Оборудование:
Intel(R) Core(TM) M-5Y10c CPU @ 0.80GHz
Оперативная память 8 ГБ
ОС - Windows 10

Задача:
Проанализировать алгоритмы на графах. Установить особенности алгоритмов.

Dfs

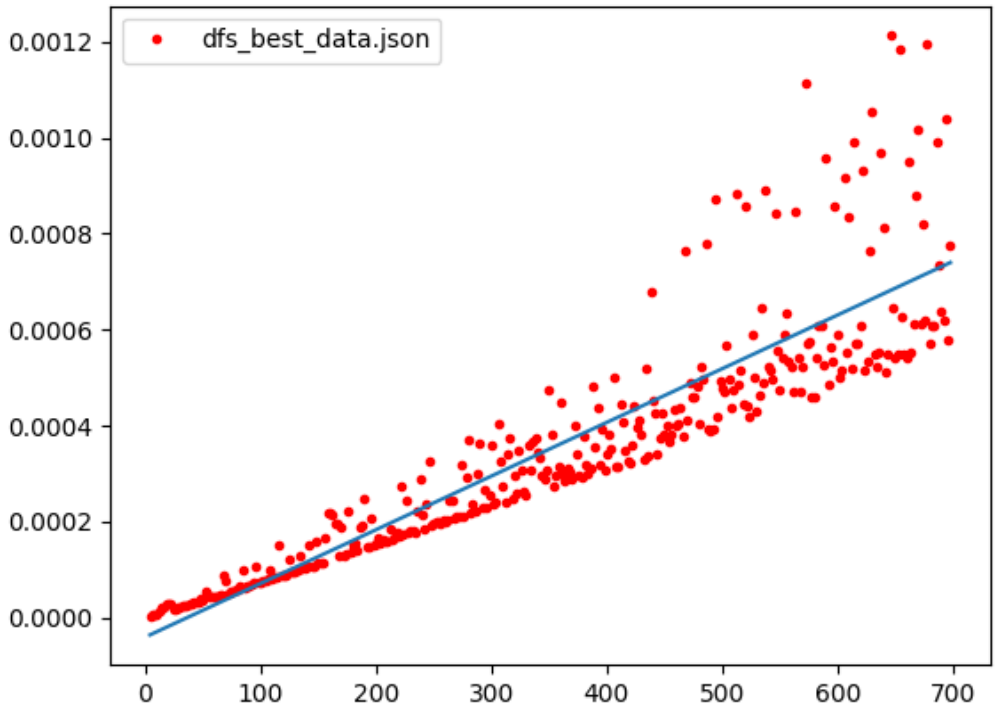
На вход принимает граф, начальную и конечную вершину. Возвращает массив - последовательность вершин, по которым можно пройти из начальной вершины до конечной.

Работает рекурсивно. Из вершины, в которой сейчас находится, пытается перейти в другие смежные вершины, в которых еще не был. Также ведется запись предков для вершин, чтобы в конце восстановить путь. Когда алгоритм доходит до конечной вершины, он восстанавливает путь и прекращается.

Время работы
Алгоритм посетит не более n вершин, и пройдет по ребру не более одного раза.
Следовательно O(n + m)

Затраты памяти
Глубина рекурсии не превышает общего числа вызовов функции Dfs — числа вершин.
Следовательно объем памяти O(n).

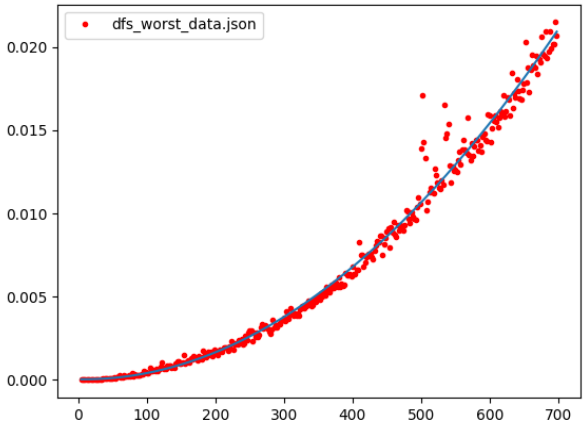
Генерация данных
Лучшие данные
Из времени работы алгоритма следует, что мы хотим минимизировать число вершин и ребер. Число вершин мы минимизировать не можем, тогда минимизируем число ребер.
Для тестирования лучшего результата возьмем графы, являющиеся цепью. Время работы будет O(n).
Из графика и линейной аппроксимации видно, что функция растет линейно.



