**Белорусский государственный технологический университет**

**Факультет информационных технологий**

**Специальность программная инженерия**

Отчёт по лабораторной работе №3

По дисциплине «Математическое программирование»

На тему «Метод ветвей и границ. Задача коммивояжера и методы её решения»

Выполнил:

Студент 2 курса 8 группы

Мамонько Денис Александрович

Преподаватель: асс. Ромыш А.С.

2025, Минск

**Вариант 7**

**ЦЕЛЬ РАБОТЫ:** освоить общие принципы решения задач методом ветвей и границ, решить задачу о коммивояжере данным методом, сравнить полученное решение задачи с комбинаторным методом перестановок.

**Задание 1.** Сформулировать условие задачи коммивояжера с параметром.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Город | 1 | 2 | 3 | 4 | 5 |
| 1 | INF | 14 | 28 | INF | 7 |
| 2 | 7 | INF | 22 | 61 | 77 |
| 3 | 9 | 21 | INF | 86 | 56 |
| 4 | 24 | 51 | 28 | INF | 21 |
| 5 | 86 | 73 | 52 | 20 | INF |

**Алгоритм коммивояжёра** – это классическая задача оптимизации, целью которой является нахождение кратчайшего маршрута, который позволяет посетить каждый город ровно один раз и вернуться в исходный город. В нашей задаче мы будем работать с матрицей городов, где в качестве городов выступают вершины, а расстояния между ними – ребра с каким-либо числовым весом. Также на некоторых рёбрах вес отсутствует и это значение помечено как INF. Это значит, что расстояние между городами отсутствует. Матрицу расстояний по-другому еще называют ***графом***.

**Граф** – структура данных, состоящая из объектов, называемых вершинами, и связей между этими вершинами, которые называются ребрами.

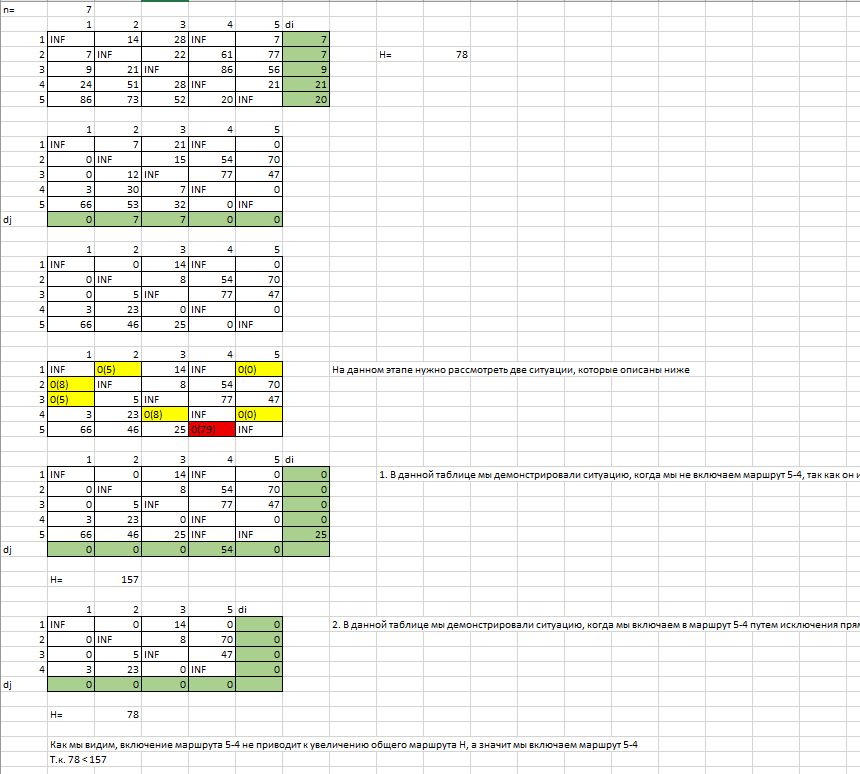
Решение задачи коммивояжера с параметром включает в себя использование ***цикла Гамильтона***.

