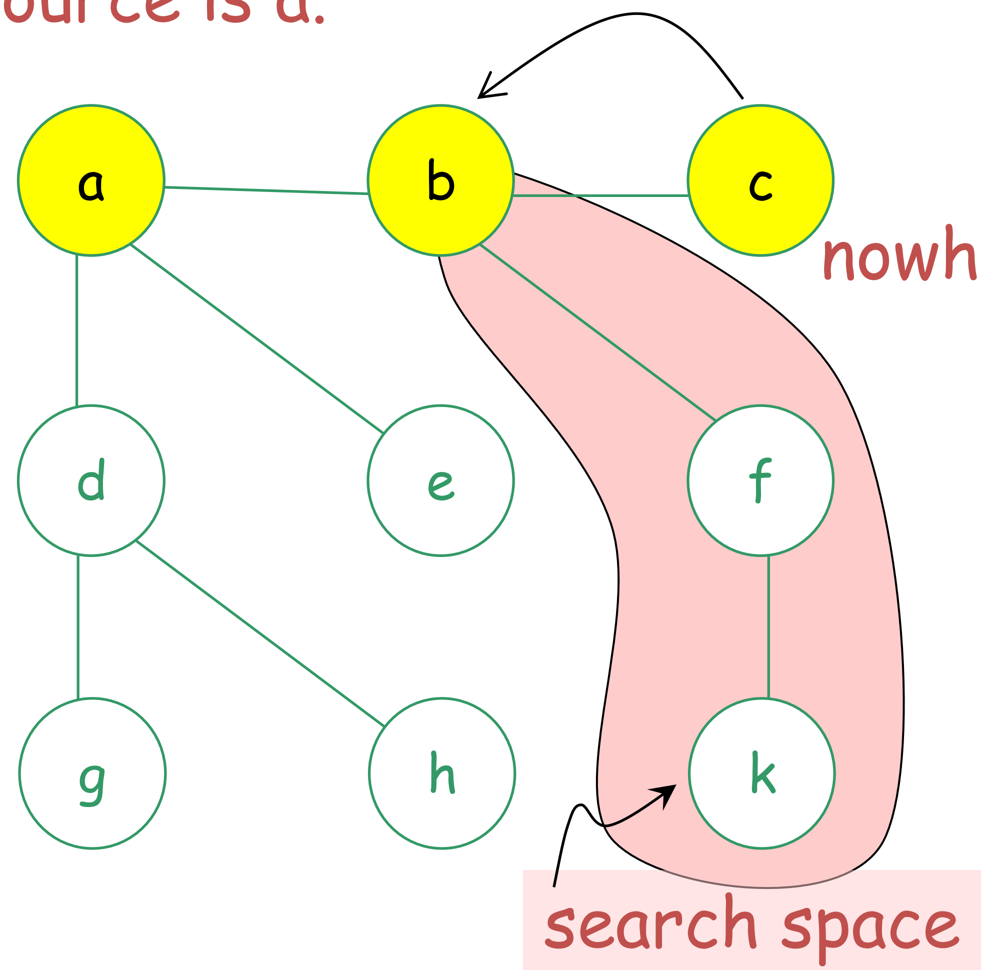
DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

