DTS203TC
Design and Analysis of Algorithms

Lecture 13: Graph Theory

Dr. Qi Chen
School of AI and Advanced Computing

Learning outcome

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

Acknowledgment: Some slides are adapted from ones by Prof. Prudence Wong

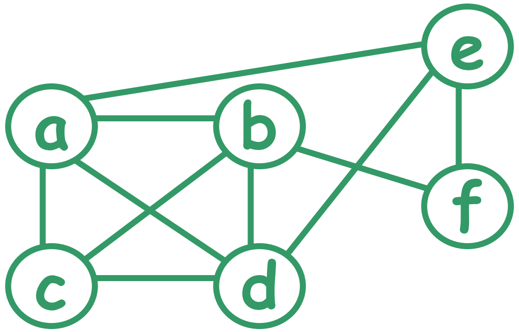
Graph

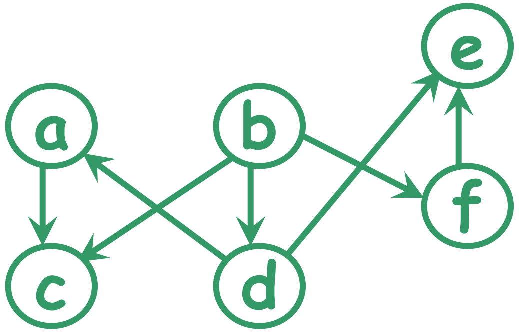
Graphs

introduced in the 18th century

- 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.)



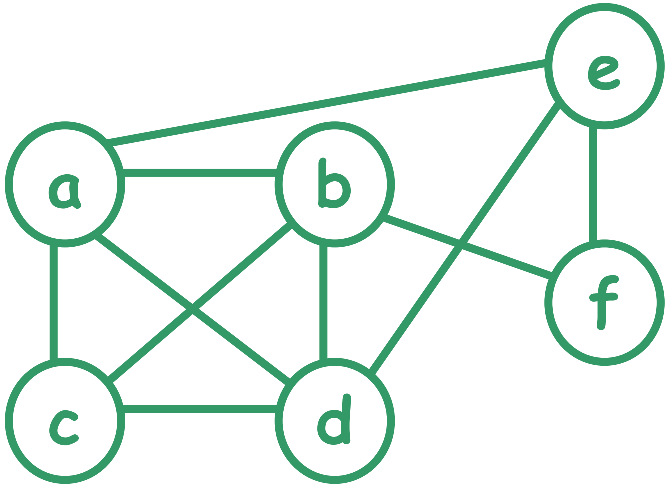


A ***directed*** graph G=(V,E) consists of … Each edge is an ***ordered*** pair of vertices. (E.g., (b,c) refer to an edge from b to c.)

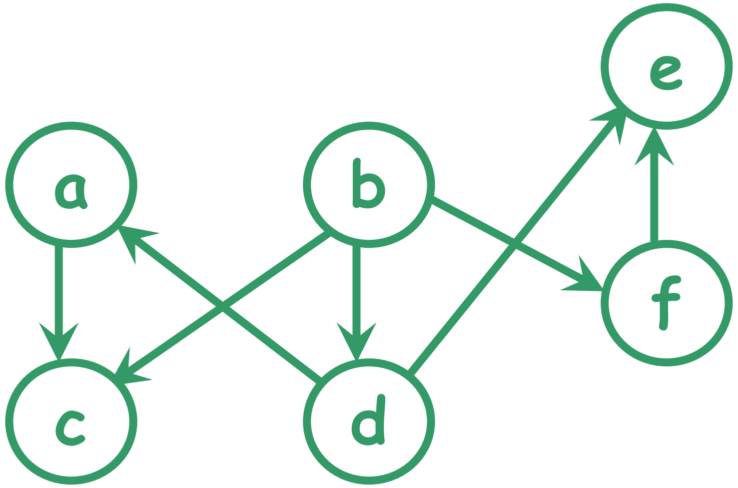
Modeling Facebook & Twitter?

Graphs

- Represent a set of interconnected objects



"friend" relationship
on Facebook



"follower" relationship
on Twitter

undirected graph

directed graph

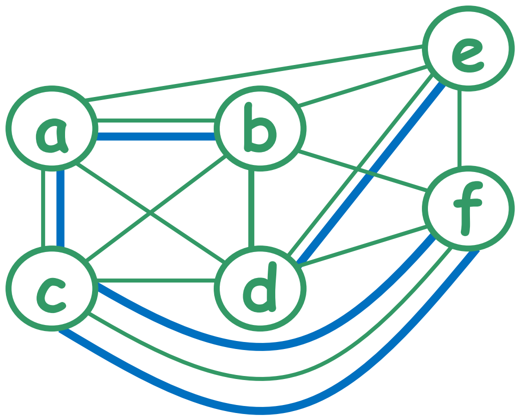
Applications of graphs

- 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 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.

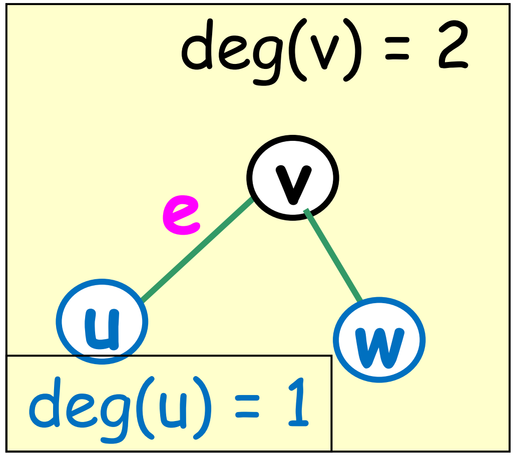
Reminder: An undirected graph G=(V,E) consists of a set of vertices V and a set of edges E. Each edge is an unordered pair of vertices.



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

Data Structure - Matrix

- 2-dimensional array
  - m-by-n matrix
    - m rows
    - n columns
  - $a_{i,j}$
    - row i, column j

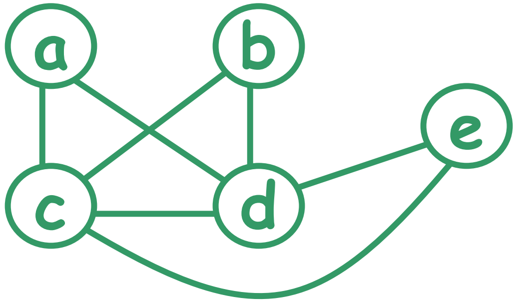
m-by-n matrix

$a_{i,j}$

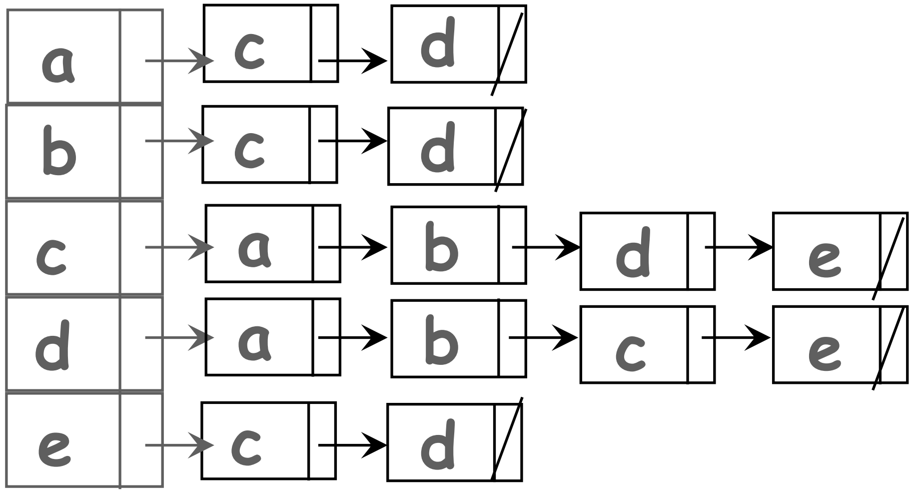
$$\begin{array}{cc} & \text{n columns} \\ \text{m rows} & \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{pmatrix} \end{array}$$

Adjacency matrix / list

- **Adjacency matrix** M for a simple **undirected** graph with n vertices is an **nxn** matrix
  - M(i, j) = 1 if vertex i and vertex j are adjacent
  - M(i, j) = 0 otherwise
- **Adjacency list:** each vertex has a list of vertices to which it is adjacent



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

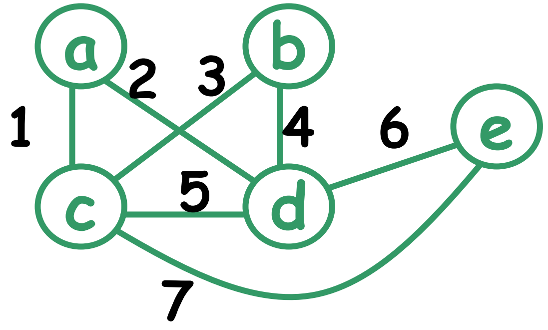


Representation (of undirected graphs)

- 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

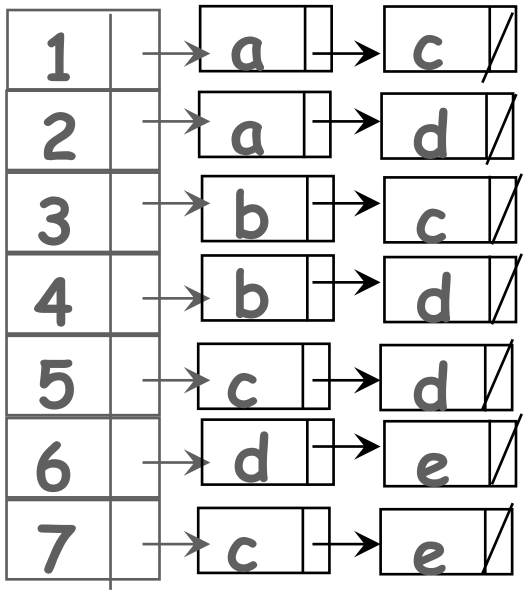
Incidence matrix / list

- **Incidence matrix** M for a simple **undirected** graph with n vertices and m edges is an **mxn** matrix
  - M(i, j) = 1 if edge i and vertex j are incidence
  - M(i, j) = 0 otherwise
- **Incidence list:** each edge has a list of vertices to which it is incident with



labels of edge
are edge number

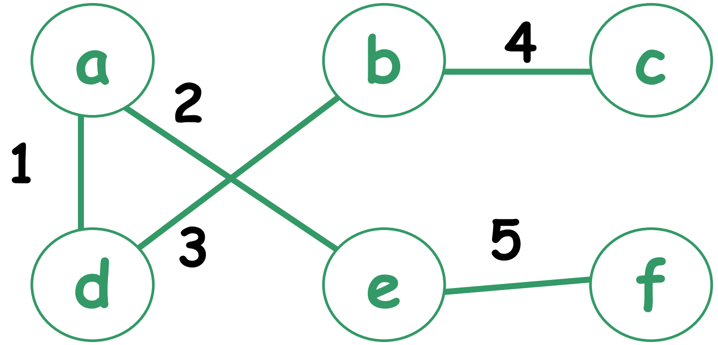
$$
\begin{array}{c} \\ 1\\ 2\\ 3\\ 4\\ 5\\ 6\\ 7 \end{array}
\begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\
\begin{pmatrix}
1 & 0 & 1 & 0 & 0\\
1 & 0 & 0 & 1 & 0\\
0 & 1 & 1 & 0 & 0\\
0 & 1 & 0 & 1 & 0\\
0 & 0 & 1 & 1 & 0\\
0 & 0 & 0 & 1 & 1\\
0 & 0 & 1 & 0 & 1
\end{pmatrix}
\end{array}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



Exercise

- Give the adjacency matrix and incidence matrix of the following graph

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



labels of edge
are edge number

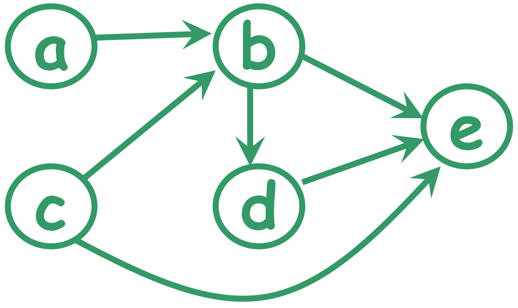
$$
\begin{array}{c}
\begin{array}{cccccc} a & b & c & d & e & f \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{pmatrix}
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1
\end{pmatrix}
\end{array}
$$

Directed graph

Directed graph

- Given a directed graph G, a vertex ***a*** is said to be **connected to** a vertex ***b*** if there is a path from ***a*** to ***b***.
  - E.g., G represents the routes provided by a certain airline. That means, a vertex represents a city and an edge represents a flight from a city to another city. Then we may ask question like: Can we fly from one city to another?

