Задача 4

**Файл: graph.hpp**

#ifndef graph\_hpp

#define graph\_hpp

#include <iostream>

#include <list>

#include <stack>

#include <vector>

#include "pair.hpp"

**class** Graph

{

**private**:

**int** v\_;

std::vector<std::list<**int**>> adj;

**public**:

Graph(**int** v);

std::list<**int**> DFS(**int** v, std::vector<**bool**> &visited, std::list<**int**> &list, std::vector<**int**> &numSCC, **int** count);

**void** addEdge(**int** v, **int** adjV);

**void** fillOrder(**int** v, std::vector<**bool**> &visited, std::stack<**int**>& Stack);

**void** printGraph();

**void** printCss(SCCPair &pair);

Graph appealGraph();

Graph metaGraph();

SCCPair SCC();

};

#endif /\* graph\_hpp \*/

**Файл: graph.cpp**

#include "graph.hpp"

Graph::Graph(**int** v)

{

**this**->v\_ = v;

adj.resize(v\_);

}

std::list<**int**> Graph::DFS(**int** v, std::vector<**bool**> &visited, std::list<**int**> &list, std::vector<**int**> &numSCC, **int** count)

{

visited[v] = **true**;

numSCC[v] = count;

list.push\_back(v);

**for** (**auto** i = adj[v].begin(); i != adj[v].end(); ++i)

{

**if** (!visited[\*i])

{

DFS(\*i, visited, list, numSCC, count);

}

}

**return** list;

}

**void** Graph::addEdge(**int** v, **int** adjV)

{

adj[v].push\_back(adjV);

}

Graph Graph::appealGraph()

{

Graph appealG(v\_);

**for** (**int** v = 0; v < v\_; v++)

{

**for** (**auto** i = adj[v].begin(); i != adj[v].end(); ++i)

{

appealG.adj[\*i].push\_back(v);

}

}

**return** appealG;

}

**void** Graph::printGraph()

{

**for** (**int** v = 0; v < v\_; v++)

{

std::cout << v << " ";

**for** (**auto** i = adj[v].begin(); i != adj[v].end(); ++i)

{

std::cout << \*i << " ";

}

std::cout << '\n';

}

}

**void** Graph::printCss(SCCPair &pair)

{

**for** (**int** v = 0; v < pair.sccList.size(); v++)

{

std::cout << "SCC " << v << ": ";

**for** (**auto** i = pair.sccList[v].begin(); i != pair.sccList[v].end(); ++i)

{

std::cout << \*i << " ";

}

std::cout << '\n';

}

}

**void** Graph::fillOrder(**int** v, std::vector<**bool**> &visited, std::stack<**int**> &Stack)

{

visited[v] = **true**;

**for** (**auto** i = adj[v].begin(); i != adj[v].end(); i++)

{

**if** (!visited[\*i])

{

fillOrder(\*i, visited, Stack);

}

}

Stack.push(v);

}

SCCPair Graph::SCC()

{

std::stack<**int**> stack;

std::vector<**bool**> visited(v\_);

**for** (**int** i = 0; i < v\_; i++)

{

visited[i] = **false**;

}

**for** (**int** i = 0; i < v\_; i++)

{

**if** (visited[i] == **false**)

{

fillOrder(i, visited, stack);

}

}

Graph gr = appealGraph();

**for** (**int** i = 0; i < v\_; i++)

{

visited[i] = **false**;

}

**int** count = 0;

std::vector<**int**> numSCC(v\_);

std::vector<std::list<**int**>> listsOfSCC;

**while** (stack.empty() == **false**)

{

**int** v = stack.top();

stack.pop();

**if** (visited[v] == **false**)

{

std::list<**int**> listCss;

gr.DFS(v, visited, listCss, numSCC, count);

listsOfSCC.push\_back(listCss);

count++;

}

}

**return** SCCPair(listsOfSCC, numSCC);

}

Graph Graph::metaGraph()

{

SCCPair graphScc = SCC();

Graph metaGraph(graphScc.sccList.size());

std::vector<**bool**> used(graphScc.sccList.size(), **false**);

