Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет *Информатика и системы управления (ИУ)*

Кафедра *"Информационная безопасность" ИУ-8*

Отчет по домашнему заданию

Алгоритмы и структуры данных

Цыденов Илья Андреевич
Группа ИУ8-53

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Постановка задачи.

Постройте алгоритм генерации лабиринтов с нужным уровнем проходимости (лабиринт гарантированно должен быть проходим).

1.2. Описание известной задачи

Исходя условия задачи принято решение в качестве известной задачи использовать задачу обхода графа в глубину. Так же существую такие алгоритмы генерации лабиринтов как: Алгоритм Крускала, Алгоритм Прима.

По условию существует задача построения лабиринта. В первоначальном виде задача не годится для реализации на языке программирования. Поэтому необходимо свести эту задачу к уже известной задаче , которую можно будет реализовать на языке программирования.

На вход первоначальной задачи подаются клетки лабиринта (пусть лабиринт будет прямоугольный). Они могут быть стенами и проходами. У каждой клетки есть координата, чтобы клетка имела свое место на поле лабиринта. Теперь поставим каждой клетке-проходу в соответствие вершину графа. Граф должен отображать двумерную координатную плоскость заданного прямоугольника, поэтому вершины необходимо расположить в виде прямоугольника с ровными рядами вершин и каждая вершина будет связана со своими соседями ребрами. На выходе первоначальной задачи получается лабиринт в любом виде , в соответствие ему ставится выход полученный в реализации задачи при помощи обхода графа в глубину, в котором получившееся дерево и есть лабиринт и между всеми не связанными вершинами соответственно находятся стены.

1.3. Описание основных подходов к решению известной задачи

Алгоритм Крускала

Вначале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса.

Алгоритм Прима

На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Таким образом, при выполнении каждого шага алгоритма, высота формируемого дерева увеличивается на 1. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимальной стоимости.

Алгоритм поиска в глубину

Шаг 1. Всем вершинам графа присваивается значение не посещенная. Выбирается первая вершина и помечается как посещенная.

Шаг 2. Для последней помеченной как посещенная вершины выбирается смежная вершина, являющаяся первой помеченной как не посещенная, и ей присваивается значение посещенная. Если таких вершин нет, то берется предыдущая помеченная вершина.

Шаг 3. Повторить шаг 2 до тех пор, пока все вершины не будут помечены как посещенные

Данный алгоритм имеет сложность O(|E| + |V|).

Как можно заметить в алгоритмах Прима и Крускала речь идет о взвешенных графах (где весом будет являться количество клеток, если брать их для генерации лабиринта). Я считаю что это только усложнит алгоритм написанный на языке программирования, потому что лабиринт будет сложнее вписать в прямоугольник, а есть желание генерировать именно такие лабиринты. Поэтому для решения задачи берется алгоритм обхода графа в глубину.

1.4. Описание метода для решения задачи

1. Сделать начальную клетку текущей и отметить ее как посещенную.
2. Пока есть не посещенные клетки
    1. Если текущая клетка имеет не посещенных «соседей»
        1. Поместить текущую клетку в стек
        2. Выбрать случайную клетку из соседних
        3. Поместить в стек ребро между текущей клеткой и выбранной
        4. Сделать выбранную клетку текущей и отметьте ее как посещенную.
    2. Иначе если стек не пуст
        1. Изъять клетку из стека
        2. Сделать ее текущей

Сложность лабиринта будет зависеть от размеров (чем больше поле тем сложнее найти выход). Так же возможно регулировка сложность посредством ограничения количества идущих подряд в одном направлении клеток.

Этот метод выбран для решения задачи, потому что наиболее близко соответствует алгоритму обхода графа в глубину, обоснование которого для решения задачи описано в предыдущих пунктах.

1.5. Описание используемых структур данных

Так как метод решения задачи предполагает наличие стека , то эта структура данных необходима.

Стек - структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека. Притом первым из стека удаляется элемент, который был помещен туда последним, то есть в стеке реализуется стратегия «последним вошел — первым вышел». Описание операций стека:

- push (запись в стек) — операция вставки нового элемента, сложность по времени O(1).
- pop (снятие со стека) — операция удаления нового элемента, сложность по времени O(1).

Существует несколько способов реализации стека:

- одномерным массивом
- односвязным списком

Список вершин и связей между ними, которые являются выходом алгоритма, будут хранится в виде строки для упрощения записи в файл.

1.6 Описание входных и выходных параметров программы

На вход программы , через параметры консоли, подаются:

1. Путь к входной файлу
2. Путь к выходной файлу

В входном файле должны содержаться следующие параметры:

1. size - размер лабиринта (конечный размер будет равен size^2)
2. in_a_row - максимальное количество клеток подряд (уровень сложности лабиринта)
3. doors_count - количество дверей в лабиринте (уровень сложности лабиринта) и соответственно ключей к ним.

Выходной файл может содержать следующие выходные данные:

1. Если программа завершилась описанным кодом ошибки (приложение1 Отчета по тестированию), то содержит только код ошибки
2. В случае успешного завершения работы:
    - список пар координат в формате x1:y1-x2:y2 соответствующие прямым путям в лабиринте.
    - список координат дверей (предпоследняя строка)
    - список координат ключей (последняя строка)

1.7 Описание реального алгоритма

Работу алгоритма можно разделить на три части (три метода):

- Генерация массива лабиринта. Генерирует массив размер size * size структур Cell и заполняет их массивы соседей.
- Выбор входа и выхода лабиринта: Выбирает случайные координаты входа и выхода с условием их нахождения на внешней границы лабиринта.
- Основной метод генерации лабиринта: Следует их вершины в вершину пока у вершины есть соседи. Каждая следующая вершина выбирается случайно из не посещенных соседних и помечается как посещенная, при этом должно соблюдаться условие - количество клеток подряд в одном направлении. Если оно не соблюдается то выбирает другая соседняя клетка. Каждая клетка пройденная алгоритмом помещается в стек. В случае если соседних клеток нет - производится возврат по стеку до ближайшей клетки у которой есть не посещенный сосед. В процессе генерации так же выбираются ключи и двери. Дверь может быть выбрана только после выбора ключа. Весь массив

поделен на отрезки в соответствии с заданным количеством дверей и в каждом отрезке должны присутствовать ключ и дверь.

1.8 Сложность реального алгоритма

Принимая за n общее количество клеток в лабиринте (size * size) получаем:

- Сложность метода генерации массива O(n) умноженное на константу - добавление соседей (от n в целом не зависит) => O(n)
- Сложность метода выбора входа и выхода O(1)
- Сложность метода генерации: метод генерации содержит один цикл , который повторяется n (умноженное на константное время выполнения вложенных операций) + количество возвратов из стека (зависит от размера и случайности , чем больше размер тем меньше зависит от размера, точно меньше чем n, так как фактически в конечном состоянии в стеке будет хотя бы одна вершина) => сложность O(n).

Итоговая сложность по времени O(n+1+n) = O(n).

Сложность по памяти: принимая объем памяти занимаемой структурой Cell за 1 единицу , в процессе генерации алгоритма расходуется n единиц дополнительной памяти на хранение массива ячеек лабиринта + константное количество памяти на дополнительные расчетные переменные и примерно n умноженное на константу - символов в строке вывода. Массивы соседей не занимаю дополнительной памяти так как содержат ссылки на уже размещенные в памяти элементы => O(n).