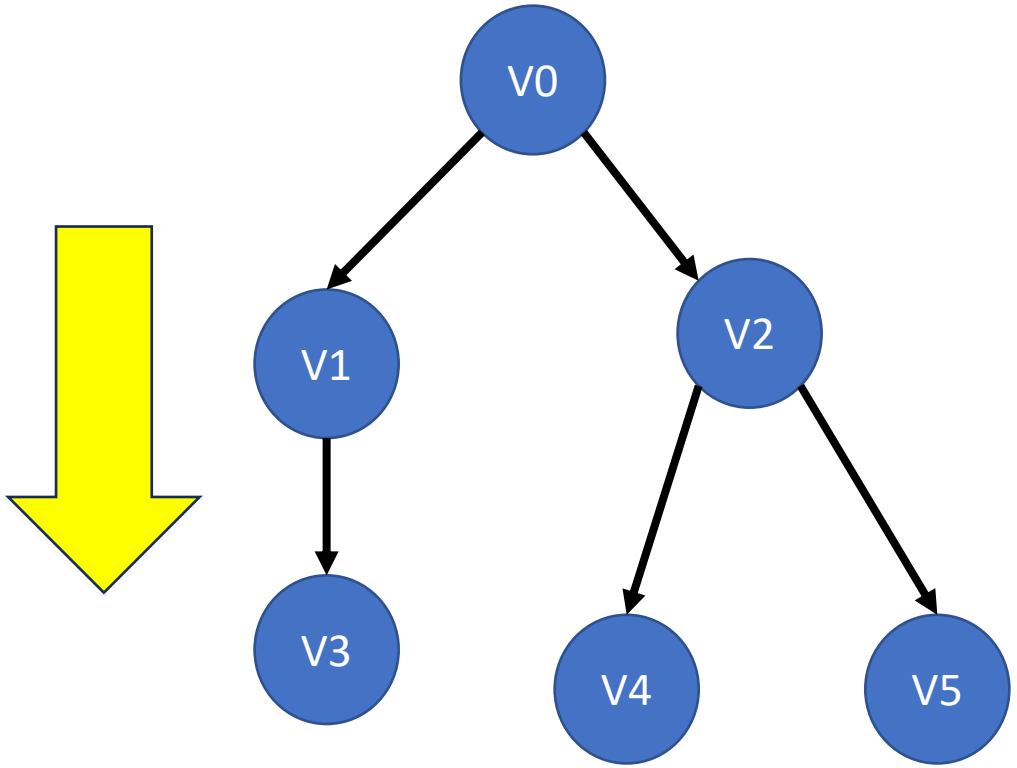
Path-finding

Daniel Nogueira

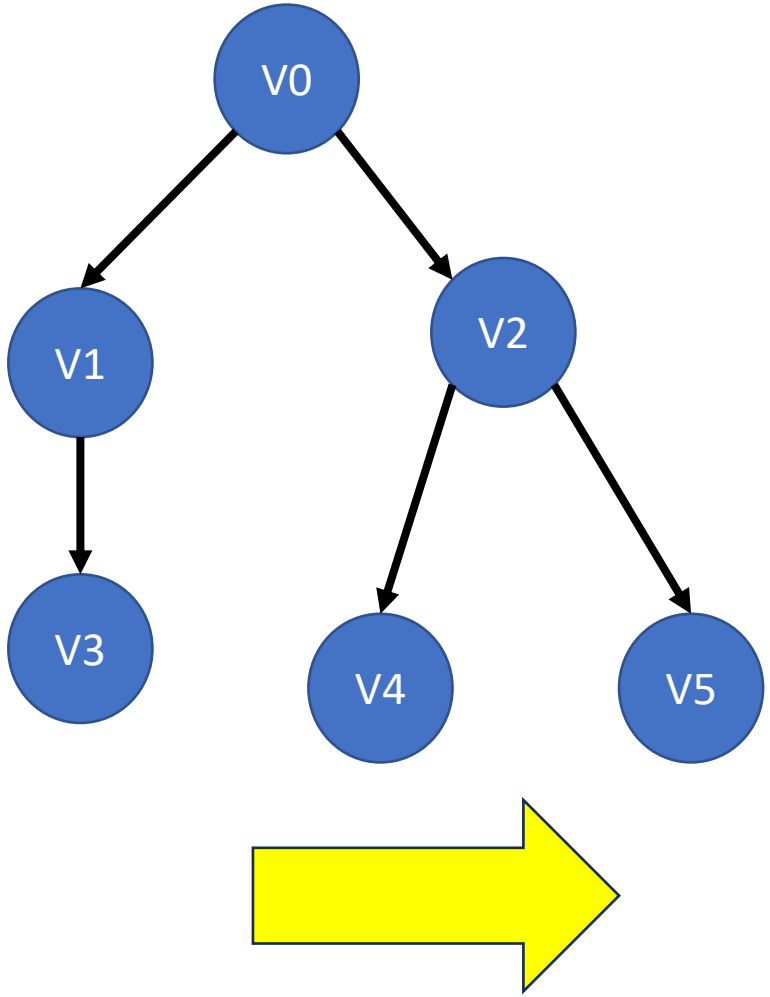
dnogueira@ipca.pt

Algorithms

Depth-First Search (DFS)



Breadth-First Search (BFS)



Algorithms

Depth-First Search (DFS)

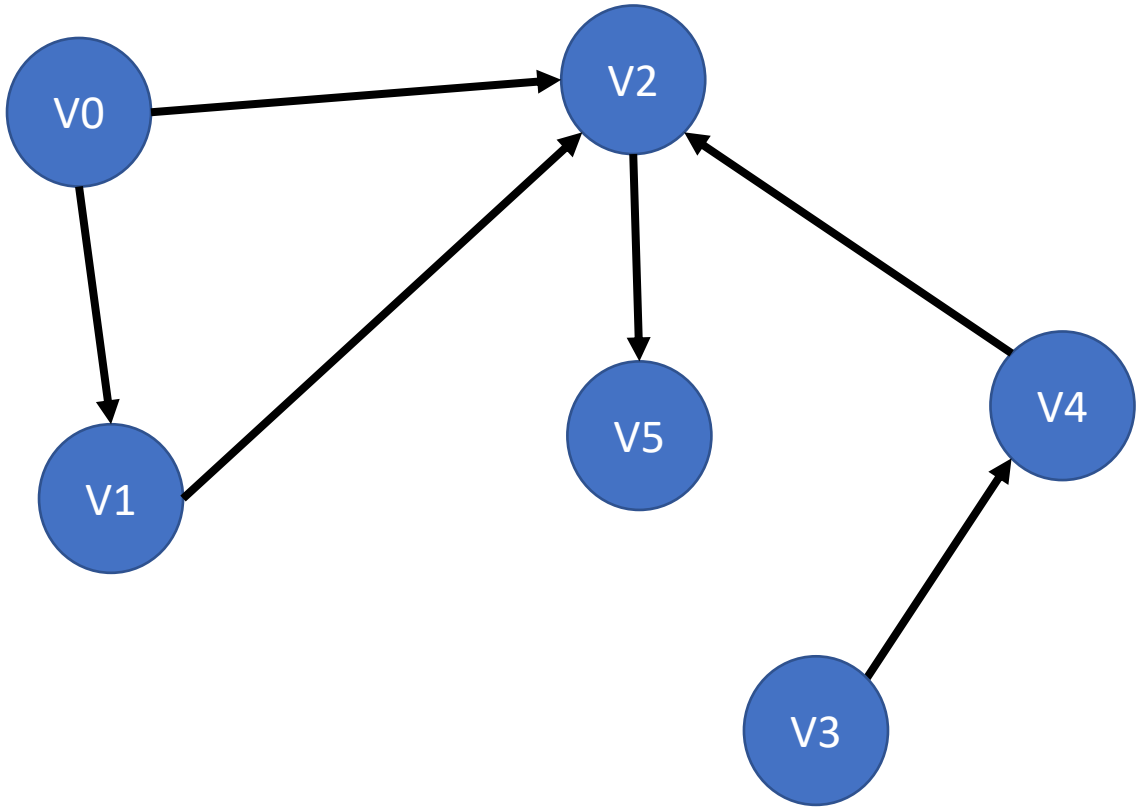
It involves thorough searches of all the nodes by ***forward tracking*** if potential, or else by ***backtracking***

The Algorithm

1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

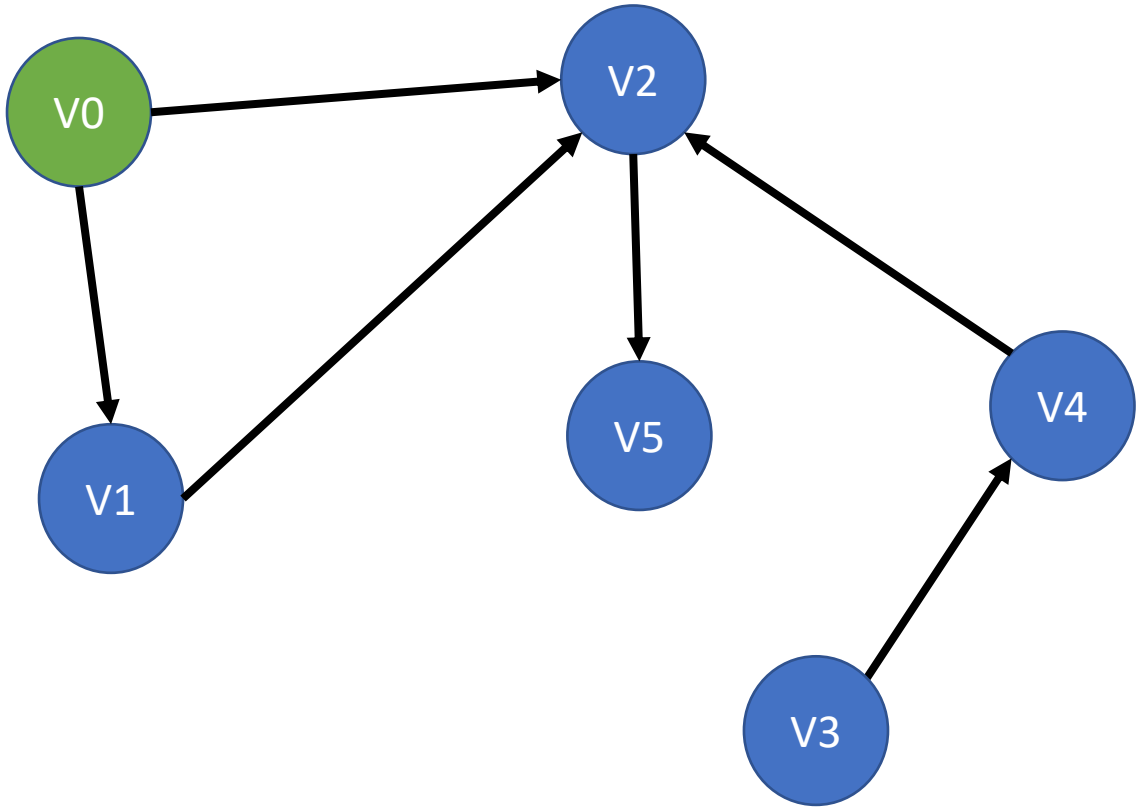
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

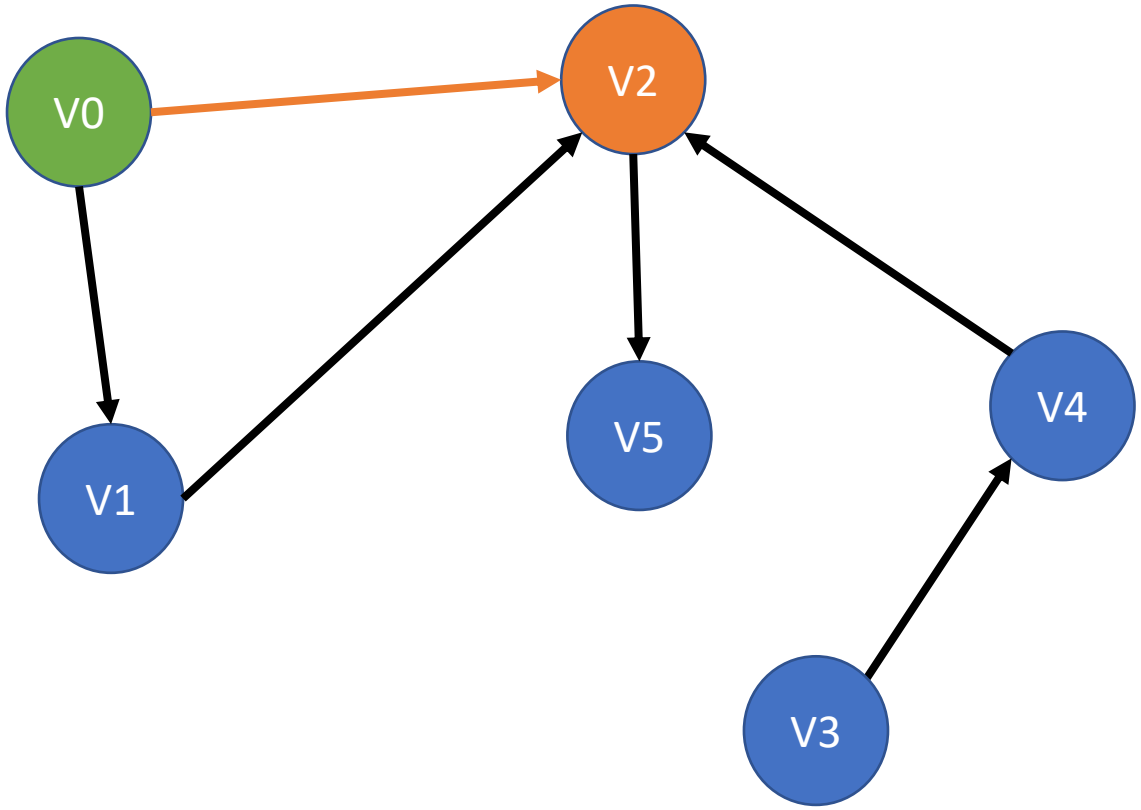
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

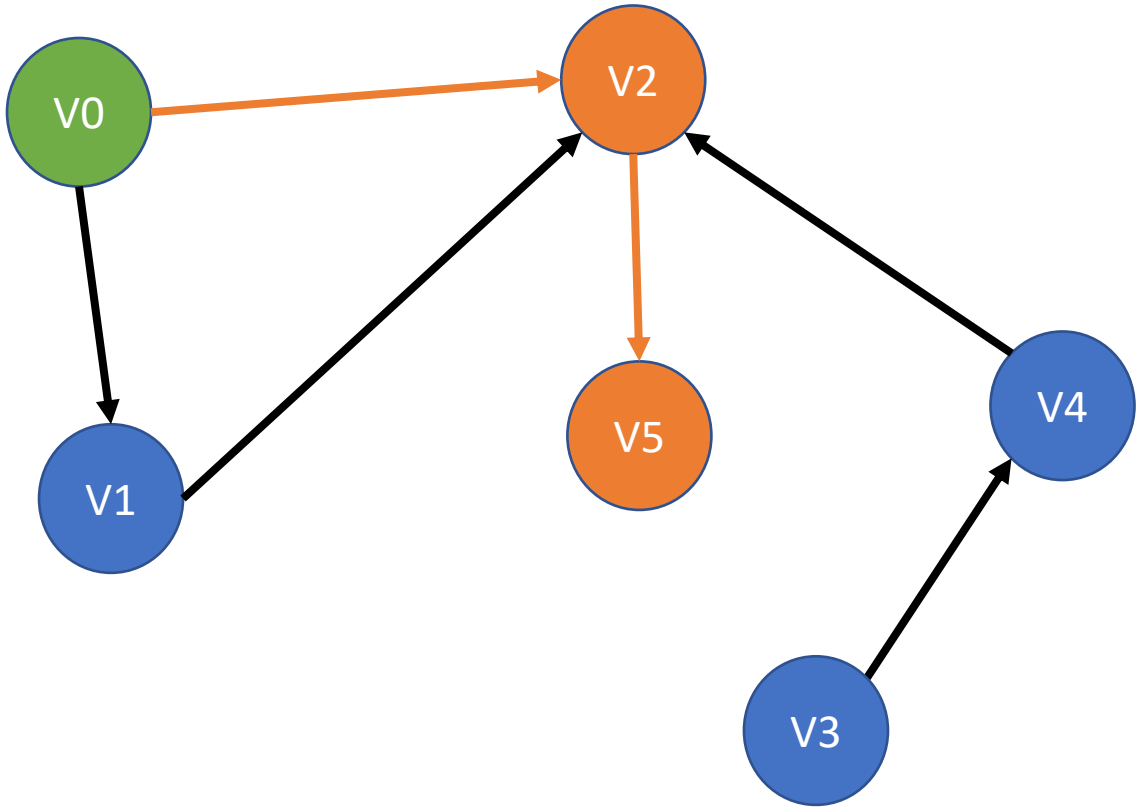
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

