```java
/**
 * Created by Kyodu on 25.03.17.
 */

import java.util.*;

public class Dinics {

        static class Edge {
            int t, rev, cap, f;

            public Edge(int t, int rev, int cap) {
                this.t = t;
                this.rev = rev;
                this.cap = cap;
            }
        }

        private List<Edge>[] graph;

        public Dinics(){

            /*graph = createGraph(6); //erster test wenig
            addEdge(graph, 0, 1, 16);
            addEdge(graph, 0, 2, 13);
            addEdge(graph, 1, 3, 12);
            addEdge(graph, 2, 1, 4);
            addEdge(graph, 2, 4, 14);
            addEdge(graph, 3, 2, 9);
            addEdge(graph, 3, 5, 20);
            addEdge(graph, 4, 3, 7);
            addEdge(graph, 4, 5, 4);
*/

            /*graph = createGraph(6); //erster test viel
            addEdge(graph, 0, 1, 16);
            addEdge(graph, 0, 2, 13);
            addEdge(graph, 1, 3, 12);
            addEdge(graph, 2, 1, 4);
            addEdge(graph, 2, 4, 14);
            addEdge(graph, 3, 2, 9);
            addEdge(graph, 3, 5, 20);
            addEdge(graph, 4, 3, 7);
            addEdge(graph, 4, 5, 4);
            //zusätzliche Kanten
            addEdge(graph, 0, 3, 5);
            addEdge(graph, 0, 4, 5);
            addEdge(graph, 1, 4, 5);
            */

```

```
 51
 52              /*graph = createGraph(8); //zweiter Test kurz
 53              addEdge(graph, 0, 1, 38);
 54              addEdge(graph, 0, 2, 1);
 55              addEdge(graph, 0, 6, 2);
 56              addEdge(graph, 1, 4, 13);
 57              addEdge(graph, 1, 2, 8);
 58              addEdge(graph, 1, 3, 10);
 59              addEdge(graph, 2, 3, 26);
 60              addEdge(graph, 3, 4, 20);
 61              addEdge(graph, 3, 5, 8);
 62              addEdge(graph, 3, 6, 24);
 63              addEdge(graph, 3, 7, 1);
 64              addEdge(graph, 4, 5, 1);
 65              addEdge(graph, 4, 2, 2);
 66              addEdge(graph, 4, 7, 7);
 67              addEdge(graph, 5, 7, 7);
 68              addEdge(graph, 6, 7, 27);
 69 */
 70              /*graph = createGraph(8); //zweiter Test lang
 71              addEdge(graph, 0, 1, 38);
 72              addEdge(graph, 0, 2, 1);
 73              addEdge(graph, 0, 6, 2);
 74              addEdge(graph, 1, 4, 13);
 75              addEdge(graph, 1, 2, 8);
 76              addEdge(graph, 1, 3, 10);
 77              addEdge(graph, 2, 3, 26);
 78              addEdge(graph, 3, 4, 20);
 79              addEdge(graph, 3, 5, 8);
 80              addEdge(graph, 3, 6, 24);
 81              addEdge(graph, 3, 7, 1);
 82              addEdge(graph, 4, 5, 1);
 83              addEdge(graph, 4, 2, 2);
 84              addEdge(graph, 4, 7, 7);
 85              addEdge(graph, 5, 7, 7);
 86              addEdge(graph, 6, 7, 27);
 87              //zusätzliche kanten
 88              addEdge(graph, 0, 4, 5);
 89              addEdge(graph, 1, 5, 5);
 90              addEdge(graph, 2, 5, 5);
 91              addEdge(graph, 6, 2, 5);*/
 92
 93              /*graph = createGraph(11); //dritter test kurz
 94              addEdge(graph, 0, 1, 38);
 95              addEdge(graph, 0, 2, 1);
 96              addEdge(graph, 0, 6, 20);
 97              addEdge(graph, 1, 4, 13);
 98              addEdge(graph, 1, 2, 8);
 99              addEdge(graph, 1, 3, 10);
100              addEdge(graph, 2, 3, 26);
```

```
101             addEdge(graph, 3, 4, 20);
102             addEdge(graph, 3, 5, 8);
103             addEdge(graph, 3, 6, 24);
