В. Ф. Очков,
Национальный исследовательский университет МЭИ, Москва

ПОЙТИ ТУДА, ЗНАЯ КУДА

Аннотация
В статье рассмотрена программа, позволяющая решить задачу коммивояжера.

Ключевые слова: задача коммивояжера, Mathcad.

Контактная информация

Очков Валерий Федорович, доктор тех. наук, профессор, Национальный исследовательский университет МЭИ, Москва; *адрес:* 111250, г. Москва, Красноказарменная ул., д. 14; *телефон:* (495) 362-71-71; *e-mail:* ochkov@twt.mpei.ac.ru

V. F. Ochkov,
National Research University MPEI, Moscow

GO THERE, KNOWING WHERE

Abstract
The article describes a program that allows to solve the traveling salesman problem.

Keywords: traveling salesman problem, Mathcad.

Сообразив, куда прежде, куда после ехать, чтоб не возвращаться, Нехлюдов прежде всего направился в сенат.

Л. Н. Толстой «Воскресенье»

Каждый из нас, отправляясь в путь по мало-мальски сложному маршруту, должен «сообразить, куда прежде ехать, куда после, чтоб не возвращаться». Многие из нас, если и не работали (подрабатывали) курьером, то получали заказанный товар из рук этого человека. Курьеру же, «направляясь в путь», обязательно нужно сообразить, «куда прежде, куда позже ехать», чтобы оптимизировать маршрут, т. е. попытаться решить **задачу коммивояжера** (странствующего торговца, коробейника): найти маршрут обхода n городов, побывав в каждом городе один раз, вернувшись в исходную точку и сведя к минимуму путь (время в пути, расходы на дорогу и т. д.).

Разработано множество алгоритмов различной сложности для решения этой задачи. Сначала ее несколько упрощают: берут n точек на плоскости с координатами, хранящимися в векторах X и Y, и считают, что путь из i-й точки ($i = 1..n$) в j-ю точку ($j = 1..n$; $i \neq j$) лежит по прямой и может быть пройден в двух направлениях (от i к j и от j к i).

Один из простейших алгоритмов решения задачи коммивояжера состоит в следующем. Берется случайная точка на плоскости, которая будет первым элементом вектора *путь*. Затем ищется ближайшая точка от этой стартовой точки. Она становится второй точкой вектора *путь*. Далее эта операция повторяется до тех пор, пока не будут перебраны все точки. В этом переборе добавляется еще одно условие: нельзя «идти» в город, т. е. в точку, где мы уже были.

На рисунке 1 показана программа, написанная в среде Mathcad Prime, реализующая этот **алгоритм ближайшего соседа**.

В программе, показанной на рисунке 1, сначала заполняется квадратная матрица M, хранящая расстояние от точки i до точки j. Если $i = j$ (диагональ матрицы), то этот элемент матрицы будет хранить значение бесконечности. Это будет знаком того, что по данному маршруту ходить не нужно. Бесконечность будет записываться в элементы матрицы $M_{i,j}$ и $M_{j,i}$, если от точки i к точке j (или наоборот) путь уже пройден. Но еще раньше нужно заполнить векторы X и Y (координаты точек на плоскости) и скаляр *старт* (номер первой точки маршрута).

Программа, реализующая метод ближайшего соседа (рис. 1), получилась такая компактная благодаря двум встроенным функциям Mathcad: *min* и *match*. Первая ищет минимальное значение в векторе или матрице, вторая определяет координаты элемента вектора или матрицы с заданным значением*.

В программе в цикле с параметром i (перебор точек около очередной точки) составляется вектор S, который хранит расстояние от этой i-й точки до всех остальных j-х точек. Далее с помощью функций *min* и *match* (см. выше) определяется точка (очередной элемент исходного вектора *путь*), ближайшая к нашей точке и в которой мы еще не были.

* Очень часто школьникам дают задание написать на Паскале такие программы методом перебора всех элементов вектора или матрицы.

```
путь := | M ← | for i ∈ 1..n
        |       |   for j ∈ 1..n
        |       |     M[i,j] ← if(i = j, ∞, sqrt((X_i − X_j)^2 + (Y_i − Y_j)^2))
        |       | M
        | путь_1 ← старт
        | for i ∈ 2..n
        |   for j ∈ 1..n
        |     s_j ← M[путь_(i−1), j]
        |   путь_i ← match(min(s), s)_1
        |   for j ∈ 1..i−1
        |     [M[путь_i, путь_j] ← ∞  M[путь_j, путь_i] ← ∞]
        | путь
```

Рис. 1. Программа «Ближайший сосед»

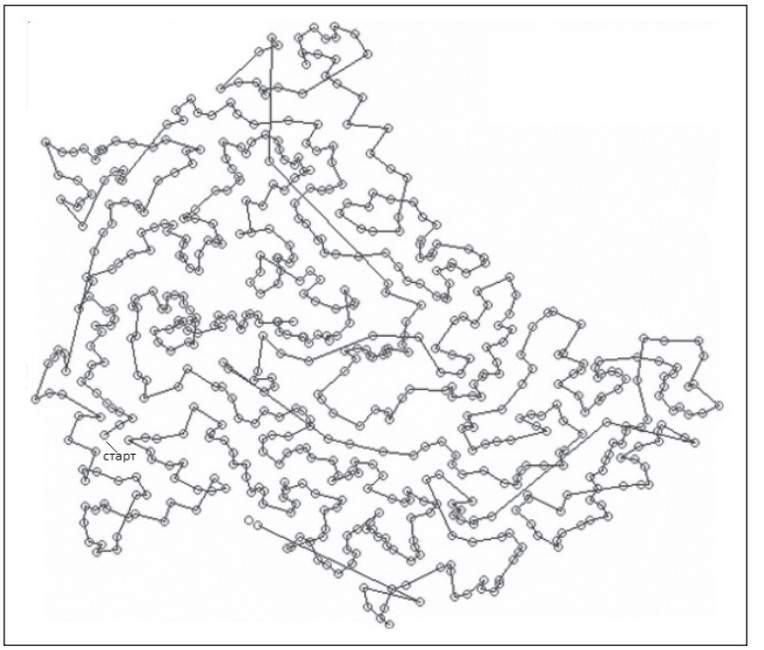

Рис. 2. Путь коммивояжера по Люксембургу

Все очень просто, но... «иная простота хуже воровства».

На рисунке 2 показан предпоследний кадр анимации обхода городов Люксембурга (одной из самых маленьких стран Европы) по алгоритму ближайшего соседа. Этот алгоритм относится к группе *жадных* алгоритмов: бросаясь к ближайшему соседу, мы делаем петли и пропускаем точки, в которые так или иначе придется «зайти», делая при этом броски из одного в другой конец «страны».

Мы намеренно не приводим здесь более сложные или оптимальные алгоритмы решения задачи коммивояжера. Предлагаем педагогам, желающим увлечь своих учеников интересной и занимательной задачей, дать школьникам представленный выше материал с тем, чтобы заинтересовать их и призвать найти в книгах или Интернете другие алгоритмы решения задачи коммивояжера и реализовать их на уроках информатики.