# Задание

Написать программу, на языке С++ в среде VisualStudio “**Расчёт рациона с минимальной стоимостью”:**

Заданы n пищевых продуктов, содержащих m различных питательных веществ. Обозначим через aij содержание (долю) j-го питательного вещества в i-ом продукте, через bj — суточную потребность организма в j-ом питательном веществе, через ci — стоимость единицы i-го продукта. Составить суточный рацион питания минимальной стоимости, удовлетворяющий потребность во всех питательных веществах.

Однородные вычислительные структуры или среды (ОВС), как правило, относятся к классу ОКМД (согласно классификации Флинна) и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

# OpenMP

Среди специалистов, занимающихся параллельными вычислениями, популярна шутка «Параллельные вычисления — технология будущего... и так будет всегда». Эта шутка не теряет актуальность уже несколько десятилетий. Аналогичные настроения были распространены в сообществе разработчиков архитектур компьютеров, обеспокоенном тем, что скоро будет достигнут предел тактовой частоты процессоров, однако частоты процессоров продолжают повышаться, хотя гораздо медленнее, чем раньше. Сплав оптимизма специалистов по параллельным вычислениям и пессимизма архитекторов систем способствовал появлению революционных многоядерных процессоров.

Главные производители процессоров сместили акцент с повышения тактовых частот на реализацию параллелизма в самих процессорах за счет использования многоядерной архитектуры. Идея проста: интегрировать в один процессор более одного ядра. Система, включающая процессор с двумя ядрами, по сути, не отличается от двухпроцессорного компьютера, а система с четырехядерным процессором — от четырехпроцессорного. Этот подход позволяет избежать многих технологических проблем, связанных с повышением тактовых частот, и создавать при этом более производительные процессоры.

Все это прекрасно, но если ваше приложение не будет использовать несколько ядер, его быстродействие никак не изменится. Именно здесь и вступает в игру технология OpenMP, которая помогает программистам на C++ быстрее создавать многопоточные приложения.

Стандарт OpenMP был разработан в 1997 г. как API, ориентированный на написание портируемых многопоточных приложений. Сначала он был основан на языке Fortran, но позднее включил в себя и C/C++. Последняя версия OpenMP — 2.0;  ее полностью поддерживает Visual C++ 2005. Стандарт OpenMP поддерживается и платформой Xbox 360.

в Visual C++ 2005 параметр компилятора /openmp. (Вы можете активизировать директивы OpenMP на страницах свойств проекта, выбрав Configuration Properties, C/C++, Language и изменив значение свойства OpenMP Support.) Встретив параметр /openmp, компилятор определяет символ \_OPENMP, с помощью которого можно выяснить, включены ли средства OpenMP. Для этого достаточно написать #ifndef \_OPENMP.

OpenMP связывается с приложениями через библиотеку импорта vcomp.lib. Соответствующая библиотека периода выполнения называется vcomp.dll. Отладочные версии библиотек импорта и периода выполнения (vcompd.lib и vcompd.dll соответственно) поддерживают дополнительные сообщения об ошибках, генерируемых при некоторых недопустимых операциях. Имейте в виду, что Visual C++ не поддерживает статическое связывание с библиотекой OpenMP периода выполнения, хотя в версии для Xbox 360 это поддерживается.

