Introduction, DFS, BFS

The beOI Instructors

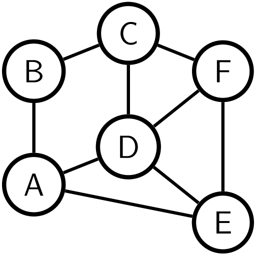
November 10, 2018

OLYMPIADE BELGE D'INFORMATIQUE
BELGISCHE INFORMATICA-OLYMPIADE

What is a graph ?

A graph can be described as a set of points linked or not together.
The points are called **Vertices** and the links **Edges**.
For example, a graph can represent a country : the cities are the vertices and the road are the edges.

Graph : example



An example of graph

- 6 Vertices : A,B,C,D,E and F
- 9 Edges : (A,B), (B,C), (A,D), (D,C), (A,E), (D,E), (D,F), (C,F) and (E,F)

Terminology and definitions

A graph can be said to be :

- **Undirected** if all edges are both-way
- **Directed** if there are one-way edges
- **Weighted** if there is a number associated with each edge (like a distance)
- **Unweighted** otherwise
- **Connected** if there is a path between all pair of vertices
- **Acyclic** if there is no loop
- **Dense** if there is an edge between all pair of vertices

Warning

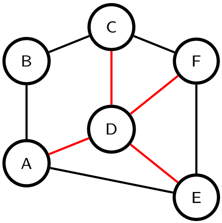
There may be more than one edge between two vertices. The graph is then called a Multigraph (otherwise it's a simple graph).

Terminology and definitions

Two main numbers in a graph

- V the vertices number
- E the edges number

The degree of a vertex is the number of edges linking to this vertex.

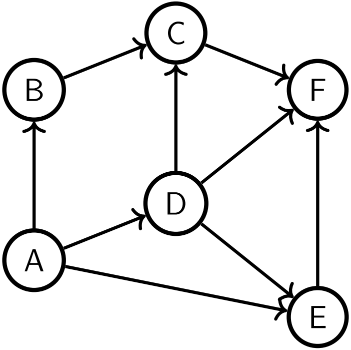


$$deg(D) = 4$$

Tree

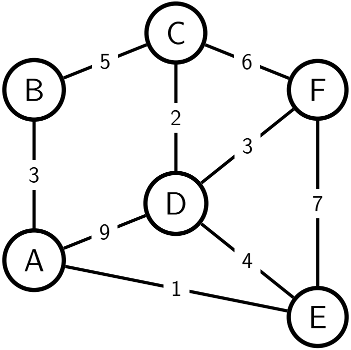
A simple undirected graph is called a **Tree** if there is exactly one path between all pair of vertices. A tree is always acyclic. Being a tree and having $E = V - 1$ are equivalent (except if there are multiple vertices between two nodes).

Directed graph : example



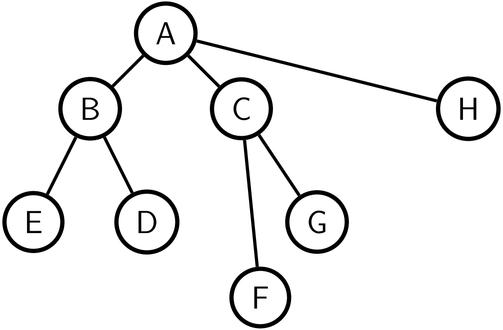
An example of directed graph

Weighted graph : example



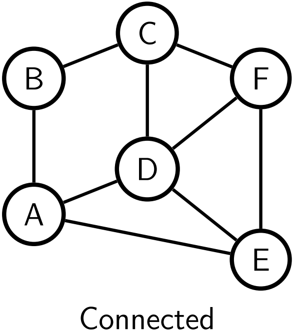
An example of weighted graph

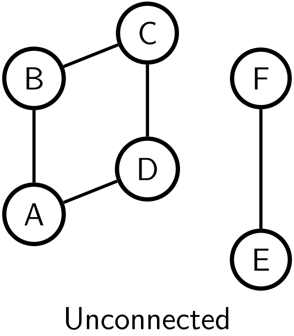
Tree : example



An example of a tree

Connected vs Unconnected



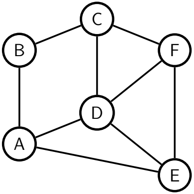


Representing a graph

There are three ways of representing a graph :

- In the form of an edges list (we won't use it here)
- In the form of an adjacency matrix
- In the form of an adjacency list.

Adjacency matrix

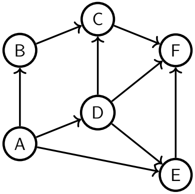


$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

- Store connection in a boolean matrix
- $a_{ij} =$ true if vertices i and j are connected
- $\mathcal{O}(V^2)$ memory

As the graph is undirected, the matrix is symmetric.

Adjacency matrix for directed graph

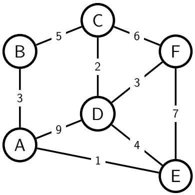


$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- $a_{ij} =$ true if there is an edge from i to j
- $a_{ij} \neq a_{ji}$ in most cases

Not symmetric

Adjacency matrix for weighted graph

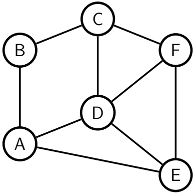


$$\begin{pmatrix} 0 & 3 & 0 & 9 & 1 & 0 \\ 3 & 0 & 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 2 & 0 & 6 \\ 9 & 0 & 2 & 0 & 3 & 4 \\ 1 & 0 & 0 & 3 & 0 & 7 \\ 0 & 0 & 6 & 4 & 7 & 0 \end{pmatrix}$$

- Replace the boolean array by an integer (or double) array
- a_{ij} = distance (or weight) from i to j
- a_{ij} symmetric if the graph is not directed

You can't use a adjacency matrix on a Multigraph !.

Adjacency list

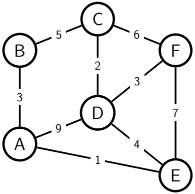


$A : \{B, D, E\}$
$B : \{A, C\}$
$C : \{B, D, F\}$
$D : \{A, C, E, F\}$
$E : \{A, D, F\}$
$F : \{C, D, E\}$

- Store connection in array of lists
- `adj[i]` is a list containing the adjacency vertices of vertex i
- Only way to work in a Multigraph.
- $\mathcal{O}(E)$ memory

If the graph is undirected, don't forget to add the edges in both ways !

Adjacency list for weighted graph

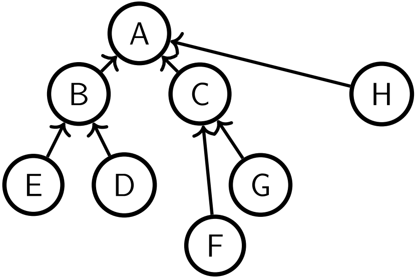


$A : \{(B,3),(D,9),(E,1)\}$
$B : \{(A,3),(C,5)\}$
$C : \{(B,5),(D,2),(F,6)\}$
$D : \{(A,9),(C,2),(E,4),(F,3)\}$
$E : \{(A,1),(D,4),(F,7)\}$
$F : \{(C,6),(D,3),(E,7)\}$

- Same principle
- For each connexion, store a pair of number : the destination vertex and the edge weight.
- Use an array of `vector<pair<int, int>>`

Tree parent representation

- Chose a vertex called the **root**
- Each vertex will have a parent except for the root
- Build it recursively

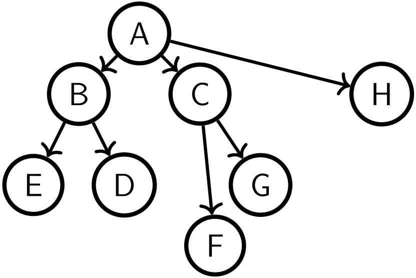


Parents :

A : has no parent (root)
B : A
C : A
D : B
E : B
F : C
G : C
H : A

Tree list of children representation

- Transposed of parents representation
- If X is parent of Y, Y is child of X
- Store the children of a vertex in an array
- A vertex without child is called a **leaf**



Children :

A : $\{B, C, H\}$
B : $\{D, E\}$
C : $\{F, G\}$
D : $\{\}$ (leaf)
E : $\{\}$ (leaf)
F : $\{\}$ (leaf)
G : $\{\}$ (leaf)
H : $\{\}$ (leaf)

Graph representations comparison

Quick summary :

	Memory	Scan all edges	Edge lookup
Adjacency matrix	$\mathcal{O}(V^2)$	$\mathcal{O}(V^2)$	$\mathcal{O}(1)$
Adjacency list	$\mathcal{O}(V+E)$	$\mathcal{O}(V+E)$	$\mathcal{O}(deg(v))$

Conclusion

Use an adjacency matrix for simple graph if there is a lot of edge lookups or if the graph is dense. Otherwise use an adjacency list.

What is a DFS ?

DFS stands for Depth First Search. It is an algorithm that visits every vertices of a connected graph.

The algorithm works recursively.

- Start by visiting one vertex.
- When visiting a vertex :
  - If visited, stop and return
  - Else, mark it as visited add call the algorithm over it's neighbours

Run time : $\mathcal{O}(V+E)$ (for adjacency list)

DFS : Usage

Use a DFS for :

- Check if a graph is connected
- Count the number of vertices in a connected subgraph

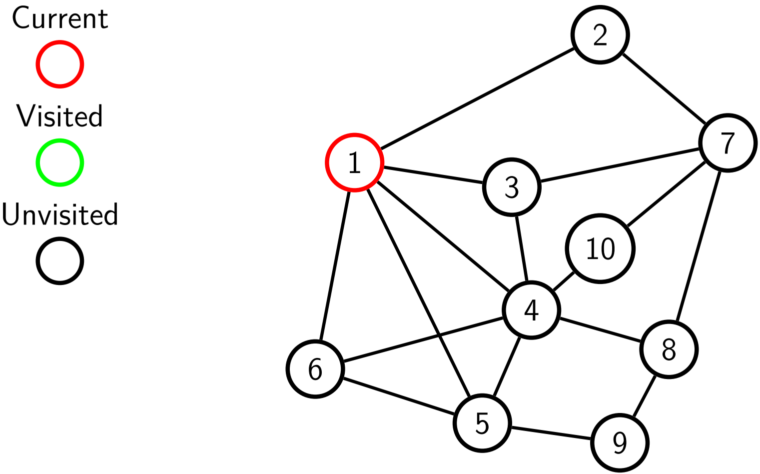
Don't use a DFS for :

- Path finding
- Distance calculation

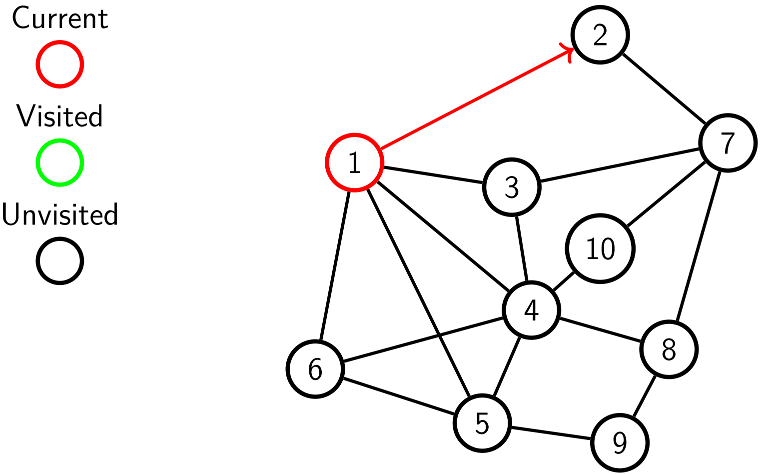
DFS : Code

```
vector<vector<int>> adj;
vector<bool> visited;

void dfs(int u) {
  if(visited[u])
    return;
  visited[u] = true;

  // Do something with u

  for(int v : adj[u]) {
    dfs(v);
  }
}
```

