НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ

ИМ. Р.Е. АЛЕКСЕЕВА

Кафедра " Вычислительные системы и технологии"

**Программирование**

**Курсовая работа**

Задача коммивояжёра.

Выполнил студент группы 19-ИВТ-3

Сухоруков Валерий Алексеевич

«18» мая 2020 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил ст. Преподаватель кафедры ВСТ

Мартынов Д.С.

«\_\_\_» \_\_\_\_\_\_\_ 2020 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2020

Оглавление

[Введение 4](#_Toc49713147)

[Цели и задачи работы 4](#_Toc49713148)

[Состояние вопроса 4](#_Toc49713149)

[Генетические алгоритмы 5](#_Toc49713150)

[Обзор 5](#_Toc49713151)

[Представление объектов 6](#_Toc49713152)

[Область применения 7](#_Toc49713153)

[Принцип действия 8](#_Toc49713154)

[Решение задачи коммивояжера генетическим алгоритмом 10](#_Toc49713155)

[Алгоритм 10](#_Toc49713156)

[Функциональные диаграммы 11](#_Toc49713157)

[Блок-схема 14](#_Toc49713158)

[Функция выбора особи 14](#_Toc49713159)

[Функция выбора трех особей 15](#_Toc49713160)

[Функция мутации 16](#_Toc49713161)

[Псевдокод 21](#_Toc49713162)

[Функция выбора особи 21](#_Toc49713163)

[Функция выбора трех особей 22](#_Toc49713164)

[Функция мутации 23](#_Toc49713165)

[Решение задачи точным методом. Метод ветвей и границ. 26](#_Toc49713166)

[Алгоритм 26](#_Toc49713167)

[Функциональные диаграммы 27](#_Toc49713168)

[Блок-схема 29](#_Toc49713169)

[Псевдокод 36](#_Toc49713170)

[Результаты работы программ 42](#_Toc49713171)

[Вывод по результатам 43](#_Toc49713172)

[Сложность алгоритмов 44](#_Toc49713173)

[Классы сложности 44](#_Toc49713174)

[Класс P 44](#_Toc49713175)

[Класс NP 44](#_Toc49713176)

[Задача коммивояжёра 45](#_Toc49713177)

[Литература 46](#_Toc49713178)

[Приложения 47](#_Toc49713179)

[Руководство пользователя к программам 47](#_Toc49713180)

[Руководство программиста к программам 48](#_Toc49713181)

[Программная реализация метода ветвей и границ 49](#_Toc49713182)

[Программная реализация генетического алгоритма 54](#_Toc49713183)

[Матрицы расстояний, используемые при проверке программ 60](#_Toc49713184)

# Введение

## Цели и задачи работы

В данной работе будет рассмотрена задача коммивояжера. Это одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. В рамках работы будет рассмотрен вариант задачи, в котором нужно посетить каждый город только один раз и вернуться в исходный.

Целью работы является сопоставление точного, но долгого метода решения-метода ветвей и границ с более быстрым, но менее точным - генетическим алгоритмом.

## Состояние вопроса

В настоящее время используются как точные, так и приближенные методы решения задачи. При сравнительно не большом количестве городов, очевидно, будет более целесообразно выбрать точный метод, ведь время, затраченное на решение задачи, будет не столь велико, а результат будет точно лучшим. Решение генетическим алгоритмом следует применять при значительном увеличении городов, когда время, затрачиваемое на решение методом ветвей и границ будет также значительно увеличиваться.

Путем проведения различных тестов решения задачи будет найдено оптимальные числа городов, для которых применим генетический алгоритм и метод ветвей и границ.

# Генетические алгоритмы

## Обзор

Генетический алгоритм — это класс эволюционных алгоритмов поиска. Идея генетических алгоритмов основана на эволюционной теории Чарльза Дарвина. Этот алгоритм симулирует процесс естественного отбора, когда более сильные особи из популяции переживают более слабых и производят следующее поколение особей.

История эволюционных вычислений началась с разработки ряда различных независимых моделей [[1](#_Литература)]. Основными из них были генетические алгоритмы и классификационные системы Голланда (Holland), опубликованные в начале 60-х годов и получившие всеобщее признание после выхода в свет книги, ставшей классикой в этой области, - "Адаптация в естественных и искусственных системах" ("Adaptation in Natural and Artifical Systems", 1975). В 70-х годах в рамках теории случайного поиска Растригиным Л.А. был предложен ряд алгоритмов, использующих идей бионического поведения особей. Развитие этих идей нашло отражение в цикле работ Букатовой И.Л. по эволюционному моделированию. Развивая идеи Цетлина М.Л. о целесообразном и оптимальном поведении стохастических автоматов, Неймарк Ю.И. предложил осуществлять поиск глобального экстремума на основе коллектива независимых автоматов, моделирующих процессы развития и элиминации особей. Большой вклад в развитие эволюционного программирования внесли Фогел (Fogel) и Уолш (Walsh). Несмотря на разницу в подходах, каждая из этих "школ" взяла за основу ряд принципов, существующих в природе, и упростила их до такой степени, чтобы их можно было реализовать на компьютере. Генетический алгоритм представляет собой метод решения задач оптимизации в виде процедур поиска, основанных на механизмах естественного отбора и наследования. В них используется эволюционный принцип выживания наиболее приспособленных особей. Они отличаются от традиционных методов оптимизации несколькими базовыми элементами. В частности, генетические алгоритмы:

* обрабатывают не значения параметров самой задачи, а их закодированную форму;
* осуществляют поиск решения исходя не из единственной точки, а из их некоторой популяции;
* используют только целевую функцию, а не ее производные либо иную дополнительную информацию,
* применяют вероятностные правила выбора.

Перечисленные свойства приводят в результате к устойчивости генетических алгоритмов и их превосходству над другими методами оптимизации.

## Представление объектов

Из биологии мы знаем, что любой организм может быть представлен своим фенотипом, который фактически определяет, чем является объект в реальном мире, и генотипом, который содержит всю информацию об объекте на уровне хромосомного набора [[2]](#_Литература). При этом каждый ген, то есть элемент информации генотипа, имеет свое отражение в фенотипе. Таким образом, для решения задач нам необходимо представить каждый признак объекта в форме, подходящей для использования в генетическом алгоритме. Все дальнейшее функционирование механизмов генетического алгоритма производится на уровне генотипа, позволяя обойтись без информации о внутренней структуре объекта, что и обуславливает его широкое применение в самых разных задачах.

При описании генетических алгоритмов используются понятия, заимствованные из генетики (популяция особей, ген, хромосома, генотип, фенотип, аллель), а также соответствующие им определения из технического лексикона (цепь, двоичная последовательность, структура и т.д.) [[3]](#_Литература).

* ***Популяция*** – это конечное множество особей.
* ***Особи*,** входящие в популяцию, представляются хромосомами с закодированным в них множествами параметров задачи, т.е. решений, которые называются точками в пространстве поиска.
* ***Хромосомы*** – это упорядоченные последовательности генов.
* ***Ген*** – это атомарный элемент генотипа, в частности, хромосомы.
* ***Генотип*** – это набор хромосом данной особи. Следовательно, особями популяции могут быть генотипы либо единичные хромосомы.
* ***Фенотип***– это набор значений, соответствующих данному генотипу, т.е. декодированная структура или множество параметров задачи.
* ***Аллель*** – это значение конкретного гена, также определяемое как значение свойства или вариант свойства.
* ***Локус*** – позиция, которая указывает место размещения данного гена в хромосоме (цепочке). Множество позиций генов – это локи.

Важным понятием в генетических алгоритмах является **функция приспособленности** (fitness function), иначе называемая функцией оценки. Она представляет меру приспособленности данной особи в популяции. Эта функция играет важнейшую роль, поскольку позволяет оценить степень приспособленности конкретных особей в популяции и выбрать из них наиболее приспособленные (т.е. имеющие наибольшие значения функции приспособленности) в соответствии с эволюционным принципом выживания «сильнейших». Функция приспособленности также получила свое название непосредственно из генетики. Она оказывает сильное влияние на функционирование генетических алгоритмов и должна иметь точное и корректное определение. В задачах оптимизации функция приспособленности, как правило, называется целевой функцией. В теории управления функция приспособленности может принимать вид функции погрешности, а в теории игр – стоимостной функции. На каждой итерации генетического алгоритма приспособленность каждой особи данной популяции оценивается при помощи функции приспособленности, и на этой основе создается следующая популяция особей, составляющих множество потенциальных решений задачи оптимизации.

## Область применения

Генетические алгоритмы в различных формах применились ко многим научным и техническим проблемам. Генетические алгоритмы использовались при создании других вычислительных структур, например, автоматов или сетей сортировки. В машинном обучении они использовались при проектировании нейронных сетей или управлении роботами. Они также применялись при моделировании развития в различных предметных областях, включая биологические (экология, иммунология и популяционная генетика), социальный (такие как экономика и политические системы) и когнитивные системы.

Тем не менее, возможно наиболее популярное приложение генетических алгоритмов - оптимизация многопараметрических функций. Многие реальные задачи могут быть сформулированы как поиск оптимального значения, где значение - сложная функция, зависящая от некоторых входных параметров. В некоторых случаях, представляет интерес найти те значения параметров, при которых достигается наилучшее точное значение функции. В других случаях, точный оптимум не требуется - решением может считаться любое значение, которое лучше некоторой заданное величины. В этом случае, генетические алгоритмы - часто наиболее приемлемый метод для поиска "хороших" значений. Сила генетического алгоритма заключена в его способности манипулировать одновременно многими параметрами, эта особенность ГА использовалось в сотнях прикладных программ, включая проектирование самолетов, настройку параметров алгоритмов и поиску устойчивых состояний систем нелинейных дифференциальных уравнений.

## Принцип действия

Генетический алгоритм производит над особями следующие действия [[4](#_Литература)]:

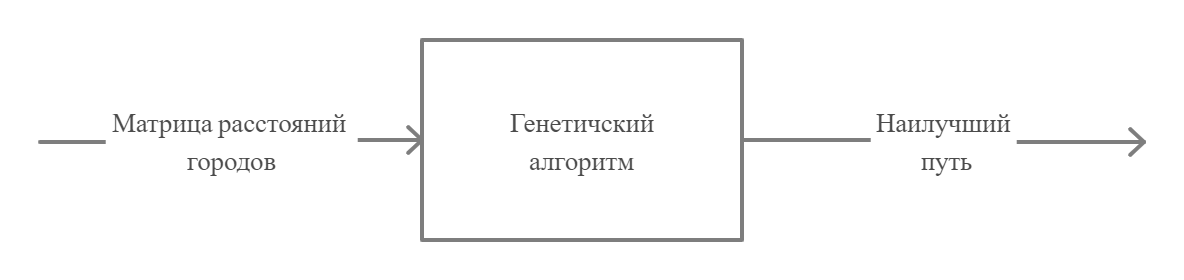
* *Генерация начальной популяции хромосом* – случайным образом выбираются значения параметров целевой функции и для этих значений параметров находится значение целевой функции.
* *Селекция* – выбор особей с наилучшей приспособленностью для воспроизводства (сортировка по значению целевой функции). Чем лучше приспособленность особи, тем выше ее шансы на скрещивание и наследование ее генов следующим поколением.
* *Кроссовер* – скрещивание. Случайным образом выбирается точка разрыва – участок между соседними битами в строке. Обе родительские структуры разрываются на два сегмента по этой точке. Затем, соответствующие сегменты различных родителей склеиваются и получаются два генотипа потомков.
* *Мутация* – случайное изменение генов. Случайным образом выбранный ген с некоторой вероятностью меняется на другой.
* *Инверсия* – изменение порядка следования частей кода. Случайным образом выбирается точка разрыва – участок между соседними битами в строке. Обе части родительские структуры, разорванной по этой точке, меняются местами, после чего склеиваются.