**for** (**int** n = 0; n < graphScc.sccList.size(); n++)

{

**for** (**auto** v = graphScc.sccList[n].begin(); v != graphScc.sccList[n].end(); v++)

{

**for** (**auto** i = adj[\*v].begin(); i != adj[\*v].end(); i++)

{

**if** (n != graphScc.numSCC[\*i] && !used[graphScc.numSCC[\*i]])

{

metaGraph.addEdge(n, graphScc.numSCC[\*i]);

used[graphScc.numSCC[\*i]] = **true**;

}

}

}

**for** (**auto** iter = metaGraph.adj[n].begin(); iter != metaGraph.adj[n].end(); iter++)

{

used[\*iter] = **false**;

}

}

**return** metaGraph;

}

**Файл: sccpair.hpp**

#ifndef pair\_hpp

#define pair\_hpp

#include <list>

#include <vector>

**struct** SCCPair

{

std::vector<std::list<**int**>> sccList;

std::vector<**int**> numSCC;

SCCPair(std::vector<std::list<**int**>> sccList, std::vector<**int**> sccCount)

{

**this**->sccList = sccList;

**this**->numSCC = sccCount;

}

};

#endif /\* pair\_hpp \*/

**Файл: main.cpp**

#include <iostream>

#include <sstream>

#include "graph.hpp"

#include "pair.hpp"

**using** **namespace** std;

**int** main()

{

**int** size = 0;

cin >> size;

cin.ignore();

Graph graph(size);

**for** (**int** i = 0; i < size; i++)

{

string str;

getline(cin, str);

std::stringstream stream(str);

std::string v;

stream >> v;

std::string u;

**int** v1 = stoi(v);

**while** (u != "-1")

{

stream >> u;

**if** (u != "-1")

{

**int** u1 = stoi(u);

graph.addEdge(v1, u1);

}

}

}

cout << "\n";

SCCPair pair = graph.SCC();

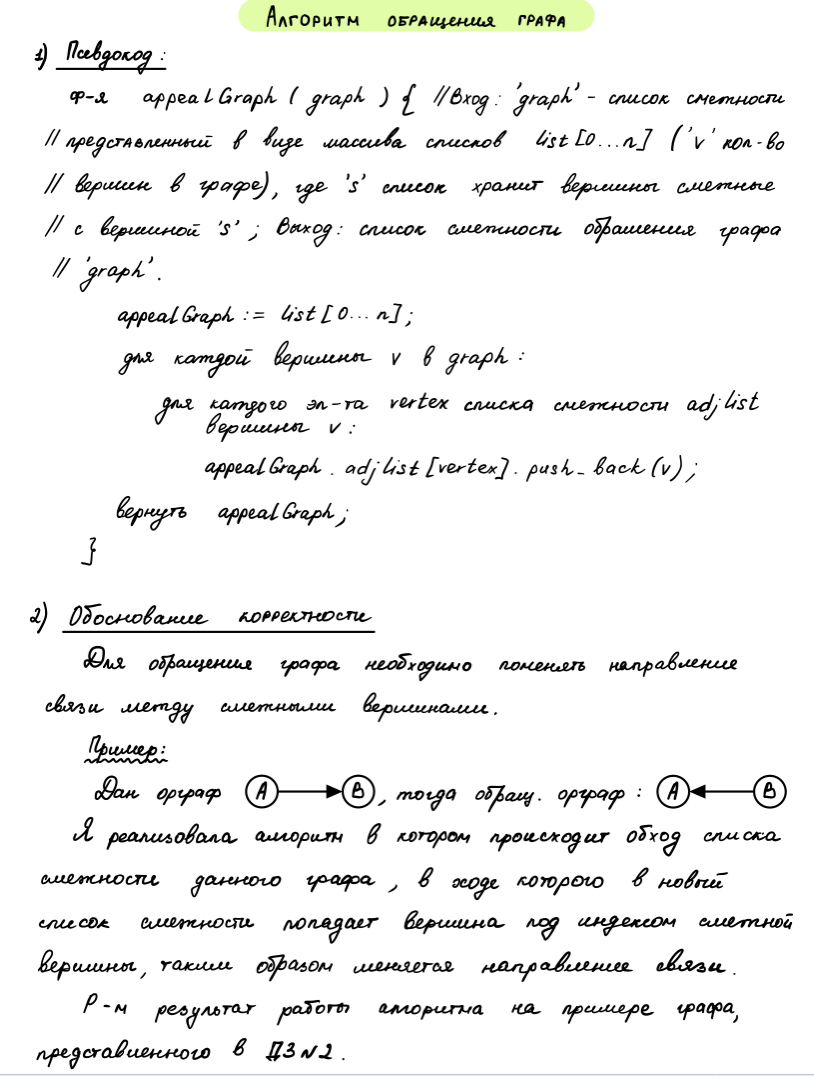
graph.printCss(pair);

