### Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение Высшего образования

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

**Всероссийская олимпиада школьников**

**«Шаг в будущее»**

АВТОМАТИЗИРОВАННАЯ СИСТЕМА УПРАВЛЕНИЯ ЛОГИСТИЧЕСКИМИ ПОТОКАМИ НА ЖД-ТРАНСПОРТЕ

### **Автор**: Бортников Евгений,

ГБОУ школа №1329, 10 «И» класс

**Научный Руководитель**: Митрузаев Александр Евгеньевич

ГБОУ школа №1329, педагог

### Москва 2017

**Содержание**

### Постановка целей и анализ задачи 3

### Теоретическая часть 4 - 7

### Рассчетная часть по решению поставленных задач 8

### 4)Заключение 9

### Список использованной литературы 10

### Приложение 1: графики быстродействия и точности методов 11

### Приложение 2: исходный код на языке с++ 14 12 - 32

### Приложение 3: пример работы программы 33

**Постановка целей и анализ задачи**

В сталеплавильном цехе жидкая сталь проходит несколько стадий обработки – сначала из жидкого чугуна происходит выплавка стали, затем, для легирования и придания стали нужных свойств, сталь обрабатывают на агрегатах внепечной обработки, после чего выполняется разливка. Внутри цеха сталь перевозится в больших ковшах: на сталевозах по ширококолейным железнодорожным путям (всего **M** путей) и в перпендикулярном направлении мостовыми кранами (всего **N** пролетов, в каждом из которых установлен 1 мостовой кран). Для минимизации затрат по времени требуется определить оптимальный план поставок плавок к агрегатам внепечной обработки. Поэтому в данном проекте рассмотрено создание автоматизированной системы управления логистическими потоками на жд-транспорте на примере решения транспортной задачи. Поскольку решение данной практической задачи можно свести к решению закрытой транспортной задачи, путем представления каждого ковша как единицы товара с фиксированной стоимостью (в данном случае временем) перевозки из пункта отправления в пункт назначения. Добиваясь минимизации затрат по времени на перевозки, получаем закрытую (количество отправляемого товара равно количеству получаемого) однородную (все товары одинаковы) транспортную задачу.

Цель проекта:

\* Создать программное решение автоматизации управления логистическими потоками на жд-транспорте.

Задачи проекта:

* 1. Рассмотреть всевозможные методы решения транспортной задачи.
  2. Произвести оценку каждого метода.
  3. Предложить собственный метод решения.
  4. Познакомиться с новыми приемами и теоретическими основами линейного и динамического программирования.
  5. Подготовить практическую реализацию решения на языке программирования C++ 14.

**Теоретическая часть**

*m n*

### Формально транспортная задача определяется так:

*F*   

*c ij x ij* 

min

### , где c -

*i*  1 *j*  1

### тариф\время перевозки от пункта отправления i в пункт назначения j, х - объем поставки, a\b - количество товара в пункте отправления\назначения соответственно. При условиях:



*m*



*i*1

*xij*

*bj* *j*  1, *n*,

*n*

# 

*j* 1

*xij*

 *ai* *i*  1, *m*,

*x*  0*i*  1, *m*; *j*  1, *n*.

### Т.е. сумма поставок по всем строкам и столбцам должна быть равна сумме всего спроса и предложения, а также поставки должны быть положительными.

Решение начинают с нахождения опорного плана (начального заполнения матрицы). Данную операцию можно осуществить методом северо-западного угла, методом минимального элемента или методом аппроксимации Фогеля. Сущность этих методов заключается в том,

что за

*n*  *m* 1

### шагов, на каждом из которых заполняют одну клетку, полностью

удовлетворяя потребности пункта отправления либо пункта назначения. Затем исключают из рассмотрения соответствующую строку либо столбец, в котором находится заполняемая

клетка. После

*n*  *m*  2

### шагов остается лишь 1 пункт отправления и 1 пункт назначения,

потребности которых равны, оставшаяся клетка заполняется. Рассмотрим подробнее методы начального заполнения.

***Метод северо-западного угла***

При заполнении матрицы данным методом на каждой итерации заполняют клетку, рассматривая первый из оставшихся пункт назначения и первый из оставшихся пункт назначения, не обращая внимания на тарифы. Т.е. заполнение матрицы начинают с северо-западного (левого верхнего) угла.

***Метод минимального элемента***

В данном методе на каждом шаге заполняется клетка с минимальным из всех оставшихся тарифом. Затем столбец либо строка, содержащая данную клетку, и чьи потребности полностью удовлетворены, более не рассматривается.

***Метод аппроксимации Фогеля***

При использовании данного метода рассчитывается разность между двумя минимальными тарифами для всех оставшихся строк и столбцов. Из полученных значений выбирают минимальное и в строке либо столбце, которому оно соответствует, находят ближайшую к столбцу либо строке, которая также содержит данное значение, клетку с минимальным тарифом и заполняют ее.

Для нахождения оптимального плана поставок разработано несколько методов. Однако наиболее часто используют метод потенциалов и метод дифференциальных рент. Я также приложил метод, использующий теорию графов в основе решения.

***Метод теории графов***

В данном методе по начальному заполнению матрицы смежности строят граф. Затем его делят на две доли, используя «цветовую покраску» обходом в ширину. После разделения по графу пускают максимальный поток с предпроталкиванием и поднятием вершин алгоритмом проталкивания предпотока, причем необходимо запоминать каждую пройденную вершину и соотносить ее с клеткой матрицы. После получения максимального потока минимизируют стоимость связанных с ним тарифов и заполняют матрицу, восстанавливая поток по пройденным клеткам тарифов.

***Метод потенциалов***

Для использования данного метода подробно рассмотрим каждый шаг и введем несколько терминов.

**Потенциалом клетки** называется

*d ij*  *bj*  *ai*  *cij*

### , где b - потенциал пункта

отправления, a - потенциал пункта назначения, c - тариф данной клетки. Также

*bj*  *ai*  *cij* .

### Получаем систему линейных уравнений, в которой число неизвестных на 1 превышает чисто уравнений. Для упрощения решения 1 произвольный потенциал пункта отправления либо

назначения можно назначить произвольным числом. Удобнее всего сказать, что

*a*1  0 .