104             addEdge(graph, 3, 10, 1);
105             addEdge(graph, 4, 5, 1);
106             addEdge(graph, 4, 2, 2);
107             addEdge(graph, 4, 10, 7);
108             addEdge(graph, 5, 10, 7);
109             addEdge(graph, 6, 10, 27);
110             addEdge(graph, 6, 7, 19);
111             addEdge(graph, 7, 8, 11);
112             addEdge(graph, 7, 9, 4);
113             addEdge(graph, 8, 9, 12);
114             addEdge(graph, 9, 5, 5);
115             addEdge(graph, 9, 10, 12);*/
116
117             graph = createGraph(11); //dritter test kurz
118             addEdge(graph, 0, 1, 38);
119             addEdge(graph, 0, 2, 1);
120             addEdge(graph, 0, 6, 20);
121             addEdge(graph, 1, 4, 13);
122             addEdge(graph, 1, 2, 8);
123             addEdge(graph, 1, 3, 10);
124             addEdge(graph, 2, 3, 26);
125             addEdge(graph, 3, 4, 20);
126             addEdge(graph, 3, 5, 8);
127             addEdge(graph, 3, 6, 24);
128             addEdge(graph, 3, 10, 1);
129             addEdge(graph, 4, 5, 1);
130             addEdge(graph, 4, 2, 2);
131             addEdge(graph, 4, 10, 7);
132             addEdge(graph, 5, 10, 7);
133             addEdge(graph, 6, 10, 27);
134             addEdge(graph, 6, 7, 19);
135             addEdge(graph, 7, 8, 11);
136             addEdge(graph, 7, 9, 4);
137             addEdge(graph, 8, 9, 12);
138             addEdge(graph, 9, 5, 5);
139             addEdge(graph, 9, 10, 12);
140             //zusätzliche kanten
141             addEdge(graph, 6, 2, 2);
142             addEdge(graph, 6, 9, 2);
143             addEdge(graph, 7, 3, 2);
144             addEdge(graph, 7, 5, 5);
145             addEdge(graph, 3, 9, 5);
146
147
148         }
149
150         public int Maxflow(){
```

```
151             //return maxFlow(graph, 0, 5); //erster test
152             //return maxFlow(graph, 0, 7); //zweiter test
153             return maxFlow(graph, 0, 10); //dritter test
154 
155         }
156 
157         public static List<Edge>[] createGraph(int nodes)
    {
158             List<Edge>[] graph = new List[nodes];
159             for (int i = 0; i < nodes; i++)
160                 graph[i] = new ArrayList<>();
161             return graph;
162         }
163 
164         public static void addEdge(List<Edge>[] graph, int
     s, int t, int cap) {
165             graph[s].add(new Edge(t, graph[t].size(), cap)
    );
166             graph[t].add(new Edge(s, graph[s].size() - 1,
    0));
167         }
168 
169         static boolean dinicBfs(List<Edge>[] graph, int
    src, int dest, int[] dist) {
170             Arrays.fill(dist, -1);
171             dist[src] = 0;
172             int[] Q = new int[graph.length];
173             int sizeQ = 0;
174             Q[sizeQ++] = src;
175             for (int i = 0; i < sizeQ; i++) {
176                 int u = Q[i];
177                 for (Edge e : graph[u]) {
178                     if (dist[e.t] < 0 && e.f < e.cap) {
179                         dist[e.t] = dist[u] + 1;
180                         Q[sizeQ++] = e.t;
181                     }
182                 }
183             }
184             return dist[dest] >= 0;
185         }
186 
187         static int dinicDfs(List<Edge>[] graph, int[] ptr,
     int[] dist, int dest, int u, int f) {
188             if (u == dest)
189                 return f;
190             for (; ptr[u] < graph[u].size(); ++ptr[u]) {
191                 Edge e = graph[u].get(ptr[u]);
192                 if (dist[e.t] == dist[u] + 1 && e.f < e.