> Reminder: A directed graph G=(V,E) consists of a set of vertices V and a set of edges E. Each edge is an ordered pair of vertices.

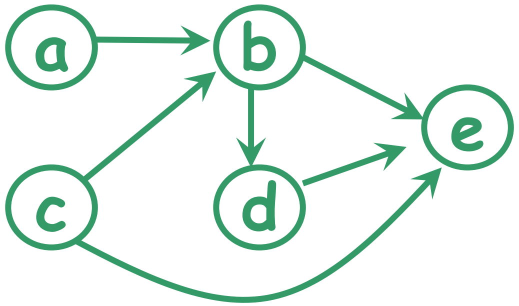


E = { (a,b), (b,d), (b,e), (c,b), (c,e), (d,e) }

N.B. (a,b) is in E, but (b,a) is NOT

In/Out degree (in directed graphs)

- The **in-degree** of a vertex ***v*** is the number of edges ***leading to*** the vertex ***v***.
- The **out-degree** of a vertex ***v*** is the number of edges ***leading away*** from the vertex ***v***.



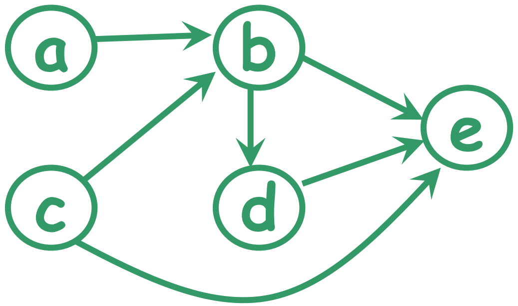
v	in-deg(v)	out-deg(v)	
a	0	1	
b	2	2	
c	0	2	
d	1	1	
e	3	0	
sum:	6	6	Always equal?

Representation (of directed graphs)

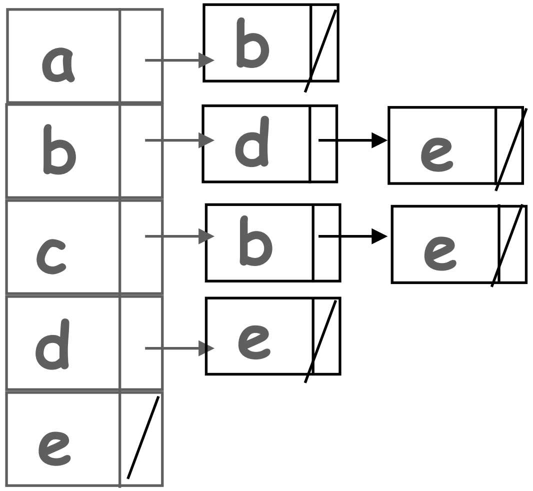
- Similar to undirected graph, a directed graph can be represented by <u>adjacency matrix</u>, <u>adjacency list</u>, <u>incidence matrix</u> or <u>incidence list</u>.

Adjacency matrix / list

- **Adjacency matrix** M for a **directed** graph with n vertices is an **nxn** matrix
  - M(i, j) = 1 if (i,j) is an edge
  - M(i, j) = 0 otherwise
- **Adjacency list:**
  - each vertex ***u*** has a list of vertices pointed to by an edge leading away from ***u***

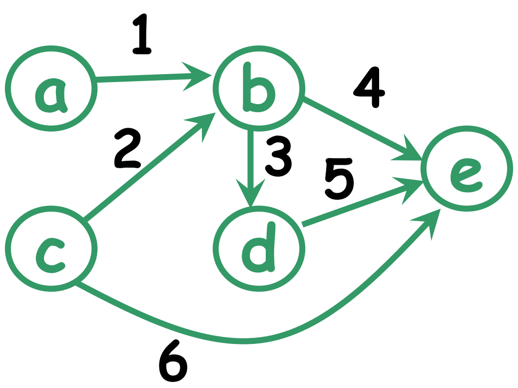


$$
\begin{array}{c c}
 & \begin{array}{ccccc} a & b & c & d & e \end{array} \\
\begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{array}
$$

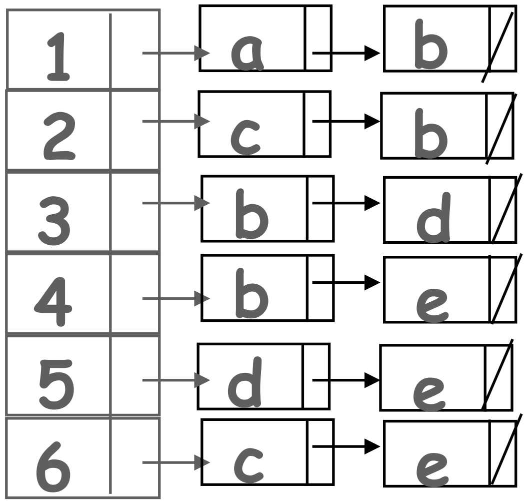


Incidence matrix / list

- **Incidence matrix** M for a **directed** graph with n vertices and m edges is an **mxn** matrix
  - M(i, j) = 1 if edge i is leading away from vertex j
  - M(i, j) = -1 if edge i is leading to vertex j
- **Incidence list:** each edge has a list of two vertices (leading away is 1st and leading to is 2nd)



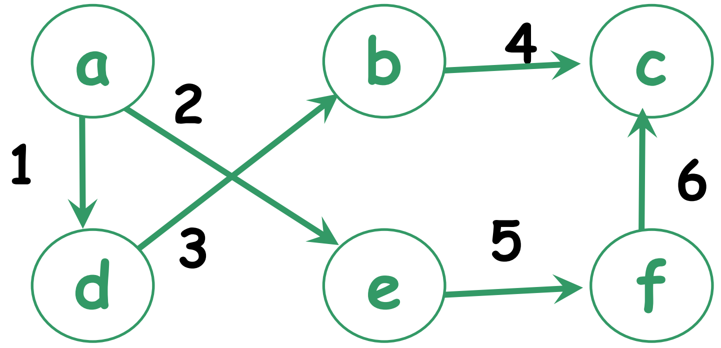
$$
\begin{array}{c}
 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6
\end{array}
\begin{array}{c}
\begin{array}{ccccc} a & b & c & d & e \end{array} \\
\begin{pmatrix}
1 & -1 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 \\
0 & 1 & 0 & 0 & -1 \\
0 & 0 & 0 & 1 & -1 \\
0 & 0 & 1 & 0 & -1
\end{pmatrix}
\end{array}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



Exercise

- Give the adjacency matrix and incidence matrix of the following graph

$$\begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline a & 0 & 0 & 0 & 1 & 1 & 0 \\ b & 0 & 0 & 1 & 0 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & 0 & 0 \\ d & 0 & 1 & 0 & 0 & 0 & 0 \\ e & 0 & 0 & 0 & 0 & 0 & 1 \\ f & 0 & 0 & 1 & 0 & 0 & 0 \end{array}$$



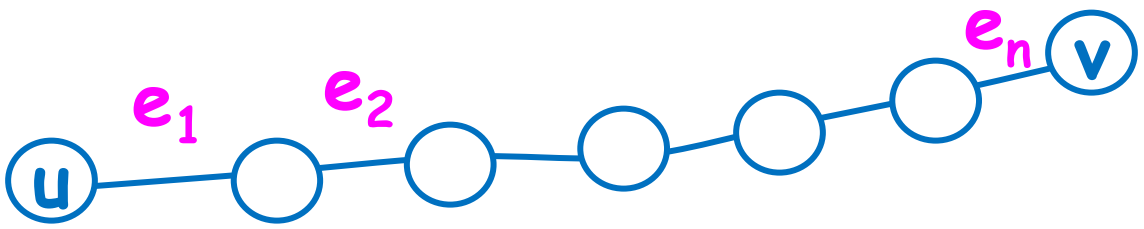
labels of edge
are edge number

$$\begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline 1 & 1 & 0 & 0 & -1 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & -1 & 0 \\ 3 & 0 & -1 & 0 & 1 & 0 & 0 \\ 4 & 0 & 1 & -1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 & -1 \\ 6 & 0 & 0 & -1 & 0 & 0 & 1 \end{array}$$

Euler circuit

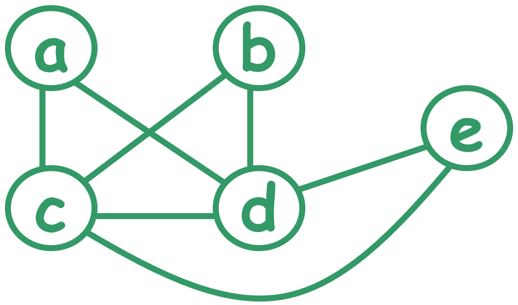
Paths, circuits (in undirected graphs)

- In an undirected graph, a **path** from a vertex ***u*** to a vertex ***v*** is a sequence of edges $e_1 = \{u, x_1\}$, $e_2 = \{x_1, x_2\}$, $\ldots e_n = \{x_{n-1}, v\}$, where **$n \geq 1$**.
- 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).

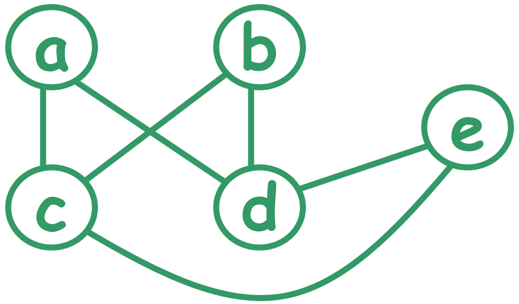


Euler circuit

- 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

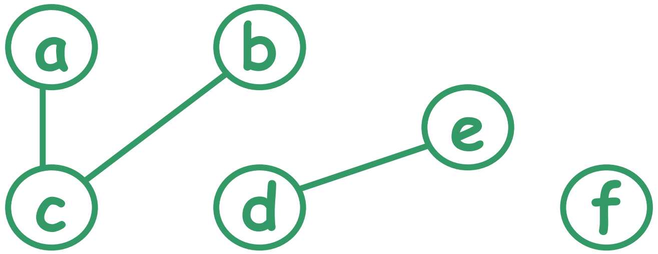


no Euler circuit

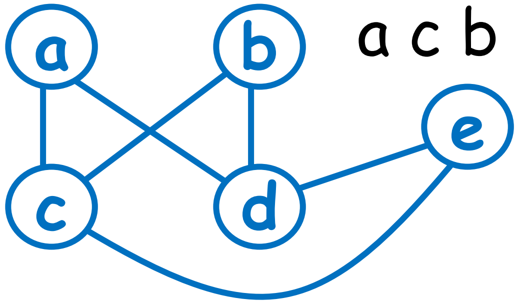
How to determine whether there is an Euler circuit in a graph?

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.



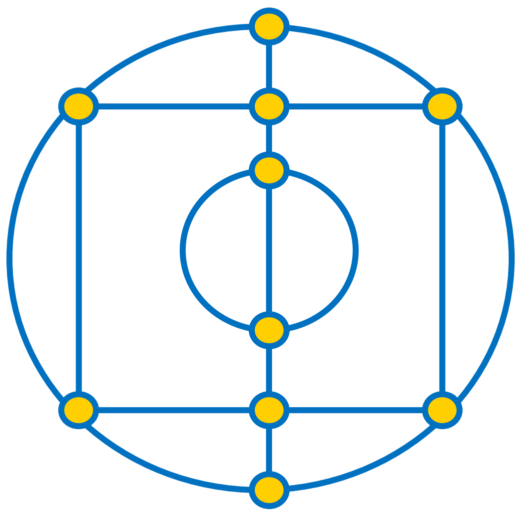
Even if the graph is connected, there may be no Euler circuit either.

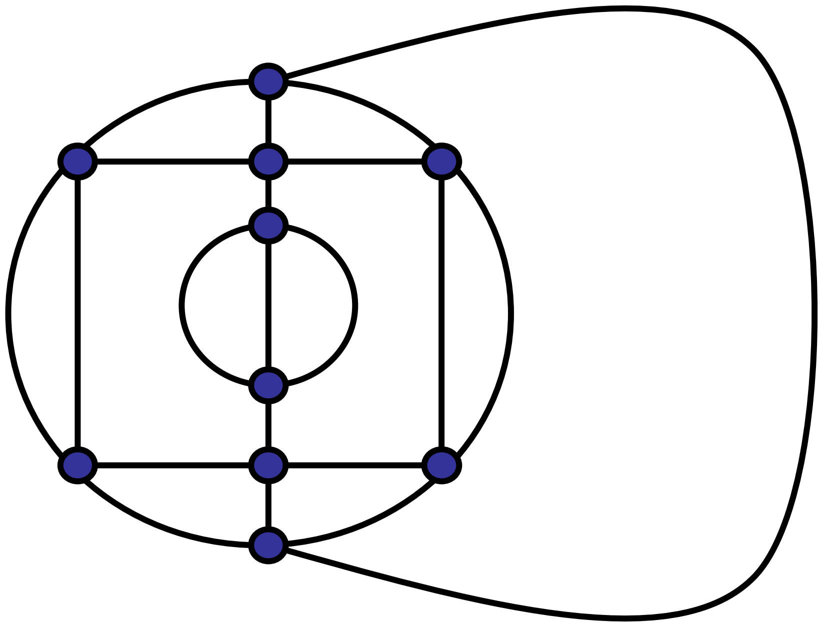


a c b d e c b d a

Necessary and sufficient condition

- Let G be a connected graph.
- **Lemma:** G contains an Euler circuit if and only if degree of every vertex is **even**.