Order of exploration
a, b, c



nowhere to go, backtrack

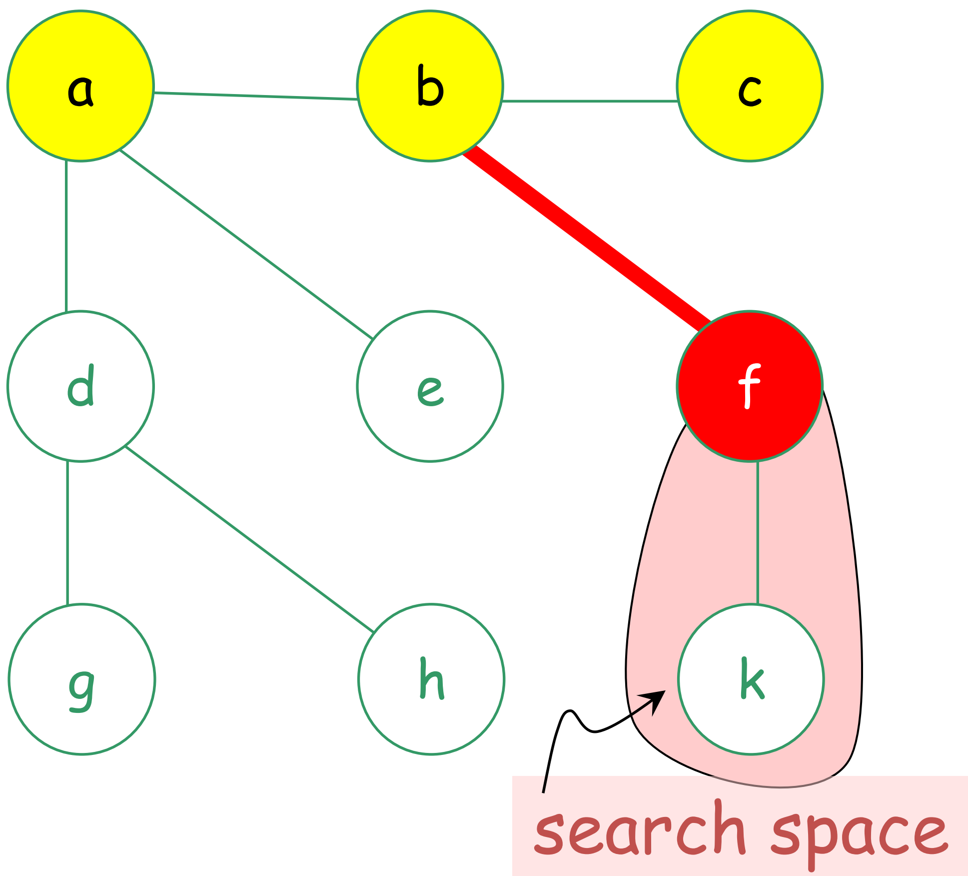
DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

Order of exploration
a, b, c, **f**



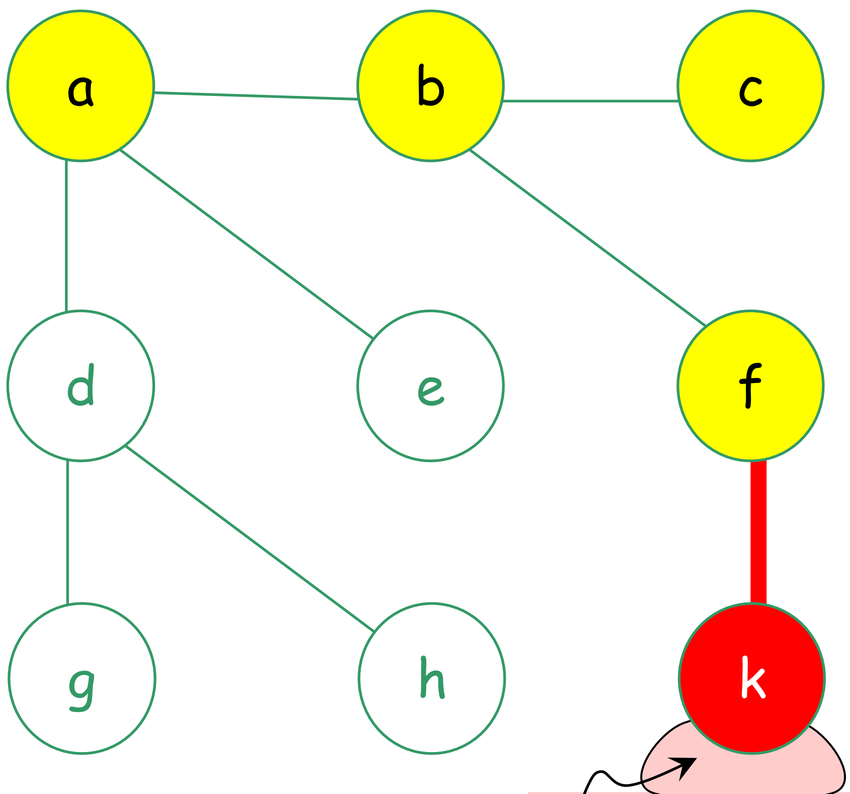
DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

