МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики»

Факультет информационных технологий и программирования

Кафедра информационных систем

Лабораторная работа №2

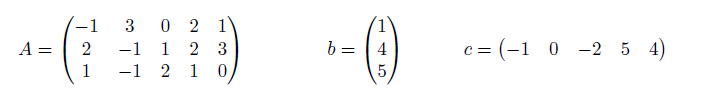
**Симплекс метод, транспортная задача**

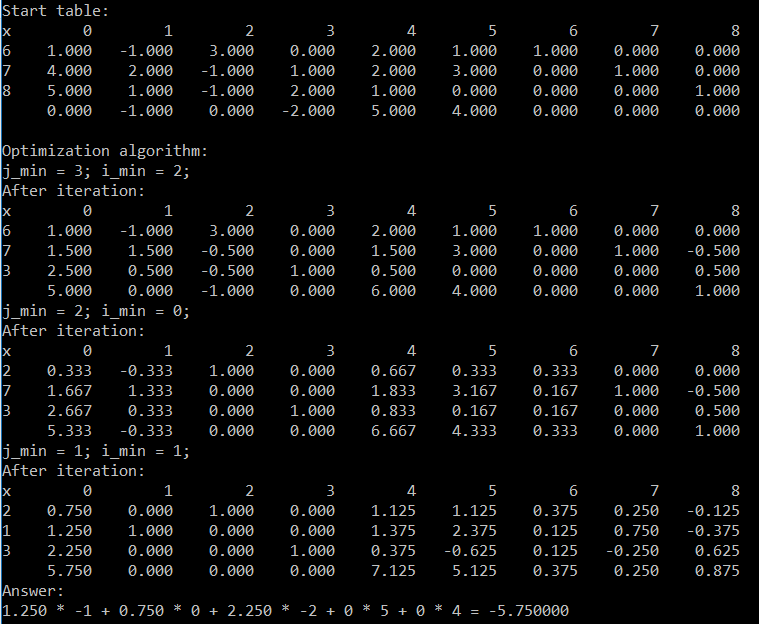
Выполнил студент группы № М3307:   
Бойцов Виталий Вячеславич

Санкт-Петербург

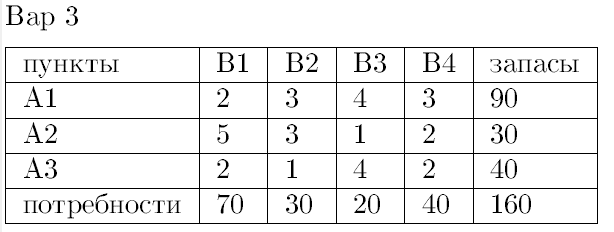
2018

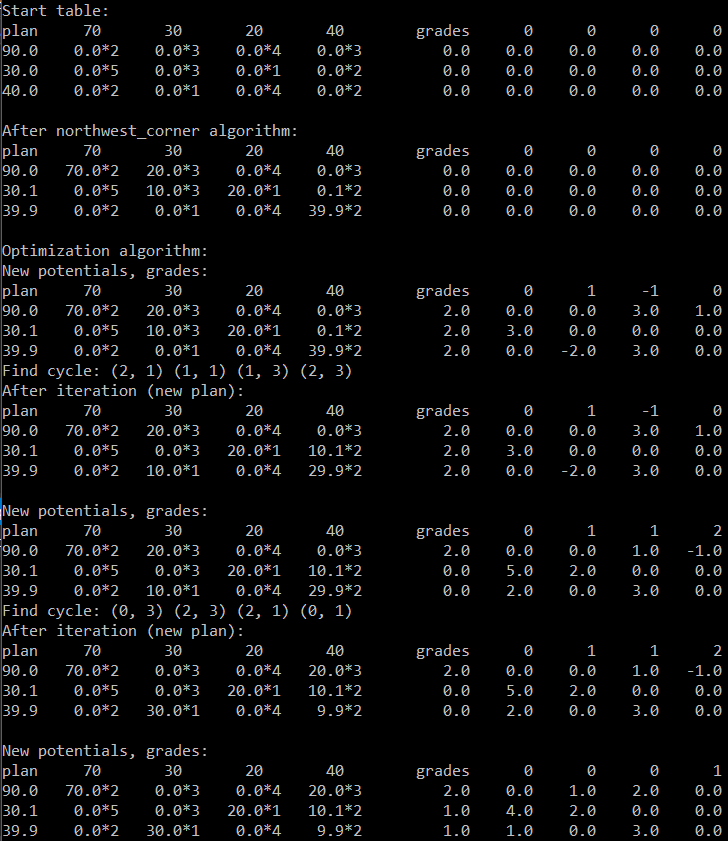
Симплекс метод





Транспортная задача





Main.cpp

#include <iostream>

#include "TPSolver.h"