    cap) {
193                     int df = dinicDfs(graph, ptr, dist,
```

```
193 dest, e.t, Math.min(f, e.cap - e.f));
194                     if (df > 0) {
195                         e.f += df;
196                         graph[e.t].get(e.rev).f -= df;
197                         return df;
198                     }
199                 }
200             }
201             return 0;
202         }
203 
204         public static int maxFlow(List<Edge>[] graph, int
    src, int dest) {
205             int flow = 0;
206             int[] dist = new int[graph.length];
207             while (dinicBfs(graph, src, dest, dist)) {
208                 int[] ptr = new int[graph.length];
209                 while (true) {
210                     int df = dinicDfs(graph, ptr, dist,
    dest, src, Integer.MAX_VALUE);
211                     if (df == 0)
212                         break;
213                     flow += df;
214                 }
215             }
216             return flow;
217         }
218 
219 }
```

```java
/**
 * Created by Kyodu on 25.03.17.
 */
// Java program for implementation of Ford Fulkerson algorithm
import java.util.*;
import java.lang.*;
import java.io.*;
import java.util.LinkedList;

public class EdmondsKarp {
    /*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */



        private final int V;
        public EdmondsKarp(int notes){
            V = notes; //Number of vertices in graph
        }
        /* Returns true if there is a path from source 's' to sink
          't' in residual graph. Also fills parent[] to store the
            path */
        boolean bfs(int rGraph[][], int s, int t, int parent[])
        {
            // Create a visited array and mark all vertices as not
            // visited
            boolean visited[] = new boolean[V];
            for(int i=0; i<V; ++i)
                visited[i]=false;

            // Create a queue, enqueue source vertex and mark
            // source vertex as visited
            LinkedList<Integer> queue = new LinkedList<Integer>();
            queue.add(s);
            visited[s] = true;
            parent[s]=-1;

            // Standard BFS Loop
            while (queue.size()!=0)
            {
```

```
43                  int u = queue.poll();
44
45                  for (int v=0; v<V; v++)
46                  {
47                      if (visited[v]==false && rGraph[u][v] >
    0)
48                      {
49                          queue.add(v);
50                          parent[v] = u;
51                          visited[v] = true;
52                      }
53                  }
54              }
55
56              // If we reached sink in BFS starting from
   source, then
57              // return true, else false
58              return (visited[t] == true);
59          }
60
61          // Returns tne maximum flow from s to t in the
   given graph
62          int fordFulkerson(int graph[][], int s, int t)
63          {
64              int u, v;
65
66              // Create a residual graph and fill the
   residual graph
67              // with given capacities in the original graph
   as
68              // residual capacities in residual graph
69
70              // Residual graph where rGraph[i][j] indicates
71              // residual capacity of edge from i to j (if
   there
72              // is an edge. If rGraph[i][j] is 0, then there
    is
73              // not)
74              int rGraph[][] = new int[V][V];
75
76              for (u = 0; u < V; u++)
77                  for (v = 0; v < V; v++)
78                      rGraph[u][v] = graph[u][v];
79
80              // This array is filled by BFS and to store
   path
81              int parent[] = new int[V];
82
83              int max_flow = 0;  // There is no flow
   initially
```

```
 84
 85             // Augment the flow while there is path from
    source
 86             // to sink
 87             while (bfs(rGraph, s, t, parent))
 88             {
 89                 // Find minimum residual capacity of the
    edges
 90                 // along the path filled by BFS. Or we can
     say
 91                 // find the maximum flow through the path
    found.
 92                 int path_flow = Integer.MAX_VALUE;
 93                 for (v=t; v!=s; v=parent[v])
 94                 {
 95                     u = parent[v];
 96                     path_flow = Math.min(path_flow, rGraph
    [u][v]);
 97                 }
 98
 99                 // update residual capacities of the edges
     and
100                 // reverse edges along the path
101                 for (v=t; v != s; v=parent[v])
102                 {
103                     u = parent[v];
104                     rGraph[u][v] -= path_flow;
105                     rGraph[v][u] += path_flow;
106                 }
107
108                 // Add path flow to overall flow
109                 max_flow += path_flow;
110             }
111
112             // Return the overall flow
113             return max_flow;
114         }
115
116 }
117
```

```java
/**
 * Created by Kyodu on 25.03.17.
