Depth First Search

Madhavan Mukund

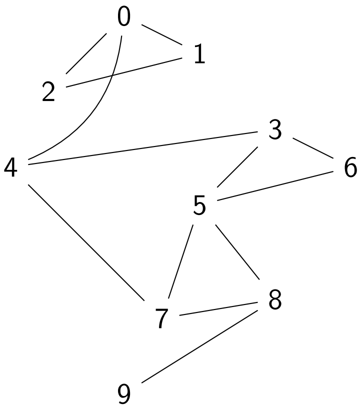
https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 4

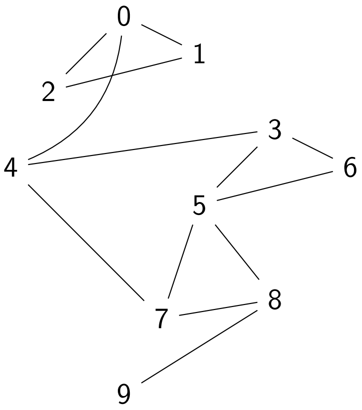
Depth first search (DFS)

- Start from i, visit an unexplored neighbour j



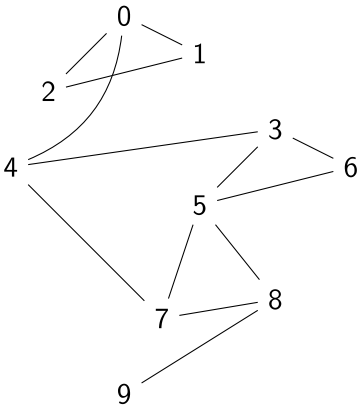
Depth first search (DFS)

- Start from i, visit an unexplored neighbour j
- Suspend the exploration of i and explore j instead



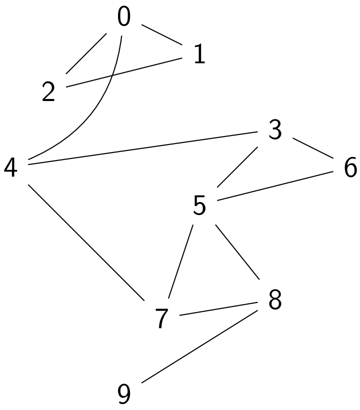
Depth first search (DFS)

- Start from i, visit an unexplored neighbour j
- Suspend the exploration of i and explore j instead
- Continue till you reach a vertex with no unexplored neighbours



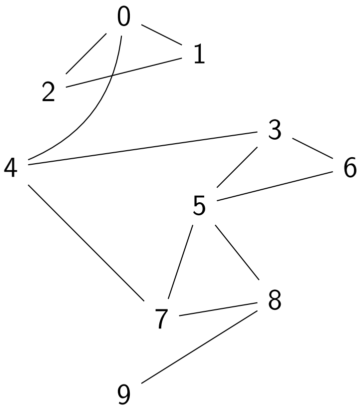
Depth first search (DFS)

- Start from i, visit an unexplored neighbour j
- Suspend the exploration of i and explore j instead
- Continue till you reach a vertex with no unexplored neighbours
- Backtrack to nearest suspended vertex that still has an unexplored neighbour



Depth first search (DFS)

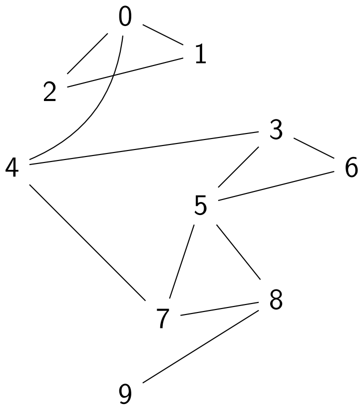
- Start from i, visit an unexplored neighbour j
- Suspend the exploration of i and explore j instead
- Continue till you reach a vertex with no unexplored neighbours
- Backtrack to nearest suspended vertex that still has an unexplored neighbour
- Suspended vertices are stored in a stack
  - Last in, first out
  - Most recently suspended is checked first



DFS from vertex 4

Visited	
0	False
1	False
2	False
3	False
4	False
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices									

