COMP108
Data Structures and Algorithms

Graphs (Part II Paths, Circuits, BFS/DFS)

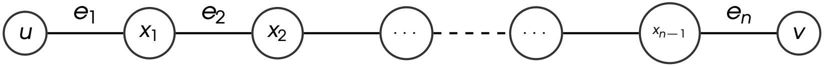
Professor Prudence Wong

pwong@liverpool.ac.uk

2022-23

Undirected graph - Paths and circuits

- In an undirected graph, a **path** from a vertex u to a vertex v is a sequence of edges $e_1 = \{u, x_1\}, e_2 = \{x_1, x_2\}, \cdots, e_n = \{x_{n-1}, v\}$, 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 in an undirected graph.
- If $u \equiv v$, this path is called a **circuit (cycle)**.

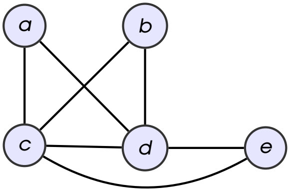


Euler Circuit

- A simple circuit visits an edge at most once.
- An **Euler circuit** is a circuit visiting every edge exactly once.
  (NB. A vertex can be repeated.)
- Does every graph has an Euler circuit ?

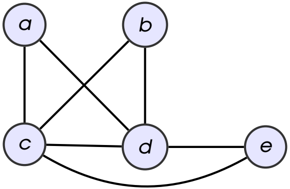
Euler Circuit

- A simple circuit visits an edge at most once.
- An **Euler circuit** is a circuit visiting every edge exactly once.
  (NB. A vertex can be repeated.)
- Does every graph has an Euler circuit ?



Euler Circuit

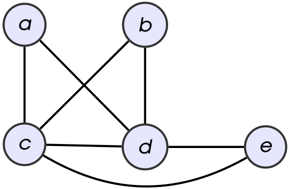
- A simple circuit visits an edge at most once.
- An **Euler circuit** is a circuit visiting every edge exactly once.
  (NB. A vertex can be repeated.)
- Does every graph has an Euler circuit ?

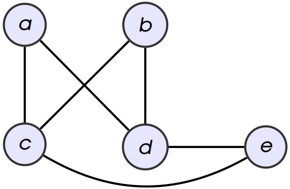


$a\ c\ b\ d\ e\ c\ d\ a$

Euler Circuit

- A simple circuit visits an edge at most once.
- An **Euler circuit** is a circuit visiting every edge exactly once.
  (NB. A vertex can be repeated.)
- Does every graph has an Euler circuit ?

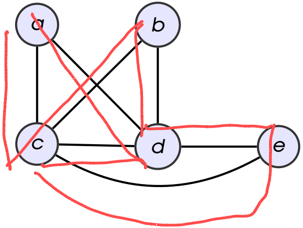




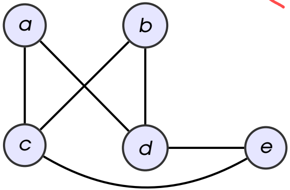
$a\ c\ b\ d\ e\ c\ d\ a$

Euler Circuit

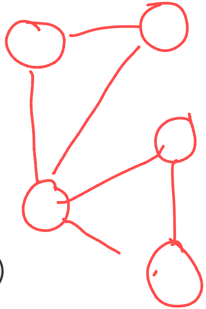
- A simple circuit visits an edge at most once.
- An **Euler circuit** is a circuit visiting every edge exactly once. (NB. A vertex can be repeated.)
- Does every graph has an Euler circuit ?



a c b d e c d a

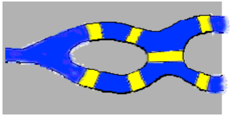


no Euler circuit



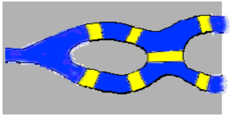
History

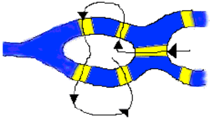
In Konigsberg, Germany, a river ran through the city with seven bridges built. People wondered whether one could go around the city by crossing each bridge exactly once.



History

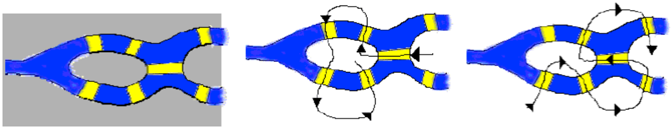
In Konigsberg, Germany, a river ran through the city with seven bridges built. People wondered whether one could go around the city by crossing each bridge exactly once.





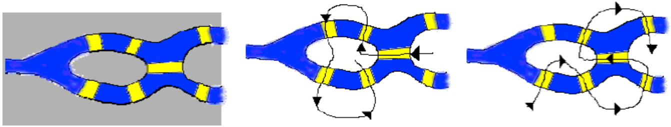
History

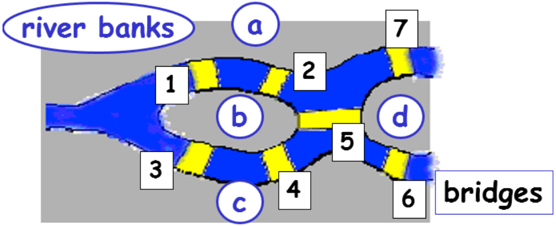
In Konigsberg, Germany, a river ran through the city with seven bridges built. People wondered whether one could go around the city by crossing each bridge exactly once.