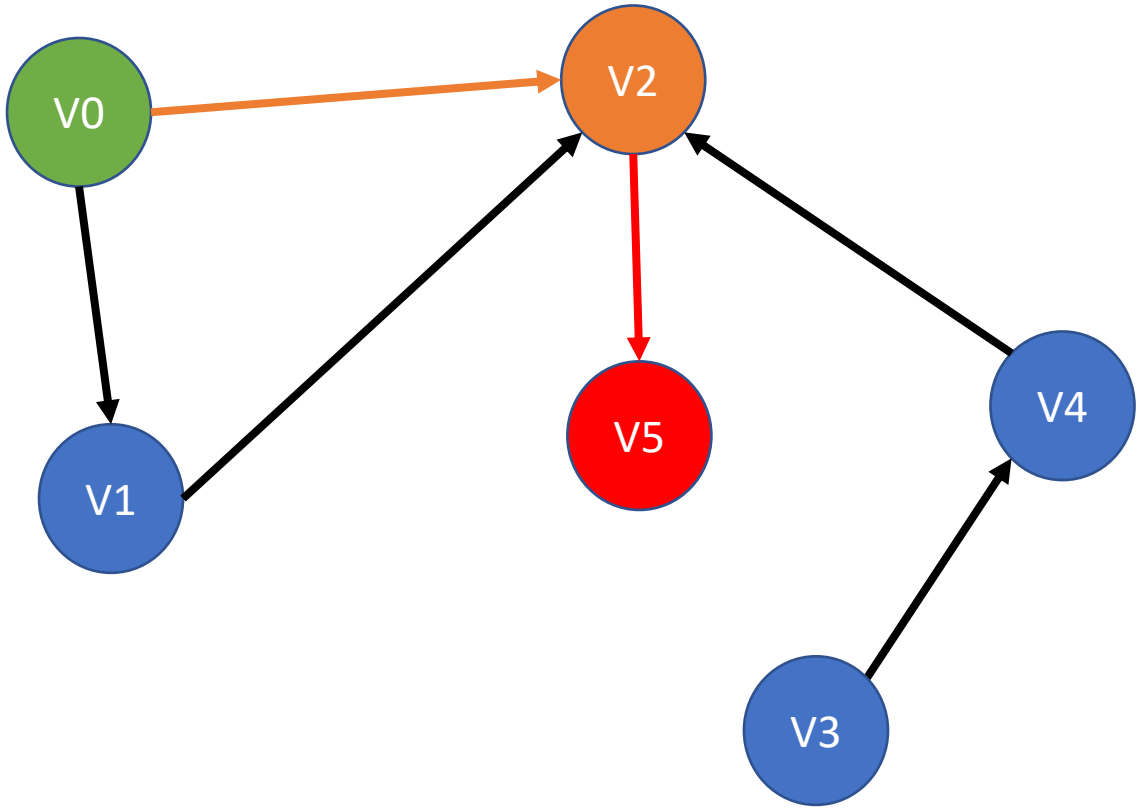
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

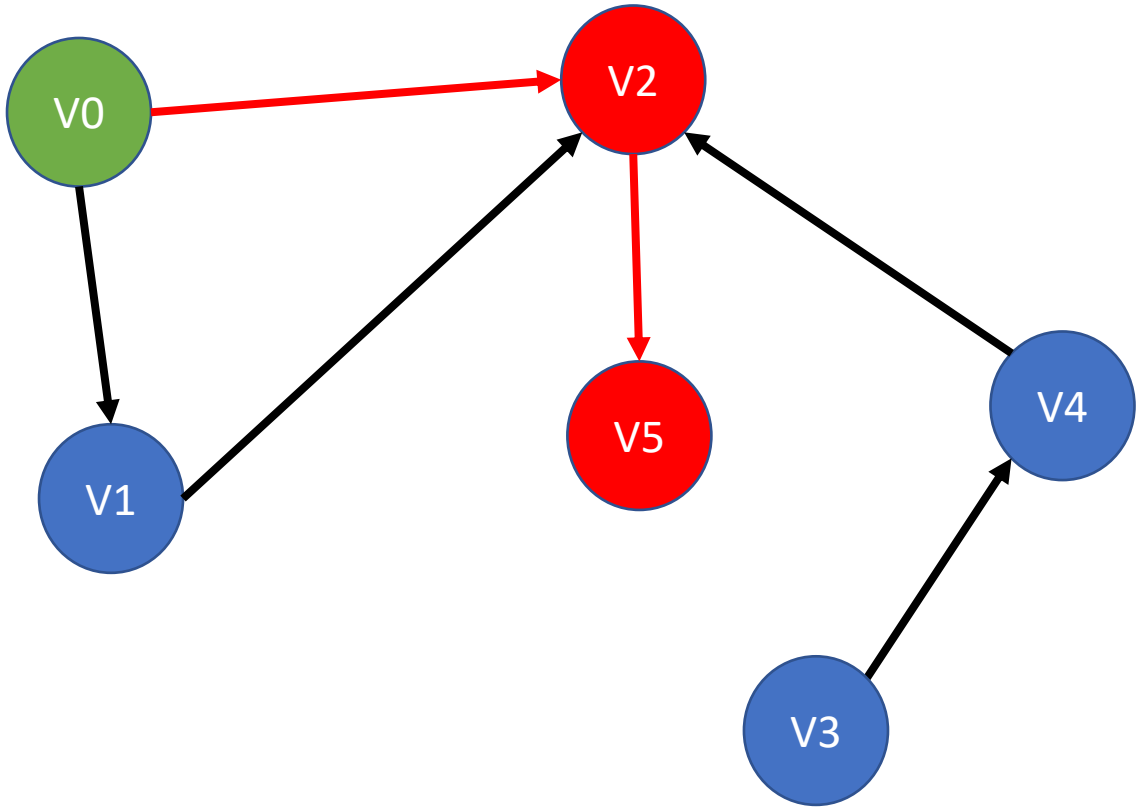
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

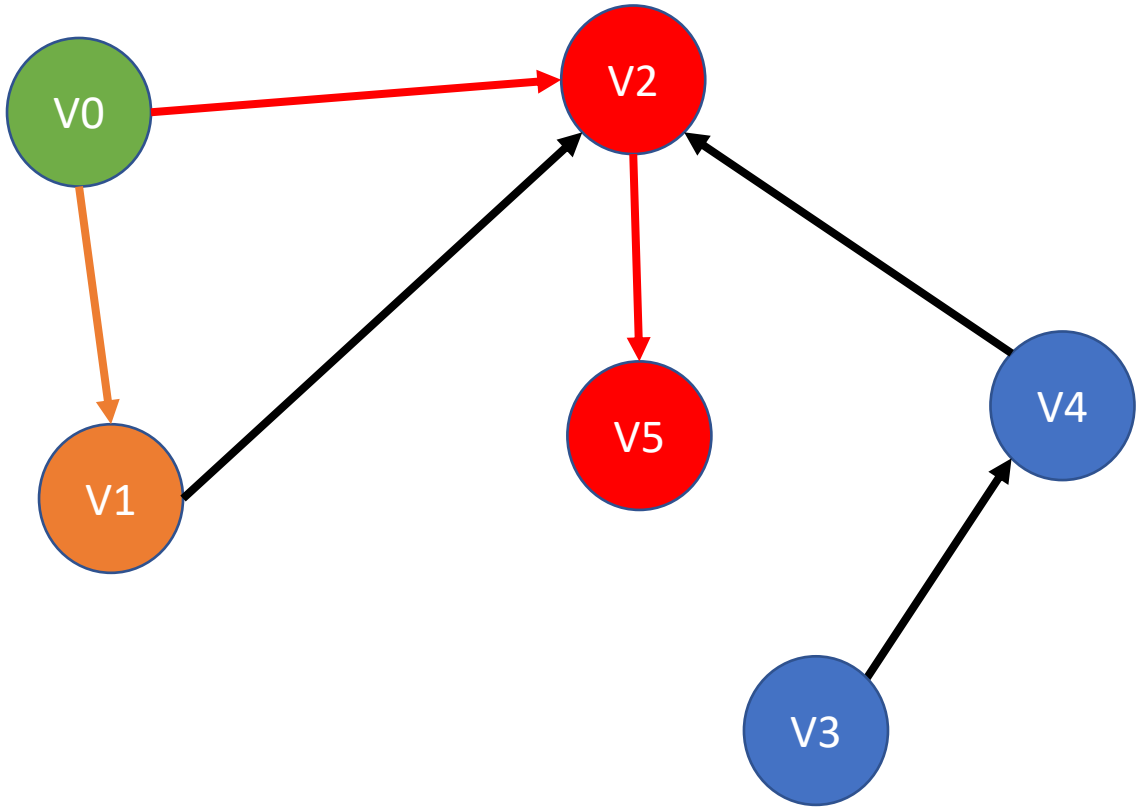
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

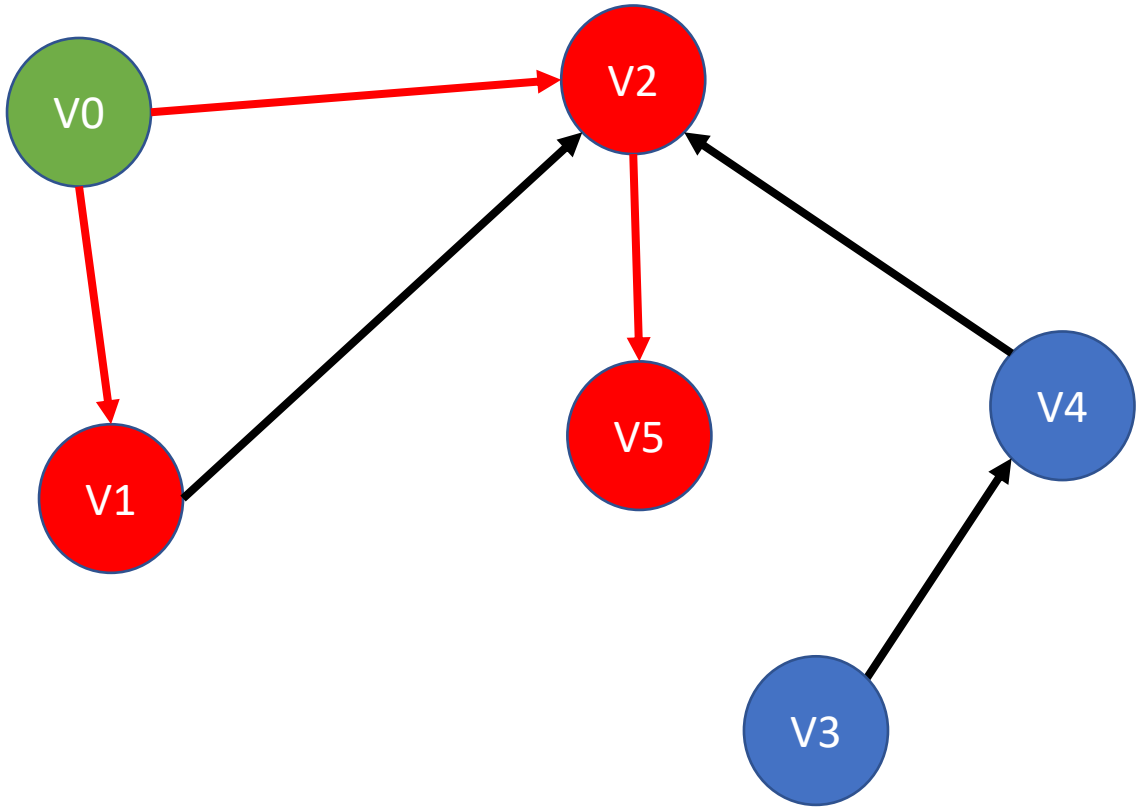
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

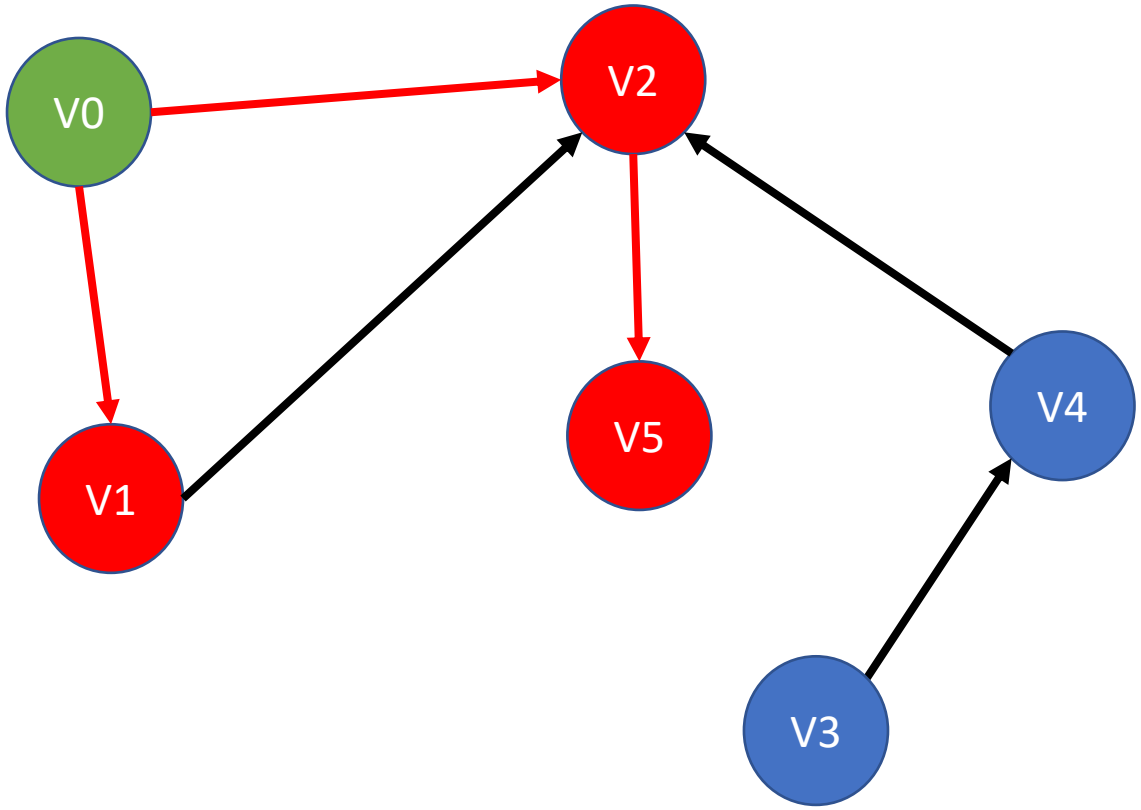
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

