Учреждение образования

«Белорусский государственный технологический университет»

**Лабораторная работа №4**

**Динамическое программирование**

Выполнил:

Студент 2 курса 7 группы ФИТ

Володькин Никифор Дмитриевич

**ЦЕЛЬ РАБОТЫ: освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.**

***Задание 1***. На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной 200.

Была реализована программа на языке C++ для генерация случайных строк.

#include <iostream>

#include <string>

int main()

{

srand(time(NULL));

std::string s1 = "";

for (int i = 0; i < 300; i++)

{

s1 += ('a' + rand() % 26);

}

std::string s2 = "";

for (int i = 0; i < 200; i++)

{

s2 += ('a' + rand() % 26);

}

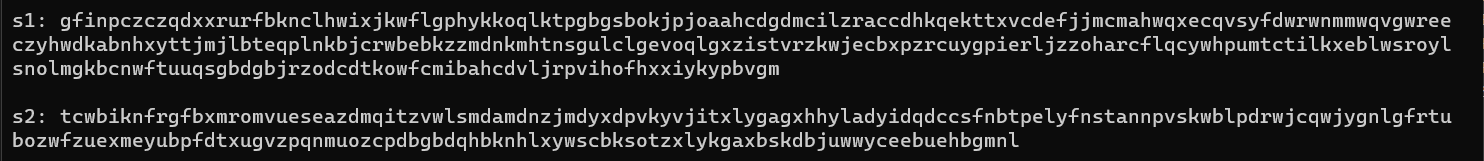
std::cout << "s1: " << s1 << std::endl << std::endl;

std::cout << "s2: " << s2 << std::endl;

}

Генерация строк на языке C++

Eсть 5 городов, соединённых между собой дорогами. Необходимо проложить между ними кратчайший замкнутый маршрут, проходящий через каждый город только один раз.



Результат выполнения программы

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

Возьмем в качестве произвольного маршрута:

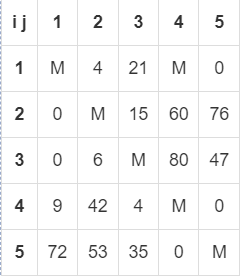
X0 = (1,2);(2,3);(3,4);(4,5);(5,1)

Тогда F(X0) = 8 + 19 + 86 + 12 + 89 = 214

Для определения нижней границы множества воспользуемся операцией редукции или приведения матрицы по строкам, для чего необходимо в каждой строке матрицы D найти минимальный элемент.



Затем вычитаем di из элементов рассматриваемой строки. В связи с этим во вновь полученной матрице в каждой строке будет как минимум один ноль.



Такую же операцию редукции проводим по столбцам, для чего в каждом столбце находим минимальный элемент:



После вычитания минимальных элементов получаем полностью редуцированную матрицу, где величины di и dj называются константами приведения.



Сумма констант приведения определяет нижнюю границу H:

H = ∑di + ∑dj

H = 4+4+6+12+17+0+4+4+0+0 = 51

Элементы матрицы dij соответствуют расстоянию от пункта i до пункта j.

Поскольку в матрице n городов, то D является матрицей nxn с неотрицательными элементами dij ≥ 0

Каждый допустимый маршрут представляет собой цикл, по которому коммивояжер посещает город только один раз и возвращается в исходный город.

Длина маршрута определяется выражением:

F(Mk) = ∑dij

Причем каждая строка и столбец входят в маршрут только один раз с элементом dij.

Определяем ребро ветвления и разобьем все множество маршрутов относительно этого ребра на два подмножества (i,j) и (i\*,j\*).