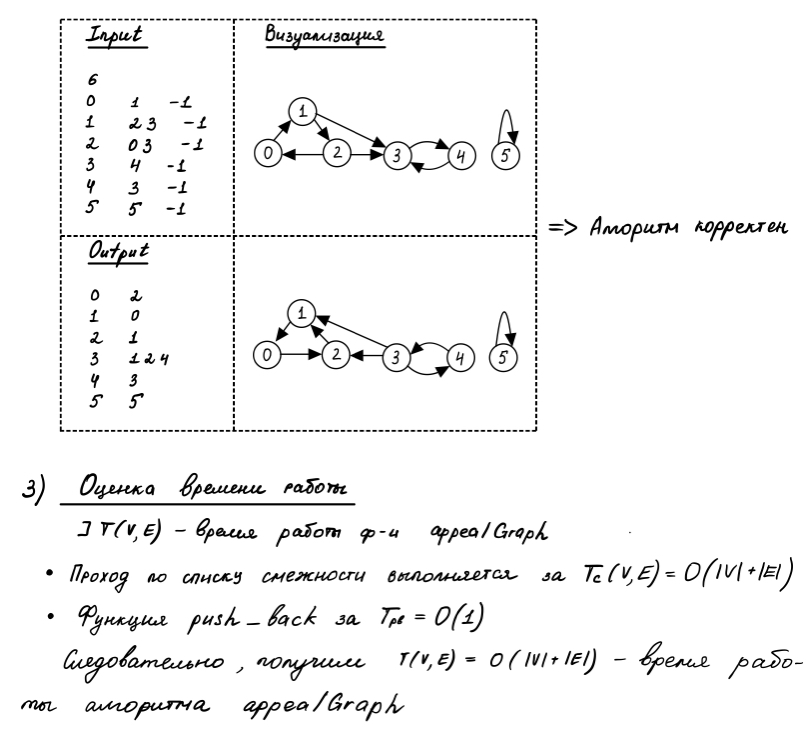
std::cout << "\nMeta-graph:\n";

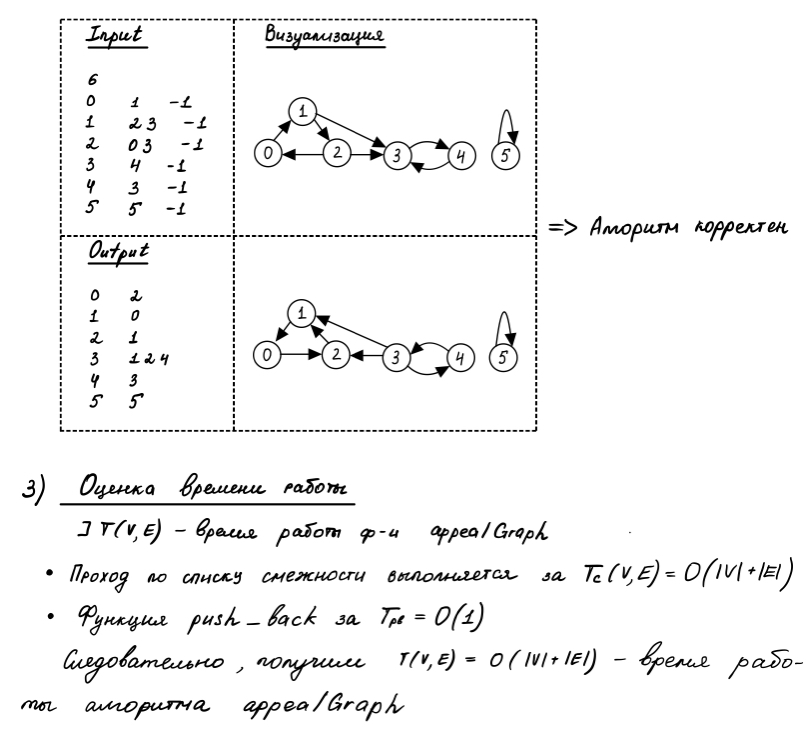
graph.metaGraph().printGraph();