# Решение задачи коммивояжера генетическим алгоритмом

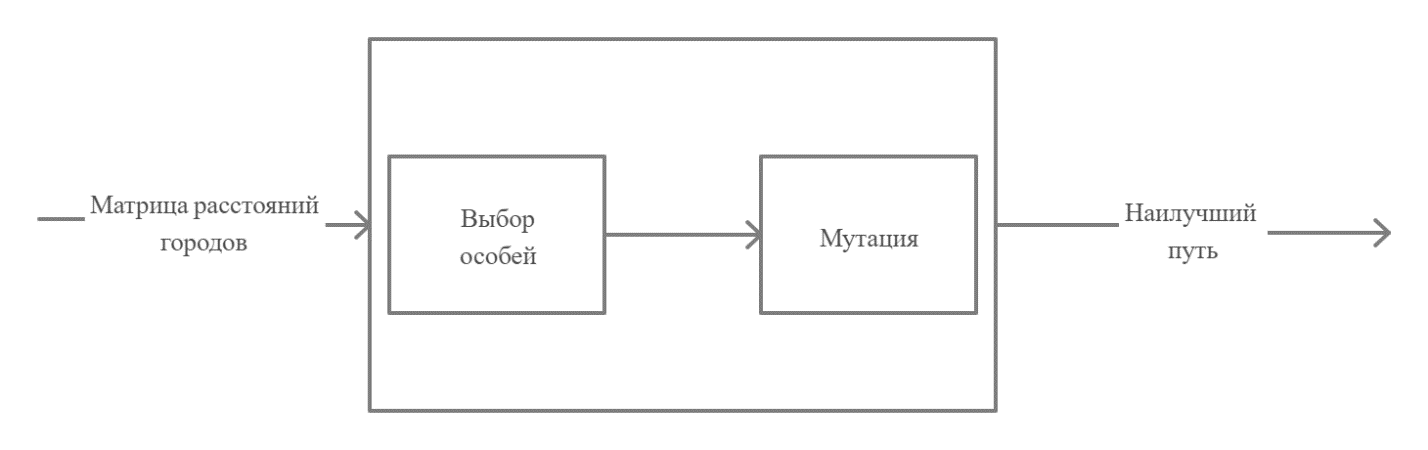
## Алгоритм

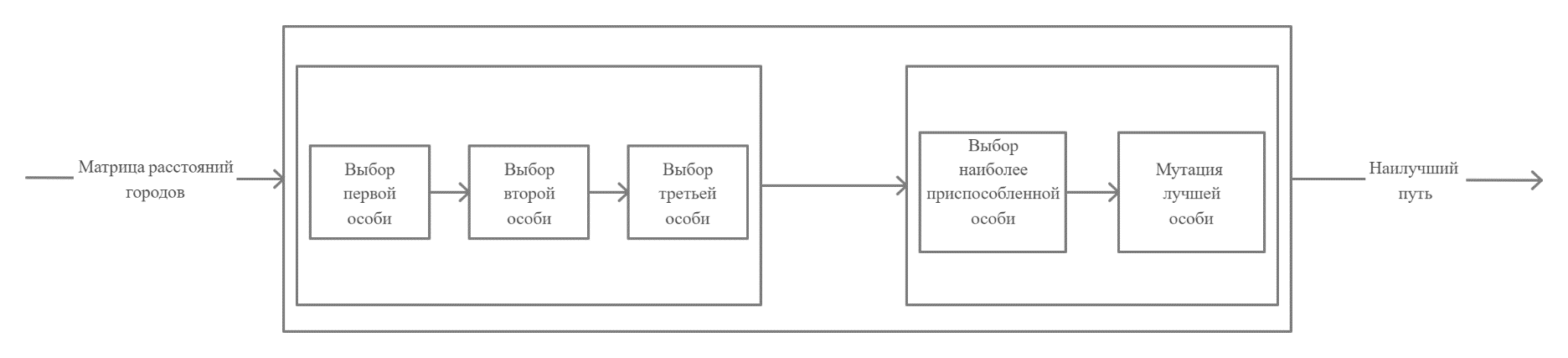
1. Случайный выбор трёх разных особей (последовательность городов, между которыми есть дорога)
2. Из полученных особей выбор самой приспособленной т.е. той, длина пути которой меньше
3. Производим мутацию особи.
4. Находим в ней два гена (города), расстояние между которыми и их соседями справа максимально.
5. Меняем местами гены.
6. Если результат мутации – уменьшение длины пути, то производим замену особи, над которой проводилась мутация, на полученную.
7. Повторение пунктов 3. - 4. пока результат мутации положительный, иначе подсчитать длину итогового пути.

