Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОННОЙ ТЕХНИКИ»

Институт системной и программной инженерии

и информационных технологий (СПИНТех)

**Отчёт**

по дисциплине «Алгоритмы и структуры данных»

**Лабораторная работа №4**

**Вариант-10**

Руководитель

\_\_\_\_\_\_\_\_\_\_\_ Волков A. C.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Студент группы ПИН-23

\_\_\_\_\_\_\_\_\_\_\_ Исламов Р. Р.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

*Москва*

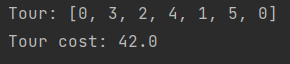
*2023*

# 1.1 Листинг. Задача Коммивояжера

package Lab4.Ex1;  
  
import java.util.\*;  
  
public class Kommivoyager {  
  
 private final int N;  
 private final int START\_NODE;  
 private final int FINISHED\_STATE;  
  
 private double[][] distance;  
 private double minTourCost = Double.*POSITIVE\_INFINITY*;  
  
 private List<Integer> tour = new ArrayList<>();  
 private boolean ranSolver = false;  
  
 public Kommivoyager(double[][] distance) {  
 this(0, distance);  
 }  
  
 public Kommivoyager(int startNode, double[][] distance) {  
  
 this.distance = distance;  
 N = distance.length;  
 START\_NODE = startNode;  
  
 // Validate inputs.  
 if (N <= 2) throw new IllegalStateException("TSP on 0, 1 or 2 nodes doesn't make sense.");  
 if (N != distance[0].length)  
 throw new IllegalArgumentException("Matrix must be square (N x N)");  
 if (START\_NODE < 0 || START\_NODE >= N)  
 throw new IllegalArgumentException("Starting node must be: 0 <= startNode < N");  
 if (N > 32)  
 throw new IllegalArgumentException(  
 "Matrix too large! A matrix that size for the DP TSP problem with a time complexity of"  
 + "O(n^2\*2^n) requires way too much computation for any modern home computer to handle");  
  
 // The finished state is when the finished state mask has all bits are set to  
 // one (meaning all the nodes have been visited).  
 FINISHED\_STATE = (1 << N) - 1;  
 }  
  
 // Returns the optimal tour for the traveling salesman problem.  
 public List<Integer> getTour() {  
 if (!ranSolver) solve();  
 return tour;  
 }  
  
 // Returns the minimal tour cost.  
 public double getTourCost() {  
 if (!ranSolver) solve();  
 return minTourCost;  
 }  
  
 public void solve() {  
  
 // Run the solver  
 int state = 1 << START\_NODE;  
 Double[][] memo = new Double[N][1 << N];  
 Integer[][] prev = new Integer[N][1 << N];  
 minTourCost = tsp(START\_NODE, state, memo, prev);  
  
 // Regenerate path  
 int index = START\_NODE;  
 while (true) {  
 tour.add(index);  
 Integer nextIndex = prev[index][state];  
 if (nextIndex == null) break;  
 int nextState = state | (1 << nextIndex);  
 state = nextState;  
 index = nextIndex;  
 }  
 tour.add(START\_NODE);  
 ranSolver = true;  
 }  
  
 private double tsp(int i, int state, Double[][] memo, Integer[][] prev) {  
  
 // Done this tour. Return cost of going back to start node.  
 if (state == FINISHED\_STATE) return distance[i][START\_NODE];  
  
 // Return cached answer if already computed.  
 if (memo[i][state] != null) return memo[i][state];  
  
 double minCost = Double.*POSITIVE\_INFINITY*;  
 int index = -1;  
 for (int next = 0; next < N; next++) {  
  
 // Skip if the next node has already been visited.  
 if ((state & (1 << next)) != 0) continue;  
  
 int nextState = state | (1 << next);  
 double newCost = distance[i][next] + tsp(next, nextState, memo, prev);  
 if (newCost < minCost) {  
 minCost = newCost;  
 index = next;  
 }  
 }  
  
 prev[i][state] = index;  
 return memo[i][state] = minCost;  
 }  
  
 // Example usage:  
 public static void main(String[] args) {  
  
 // Create adjacency matrix  
 int n = 6;  
 double[][] distanceMatrix = new double[n][n];  
 for (double[] row : distanceMatrix) java.util.Arrays.*fill*(row, 10000);  
 distanceMatrix[1][4] = distanceMatrix[4][1] = 2;  
 distanceMatrix[4][2] = distanceMatrix[2][4] = 4;  
 distanceMatrix[2][3] = distanceMatrix[3][2] = 6;  
 distanceMatrix[3][0] = distanceMatrix[0][3] = 8;  
 distanceMatrix[0][5] = distanceMatrix[5][0] = 10;  
 distanceMatrix[5][1] = distanceMatrix[1][5] = 12;  
  
 // Run the solver  
 Kommivoyager solver = new Kommivoyager(distanceMatrix);  
  