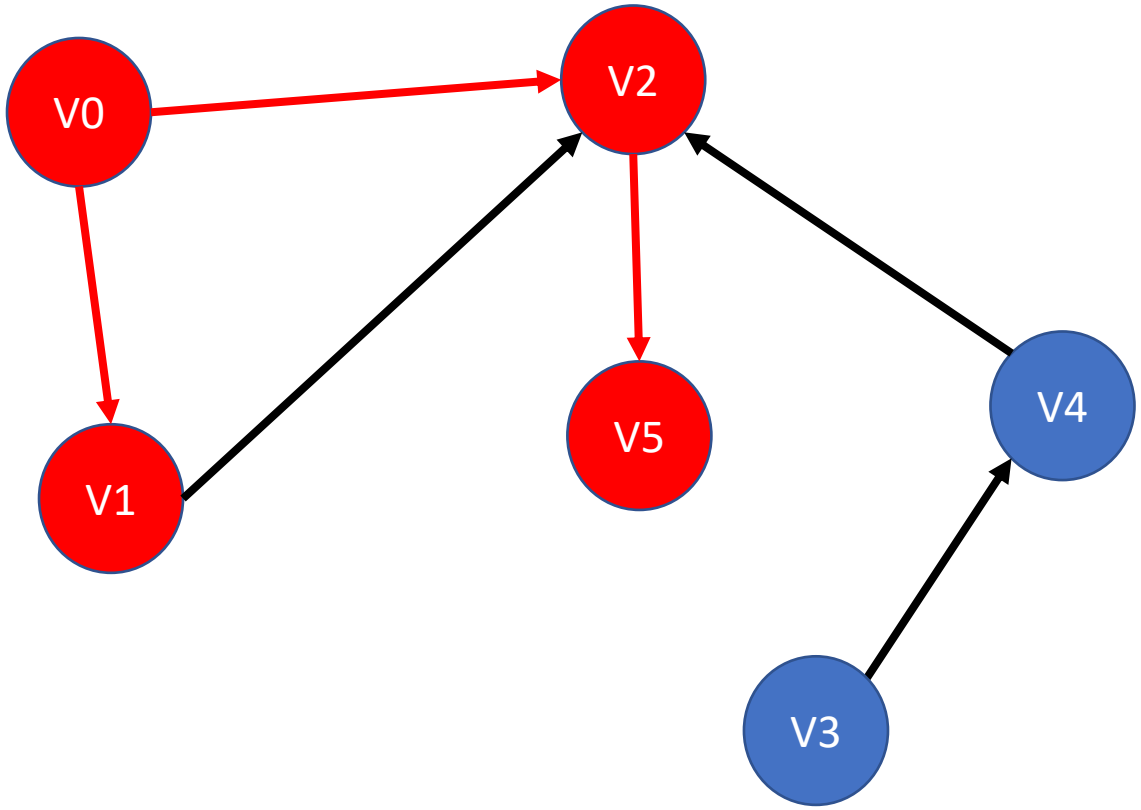
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

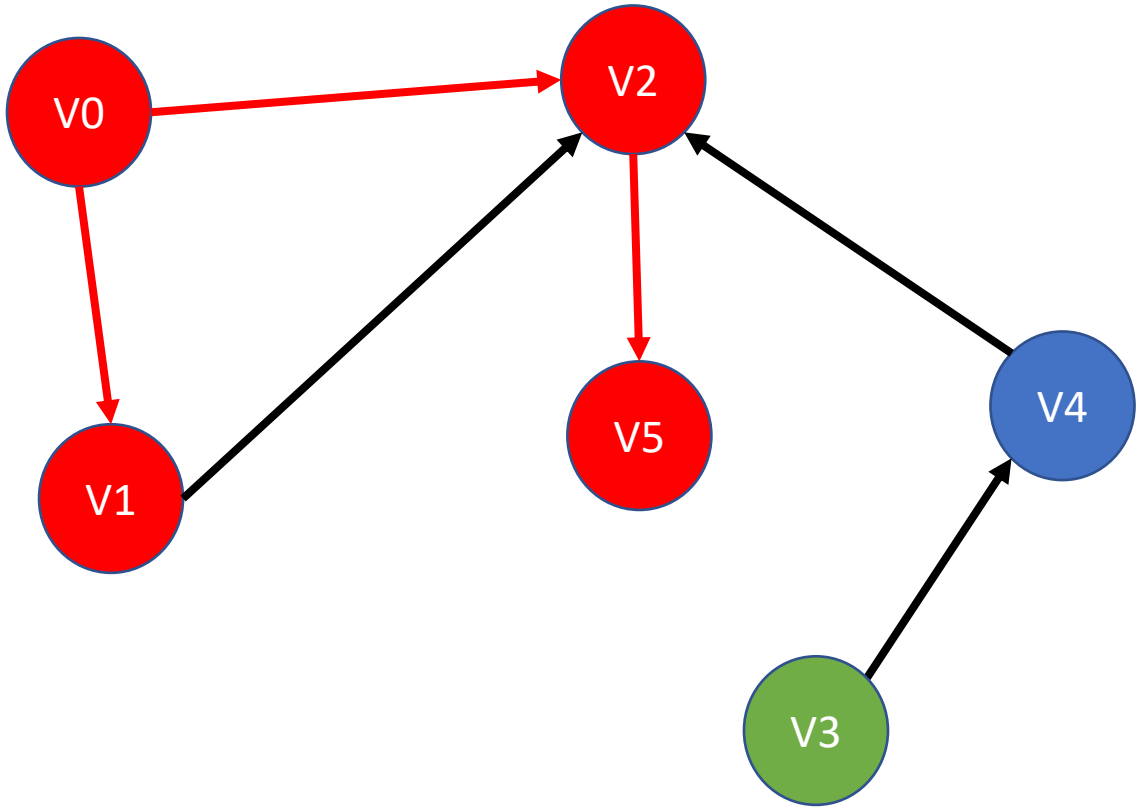
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

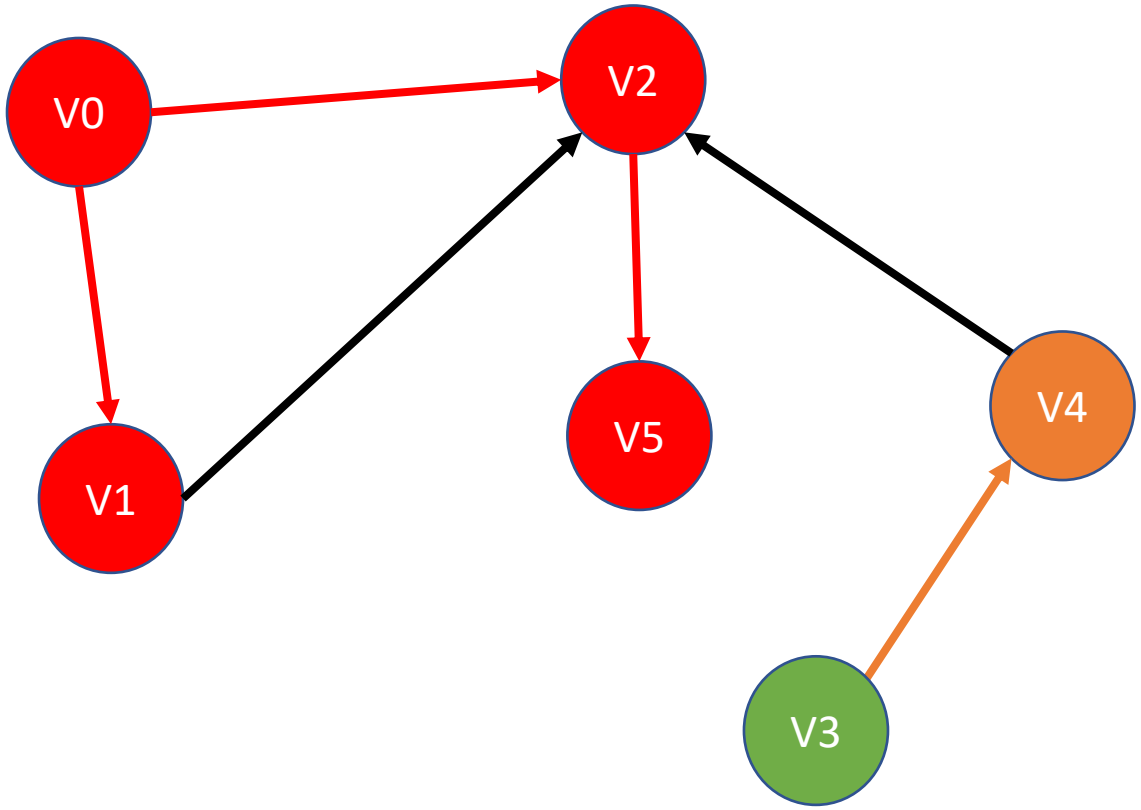
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

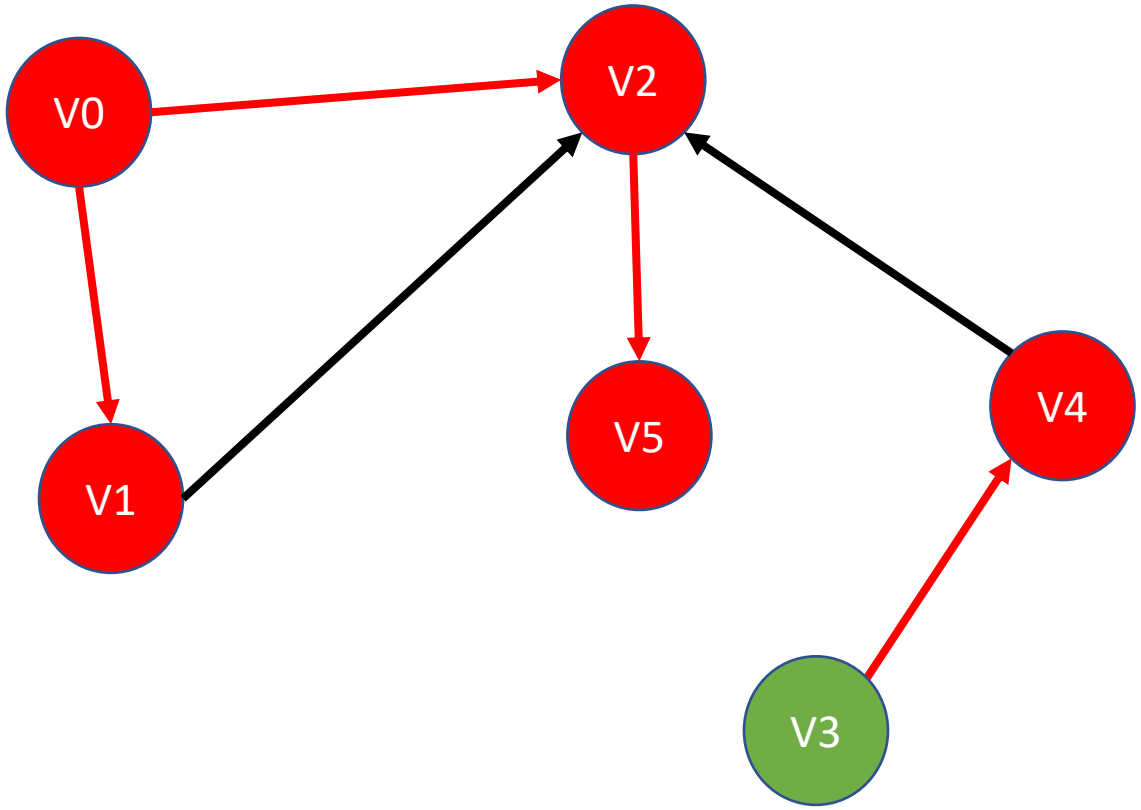
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

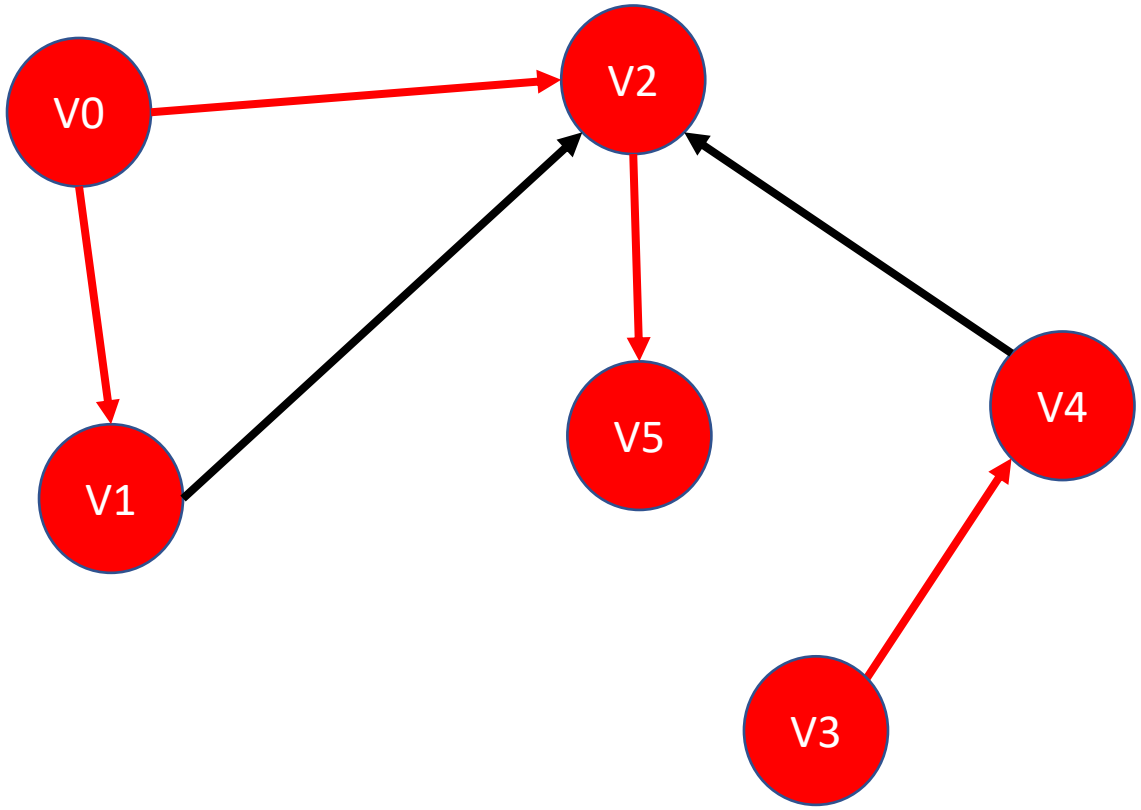
Depth-First Search (DFS)



1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

Depth-First Search (DFS)

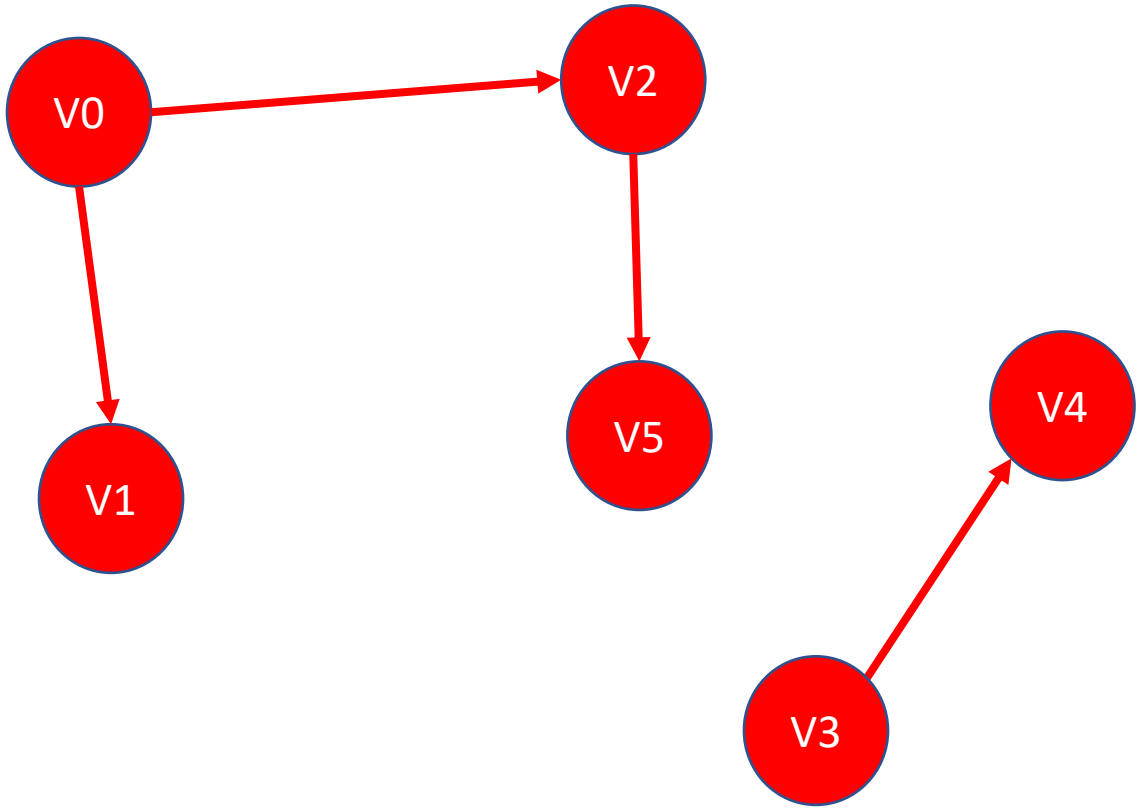


1. Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
   - Choose an adjacent node not yet visited
   - Visit it
3. If it is a non-objective end node:
   - Return to this father
   - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

