Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ОТЧЁТ

по учебной (ознакомительной) практике

по специальности 1-40 02 01 «Вычислительные машины, системы и сети»

БГУИР 1-40 02 04 01 ПЗ

Студент И.Р. Лагодич

Руководитель практики:

от университета:

УО БГУИР, кафедра ЭВМ И.Г. Скиба

МИНСК 2023

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ............................................................................................................3

1. ПОСТАНОВКА ЗАДАЧИ……………………………………….…………...4

2. ОПИСАНИЕ ОСНОВНОГО АЛГОРИТМА………………………………...5

3. БЛОК-СХЕМА АЛГОРИТМА….....................................................................6

4. РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ……………………………...7

5. ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ……………………………..10

ЗАКЛЮЧЕНИЕ…………………………………………………………………11

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ…………………………..12

# **ВВЕДЕНИЕ**

Подмножество S множества вершин V графа G называется независимым множеством графа G, если выполняется условие S ∩ N(S) = ∅, т. е. любые две вершины из S не смежны. Если S – независимое множество, то любое его подмножество также является независимым. Поэтому представляет интерес задача нахождения в графе максимальных независимых множеств, т. е. таких, которые не являются собственными подмножествами никаких других независимых множеств.

Независимое множество, имеющее наибольшую мощность среди всех независимых множеств графа G, называют наибольшим независимым множеством, а его мощность называют числом независимости графа G и обозначают символом α(G).

# **ПОСТАНОВКА ЗАДАЧИ**

Необходимо разработать программу для нахождения максимального независимого множества в графе

Примером задачи нахождения наибольшего независимого множества является другая задача о ферзях, в которой надо расставить на шахматной доске наибольшее число ферзей так, чтобы ни один из них не находился под ударом другого. Наибольшее независимое множество графа, представляющего шахматную доску, как определено выше, покажет, на какие клетки надо поставить ферзей. Наибольшее число ферзей, расставленных при указанном условии, которое в данном случае равно восьми, есть число независимости данного графа.

Пусть Si – одно из независимых множеств графа G, формируемых на i-м этапе. За начальное значение множества Si принимается множество, состоящее из единственной вершины vi. Множество Si расширяется за счет поочередного включения в него элементов vj ∈ V, удовлетворяющих следующим условиям: i < j ≤ n и vj ∈ ∩(v ∈ Si) {V\ N(v)}

Каждый раз при соблюдении этих условий выбирается vj с минимальным j. Это расширение множества Si продолжается до тех пор, пока множество ∩(v ∈ Si) {V\ N(v)} не станет пустым.

# **2 ОПИСАНИЕ ОСНОВНОГО АЛГОРИТМА**

Для поиска максимального независимого множества берется одна случайная из n вершин, заданных пользователем, которая будет первой занесена в массив максимального независимого множества. Все вершины и их отношения записаны в матрице смежности. Далее будем искать не смежные вершины по отношению к вершинам, занесенным в массив с максимальным независимым множеством, и делая проверку на встречу уже записанных вершин в массив. При нахождении новой вершины она обязательна должна быть не смежна с уже находящимися вершинами массива максимального независимого множества. Поиск будет выполняться до тех пор, пока не будут проверены каждые вершины. На рисунке 2.2 изображен граф, где выделены не смежные вершины, множество которых {3, 4, 5} является одним из максимальных независимых множеств для данного графа.

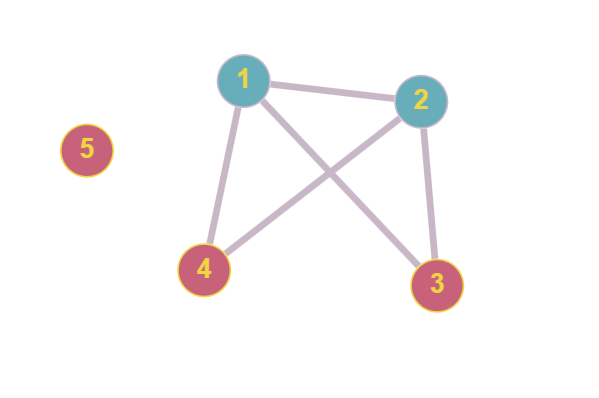


Рисунок 2.2 – Граф с 5 вершинами

# **3 БЛОК-СХЕМА АЛГОРИТМА**

На рисунке 3.1 приведена блок-схема алгоритма программы для поиска максимального независимого множества.

