Лабораторная работа №7. Графы.

Студент Варин Дмитрий Владимирович
Группа ИУ7-36Б
Вариант 3

Цель

Реализовать алгоритмы обработки графовых структур:
поиск различных путей, проверка связности, построение остовых деревьев минимальной стоимости.

Задание

```
Найти самый длинный простой путь в графе
```

Техническое задание

Входные данные

В зависимости от режима работы:

1. *Ввод графа из файла* - число *1*;
2. *Решение задачи поиска самого длинного пути* - вершина для который ищем путь.
3. *Вывод справки* - число *3*
4. *Вывод графа в файл*;
5. *Завершение работы* - число 0.

Выходные данные

1. Максимальные пути до вершин графа от текущей.
2. Файл в расширении *.gv* для визуализации деревьев.
3. Справка.
   Файл хранит список смежности, такой формат входных данных обрабатывается программой.

Функции программы:

1. Ввод графа.
2. Поиск макс. пути от заданной вершины.
3. Вывод графа.
4. Справочная информация.

Возможные аварийные ситуации

1. Некорректный диапазон вершин (не соответствует введнём данным)
   На входе: число, не входящее в диапазон вершин.
   На выходе: сообщение «Неверный ввод, повторите попытку»

2. Не удалось открыть файл.
   На входе: не существующий файл;
   На выходе : сообщение «Не удалось открыть файл;
3. Данные в файле некорректны с точки зрения заданного формата.
   На входе: имя файла;
   На выходе: сообщение «Ошибка чтения из файла»;
4. Решение задачи без введения данных.
   На входе: число *2*;
   На выходе: сообщение «Матрица не заполнена»;
5. Вывод пустой матрицы.
   На входе: число;
   На выходе: сообщение «Матрица не заполнена»;

Структуры данных

В программе используется несколько АТД:

Хранения графа

Граф в программе хранится в виде `матрицы смежности`.

Также в программе используется АТД - Стек, на основе статического массива.
Он (АТД) нужен для обхода вершин, которые можно посетить из текущей.

```
typedef struct {
    int *data;
    int size;
    int capacity;
} stack_t;
```

`int data` - массив целых чисел;
`int size` - текущий размер стека;
`int capacity` - максимальный размер стека;

Алгоритм

1. На экран пользователю выводится меню
2. Пользователь вводит номер команды
3. Выполняется действие согласно номеру команды

Поиск максимального пути осуществляется следующим образом:

1. Заводятся массивы для :

- посещенных вершин;
- расстояний до остальных вершин - результирующий;
- массив для записи родителей вершины -> восстановление пути.
  Также создаётся стек для хранения вершин.

```
stack_t stack;
    stack_init(&stack, n);

    int dist[n];
    int parent[n];
    int visited[n];

for (int i = 0; i < n; ++i) {
        visited[i] = -1;
        parent[i] = -1;
    }
```

2. Вершины сортируются топологически, то есть слева направо, записываются в стек в том же порядке.

```
for (int i = 0; i < n; ++i) {
        if (visited[i] == -1) {
            topological_sort(i, visited, &stack, arr, n);
        }
    }
```

3. Пока стек не пуст, проверяется расстояние до вершин, используя динамическое программирование:
   обходится каждая вершина, которая связана с полученной из стека, и проверяется расстояние для неё от полученной + переход по дуге,
   берётся максимальный, записывается в массив расстояний.

```
while (!stack_is_empty(&stack)) {
        int u = stack_pop(&stack);

        if (dist[u] != INT_MIN) {
            for (int i = 0; i < n; ++i) {
                if (arr[u][i] && arr[u][i] != INT_MIN) {
                    if (dist[i] < dist[u] + arr[u][i]) {
                        dist[i] = dist[u] + arr[u][i];
                        parent[i] = u;
                    }
                }
                }
            }
        }
```

4. Вывод путей от заданной вершины до остальных, а также путь до этих вершин.

```

```

```
for (int i = 0; i < n; ++i) {
        if (dist[i] == INT_MIN) {
            printf("Вершина %d : INF\tПуть: ", i);
        }
        else {
            printf("Вершина %d : %d\tПуть: ", i,dist[i]);
        }
        stack_t st;
        stack_init(&st, n);
        int j = i;

        while (parent[j] != -1) {
            stack_push(&st, j);
            j = parent[j];
        }
        if (dist[i] != INT_MIN) {
            stack_push(&st, s);
        }
        while (!stack_is_empty(&st)) {
            printf("%d ", stack_pop(&st));
        }
        puts("");

    }
```

Выводы по проделанной работе

Временная сложность топологической сортировки `O(V + E)`, где V - кол-во вершин, E - кол-во рёбер. После определения топологического порядка алгоритм обрабатывает все вершины и для каждой вершины запускает цикл для всех смежных вершин.
Общее количество смежных вершин в графе равно `O(E)`.
Таким образом, внутренний цикл выполняется `O(V + E)` раз.Следовательно, общая временная сложность этого алгоритма составляет `O(V + E)`.

Контрольные вопросы

1. Что такое граф?

- Граф – конечное множество вершин и соединяющих их ребер; $G = <V, E>$.
  Если пары E (ребра) имеют направление, то граф называется ориентированным;
  если ребро имеет вес, то граф называется взвешенным.

2. Как представляются графы в памяти?

- С помощью матрицы смежности.

3. Какие операции возможны над графами?

- Обход вершин, поиск различных путей, исключение и включение вершин.

4. Какие способы обхода графов существуют?

- Обход в ширину
- обход в глубину

5. Где используются графовые структуры?

- Графовые структуры могут использоваться в задачах, в которых между элементами могут быть установлены произвольные связи, необязательно иерархические.

6. Какие пути в графе Вы знаете?

- Эйлеров путь(по каждому ребру 1 раз), простой путь(каждая вершина встречается не более 1 раза), сложный путь, гамильтонов путь(по каждой вершине 1 раз).

7. Что такое каркасы графа?

- Каркас графа – дерево, в которое входят все вершины графа, и некоторые (необязательно все) его рёбра.