#include "SMSolver.h"

using namespace *std*;

void solve\_SM() {

SMSolver::vec2d a = {

{ -1, 3, 0, 2, 1 },

{ 2, -1, 1, 2, 3 },

{ 1, -1, 2, 1, 0 }

};

SMSolver::vec1d b = { 1, 4, 5 };

SMSolver::vec1d c = { -1, 0, -2, 5, 4 };

SMSolver solver(a, b, c);

*printf*("Start table:\n");

solver.print();

*printf*("\n");

*printf*("Optimization algorithm:\n");

solver.optimize();

*cout* << "Answer:\n";

solver.print\_ans();

}

void solve\_TP() {

TPSolver::vec2d cost = {

{ 2, 3, 4, 3 },

{ 5, 3, 1, 2 },

{ 2, 1, 4, 2 }

};

TPSolver::vec1d supply = { 90, 30, 40 };

TPSolver::vec1d claim = { 70, 30, 20, 40 };

TPSolver solver(supply, claim, cost);

*cout* << "Start table:\n";

solver.print();

*cout* << '\n';

solver.northwest\_corner();

*cout* << "After northwest\_corner algorithm:\n";

solver.print();

*cout* << '\n';

*cout* << "Optimization algorithm:\n";

solver.optimize();

}

int *main*() {

solve\_SM();

solve\_TP();

# }

SMSolver.h

#pragma once

#include <iostream>

#include <vector>

#include <queue>

using namespace *std*;

class SMSolver {

public:

typedef *vector*<double> vec1d;

typedef *vector*<int> vec1i;

typedef *vector*<*vector*<double>> vec2d;

vec2d a;

vec1i b;

vec1d c;

SMSolver(vec2d a, vec1d b, vec1d c) {

this->c = c;

for (int i = 0; i < b.*size*(); i++) {

a[i].*insert*(a[i].*begin*(), b[i]);

for (int j = 0; j < b.*size*(); j++) {

a[i].*push\_back*(i == j);

}

}

c.*insert*(c.*begin*(), 0);

for (int i = 0; i < b.*size*(); i++) {

this->b.*push\_back*(c.*size*());

c.*push\_back*(0);

}

a.*push\_back*(c);

this->a = a;

}

void print() {

*size\_t* w = 8;

//name

*printf*("x");

//space

*printf*(" ");

for (int j = 0; j < a[0].*size*(); j++) {

//x1

*printf*("%\*d", w, j);

//space

*printf*(" ");

}

*printf*("\n");

for (int i = 0; i < b.*size*(); i++) {

//name

*printf*("%d", b[i]);

//space

*printf*(" ");

for (int j = 0; j < a[i].*size*(); j++) {

*printf*("%\*.3lf ", w, a[i][j]);

}

*printf*("\n");

}

//name

*printf*(" ");

//space

*printf*(" ");

for (int j = 0; j < a[0].*size*(); j++) {

*printf*("%\*.3lf ", w, a.*back*()[j]);

}

*printf*("\n");

}

void print\_ans() {

double ans = 0;

for (int j = 0; j < c.*size*(); j++) {

bool pr = 0;

for (int i = 0; i < b.*size*(); i++) {

if (b[i] == j + 1) {

ans += a[i][0] \* c[j];

*printf*("%.3lf \* %.0lf", a[i][0], c[j]);

goto label;

}

}

*printf*("0 \* %.0lf", c[j]);

label:

if (j == c.*size*() - 1) {

*printf*(" = %lf\n", ans);

} else {

*printf*(" + ");

}

}

}

void add\_string\_to\_string(int i1, double multy, int i2) {

for (*size\_t* j = 0; j < a[0].*size*(); j++) {

a[i2][j] += a[i1][j] \* multy;

}

}

void multy\_string(int i, double multy) {

for (*size\_t* j = 0; j < a[0].*size*(); j++) {

a[i][j] \*= multy;

}

}

void optimize() {

while (true) {

int j\_min = *min\_element*(a.*back*().*begin*(), a.*back*().*end*()) - a.*back*().*begin*();

if (a.*back*()[j\_min] >= 0) {

break;

}

int i\_min = -1;

for (int i = 0; i < b.*size*(); i++) {

if (a[i][j\_min] > 0 && (i\_min == -1 || (a[i\_min][0] / a[i\_min][j\_min] > a[i][0] / a[i][j\_min]))) {

i\_min = i;

