

Universidade Federal de Ouro Preto

Lecture Notes

Graphs

Prof. Rodrigo Silva

April 3, 2023

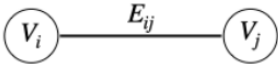
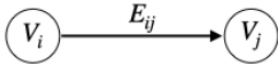
Source

- Super Study Guide - Graphs (<https://superstudy.guide/algorithms-data-structures/graphs-trees/graphs>)

1 Graphs

1.1 Definition

A graph G is defined by its vertices V and edges E and is often noted $G = (V, E)$. The following table summarizes the two main types of graph:

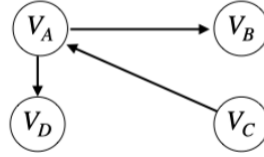
Undirected graph	Directed graph
Edges do not have a direction.	Edges have a direction.
	

! REMARK

The terms "node" and "vertex" can be used interchangeably.

1.2 Representation

Let's consider the following graph:



It can be represented in two different ways:

Type	Description	Illustration																									
Adjacency list	Collection of unordered lists where each entry V_i maps to all the nodes V_j such that E_{ij} exists in the graph.	$V_A \rightarrow \{V_B, V_D\}$ $V_B \rightarrow \emptyset$ $V_C \rightarrow \{V_A\}$ $V_D \rightarrow \emptyset$																									
Adjacency matrix	Matrix of boolean values where the entry (i, j) indicates whether E_{ij} is present in the graph. <i>In an undirected graph, this matrix is always symmetric.</i>	<table><tr><th></th><th>V_A</th><th>V_B</th><th>V_C</th><th>V_D</th></tr><tr><th>V_A</th><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><th>V_B</th><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>V_C</th><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><th>V_D</th><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>		V_A	V_B	V_C	V_D	V_A	0	1	0	1	V_B	0	0	0	0	V_C	1	0	0	0	V_D	0	0	0	0
	V_A	V_B	V_C	V_D																							
V_A	0	1	0	1																							
V_B	0	0	0	0																							
V_C	1	0	0	0																							
V_D	0	0	0	0																							

2 Graphs in C++

2.1 Adjacency list

```
1 #include <iostream>
2 #include <unordered_map>
3 #include <vector>
4 #include <stack>
5 #include <algorithm>
6
7 using namespace std;
8
9 class Graph {
10 private:
11     unordered_map<string, vector<pair<string, int>>> adj_list;
12
13 public:
14     void add_node(string node) {
15         if (adj_list.find(node) == adj_list.end()) {
16             adj_list[node] = {};
17         }
18     }
19
20     void add_edge(string node1, string node2, int weight = 1) {
21         add_node(node1);
22         add_node(node2);
```

```

23     adj_list[node1].push_back({node2, weight});
24     adj_list[node2].push_back({node1, weight});
25 }
26
27 void remove_edge(string node1, string node2) {
28     auto it = find_if(adj_list[node1].begin(), adj_list[node1].end(),
29                     [node2](const pair<string, int>& p) {
30                         return p.first == node2;
31                     });
32     if (it != adj_list[node1].end()) {
33         adj_list[node1].erase(it);
34     }
35     it = find_if(adj_list[node2].begin(), adj_list[node2].end(),
36                 [node1](const pair<string, int>& p) {
37                     return p.first == node1;
38                 });
39     if (it != adj_list[node2].end()) {
40         adj_list[node2].erase(it);
41     }
42 }
43
44 void remove_node(string node) {
45     adj_list.erase(node);
46     for (auto& [other_node, neighbours] : adj_list) {
47         auto it = find_if(neighbours.begin(), neighbours.end(),
48                         [node](const pair<string, int>& p) {
49                             return p.first == node;
50                         });
51         if (it != neighbours.end()) {
52             neighbours.erase(it);
53         }
54     }
55 }
56
57 bool is_path(string node1, string node2) {
58     unordered_set<string> visited;
59     stack<string> st;
60     st.push(node1);
61     while (!st.empty()) {
62         string node = st.top();
63         st.pop();
64         if (node == node2) {
65             return true;
66         }
67         if (visited.find(node) == visited.end()) {
68             visited.insert(node);
69             for (auto& [neighbour, weight] : adj_list[node]) {
70                 st.push(neighbour);
71             }
72         }
73     }
74     return false;
75 }
76 };
77
78 int main() {
79     Graph g;
80     g.add_edge("A", "B"); // add an edge between nodes 'A' and 'B'
81     g.add_edge("B", "C"); // add an edge between nodes 'B' and 'C'
82     for (auto& [node, neighbours] : g.adj_list) {
83         cout << node << ": ";
84         for (auto& [neighbour, weight] : neighbours) {
85             cout << "(" << neighbour << ", " << weight << ") ";
86         }
87         cout << endl;

