Министерство образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №8

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в ширину»

Выполнили

студент группы 21ВВ3:

Савицкий Макар

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2022

**Цель работы**

Реализация алгоритмов обхода графа в ширину для списков и матриц смежности.

**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.

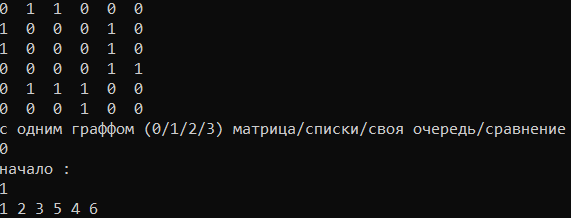
**Задание 2\***

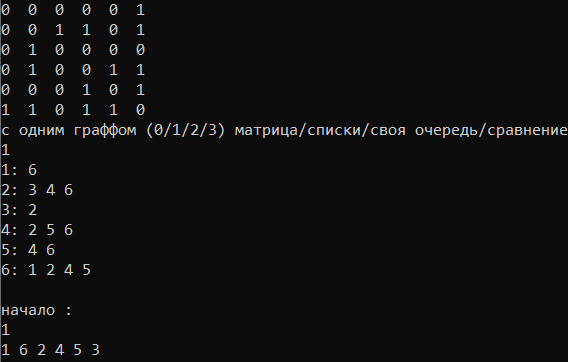
1. Для матричной формы представления графов реализуйте алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», самостоятельно созданной в лабораторной работе № 3.
2. Оцените время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков.

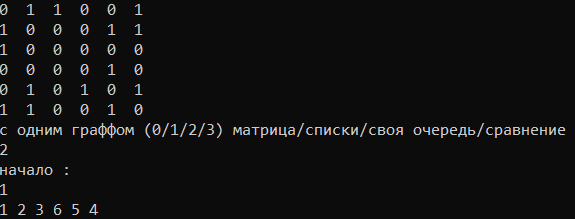
**Описание метода решения задачи**

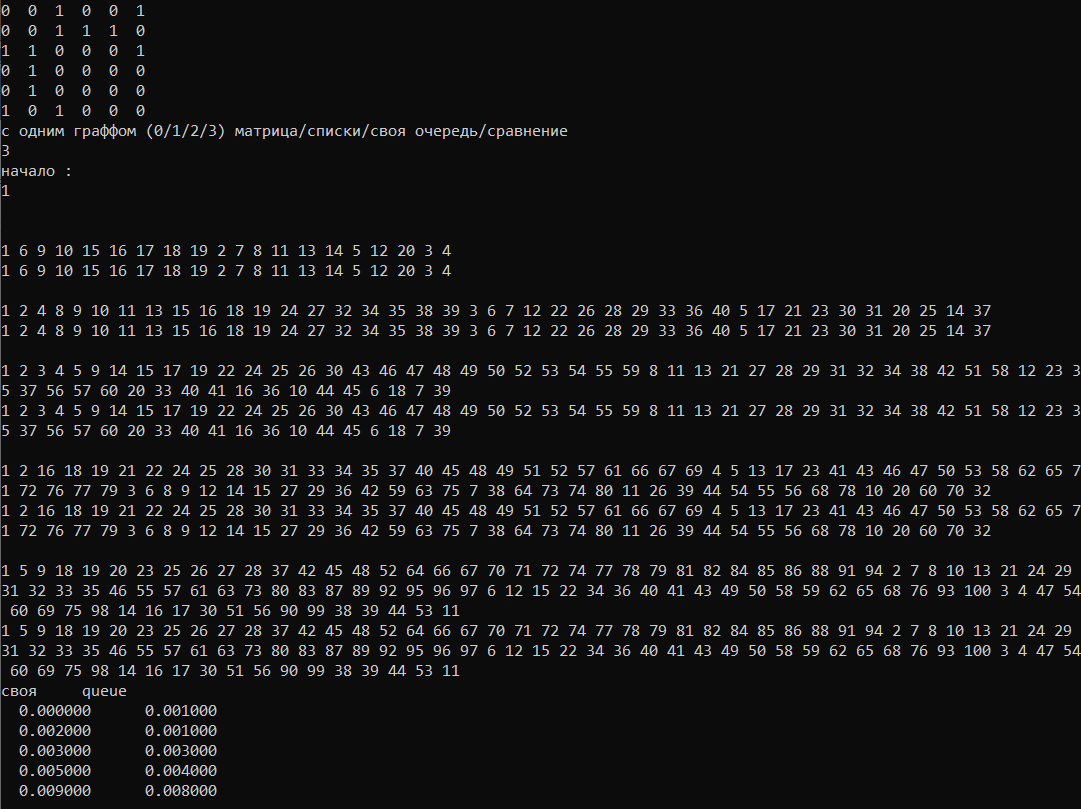
1. *Генерируем матрицу смежности*
2. *В зависимости от введенного значения выполняем соответствующую операцию*
3. *Выбираем с какой вершины начать обход*
4. *В зависимости от введённого значения выполняем ту или иную команду*
5. *Выводим очередность вершин, в порядке обхода*