Order of exploration
a, b, c, f, **k**



DFS searches "deeper" in the graph whenever possible

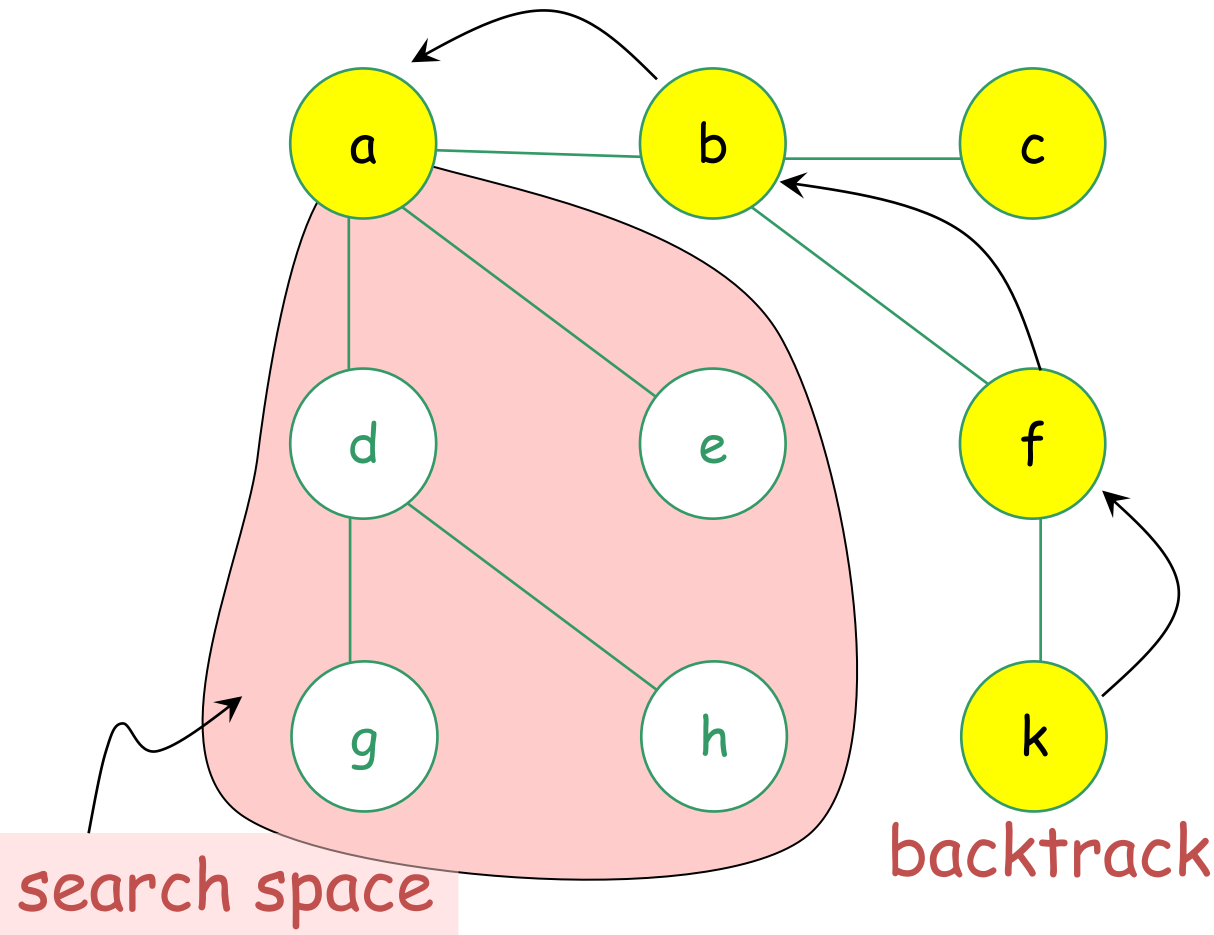
search space is

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

Order of exploration
a, b, c, f, k



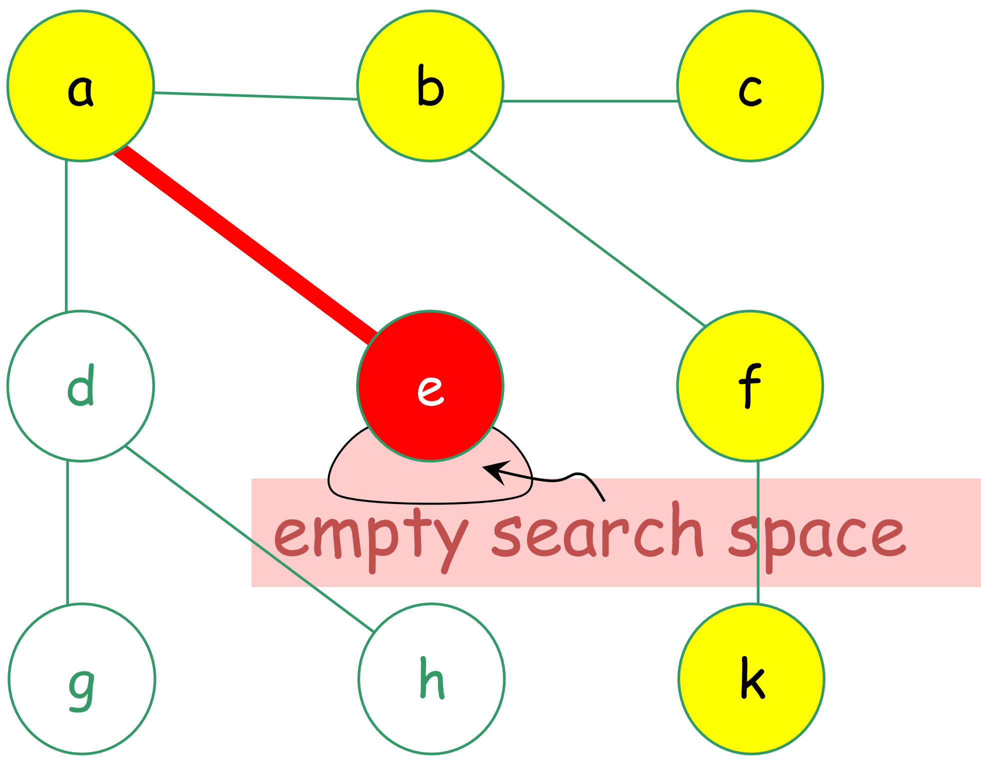
DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

