1. **Цель работы**

   Целью работы является изучение графов и получение практических навыков их использования.

2. **Задание**

Согласно варианту №18:

Составить программу для нахождения произвольного разбиения N студентов на M команд, численность которых отличается не более чем в 2 раза, если известно, что в любой команде должны быть студенты, обязательно не знакомые друг с другом. Круг знакомств задается графом, где вершина – это студент, а ребро отображает его знакомство с другим студентом. Решить задачи:

1) Определить наименьшее количество команд, на которое можно разбить множество студентов;

2) Проверить возможность разбиения множества студентов на заданное количество команд.

3. **Листинг программы**

```
// Вариант 18
//

#include <iostream>
#include <math.h>

using namespace std;

struct Links
{
   int node_1;
   int node_2;
};

struct Node
{
   int node = 0;
   bool taken = false;
};
```

```
int get_num_int() // Запрос и проверка числа на корректность
{
   int x;

   cin >> x;
   while (cin.fail() || (cin.peek() != '\n')) // Проверка на корректность
   {
      cin.clear(); // Очищение флага ошибки
      cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Очистка буфера запроса
      cout << "Повторите ввод: ";
      cin >> x;
   }

   return x;
}

void show_grap(Links* array, int n)
{
   for (int i = 0; i < n; i++)
   {
      cout << "Ребро " << array[i].node_1 << " - " << array[i].node_2 << endl;
   }
}

void show_nodes(Node* array, int n)
{
   for (int i = 0; i < n; i++)
   {
      cout << "Вершина " << array[i].node << endl;
   }
}

bool find_node(Node* array, int n, int value)
{
```

```
    for (int i = 0; i < n; i++)
    {
        if (array[i].node == value)
            return true;
    }
    return false;
}

void add_node(Node*& array, int& n, int value)
{
    Node* arr = new Node[n + 1];

    for (int i = 0; i < n; i++)
    {
        arr[i].node = array[i].node;
    }

    arr[n].node = value;

    delete[] array;
    array = arr;
    n++;

}

void add_elem(Links*& array, Node*& array_2, int& n, int& n_2, int value_1, int
value_2)
{
    Links* arr = new Links[n + 1];

    for (int i = 0; i < n; i++)
    {
        arr[i].node_1 = array[i].node_1;
        arr[i].node_2 = array[i].node_2;
    }
```

```
    arr[n].node_1 = value_1;
    arr[n].node_2 = value_2;

    delete[] array;
    array = arr;
    n++;

    if (not(find_node(array_2, n, value_1)))
    {
        add_node(array_2, n_2, value_1);
    }
    if (not(find_node(array_2, n, value_2)))
    {
        add_node(array_2, n_2, value_2);
    }

}

bool find_rebr(Links* array, int n, int value_1, int value_2)
{
    for (int i = 0; i < n; i++)
    {
        if ((array[i].node_1 == value_1 && array[i].node_2 == value_2) || (array[i].node_1
== value_2 && array[i].node_2 == value_1))
            return true;
    }
    return false;
}

int find_maxs(Node* array, int n)
{
    int maxs = 1;
    for (int i = 0; i < n; i++)
    {
```

```
        maxs = max(maxs, array[i].node);
    }
    return maxs;
}

void mark_untaken(Node*& array, int n)
{
    for (int i = 0; i < n; i++)
    {
        array[i].taken = false;
    }
}



void sorting_nodes(Node*& array_2, int n_2)
{
    for (int i = 0; i < n_2; i++) {
        for (int j = 0; j < n_2-1; j++) {
            if (array_2[j].node > array_2[j + 1].node) {
                int b = array_2[j].node;
                array_2[j].node = array_2[j + 1].node;
                array_2[j + 1].node = b;
            }
        }
    }
}

bool isfree(Node* array, int n)