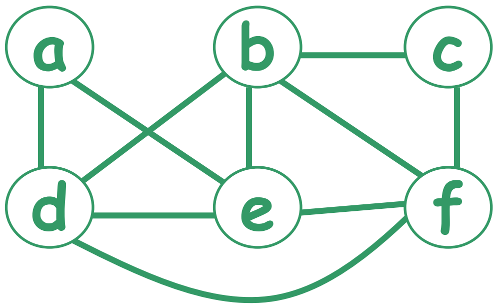


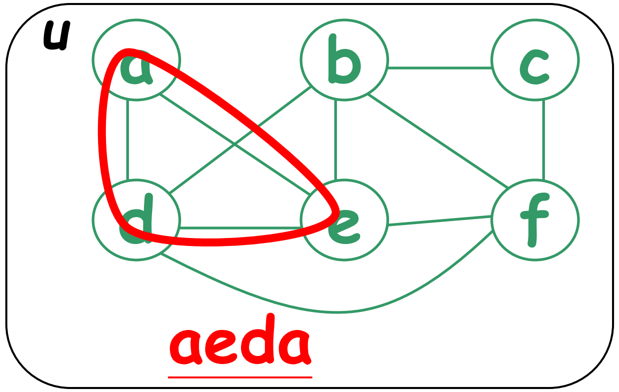


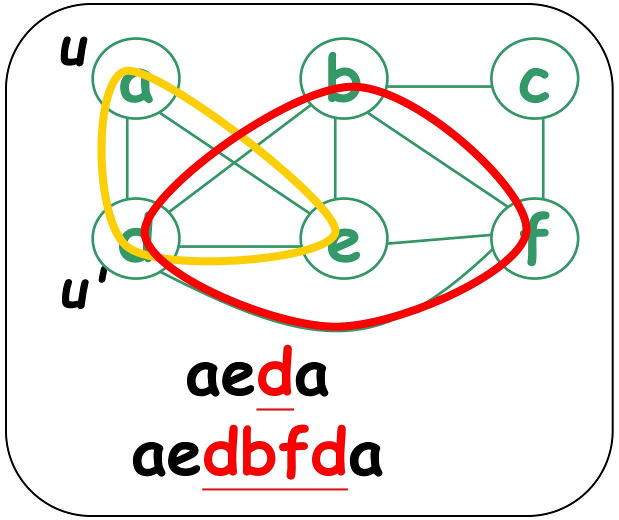
Necessary and sufficient condition

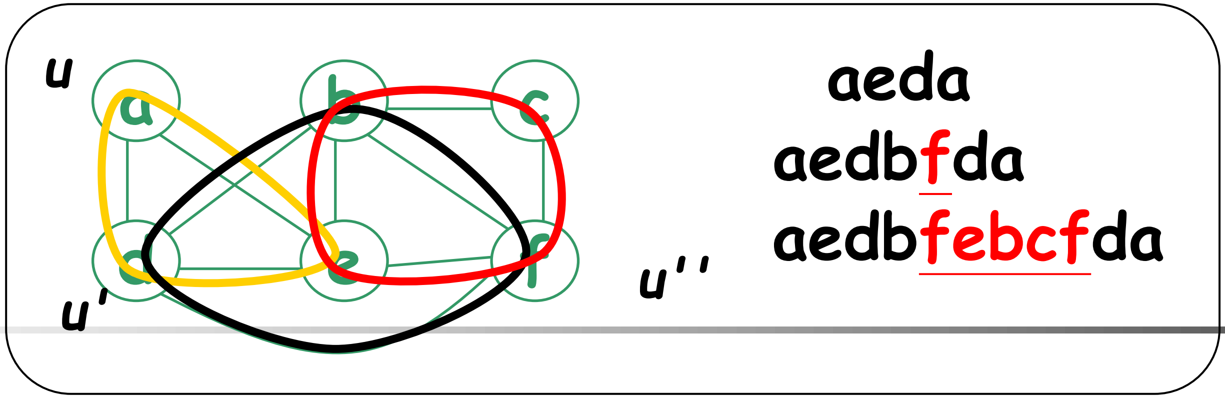
How to find it?

- Let G be a connected graph.
- **Lemma:** G contains an Euler circuit if and only if degree of every vertex is **even**.









Hamiltonian circuit

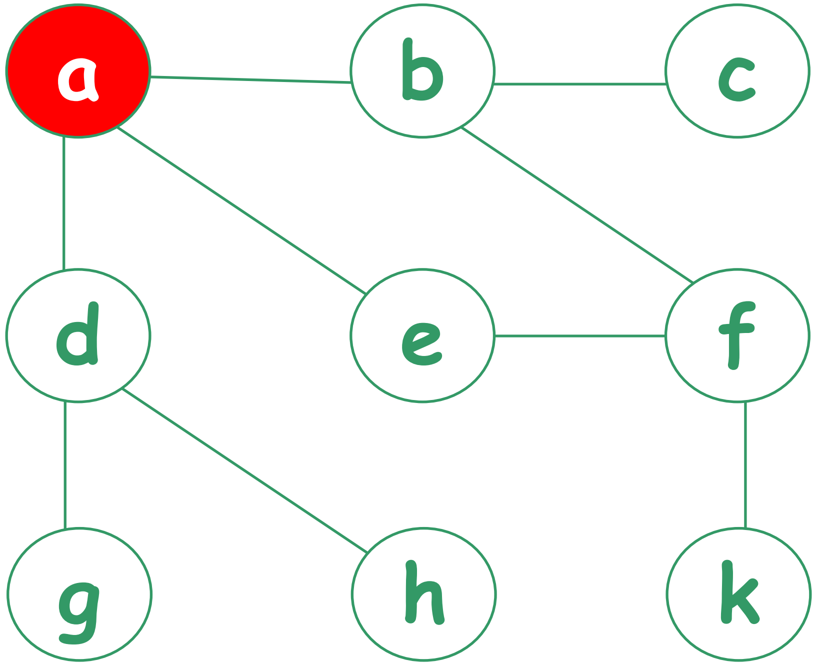
- Let G be an undirected graph.
- A **Hamiltonian circuit** is a circuit containing **every vertex** of G **exactly once**.
- Note that a Hamiltonian circuit may NOT visit all edges.
- Unlike the case of Euler circuits, determining whether a graph contains a Hamiltonian circuit is a very ***difficult*** problem. (NP-hard)

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.

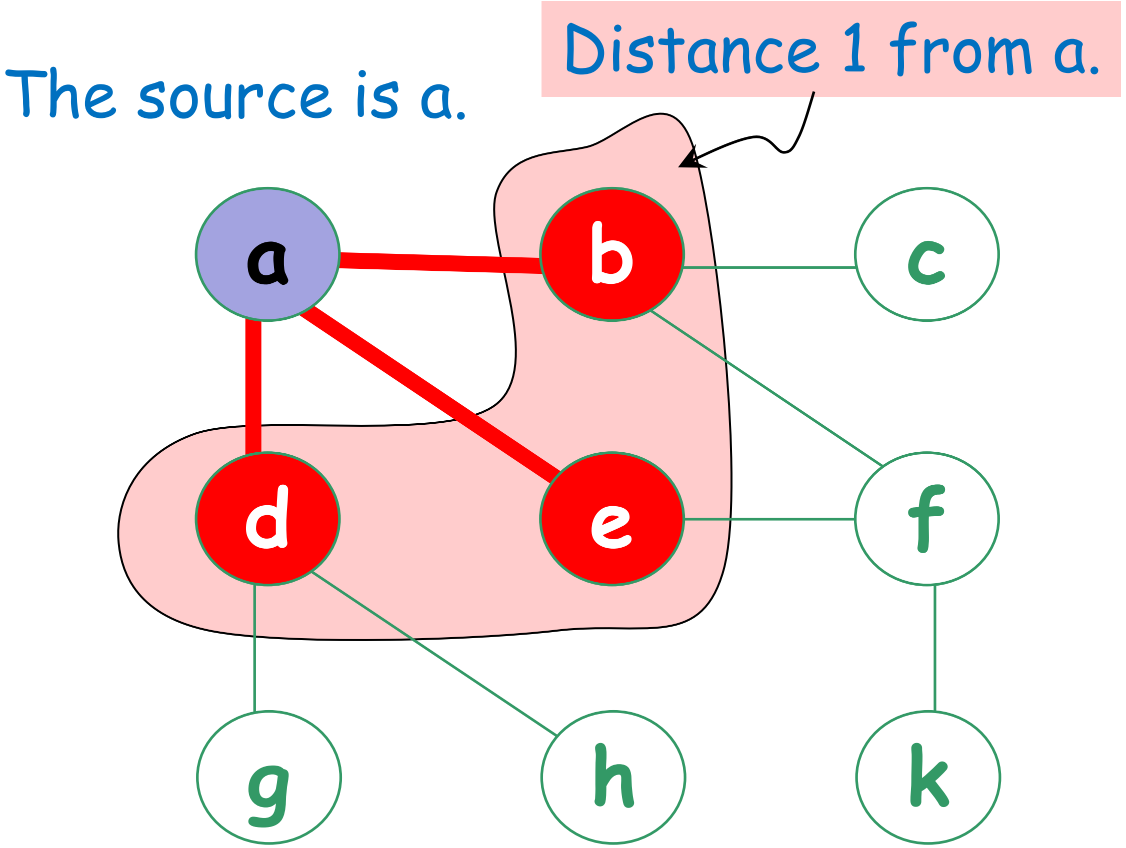


Order of exploration

a,

Breadth First Search (BFS)

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

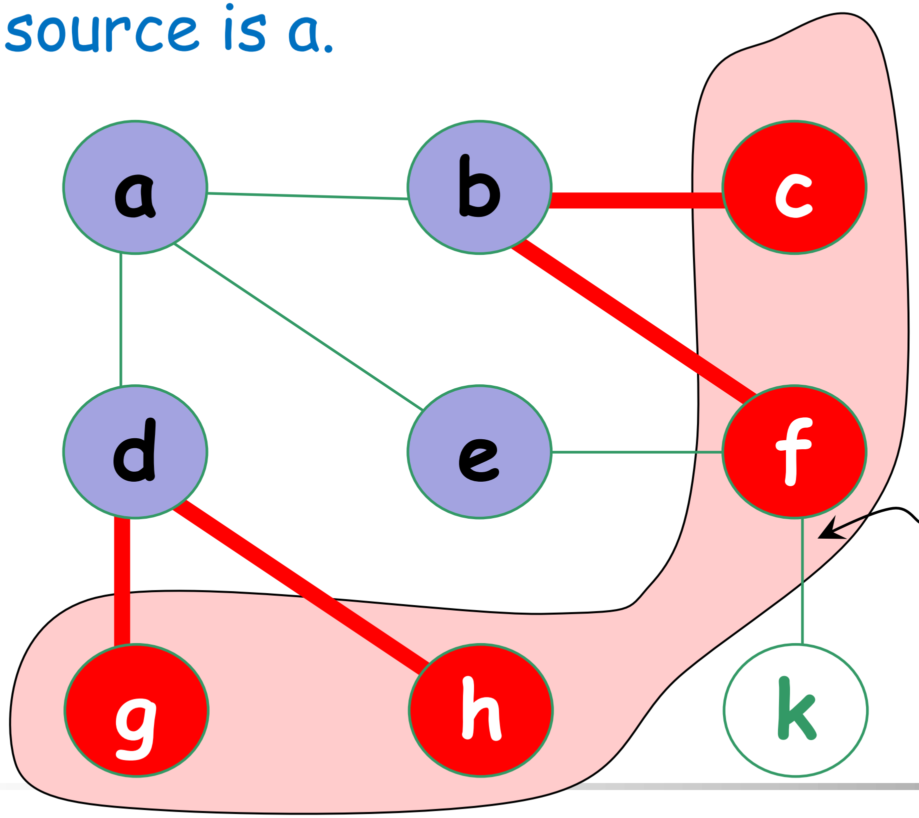


Order of exploration
a, b, e, d

Breadth First Search (BFS)

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

The source is a.



Distance 2 from a.