 */

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.io.*;


public class MaximumFlow {

    public static void main(String...arg)throws IOException
{

        int[][] graph;
        int numberOfNodes;
        int source;
        int sink;
        int maxFlowD =0;
        int maxFlowK =0;
        long timeStart;
        long timeEnd;
        String filename = "log3_viel.csv";
        FileWriter fw = new FileWriter(filename, true); //
the true will append the new data

       /* Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of nodes");
        numberOfNodes = scanner.nextInt();
        graph = new int[numberOfNodes + 1][numberOfNodes +
1];

        /*System.out.println("Enter the graph matrix");
        for (int sourceVertex = 0; sourceVertex <=
numberOfNodes; sourceVertex++)
        {
            for (int destinationVertex = 0;
destinationVertex <= numberOfNodes; destinationVertex++)
            {
                graph[sourceVertex][destinationVertex] =
scanner.nextInt();
            }
        }

        System.out.println("Enter the source of the graph
");
        source= scanner.nextInt();

        System.out.println("Enter the sink of the graph");
```

```
44          sink = scanner.nextInt();
45  */
46          /*FordFulkerson fordFulkerson = new FordFulkerson(
    numberOfNodes);
47          long timeStart = System.nanoTime();
48          maxFlow = fordFulkerson.fordFulkerson(graph, source
    , sink);
49          long timeEnd = System.nanoTime();
50          System.out.println("The Max Flow is " + maxFlow);
51          System.out.println("Verlaufszeit der Schleife
    : " + (timeEnd - timeStart) + " Nanosekunden.");
52          */
53
54
55          // Usage example
56          // Driver program to test above functions
57          // Let us create a graph shown in the above example
58
59
60          /*int graph2[][] =new int[][]{ //test 1 wenig
61                  {0, 16, 13, 0, 0, 0}, //0
62                  {0, 0, 0, 12, 0, 0}, //1
63                  {0, 4, 0, 0, 14, 0},  //2
64                  {0, 0, 9, 0, 0, 20},  //3
65                  {0, 0, 0, 7, 0, 4},   //4
66                  {0, 0, 0, 0, 0, 0}    //5
67
68
69          };*/
70
71         /* int graph2[][] =new int[][] { //test 1 viel
72                  {0, 16, 13, 5, 5, 0}, //0
73                  {0, 0, 0, 12, 5, 0}, //1
74                  {0, 4, 0, 0, 14, 0},  //2
75                  {0, 0, 9, 0, 0, 20},  //3
76                  {0, 0, 0, 7, 0, 4},   //4
77                  {0, 0, 0, 0, 0, 0}    //5
78  };*/
79
80          /*int graph2[][] =new int[][] { //zweiter test
    wenig
81                  {0, 38, 1, 0, 0, 0, 2 ,0}, //0
82                  {0, 0, 8, 10, 13, 0, 0 ,0}, //1
83                  {0, 0, 0, 26, 0, 0, 0 ,0}, //2
84                  {0, 0, 0, 0, 20, 8, 24, 1}, //3
85                  {0, 0, 2, 0, 0, 1, 0 ,7}, //4
86                  {0, 0, 0, 0, 0, 0, 0 ,7}, //5
87                  {0, 0, 0, 0, 0, 0, 0 ,27}, //6
88                  {0, 0, 0, 0, 0, 0, 0 ,0}, //7
89          };*/
```

```
 90
 91          /*int graph2[][] =new int[][] { //zweiter test
    viel
 92                  {0, 38, 1, 0, 5, 0, 2 ,0}, //0
 93                  {0, 0, 8, 10, 13, 5, 0 ,0}, //1
 94                  {0, 0, 0, 26, 0, 5, 0 ,0}, //2
 95                  {0, 0, 0, 0, 20, 8, 24, 1}, //3
 96                  {0, 0, 2, 0, 0, 1, 0 ,7}, //4
 97                  {0, 0, 0, 0, 0, 0, 0 ,7}, //5
 98                  {0, 0, 5, 0, 0, 0, 0 ,27}, //6
 99                  {0, 0, 0, 0, 0, 0, 0 ,0}, //7
100          };*/
101 /*
102          int graph2[][] =new int[][]{ //dritter test wenig
