Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

По лабораторной работе №7

По курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обходграфа в глубину»

**Выполнили**

**студенты группы 23ВВВ2:**

Пырков Д. А.

Родионов А. А.

**Приняли:**

Митрохин М. А.

Юрова О. В.

Пенза 2024

**Цель работы:** изучить и реализовать процедуру обхода графа в глубину для графов, представленных в матричном виде и в виде списков смежности, а также выполнить преобразование рекурсивной реализации обхода графа к не рекурсивной.

**Задание 1:**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран. Для сгенерированного графа осуществите процедуру обхода в глубину, реализованную в соответствии с приведенным описанием.
2. (\*) Реализуйте процедуру обхода в глубину для графа, представленного списками смежности.

\* **Задание 2:**

Для матричной формы представления графов выполните преобразование рекурсивной реализации обхода графа к не рекурсивной.

**Ход работы**

**Задание 1**

**1.** **Обход в глубину для матричной формы представления графов**

1. Функция depth\_search инициирует процесс обхода:

void depth\_search(int size, int\_fast8\_t\*\* graph) {

// Создаёт массив для отметки посещённых вершин

bool\* visited = (bool\*)calloc(size, sizeof(bool));

// Проходит по всем вершинам графа

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

// Если вершина ещё не посещена, запускает DFS с этой вершины

if (!visited[v]) {

DFS(v, size, graph, visited);

}

}

}

1. Основной алгоритм реализован в рекурсивной функции DFS:

void DFS(int v, int size, int\_fast8\_t\*\* graph, bool\* visited) {

// Отмечает текущую вершину как посещённую

visited[v] = true;

// Выводит номер текущей вершины

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

// Ищет все смежные вершины

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

// Если есть ребро и вершина ещё не посещена

if (graph[v][i] == 1 && !visited[i]) {

// Рекурсивно запускает DFS для этой вершины

DFS(i, size, graph, visited);

}

}

}

Алгоритм работает следующим образом:

1. Начинает с первой непосещённой вершины;
2. Отмечает её как посещённую;
3. Для каждой смежной непосещённой вершины рекурсивно запускает тот же процесс;
4. Если все смежные вершины посещены, возвращается к предыдущей вершине;
5. Процесс продолжается, пока не будут посещены все достижимые вершины;
6. Если остались непосещённые вершины (в случае несвязного графа), начинает процесс с новой непосещённой вершины.

Например, для графа:

0 1 0

1 0 1

0 1 0

Порядок обхода будет: 1 → 2 → 3, так как:

1. Начинает с вершины 1;
2. Находит связь с вершиной 2;
3. Из вершины 2 находит связь с вершиной 3;
4. Завершает обход, так как все вершины посещены.

**Результат работы программы**

На рисунке 1 приведён результат работы программы №1.1.

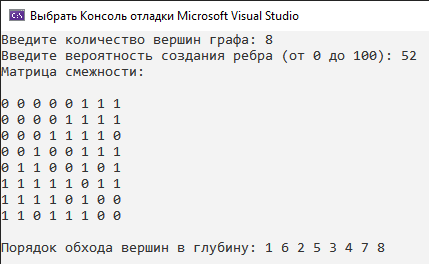


Рисунок 1 – результат работы программы №1.1

**2.** \***Обход в глубину для графа, представленного списками смежности**

1. Функция depth\_search инициирует процесс обхода:

void depth\_search(Node\*\* adj\_list, int size) {

// Создаём массив для отметки посещённых вершин

bool\* visited = (bool\*)calloc(size, sizeof(bool));

// Проходим по всем вершинам

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

// Если вершина ещё не посещена, начинаем обход с неё

if (!visited[v]) {

DFS(adj\_list, v, visited, size);

}

}

}

1. Основной алгоритм реализован в рекурсивной функции DFS:

void DFS(Node\*\* adj\_list, int vertex, bool\* visited, int size) {

// Помечаем текущую вершину как посещённую

visited[vertex] = true;

// Выводим её

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

// Проходим по всем смежным вершинам

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

while (current != NULL) {

// Если смежная вершина ещё не посещена

if (!visited[current->vertex]) {

// Рекурсивно запускаем обход из этой вершины

DFS(adj\_list, current->vertex, visited, size);

}

current = current->next;

}

}

Алгоритм работает следующим образом:

1. Начинает с произвольной вершины;
2. Помечает её как посещённую и выводит;
3. Для каждой смежной непосещённой вершины рекурсивно запускает тот же алгоритм;
4. Если все смежные вершины посещены, возвращается назад;
5. Процесс продолжается, пока не будут посещены все достижимые вершины.