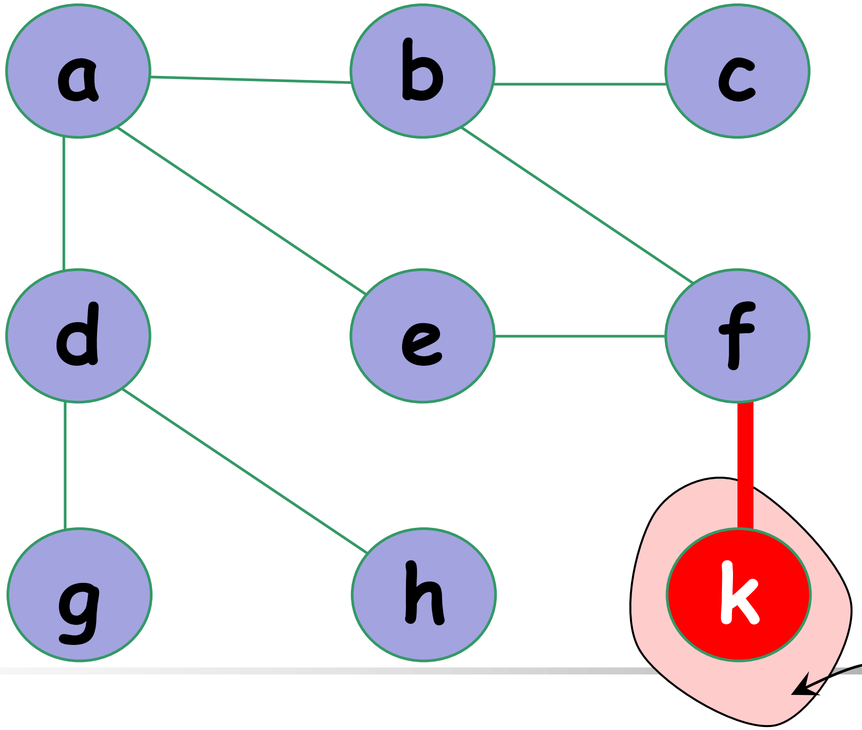
Order of exploration
a, b, e, d, c, f, h, g

Breadth First Search (BFS)

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

The source is a.

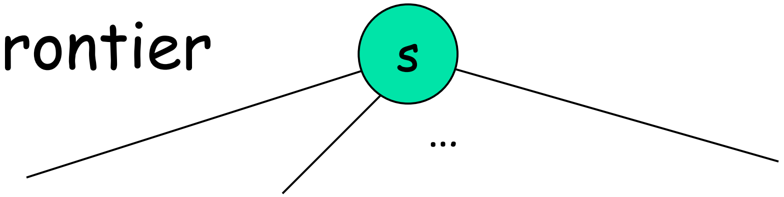
Order of exploration
a, b, e, d, c, f, h, g, k



Distance 3 from a.

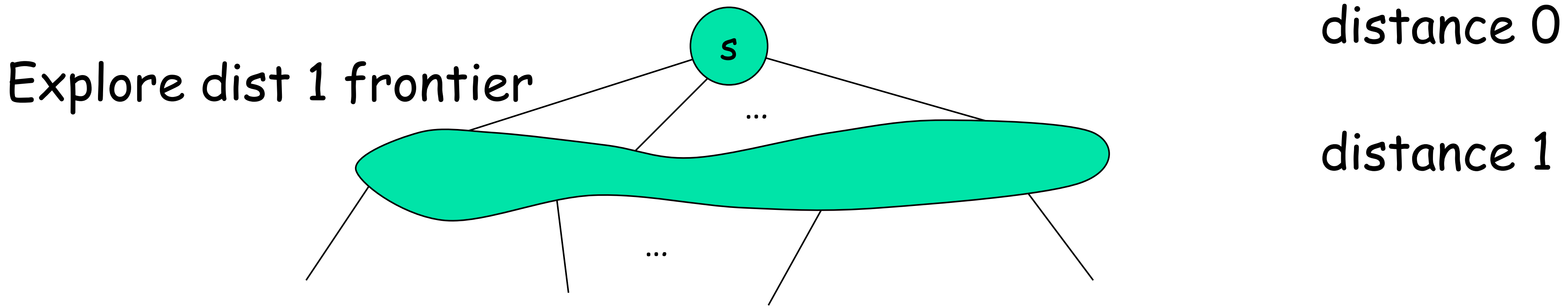
In general (BFS)

Explore dist 0 frontier

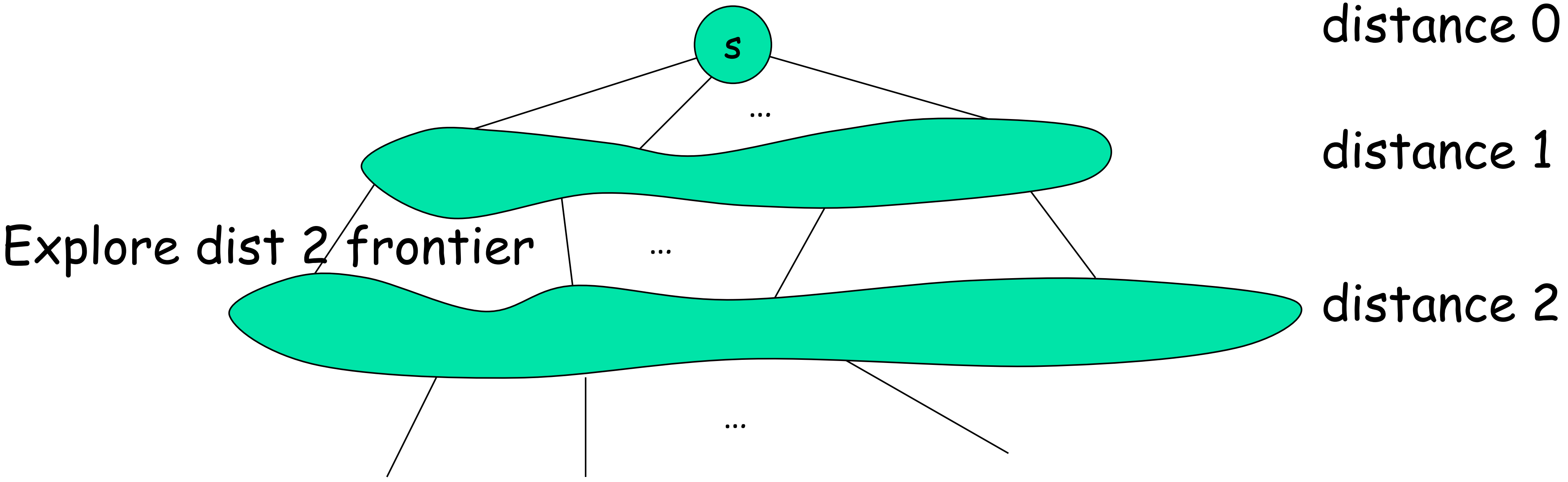


distance 0

In general (BFS)



In general (BFS)

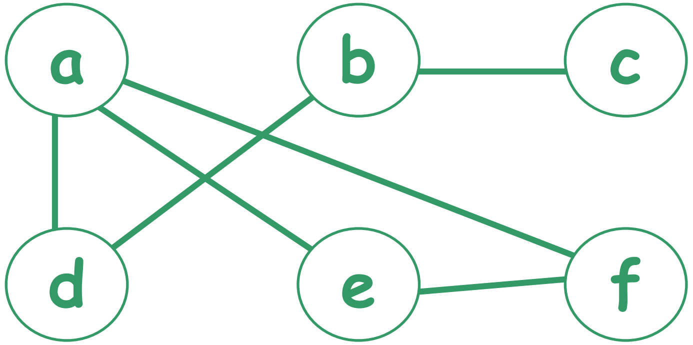


Breadth First Search (BFS)

- 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***.

Exercise – BFS

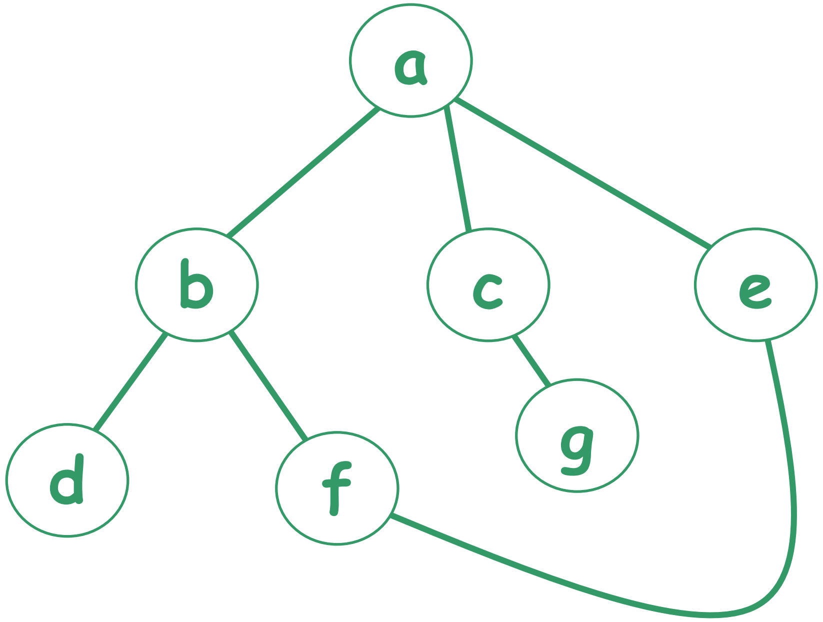
- Apply **BFS** to the following graph starting from vertex **a** and list the order of exploration