{
    for (int i = 0; i < n; i++)
    {
        if (array[i].taken == false)
            return true;
    }
    return false;
```

```
}

void add_pair(Links*& array, int& n, int value_1, int value_2)
{
   Links* arr = new Links[n + 1];

   for (int i = 0; i < n; i++)
   {
      arr[i].node_1 = array[i].node_1;
      arr[i].node_2 = array[i].node_2;
   }

   arr[n].node_1 = value_1;
   arr[n].node_2 = value_2;

   delete[] array;
   array = arr;
   n++;
}

void add_team_member(Node*& array, int& n, int value)
{
   Node* arr = new Node[n + 1];

   for (int i = 0; i < n; i++)
   {
      arr[i].node = array[i].node;
   }

   arr[n].node = value;
   delete[] array;
   array = arr;
   n++;
}
```

```
Links find_commands(Links* array, Node*& array_2, int n, int n_2, bool mins, int&
command_need)
{


   int maxs = find_maxs(array_2, n_2);

   int** matrix_inc{ new int* [maxs] {} };

   for (int i = 0; i < maxs; i++)
   {
      matrix_inc[i] = new int[maxs] {};
   }

   for (int i = 0; i < n; i++)
   {
      matrix_inc[array[i].node_1 - 1][array[i].node_2 - 1] = 1;
      matrix_inc[array[i].node_2 - 1][array[i].node_1 - 1] = 1;
   }

   for (int i = 0; i < n_2; i++)
   {
      matrix_inc[i][i] = 0;
   }

   /*
   for (int i = 0; i < maxs; i++)
   {
      for (int j = 0; j < maxs; j++)
      {
         cout << matrix_inc[i][j] << '\t';
      }
      cout << endl;
   }
   */
```

```
int can_commands = 0;
int n_3 = 0;
Links* pairs = new Links[n_3];
for (int i = 0; i < maxs; i++)
{
   if (array_2[i].taken) continue;
   for (int j = 0; j < maxs; j++)
   {
      if (i == j) continue;
      if (array_2[j].taken) continue;
      if (matrix_inc[i][j] == 0)
      {
         add_pair(pairs, n_3, i + 1, j + 1);
         array_2[i].taken = true;
         array_2[j].taken = true;
         can_commands += 1;
         break;
      }
   }
}

mark_untaken(array_2, n_2);

for (int i = 0; i < maxs; i++)
{
   delete[] matrix_inc[i];
}
delete[] matrix_inc;

Links commands;
commands.node_2 = can_commands;

if (mins)
{
   commands.node_1 = (min(can_commands, 2));
```

```
    delete[] pairs;
    return commands;
}
else if (not(mins) && (command_need > can_commands))
{
    commands.node_1 = -1;
    delete[] pairs;
    return commands;
}
else if (not(mins) && (command_need <= can_commands))
{
    commands.node_1 = 1;
    Node** comms{ new Node* [command_need]{} };
    int* sizes = new int[command_need]{};
    for (int i = 0; i < command_need; i++)
    {
        comms[i] = new Node[0]{};
    }


    int ost = maxs;
    for (int i = 0; i < command_need; i++)
    {
        add_team_member(comms[i], sizes[i], pairs[i].node_1);
        add_team_member(comms[i], sizes[i], pairs[i].node_2);
        array_2[pairs[i].node_1-1].taken = true;
        array_2[pairs[i].node_2-1].taken = true;
        ost -= 2;
    }

    int team = 0;
    for (int i = command_need; i < n_3; i++)
    {
        add_team_member(comms[team], sizes[team], pairs[i].node_1);
        team++;
```

```
    if (team == command_need)
        team = 0;
    add_team_member(comms[team], sizes[team], pairs[i].node_2);
    array_2[pairs[i].node_1 - 1].taken = true;
    array_2[pairs[i].node_2 - 1].taken = true;
    team++;
    if (team == command_need)
        team = 0;
    ost -= 2;
}

cout << ost << endl;

while (ost > 0)
{
    for (int i = 0; i < maxs; i++)
    {
        if (array_2[i].taken) continue;
        add_team_member(comms[team], sizes[team], array_2[i].node);
        array_2[i].taken = true;
        ost--;
