Основная концепция:  
Распределятор (контроллер) не знает куда отправлять задачи, т.е. не знает о выч.узлах, но узлы знают на каком порту находится распределятор, чтобы к нему постучать и взять работу

Client (вычислительный узел) просто получает работу и с помощью класса Executor запускает работу на выполнение, т.е. подсчет стоимости каждого пути и нахождения из подсчитанных, пути с минимальной стоимостью и длиной n

Server (UI) - прокидывается файл с матрицей и длина пути (т.е. количество городов, которое должно быть посещено)

После чего в методе start с помощью класса для формирования работ JobsFactory передаем исходные данные (матрицу целиком и длину пути)

Controller (распределятор задач) - просто передает узлу (Клиенту) сформированную с помощью класса JobsFactory задачу

Класс JobsFactory. Методы

Концепция: Подзадачей является перебор всех возможных путей.

1. next () - формирует все возможные переборы для каждого из индексов городов, типа куда из какого города можем пойти в разных вариациях. Индекс городов начинается с нуля.
2. CompHeadSize() - расчет размера подзадачи (Почему 9? Проведены эксперименты и время выполнения полных переборов зависит от количества городов, которые должны быть посещены. Для 8 городов и длине пути = 7 - меньше секунды, для 9 городов и длине пути равной 8 - примерно 1 секунда, для 10 городов и длине пути = 8 - 9 с копейками секунд, а для 10 городов и длине пути = 9 - уже 15-20 секунд.

Было принято решение взять размер подзадачи (диапазон) равной ¼ от длины маршрута. Например, ¼ от 9 это 2,25 города, округляем и получаем 2 города. Почему именно ¼? Есть точка старта, есть точка выхода - т.е. 2 точки - 2 города. То есть размер подзадачи (по сколько городов брать и проверять) формально равен 2 городам. Нам это в принципе подходит)

1. Ну и метод GetJob, который формирует Job (работу, т.е. подзадачу) на основе целиком матрицы, размера пути и сформированного массива переборов.

Класс JobExecutor

Из полученной работы (подзадачи) считаем стоимости по каждому пути и выбираем из них наименьший, который сохраняем в результат, который потом будем выдавать на ui. Реализован поиск в глубину. (Надеюсь Ермаков спрашивать про это не будет)