ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

Департамент прикладной математики

ОТЧЕТ ПО ПРОДЕЛАННОЙ РАБОТЕ В РАМКАХ НАУЧНО-ИССЛЕДОВАТЕЛЬСКОГО СЕМИНАРА «ВВЕДЕНИЕ В СПЕЦИАЛЬНОСТЬ»

ТЕМА: ЗАДАЧА КОММИВОЯЖЁРА

Работу выполнил: Башкиров Данил Павлович
Научный руководитель: Посыпкин Михаил Анатольевич

Москва 2018

Введение

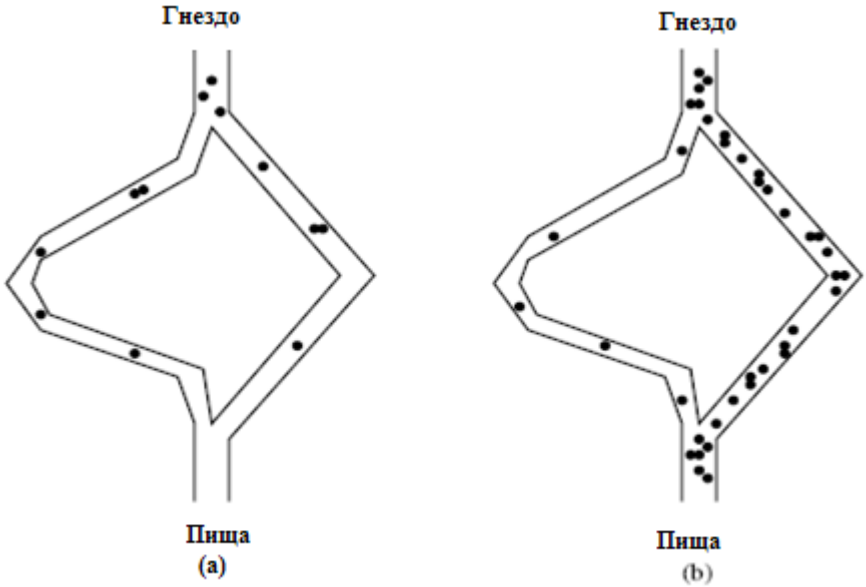
Задача комивояжёра - задача, заключающаяся в поиске кратчайшего гамильтонова цикла. Гамильтонов цикл - цикл наименьшей длины или стоимости, проходящий через все вершины графа, при этом степень каждой вершины должна быть равна двум.

Точно неизвестно, когда задача была поставлена впервые, однако подобная проблема поднималась в книге 1832 года *«Коммивояжёр — как он должен вести себя и что должен делать для того, чтобы доставлять товар и иметь успех в своих делах — советы старого курьера»*. В ней предлагались наилучшие маршруты в Германии и Швейцарии для странствующих торговцев. Во второй половине XX века задача коммивояжера привлекла внимание ученых из США и Европы. На ее основе были построены многие современные алгоритмы.