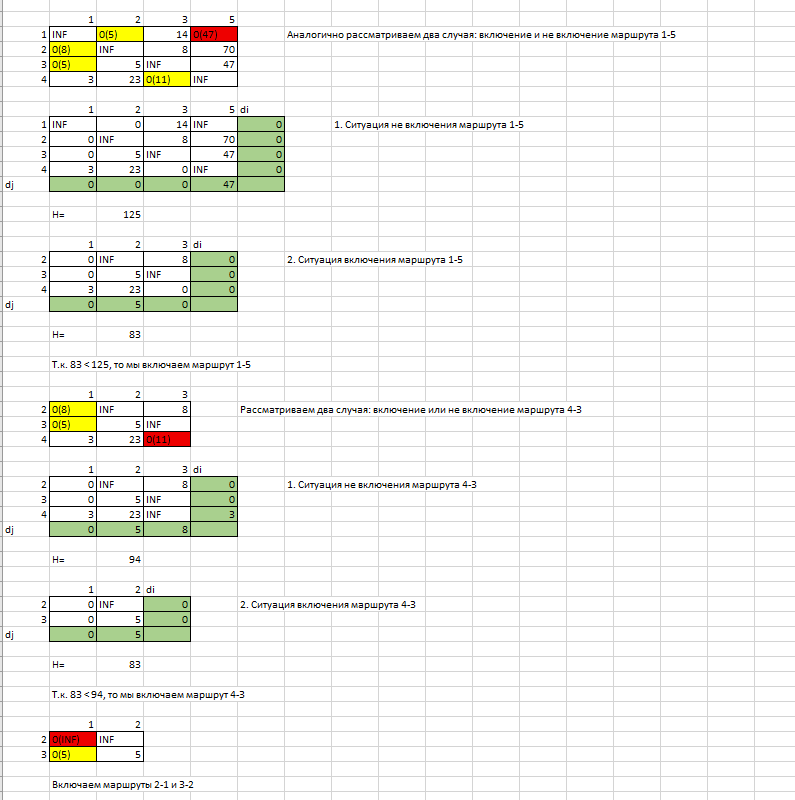
**Цикл Гамильтона** – маршрут, который проходит каждую вершину графа ровно один раз и возвращается в исходную вершину.  
Результатом задачи является нахождения кратчайшего пути в цикле Гамильтона.

**Стоимость пути (кратчайший путь)** – сумма длин всех ребер в цикле Гамильтона.

**Цель задачи коммивояжера с параметром** – найти гамильтонов цикл минимального веса в заданном графе, который удовлетворяет ограничению на максимальный вес ребра. Значение параметра может быть задано заранее или получено в результате выч ислений, и влияет на решение задачи.

**Задание 2.** Решить сформулированную задачу методом ветвей и границ.





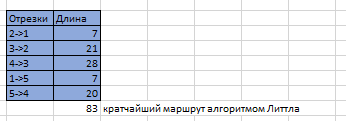


Рисунок 1 – Решение алгоритма Литтла вручную

В результате решения данной задачи, мы получили, что оптимальным маршрутом является следующий: **1->5->4->3->2->1** и его длина составляет **83**.

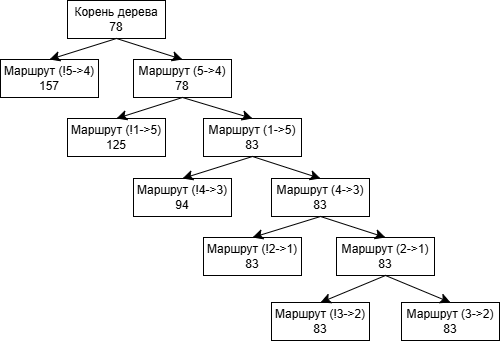


Рисунок 2 – Обход исходной матрицы

**Задание 2\*.** Предложить решение задачи коммивояжера методом вервей и границ на языке C++. Код, анализ и вывод включить в отчет.