a, d, e, f, b, c

Exercise (2) – BFS

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



a, b, c, e, d, f, g

a, b, e, c, d, f, g

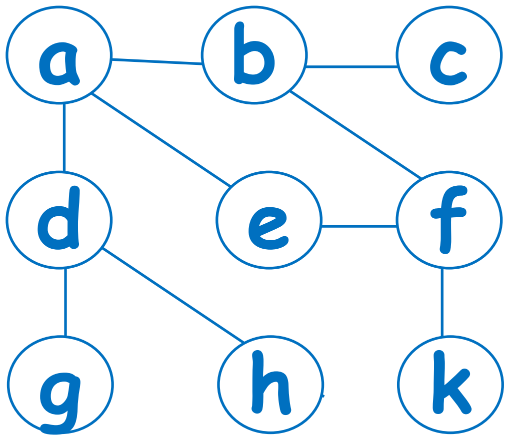
a, c, e, b, g, f, d

…

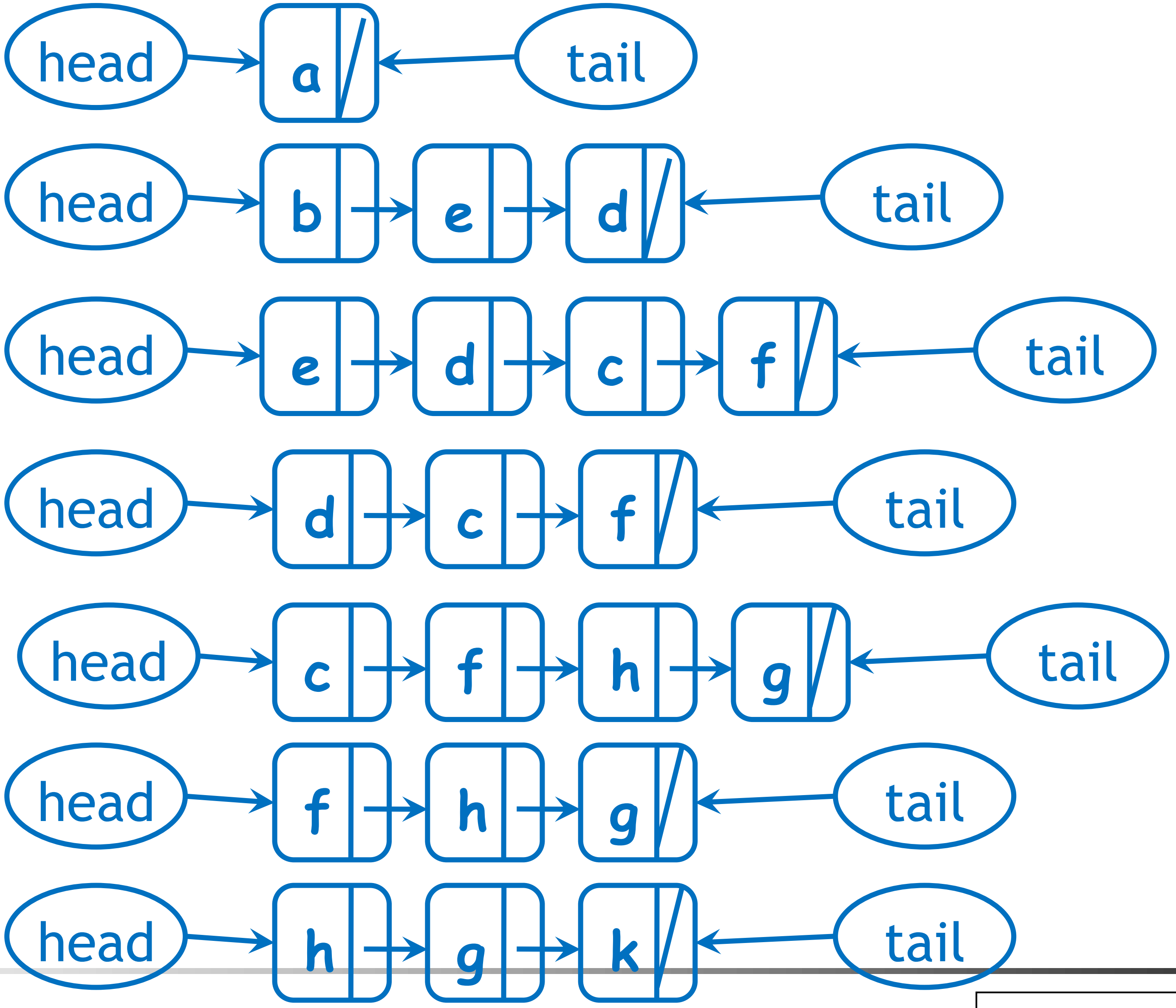
BFS – Pseudo code

```
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 using linked list



a, b, e, d, c, f, h, g, k



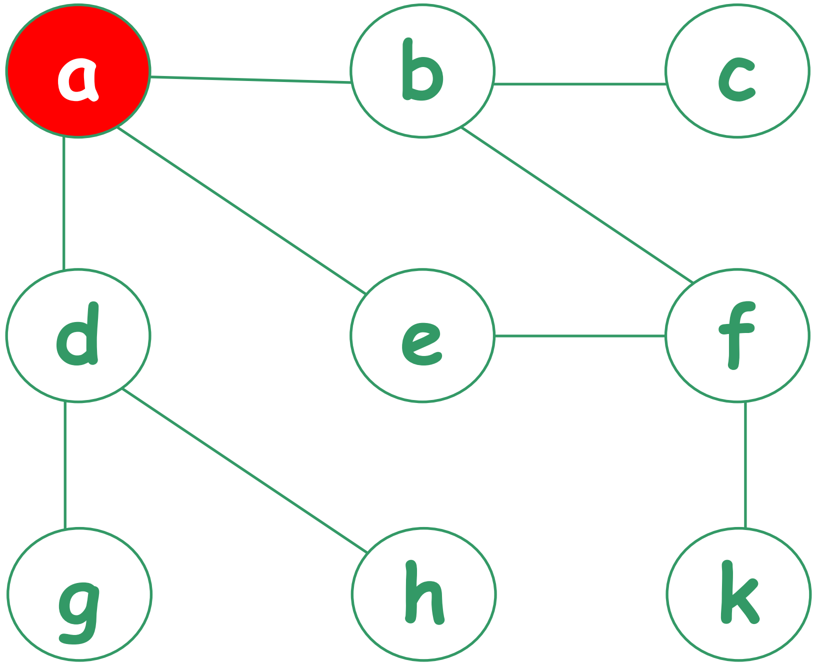
& so on …

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

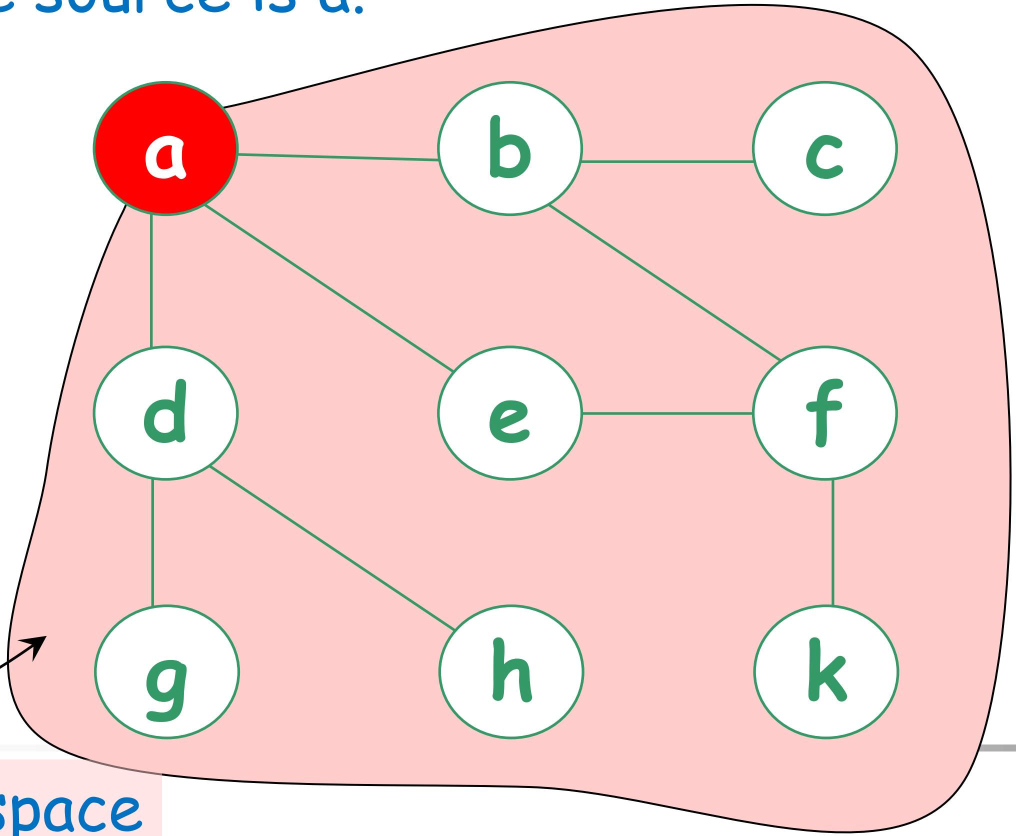
a,

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.



search space

Order of exploration

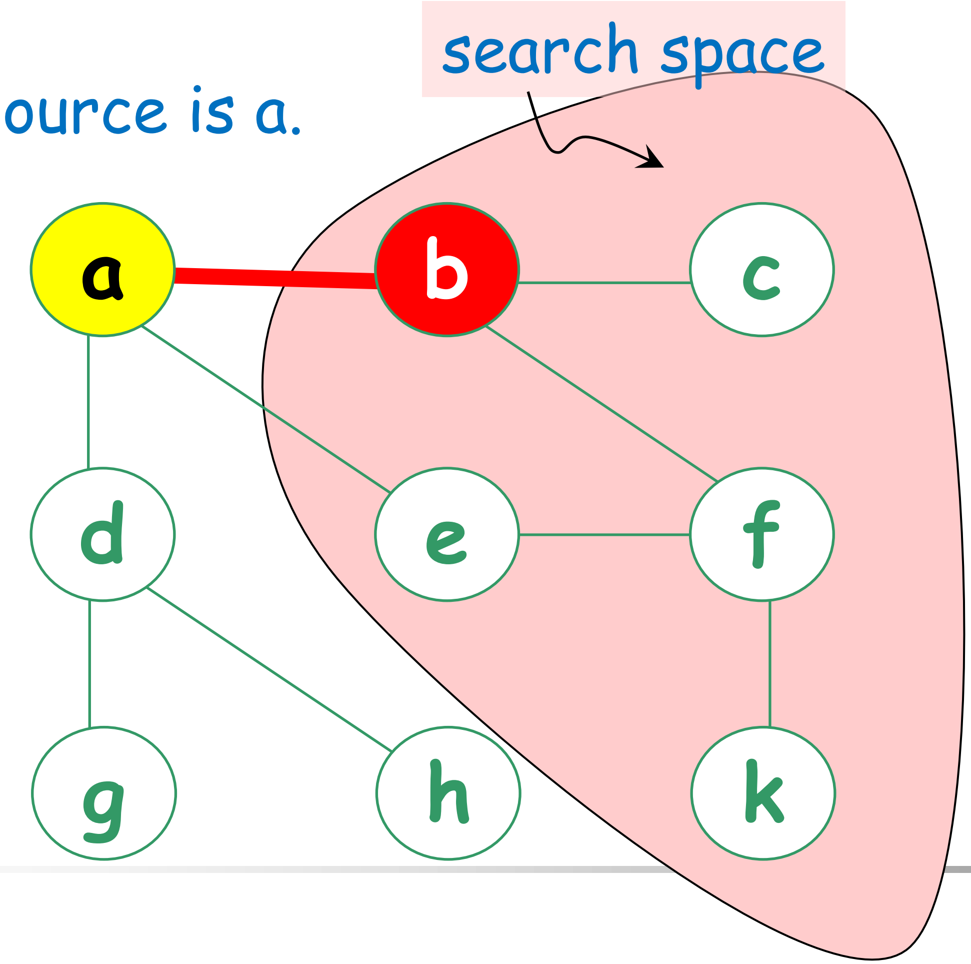
a,

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.

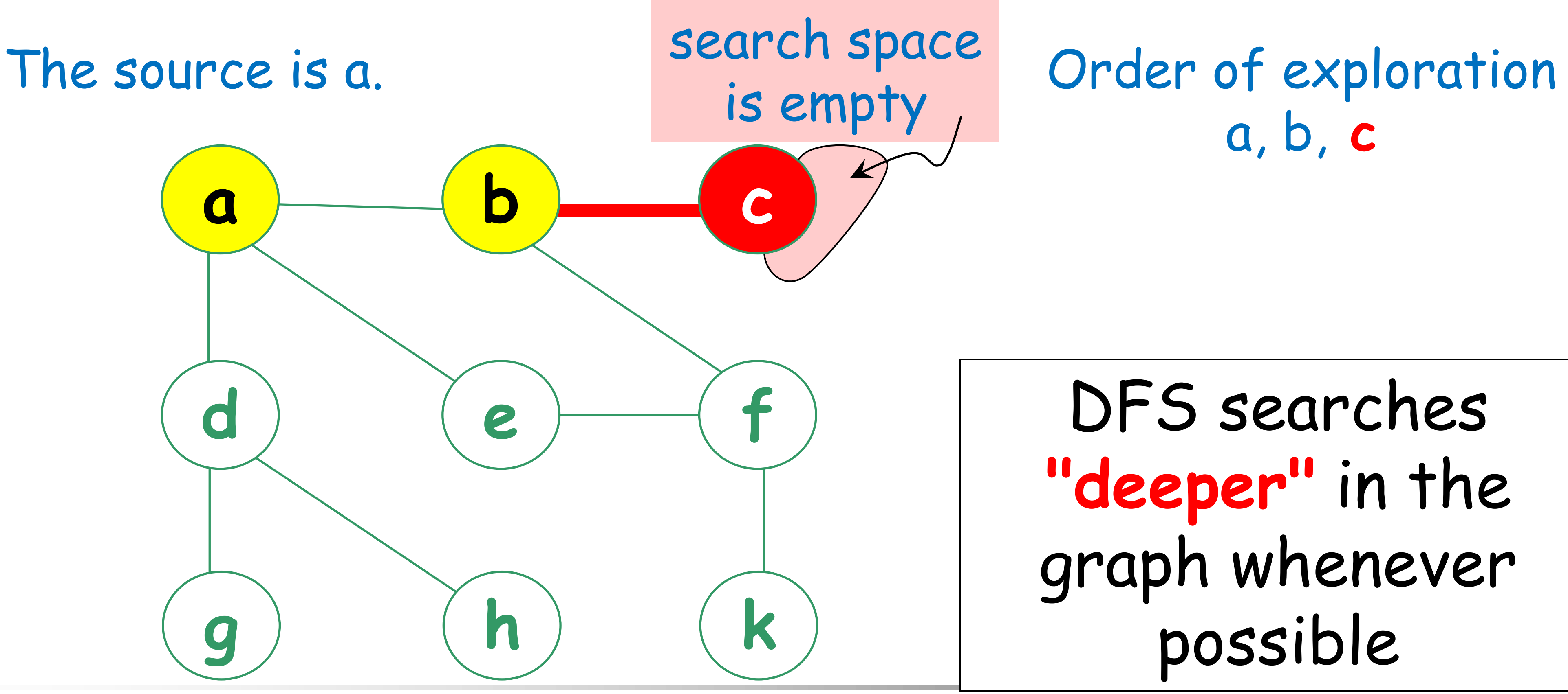


Order of exploration
a, b

DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

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



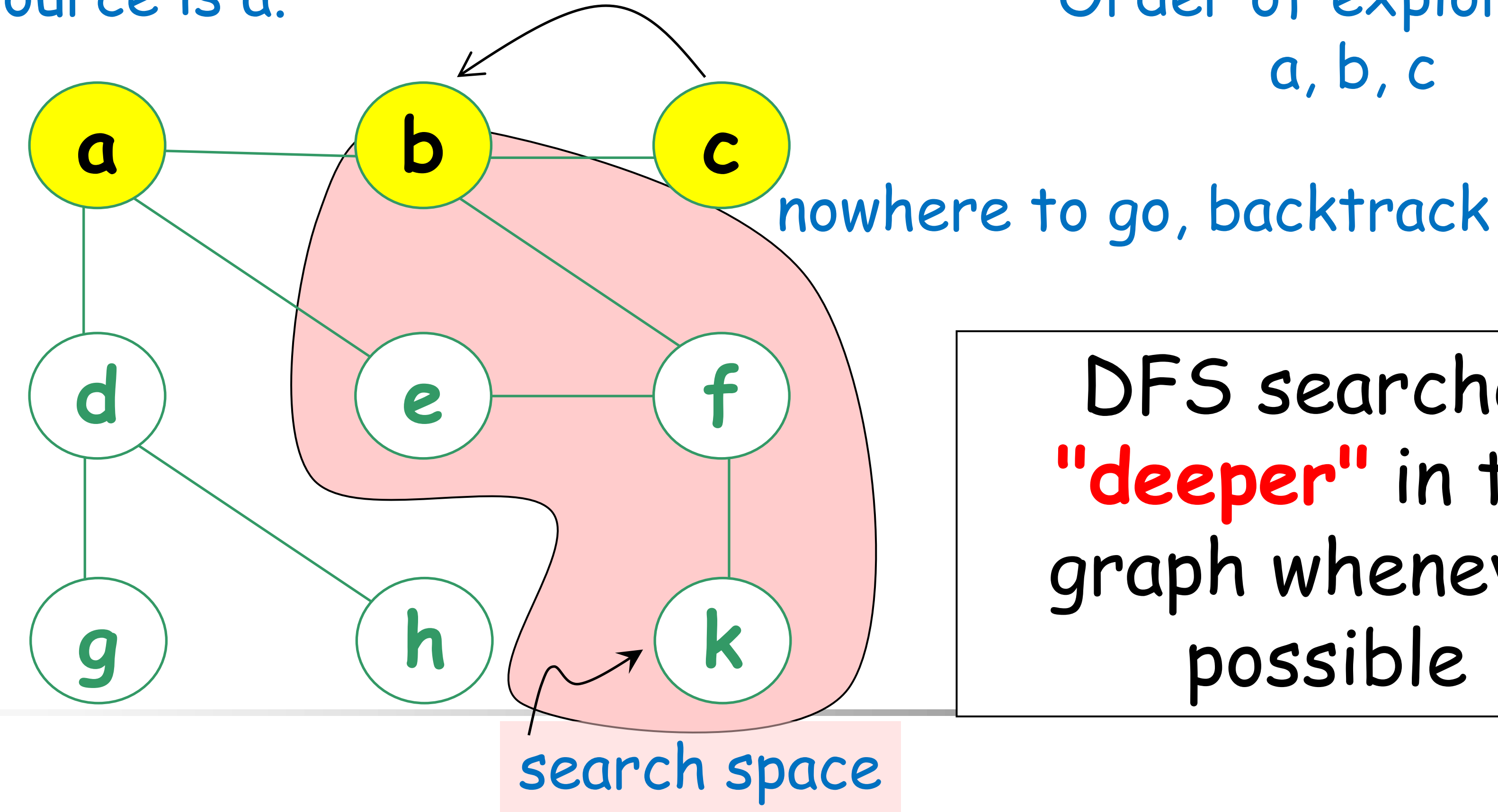
DFS searches "deeper" in the graph whenever possible

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



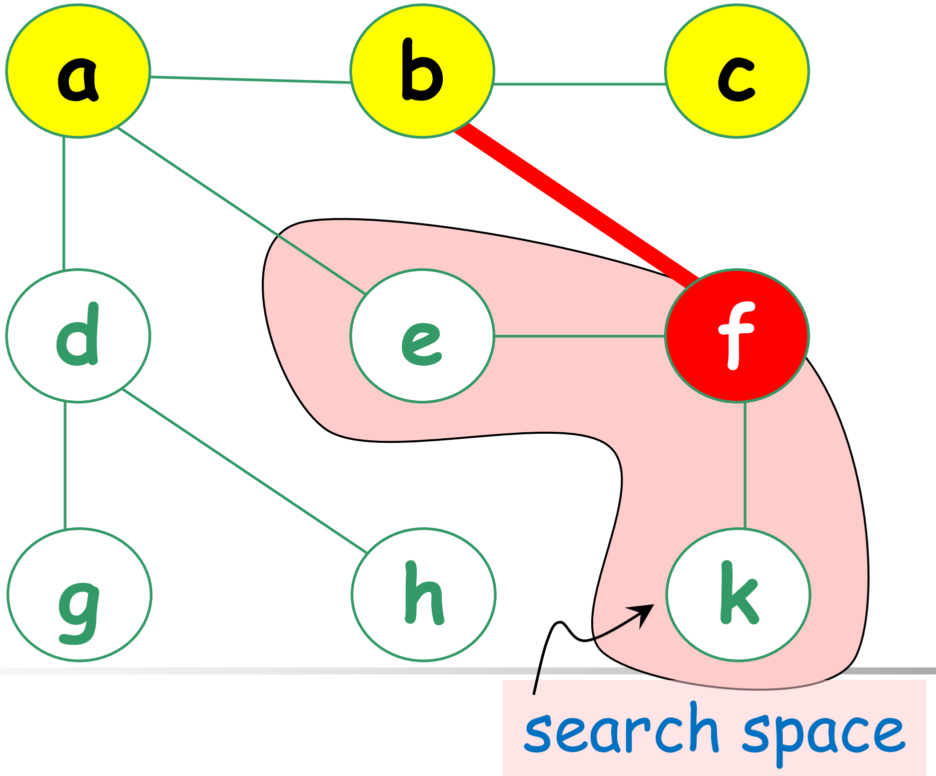
DFS searches **"deeper"** in the graph whenever possible

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