History

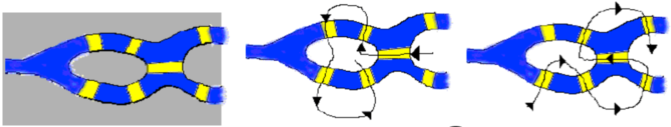
In Konigsberg, Germany, a river ran through the city with seven bridges built. People wondered whether one could go around the city by crossing each bridge exactly once.

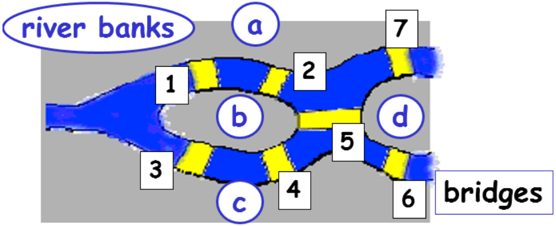


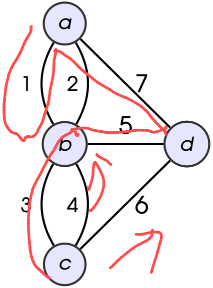


History

In Konigsberg, Germany, a river ran through the city with seven bridges built. People wondered whether one could go around the city by crossing each bridge exactly once.

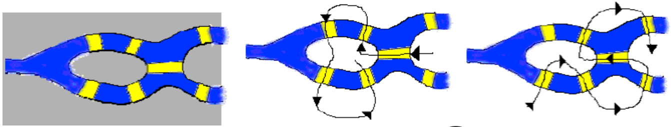


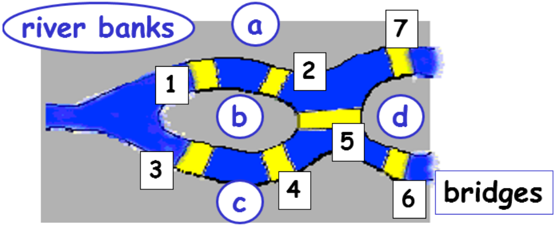


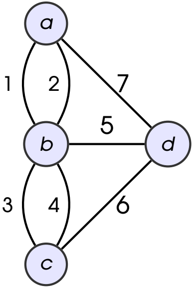


History

In Konigsberg, Germany, a river ran through the city with seven bridges built. People wondered whether one could go around the city by crossing each bridge exactly once.



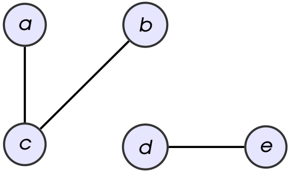




How to determine if there is Euler circuit?

A trivial condition for Euler circuit

- 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.
- Being connected is a necessary condition but not sufficient.



Necessary and Sufficient Condition

Let G be a connected graph.

Lemma

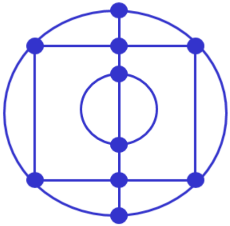
G contains an Euler circuit if and only if the degree of every vertex is even.

Necessary and Sufficient Condition

Let G be a connected graph.

Lemma

G contains an Euler circuit if and only if the degree of every vertex is even.



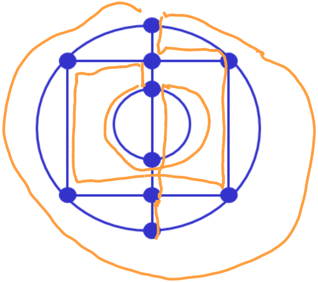
Necessary and Sufficient Condition

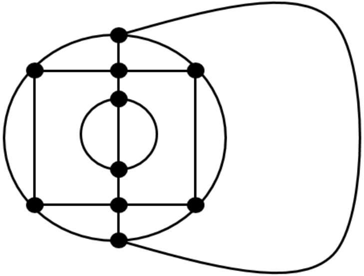
Let G be a connected graph.

Euler path

Lemma

G contains an Euler circuit if and only if the degree of every vertex is even.

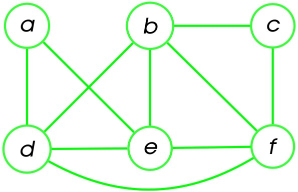




Necessary and Sufficient Condition

Lemma

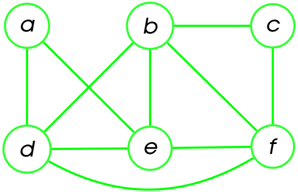
G contains an Euler circuit if and only if the degree of every vertex is even.