Order of exploration
a, b, c, f, k, **e**



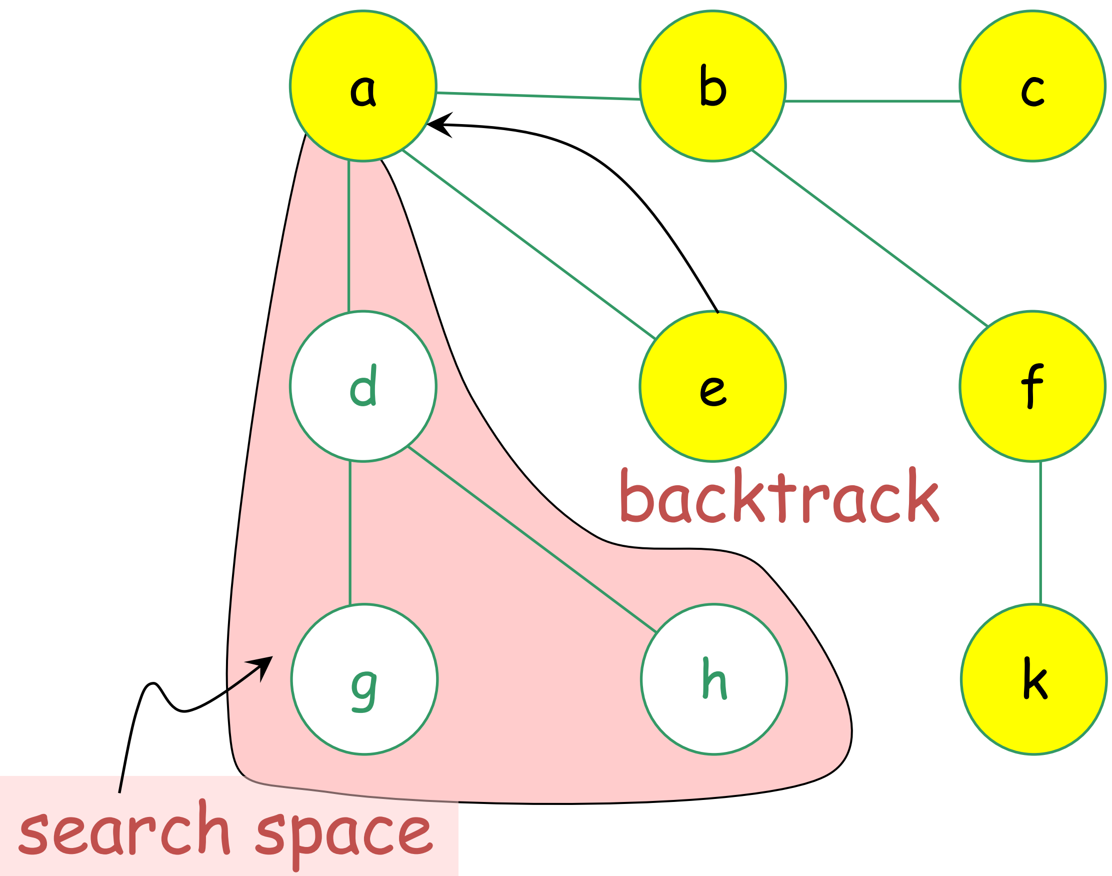
DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