С этой целью для всех клеток матрицы с нулевыми элементами заменяем поочередно нули на М(бесконечность) и определяем для них сумму образовавшихся констант приведения, они приведены в скобках.



d(1,2) = 0 + 2 = 2; d(1,5) = 0 + 0 = 0; d(2,1) = 11 + 0 = 11; d(3,1) = 2 + 0 = 2; d(4,3) = 0 + 11 = 11; d(4,5) = 0 + 0 = 0; d(5,4) = 31 + 60 = 91;

Наибольшая сумма констант приведения равна (31 + 60) = 91 для ребра (5,4), следовательно, множество разбивается на два подмножества (5,4) и (5\*,4\*).

Исключение ребра (5,4) проводим путем замены элемента d54 = 0 на M, после чего осуществляем очередное приведение матрицы расстояний для образовавшегося подмножества (5\*,4\*), в результате получим редуцированную матрицу.



Нижняя граница гамильтоновых циклов этого подмножества:

H(5\*,4\*) = 51 + 91 = 142

Включение ребра (5,4) проводится путем исключения всех элементов 5-ой строки и 4-го столбца, в которой элемент d45 заменяем на М, для исключения образования негамильтонова цикла.

В результате получим другую сокращенную матрицу (4 x 4), которая подлежит операции приведения.

После операции приведения сокращенная матрица будет иметь вид:



Сумма констант приведения сокращенной матрицы:

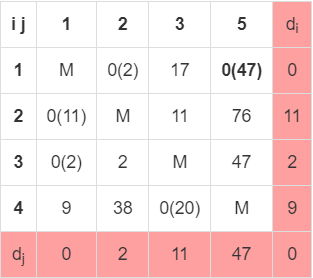
∑di + ∑dj = 0

Нижняя граница подмножества (5,4) равна:

H(5,4) = 51 + 0 = 51 ≤ 142

Поскольку нижняя граница этого подмножества (5,4) меньше, чем подмножества (5\*,4\*), то ребро (5,4) включаем в маршрут с новой границей H = 51

Определяем ребро ветвления.



d(1,2) = 0 + 2 = 2; d(1,5) = 0 + 47 = 47; d(2,1) = 11 + 0 = 11; d(3,1) = 2 + 0 = 2; d(4,3) = 9 + 11 = 20;

Наибольшая сумма констант приведения равна (0 + 47) = 47 для ребра (1,5).

Исключение ребра (1,5).



Нижняя граница гамильтоновых циклов этого подмножества:

H(1\*,5\*) = 51 + 47 = 98

Включение ребра (1,5) проводится путем исключения всех элементов 1-ой строки и 5-го столбца, в которой элемент d51 заменяем на М, для исключения образования негамильтонова цикла.

В результате получим другую сокращенную матрицу (3 x 3), которая подлежит операции приведения.

После операции приведения сокращенная матрица будет иметь вид:



Сумма констант приведения сокращенной матрицы:

∑di + ∑dj = 2

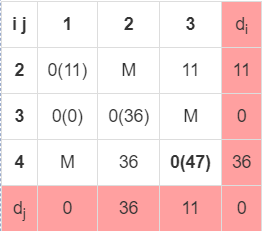
Нижняя граница подмножества (1,5) равна:

H(1,5) = 51 + 2 = 53 ≤ 98

Чтобы исключить подциклы, запретим следующие переходы: (4,1),

Поскольку нижняя граница этого подмножества (1,5) меньше, чем подмножества (1\*,5\*), то ребро (1,5) включаем в маршрут с новой границей H = 53

Определяем ребро ветвления.



d(2,1) = 11 + 0 = 11; d(3,1) = 0 + 0 = 0; d(3,2) = 0 + 36 = 36; d(4,3) = 36 + 11 = 47;

Наибольшая сумма констант приведения равна (36 + 11) = 47 для ребра (4,3).

Исключение ребра (4,3).



Нижняя граница гамильтоновых циклов этого подмножества:

H(4\*,3\*) = 53 + 47 = 100

Включение ребра (4,3) проводится путем исключения всех элементов 4-ой строки и 3-го столбца, в которой элемент d34 заменяем на М, для исключения образования негамильтонова цикла.