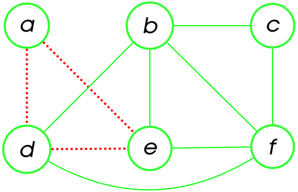


Necessary and Sufficient Condition

Lemma

G contains an Euler circuit if and only if the degree of every vertex is even.



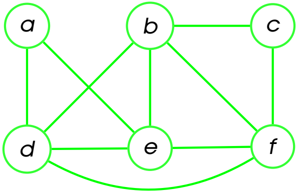


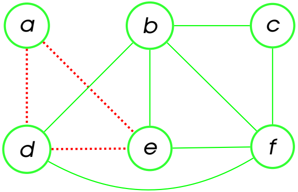
aeda

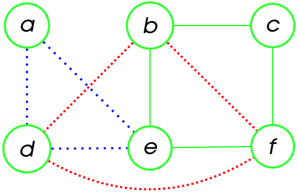
Necessary and Sufficient Condition

Lemma

G contains an Euler circuit if and only if the degree of every vertex is even.







aeda

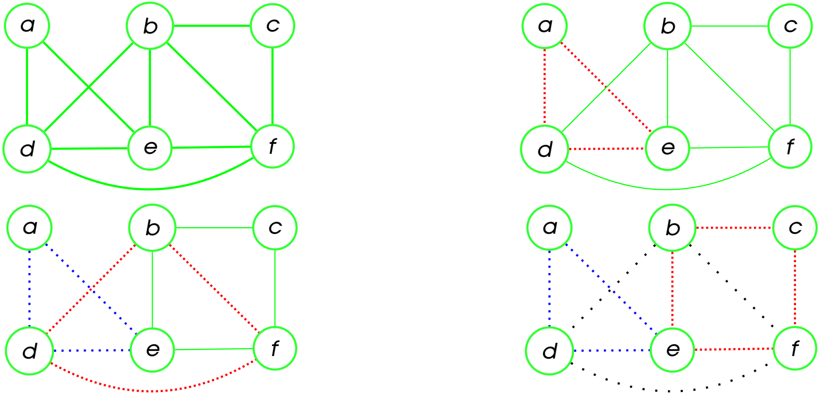
aeda

aedbfda

Necessary and Sufficient Condition

Lemma

G contains an Euler circuit if and only if the degree of every vertex is even.



aeda

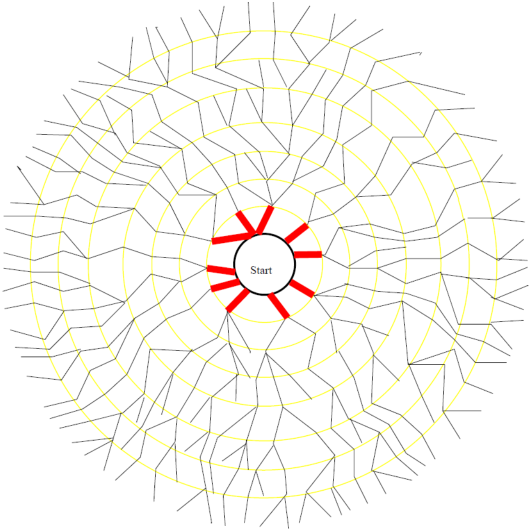
aeda
aedbfda

aedbfda
aedbfebcfda

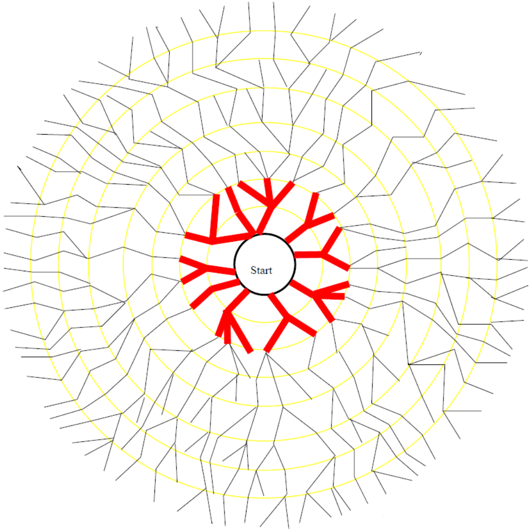
Hamiltonian circuit

- 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)

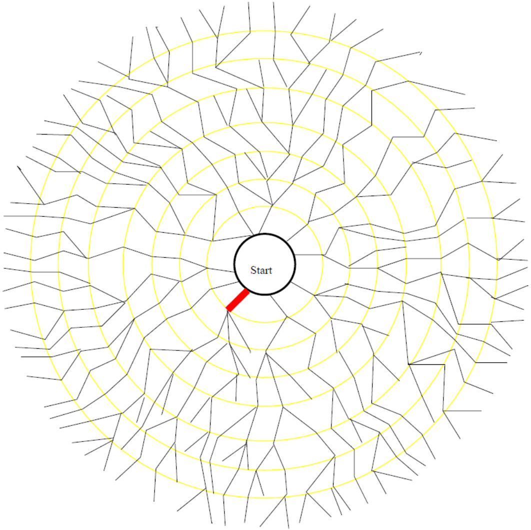
Breadth First Search (BFS) - figure adopted from COMP111