Order of exploration
a, b, c, f, k, e



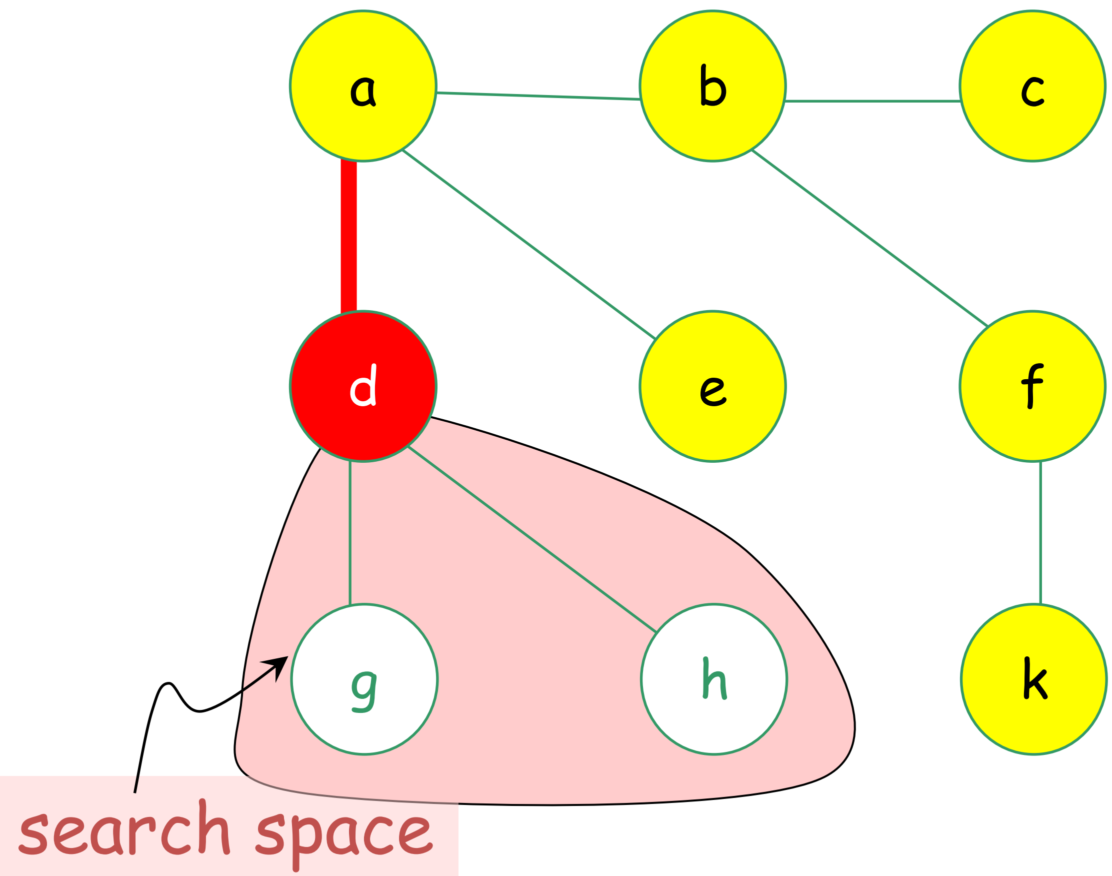
DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