## Функциональные диаграммы

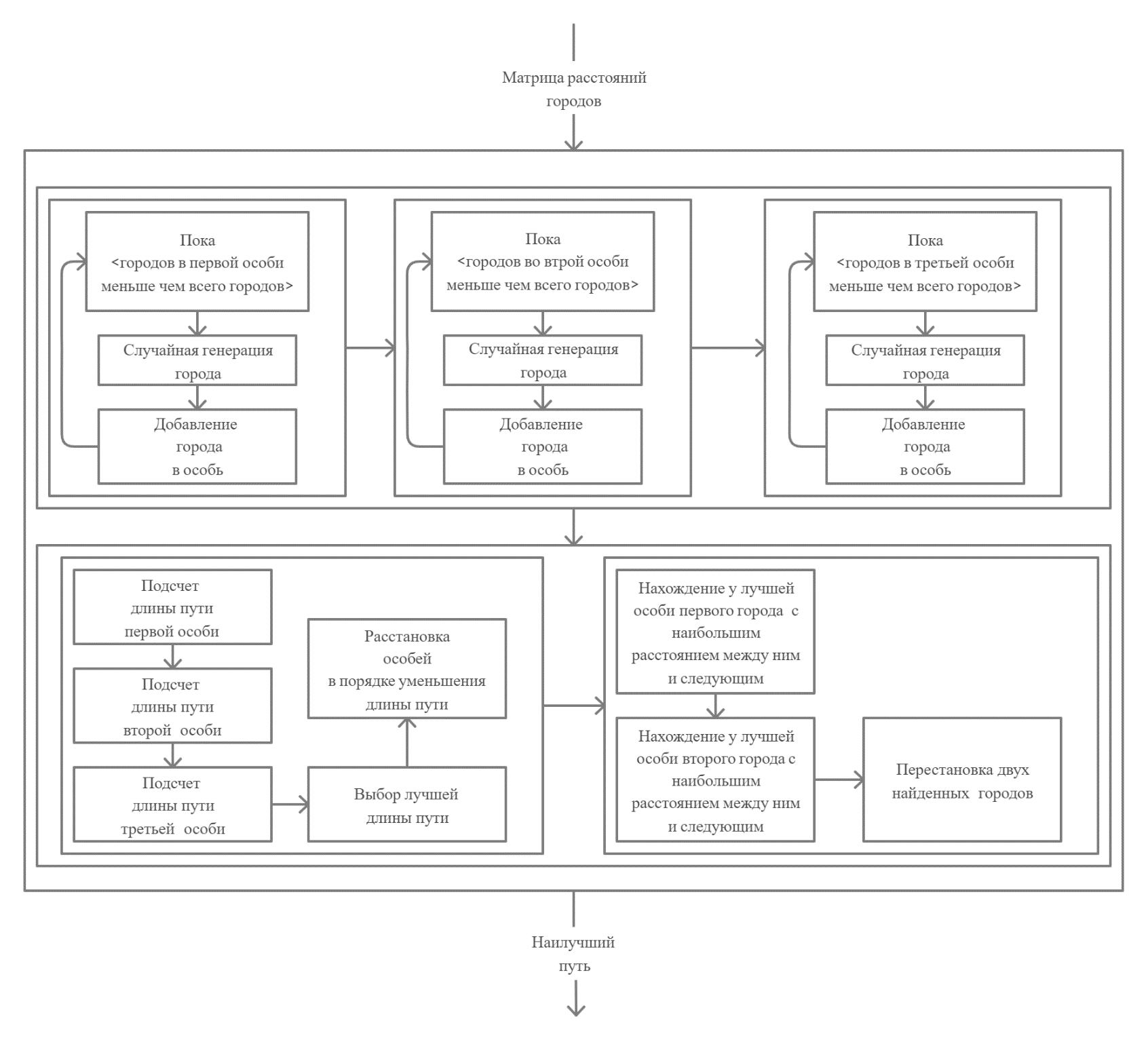


Application0

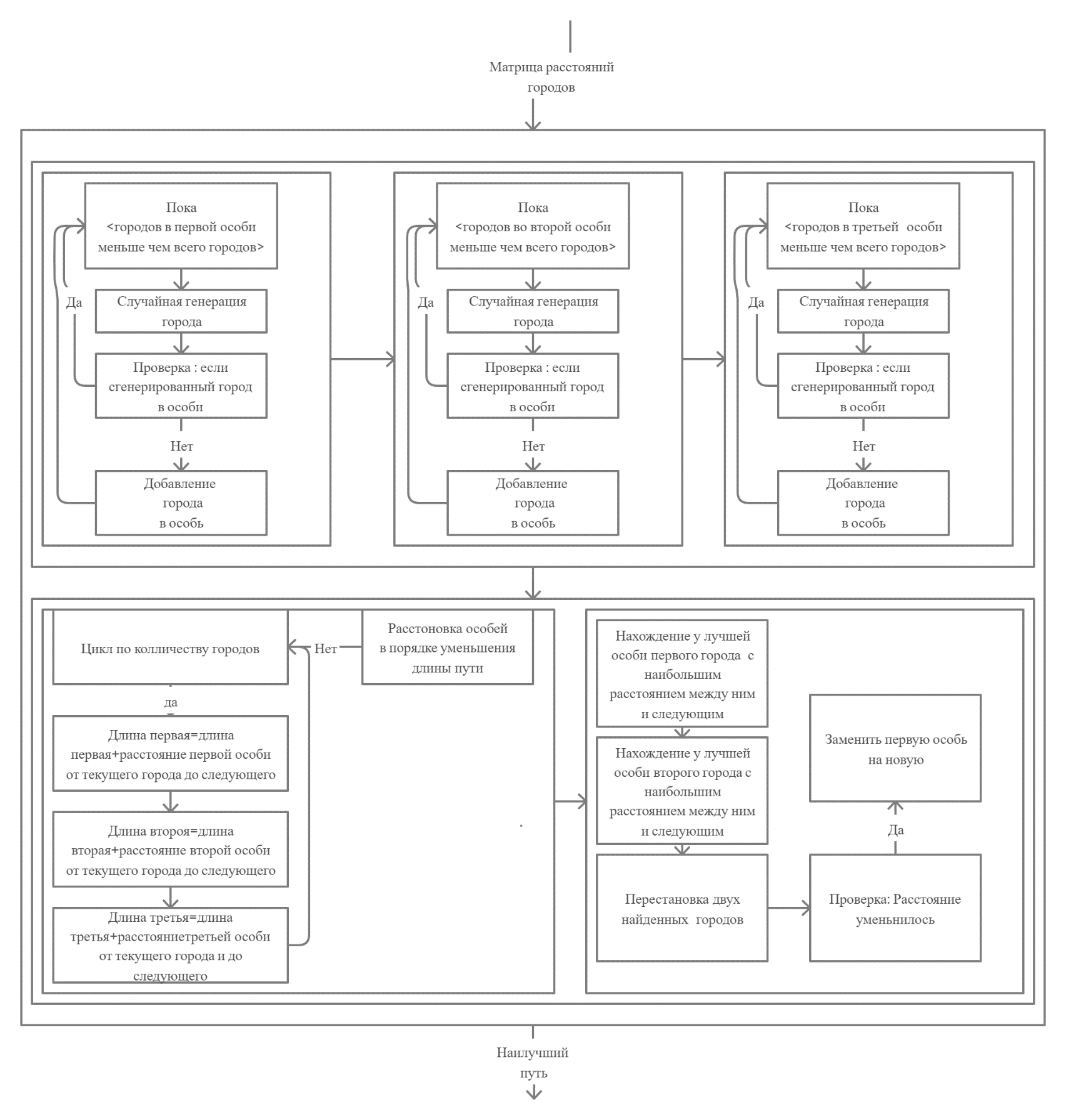


Application1

Application2



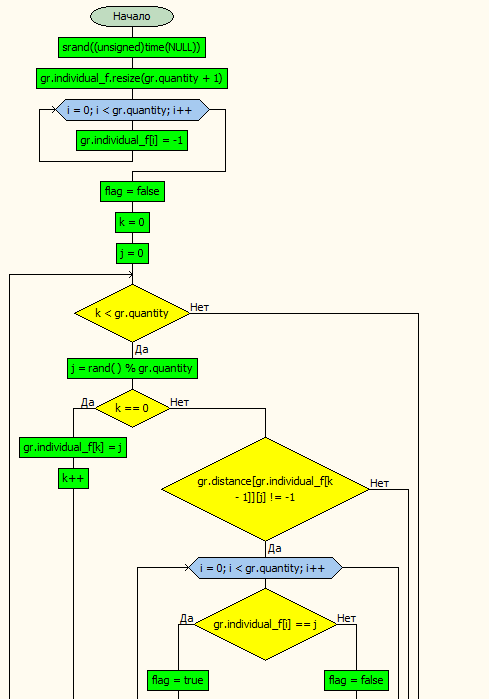
Application3

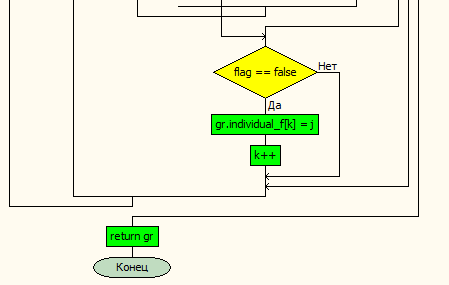


Application4

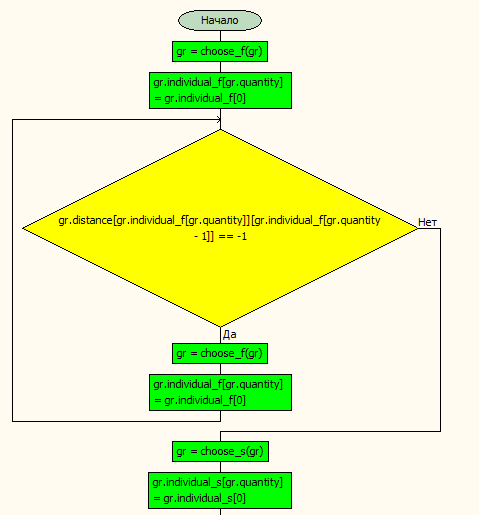
## Блок-схема

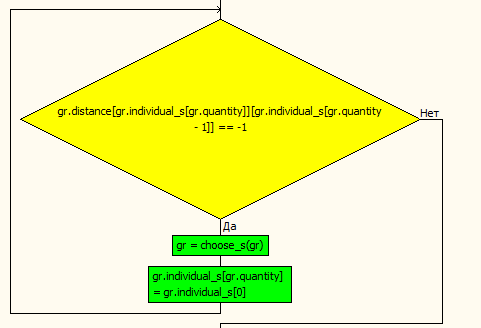
### Функция выбора особи

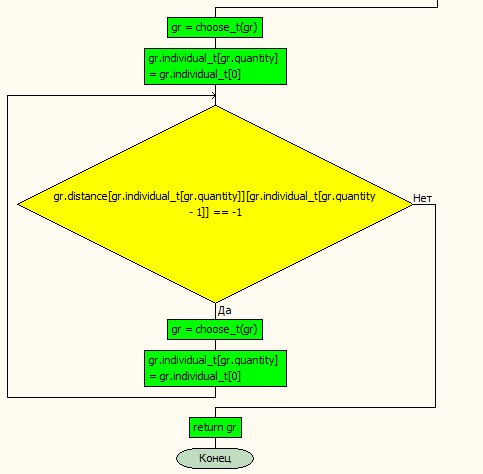




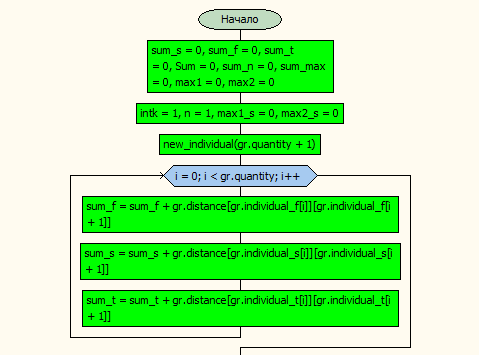
### Функция выбора трех особей

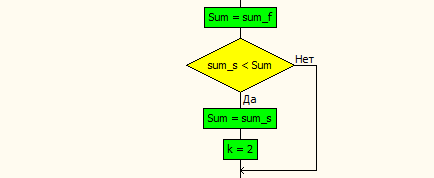


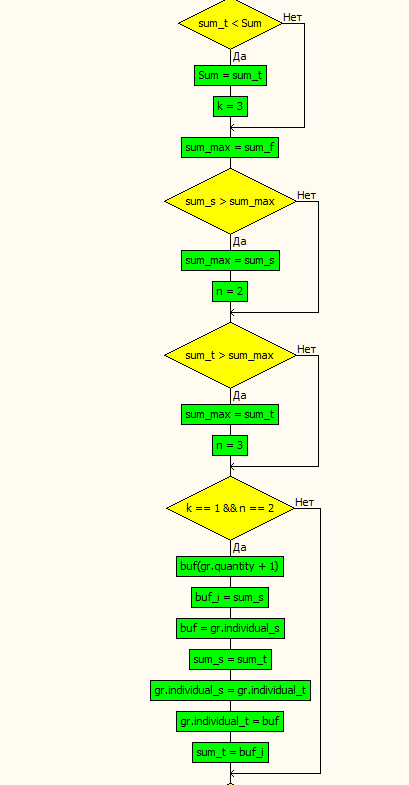


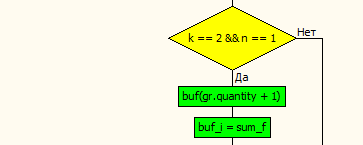


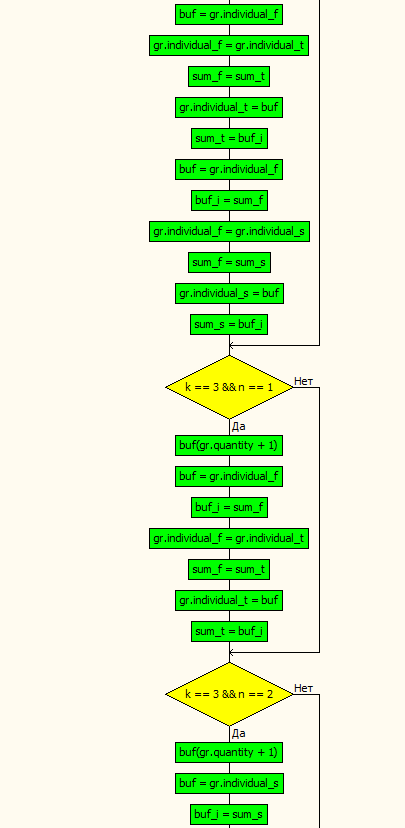
### Функция мутации

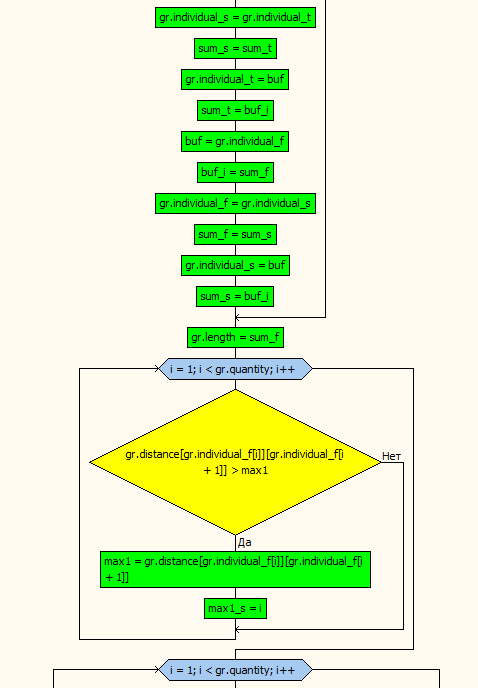


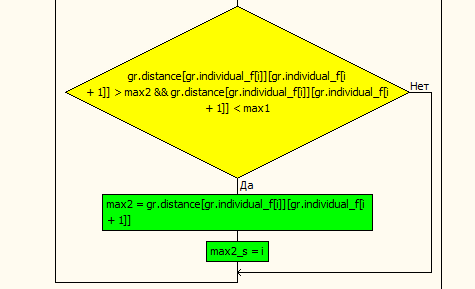


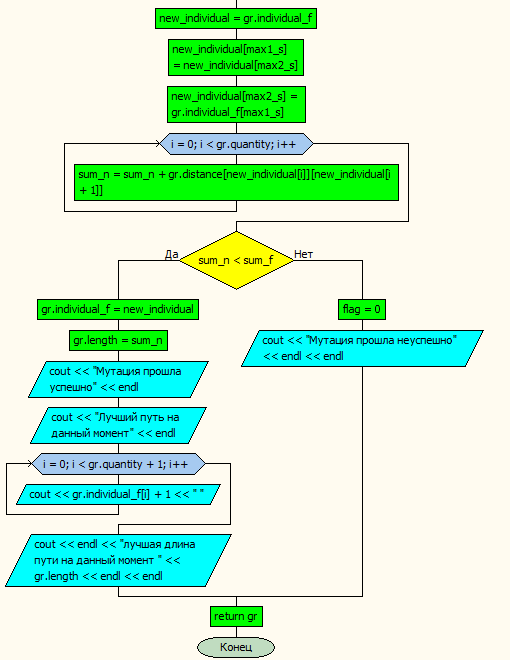












## Псевдокод

### Функция выбора особи

Алгоритм choose\_f (Аргументы: distance: Таблица: Значение с плавающей точкой, quantity: Целое, Результат: individual: Массив: Значение с плавающей точкой)

Начало

| i: Целое

|Для<i: =0, i<quantity, i++>

| | individual[i]:= -1

|Всё-цикл

|flag: Логическое

|k, j: Целое

|k: =0

|j: =0

|Пока (k<quantity)

| |j= Случайно (от 0 до quantity)

| |Если (k=0) то

| | |individual [k] =j

| | |k:=k+1

| |Иначе

| | |Если (distance [k-1] [j] <>-1) то

| | | |Для <i=0, I<quantity, i++>

| | | | | Если (individual [i] = j) то

| | | | | |flag: = true

| | | | | |Прервать-цикл

| | | | |Иначе

| | | | | |flag: = false

| | | | |Всё-если

| | | | | Если (flag=false) то

| | | | | |individual[k]: =j

| | | | | |k: =k+1

| | | | |Всё-если

| | | |Всё-цикл

| | |Всё-если

| |Всё-если

|Всё-цикл

|Вернуть: individual

Конец

### Функция выбора трех особей

Алгоритм choose\_f (Аргументы: distance: Таблица: Значение с плавающей точкой, quantity: Целое, Результат: individual\_f, individual\_s, individual\_t: Массив: Значение с плавающей точкой)

Начало

|individual\_f: =choose\_f (distance, quantity)

| individual\_f [quantity]: =individual\_f [0]

|Пока (distance [individual\_f[quantity]] [individual\_f[quantity-1]] =-1)

| |individual\_f: =choose\_f (distance, quantity)

| | individual\_f [quantity]: =individual\_f [0]

|Всё-цикл

|individual\_s: =choose\_s (distance, quantity)

| individual\_s [quantity]: =individual\_s [0]

|Пока (distance [individual\_s [quantity]] [individual\_s [quantity-1]] =-1)

| |individual\_s: =choose\_f (distance, quantity)

| | individual\_s [quantity]: =individual\_f [0]

|Всё-цикл

|individual\_t: =choose\_t (distance, quantity)

| individual\_t [quantity]: =individual\_t [0]

|Пока (distance [individual\_t [quantity]] [individual\_t[quantity-1]] =-1)

| |individual\_t: =choose\_t (distance, quantity)

| | individual\_t [quantity]: =individual\_t [0]

|Всё-цикл

|Вернуть: individual\_f, individual\_s, individual\_t

Конец

### Функция мутации

Алгоритм mutation (Аргументы: distance: Таблица: Значение с плавающей точкой, individual\_f, individual\_s, individual\_t: Массив: Целое, quantity: Целое, Результат: individual\_f, individual\_s, individual\_t, length: Значение с плавающей точкой)

Начало

| sum\_s, sum\_f, sum\_t, Sum, sum\_n, sum\_max, max1, max2: Значение с плавающей |точкой

| sum\_s: = 0, sum\_f: = 0, sum\_t: = 0, Sum: =0, sum\_n: =0, sum\_max: =0, max1: = 0, max2: = 0

| k, n, max1\_s, max2\_s,i: Целое

|k: =1, n: =1, max1\_s: =0, max2\_s: =0

|new\_individual: Массив: Целое

|Для<i: =0, i<quantity, i++>

| |sum\_f: =sum\_f + distance [individual\_f[i]] [individual\_f [i+1]]

| |sum\_s: =sum\_s + distance [individual\_s[i]] [individual\_s [i+1]]

| |sum\_t: =sum\_t + distance [individual\_t[i]] [individual\_t [i+1]]

|Всё-цикл

|Sum:= sum\_f

|Если (sum\_s < Sum) то

| |Sum: =sum\_s

| |k: =2

|Всё-если

|Если (sum\_t < Sum) то

| |Sum: =sum\_t

| |k: =3

|Всё-если

|Если (sum\_s > sum\_max) то

| |sum\_max: =sum\_s

| |n: =2

|Всё-если

|Если (sum\_t > sum\_max) то

| |sum\_max: =sum\_t

| |n: =3

|Всё-если

|Если (k=1 и n=2) то

| |Обмен (sum\_s, sum\_t)

| | Обмен (individual\_s, individual\_t)

|Всё-если

|Если (k=2 и n=3) то

| |Обмен (sum\_s, sum\_f)

| | Обмен (individual\_s, individual\_f)

|Всё-если

|Если (k=2 и n=1) то

| |Обмен (sum\_f, sum\_t)

| | Обмен (individual\_f, individual\_t)

| |Обмен (sum\_f, sum\_s)

| | Обмен (individual\_f, individual\_s)

|Всё-если

|Если (k=3 и n=1) то

| |Обмен (sum\_t, sum\_f)

| | Обмен (individual\_t, individual\_f)

|Всё-если

|Если (k=3 и n=2) то

| |Обмен (sum\_s, sum\_t)

| | Обмен (individual\_s, individual\_t)

| |Обмен (sum\_f, sum\_s)

| | Обмен (individual\_f, individual\_s)

|Всё-если

|length: =sum\_f

|Для<i: =1, i<quantity, i++>

| |Если (distance [individual\_f[i]] [individual\_f [i+1]] >max1) то

| | |max1: = distance [individual\_f[i]] [individual\_f [i+1]]

| | |max1\_s: =i

| |Всё-если

| |Если (distance [individual\_f[i]][individual\_f [i+1]] >max2 и distance [individual\_f | |[i]] [individual\_f [i+1]] <max1) то

| | |max2: = distance [individual\_f[i]] [individual\_f [i+1]]

| | |max2\_s: =i

| |Всё-если

|Всё-цикл

|new\_individual: = individual\_f

|Обмен (new\_individual [max1\_s], new\_individual [max2\_s])

|Для<i: = 0, i<quantity, i++>

| |sum\_n: = sum\_n + distance [new\_individual[i]][new\_individual[i + 1]]

|Всё-цикл

|Если (sum\_n < sum\_f) то

| | individual\_f:= new\_individual

| | length: = sum\_n

|Всё-если

|Вернуть: individual\_f, individual\_s, individual\_t, length

Конец

# Решение задачи точным методом. Метод ветвей и границ.

## Алгоритм

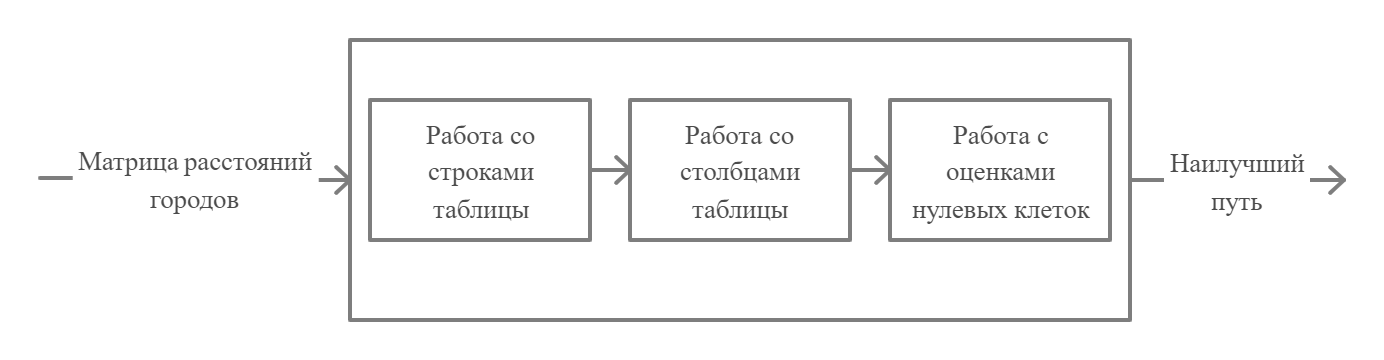
Для решения задачи коммивояжера методом ветвей и границ необходимо выполнить следующий алгоритм (последовательность действий):