1 Методы решения

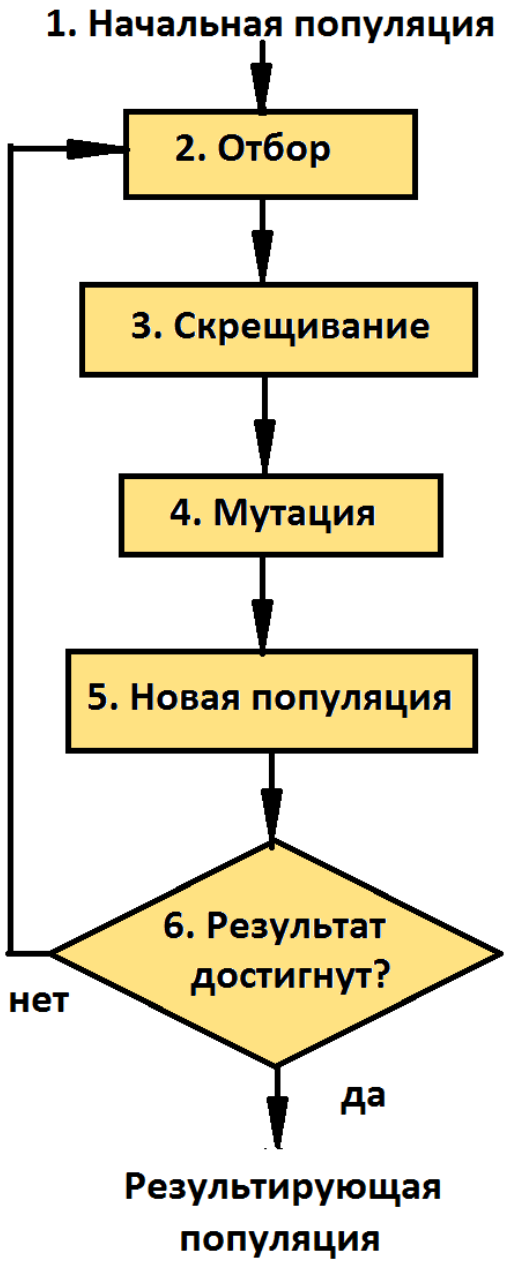
1.1 Муравьиный алгоритм

Данный алгоритм моделирует поведение колонии муравьев при поиске пищи. В реальной жизни муравьи ходят случайными путями, а при обнаружении провизии прокладывают феромонные тропы. Они помогают другим муравьям находить дорогу к пище, что значительно ускоряет её добычу. Феромоны имеют свойство испаряться, но если путь пользуется популярностью, то она постоянно обновляется. Таким образом, если дорога занимает достаточно много времени, то она начинает слабеть. И наоборот, если феромонный уровень достаточно высок то это поможет найти кратчайший путь.



1.2 Генетический алгоритм

Данный алгоритм моделирует механизм естесвенного отбора в природе. Генерируется случайный вектор генов, где каждый ген может быть представлен числом, битом и каким-то другим объектом. Случайным образом генерируется множество таких векторов и к нему применяется «функция приспособленности», которая проверит приспособленность данного множества генотипов. В нашем случае это будет функция, которая будет проверять условие минимальной стоимости гамильтонова цикла. Каждому такому множеству присваивается оценка, насколько хорошо оно справляется с поставленной задачей. Выбираются несколько лучших решений и к ним снова применяется «функция приспособленности». Этот набор действий подобен «эволюционному процессу», и для его прекращения нужен критерий остановки.



1.3 Метод эластичной сети

Данный алгоритм был предложен в 1987 году Дурбином и Уиллшоу, которые указали на его аналогичность механизмам установления упорядоченных нейронных связей. Суть заключается в установке небольшой окружности на плоскость, которая будет неравномерно расширяться, становясь, в итоге, искомым гамильтоновым циклом. У каждой точки окружности есть две цели: достичь ближайшего города и держаться как можно ближе к соседям, чтобы длина кольца была наименьшей. В дальнейшем города будут притягивать к себе только ближайшие точки окружности, так что конечная окружность будет искомым маршрутом.

1.4 Метод ветвей и границ

Данный метод впервые был предложен для решения задач целочисленного программирования в 1960 году Лендом и Дойгом. Алгоритм может быть рассмотрен на примере поиска минимума для функции $f(x)$. Разобьем множество определения x на подмножества, и представим их в виде узлов дерева. Полученное дерево называется *деревом поиска*. Тогда, если A и B - подмножества множества значений x, а $inf(f(A)) > sup(f(B))$, то множество A дальше можно не рассматривать. Если рекурсивно применить метод ко всем узлам дерева, то в итоге останется область, максимальное и нижнее значение функции от которой будут совпадать. Именно эта область и будет минимумом.

2 Ход решения

1. Пусть дана матрица смежности связного графа:

	1	2	3	4	5
1	∞	20	18	12	8
2	5	∞	14	7	11
3	12	18	∞	6	11
4	11	17	11	∞	12
5	5	5	5	5	∞

   Для того, чтобы исключить зацикливание, элементам, лежащим на главной диагонали присваивается бесконечно большое значение.