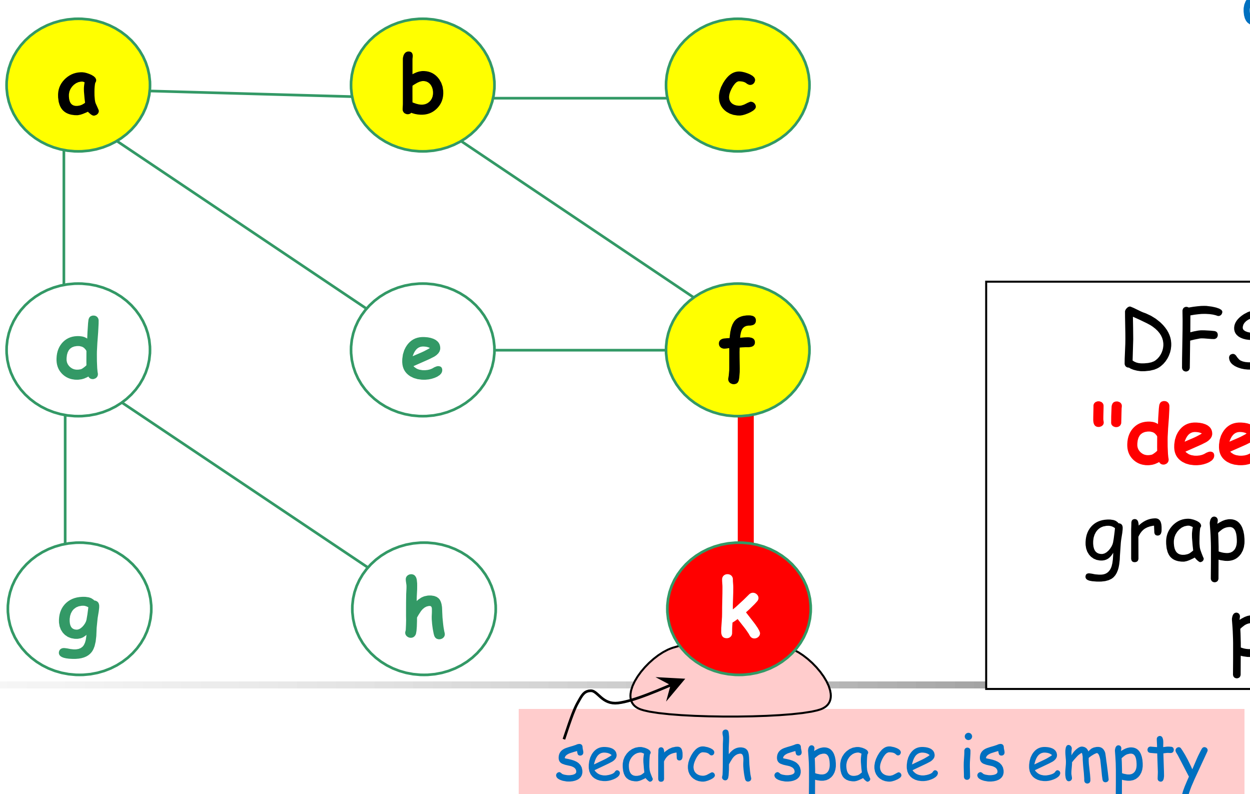
DFS searches "deeper" in the graph whenever possible

Depth First Search (DFS)

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

The source is a.

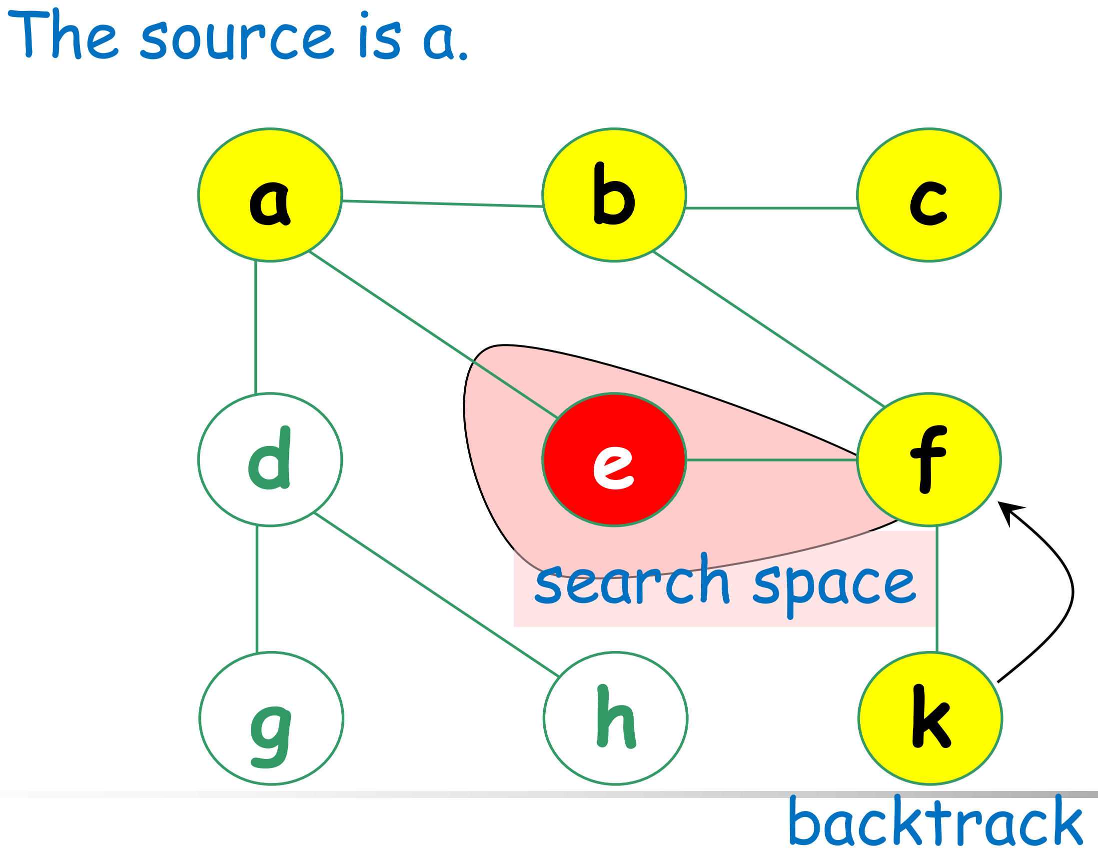
Order of exploration
a, b, c, f, **k**



DFS searches **"deeper"** in the graph whenever possible

Depth First Search (DFS)

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

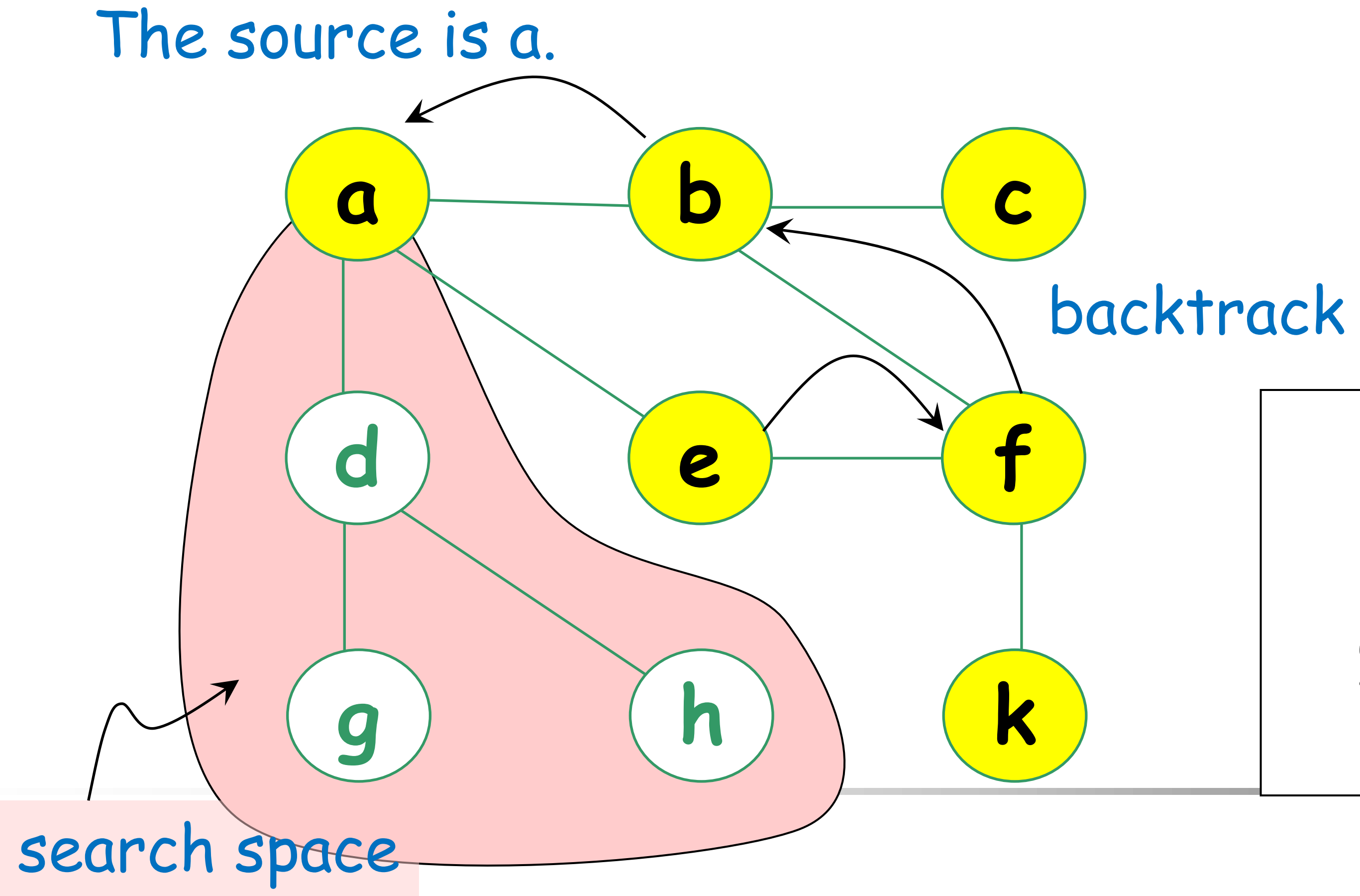


Order of exploration
a, b, c, f, k, e

DFS searches "**deeper**" in the graph whenever possible

Depth First Search (DFS)

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



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

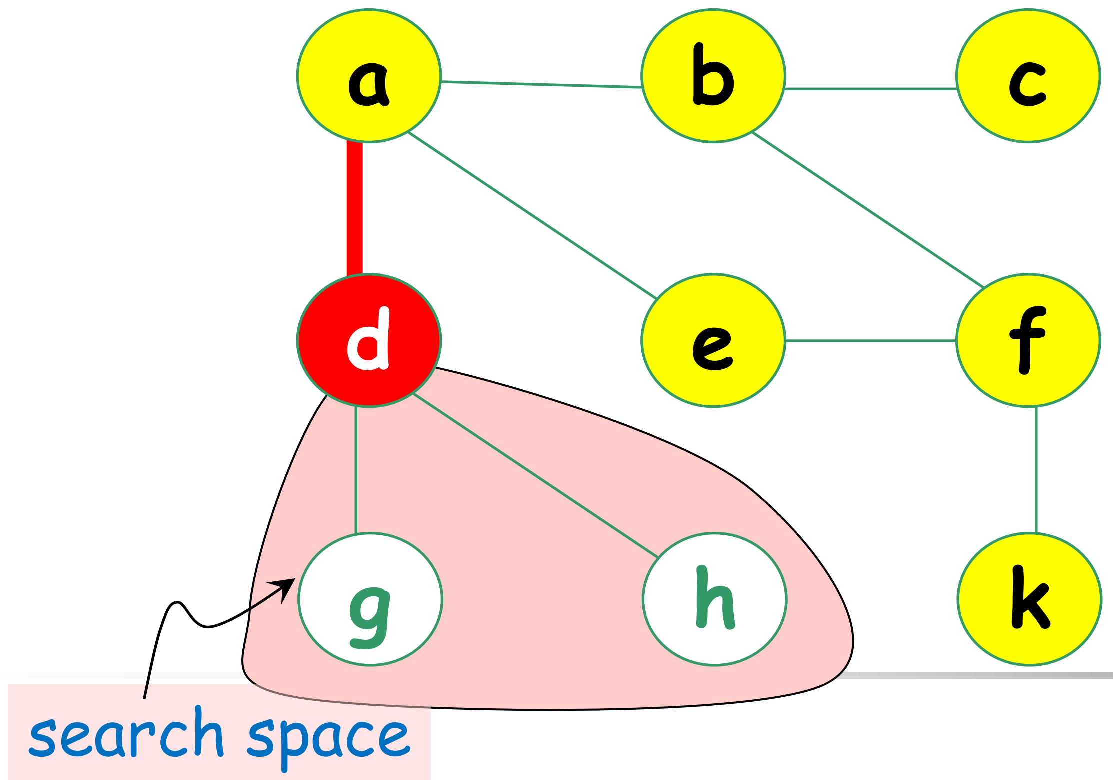
DFS searches **"deeper"** in the graph whenever possible

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



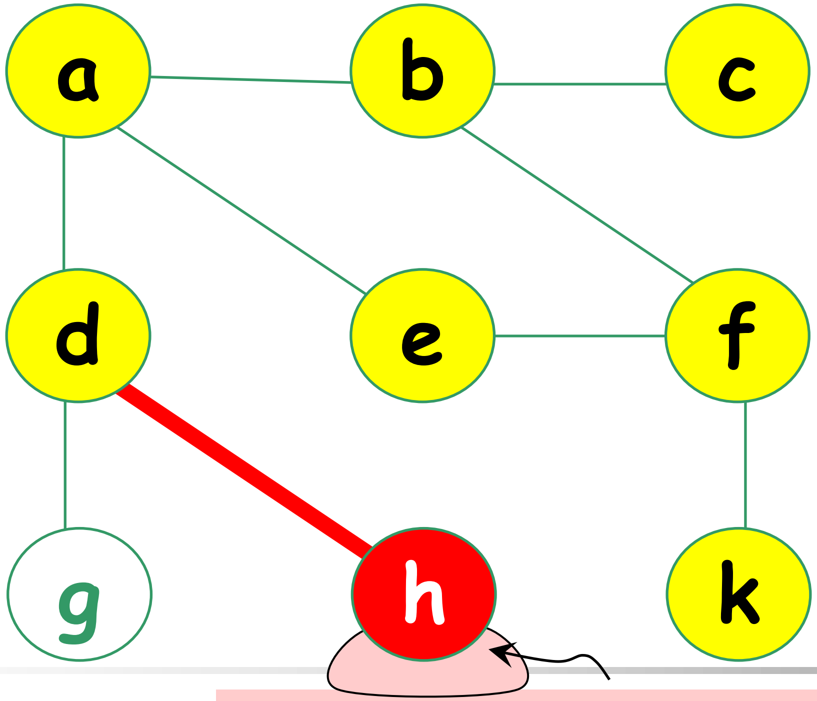
DFS searches **"deeper"** in the graph whenever possible

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



search space is empty

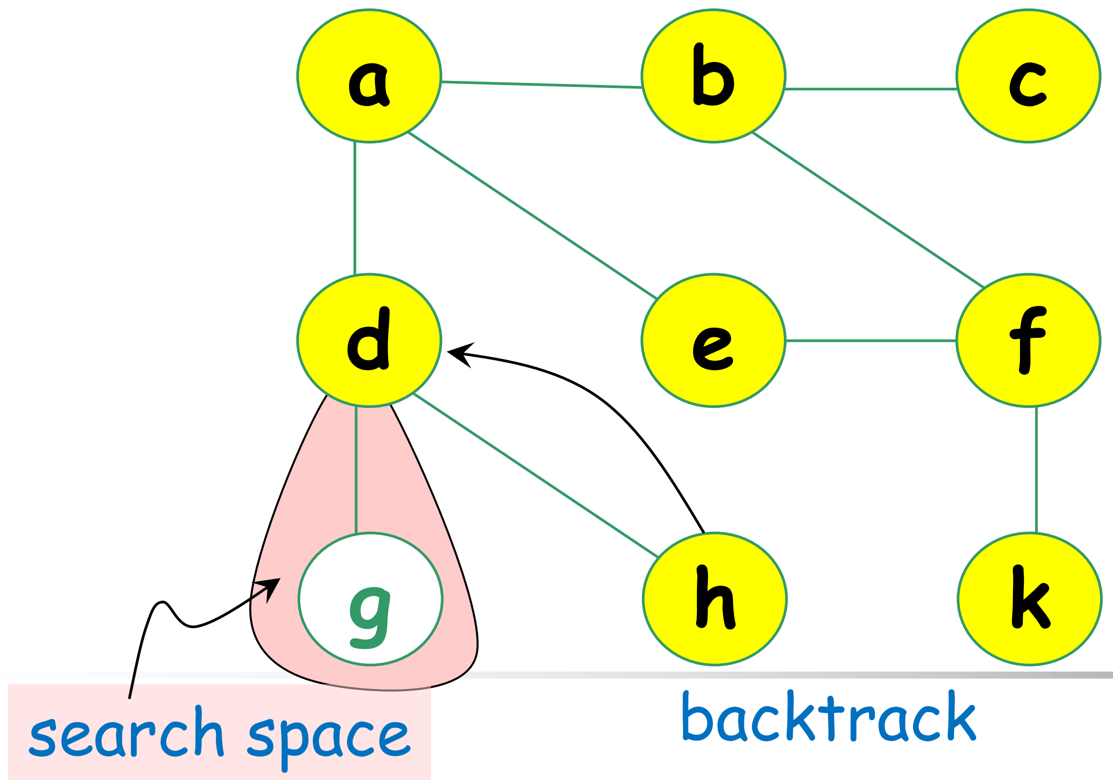
DFS searches "deeper" in the graph whenever possible

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