}

}

*cout* << "j\_min = " << j\_min << "; i\_min = " << i\_min << ";\n";

b[i\_min] = j\_min;

multy\_string(i\_min, 1 / a[i\_min][j\_min]);

for (int i = 0; i < a.*size*(); i++) {

if (i != i\_min && a[i][j\_min] != 0) {

add\_string\_to\_string(i\_min, -a[i][j\_min], i);

}

}

*printf*("After iteration:\n");

print();

}

}

};

TPSolver

#pragma once

#include <iostream>

#include <vector>

#include <queue>

using namespace *std*;

struct Coordinate {

int i, j;

Coordinate(int i, int j) : i(i), j(j) {}

bool operator == (Coordinate b) {

return i == b.i && j == b.j;

}

bool operator != (Coordinate b) {

return !((\*this) == b);

}

};

*ostream*& operator << (*ostream* &os, Coordinate const& c) {

return os << "(" << c.i << ", " << c.j << ")";

}

class TPSolver {

public:

const double EPS = 0.1;

typedef *vector*<double> vec1d;

typedef *vector*<*vector*<double>> vec2d;

typedef *vector*<int> vec1i;

typedef *vector*<*vector*<int>> vec2i;

typedef *vector*<Coordinate> vec1c;

typedef *vector*<*vector*<Coordinate>> vec2c;

static vec2d make\_vec2d(int n, int m) {

return vec2d(n, vec1d(m, 0));

}

static vec2i make\_vec2i(int n, int m) {

return vec2i(n, vec1i(m, 0));

}

static vec2c make\_vec2c(int n, int m) {

return vec2c(n, vec1c(m, Coordinate(-1, -1)));

}

vec1d resource;

vec1d request;

vec2d cost;

vec2d plan;

vec1d u\_res;

vec1d u\_req;

vec1i u\_res\_used;

vec1i u\_req\_used;

vec2d grade;

vec2i used;

vec2c previous;

TPSolver(vec1d resource, vec1d request, vec2d cost) : resource(resource), request(request), cost(cost) {

plan = make\_vec2d(resource.*size*(), request.*size*());

grade = make\_vec2d(resource.*size*(), request.*size*());

u\_res = vec1d(resource.*size*());

u\_req = vec1d(request.*size*());

}

double resource\_sum(int i) {

double ans = 0;

for (int j = 0; j < request.*size*(); j++) {

ans += plan[i][j];

}

return ans;

}

double request\_sum(int j) {

double ans = 0;

for (int i = 0; i < resource.*size*(); i++) {

ans += plan[i][j];

}

return ans;

}

void print() {

*size\_t* w = 6;

*printf*("plan ");

for (int j = 0; j < request.*size*(); j++)

*printf*("%\*.0lf ", w, request[j]);

*printf*(" grades ");

for (int j = 0; j < request.*size*(); j++)

*printf*("%\*.0lf ", w, u\_req[j]);

*printf*("\n");

for (int i = 0; i < resource.*size*(); i++) {

*printf*("%\*.1lf ", 4, resource[i]);

for (int j = 0; j < request.*size*(); j++) {

*printf*("%\*.1lf\*%.0lf ", w, plan[i][j], cost[i][j]);

}

*printf*("%s", *string*(5, ' ').*c\_str*());

*printf*("%\*.1lf ", w, u\_res[i]);

for (int j = 0; j < request.*size*(); j++) {

*printf*("%\*.1lf ", w, grade[i][j]);

}

*printf*("\n");

}

}

void northwest\_corner() {

int i = 0, j = 0;

while (true) {

if (i == resource.*size*() - 1 && j == request.*size*() - 1) {

plan[i][j] = resource[i] - resource\_sum(i);

break;

}

if (resource[i] - resource\_sum(i) > request[j] - request\_sum(j)) {

plan[i][j] = request[j] - request\_sum(j);

j++;

}

else if (resource[i] - resource\_sum(i) < request[j] - request\_sum(j)) {

plan[i][j] = resource[i] - resource\_sum(i);

i++;

}

else if (resource[i] - resource\_sum(i) == request[j] - request\_sum(j)) {

resource[i] += EPS;

resource.*back*() -= EPS;

}

}

}

void dfs\_potentials(Coordinate c) {

for (int i = 0; i < resource.*size*(); i++) {

int j = c.j;

if (!used[i][j] && plan[i][j]) {

used[i][j] = 1;

if (!u\_req\_used[j]) {

u\_req[j] = cost[i][j] - u\_res[i];

u\_req\_used[j] = 1;

