

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

|  |
| --- |
| **Институт информационных технологий (ИТ)** |
| **Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)** |

**ОТЧЁТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ №6**

**по дисциплине «Структуры и алгоритмы обработки данных»**

Тема: **«**Алгоритмические стратегии. Перебор и методы его сокращения.»

|  |  |  |  |
| --- | --- | --- | --- |
| Отчет представлен к рассмотрению:  Студентка группы ИНБО-01-20 | «10» ноября 2021 г. |  | Тульцова А.Д. |
| (подпись) | | | |
| Преподаватель | «10» ноября 2021 г. |  | Сорокин А.В. |
|  | (подпись) | | |

Москва, 2021 г.

CОДЕРЖАНИЕ

[Цель работы 3](#_Toc86702501)

[Постановка задачи 3](#_Toc86702502)

[Подход к решению. 3](#_Toc86702503)

[Алгоритмы операций на псевдокоде. 5](#_Toc86702504)

[Код программы. 7](#_Toc86702505)

[Тестирование программы. 9](#_Toc86702506)

[Вывод 12](#_Toc86702507)

[Список информационных источников 13](#_Toc86702508)

# Цель работы

Разработка и программная реализация задач с применением метода сокращения числа переборов.

# Постановка задачи

Разработать алгоритм решения задачи варианта с применением метода, указанного в варианте; реализовать программу; оценить количество переборов при решении задачи стратегией «в лоб» – методом «грубой силы»; привести анализ снижения числа переборов при применении метода, указанного в варианте.

**Вариант 8.**

|  |  |
| --- | --- |
| Задача | Метод |
| Черепашке нужно попасть из пункта А в пункт В. Поле движения разбито на квадраты. Известно время движения вверх и вправо в каждой клетке (улицы). На каждом углу она может поворачивать только на север или только на восток. Найти минимальное время, за которое черепашка может попасть из А в B. | Динамическое программирование |

**Дано:**

Размер поля, время движения от одного угла к другому.

**Результат.**

Минимальное время, за которое черепашка может попасть из пункта A в пункт B.

## Подход к решению.

1. Было решено представить поле движения в виде ориентированного графа. Для этого был разработан класс графа, реализующий следующие методы: создание графа посредством применения операций вставки ребра в граф, нахождение величины кратчайшего пути от заданной вершины к другой заданной вершине с помощью метода «грубой силы», а также с помощью метода «Дейкстры» как одного из методов динамического программирования для снижения числа переборов.
2. Разработан консольный пользовательский интерфейс для тестирования работоспособности программы.
3. Разработаны методы обработки графа:
   1. Метод вставки ребра в граф – добавление ребра с заданным весом между двумя заданными вершинами в список смежных вершин графа и в матрицу смежности графа;
   2. Метод перебора вершин («грубой силы») – поиск величин кратчайших путей от каждой вершины графа к другим вершинам графа путём перебора и возврат матрицы кратчайших путей из найденных величин;
   3. Метод «Дейкстры» – поиск величин кратчайших путей до каждой вершины для заданной вершины и возврат массива кратчайших путей из найденных величин;
   4. Метод вызова алгоритма перебора вершин – вызов метода «грубой силы» и возврат величины кратчайшего пути из одной заданной вершины в другую заданную вершину по матрице кратчайших путей;
   5. Метод вызова алгоритма Дейкстры – вызов метода «Дейкстры» и возврат величины кратчайшего пути между двумя заданными вершинами графа по массиву кратчайших путей.
4. Разработан метод приложения для тестирования:
   1. Метод для тестирования нахождения кратчайшего пути – организация нахождения кратчайшего пути от одной заданной вершины к другой заданной вершине с помощью алгоритма перебора вершин и с помощью алгоритма Дейкстры, а также вывода результатов работы алгоритмов в консоль.

## Алгоритмы операций на псевдокоде.

**Метод вставки ребра в граф:**

процедура connect(вершина1, вершина2, вес):

список\_смежных\_вершин[вершина1].добавить\_элемент\_в\_конец

([вершина2, вес]);

матрица\_смежности[вершина1][вершина2] := вес.

**Метод перебора вершин («грубой силы»):**

функция brute\_force():  
 количество\_вершин := длина(матрица\_смежности);  
 min\_dist := матрица\_смежности;  
 количество\_сравнений := 1;  
 для k от 0 до количество\_вершин – 1 выполнять:  
 количество\_сравнений := количество\_сравнений + 1;  
 для i от 0 до количество\_вершин – 1 выполнять:  
 количество\_сравнений := количество\_сравнений + 1;  
 для j от 0 до количество\_вершин – 1 выполнять:  
 количество\_сравнений := количество\_сравнений + 1;  
 если min\_dist[i][j] > min\_dist[i][k] + min\_dist[k][j]:  
 min\_dist[i][j] := min\_dist[i][k] + min\_dist[k][j];  
 количество\_сравнений := количество\_сравнений + 1;  
 количество\_сравнений := количество\_сравнений + 1;  
 количество\_сравнений := количество\_сравнений + 1;  
 вывод(количество\_сравнений);  
 возврат min\_dist.

