**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский Нижегородский государственный университет**

**им. Н.И. Лобачевского»**

Институт экономики и предпринимательства

Кафедра информационных технологий и инструментальных методов в экономике

Курсовая работа по дисциплине:

«Алгоритмизация и программирование»

На тему: «Решение задачи линейного программирования по нахождению максимума целевой функции»

Работу выполнил студент:

Проверил:

Нижний Новгород

Содержание

1. Вводная часть
2. Теоретическая часть
3. Практическая часть
4. Листинг

Вводная часть

В данной работе будет рассмотрена задача линейного программирования по нахождению максимума целевой функции при заданной системе ограничений.

Задача:

Для производства видов продукции используется типов ресурсов. Известны нормы расхода -го ресурса на единицу -той продукции,

, , запасы ресурсов , , доход , от реализации продукции единицы -той продукции. Определить такой план продукции, при котором суммарный доход достигает максимального значения.

В условиях задачи отсутствует информация, должен ли ответ быть выражен в целочисленном формате, по этой причине, получение ответа с дробью будем считать допустимым.

Таким образом, найти решение данной задачи поможет симплекс-метод.

Эвристического решения для описанной задачи не предусмотрено, однако получить допустимое значение можно в первой итерции симплекс-метода.

Программа была написана на языке программирования C++, запускалась под операционной системой Fedora Linux, компилировалась с помощью компилятора GNU для C++ - g++.

Теоретическая часть

Линейное программирование – это математический численный метод для оптимизации моделей, в которых целевые функции и ограничения строго являются уравнениями линейной алгебры. Модель линейного программирования включает целевую функцию, ограничения в виде линейных уравнений или неравенств и требование неотрицательности переменных.

Целевая функция:

Если функцию требуется обратить не в максимум, а в минимум, достаточно изменить знак коэффициентов на противоположный.

Ограничения

a и b — постоянные числа, заданные условиями задачи. Если по условиям задачи вместо равенств предполагаются неравенства, то для неравенства вида «» для преобразования его в равенство надо добавить дополнительную переменную или несколько таких переменных ( и т.д. по числу неравенств). Аналогично, для неравенств вида «» дополнительную неотрицательную переменную следует вычесть (или, что то же самое, прибавить с коэффициентом –1).

Основным методом в решении задач линейного программирования выступает симплекс-метод.

Он основан на том, что область точек пространства, удовлетворяющих линейным ограничениям, является многогранником, и максимум линейной функции должен достигаться в одной из вершин этого многогранника. На каждом шаге симплекс-метода происходит переход в соседнюю вершину с не худшим значением целевой функции.

Применение симплекс-метода будет рассмотрено в практической части, на примере решения задачи вручную.

Практическая часть

Решение задачи вручную

Начнём практическую часть с ручного решения задачи.

Приведём начальные данные:

Видов продукции: 2

Типов ресурсов: 2

Расход ресурса i на единицу продукции j:

Запас ресурсов:

Стоимость продукции:

Составим целевую функцию и систему ограничений

Приведём задачу к каноническому виду

Перенесём значения в симплекс таблицу:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | 4 | 4 | 1 | 0 | 24 |
|  | 6 | 4 | 0 | 1 | 27 |
|  | -14 | -12 | 0 | 0 | 0 |

Значения переносятся в нижнюю строку с отрицательным знаком. Для вспомогательных переменных, указываетмя ноль.

Определим разрешающий столбец по наименьшему значению в строке F. Затем находим разрешающую строку по наименьшему отношению элементов строк x3 и x4 столбца x1 к значениям столбца x\*.

Разрешающей строкой оказалась строка x4.

Поделим значения данной строки на разрешающий элемент (6):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2/3 | 0 | 1/6 | 4.5 |

Теперь, для составления следующей таблицы, к строкам x3 и F нужно прибавить произведение изменённой разрешающей строки на элементы строк x3 и F из разрешающего столбца, умноженные на -1.

1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| + | -4 | -8/3 | 0 | -4/6 | -18 |
| 4 | 4 | 1 | 0 | 24 |
| x3 | 0 | 4/3 | 1 | -4/6 | 6 |

2)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| + | 14 | 28/3 | 0 | 7/3 | 63 |
| -14 | -12 | 0 | 0 | 0 |
| F | 0 | -8/3 | 0 | 7/3 | 63 |

Перенесём полученные значения в новую симплекс-таблицу:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | 0 | 4/3 | 1 | -4/6 | 6 |
|  | 1 | 2/3 | 0 | 1/6 | 4.5 |
|  | 0 | -8/3 | 0 | 7/3 | 63 |

Строка x4 переименуется в строку x1, в соответствии с разрешающим столбцом, в её позиции подставляются значения, поделённые на разрешающий элемент.