        team++;
        if (team == command_need)
            team = 0;
    }
}

mark_untaken(array_2, n_2);

cout << "Команды: " << endl;

for (int i = 0; i < command_need; i++)
{
    cout << i + 1 << "." << '\t';
    for (int j = 0; j < sizes[i]; j++)
```

```
            {
                cout << comms[i][j].node << '\t';
            }
            cout << endl;
        }

        for (int i = 0; i < command_need; i++)
        {
            delete[] comms[i];
        }
        delete[] comms;
        delete[] pairs;

        return commands;
    }

}

int main()
{
    setlocale(LC_ALL, "RUS");

    cout << "Добро пожаловать в программу разбиения студентов на команды по знакомству." << endl;

    int n = 0;
    int n_2 = 0;
    int comms;
    Links* grap = new Links[n];
    Node* nodes = new Node[n_2];

    add_elem(grap, nodes, n, n_2, 1, 2);
    add_elem(grap, nodes, n, n_2, 1, 3);
    add_elem(grap, nodes, n, n_2, 2, 3);
    add_elem(grap, nodes, n, n_2, 2, 4);
```

```
add_elem(grap, nodes, n, n_2, 2, 6);
add_elem(grap, nodes, n, n_2, 2, 7);
add_elem(grap, nodes, n, n_2, 3, 5);
add_elem(grap, nodes, n, n_2, 4, 5);
add_elem(grap, nodes, n, n_2, 4, 6);
sorting_nodes(nodes, n_2);

while (true)
{

   cout << "1. Вывод ребёр" << endl;
   cout << "2. Вывод вершин" << endl;
   cout << "3. Разделение на M команд" << endl;
   cout << "4. Узнать наименьшее разбиение на команды" << endl;
   cout << "5. Выход" << endl;

   cout << "Выберите дальнейшее действие: ";
   int choice = get_num_int();

   if (choice == 1)
   {
      system("cls");

      if (n > 0)
      {
         cout << "Рёбра в графе: " << endl;
         show_grap(grap, n);
      }
      else
      {
         cout << "Граф пуст" << endl;
      }
   }
   else if (choice == 2)
   {
```

```
            system("cls");
            if (n_2 > 0)
            {
                cout << "Вершины в графе: " << endl;
                show_nodes(nodes, n_2);
            }
            else
            {
                cout << "Вершин нет" << endl;
            }
        }
        else if (choice == 3)
        {
            system("cls");
            cout << "Введите количетсво команд (>1): " << endl;
            comms = get_num_int();
            while (comms <= 1)
            {
                cout << "Введите корректное значение (>1): " << endl;
                comms = get_num_int();
            }

            Links commands = find_commands(grap, nodes, n, n_2, false, comms);

            if (commands.node_1 == -1)
            {
                cout << "Студентов невозможно разделить на команды" << endl;
                cout << "Максимально возможное количество команд: " <<
commands.node_2 << endl;
            }
            else if (commands.node_1 == 1)
            {
                cout << "Студенты разделены на " << comms << " команд(ы)" << endl;
            }
```

```
        }
        else if (choice == 4)
        {
            system("cls");
            int comms;
            Links commands = find_commands(grap, nodes, n, n_2, true, comms);
            if (commands.node_1 <= 1)
            {
                cout << "Студентов невозможно разделить на команды в количестве 2 и
более" << endl;
            }
            else
            {
                cout << "Минимальное количество команд: " << commands.node_1 <<
endl;
            }

        }
        else if (choice == 5)
        {
            system("cls");
            exit(0);
        }
        else
        {
            system("cls");
        }
    }
}
```

