Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Основы алгоритмизации и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к учебной практике (ознакомительной)  
на тему

ПОСЛЕДОВАТЕЛЬНАЯ РАСКРАСКА НЕОРИЕНТИРОВАННОГО ГРАФА, ПРЕДСТАВЛЕННОГО МОДИФИЦИРОВАННЫМИ

СПИСКАМИ СМЕЖНОСТИ

БГУИР КР 1-40 02 01 130 ПЗ

Студент

Руководитель

А.В.Шаповалова

А.М.Ковальчук

МИНСК 2022

# **ВВЕДЕНИЕ**

Язык С (С++) часто называют языком среднего уровня. Это означает, что С (С++) объединяет элементы языков высокого уровня с функциональностью Ассемблера.

Языки высокого уровня поддерживают концепцию типов данных. Тип данных определяет набор значений, которые переменная может хранить, и набор операций, которые могут выполняться над переменными. Наряду с тем, что в языке С (С++) представлены все основные типы данных, он не так жестко типизирован, как языки Паскаль или Ада. Язык С (С++) позволяет осуществлять большинство преобразований типов. Контроль за выполнением этих преобразований, а также проверка некоторых ошибок (например, выход за границы массива) возлагается на программиста. Реализованная в С (С++) возможность напрямую манипулировать битами, байтами, словами и указателями необходима для программирования на системном уровне. Язык С (С++) считается структурированным языком. Отличительной чертой структурированного языка является разделение кода и данных. Одним из способов решения этой проблемы является использование подпрограмм (функций), широко использующих локальные переменные. Необходимо отметить, что излишнее использование глобальных переменных может приводить к фатальным ошибкам. Как и ряд других структурированных языков, С (С++) поддерживает ряд операторов цикла, условных операторов и операторов ветвления.

Язык С (С++) содержит стандартные библиотеки, предоставляющие функции, выполняющие наиболее типичные задачи. Эти библиотеки легко могут быть подключены, а также дополнены. Язык С (С++) позволяет разбивать программу на части и выполнять их раздельную компиляцию. Откомпилированные таким образом файлы объединяются для создания полного объектного кода. Преимущество раздельной компиляции в том, что при изменении одного файла не требуется перекомпиляции всей программы.

**Тема работы:** реализация алгоритма последовательной раскраски неориентированного графа на языке програмирования С, используя принцип динамического программирования.

**Динамическое программирование в теории вычислительных систем**– способ решения сложных задач путём разбиения их на несколько более лёгких одинаковых подзадач, реккурентно связанных между собой.

**Граф** – совокупность точек, соединенных линиями. Точки называются вершинами, или узлами, а линии – ребрами.

**Неориентированный граф G** — это [упорядоченная пара](https://dic.academic.ru/dic.nsf/ruwiki/1157219) G=(V,E), для которой выполнены следующие условия:

1. V – это множество вершин, или узлов.
2. Е – это множество пар (в случае неориентированного графа – неупорядоченных) различных вершин, называемых ребрами.

В неориентированном графе по каждому из ребер можно осуществлять переход в обоих направлениях.

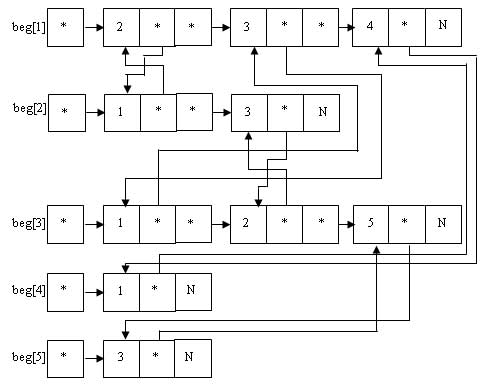
[**Раскраска**](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BA%D1%80%D0%B0%D1%81%D0%BA%D0%B0)**графа** — [теоретико-графовая](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) конструкция, при которой вершинам графа ставятся в соответствие метки, традиционно называемые «цветами». Когда говорят о раскраске графов, почти всегда подразумевают под этим раскраску их вершин, то есть присвоение цветовых меток вершинам графа так, чтобы любые две вершины, имеющие общее ребро, имели разные цвета.