На данном этапе уже получено допустимое решение:

при x1 = 4.5 (x2 = 0) F = 63.

Следующим разрешающим столбцом является столбец x2, а разрешающей строкой является строка x3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | ¾ | -½ | 4.5 |

Вычислим новые значения для следующей симплекс таблицы:

1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| + | 0 | -2/3 | -1/2 | 1/3 | -3 |
| 1 | 2/3 | 0 | 1/6 | 4.5 |
| X1 | 1 | 0 | -1/2 | ½ | 1.5 |

2)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| + | 0 | 8/3 | 2 | -4/3 | 12 |
| 0 | -8/3 | 0 | 7/3 | 63 |
| F | 0 | 0 | 2 | 1 | 75 |

Переносим значения в таблицу:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | 0 | 1 | ¾ | -½ | 4.5 |
|  | 1 | 0 | -1/2 | ½ | 1.5 |
|  | 0 | 0 | 2 | 1 | 75 |

Поскольку в строке F не осталось отрицательных значений, найдено оптимальное рещение.

Значения из последнего столбца приравниваются к элементам, указанным в первом сталбце. Элементы, отсутствующие в первом столбце, приравниваются к нулю. Таким образом, получаем ответ:

Учитывая отсутствие вспомогательных переменных x3 и x4 в целевой функции, их значения не представляют интереса.

Описание структуры программы

Перейдём к описанию структуры порграммы:

В программе реализован класс Task, в поле private которого объявлены переменные количества видов продукции productsAmount и типов ресурсов resourcesAmount. Массив A хранит значения расхода ресурсов на продукцию, массив b содержит значения запасов каждого ресурса, а массив c хранит значения стоимости каждого вида продукции.

Для удобства, программа была разбита на 3 файла:

заголовочный файл с классом – task.h;

файл с реализацией методов класса – task\_methods.cpp;

и основной файл с функцией main – main.cpp.

Рассмотрим заголовочный файл:

#ifndef TASK\_H

#define TASK\_H

**class** Task {

**private**:

**int** productsAmount;

**int** resourcesAmount;

**float** \*\*A;

**float** \*b;

**float** \*c;

**void** deleteData();

**public**:

**int** saveData(**const** **char**\* string);

**int** restoreData(**const** **char**\* string);

**void** getData();

**void** printEquationsSystem();

**void** iteration\_Method();

**void** symplex\_Method(**unsigned** **int** mode);

**void** setRandomData();

**void** setManually();

Task();

~Task();

};

#endif /\*TASK\_H\*/

Продолжим описание пространства private:

Функция deleteData() используется для очистки памяти, выделенной под массивы. Вызывается из деструктора, при удалении экземпляров класса.

Перейдём к public методам:

saveData() – функция используется для сохранения имеющихся входных значений в файл, обозначенный в аргументе. По умолчанию, сохранение производится в файл save.txt, который будет создан (или перезаписан) в каталоге, в котором находится исполняемая программа.

restoreData() – функция используется для восстановления входных данных из файла, путь к которому указан в аргументе функции. По умолчанию, восстановление производится из файла save.txt из директории, в которой находится и запускается программа.

Обе функции, описанные выше, возвращают целочисленные значения. При удачном выполнении, функции возвращают 0.

getData() – функция выводит на экран (в консоль) входные значения.

printEquationsSystem() – функция печатает в терминал целевую функцию и систему условий.

iteration\_Method() – функция, вычисляющая значения переменных и определяющая максимум целевой функции для целочисленных значений, путём перебора всех возможных комбинаций переменных. По окончании работы, выводит полученный результат в терминал.

symplex\_Method(unsigned int mode) – функция, реализующая симплекс-метод. Аргумент функции используется для ограничения количества итераций симплекс-метода. По умолчанию, указывается число 0, в таком случае ограничение будет установлено в 1000. Большее количество итераций нецелесообразно. По окончании работы, выводит полученный результат в терминал.

setRandomData() – функция устанавливает случайные значения в качестве входных данных. Диапазоны возможных значений заранее определены в коде.

setManually() – функция, позволяющая ввести входные значения вручную.

Task() – конструктор класса. Устанавливает базовые значения в приватные переменные, при создании экземпляра класса.

~Task() – деструктор класса. Вызывается при удалении экземпляра класса. Внутри вызывает функцию deleteData() для очистки выделенной под массивы памяти.

Перейдём к описанию файла main.cpp

Внутри данного файла описано взаимодействие с программой через консоль.

Реализованы три оператора выбора switch-case:

Первый запускает одну из следующий функций: setRandomData(), restoreData(), setManually() – для установки входных значений.

Второй предлагает запустить одну из следующих функций: iteration\_Method(), symplex\_Method(0), symplex\_Method(1) – для вычисления целевой функции и переменных x(икс).