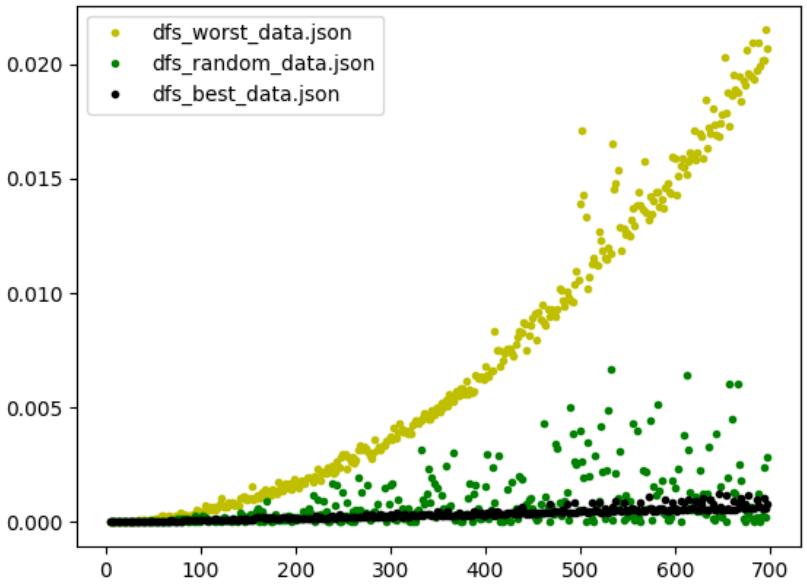
Худшие данные

Из времени работы алгоритма следует, что мы хотим максимизировать число вершин и ребер. Число вершин мы максимизировать не можем, тогда максимизируем число ребер. Для тестирования худшего результата возьмем графы, являющиеся полными. Их время работы будет O(n**2).

Из графика и квадратичной аппроксимации видно, что функция растет квадратично.



Случайные данные
За основу было взято дерево. Добавлялось неизвестное число ребер.



На графике видно, что в большинстве случаев алгоритм вне зависимости от размера графа работал за время, близкое к линейному и редко за время, близкое к квадратичному. Это можно объяснить тем, что в неориентированном графе либо будет примерно линейное количество ребер и тогда алгоритм будет работать линейное время, а если число ребер близко к квадратичному, то скорее всего быстро найдется путь, содержащий маленькое число ребер и его поиск потребует также просмотра маленького числа ребер.

Bfs

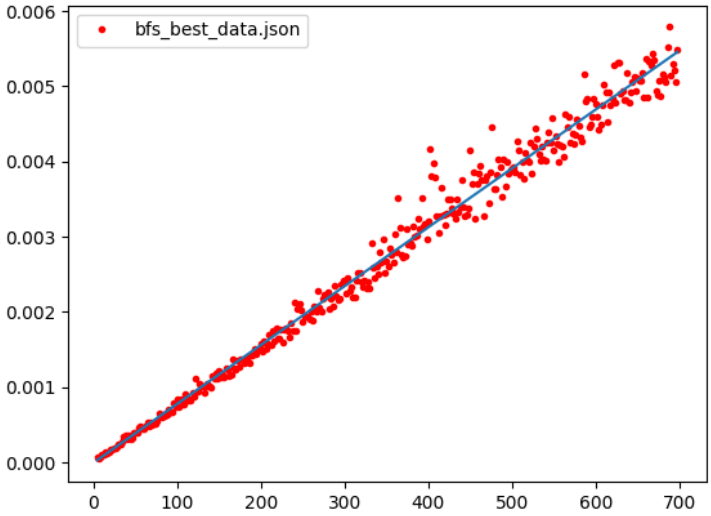
На вход принимает граф, начальную и конечную вершину. Возвращает массив - последовательность вершин, по которым можно пройти из начальной вершины до конечной, причем путь будет кратчайшим.

В очередь помещается начальная вершина, затем на каждом шаге из очереди извлекается, добавленная раньше всех вершина, из нее просматриваются все рёбра, если ребро ведет в вершину, которая еще не добавлялась в очередь, то она добавляется в очередь. Как только метод доходит до конечной вершины, он восстанавливает путь и заканчивает работу..