1. Построение матрицы расстояний.
2. Нахождение минимума по строкам.
3. Редукция строк.
4. Нахождение минимума по столбцам.
5. Редукция столбцов.
6. Вычисление оценок нулевых клеток.
7. Редукция матрицы.
8. Если полный путь еще не найден, переходим к пункту 2, если найден к пункту 9
9. Вычисление итоговой длины пути и построение маршрута

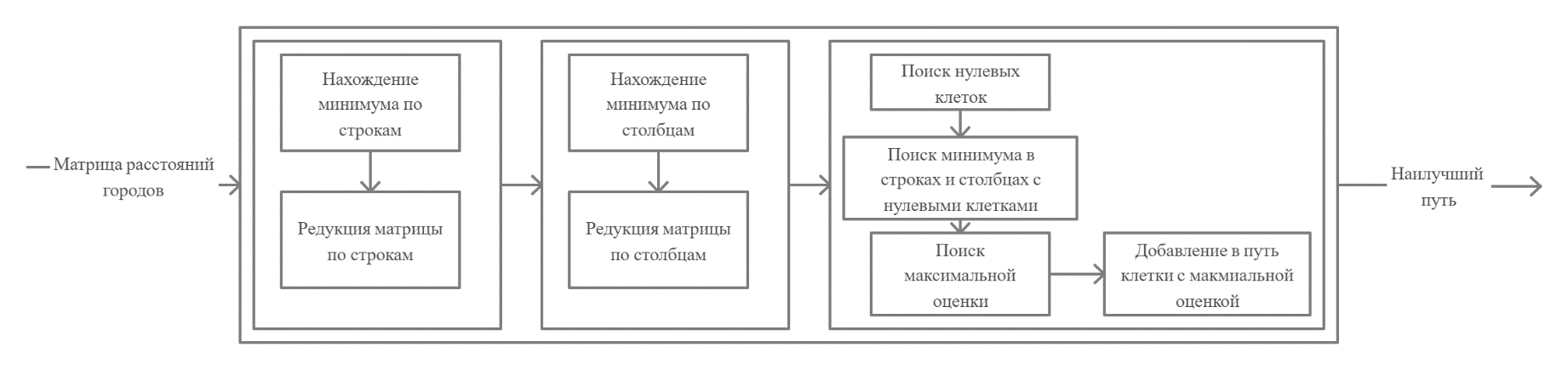
## Функциональные диаграммы



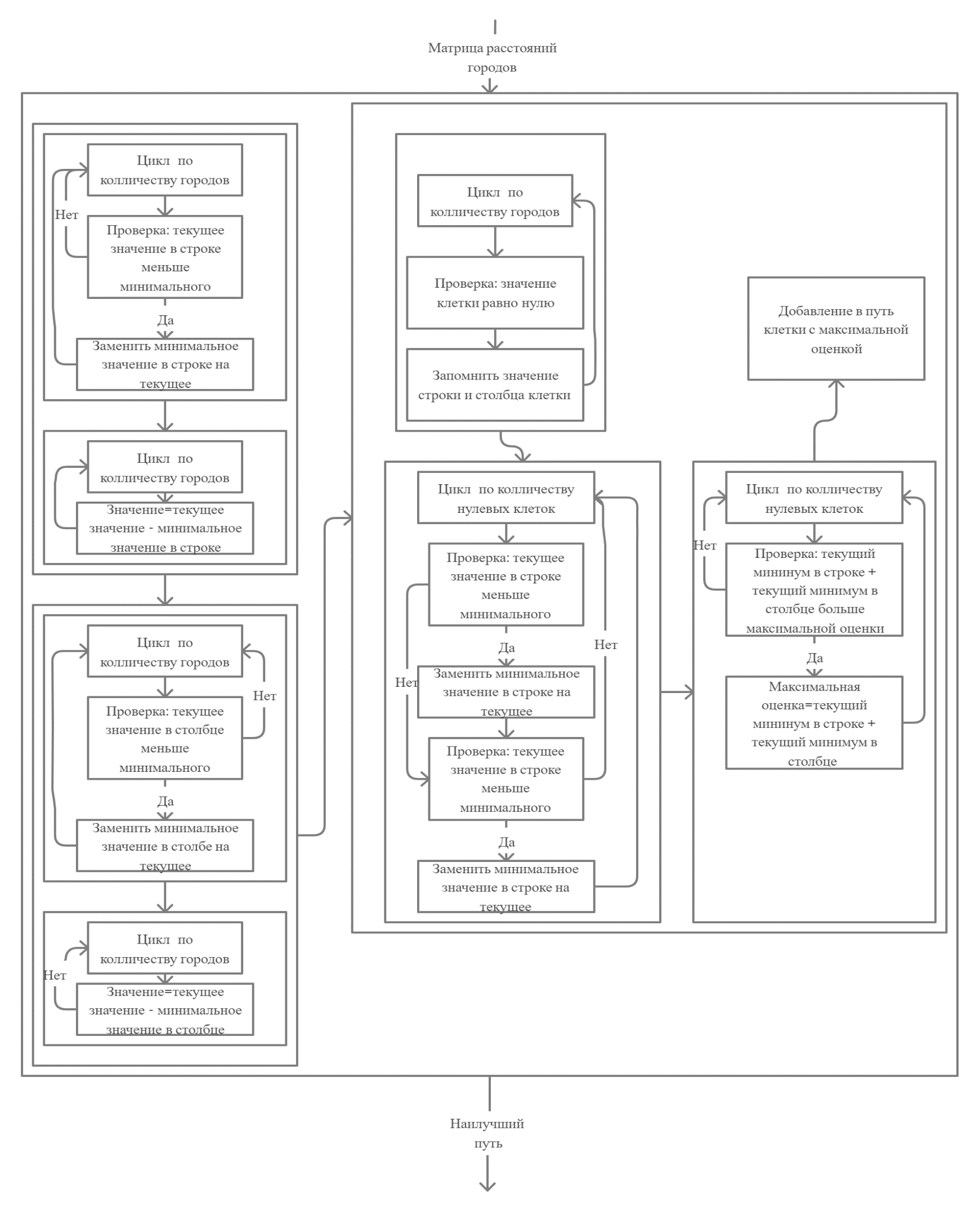
Application0



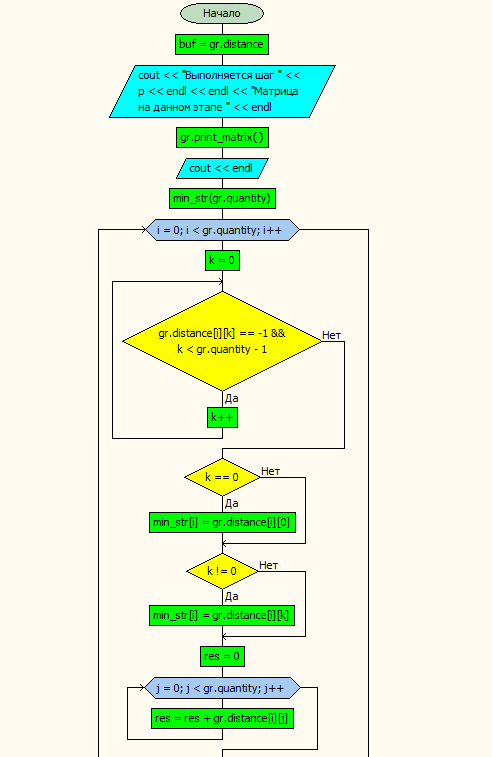
Application1

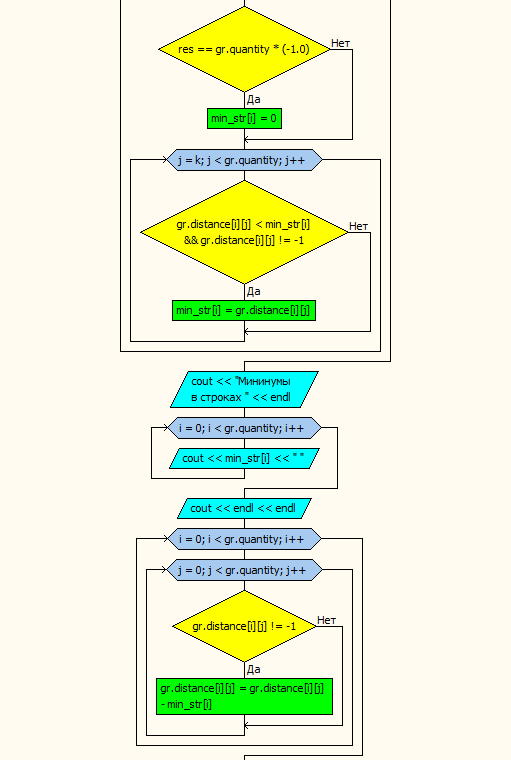
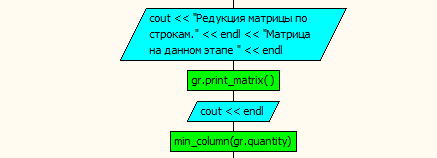


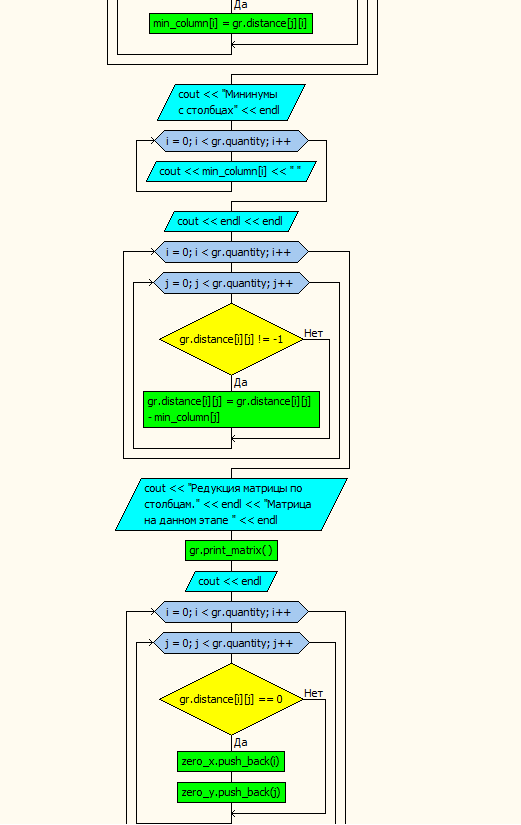
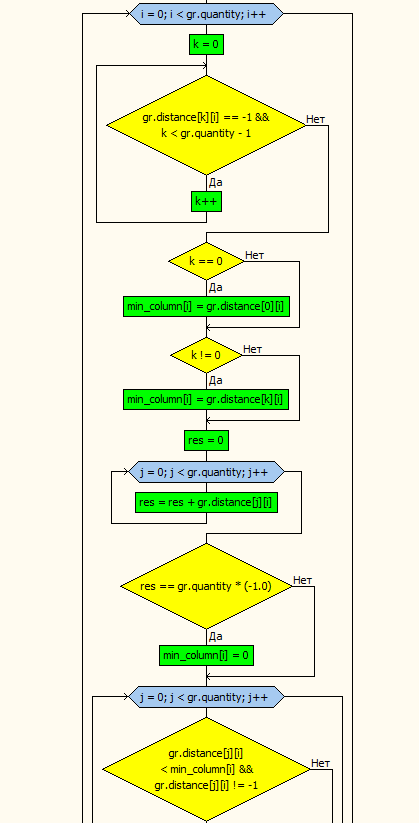
Application2