Первые два оператора помещены в циклы for с ограничением на количество операций. Это сделано для возможности совершить ошибку. При многократном совершении ошибки программа завершится.

Если при корректном выборе в первом операторе, цикл завершится, то второй цикл не завершается, позволяя выбрать другой метод. Для выхода из второго цикла используется выделенное значение.

Последний оператор предлагает сохранить входные данные. Если выбрать корректное значение, будет вызвана функция saveData(). Если выбрать неправильное значение, данные сохранены не будут и программа завершится.

Код из файла приведён в листинге.

Из файла task\_methods.cpp наибольший интерес представляет реализация функции symplex\_Method(). Ниже приведу его реализацию в формате псевдокода:

*Начало*

*Созать переменную “n”, присвоить значение суммы количества видов продукции(productsAmount), типов ресурсов (resourcesAmount) и единицы.*

*Создать переменную “m”, присвоить значение суммы количества типов ресурсов и единицы.*

*Создать двухмекрный массив “L”, размером n(по горизонтали)\*m(по вертикали).*

*Заполнить массив “L” нулями.*

*Занести единицы в позиции, соответствующие основным переменным, по диагонали, начиная от столбца первой основной переменной.*

*Занести элементы массива “b” в последний столбец массива “L”.*

*Занести элементы массива “c” в последнюю строку массива “L”.*

*Создать массив “Xc” размером “n – 1”.*

*Заполнить с 0 до “productsAmount ” элементы массива Xc значением:*

*(-1).*

*Заполнить элементы массива с “productsAmount” до “resourcesAmount” значениями от 0 до “resourcesAmount”.*

*Выполнять в цикле код ниже, пока счётчик не достигнет 1000 (при значении аргумента функции равного нулю), либо значения, указанного в аргументе:*

*Создать переменную “iLmin” и присвоить значение -1.*

*Проверить в цикле последнюю строку массива L на наличие элементов ниже нуля. Присвоить позицию минимального элемента в массиве переменной iLmin.*

*Проверить значение переменной iLmin – если оно равно -1 – выйти из цикла.*

*Создать переменную “jLmin” и присвоить значение -1.*

*Порверить в цикле, в столбце iLmin массива L отношения последнего элемента каждой строки к элементам разрешающего столбца iLmin. Минимальное значение присвоить переменной jLmin. Если переменная столбца iLmin меньше нуля, а переменная в конце смежной с переменной строки положительная – пропустить это значение.*

*Проверить значение переменной jLmin – если оно равно -1 – выйти из цикла.*

*Создать переменную minL и присвоить ей значение разрешающего элемента L[iLmin][jLmin].*

*Поделить элементы разрешающей строки jLmin на значение разрешающего элемента minL.*

*Создать элемент j и в цикле, пока j меньше m, выполнить код ниже:*

*Создать переменную iL, равную произведению*

*L[iLmin][j] и -1.*

*Создать переменную i и в цикле, пока i меньше n выполнять команду:*

*Прибавить к переменной массива L[i][j] произведение L[i][jLmin] и iL.*

*Присвоить элементу iLmin массива Xc значение jLmin.*

*Присвоить элементу “productsAmount + jLmin” массива Xc значение -1.*

*Выполнять код в цикле, пока i меньше productsAmount, увеличивая i на единицу:*

*Напечатать “X[i] = 0”, если Xc[i] равен -1 и напечатать значение из последнего столбца массива L в строке Xc[i], в обратном случае.*

*Создать переменную F и приравнять к нулю.*

*Создать i и в цикле, пока i меньше productsAmount, выполнять код:*

*Добавить к переменой F произведение c[i] и L[n-1][Xc[i]], если Xc[i] не равен -1.*

*Вывести в консоль значение переменной F.*

*Очистить память, выделенную под массив Xc.*

*Очистить память, выделенную под массив L.*

*Конец.*

Результат работы программы

Сделаю проверку работы программы для трёх сценариев.

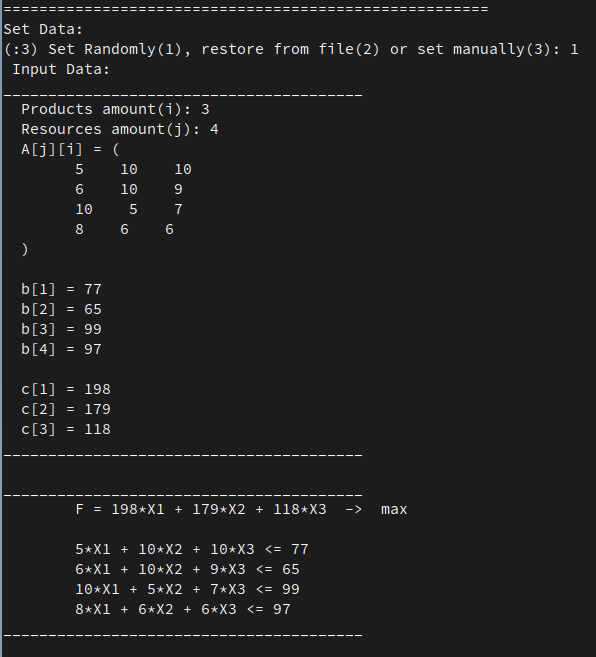
1.