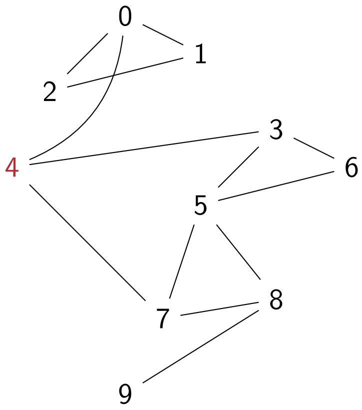


DFS from vertex 4

Visited	
0	False
1	False
2	False
3	False
4	True
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices									

- Mark 4,

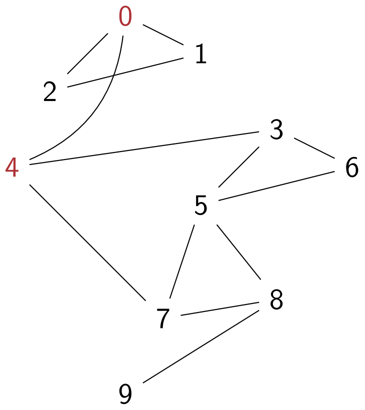


DFS from vertex 4

Visited	
0	True
1	False
2	False
3	False
4	True
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices									
4									

- Mark 4, Suspend 4, explore 0

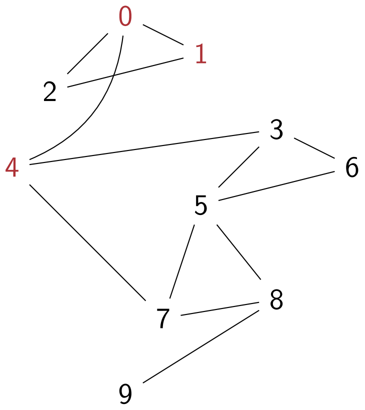


DFS from vertex 4

Visited	
0	True
1	True
2	False
3	False
4	True
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices									
4	0								

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1

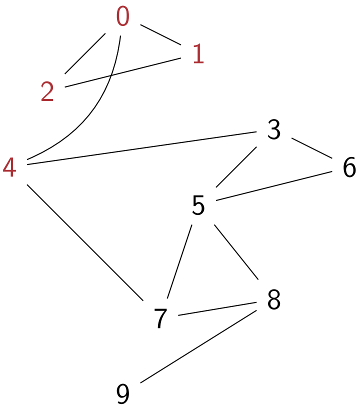


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	False
4	True
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices									
4	0	1							

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2

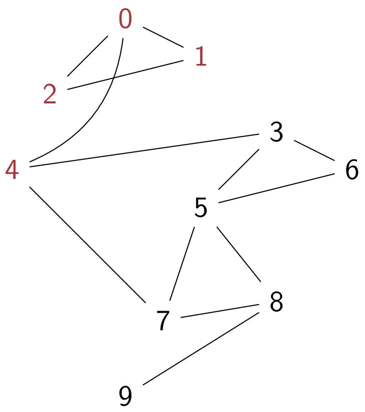


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	False
4	True
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices								
4	0							

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1,

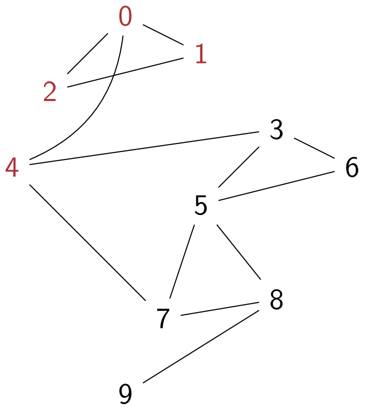


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	False
4	True
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices									
4									