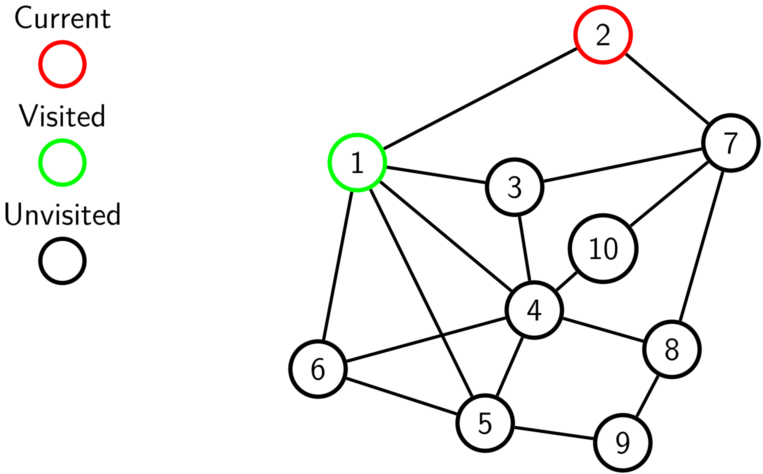
DFS : Example



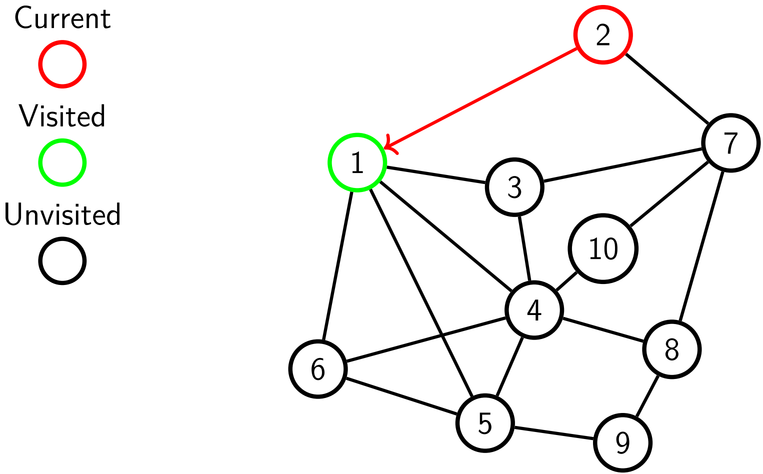
DFS : Example



DFS : Example



DFS : Example

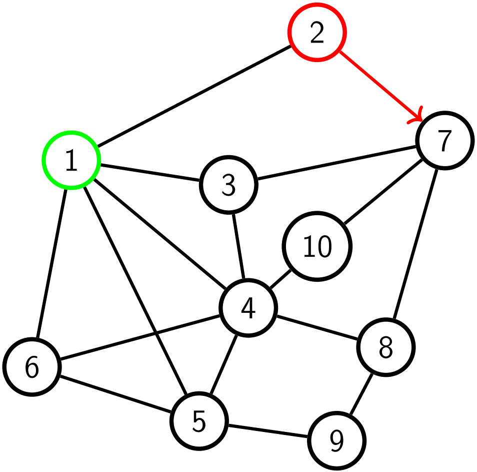


DFS : Example

Current

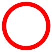
Visited

Unvisited

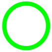


DFS : Example

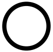
Current

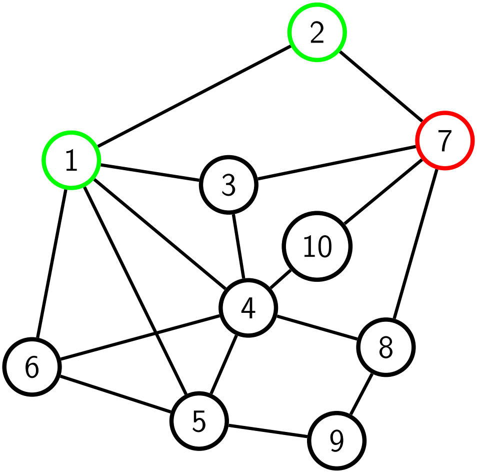


Visited

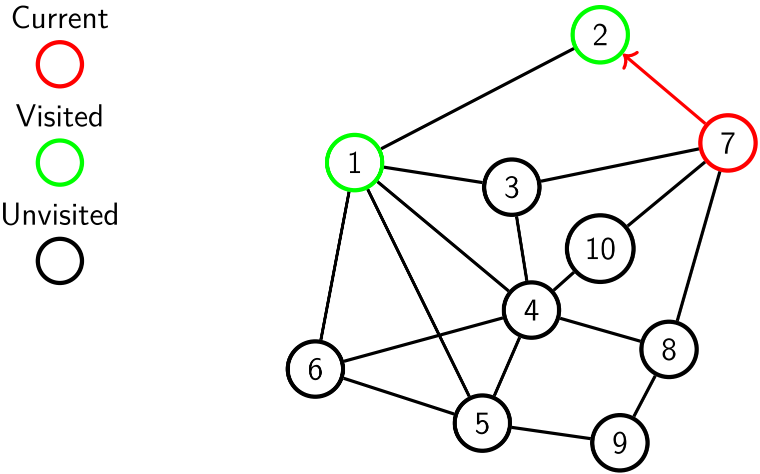


Unvisited

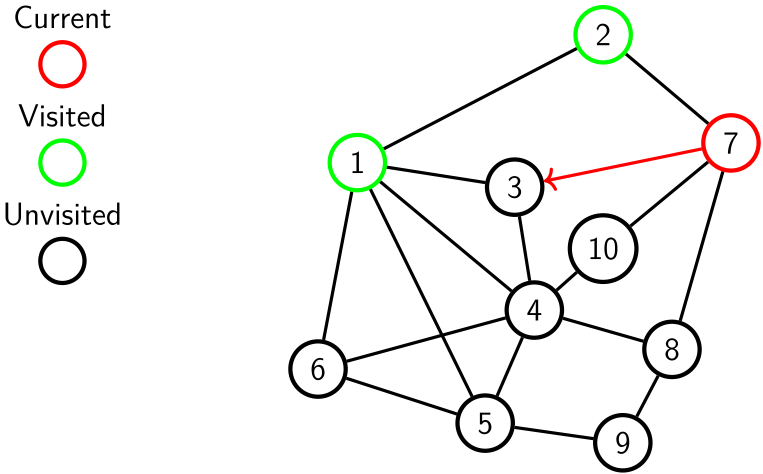




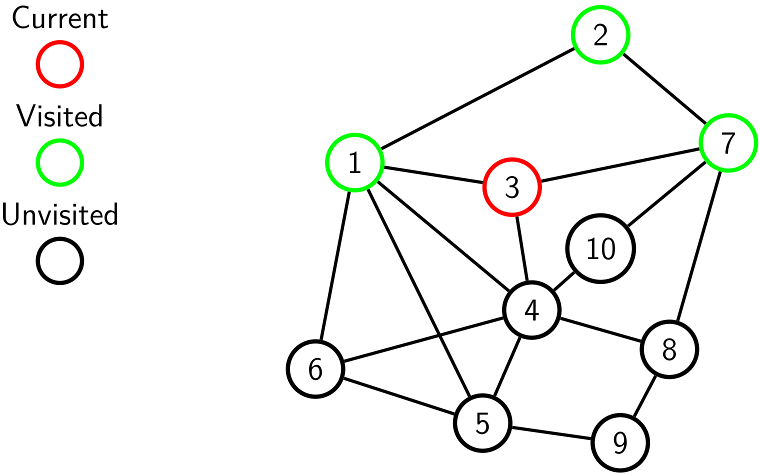
DFS : Example



DFS : Example



DFS : Example

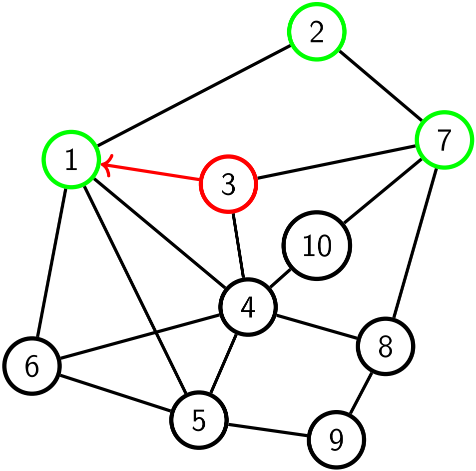


DFS : Example

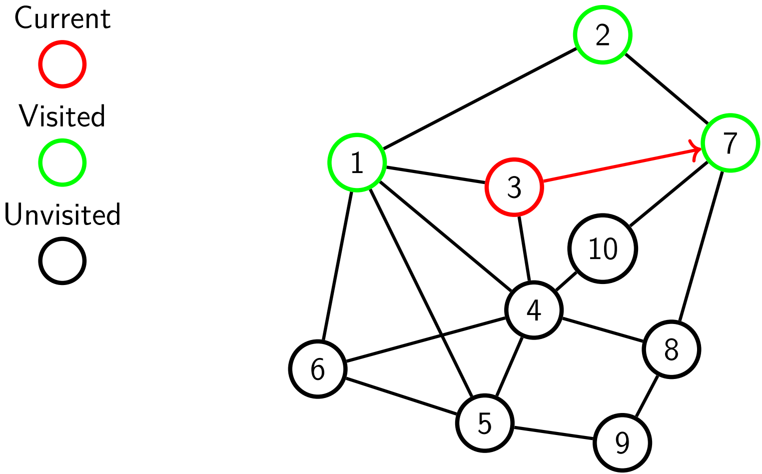
Current

Visited

Unvisited



DFS : Example

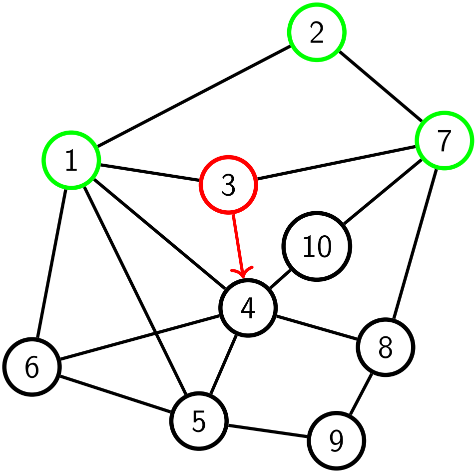


DFS : Example

Current

Visited

Unvisited

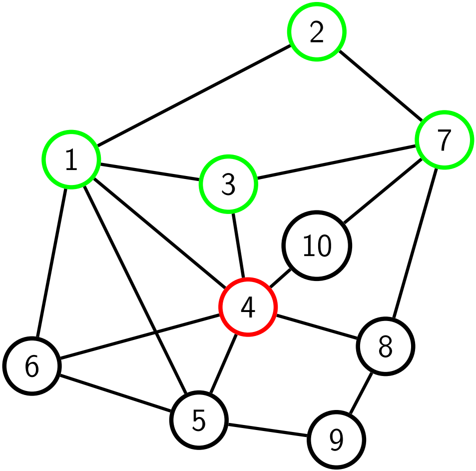


DFS : Example

Current

Visited

Unvisited

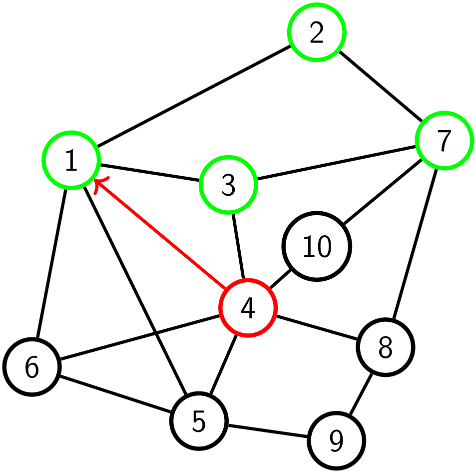


DFS : Example

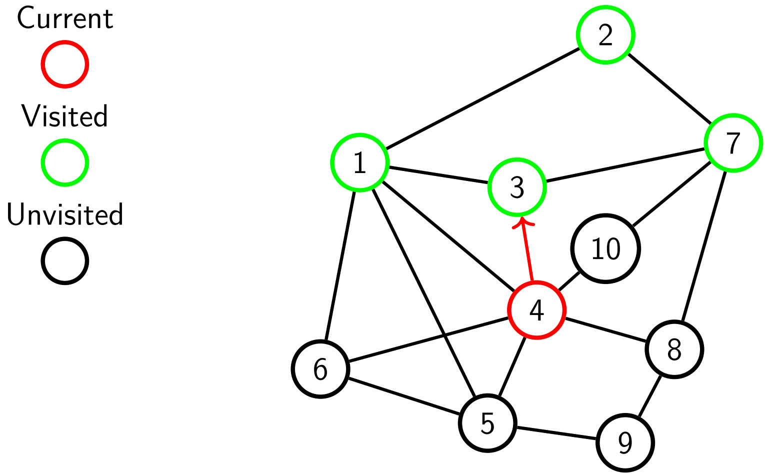
Current

Visited

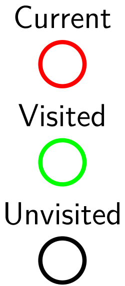
Unvisited

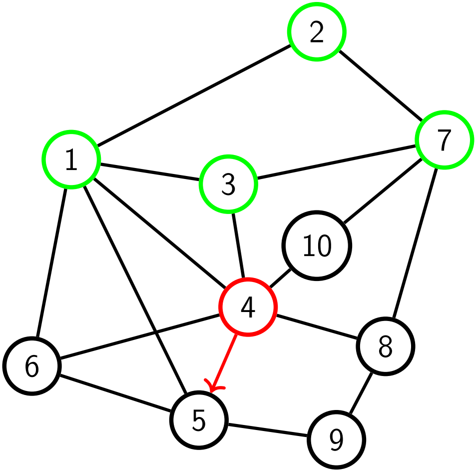


DFS : Example

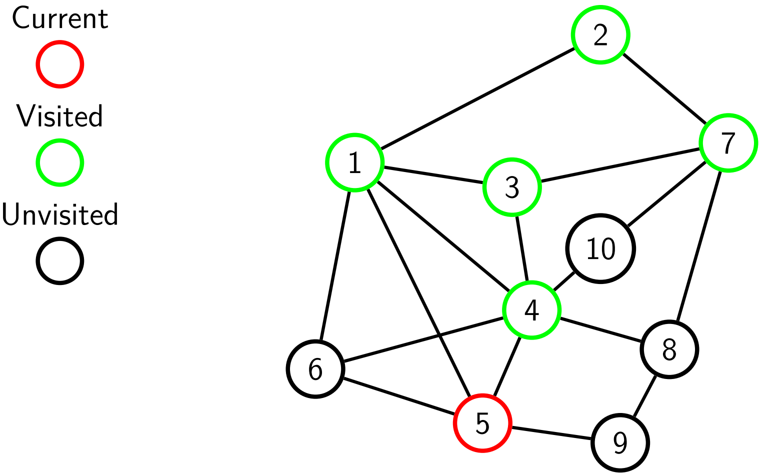


DFS : Example





DFS : Example

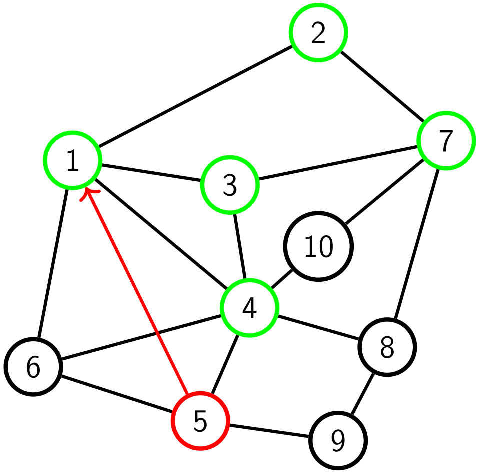


DFS : Example

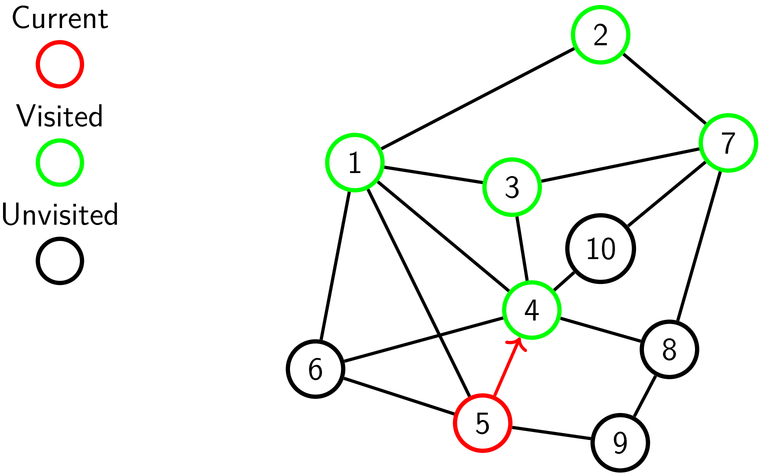
Current

Visited

Unvisited



DFS : Example

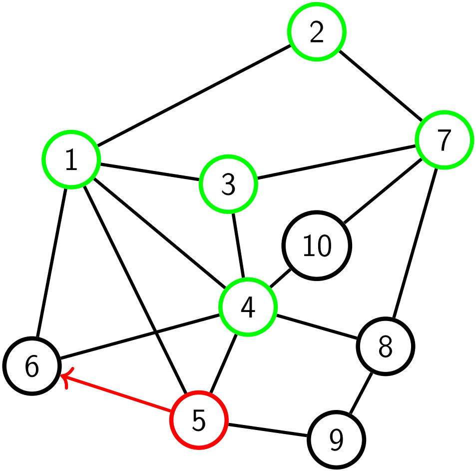


DFS : Example

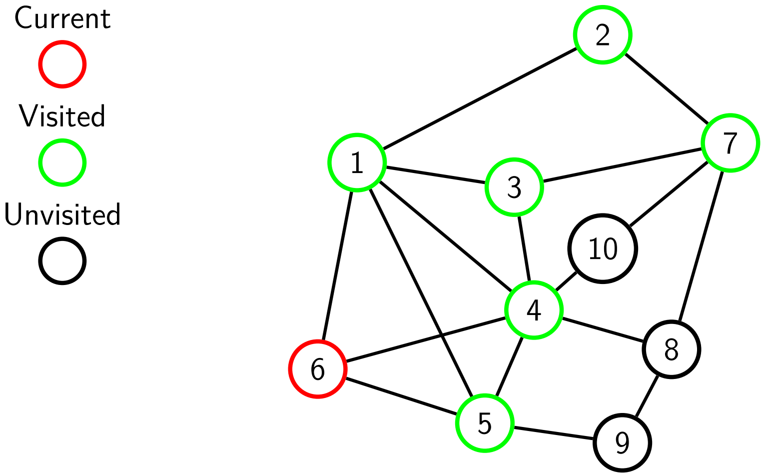
Current

Visited

Unvisited



DFS : Example