### Тогда будет довольно нетрудно последовательно рассчитать все остальные потенциалы.

Если все потенциалы клеток неположительны, то план поставок является оптимальным, иначе следует найти максимальный из положительных потенциал клетки и заполнить ее, затем необходимо изменить значение в ряде других клеток, связанных с заполняемой так называемым циклом.

**Циклом** в данном методе потенциалов называется ломаная линия, звенья которой расположены вдоль строк и столбцов, а вершины в занятых клетках. Причем в каждой вершине цикла встречается ровно одно вертикальное и одно горизонтальное звено, точки самопересечения вершинами не являются.

После нахождения цикла, данной клетке приписывают знак «+», всем остальным поочередно знак «-» или «+». Затем в заполняемую клетку переносят меньшее из отрицательных чисел, записанное в клетках цикла, вычитают из всех отрицательных клеток и прибавляют ко всем положительным клеткам. В результате определяют новый опорный план задачи. Описанные выше действия называют сдвигом по циклу пересчета.

Найденный опорный план снова проверяют на оптимальность, при отрицательном результате переходят к новому опорному плану, производя сдвиг по циклу пересчета. После конечного числа итераций получают оптимальный план.

***Метод дифференциальных рент***

В данном методе производится начальное **условно-оптимальное** заполнение: в каждом столбце помечают клетку с минимальным тарифом, затем заполняют ее максимально возможным числом. В общих случаях данное заполнение не соответствует ограничениям задачи. Для нормализации распределения производят поиск избыточных и недостаточных строк.

**Избыточная строка** - строка, запасы поставщика в которой не распределены полностью, когда **недостаточная строка** - строка, запасы поставщика в которой полностью распределены, но потребности связанных с ней столбцов не удовлетворены полностью. При нулевых запасах строка считается положительной, если вторая заполненная клетка в столбце, связанным еще одной заполненной клеткой с данной строкой, расположена в положительной строке.

На следующем шаге для каждого столбца находят разницу между помеченным числом и ближайшим к нему тарифом в избыточной строке, но если помеченное число уже находится в избыточной строке, то разность не определяют. Среди полученных чисел выбирают наименьшее, оно называется **промежуточной рентой**. Затем происходит переход к новой таблице путем прибавления к тарифам в недостаточных строках промежуточной ренты, причем все клетки полученной таблицы считаются свободными.

После перехода к новой матрице начинают ее заполнение по следующему правилу: выбирают столбец либо строку, содержащую помеченное число, затем данную клетку заполняют, а строку либо столбец исключают из рассмотрения. После конечного числа итераций заполняются все клетки с помеченными числами, если к тому моменту уже удается распределить весь груз, то план считается оптимальным, иначе снова переходят к новой таблице. После повторения выше описанных действий получают оптимальный план поставок.

**Практическая часть по решению поставленных задач**

***Оценка эффективности методов начального заполнения***

1. Ассимптотика метода северо-западного угла:

*O**n* \* *m*. Время работы: 0.00с.

### Точность (количество верных ответов / количество ответов \* 100%) : 10%.

1. Ассимптотика метода минимального элемента: Точность: 20%.
2. Ассимптотика метода аппроксимации Фогеля: Точность: 30%.

*O**n*2\**m*2 . Время работы: 0.01с.

*O**n*3 \* *m*3. Время работы: 0.04с.

***Оценка эффективности методов конечного заполнения***

### Метод потенциалов имеет ассимптотику

*O* 5 \* 5

### . Время работы: 0, 17с.

*n m* 

1. Метод дифференциальных рент имеет ассимптотику

*O**n*4 \* *m*4 

### и более простую логику

перехода матрицы, что делает его наиболее эффективным при использовании ЭВМ. Время работы: 0.09с.

1. Метод теории графов имеет ассимптотику:

*O* 4 \* 3

### . Время работы: 0.08с.

*n m* 

**Заключение**

В заключение хотелось бы отметить, что цель проекта достигнута - успешно создано программное решение поставленной проблемы.

Также во время работы над данным проектом я познакомился с методами линейного и динамического программирования, многими аспектами теории графов и математическими теоремами.

Более того, я планирую разработать собственный метод решения транспортной задачи, основанный на подходе к полиномиальной задаче методом неполиномиального решения.

К тому же я узнал о подходах к задачам методом математического программирования, что увеличило мои результаты в регулярных личных зачетах по спортивному программированию.

**Список использованной литературы**

[*Книги*]

### 1.С. Дасгупта, Х. Пападимитриу, У. Вазирани, «*Алгоритмы*». Перевод под редакцией А. Шеня. 2014г. - 320с.

2.И.Л. Акулич, «*Математическое программирование в примерах и задачах*». 2011г. - 352с. 3.М. Иванов, «*e-maxx :: algo*». 2012г. - 491с.

4.Б. Страуструп, «*Язык программирования С++*». 1985г. - 589с.

5.Д. Кнут, О. Паташник, Р. Грэхем, «*Конкретная математика*». 1988г. - 784с. 6.С. С. Эпп, «*Дискретная математика и ее применение*». 1990г. - 414с.

7.С. Марсленд, «*Машинное обучение*». 2009г. - 284 с. 8.Т. М. Митчел, «*Машинное обучение*». 1997г. - 319с. 9.С. Хайкин, «*Нейронные сети*». 1994г. - 470с.

10.С. Хайкин, «*Нейронные сети и машинное обучение*». 1993г. - 361с. 11.Д. Кнут, «*Искусство программирования*». 2008г. - 720с.

12. Д. Бентли, «*Жемчужины программирования*». 2002г. - 272с.