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0,

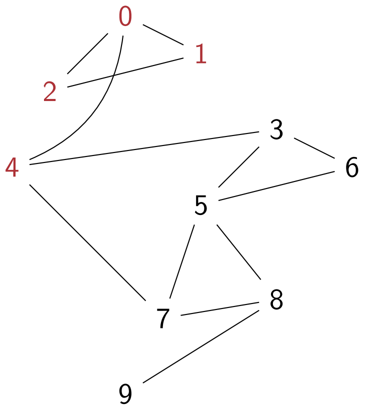


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices									

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4

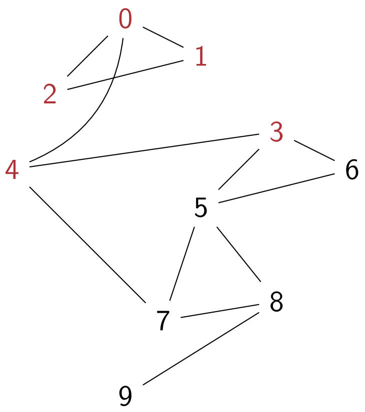


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices									
4									

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3

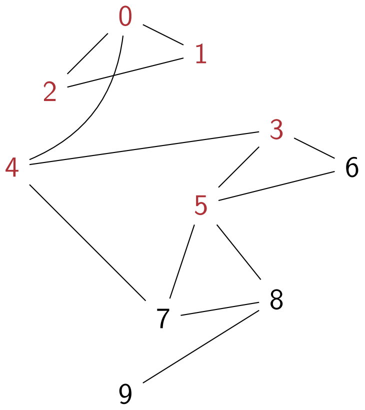


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	False
7	False
8	False
9	False

Stack of suspended vertices									
4	3								

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5

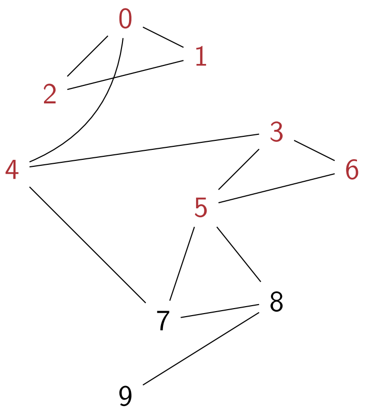


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	False
8	False
9	False

Stack of suspended vertices									
4	3	5							

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6

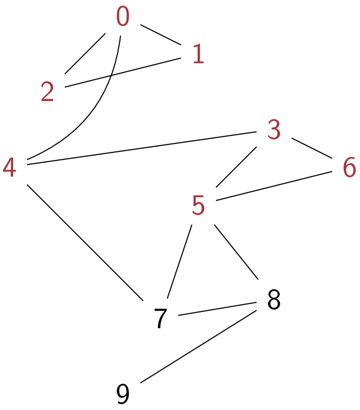


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	False
8	False
9	False

Stack of suspended vertices									
4	3								

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5,

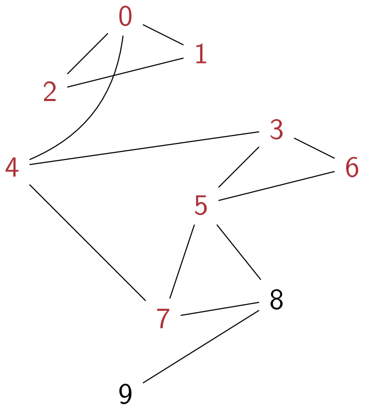


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	False
9	False

Stack of suspended vertices									
4	3	5							

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5, suspend 5, explore 7

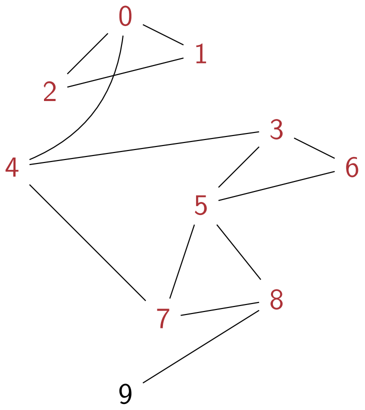


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	False

Stack of suspended vertices									
4	3	5	7						

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5, suspend 5, explore 7
- Suspend 7, explore 8

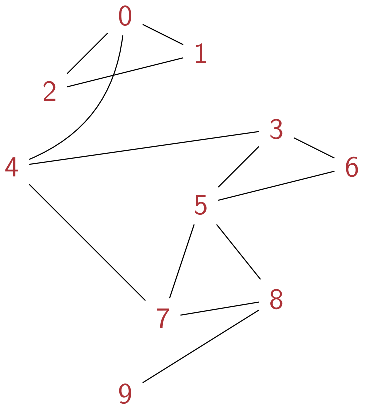


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

Stack of suspended vertices									
4	3	5	7	8					

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5, suspend 5, explore 7
- Suspend 7, explore 8
- Suspend 8, explore 9