|  |
| --- |
| #include <iostream>  #include <limits>  using namespace std;  const int N = 5;  int final\_path[N + 1];  bool visited[N];  int final\_res = INT\_MAX;  void copyToFinal(int curr\_path[])  {  for (int i = 0; i < N; i++)  final\_path[i] = curr\_path[i];  final\_path[N] = curr\_path[0];  }  int firstMin(int matrix[N][N], int i)  {  int min = INT\_MAX;  for (int k = 0; k < N; k++)  if (matrix[i][k] < min && i != k)  min = matrix[i][k];  return min;  }  int secondMin(int matrix[N][N], int i)  {  int first = INT\_MAX, second = INT\_MAX;  for (int j = 0; j < N; j++)  {  if (i == j)  continue;  if (matrix[i][j] <= first)  {  second = first;  first = matrix[i][j];  }  else if (matrix[i][j] <= second &&  matrix[i][j] != first)  second = matrix[i][j];  }  return second;  }  void TSPRec(int matrix[N][N], int curr\_bound, int curr\_weight,  int level, int curr\_path[])  {  if (level == N)  {  if (matrix[curr\_path[level - 1]][curr\_path[0]] != 0)  {  int curr\_res = curr\_weight +  matrix[curr\_path[level - 1]][curr\_path[0]];  if (curr\_res < final\_res)  {  copyToFinal(curr\_path);  final\_res = curr\_res;  }  }  return;  }  for (int i = 0; i < N; i++)  {  if (matrix[curr\_path[level - 1]][i] != 0 &&  visited[i] == false)  {  int temp = curr\_bound;  curr\_weight += matrix[curr\_path[level - 1]][i];  if (level == 1)  curr\_bound -= ((firstMin(matrix, curr\_path[level - 1]) +  firstMin(matrix, i)) / 2);  else  curr\_bound -= ((secondMin(matrix, curr\_path[level - 1]) +  firstMin(matrix, i)) / 2);  if (curr\_bound + curr\_weight < final\_res)  {  curr\_path[level] = i;  visited[i] = true;  TSPRec(matrix, curr\_bound, curr\_weight, level + 1,  curr\_path);  }  curr\_weight -= matrix[curr\_path[level - 1]][i];  curr\_bound = temp;  memset(visited, false, sizeof(visited));  for (int j = 0; j <= level - 1; j++)  visited[curr\_path[j]] = true;  }  }  }  void TSP(int matrix[N][N])  {  int curr\_path[N + 1];  int curr\_bound = 0;  memset(curr\_path, -1, sizeof(curr\_path));  memset(visited, 0, sizeof(curr\_path));  for (int i = 0; i < N; i++)  curr\_bound += (firstMin(matrix, i) +  secondMin(matrix, i));  curr\_bound = (curr\_bound & 1) ? curr\_bound / 2 + 1 :  curr\_bound / 2;  visited[0] = true;  curr\_path[0] = 0;  TSPRec(matrix, curr\_bound, 0, 1, curr\_path);  }  int main()  {  setlocale(LC\_ALL, "RU");  int matrix[N][N] = {  {0, 14, 28, 0, 7},  {7, 0, 22, 61, 77},  {9, 21, 0, 86, 56},  {24, 51, 28, 0, 21},  {86, 73, 52, 20, 0},  };  TSP(matrix);  cout << "Длина минимального маршрута : " << final\_res << endl;  cout << "Последовательность вершин : ";  for (int i = 0; i <= N; i++)  cout << final\_path[i] << " ";  return 0;  } |

Листинг 2.1 – Алгоритм Литтла



Рисунок 3 – Результат работы программы

**Задание 3.** Проверить полученное решение при помощи генератора перестановок (см. лаб. 2, задание 5.1.) и включить копию экрана с решением в отчет.

|  |
| --- |
| #pragma once  namespace combi  {  struct permutation  {  const static bool L = true;  const static bool R = false;  short n,  \* sset;  bool\* dart;  permutation(short n = 1);  void reset();  \_\_int64 getfirst();  \_\_int64 getnext();  short ntx(short i);  unsigned \_\_int64 np;  unsigned \_\_int64 count() const;  };  }; |