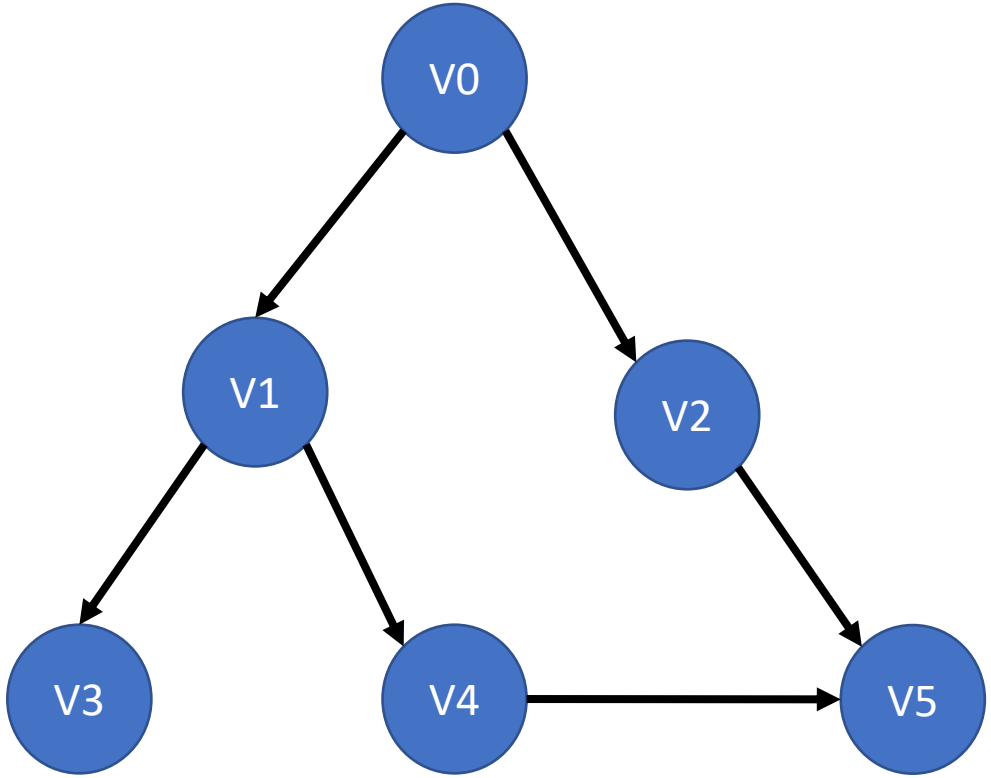
Depth-First Search (DFS)

Two Trees or Forest



Algorithms

Depth-First Search (DFS)



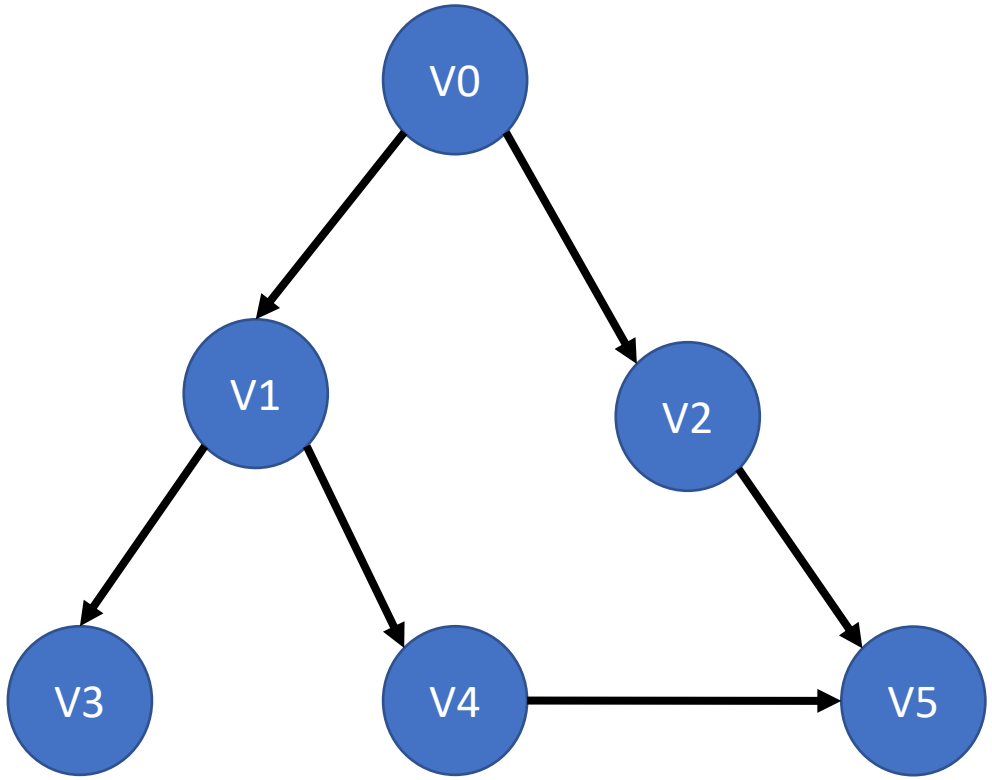
Algorithms

Depth-First Search (DFS)

```
using System;
using System.Collections.Generic;

class Graph
{
    private Dictionary<string, List<string>> adjacencyList;

    public Graph()
    {
        adjacencyList = new Dictionary<string, List<string>>();
    }

    public void AddVertex(string vertex)
    {
        if (!adjacencyList.ContainsKey(vertex))
        {
            adjacencyList[vertex] = new List<string>();
        }
    }

    public void AddEdge(string fromVertex, string toVertex)
    {
        if (!adjacencyList.ContainsKey(fromVertex) || !adjacencyList.ContainsKey(toVertex))
        {
            throw new ArgumentException("Vertices not found in the graph.");
        }

        adjacencyList[fromVertex].Add(toVertex);
    }

    public void DFS(string startVertex)
    {
        HashSet<string> visited = new HashSet<string>();
        DFSRecursive(startVertex, visited);
    }

    private void DFSRecursive(string vertex, HashSet<string> visited)
    {
        visited.Add(vertex);
        Console.WriteLine("Visiting vertex: " + vertex);

        foreach (string neighbor in adjacencyList[vertex])
        {
            if (!visited.Contains(neighbor))
            {
                DFSRecursive(neighbor, visited);
            }       }    }  }
class Program
{
    static void Main(string[] args)
    {
        Graph graph = new Graph();

        // Adicionar vértices
        graph.AddVertex("V0");
        graph.AddVertex("V1");
        graph.AddVertex("V2");
        graph.AddVertex("V3");
        graph.AddVertex("V4");
        graph.AddVertex("V5");

        // Adicionar arestas
        graph.AddEdge("V0", "V1");
        graph.AddEdge("V0", "V2");
        graph.AddEdge("V1", "V3");
        graph.AddEdge("V1", "V4");
        graph.AddEdge("V2", "V5");
        graph.AddEdge("V4", "V5");

        Console.WriteLine("DFS starting from vertex V0:");
        graph.DFS("V0");    } }
```

Algorithms

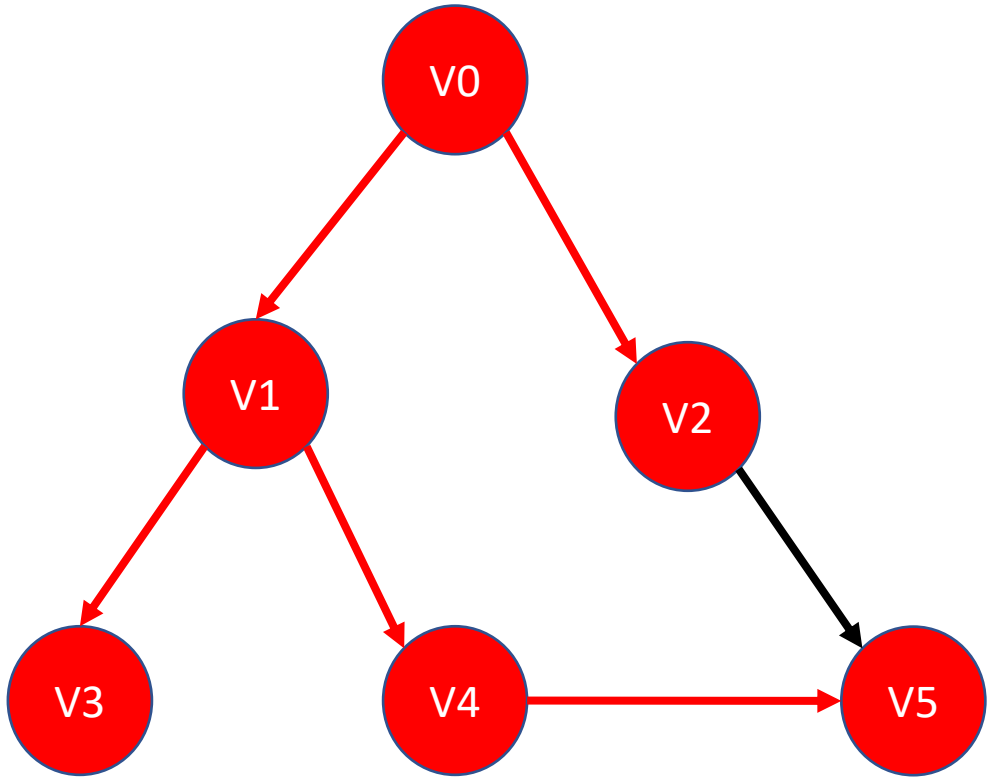
Depth-First Search (DFS)




```
def dfs(graph, vertex, visited):
    # Marcar o vértice como visitado
    visited[vertex] = True
    print("Visitando vértice:", vertex)

    # Recursivamente visitar os vértices adjacentes não visitados
    for neighbor in graph[vertex]:
        if not visited[neighbor]:
            dfs(graph, neighbor, visited)

# Grafo representado como um dicionário de adjacências
graph = {
    'V0': ['V1', 'V2'],
    'V1': ['V3', 'V4'],
    'V2': ['V5'],
    'V3': [],
    'V4': ['V5'],
    'V5': []
}

# Inicializar um vetor de visitados
visited = {vertex: False for vertex in graph}

# Chamar o DFS a partir de todos os vértices não visitados
for vertex in graph:
    if not visited[vertex]:
        dfs(graph, vertex, visited)
```

Algorithms

Depth-First Search (DFS)




```
Following is the Depth-First Search
V0
V1
V3
V4
V5
V2
```

Algorithms

Breadth-First Search (BFS)

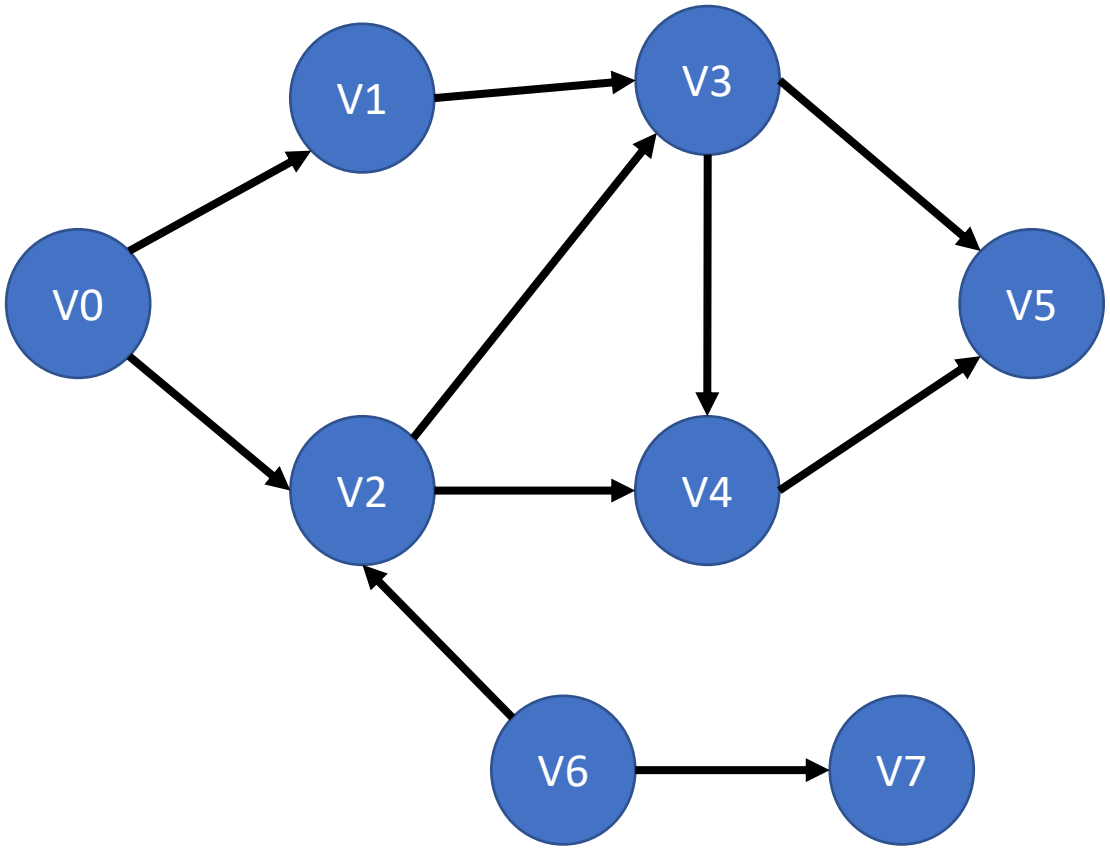
It is used for tree traversal on graphs or tree data structures. BFS can be easily implemented using recursion and data structures like dictionaries and lists.

