DFS:

Public class DepthFirstPath{

Private boolean[] marked;

Private int[] edgeTo;

Public void DepthFirstPath(){

... // Initialize the graph

Marked = new boolean[G.v()];

edgeTo = new int[G.v()];

For(int v = 0; v < G.v(); v++){

If(!marked[v]){

DFS(G, v);

}

}

}

Public void DFS(Graph G, int v){

Marked[v] = true;

For(int w : G.adj(v)){

If(!marked[w]){

DFS(G, w);

edgeTo[w] = v;

}

}

}

}

BFS:

Public class BreadthFirstPath{

Private boolean[] marked;

Private int[] edgeTo;

Public void BreadthFirstPath(Graph G, int s){

... // Initialize Graph

Queue<Integer> q = new Queue<>();

Marked[s] = true;

q.enqueue(s);

While(!q.isEmpty()){

Int v = q.dequeue();

For(int w : G.adj(v)){

If(!marked[w]){

Marked[w] = true;

q.enqueue(w);

edgeTo(w) = v;

}

}

}

}

}

DFS:

Public class DepthFirstPathes{

Private boolean[] marked;

Private int[] edgeTo;

Public void DepthFirstPathes(){

... // Initialize the graph

Marked = new boolean[G.v()];

edgeTo = new int[G.v()];

For(int v = 0; v < G.v(); v ++){

If(!marked[v]){

Dfs(G, v);

}

}

}

Private void dfs(Graph G, int v){

Marked[v] = true;

For(int w : G.adj(v)){

If(!marked[w]){

Dfs(G, w);

edgeTo[w] = true;

}

}

}

}

BFS:

Public class BreadFirstPathes{

Private boolean[] marked;

Private int[] edgeTo;

Public void BreadFirstPathes(Graph G, int s){

... // Initialize the graph

Marked = new boolean[G.v()];

edgeTo = new int[G.v()];

Queue<Integer> q = new Queue<>();

Marked[s] = true;

Q.enqueue(s);

While(!q.isEmpty()){

Int v = q.dequeue();

For(int w : G.adj(v)){

If(!marked[w]){

Marked[w] = true;

Q.enqueue(w);

edgeTo[w] = v;

}

}

}

}

}