4. **Контрольный пример**

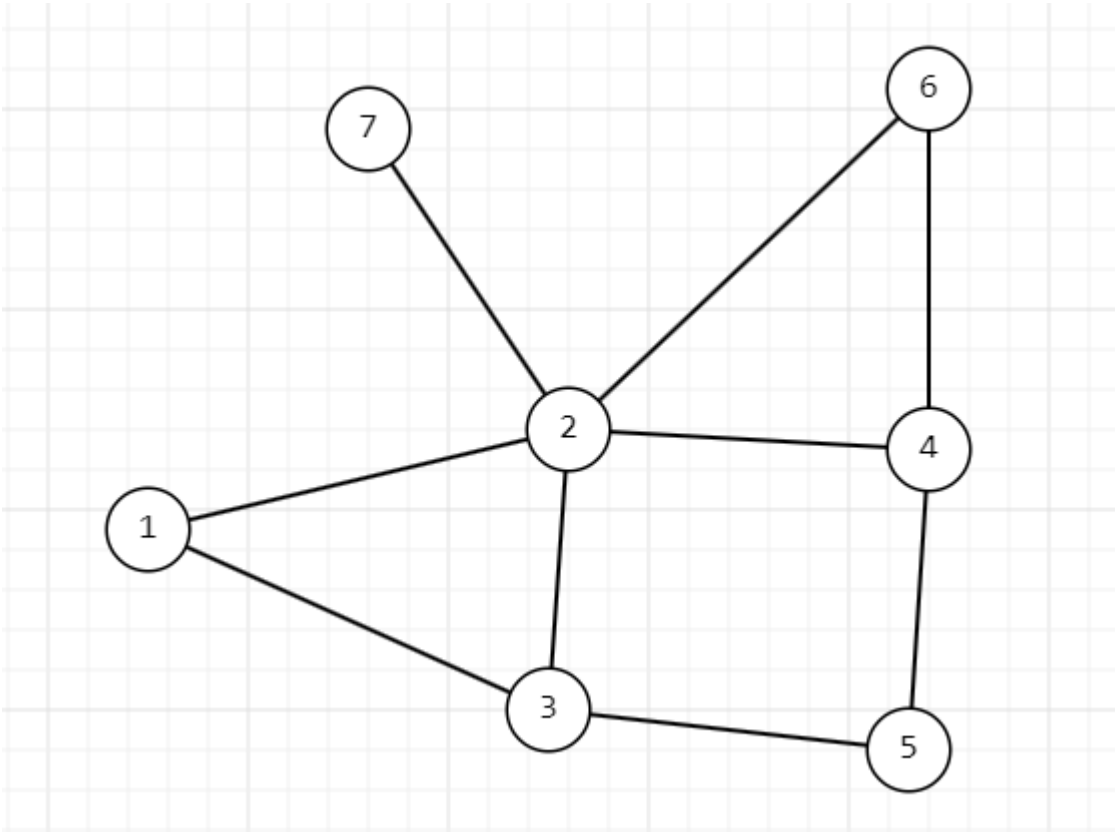

Рисунок 1 – Граф

Дан граф на рисунке 1. Алгоритм ищет пары, незнакомых друг с другом студентов. Количество пар определяет максимальное количество команд. Если нужно разбить на М команд, то первые М пар составляют М команд и потом поочерёдно записывается студент, не состоящий ни в одной из команд.

По рисунку:

Проанализировав все пары, станет ясно, что максимальное число пар 3.

Например(могут быть и другие):

- 1-4
- 2-5
- 3-6

При разбитии студентов на 2 команды применяется следующий алгоритм:

1-4 и 2-5 становяться в первую и вторую команду соответственно.

Затем поочерёдно записывается каждый студент, не состоящий в команде.

Как итог команды:

- 1 4 3 7
- 2 5 6

Таким образом в любой команде обязательно найдутся студенты, которые не знакомы.

5. Выводы

В процессе лабораторной работы были изучены графы, а так их алгоритмы и применены на практике.