**СУТЬ АЛГОРИТМА И РЕАЛИЗАЦИЯ ПРОГРАММЫ**

**Алгоритм последовательной раскраски –** динамический алгоритм присвоения цвета каждой из вершин графа, основываясь на следующем методе: произвольной вершине **v1** графа **G** присваивается цвет 1. Если вершины **v1, v2, ...,vi** раскрашены **k** цветами 1, 2, ..., **k**, **k <= i**, то новой произвольно взятой вершине **vi+k** присваивается минимальный цвет, не использованный при раскраске смежных вершин.

Рассмотрим неориентированный граф **G** с вершинами, пронумерованными от **1** до **num\_vertex**. Алгоритм может быть выполнен за время **O(V+E)** сравнений в графе, даже если в графе может быть до **Ʊ(n2)** рёбер, и каждая комбинация рёбер повторяется. Это достигается путём того, что в модифицированных списках смежности элемент списка **sp\_adj[vertex\_1]**, содержащий вершину **vertex\_1**, снабжен указателем на элемент списка **sp\_adj[vertex\_2]**, содержащий вершину **vertex\_2**, и наоборот, что схематически изображено на рисунке 1.

Рис. 1 – структура модифицированных списков смежности



Тогда удаляя «часть» ребра **(vertex\_1, vertex\_2)** из списка, можно за число шагов, ограниченное константой (**O(n)**), удалить и другую «часть» ребра, а именно **(vertex\_2, vertex\_1)**, не просматривая весь список смежности **sp\_adj[vertex\_2]**.

Формат ввода:

1. Ввод количества вершин и модифицированных списков смежности графа в консоли
2. Считывание количества вершин и модифицированных списков смежности графа из текстового файла

Формат вывода:

1. Вывод результата в виде списка вершин и их цветов в консоль
2. Вывод результаты в виде списка вершин и их цветов в текстовый файл

Структура звена графа:

struct zveno

{

int vertex; // вершина графа

zveno\* adj; // указатель на список смежности вершины vertex

zveno\* next\_adj; // указатель на следующую смежную вершину

};

**Алгоритм функции последовательной раскраски графа**

1. Установка значения вершины, равного единице и значения цвета, равного единице (i = 1, color = 1).
2. Присваивание i-й вершине в списке смежности цвета сolor.
3. Увеличение счетчика вершин на единицу (i++).
4. Цикл по количеству вершин (i), пока i не больше количества вершин графа.
5. Если содержимое i-го списка смежности пусто, переход к пункту 9.
6. Установка значения цвета сolor = 1.
7. Цикл по возвращаемому значению функции выявления возможности раскраски i-й вершины цветом сolor (Функция find\_color возвращает 1, если вершину можно раскрасить. Указатель ukaz\_zv возвращает указатель на вершину, смежную с i-й и раскрашенную цветом сolor. Если i-ю вершину можно раскрасить, то указатель есть NULL): пока вершину нельзя раскрасить в цвет сolor, увеличиваем его на единицу (сolor++).
8. Присваивание i-й вершине в списке смежности цвета сolor.
9. Увеличение счетчика вершин на единицу (i++).

**Алгоритм функции создания ребра графа**

(вершины vertex\_1 и vertex\_2 смежные, ukaz – NULL-указатель типа zveno\*\*)

1. Проверка смежности вершин функцией find (Функция возвращает 1, если вершины смежны. Указатель возвращает указатель на место на место одной вершины в списке смежности другой. Если вершины не смежны, или уже связаны, то указатель есть NULL).
2. Если возвращаемое значение не есть NULL (вершины смежны и еще не связаны между собой), вызов функции создания дуги (vertex\_1, vertex\_2) и дуги (vertex\_2, vertex\_1):
3. выделение памяти под 1 звено графа.
4. запись содержимого поля следующей смежной вершины vertex\_1 в поле следующей смежной вершины нового звена.
5. запись номера вершины vertex\_2 в поле номера вершины нового звена.
6. запись нового звена в поле следующей смежной вершины vertex\_1.
7. Запись списка смежности вершины vertex\_2 в поле смежной вершины vertex\_1.
8. Запись списка смежности вершины vertex\_1 в поле смежной вершины vertex\_2.