**Приложение 1: графики быстродействия и точности методов**

**Приложение 2: исходный код на языке c++ 14**

#define CLR\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <vector>

#include <map>

#include <tuple>

#include <set>

#include <string>

#include <queue>

#include <deque>

#include <stack>

#include <cstdio>

#include <regex>

#include <limits.h>

#include <float.h>

#include <utility>

#include <sstream>

#include <bitset>

#include <stdio.h>

#include <list>

#include <Windows.h>

#include <WinBase.h>

#include <ctime>

#include <io.h>

#include <fcntl.h>

#define pub push\_back

#define puf push\_front

#define pob pop\_back

#define pof pop\_front

using namespace std;

typedef long long ll;

typedef long double ld;

typedef vector<ll> arr;

typedef vector<arr> matrix;

typedef vector<matrix> cube;

const int inf = 1e9 + 7;

ll m, n, main\_counter, f\_i, f\_j, i\_startcycle, j\_startcycle, max\_dr = 0; //размер таблицы M\*N; главный счетчик итераций поиска цикла; разницы по столбцу и строке; начало и конец цикла; абсолютный максимум по спросу и предложению

arr cycle\_i, cycle\_j, request, offer, request\_leftover, offer\_leftover, u, v; //массивы с координатами цикла; спроса, предложения и их остатков; потенциалов по строкам и столбцам

matrix prices, shipment, basis\_cells; //матрицы тарифов, объемов поставок и базисных ячеек

void copy\_arr(arr& from, arr& to) {

to.resize(from.size());

for (ll i = 0; i < from.size(); ++i) {

to[i] = from[i];

}

}

void copy\_matrix(matrix& from, matrix& to) {

to.resize(from.size());

for (ll i = 0; i < from.size(); ++i) {

copy\_arr(from[i], to[i]);

}

}

void copy\_cube(cube& from, cube& to) {

to.resize(from.size());

for (ll i = 0; i < from.size(); ++i) {

copy\_matrix(from[i], to[i]);

}

}

void input() {

/\*fstream fin("‪d:\example1.txt");\*/

cin >> m >> n;

prices.resize(m + 1);

shipment.resize(m + 1);

basis\_cells.resize(m + 1);

for (ll i = 0; i <= m; ++i) {

prices[i].resize(n + 1);

shipment[i].resize(n + 1);

basis\_cells[i].resize(n + 1);

}

request.resize(n + 1);

request\_leftover.resize(n + 1);

u.resize(m + 1);

v.resize(n + 1);

offer.resize(m + 1);

offer\_leftover.resize(m + 1);

for (ll i = 1; i <= m; ++i) {

for (ll j = 1; j <= n; ++j) {

cin >> prices[i][j];

}

}

for (ll i = 1; i <= n; ++i) {

cin >> request[i];

if (request[i] > max\_dr) {

max\_dr = request[i];

}

}

for (ll i = 1; i <= m; ++i) {

cin >> offer[i];

if (offer[i] > max\_dr) {

max\_dr = offer[i];

}

}

return;

}

void output() {

for (ll i = 1; i <= m; ++i) {

cout << endl;

for (ll j = 1; j <= n; ++j) {

cout << shipment[i][j] << '\*' << prices[i][j] << '\t';

}

}

cout << endl;

return;

}

void to\_zero() {

for (ll j = 1; j <= n; ++j) {

request\_leftover[j] = request[j];

}

for (ll i = 1; i <= m; ++i) {

offer\_leftover[i] = offer[i];

}

for (ll i = 1; i <= m; ++i) {

for (ll j = 1; j <= n; ++j) {

basis\_cells[i][j] = 0;

shipment[i][j] = 0;

}

}

return;

}

//---МЕТОД СЕВЕРО-ЗАПАДНОГО УГЛА---

//также заполняет БазисныеЯчейки[i][j] значениями 1 или 0 (если есть ненулевая отгрузка, то ячейка - базисная)

void nc\_method() {

for (ll i = 1; i <= m; ++i) {

for (ll j = 1; j <= n; ++j) {

if (offer\_leftover[i] == 0) {

break;

}

else {

if (offer\_leftover[i] < 0) {

cerr << "Ошибка: leftover предложения меньше 0\n";

}

}

ll space = request\_leftover[j];

if (space == 0) {

continue;

}

else {

if (space < 0) {

cerr << "Ошибка: leftover спроса меньше 0\n";

}

}

if (offer\_leftover[i] < space) {

space = offer\_leftover[i];

}

request\_leftover[j] -= space;

offer\_leftover[i] -= space;

basis\_cells[i][j] = 1;

shipment[i][j] = space;

}

}

return;

}

//---МЕТОД МИНИМАЛЬНОГО ЭЛЕМЕНТА---

void me\_method() {

for (ll max\_count = 1; max\_count <= m\*n; ++max\_count) { //максимальное число итераций: M\*N

ll min\_rate = inf, min\_i = inf, min\_j = inf;

bool allcargowasshipped = true;

ll i, j;

for (i = 1; i <= m; ++i) {

for (j = 1; j <= n; ++j) {

if (basis\_cells[i][j] == 1) {

continue;

}

if (offer\_leftover[i] == 0) {

continue;

}

else {

if (offer\_leftover[i] < 0) {

cerr << "Ошибка: leftover предложения меньше 0\n";

}

}

if (request\_leftover[j] == 0) {

continue;

}

else {

if (request\_leftover[j] < 0) {

cerr << "Ошибка: leftover спроса меньше 0\n";

}

}

allcargowasshipped = false;

ll fs = prices[i][j];

if (min\_rate == inf) {

min\_rate = fs;

min\_i = i;

min\_j = j;

}

else {

if (fs < min\_rate) {

min\_rate = fs;

min\_i = i;

min\_j = j;

}

}

}

}

i = min\_i;

j = min\_j;

if (allcargowasshipped) {

return;

}

if (min\_rate == inf) {

cerr << "Не удалось определить минимальный тариф\n";

}

ll space = request\_leftover[j]; //минимальный тариф найден в ячейке i, j

if (offer\_leftover[i] < space) {

space = offer\_leftover[i];

}

request\_leftover[j] = request\_leftover[j] - space;

offer\_leftover[i] = offer\_leftover[i] - space;

basis\_cells[i][j] = 1;

shipment[i][j] = space;

}

cerr << "Не удалось распределить методом минимальных тарифов\n";

return;

}

//вычисляет первую минимальную цену в указанной строке, обходит стороной ячейки, где остаток спроса = 0, возвращает минимальную цену, а j минимальной ячейки - в переменной f\_j

ll f\_first\_min\_price\_by\_string(ll i) {

ll min1 = inf, f\_j = inf;

for (ll j = 1; j <= n; ++j) {

if (request\_leftover[j] == 0) {

continue;

