Министерство образования Республики Беларусь
Учреждение образования
“Брестский Государственный технический университет”
Кафедра ИИТ

Лабораторная работа №1

По дисциплине “Математические основы интеллектуальных систем”

Тема: “Нахождение компонент связности неориентированного графа”

Выполнил:
Студент 2 курса
Группы ИИ-21
Кирилович А. А.
Проверил:
Козинский А. А.

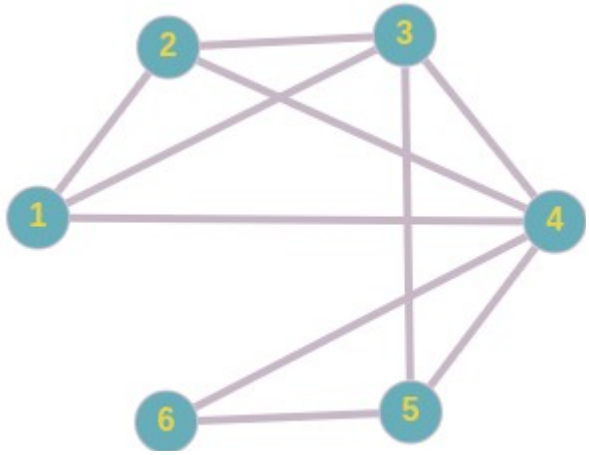
Брест 2022

Цель: научиться находить компонент связности неориентированного графа.

Ход работы:

Задание 1

Построить матрицу смежности и инцидентности для заданного графа. Изобразить граф.



Файл connections.txt
(1,2),(1,3),(1,4),(2,3),(2,4),(3,4),(3,5),(4,5),(4,6),(5,6)

```
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <fstream>
#include <algorithm>

static std::vector<int> split(const std::string &s, char
delim) {
	std::vector<int> elems;
	std::stringstream ss(s);
	std::string item;
	while (getline(ss, item, delim)) {
		elems.push_back(atoi(item.c_str()));
	}
	return elems;
	}

void removeAllOccurrence(std::string &s, char symbolToDelete) {
	size_t pos = 0;
	size_t i = 0;
	while ((i = s.find(symbolToDelete, pos)) != std::string::npos) {
		s.erase(i, 1);
		pos += i;
	}
}

int main() {
	std::ifstream file;
	file.open("connections.txt", std::ios_base::in);
	std::string str;
	getline(file, str);
	file.close();
	str.erase(std::remove(str.begin(), str.end(), ')'), str.end());
	str.erase(std::remove(str.begin(), str.end(), '('), str.end());
	std::vector<int> nodes;
	nodes = split(str, ',');

	int n = nodes.size() / 2, m = 2;
	int max_node = *max_element(nodes.begin(), nodes.end());
	std::vector<std::vector<int>> connections(n, std::vector<int>(m));

	for (int i = 0, j = 0; i < n, j < 2 * n; i++, j+=2) {
		connections[i][0] = nodes[j];
	}
	for (int i = 0, j = 1; i < n, j < 2 * n; i++, j+=2) {
	connections[i][1] = nodes[j];
	}
	// adjance matrix
	int adjacencyMatrix[max_node][max_node];
	for (int i = 0; i < max_node; i++) {
		for (int j = 0; j < max_node; j++) {
			adjacencyMatrix[i][j] = 0;
		}
	}
	for (int i = 0; i < n; i++) {
		adjacencyMatrix[connections[i][0] - 1][connections[i][1] - 1] = 1;
		adjacencyMatrix[connections[i][1] - 1][connections[i][0] - 1] = 1;
	}
	for (int i = 0; i < max_node; i++) {
		for (int j = 0; j < max_node; j++) {
			std::cout << adjacencyMatrix[i][j];
		}
```

```
011100
101100
110110
111011
001101
000110

1110000000
1001100000
0101011000
0010110110
0000001101
0000000011
```

```
                std::cout << std::endl;
        }
        std::cout << std::endl;
        // incidence matrix
        int incidenceMatrix[max_node][n];
        for (int i = 0; i < max_node; i++) {
                for (int j = 0; j < n; j++) {
                        incidenceMatrix[i][j] = 0;
                }
        }
        for (int i = 0; i < n; i++) {
                incidenceMatrix[connections[i][0] - 1][i] = 1;
                incidenceMatrix[connections[i][1] - 1][i] = 1;
        }
        for (int i = 0; i < max_node; i++)   {
                for (int j = 0; j < n; j++) {
                        std::cout << incidenceMatrix[i][j];
                }
                std::cout << std::endl;
        }
}
```