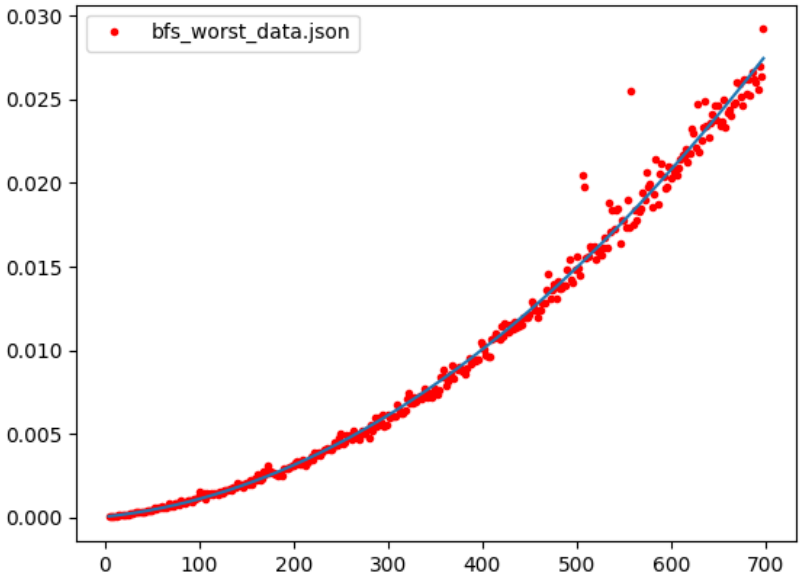
Время работы
Алгоритм посетит не более n вершин, и пройдет по ребру не более одного раза.
Следовательно O(n + m)

Затраты памяти
Максимальное число вершин, одновременно хранящихся в очереди — n, то есть, максимальный объем используемой памяти — O(n).

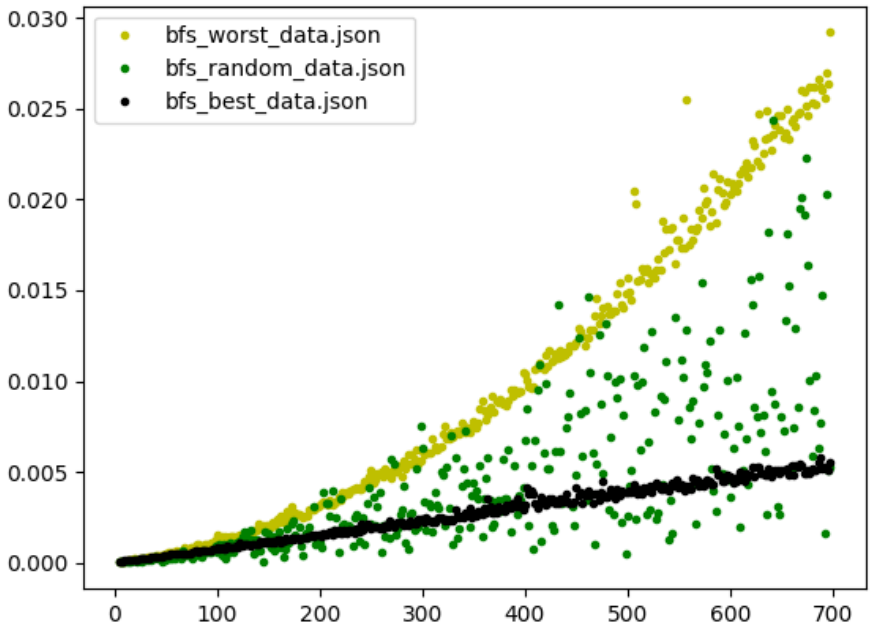
Генерация данных
Лучшие данные
Из времени работы алгоритма следует, что мы хотим минимизировать число вершин и ребер. Число вершин мы минимизировать не можем, тогда минимизируем число ребер. Для тестирования лучшего результата возьмем графы, являющиеся цепью. Их время работы будет O(n).
Из графика и линейной аппроксимации видно, что функция растет линейно.



