Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в ширину»

Выполнили студенты группы 20ВВ3:

Мальков И.А.

Педай Н.Д.

Приняли:

Митрохин М.А.

Юрова О.В.

Пенза 2021

**Лабораторное задание**

**Задание 1**

1.Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для неориентированного графа G. Выведите матрицу на экран.

2. Для сгенерированного графа осуществите процедуру обхода в ширину,

реализованную в соответствии с приведенным выше описанием. При

реализации алгоритма в качестве очереди используйте класс queue из

стандартной библиотеки С++.

3.\* Реализуйте процедуру обхода в ширину для графа, представленного

списками смежности.

**Задание 2\***

1. Для матричной формы представления графов реализуйте алгоритм

обхода в ширину с использованием очереди, построенной на основе

структуры данных «список», самостоятельно созданной в лабораторной

работе № 3.

2. Оцените время работы двух реализаций алгоритмов обхода в ширину

(использующего стандартный класс queue и использующего очередь,

реализованную самостоятельно) для графов разных порядков.

**Ход работы:**

1. Код программы

#include <stdio.h>

#include <iostream>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <queue>

#include <stack> // стек

using namespace std;

typedef struct List {

int Data;

struct List\* next;

};

typedef struct Node {

int size;

List\* head;

List\* end;

};

void creation(Node \*tmp) //создание очереди

{

tmp->head = new List;

tmp->head->next = NULL;

tmp->end = tmp->head;

tmp->size = 0;

}

bool empty(Node \*tmp) // проверка на пустоту

{

if (tmp->head == tmp->end) return false;

else return true;

}

int head(Node \*tmp)

{

return tmp->head->next->Data;

}

void addition(Node \*tmp, int num) // добавление элемента

{

tmp->end->next = new List;

tmp->end = tmp->end->next;

tmp->end->Data = num;

tmp->end->next = NULL;

tmp->size++;

}

void del(Node \*tmp) // удаление элемента

{

List\* p;

p = tmp->head;

tmp->head = tmp->head->next; // смещение указателя

tmp->size--;

free(p);

}

void BFS\_list\_q(int\*\* A1, int n, bool\* A2, int a3)

{

Node tmp;

creation(&tmp);

addition(&tmp, a3); // помещаем в очередь вершину

A2[a3] = true;

cout << "list BFS queue result: ";

while (empty(&tmp)) // пока не пуст

{

a3 = head(&tmp); // извлекаем вершину

del(&tmp);

cout << a3 << " ";

//for (int j = n - 1; j >= 0; j--) // проверяем для нее все смежные вершины

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

{

if (A1[a3][j] == 1 && A2[j] == false) // если вершина смежная и не обнаружена

{

addition(&tmp, j); // добавляем ее в очередь

A2[j] = true; // отмечаем вершину как обнаруженную

}

}

}

cin.get();

}

List\* init(int num)

{

List\* node = (List\*)malloc(sizeof(List));

node->Data = num;

node->next = NULL;

return node;

}

List\* add(List\* lst, int num)

{

struct List\* temp, \* p;

temp = (List\*)malloc(sizeof(List));

p = lst->next;

lst->next = temp;

temp->Data = num;

temp->next = p;

return temp;

}

void initialize(int n, List\*\* vrt)

{

int i;

for (i = 0; i < n; i++)

{

vrt[i] = init(i);

}

}

void list(int n, int i, int\*\* A1, List\* lst)

{

int j;

for (j = n; j >= 0; j--)

{

if (A1[i][j] == 1)

add(lst, j);

}

}

void result(List\*\* vrt, int\*\* A1, int n)

{

int i;

for (i = 0; i < n; i++)

{

list(n, i, A1, vrt[i]);

}

}

void print(List\*\* vrt, int n)

{

int i, j;

for (i = 0; i < n; i++)

{

cout << endl;

List\* x = vrt[i];

for (j = 0; j < n; j++)

{

if (vrt[i] != NULL)

{

cout << vrt[i]->Data<< " ";

vrt[i] = vrt[i]->next;

}

}

vrt[i] = x;

}

cout << endl;

}

void BFS\_list(bool\* A2, int n, List\*\* vrt, int a4)

{

queue<int> q;