2. Первым этапом алгоритма является редуцирование матрицы по строкам и столбцам. Для этого для каждой строки найдем минимальный элемент d_i:

	1	2	3	4	5	d_i
1	∞	20	18	12	8	8
2	5	∞	14	7	11	5
3	12	18	∞	6	11	6
4	11	17	11	∞	12	11
5	5	5	5	5	∞	5

3. Вычтем минимальный элемент d_i строки из каждого ее элемента:

	1	2	3	4	5
1	∞	12	10	4	0
2	0	∞	9	2	6
3	6	12	∞	0	5
4	0	6	0	∞	1
5	0	0	0	0	∞

4. Такую же операцию редукции проведем для столбцов. Найдем минимальный элемент d_j для каждого столбца:

	1	2	3	4	5
1	∞	12	10	4	0
2	0	∞	9	2	6
3	6	12	∞	0	5
4	0	6	0	∞	1
5	0	0	0	0	∞
d_j	0	0	0	0	0

5. Вычтем минимальный элемент d_j столбца из каждого его элемента

	1	2	3	4	5
1	∞	12	10	4	0
2	0	∞	9	2	6
3	6	12	∞	0	5
4	0	6	0	∞	1
5	0	0	0	0	∞

6. После проделанных операций, в каждой строке и в каждом столбце гарантированно найдется элемента$a[i][j] = 0$. Для кажого такого элемента найдем коэффициент, который будет равен сумме минимального элемента в строке i и столбце j. Затем удалим из матрицы строку и столбец, в которой будет находиться нулевой элемент с наибольшим коэффициентом, а обратному пути $a[j][i]$ (если он существует) присвоим бесконечно большое значение. Если же нулевых элементов с максимальным коэффициентом несколько, можно удалить любой. Координаты удаленного элемента i, j будут являться ребром, составляющим кратчайший гамильтонов граф. Как видно из матрицы, приведенной ниже, искомый элемент имеет координаты (5, 2), следовательно, 5 строку и 2 столбец необходимо исключить.

	1	2	3	4	5
1	∞	12	10	4	0(5)
2	0(2)	∞	9	2	6
3	6	12	∞	0(5)	5
4	0(0)	6	0(0)	∞	1
5	0(0)	0(6)	0(0)	0(0)	∞

7. Для получившейся матрицы повторим шаги 1 - 6, пока ее размерность не станет 1x1.

	1	3	4	5
1	∞	10	4	0
2	0	9	2	∞
3	6	∞	0	5
4	0	0	∞	1

8. В итоге получим матрицу:

	5
1	0

Нетрудно заметить, что единственной незамкнутой парой вершин остались 1 и 5, следовательно, ребро (1, 5) составляет искомый гамильтонов граф

В результате, получим, что искомый путь: $1 \to 5 \to 2 \to 4 \to 3$, а его длина равна 43

3 Реализация

Алгоритм был реализован на языке C++. Программа считывает из файла «input.txt» размеры матрицы смежности и ее содержимое, а ребра гамильтонова графа и его длину выводит в файл «output.txt».

- Функция reductionMatrix(vector<vector<double» a) производит редукцию матрицы a по столбцам и строкам.
- Функция deleteRC(vector<vector<double» &a, int i, int j) удаляет из матрицы a строку с номером i и столбец с номером j.
- В функции bbmethod(vector<vector<double» &a) реализуется непосредственно сам алгоритм ветвей и границ. Функция производит редукцию матрицы a с помощью функции reductionMatrix(vector<vector<double» a), а затем находит нулевой элемент с наибольшим коэффициентом приведения. Затем этот элемент удаляется из матрицы с помощью функции deleteRC(vector<vector<double» &a, int i, int j) вместе со строкой и столбцом, в которых он находится. После от получившейся матрицы рекурсионно вызывается функция bbmethod(vector<vector<double» &a), а критерием выхода является размерность матрицы. Как только она становится равна 1, функция выводит координаты оставшегося элемента и прекращает работу.