Если в графе есть несколько несвязных компонент, внешний цикл в depth\_search обеспечивает обход всех вершин, начиная новый обход DFS для каждой непосещённой вершины.

**Результат работы программы**

На рисунке 2 приведён результат работы программы №1.2.

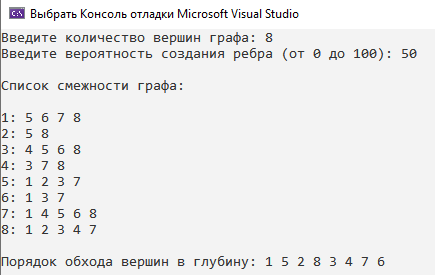


Рисунок 2 – результат работы программы №1.2

\***Задание 2**

**Нерекурсивный обход в глубину для матричной формы представления графов**

1. depth\_search(int size, int\_fast8\_t\*\* graph):

Эта функция является главным методом для запуска DFS. Она инициализирует массив visited размером size, который отслеживает посещенные вершины. Важно, что цикл for проходит по всем вершинам графа:

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

if (!visited[v]) {

DFS(v, size, graph, visited);

}

}

Если вершина v еще не посещена, вызывается функция DFS, чтобы начать обход в глубину от этой вершины. Этот итеративный подход гарантирует, что все компоненты связности графа будут обработаны.

2. DFS(int v, int size, int\_fast8\_t\*\* graph, bool\* visited):

Эта функция реализует сам алгоритм DFS. Она использует стек stack для отслеживания вершин, которые нужно посетить.

Stack\* stack = createStack(size \* size); // Стек для хранения вершин

Размер стека size \* size — это потенциальный максимальный размер стека, что может быть полезно для предотвращения переполнения стека, но в реальности может быть меньше.

push(stack, start); // Добавляем стартовую вершину в стек

while (!isEmpty(stack)) {

int v = pop(stack);

if (!visited[v]) {

visited[v] = true;

printf("%d ", v + 1); // Выводим вершину

for (int i = size - 1; i >= 0; i--) {

if (graph[v][i] == 1 && !visited[i]) {

push(stack, i);

}

}

}

}

Цикл while проходит по стеку, пока он не опустеет. В каждом цикле:

* int v = pop(stack);: Извлекается вершина из стека.
* if (!visited[v]): Если вершина не посещена:
  + visited[v] = true;: Помечаем вершину как посещенную.
  + printf("%d ", v + 1);: Выводим значение вершины (прибавляем 1, чтобы индексы соответствовали нумерации вершин в задаче).
  + for цикл: Просматриваем все смежные вершины. Если смежная вершина не посещена, она добавляется в стек с помощью push. Обратите внимание на порядок обхода смежных вершин (от size - 1 до 0). Это определяет порядок добавления вершин в стек и, соответственно, порядок обхода графа в глубину.

После завершения цикла while функция освобождает память, выделенную для стека.

**В итоге:**

Программа использует стек для хранения вершин, которые нужно посетить. В каждом шаге алгоритма она извлекает вершину из стека, проверяет, была ли она посещена. Если нет, то она помечает ее как посещенную, выводит её номер, а затем добавляет в стек все непосещенные смежные вершины. Этот процесс повторяется, пока стек не опустеет. Итеративный подход, использующий стек, обеспечивает правильный обход графа в глубину без рекурсии.

**Результат работы программы**

На рисунке 3 приведён результат работы программы №2.

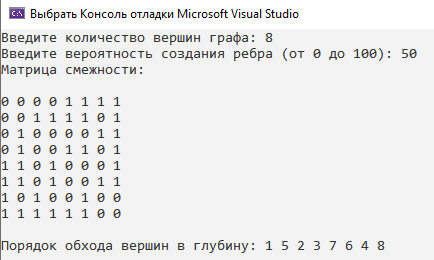


Рисунок 3 – результат работы программы №2

**Выводы**

Был изучен и реализован алгоритм обхода графа в глубину над графами в матричном виде и в виде списков смежности, а также реализован нерекурсивный способ обхода графа в глубину, представленного матрицами смежности.