При запуске программы, первым появляется выбор метода заполнения данных.

В первой проверке, выберу пункт случайного заполнения данных. Количество видов продукции и типов ресурсов генерируется случайно в диапазоне от 2 до 5.

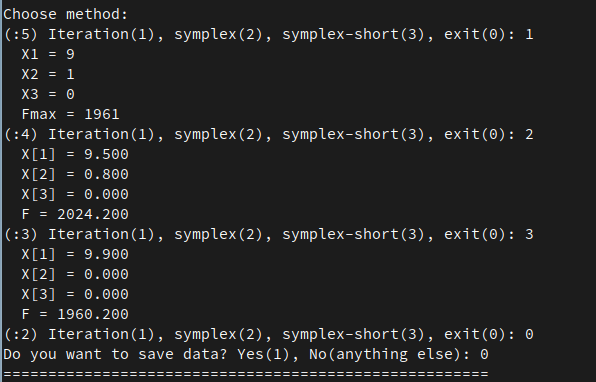
Запас ресурсов генерируется в диапазоне от 50 до 100.

Стоимость продукции генерируется в диапазоне от 100 до 200.



На экран выводятся сгенерированные данные, а также целевая функция и система ограничений.

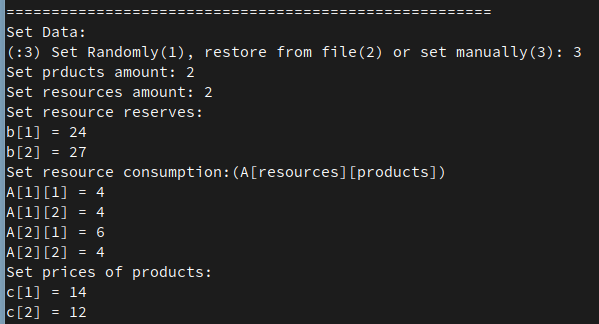
Средующим, программа выводит сообщение о выборе метода получения значения целевой функции. Выберем поочерёдно все и посмотрим на результат.

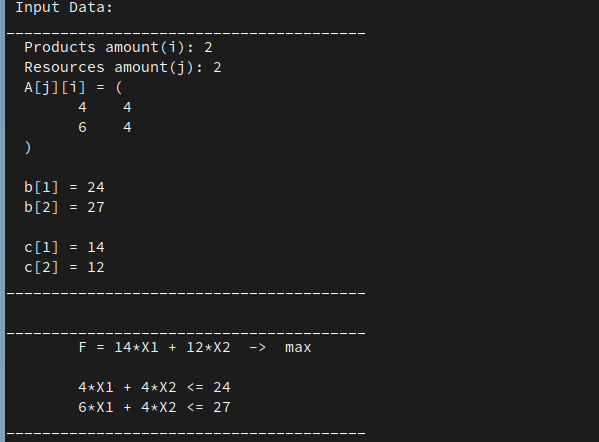


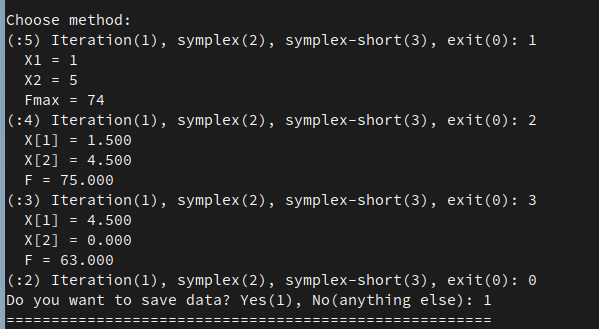
Первым был выбран метод перебора целочисленных значений, вторым – симплекс-метод, а третьим – симплекс-метод, ограниченный одной итерацией. Можно увидеть, что симплекс-метод даёт наибольший результат, однако его первая итерация выдаёт достаточно близкое значение.

2.

Далее, запущу программу повторно и выберу ввод значений вручную. Укажу параметры, приведённые при решении задачи вручную.







