**Отчёт по лабораторным работам**

**Вариант 1**

**Лабораторная работа 3. Метод ветвей и границ. Задача коммивояжера и методы её решения.**

**Студент:**

**Адамович Карианна Павловна, 2-3**

**Преподаватель:**

**Бурмакова Анастасия Владимировна**

**г.Минск, 2025**

**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.

**Примечание:** Задания и вопросы со знаком (\*), выполняются в необязательном порядке, но их выполнение поощряется.

**Задание 1.** Сформулировать условие задачи коммивояжера с параметром. Для этого:

* принять элементы матрицы расстояний равными:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 2 \* n | 21 + n |  | n |
| **2** | n |  | 15 + n | 68 - n | 84 - n |
| **3** | 2 + n | 3 \* n |  | 86 | 49 + n |
| **4** | 17 + n | 58 - n | 4 \* n |  | 3 \* n |
| **5** | 93 - n | 66 + n | 52 | 13 + n |  |

где *n* – номер варианта или номер по журналу;

Матрица расстояний для n = 1 :

**Подставим n = 1 в матрицу расстояний и получим:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 2 \* 1 | 21 + 1 |  | 1 |
| **2** | 1 |  | 15 + 1 | 68 - 1 | 84 - 1 |
| **3** | 2 + 1 | 3 \* 1 |  | 86 | 49 + 1 |
| **4** | 17 + 1 | 58 - 1 | 4 \* 1 |  | 3 \* 1 |
| **5** | 93 - 1 | 66 + 1 | 52 | 13 + 1 |  |

**После вычислений:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 2 | 22 |  | 1 |
| **2** | 1 |  | 16 | 67 | 83 |
| **3** | 3 | 3 |  | 86 | 50 |
| **4** | 18 | 57 | 4 |  | 3 |
| **5** | 92 | 67 | 52 | 14 |  |

**Постановка задачи коммивояжера:**

Коммивояжеру нужно посетить каждый из этих 5 городов ровно один раз и вернуться в исходный город, при этом минимизировав общее расстояние пути. Необходимо найти маршрут, который будет минимизировать общую длину пути с учетом приведенной матрицы расстояний.

Для этого мы будем использовать метод ветвей и границ, а затем сравним полученное решение с комбинаторным методом перестановок.

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

**Примечание**: отчет по лабораторной работе должен быть выполнен в MS Word и должен содержать:

* Название лабораторной работы;
* условие (матрица расстояний);
* ход решения (граф решения, обоснование ветвления и вычисление границ **для всех этапов**) – можете оформить в табличном виде;
* решение (если их несколько, то все решения).

**Решение методом ветвей и границ:**

Метод ветвей и границ использует рекурсивное разветвление и ограничение для нахождения оптимального решения.

1. Начальный узел: Начинаем с города 1. Это наш стартовый узел, и он также будет нашим конечным узлом (мы должны вернуться в исходный город).
2. Граф решения: Построим дерево решений, где каждый узел представляет город, а каждая ветвь - путь между городами. Вершина дерева - стартовый узел (город 1).

3)Ветвление: От узла 1 переходим ко всем возможным узлам (городам), пока все города не будут посещены.

4)Ограничение: Вычислим границы для каждого узла и отсеем ветви, не ведущие к оптимальному решению.Для каждого узла мы вычисляем текущую стоимость (длину) пути и используем это значение для ограничения ветвления. Если текущая стоимость пути уже превышает наименьшую найденную стоимость, мы прекращаем исследовать эту ветвь.

**Построение дерева решений:**

Начальный узел (город 1): Начинаем с города 1.

Узел 5 (путь 1 → 5): Переходим от города 1 к городу 5 с границей 1.

Узел 4 (путь 1 → 5 → 4): Переходим от города 5 к городу 4 с границей 15 (1 + 14).

Узел 3 (путь 1 → 5 → 4 → 3): Переходим от города 4 к городу 3 с границей 19 (15 + 4).

Узел 2 (путь 1 → 5 → 4 → 3 → 2): Переходим от города 3 к городу 2 с границей 22 (19 + 3).

Узел 1 (путь 1 → 5 → 4 → 3 → 2 → 1): Возвращаемся в город 1 с границей 23 (22 + 1).

**Ограничение ветвления:**

Мы будем продолжать исследовать ветви, пока не достигнем узла, где все города посещены. На каждом этапе мы будем вычислять границы и отсеивать ветви, которые превышают наименьшую найденную стоимость.