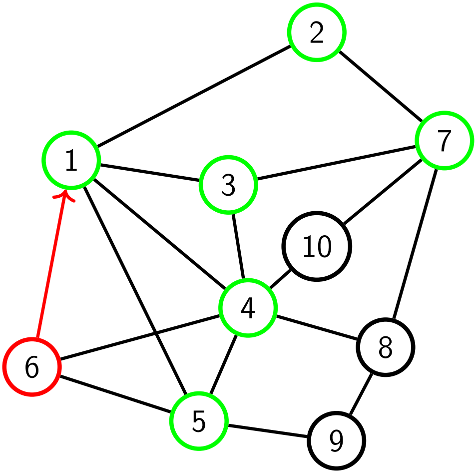


DFS : Example

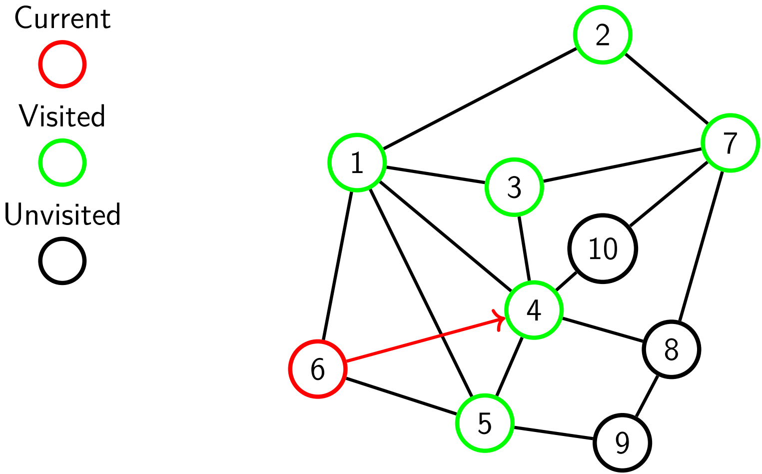
Current

Visited

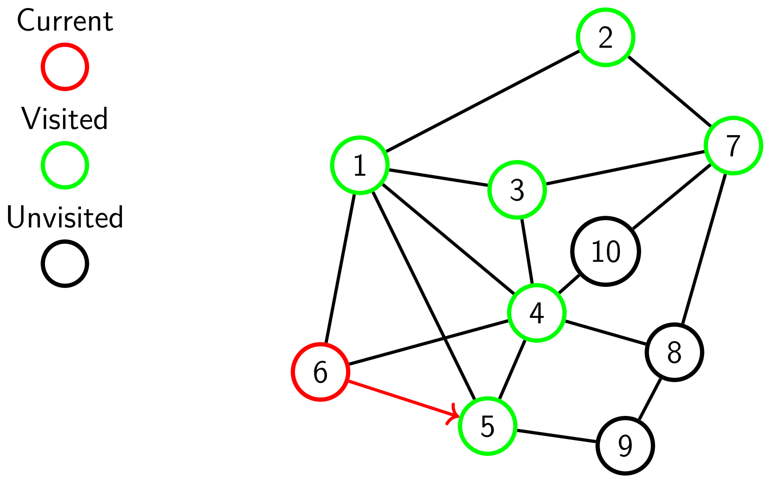
Unvisited



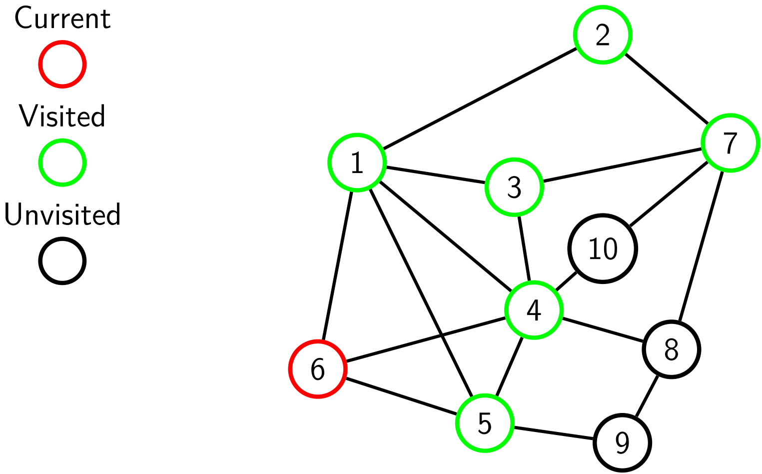
DFS : Example



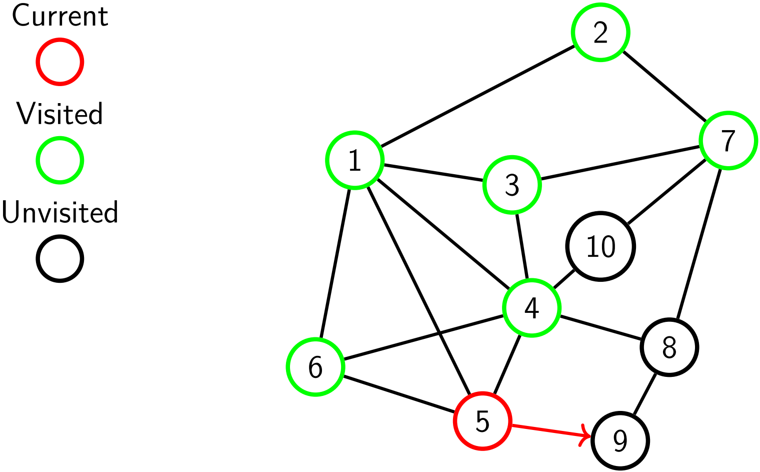
DFS : Example



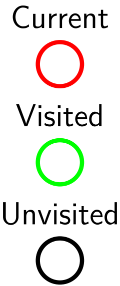
DFS : Example

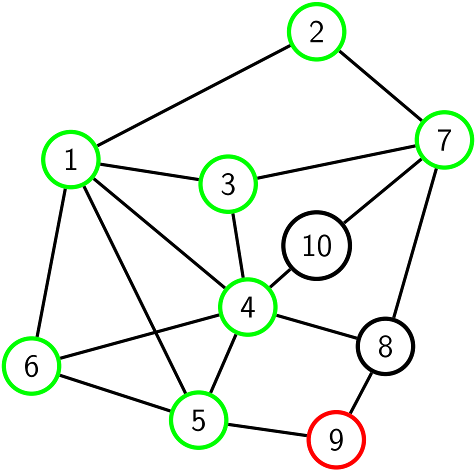


DFS : Example

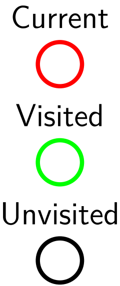


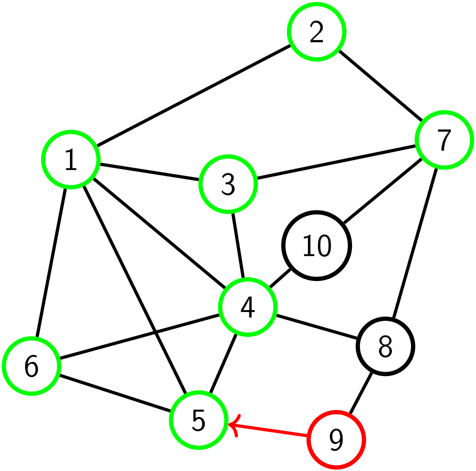
DFS : Example





DFS : Example



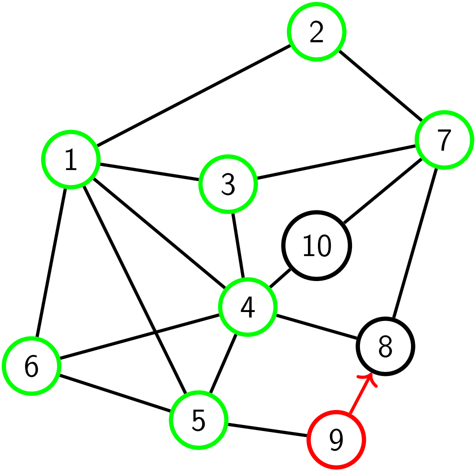


DFS : Example

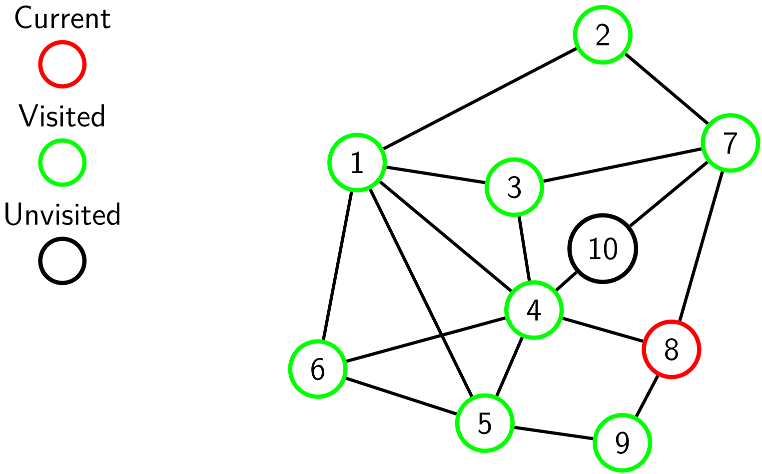
Current

Visited

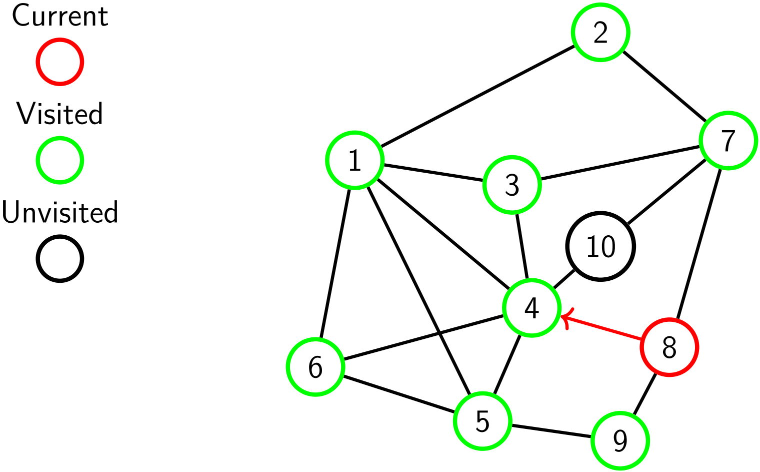
Unvisited



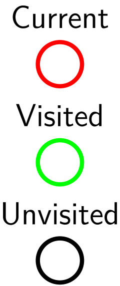
DFS : Example

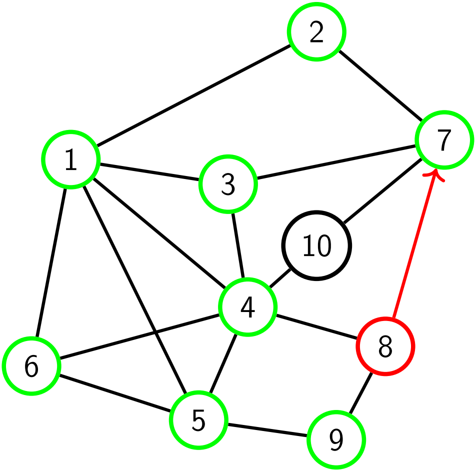


DFS : Example

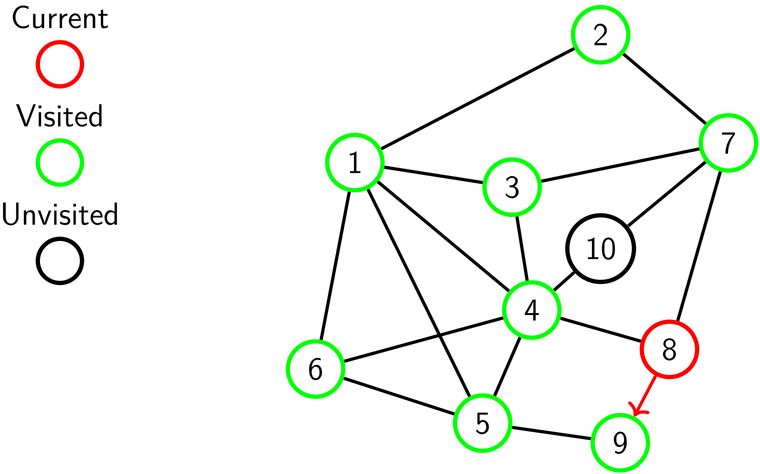


DFS : Example





DFS : Example

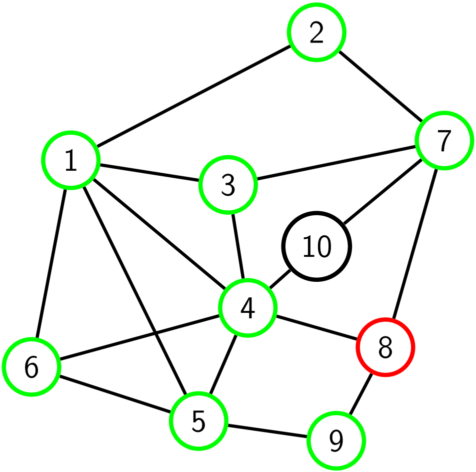


DFS : Example

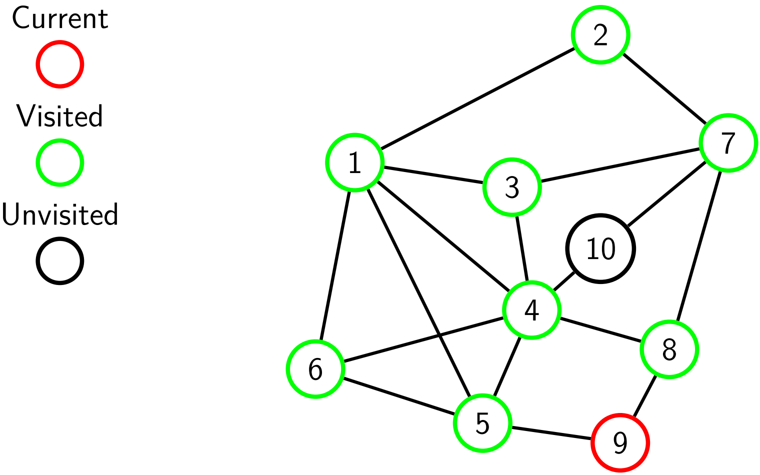
Current

Visited

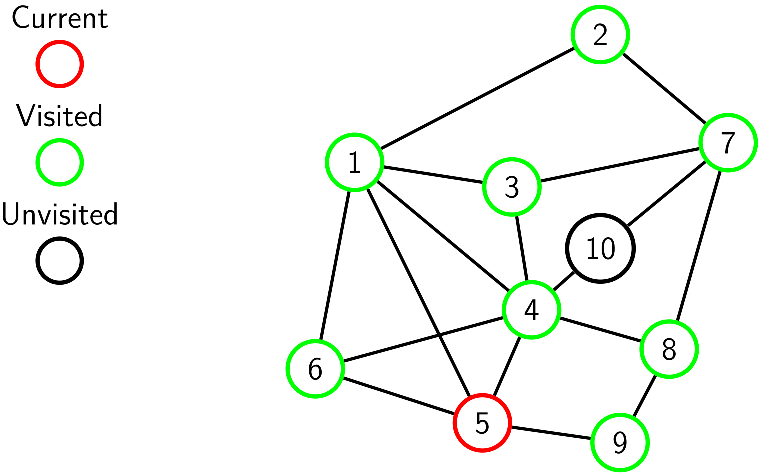
Unvisited



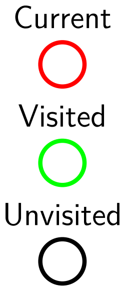
DFS : Example

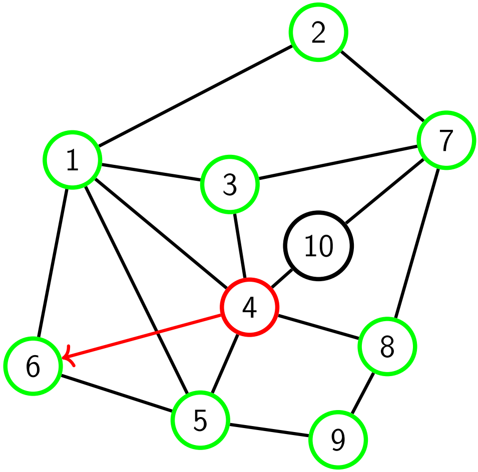


DFS : Example

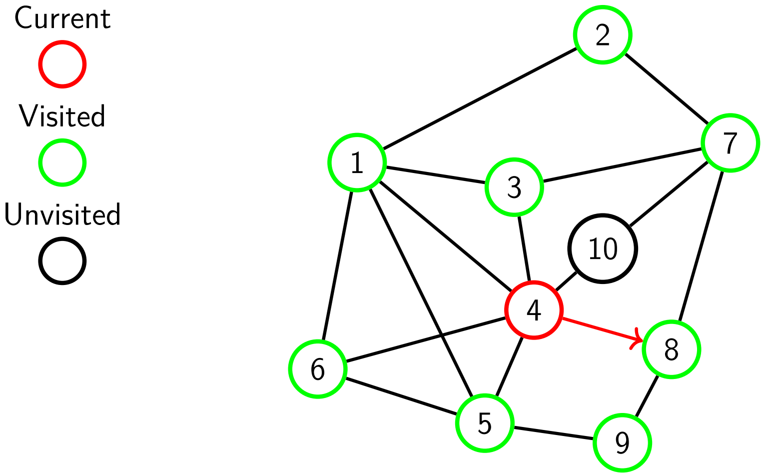


DFS : Example

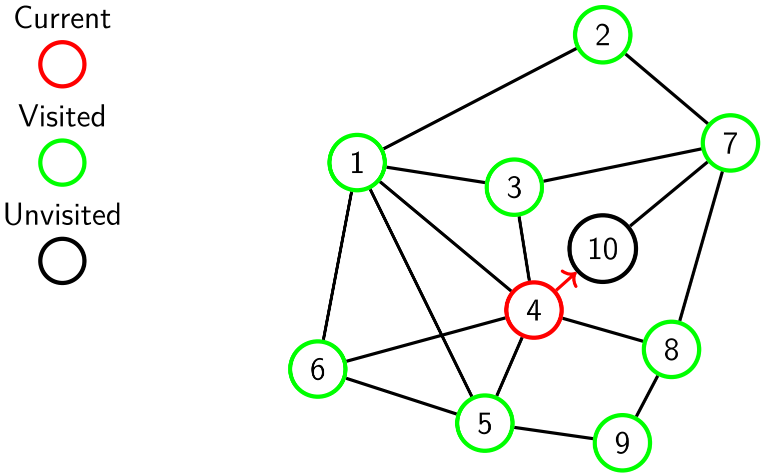




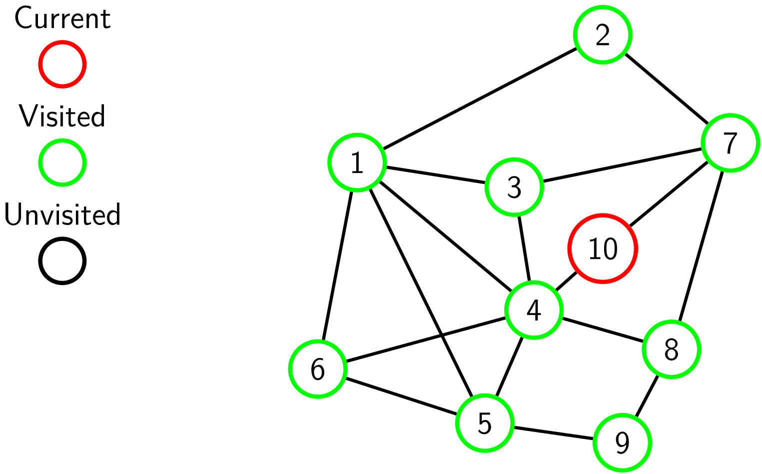
DFS : Example



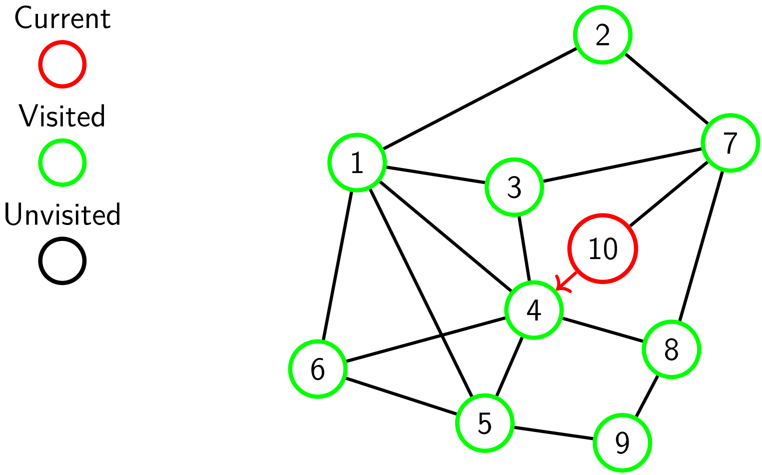
DFS : Example



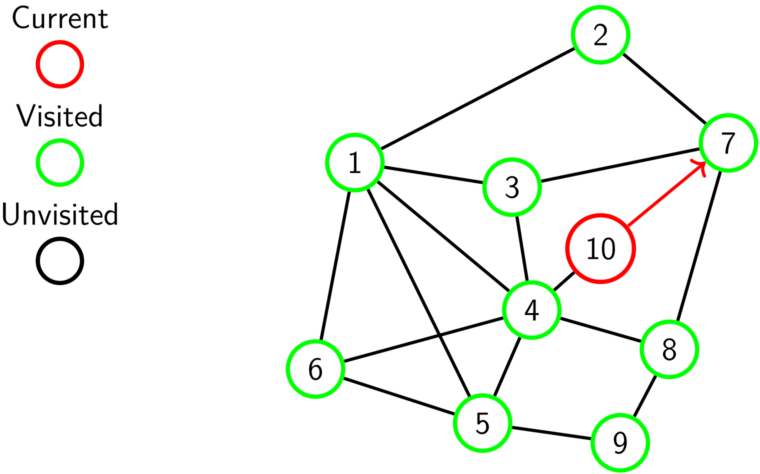
DFS : Example



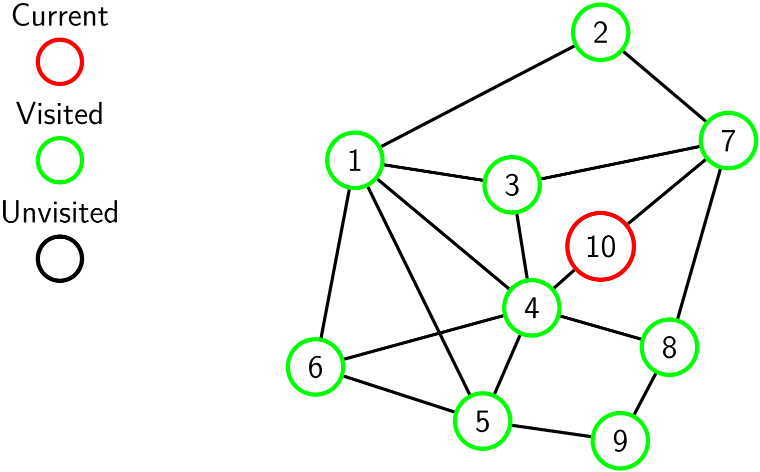
DFS : Example



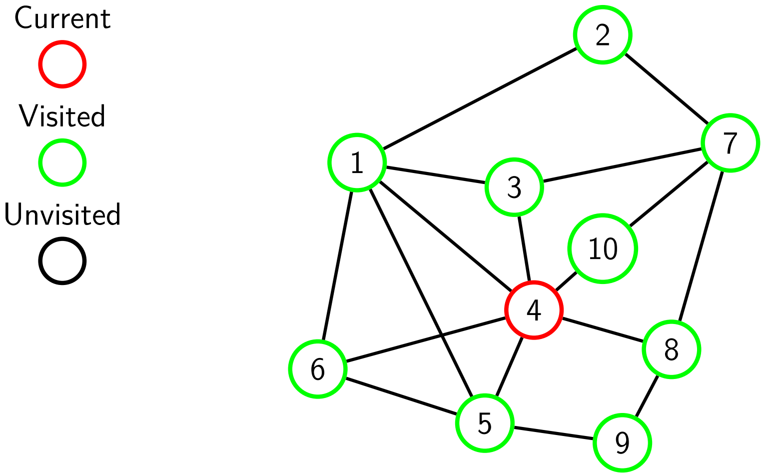