Блок-схема функции main представлена в приложении А.

Далее представлен код выполненной программы с комментариями по осуществлению алгоритма последовательной раскраски на языке программирования С.

Функция main():

#include"Header.h"

int main()

{

setlocale(LC\_ALL, "Rus");

char x;

FILE\* f;

do

{

system("cls");

int choose, num\_vertex;

zveno\*\* sp\_adj;

printf("Выберите метод ввода:\n1.С клавиатуры\n2.Считать из файла\n");

scanf\_s("%d", &choose);

if (choose == 1) num\_vertex = enter\_num\_vertex(); // ввод числа вершин графа

else num\_vertex = read\_num(); // считывание числа вершин графа

// из файла

sp\_adj = (zveno\*\*)calloc(num\_vertex + 1, sizeof(zveno\*)); // выделение памяти под массив

// указателей на списки смежности

spisok(num\_vertex, sp\_adj); // инициализация списков смежности

init\_graph(num\_vertex, sp\_adj); // инициализация графа

if (choose == 1) make\_graph(num\_vertex, sp\_adj); // построение графa с клавиатуры

else file\_make\_graph(num\_vertex, sp\_adj); // считывание модифицированных

// списков смежности графа из файла

print\_graph(num\_vertex, sp\_adj); // печать содержимого списков

color(num\_vertex, sp\_adj); // последовательная раскраска графа

printf("Раскраска графа:\n");

print\_color(num\_vertex, sp\_adj); // печать раскраски графа в консоль

print\_color\_to\_file(num\_vertex, sp\_adj); // печать раскраски графа в файл

delete\_all(num\_vertex, sp\_adj); // удаление ребер графа

delete\_vertex(1, num\_vertex, sp\_adj); // удаление вершин графа

printf("Если хотите повторить программу, введите 'a'\n");

x = \_getch();

} while (x == 'a');

return 0;

}

Функции для выполнения раскраски графа:

void color(int& num\_vertex, zveno\*\* sp\_adj) // функция последовательной раскраски графа

{

int i = 1, color = 1; // установка индекса первой вершины и

// начального значения цвета

zveno\* ukaz\_zv;

sp\_adj[i]->vertex = color; // присваивание i-ой вершине цвета сolor

i++; // увеличение счетчика числа вершин

while (i <= num\_vertex) // пока значение вершины не больше числа

{ // вершин графа

if (sp\_adj[i] != NULL)

{

color = 1; // установка начального значения цвета

while (!find\_color(i, color, &ukaz\_zv, sp\_adj)) // пока вершину нельзя раскрасить в цвет сolor

color++; // увеличение значения цвета

sp\_adj[i]->vertex = color; // присваивание i-ой вершине цвета сolor

i++; // переход к следующей вершине

}

else

i++; // переход к следующей вершине

}

}

int find\_color(int vertex\_1, int color, // функция выявления возможности раскраски

zveno\*\* ukaz\_zv, zveno\*\* sp\_adj) // вершины vertex\_1 цветом color

{ /\* функция возвращает 1, если вершину vertex\_1 можно раскрасить.

yказатель \*ukaz\_zv возвращает указатель на вершину, смежную с vertex\_1

и раскрашенную цветом color. Если вершину vertex\_1 можно раскрасить,

то \*ukaz\_zv есть NULL\*/

int i = 1;

while (!(find(vertex\_1, i, &(\*ukaz\_zv), sp\_adj)

&&sp\_adj[i]->vertex == color) && i != vertex\_1)

i++;

return (i == vertex\_1);

}

int find(int vertex\_1, int vertex\_2, // функция проверки смежности вершин vertex\_2 и vertex\_1

zveno\*\* ukaz\_zv, zveno\*\* sp\_adj)

/\* функция возвращает 1, если вершины смежны.указатель\* ukaz\_zv

возвращает указатель на место vertex\_2 в списке смежности vertex\_1.