Задание 2

Используя поиск в ширину и поиск в глубину написать программу, определяющую число компонент связнозности графа. Методы представляются в виде функций.

```
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <fstream>
#include <algorithm>
#include <queue>

void dfs(std::vector<std::vector<int>> &adjacencyMatrix, int start, int max_node, std::vector<int> &visited) {
        visited[start] = 1;
        for (int i = 0; i < max_node; i++) {
                if (adjacencyMatrix[start][i] == 1 && visited[i] == 0) {
                        dfs(adjacencyMatrix, i, max_node, visited);
                }
        }
}

void bfs (std::vector<std::vector<int>> &adjacencyMatrix, std::queue <int> q, int n, std::vector<int> &visited)
{
        while (!q.empty()) {
                int start = q.front();
                visited[start] = 1;
                q.pop();
                for (int i = 0; i < n; i++)
                {
                        if (adjacencyMatrix[start][i] == 1 && visited[i] == 0) {
                                q.push(i);
                        }
                }
        }
}

static std::vector<int> split(const std::string &s, char delim) {
        std::vector<int> elems;
        std::stringstream ss(s);
        std::string item;
        while (getline(ss, item, delim)) {
                elems.push_back(atoi(item.c_str()));
        }
        return elems;
}

void removeAllOccurrence(std::string &s, char symbolToDelete) {
        size_t pos = 0;
        size_t i = 0;
        while ((i = s.find(symbolToDelete, pos)) != std::string::npos) {
                s.erase(i, 1);
                pos += i;
        }
}
```

```
int main() {
	std::ifstream file;
	file.open("connections.txt", std::ios_base::in);
	std::string str;
	getline(file, str);
	file.close();
	str.erase(std::remove(str.begin(), str.end(), ')'), str.end());
	str.erase(std::remove(str.begin(), str.end(), '('), str.end());
	std::vector<int> nodes;
	nodes = split(str, ',');

	Int n = nodes.size() / 2, m = 2;
	int max_node = *max_element(nodes.begin(), nodes.end());
	std::vector<std::vector<int>> connections(n, std::vector<int>(m));

	for (int i = 0, j = 0; i < n, j < 2 * n; i++, j+=2) {
		connections[i][0] = nodes[j];
	}
	for (int i = 0, j = 1; i < n, j < 2 * n; i++, j+=2) {
		connections[i][1] = nodes[j];
	}

	std::vector<std::vector<int>> adjacencyMatrix(max_node,    std::vector<int>(max_node));
	for (int i = 0; i < max_node; i++) {
		for (int j = 0; j < max_node; j++) {
			adjacencyMatrix[i][j] = 0;
		}
	}
	for (int i = 0; i < n; i++) {
		adjacencyMatrix[connections[i][0] - 1][connections[i][1] - 1] = 1;
		adjacencyMatrix[connections[i][1] - 1][connections[i][0] - 1] = 1;
	}

	std::vector<int> visited(max_node);

	//searching count by DFS
	for (int i = 0; i < max_node; i++) {
		visited[i] = 0;
	}
	dfs(adjacencyMatrix, 0, n, visited);
	int count = 1;
	for (int i = 0; i < max_node; i++) {
		if (visited[i] == 0) {
			dfs(adjacencyMatrix, i, n, visited);
			count++;
		}
	}
	std::cout << "DFS: count = " << count << std::endl;

	//searching count by BFS
	for (int i = 0; i < max_node; i++) {
		visited[i] = 0;
	}
	std::queue <int> q;
	q.push(0);
	bfs(adjacencyMatrix, q, n, visited);
	count = 1;
	for (int i = 0; i < max_node; i++) {
		if (visited[i] == 0) {
			q.push(i);
			bfs(adjacencyMatrix, q, n, visited);
			count++;
		}
	}
	std::cout << "BFS: count = " << count << std::endl;
}
```

```
DFS: count = 1
BFS: count = 1
```

Вывод: в ходе лабораторной работы я научился работать с графами, научился находить компонент связности.