DFS : Example



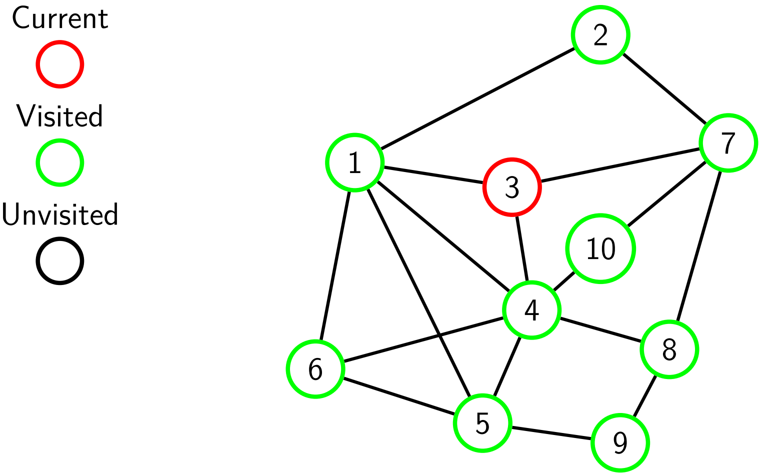
DFS : Example



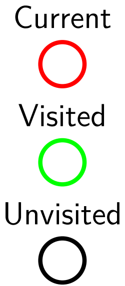
DFS : Example

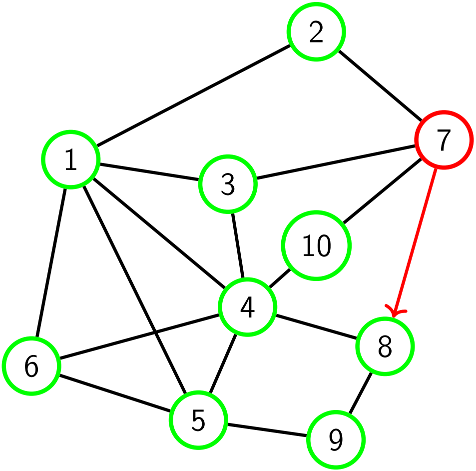


DFS : Example

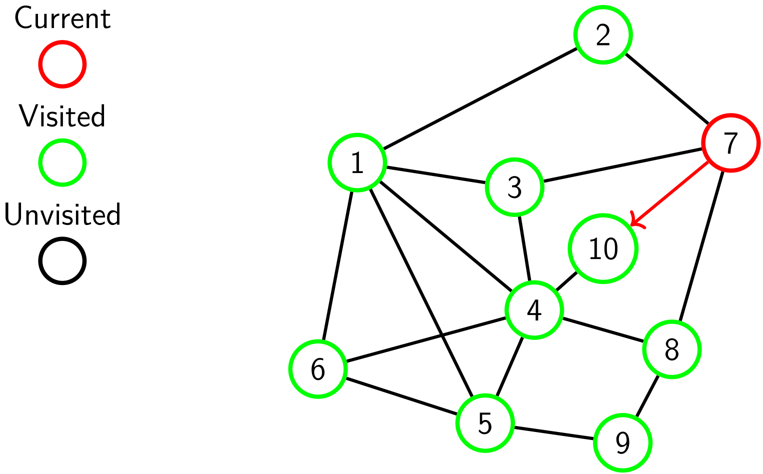


DFS : Example

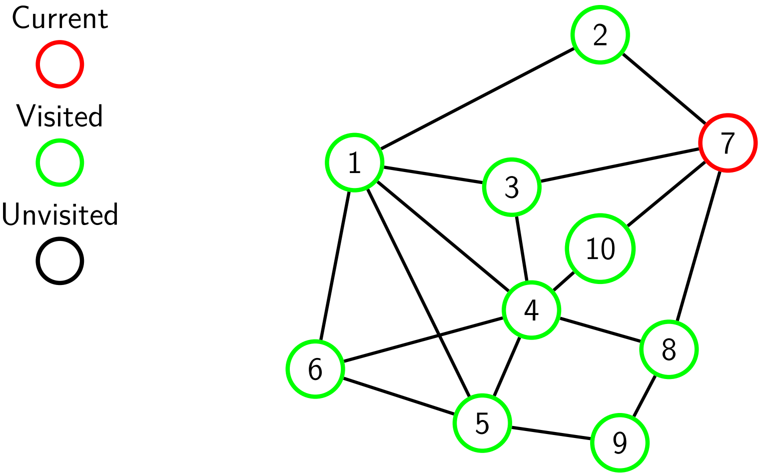




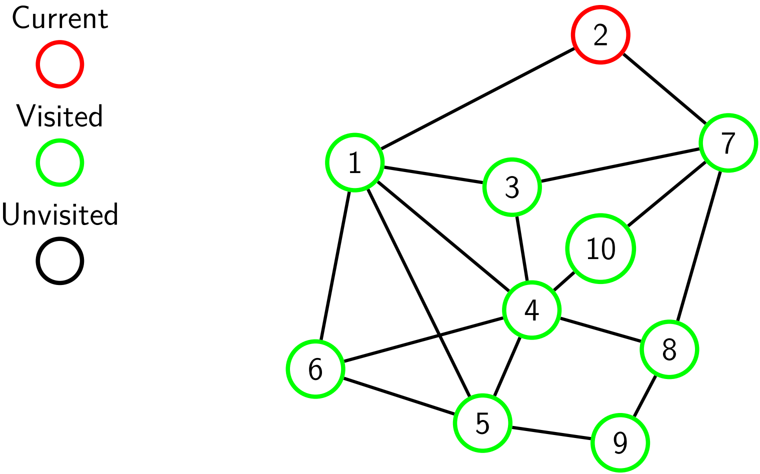
DFS : Example



DFS : Example



DFS : Example

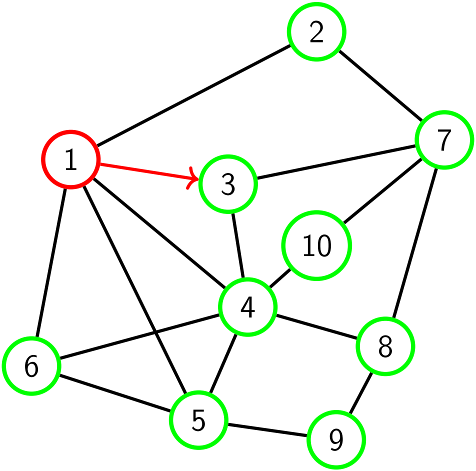


DFS : Example

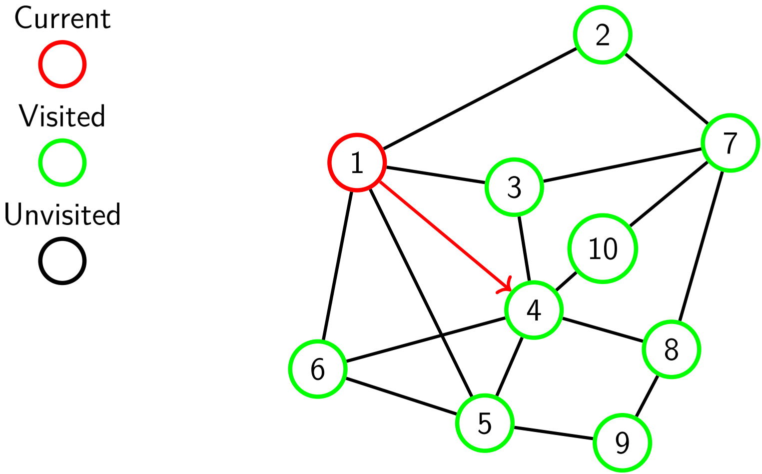
Current

Visited

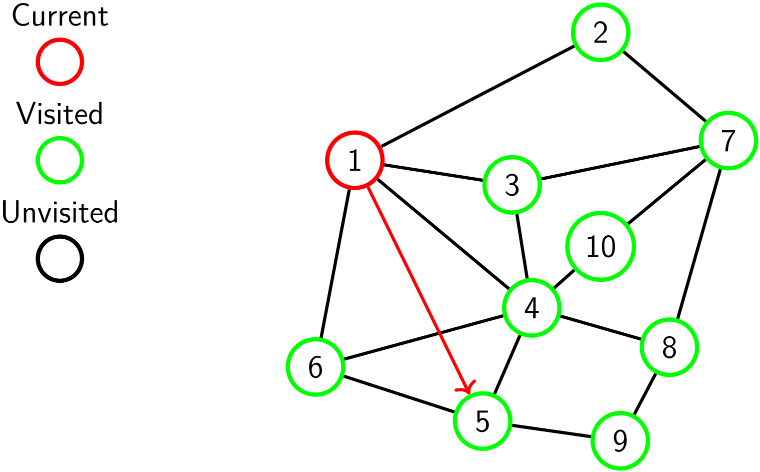
Unvisited



DFS : Example



DFS : Example

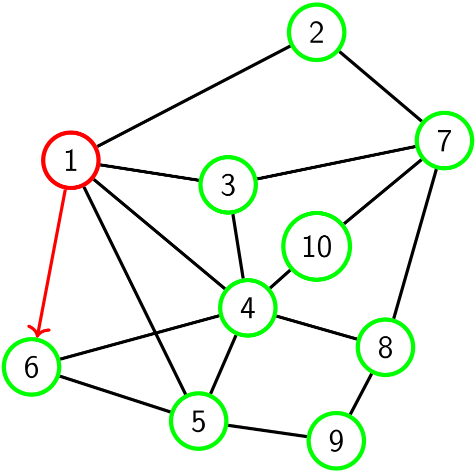


DFS : Example

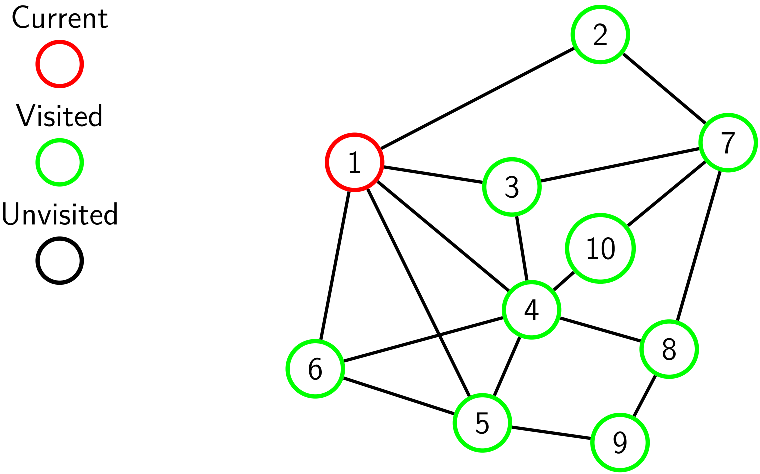
Current

Visited

Unvisited



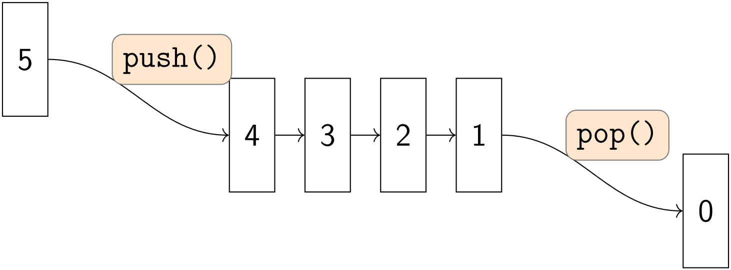
DFS : Example



What is a BFS ?

BFS stands for Breadth First Search. It is an algorithm used to compute distance and shortest path on unweighed graphs.

This algorithm uses a **queue** datastucture (first-in, first-out).



BFS : Algorithm

Star by putting the first vertex in the queue add set its distance to be 0, set all nodes as unvisited (distance $= \infty$). Then as long as the queue is not empty :

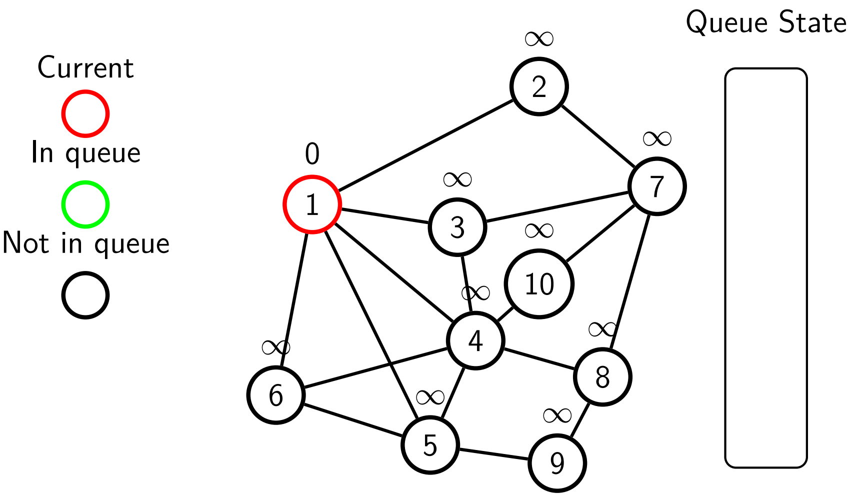
- Poll the first vertex in the queue.
- For each of its unvisited (distance $= \infty$) neighbour :
  - Otherwise, set its distance as the distance of the polled vertex $+ 1$ and add it to the queue.
  - Set the parent of the neighbour as the current vertex.

To track the shortest path, begin at the destination and go up from parent to parent until reaching the starting vertex.