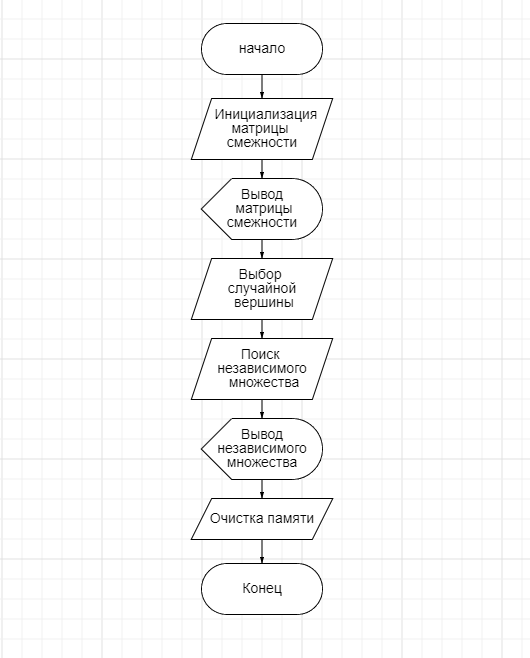


Рисунок 3.1 – Основная функция

**4 РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int\*\* memory\_alloc(int size)

{

int memory\_size = size;

int\*\* temp = (int\*\*)calloc(memory\_size, sizeof(int\*));

if(!temp)

{

printf("Error: can't allocate memory");

exit(1);

}

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

{

temp[i] = (int\*)calloc(size, sizeof(int));

if (!temp[i])

{

printf("Error: can't allocate memory");

exit(1);

}

}

return temp;

}

void matrix\_filling(int\*\* array, int size)

{

printf("Fill the adjacency matrix: \n");

for (int row = 0; row < size; row++)

{

for (int col = row; col < size; col++)

{

if (col != row)

{

printf("Enter the connection of %d verticy with %d verticy (1 - connected; 0 - not connected): ", row + 1, col + 1);

while ((scanf\_s("%d", &array[row][col]) != 1) || (array[row][col] < 0) || (array[row][col] > 1) || (getchar() != '\n'))

{

printf("Wrong input. Try again.\n");

}

array[col][row] = array[row][col];

}

else

array[col][row] = 0;

}

}

}

void matrix\_printing(int\*\* array, int size)

{

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

{

printf("%d ", i + 1);

}

printf("\n");

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

{

printf("--");

}

printf("\n");

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

{

for (int j = 0; j < size; j++)

{

printf("%d ", array[i][j]);

}

printf("|%d \n", i + 1);

}

}

void set\_printing(int\* set, int set\_size)

{

printf("\nMaximum indepent set is {");

for (int i = 0; i < set\_size; i++)

{

if (i != set\_size - 1)

printf("%d, ", set[i]);

else

printf("%d", set[i]);

}

printf("}");

}

void free\_memory(int\*\* array, int\* set, int vertices)

{

for (int i = 0; i < vertices; i++)

free(array[i]);

free(array);

free(set);

}

int main()

{

int\*\* matrix;

int\* set;

int set\_size = 1;

int vertices;

srand(time(NULL));

printf("Enter the number of vertices: ");

while ((scanf\_s("%d", &vertices) != 1) || (vertices < 1) || (getchar() != '\n'))

{

printf("Wrong input. Try again\n");

rewind(stdin);

}

matrix = memory\_alloc(vertices);

matrix\_filling(matrix, vertices);

matrix\_printing(matrix, vertices);

set = (int\*)calloc(vertices, sizeof(int));

if (!set)