```

```

88     }
89     // Output: A: (B, 1)
90     //           B: (A, 1) (C, 1)
91     //           C: (B, 1)
92
93     g.add_edge("A", "D"); // add an edge between nodes 'A' and 'D'
94     for (auto& [node, neighbours] : g.adj_list) {
95         cout << node << ": ";
96         for (auto& [neighbour, weight] : neighbours) {
97             cout << "(" << neighbour << ", " << weight << ") ";
98         }
99         cout << endl;
100     }
101 }

```

Listing 1: Example 1 - Graph as Adjacency List C++ (Interactive)

2.2 Adjacency matrix

```

1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  class Graph {
6  public:
7      Graph(int num_nodes) : num_nodes(num_nodes), adj_matrix(num_nodes, vector<int>(
num_nodes, 0)) {}
8
9      void add_node() {
10         num_nodes++;
11         for (auto& row : adj_matrix) {
12             row.push_back(0);
13         }
14         adj_matrix.push_back(vector<int>(num_nodes, 0));
15     }
16
17     void add_edge(int node1, int node2, int weight = 1) {
18         adj_matrix[node1][node2] = weight;
19         adj_matrix[node2][node1] = weight;
20     }
21
22     void remove_edge(int node1, int node2) {
23         adj_matrix[node1][node2] = 0;
24         adj_matrix[node2][node1] = 0;
25     }
26
27     void remove_node(int node) {
28         num_nodes--;
29         adj_matrix.erase(adj_matrix.begin() + node);
30         for (auto& row : adj_matrix) {
31             row.erase(row.begin() + node);
32         }
33     }
34
35     bool is_path(int node1, int node2) {
36         vector<bool> visited(num_nodes, false);
37         vector<int> queue;
38         queue.push_back(node1);
39         visited[node1] = true;
40
41         while (!queue.empty()) {
42             int current_node = queue.front();
43             queue.erase(queue.begin());
44             if (current_node == node2) {
45                 return true;

```

```

46         }
47         for (int neighbor = 0; neighbor < num_nodes; neighbor++) {
48             int weight = adj_matrix[current_node][neighbor];
49             if (weight > 0 && !visited[neighbor]) {
50                 visited[neighbor] = true;
51                 queue.push_back(neighbor);
52             }
53         }
54     }
55
56     return false;
57 }
58
59 private:
60     int num_nodes;
61     vector<vector<int>> adj_matrix;
62 };
63
64 int main() {
65     Graph g(3); // create a graph with 3 nodes
66     g.add_edge(0, 1); // add an edge between nodes 0 and 1
67     g.add_edge(1, 2); // add an edge between nodes 1 and 2
68     for (auto row : g.adj_matrix) {
69         for (auto val : row) {
70             cout << val << " ";
71         }
72         cout << endl;
73     } // print the adjacency matrix
74     // Output: 0 1 0
75     //          1 0 1
76     //          0 1 0
77
78     g.add_node(); // add a node to the graph
79     g.add_edge(0, 3); // add an edge between nodes 0 and 3
80     for (auto row : g.adj_matrix) {
81         for (auto val : row) {
82             cout << val << " ";
83         }
84         cout << endl;
85     } // print the adjacency matrix
86     // Output: 0 1 0 1
87     //          1 0 1 0
88     //          0 1 0 0
89     //          1 0 0 0
90
91     g.remove_edge(0, 3); // remove the edge between nodes 0 and 3
92     g.remove_node(2); // remove node 2 from the graph
93     for (auto row : g.adj_matrix) {
94         for (auto val : row) {
95             cout << val << " ";
96         }
97         cout << endl;
98     } // print the adjacency matrix
99 }