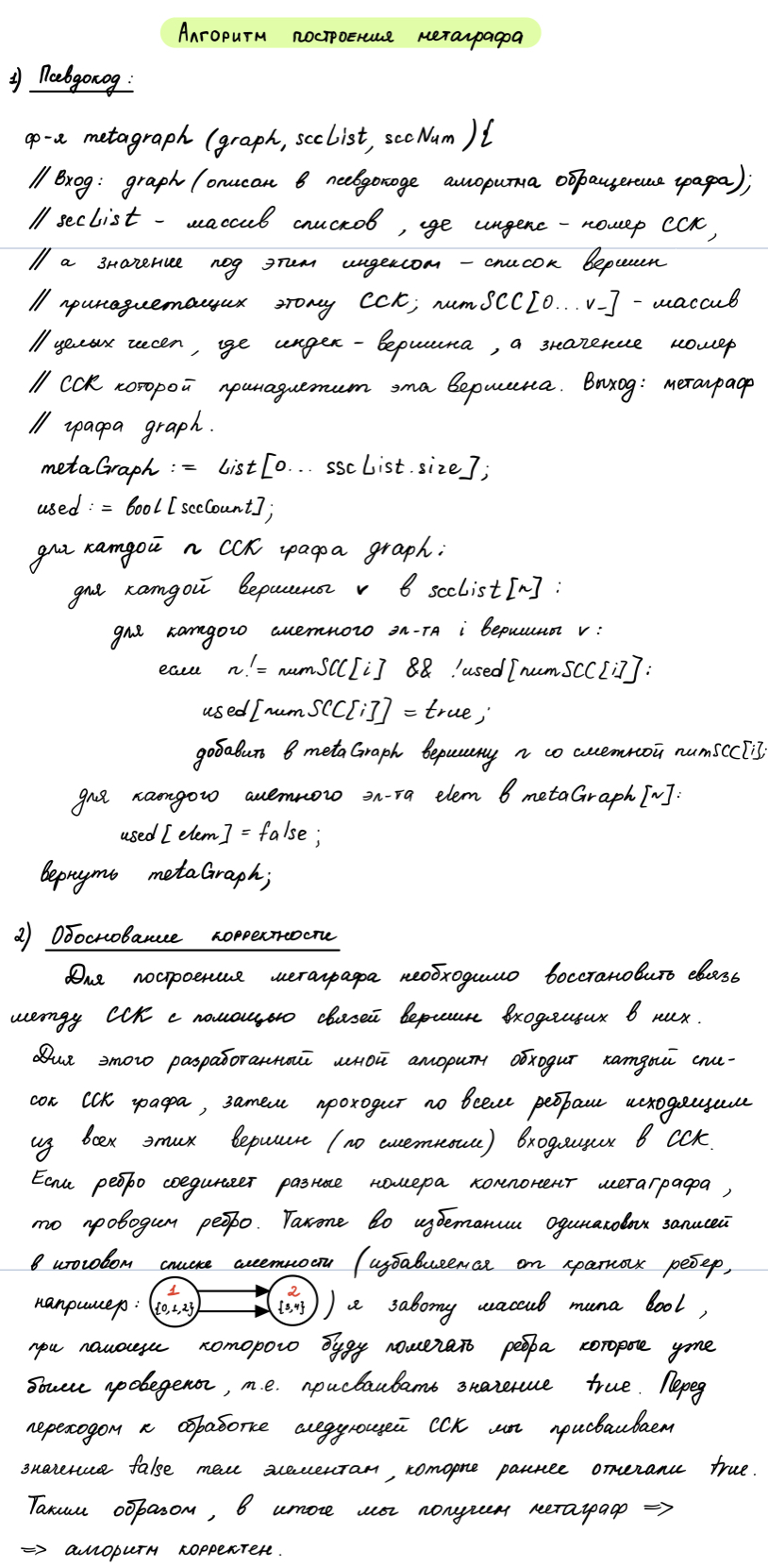
**return** 0;

