Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

По лабораторной работе №9

По курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний в графе»

**Выполнил**

**студент группы 23ВВВ2:**

Родионов А. А.

**Приняли:**

Митрохин М. А.

Юрова О. В.

Пенза 2024

**Цель работы:** изучить и реализовать процедуру поиска расстояний для графов, представленных в матричном виде и в виде списков смежности, а также разработать алгоритм поиска расстояний на основе обхода в глубину и оценить время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину.

**Задание 1:**

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

\* **Задание 2:**

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.
2. Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

**Ход работы**

**Задание 1**

**1.** **Поиск расстояний для матричной формы представления графов**

1. Функция find\_distances инициирует процесс обхода:

void find\_distances(int size, int\_fast8\_t\*\* graph\_pointer) {

// Запрашивает начальную вершину у пользователя

int start\_vertex;

printf("\nВведите начальную вершину для поиска расстояний: ");

scanf("%d", &start\_vertex);

start\_vertex--; // Преобразование номера вершины в индекс массива

// Проверка корректности введенной вершины

if (start\_vertex < 0 || start\_vertex >= size) {

printf("Ошибка: некорректный номер вершины\n");

return;

}

// Создание массива расстояний

int\* DIST = new int[size];

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

DIST[i] = -1; // Инициализация расстояний значением -1

}

// Вызов алгоритма поиска в ширину

BFSD(start\_vertex, size, graph\_pointer, DIST);

// Вывод результатов

printf("\n\nРасстояния от вершины %d до остальных вершин:\n\n", start\_vertex + 1);

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

printf("До вершины %d: %d\n", i + 1, DIST[i]);

}

delete[] DIST;

}

1. Алгоритм поиска в ширину (BFSD):

void BFSD(int v, int size, int\_fast8\_t\*\* graph\_pointer, int\* DIST) {

queue<int> Q; // Создание очереди для BFS

Q.push(v); // Добавление начальной вершины в очередь

DIST[v] = 0; // Расстояние до начальной вершины = 0

while (!Q.empty()) {

int current = Q.front(); // Получение текущей вершины

Q.pop(); // Удаление вершины из очереди

printf("%d ", current + 1); // Вывод порядка обхода

// Перебор всех вершин

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

// Если есть ребро и вершина не посещена

if (graph\_pointer[current][i] == 1 && DIST[i] == -1) {

Q.push(i); // Добавление вершины в очередь

DIST[i] = DIST[current] + 1; // Установка расстояния

}

}

}

}

Основные этапы работы:

1. Пользователь вводит начальную вершину
2. Создается массив DIST для хранения расстояний, изначально заполненный -1
3. Запускается поиск в ширину (BFS):
   * Начальная вершина помещается в очередь
   * Для начальной вершины устанавливается расстояние 0
   * Пока очередь не пуста:
     + Извлекается вершина из очереди
     + Просматриваются все её соседи
     + Если сосед не посещен (DIST[i] == -1), то:
       - Он добавляется в очередь
       - Для него устанавливается расстояние на 1 больше, чем до текущей вершины
4. В результате массив DIST содержит кратчайшие расстояния от начальной вершины до всех достижимых вершин

Если какая-то вершина недостижима из начальной, то соответствующее значение в массиве DIST останется равным -1.

**Результат работы программы**

На рисунке 1 приведён результат работы программы №1.1.

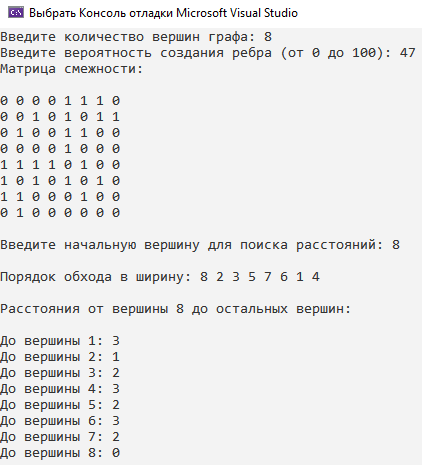


Рисунок 1 – результат работы программы №1.1

**2.** \***Поиск расстояний в графе, представленном списками смежности**

1. Инициализация:

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

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

DIST[i] = -1; // Изначально все расстояния помечаются как недостижимые (-1)

}

1. Создание очереди и добавление начальной вершины:

Queue\* Q = createQueue(size);

enqueue(Q, start\_vertex);

DIST[start\_vertex] = 0; // Расстояние до начальной вершины = 0

1. Основной цикл BFS:

while (!isEmpty(Q)) {

int v = dequeue(Q); // Извлекаем вершину из очереди

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

Node\* current = adj\_list[v];

while (current != NULL) { // Перебираем всех соседей текущей вершины

int i = current->vertex;

if (DIST[i] == -1) { // Если вершина ещё не посещена

enqueue(Q, i); // Добавляем её в очередь

DIST[i] = DIST[v] + 1; // Расстояние = расстояние до предыдущей + 1

}

current = current->next;