Листинг 3.1 – Файл Combi.h

|  |
| --- |
| #include "Combi.h"  #include <algorithm>  #define NINF ((short)0x8000)  namespace combi  {  permutation::permutation(short n)  {  this->n = n;  this->sset = new short[n];  this->dart = new bool[n];  this->reset();  };  void permutation::reset()  {  this->getfirst();  };  \_\_int64 permutation::getfirst()  {  this->np = 0;  for (int i = 0; i < this->n; i++)  {  this->sset[i] = i; this->dart[i] = L;  };  return (this->n > 0) ? this->np : -1;  };  \_\_int64 permutation::getnext()  {  \_\_int64 rc = -1;  short maxm = NINF, idx = -1;  for (int i = 0; i < this->n; i++)  {  if (i > 0 &&  this->dart[i] == L &&  this->sset[i] > this->sset[i - 1] &&  maxm < this->sset[i]) maxm = this->sset[idx = i];  if (i < (this->n - 1) &&  this->dart[i] == R &&  this->sset[i] > this->sset[i + 1] &&  maxm < this->sset[i]) maxm = this->sset[idx = i];  };  if (idx >= 0)  {  std::swap(this->sset[idx],  this->sset[idx + (this->dart[idx] == L ? -1 : 1)]);  std::swap(this->dart[idx],  this->dart[idx + (this->dart[idx] == L ? -1 : 1)]);  for (int i = 0; i < this->n; i++)  if (this->sset[i] > maxm) this->dart[i] = !this->dart[i];  rc = ++this->np;  }  return rc;  };  short permutation::ntx(short i) { return this->sset[i]; };  unsigned \_\_int64 fact(unsigned \_\_int64 x) { return (x == 0) ? 1 : (x \* fact(x - 1)); };  unsigned \_\_int64 permutation::count() const { return fact(this->n); };  } |

Листинг 3.2 – Файл Combi.cpp

|  |
| --- |
| #pragma once  #define INF 0x7fffffff  #include "Combi.h"  int salesman(  int n,  const int\* d,  int\* r  ); |

Листинг 3.3 – Файл TSP.h

|  |
| --- |
| #include "TSP.h"  int sum(int x1, int x2)  {  return (x1 == INF || x2 == INF) ? INF : (x1 + x2);  };  int\* firstpath(int n)  {  int\* rc = new int[n + 1];  rc[n] = 0;  for (int i = 0; i < n; i++)  rc[i] = i;  return rc;  };  int\* source(int n)  {  int\* rc = new int[n - 1];  for (int i = 1; i < n; i++) rc[i - 1] = i;  return rc;  };  void copypath(int n, int\* r1, const int\* r2)  {  for (int i = 0; i < n; i++)  r1[i] = r2[i];  };  int distance(int n, int\* r, const int\* d)  {  int rc = 0;  for (int i = 0; i < n - 1; i++)  rc = sum(rc, d[r[i] \* n + r[i + 1]]);  return sum(rc, d[r[n - 1] \* n + 0]);  };  void indx(int n, int\* r, const int\* s, const short\* ntx)  {  for (int i = 1; i < n; i++)  r[i] = s[ntx[i - 1]];  }  int salesman(  int n,  const int\* d,  int\* r  )  {  int\* s = source(n), \* b = firstpath(n), rc = INF, dist = 0;  combi::permutation p(n - 1);  int k = p.getfirst();  while (k >= 0)  {  indx(n, b, s, p.sset);  if ((dist = distance(n, b, d)) < rc)  {  rc = dist; copypath(n, r, b);  }  k = p.getnext();  };  return rc;  } |

Листинг 3.4 – Файл TSP.cpp

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include "TSP.h"  #define N 5  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  int d[N][N] = {  { INF, 14, 28, INF, 7 },  { 7, INF, 22, 61, 77 },  { 9, 21, INF, 86, 56 },  { 24, 51, 28, INF, 21 },  { 86, 73, 52, 20, INF }  };  int r[N];  int s = salesman(  N,  (int\*)d,  r  );  cout << "\n-- Задача коммивояжера -- ";  cout << "\n-- количество городов: " << N;  cout << "\n-- матрица расстояний : ";  for (int i = 0; i < N; i++)  {  cout << "\n";  for (int j = 0; j < N; j++)  if (d[i][j] != INF) cout << setw(3) << d[i][j] << " ";  else cout << setw(3) << "INF" << " ";  }  cout << "\n-- оптимальный маршрут: ";  for (int i = 0; i < N; i++)  cout << r[i] + 1 << "-->";  cout << 1;  cout << "\n-- длина маршрута : " << s << "\n";  system("pause");  return 0;  } |

Листинг 3.5 – Решение задачи коммивояжера методом генератора перестановок

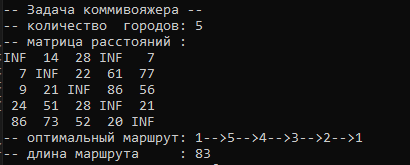


Рисунок 4 – Результат работы программы

Исходя из результатов программ, можно заметить, что решения всеми 3 способами, которые были упомянуты выше, совпали. Был найден одинаковый маршрут и его длина.

**Вывод**: результаты после решения вручную, после использования генератора перестановок и после решения методом ветвей и границ совпали.