Project-8.2

Method 1 -- Ford-Fulkerson using BFS to find a path in the residual graph

I create an array to store the residual graph and an array to store the augmenting path.

```
int[][] rGraph = new int[size][size];
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        rGraph[i][j] = graph[i][j];
    }
}

int[] path = new int[size];
```

Traverse the graph to find an augmenting path. If find, the flow on this augmenting path is equal to the minimum flow of these edges, and for each edge, I reduce these flow:

```
int maxFlow = 0;
while (isPathBFS(rGraph, s, t, path)) {
    // computer the minimum flow of this path
    int minFlow = Integer.MAX_VALUE;
    for (int v = t; v != s; v = path[v]) {
        int u = path[v];
        minFlow = Math.min(minFlow, rGraph[u][v]);
    }

    // compute the residual graph
    for (int v = t; v != s; v = path[v]) {
        int u = path[v];
        rGraph[u][v] = rGraph[u][v] - minFlow;
        rGraph[v][u] = rGraph[v][u] + minFlow;
    }

    // compute the maximum flow
    maxFlow += minFlow;
}
```

I create a queue to store the path -- first in first out, and put the first node in the queue. And I create an array to store whether the node is visited.

```
GenericQueue<Integer> queue = new GenericQueue<>();
queue.enqueue(s);

boolean[] visited = new boolean[rGraph.length];
visited[s] = true;
```

For first node A in the queue, I will find if there is another node B that is not visited, and is connected with A, and the residual flow on this edge AB is not zero. If true, I will put all nodes like B in the queue and delete A, and store these nodes in the path array.

```
while (queue.getSize() > 0) {
    int u = queue.dequeue();
    for (int v = 0; v < rGraph.length; v++) {
        if (visited[v] == false && rGraph[u][v] > 0) {
            queue.enqueue(v);
            path[v] = u;
            visited[v] = true;
        }
    }

    if (visited[t] == true) {
        return true;
    }
}
```

Method 2 -- Ford-Fulkerson using DFS to find a path in the residual graph

Except for the method which is to find the augmenting path, others are same as method 1.

I create a stack to store the path -- last in first out, and put the first node in the stack. And I create an array to store whether the node is visited.

```
GenericStack<Integer> stack = new GenericStack<>();
stack.push(s);

boolean[] visited = new boolean[rGraph.length];
visited[s] = true;
```

For top node A in the stack, I will find if there is another node B that is not visited, and is connected with A, and the residual flow on this edge AB is not zero. If true, I will put all nodes like B in the stack and delete A at the first position in the queue, and store these node in the path array.

```
while (stack.getSize() > 0) {
    int u = stack.pop();
    for (int v = 0; v < rGraph.length; v++) {
        if (visited[v] == false && rGraph[u][v] > 0) {
            stack.push(v);
            path[v] = u;
            visited[v] = true;
        }
    }

    if (visited[t] == true) {
        return true;
    }
}
```

Method 3 -- Ford-Fulkerson using DFS to find a path in the original graph

I create an array to store whether the node is visited. And I also create an array to store the flow.

```
boolean[] visited = new boolean[size];

int[][] flow = new int[size][size];
```

I create a recursive method. If I find a path, I will compute the maximum flow on this path, which is the minimum flow of the flow on each edge. And all the result flow will be stored in the array flow [][].

```
while (DFS(graph, flow, s, t, visited, Integer.MAX_VALUE) > 0) {
    for (int i = 0; i < size; i++) {
        visited[i] = false;
    }
}
```

Recursive method:

```
if (u == t) {
    return minFlow;
}

for (int v = 0; v < graph.length; v++) {
    if (visited[v] == false && graph[u][v] - flow[u][v] > 0) {
        int i = DFS(graph, flow, v, t, visited, Math.min(minFlow, graph[u][v] - flow[u][v]));
        if (i > 0) {
            flow[u][v] = flow[u][v] + i;
            flow[v][u] = flow[v][u] - i;
            return i;
        }
    }
}
```

Due to the theorem that the value of a maximum flow is equal to the capacity of a minimum cut, I add the whole flow depart from the source node as the maximum flow.

```
int maxFlow = 0;
for (int i = 0; i < size; i++) {
    maxFlow = maxFlow + flow[0][i];
}
```