если вершины не смежны или уже связаны, то ukaz есть NULL.\*/

{

zveno\* ukaz = sp\_adj[vertex\_1]->next\_adj; // установка указателя на смежную вершину одной

// из проверяемых вершин

while (ukaz != NULL && // пока указатель не пустой и вторая вершина

ukaz->vertex != vertex\_2) // не является искомой

ukaz = ukaz->next\_adj; // сдвиг указателя на следующую смежную вершину

\*ukaz\_zv = ukaz; // присвоение найденной информации указателю

return (ukaz != NULL);

}

Функции инициализации и построения графа:

int enter\_num\_vertex() // функция ввода числа вершин

{

printf("Введите число вершин графа: ");

int num\_vertex = 0;

int p = scanf\_s("%d", &num\_vertex);

while (!p || num\_vertex < 2) // проверка введенного значения

{

printf("Малое число вершин. Повторите попытку.\n");

p = scanf\_s("%d", &num\_vertex);

}

return num\_vertex;

}

void spisok(int& num\_vertex, zveno\*\* sp\_adj) // функция инициализации списков смежности

{

for (int i = 1; i <= num\_vertex; i++)

sp\_adj[i] = NULL;

}

void init\_graph(int& num\_vertex, zveno\*\* sp\_adj) // функция инициализации модифицированных

{ // списков смежности

for (int i = 1; i <= num\_vertex; i++)

add\_vertex(i, sp\_adj); // вызов функции создания вершины

}

void add\_vertex(int vertex\_1, zveno\*\* sp\_adj) // функция создания вершины vertex\_1

{

zveno\* Ukaz = (zveno\*)calloc(1, sizeof(zveno)); // выделение памяти под звено графа

Ukaz->next\_adj = NULL;

sp\_adj[vertex\_1] = Ukaz; // запись звена в массив

}

void make\_graph(int& num\_vertex, zveno\*\* sp\_adj) // функция построения графа

{

int adj\_vertex = 0;

for (int i = 1; i <= num\_vertex; i++)

{

printf("Введите вершины, смежные с %d-й вершиной (в конце введите 0): ", i);

int p = scanf\_s("%d", &adj\_vertex);

while (!p || adj\_vertex < 0 || // проверка введенного значения

adj\_vertex > num\_vertex || adj\_vertex == i)

{

printf("Введено некорректное значение. Повторите попытку.\n");

p = scanf\_s("%d", &adj\_vertex);

}

while (adj\_vertex != 0)

{

make(i, adj\_vertex, sp\_adj);

p = scanf\_s("%d", &adj\_vertex);

while (!p || adj\_vertex < 0 || // проверка введенного значения

adj\_vertex > num\_vertex || adj\_vertex == i)

{

printf("Введено некорректное значение. Повторите попытку.\n");

p = scanf\_s("%d", &adj\_vertex);

}

}

}

}

void make(int vertex\_1, int vertex\_2, zveno\*\* sp\_adj) // функция создания ребра {vertex\_1, vertex\_2}

{

zveno\* ukaz;

if (!find(vertex\_1, vertex\_2, &ukaz, sp\_adj)) // если вершины смежны и еще не связаны

{

add\_duga(vertex\_1, vertex\_2, sp\_adj); // создание дуг графа

add\_duga(vertex\_2, vertex\_1, sp\_adj);

sp\_adj[vertex\_1]->next\_adj->list\_adj = sp\_adj[vertex\_2]; // запись списка смежности vertex\_2 в