Order of exploration
a, b, c, f, k, e, **d**

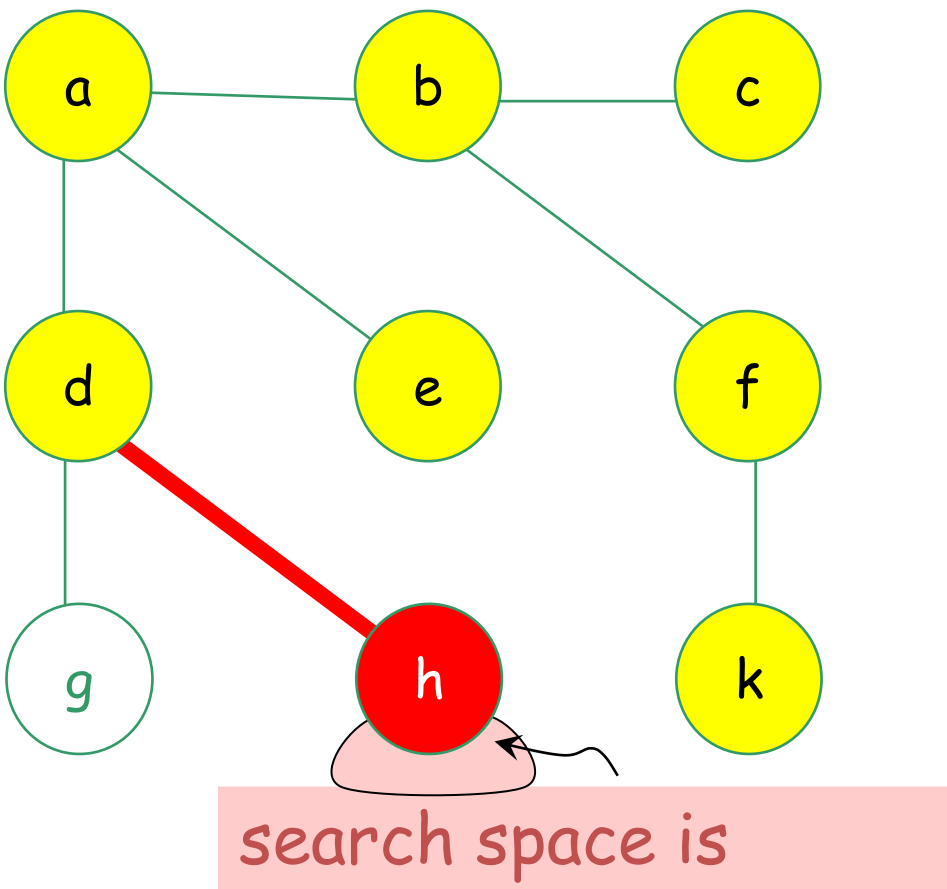


DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.



Order of exploration
a, b, c, f, k, e, d, **h**

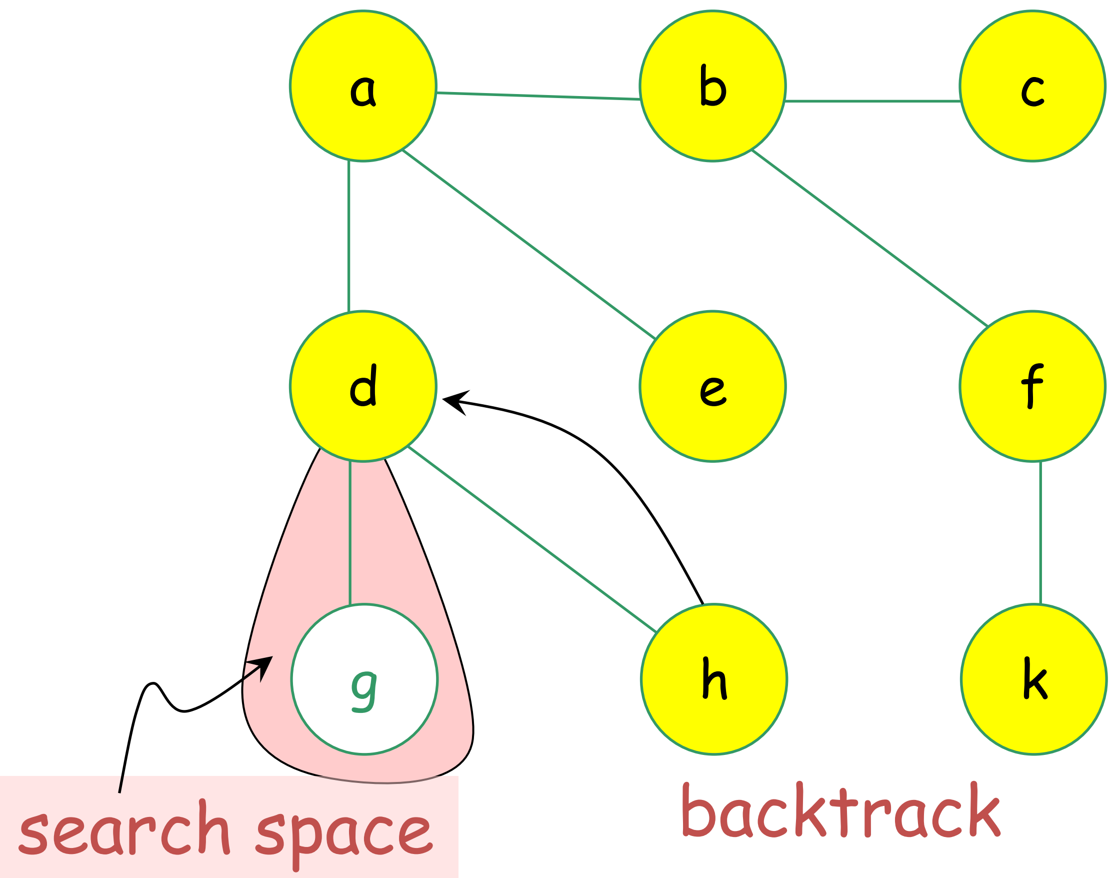
DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