}

else {

if (request\_leftover[j] < 0) {

cerr << "Ошибка: leftover спроса меньше 0\n";

}

ll max\_count = prices[i][j];

if (min1 == inf) {

min1 = max\_count;

f\_j = j;

}

else {

if (min1 > max\_count) {

min1 = max\_count;

f\_j = j;

}

}

}

return min1;

}

}

//вычисляет вторую минимальную цену в указанной строке, обходит стороной ячейки, где остаток спроса = 0 и f\_j, возвращает минимальную цену, а j минимальной ячейки - в переменной f\_j

ll f\_second\_min\_price\_by\_string(ll i) {

ll min2 = inf;

if (f\_j == inf) {

return inf;

}

for (ll j = 1; j <= n; ++j) {

if (j == f\_j) {

continue;

}

if (request\_leftover[j] == 0) {

continue;

}

ll fs = prices[i][j];

if (min2 == inf) {

min2 = fs;

}

else {

if (min2 > fs) {

min2 = fs;

}

}

}

return min2;

}

//вычисляет первую минимальную цену по указанному столбцу, обходит стороной ячейки, где остаток предложения = 0, возвращает минимальную цену, а i минимальной ячейки - в переменной f\_i

ll f\_first\_min\_price\_by\_column(ll j) {

ll min1 = inf, f\_i = inf;

for (ll i = 1; i <= m; ++i) {

if (offer\_leftover[i] == 0) {

continue;

}

else {

if (offer\_leftover[i] < 0) {

cerr << "Ошибка: leftover предложения меньше 0\n";

}

}

ll max\_count = prices[i][j];

if (min1 == inf) {

min1 = max\_count;

f\_i = i;

}

else {

if (min1 > max\_count) {

min1 = max\_count;

f\_i = i;

}

}

}

return min1;

}

//вычисляет вторую минимальную цену по указанному столбцу, обходит стороной ячейки, где остаток предложения = 0 и f\_i, возвращает минимальную цену, а i минимальной ячейки - в переменной f\_i

ll f\_second\_min\_price\_by\_column(ll j) {

ll min2 = inf;

if (f\_i == inf) {

return inf;

}

for (ll i = 1; i <= m; ++i) {

if (i == f\_i) {

continue;

}

if (offer\_leftover[i] == 0) {

continue;

}

ll max\_count = prices[i][j];

if (min2 == inf) {

min2 = max\_count;

f\_i = i;

}

else {

if (min2 > max\_count) {

min2 = max\_count;

f\_i = i;

}

}

}

return min2;

}

ll f\_difference\_between\_min\_prices\_by\_string(ll i) {

ll price1 = f\_first\_min\_price\_by\_string(i), price2 = f\_second\_min\_price\_by\_string(i);

if (price2 == inf || price1 == inf) {

return 0;

}

return price2 - price1;

}

ll f\_difference\_by\_min\_prices\_by\_column(ll j) {

ll price1 = f\_first\_min\_price\_by\_column(j), price2 = f\_second\_min\_price\_by\_column(j);

if (price1 == inf || price2 == inf) {

return 0;

}

return price2 - price1;

}

//---МЕТОД АППРОКСИМАЦИИ ФОГЕЛЯ---

void f\_method() {

for (ll j = 1; j <= n; ++j) {

request\_leftover[j] = request[j];

}

for (ll i = 1; i <= m; ++i) {

offer\_leftover[i] = offer[i];

}

for (ll i = 1; i <= m; ++i) {

for (ll j = 1; j <= n; ++j) {

basis\_cells[i][j] = 0;

shipment[i][j] = 0;

}

}

for (ll max\_count = 1; max\_count <= 100; ++max\_count) {

ll max1 = inf, max1\_i = inf, i, max2 = inf, max2\_j = inf, j;

for (i = 1; i <= m; ++i) {

if (offer\_leftover[i] == 0) {

continue;

}

ll delta = f\_difference\_between\_min\_prices\_by\_string(i);

if (max1\_i == inf) {

max1 = delta;

max1\_i = i;

}

else {

if (max1 < delta) {

max1 = delta;

max1\_i = i;

}

}

}

for ( j = 1; j <= n; ++j) {

if (request\_leftover[j] == 0) {

continue;

}

ll delta = f\_difference\_by\_min\_prices\_by\_column(j);

if (max2\_j == inf) {

max2 = delta;

max2\_j = j;

}

else {

if (max2 < delta) {

max2 = delta;

max2\_j = j;

}

}

}

if (max1 == inf && max2 == inf) { //все отгрузили

break;

}

else {

if (max1 == inf) {

max1 = max2 - 1;

}

else {

if (max2 == inf) {

max2 = max1 - 1;

}

}

}

if (max1 > max2) { //предпочтительна строка max1

//mn = f\_first\_min\_price\_by\_string(max1\_i);

i = max1\_i;

j = f\_j;

}

else { //предпочтителен столбец max2

//mn = f\_first\_min\_price\_by\_column(max2\_j);

i = f\_i;

j = max2\_j;

}

ll space = request\_leftover[j];

if (offer\_leftover[i] < space) {

space = offer\_leftover[i];

}

request\_leftover[j] = request\_leftover[j] - space;

offer\_leftover[i] = offer\_leftover[i] - space;

basis\_cells[i][j] = 1;

shipment[i][j] = space;

}

return;

}

//---МЕТОД ПОТЕНЦИАЛОВ---

//проверяет правильность отгрузок: отгрузки по каждой строке должны быть равны предложению, а отгрузки по каждому столбцу - спросу потребителя

bool check\_correctness\_of\_shipment() {

for (ll i = 1; i <= m; ++i) {

ll space = 0;

for (ll j = 1; j <= n; ++j) {

space += shipment[i][j];

}

if (space != offer[i]) {

cerr << "Ошибка: shipment по строке не равны предложению в строке " << i << endl;

}

}

for (ll j = 1; j <= n; ++j) {

ll space = 0;

for (ll i = 1; i <= m; ++i) {

space += shipment[i][j];

}

if (space != request[j]) {

cerr << "Ошибка: shipment по столбцу не равны спросу в столбце " << j << endl;

}

}

return true;