{

printf("Error: can't allocate memory");

exit(1);

}

set[0] = rand() % vertices + 1;

for (int col = 0, found = 0; col < vertices; found = 0, col++)

{

for (int set\_pos = 0, repeat = 0; set\_pos < set\_size; repeat = 0, set\_pos++)

{

if (set[set\_pos] != col + 1)

{

if ((matrix[set[set\_pos] - 1][col] == 0) && (repeat == 0))

found = 1;

else

{

set\_pos = set\_size;

found = 0;

}

}

else

{

set\_pos = set\_size;

found = 0;

}

}

if (found == 1)

{

set[set\_size] = col + 1;

set\_size++;

}

}

set\_printing(set, set\_size);

free\_memory(matrix, set, vertices);

return 0;

}

**5 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ**

Скриншот работы программы представлен на рисунке 5.1

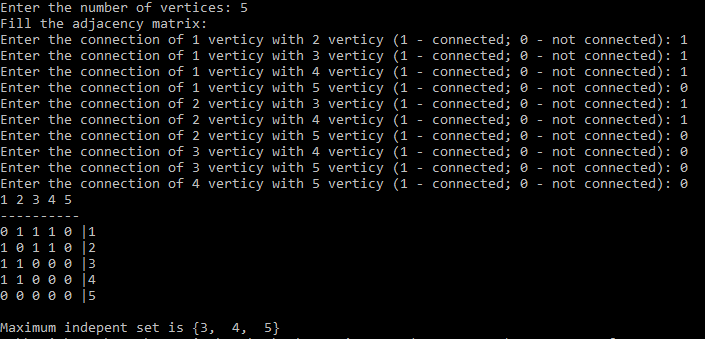


Рисунок 4.1 – Скриншот работы программы

# **ЗАКЛЮЧЕНИЕ**

Задача о независимом множестве относится к классу [NP-полных задач](https://ru.wikipedia.org/wiki/NP-%D0%BF%D0%BE%D0%BB%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0) в области [теории графов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2).

В информатике изучается несколько вычислительных задач, связанных с независимыми множествами:

* В задаче о наибольшем независимом множестве входом служит неориентированный граф, а выходом — наибольшее независимое множество в этом графе. Если существует несколько таких множеств, достаточно найти одно.
* В задаче о независимом множестве максимального веса входом служит неориентированный граф с весами, заданными для вершин, а выходом — независимое множество с максимальным общим весом. Задача о максимальном по включению независимом множестве является частным случаем этой задачи с весами, равными единице.
* В задаче перечисления максимальных по включению независимых множеств входом служит неориентированный граф, а выходом — список всех максимальных по включению независимых множеств. Задачу о наибольшем независимом множестве можно решать как подзадачу данной задачи, поскольку наибольшее независимое множество является максимальным по включению и должно попасть в этот список.
* В задаче о наличии независимого множества заданного размера входом служит неориентированный граф и число k, а выходом — Да/Нет: Да, если граф содержит независимое множество размера k, и Нет в противном случае.

Первые три задачи важны в практических приложениях, последняя же важна для теории [NP-полноты](https://ru.wikipedia.org/wiki/NP-%D0%BF%D0%BE%D0%BB%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0) для задач, связанных с независимыми множествами.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Брайан Керниган, Деннис Ритч. Язык программирования Си, 1978 г.. — 343 с.

[2] А. Б. Дайняк, А. А. Сапоженко, Независимые множества в графах, Дискрет. матем., том 28, выпуск 1, 44–77, 2016. — 35 с.

[3] Поттосин Ю.В. Методы дискретной математики в логическом проектировании цифровых устройств, 2017. — 202 с.

[4] Андерсон Д.А. Дискретная математика и комбинаторика. – М.: Издательский дом «Вильямс», 2003. – 960 с.

[5] Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 536 с

[6] Кристофидес Н. Теория графов. Алгоритмический подход. – М.: Мир, 1978. – 432 с.