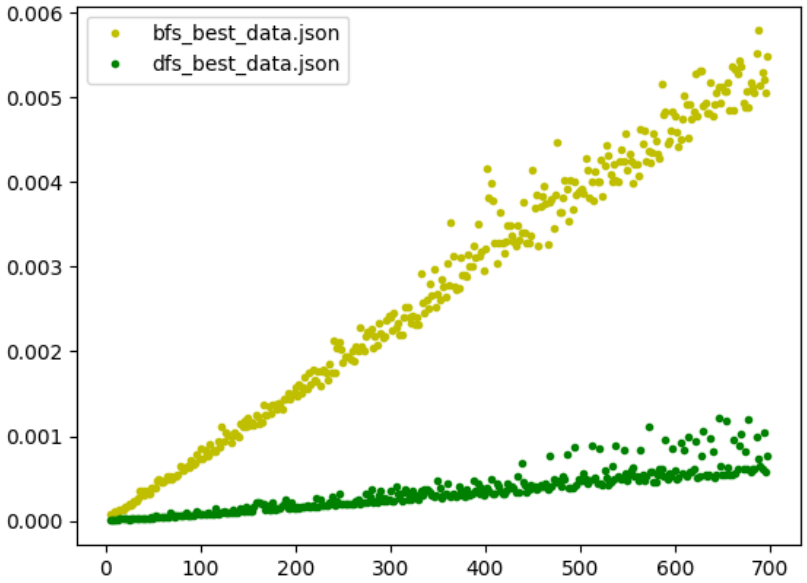
Худшие данные
Из времени работы алгоритма следует, что мы хотим максимизировать число вершин и ребер. Число вершин мы максимизировать не можем, тогда максимизируем число ребер. Для тестирования худшего результата возьмем графы, являющиеся полными. Их время работы будет O(n**2).
Из графика и квадратичной аппроксимации видно, что функция растет квадратично



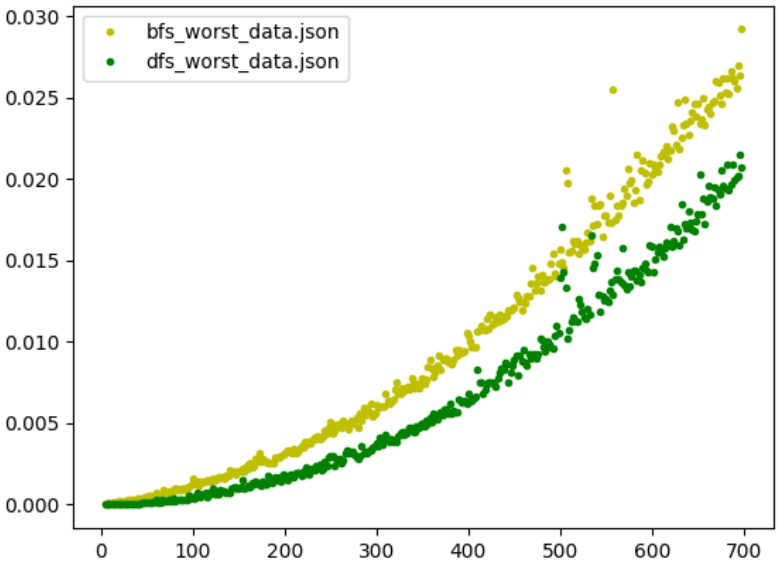
Случайные данные
За основу было взято дерево. Добавлялось неизвестное число ребер.
Видно, что в большинстве случаев время линейное.



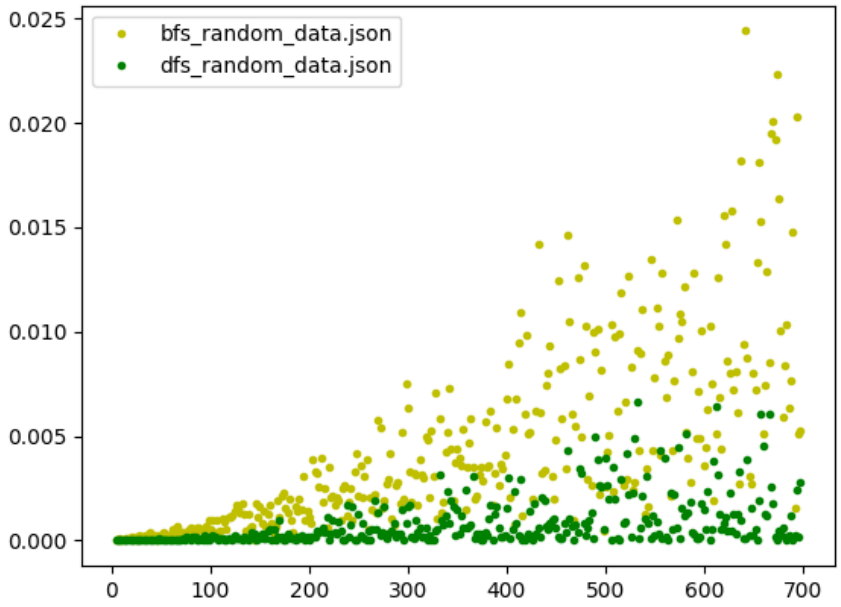
Dfs и Bfs



У поиска в ширину коэффициент k во много раз больше, чем у лучшего случая поиска в глубину.