В результате получим другую сокращенную матрицу (2 x 2), которая подлежит операции приведения.

После операции приведения сокращенная матрица будет иметь вид:



Сумма констант приведения сокращенной матрицы:

∑di + ∑dj = 0

Нижняя граница подмножества (4,3) равна:

H(4,3) = 53 + 0 = 53 ≤ 100

Поскольку нижняя граница этого подмножества (4,3) меньше, чем подмножества (4\*,3\*), то ребро (4,3) включаем в маршрут с новой границей H = 53

В соответствии с этой матрицей включаем в гамильтонов маршрут ребра (2,1) и (3,2).

В результате по дереву ветвлений гамильтонов цикл образуют ребра:

(5,4), (4,3), (3,2), (2,1), (1,5),

Длина маршрута равна F(Mk) = 53

***Задание 2\****. Предложить решение задачи коммивояжера методом вервей и границ на языке C++.

// Программа на C++, решающая задачу коммивояжера

// с помощью метода ветвей и границ.

#include <Bits.h>

#include <iostream>

using namespace std;

const int N = 5;

// final\_path[] хранит финальное решение - путь коммивояжера.

int final\_path[N + 1];

// visited[] отслеживает уже посещенные узлы

// в конкретном пути

bool visited[N];

// Хранит финальный минимальный вес кратчайшего маршрута.

int final\_res = INT\_MAX;

// Функция копирует временное решение в

// финальное решение

void copyToFinal(int curr\_path[])

{

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

final\_path[i] = curr\_path[i];

final\_path[N] = curr\_path[0];

}

// Функция находит минимальную стоимость ребра,

// заканчивающегося на вершине i

int firstMin(int adj[N][N], int i)

{

int min = INT\_MAX;

for (int k = 0; k < N; k++)

if (adj[i][k] < min && i != k)

min = adj[i][k];

return min;

}

// Функция находит второе минимальное ребро

// заканчивающееся на вершине i

int secondMin(int adj[N][N], int i)

{

int first = INT\_MAX, second = INT\_MAX;

for (int j = 0; j < N; j++)

{

if (i == j)

continue;

if (adj[i][j] <= first)

{

second = first;

first = adj[i][j];

}

else if (adj[i][j] <= second &&

adj[i][j] != first)

second = adj[i][j];

}

return second;

}

// Функция, которая принимает в качестве аргументов:

// curr\_bound -> нижняя граница корневого узла

// curr\_weight -> хранит вес пути до сих пор

// level -> текущий уровень при перемещении в поиске

// пространства дерева

// curr\_path[] -> где хранится решение, которое

// впоследствии будет скопировано в final\_path[]

void TSPRec(int adj[N][N], int curr\_bound, int curr\_weight,

int level, int curr\_path[])

{

// Базовый случай - достижение уровня N, что

// означает, что мы прошли по всем узлам один раз

if (level == N)

{

// проверяем, есть ли ребро от последней вершины

// в пути обратно к первой вершине

if (adj[curr\_path[level - 1]][curr\_path[0]] != 0)

{

// curr\_res содержит общий вес решения

int curr\_res = curr\_weight +

adj[curr\_path[level - 1]][curr\_path[0]];

// Обновляем конечный результат и конечный путь,

// если текущий результат лучше.

if (curr\_res < final\_res)

{

copyToFinal(curr\_path);

final\_res = curr\_res;

}

}

return;

}

// для любого другого уровня перебираем все вершины

// рекурсивно создавая дерево пространства поиска

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

{

// Рассматриваем следующую вершину, если она не совпадает (диагональный

// элемент в матрице смежности и еще не посещена)

if (adj[curr\_path[level - 1]][i] != 0 &&

visited[i] == false)

{

int temp = curr\_bound;

curr\_weight += adj[curr\_path[level - 1]][i];