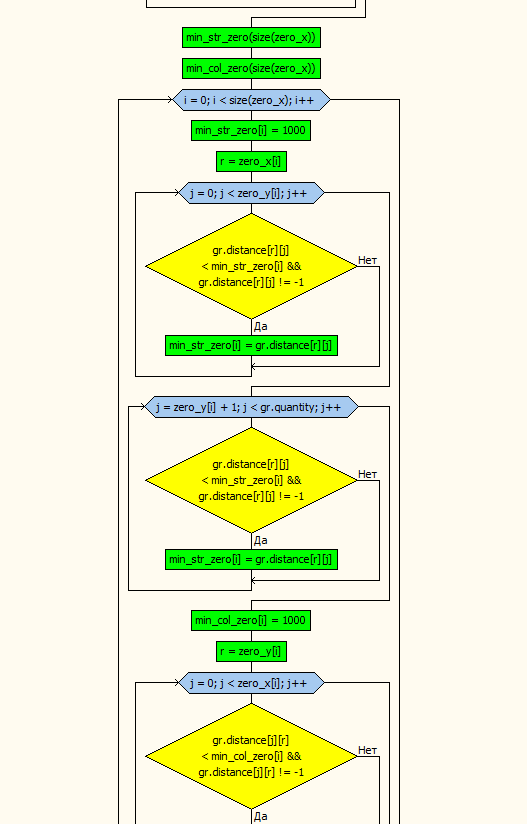
 Application4

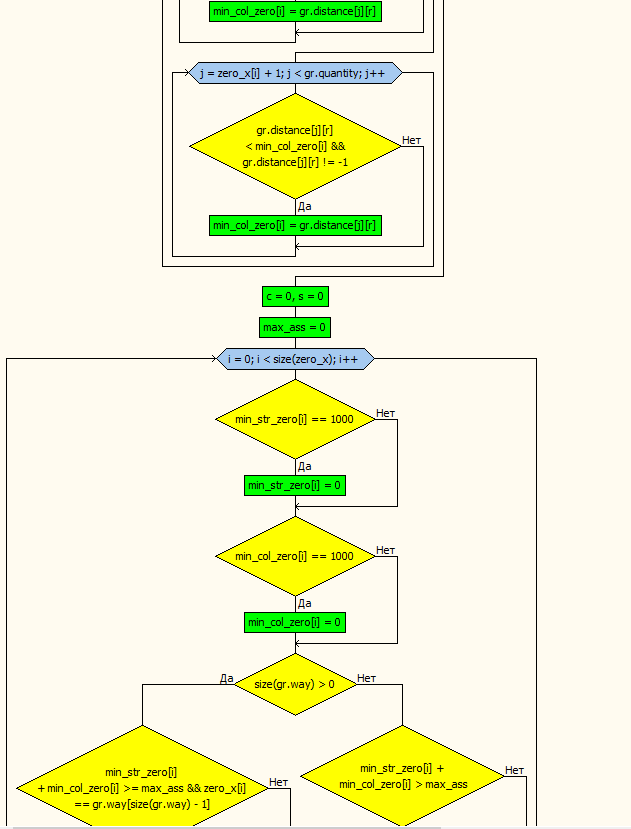
## Блок-схема

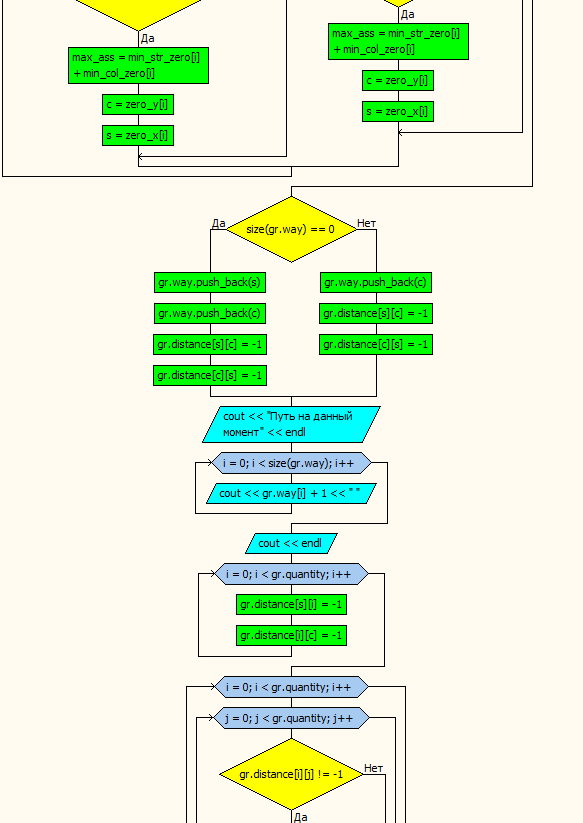
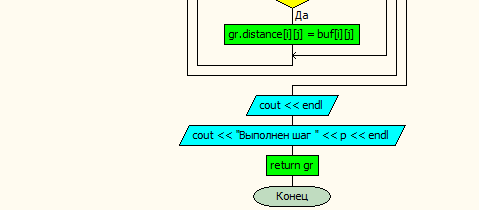












## Псевдокод

Алгоритм step (Аргументы: distance: Таблица: Значение с плавающей точкой, quantity: Целое, Результат: way: Массив: Целое)

Начало

|Вывод: distance

|min\_str, min\_col, buf: Таблица: Число с плавающей точкой

|buf=distance

|i, j: Целое

|Для <i=0, i<quantity, i++>

| |k: Целое

| |k: =0

| | Пока < distance[i] [k] =-1 и k<quantity -1>

| | |k:=k+1

| |Всё-цикл

| | Если (k=0) то

| | |min\_str[i]: = distance[i] [0]

| |Всё-если

| | Если (k<>0) то

| | |min\_str[i]: = distance[i] [k]

| |Всё-если

| | res: Число с плавающей точкой

| |res: =0

| | Для <j: =0, j <quantity, j++>

| | |res: =res+distance [i] [j]

| | Всё-цикл

| | Если (res=quantity\*-1) то

| | |min\_str[i]: = 0

| |Всё-если

| | Для <j:=k, j< quantity, j++>

| | |Если (distance[i] [j] < min\_str[i] и distance[i] [j] <>-1) то

| | | | min\_str[i]: = distance[i] [j]

| | | Всё-если

| | Всё-цикл

|Всё-цикл

|Для <i: =0, i<quantity, i++>

| | Для <j: =0, j< quantity, j++>

| | | Если (distance[i] [j] <>-1) то

| | | | distance[i] [j]: = distance[i] [j] - min\_str[i]

| | | Всё-если

| | Всё-цикл

|Всё-цикл

|Вывод: distance

|Для <i:=0, i<quantity, i++>

| |k: Целое

| |k:=0

| | Пока < distance[k] [i] =-1 и k<quantity -1>

| | |k:=k+1

| |Всё-цикл

| | Если (k=0) то

| | |min\_col [i]: = distance [0] [i]

| |Всё-если

| | Если (k<>0) то

| | |min\_str[i]: = distance[k] [i]

| |Всё-если

| | res: Число с плавающей точкой

| |res: =0

| | Для <j: =0, j <quantity, j++>

| | |res=res+distance [j] [i]

| | Всё-цикл

| | Если (res=quantity\*-1) то

| | |min\_col[i]: = 0

| |Всё-если

| | Для <j: =k, j< quantity, j++>

| | |Если (distance[j] [i] < min\_col[i] и distance[i] [j] <>-1) то

| | | | min\_col [i]: = distance[j] [i]

| | | Всё-если

| | Всё-цикл

|Всё-цикл

|Для <i: =0, i<quantity, i++>

| | Для <j: =0, j< quantity, j++>

| | | Если (distance[i] [j] <>-1) то

| | | | distance[i] [j]: = distance[i] [j] - min\_col [j]

| | | Всё-если

| | Всё-цикл

| Всё-цикл

|Вывод: distance

|zero\_x, zero\_y: Массив: Целое

|Для <i: =0, i<quantity, i++>

| | Для <j=0, j< quantity, j++>

| | | Если (distance[i] [j] =0) то

| | | | zero\_x. добавить\_в \_конец (i)

| | | | zero\_y. добавить\_в \_конец(j)

| | | Всё-если

| | Всё-цикл

| Всё-цикл

|min\_col\_zero, min\_str\_zero: Таблица: Число с плавающей точкой

|Для<i: =0, i<Размер (zero\_x), i++>

| |min\_str\_zero[i] = Max

| |r: Целое

| |r:=zero\_x[i]

| |Для<j=0, j< zero\_y[i], j++>

| | | Если (distance[r] [j] < min\_srt\_zero[i] и distance[r] [j] <>-1) то

| | | |min\_str\_zero\_ [i]: =distance[r] [j]

| | |Всё-если

| |Всё-цикл

| | Для<j: = zero\_y[i] +1, j< quantity, j++>

| | | Если (distance[r] [j] < min\_srt\_zero[i] и distance[r] [j] <>-1) то

| | | |min\_str\_zero\_ [i]: =distance[r] [j]

| | |Всё-если

| |Всё-цикл

|min\_col\_zero[i]: =Max

|r: =zero\_y[i]

| |Для<j: =0, j< zero\_x[i], j++>

| | | Если (distance[j] [r] < min\_col\_zero[i] и distance[j] [r] <>-1) то

| | | |min\_col\_zero\_ [i]: =distance[j] [r]

| | |Всё-если

| |Всё-цикл

| | Для<j: = <zero\_x[i] +1, j< quantity, j++>

| | | Если (distance[j] [r] < min\_col\_zero[i] и distance[j] [r] <>-1) то

| | | |min\_col\_zero\_ [i]: =distance[j] [r]

| | |Всё-если

| |Всё-цикл

| Всё-цикл

|c, s: Целое

|c: =0

|s: =0

|max\_ass: Число с плавающей точкой

|max\_ass:= 0

| Для<i=0, i<Размер (zero\_x), i++>

| |Если (min\_col\_zero\_ [i] =Max) то

| | | min\_col\_zero\_ [i]: =0

| |Всё-если

| |Если (min\_str\_zero\_ [i] =Max) то

| | | min\_str\_zero\_ [i]: =0

| |Всё-если

| | Если (Размер (way>0)) то

| | | Если (min\_str\_zero[i] +min\_col\_zero[i]>=max\_ass и zero\_x[i] =

| | |way [Размер (way) - 1] и zero\_y[i] <> way [0]) то

| | | |max\_ass: = min\_str\_zero[i] +min\_col\_zero[i]

| | | |c:=zero\_y[i]

| | | |s:=zero\_x[i]

| | | Всё-если

| |Иначе

| | | Если (min\_str\_zero[i] +min\_col\_zero[i]>=max\_ass) то

| | | |max\_ass:= min\_str\_zero[i] +min\_col\_zero[i]

| | | |c:=zero\_y[i]

| | | |s:=zero\_x[i]

| | | Всё-если

| |Всё-если

| Всё-цикл

|Если (Размер (way)=0) то

| | way. добавить\_в \_конец(s)

| | way. добавить\_в \_конец(c)

| |distance[s] [c]: = -1

| |distance[c] [s]: = -1

|Иначе

| |way. добавить\_в \_конец(c)

| |distance[s] [c]: = -1

| |distance[c] [s]: = -1

|Всё-если

|Если (Размер (way) = quantity) то

| | way. добавить\_в \_конец [way [0]]

|Всё-если

|Вывод: way

|Для <i=0, i<quantity, i++>

| |distance[s] [i]: = -1

| |distance[i] [c]:= -1

|Всё-цикл

|Для <i=0, i<quantity, i++>

| | Для <j=0, j< quantity, j++>

| | | Если (distance[i] [j] <>-1) то

| | | | distance[i] [j]:= buf[i] [j]

| | | Всё-если

| | Всё-цикл

|Всё-цикл

|Вернуть: way

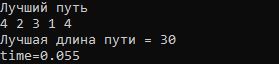
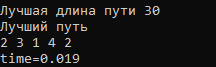
Конец

# Результаты работы программ

Программы были испытаны на 4, 10, 20, 30 и 40 городах. При всех исходных данных они повели себя корректно.

1. 4 города

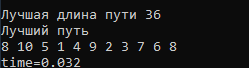
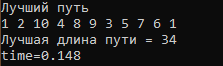
Генетический алгоритм (лучшая длина пути за 10 запусков) Метод ветвей и границ



Генетический алгоритм смог найти кратчайший путь примерно в 3 раза быстрее.

1. 10 городов

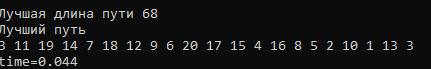
Генетический алгоритм (лучшая длина пути за 20 запусков) Метод ветвей и границ



Генетический алгоритм не смог найти кратчайший путь. Найденный им путь на несколько значений отличается от лучшего. Время его работы примерно в 5 раз меньше.

1. 20 городов

Генетический алгоритм (лучшая длина пути за 20 запусков)

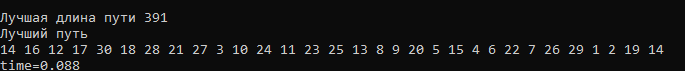


C:\Users\Валерий\Desktop\дз\програмирование\2 семетср\метод ветвей и границ\20 городов.png Метод ветвей и границ

Генетический алгоритм не смог найти наикратчайший путь. Найденный им путь уже более значительно отличается от лучшего. Время его работы примерно в 8 раз меньше.

1. 30 городов

Генетический алгоритм (лучшая длина пути за 20 запусков)

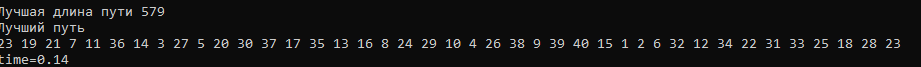


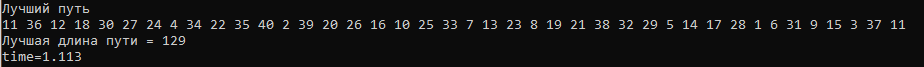
Метод ветвей и границ

C:\Users\Валерий\Desktop\дз\програмирование\2 семетср\метод ветвей и границ\30 городов.png

Генетический алгоритм вновь не смог найти наикратчайший путь. Найденный им путь уже значительно отличается от лучшего. Время его работы примерно в 7 раз меньше.

1. 40 городов

Генетический алгоритм (лучшая длина пути за 20 запусков)

Метод ветвей и границ

Время работы генетического алгоритма в 8 раз меньше, но значение его лучшей длины пути уже в несколько раз выше, чем значение, полученное с помощью метода ветвей и границ.