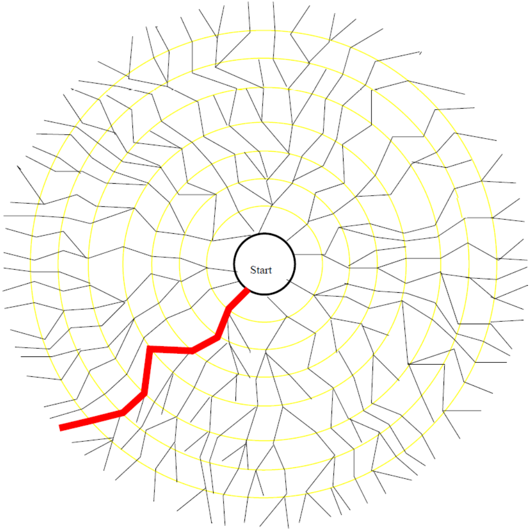
Breadth First Search (BFS) - figure adopted from COMP111



Depth First Search (DFS) - figure adopted from COMP111



Depth First Search (DFS) - figure adopted from COMP111

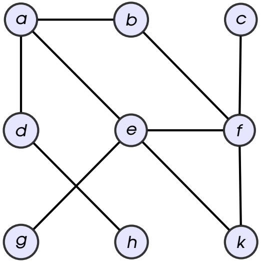


BFS...

BFS - with queue / linked list

BFS:

Suppose a is the starting vertex.

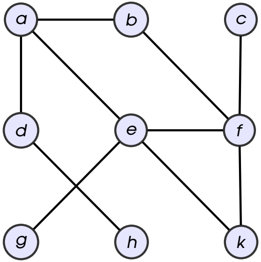


BFS - with queue / linked list

BFS: a,

Suppose a is the starting vertex.

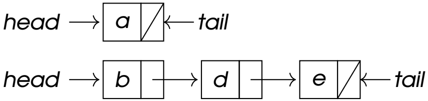
head $\longrightarrow$ | a | / | $\longleftarrow$ *tail*

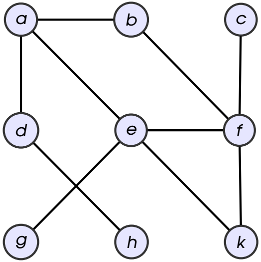


BFS - with queue / linked list

BFS: a, b,

Suppose a is the starting vertex.

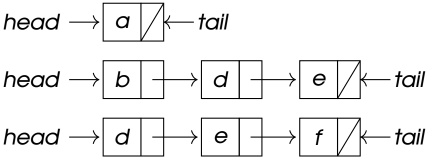


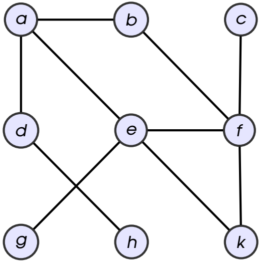


BFS - with queue / linked list

BFS: *a*, *b*, *d*,

Suppose *a* is the starting vertex.

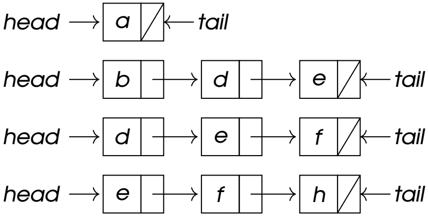


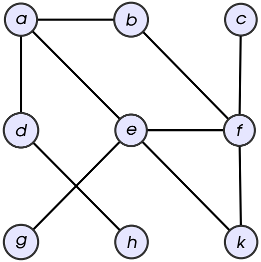


BFS - with queue / linked list

BFS: *a*, *b*, *d*, *e*,

Suppose *a* is the starting vertex.

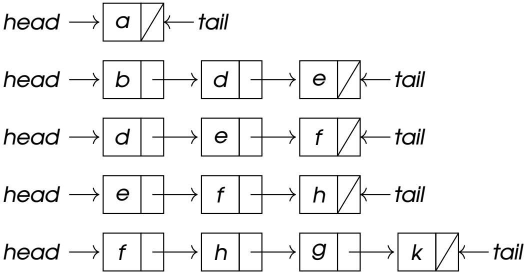


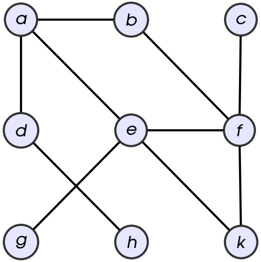


BFS - with queue / linked list

BFS: a, b, d, e, f,

Suppose a is the starting vertex.