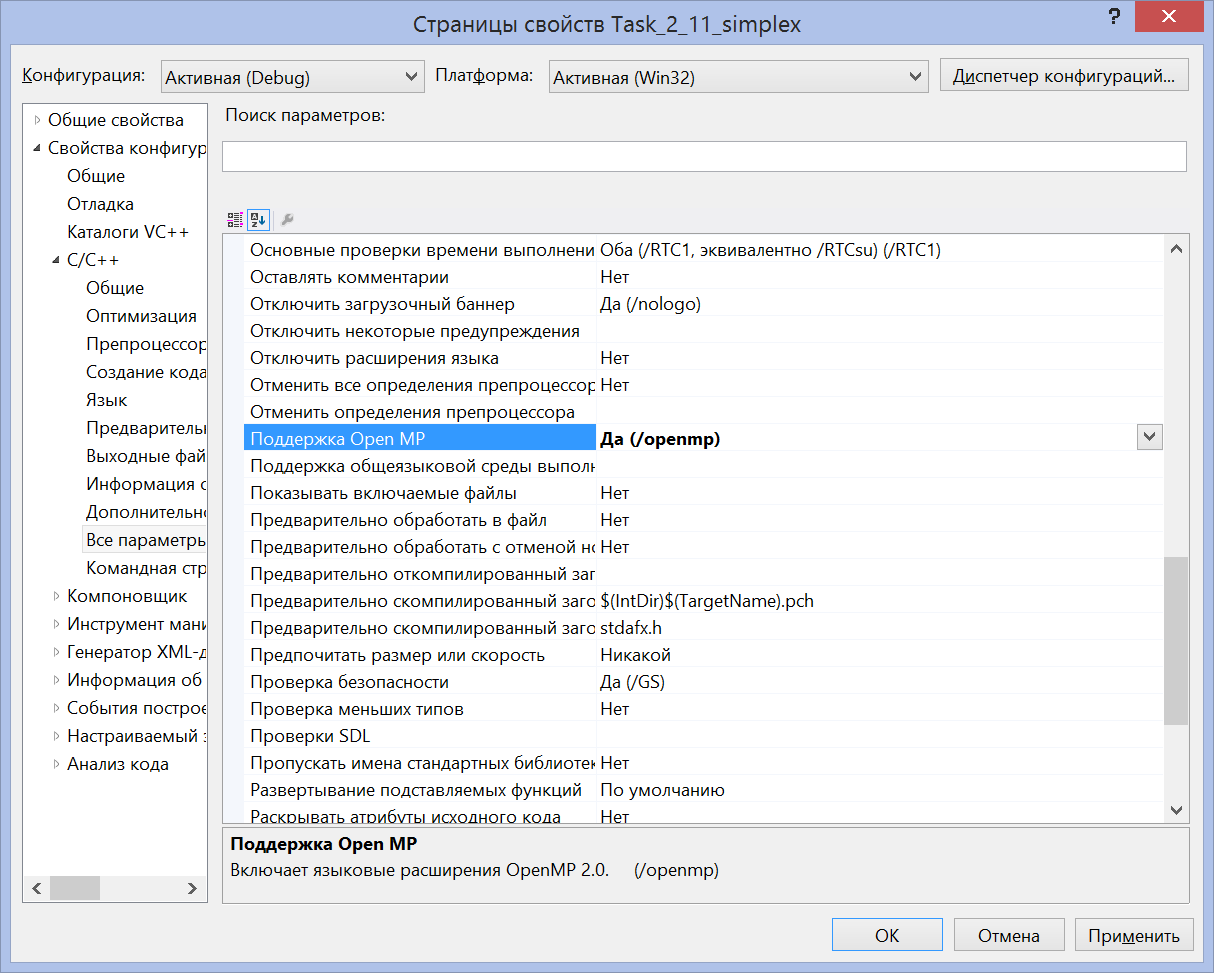
Работа OpenMP-приложения начинается с единственного потока — основного. В приложении могут содержаться параллельные регионы, входя в которые, основной поток создает группы потоков (включающие основной поток). В конце параллельного региона группы потоков останавливаются, а выполнение основного потока продолжается. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждый поток первоначального региона становится основным для своей группы потоков. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности.

OpenMP прост в использовании и включает лишь два базовых типа конструкций: директивы pragma и функции исполняющей среды OpenMP. Директивы pragma, как правило, указывают компилятору реализовать параллельное выполнение блоков кода. Все эти директивы начинаются с #pragma omp. Как и любые другие директивы pragma, они игнорируются компилятором, не поддерживающим конкретную технологию — в данном случае OpenMP.

Функции OpenMP служат в основном для изменения и получения параметров среды. Кроме того, OpenMP включает API-функции для поддержки некоторых типов синхронизации. Чтобы задействовать эти функции библиотеки OpenMP периода выполнения (исполняющей среды), в программу нужно включить заголовочный файл omp.h. Если вы используете в приложении только OpenMP-директивы pragma, включать этот файл не требуется.

Для реализации параллельного выполнения блоков приложения нужно просто добавить в код директивы pragma и, если нужно, воспользоваться функциями библиотеки OpenMP периода выполнения.

OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic, которые определяют или механизмы разделения работы или конструкции синхронизации.



# Расчёт рациона минимальной стоимости

Задача расчёта рациона минимальной стоимости является одной из классических задач линейного программирования – нахождение точки минимума линейной функции f(x)=cx при условии ограничений ax>=b, где b>=0 и x>=0.