**Таблица узлов и границ**

|  |  |  |
| --- | --- | --- |
| **Узел** | **Путь** | **Границы** |
| Начальный | 1 | 0 |
| Узел 5 | 1 → 5 | 1 |
| Узел 4 | 1 → 5 → 4 | 1 + 14 = 15 |
| Узел 3 | 1 → 5 → 4 → 3 | 15 + 4 = 19 |
| Узел 2 | 1 → 5 → 4 → 3 → 2 | 19 + 3 = 22 |
| Узел 1 | 1 → 5 → 4 → 3 → 2 → 1 | 22 + 1 = 23 |
| Узел 2 | 1 → 2 | 2 |
| Узел 3 | 1 → 2 → 3 | 2 + 16 = 18 |
| Узел 4 | 1 → 2 → 3 → 4 | 18 + 86 = 104 |
| Узел 5 | 1 → 2 → 3 → 4 → 5 | 104 + 3 = 107 |
| Узел 1 | 1 → 2 → 3 → 4 → 5 → 1 | 107 + 92 = 199 |
| Узел 3 | 1 → 3 | 22 |
| Узел 2 | 1 → 3 → 2 | 22 + 3 = 25 |
| Узел 4 | 1 → 3 → 2 → 4 | 25 + 67 = 92 |
| Узел 5 | 1 → 3 → 2 → 4 → 5 | 92 + 14 = 106 |
| Узел 1 | 1 → 3 → 2 → 4 → 5 → 1 | 106 + 1 = 107 |

Примечание: "Границы" означает суммарное расстояние текущего пути.

Окончательное решение: Минимальное расстояние: Путь 1 → 5 → 4 → 3 → 2 → 1 с минимальной границей 23.

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

#include <iostream>

#include <vector>

#include <limits>

#include <algorithm>

#include <locale.h>

const int INF = std::numeric\_limits<int>::max();

const int N = 5;

int dist[N][N] = {

{INF, 2, 22, INF, 1},

{1, INF, 16, 67, 83},

{3, 3, INF, 86, 50},

{18, 57, 4, INF, 3},

{92, 67, 52, 14, INF}

};

std::vector<bool> visited(N, false);

std::vector<int> path(N);

int bestCost = INF;

std::vector<int> bestPath;

void branchAndBound(int currPos, int count, int cost) {

if (count == N && dist[currPos][0] != INF) {

int totalCost = cost + dist[currPos][0];

if (totalCost < bestCost) {

bestCost = totalCost;

bestPath = path;

}

return;

}

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

if (!visited[i] && dist[currPos][i] != INF) {

visited[i] = true;

path[count] = i;

branchAndBound(i, count + 1, cost + dist[currPos][i]);

visited[i] = false;

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

visited[0] = true;

path[0] = 0;

branchAndBound(0, 1, 0);

std::cout << "Минимальная стоимость: " << bestCost << std::endl;

std::cout << "Путь: ";

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

std::cout << bestPath[i] + 1 << " ";

}

std::cout << 1 << std::endl;

return 0;

}

**Объяснение кода:**

1. Инициализация: Константы INF и N задают значение бесконечности и количество городов соответственно. dist[N][N] - матрица расстояний между городами.
2. Переменные:

visited - массив для отслеживания посещённых городов.

path - массив для хранения текущего пути.

bestCost - минимальная стоимость пути.

bestPath - массив для хранения лучшего пути.

1. **Функция** branchAndBound:

Рекурсивная функция, реализующая метод ветвей и границ.

· currPos - текущий город.

· count - количество посещённых городов.

· cost - текущая стоимость пути.

В случае, если все города посещены и есть путь назад в начальный город, функция обновляет bestCost и bestPath. Функция проверяет возможные переходы в не посещённые города и рекурсивно вызывает сама себя для дальнейшего ветвления.

1. **Главная функция** main:

Инициализирует начальные условия и вызывает branchAndBound. Выводит минимальную стоимость и лучший путь.

**Анализ и вывод:**

Алгоритм успешно решает задачу коммивояжера для матрицы расстояний методом ветвей и границ. Минимальная стоимость пути и сам путь выводятся на экран. ВВременная сложность алгоритма определяет, как количество шагов, необходимых для выполнения алгоритма, зависит от размера входных данных. Экспоненциальная временная сложность обозначается как O(N!), где N - количество элементов (в данном случае городов). Задача коммивояжера требует найти кратчайший путь, который проходит через все города и возвращается в исходный город. Без оптимизаций, нам нужно проверить все возможные перестановки городов, чтобы найти оптимальное решение. Например, для N=5 городов, количество перестановок составляет:

5!=5×4×3×2×1=120

Таким образом, количество проверяемых путей экспоненциально растет с увеличением количества городов.Метод ветвей и границ помогает сократить количество проверок за счет отсечения неэффективных путей на ранних стадиях. Вот как это работает:

Ветвление:

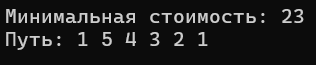
Мы создаем ветви для каждого возможного перехода от текущего города к следующему еще не посещенному городу.