The Algorithm

1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
    - Remove the 1st node from the queue, u
    - For each neighbour v of u:
        * If v is not explored:
            ** Mark v as explored
            ** Put v at the end of the queue
4. Repeat from another starting node, if there is one

Algorithms

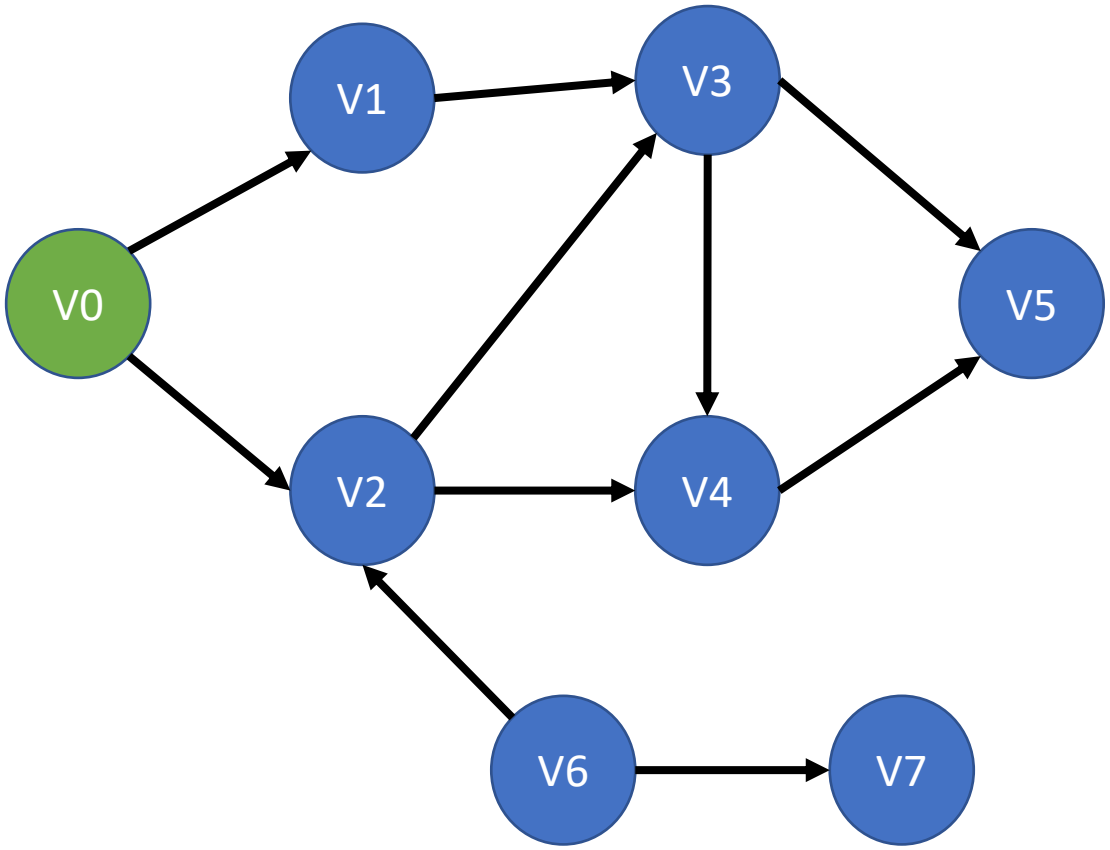
Breadth-First Search (BFS)



1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
    - Remove the 1st node from the queue, u
    - For each neighbour v of u:
        * If v is not explored:
            ** Mark v as explored
            ** Put v at the end of the queue
4. Repeat from another starting node, if there is one

Algorithms

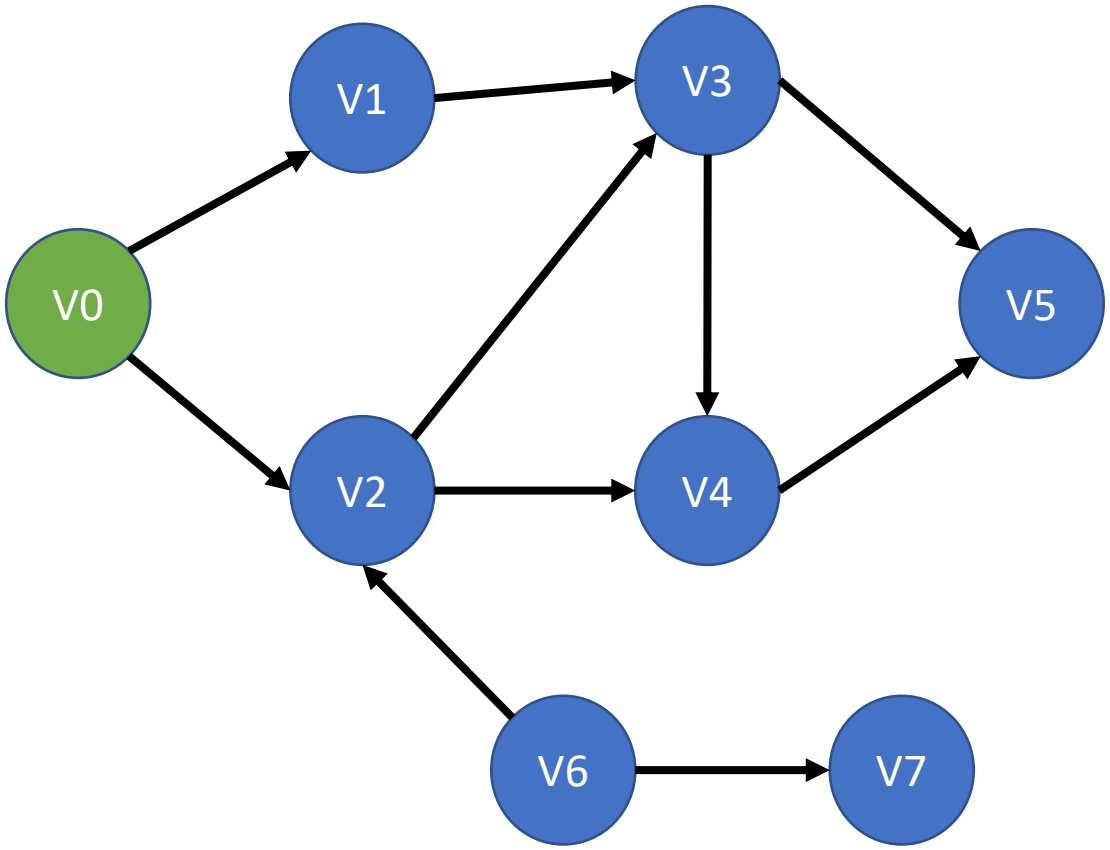
Breadth-First Search (BFS)



1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the queue, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the queue
4. Repeat from another starting node, if there is one

Algorithms

Breadth-First Search (BFS)

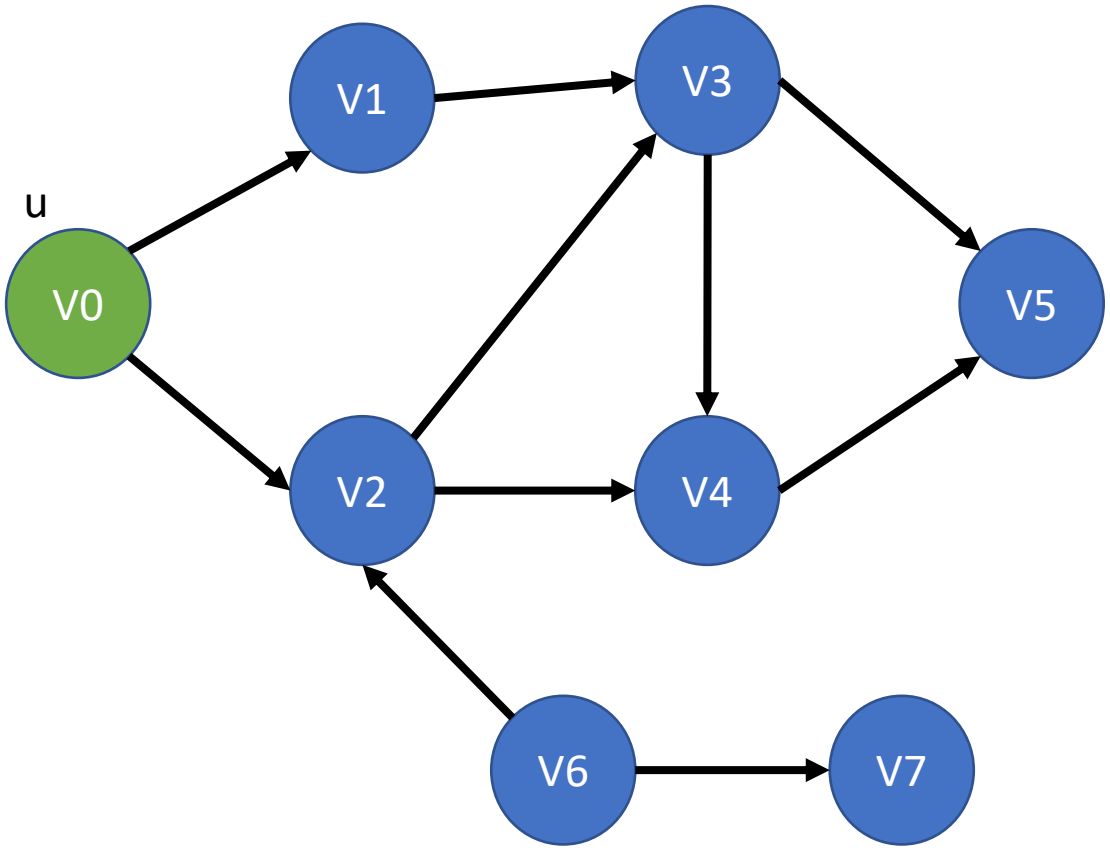


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the queue, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the queue
4. Repeat from another starting node, if there is one

V0

Algorithms

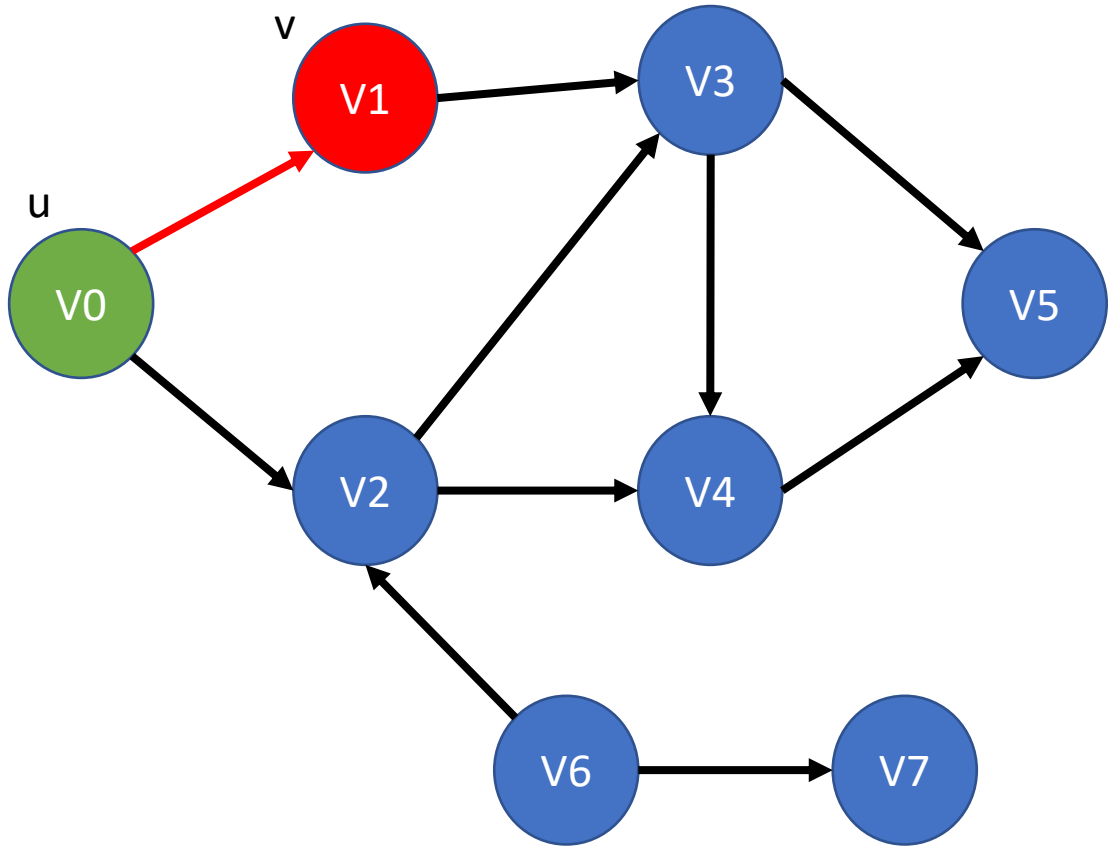
Breadth-First Search (BFS)



1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

Algorithms

Breadth-First Search (BFS)

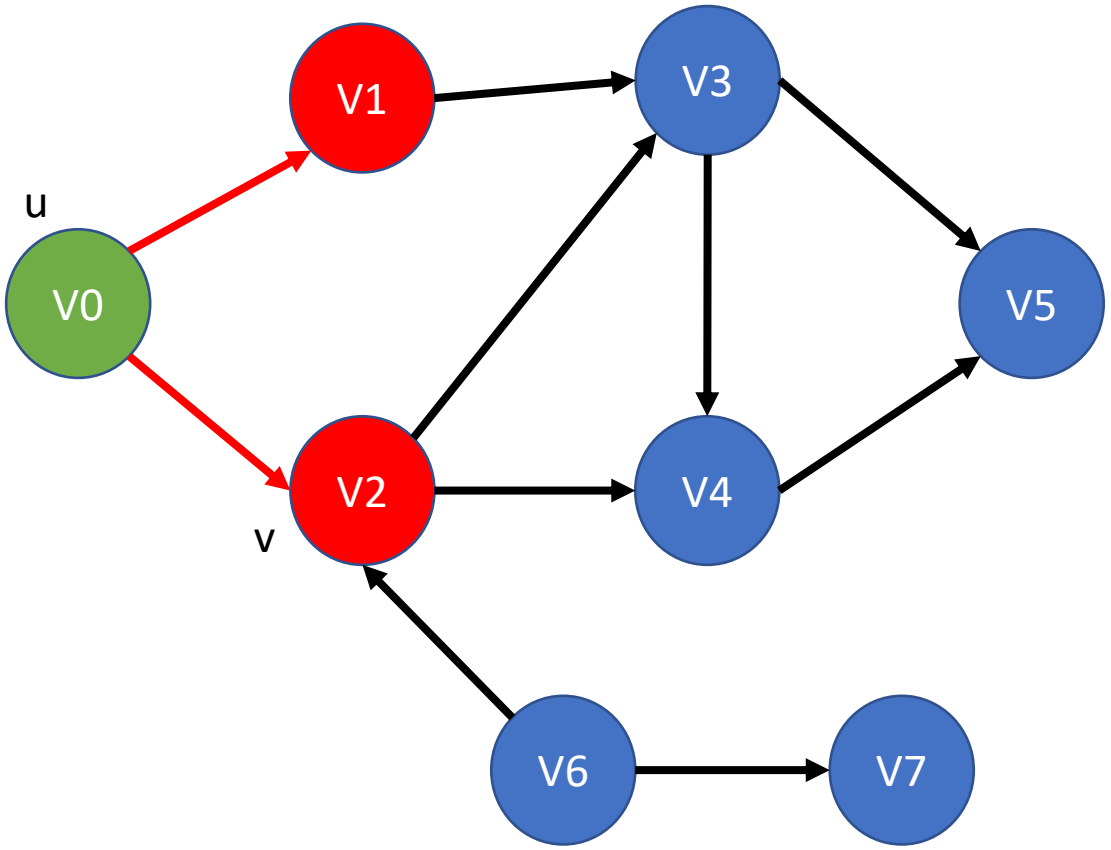


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V1

Algorithms

Breadth-First Search (BFS)