**Результаты работ программы**

****

****

****

****

**Листинг**#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h> //указание прототипов

#include <stdlib.h> //библиотечных функций

#include <time.h>

#include <windows.h>

#include <queue>

using namespace std;

struct node {

int inf;

struct node\* next;

};

struct node\* head = NULL, \* last = NULL;

void push(int t) {

struct node\* p = NULL, \* f = NULL;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL) {

exit(1);

}

p->inf = t;

p->next = NULL;

if (head == NULL) {

head = p;

last = p;

}else {

p->next = head;

head = p;

}

return;

}

int pop() {

int t = last->inf;

if (head == NULL)

return -1;

if (head == last) {

free(head);

head = NULL;

last = NULL;

}

else {

struct node\* struc = head;

while (struc->next != last)

struc = struc->next;

last = struc;

free(struc->next);

struc->next = NULL;

}

return t;

}

struct list {

int i;

struct list\* next;

};

struct branch {

int t;

struct list\* head;

struct list\* last;

struct branch\* next;

};

struct branch\* head\_b = NULL, \* last1 = NULL;

void spstore\_branch(int t) {

struct branch\* p = NULL;

if ((p = (branch\*)malloc(sizeof(struct branch))) == NULL) // выделяем память под новый элемент списка

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

p->t = t;

p->head = NULL;

p->last = NULL;

p->next = NULL;

if (p != NULL) {

if (head\_b == NULL) {

head\_b = p;

last1 = p;

}

else {

last1->next = p;

last1 = p;

}

}

}

void spstore\_list(int i) {

struct branch\* struc1 = last1; // указатель, проходящий по списку установлен на начало списка

struct branch\* prev1 = NULL;

if (head\_b == NULL) {

return;

}

struct list\* p = NULL, \* f = NULL;

if ((p = (list\*)malloc(sizeof(struct list))) == NULL) { // выделяем память под новый элемент списка

printf("Ошибка при распределении памяти\n");

exit(1);

}

p->i = i;

p->next = NULL;

if (struc1->head == NULL && p != NULL) { // если списка нет, то устанавливаем голову списка

struc1->head = p;

struc1->last = p;

}

else if (struc1->head != NULL && p != NULL) {

f = struc1->head;

int ii = 0;

if (f->i > p->i) { //если в начало ставим

p->next = struc1->head;

struc1->head = p;

ii = 1;

}

if (struc1->last->i < p->i) { //если в конец ставим

struc1->last->next = p;

struc1->last = p;

ii = 1;

}

if (struc1->last->i > p->i && f->i < p->i) {

while (f->next->i < p->i) {

f = f->next;

}

p->next = f->next;

f->next = p;

ii = 1;

}

if (ii == 0) {

free(p);

}

}

}

void review\_uz() {

struct branch\* struc = head\_b;

struct list\* struc1;

while (struc) {

printf("%d: ", struc->t + 1);

struc1 = struc->head;

while (struc1) {

printf("%-2d", struc1->i);

struc1 = struc1->next;

}

printf("\n");

struc = struc->next;

}

}

int\* craft(int size\_G) {

int\* G = (int\*)malloc(size\_G \* size\_G \* sizeof(int));

for (int i = 0; i < size\_G; i++) {

for (int l = i; l < size\_G; l++) {

\*(G + i \* size\_G + l) = (rand() % 5 + 7) / 10;