Для решения данной задачи был выбран симплекс-метод с введением дополнительных и искусственных переменных. Использовалась расширенная таблица плана симплекс-метода со строками F и M, и задача решалась в два этапа – сперва исключались из базиса искусственные переменные вычислениями по строке M, а затем применялся алгоритм нахождения оптимального плана по строке F.

Программа работает в 2-х режимах:

* с выводом результатов работы на экран (для демонстрации работоспособности);
* без вывода результатов работы на экран, но с определением времени, затрачиваемого на вычисления;

Листинг кода

#include "Header.hpp"

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Вычисление точки минимума линейной функции f(x)=cx

Для неравенства вида ax>=b и ограничения b>=0,x>=0

\*\*a - матрица коэффициентов левой части неравенства

\*b - вектор значений правой части неравенства

\*c - коэффициенты линейной функции

m - количество неравенств (строк)

n - количество переменных (столбцов)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double \*ration(double \*\*a, double \*b, double \*c, int m, int n, int demo){

// Метод искусственного базиса

double \*\*plan; // Симплекс-таблица

int \*index; // Индексы переменных в базисе

int \*flags; // Флаги переменных в базисе

index = new int[m];

flags = new int[m+n];

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

index[i] = m+n+i; // индексы исскуственных переменных

for(int i = 0; i < m+n; i++)

flags[i] = 0; // флаги переменных

plan = new double \*[m+2];

for(int i = 0; i < m+2; i++)

plan[i] = new double[1+m+n];

// используется параллельная обработка элементов массива

// Однородные вычислительные структуры или среды (ОВС), как правило,

// относятся к классу ОКМД (согласно классификации Флинна)

// и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

// Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором

// операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы

// на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

// OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic,

// которые определяют или механизмы разделения работы или конструкции синхронизации.

// Общие переменные доступны всем потокам из группы, поэтому изменения таких переменных в одном потоке видимы другим потокам

// в параллельном регионе. Что касается частных переменных, то каждый поток из группы располагает их отдельными экземплярами,

// поэтому изменения таких переменных в одном потоке никак не сказываются на их экземплярах,

// принадлежащих другим потокам.

// частными являются индексы параллельных циклов for.

#pragma omp parallel for

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

#pragma omp parallel for

for(int j = 0; j < m+n; j++)

plan[i][j+1]=(j<n)?(a[i][j]):((j==(n+i))?-1:0);

// Формируем F строку матрицы

#pragma omp parallel for

for(int j = 0; j < m+n; j++) {

plan[m][j+1] = (j<n)?(-c[j]):0;

}

// Формируем M строку матрицы

#pragma omp parallel for

for(int j = 0; j < m+n; j++) {

plan[m+1][j+1] = (j<n)?0:-1;

if(j<n) for(int i = 0; i < m; i++) plan[m+1][j+1] += a[i][j];

}

#pragma omp parallel for

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

plan[i][0]=b[i];

plan[m][0]=0;

plan[m+1][0]=0;

if(simplex(plan, index, flags, m, n, 1, demo)!=0) return NULL; // Исключение искусственных переменных

if(simplex(plan, index, flags, m, n, 0, demo)!=0) return NULL; // Нахождение решения

double \* result;

result = new double[n]; //результат

for(int i = 0; i < n; i++) result[i]=0;

for(int i = 0; i < m; i++) if(index[i]<n) result[index[i]]=plan[i][0];

for(int i = 0; i < m+2; i++) delete plan[i];

delete plan;

delete index;

delete flags;

return result;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Вычисление симлекс-метода по таблице плана

\*\*plan - таблица плана

\*index - индексы переменных в базисе

\*flags - флаги переменных в базисе

m - количество базисных переменных

n - количество переменных

extended = 0,1 - флаг использования расширенной таблицы

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int simplex(double \*\*plan, int \*index, int \*flags, int m, int n, int extended, int demo) {

while(1){

if(demo!=0) {

cout << "\tТекущий план plan["<<1+m+extended<<","<<1+m+n<<"]:" << endl;

for(int i = 0; i < 1+m+extended; i++) {

for(int j = 0; j < 1+m+n; j++)

cout << plan[i][j] << "\t";

cout << endl;

}

cout << "\tПеременные в базисе index["<<m<<"]:" << endl;

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

cout << index[i] << "\t";

cout << endl;

