Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра "Вычислительная техника"

Отчет

по лабораторной работе №8

по курсу “Л и ОА в ИЗ”

на тему “Обход графа в ширину”

Выполнили студенты группы 22ВВС1:

Агапов И.Е.

Братчиков А.А.

Приняли:

Юрова О.В.

Акифьев И.В.

Пенза 2023

### 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.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <iostream>

#include <queue>

using namespace std;

typedef struct Node {

int value;

struct Node\* next;

} Node;

struct Node\* head = NULL, \* last = NULL, \* f = NULL; // указатели на первый и последний элементы списка

Node\* create(int data)

{

// Выделение памяти под корень списка

Node\* tmp = (Node\*)malloc(sizeof(Node));

tmp->value = data;

// Присваивание указателю на следующий элемент значения NULL

tmp->next = NULL;

return(tmp);

}

Node\* pushStart(int data, Node\* head)

{

Node\* tmp = create(data);

// Присваивание указателю на следующий элемент значения указателя на «голову»

// первоначального списка

tmp->next = head;

return(tmp);

}

char find\_el[256];

void spstore(int x), pop(void);

struct Node\* get\_struct(int x); // функция создания элемента

void BFS(queue <int> Q, int size, bool\* vis, int\*\* mas, int S) {

Q.push(S);

vis[S] = true;

while (!Q.empty()) {

S = Q.front();

printf("%d\t", S + 1);

Q.pop();

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

if (mas[S][i] == 1 && !vis[i]) {

vis[i] = true;

Q.push(i);

}

}

}

return;

}

void BFSselfqueue(int size, bool\* vis, int\*\* mas, int S) {

spstore(S);

vis[S] = true;

while (last!=NULL) {

if (head == NULL) {

return;

}

S = head->value;

printf("%d\t", S + 1);

pop();

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

if (mas[S][i] == 1 && !vis[i]) {

vis[i] = true;

spstore(i);

}

}

}

return;

}

void BFSsps(queue <int> Q, int size, bool\* vis, Node\*\* last, int S) {

int y = 0;

printf("%d\t", S + 1);

Q.push(last[S]->value);

vis[S] = true;

vis[last[S]->value] = true;

while (!Q.empty()) {

S = Q.front();

printf("%d\t", S + 1);

Q.pop();

y = last[S]->value;

while (last[S] != NULL) {

if (!vis[last[S]->value]) {

Q.push(last[S]->value);

vis[last[S]->value] = true;

}

last[S] = last[S]->next;

}

}

return;

}

int main() {

queue <int> SomeQ;

Node\*\* sps;

Node\*\* last;

int\*\* mas;

int size;

bool\* Num;

int w;

srand(241324);

scanf("%d", &size);

sps = (Node\*\*)malloc(sizeof(Node\*) \* size);

last = (Node\*\*)malloc(sizeof(Node\*) \* size);

w = size;

Num = (bool\*)malloc(sizeof(bool) \* size);

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

int j = 0;

int i = 0;

setlocale(LC\_ALL, "");

for (int k = 0; k < size; k++) {

Num[k] = false;

}

while (size > i) {

j = 0;

mas[i] = (int\*)malloc(sizeof(int) \* w);

while (w > j) {

mas[i][j] = 0;

j++;

}

i++;

}

i = 0;

j = 0;

while (size > i) {

j = i;

while (w > j) {

if (i == j) {

mas[i][j] = 0;

}

else {

mas[i][j] = rand() % 2;

mas[j][i] = mas[i][j];

}

j++;

}

i++;

}

i = 0;

j = 0;

bool fl = true;

while (size > i) {

j = 0;

while (w > j) {

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

if (fl) {

last[i] = pushStart(j, NULL);

fl = false;

}

else {

last[i] = pushStart(j, last[i]);

}

}

j++;

}

i++;

fl = true;

}

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

sps[i] = last[i];

}

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

printf("%d\t", i + 1);

while (last[i] != NULL) {

printf("%d\t", last[i]->value + 1);

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

}

printf("\n");

}

printf("\n");

printf("\n");

printf("\n");

printf("\t");

for (int o = 0; o < size; o++) {

printf("%d\t", o + 1);

}

printf("\n");

for (int o = 0; o < size; o++) {

printf("%d\t", o + 1);

for (int n = 0; n < w; n++) {

printf("%d\t", mas[o][n]);

}

printf("\n");

}

printf("\n");

printf("\n");

printf("\n");

w = 0;

scanf("%d", &w);

w--;

int w2 = w;

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

last[i] = sps[i];

}

//BFSselfqueue(SomeQ,size,Num,mas,w);

clock\_t start = clock();

BFSsps(SomeQ, size, Num, sps, w2);

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\nВремя списки : %f сек\n", seconds);

printf("\n");

printf("\n");

printf("\n");

for (int k = 0; k < size; k++) {

Num[k] = false;

}

start = clock();

BFSselfqueue(size, Num, mas, w);

end = clock();

seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\nВремя своя очередь: %f сек\n", seconds);

for (int k = 0; k < size; k++) {

Num[k] = false;

}

start = clock();

BFS(SomeQ,size, Num, mas, w);

end = clock();

seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\nВремя библеотечная: %f сек\n", seconds);

}

void spstore(int x)

{

struct Node\* p = NULL;

p = get\_struct(x);

if (head == NULL) {

head = p;

last = p;

}

else {

last->next = p;

last = p;

f = p;

}

return;

}

struct Node\* get\_struct(int x)

{

struct Node\* p = NULL;

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

{

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

exit(1);

}

p->value = x;

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

void pop() {

struct Node\* tmp;

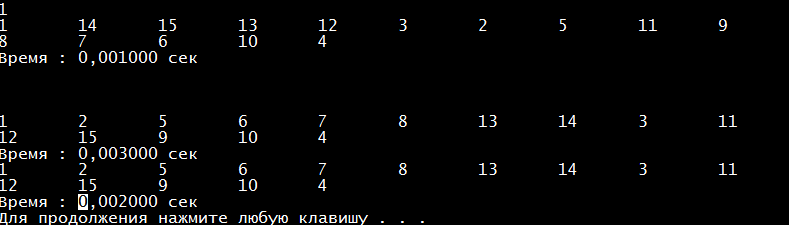
tmp = head;

head = tmp->next;

free(tmp);

}

**Результат работы программы**

****

**Вывод**

### В ходе выполнения лабораторной работы были реализованы функции поиска в ширину для графа с помощью библиотечной очереди queue, и при помощи собственной очереди основанной на односвязном списке из лабораторной работы №3, также был реализован алгоритм использующий списки связности.

### Были выполнены измерения скорости выполнения алгоритмов, результаты измерений представлены в данном отчете, мы изучили зависимость скорости каждой функции от размера графа.