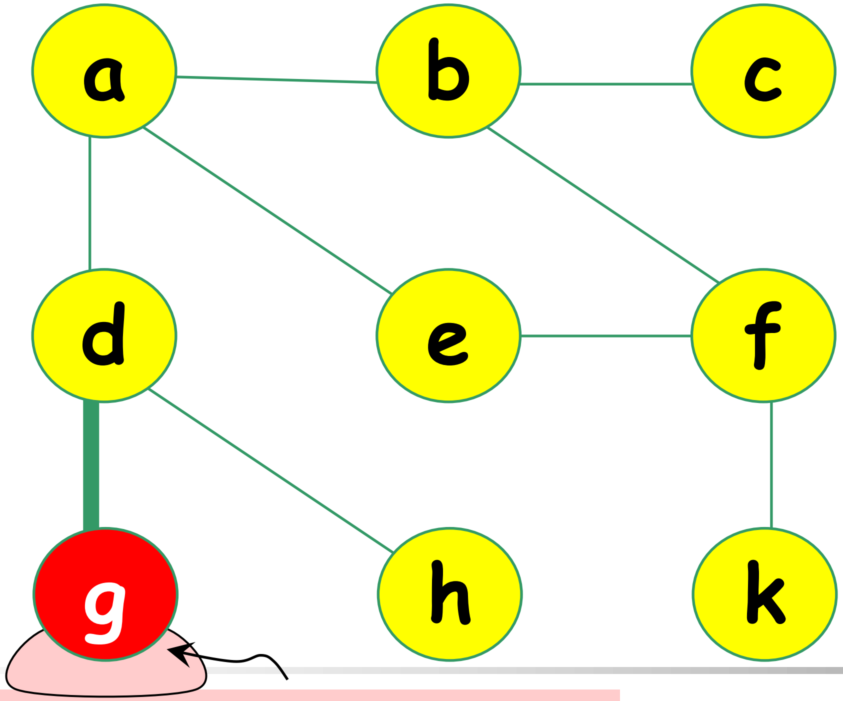
DFS searches **"deeper"** in the graph whenever possible

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 empty

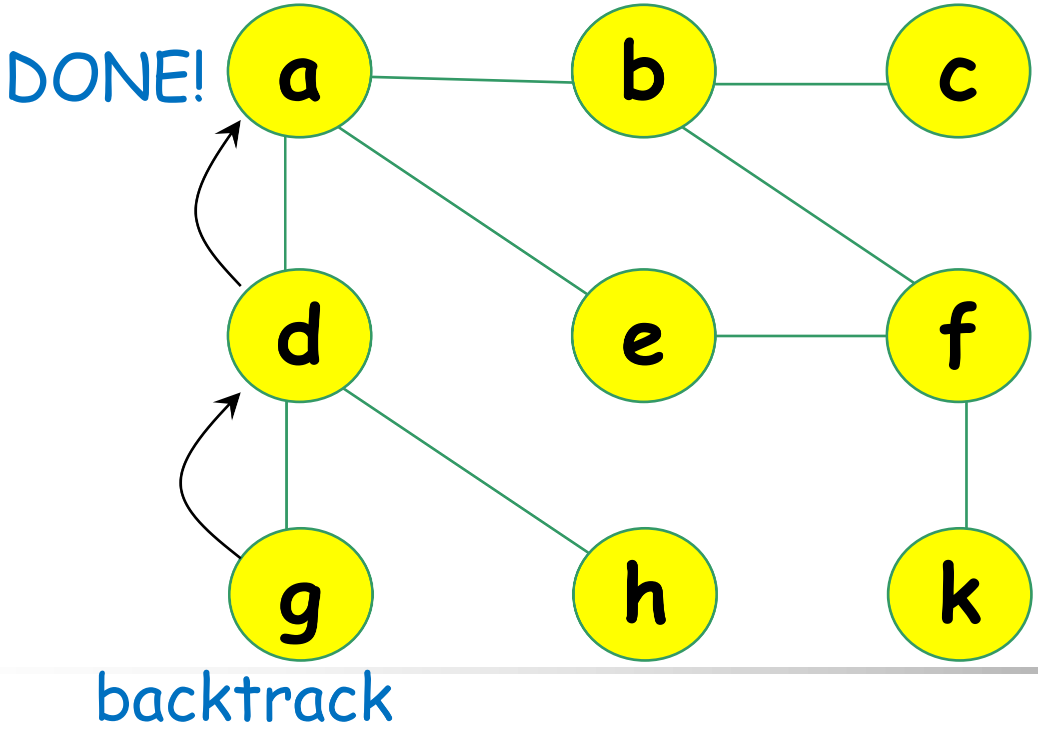
DFS searches **"deeper"** in the graph whenever possible

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

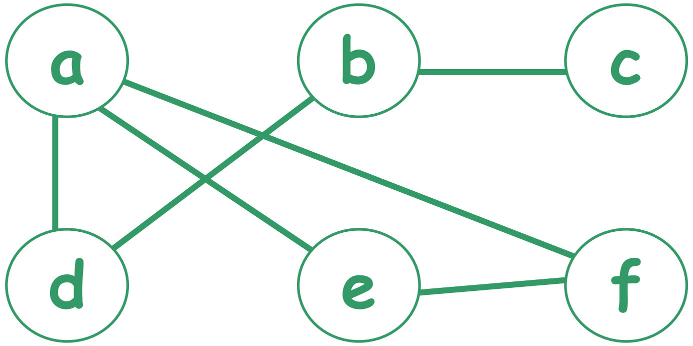
Depth First Search (DFS)

- **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.

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

Exercise – DFS

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



a, d, b, c, e, f

a, e, f, d, b, c

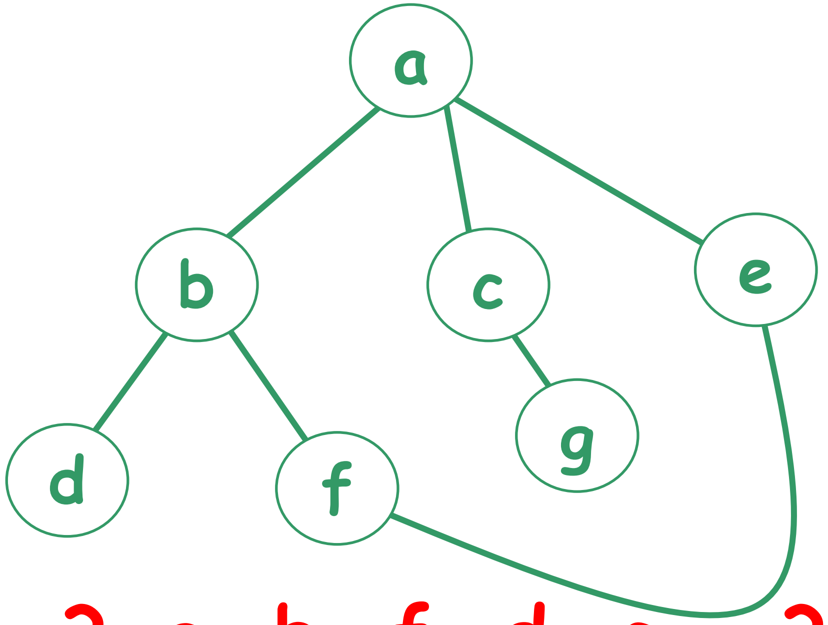
a, f, e, d, b, c

a, f, d, b, c, e??

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

Exercise (2) – DFS

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



a, b, d, f, e, c, g
a, b, f, e, d, c, g
a, c, g, b, d, f, e
a, c, g, b, f, e, d
a, c, g, e, f, b, d
a, e, f, b, d, c, g

a, e, b, …? a, b, f, d, c, …?

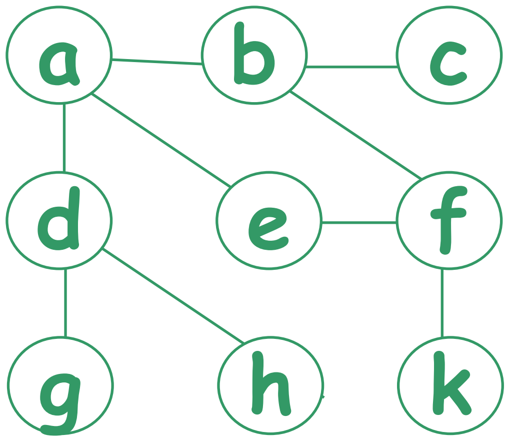
DFS – Pseudo code (recursive)

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

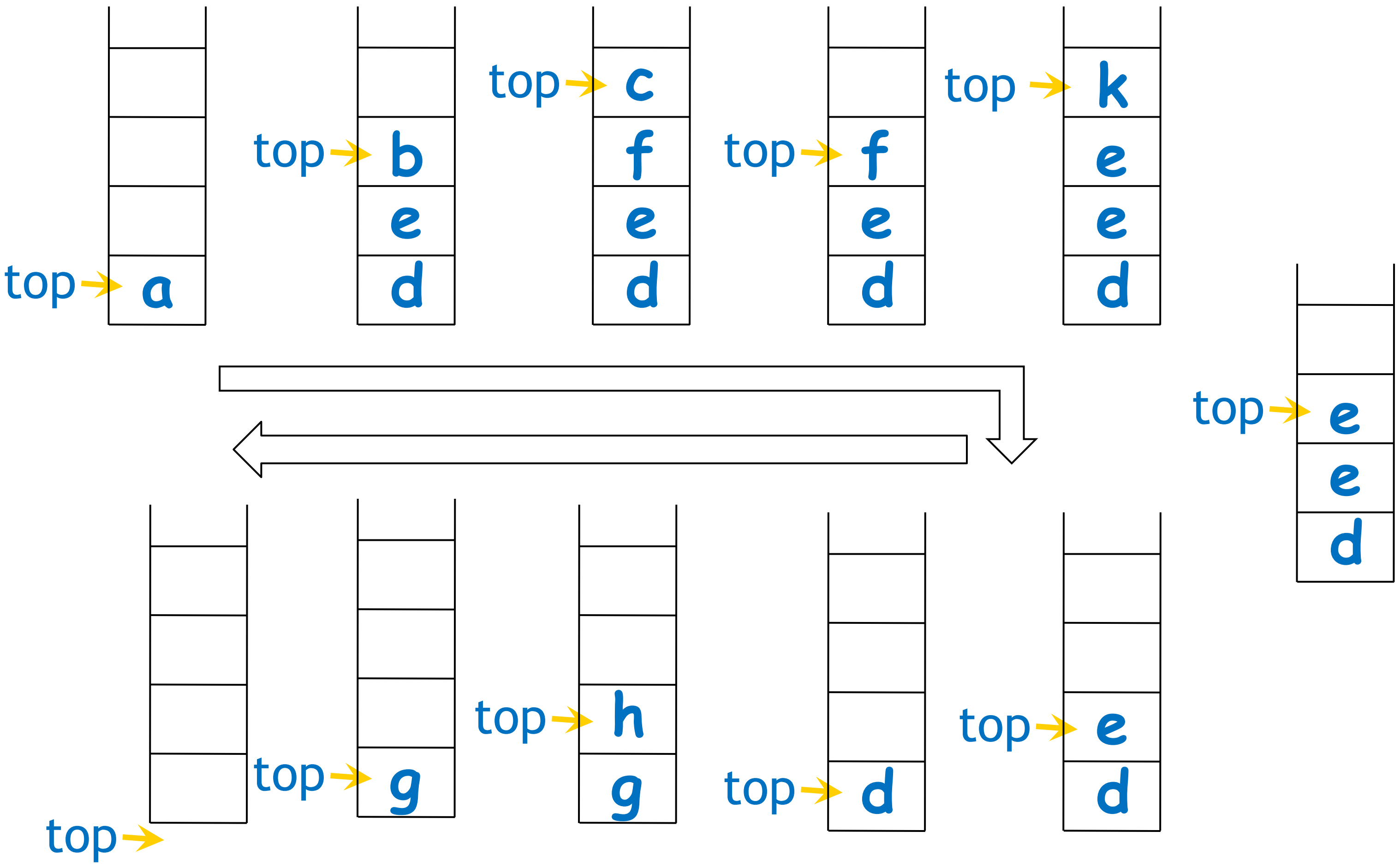
DFS – Pseudo code (using stack)