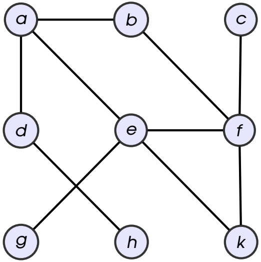


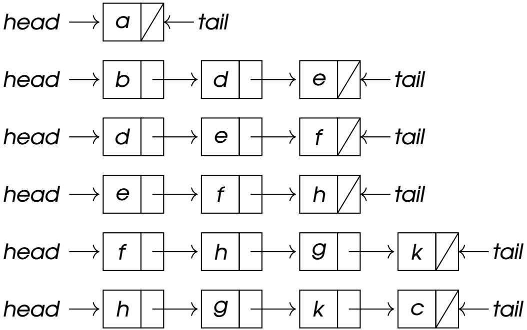


BFS - with queue / linked list

BFS: *a*, *b*, *d*, *e*, *f*, *h*,

Suppose *a* is the starting vertex.

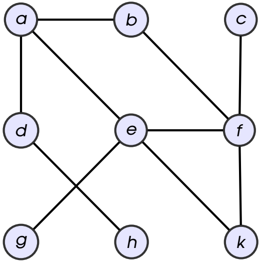


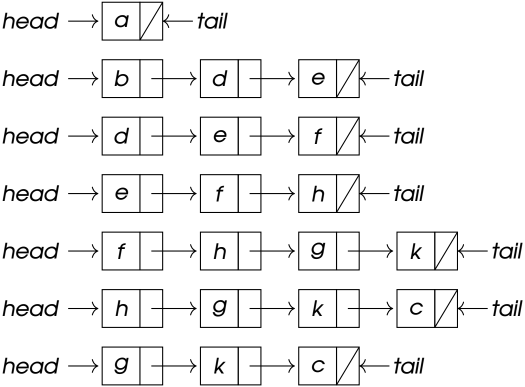


BFS - with queue / linked list

BFS: *a*, *b*, *d*, *e*, *f*, *h*, *g*,

Suppose *a* is the starting vertex.

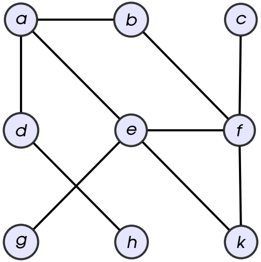


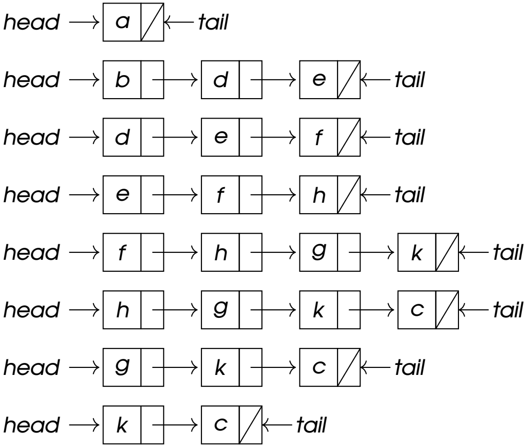


BFS - with queue / linked list

BFS: *a, b, d, e, f, h, g, k,*

Suppose *a* is the starting vertex.

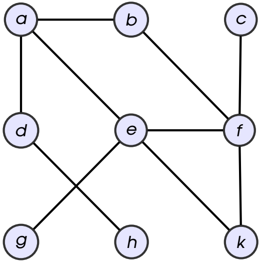


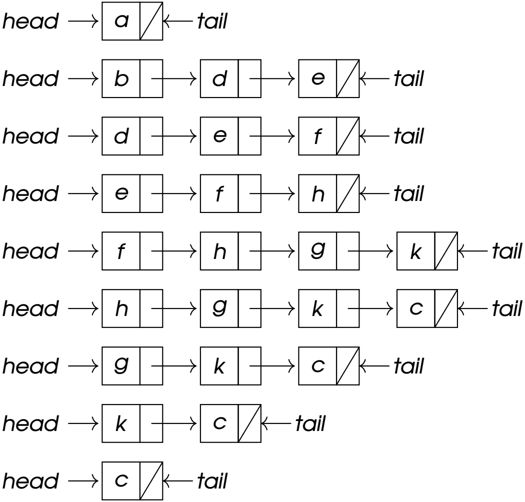


BFS - with queue / linked list

BFS: *a, b, d, e, f, h, g, k, c*

Suppose *a* is the starting vertex.





BFS - pseudo code - with linked list / queue

unmark all vertices

choose some starting vertex s

mark s and insert s into tail of list L

BFS - pseudo code - with linked list / queue

unmark all vertices
choose some starting vertex s
mark s and insert s into tail of list L
while L is nonempty do
begin

end

BFS - pseudo code - with linked list / queue

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 **head of L**

end

BFS - pseudo code - with linked list / queue

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 **head of L**
    visit v // e.g., print its data

end

BFS - pseudo code - with linked list / queue

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 **head of *L***
    visit *v* // e.g., print its data
    for each **unmarked neighbor *w* of *v*** do

end

BFS - pseudo code - with linked list / queue

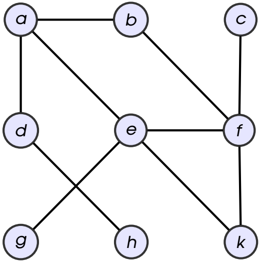
```
unmark all vertices
choose some starting vertex s
mark s and insert s into tail of list L
while L is nonempty do
begin
    remove a vertex v from head of L
    visit v // e.g., print its data
    for each unmarked neighbor w of v do
        mark w and insert w into tail of list L
end
```

DFS...