Order of exploration
a, b, c, f, k, e, d, h



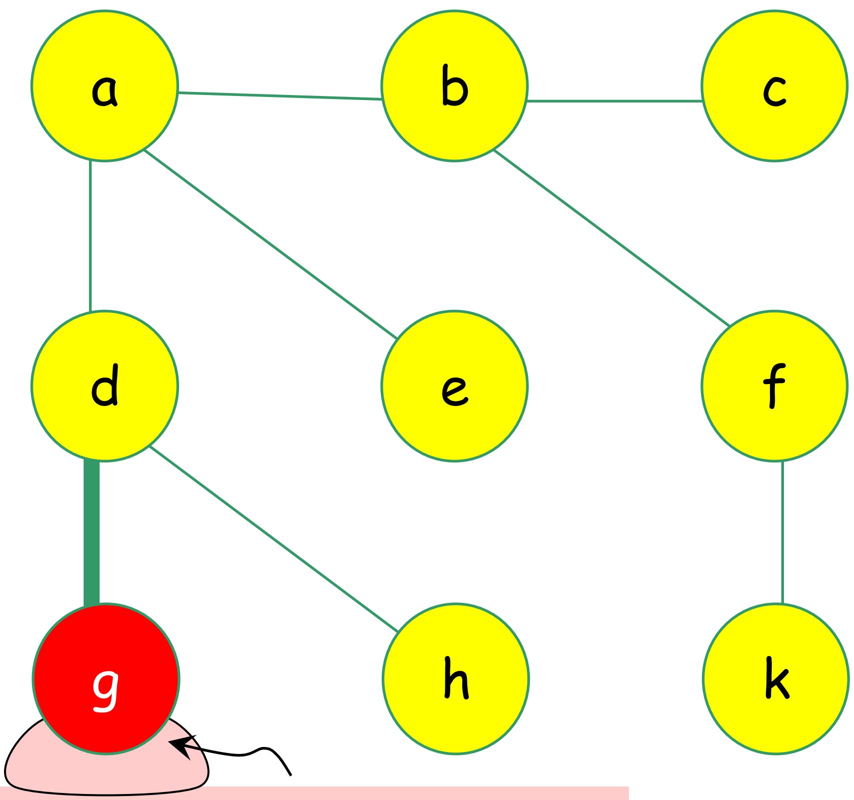
DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

Order of exploration
a, b, c, f, k, e, d, h, ***g***



search space is

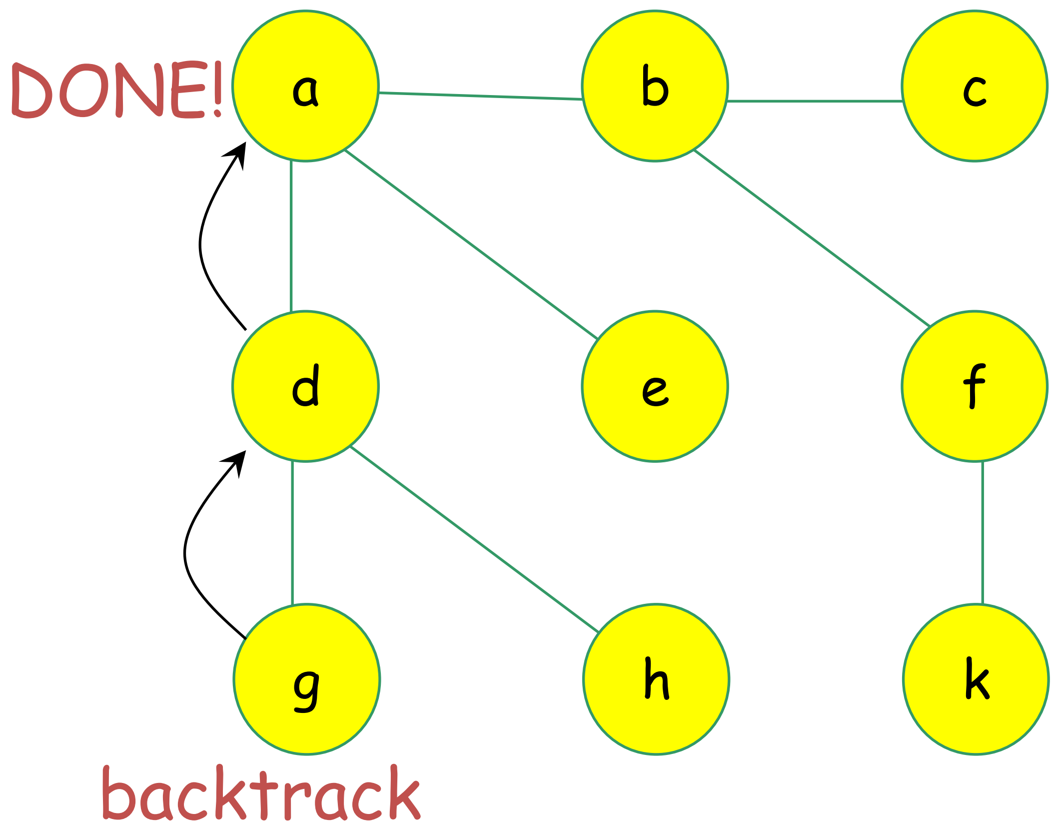
DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Edges are explored from the most recently discovered vertex, backtracks when finished