}

void calc\_potential\_by\_horizontal(ll i);

void calc\_potential\_by\_vertical(ll j);

void calc\_potential\_by\_vertical(ll j) {

if (v[j] == inf) {

cerr << "Ошибка Получения потенциала v[" << j << "]\n";

}

for (ll i = 1; i <= m; ++i) {

if (basis\_cells[i][j] == 0) {

continue;

}

if (u[i] != inf) {

continue;

}

else {

u[i] = prices[i][j] - v[j];

calc\_potential\_by\_horizontal(i);

}

}

}

void calc\_potential\_by\_horizontal(ll i) {

main\_counter--;

if (main\_counter == 0) {

cerr << "Зацикливание при вычислении Потенциалов\n";

}

if (u[i] == inf) {

cerr << "Ошибка получения потенциала u[" << i << "]\n";

}

for (ll j = 1; j <= n; ++j) {

if (basis\_cells[i][j] == 0) {

continue;

}

if (v[j] != inf) {

continue;

}

else {

v[j] = prices[i][j] - u[i];

calc\_potential\_by\_vertical(j);

}

}

}

//вычисление потенциалов u[i] и v[j]

bool calc\_potentials() {

ll i, j;

for (i = 1; i <= m; ++i) {

u[i] = inf;

}

for (j = 1; j <= n; ++j) {

v[j] = inf;

}

u[1] = 0;

main\_counter = m\*n; //максимальное число итераций

calc\_potential\_by\_horizontal(1); //начало рекурсии

for (ll i = 1; i <= m; ++i) { //проверка

if (u[i] == inf) {

cout << "\nНе удалось вычислить Потенциал u[" << i << "]\n";

return false;

}

}

for (j = 1; j <= n; ++j) {

if (v[j] == inf) {

cout << "\nНе удалось вычислить Потенциал v[" << j << "]\n";

return false;

}

}

return true;

}

//по известным потенциалам u и v, а также ценам, вычисляет, оптимальное ли решение, если решение не оптимально, находит ячейку i1,j1 с минимальной отрицательной дельтой, откуда будем строить цикл

bool check\_optimality\_of\_solving() {

bool solvingisoptimal = true;

ll mindelta = inf, i, j, delta;

for (ll i = 1; i <= m; ++i) {

for (ll j = 1; j <= n; ++j) {

if (basis\_cells[i][j] == 1) {

delta = 0;

}

else {

delta = prices[i][j] - u[i] - v[j];

}

if (delta < 0) {

solvingisoptimal = false;

}

if (mindelta == inf) {

mindelta = delta;

i\_startcycle = i;

j\_startcycle = j;

}

else {

if (delta < mindelta) {

mindelta = delta;

i\_startcycle = i;

j\_startcycle = j;

}

}

}

}

return solvingisoptimal;

}

//вычисляет суммарную стоимость транспортировки

ll full\_price\_of\_shipment() {

ll sum = 0;

for (ll i = 1; i <= m; ++i) {

for (ll j = 1; j <= n; ++j) {

sum += (shipment[i][j] \* prices[i][j]);

}

}

return sum;

}

//если решение вырождено, то надо ввести в число базисных переменную с нулевой отгрузкой

bool find\_zero\_cell\_to\_insert\_in\_basis() {

ll is\_ok = 0; //проверка вселенной на стабильность

for (ll i = 1; i <= m; ++i) {

for (ll j = 1; j <= n; ++j) {

if (basis\_cells[i][j] == 0) {

is\_ok = 1;

break;

}

}

if (is\_ok == 1) {

break;

}

}

if (is\_ok == 0) {

cerr << "Не существует не базисной (нулевой) ячейки цикла ввода в базис\n";

}

while (1) {

/\*случайные значения для предотвращения зацикливания, согласно рекомендации Данцига:

Дж. Данциг «Линейное программирование, его применения и обобщения» 1966г. - стр. 312\*/

ll i = 1 + rand() % m, j = 1 + rand() % n;

if (basis\_cells[i][j] == 1) { //пропускаем базисные ячейки

continue;

}

if (shipment[i][j] != 0) {

cerr << "Ненулевые shipment цикла не базисной ячейки\n";

}

/\*if (Найти(i, j)) { //Пропускаем ячейки, которые образуют цикл

continue;

}\*/

basis\_cells[i][j] = 1;

cout << "В базис введена ячейка " << i << " " << j << endl;

return true; //удалось найти ячейку

}

}

/\*bool find\_cycle\_by\_horizontal(ll i0, ll j0);\*/

/\*bool find\_cycle\_by\_vertical(ll i0, ll j0);\*/

/\*bool find\_cycle\_by\_horizontal(ll i0, ll j0) {

main\_counter--;

if (main\_counter == 0) {

cerr << "Слишком большое число итераций при поиске цикла\n";

return false;

}

for (ll j = 1; j <= n; ++j) {

if (j == j0 || basis\_cells[i0][j] == 0) {

continue;

}

if (find\_cycle\_by\_vertical(i0, j)) {

cycle\_i.push\_back(i0);

cycle\_j.push\_back(j);

return true;

}

}

return false;

}\*/

/\*bool find\_cycle\_by\_vertical(ll i0, ll j0) {

for (ll i = 1; i <= m; ++i) {

if (j0 == j\_startcycle && i == i\_startcycle) { //попали в начальную точку

cycle\_i.push\_back(i);

cycle\_j.push\_back(j0);

return true; //это победа, цикл найден

}

if (i == i0 || basis\_cells[i][j0] == 0) {

continue;

}

if (find\_cycle\_by\_horizontal(i, j0)) {

cycle\_i.push\_back(i);

cycle\_j.push\_back(j0);

return true;

}

}

return false; //это не победа, цикл не найден

}\*/

//поиск цикла для перераспределения поставок, заполняет массивы cycle\_i и cycle\_j с координатами вершин цикла