```
unmark all vertices
push starting vertex u onto top of stack S
while S is nonempty do
begin
    pop a vertex v from top of S
    if (v is unmarked) then
    begin
        visit and mark v
        for each unmarked neighbor w of v do
            push w onto top of S
    end
end
```

DFS using Stack



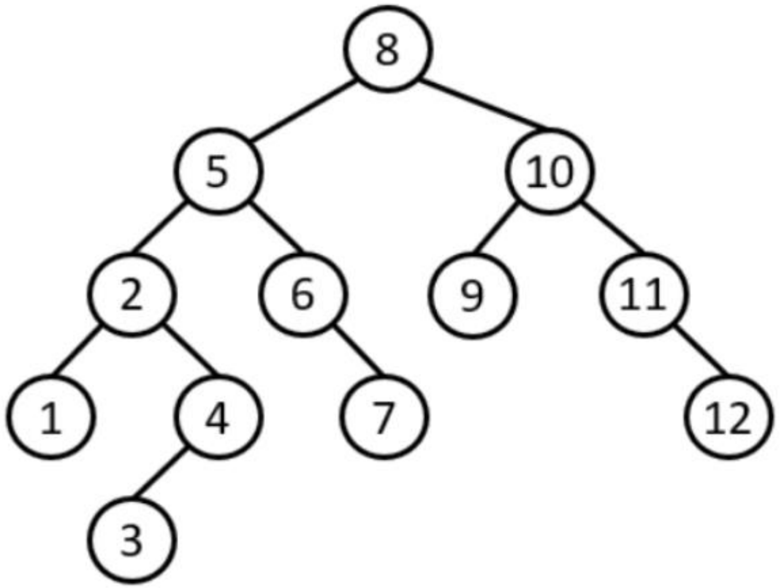
a, b, c, f, k, e, d, h, g



Exercise

- Implement BFS and DFS with Python

Exercise



Apply breadth-first search and depth-first search to the given tree starting from node '8' and show the order of exploration. **[4 marks]**

Learning outcome

- Able to tell what an undirected graph is and what a directed graph is
  - Know how to represent a graph using matrix and list

- Understand what Euler circuit is and able to determine whether such circuit exists in an undirected graph

- Able to apply BFS and DFS to traverse a graph