The source is a.

Order of exploration
a, b, c, f, k, e, d, h, g



DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

Depth-first search is another strategy for exploring a graph; it search **"deeper"** in the graph whenever possible.

- Edges are explored from the most recently discovered vertex ***v*** that still has unexplored edges leaving it.
- When all edges of ***v*** have been explored, the search ***"backtracks"*** to explore edges leaving the vertex from which ***v*** was discovered.

DFS – pseudo code (recursive)

```
Algorithm DFS(vertex v)
  visit v
  for each unvisited neighbor w of v do
  begin
     DFS(w)
  end
```

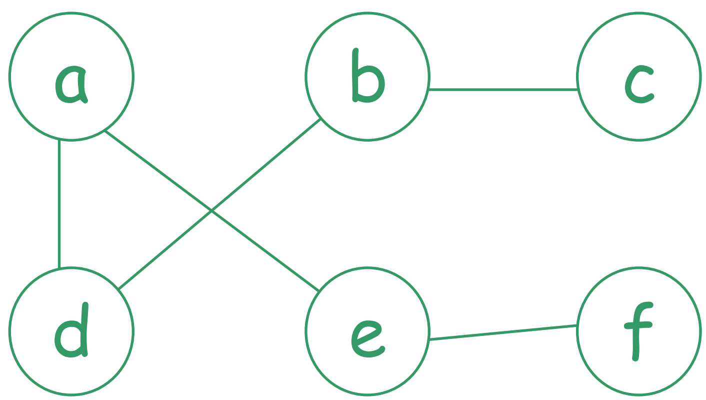
DFS – pseudo code (recursive)

```
Algorithm DFS(G)            //G=(V,E)
  for each v in V
     mark v with 0          //means v is not visited yet
  count = 0
  for each vertex in V do
     if v is marked with 0
        dfs(v)
```

```
dfs(v)
count = count +1
Mark v with count
  for each vertex w in Adj(v)
   do
    if w is marked with 0
       dfs(w)
```

Exercise

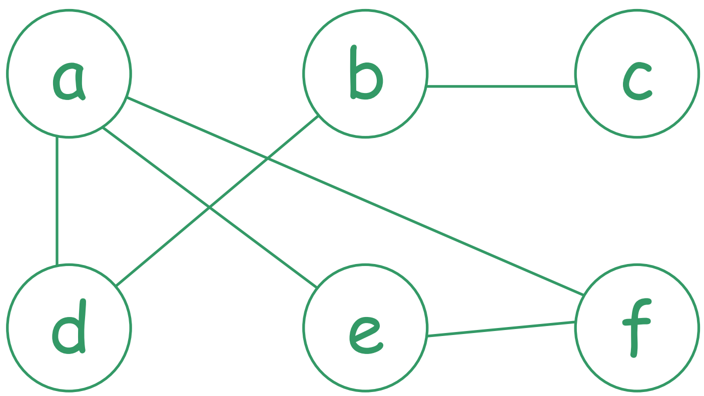
Apply **DFS** to the following graph starting from vertex **a** and list the order of exploration



Compare your results with slide #39

Exercise (2)

Apply **DFS** to the following graph starting from vertex **a** and list the order of exploration



Compare your results with slide #40

Learning outcomes

- ✓ Able to tell what is an undirected graph **and what is a directed graph**
  - ✓ Know how to represent a graph using matrix and list
- ✓ Understand what Euler path / circuit and able to determine whether such path / circuit exists in an undirected graph
- ✓ Able to apply BFS and DFS to traverse a graph
- ➢ Able to tell what a tree is