}

// Проверяем текущий план на оптимальность

// Если все оценки в последней строке плана неположительные, то план оптимален

double maxByRow=0;

for(int j = 0; j < m+n; j++)

if(flags[j]==0 && maxByRow < plan[m+extended][j+1])

maxByRow = plan[m+extended][j+1];

if(maxByRow<=0) return 0; // Найден оптимальный план

// Находим решающий элемент матрицы плана

double maxByMod = 0;

int r=0;

int c=0;

for(int j = 0; j < m+n; j++) if(flags[j]==0 && plan[m+extended][j+1]>0) {

double minByCol=-1;

for(int i = 0; i < m; i++) if(plan[i][j+1]>0) {

minByCol = plan[i][0]/plan[i][j+1];

for(int i1 = i+1; i < m; i++)

if(plan[i1][j+1]>0 && minByCol > plan[i1][0]/plan[i1][j+1])

minByCol = plan[i1][0]/plan[i1][j+1];

break;

}

if(minByCol < 0) return -1; // Решение не найдено

if(maxByMod<minByCol\*plan[m+extended][j+1]) {

maxByMod=minByCol\*plan[m+extended][j+1];

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

if(plan[i][j+1]>0 && minByCol==plan[i][0]/plan[i][j+1]) {

r = i;

c = j+1;

}

}

}

if(demo!=0) cout << "\tРешающий элемент: (" << r << "," << c << ")" << endl;

// Выполняем алгоритм Гаусса-Жордана для заданного решающего элемента матрицы

// используется параллельная обработка элементов массива

gauss\_jordan(plan, r, c, 1+m+extended, 1+m+n);

// Переменную c-1 отмечаем как включённыю в базис вместо предыдущей

if(index[r]<m+n) flags[index[r]]=0;

index[r] = c-1;

flags[index[r]]=1;

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Алгоритм Гаусса-Жордана для заданного решающего элемента матрицы

\*\*data - матрица

r - номер строки решающего элемента

с - номер столбца решающего элемента

m - количество строк матрицы

n - количество столбцов матрицы

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void gauss\_jordan(double \*\*data, int r, int c, int m, int n) {

double d = data[r][c];

double epsilon=1e-7; // точность вычислений с плавающей точкой

// используется параллельная обработка элементов массива

// Однородные вычислительные структуры или среды (ОВС), как правило,

// относятся к классу ОКМД (согласно классификации Флинна)

// и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

// Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором

// операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы

// на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

// OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic,

// которые определяют или механизмы разделения работы или конструкции синхронизации.

// Общие переменные доступны всем потокам из группы, поэтому изменения таких переменных в одном потоке видимы другим потокам

// в параллельном регионе. Что касается частных переменных, то каждый поток из группы располагает их отдельными экземплярами,

// поэтому изменения таких переменных в одном потоке никак не сказываются на их экземплярах,

// принадлежащих другим потокам.

// частными являются индексы параллельных циклов for.

#pragma omp parallel for

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

#pragma omp parallel for

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

if(i!=r && j!=c) {

data[i][j] -= data[r][j]\*data[i][c]/d;

if(abs(data[i][j])<epsilon) data[i][j]=0;

}

#pragma omp parallel for

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

if(j!=c)

data[r][j] /= d;

#pragma omp parallel for

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

if(i!=r)

data[i][c] = 0;

data[r][c] = 1;

# }

#include "Header.hpp"

using namespace std;

// 11. Заданы n пищевых продуктов, содержащих m различных питательных веществ.

// Обозначим через aij содержание (долю) j-го питательного вещества в i-ом продукте,

// через bj — суточную потребность организма в j-ом питательном веществе,

// через ci — стоимость единицы i-го продукта.

// Составить суточный рацион питания минимальной стоимости,

// удовлетворяющий потребность во всех питательных веществах.

int main()

{

int m; // количество неравенств

int n; // количество переменных

double \*\*a;

double \*b;

double \*c;

double \*x;

int demo; // режим программы

setlocale(LC\_ALL, "Russian");

cout << "\tВычисление точки минимума линейной функции f(x)=cx" << endl;

cout << "\tДля неравенства вида ax>=b и ограничения b>=0,x>=0" << endl;

cout << "\tВведите количество неравенств:" << endl; cin >> m;

cout << "\tВведите количество переменных:" << endl; cin >> n;

a = new double\*[m];

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

a[i] = new double[n];

b = new double[m];

c = new double[n];

cout << "\tВведите матрицу a["<<m<<","<<n<<"]:" << endl;

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

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

cin >> a[i][j];

cout << "\tВведите вектор b["<<m<<"]:" << endl;

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

cin >> b[i];

cout << "\tВведите вектор c["<<n<<"]:" << endl;

