# Клещенко В

# ЛР #2: Структуры данных и динамическое программирование

## Цель

Познакомить студента с основами структур данных и методикам динамического программирования.

## Задача

Поиск оптимального (кратчайшего, быстрейшего или самого дешевого) пути, проходящего через промежуточный пункты по одному разу и возвращающегося в исходную точку. К примеру, нахождение наиболее выгодного маршрута, позволяющего коммивояжеру посетить со своим товаром определенные города по одному разу и вернуться обратно. Мерой выгодности маршрута может быть минимальная длина пути.

Задать перечень точек с координатами X и Y. Пример:

* Точка 1, X1, Y1
* Точка 2, X2, Y2
* Точка 3, X3, Y3
* Точка ...

В качестве отправной и конечной точки брать первую введенную точку.

1. **Спроектировать оптимальную структуру для решения задачи с точки зрения затрат памяти. Реализовать на языке C++ или Python**
2. **Спроектировать оптимальный алгоритм решения задачи с использованием технологий динамического программирования. Оценить вычислительные и емкостные затраты. Реализовать на языке C++ или Python**
3. **Всё, что было сделано в шагах 1-2, сохранить в репозиторий (+ отчет по данной ЛР в папку doc).**

## Решение

У данной задачи может быть много решений как точных, так и приближенных. При выполнении лабораторной работы был выбран метод использующий подход динамического программирования для нахождения точного решения задачи. Динамичность заключалась в использовании на каждом шаге результат предыдущих вычислений и обновления всего решения в ходе алгоритма.

1. **Спроектировать оптимальную структуру для решения задачи с точки зрения затрат памяти. Реализовать на языке C++ или Python**

В качестве структуры для решения задачи был выбран двумерный массив, поскольку он обеспечивал бы О(1) для вставки, а это полезно учитывая динамичность подхода.

1. **Спроектировать оптимальный алгоритм решения задачи с использованием технологий динамического программирования. Оценить вычислительные и емкостные затраты. Реализовать на языке C++ или Python**

Сам алгоритм будет описан ниже, но мотивация для его применения, следующая: очевидным решением было бы перебрать всевозможные варианты путей, по которым можно было бы объехать n городов по разу и вернуться в тот же – временная сложность такого алгоритма будет n! – количество перестановок. Существенный прирост в скорости обеспечило бы учет следующего обстоятельства: когда перебираются все пути то для части пути одни и те же значения перебираются много раз, например:

1 2 3 | 4 9 8 7 --- 1 2 3 | 4 9 7 8 --- …

3 2 1 | 4 9 8 7 --- 3 2 1 | 4 9 7 8 --- …

…

Чтобы такого не происходило, можно ввести функцию, которая для определенного поднабора городов хранила кратчайший путь через них (4 9 7 8):

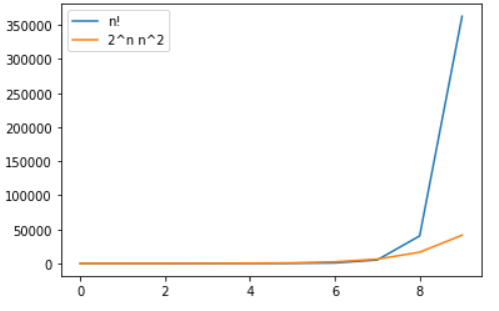
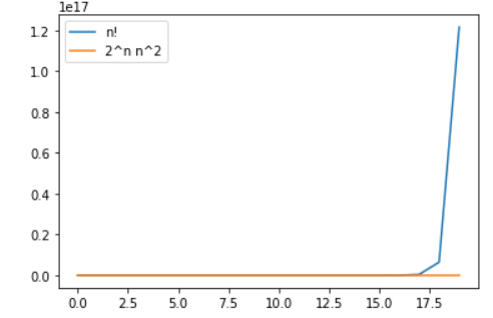
C[S, j] – длина кратчайшего пути, начинающегося в 1 и проходящего по всех городам из множества S и приводящего к j, тогда пересчет на каждом шаге:

C[S, j] = min { C[S \ {j}, i] + Dij } – функция Беллмана

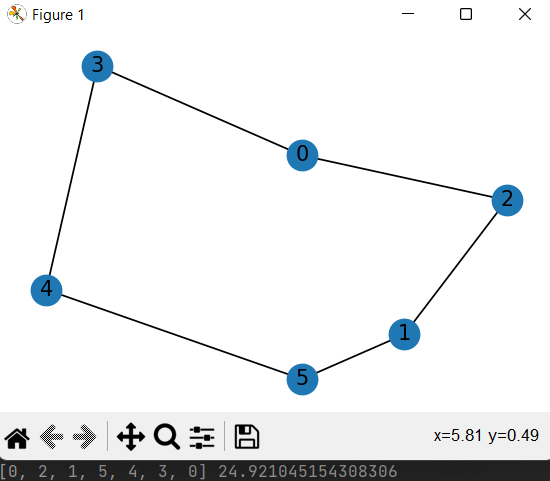
Тогда оценим сложность этого алгоритма: теперь перебираются не все n! перестановок, а только всевозможные подмножества, их - 2^n и в каждом подмножестве нужно перебрать для каждого j все i (см. формулу), итого 2^n n^2.

Затраты по памяти, опять же из формулы 2^n n.

Кажется, что алгоритм работает не сильно лучше и тратит очень много памяти, но на самом деле это существенный прирост к скорости:



В качестве визуализации использовался граф, результат работы:



## Вывод

В результате выполнения лабораторной работы был реализован алгоритм для решения задачи коммивояжера. Показано, что оптимизация решения на каждом шаге, то есть подход динамического программирования дает существенный прирост к скорости работы программы.