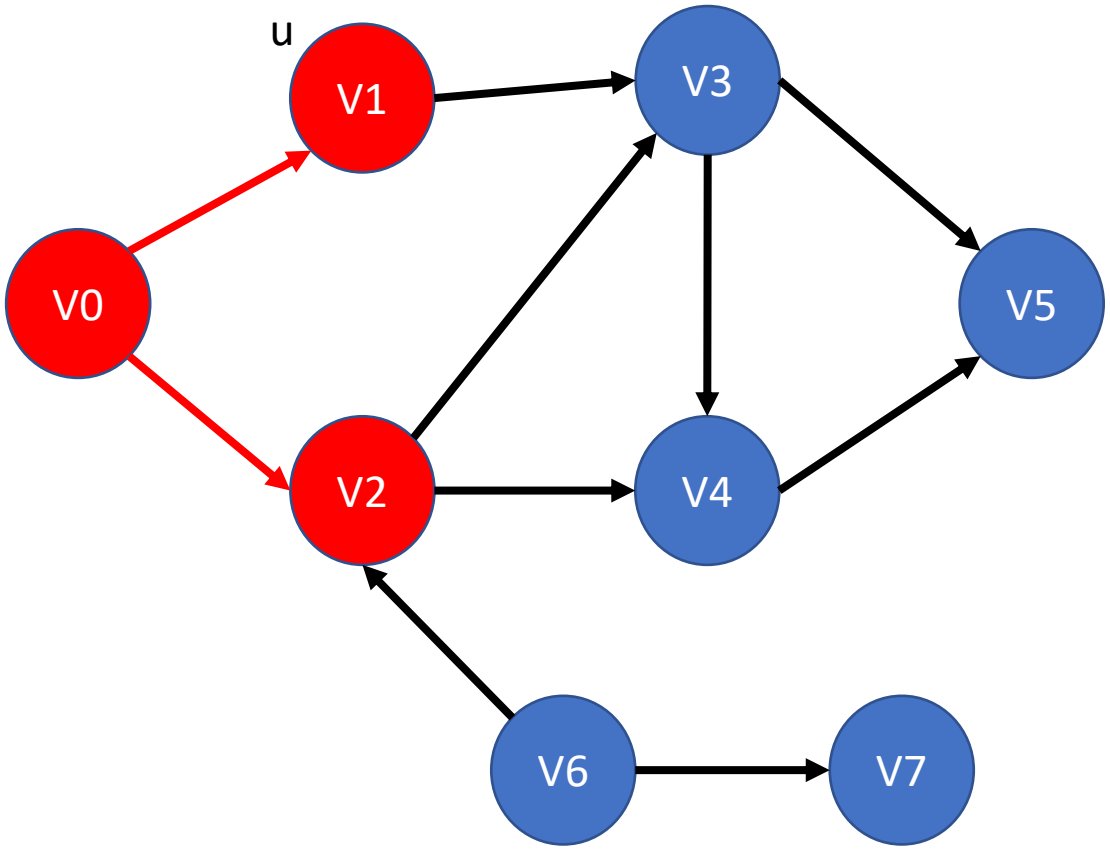


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V1	V2

Algorithms

Breadth-First Search (BFS)

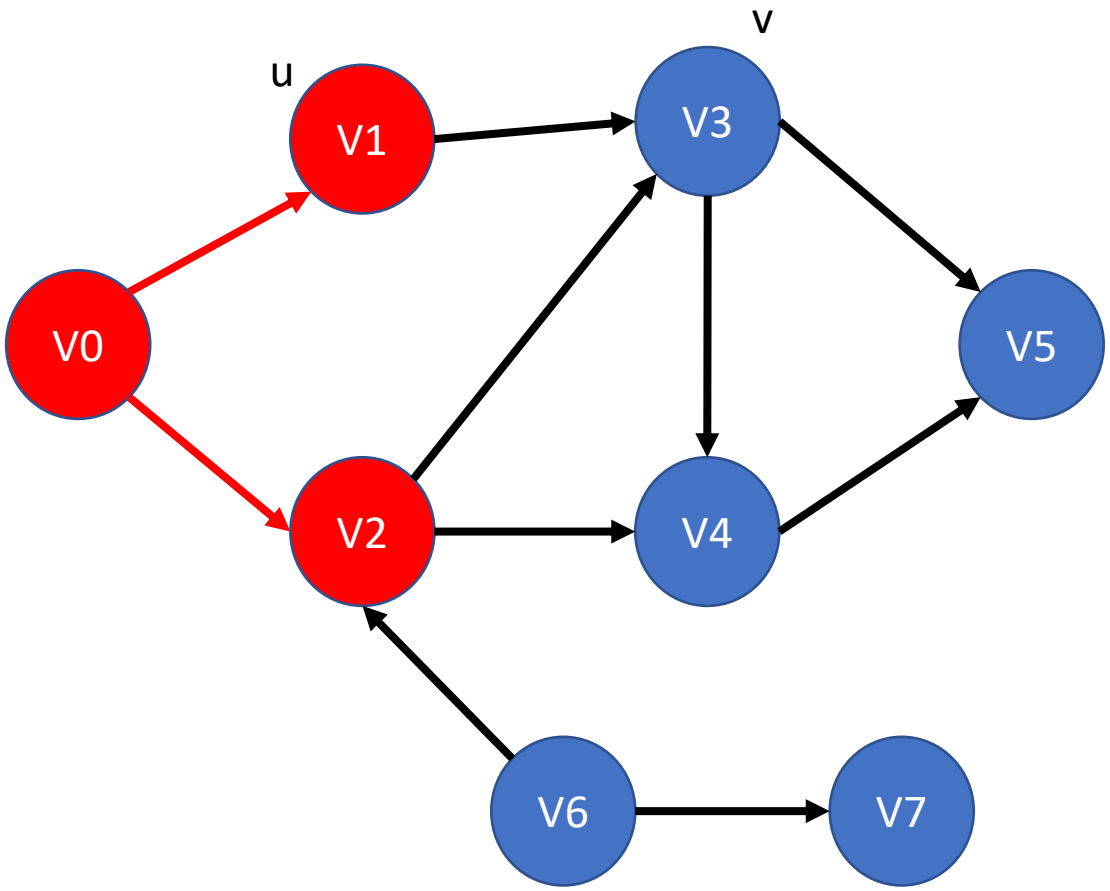


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V2

Algorithms

Breadth-First Search (BFS)

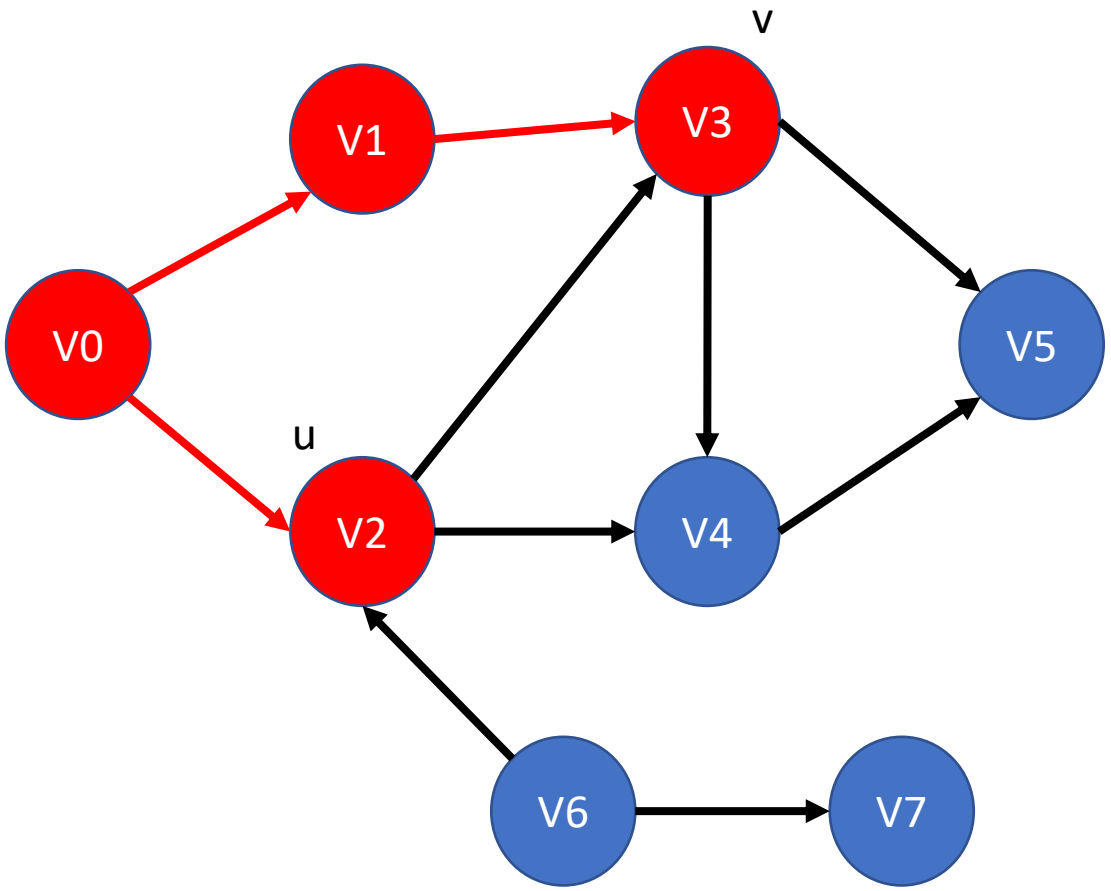


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V2	V3

Algorithms

Breadth-First Search (BFS)

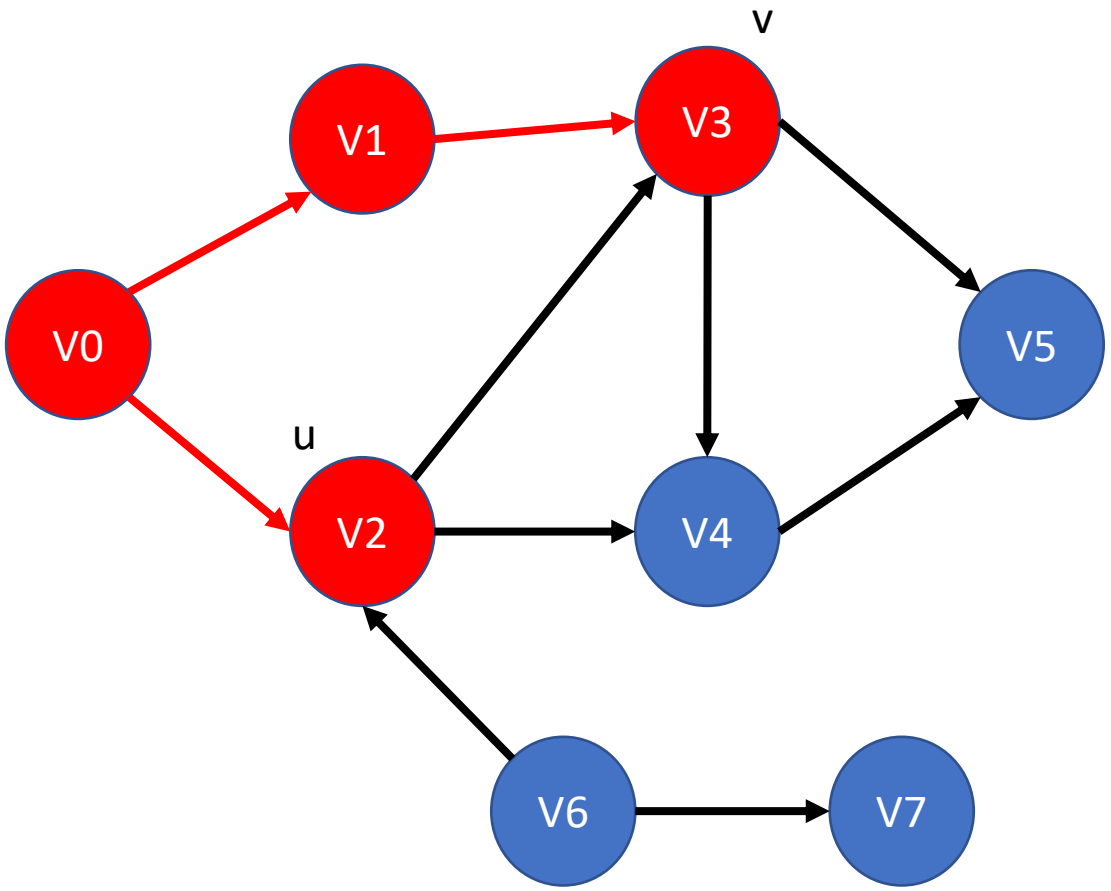


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V2	V3

Algorithms

Breadth-First Search (BFS)

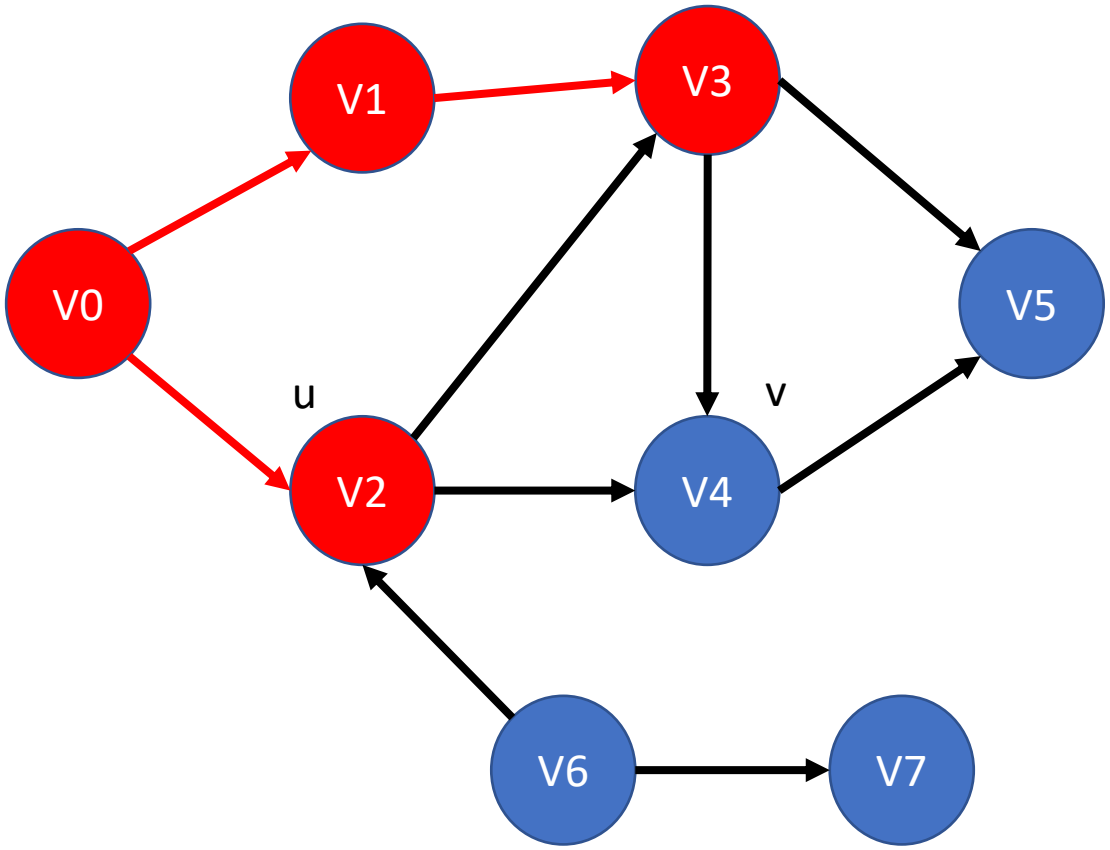


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V3

Algorithms

Breadth-First Search (BFS)

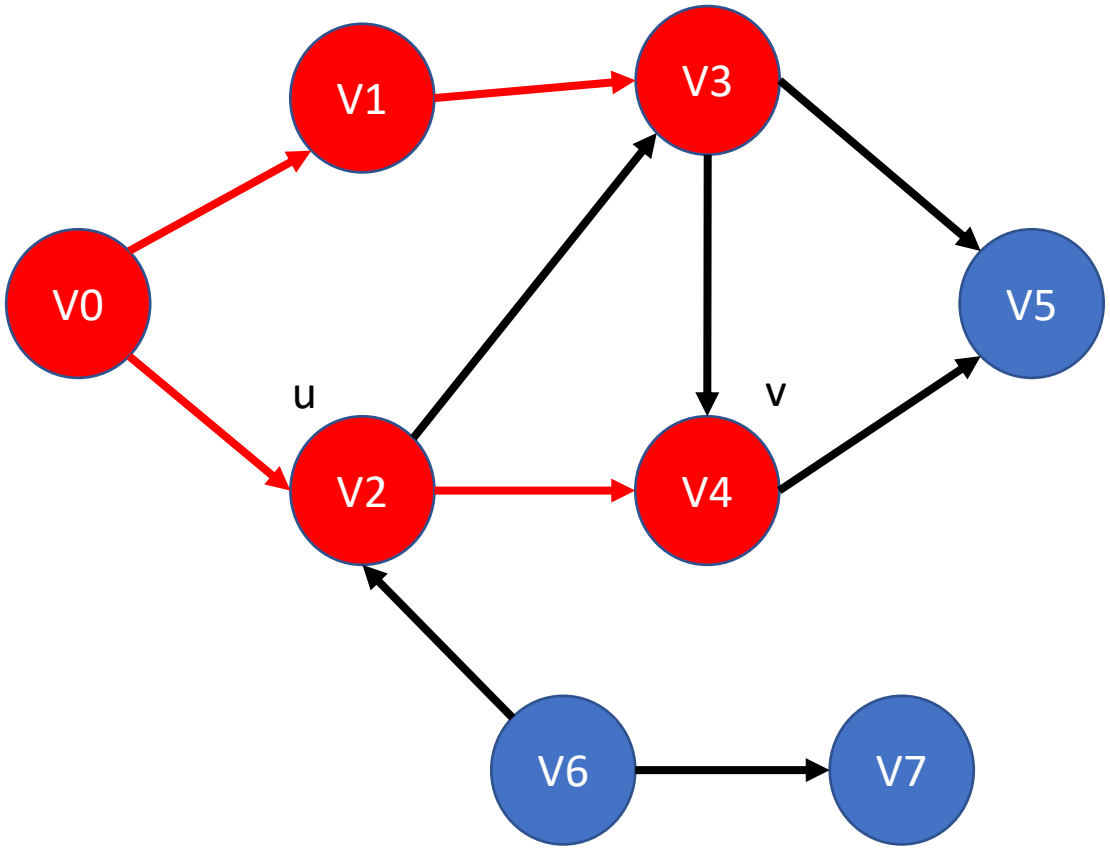


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V3	V4

Algorithms

Breadth-First Search (BFS)