/\*bool find\_cycle(ll i0, ll j0) {

main\_counter = 4\*m\*n; //максимальное число итераций: M\*N

cycle\_i.clear();

cycle\_j.clear();

if (find\_cycle\_by\_horizontal(i0, j0)) {

return true; //нашли цикл

}

return false; //цикл не найден

}\*/

//обходит граф в глубину

bool dfs(matrix& graph, arr& used, arr& parent, ll v, ll x, bool flag) {

for (ll i = 0; i < graph[v].size(); ++i) {

if (!used[graph[i][v]] && used[graph[i][v]] != 2) { //если не были в вершине и с нее не начинается цикл, неокружающий клетку i0, j0

used[graph[i][v]] = 1;

parent[graph[i][v]] = v;

dfs(graph, used, parent, graph[i][v], x, flag);

}

else {

arr cycle;

ll t = v;

while (parent[t] != v) {

cycle.pub(t);

v = parent[t];

}

cycle.pub(v);

sort(cycle.begin(), cycle.end());

if (cycle[0] < x && cycle[n - 1] > x && cycle[0] < cycle[((cycle.size()) / 2 - 1) / 2] &&

cycle[((cycle.size()) / 2 - 1) / 2] < x && cycle[cycle.size() - 1] > cycle[cycle.size() / 2 - 1 + (((cycle.size()) / 2 - 1) / 2)] &&

cycle[cycle.size() / 2 - 1 + (((cycle.size()) / 2 - 1) / 2)] > x) {

return true;

}

else { //если цикл клетку окружает, то все плохо: помечаем вершину ОЧЕНЬ ПЛОХОЙ

used[graph[i][v]] = 2;

for (ll i = 0; i < n \* m + n + m; ++i) { //удалим все значения из массива предков и путей, кроме ОЧЕНЬ ПЛОХИХ КЛЕТОК

if (used[i] == 1) {

used[i] = 0;

}

parent[i] = -1;

}

if (flag) {

flag = false;

dfs(graph, used, parent, v, x, flag); //попробуем еще разок

}

return false;

}

}

}

return false;

}

void dfs(matrix& graph, arr& used, ll v, ll counter) {

for (ll i = 0; i < graph[v].size(); ++i) {

if (!used[graph[v][i]]) {

used[graph[v][i]] = 1;

dfs(graph, used, graph[v][i], counter);

}

}

return;

}

bool find\_cycle(ll i0, ll j0) {

cycle\_i.clear();

cycle\_j.clear();

vector<pair<ll, ll>>cycle;

pair<ll, ll>top, top\_right, right, down\_right, down, down\_left, left, top\_left;

ll i\_now = i0, j\_now = j0;

while (i\_now < m && j\_now < n && !basis\_cells[i\_now][j\_now]) {

i\_now++;

j\_now++;

}

if (i\_now <= m && j\_now <= n && basis\_cells[i\_now][j\_now]) {

top\_right = { i\_now, j\_now };

}

else {

return false;

}

i\_now = i0;

j\_now = j0;

while (i\_now > 1 && j\_now > 1 && !basis\_cells[i\_now][j\_now]) {

i\_now--;

j\_now--;

}

if (i\_now >= 1 && j\_now >= 1 && basis\_cells[i\_now][j\_now]) {

down\_left = { i\_now, j\_now };

}

else {

return false;

}

i\_now = i0;

j\_now = j0;

while (i\_now > 1 && !basis\_cells[i\_now][j\_now]) {

i\_now--;

}

if (i\_now >= 1 && basis\_cells[i\_now][j\_now]) {

down = { i\_now, j\_now };

}

else {

return false;

}

i\_now = i0;

j\_now = j0;

while (i\_now < m && !basis\_cells[i\_now][j\_now]) {

i\_now++;

}

if (i\_now <= m && basis\_cells[i\_now][j\_now]) {

top = { i\_now, j\_now };

}

else {

return false;

}

i\_now = i0;

j\_now = j0;

while (i\_now > 1 && j\_now < n && !basis\_cells[i\_now][j\_now]) {

i\_now--;

j\_now++;

}

if (i\_now > 1 && j\_now < n && basis\_cells[i\_now][j\_now]) {

down\_right = { i\_now, j\_now };

}

else {

return false;

}

i\_now = i0;

j\_now = j0;

while (i\_now < m && j\_now > 1 && !basis\_cells[i\_now][j\_now]) {

i\_now++;

j\_now--;

}

if (i\_now <= m && j\_now >= 1 && basis\_cells[i\_now][j\_now]) {

top\_left = { i\_now, j\_now };

}

else {

return false;

}

i\_now = i0;

j\_now = j0;

while (j\_now < n && !basis\_cells[i\_now][j\_now]) {

j\_now++;

}

if (j\_now <= n && basis\_cells[i\_now][j\_now]) {

right = { i\_now, j\_now };

}

else {

return false;

}

i\_now = i0;

j\_now = j0;

while (j\_now > 1 && !basis\_cells[i\_now][j\_now]) {

j\_now--;

}

if (j\_now >= 1 && basis\_cells[i\_now][j\_now]) {

left = { i\_now, j\_now };

}

else {

return false;

}

i\_now = i0;

j\_now = j0;

matrix graph(n \* m + n + m);

for (ll i = 1; i <= m; ++i) { //построим списки смежности для графа

for (ll j = 1; j <= n; ++j) {

if (basis\_cells[i][j]) {

if (j < n && basis\_cells[i][j + 1]) {

graph[(i - 1)\*j + j].pub((i - 1) \* j + j + 1);

}

if (j > 1 && basis\_cells[i][j - 1]) {

graph[(i - 1)\*j + j].pub((i - 1) \* j + j - 1);

}

if (i < m && basis\_cells[i + 1][j]) {

graph[(i - 1)\*j + j].pub(i \* j + j);

}

if (i > 1 && graph[i - 1][j]) {

graph[(i - 1)\*j + j].pub((i - 2) \* j + j);

}

}

}

}

arr used(n \* m + n + m), parent(n \* m + n + m);

ll counter = 0;

for (ll i = 1; i <= n \* m + n + m; ++i) { //разбиение и проверка компонент связности

if (!used[i]) {

counter++;

dfs(graph, used, i, counter);

}

}

if (!dfs(graph, used, parent, (top.first - 1) \* top.second + top.second, (i0 - 1) \* j0 + j0, true) &&

!dfs(graph, used, parent, (down.first - 1) \* down.second + down.second, (i0 - 1) \* j0 + j0, true) &&

!dfs(graph, used, parent, (left.first - 1) \* left.second + left.second, (i0 - 1) \* j0 + j0, true) &&