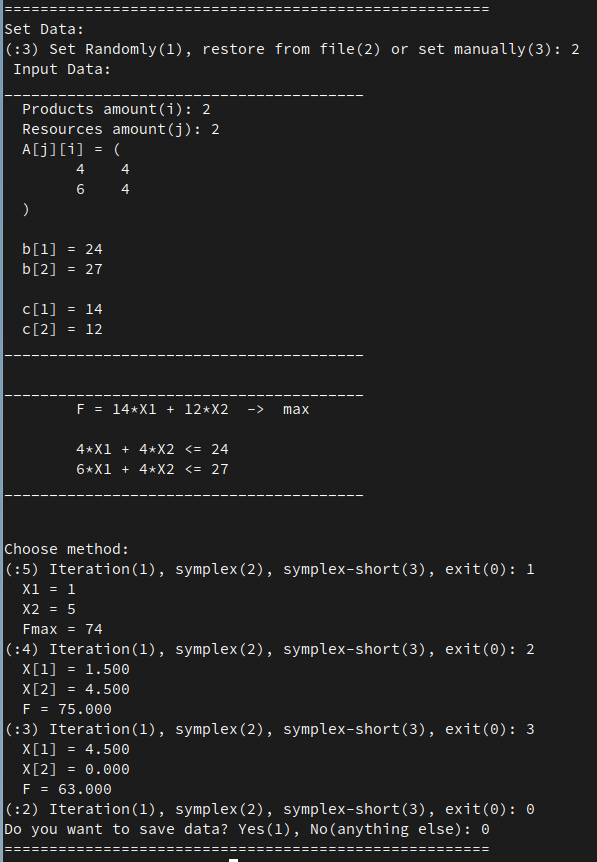
При выборе симплекс-метода, программа выдаёт идентичные значения полученным мною вручную.

Первая итерация симплекс-метода в этот раз привела не самое оптимальное, однако допустимое значение.

На этот раз, выберем опцию сохранения данных.

3.

При третьем запуске, выберем восстановление входных данных из сохранённых ранее и проверим результат.



Данные были успешно восстановлены, результаты идентичны.

Вывод

В ходе этой работы, на языке C++ была реализована программа нахождения максимума целевой функции задачи линейного программирования с соблюдением указанных критериев:

Программа работает в диалоговом режиме;

В системе реализованы следующие блоки:

* формирование исходных данных случайным образом или вручную;
* вывод исходных данных в консоль;
* вывод результатов решения задачи;
* сохранение исходных данных в текстовом файле в каталоге, где запускается программа;
* восстановление исходных данных из файла сохранения.

Вывод данных осуществляется в читаемом виде, отдельно формируются и выводятся на экран система условий и целевая функция;

Блок решений включает три метода решения задачи: метод полного перебора целочисленных значений, симплекс-метод и подобие эвристического решения – симплекс-метод с одной итерацией;

Сохранение и восстановление исходных данных осуществляется в текстовый файл.

Если сравнить результаты работы обычного симплекс-метода и полного перебора целочисленных значений, можно заметить, что разница бывает значительной. Поэтому, симплекс-метод может давать не оптимальные результаты, когда продукцию и ресурсы не представляется возможным делить на части.

В качестве предложения по совершенствованию системы, можно включить метод Гомори для нахождения целочисленных результатов решения – в том случае, когда результат с дробью не является приемлемым. Он представляет собой изменённый симплекс-метод.

Листинг

task.h

#ifndef TASK\_H

#define TASK\_H

**class** Task {

**private**:

**int** productsAmount;

**int** resourcesAmount;

**float** \*\*A;

**float** \*b;

**float** \*c;

**void** deleteData();

**public**:

**int** saveData(**const** **char**\* string);

**int** restoreData(**const** **char**\* string);

**void** getData();

**void** printEquationsSystem();

**void** iteration\_Method();

**void** symplex\_Method(**unsigned** **int** mode);

**void** setRandomData();

**void** setManually();

Task();

~Task();

};

#endif /\*TASK\_H\*/

main.cpp

#include <iostream>

#include "task.h"

**int** main (**int** argc, **char** \* argv[]) {

std::cout << "======================================================\n";

Task \*task = **new** Task();

**int** a = 0;

std::cout << "Set Data:" << std::endl;

**for** (**int** i = 3; i >= 0; i--) {

**int** flag = 0;

**if** (i == 0) {

std::cout << "Program shutdown." << std::endl;

**delete** task;

**return** 0;

}

std::cout << "(:" << i << ") Set Randomly(1), restore from file(2) or set manually(3): ";

std::cin >> a;

**switch**(a) {

**case** 1: {

task->setRandomData();

**break**;

}

**case** 2: {

**if** (task->restoreData("save.txt") != 0) {

std::cout << "Unable to restore. Select another option." << std::endl;

flag = 1;

}

**break**;

}

**case** 3: {

task->setManually();

**break**;

}

**default**: {

std::cout << "Try again" << std::endl;

flag = 1;

**break**;

}

}

**if** (flag == 0) **break**;

}

task->getData();

task->printEquationsSystem();

std::cout << "\nChoose method:\n";

**for** (**int** i = 5; i >= 0; i--) {

**int** flag = 0;

**if** (i == 0) {

std::cout << "Program shutdown." << std::endl;

