

```
1 class Graph {
2   constructor() {
3     this.vertices = [];
4     this.edges = [];
5     this.numberOfEdges = 0;
6   }
7
8   addVertex(vertex) {
9     this.vertices.push(vertex);
10    this.edges[vertex] = [];
11  }
12
13  removeVertex(vertex) {
14    const index = this.vertices.indexOf(vertex);
15    if(~index) {
16      this.vertices.splice(index, 1);
17    }
18    while(this.edges[vertex].length) {
19      const adjacentVertex = this.edges[vertex].pop();
20      this.removeEdge(adjacentVertex, vertex);
21    }
22  }
23
24  addEdge(vertex1, vertex2) {
25    this.edges[vertex1].push(vertex2);
26    this.edges[vertex2].push(vertex1);
27    this.numberOfEdges++;
28  }
29
30  removeEdge(vertex1, vertex2) {
31    const index1 = this.edges[vertex1] ? this.edges[vertex1].indexOf(vertex2) : -1;
32    const index2 = this.edges[vertex2] ? this.edges[vertex2].indexOf(vertex1) : -1;
33    if(~index1) {
34      this.edges[vertex1].splice(index1, 1);
35      this.numberOfEdges--;
36    }
37    if(~index2) {
38      this.edges[vertex2].splice(index2, 1);
39    }
40  }
41
42  size() {
43    return this.vertices.length;
44  }
45
46  relations() {
47    return this.numberOfEdges;
48  }
49
50  traverseDFS(vertex, fn) {
51    if(!~this.vertices.indexOf(vertex)) {
52      return console.log('Vertex not found');
53    }
54    const visited = [];
55    this._traverseDFS(vertex, visited, fn);
56  }
57
58  _traverseDFS(vertex, visited, fn) {
59    visited[vertex] = true;
60    if(this.edges[vertex] !== undefined) {
```

```
61     fn(vertex);
62   }
63   for(let i = 0; i < this.edges[vertex].length; i++) {
64     if(!visited[this.edges[vertex][i]]) {
65       this._traverseDFS(this.edges[vertex][i], visited, fn);
66     }
67   }
68 }
69
70 traverseBFS(vertex, fn) {
71   if(!~this.vertices.indexOf(vertex)) {
72     return console.log('Vertex not found');
73   }
74   const queue = [];
75   queue.push(vertex);
76   const visited = [];
77   visited[vertex] = true;
78
79   while(queue.length) {
80     vertex = queue.shift();
81     fn(vertex);
82     for(let i = 0; i < this.edges[vertex].length; i++) {
83       if(!visited[this.edges[vertex][i]]) {
84         visited[this.edges[vertex][i]] = true;
85         queue.push(this.edges[vertex][i]);
86       }
87     }
88   }
89 }
90
91 pathFromTo(vertexSource, vertexDestination) {
92   if(!~this.vertices.indexOf(vertexSource)) {
93     return console.log('Vertex not found');
94   }
95   const queue = [];
96   queue.push(vertexSource);
97   const visited = [];
98   visited[vertexSource] = true;
99   const paths = [];
100
101   while(queue.length) {
102     const vertex = queue.shift();
103     for(let i = 0; i < this.edges[vertex].length; i++) {
104       if(!visited[this.edges[vertex][i]]) {
105         visited[this.edges[vertex][i]] = true;
106         queue.push(this.edges[vertex][i]);
107         // save paths between vertices
108         paths[this.edges[vertex][i]] = vertex;
109       }
110     }
111   }
112   if(!visited[vertexDestination]) {
113     return undefined;
114   }
115
116   const path = [];
117   for(var j = vertexDestination; j !== vertexSource; j = paths[j]) {
118     path.push(j);
119   }
120   path.push(j);
```

```
121     return path.reverse().join('-');
122 }
123
124 print() {
125     console.log(this.vertices.map(function(vertex) {
126         return (`${vertex} -> ${this.edges[vertex].join(', ')}').trim();
127     }, this).join(' | '));
128 }
129 }
130
131 const graph = new Graph();
132 graph.addVertex(1);
133 graph.addVertex(2);
134 graph.addVertex(3);
135 graph.addVertex(4);
136 graph.addVertex(5);
137 graph.addVertex(6);
138 graph.print(); // 1 -> | 2 -> | 3 -> | 4 -> | 5 -> | 6 ->
139 graph.addEdge(1, 2);
140 graph.addEdge(1, 5);
141 graph.addEdge(2, 3);
142 graph.addEdge(2, 5);
143 graph.addEdge(3, 4);
144 graph.addEdge(4, 5);
145 graph.addEdge(4, 6);
146 graph.print(); // 1 -> 2, 5 | 2 -> 1, 3, 5 | 3 -> 2, 4 | 4 -> 3, 5, 6 | 5 -> 1, 2, 4
    | 6 -> 4
147 console.log('graph size (number of vertices):', graph.size()); // => 6
148 console.log('graph relations (number of edges):', graph.relations()); // => 7
149 graph.traverseDFS(1, vertex => { console.log(vertex); }); // => 1 2 3 4 5 6
150 console.log('---');
151 graph.traverseBFS(1, vertex => { console.log(vertex); }); // => 1 2 5 3 4 6
152 graph.traverseDFS(0, vertex => { console.log(vertex); }); // => 'Vertex not found'
153 graph.traverseBFS(0, vertex => { console.log(vertex); }); // => 'Vertex not found'
154 console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-5-1
155 console.log('path from 3 to 5:', graph.pathFromTo(3, 5)); // => 3-2-5
156 graph.removeEdge(1, 2);
157 graph.removeEdge(4, 5);
158 graph.removeEdge(10, 11);
159 console.log('graph relations (number of edges):', graph.relations()); // => 5
160 console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-3-2-5-1
161 graph.addEdge(1, 2);
162 graph.addEdge(4, 5);
163 console.log('graph relations (number of edges):', graph.relations()); // => 7
164 console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-5-1
165 graph.removeVertex(5);
166 console.log('graph size (number of vertices):', graph.size()); // => 5
167 console.log('graph relations (number of edges):', graph.relations()); // => 4
168 console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-3-2-1
169
```