 // Prints: [0, 3, 2, 4, 1, 5, 0]  
 System.*out*.println("Tour: " + solver.getTour());  
  
 // Print: 42.0  
 System.*out*.println("Tour cost: " + solver.getTourCost());  
 }  
}

}

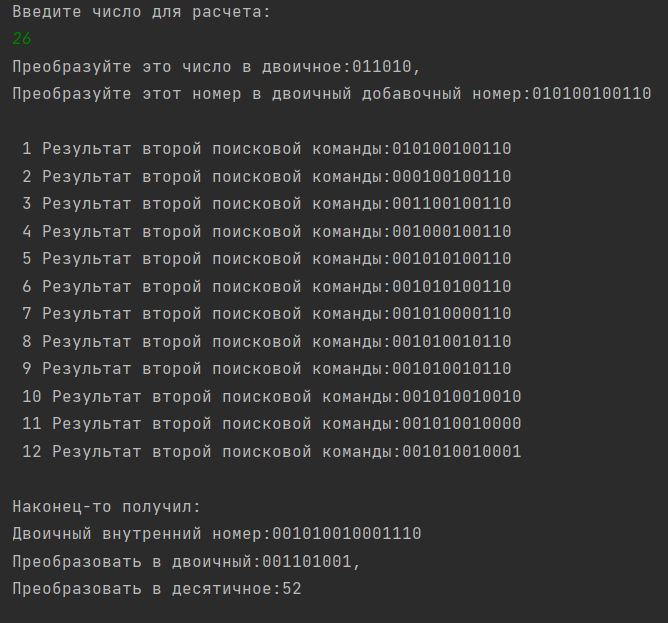
# 1.2 Результат работы



# 2.1 Листинг. Машина Тьюринга

package Lab4.Ex2;  
  
import java.util.Scanner;  
  
public class Turing {  
 public static void main(String[] args) {  
  
 System.*out*.println("Введите число для расчета:");  
 int num = new Scanner(System.*in*).nextInt();  
 if(num == 0) {System.*out*.println("Результат - 0");return;}  
  
 String string = *Transform1*(num);  
  
 char[] ch = *Transform2*(string);  
  
 ch = *Tututu*(ch);  
  
 *Transform3*(ch);  
 }  
  
 public static String Transform1(int num){  
 String string = Integer.*toBinaryString*(num);  
 string = "0" + string + ",";  
 System.*out*.println("Преобразуйте это число в двоичное:"+string);  
 return string;  
 }  
 public static char[] Transform2(String string){  
  
 string = string.replace("1","10");  
 string = string.replace(",","110");  
  
 System.*out*.println("Преобразуйте этот номер в двоичный добавочный номер:"+string);  
 System.*out*.println();  
  
 char[] ch = string.toCharArray();  
 return ch;  
 }  
  
 public static char[] Tututu(char[] ch){  
 int[][] array1 = {{0,0},{0,1},{1,0},{1,1},{10,0},{11,0}};  
 int[][] array2 = {{0,0},{1,0},{0,1},{10,0},{11,1},{0,1}};  
  
 int NT = 0;  
  
 for(int i = 0;i < ch.length;i++){  
 int index = *Search*(NT,ch[i]-'0',array1);  
 NT = array2[index][0];  
 ch[i] = (char)(array2[index][1]+'0');  
 System.*out*.print(" "+(i+1)+" Результат второй поисковой команды:");  
 System.*out*.println(ch);  
 }  
 return ch;  
  
 }  
 public static int Search(int NT,int SR,int[][] array1){  
 for(int i = 0;i < 6;i++) {  
 if (NT == array1[i][0] && SR == array1[i][1]) {  
 return i;  
 }  
 }  
 return -1;  
 }  
 public static void Transform3(char[] ch){  
 String str = new String(ch);  
 str +="110";  
 System.*out*.println();  
 System.*out*.println("Наконец-то получил:");  
 System.*out*.println("Двоичный внутренний номер:"+str);  
  
 str = str.replace("110",",");  
 str = str.replace("10","1");  
 System.*out*.println("Преобразовать в двоичный:"+str);  
  
 int index1 = str.indexOf("1");  
 str = str.substring(index1,str.length()-2);  
 System.*out*.print("Преобразовать в десятичное:");  
 System.*out*.println(Integer.*parseInt*(str,2));  
  
 }  
}

# 2.2 Результат работы



# 3 Контрольные вопросы

Контрольные вопросы по тексту лабораторной 2:

Контрольные вопросы 1. Каковы основные принципы метода ветвей и границ? 2. Какое множество называется рекордом? 3. Как сформулировать условие задачи коммивояжера? 4. Что означает привести матрицу по строкам? 5. Как строится дерево перебора для расшифровки криптограмм? 6. Что такое функция штрафа? 7. Как исключается досрочное завершение тура? 8. Что такое нижняя граничная оценка?

