```
class Graph {
  constructor() {
    this.vertices = [];
    this.edges = [];
    this.numberOfEdges = 0;
  }

  addVertex(vertex) {
    this.vertices.push(vertex);
    this.edges[vertex] = [];
  }

  removeVertex(vertex) {
    const index = this.vertices.indexOf(vertex);
    if(~index) {
      this.vertices.splice(index, 1);
    }
    while(this.edges[vertex].length) {
      const adjacentVertex = this.edges[vertex].pop();
      this.removeEdge(adjacentVertex, vertex);
    }
  }

  addEdge(vertex1, vertex2) {
    this.edges[vertex1].push(vertex2);
    this.edges[vertex2].push(vertex1);
    this.numberOfEdges++;
  }

  removeEdge(vertex1, vertex2) {
    const index1 = this.edges[vertex1] ? this.edges[vertex1].indexOf(vertex2) : -1;
    const index2 = this.edges[vertex2] ? this.edges[vertex2].indexOf(vertex1) : -1;
    if(~index1) {
      this.edges[vertex1].splice(index1, 1);
      this.numberOfEdges--;
    }
    if(~index2) {
      this.edges[vertex2].splice(index2, 1);
    }
  }

  size() {
    return this.vertices.length;
  }

  relations() {
    return this.numberOfEdges;
  }

  traverseDFS(vertex, fn) {
    if(!~this.vertices.indexOf(vertex)) {
      return console.log('Vertex not found');
    }
    const visited = [];
    this._traverseDFS(vertex, visited, fn);
  }

  _traverseDFS(vertex, visited, fn) {
    visited[vertex] = true;
    if(this.edges[vertex] !== undefined) {
```

```
      fn(vertex);
    }
    for(let i = 0; i < this.edges[vertex].length; i++) {
      if(!visited[this.edges[vertex][i]]) {
        this._traverseDFS(this.edges[vertex][i], visited, fn);
      }
    }
  }

  traverseBFS(vertex, fn) {
    if(!~this.vertices.indexOf(vertex)) {
      return console.log('Vertex not found');
    }
    const queue = [];
    queue.push(vertex);
    const visited = [];
    visited[vertex] = true;

    while(queue.length) {
      vertex = queue.shift();
      fn(vertex);
      for(let i = 0; i < this.edges[vertex].length; i++) {
        if(!visited[this.edges[vertex][i]]) {
          visited[this.edges[vertex][i]] = true;
          queue.push(this.edges[vertex][i]);
        }
      }
    }
  }

  pathFromTo(vertexSource, vertexDestination) {
    if(!~this.vertices.indexOf(vertexSource)) {
      return console.log('Vertex not found');
    }
    const queue = [];
    queue.push(vertexSource);
    const visited = [];
    visited[vertexSource] = true;
    const paths = [];

    while(queue.length) {
      const vertex = queue.shift();
      for(let i = 0; i < this.edges[vertex].length; i++) {
        if(!visited[this.edges[vertex][i]]) {
          visited[this.edges[vertex][i]] = true;
          queue.push(this.edges[vertex][i]);
          // save paths between vertices
          paths[this.edges[vertex][i]] = vertex;
        }
      }
    }
    if(!visited[vertexDestination]) {
      return undefined;
    }

    const path = [];
    for(var j = vertexDestination; j != vertexSource; j = paths[j]) {
      path.push(j);
    }
    path.push(j);
```

```
    return path.reverse().join('-');
  }

  print() {
    console.log(this.vertices.map(function(vertex) {
      return (`${vertex} -> ${this.edges[vertex].join(', ')}`).trim();
    }, this).join(' | '));
  }
}

const graph = new Graph();
graph.addVertex(1);
graph.addVertex(2);
graph.addVertex(3);
graph.addVertex(4);
graph.addVertex(5);
graph.addVertex(6);
graph.print(); // 1 -> | 2 -> | 3 -> | 4 -> | 5 -> | 6 ->
graph.addEdge(1, 2);
graph.addEdge(1, 5);
graph.addEdge(2, 3);
graph.addEdge(2, 5);
graph.addEdge(3, 4);
graph.addEdge(4, 5);
graph.addEdge(4, 6);
graph.print(); // 1 -> 2, 5 | 2 -> 1, 3, 5 | 3 -> 2, 4 | 4 -> 3, 5, 6 | 5 -> 1, 2, 4
| 6 -> 4
console.log('graph size (number of vertices):', graph.size()); // => 6
console.log('graph relations (number of edges):', graph.relations()); // => 7
graph.traverseDFS(1, vertex => { console.log(vertex); }); // => 1 2 3 4 5 6
console.log('---');
graph.traverseBFS(1, vertex => { console.log(vertex); }); // => 1 2 5 3 4 6
graph.traverseDFS(0, vertex => { console.log(vertex); }); // => 'Vertex not found'
graph.traverseBFS(0, vertex => { console.log(vertex); }); // => 'Vertex not found'
console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-5-1
console.log('path from 3 to 5:', graph.pathFromTo(3, 5)); // => 3-2-5
graph.removeEdge(1, 2);
graph.removeEdge(4, 5);
graph.removeEdge(10, 11);
console.log('graph relations (number of edges):', graph.relations()); // => 5
console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-3-2-5-1
graph.addEdge(1, 2);
graph.addEdge(4, 5);
console.log('graph relations (number of edges):', graph.relations()); // => 7
console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-5-1
graph.removeVertex(5);
console.log('graph size (number of vertices):', graph.size()); // => 5
console.log('graph relations (number of edges):', graph.relations()); // => 4
console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-3-2-1

```