DFS - with stack

DFS:

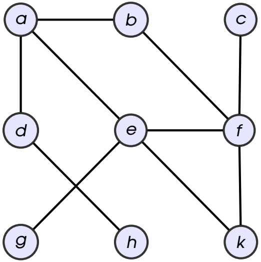
Suppose a is the starting vertex.

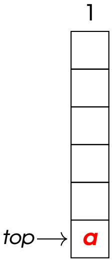


DFS - with stack

DFS: a,

Suppose a is the starting vertex.

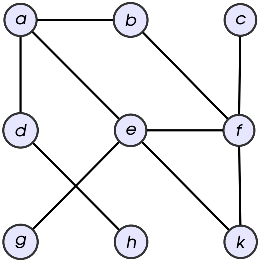


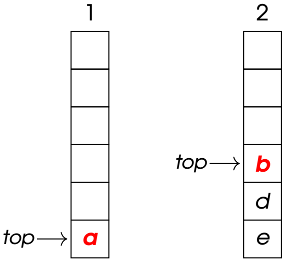


DFS - with stack

DFS: a, b,

Suppose a is the starting vertex.

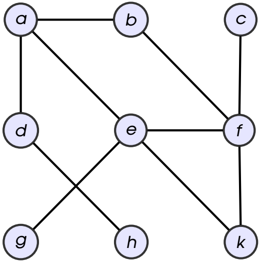


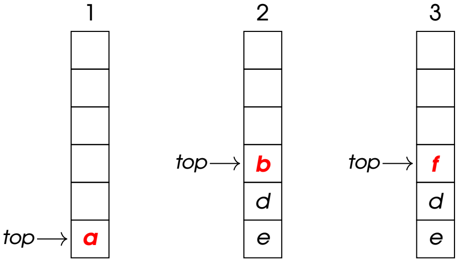


DFS - with stack

DFS: a, b, f,

Suppose a is the starting vertex.

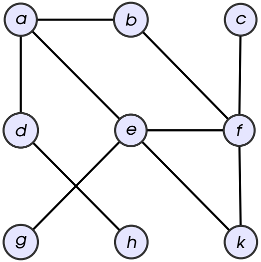


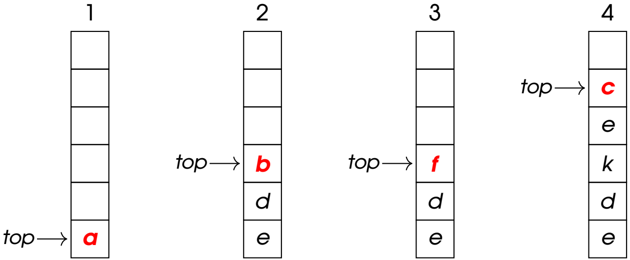


DFS - with stack

DFS: a, b, f, c,

Suppose a is the starting vertex.

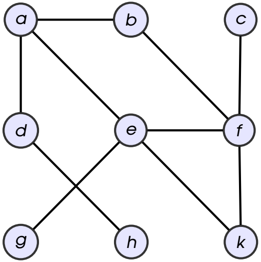


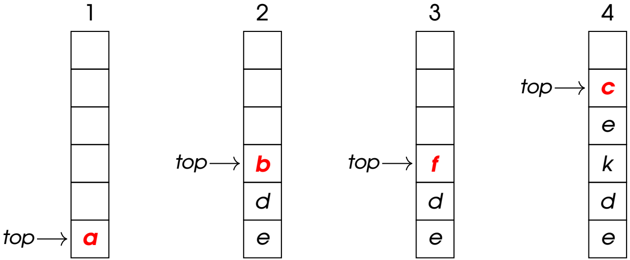


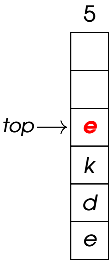
DFS - with stack

DFS: a, b, f, c, e,

Suppose a is the starting vertex.