BFS : Code

```
vector<vector<int>> adj;

void bfs(int n, int start) {
  vector<int> dist(n, 1e9); // 1e9 is infinity
  queue<int> q;
  // Add start
  q.push(start);
  dist[start] = 0;
  while(!q.empty()) {
    int u = q.front(); q.pop();
    // Do something with v
    for(int v : adj[u]) {
      if(dist[v] == 1e9) {
        dist[v] = dist[u] + 1;
        q.push(v);
      }
    }
  }
}
```

BFS : Example

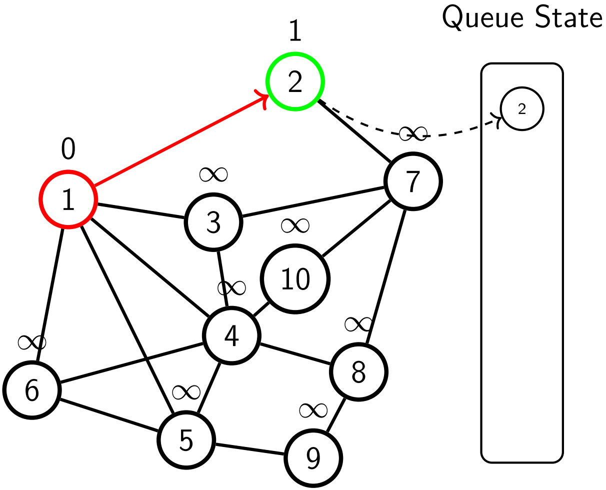


BFS : Example

Current

In queue

Not in queue

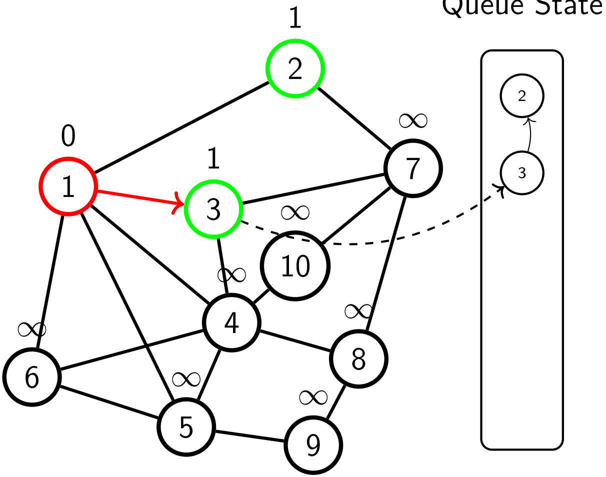


BFS : Example

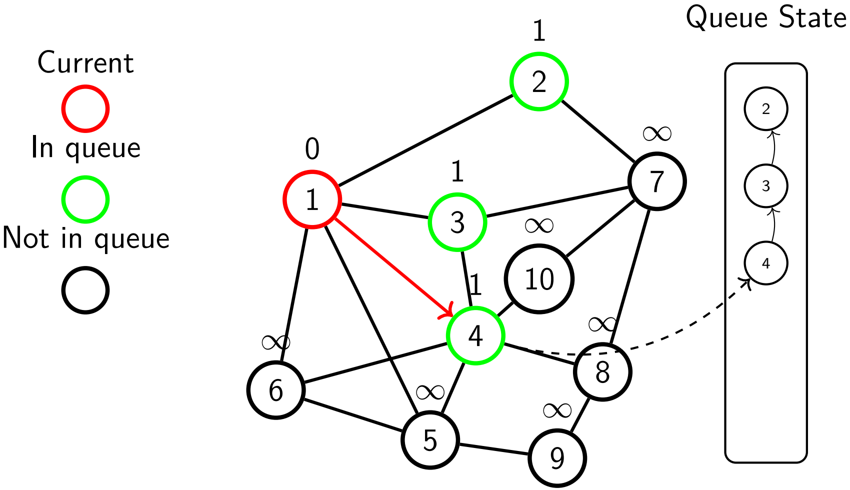
Current

In queue

Not in queue



BFS : Example

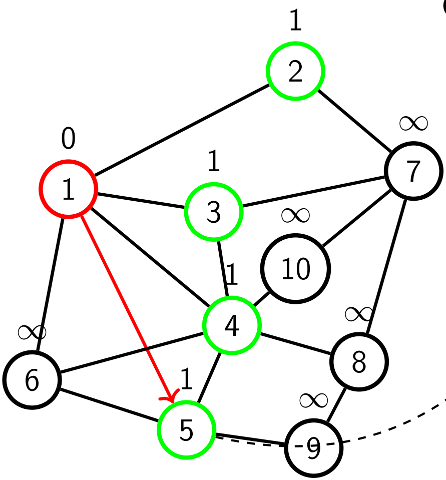

Current
In queue
Not in queue
Queue State
0
1
1
1
1
∞
∞
∞
∞
∞
∞
1
2
3
4
5
6
7
8
9
10
2
3
4

BFS : Example

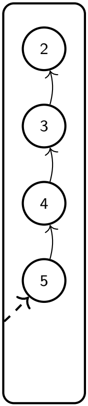
Current

In queue

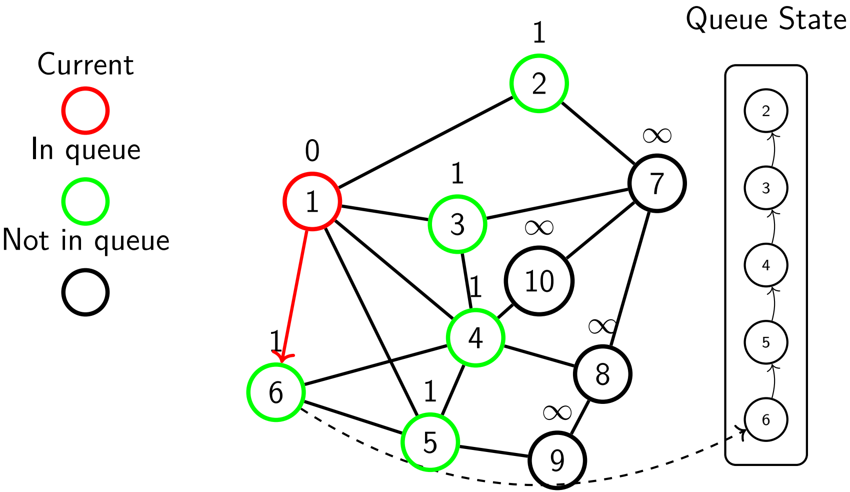
Not in queue



Queue State



BFS : Example


Current
In queue
Not in queue
Queue State
1
2
0
1
∞
1
3
7
∞
10
1
1
4
∞
6
1
8
∞
5
9
2
3
4
5
6

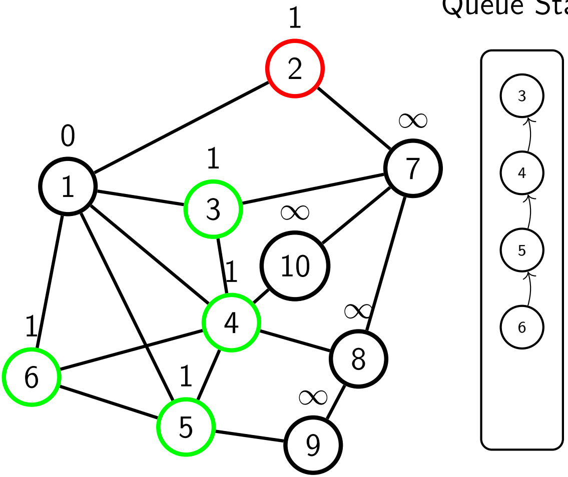
BFS : Example

Current

In queue

Not in queue

Queue State

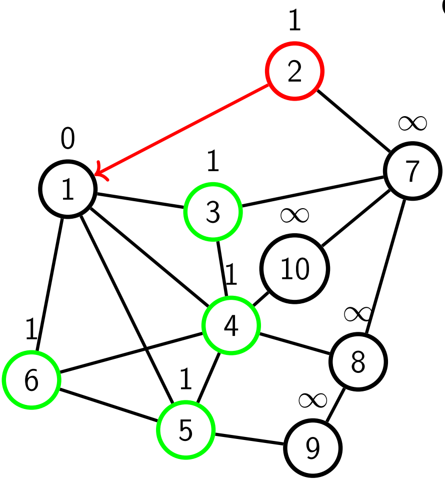


BFS : Example

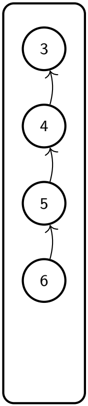
Current

In queue

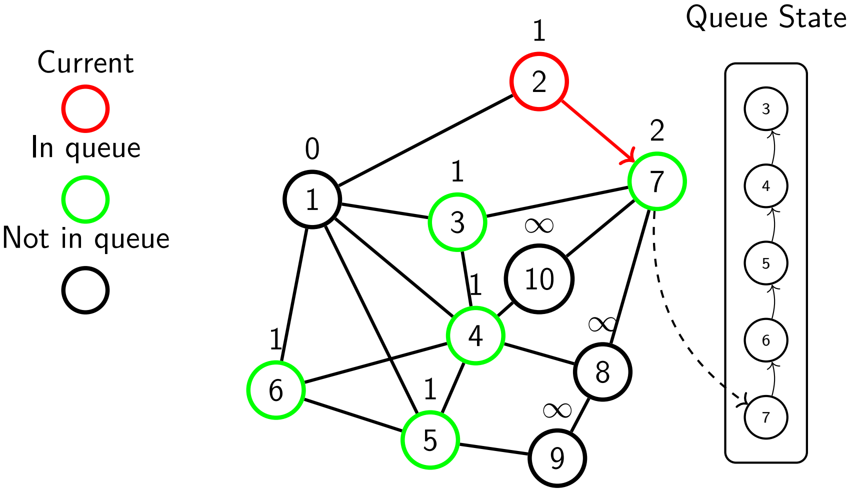
Not in queue



Queue State



BFS : Example

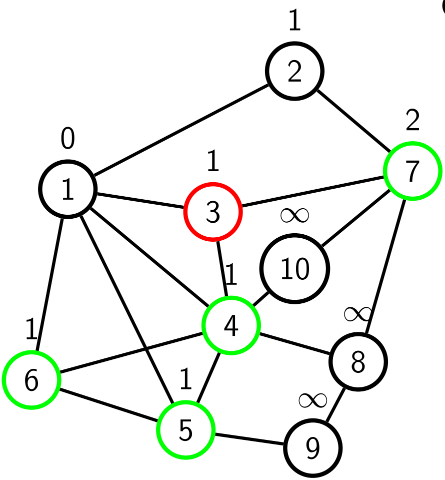

Current
In queue
Not in queue
Queue State
1
2
0
1
2
1
3
7
∞
10
1
1
4
∞
8
6
1
5
∞
9
3
4
5
6
7