4 Код программы

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <vector>
#include <fstream>
#include <limits>
#include <locale>

using namespace std;

vector<vector<double>> init(66);
int min_way = 0;

vector <vector<double>> reductionMatrix(vector <vector<double>> a) {
    double red_const;
    for (int i = 1; i < a[0].size(); i++) {
        red_const = numeric_limits <double>::infinity();
        for (int j = 1; j < a[0].size(); j++)
            if (a[i][j] < red_const) red_const = a[i][j];
        for (int j = 1; j < a[0].size(); j++)
            a[i][j] -= red_const;
    }
    for (int i = 1; i < a[0].size(); i++) {
        red_const = numeric_limits <double>::infinity();
        for (int j = 1; j < a[0].size(); j++)
            if (a[j][i] < red_const) red_const = a[j][i];
        for (int j = 1; j < a[0].size(); j++)
            a[j][i] -= red_const;
    }
    return a;
}

void deleteRC(vector<vector<double>> &a, int i, int j) {
    int n = a[0].size();
    double b;
    for (int k = i; k < n - 1; k++)
        for (int q = 0; q < n; q++) {
            b = a[k][q];
            a[k][q] = a[k + 1][q];
            a[k + 1][q] = b;
```

```
        }
    a.resize(n - 1);
    for (int k = j; k < n - 1; k++)
        for (int q = 0; q < n - 1; q++) {
            b = a[q][k];
            a[q][k] = a[q][k + 1];
            a[q][k + 1] = b;
        }
    for (int i = 0; i < n - 1; i++)
        a[i].resize(n - 1);
}

void bbmethod(vector<vector<double>> &a) {
    if (a[0].size() > 2) {
        a = reductionMatrix(a);
        int maxi, maxj;
        double branch_edge_column, branch_edge_row, max_b_l = -1;
        for (int i = 1; i < a[0].size(); i++) {
            for (int j = 1; j < a[0].size(); j++) {
                if (a[i][j] == 0) {
                    branch_edge_column = numeric_limits <double>::infinity();
                    branch_edge_row = numeric_limits <double>::infinity();
                    for (int k = 1; k < a[0].size(); k++) {
                        if (a[i][k] < branch_edge_row && k != j)
                            branch_edge_row = a[i][k];
                        if (a[k][j] < branch_edge_column && k != i)
                            branch_edge_column = a[k][j];
                    }
                    if (branch_edge_column + branch_edge_row > max_b_l) {
                        max_b_l = branch_edge_column + branch_edge_row;
                        maxi = i; maxj = j;
                    }
                }
            }
        }
        a[maxi][maxj] = numeric_limits <double>::infinity();
        for (int i = 1; i < a[0].size(); i++)
            for (int j = 1; j < a[0].size(); j++)
                if (a[maxi][0] == a[0][j] && a[0][maxj] == a[i][0])
                    a[i][j] = numeric_limits <double>::infinity();
```

```
        printf("(%.0f, %.0f)\n", a[maxi][0], a[0][maxj]);
        min_way += init[a[maxi][0] - 1][a[0][maxj] - 1];
        deleteRC(a, maxi, maxj);
        bbmethod(a);
    } else {
        printf("(%.0f, %.0f)\n", a[1][0], a[0][1]);
        min_way += init[a[1][0] - 1][a[0][1] - 1];
    }
}

int main() {
    setlocale(LC_ALL, "Russian");
    ifstream fin("input.txt");
    ofstream fout("output.txt");
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    vector<vector<double>> a(66);
    int n, x;
    cin >> n;
    for (int i = 0; i <= n; i++)
        a[0].push_back(i);
    for (int i = 1; i <= n; i++) {
        a[i].push_back(i);
        for (int j = 0; j < n; j++) {
            cin >> x;
            a[i].push_back(x);
            init[i - 1].push_back(x);
        }
    }
    for (int i = 1; i < a[0].size(); i++)
        a[i][i] = numeric_limits <double>::infinity();
    cout << "Ребра, составляющие кратчайший гамильтонов цикл:" << endl;
    bbmethod(a);
    cout << "Длина этого цикла: ";
    cout << min_way << endl;
    return 0;
}
```

Итоги

В результате данного исследования были изучены новые алгоритмы и методы комбинаторной оптимизации, а также улучшены навыки владения языком программирования C++ и системой подготоки документов LaTex.

Список литературы

[1] Сигал И. Х., *«Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы»*, 2002

[2] URL: *«Travelling salesman problem»*, *https : //en.wikipedia.org/wiki/Travelling_salesman_proble*

[3] URL: *«Пример решения задачи коммивояжера»*, *https : //math.semestr.ru/kom/komm.php*

[4] Воронцов К. В. *«LaTeXв примерах»*, 2005