\*(G + l \* size\_G + i) = \*(G + i \* size\_G + l);

if (l == i) \*(G + i \* size\_G + l) = 0;

}

}

return G;

}

void print(int\* G, int size\_G) {

for (int i = 0; i < size\_G; i++) {

for (int l = 0; l < size\_G; l++) {

printf("%d ", \*(G + i \* size\_G + l));

}

printf("\n");

}

}

void matrixs(int\* G, int\* NUM, int size\_G, int v) {

for (int l = 0; l < size\_G; l++)

\*(NUM + l) = 0;

queue<int> Queue;

Queue.push(v - 1);

while (!Queue.empty()) {

int nod = Queue.front();

Queue.pop();

if (\*(NUM + nod) == 0){

\*(NUM + nod) = 1;

for (int i = 0; i < size\_G; i++) {

if (\*(G + nod \* size\_G + i) == 1)

if (\*(NUM + i) == 0)

Queue.push(i);

}

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

}

}

}

void matrixs11(int\* G, int\* NUM, int size\_G, int v) {

for (int l = 0; l < size\_G; l++)

\*(NUM + l) = 0;

push(v - 1);

while (head) {

int nod = pop();

if (\*(NUM + nod) == 0) {

\*(NUM + nod) = 1;

for (int i = 0; i < size\_G; i++) {

if (\*(G + nod \* size\_G + i) == 1)

if (\*(NUM + i) == 0)

push(i);

}

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

}

}

}

void lists(int\* NUM, int size\_G, int v) {

struct branch\* p = head\_b;

struct list\* pp = p->head;

for (int l = 0; l < size\_G; l++)

\*(NUM + l) = 0;

queue<int> Queue;

Queue.push(v);

while (!Queue.empty()) {

int nod = Queue.front();

Queue.pop();

p = head\_b;

for(int i = 1; i < nod; i++)

p = p->next;

pp = p->head;

if (\*(NUM + nod - 1) == 0) {

\*(NUM + nod - 1) = 1;

while (pp) {

Queue.push(pp->i);

pp = pp->next;

}

printf("%d ", nod);

}

}

}

void main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

clock\_t begin, end;

double time[2][5];

int size\_G = 6, t1;

int\* G = craft(size\_G);

int\* NUM = (int\*)malloc(size\_G \* sizeof(int));

for (int i = 0; i < size\_G; i++) {

for (int l = 0; l < size\_G; l++) {

printf("%d ", \*(G + i \* size\_G + l));

}

printf("\n");

}

printf("с одним граффом (0/1/2/) матрица/списки/своя очередь/сравнение\n");

scanf("%d", &t1);

switch (t1) {

case 0:

printf("начало : \n");

scanf("%d", &t1);

matrixs(G, NUM, size\_G, t1);

break;

case 1:

for (int i = 0; i < size\_G; i++) {

spstore\_branch(i);

for (int l = 0; l < size\_G; l++)

if (\*(G + i \* size\_G + l) == 1)

spstore\_list(l + 1);

}

review\_uz();

printf("\n");

printf("начало : \n");

scanf("%d", &t1);

lists(NUM, size\_G, t1);

break;

case 2:

printf("начало : \n");

scanf("%d", &t1);

matrixs11(G, NUM, size\_G, t1);

break;

case 3:

printf("начало : \n");

scanf("%d", &t1);

for (int i = 20; i < 101; i += 20) {

printf("\n\n");

G = craft(i);

size\_G = i;

begin = clock();

matrixs11(G, NUM, size\_G, t1);

end = clock();

time[0][i / 20 - 1] = (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("\n");

begin = clock();

matrixs(G, NUM, size\_G, t1);

end = clock();

time[1][i / 20 - 1] = (double)(end - begin) / CLOCKS\_PER\_SEC;

}

printf("\nсвоя queue\n");

for (int i = 0; i < 5; i++) {

printf("%10f %10f\n", time[0][i], time[1][i]);

}

break;

}

}

**Вывод**

В ходе выполнения данной лабораторной работы мы научились реализовывать алгоритмы обхода графа в ширину для списков и матриц смежности.