```

Listing 2: Example 1 - Graph as Adjacency List C++

3 Graphs in Python

3.1 Adjacency list

```

1 class Graph:
2     def __init__(self):
3         self.adj_list = {}

```

```

4
5     def add_node(self, node):
6         if node not in self.adj_list:
7             self.adj_list[node] = []
8
9     def add_edge(self, node1, node2, weight=1):
10        self.add_node(node1)
11        self.add_node(node2)
12        self.adj_list[node1].append((node2, weight))
13        self.adj_list[node2].append((node1, weight))
14
15    def remove_edge(self, node1, node2):
16        for i, (node, weight) in enumerate(self.adj_list[node1]):
17            if node == node2:
18                del self.adj_list[node1][i]
19                break
20        for i, (node, weight) in enumerate(self.adj_list[node2]):
21            if node == node1:
22                del self.adj_list[node2][i]
23                break
24
25    def remove_node(self, node):
26        del self.adj_list[node]
27        for other_node in self.adj_list:
28            for i, (n, w) in enumerate(self.adj_list[other_node]):
29                if n == node:
30                    del self.adj_list[other_node][i]
31                    break
32
33    def is_path(self, node1, node2):
34        visited = set()
35        stack = [node1]
36        while stack:
37            node = stack.pop()
38            if node == node2:
39                return True
40            if node not in visited:
41                visited.add(node)
42                stack.extend(neighbour for neighbour, weight in self.adj_list[node])
43        return False
44
45    # example usage
46    g = Graph()
47    g.add_edge('A', 'B') # add an edge between nodes 'A' and 'B'
48    g.add_edge('B', 'C') # add an edge between nodes 'B' and 'C'
49    print(g.adj_list) # print the adjacency list
50    # Output: {'A': [('B', 1)], 'B': [('A', 1), ('C', 1)], 'C': [('B', 1)]}
51
52    g.add_edge('A', 'D') # add an edge between nodes 'A' and 'D'
53    print(g.adj_list) # print the adjacency list
54    # Output: {'A': [('B', 1), ('D', 1)], 'B': [('A', 1), ('C', 1)], 'C': [('B', 1)], 'D':
55        [('A', 1)]}
56
57    g.remove_edge('A', 'D') # remove the edge between nodes 'A' and 'D'
58    g.remove_node('B') # remove node 'B' from the graph
59    print(g.adj_list) # print the adjacency list
60    # Output: {'A': [], 'C': []}
61
62    print(g.is_path('A', 'C')) # check if a path exists between nodes 'A' and 'C'
63    # Output: False
64    print(g.is_path('A', 'D')) # check if a path exists between nodes 'A' and 'D'
65    # Output: False
66    print(g.is_path('D', 'A')) # check if a path exists between nodes 'D' and 'A'

```

```
66 # Output: True
```

Listing 3: Example 1 - Graph as Adjacency List Python

3.2 Adjacency matrix

```
1 class Graph:
2     def __init__(self, num_nodes):
3         self.num_nodes = num_nodes
4         self.adj_matrix = [[0 for _ in range(num_nodes)] for _ in range(num_nodes)]
5
6     def add_node(self):
7         self.num_nodes += 1
8         for row in self.adj_matrix:
9             row.append(0)
10        self.adj_matrix.append([0 for _ in range(self.num_nodes)])
11
12    def add_edge(self, node1, node2, weight=1):
13        self.adj_matrix[node1][node2] = weight
14        self.adj_matrix[node2][node1] = weight
15
16    def remove_edge(self, node1, node2):
17        self.adj_matrix[node1][node2] = 0
18        self.adj_matrix[node2][node1] = 0
19
20    def remove_node(self, node):
21        self.num_nodes -= 1
22        self.adj_matrix.pop(node)
23        for row in self.adj_matrix:
24            row.pop(node)
25
26    def is_path(self, node1, node2):
27        visited = [False] * self.num_nodes
28        queue = [node1]
29        visited[node1] = True
30
31        while queue:
32            current_node = queue.pop(0)
33            if current_node == node2:
34                return True
35            for neighbor, weight in enumerate(self.adj_matrix[current_node]):
36                if weight > 0 and not visited[neighbor]:
37                    visited[neighbor] = True
38                    queue.append(neighbor)
39
40        return False
41
42
43 # example usage
44 g = Graph(3) # create a graph with 3 nodes
45 g.add_edge(0, 1) # add an edge between nodes 0 and 1
46 g.add_edge(1, 2) # add an edge between nodes 1 and 2
47 print(g.adj_matrix) # print the adjacency matrix
48 # Output: [[0, 1, 0], [1, 0, 1], [0, 1, 0]]
49
50 g.add_node() # add a node to the graph
51 g.add_edge(0, 3) # add an edge between nodes 0 and 3
52 print(g.adj_matrix) # print the adjacency matrix
53 # Output: [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0]]
54
55 g.remove_edge(0, 3) # remove the edge between nodes 0 and 3
56 g.remove_node(2) # remove node 2 from the graph
57 print(g.adj_matrix) # print the adjacency matrix
58 # Output: [[0, 1, 1], [1, 0, 0], [1, 0, 0]]
59
```

```
60 print(g.is_path(0, 2)) # check if a path exists between nodes 0 and 2
61 # Output: True
62 print(g.is_path(0, 1)) # check if a path exists between nodes 0 and 1
63 # Output: True
64 print(g.is_path(1, 2)) # check if a path exists between nodes 1 and 2
65 # Output: False
```

Listing 4: Example 1 - Graph as Matrix Python