}

}

1. Вывод результатов:

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

printf("До вершины %d: %d\n", i + 1, DIST[i]);

}

Алгоритм работает следующим образом:

1. Начинает с заданной вершины, помечая расстояние до неё как 0
2. Посещает всех её непосредственных соседей (расстояние 1)
3. Затем посещает соседей соседей (расстояние 2)
4. И так далее, пока не будут найдены все достижимые вершины

Массив DIST хранит кратчайшие расстояния от начальной вершины до всех остальных вершин графа. Если какая-то вершина остается с значением -1, это означает, что она недостижима из начальной вершины.

**Результат работы программы**

На рисунке 2 приведён результат работы программы №1.2.

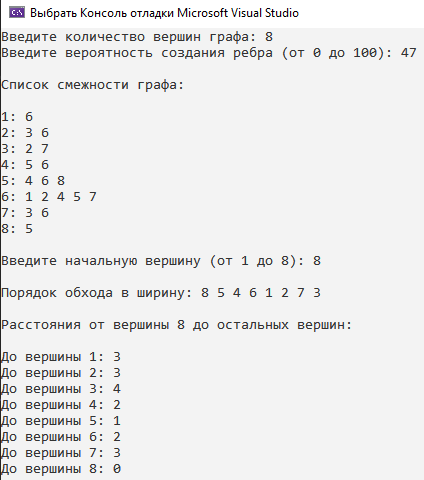


Рисунок 2 – результат работы программы №1.2

\***Задание 2**

**1. Поиск расстояний на основе обхода в глубину для матричной формы представления графов**

1. Функция find\_distances():
   * Запрашивает у пользователя начальную вершину для поиска расстояний
   * Создает массив DIST размером с граф, инициализируя все значения -1
   * Вызывает функцию DFSD() для обхода графа в глубину
2. Функция DFSD() (Depth-First Search with Distance):
   * Это рекурсивная функция обхода графа в глубину с одновременным вычислением расстояний
   * Параметры:
     + v - текущая вершина
     + size - размер графа
     + graph\_pointer - матрица смежности
     + DIST - массив расстояний
     + current\_dist - текущее расстояние от начальной вершины
3. Механизм работы DFSD():
   * Помечает текущую вершину расстоянием current\_dist
   * Просматривает всех соседей текущей вершины
   * Если сосед еще не посещен (DIST[i] == -1), рекурсивно вызывает себя для этого соседа
   * Увеличивает расстояние на 1 при каждом рекурсивном вызове
4. Результат:
   * Массив DIST содержит кратчайшие расстояния от начальной вершины до всех остальных
   * Выводит порядок обхода и расстояния до каждой вершины

**Результат работы программы**

На рисунке 3 приведён результат работы программы №2.1.

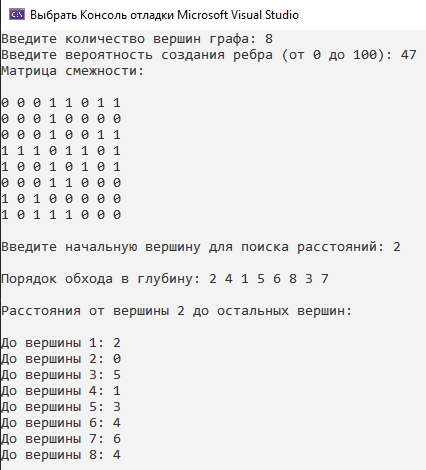


Рисунок 3 – результат работы программы №2.1

**2. Поиск расстояний на основе обхода в глубину для графов, представленных списками смежности**

1. Функция find\_distances:

void find\_distances(Node\*\* adj\_list, int size, int start\_vertex) {

// Создаем массив расстояний

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

// Инициализируем все расстояния как -1 (непосещенные вершины)

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

DIST[i] = -1;

}

// Запускаем DFS из начальной вершины

printf("\nПорядок обхода в глубину: ");

DFS(adj\_list, start\_vertex, DIST, 0);

// Выводим найденные расстояния

printf("\n\nРасстояния от вершины %d до остальных вершин:\n\n", start\_vertex + 1);

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

printf("До вершины %d: %d\n", i + 1, DIST[i]);

}

free(DIST); }

1. Функция DFS (поиск в глубину):

void DFS(Node\*\* adj\_list, int vertex, int\* DIST, int current\_dist) {

// Записываем текущее расстояние для вершины

DIST[vertex] = current\_dist;