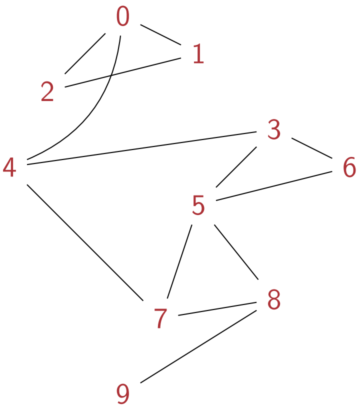


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

Stack of suspended vertices									
4	3	5	7						

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5, suspend 5, explore 7
- Suspend 7, explore 8
- Suspend 8, explore 9
- Backtrack to 8,

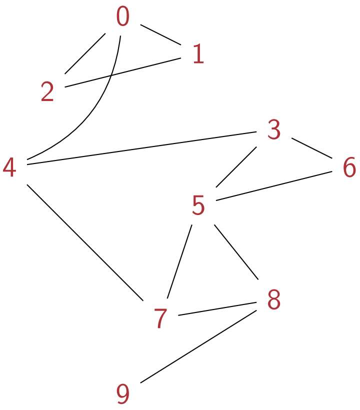


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

Stack of suspended vertices									
4	3	5							

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5, suspend 5, explore 7
- Suspend 7, explore 8
- Suspend 8, explore 9
- Backtrack to 8, 7,

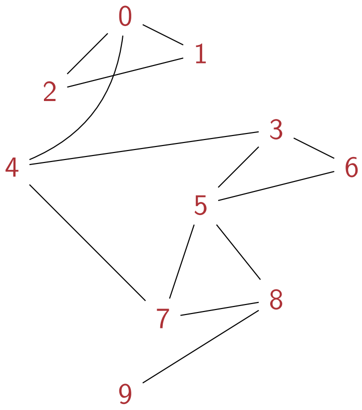


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

Stack of suspended vertices								
4	3							

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5, suspend 5, explore 7
- Suspend 7, explore 8
- Suspend 8, explore 9
- Backtrack to 8, 7, 5,

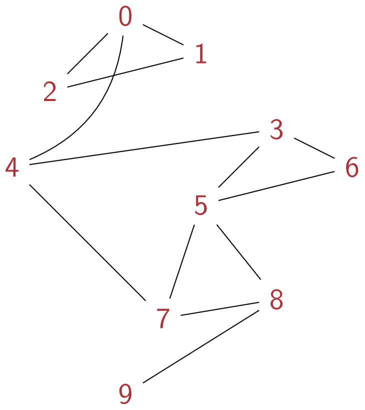


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

Stack of suspended vertices									
4									

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5, suspend 5, explore 7
- Suspend 7, explore 8
- Suspend 8, explore 9
- Backtrack to 8, 7, 5, 3,

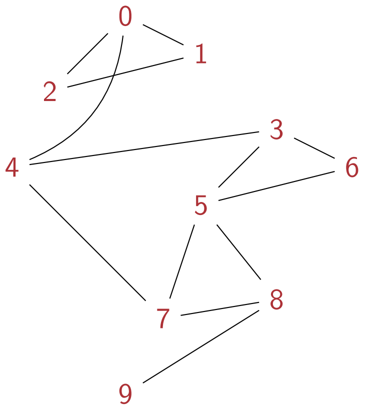


DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

Stack of suspended vertices									

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5, suspend 5, explore 7
- Suspend 7, explore 8
- Suspend 8, explore 9
- Backtrack to 8, 7, 5, 3, 4



Implementing DFS

- DFS is most natural to implement recursively
  - For each unvisited neighbour of v, call $DFS(v)$

```
def DFSInit(AMat):
  # Initialization
  (rows,cols) = AMat.shape
  (visited,parent) = ({},{})
  for i in range(rows):
    visited[i] = False
    parent[i] = -1
  return(visited,parent)

def DFS(AMat,visited,parent,v):
  visited[v] = True

  for k in neighbours(AMat,v):
    if (not visited[k]):
      parent[k] = v
      (visited,parent) =
          DFS(AMat,visited,parent,k)

  return(visited,parent)
```