}

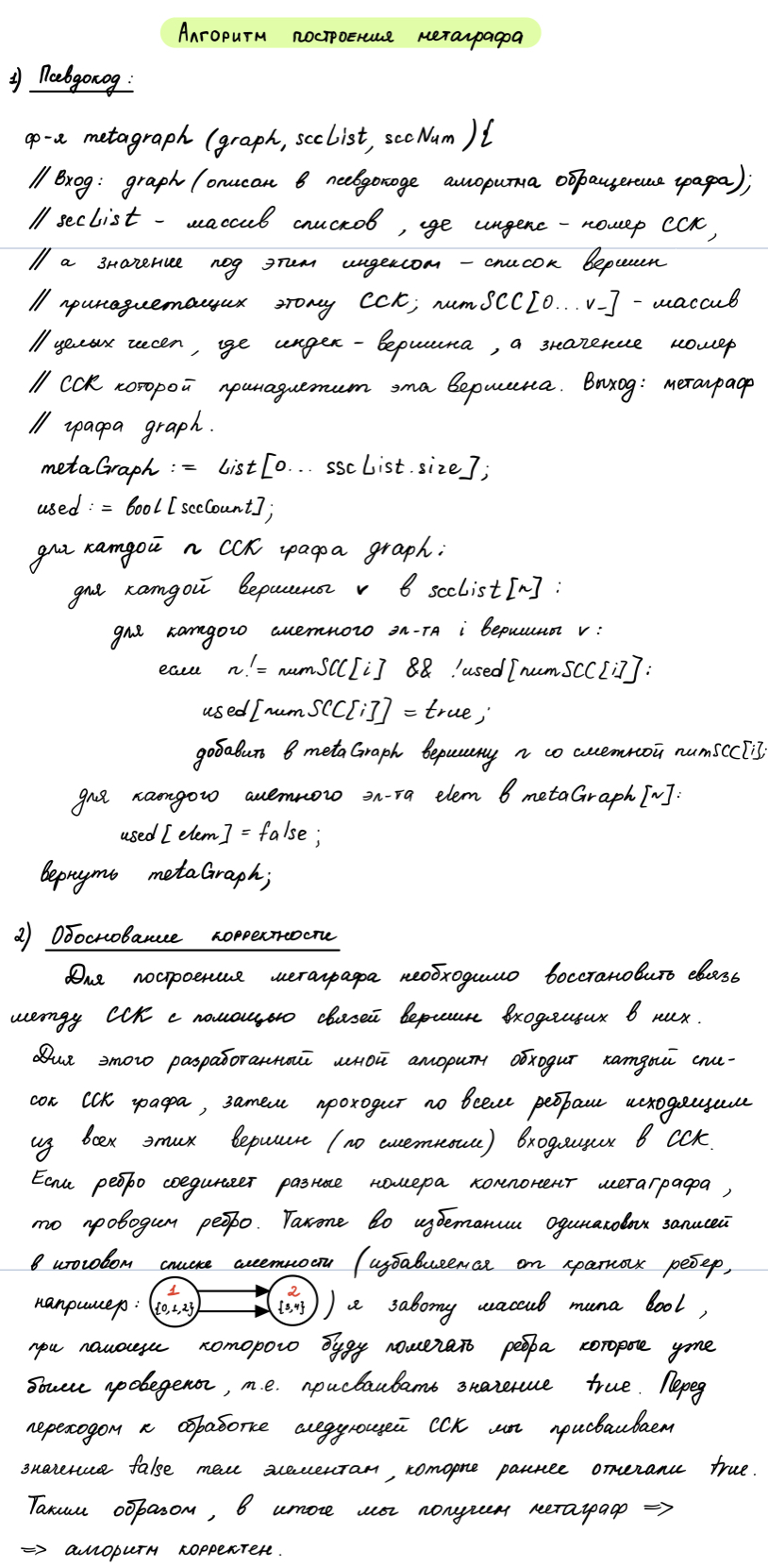
****

****

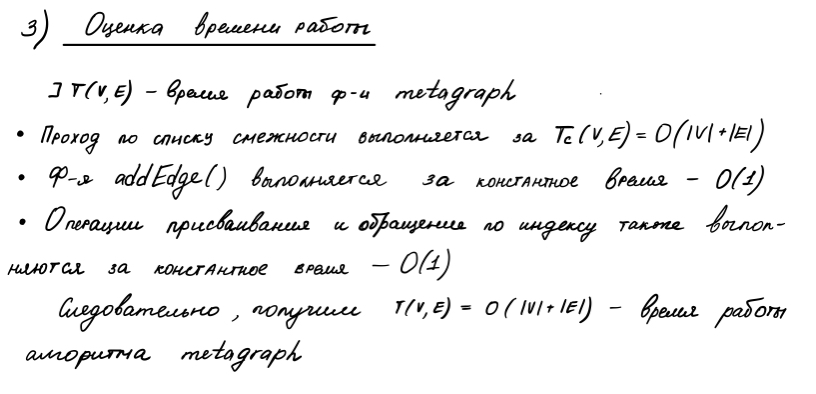
****

****



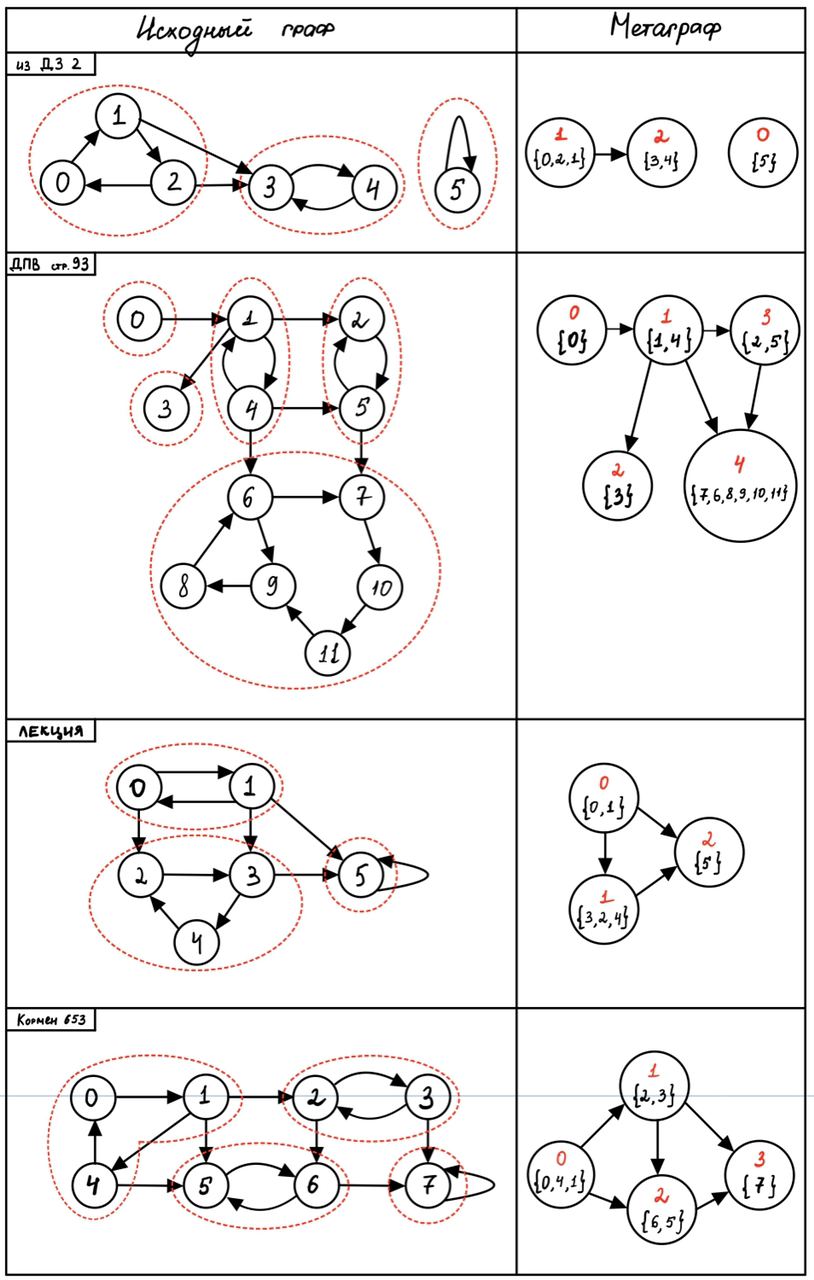


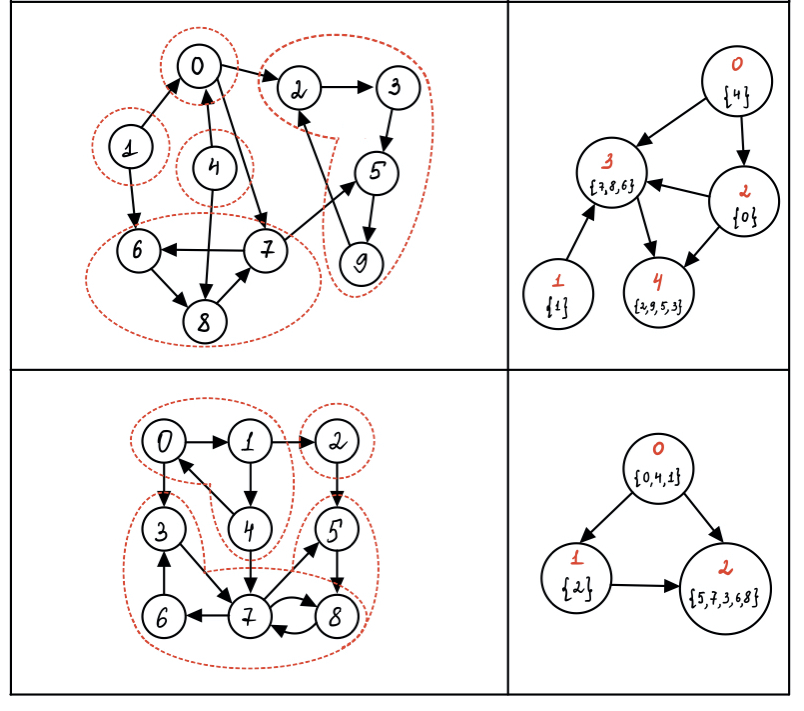






Визуализация построения метографа





|  |  |
| --- | --- |
| Input | Output |
| 6  0 1 -1  1 2 3 -1  2 0 3 -1  3 4 -1  4 3 -1  5 5 -1 | SCC 0: 5  SCC 1: 0 2 1  SCC 2: 3 4  Meta-graph:  0  1 2  2 |