**delete** task;

**return** 0;

}

std::cout << "(:" << i << ") Iteration(1), symplex(2), symplex-short(3), exit(0): ";

std::cin >> a;

**switch**(a) {

**case** 0: {

flag = 1;

**break**;

}

**case** 1: {

task->iteration\_Method();

**break**;

}

**case** 2: {

task->symplex\_Method(0);

**break**;

}

**case** 3: {

task->symplex\_Method(1);

**break**;

}

**default**: {

**break**;

}

}

**if** (flag == 1) **break**;

}

std::cout << "Do you want to save data? Yes(1), No(anything else): ";

std::cin >> a;

**switch** (a) {

**case** 1: {

**if**(task->saveData("save.txt") != 0) {

std::cout << "Unable to save data" << std::endl;

}

**break**;

}

**case** 2: {

**break**;

}

**default**: {

**break**;

}

}

**delete** task;

std::cout << "======================================================\n";

**return** 0;

}

task\_methods.cpp

#include <iostream>

#include <cstdlib>

#include <iomanip>

#include <typeinfo>

#include <cmath>

#include <fstream>

#include "task.h"

// PRIVATE:

**void** Task::deleteData() {

**for** (**int** i = 0; i < productsAmount; i++) {

**delete** [] A[i];

}

**delete** [] A;

**delete** [] b;

**delete** [] c;

}

// PUBLIC:

**int** Task::saveData(**const** **char**\* string) {

std::ofstream file (string);

**if** (!file.is\_open()) {

std::cerr << "File is not opened" << std::endl;

**return** 1;

}

file << productsAmount << ' ' << resourcesAmount << std::endl;

**for** ( **int** j = 0; j < resourcesAmount; j++) {

**for** (**int** i = 0; i < productsAmount; i++) {

file << A[i][j];

**if** (i < productsAmount - 1) file << ' ';

}

file << std::endl;

}

**for** (**int** j = 0; j < resourcesAmount; j++) {

file << b[j];

**if** (j < resourcesAmount - 1) file << ' ';

}

file << std::endl;

**for** (**int** i = 0; i < productsAmount; i++) {

file << c[i];

**if** (i < productsAmount - 1) file << ' ';

}

file << std::endl;

file.close();

**return** 0;

}

**int** Task::restoreData(**const** **char**\* string) {

std::ifstream file;

file.open(string);

**if** (!file.is\_open()) {

std::cerr << "Error" << std::endl;

**return** 1;

}

file >> productsAmount >> resourcesAmount;

A = **new** **float**\* [productsAmount];

**for** (**int** i = 0; i < productsAmount; i++) {

A[i] = **new** **float**[resourcesAmount];

}

**for** (**int** j = 0; j < resourcesAmount; j++) {

**for** (**int** i = 0; i < productsAmount; i++) {

file >> A[i][j];

}

}

b = **new** **float**[resourcesAmount];

**for** (**int** j = 0; j < resourcesAmount; j++) {

file >> b[j];

}

c = **new** **float**[productsAmount];

**for** (**int** i = 0; i < productsAmount; i++) {

file >> c[i];

}

file.close();

**return** 0;

}

**void** Task::getData() {

std::cout << " Input Data: " << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << " Products amount(i): " << productsAmount << std::endl;

std::cout << " Resources amount(j): " << resourcesAmount << std::endl;

std::cout << " A[j][i] = (" << std::endl;

**for** (**int** j = 0; j < resourcesAmount; j++) {

std::cout << "\t";

**for** (**int** i = 0; i < productsAmount; i++) {

std::cout << A[i][j] << " ";

}

std::cout << std::endl;

}

std::cout << " )" << std::endl << std::endl;

**for** (**int** j = 0; j < resourcesAmount; j++) {

std::cout << " b[" << j+1 << "] = " << b[j] << std::endl;

}

std::cout << std::endl;

**for** (**int** i = 0; i < productsAmount; i++) {

std::cout << " c[" << i+1 << "] = " << c[i] << std::endl;

}

//std::cout << typeid(c[0]).name() << endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << std::endl;

}

**void** Task::printEquationsSystem() {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "\tF = ";

**for** (**int** i = 0; i < productsAmount; i++) {

std::cout << c[i] << "\*X" << i+1 << " ";

**if** (i < productsAmount - 1) std::cout << "+ ";

}

std::cout << " -> max" << std::endl << std::endl;

**for** (**int** j = 0; j < resourcesAmount; j++) {

std::cout << "\t";

**for** (**int** i = 0; i < productsAmount; i++) {

std::cout << A[i][j] << "\*X" << i+1 << " ";

**if** (i < productsAmount - 1) std::cout << "+ ";

}

std::cout << "<= " << b[j] << std::endl;