## Вывод по результатам

Генетический алгоритм очень эффективен при небольшом количестве городов. Его значения близки к истинным и время работы значительно меньше, чем время работы метода ветвей и границ. При увеличении количества городов сложнее случайно выбрать «особь»-последовательность городов-, которая будет близка к самой приспособленной особи, поэтому результаты, получаемые генетическим алгоритмом, ухудшаются.

# Сложность алгоритмов

Сложностьалгоритма — понятие в теории алгоритмов, обозначающее функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных. [[5]](#_Литература)

Наиболее значимыми характеристиками ресурсной эффективности алгоритмов являются оценки временной и емкостнойсложности, отражающие ресурсы процессора, оперативной памяти, а также внешних носителей данных (при использовании). Наиболее важной (критической) характеристикой является время.

## Классы сложности

В теории алгоритмов классами сложности называются множества вычислительных задач, примерно одинаковых по сложности вычисления. Для каждого класса существует категория задач, которые являются «самыми сложными». Это означает, что любая задача из класса сводится к такой задаче, и притом сама задача лежит в классе. Такие задачи называют полными задачами для данного класса. Наиболее известными являются NP-полные задачи.

### Класс P

В теории алгоритмов классом P (polynomial) называют множество задач, для которых существуют «быстрые» алгоритмы решения (время работы которых полиномиально зависит от размера входных данных).

Пример задач, принадлежащих классу Р:

Примерами задач из класса P являются целочисленное сложение, умножение, деление, взятие остатка от деления, умножения матриц, выяснение связности графов, сортировка множества из n чисел, нахождение эйлерова цикла на графе из m рёбер, обнаружение в тексте длиной n некоторого слова, построение покрывающего дерева минимальной стоимости, линейное программирование.

### Класс NP

Классом NP (non-deterministic polynomial) называют множество проблем разрешимости, решение которых возможно проверить на машине Тьюринга за время, не превосходящее полинома от размера входных данных, при наличии некоторых дополнительных сведений (так называемого сертификата решения).

Эквивалентно класс NP можно определить, как содержащий задачи, которые можно за полиномиальное время решить на недетерминированной машине Тьюринга (то есть такой машине Тьюринга, у программы которой могут существовать разные строки с одинаковой левой частью). Если машина встретила «развилку», то есть неоднозначность в программе, то дальше возможны разные варианты вычисления. Предикат R(x), который представляет данная недетерминированная машина Тьюринга, считается равным единице, если существует хоть один вариант вычисления, возвращающий 1, и нулю, если все варианты возвращают 0. Если длина вычисления, дающего 1, не превосходит некоторого многочлена от длины x, то предикат называется принадлежащим классу NP.

Сложность задач, образующих класс NP (недетерминированной полиномиальной сложности) либо экспоненциальна, либо факториальна. Сложность некоторых из них равна 2N, где N — количество входных данных.

Словосочетание «недетерминированные полиномиальные», характеризующее задачи из класса NP, объясняется следующим двухшаговым подходом к их решению.

1) *На* первом шаге имеется недетерминированный алгоритм, генерирующий возможное решение такой задачи — что-то вроде попытки угадать решение; иногда такая попытка оказывается успешной (и мы получаем оптимальный или близкий к оптимальному ответ), иногда безуспешной (ответ далек от оптимального).

2) *На* втором шаге проверяется, действительно ли ответ, полученный на первом шаге, является решением исходной задачи.

## Задача коммивояжёра

Задача коммивояжера относится к классу NP задач. Сложность алгоритма определяется количеством входных данных, чем их больше, тем больше нужно провести итераций, затратить больше времени и памяти. Метод ветвей и границ имеет сложность О (n\*6), где n- количество городов, 4- количество действий на каждом этапе. Генетический алгоритм имеет сложность О (n) где n- количество городов, она зависит только от количества городов.

# Литература

1. <http://algolist.manual.ru/ai/ga/ga1.php>
2. <https://basegroup.ru/community/articles/ga-math>
3. <https://studopedia.info/6-90615.html>
4. <https://prog-cpp.ru/genetic/>
5. Ломакина Л.С. Алгоритмы и теория сложности. Методические материалы. Электронный ресурс, НГТУ им. Р.Е.Алексеева, Нижний Новгород
6. Ломакина Л.С. Методические материалы к изучению курса Теоретические основы алгоритмизации 2020, Электронный ресурс, НГТУ им. Р.Е.Алексеева, Нижний Новгород

# Приложения

## Руководство пользователя к программам

Данная программа работает предназначена для решения задачи коммивояжера. Она считывает данные о расстояниях между городами из файла и выводит на консоль лучшее решение задачи для предоставленных данных.

Требования к файлу:

1. В первой строке записано количество городов
2. В каждой последующей строке записаны n чисел – расстояния между городом, из которого идет дорога, - его номер-строка и городом, в который идет дорога.
3. Если прямой дороги из какого-либо города в другой нет, то значение расстояния -1.
4. Если город отправления совпадает с городом прибытия, то значения расстояния -1.

## Руководство программиста к программам

Программы написаны в среде разработки microsoft visual studio, что не гарантирует корректную работу на других платформах. Для работы программы не требуется установка дополнительных библиотек. Все использованные библиотеки- стандартные.

Использованные библиотеки:

1. iostream –для работы со стандартным потоком ввода/вывода.
2. fstream – для работы с файловыми потоками ввода/вывода.
3. vector - для работы с последовательными шаблонными контейнерами.
4. string – для работы со строковыми контейнерами.
5. ctime –для работы с типом time, позволяющим отслеживать время работы программы.
6. iomanip- для манипуляции со стандартным потоком ввода/вывода.

## Программная реализация метода ветвей и границ

#include<iostream>

#include<fstream>

#include<vector>

#include<string>

#include<iomanip>

using namespace std;

struct Graph {

int quantity=0;//количество городов

double length = 0;//длина пути

vector<vector<double>>distance;//таблица расстояний

vector<int>way;//путь

vector<vector<double>>assessment;//оценка нулевых клеток

void print\_matrix() {

for (int j = 0; j < quantity; j++){

for (int i = 0; i < quantity; i++){

cout << setw(5)<<distance[j][i];

}

cout << endl;

}

}

};

Graph read(string file\_name) {//Функция чтения данных из файла

cout << endl << "Режим чтения" << endl;

Graph gr;

ifstream in;

in.open(file\_name);//открываем файл на чтение

if (!in)//проверка на корректность файла

{

if (in.eof()){

cout << "Файл пуст" << endl;

exit(-1);

}

else{

cout << "Неверное имя файла" << endl;

exit(-1);

}

}

if (in.is\_open()) {

int num;

in >> num;

vector<vector<double>> input\_data;

vector<double>string(num);

//vector<double>a(num);

for (int i = 0; i < num; i++) {

for (int j = 0; j < num; j++) {

in >> string[j];//cчитывем данные i-той строки

//a[i] = 0;

}

input\_data.push\_back(string);//записываем данные в таблицу расстояний

}

gr.quantity = num;//записываем количество городов

gr.distance = input\_data;//переписываем даные расстояний

cout << "Данные считаны" << endl;

}

in.close();

return gr;

}

Graph step(Graph gr,int p) {

vector<vector<double>>buf;

buf = gr.distance;

cout << "Выполняется шаг " << p << endl << endl << "Матрица на данном этапе " << endl;

gr.print\_matrix();

cout << endl;

//вычисление минимума в строке

vector<double>min\_str(gr.quantity);

for (int i = 0; i < gr.quantity; i++)

{

int k = 0;

while (gr.distance[i][k] == -1 && k<gr.quantity-1) { k++; }

if (k == 0) { min\_str[i] = gr.distance[i][0]; }

if (k != 0) { min\_str[i] = gr.distance[i][k]; }

//проверка на то, что строка не вычерктнута

double res = 0;

for (int j = 0; j < gr.quantity; j++){res = res+gr.distance[i][j];}

if (res==gr.quantity\*(-1.0)){min\_str[i] = 0;}

for (int j = k; j < gr.quantity; j++){

if (gr.distance[i][j] < min\_str[i] && gr.distance[i][j] != -1) { min\_str[i] = gr.distance[i][j]; }

}

}

cout << "Мининумы в строках "<<endl;

for (int i = 0; i < gr.quantity; i++){cout << min\_str[i]<<" ";}

cout << endl<<endl;

//редукция матрицы по строкам

for (int i = 0; i < gr.quantity; i++){

for (int j = 0; j < gr.quantity; j++){

if (gr.distance[i][j]!=-1){gr.distance[i][j] = gr.distance[i][j] - min\_str[i];}

}

}

cout << "Редукция матрицы по строкам." << endl << "Матрица на данном этапе " << endl;

gr.print\_matrix();

cout << endl ;

//вычисление минимума по столбцам

vector<double>min\_column(gr.quantity);

for (int i = 0; i < gr.quantity; i++){

int k = 0;

while (gr.distance[k][i] == -1 && k < gr.quantity - 1) { k++; }

if (k == 0) { min\_column[i] = gr.distance[0][i]; }

if (k != 0) { min\_column[i] = gr.distance[k][i]; }

//проверка на то, что столбец не вычеркнут

double res = 0;

for (int j = 0; j < gr.quantity; j++) {res = res + gr.distance[j][i];}

if (res == gr.quantity \* (-1.0)) { min\_column[i] = 0; }

for (int j = k; j < gr.quantity; j++){

if (gr.distance[j][i] < min\_column[i] && gr.distance[j][i] != -1) { min\_column[i] = gr.distance[j][i]; }

}

}

cout << "Мининумы с столбцах"<<endl;

for (int i = 0; i < gr.quantity; i++){

cout << min\_column[i] << " ";

}

cout << endl << endl;

//редукция матрицы по столбцам

for (int i = 0; i < gr.quantity; i++){

for (int j = 0; j < gr.quantity; j++){

if (gr.distance[i][j] != -1) { gr.distance[i][j] = gr.distance[i][j] - min\_column[j]; }

}

}

cout << "Редукция матрицы по столбцам." << endl<< "Матрица на данном этапе " << endl;

gr.print\_matrix();

cout << endl ;

//Нахождение нулевых клеток

vector<int> zero\_x;

vector<int> zero\_y;

for (int i = 0; i < gr.quantity; i++){

for (int j = 0; j < gr.quantity; j++){

if (gr.distance[i][j] == 0) { zero\_x.push\_back(i); zero\_y.push\_back(j); }

}

}

//Поиск минимума в столбцах и строках с нулевыми клетками

vector<double>min\_str\_zero(size(zero\_x));

vector<double>min\_col\_zero(size(zero\_x));

for (int i = 0; i < size(zero\_x); i++) {

min\_str\_zero[i] = DBL\_MAX;

int r = zero\_x[i];

for (int j = 0; j < zero\_y[i]; j++){

if (gr.distance[r][j] < min\_str\_zero[i] && gr.distance[r][j] != -1) { min\_str\_zero[i] = gr.distance[r][j]; }

}

for (int j = zero\_y[i]+1; j < gr.quantity; j++) {

if (gr.distance[r][j] < min\_str\_zero[i] && gr.distance[r][j] != -1) { min\_str\_zero[i] = gr.distance[r][j]; }

}

min\_col\_zero[i] = DBL\_MAX;

r = zero\_y[i];

for (int j = 0; j < zero\_x[i]; j++) {

if (gr.distance[j][r] < min\_col\_zero[i] && gr.distance[j][r] != -1) { min\_col\_zero[i] = gr.distance[j][r]; }

}

for (int j = zero\_x[i]+1; j < gr.quantity; j++) {

if (gr.distance[j][r] < min\_col\_zero[i] && gr.distance[j][r] != -1) { min\_col\_zero[i] = gr.distance[j][r]; }

}

}

//Поиск максимальной оценки нулевых клеток

int c = 0, s = 0;

double max\_ass = 0;

for (int i = 0; i < size(zero\_x); i++){

if (min\_str\_zero[i] == DBL\_MAX) { min\_str\_zero[i] = 0; }

if (min\_col\_zero[i] == DBL\_MAX) { min\_col\_zero[i] = 0; }

if (size(gr.way) > 0) {//если в пути есть города, то нужно проверить, что город из которого идем равен последнему записанному городу

if (min\_str\_zero[i] + min\_col\_zero[i] >= max\_ass && zero\_x[i] == gr.way[size(gr.way) - 1] && zero\_y[i] != gr.way[0]) {

max\_ass = min\_str\_zero[i] + min\_col\_zero[i];

c = zero\_y[i];

s = zero\_x[i];}

}

else {

if (min\_str\_zero[i] + min\_col\_zero[i] > max\_ass ) {

max\_ass = min\_str\_zero[i] + min\_col\_zero[i];

c = zero\_y[i];

s = zero\_x[i];}

}

}

if (size(gr.way) == 0) {

gr.way.push\_back(s);//добавляем в путь город s, из которого идем

gr.way.push\_back(c);//добавляем в путь город с, в который идем

gr.distance[s][c] = -1;//вычеркиваем клетку, что бы ей больше не пользоваться

gr.distance[c][s] = -1;//вычеркиваем противоположную клетку

}else {

gr.way.push\_back(c);//добавляем в путь город с, в который идем

gr.distance[s][c] = -1;//вычеркиваем клетку, что бы ей больше не пользоваться

gr.distance[c][s] = -1;//вычеркиваем противоположную клетку

}

if (size(gr.way) == gr.quantity) { gr.way.push\_back(gr.way[0]);}

cout<<"Путь на данный момент"<<endl;

for (int i = 0; i < size(gr.way); i++){cout << gr.way[i]+1<<" ";}

cout << endl;