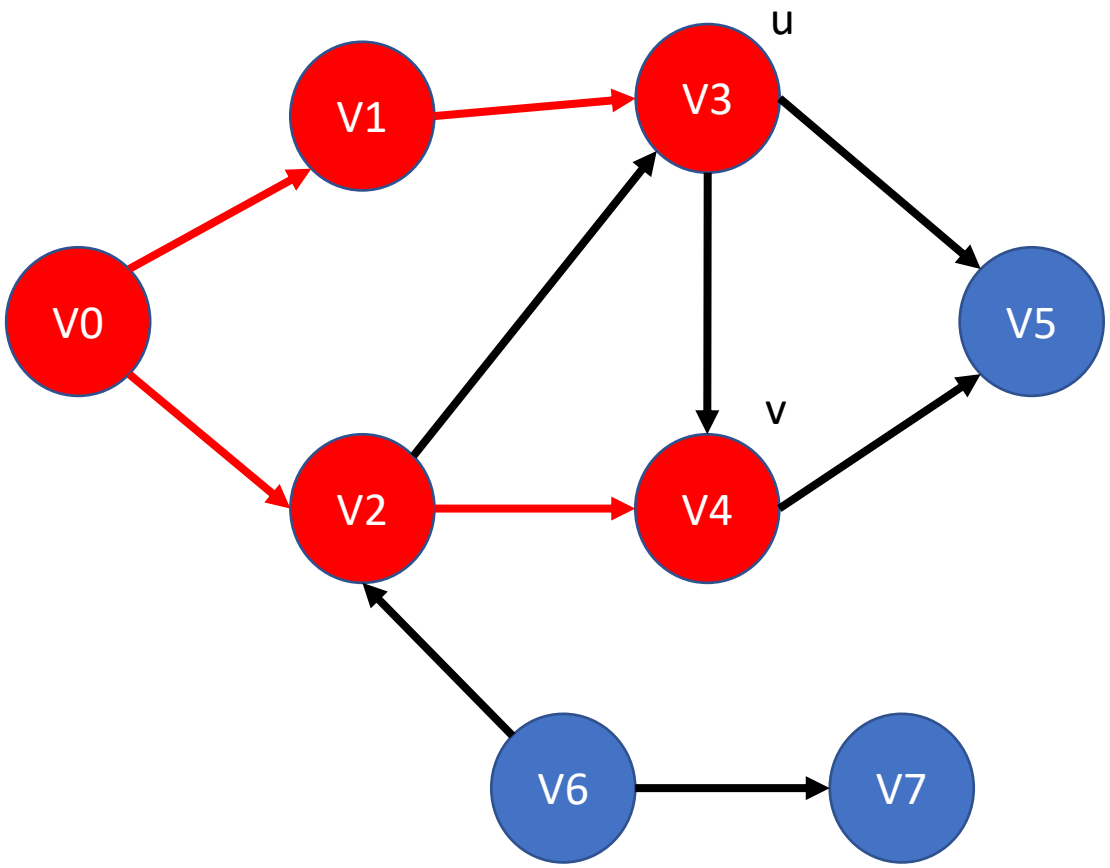


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V3	V4

Algorithms

Breadth-First Search (BFS)

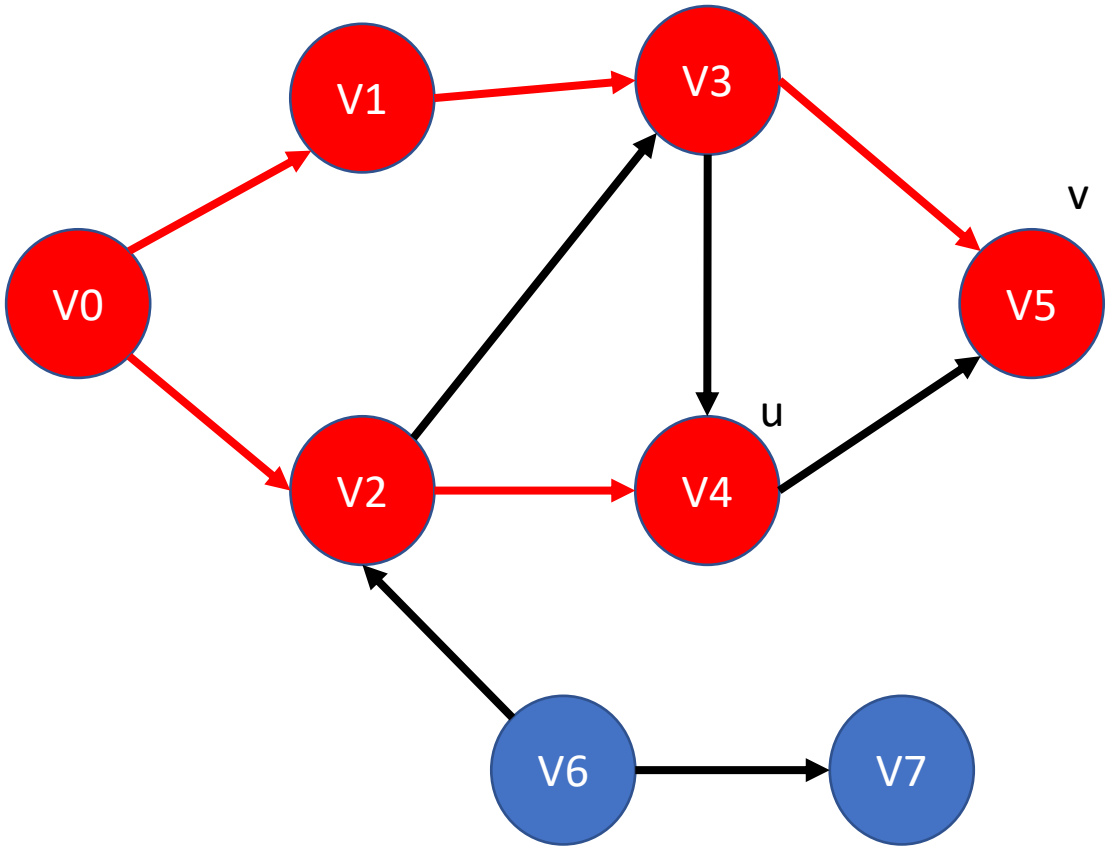


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V4

Algorithms

Breadth-First Search (BFS)

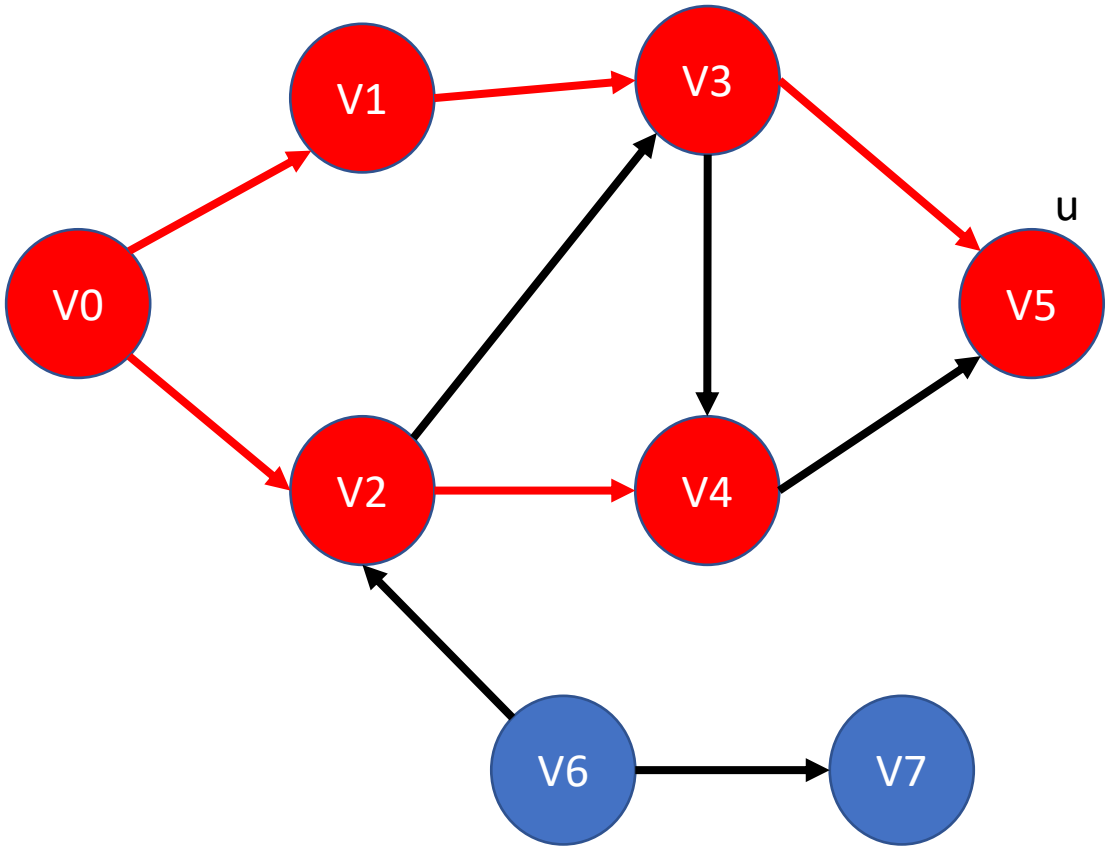


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V5

Algorithms

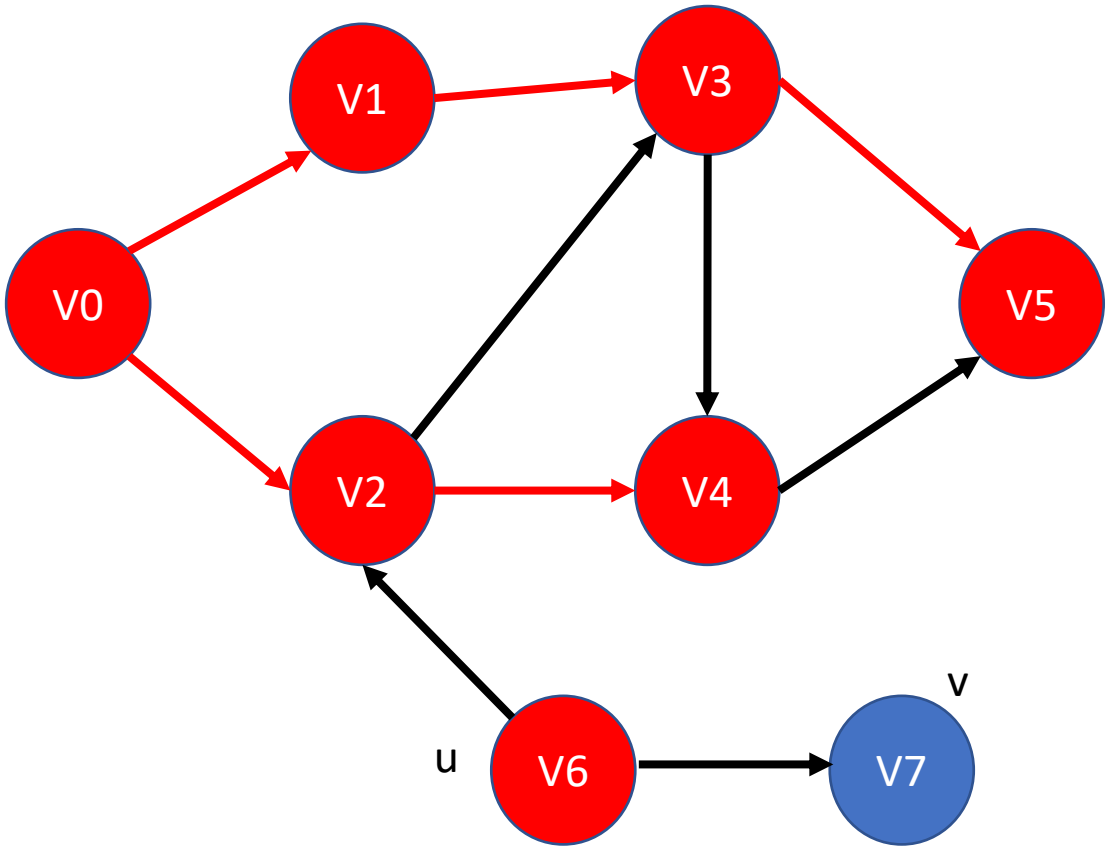
Breadth-First Search (BFS)



1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

Algorithms

Breadth-First Search (BFS)

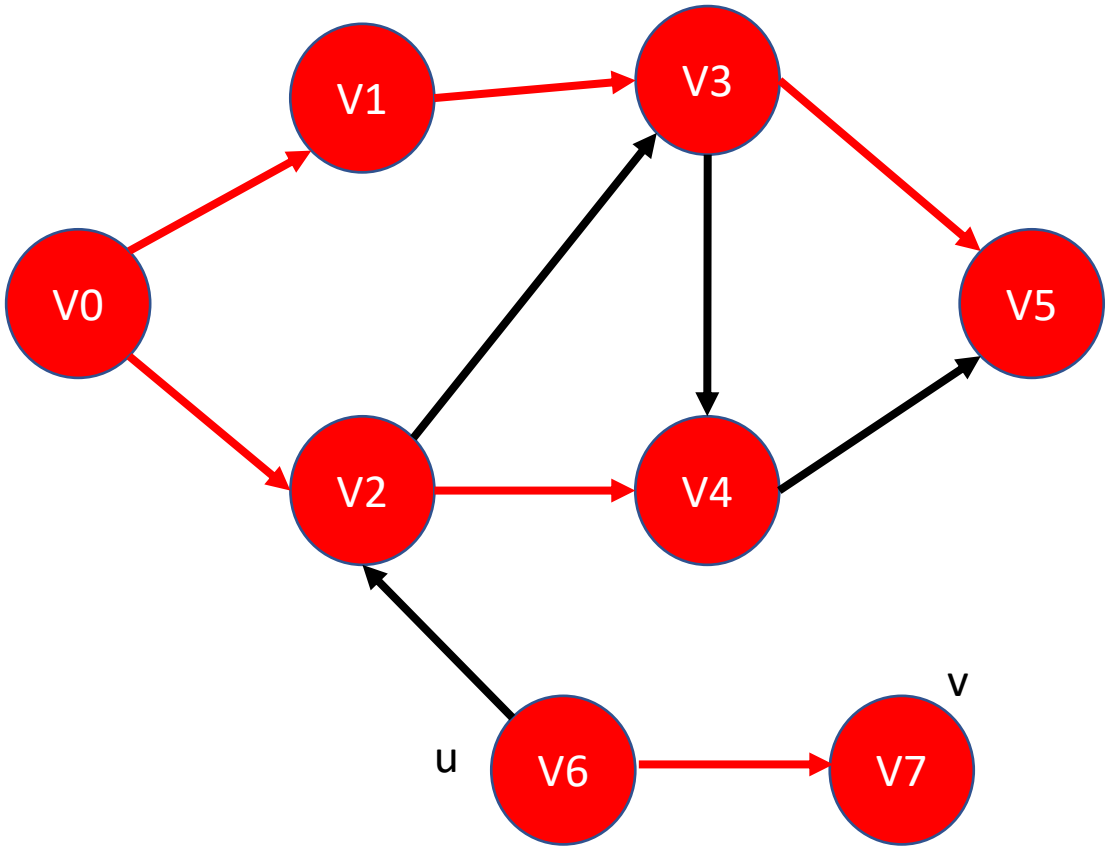


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V6

Algorithms

Breadth-First Search (BFS)