List\* h;

int i = a4;

q.push(a4); // помещаем в очередь вершину

A2[i] = true;

cout << "BFS list queue result: ";

while (!q.empty()) // пока не пуст

{

i = q.front(); // извлекаем вершину

cout << i << " ";

q.pop();

h = vrt[i];

//for (int j = n - 1; j >= 0; j--) // проверяем для нее все смежные вершины

while(vrt[i] != 0)

{

if(A2[vrt[i]->Data] == false)

{

A2[vrt[i]->Data] = true;

q.push(vrt[i]->Data);

}

vrt[i] = vrt[i]->next;

}

vrt[i] = h;

}

cout << endl;

cin.get();

}

void BFS(int\*\* A1, int n, bool\* A2, int a1)

{

queue<int> q;

q.push(a1); // помещаем в очередь вершину

A2[a1] = true;

cout << "BFS queue result: ";

while (!q.empty()) // пока не пуст

{

a1 = q.front(); // извлекаем вершину

q.pop();

cout << a1 << " ";

//for (int j = n - 1; j >= 0; j--) // проверяем для нее все смежные вершины

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

{

if (A1[a1][j] == 1 && A2[j] == false) // если вершина смежная и не обнаружена

{

q.push(j); // добавляем ее в очередь

A2[j] = true; // отмечаем вершину как обнаруженную

}

}

}

cin.get();

}

int\*\* create(int n)

{

int\*\* A1, k = 1;

A1 = (int\*\*)malloc(n \* sizeof(int\*));

srand(time(NULL));

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

{

A1[i] = (int\*)malloc(n \* sizeof(int));

A1[i][i] = 0;

for (int j = k; j < n; j++)

{

if (rand() % 100 > 50)

{

A1[i][j] = 0;

}

else {

A1[i][j] = 1;

}

}

k++;

}

k = 1;

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

{

for (int j = k; j < n; j++)

{

A1[j][i] = A1[i][j];

}

k++;

}

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

cout << "V" << i << " ";

/\* for (int i = 0; i < n; i++)

{

cout << endl;

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

{

cout << A1[i][j] << " ";

}

}\*/

return A1;

}

int main()

{

clock\_t start, end;

int n, \*\* t, s, o, a1, a3, a4;

cout << "Enter array size: ";

cin >> n;

List\*\* vrt = (List\*\*)malloc(n \* sizeof(List\*));

bool\* A2 = (bool\*)malloc(n \* sizeof(bool));

t = create(n);

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

A2[i] = false;

cout << endl;

cout << "Enter vertex: ";

cin >> a1;

start = clock(); // старт таймера

BFS(t, n, A2, a1);

end = clock(); // остановка таймера

float diff = (end - start);

cout << endl;

cout << "time is: " << diff << endl;

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

A2[i] = false;

initialize(n, vrt);

result(vrt, t, n);

//print(vrt, n);

cout << "Enter vertex: ";

cin >> a4;

BFS\_list(A2, n, vrt, a4);

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

A2[i] = false;

cout << "Enter vertex: ";

cin >> a3;

start = clock(); // старт таймера

BFS\_list\_q(t, n, A2, a3);

end = clock(); // остановка таймера

float diff1 = (end - start);

cout << endl;

cout << "time is: " << diff1 << endl;

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

free(t[i]);

free(t);

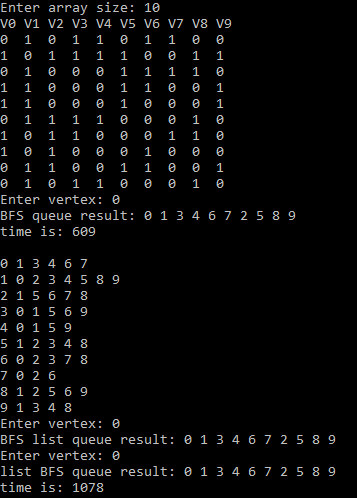
free(A2);

system("pause");

return 0;

}

1. Результат работы программы.



t1 – время с использованием стандартного класса queue.

t2 – время с использованием очереди, реализованной самостоятельно.

Матрица 5000\*5000

t­1 = 1256

t­2 = 1444

Матрица 2500\*2500

t­1 = 531

t­2 = 593

Матрица 1250\*1250

t­1 = 219

t­2 = 250

При использовании стандартного класса queue алгоритм обхода в ширину выполняется быстрее.

Вывод: в ходе выполнения данной лабораторной работы разработали программу, согласно заданию.