Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(ФГБОУ ВО «СибГУТИ»)

*Факультет информатики и вычислительной техники*

*Кафедра прикладной математики и кибернетики (ПМ и К)*

Расчетно-графическое задание по дисциплине

«Теория сложности вычислительных процессов и структур»

Вариант 8

Выполнил:

студент гр. ИП-513

Майоров С. А.

Проверил:

старший преподаватель

Разинкина Т. Э.

Новосибирск

2017

**Оглавление**

[I.](#_30j0zll) Задача: 3

[II.](#_1fob9te) Теоретическая часть. 3

[III.](#_3znysh7) Листинг 4

[IV.](#_2et92p0) Результаты 8

# Задача:

Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одновременного ветвления). Сравнить трудоемкость двух алгоритмов.

С=

(b 13 7 5 2 9

8 b 4 7 5 b

8 4 b 3 6 2

5 8 1 b 0 1

b 6 1 4 b b

0 0 b 3 7 b)

# Теоретическая часть.

**Элементы теории графов**

Граф – пара G(V,E), где V = {V1,V2, …,Vn} – множество вершин графа G, E – множество ребёр (дуг) графа: Е={ (V1i, V2i) }.  
Рёбра можно задавать разными способами, например матрицей инцидентности или списком ребёр.  
Взвешенный граф – граф, каждой дуге которого приписан её вес.  
Цикл в графе – маршрут, заданный последовательностью вершин, каждая вершина посещается по одному разу, и начальная вершина совпадает с конечной.

**Задача коммивояжера**

*Имеется взвешенный неориентированный связный граф. Необходимо найти гамильтонов цикл наименьшей длины, т.е. нужно обойти все вершины графа, побывав в каждой из них по одному разу, затратив как можно меньше денег.*  
Оценим трудоемкость методом прямого перебора.  
Имеем n! всевозможных маршрутов. Стоимость каждого маршрута – сумма n ребер. Следовательно,  
T(n) = n!·n  
n! можно заменить (n-1)!, т.к. маршрут проходит через все вершины, и поэтому в качестве стартовой мы можем взять вершину vi, располагая оставшиеся вершины (n-1) произвольным образом.  
T(n) = (n-1)!·n = n! – неполиномиальная.  
Для более быстрого решения нашей задачи существует метод ветвей и границ. На самом деле это целая группа методов, они используются для решения множества задач. Их объединяет общая идея, но в каждом случае реализация метода ветвей и границ своя. Впервые этот мтеод был придуман для решения задачи коммивояжера.

**Метод ветвей и границ**

Метод ветвей и границ — общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. По существу, метод является вариацией полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.  
Общая идея метода может быть описана на примере поиска минимума функции f(x) на множестве допустимых значений переменной x. Функция f и переменная x могут быть произвольной природы. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).  
Процедура ветвления состоит в разбиении множества допустимых значений переменной x на подобласти (подмножества) меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое деревом поиска или деревом ветвей и границ. Узлами этого дерева являются построенные подобласти (подмножества множества значений переменной x).  
  
Процедура нахождения оценок заключается в поиске верхних и нижних границ для решения задачи на подобласти допустимых значений переменной x.  
В основе метода ветвей и границ лежит следующая идея: если нижняя граница значений функции на подобласти A дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (правило отсева). Обычно минимальную из полученных верхних оценок записывают в глобальную переменную m; любой узел дерева поиска, нижняя граница которого больше значения m, может быть исключён из дальнейшего рассмотрения.  
Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