1. Основные принципы метода ветвей и границ:

* Ветвление: задача разделяется на более мелкие подзадачи (ветви), которые могут быть решены независимо друг от друга.
* Оценка: для каждой подзадачи вычисляется нижняя граница (оценка), которая представляет собой оценку оптимального значения для данной подзадачи.
* Отсечение: если текущая оценка для подзадачи хуже, чем текущий лучший результат (рекорд), то данная подзадача отсекается, и поиск продолжается в других ветвях.
* Обновление рекорда: если для подзадачи получено значение лучше текущего рекорда, то рекорд обновляется.

1. Рекорд - это лучшее найденное значение в процессе решения задачи оптимизации. Он представляет собой оптимальное решение или наилучшую известную оценку для данной задачи.
2. Условие задачи коммивояжера формулируется следующим образом: необходимо найти самый короткий путь, проходящий через все заданные города и возвращающийся в исходный город, при условии, что каждый город должен быть посещен только один раз.
3. Привести матрицу по строкам означает переставить элементы матрицы таким образом, чтобы элементы каждой строки находились в определенном порядке. Это может быть сортировка строк по возрастанию или упорядочение элементов строк по определенному критерию.
4. Дерево перебора для расшифровки криптограмм строится путем рассмотрения всех возможных перестановок букв в соответствии с заданным алфавитом и ограничениями. Каждая вершина дерева представляет собой одну перестановку, а ребра соответствуют возможным переходам от одной перестановки к другой, при введении определенных ограничений.
5. Функция штрафа - это функция, используемая для оценки качества или нежелательности определенного решения в задаче оптимизации. Она может учитывать различные факторы, такие как ограничения, цели и предпочтения, и влиять на процесс принятия решений или на алгоритм поиска оптимального решения.
6. Досрочное завершение тура исключается путем применения различных правил или ограничений, которые гарантируют, что тур будет завершен по определенным условиям. Например, можно установить правило, по которому тур не может быть завершен, пока не будут посещены все города, или можно ввести штрафные баллы за досрочное завершение тура, чтобы обеспечить полное выполнение задачи.
7. Нижняя граничная оценка (lower bound) - это оценка минимального значения, которое может принимать целевая функция в задаче оптимизации. Она используется для сравнения с текущими значениями и отсечения подзадач с более плохими оценками. Нижняя граница позволяет уменьшить пространство поиска и сосредоточиться на потенциально оптимальных решениях.

Контрольные вопросы по тексту лабораторной 5:

1. Какова структура машины Тьюринга? 2. Что такое внутренняя и внешняя память машины Тьюринга? 3. В чем заключается особенность построения функциональных таблиц? 4. Какова связь функциональной таблицы с функциональной диаграммой? 5. В чем смысл умножения машин Тьюринга? 6. Что такое итерация машин Тьюринга?

1. Структура машины Тьюринга состоит из следующих элементов:

* Бесконечная лента, разделенная на ячейки, каждая из которых содержит символ.
* Головка, которая может перемещаться влево или вправо по ленте и считывать или записывать символы.
* Состояния, включая начальное состояние и различные промежуточные состояния.
* Функциональная таблица или функциональная диаграмма, определяющая переходы машины между состояниями в зависимости от символов, считанных с ленты.

1. Внутренняя память машины Тьюринга представляет собой состояния и символы, хранящиеся в состояниях и ячейках ленты. Внешняя память машины Тьюринга - это сама лента, на которой хранятся символы.
2. Построение функциональных таблиц заключается в определении переходов машины Тьюринга между состояниями в зависимости от символов, считываемых с ленты. Функциональная таблица представляет собой таблицу, в которой каждая строка соответствует комбинации текущего состояния и символа, а каждый столбец представляет следующее состояние, символ для записи и направление движения головки.
3. Функциональная таблица и функциональная диаграмма связаны и представляют альтернативные способы представления переходов машины Тьюринга. Функциональная таблица позволяет ясно описать все возможные переходы в виде таблицы, где каждая ячейка определяет следующее состояние и действия машины. Функциональная диаграмма представляет собой графическое представление переходов с использованием узлов и дуг, где каждый узел представляет состояние, а каждая дуга - переход между состояниями в зависимости от символов.
4. Умножение машин Тьюринга относится к задаче умножения двух чисел с использованием машины Тьюринга. Машина Тьюринга может быть настроена таким образом, чтобы она могла перемещаться по ленте, складывать, перемножать и выполнять другие операции с символами на ленте, чтобы выполнить умножение чисел.
5. Итерация машин Тьюринга относится к повторяющемуся выполнению операций или переходов машины Тьюринга в цикле. Это может быть использовано для выполнения повторяющихся задач или операций на ленте, пока не будет выполнено определенное условие или достигнуто желаемое состояние.