// предназначенное поле vertex\_1

sp\_adj[vertex\_2]->next\_adj->list\_adj = sp\_adj[vertex\_1]; // запись списка смежности vertex\_1 в

} // предназначенное поле vertex\_2

}

Функции для работы с файлами:

int read\_num() // функция считывания числа вершин из файла

{

int num\_vertex = 0;

FILE\* f = fopen("input.txt", "r"); // открытие файла в режиме чтения

fscanf(f, "%d", &num\_vertex); // считывание числа

fclose(f); // закрытие файла

return num\_vertex;

}

void print\_color\_to\_file(int& num\_vertex, zveno\*\* sp\_adj) // функция печати раскраски графа в файл

{

FILE\* g = fopen("output.txt", "w+"); // открытие файла

for (int i = 1; i <= num\_vertex; i++)

if (sp\_adj[i] != NULL) // если i-я вершина не пуста

fprintf(g, "Цвет %d-й вершины - %d\n", i, sp\_adj[i]->vertex); // печать вершины и ее цвета

fprintf(g, "\n");

fclose(g); // закрытие файла

}

void file\_make\_graph(int& num\_vertex, zveno\*\* sp\_adj) // функция считывания модифицированных { // списков смежности графа из файла

FILE\* f = fopen("input.txt", "r"); // открытие файла в режиме чтения

int adj\_vertex = 0, x;

fscanf(f, "%d", &x);

for (int i = 1; i <= num\_vertex; i++)

{

int p = fscanf(f, "%d", &adj\_vertex); // считывание вершины из файла

while (adj\_vertex != 0) // пока не будет считан 0

{

if (adj\_vertex <= num\_vertex && adj\_vertex > 0)

make(i, adj\_vertex, sp\_adj); // вызов функции создания ребра

p = fscanf(f, "%d", &adj\_vertex);

}

}

fclose(f); // закрытие файла

}

Функции очистки памяти:

void delete\_duga(int vertex\_1, // функция удаления дуги (vertex\_1, vertex\_2)

int vertex\_2, zveno\*\* sp\_adj)

{

zveno\* ukaz, \* ukaz\_1;

ukaz\_1 = sp\_adj[vertex\_1]; // установка указателя на 1 вершину

ukaz = sp\_adj[vertex\_1]->next\_adj; // установка указателя на следующую смежную вершину

while (ukaz != NULL && // пока указатель не пустой и

ukaz->vertex != vertex\_2) // не найдена нужная вершина

{

ukaz\_1 = ukaz;

ukaz = ukaz->next\_adj; // сдвиг указателя на следующую смежную вершину

}

if (ukaz != NULL) // если указатель не пустой

{

ukaz\_1->next\_adj = ukaz->next\_adj; // сдвига его на место предыдущей смежной вершины

free(ukaz); // освобождение памяти

}

}

void delete\_(int vertex\_1, int vertex\_2, // функция удаления ребра {vertex\_1, vertex\_2}

zveno\*\* sp\_adj)

{

delete\_duga(vertex\_1, vertex\_2, sp\_adj); // вызов функции удаления дуги (vertex\_1, vertex\_2)

delete\_duga(vertex\_2, vertex\_1, sp\_adj); // вызов функции удаления дуги (vertex\_2, vertex\_1)

}

void delete\_all(int& num\_vertex, // функция удаления всех ребер графа

zveno\*\* sp\_adj)

{

for (int i = 1; i < num\_vertex; i++)

{

int k = i + 1;

delete\_(i, k, sp\_adj); // вызов функции удаления ребра

}

}

void delete\_vertex(int vertex\_1, int& num\_vertex, // функция удаления вершин графа

zveno\*\* sp\_adj)