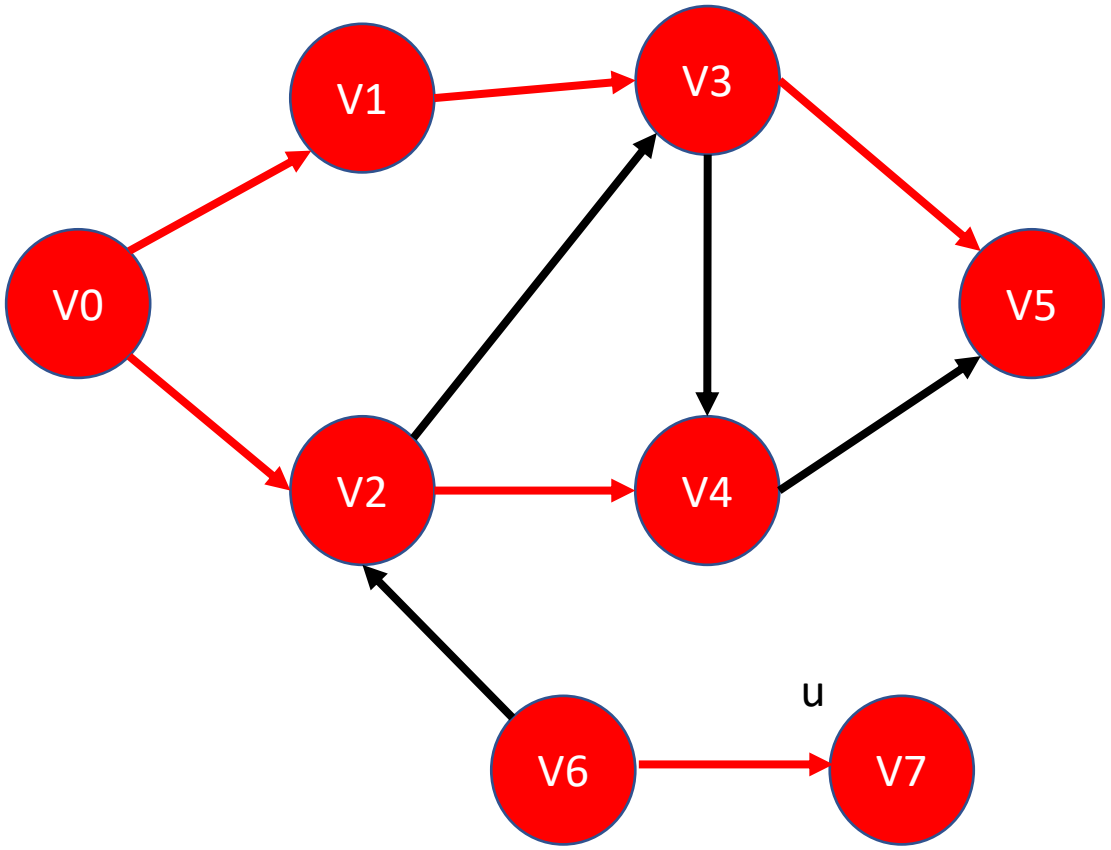


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V6	V7

Algorithms

Breadth-First Search (BFS)

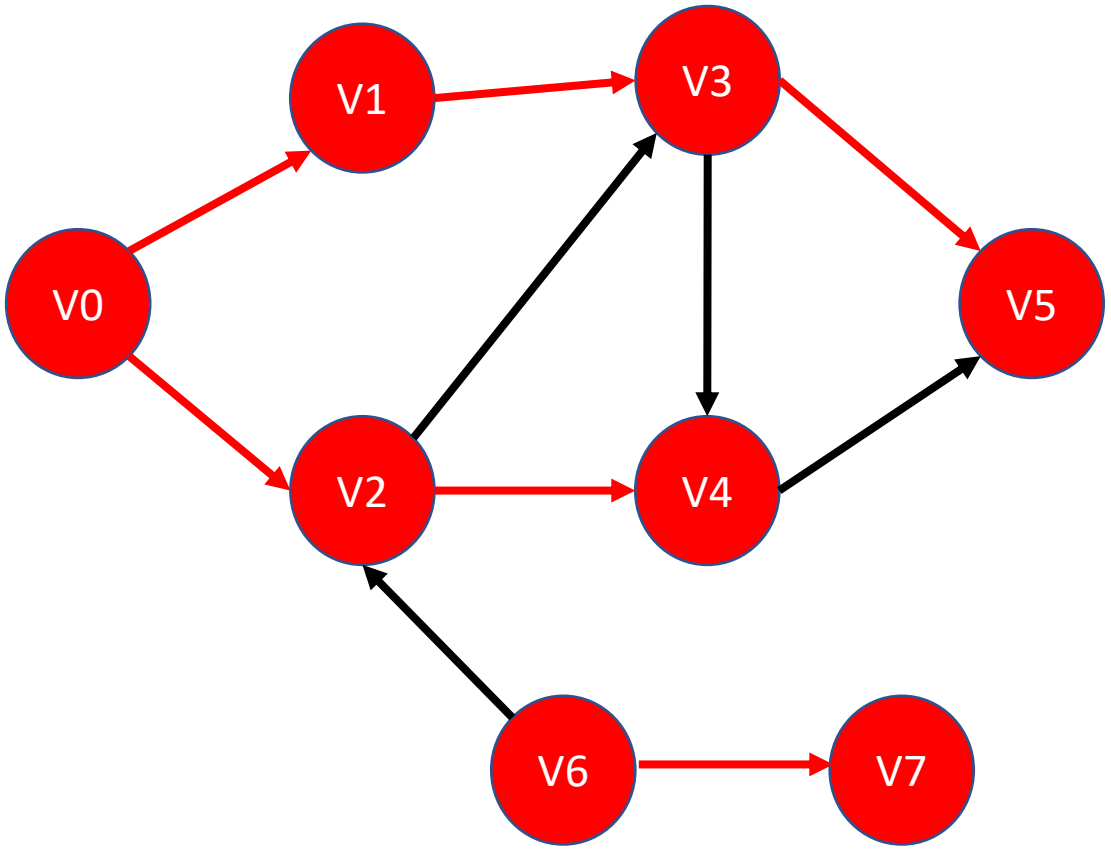


1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

V7

Algorithms

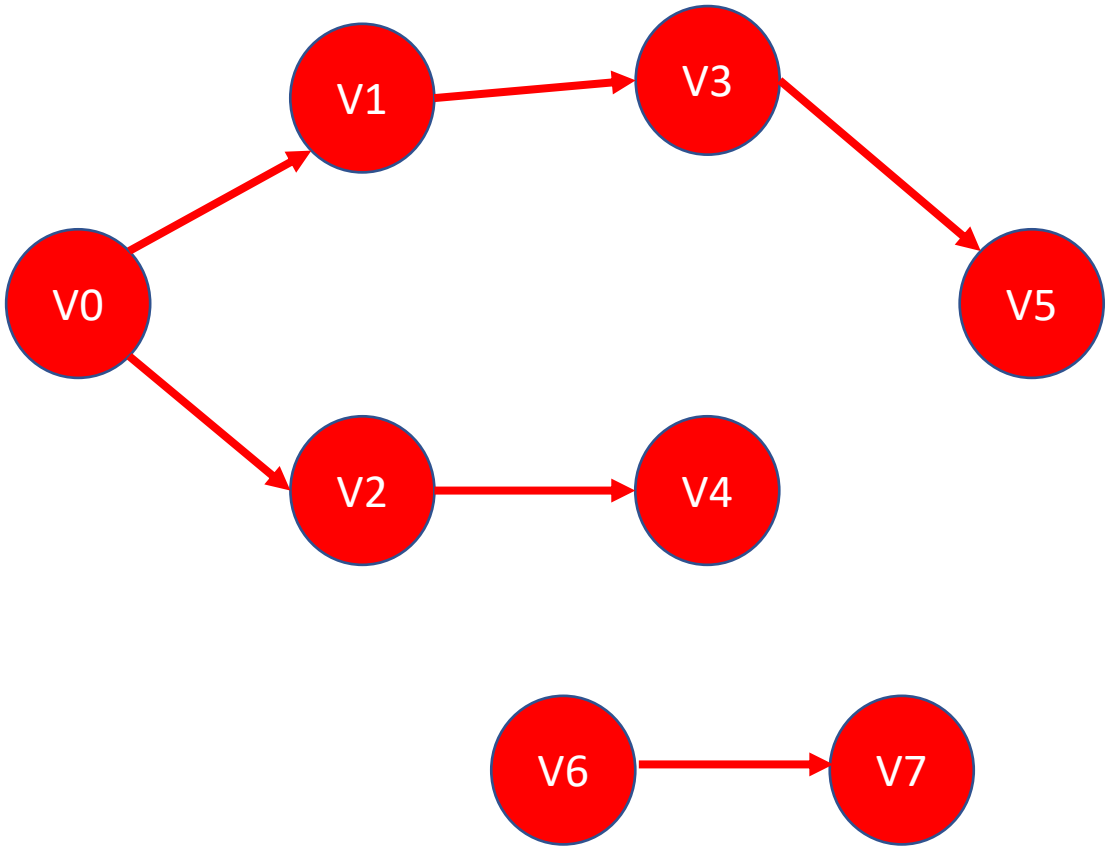
Breadth-First Search (BFS)



1. Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
   - Remove the 1st node from the list, u
   - For each neighbour v of u:
     * If v is not explored:
       ** Mark v as explored
       ** Put v at the end of the list
4. Repeat from another starting node, if there is one

Algorithms

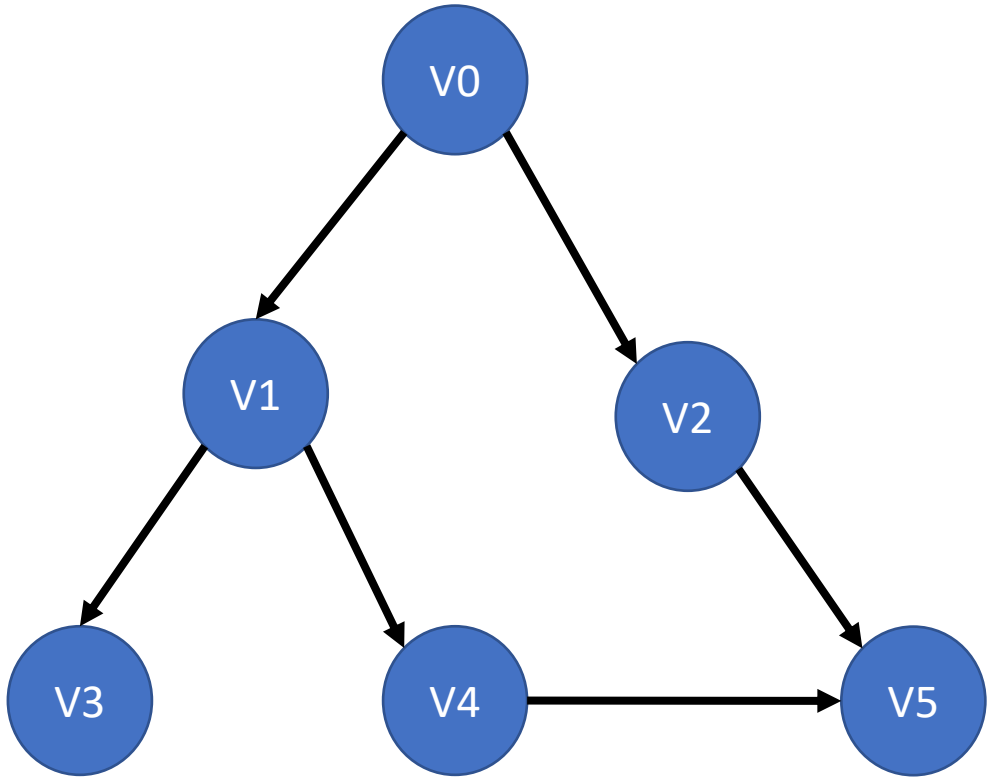
Breadth-First Search (BFS)



Two Trees or Forest

Algorithms

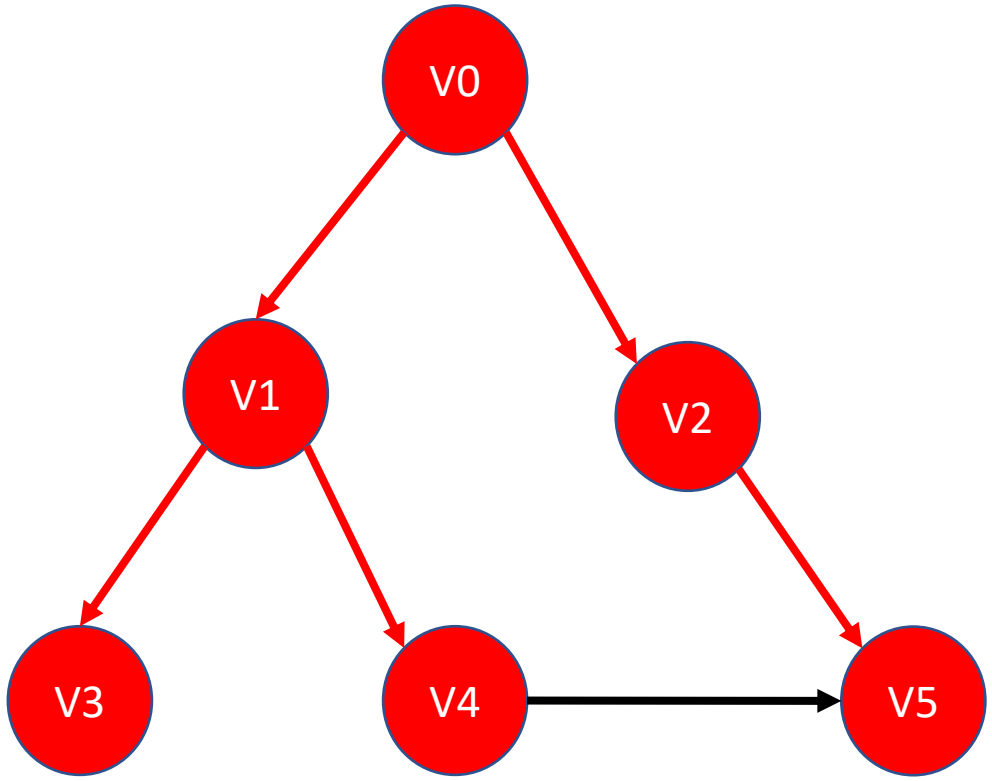
Breadth-First Search (BFS)




```
graph = {
    'V0' : ['V1','V2'],
    'V1' : ['V3', 'V4'],
    'V2' : ['V5'],
    'V3' : [],
    'V4' : ['V5'],
    'V5' : []
}

visited = [] # List to keep track of visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    s = queue.pop(0)
    print (s, end = " ")

    for neighbour in graph[s]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, 'V0')
```

Algorithms

Breadth-First Search (BFS)




```
Following is the Breadth-First Search
V0 V1 V2 V3 V4 V5
```

DFS V0 V1 V3 V4 V5 V2

BFS V0 V1 V2 V3 V4 V5

Daniel Nogueira

dnogueira@ipca.pt