# Листинг

|  |
| --- |
| package ru.labs.fnick;  import java.util.ArrayList; import java.util.Arrays;  public class TSP {  static int T = 0;  static Route bestRoute = new Route();  final static int Infinity = Integer.MAX\_VALUE;   private static class Route  {  ArrayList<Integer> route;  int lentgth;    public Route()  {  route = new ArrayList<>();  lentgth = 0;  }  }   private static void bfRouteSearcher(int[][] c, Route r, ArrayList<Integer> verticesNotInRoute)  {  int firstIndex = r.route.get(0) - 1;  int lastIndex = r.route.get(r.route.size() - 1) - 1;   if(!verticesNotInRoute.isEmpty())  {  for(int i = 0; i<verticesNotInRoute.size(); ++i)  {  Integer justRemoved = (Integer) verticesNotInRoute.remove(0);  Route newRoute = new Route();  newRoute.route = (ArrayList<Integer>) r.route.clone();  newRoute.route.add(justRemoved);  if (c[lastIndex][justRemoved - 1] != Infinity && r.lentgth != Infinity)  newRoute.lentgth = r.lentgth + c[lastIndex][justRemoved - 1];  else  newRoute.lentgth = Infinity;   bfRouteSearcher(c, newRoute, verticesNotInRoute);  verticesNotInRoute.add(justRemoved);  }  }  else  {  T += r.route.size();  if (c[lastIndex][firstIndex] == Infinity || r.lentgth == Infinity)  return;  r.lentgth += c[lastIndex][firstIndex];  if(r.lentgth < bestRoute.lentgth)  bestRoute = r;  }  }   public static void bruteforce(int[][] c)  {  T = 0;  int n = c.length;  bestRoute = new Route();  bestRoute.lentgth = Infinity;   ArrayList<Integer> lst = new ArrayList<Integer>();  for (int i = 2; i <= n; ++i)  lst.add(i);  Route route = new Route();  route.route.add(1);   bfRouteSearcher(c, route, lst);  }   public static void bnbRouteSearcher(int[][] c, Route r, ArrayList<Integer> verticesNotInRoute, int record)  {  int n = c.length;   int firstIndex = r.route.get(0) - 1;  int lastIndex = r.route.get(r.route.size() - 1) - 1;   for(int i = 0; i<verticesNotInRoute.size(); ++i)  {  Integer justRemoved = (Integer) verticesNotInRoute.remove(0);  Route newRoute = new Route();  newRoute.route = (ArrayList<Integer>) r.route.clone();  newRoute.route.add(justRemoved);   if (c[lastIndex][justRemoved - 1] != Infinity)  {  newRoute.lentgth = r.lentgth + c[lastIndex][justRemoved - 1];   int alpha, beta;  int h = newRoute.lentgth;   int[][] c\_new = new int[n][n];  for (int j = 0; j < n; ++j)  System.arraycopy(c[j], 0, c\_new[j], 0, n);  c\_new[justRemoved - 1][firstIndex] = Infinity;   for (Integer j = 0; j < n; ++j)  {  boolean out = false;  if (r.route.contains(j + 1))  out = true;  if (!out)  {  alpha = Infinity;   for (Integer k = 0; k < n; ++k)  {  boolean in = false;  newRoute.route.remove(0);  if (newRoute.route.contains(k + 1))  in = true;  newRoute.route.add(0, firstIndex + 1);   if (!in)  alpha = Math.min(alpha, c\_new[j][k]);  }   if (alpha != Infinity)  h += alpha;  else  {  h = Infinity;  break;  }   for (Integer k = 0; k < n; ++k)  if (c\_new[j][k] != Infinity)  c\_new[j][k] -= alpha;  }  }   for (Integer k = 0; k < n; ++k)  {  boolean in = false;  newRoute.route.remove(0);  if (newRoute.route.contains(k + 1))  in = true;  newRoute.route.add(0, firstIndex + 1);  if (!in)  {  beta = Infinity;   for (Integer j = 0; j < n; ++j)  {  boolean out = false;  if (r.route.contains(j + 1))  out = true;   if (!out)  beta = Math.min(beta, c\_new[j][k]);  }   if (beta != Infinity && h != Infinity)  h += beta;  else  {  h = Infinity;  break;  }  }  }   if (h < record)  {  if (verticesNotInRoute.size() == 1)  {  Integer lastRemoved = (Integer) verticesNotInRoute.remove(0);  newRoute.route.add(lastRemoved);  newRoute.lentgth = h;  if (h < bestRoute.lentgth)  bestRoute = newRoute;  verticesNotInRoute.add(lastRemoved);   T += newRoute.route.size();  } else  bnbRouteSearcher(c, newRoute, verticesNotInRoute, record);  } else  T += (newRoute.route.size() - 1);  } else  T += (newRoute.route.size() - 1);   verticesNotInRoute.add(justRemoved);  }  }   public static void branchNbound(int[][] c)  {  int n = c.length;   bestRoute = new Route();  bestRoute.route.add(1);  Route route = new Route();  route.route.add(1);   ArrayList<Integer> lst = new ArrayList<Integer>();  for (int i = 2; i <= n; ++i)  lst.add(i);  ArrayList<Integer> vertices = (ArrayList<Integer>) lst.clone();   int lastVertex = 1;  while (!lst.isEmpty())  {  Integer min = lst.get(0);  for (Integer v : lst)  if (c[lastVertex - 1][v - 1] < c[lastVertex - 1][min - 1])  min = v;  bestRoute.route.add(min);  if (bestRoute.lentgth != Infinity && c[lastVertex - 1][min - 1] != Infinity)  bestRoute.lentgth += c[lastVertex - 1][min - 1];  else  bestRoute.lentgth = Infinity;  lastVertex = min;  lst.remove(min);  }  if (bestRoute.lentgth != Infinity && c[lastVertex - 1][0] != Infinity)  bestRoute.lentgth += c[lastVertex - 1][0];  else  bestRoute.lentgth = Infinity;   T = bestRoute.route.size();   bnbRouteSearcher(c, route, vertices, bestRoute.lentgth);  }   public static void main(String[] args)  {  int b = Infinity;  int[][] c = {{b, 13, 7, 5, 2, 9},  {8, b, 4, 7, 5, b},  {8, 4, b, 3, 6, 2},  {5, 8, 1, b, 0, 1},  {b, 6, 1, 4, b, b},  {0, 0, b, 3, 7, b}};  bruteforce(c);  System.out.println("Best route: " + bestRoute.route.toString() + " for " + bestRoute.lentgth + "\nT = " + T);   branchNbound(c);  System.out.println("Best route: " + bestRoute.route.toString() + " for " + bestRoute.lentgth + "\nT = " + T);  } } |

# Результаты

С заданными входными данными результат получился следующим (сверху - простой перебор, снизу - метод ветвей и границ):