| 12  0 1 -1  1 2 3 4 -1  2 5 -1  3 -1  4 1 5 6 -1  5 2 7 -1  6 7 9 -1  7 10 -1  8 6 -1  9 8 -1  10 11 -1  11 9 -1 | SCC 0: 0  SCC 1: 1 4  SCC 2: 3  SCC 3: 2 5  SCC 4: 7 6 8 9 11 10  Meta-graph:  0 1  1 3 2 4  2  3 4  4 |
| 6  0 1 2 -1  1 0 3 5 -1  2 3 -1  3 4 5 -1  4 2 -1  5 5 -1 | SCC 0: 0 1  SCC 1: 3 2 4  SCC 2: 5  Meta-graph:  0 1 2  1 2  2 |
| 8  0 1 -1  1 2 4 5 -1  2 3 6 -1  3 2 7 -1  4 0 5 -1  5 6 -1  6 5 7 -1  7 7 -1 | SCC 0: 0 4 1  SCC 1: 2 3  SCC 2: 6 5  SCC 3: 7  Meta-graph:  0 1 2  1 2 3  2 3  3 |
| 10  0 2 7 -1  1 6 -1  2 3 -1  3 5 -1  4 0 8 -1  5 9 -1  6 8 -1  7 5 6 -1  8 7 -1  9 2 -1 | SCC 0: 4  SCC 1: 1  SCC 2: 0  SCC 3: 7 8 6  SCC 4: 2 9 5 3  Meta-graph:  0 2 3  1 3  2 4 3  3 4  4 |
| 9  0 1 3 -1  1 2 4 -1  2 5 -1  3 7 -1  4 7 0 -1  5 8 -1  6 3 -1  7 5 6 8 -1  8 7 -1 | SCC 0: 0 4 1  SCC 1: 2  SCC 2: 5 7 3 6 8  Meta-graph:  0 2 1  1 2  2 |

Тесты