BFS : Example

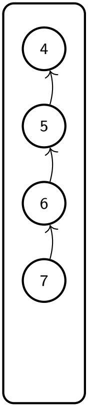
Current

In queue

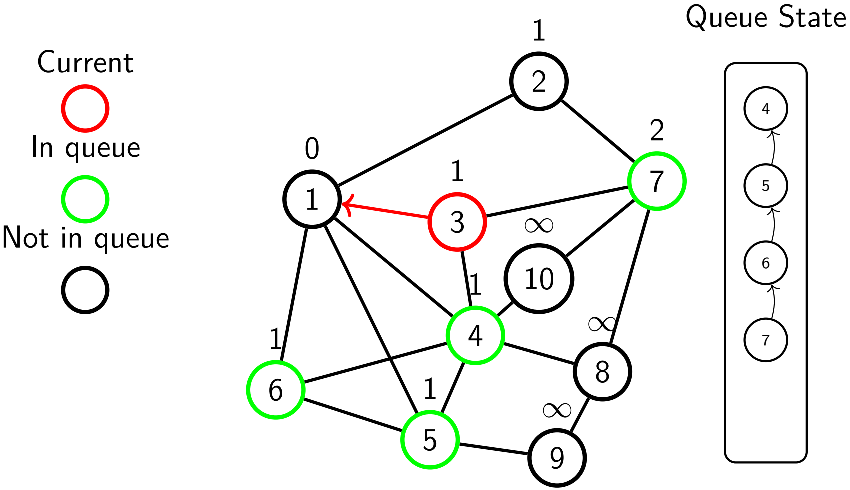
Not in queue



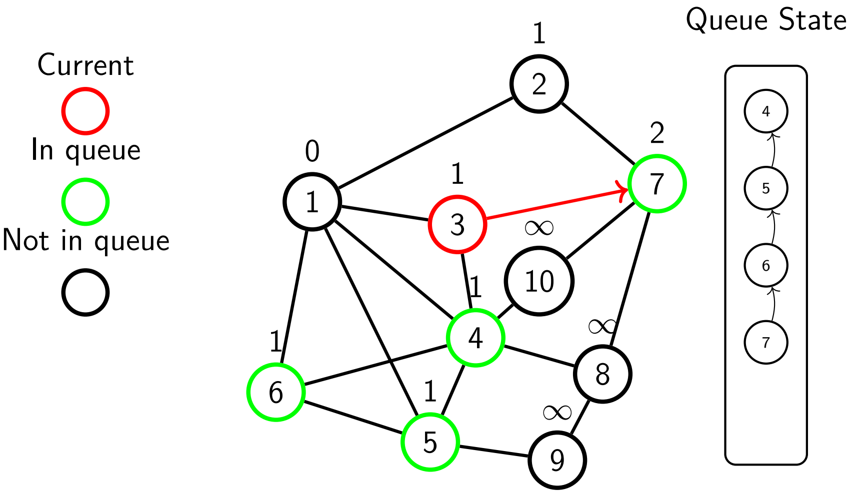
Queue State



BFS : Example


Current
In queue
Not in queue
0
1
1
2
1
1
1
∞
∞
∞
1
2
3
4
5
6
7
8
9
10
Queue State
4
5
6
7

BFS : Example

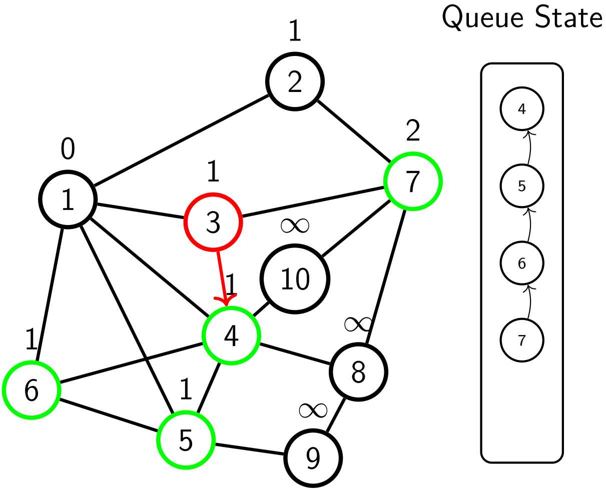

Current
In queue
Not in queue
Queue State
0
1
1
2
1
1
∞
∞
∞
1
2
3
4
5
6
7
8
9
10
4
5
6
7

BFS : Example

Current

In queue

Not in queue

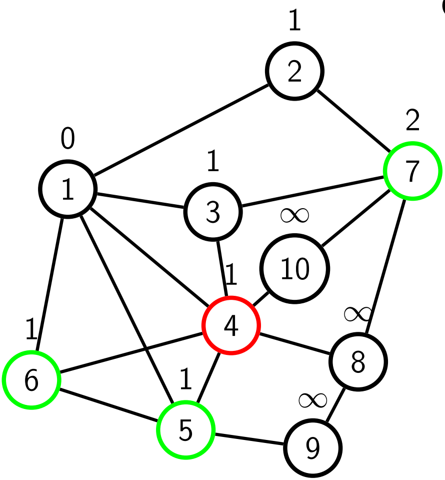


BFS : Example

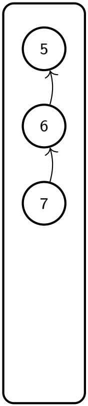
Current

In queue

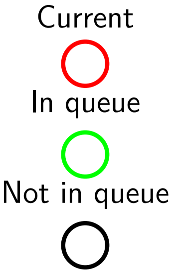
Not in queue

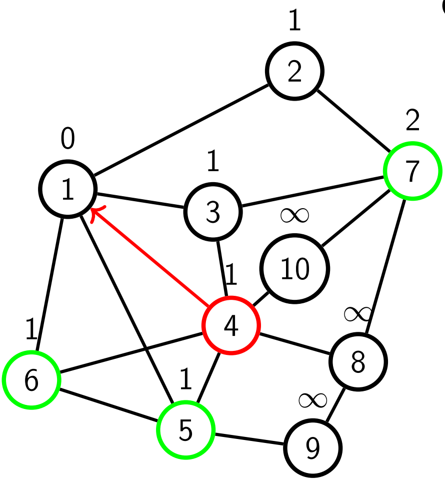


Queue State

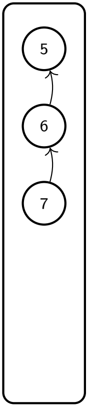


BFS : Example

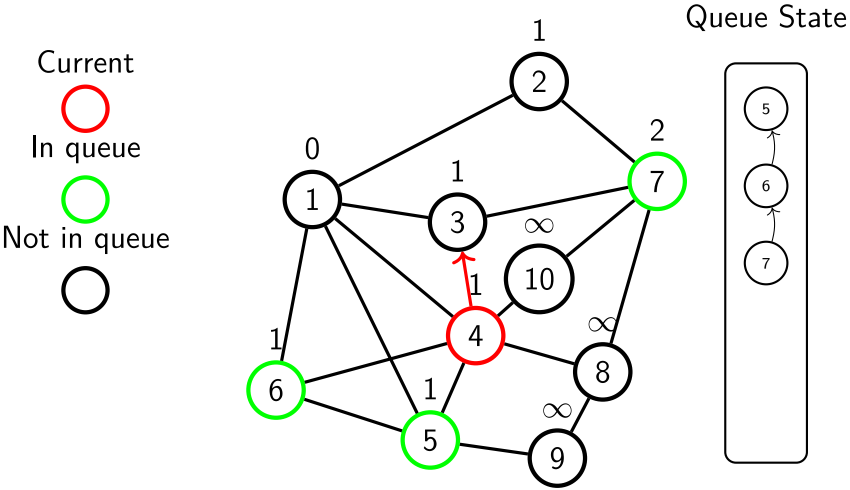




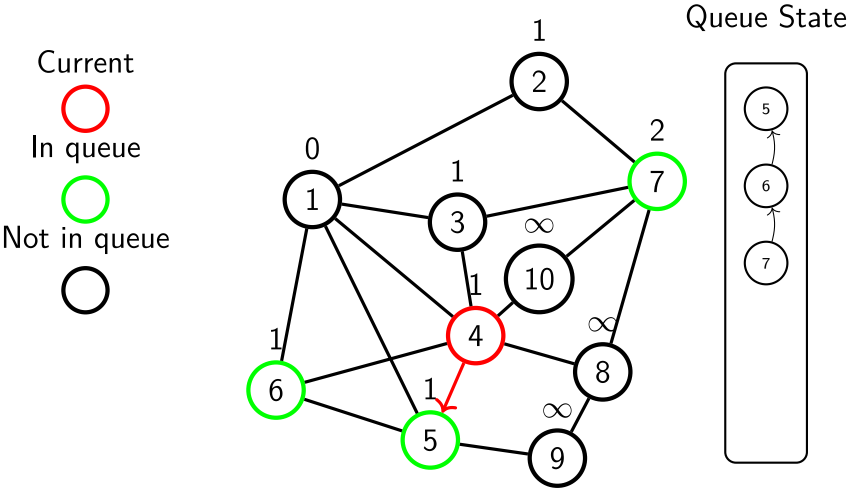
Queue State



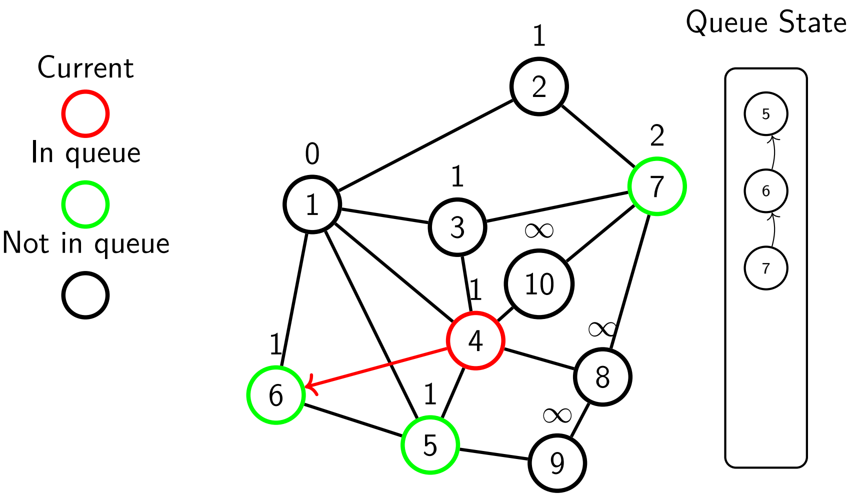
BFS : Example


Current
In queue
Not in queue
0
1
1
2
1
1
1
∞
∞
∞
1
2
3
4
5
6
7
8
9
10
Queue State
5
6
7

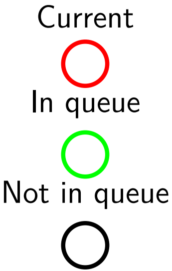
BFS : Example

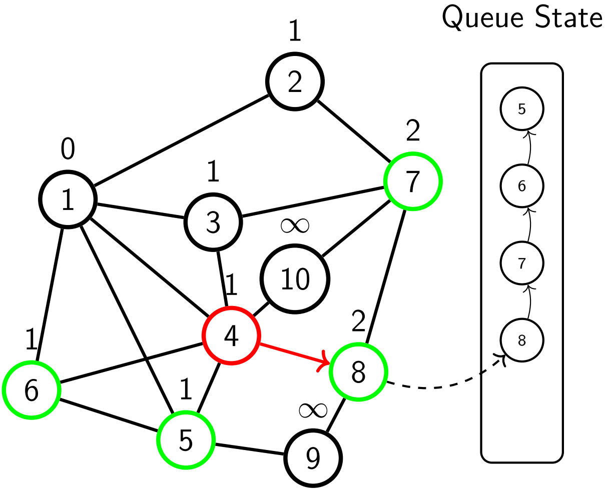

Current
In queue
Not in queue
1
2
2
7
0
1
1
3
∞
10
1
4
∞
1
6
1
5
8
∞
9
Queue State
5
6
7

BFS : Example

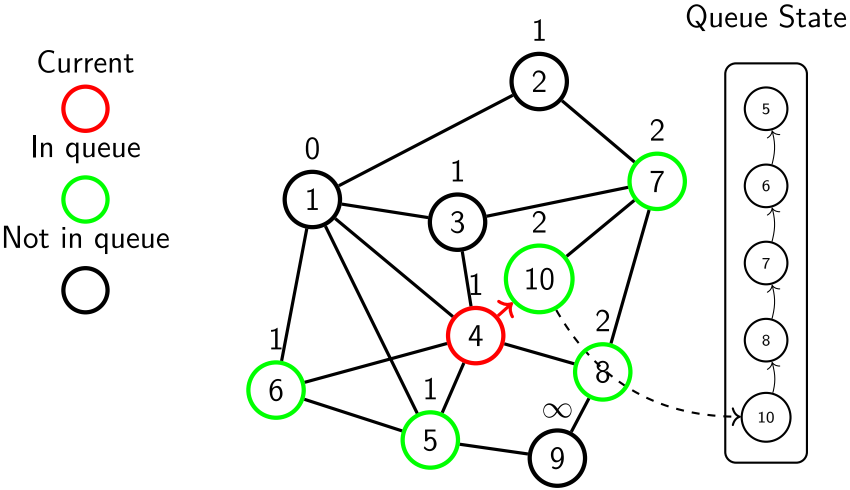

Current
In queue
Not in queue
Queue State
1
2
2
7
0
1
1
3
∞
10
1
4
∞
8
1
6
1
5
∞
9
5
6
7

BFS : Example

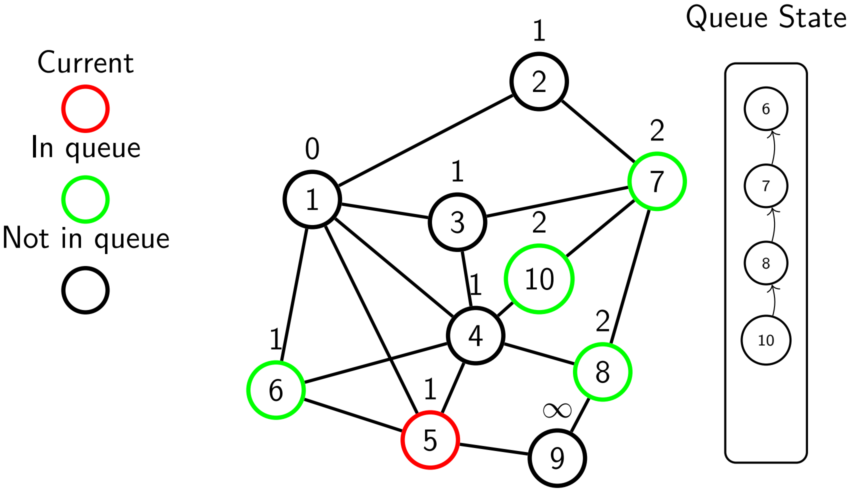

Current
In queue
Not in queue


Queue State
1
2
2
0
1
1
∞
1
1
2
1
∞
5
6
7
8
3
10
4
9
5
6
7
8

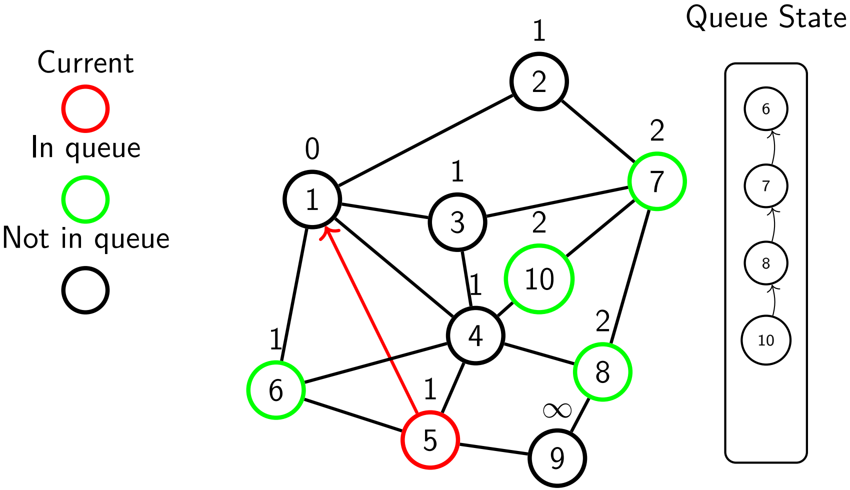
BFS : Example


Current
In queue
Not in queue
Queue State
0
1
1
2
1
3
2
7
2
10
1
4
2
8
1
6
1
5
∞
9
5
6
7
8
10