//уничтожение из таблицы расстояний городов s и с

for (int i = 0; i < gr.quantity; i++) {

gr.distance[s][i] = -1;

gr.distance[i][c] = -1;}

//возвращение невычеркнутым клеткам изначального значения

for (int i = 0; i < gr.quantity; i++){

for (int j = 0; j < gr.quantity; j++){

if (gr.distance[i][j] != -1) { gr.distance[i][j] = buf[i][j]; }

}

}

cout << endl;

cout << "Выполнен шаг "<<p<<endl;

return gr;

}

int main() {

double start = clock();

setlocale(LC\_ALL, "RUS");

cout << "Введите имя файла для чтения данных"<<endl;

string file\_name;

cin >> file\_name;

Graph gr;

gr = read(file\_name);

vector<vector<double>>distance;

distance = gr.distance;

int i = 1;

while (size(gr.way) != gr.quantity+1 ){

gr = step(gr, i);

i++;}

cout << "Лучший путь "<<endl;

for (int i = 0; i < gr.quantity; i++) {

gr.length = gr.length + distance[gr.way[i]][gr.way[i + 1]];

cout << gr.way[i]+1 << " ";}

cout << gr.way[gr.quantity]+1;

cout << endl<<"Лучшая длина пути = "<<gr.length<<endl << "time=" << (clock() - start) / CLOCKS\_PER\_SEC;

return 0;

}

## Программная реализация генетического алгоритма

#include<iostream>

#include<fstream>

#include<vector>

#include<string>

#include <ctime>

using namespace std;

struct Graph {

int quantity = 0;//колличество городов

int length = 0;//длина пути

vector<vector<double>>distance;//таблица расстояний

vector<int>individual\_f;//особь 1

vector<int>individual\_s;//особь 2

vector<int>individual\_t;//особь 3

};

Graph read(string file\_name) {//Функция чтения данных из файла

cout << "Режим чтения" << endl;

Graph gr;

vector<vector<double>> input\_data;

ifstream in;

in.open(file\_name);//открываем файл на чтение

if (!in)//проверка на корректность файла

{

if (in.eof()) {

cout << "Файл пуст" << endl;

exit(-1);

}

else {

cout << "Неверное имя файла" << endl;

exit(-1);

}

}

if (in.is\_open()) {

int num;

in >> num;

vector<double>string(num);

vector<double>a(num);

for (int i = 0; i < num; i++) {

for (int j = 0; j < num; j++) {

in >> string[j];//cчитывем данные i-той строки

a[i] = 0;

}

input\_data.push\_back(string);//записываем данные в таблицу расстояний

}

gr.quantity = num;//записываем количество городов

gr.distance = input\_data;//переписываем даные расстояний

cout << "Данные считаны" << endl;

}

in.close();

return gr;

}

Graph choose\_f(Graph gr) {//выбор особи 1

srand((unsigned)time(NULL));//изменение начального значения ГПСЧ

gr.individual\_f.resize(gr.quantity + 1);//устанавливаем размер "особи"

for (int i = 0; i < gr.quantity; i++) { gr.individual\_f[i] = -1; }

bool flag = false;

int k = 0;//число городов в "особи"

int j = 0;//переменная, в которую будет записываться псевдослучайное число

while (k < gr.quantity) {

j = rand() % gr.quantity;

if (k == 0) {//если в особи еще нет ни одного города

gr.individual\_f[k] = j;

k++;}

else {//если в особи есть хотя бы 1 город

if (gr.distance[gr.individual\_f[k - 1]][j] != -1) {//если расстояние между последним и сгенерированным городом не равно -1

for (int i = 0; i < gr.quantity; i++)

{

if (gr.individual\_f[i] == j) {//проверяем есть ли в особи город j

flag = true;

break;

}

else { flag = false; }

}

if (flag == false) { gr.individual\_f[k] = j; k++; }//если город j в особи не обнаружен, то добавляем его

}

}

}

return gr;

}

Graph choose\_s(Graph gr) {//выбор особи 2

srand((unsigned)time(NULL) \* 2);//изменение начального значения ГПСЧ для того, чтобы особи были разные

gr.individual\_s.resize(gr.quantity + 1);

for (int i = 0; i < gr.quantity; i++) { gr.individual\_s[i] = -1; }

int k = 0, j;

bool flag = false;

while (k < gr.quantity) {

j = rand() % gr.quantity;

if (k == 0) {//если в особи еще нет ни одного города

gr.individual\_s[k] = j;

k++;}

else {//если в особи есть хотя бы 1 город

if (gr.distance[gr.individual\_s[k - 1]][j] != -1) {//если расстояние между последним и сгенерированным городом не равно -1

for (int i = 0; i < gr.quantity; i++)

{

if (gr.individual\_s[i] == j) {//проверяем есть ли в особи город j

flag = true;

break;

}

else { flag = false; }

}

if (flag == false) { gr.individual\_s[k] = j; k++; }//если город j в особи не обнаружен, то добавляем его

}

}

}

return gr;

}

Graph choose\_t(Graph gr) {//выбор особи 3

srand((unsigned)time(NULL)\*3);//изменение начального значения ГПСЧ для того, чтобы особи были разные

gr.individual\_t.resize(gr.quantity + 1);

for (int i = 0; i < gr.quantity; i++) { gr.individual\_t[i] = -1; }

int k = 0, j;

bool flag = false;

while (k < gr.quantity) {

j = rand() % gr.quantity;

if (k == 0) {//если в особи еще нет ни одного города

gr.individual\_t[k] = j;

k++;}

else {//если в особи есть хотя бы 1 город

if (gr.distance[gr.individual\_t[k - 1]][j] != -1) {//если расстояние между последним и сгенерированным городом не равно -1

for (int i = 0; i < gr.quantity; i++)

{

if (gr.individual\_t[i] == j) {//проверяем есть ли в особи город j

flag = true;

break;

}

else { flag = false; }

}

if (flag == false) { gr.individual\_t[k] = j; k++; }//если город j в особи не обнаружен, то добавляем его

}

}

}

return gr;

}

Graph choose(Graph gr) {//выбор трех особей

gr = choose\_f(gr);// выбор особи

gr.individual\_f[gr.quantity] = gr.individual\_f[0];//запись в последнюю ячейку первого города

// если расстояние между последним и предпоследним городом равно -1, то нужно перевыбрать особь

while (gr.distance[gr.individual\_f[gr.quantity]][gr.individual\_f[gr.quantity - 1]] == -1) {

gr = choose\_f(gr);

gr.individual\_f[gr.quantity] = gr.individual\_f[0];}

gr = choose\_s(gr);

gr.individual\_s[gr.quantity] = gr.individual\_s[0];

while (gr.distance[gr.individual\_s[gr.quantity]][gr.individual\_s[gr.quantity - 1]] == -1) {

gr = choose\_s(gr);

gr.individual\_s[gr.quantity] = gr.individual\_s[0];}

gr = choose\_t(gr);

gr.individual\_t[gr.quantity] = gr.individual\_t[0];

while (gr.distance[gr.individual\_t[gr.quantity]][gr.individual\_t[gr.quantity - 1]] == -1) {

gr = choose\_t(gr);

gr.individual\_t[gr.quantity] = gr.individual\_t[0];}

return gr;

}

Graph mutation(Graph gr,int& flag) {//мутация наилучшей особи

double sum\_s = 0, sum\_f = 0, sum\_t = 0,Sum=0,sum\_n=0,sum\_max=0,max1 = 0, max2 = 0;

int k=1,n=1,max1\_s=0,max2\_s=0;

vector<int>new\_individual(gr.quantity+1);//новая особь

for (int i = 0; i < gr.quantity; i++){

sum\_f = sum\_f + gr.distance[gr.individual\_f[i]][gr.individual\_f[i + 1]];//подсчет длины пути 1

sum\_s = sum\_s + gr.distance[gr.individual\_s[i]][gr.individual\_s[i + 1]];//подсчет длины пути 2

sum\_t = sum\_t + gr.distance[gr.individual\_t[i]][gr.individual\_t[i + 1]];//подсчет длины пути 3

}

//выбор наилучшего пути

Sum = sum\_f;

if (sum\_s < Sum) { Sum = sum\_s; k = 2; }

if (sum\_t < Sum) { Sum = sum\_t; k = 3; }

//выбор наихудшего пути

sum\_max = sum\_f;

if (sum\_s > sum\_max) { sum\_max = sum\_s; n = 2; }

if (sum\_t >sum\_max) { sum\_max = sum\_t; n = 3; }

//расстановка особей в порядке возрастания длины пути

if (k == 1 && n == 2) {

double buf\_i;

vector<int>buf(gr.quantity + 1);

buf\_i = sum\_s;

buf = gr.individual\_s;

sum\_s = sum\_t;

gr.individual\_s = gr.individual\_t;

gr.individual\_t = buf;

sum\_t = buf\_i;}

if (k == 2 && n == 3) {

double buf\_i;

vector<int>buf(gr.quantity + 1);

buf = gr.individual\_s;

buf\_i = sum\_s;

gr.individual\_s = gr.individual\_f;

sum\_s = sum\_f;

gr.individual\_f = buf;

sum\_f = buf\_i;}

if (k == 2 && n == 1) {

double buf\_i;

vector<int>buf(gr.quantity + 1);

buf\_i = sum\_f;

buf = gr.individual\_f;

gr.individual\_f = gr.individual\_t;

sum\_f = sum\_t;

gr.individual\_t = buf;

sum\_t = buf\_i;

buf = gr.individual\_f;

buf\_i = sum\_f;

gr.individual\_f = gr.individual\_s;

sum\_f = sum\_s;

gr.individual\_s = buf;

sum\_s = buf\_i;}

if (k == 3 && n == 1){

double buf\_i;

vector<int>buf(gr.quantity + 1);

buf = gr.individual\_f;

buf\_i = sum\_f;

gr.individual\_f = gr.individual\_t;

sum\_f = sum\_t;

gr.individual\_t = buf;

sum\_t = buf\_i;}

if (k == 3 && n == 2){

double buf\_i;

vector<int>buf(gr.quantity + 1);

buf = gr.individual\_s;

buf\_i = sum\_s;

gr.individual\_s = gr.individual\_t;

sum\_s = sum\_t;

gr.individual\_t = buf;

sum\_t = buf\_i;

buf = gr.individual\_f;

buf\_i = sum\_f;

gr.individual\_f = gr.individual\_s;

sum\_f = sum\_s;

gr.individual\_s = buf;

sum\_s = buf\_i;}

gr.length = sum\_f;//лучшая длина пути- длина пути первой особи

//Мутация проиходит у той особи, длина пути которой меньше(первой), чтобы достичь ещё меньшей длины.

//Она осуществляется по двум генам с максимальным расстоянием между городами.

//Если расстояние уменьшилось, то новая заменяет ту, у которой проводилась мутация.

for (int i = 1; i < gr.quantity; i++){

if (gr.distance[gr.individual\_f[i]][gr.individual\_f[i+1]]>max1){

max1 = gr.distance[gr.individual\_f[i]][gr.individual\_f[i + 1]];

max1\_s = i;

}

}

for (int i = 1; i < gr.quantity; i++) {

if (gr.distance[gr.individual\_f[i]][gr.individual\_f[i + 1]] > max2 && gr.distance[gr.individual\_f[i]][gr.individual\_f[i + 1]] < max1) {

max2 = gr.distance[gr.individual\_f[i]][gr.individual\_f[i + 1]];

max2\_s = i;

}

}

new\_individual = gr.individual\_f;

new\_individual[max1\_s] = new\_individual[max2\_s];

new\_individual[max2\_s] = gr.individual\_f[max1\_s];

for (int i = 0; i < gr.quantity; i++){sum\_n = sum\_n + gr.distance[new\_individual[i]][new\_individual[i + 1]];}//подсчет длины пути новой особи

//Если мутация прошла успешно т.е длина уменьшилась, то заменяем особь.