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

cin >> c[j];

cout << "\tДемонстрационный режим програмы (0 - замер времени, 1 - демонстрация работоспособности):" << endl; cin >> demo;

clock\_t t = clock();

if((x = ration(a, b, c, m, n, demo)) == NULL)

{

cout << "Решение не найдено." << endl;

getchar();

getchar();

return -1;

}

t = clock()-t;

double seconds = ((double)t)/CLOCKS\_PER\_SEC;

cout << "\tВремя выполнения: " << seconds << "sec" << endl;

cout << "\tРешение: " << endl;

for(int j = 0; j < n; j++) cout << x[j] << '\t'; cout << endl;

double f=0; for(int j = 0; j < n; j++) f+=c[j]\*x[j];

cout << "\tЗначение: " << endl;

cout << f << endl;

for(int i = 0; i < m; i++) delete a[i];

delete a;

delete b;

delete c;

delete x;

getchar();

getchar();

return 0;

}

# Контрольные примеры работы программы

Пример 1.

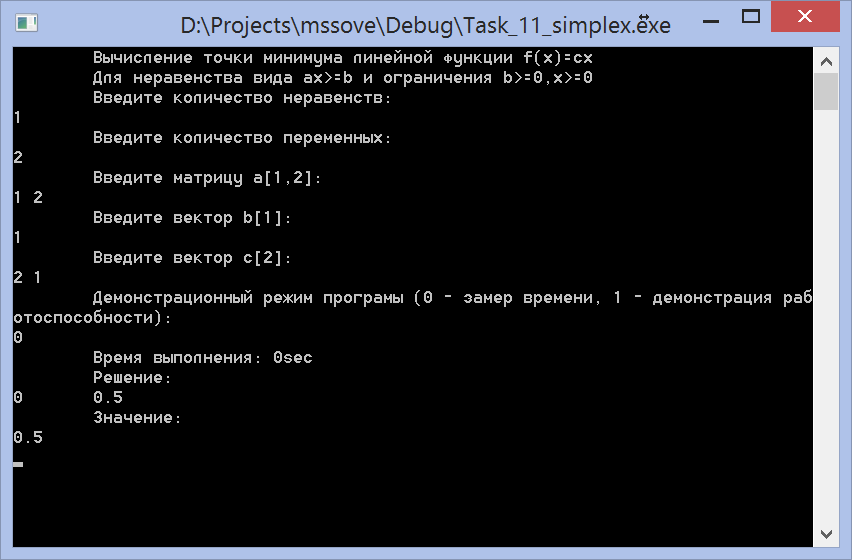


Рис.1..

Пример 2.

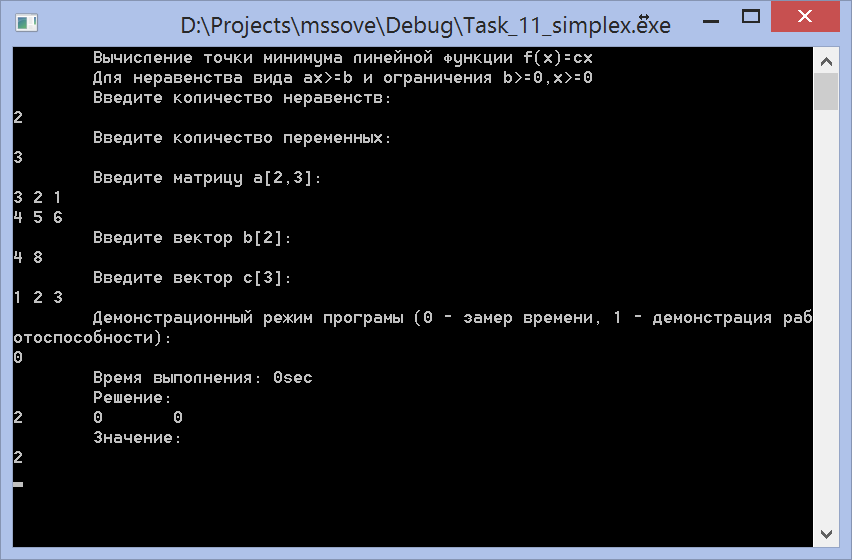


Рис.2..

# Заключение

В ходе данной лабораторной был изучен алгоритм симплекс-метода с введением дополнительных и искусственных переменных. Была сделана программная реализация данного алгоритма. Для данной программной реализации были проведены ряд тестов, показывающие правильность работы алгоритма.