}

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << std::endl;

}

**void** Task::iteration\_Method() {

// вычисляем максимально возможное количество продкции,

// которое можно получить из доступных ресурсов

// (для каждого типа отдельно, независимо от других)

**int** \*max\_copies = **new** **int**[productsAmount];

**for** (**int** i = 0; i < productsAmount; i++) {

**int** min = b[0]/A[i][0];

**for** (**int** j = 1; j < resourcesAmount; j++) {

**if** (b[j]/A[i][j] < min) min = b[j]/A[i][j];

}

max\_copies[i] = min;

}

// создаём матрицу для всех возможных и невозможных комбинаций

// производства продукции

**int** size = max\_copies[0] + 1;

**for** (**int** i = 1; i < productsAmount; i++) {

size \*= (max\_copies[i] + 1);

}

**int** \*\*Mattrix = **new** **int**\* [productsAmount];

**for** (**int** i = 0; i < productsAmount; i ++) {

Mattrix[i] = **new** **int**[size];

}

// заполняем матрицу

**for** (**int** i = 0; i < productsAmount; i++) {

**int** counter = 1;

**for** (**int** j = i + 1; j < productsAmount; j++) {

counter \*= (max\_copies[j] + 1);

}

**int** max\_c = max\_copies[i];

**int** ctr = counter;

**for** (**int** s = 0; s < size; s++) {

**if** (ctr < 1) {

ctr = counter;

max\_c--;

}

**if** (max\_c < 0) {

max\_c = max\_copies[i];

}

Mattrix[i][s] = max\_c;

ctr--;

}

}

// создаём массив значений выгоды

// для каждого набора продукции

**int** \*F = **new** **int**[size];

// заполняем массив

**for** (**int** s = 0; s < size; s++) {

F[s] = 0;

**for** (**int** j = 0; j < resourcesAmount; j++) {

**int** Z = 0;

**for** (**int** i = 0; i < productsAmount; i++) {

Z += Mattrix[i][s] \* A[i][j];

}

**if** (Z > b[j]) {

**break**;

}

**if** (j == resourcesAmount - 1) {

**for** (**int** i = 0; i < productsAmount; i++) {

F[s] += c[i] \* Mattrix[i][s];

}

}

}

}

// находим максимальное значение

**int** max\_index = -1;

**for** (**int** s = 0; s < size; s++) {

**if** (F[s] == 0) **continue**;

**if** (F[s] > 0 && max\_index == -1) {

max\_index = s;

**continue**;

}

**if** (F[s] > F[max\_index]) max\_index = s;

}

// Выводим результаты

**for** (**int** i = 0; i < productsAmount; i++) {

std::cout << " X" << i + 1 << " = " << Mattrix[i][max\_index] << std::endl;

}

std::cout << " Fmax = " << F[max\_index] << std::endl;

**delete** [] max\_copies;

**for** (**int** i = 0; i < productsAmount; i++) {

**delete** [] Mattrix[i];

}

**delete** [] Mattrix;

**delete** [] F;

}

**void** Task::symplex\_Method(**unsigned** **int** mode) {

**int** n = productsAmount + resourcesAmount + 1;

**int** m = resourcesAmount + 1;

**float** \*\*L = **new** **float**\* [n];

**for** (**int** i = 0; i < n; i++) {

L[i] = **new** **float**[m];

}

// заполнение массива L нулями

**for** (**int** i = 0; i < n; i++) {

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

L[i][j] = 0.f;

}

}

// заносим единицы в позиции основных переменных

**for** (**int** j = 0; j < resourcesAmount; j++) {

L[j + productsAmount][j] = 1.f;

}

// переносим элементы массива A[][] в массив L

**for** (**int** i = 0; i < productsAmount; i++) {

**for** (**int** j = 0; j < resourcesAmount; j++) {

L[i][j] = A[i][j];

}

}

// заносим элементы массива b[] в массив L

**for** (**int** j = 0; j < resourcesAmount; j++) {

L[productsAmount + resourcesAmount][j] = b[j];

}

// заносим элементы массива c[] в массив L

**for** (**int** i = 0; i < productsAmount; i++) {

L[i][resourcesAmount] = c[i] \* (-1.f);

}

// создаём вспомогательный массив для получения

// в конце исполнения функции значений X и функции F

**int** \*Xc = **new** **int**[n - 1];

**for** (**int** i = 0; i < productsAmount; i++) {

Xc[i] = -1;

}

**for** (**int** i = productsAmount; i < n - 1; i++) {

Xc[i] = i - productsAmount;

}

// MAIN BLOCK ////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////

**for** (**int** counter = 0; counter < ((mode == 0) ? 1000 : mode); counter++) {

// 1

**int** iLmin = -1;

**for** (**int** i = 0; i < n - 1; i++) {

**if** ((L[i][m-1] < -0.001) && (iLmin == -1)) {

iLmin = i;