103                  {0, 38, 1, 0, 0, 0, 20, 0, 0, 0, 0}, //0
104                  {0, 0, 8, 10, 13, 0, 0, 0, 0, 0, 0}, //1
105                  {0, 0, 0, 26, 0, 0, 0, 0, 0, 0, 0}, //2
106                  {0, 0, 0, 0, 20, 8, 24, 0, 0, 0, 1}, //3
107                  {0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 7}, //4
108                  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7}, //5
109                  {0, 0, 0, 0, 0, 0, 0, 19, 0, 0, 27}, //6
110                  {0, 0, 0, 0, 0, 0, 0, 0, 11, 4, 0}, //7
111                  {0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0}, //8
112                  {0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 12}, //9
113                  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0} //10
114          };*/
115
116          int graph2[][] =new int[][]{ //dritter test mit
    viel
117                  {0, 38, 1, 0, 0, 0, 20, 0, 0, 0, 0}, //0
118                  {0, 0, 8, 10, 13, 0, 0, 0, 0, 0, 0}, //1
119                  {0, 0, 0, 26, 0, 0, 0, 0, 0, 0, 0}, //2
120                  {0, 0, 0, 0, 20, 8, 24, 0, 0, 5, 1}, //3
121                  {0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 7}, //4
122                  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7}, //5
123                  {0, 0, 2, 0, 0, 0, 0, 19, 0, 2, 27}, //6
124                  {0, 0, 0, 2, 0, 5, 0, 0, 11, 4, 0}, //7
125                  {0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0}, //8
126                  {0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 12}, //9
127                  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0} //10
128          };
129
130          //EdmondsKarp m = new EdmondsKarp(6); //erster
    test
131          //EdmondsKarp m = new EdmondsKarp(8); //zweiter
    test
132          EdmondsKarp m = new EdmondsKarp(11); //dritter
    test
133
134          long tmp =0;
```

```
135         try {
136             fw.write("Edmonds Karp" + "," + "Dinic \n");//
    appends the string to the file
137         } catch (IOException ioe) {
138             System.err.println("IOException: " + ioe.
    getMessage());
139         }
140         int faktor =1000000; //zum versuch 10000000
141         for(int j= 0 ;j<100; j++) {
142             long TimeK =0;
143             long TimeD =0;
144             for (int i = 0; i < faktor; i++) {
145                 timeStart = System.nanoTime();
146                 //maxFlowK = m.fordFulkerson(graph2, 0, 5
    ); //erster test
147                 //maxFlowK = m.fordFulkerson(graph2, 0, 7
    ); //zweiter test
148                 maxFlowK = m.fordFulkerson(graph2, 0, 10);
     //dritter test
149                 timeEnd = System.nanoTime();
150                 TimeK = TimeK + (timeEnd - timeStart);
151 
152             }
153 
154             for (int i = 0; i < faktor; i++) {
155                 Dinics d = new Dinics();
156                 timeStart = System.nanoTime();
157                 maxFlowD = d.Maxflow();
158                 timeEnd = System.nanoTime();
159                 TimeD = TimeD + (timeEnd - timeStart);
160 
161             }
162             try {
163                 fw.write(TimeK/faktor + "," + TimeD/faktor
    +"\n");//appends the string to the file
164 
165             } catch (IOException ioe2) {
166                 System.err.println("IOException: " + ioe2.
    getMessage());
167             }
168             System.out.println("Durchlauf :" + j + " Max
    Flow Karp: " + maxFlowK + " Max Flow Dinic: "+ maxFlowD);
169         }
170         fw.close();
171         //scanner.close();
172     }
173 }
174 
175 
```