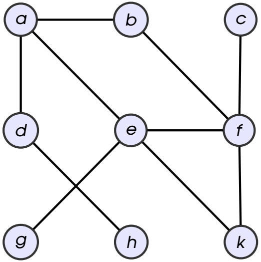


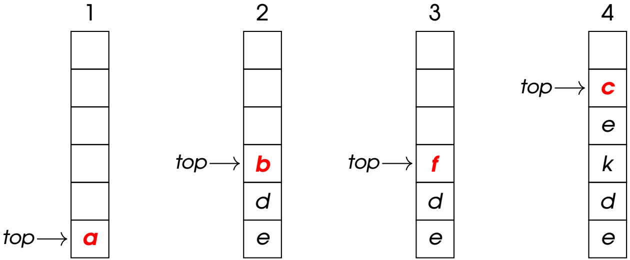


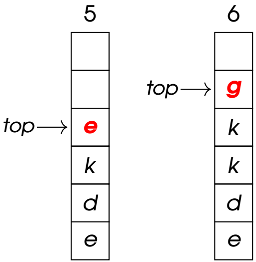
DFS - with stack

DFS: a, b, f, c, e, g,

Suppose a is the starting vertex.



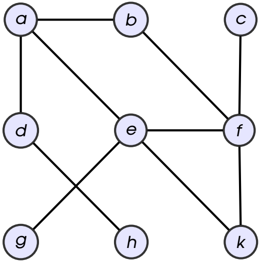


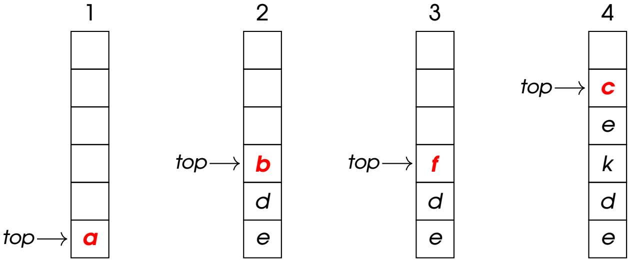


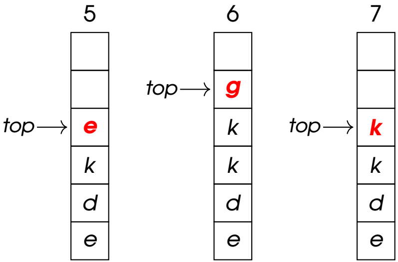
DFS - with stack

DFS: a, b, f, c, e, g, k,

Suppose a is the starting vertex.



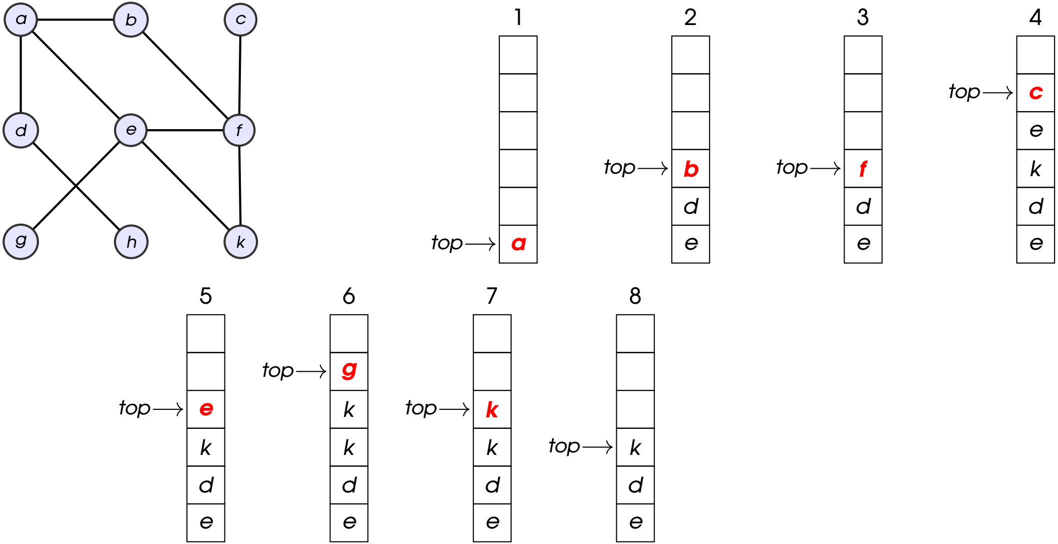




DFS - with stack

DFS: a, b, f, c, e, g, k,

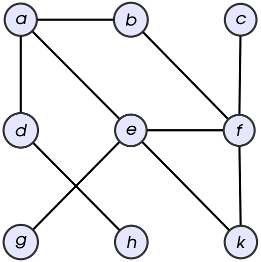
Suppose a is the starting vertex.