В среднем на худших данных поиск в ширину работает примерно в 1.5 раза медленнее поиска в глубину. Возможно, это связано с тем, что обход в ширину гарантированно посмотрит все ребра из каждой рассматриваемой вершины, а обход в глубину может случайно найти нужное ребро, не рассматривая до конца остальные, и сразу перейти по нему в следующую вершину.



На больших случайных данных поиск в ширину работает намного медленнее поиска в глубину.

Итог:
Во всех случаях обход в ширину в среднем работает медленнее.

Dijkstra

На вход принимает граф, начальную вершину. Возвращает массив, в котором находятся минимальные расстояния от начальной вершины до всех остальных.

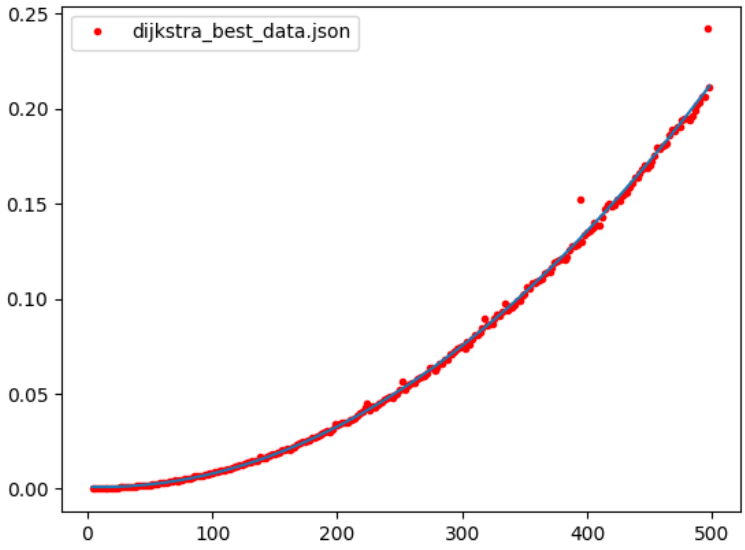
Каждой вершине из *n* сопоставим метку — минимальное известное расстояние от этой вершины до первоначальной..Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки.Работа алгоритма завершается, когда все вершины посещены.

Время работы
Сложность алгоритма Дейкстры складывается из двух операций: время нахождения вершины с наименьшей величиной расстояния, и время совершения релаксации. Эти операции потребуют O(n), O(1) времени.Первая операция выполняется O(n) раз, вторая O(m). Итоговая асимптотика O(n**2+m).

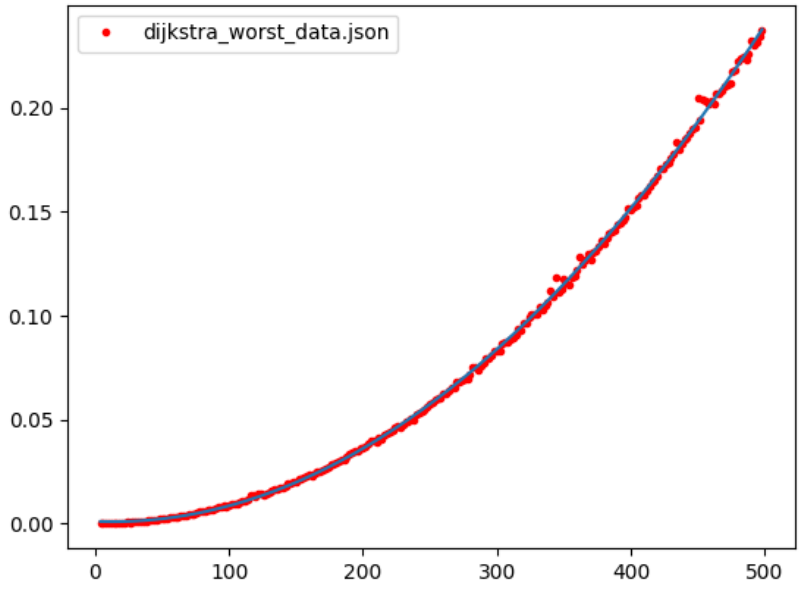
Затраты памяти

Во время алгоритма мы храним матрицу n**2. Следовательно мы используем O(n**2) памяти.