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

// Обходим всех соседей текущей вершины

Node\* current = adj\_list[vertex];

while (current != NULL) {

int next\_vertex = current->vertex;

// Если вершина еще не посещена (расстояние = -1)

if (DIST[next\_vertex] == -1) {

// Рекурсивно вызываем DFS для этой вершины

DFS(adj\_list, next\_vertex, DIST, current\_dist + 1);

}

current = current->next;

}

}

Процесс работы:

1. Создается массив DIST для хранения расстояний до всех вершин.
2. Изначально все расстояния устанавливаются в -1 (непосещенные вершины).
3. Запускается DFS из начальной вершины с начальным расстоянием 0.
4. При посещении каждой вершины:
   * Записывается текущее расстояние до нее
   * Выводится номер вершины (показывая порядок обхода)
   * Рекурсивно обходятся все непосещенные соседи с увеличением расстояния на 1
5. После завершения обхода выводятся все найденные расстояния.

**Результат работы программы**

На рисунке 4 приведён результат работы программы №2.2.

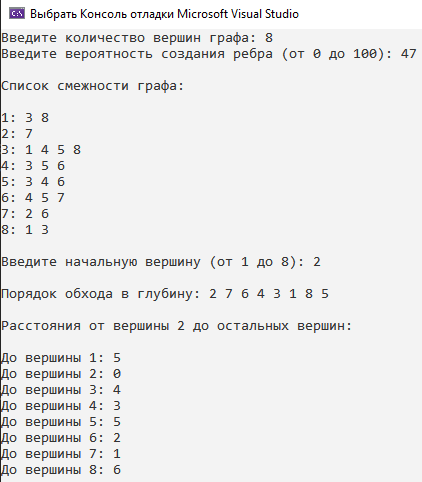


Рисунок 4 – результат работы программы №2.2

**3. Оценка времени работы двух реализаций алгоритмов обхода в ширину**

В программе время работы алгоритмов оценивается следующим образом:

1. Для каждого алгоритма замеряется время начала выполнения с помощью функции clock():

search1\_start = clock();

1. После выполнения алгоритма замеряется время окончания:

search1\_end = clock();

1. Вычисляется разница между временем окончания и началом, которая переводится в миллисекунды:

search\_time1 = ((double)(search1\_end - search1\_start)) / CLOCKS\_PER\_SEC \* 1000;

Что касается сравнения скорости работы алгоритмов:

Поиск в ширину (BFS) обычно работает быстрее поиска в глубину (DFS) для задачи поиска кратчайших расстояний по следующим причинам:

1. BFS гарантированно находит кратчайший путь до каждой вершины, посещая вершины уровень за уровнем.
2. BFS использует очередь, что делает его работу более эффективной в плане использования памяти и времени выполнения.
3. DFS может уходить в "глубокие" ветви графа, даже если более короткий путь существует через другие вершины, что приводит к дополнительным операциям.
4. В случае поиска расстояний, BFS не требует перезаписи уже найденных расстояний, в то время как DFS может несколько раз посещать одну и ту же вершину через разные пути.

Однако стоит отметить, что конкретное время выполнения может зависеть от:

* структуры графа
* количества вершин и рёбер
* начальной вершины
* вероятности создания рёбер

**Результат работы программы**

На рисунке 5 приведён результат работы программы №2.3.

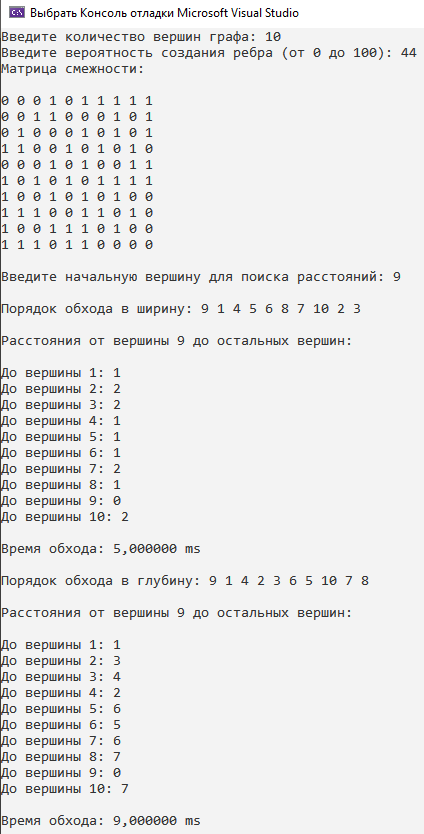


Рисунок 5 – результат работы программы №2.3

**Выводы**

Был изучен и реализован алгоритм поиска расстояний в графах на основе обхода в ширину и глубину в матричном виде и в виде списков смежности, а также было оценено время работы двух реализаций алгоритмов поиска расстояний.