**Алгоритм**

Сначала необходимо длины дорог, соединяющих города представить в виде следующей таблицы:

В нашем примере у нас 4 города и в таблице указано расстояние от каждого города к 3-м другим, в зависимости от направления движения (т.к. некоторые ж/д пути могут быть с односторонним движением и т.д.).

Расстояние от города к этому же городу обозначено буквой M. Также используется знак бесконечности. Это сделано для того, чтобы данный отрезок путь был условно принят за бесконечно длинный. Тогда не будет смысла выбрать движение от 1-ого города к 1-му, от 2-ого ко 2-му, и т.п. в качестве отрезка маршрута.

Находим минимальное значение в каждой строке (di) и выписываем его в отдельный столбец.

Производим редукцию строк – из каждого элемента в строке вычитаем соответствующее значение найденного минимума (di).



В итоге в каждой строке будет хотя бы одна нулевая клетка.

Далее находим минимальные значения в каждом столбце (dj). Эти минимумы выписываем в отдельную строку.



Вычитаем из каждого элемента матрицы соответствующее ему dj.

В итоге в каждом столбце будет хотя бы одна нулевая клетка.

Для каждой нулевой клетки получившейся преобразованной матрицы находим «оценку». Ею будет сумма минимального элемента по строке и минимального элемента по столбцу, в которых размещена данная нулевая клетка. Сама она при этом не учитывается. Найденные ранее di и dj не учитываются. Полученную оценку записываем рядом с нулем, в скобках.  


И так по всем нулевым клеткам:



Выбираем нулевую клетку с наибольшей оценкой. Заменяем ее на «М». Мы нашли один из отрезков пути. Выписываем его (от какого города к какому движемся, в нашем примере от 4-ого к 2-му).  


Ту строку и тот столбец, где образовалось две «М» полностью вычеркиваем. В клетку, соответствующую обратному пути ставим еще одну букву «М» (т.к. мы уже не будем возвращаться обратно).



Если мы еще не нашли все отрезки пути, то возвращаемся ко 2-му пункту и вновь ищем минимумы по строкам и столбцам, проводим их редукцию, считаем оценки нулевых клеток и т.д. Если все отрезки пути найдены (или найдены еще не все отрезки, но оставшаяся часть пути очевидна) – переходим к пункту 9.

Найдя все отрезки пути, остается только соединить их между собой и рассчитать общую длину пути (стоимость поездки по этому маршруту, затраченное время и т.д.). Длины дорог, соединяющих города, берем из самой первой таблицы с исходными данными. В нашем примере маршрут получился следующий: 4 → 2 → 3 → 1 → 4. Общая длина пути: L = 30.

Применение задачи коммивояжера на практике довольно обширно. В частности, ее можно использовать для поиска кратчайшего маршрута при гастролях эстрадной группы по городам, нахождения последовательности технологических операций обеспечивающей наименьшее время выполнения всего производственного цикла и пр.

**Практика**

Постановка задачи:

Построение оптимального маршрута для экскурсии (обхода) объектов одного или заданных типов данных.

В чем смысловая нагрузка? – создание рабочего прототипа приложения, реализующего данный функционал.

Как это сделать? Разберем этапы.

Что нам понадобится?

1. Исходные данные (координаты объектов, путь между которыми мы будем искать)
2. Алгоритм, решающий задачу коммивояжера
3. Результат в виде карты с маршрутом

Все просто. Осталось придумать, как это правильно реализовать. Живем мы в России – делаем выводы, что пользоваться лучше Российскими компаниями, предоставляющими нужную информацию.

Для программирования выбран модный нынче язык – Go. Принципиальной разницы нет, просто необходима была практика.

Что дальше? **Исходные данные!**

Пользуемся API (Application Programming Interface) Yandex.Maps. С его помощью получаем исходные данные:

1. Названия объектов
2. Географические координаты объектов

Как в целом работает программа:

1. Получаем данные от API для всех запросов в списке, если они удовлетворяют заранее заданным параметрам.
2. Объединяем эти запросы.
3. Заполняем пары элементов. Это необходимо для правильной исходной нумерации пар и легкого заполнения исходной матрицы весов.
4. Заполняем матрицу весов.
5. Дальше по алгоритму
   1. Редуцируем строки
   2. Редуцируем столбцы
   3. Читаем коэффициенты
   4. Находим максимальный и добавляем его в список
   5. Удаляем строку и столбец
   6. Пересчитываем минимальный путь и сравниваем его с порогом
6. Как только найдены все точки – сортируем элементы в списке для более простого построения маршрута (все стартовые точки будут соответствовать позициям в массиве).
7. Попадаем в модуль построения карт.

Что уж лукавить, я решил сэкономить время и воспользовался теми же Yandex.Maps для построения своих маршрутов.

Правильно передавая параметры в API я получил изображение карты, центрированное по всем точкам, которые я сам пометил на карте с нанесенными на неё метками, пронумерованными в порядке посещения.

Задачи в курсовой работе показывают, что проект работает и строит оптимальные маршруты. Проверено это было несколькими способами.

1. Пример с церквями Москвы (программа посещает сначала церкви в центре города, а затем попадает на кольцевые).
2. Пример с магазинами Кургана (программа строит маршрут не пересекая пути, потому что расстояния между магазинами по окружности меньше, чем между напротив расположенными).
3. Пример с аэропортами (Из города в город приложение попадает только один раз).