Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему « Поиск расстояний в графе»

Выполнил:

студент группы 20ВВ2

Кочергин В.П.

Принял:

к.т.н. доцент Юрова О.В.

д.т.н. профессор Митрохин М. А.

Пенза 2021

**Цель работы:** Реализовать алгоритм поиска расстояний в матрице и списке смежности

**Задание 1** Реализовать алгоритм на основе обхода в ширину

Листинг:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#include <queue>

#include <stack>

using namespace std;

struct Node

{

int items;

Node\* next;

};

typedef Node\* PNode;

PNode\* Head;

void InitializHead(int size)

{

int i;

Head = (PNode\*)malloc(size \* sizeof(PNode));

for (i = 0; i < size; i++) Head[i] = 0;

}

PNode CreateNode(int items)

{

PNode NewNode = (Node\*)malloc(sizeof(Node));

NewNode->items = items;

NewNode->next = NULL;

return NewNode;

}

void AddFirst(PNode NewNode, int i)

{

NewNode->next = Head[i];

Head[i] = NewNode;

}

void AddAfter(PNode q, PNode NewNode)

{

NewNode->next = q->next;

q->next = NewNode;

}

void AddLast(PNode NewNode, int i)

{

PNode q = Head[i];

while (q->next) q = q->next;

AddAfter(q, NewNode);

}

void print(int i)

{

PNode q = Head[i];

while (q)

{

printf("%d ", q->items);

q = q->next;

}

printf("\n");

}

void BFS(int\* dist ,int\*\* a, int size, int num)

{

queue <int> q;

int i, v;

q.push(num);

dist[num] = 0;

while (!q.empty())

{

num = q.front();

q.pop();

printf("%d ", num);

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

{

if (dist[i] == -1 && a[num][i] == 1)

{

q.push(i);

dist[i] = dist[num] + 1;

}

}

}

}

void BFS2(int\* dist, int\*\* a, int size, int num)

{

queue <int> q;

int i;

q.push(num);

dist[num] = 0;

while (!q.empty())

{

num = q.front();

q.pop();

printf("%d ", num);

while (Head[num])

{

if (dist[Head[num]->items] == -1)

{

q.push(Head[num]->items);

dist[Head[num]->items] = dist[num] + 1;

}

Head[num] = Head[num]->next;

}

}

}

void DFS(int\*\* a, int\* dist, int size, int num) {

stack <int> s;

int i;

s.push(num);

dist[num] = 0;

while (!s.empty())

{

num = s.top();

s.pop();

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

{

if ((a[num][i] == 1) && (dist[i] == -1))

{

s.push(i);

dist[i] = dist[num] + 1;

}

}

}

}

void DFS2(int\*\* a, int\* dist, int num, int size)

{

std::stack <int> s;

s.push(num);

dist[num] = 0;

while (!s.empty())

{

num = s.top();

s.pop();

while (Head[num])

{

if (dist[Head[num]->items] == -1)

{

s.push(Head[num]->items);

dist[Head[num]->items] = dist[num] + 1;

}

Head[num] = Head[num]->next;

}

}

}

int main()

{

int i, j, size, \*\* a, \*dist, ran, num, type, \*parent;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setvbuf(stdin, NULL, \_IONBF, 0);

setvbuf(stdout, NULL, \_IONBF, 0);

PNode NewNode = NULL;

printf("Введите размеры массива ");

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

printf("Введите начальную строку ");

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

printf("Выберите тип обхода\n1-В ширину\n2-В глубину\n ");

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

printf("Матрица смежности:\n");

a = (int\*\*)malloc(size \* sizeof(int));

parent = (int\*)malloc(size \* sizeof(int));

dist = (int\*)malloc(size \* sizeof(int));

srand(time(NULL));

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

{

dist[i] = -1;

}

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

{

a[i] = (int\*)malloc(size \* sizeof(int));

}

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

{

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

{

a[i][j] = 0;

}

}

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

{

for (j = 1 + i; j < size; j++)

{

ran = rand() % 101;

if (ran > 50) a[i][j] = 1;

else a[i][j] = 0;

}

}

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

{

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

{

a[j][i] = a[i][j];

}

}

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

{

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

{

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

}

printf("\n");

}

switch (type)

{

case 1:

{

printf("\nОчерёдность посещения вершин: ");

BFS(dist, a, size, num);

printf("\nРасстояния до вершин: ");

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

{

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

}

InitializHead(size);

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

{

NewNode = CreateNode(i);

AddFirst(NewNode, i);

}

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

{

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

{

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

{

NewNode = CreateNode(j);

AddLast(NewNode, i);

}

}

}

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

{

dist[i] = -1;

}

printf("\nОчерёдность посещения вершин: ");

BFS2(dist, a, size, num);

printf("\nРасстояния до вершин: ");

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

{

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

}

break;

}

case 2:

{

printf("\nРасстояние пройденное до каждой вершины:\n");

dist[num] = 0;

DFS(a, dist, size, num);

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

{

printf("Вершина %d - %d\n", i, dist[i]);

}

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

{

dist[i] = -1;

}

InitializHead(size);

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

NewNode = CreateNode(i);

AddFirst(NewNode, i);

}

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

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

if (a[i][j] == 1) {

NewNode = CreateNode(j);

AddLast(NewNode, i);

}

}

}

printf("\n\nРасстояние и обход для глубины представленной списками\n");

printf("Расстояние пройденное до каждой вершины:\n");

DFS2(a, dist, num, size);

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

{

printf("Вершина %d - %d\n", i, dist[i]);

}

break;

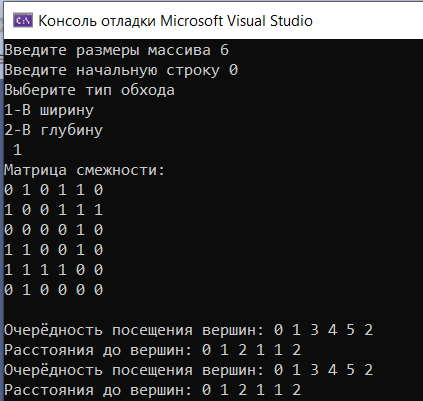
}

}

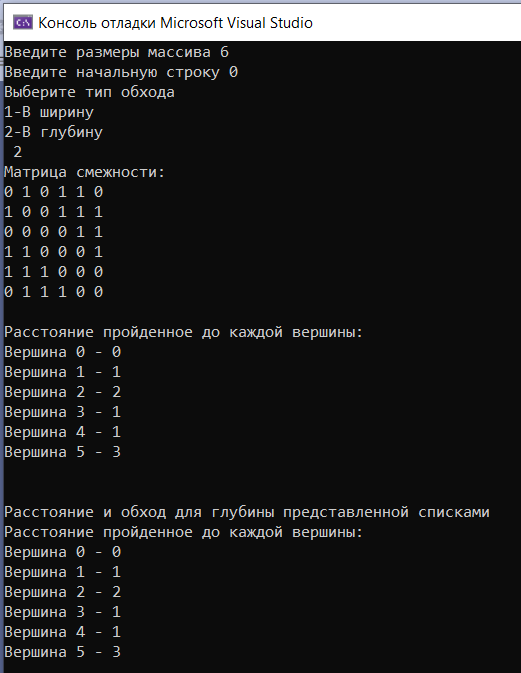
}

Результат работы программы

Запуск подсчёта расстояний в ширину



Запуск подсчёта расстояний в глубину



**Вывод:** мне удалось написать алгоритм поиска расстояний в списке и матрице смежности