//Иначе присваиваем переменной flag, переданной как параметр значение равное 0 - показатель неуспешной мутации

if (sum\_n < sum\_f) {

gr.individual\_f = new\_individual;

gr.length = sum\_n;

cout << "Мутация прошла успешно"<<endl;

cout << "Лучший путь на данный момент" << endl;

for (int i = 0; i < gr.quantity + 1; i++) { cout << gr.individual\_f[i] + 1 << " "; }

cout << endl << "лучшая длина пути на данный момент " << gr.length << endl << endl;

}

else {

flag = 0;

cout << "Мутация прошла неуспешно" << endl << endl;}

return gr;

}

int main() {

double start = clock();

int flag=1;

cout << "Введите имя файла для чтения данных"<<endl;

string file\_name;

cin >> file\_name;

Graph gr;

gr = read(file\_name);

gr=choose(gr);

cout << "сформированные особи:"<<endl;

for (int i = 0; i < gr.quantity + 1; i++) { cout << gr.individual\_f[i] + 1 << " "; }

cout << endl;

for (int i = 0; i < gr.quantity + 1; i++) { cout << gr.individual\_s[i] + 1 << " "; }

cout << endl;

for (int i = 0; i < gr.quantity + 1; i++) { cout << gr.individual\_t[i] + 1 << " "; }

cout << endl<<endl;

while (flag == 1) {

gr = mutation(gr, flag);

}

cout << "Лучшая длина пути "<<gr.length << endl;

cout << "Лучший путь " << endl;

for (int i = 0; i < gr.quantity+1; i++){cout << gr.individual\_f[i] + 1 << " " ;}

cout << endl<<"time="<< (clock() - start) / CLOCKS\_PER\_SEC;

return 0;

}

## Матрицы расстояний, используемые при проверке программ

1. 4 города

|  |  |  |  |
| --- | --- | --- | --- |
| -1 | 5 | 11 | 9 |
| 10 | -1 | 8 | 7 |
| 7 | 14 | -1 | 8 |
| 12 | 6 | 15 | -1 |

1. 10 городов

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 5 |
| 8 | -1 | 4 | 7 | 8 | 9 | 6 | 7 | 5 | 4 |
| 3 | 4 | -1 | 2 | 3 | 4 | 5 | 6 | 7 | 3 |
| 8 | 7 | 8 | -1 | 6 | 8 | 6 | 4 | 5 | 6 |
| 1 | 1 | 1 | 1 | -1 | 1 | 1 | 1 | 1 | 1 |
| 6 | 8 | 9 | 6 | 8 | -1 | 9 | 2 | 8 | 3 |
| 8 | 9 | 7 | 8 | 7 | 5 | -1 | 8 | 6 | 7 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | -1 | 2 | 2 |
| 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | -1 | 6 |
| 7 | 8 | 9 | 4 | 5 | 6 | 7 | 8 | 9 | -1 |

1. 20 городов

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 5 | -1 | 8 | 7 | 8 | 6 | 8 | 4 | 9 | -1 | 8 |
| 8 | -1 | 4 | 7 | 8 | 9 | 6 | 7 | 5 | 4 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | -1 |
| 3 | 4 | -1 | 2 | 3 | 4 | 5 | 6 | 7 | 3 | 7 | 7 | 7 | 2 | 8 | 6 | 8 | 3 | 2 | 1 |
| 8 | 7 | 8 | -1 | 6 | 8 | 6 | 4 | 5 | 6 | 6 | 8 | 17 | 4 | 8 | 5 | 8 | 9 | 7 | 8 |
| 1 | 1 | 1 | 1 | -1 | 1 | 1 | 1 | 1 | 1 | 2 | 7 | 8 | 1 | 5 | 7 | 8 | 9 | 7 | 1 |
| 6 | 8 | 9 | 6 | 8 | -1 | 2 | 2 | 8 | 3 | 8 | 2 | 3 | 1 | 4 | 5 | 7 | 8 | 9 | 5 |
| 8 | 9 | 7 | 8 | 1 | 5 | -1 | 8 | 6 | 7 | 8 | 9 | 10 | 11 | 2 | 5 | 6 | -1 | 8 | 7 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | -1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | -1 | 6 | 8 | 9 | 2 | 7 | 5 | 3 | 8 | 9 | 4 | 1 |
| 7 | 8 | 9 | 4 | 5 | 6 | 7 | 8 | 9 | -1 | 8 | 9 | -1 | -1 | 8 | 9 | 7 | 2 | 9 | 8 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 5 | -1 | 8 | 7 | 8 | 6 | 8 | 4 | 9 | -1 | 8 |
| 8 | -1 | 4 | 7 | 8 | 9 | 6 | 7 | 5 | 4 | 8 | -1 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | -1 |
| 3 | 4 | -1 | 2 | 3 | 4 | 5 | 6 | 7 | 3 | 7 | 7 | -1 | 2 | 8 | 9 | 8 | 3 | 2 | 1 |
| 8 | 7 | 8 | -1 | 6 | 8 | 6 | 4 | 5 | 6 | 6 | 8 | 7 | -1 | 8 | 5 | 8 | 9 | 7 | 8 |
| 1 | 1 | 1 | 1 | -1 | 1 | 1 | 1 | 1 | 1 | 2 | 7 | 8 | 10 | -1 | 7 | 8 | 9 | 7 | 10 |
| 6 | 8 | 9 | 6 | 8 | -1 | 9 | 2 | 8 | 3 | 8 | 2 | 3 | 1 | 4 | -1 | 7 | 8 | 9 | 5 |
| 8 | 9 | 7 | 8 | 7 | 5 | -1 | 8 | 6 | 7 | 8 | 9 | 10 | 11 | 2 | -1 | -1 | 8 | 7 | 2 |
| 2 | 2 | 2 | 2 | 2 | 2 | -1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | -1 | 1 | 1 | 3 |
| 3 | 3 | 3 | 4 | 4 | 4 | 4 | 8 | 6 | 8 | 9 | 2 | 7 | 5 | 3 | 8 | 9 | -1 | 1 | 7 |
| 7 | 8 | 9 | 4 | 5 | 6 | 7 | 8 | 9 | -1 | 8 | 9 | -1 | -1 | 8 | 9 | 7 | 2 | 6 | -1 |

1. 30 городов

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | 8 | 22 | 48 | 25 | 8 | 20 | 41 | 23 | 39 | 24 | 19 | 45 | 25 | 4 | 25 | 21 | 3 | 49 | 41 | 43 | 8 | 50 | 46 | 17 | 34 | 37 | 2 | 22 | 16 |
| 22 | -1 | 3 | 16 | 47 | 40 | 18 | 34 | 42 | 10 | 38 | 15 | 40 | 17 | 46 | 24 | 19 | 48 | 5 | 31 | 35 | 24 | 39 | 28 | 28 | 48 | 44 | 26 | 19 | 12 |
| 32 | 39 | -1 | 5 | 21 | 10 | 34 | 27 | 31 | 1 | 8 | 24 | 26 | 24 | 3 | 50 | 6 | 32 | 24 | 46 | 1 | 21 | 45 | 14 | 35 | 14 | 21 | 2 | 33 | 35 |
| 21 | 36 | 40 | -1 | 35 | 14 | 8 | 48 | 50 | 26 | 12 | 3 | 24 | 5 | 13 | 3 | 2 | 37 | 26 | 45 | 27 | 37 | 38 | 41 | 5 | 48 | 21 | 22 | 1 | 43 |
| 11 | 5 | 34 | 9 | -1 | 12 | 23 | 7 | 8 | 15 | 12 | 7 | 13 | 11 | 14 | 47 | 44 | 27 | 21 | 42 | 48 | 8 | 6 | 25 | 2 | 42 | 5 | 9 | 31 | 30 |
| 28 | 43 | 40 | 17 | 30 | -1 | 23 | 24 | 49 | 2 | 33 | 50 | 46 | 6 | 43 | 12 | 50 | 25 | 43 | 2 | 40 | 4 | 48 | 23 | 50 | 14 | 36 | 16 | 20 | 45 |
| 30 | 16 | 1 | 17 | 5 | 9 | -1 | 34 | 15 | 30 | 45 | 3 | 7 | 7 | 6 | 6 | 13 | 38 | 50 | 25 | 3 | 39 | 21 | 20 | 3 | 17 | 48 | 24 | 32 | 32 |
| 6 | 50 | 36 | 22 | 6 | 26 | 43 | -1 | 29 | 1 | 6 | 18 | 38 | 29 | 12 | 50 | 28 | 49 | 13 | 32 | 46 | 50 | 22 | 18 | 20 | 36 | 37 | 1 | 20 | 36 |
| 34 | 26 | 10 | 42 | 27 | 40 | 42 | 10 | -1 | 11 | 5 | 44 | 18 | 25 | 48 | 41 | 46 | 47 | 32 | 21 | 31 | 18 | 18 | 2 | 44 | 16 | 40 | 47 | 26 | 9 |
| 22 | 34 | 28 | 31 | 48 | 39 | 26 | 27 | 35 | -1 | 11 | 50 | 31 | 12 | 22 | 37 | 35 | 41 | 50 | 7 | 4 | 20 | 33 | 1 | 24 | 26 | 8 | 12 | 45 | 29 |
| 35 | 17 | 20 | 20 | 14 | 42 | 49 | 17 | 14 | 36 | -1 | 6 | 40 | 34 | 47 | 22 | 41 | 25 | 29 | 10 | 27 | 7 | 4 | 22 | 42 | 5 | 16 | 11 | 34 | 27 |
| 29 | 34 | 12 | 43 | 46 | 35 | 8 | 50 | 27 | 39 | 45 | -1 | 35 | 26 | 40 | 7 | 12 | 28 | 38 | 37 | 6 | 44 | 9 | 27 | 24 | 42 | 50 | 4 | 32 | 30 |
| 29 | 4 | 42 | 12 | 15 | 37 | 35 | 26 | 36 | 18 | 24 | 5 | -1 | 33 | 40 | 25 | 14 | 35 | 26 | 48 | 1 | 50 | 1 | 8 | 46 | 18 | 35 | 3 | 50 | 34 |
| 37 | 17 | 40 | 41 | 22 | 13 | 23 | 27 | 38 | 16 | 17 | 3 | 48 | -1 | 13 | 1 | 46 | 18 | 1 | 48 | 45 | 6 | 41 | 50 | 42 | 2 | 15 | 42 | 43 | 7 |
| 24 | 20 | 5 | 7 | 22 | 23 | 44 | 3 | 10 | 5 | 37 | 20 | 49 | 39 | -1 | 31 | 2 | 47 | 24 | 8 | 12 | 2 | 30 | 26 | 43 | 11 | 8 | 18 | 27 | 33 |
| 15 | 6 | 8 | 36 | 25 | 30 | 16 | 46 | 17 | 18 | 6 | 20 | 43 | 27 | 41 | -1 | 19 | 9 | 34 | 3 | 13 | 40 | 49 | 25 | 25 | 48 | 47 | 21 | 35 | 32 |
| 40 | 37 | 47 | 28 | 41 | 40 | 16 | 44 | 34 | 20 | 49 | 21 | 38 | 5 | 44 | 17 | -1 | 33 | 8 | 37 | 20 | 22 | 28 | 26 | 48 | 14 | 17 | 7 | 33 | 8 |
| 39 | 32 | 25 | 9 | 10 | 11 | 24 | 2 | 35 | 48 | 23 | 46 | 42 | 31 | 27 | 14 | 48 | -1 | 18 | 6 | 34 | 39 | 22 | 45 | 37 | 24 | 4 | 18 | 45 | 30 |
| 25 | 24 | 47 | 22 | 27 | 35 | 39 | 46 | 38 | 3 | 37 | 20 | 19 | 17 | 26 | 30 | 47 | 10 | -1 | 8 | 31 | 6 | 9 | 46 | 30 | 15 | 15 | 4 | 22 | 7 |
| 25 | 19 | 28 | 33 | 26 | 10 | 28 | 37 | 5 | 44 | 27 | 48 | 47 | 32 | 18 | 43 | 9 | 49 | 14 | -1 | 14 | 35 | 40 | 5 | 45 | 28 | 12 | 20 | 47 | 8 |
| 31 | 43 | 4 | 8 | 12 | 42 | 22 | 1 | 49 | 17 | 35 | 6 | 3 | 42 | 37 | 49 | 18 | 43 | 12 | 41 | -1 | 41 | 44 | 32 | 14 | 36 | 11 | 47 | 37 | 46 |
| 15 | 11 | 16 | 43 | 24 | 35 | 20 | 47 | 9 | 9 | 25 | 28 | 23 | 36 | 36 | 10 | 32 | 20 | 28 | 25 | 43 | -1 | 49 | 21 | 6 | 44 | 31 | 38 | 2 | 38 |
| 9 | 9 | 27 | 6 | 1 | 38 | 10 | 8 | 1 | 19 | 48 | 38 | 33 | 21 | 9 | 10 | 12 | 24 | 9 | 27 | 28 | 32 | -1 | 16 | 4 | 45 | 2 | 20 | 21 | 48 |
| 24 | 42 | 31 | 10 | 27 | 45 | 9 | 38 | 21 | 35 | 6 | 5 | 29 | 3 | 16 | 27 | 34 | 38 | 18 | 6 | 40 | 26 | 21 | -1 | 21 | 38 | 6 | 46 | 40 | 25 |
| 19 | 50 | 43 | 50 | 8 | 39 | 13 | 20 | 26 | 36 | 1 | 37 | 9 | 2 | 45 | 38 | 32 | 26 | 10 | 11 | 45 | 23 | 38 | 35 | -1 | 1 | 34 | 43 | 35 | 19 |
| 30 | 35 | 35 | 7 | 12 | 31 | 8 | 23 | 27 | 9 | 41 | 46 | 6 | 41 | 41 | 47 | 28 | 11 | 22 | 23 | 33 | 31 | 38 | 23 | 31 | -1 | 38 | 37 | 10 | 17 |
| 23 | 37 | 17 | 31 | 4 | 15 | 20 | 27 | 2 | 2 | 1 | 34 | 14 | 24 | 10 | 27 | 3 | 10 | 2 | 7 | 29 | 23 | 11 | 20 | 8 | 15 | -1 | 14 | 44 | 3 |
| 37 | 25 | 46 | 35 | 23 | 11 | 33 | 31 | 11 | 44 | 29 | 8 | 8 | 29 | 31 | 11 | 33 | 24 | 39 | 50 | 4 | 24 | 13 | 49 | 42 | 14 | 26 | -1 | 50 | 4 |
| 26 | 8 | 26 | 50 | 33 | 11 | 4 | 48 | 39 | 18 | 17 | 31 | 12 | 23 | 3 | 6 | 38 | 48 | 32 | 23 | 43 | 35 | 11 | 26 | 12 | 22 | 34 | 30 | -1 | 31 |
| 11 | 9 | 25 | 33 | 28 | 38 | 8 | 40 | 45 | 35 | 14 | 26 | 41 | 12 | 14 | 40 | 7 | 31 | 48 | 45 | 13 | 11 | 31 | 36 | 32 | 30 | 22 | 32 | 33 | -1 |