Implementing DFS

- DFS is most natural to implement recursively
  - For each unvisited neighbour of v, call *DFS*(v)
- No need to maintain a stack
  - Recursion implicilty maintains stack
  - Separate initialization step

```
def DFSInit(AMat):
  # Initialization
  (rows,cols) = AMat.shape
  (visited,parent) = ({},{})
  for i in range(rows):
    visited[i] = False
    parent[i] = -1
  return(visited,parent)

def DFS(AMat,visited,parent,v):
  visited[v] = True

  for k in neighbours(AMat,v):
    if (not visited[k]):
      parent[k] = v
      (visited,parent) =
          DFS(AMat,visited,parent,k)

  return(visited,parent)
```

Implementing DFS

- DFS is most natural to implement recursively
  - For each unvisited neighbour of v, call $DFS(v)$
- No need to maintain a stack
  - Recursion implicilty maintains stack
  - Separate initialization step
- Can make `visited` and `parent` global
  - Still need to initialize them according to the size of input adjacency matrix/list

```
(visited,parent) = ({},{})

def DFSInitGlobal(AMat):
  # Initialization
  (rows,cols) = AMat.shape
  for i in range(rows):
    visited[i] = False
    parent[i] = -1
  return

def DFSGlobal(AMat,v):
  visited[v] = True

  for k in neighbours(AMat,v):
    if (not visited[k]):
      parent[k] = v
      DFSGlobal(AMat,k)

  return
```

Implementing DFS

- DFS is most natural to implement recursively
  - For each unvisited neighbour of v, call $DFS(v)$
- No need to maintain a stack
  - Recursion implicilty maintains stack
  - Separate initialization step
- Can make `visited` and `parent` global
  - Still need to initialize them according to the size of input adjacency matrix/list
- Use an adjacency list instead

```
def DFSInitList(AList):
  # Initialization
  (visited,parent) = ({},{})
  for i in AList.keys():
    visited[i] = False
    parent[i] = -1
  return(visited,parent)

def DFSList(AList,visited,parent,v):
  visited[v] = True

  for k in AList[v]:
    if (not visited[k]):
      parent[k] = v
      (visited,parent) =
          DFSList(AList,visited,parent,k)

  return(visited,parent)
```

Implementing DFS

- DFS is most natural to implement recursively
  - For each unvisited neighbour of v, call $DFS(v)$
- No need to maintain a stack
  - Recursion implicilty maintains stack
  - Separate initialization step
- Can make `visited` and `parent` global
  - Still need to initialize them according to the size of input adjacency matrix/list
- Use an adjacency list instead

```
(visited,parent) = ({},{})

def DFSInitListGlobal(AList):
  # Initialization
  for i in AList.keys():
    visited[i] = False
    parent[i] = -1
  return

def DFSListGlobal(AList,v):
  visited[v] = True

  for k in AList[v]:
    if (not visited[k]):
      parent[k] = v
      DFSListGlobal(AList,k)

  return
```

Complexity of DFS

- Like BFS, each vertex is marked and explored once

Complexity of DFS

- Like BFS, each vertex is marked and explored once
- Exploring vertex v requires scanning all neighbours of v
  - $O(n)$ time for adjacency matrix, independent of $degree(v)$
  - $degree(v)$ time for adjacency list
    - Total time is $O(m)$ across all vertices

Complexity of DFS

- Like BFS, each vertex is marked and explored once
- Exploring vertex v requires scanning all neighbours of v
  - $O(n)$ time for adjacency matrix, independent of $degree(v)$
  - $degree(v)$ time for adjacency list
    - Total time is $O(m)$ across all vertices
- Overall complexity is same as BFS
  - $O(n^2)$ using adjacency matrix
  - $O(m+n)$ using adjacency list

Summary

- DFS is another systematic strategy to explore a graph

Summary

- DFS is another systematic strategy to explore a graph
- DFS uses a stack to suspend exploration and move to unexplored neighbours

Summary

- DFS is another systematic strategy to explore a graph
- DFS uses a stack to suspend exploration and move to unexplored neighbours
- Paths discovered by DFS are not shortest paths, unlike BFS

Summary

- DFS is another systematic strategy to explore a graph
- DFS uses a stack to suspend exploration and move to unexplored neighbours
- Paths discovered by DFS are not shortest paths, unlike BFS
- Useful features can be found by recording the order in which DFS visits vertices