!dfs(graph, used, parent, (top\_right.first - 1) \* top\_right.second + top\_right.second, (i0 - 1) \* j0 + j0, true) &&

!dfs(graph, used, parent, (top\_left.first - 1) \* top\_left.second + top\_left.second, (i0 - 1) \* j0 + j0, true) &&

!dfs(graph, used, parent, (down\_right.first - 1) \* down\_right.second + down\_right.second, (i0 - 1) \* j0 + j0, true) &&

!dfs(graph, used, parent, (down\_left.first - 1) \* down\_left.second + down\_left.second, (i0 - 1) \* j0 + j0, true)) { //попытаемся найти цикл обходом в глубину, использующим отсечение неокружающих клетку i0, j0

cerr << "\nНе удалось найти цикл.\n";

return false;

}

else {

cycle\_i.resize(cycle.size());

cycle\_j.resize(cycle.size());

for (ll i = 0; i < cycle.size(); ++i) {

cycle\_i[i] = cycle[i].first;

cycle\_j[i] = cycle[i].second;

}

return true;

}

}

//сдвиг по циклу пересчета cycle\_i, cycle\_j

bool recalc\_by\_cycle() {

cout << "\nПерераспределение по циклу " << cycle\_i.size() << endl;

if (cycle\_j.size() != cycle\_i.size()) {

cerr << "Не одинаковые размерности циклов координат цикла\n";

}

if (cycle\_i.size() < 4) {

cerr << "Цикл имеет меньше 4 элементов\n";

}

ll tetha = inf, sign = 1;

for (ll max\_count = 0; max\_count <= cycle\_i[cycle\_i.size() - 1]; ++max\_count) {

ll i = cycle\_i[max\_count], j = cycle\_j[max\_count], amount;

if (sign == -1) {

amount = shipment[i][j];

if (tetha == inf) {

tetha = amount;

}

else {

if (amount < tetha) {

tetha = amount;

}

}

sign = 1;

}

else {

sign = -1;

}

}

if (tetha == inf) {

cerr << "Не удалось вычислить tetha\n";

}

cout << "\ntetha = " << tetha << endl;

if (tetha == 0) {

return false;

}

sign = 1;

for (ll max\_count = 0; max\_count <= cycle\_i[cycle\_i.size() - 1]; ++max\_count) {

ll i = cycle\_i[max\_count], j = cycle\_j[max\_count];

if (sign == -1) {

shipment[i][j] -= tetha;

sign = 1;

}

else {

shipment[i][j] += tetha;

sign = -1;

}

}

return true;

}

void p\_method() {

cout << "\n\nМЕТОД ПОТЕНЦИАЛОВ\n\n";

while (1) {

check\_correctness\_of\_shipment();

ll countofbasis = 0;

for (ll i = 1; i <= m; ++i) {

for (ll j = 1; j <= n; ++j) {

if (shipment[i][j] > 0) {

basis\_cells[i][j] = 1;

countofbasis++;

}

else {

if (shipment[i][j] < 0) {

cerr << "shipment не должны быть отрицательными\n";

}

else {

basis\_cells[i][j] = 0;

}

}

}

while (countofbasis < (m + n - 1)) {

cout << "Решение вырождено\n";

find\_zero\_cell\_to\_insert\_in\_basis();

countofbasis++;

}

if (!calc\_potentials()) {

continue;

}

if (check\_optimality\_of\_solving()) {

cout << "\nРЕШЕНИЕ ОПТИМАЛЬНО\n"; //нашли опимальное решение

output();

cout << "\nСтоимость перевозки: " << full\_price\_of\_shipment() << endl;

return;

}

cout << "Решение не оптимально\n";

if (!find\_cycle(i\_startcycle, j\_startcycle)) {

cerr << "Не удалось найти цикл\n";

}

else {

recalc\_by\_cycle();

}

cout << "Стоимость перевозки: " << full\_price\_of\_shipment();

}

}

}

//---МЕТОД ДИФФЕРЕНЦИАЛЬНЫХ РЕНТ---

void dr\_method() {

cout << "\n\nМЕТОД ДИФФЕРЕНЦИАЛЬНЫХ РЕНТ\n\n";

for (ll i = 1; i <= m; ++i) {

for (ll j = 1; j <= n; ++j) {

if (prices[i][j]) {

}

}

}

}

/\*bool kuhn(int from, arr& mt, arr& used, matrix& graph) {

if (used[from]) {

return false;

}

used[from] = true;

for (size\_t i = 0; i < graph[from].size(); ++i) {

int to = graph[from][i];

if (mt[to] == -1 || kuhn(mt[to], mt, used, graph)) {

mt[to] = from;

return true;

}

}

return false;

}\*/

//---МЕТОД ТЕОРИИ ГРАФОВ---

/\*bool gt\_method() {

ll n, s, counter = 0;

matrix graph(s);

arr used, mt(n, -1);

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

used.assign(s, false);

if (kuhn(i, mt, used, graph)) {

counter++;

}

}

}\*/

//---МЕТОД

/\*bool s\_method() {

}\*/

int main() {

setlocale(LC\_ALL, "Russian");

input();

ll frequest = 0;

for (ll j = 1; j <= n; ++j) {

frequest += request[j];

}

ll foffer = 0;

for (ll i = 1; i <= m; ++i) {

foffer += offer[i];

}

if (foffer > frequest) {

cout << "offer больше спроса на " << foffer - frequest << " единиц груза. Создайте фиктивного потребителя.\n";

}

else {

if (foffer < frequest) {

cout << "offer меньше спроса на " << frequest - foffer << " единиц груза. Создайте фиктивного поставщика.\n";

}

}

/\*добавление возмущений (эпсилон) во избежание зацикливания транспортной задачи

литература:

1) Дж. Данциг «Линейное программирование, его применения и обобщения», 1966г. - стр. 303

2) С.Гасс. «Линейное программирование (методы и приложения)», - стр. 195,

в данном случае не используется, поскольку применен метод случайного включения нулевой перевозки в базис, который также, согласно Данцигу, страхует от зацикливания\*/