// Разное вычисление curr\_bound для

// уровня 2 от других уровней

if (level == 1)

curr\_bound -= ((firstMin(adj, curr\_path[level - 1]) +

firstMin(adj, i)) / 2);

else

curr\_bound -= ((secondMin(adj, curr\_path[level - 1]) +

firstMin(adj, i)) / 2);

// curr\_bound + curr\_weight - это реальная нижняя граница

// для узла, на котором мы находимся

// Если текущая нижняя граница < final\_res, то мы должны дальше исследовать

// этот узел

if (curr\_bound + curr\_weight < final\_res)

{

curr\_path[level] = i;

visited[i] = true;

// вызываем TSPRec для следующего уровня

TSPRec(adj, curr\_bound, curr\_weight, level + 1,

curr\_path);

}

// Иначе нам нужно обрезать узел, сбросив

// все изменения в curr\_weight и curr\_bound

curr\_weight -= adj[curr\_path[level - 1]][i];

curr\_bound = temp;

// Также сбрасываем массив visited

memset(visited, false, sizeof(visited));

for (int j = 0; j <= level - 1; j++)

visited[curr\_path[j]] = true;

}

}

}

// Эта функция устанавливает конечный путь final\_path[]

void TSP(int adj[N][N])

{

int curr\_path[N + 1];

// Вычисляем начальную нижнюю границу для корневого узла

// с помощью формулы 1/2 \* (сумма первой минимальной +

// второй минимальной) для всех ребер.

// Также инициализируем массив curr\_path и visited

int curr\_bound = 0;

memset(curr\_path, -1, sizeof(curr\_path));

memset(visited, 0, sizeof(curr\_path));

// Вычисляем начальную границу

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

curr\_bound += (firstMin(adj, i) +

secondMin(adj, i));

// Округляем нижнюю границу до целого числа

curr\_bound = (curr\_bound & 1) ? curr\_bound / 2 + 1 :

curr\_bound / 2;

// Мы начинаем с вершины 1, поэтому первая вершина

// в curr\_path[] равна 0

visited[0] = true;

curr\_path[0] = 0;

// Вызываем TSPRec для curr\_weight равного 0 и уровня 1

TSPRec(adj, curr\_bound, 0, 1, curr\_path);

}

// Код вызова функций

int main()

{

// Матрица смежности для данного графа

int adj[N][N] = {

{INFINITY, 8, 25, INFINITY, 4},

{4, INFINITY, 19, 64, 80},

{6, 12, INFINITY, 86, 53},

{21, 54, 16, INFINITY, 12},

{89, 70, 52, 17, INFINITY}

};

TSP(adj);

printf("Minimum cost : %d\n", final\_res);

printf("Path Taken : ");

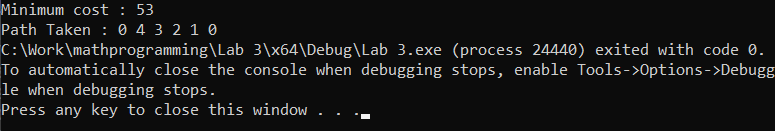
for (int i = 0; i <= N; i++)

printf("%d ", final\_path[i]);

return 0;

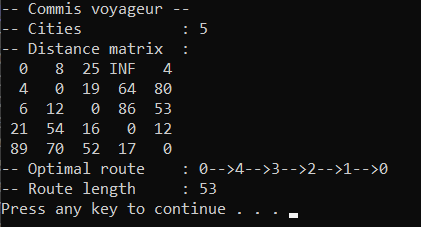
}

Решение задачи на языке C++



Решение задачи методом ветвей и границ

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



Решение задачи с использованием генератора перестановок

Результаты решения задач совпадают, что говорит о верности решения.

**ВЫВОД: Освоены общие принципы решения задач методом ветвей и границ, решена задача о коммивояжере данным методом, проведено сравнение полученного решения задачи с комбинаторным методом перестановок.**