{

zveno\* ukaz, \*ukaz\_1;

for (int i = 1; i <= num\_vertex; i++)

delete\_duga(i, vertex\_1, sp\_adj);

ukaz = sp\_adj[vertex\_1]; // установка указателя на вершину

sp\_adj[vertex\_1] = NULL;

while (ukaz != NULL) // пока указатель не пустой

{

ukaz\_1 = ukaz->next\_adj; // сдвиг указателя

free (ukaz); // освобождение памяти

ukaz = ukaz\_1; // переход к следующей вершине

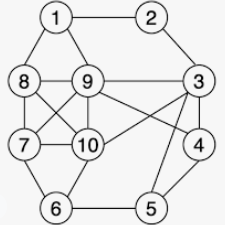
}

}

**ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАЧИ ПРОГРАММОЙ**

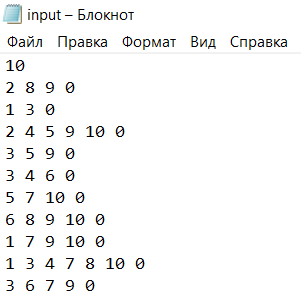
Дан неориентированный граф, заданный на рисунке 2. Найти раскраску его вершин.

Рис. 2 - граф



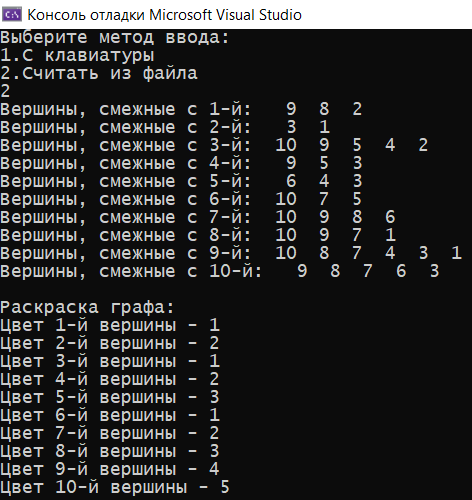
Используя данный граф, построим его списки смежности. Количество вершин и списки смежности и приведены на рисунке 3 (первая строка файла – количество вершин, последующие – списки смежных вершин).

Рис. 3 – пример файла input.txt



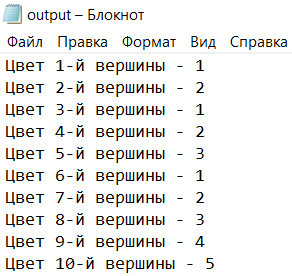
На рисунке 4 представлен пример работы программы по исходным данным.

Рис. 4 – пример работы программы



После завершения работы программы полученная раскраска сохраняется в текстовый файл под именем output.txt (рис. 5).

Рис. 5 – пример записанной раскраски в файл output.txt.



**ЗАКЛЮЧЕНИЕ**

В процессе выполнения учебной практики были выполнены первоначально заданные цели, а именно реализован алгоритм последовательной раскраски графа. Граф был представлен модифицированными списками смежности вершин, а не матрицей смежности, что позволило существенно сократить время работы программы (O(V+E) вместо O(V2)).

Раскраска графов находит применение и во многих практических областях, а не только в теоретических задачах. Помимо классических типов проблем, различные ограничения могут также быть наложены на граф, на способ присвоения цветов или на сами цвета. Этот метод, например, используется в популярной головоломке [Судоку](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%B4%D0%BE%D0%BA%D1%83" \o "Судоку). Разнообразные задачи, возникающие при планировании производства, составлении графиков осмотра, хранении и транспортировке товаров и т.д. могут быть представлены часто как задачи теории графов, тесно связанные с задачей раскраски.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

Луцик Ю.А., Ковальчук А.М., Сасин Е.А. – Учебное пособие по курсу «Основы алгоритмизации и программирования: язык С». – Минск: БГУИР, 2015 г.

Луцик Ю.А., Ковальчук А.М., Лукьянова И.В., Бушкевич А.В. – Лабораторный практикум по курсу «Основы алгоритмизации и программирования» (в 2-х частях). – Минск: БГУИР, 2008 г.

**ПРИЛОЖЕНИЕ А**

*(обязательное)*

Блок-схема алгоритма функции main()