/\*ld epsilon = 0.0001, epsilon1= 0;

for (ll j = 1; j <= n; ++j) {

request[j] += epsilon;

epsilon1 += epsilon;

}

offer[1] += epsilon1;\*/

cout << endl;

to\_zero();

ll tick = clock();

nc\_method();

ll now = clock();

cout << endl << "Для вычисления понадобилось (сек): " << ((ld)(now - tick)) / CLOCKS\_PER\_SEC << endl;

cout << "Стоимость перевозки методом северо-западного угла: " << full\_price\_of\_shipment();

output();

to\_zero();

cout << endl;

tick = clock();

me\_method();

now = clock();

cout << endl << "Для вычисления понадобилось (сек): " << ((ld)(now - tick)) / CLOCKS\_PER\_SEC << endl;

cout << "Стоимость перевозки методом минимальных тарифов: " << full\_price\_of\_shipment();

output();

to\_zero();

cout << endl;

tick = clock();

f\_method();

now = clock();

cout << endl << "Для вычисления понадобилось (сек): " << ((ld)(now - tick)) / CLOCKS\_PER\_SEC << endl;

cout << "Стоимость перевозки методом Фогеля: " << full\_price\_of\_shipment();

output();

cout << endl;

tick = clock();

p\_method();

now = clock();

cout << endl << "Для вычисления понадобилось (сек): " << ((float)(now - tick)) / CLOCKS\_PER\_SEC << endl;

tick = clock();

dr\_method();

now = clock();

cout << endl << "Для вычисления понадобилось (сек): " << ((float)(now - tick)) / CLOCKS\_PER\_SEC << endl;

tick = clock();

//to be continued...

system("pause");

}

/\*EXAMPLES:

2 2

4 7

3 5

10 20

25 5

3 5

5 3 6 2 0

4 7 9 1 0

3 4 7 5 0

16 18 30 25 1

19 37 34

4 6

1 2 1 4 5 2

3 3 2 1 4 3

4 2 5 9 6 2

3 1 7 3 4 6

20 40 30 10 50 25

30 50 75 20

4 6

2 1 3 3 2 5

3 2 2 4 3 4

3 5 4 2 4 1

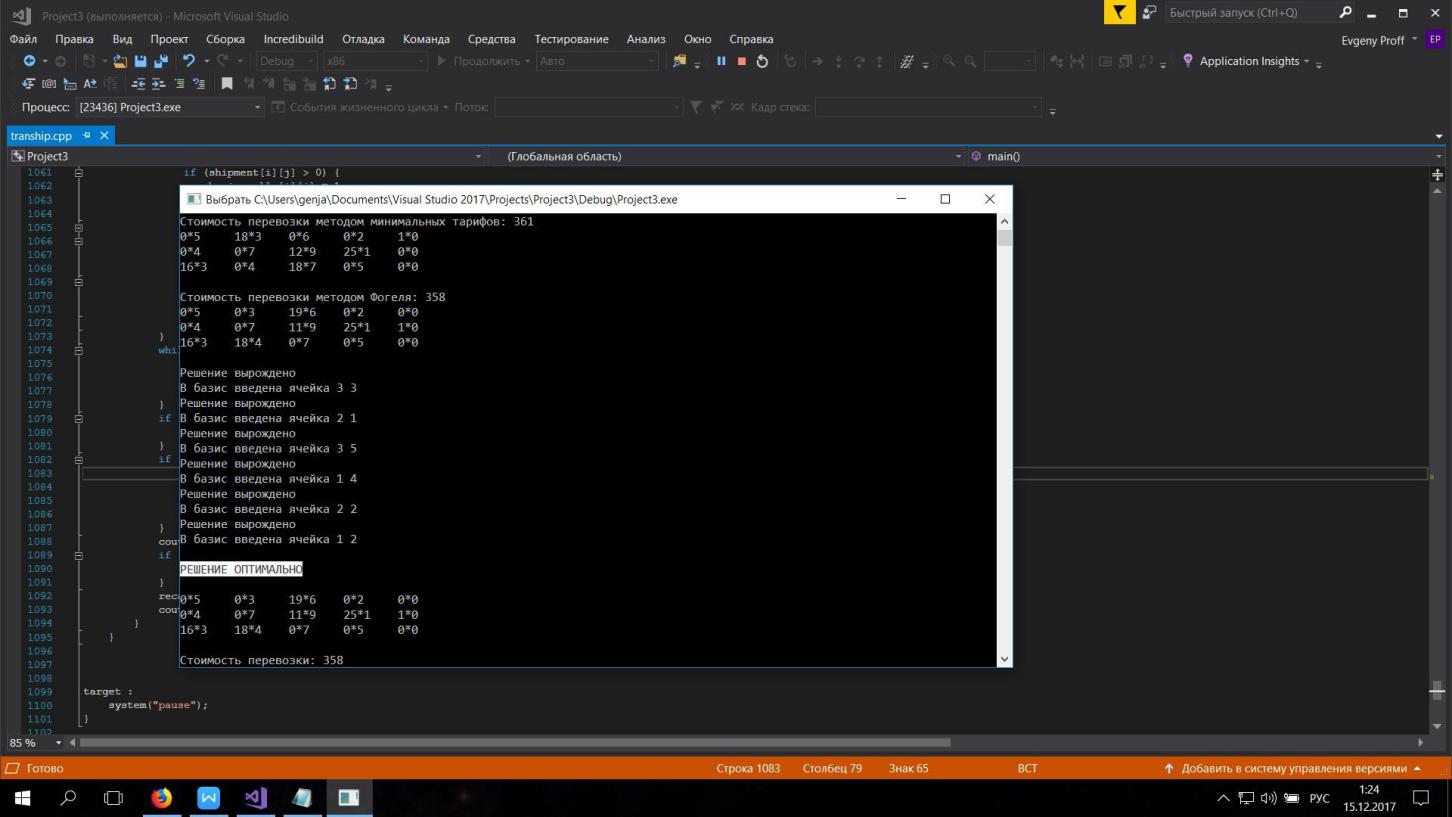
4 2 2 1 2 2

30 50 20 40 30 11

50 40 60 31

\*/

## Приложение 3: пример работы программы



### На данный момент подобрано 20 тестов, ограничение на **M \* N** не превышает 400.