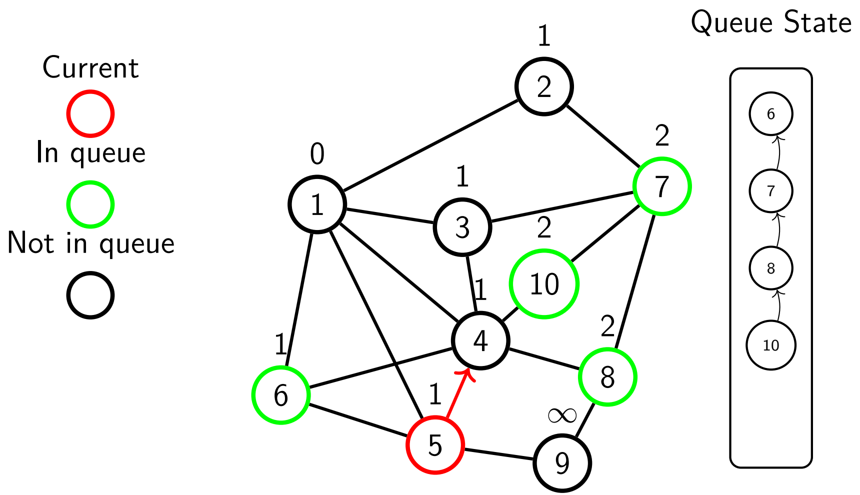
BFS : Example


Current
In queue
Not in queue
Queue State
0
1
1
2
2
2
1
1
1
2
∞
1
2
3
4
5
6
7
8
9
10
6
7
8
10

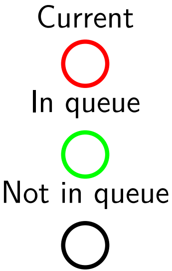
BFS : Example

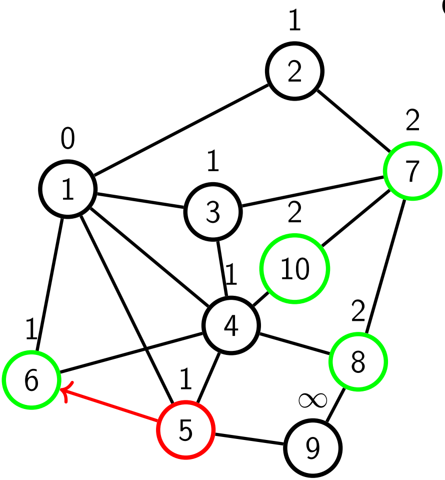

Current
In queue
Not in queue
1
2
0
1
3
2
7
2
10
1
1
4
2
8
1
6
1
5
∞
9
Queue State
6
7
8
10

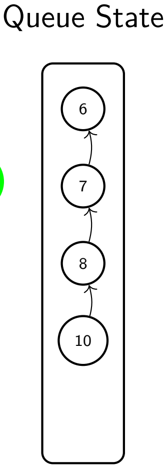
BFS : Example


Current
In queue
Not in queue
0
1
1
2
1
2
3
2
7
10
1
1
4
2
1
6
8
1
5
∞
9
Queue State
6
7
8
10

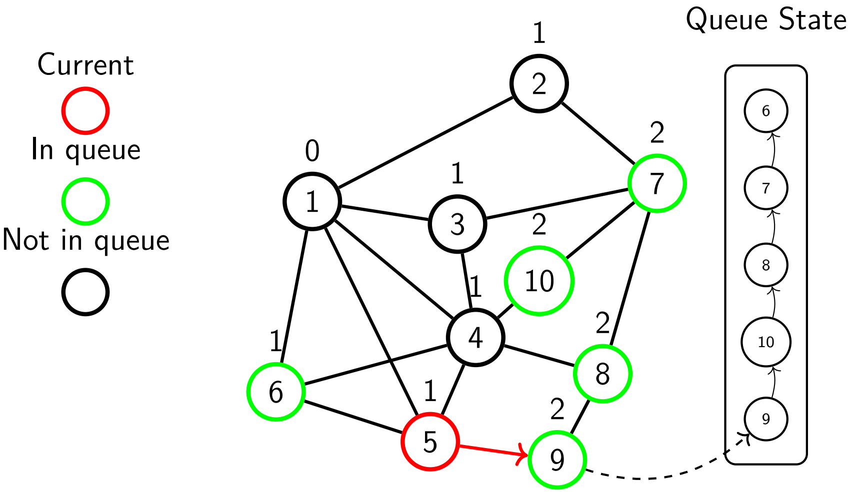
BFS : Example


Current
In queue
Not in queue

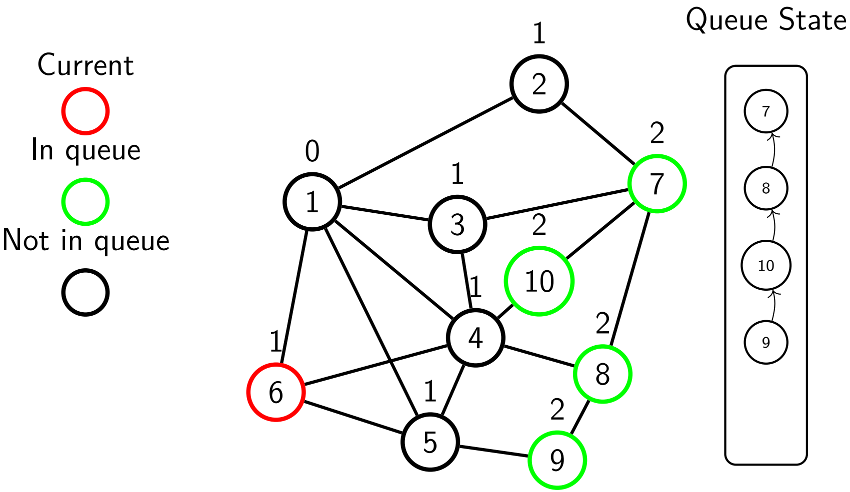

1
2
0
1
1
3
2
7
2
10
1
1
4
2
6
1
8
∞
5
9
2
1


Queue State
6
7
8
10

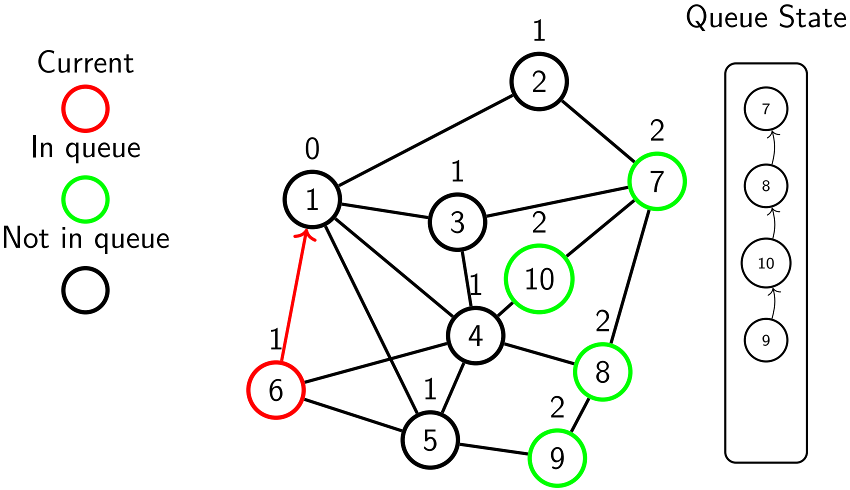
BFS : Example


Current
In queue
Not in queue
Queue State
0
1
1
2
2
1
2
1
1
2
2
1
1
2
3
4
5
6
7
8
9
10
6
7
8
10
9

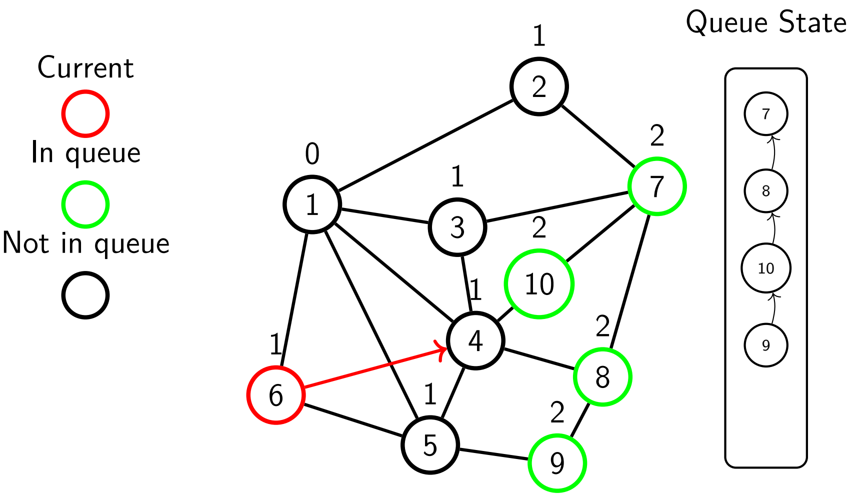
BFS : Example


Current
In queue
Not in queue
0
1
1
2
1
3
2
7
2
10
1
4
2
8
1
6
1
5
2
9
Queue State
7
8
10
9

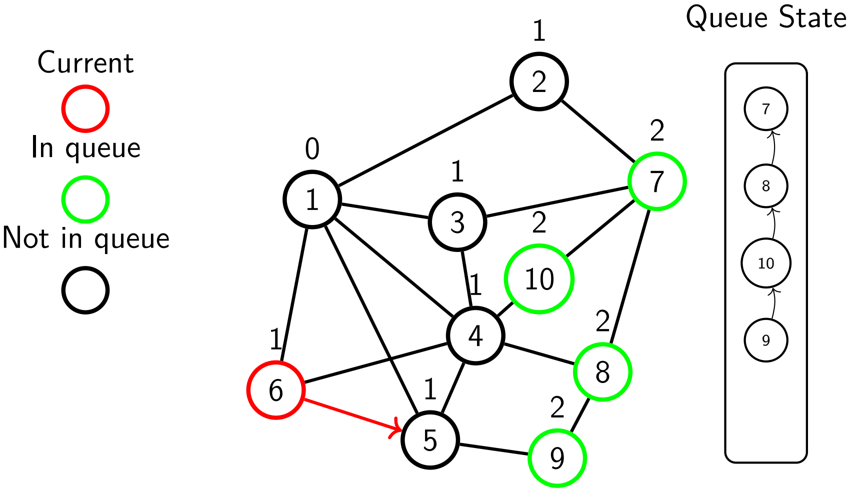
BFS : Example


Current
In queue
Not in queue
0
1
1
2
1
3
2
7
2
10
1
4
2
8
1
6
1
5
2
9
Queue State
7
8
10
9

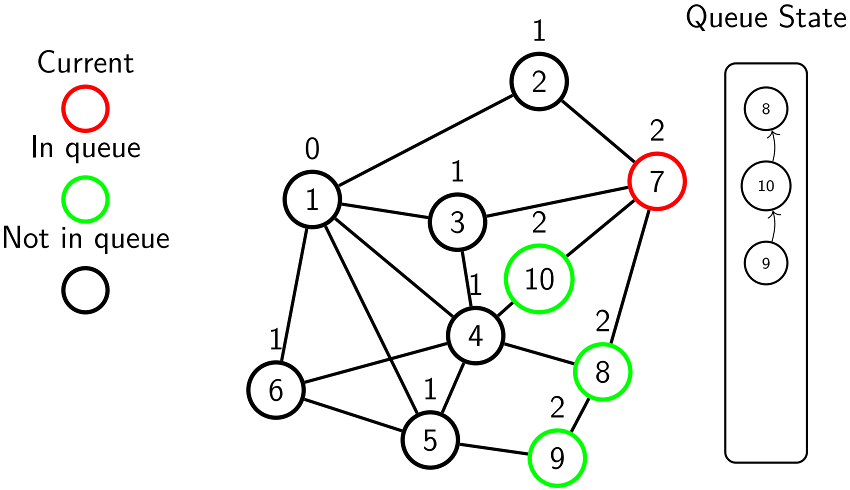
BFS : Example


Current
In queue
Not in queue
1
2
0
1
2
1
7
3
2
10
1
4
1
2
6
1
8
5
2
9
Queue State
7
8
10
9

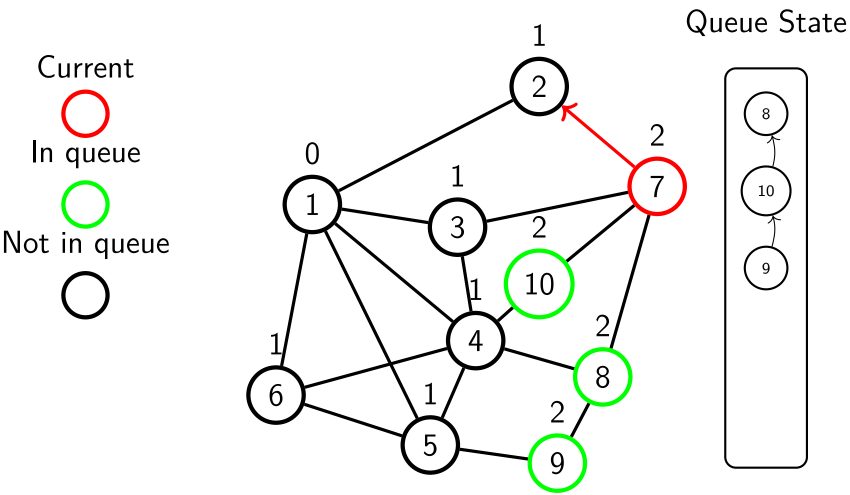
BFS : Example


Current
In queue
Not in queue
0
1
1
2
1
3
2
2
7
10
1
4
2
8
1
6
1
5
2
9
Queue State
7
8
10
9

BFS : Example


Current
In queue
Not in queue
0
1
1
2
3
4
5
6
7
8
9
10
1
1
1
1
1
2
2
2
2
Queue State
8
10
9

BFS : Example

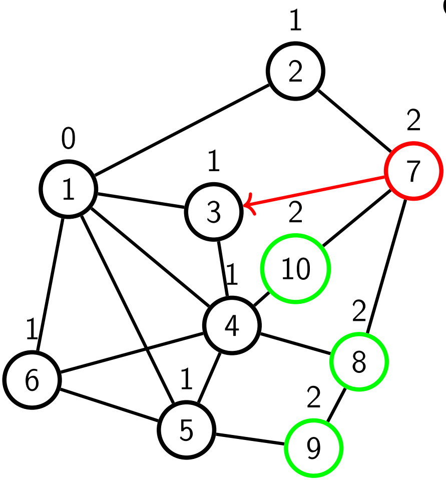

Current
In queue
Not in queue
0
1
1
2
1
2
2
10
3
1
4
1
2
8
6
1
2
5
9
7
Queue State
8
10
9