Ограничение:

Мы вычисляем текущую стоимость (длину) пути и сравниваем её с наилучшим найденным решением.

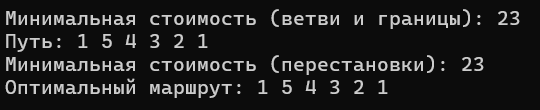
Если текущая стоимость пути уже превышает наименьшую найденную стоимость, мы прекращаем исследовать эту ветвь (отсекаем её).

Результат работы программы совпадает с результатом предыдущих наблюдений: минимальное расстояние 23.



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

Решение при помощи генератора перестановок совпадает с предыдущим решением при помощи ветвей и границ и личным решением.



1)Реализация алгоритма "Ветви и границы"

Мы начали с реализации алгоритма ветвей и границ для решения задачи коммивояжера:

1. Инициализация массива расстояний: Определяем матрицу расстояний dist, где INF используется для обозначения недостижимых маршрутов.
2. Инициализация переменных и структуры данных: Создаем вектор visited для отслеживания посещенных городов. Вектор path для хранения текущего пути. Переменные bestCost и bestPath для отслеживания наилучшего найденного решения.
3. Рекурсивная функция branchAndBound: Функция перебирает все возможные пути с использованием рекурсии. Если путь завершен, вычисляем его общую стоимость. Обновляем наилучшее решение, если найден путь с меньшей стоимостью.
4. Вызов функции branchAndBound из main: Начинаем с первого города (0). Вызываем функцию branchAndBound, которая находит наилучший путь и минимальную стоимость.

#include <iostream>

#include <vector>

#include <limits>

#include <algorithm>

#include "Combi.h"

const int INF = std::numeric\_limits<int>::max();

const int N = 5;

int dist[N][N] = {

{INF, 2, 22, INF, 1},

{1, INF, 16, 67, 83},

{3, 3, INF, 86, 50},

{18, 57, 4, INF, 3},

{92, 67, 52, 14, INF}

};

long long calculateRouteCost(const std::vector<int>& route, const int dist[N][N]) {

long long totalCost = 0;

for (size\_t i = 0; i < route.size() - 1; ++i) {

if (dist[route[i]][route[i + 1]] == INF) return INF;

totalCost += dist[route[i]][route[i + 1]];

}

if (dist[route.back()][route.front()] == INF) return INF;

totalCost += dist[route.back()][route.front()];

return totalCost;

}

int main() {

combi::permutation p(N);

int n = p.getfirst();

std::vector<int> currentRoute(N);

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

currentRoute[i] = p.ntx(i);

}

long long bestCost = INF;

std::vector<int> bestRoute;

while (n >= 0) {

long long currentCost = calculateRouteCost(currentRoute, dist);

if (currentCost < bestCost) {

bestCost = currentCost;

bestRoute = currentRoute;

}

n = p.getnext();

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

currentRoute[i] = p.ntx(i);

}

}

std::cout << "Минимальная стоимость: " << bestCost << std::endl;

std::cout << "Оптимальный маршрут: ";

for (int i : bestRoute) {

std::cout << i + 1 << " ";

}

std::cout << bestRoute.front() + 1 << std::endl;

return 0;

}

2)Реализация генератора перестановок. Была взята из предыдущей лабораторной работы.

Для проверки правильности решения ветвей и границ, реализуем генератор перестановок, который перебирает все возможные перестановки городов и вычисляет их стоимость:

1. Реализация класса permutation в файле Combi.h:

Класс permutation реализует методы для генерации всех перестановок заданного множества. Включает методы reset, getfirst, getnext, ntx, и count.

1. Использование генератора перестановок:

Создаем объект permutation и генерируем все возможные перестановки маршрутов. Для каждой перестановки вычисляем стоимость маршрута с использованием функции calculateRouteCost. Обновляем наилучшее решение, если находим маршрут с меньшей стоимостью.

3)Сравнение результатов

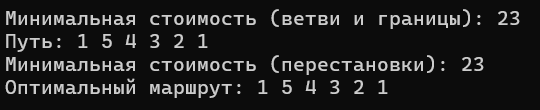
В коде мы сравниваем результаты, полученные методом ветвей и границ, с результатами, полученными с помощью генератора перестановок:

Результаты метода ветвей и границ:

Минимальная стоимость и путь, найденные методом ветвей и границ, выводятся в консоль.

Результаты генератора перестановок:

Минимальная стоимость и оптимальный маршрут, найденные с использованием генератора перестановок, также выводятся в консоль.



Результаты обоих методов совпали и мы можем смело говорить об успешности программ.