**Отчёт по лабораторной работе №2**

**по дисциплине**

**Методы вычислений**

Выполнил: студент ИУ7-17

Пахомов А.А.

Вариант 1

МГТУ, 2015 г.

**Теоретическая часть**

**Содержательная постановка задачи коммивояжера:**

Коммивояжер должен объехать населенных пунктов, закончив маршрут в том же пункте, в котором начал. Стоимость непосредственного переезда из -го пункта в -й составляет единиц. Допускается случай . Это означает, что напрямую проехать из -го пункта в -й нельзя. В каждом городе коммивояжер должен побывать ровно один раз. Требуется составить такой маршрут, стоимость которого будет минимальной.

**Математическая постановка задачи коммивояжера:**

**Входные данные:**

Входные данные задаются в виде матрицы стоимостей C, для которой требуется решить задачу коммивояжера с использованием метода ветвей и границ.

**Метод ветвей и границ для решения задачи коммивояжера**

**C:\Users\Peter\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Gotovo.png**

**Текст программы**

*lab2.m*

m=dlmread('input2.txt');

fprintf('Матрица стоимостей = \n');

disp(m);

% с отладочной печатью

[assignment,cost] = knight\_of\_the\_road(m, true);

fprintf('\nx\* = \n');

disp(assignment);

fprintf('\nf\* = %d\n\n', cost)

*knight\_of\_the\_road.m*

function [assignment,cost] = knight\_of\_the\_road(matr, debugging)

xx = [0 1 0 0 0; 0 0 1 0 0; 0 0 0 1 0; 0 0 0 0 1; 1 0 0 0 0];

ff = sum(sum(mult\_matrixes\_mod\_inf(matr, xx)));

if debugging

fprintf('\nНачальное x\* = \n');

disp(xx);

fprintf('\nНачальное f\* = %d\n\n', ff)

end;

S = (matr);

[~,n] = size(matr);

k = 0;

while ~isempty(S)

k = k + 1;

m = S(:,:,1);

S(:,:,1) = [];

if debugging

fprintf('Метод ветвей и границ для З.К. Итерация %d\n', k);

end;

[assignment,cost] = hungarian\_method\_mod\_inf(m, false, debugging);

if debugging

fprintf('\nx(opt) = \n');

disp(assignment);

fprintf('\nf(opt) = %d\n\n', cost)

end;

if cost < ff

cycles = getCycles(assignment);

if debugging

fprintf('Циклы:\n');

for i=1:size(cycles,2)

disp(cycles{i}.');

end;

end;

if size(cycles, 2) == 1 && size(cycles{1},1) == n

if debugging

fprintf('Новое минимальное решение f\* = %d\n', cost);

end;

xx = assignment;

ff = cost;

else

res\_cycle = cycles{1};

min\_size = size(res\_cycle, 1);

for i=2:size(cycles,2)

if size(cycles{i},1) < min\_size

min\_size = size(cycles{i}, 1);

res\_cycle = cycles{i};

end;

end;

subtask\_cnt = size(S, 3);

sub = m;

sub(res\_cycle(min\_size), res\_cycle(1)) = Inf;

S(:,:,subtask\_cnt+1) = sub;

for i=1:min\_size-1

sub = m;

sub(res\_cycle(i), res\_cycle(i+1)) = Inf;

S(:,:,subtask\_cnt+1+i) = sub;

end;

end;

end;

end;

*getCycles.m*

function cycles = getCycles(x)

cycles = {};

localCycle = 1;

found = 1;

indexes = 1: size(x,1);

k = 1;

while size(found,1) ~= size(x,1)

localCycleSize = size(localCycle,1);

ind = localCycle(localCycleSize);

cur = find(x(ind,:) == 1);

if cur ~= localCycle(1)

found = [found; cur];

localCycle = [localCycle; cur];

end;

if cur == localCycle(1) || size(found,1) == size(x,1)

cycles{k} = localCycle;

k = k + 1;

tmp = setdiff(indexes, found);

if ~isempty(tmp)

localCycle = tmp(1);

found = [found; tmp(1)];

end;

end;

end;

*mult\_matrixes\_mod\_inf.m*

function res = mult\_matrixes\_mod\_inf(m1, m2)

[m,n] = size(m1);

res = zeros(n, m);

for i=1:n

for j=1:m

if(m1(i,j) == 0 || m2(i,j) == 0)

res(i, j) = 0;

else

res(i, j) = m1(i,j)\*m2(i,j);

end;

end;

end;

Венгерский метод, реализованный в лабораторной работе №1 модифицирован для умножения матриц, содержащих ∞.

**Результаты вычислений**

**Задача минимизации:**

Матрица стоимостей =

Inf 1 15 10 9

9 Inf 2 2 3

11 1 Inf 9 4

2 9 10 Inf 12

9 1 9 10 Inf

x\* =

0 1 0 0 0

0 0 1 0 0

0 0 0 0 1

1 0 0 0 0

0 0 0 1 0

f\* = 19