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 № M3307:
Бойцов Виталий Вячеславич

Санкт-Петербург

2018

Симплекс метод

$$A = \begin{pmatrix} -1 & 3 & 0 & 2 & 1 \\ 2 & -1 & 1 & 2 & 3 \\ 1 & -1 & 2 & 1 & 0 \end{pmatrix} \qquad b = \begin{pmatrix} 1 \\ 4 \\ 5 \end{pmatrix} \qquad c = \begin{pmatrix} -1 & 0 & -2 & 5 & 4 \end{pmatrix}$$

```
Start table:
x        0        1        2        3        4        5        6        7        8
6    1.000   -1.000    3.000    0.000    2.000    1.000    1.000    0.000    0.000
7    4.000    2.000   -1.000    1.000    2.000    3.000    0.000    1.000    0.000
8    5.000    1.000   -1.000    2.000    1.000    0.000    0.000    0.000    1.000
     0.000   -1.000    0.000   -2.000    5.000    4.000    0.000    0.000    0.000

Optimization algorithm:
j_min = 3; i_min = 2;
After iteration:
x        0        1        2        3        4        5        6        7        8
6    1.000   -1.000    3.000    0.000    2.000    1.000    1.000    0.000    0.000
7    1.500    1.500   -0.500    0.000    1.500    3.000    0.000    1.000   -0.500
3    2.500    0.500   -0.500    1.000    0.500    0.000    0.000    0.000    0.500
     5.000    0.000   -1.000    0.000    6.000    4.000    0.000    0.000    1.000
j_min = 2; i_min = 0;
After iteration:
x        0        1        2        3        4        5        6        7        8
2    0.333   -0.333    1.000    0.000    0.667    0.333    0.333    0.000    0.000
7    1.667    1.333    0.000    0.000    1.833    3.167    0.167    1.000   -0.500
3    2.667    0.333    0.000    1.000    0.833    0.167    0.167    0.000    0.500
     5.333   -0.333    0.000    0.000    6.667    4.333    0.333    0.000    1.000
j_min = 1; i_min = 1;
After iteration:
x        0        1        2        3        4        5        6        7        8
2    0.750    0.000    1.000    0.000    1.125    1.125    0.375    0.250   -0.125
1    1.250    1.000    0.000    0.000    1.375    2.375    0.125    0.750   -0.375
3    2.250    0.000    0.000    1.000    0.375   -0.625    0.125   -0.250    0.625
     5.750    0.000    0.000    0.000    7.125    5.125    0.375    0.250    0.875
Answer:
1.250 * -1 + 0.750 * 0 + 2.250 * -2 + 0 * 5 + 0 * 4 = -5.750000
```

Транспортная задача

Вар 3

пункты	B1	B2	B3	B4	запасы
A1	2	3	4	3	90
A2	5	3	1	2	30
A3	2	1	4	2	40
потребности	70	30	20	40	160

```
Start table:
plan      70        30        20        40        grades      0      0      0      0
90.0     0.0*2     0.0*3     0.0*4     0.0*3          0.0    0.0    0.0    0.0    0.0
30.0     0.0*5     0.0*3     0.0*1     0.0*2          0.0    0.0    0.0    0.0    0.0
40.0     0.0*2     0.0*1     0.0*4     0.0*2          0.0    0.0    0.0    0.0    0.0

After northwest_corner algorithm:
plan      70        30        20        40        grades      0      0      0      0
90.0    70.0*2    20.0*3     0.0*4     0.0*3          0.0    0.0    0.0    0.0    0.0
30.1     0.0*5    10.0*3    20.0*1     0.1*2          0.0    0.0    0.0    0.0    0.0
39.9     0.0*2     0.0*1     0.0*4    39.9*2          0.0    0.0    0.0    0.0    0.0

Optimization algorithm:
New potentials, grades:
plan      70        30        20        40        grades      0      1     -1      0
90.0    70.0*2    20.0*3     0.0*4     0.0*3          2.0    0.0    0.0    3.0    1.0
30.1     0.0*5    10.0*3    20.0*1     0.1*2          2.0    3.0    0.0    0.0    0.0
39.9     0.0*2     0.0*1     0.0*4    39.9*2          2.0    0.0   -2.0    3.0    0.0
Find cycle: (2, 1) (1, 1) (1, 3) (2, 3)
After iteration (new plan):
plan      70        30        20        40        grades      0      1     -1      0
90.0    70.0*2    20.0*3     0.0*4     0.0*3          2.0    0.0    0.0    3.0    1.0
30.1     0.0*5     0.0*3    20.0*1    10.1*2          2.0    3.0    0.0    0.0    0.0
39.9     0.0*2    10.0*1     0.0*4    29.9*2          2.0    0.0   -2.0    3.0    0.0

New potentials, grades:
plan      70        30        20        40        grades      0      1      1      2
90.0    70.0*2    20.0*3     0.0*4     0.0*3          2.0    0.0    0.0    1.0   -1.0
30.1     0.0*5     0.0*3    20.0*1    10.1*2          0.0    5.0    2.0    0.0    0.0
39.9     0.0*2    10.0*1     0.0*4    29.9*2          0.0    2.0    0.0    3.0    0.0
Find cycle: (0, 3) (2, 3) (2, 1) (0, 1)
After iteration (new plan):
plan      70        30        20        40        grades      0      1      1      2
90.0    70.0*2     0.0*3     0.0*4    20.0*3          2.0    0.0    0.0    1.0   -1.0
30.1     0.0*5     0.0*3    20.0*1    10.1*2          0.0    5.0    2.0    0.0    0.0
39.9     0.0*2    30.0*1     0.0*4     9.9*2          0.0    2.0    0.0    3.0    0.0

New potentials, grades:
plan      70        30        20        40        grades      0      0      0      1
90.0    70.0*2     0.0*3     0.0*4    20.0*3          2.0    0.0    1.0    2.0    0.0
30.1     0.0*5     0.0*3    20.0*1    10.1*2          1.0    4.0    2.0    0.0    0.0
39.9     0.0*2    30.0*1     0.0*4     9.9*2          1.0    1.0    0.0    3.0    0.0
```

Main.cpp

```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';
		}
	}

};
```