BFS : Example

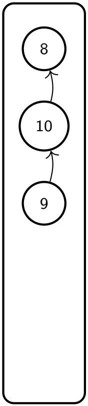
Current

In queue

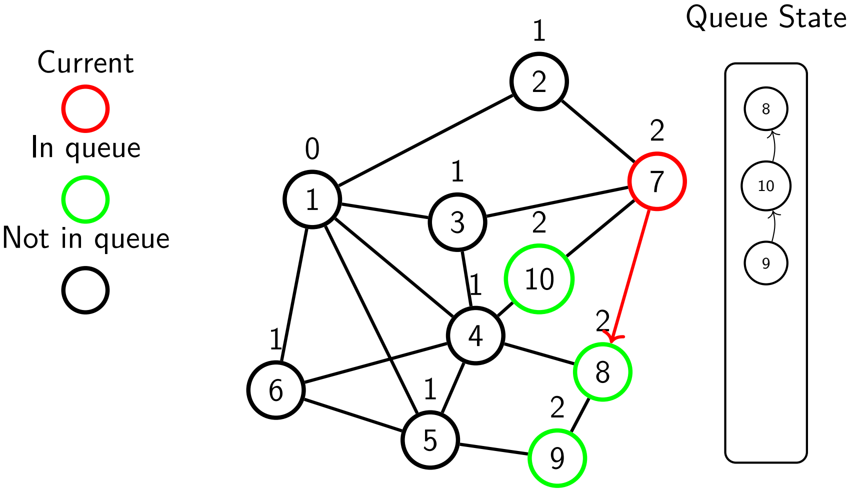
Not in queue



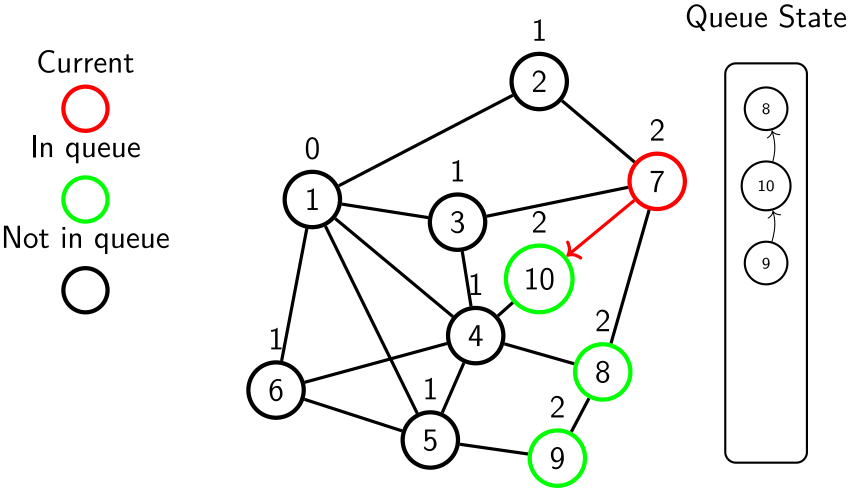
Queue State



BFS : Example


Current
In queue
Not in queue
Queue State
0
1
1
1
1
1
1
2
2
2
2
1
2
3
4
5
6
7
8
9
10
8
10
9

BFS : Example

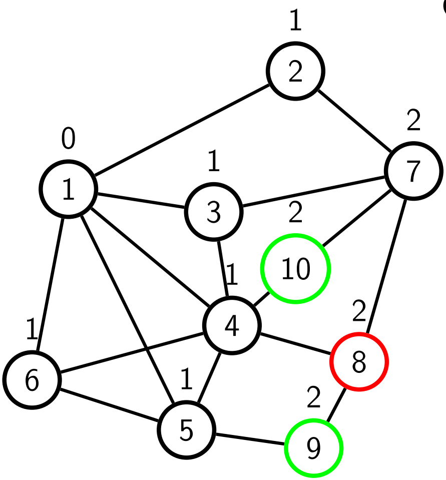

Current
In queue
Not in queue
Queue State
0
1
1
2
3
1
2
7
2
10
2
1
4
1
6
1
5
8
2
2
9
8
10
9

BFS : Example

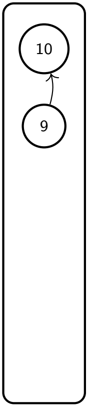
Current

In queue

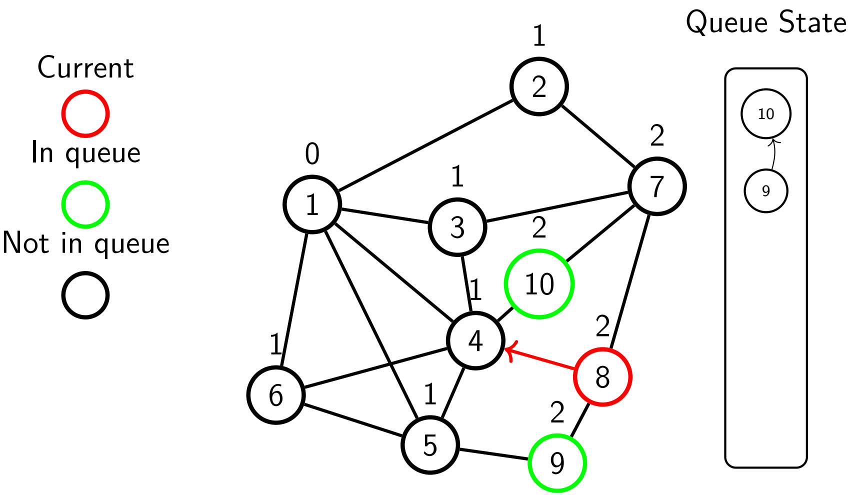
Not in queue



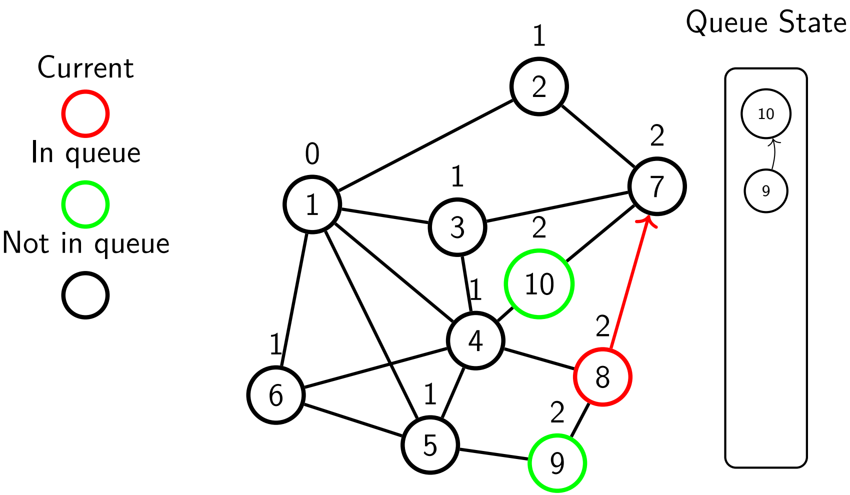
Queue State



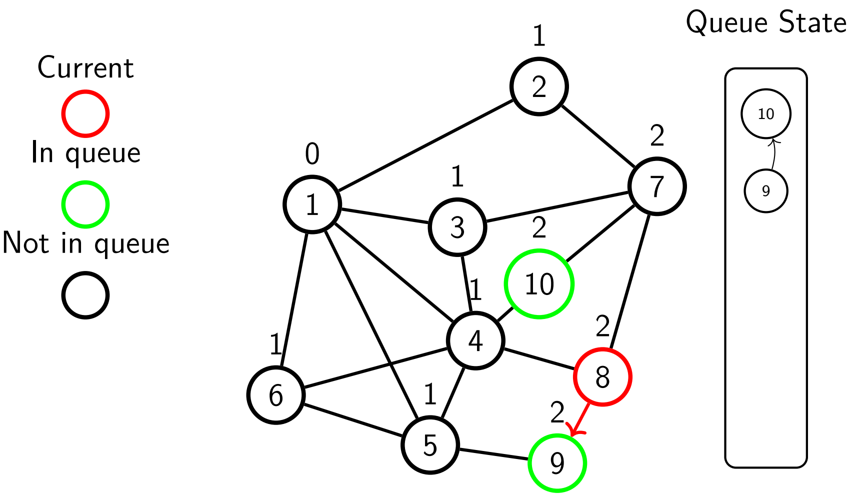
BFS : Example


Current
In queue
Not in queue
1
2
2
7
0
1
1
3
2
10
1
4
2
8
1
6
1
5
2
9
Queue State
10
9

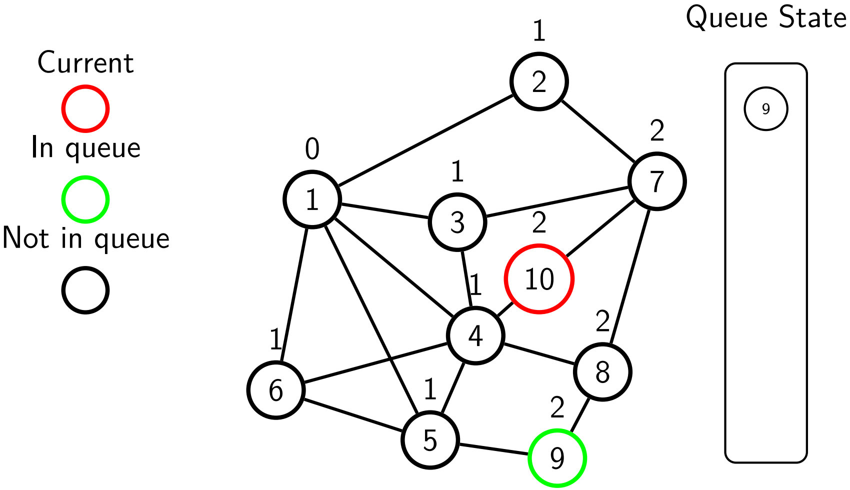
BFS : Example


Current
In queue
Not in queue
0
1
1
2
1
2
1
2
2
1
1
2
3
4
5
6
7
8
9
10
Queue State
10
9

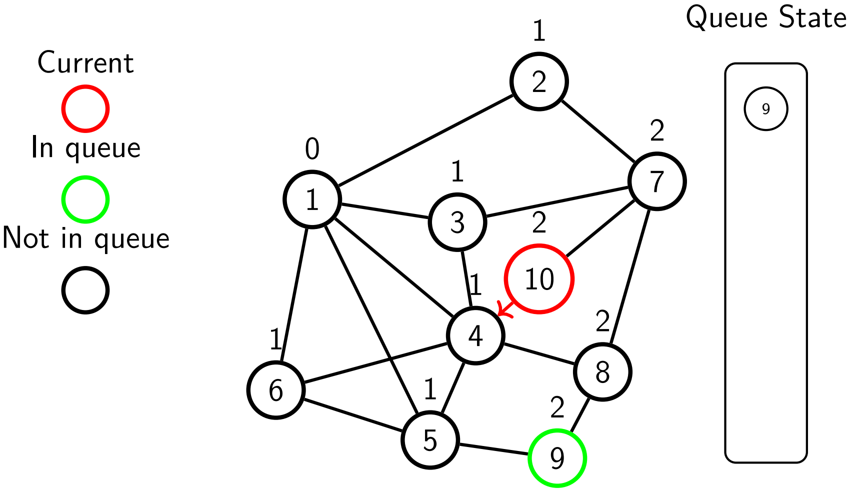
BFS : Example


Current
In queue
Not in queue
0
1
1
2
2
1
1
2
2
1
2
3
4
5
6
7
8
9
10
Queue State
10
9

BFS : Example



BFS : Example

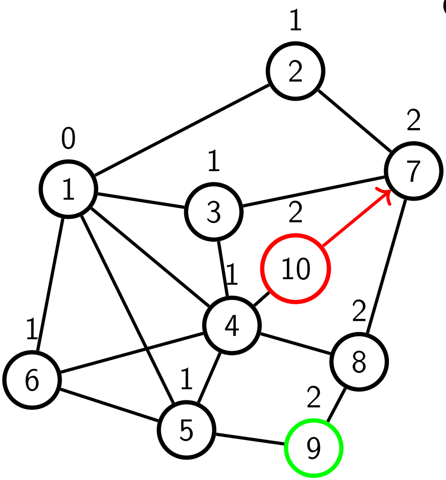

Current
In queue
Not in queue
0
1
1
2
2
1
1
1
2
2
1
2
3
4
5
6
7
8
9
10
Queue State
9

BFS : Example

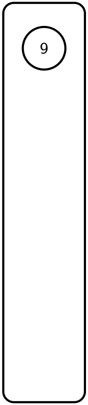
Current

In queue

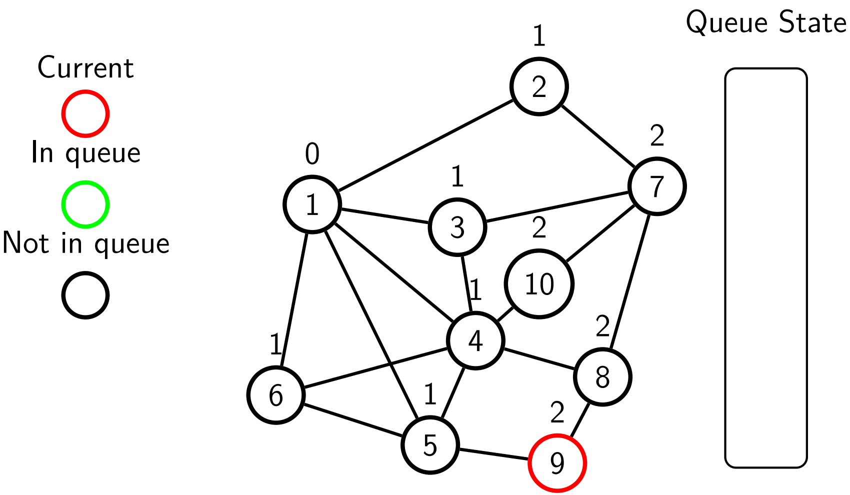
Not in queue



Queue State



BFS : Example

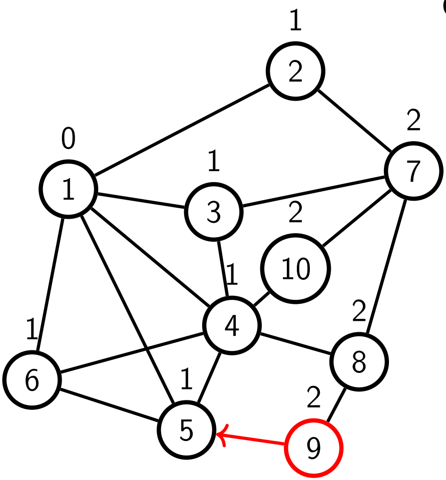

Current
In queue
Not in queue
Queue State
1
2
0
1
2
7
1
3
2
10
1
4
1
6
2
8
1
5
2
9

BFS : Example

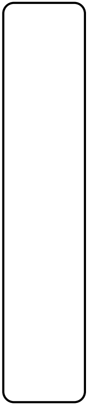
Current

In queue

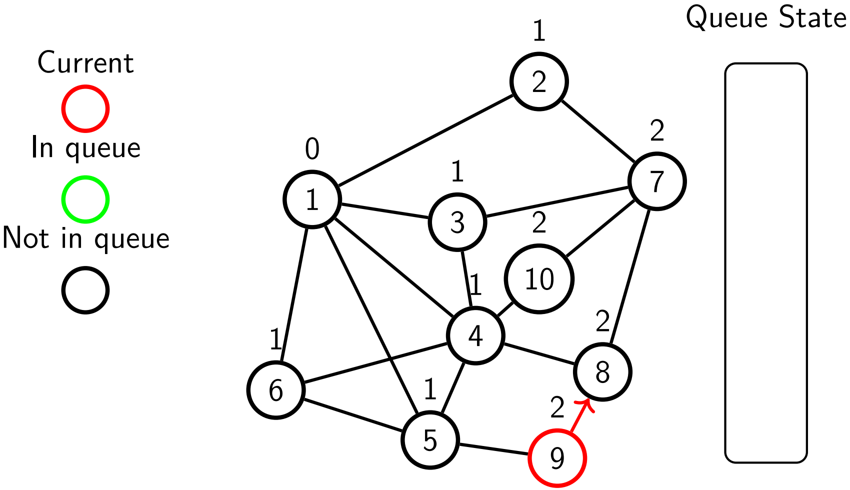
Not in queue



Queue State



BFS : Example


Current
In queue
Not in queue
0
1
1
2
3
7
2
1
10
2
1
4
2
8
1
6
1
5
2
9
Queue State