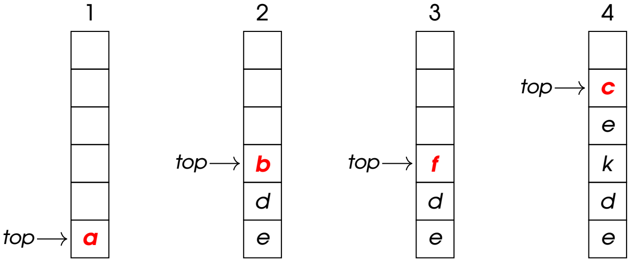


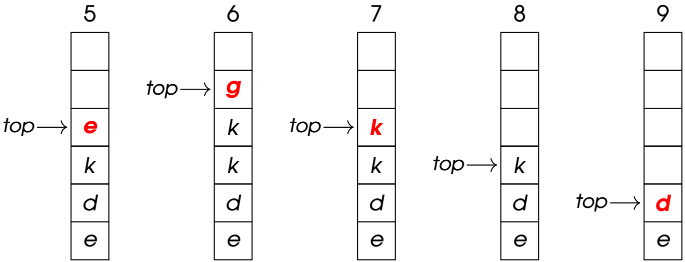
DFS - with stack

DFS: *a*, *b*, *f*, *c*, *e*, *g*, *k*, *d*,

Suppose *a* is the starting vertex.



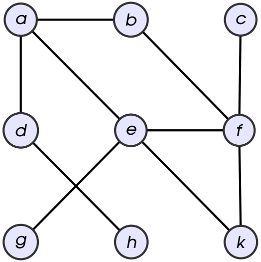


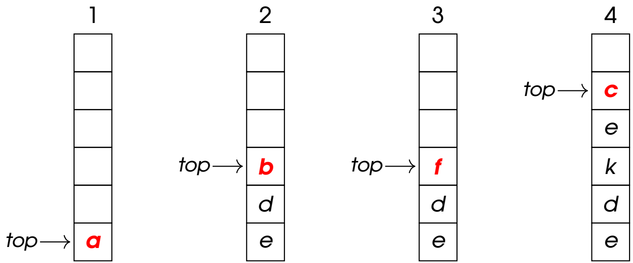


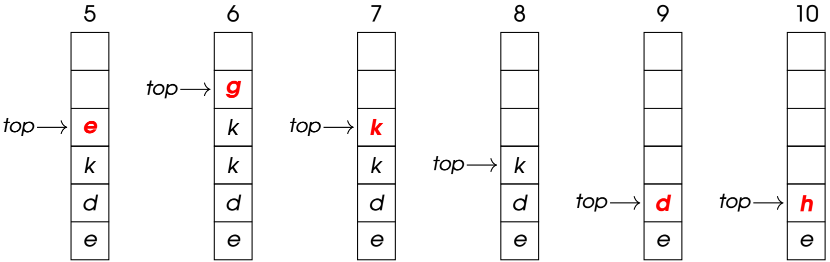
DFS - with stack

DFS: *a*, *b*, *f*, *c*, *e*, *g*, *k*, *d*, *h*

Suppose *a* is the starting vertex.



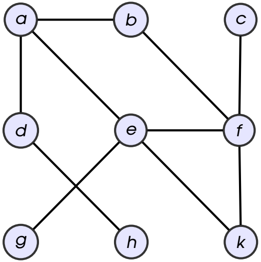


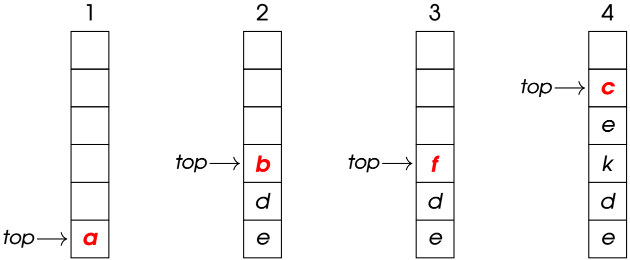


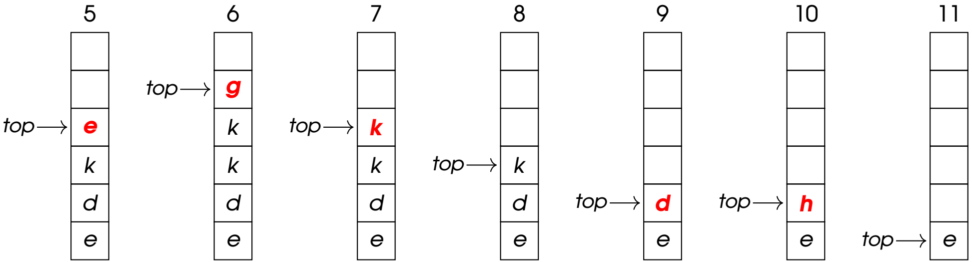
DFS - with stack

DFS: *a, b, f, c, e, g, k, d, h*

Suppose *a* is the starting vertex.







DFS - pseudo code - with stack

unmark all vertices

DFS - pseudo code - with stack

unmark all vertices

push starting vertex u onto **top of stack** S

DFS - pseudo code - with stack

```
unmark all vertices
push starting vertex u onto top of stack S
while S is nonempty do
begin



end
```

DFS - pseudo code - with 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***

end

DFS - pseudo code - with 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



    end
end
```

DFS - pseudo code - with 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


    end
end
```

DFS - pseudo code - with 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

    end
end
```

DFS - pseudo code - with 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
```

Summary

Summary: Traversals

Next: Greedy Algorithms

For note taking