|  |
| --- |
| Министерство образования Российской Федерации  Пензенский государственный университет  Кафедра «Вычислительная техника» |
| Отчет  по лабораторной работе №7  по курсу «Логика и основы алгоритмизации в инженерных задачах»  на тему «Поиск расстояний во взвешенном графе» |
|  |
|  |
| Выполнила студентка группы 19ВВ3:  Ханбекова Е. В.  Принял:  Митрохин М. А. |
| Пенза  2020 |

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При  реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**Листинг**

#include <stdio.h>

#include <conio.h>

#include <Windows.h>

#include <stdlib.h>

#include <queue>

#include <iostream>

using namespace std;

typedef int tData;

typedef struct sNode {

tData value;

tData nom;

struct sNode \*next;

} tNode;

tNode\* create\_list(int N, int k, int \*\*A, int i)

{

tNode \*p\_begin = NULL;

tNode \*p = NULL;

//заполним односвязный список

p\_begin = (tNode \*)malloc(sizeof(tNode));

p = p\_begin;

p->next = NULL;

p->value = i + 1;

p->nom = i + 1;

for (int x = 0; x < N; x++) {

if (A[i][x] != 0) {

p->next = (tNode \*)malloc(sizeof(tNode));

p = p->next;

//заполнить \_новую\_ структуру данных

p->next = NULL;

p->nom = x + 1;

p->value = A[i][x];

}

}

return p\_begin;

}

void print\_list(tNode \*p\_begin)

{

tNode \*p = p\_begin;

while (p != NULL) {

printf("-> %d ", p->nom);

p = p->next;

}

}

tNode \*\*p\_begin;

void BFSSPISOK(int i, int \*M, int \*\*A) {

queue <int> Q;

Q.push(i);

M[i] = 0;

while (Q.empty() != true) {

i = Q.front();

tNode \*p\_b = p\_begin[i];

Q.pop();

while (p\_b->next != NULL) {

int j = p\_b->next->nom -1;

p\_b = p\_b->next;

if (M[j] > M[i] + A[i][j]) {

M[j] = 0;

Q.push(j);

M[j] = M[i] + p\_b->value;

}

}

}

}

int BFS(int\*\*A, int\*M, int i, int N) {

queue <int> Q;

int j;

Q.push(i);

M[i] = 0;

while (Q.empty() != true) {

i = Q.front();

Q.pop();

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

if ((A[i][j] != 0) && (M[j] > M[i] + A[i][j])) {

M[j] = 0;

Q.push(j);

M[j] = M[i] + A[i][j];

}

}

}

return 0;

}

int main(void)

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int N, i, j;

printf("Ведите размер матрицы \n");

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

srand(12);

int \*M = (int\*)malloc(N \* sizeof(int));

int \*\*A = (int \*\*)malloc(N \* sizeof(int \*));

for (int i = 0; i < N; i++) {

A[i] = (int \*)malloc(N \* sizeof(int));

}

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

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

A[i][j] = rand() % 4;

A[j][i] = A[i][j];

if (i == j) {

A[i][j] = 0;

}

}

}

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

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

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

}

printf("\n");

}

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

M[i] = 1000;

}

printf("\n\nПоиск расстояний :\n");

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

BFS(A, M, i, N);

printf("\n");

for (int l = 0; l < N; l++) {

printf("%d ", M[l]);

}

for (int l = 0; l < N; l++) {

M[l] = 1000;

}

}

printf("\n\n Список :\n");

p\_begin = (tNode\*\*)malloc(N \* sizeof(tNode\*));

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

int k = 0;

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

if (A[i][j] != 0)

k++;

}

p\_begin[i] = create\_list(N, k, A, i);

print\_list(p\_begin[i]);

printf("\n");

}

printf("\n\nПоиск расстояний списками:\n");

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

M[i] = 1000;

}

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

BFSSPISOK(i, M, A);

printf("\n");

for (int l = 0; l < N; l++) {

printf("%d ", M[l]);

}

for (int l = 0; l < N; l++) {

M[l] = 1000;

}

}

free(A);

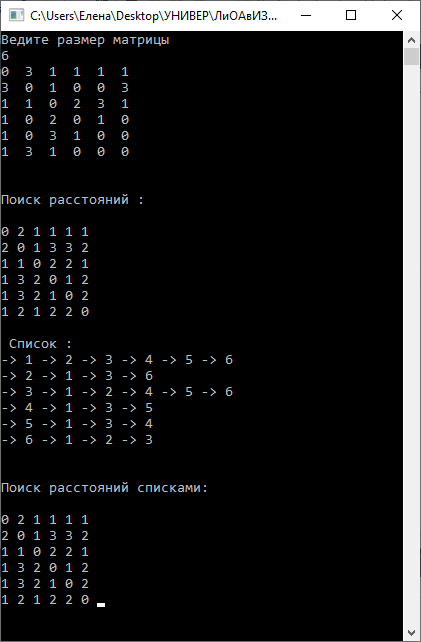
free(M);

free(p\_begin);

\_getch();

}

**Результат работы программы**



**Вывод:**

В результате работы был реализован поиск кратчайших путей во взвешенном графе на основе обхода графа в ширину. Отличие от поиска расстояний в не взвешенном графе, при обновлении расстояния до вершины при ее посещении оно увеличивается не на 1, а на величину веса ребра. Для поиска кратчайшего путь до вершины проверяется, является ли он наименьшим.