**Метод вызова алгоритма перебора вершин:**

функция shortest\_path\_brute\_force(вершина1, вершина2):  
 возврат brute\_force()[вершина1][ вершина2].

**Метод «Дейкстры»:**

функция dijkstra(вершина1):  
 nodes\_to\_visit := массив[];  
 nodes\_to\_visit.добавить\_элемент\_в\_конец((0, вершина1));  
 visited := множество();

количество\_сравнений := 1;  
 min\_dist := {i: ∞ для каждого i от 0 до длина(список\_смежных\_вершин) –

1};

количество\_сравнений := количество\_сравнений +

длина(список\_смежных\_вершин);  
 min\_dist[вершина1] := 0;  
 пока длина(nodes\_to\_visit) > 0, выполнять:

вес, текущая\_вершина := минимальный\_элемент(nodes\_to\_visit);  
 nodes\_to\_visit.удалить\_элемент((вес, текущая\_вершина));

количество\_сравнений := количество\_сравнений + 1;  
 если текущая\_вершина в visited:  
 принудительный\_запуск\_следующего\_прохода\_цикла;  
 visited.добавить\_элемент(текущая\_вершина);

количество\_сравнений := количество\_сравнений + 1;  
 для след\_вес, след\_вершина в

список\_смежных\_вершин[текущая\_вершина] выполнять:

количество\_сравнений := количество\_сравнений + 1;  
 если вес + след\_вес < min\_dist[след\_вершина] и след\_вершина не в

visited:  
 min\_dist[след\_вершина] := вес + след\_вес;  
 nodes\_to\_visit. добавить\_элемент\_в\_конец((вес + след\_вес,

след\_вершина));

количество\_сравнений := количество\_сравнений + 1;

количество\_сравнений := количество\_сравнений + 1;

вывод(количество\_сравнений);

возврат min\_dist.

**Метод вызова алгоритма Дейкстры:**

функция shortest\_path\_dijkstra(вершина1, вершина2):  
 возврат dijkstra(вершина1)[вершина2].

## Код программы.

Класс графа:



Рисунок 1 – Класс графа.

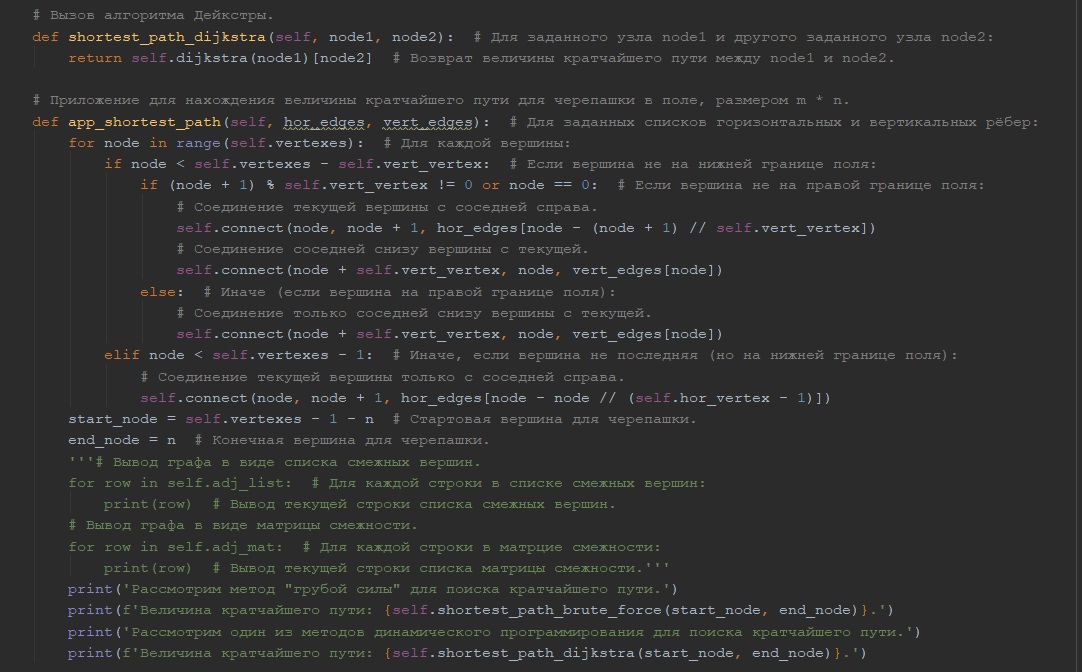


Рисунок 2 – Класс графа (продолжение).