**continue**;

}

**if** ((L[i][m-1] < -0.001) && (iLmin != -1)) {

**if** (L[i][m-1] < L[iLmin][m-1]) iLmin = i;

}

}

**if** (iLmin == -1) {

**break**;

}

// 2

**int** jLmin = -1;

**for** (**int** j = 0; j < m - 1; j++) {

**if** (L[iLmin][j] == 0 ||

((L[iLmin][j] > -0.001) && (L[iLmin][j] < 0.001))) **continue**;

**if** ((L[iLmin][j] < 0) && (L[n-1][j] < 0)) **continue**;

**if** (jLmin == -1) {

jLmin = j;

**continue**;

}

**if** (L[n-1][j]/L[iLmin][j] < L[n-1][jLmin]/L[iLmin][jLmin]) jLmin = j;

}

**if** (jLmin == -1) **break**;

// 3

**float** minL = L[iLmin][jLmin];

**for** (**int** i = 0; i < n; i++) {

L[i][jLmin] = L[i][jLmin]/minL;

}

// 4

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

**if** (j == jLmin) **continue**;

**float** iL = L[iLmin][j] \* (-1.f);

**for** (**int** i = 0; i < n; i++) {

L[i][j] = L[i][j] + L[i][jLmin] \* iL;

}

}

Xc[iLmin] = jLmin;

Xc[productsAmount + jLmin] = -1;

}

//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////// MAIN BLOCK //

// Выводим полученные значения X и функции F

**for** (**int** i = 0; i < productsAmount; i++) {

std::cout << " X[" << i + 1 << "] = ";

std::cout << std::fixed << std::setprecision(3) << ( (Xc[i] == -1) ? 0 : L[n-1][Xc[i]] );

std::cout << std::endl;

}

**float** F = 0;

**for** (**int** i = 0; i < productsAmount; i++) {

F += c[i] \* ( (Xc[i] == -1) ? 0 : L[n-1][Xc[i]] );

}

std::cout << std::fixed << std::setprecision(3) << " F = " << F << std::endl;

// Очищаем память, выделенную под массивы

**delete** [] Xc;

**for** (**int** i = 0; i < n; i++) {

**delete** [] L[i];

}

**delete** [] L;

}

**void** Task::setRandomData() {

srand(time(0));

**int** min = 2;

**int** max = 5;

productsAmount = min + (rand() % (max - min + 1));

resourcesAmount = min + (rand() % (max - min + 1));

A = **new** **float**\* [productsAmount];

**for** (**int** i = 0; i < productsAmount; i++) {

A[i] = **new** **float** [resourcesAmount];

}

min = 5;

max = 10;

**for** (**int** i = 0; i < productsAmount; i++) {

**for** (**int** j = 0; j < resourcesAmount; j++) {

A[i][j] = **static\_cast**<**float**>(min + (rand() % (max - min + 1)));

}

}

min = 50;

max = 100;

b = **new** **float**[resourcesAmount];

**for** (**int** j = 0; j < resourcesAmount; j++) {

b[j] = **static\_cast**<**float**>(min + (rand() % (max - min + 1)));

}

min = 100;

max = 200;

c = **new** **float**[productsAmount];

**for** (**int** i = 0; i < productsAmount; i++) {

c[i] = **static\_cast**<**float**>(min + (rand() % (max - min + 1)));

}

}

**void** Task::setManually() {

std::cout << "Set prducts amount: ";

std::cin >> productsAmount;

std::cout << "Set resources amount: ";

std::cin >> resourcesAmount;

b = **new** **float**[resourcesAmount];

std::cout << "Set resource reserves:" << std::endl;

**for** (**int** j = 0; j < resourcesAmount; j++) {

std::cout << "b[" << j + 1 << "] = ";

std::cin >> b[j];

}

A = **new** **float**\* [productsAmount];

**for** (**int** i = 0; i < productsAmount; i++) {

A[i] = **new** **float**[resourcesAmount];

}

std::cout << "Set resource consumption:(A[resources][products])" << std::endl;

**for** (**int** j = 0; j < resourcesAmount; j++) {

**for** (**int** i = 0; i < productsAmount; i++) {

std::cout << "A[" << j + 1 << "][" << i + 1 << "] = ";

std::cin >> A[i][j];

}

}

c = **new** **float**[productsAmount];

std::cout << "Set prices of products:" << std::endl;

**for** (**int** i = 0; i < productsAmount; i++) {

std::cout << "c[" << i + 1 << "] = ";

std::cin >> c[i];

}

}

Task::Task() {

productsAmount = 0;

resourcesAmount = 0;

A = **NULL**;

b = **NULL**;

c = **NULL**;

}

Task::~Task() {

deleteData();

}