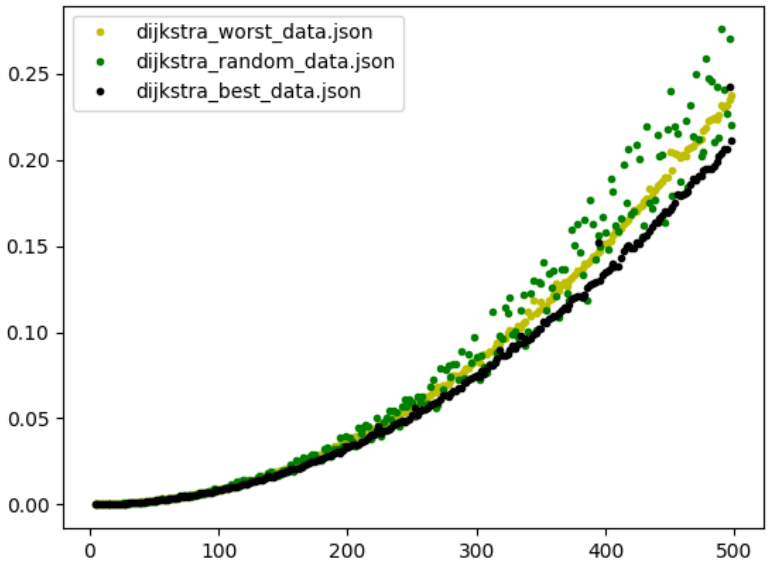
Генерация данных
Лучшие данные
Из времени работы алгоритма следует, что мы хотим минимизировать число вершин и ребер. Число вершин мы минимизировать не можем, тогда минимизируем число ребер. Для тестирования лучшего результата возьмем графы, являющиеся цепью. Их время работы будет O(n**2).
Из графика и квадратичной аппроксимации видно, что функция растет квадратично.



Худшие данные
Из времени работы алгоритма следует, что мы хотим максимизировать число вершин и ребер. Число вершин мы максимизировать не можем, тогда максимизировать число ребер. Для тестирования худшего результата возьмем графы, являющиеся полным. Их время работы будет O(n**2 + m).



Случайные данные
За основу был взят связный граф. Добавлялось неизвестное число ребер.



Floyd
На вход принимает граф. Возвращает матрицу n на n, в которой находятся минимальные расстояния между двумя любыми вершинами.

Каждой вершине из *n* сопоставим метку — минимальное известное расстояние от этой вершины до первоначальной..Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки.Работа алгоритма завершается, когда все вершины посещены.

Время работы
Сложность алгоритма Дейкстры складывается из двух операций: время нахождения вершины с наименьшей величиной расстояния, и время совершения релаксации. Эти операции потребуют O(n), O(1) времени.Первая операция выполняется O(n) раз, вторая O(m). Итоговая асимптотика O(n**2+m).

Обозначим длину кратчайшего пути между вершинами u и v, содержащего, помимо u и v только вершины из множества {1..i} как d(i)(uv) d(0)(uv) = w(uv). На каждом шаге алгоритма, мы будем брать очередную вершину (пусть её номер — i) и для всех пар вершин u и v вычислять d(i)(uv) = min(d(i−1)(uv),d(i−1)(ui)+d(i−1)(iv)).То есть, если кратчайший путь из u в v, содержащий только вершины из множества {1..i}, проходит через вершину i, то кратчайшим путем из u в v является кратчайший путь из u в i, объединенный с кратчайшим путем из i в v. В противном случае, когда этот путь не содержит вершины i, кратчайший путь из u в v, содержащий только вершины из множества {1..i} является кратчайшим путем из u в v, содержащим только вершины из множества {1..i−1}.

Время работы
O(n**3), так как мы всегда проходим 3 вложенных цикла длинной n

Затраты памяти
Во время алгоритма мы храним матрицу n**2. Следовательно мы используем O(n**2) памяти.

Для данных с одинаковым количеством вершин время работы алгоритма не изменится.