Основная функция для тестирования:

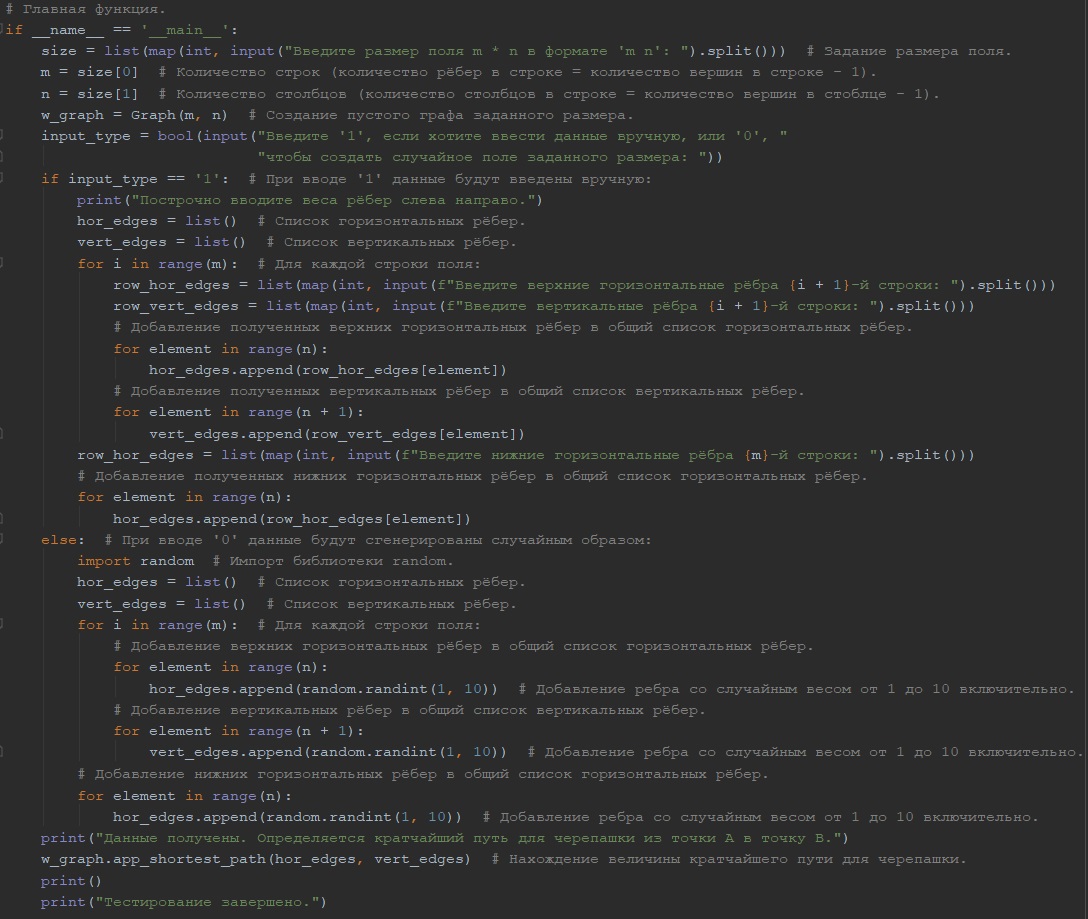


Рисунок 3 – Основная функция.

## Тестирование программы.

**Тестирование программы на заданном поле движения.**

Для тестирования было выбрано поле движения черепашки размером 3\*3 (см. Рисунок 4). Стрелками выделен кратчайший путь (минимальное затрачиваемое время) для черепашки из точки A в точку B; его величина – 21.

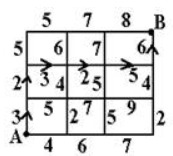


Рисунок 4 – Поле движения черепашки.

Итак, было запущено тестирование программы. Сначала был указан размер поля движения черепашки, затем построчно вводились рёбра поля движения (см. Рисунок 5) после выбора формата ввода вручную.

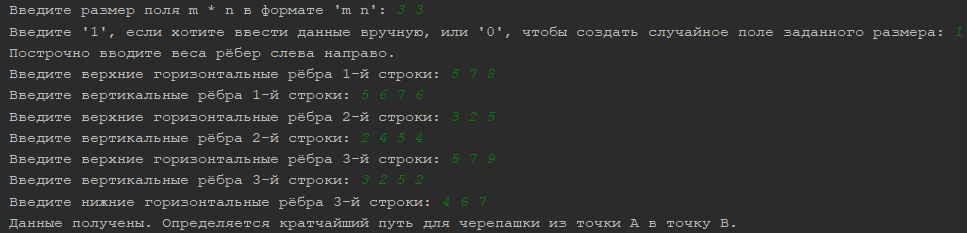


Рисунок 5 – Ввод данных.