}

if (!u\_res\_used[i]) {

u\_res[i] = cost[i][j] - u\_req[j];

u\_res\_used[i] = 1;

}

dfs\_potentials(Coordinate(i, j));

}

}

for (int j = 0; j < request.*size*(); j++) {

int i = c.i;

if (!used[i][j] && plan[i][j]) {

used[i][j] = 1;

if (!u\_req\_used[j]) {

u\_req[j] = cost[i][j] - u\_res[i];

u\_req\_used[j] = 1;

}

if (!u\_res\_used[i]) {

u\_res[i] = cost[i][j] - u\_req[j];

u\_res\_used[i] = 1;

}

dfs\_potentials(Coordinate(i, j));

}

}

}

void find\_potentials() {

used = make\_vec2i(resource.*size*(), request.*size*());

u\_res = vec1d(resource.*size*());

u\_req = vec1d(request.*size*());

u\_res\_used = vec1i(resource.*size*());

u\_req\_used = vec1i(request.*size*());

for (int i = 0; i < resource.*size*(); i++) {

for (int j = 0; j < request.*size*(); j++) {

if (!used[i][j] && plan[i][j]) {

u\_req[j] = 0;

u\_req\_used[j] = 1;

u\_res[i] = cost[i][j] - u\_req[j];

u\_res\_used[i] = 1;

used[i][j] = 1;

dfs\_potentials(Coordinate(i, j));

}

}

}

}

bool dfs\_cycle(Coordinate c) {

if (used[c.i][c.j] % 2) {

for (int j = 0; j < request.*size*(); j++) {

if (plan[c.i][j] && j != c.j) {

if (used[c.i][j] == 1) {

previous[c.i][j] = c;

return true;

}

if (!used[c.i][j]) {

used[c.i][j] = used[c.i][c.j] + 1;

previous[c.i][j] = c;

if (dfs\_cycle(Coordinate(c.i, j)))

return true;

previous[c.i][j] = Coordinate(-1, -1);

used[c.i][j] = 0;

}

}

}

}

else {

for (int i = 0; i < resource.*size*(); i++) {

if (i != c.i && (used[i][c.j] == 1 || plan[i][c.j])) {

if (used[i][c.j] == 1) {

previous[i][c.j] = c;

return true;

}

if (!used[i][c.j]) {

used[i][c.j] = used[i][c.j] + 1;

previous[i][c.j] = c;

if (dfs\_cycle(Coordinate(i, c.j)))

return true;

previous[i][c.j] = Coordinate(-1, -1);

used[i][c.j] = 0;

}

}

}

}

return false;

}

void optimize() {

while (true) {

find\_potentials();

for (int i = 0; i < resource.*size*(); i++) {

for (int j = 0; j < request.*size*(); j++) {

grade[i][j] = cost[i][j] - u\_res[i] - u\_req[j];

}

}

*cout* << "New potentials, grades:\n";

print();

Coordinate c\_min = { 0, 0 };

for (int i = 0; i < resource.*size*(); i++) {

for (int j = 0; j < request.*size*(); j++) {

if (grade[i][j] < grade[c\_min.i][c\_min.j]) {

c\_min.i = i;

c\_min.j = j;

}

}

}

if (grade[c\_min.i][c\_min.j] >= 0) {

break;

}

used = make\_vec2i(resource.*size*(), request.*size*());

previous = make\_vec2c(resource.*size*(), request.*size*());

used[c\_min.i][c\_min.j] = 1;

dfs\_cycle(c\_min);

Coordinate now = c\_min;

vec1c cycle;

cycle.*push\_back*(now);

now = previous[now.i][now.j];

while (now != c\_min) {

cycle.*push\_back*(now);

now = previous[now.i][now.j];

}

*cout* << "Find cycle: ";

for (auto val : cycle)

*cout* << val << ' ';

*cout* << "\n";

Coordinate cycle\_c\_min = cycle[1];

for (int k = 3; k < cycle.*size*(); k++) {

if (plan[cycle\_c\_min.i][cycle\_c\_min.j] > plan[cycle[k].i][cycle[k].j]) {

cycle\_c\_min = cycle[k];

}

}

double delta = plan[cycle\_c\_min.i][cycle\_c\_min.j];

for (int k = 0; k < cycle.*size*(); k++) {

if (k % 2) {

plan[cycle[k].i][cycle[k].j] -= delta;

} else {

plan[cycle[k].i][cycle[k].j] += delta;

}

}

*cout* << "After iteration (new plan):\n";

print();

*cout* << '\n';

}

}

};