После завершения ввода данных программа осуществляет поиск величины кратчайшего пути двумя способами: методом «грубой силы» (алгоритмом перебора вершин) и методом динамического программирования (алгоритмом Дейкстры) (см. Рисунок 6). Чтобы сделать анализ результатов более удобным, в процессе выполнения алгоритмов также подсчитывалось количество сравнений и замерялось затраченное время.

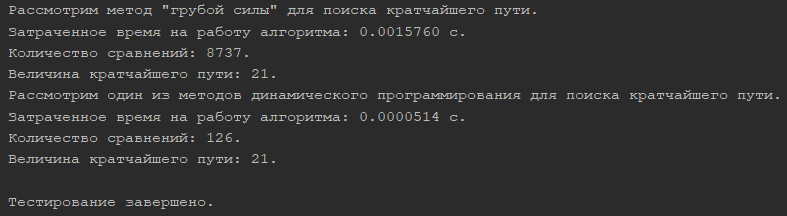


Рисунок 6 – Полученные результаты.

В результате для данного примера метод «грубой силы» уступает выбранному методу динамического программирования по скорости примерно в 3 раза, а количество сравнений для метода «грубой силы» – 8737. Алгоритм Дейкстры способен уменьшить затраченное время на поиск величины кратчайшего пути и количество сравнений (в ~69 раз) путём снижения общего количества переборов за счёт использования принципа динамического программирования – разбиения общей задачи на более простые подзадачи, а именно – следуя методу «Дейкстры», нет необходимости осуществлять проход по каждой вершине более одного раза; также данный алгоритм постоянно осуществляет движение к наиболее «выгодной» вершине, на каждом шаге выбирая наиболее короткий путь из возможных. Таким образом сложность алгоритма значительно снижается, как и время его выполнения с количеством выполняемых операций сравнения.

**Тестирование программы на сгенерированном случайным образом поле движения.**

Теперь для тестирования было выбрано поле движения черепашки размером 14\*16. Для упрощения ввода оно было заполнено автоматически: веса рёбер были сгенерированы случайным образом в пределах от 1 до 10 включительно. Благодаря реализованному интерфейсу, после запуска программы достаточно было задать размер поля и указать в качестве формата ввода заполнение поля случайным образом (см. Рисунок 7).

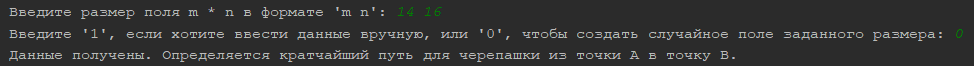


Рисунок 7 – Ввод данных.

Далее программа выполнила для созданного поля движения поиск величины кратчайшего пути двумя способами: методом «грубой силы» (алгоритмом перебора вершин) и методом динамического программирования (алгоритмом Дейкстры) (см. Рисунок 8).

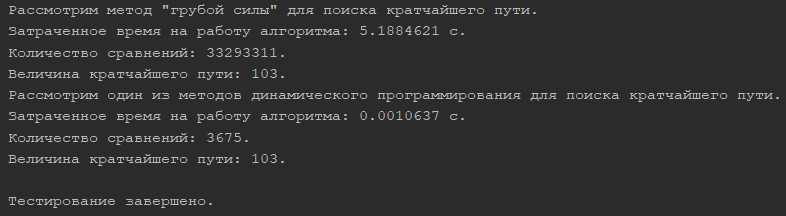


Рисунок 8 – Полученные результаты.

Как видно по результатам – и тот, и другой методы так же вывели одинаковый ответ. Однако теперь метод «грубой силы» уступает выбранному методу динамического программирования (алгоритму Дейкстры) по скорости примерно в 4878 раз, а количество сравнений для метода «грубой силы» уже более 33 млн., что делает его очень неэффективным и по памяти, и по времени. При этом с применением методов динамического программирования (в данном случае – алгоритма Дейкстры) поставленная задача решается примерно за тысячную долю секунды, а количество сравнений относительно небольшое (~ в 10000 раз меньше в сравнении с решением методом «грубой силы»).

Таким образом, методы динамического программирования позволяют значительно упростить решение поставленной задачи, разбивая её на более простые подзадачи; посредством снижения количества переборов уменьшается затрачиваемое время на выполнения алгоритма, а также количество выполняемых операций сравнения.

# Вывод

В ходе работы был разработан алгоритм решения задачи поиска кратчайшего пути с применением метода динамического программирования; реализована программа; приведена оценка количества переборов при решении задачи стратегией «в лоб» – методом «грубой силы»; произведён анализ снижения числа переборов при применении одного из методов динамического программирования.

# Список информационных источников

1. Лекции по дисциплине «Структуры и алгоритмы обработки данных» / Л. А. Скворцова, МИРЭА – Российский технологический университет, 2021.