Лабораторная работа №8 по курсу дискретного анализа: Откорм бычков

Выполнил студент МАИ группы М8О-201Б *Ефимов Александр.*

Постановка задачи

Разработать жадный алгоритм решения задачи, определяемой своим вариантом. Доказать его корректность, оценить скорость и объём затрачиваемой оперативной памяти.

Реализовать программу на языке C или C++, соответсвующую построенному алгоритму. Формат входных и выходных данных описан в варианте задания.

Бычкам дают пищевые добавки, чтобы ускорить их рост. Каждая добавка содержит некоторые из N действующих веществ. Соотношения количеств веществ в добавках могут отличаться. Воздействие добавки определяется как $c_1a_1 + c_2a_2 + \ldots + c_Na_N$, где a_i количество i-го вещества в добавке, c_i — неизвестный коэффициент, связанный с веществом и не зависящий от добавки. Чтобы найти неизвестные коэффициенты c_i, Биолог может измерить воздействие любой добавки, использовав один её мешок. Известна цена мешка каждой из M ($M \geq N$) различных добавок. Нужно помочь Биологу подобрать самый дешевый наобор добавок, позволяющий найти коэффициенты c_i. Возможно, соотношения веществ в добавках таковы, что определить коэффициенты нельзя.

Метод решения

Задача обобщается в простое исключение методом Гаусса, причем наиболее оптимальные мешков можно найти жадным способом, заранее отсортировав все уравнения мешков по их цене. Так как здесь не требуется найти коэффиценты, для решения необходимо и достаточно ровно N линейно независимых уравнений, начиная поиск с уравнений с наименьшей ценой. Остальные $M > N$ уравнений будут линейно зависимы по определению. Если же линейно независимых уравнений меньше N, то в системе присутствуют независимые перменные, значения которых нельзя определить однозначно, соответственно их цену вывести невозможно.

Установление корректности можно провести методом индукции. В начале выбранное уравнение может быть либо линейно независимым (т.к. еще нет строк для проверки зависимости), либо пустая. Все пустые в начале строки можно пропускать до нахождения непустной строки. Так как все строки отсортированы, первая найденная строка обязательно минмальная.

Каждое новое уравнение может быть либо минимальным линейно независимым, либо линейно зависимым (которое также пропускается, т.к. для решения не подходит).

Появление меньшего линейно независимого уравнения в процессе решения невозможно, так как к моменту, когда уравнение было обнаружено линейно зависимым, обязательно было обнаружено(-на) меньшее уравнение (комбинация уравнений), через которое проверялась зависимость.

Описание программы

1. *equation.h(cpp)*
   Реализация класса уравнения с методом, проверяющим на пустоту, и операторами, позволяющими работать над уравнениями как над строками в матрице;

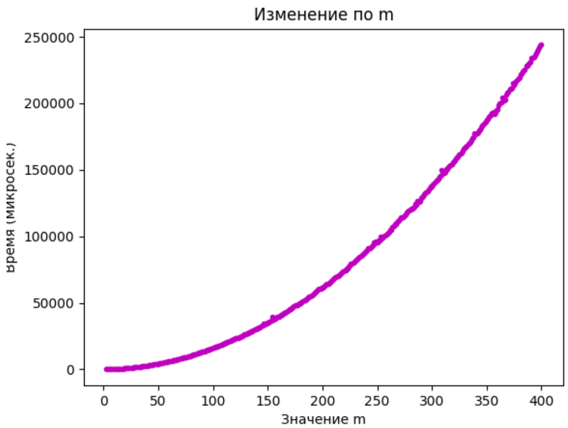
2. *matrix.h(cpp)*
   Содержит в себе реализацию матрицы СЛАУ вместе с методом, выполняющим гауссово исключение. Если система имеет решение, то метод вернет пару из новой системы и логической истинны. Если система не решаема, то она вместе с новой системой вернет логическую ложь;

3. *main.cpp*
   Начало потока программы; принимает на вход значения СЛАУ.

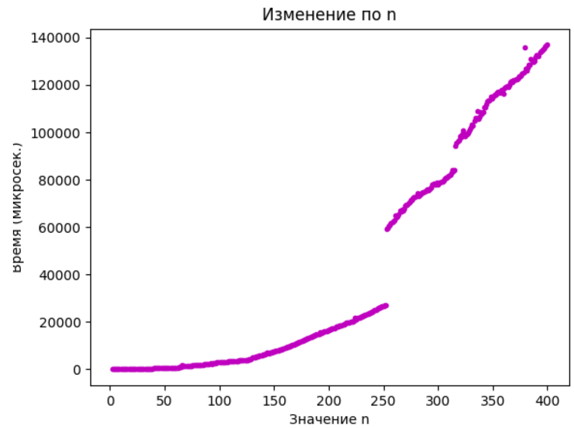
Дневник отладки

Номер	Ошибка	Обнаруженная причина	Исправление
1	Ошибка компиляции	Makefile чистит с проверкой существования файла	Избавится от проверки с помощью флага '-f'

Тест производительности



(a) Постоянная $N = 1000$.



(b) Постоянная $M = 1000$.

Анализируя алгоритм в функции *EliminateGauss()*, можно сделать предположение, что его сложность равна $O(MN^2)$ (т.е. во время прохождения матрицы и M уравнений, из каждого из них вычитается в худшем случае $N - 1$ найденных ЛНЗ уравнений, причем вычитание происходит между N константами).

Во время работы алгоритм использует дополнительно $O(N^2)$ памяти в худшем случае для хранения полученной системы, а также массива индексов, с которых начинаются ненулевые коэффиценты в нем.

На рисунках показано время, затраченное для поиска решенеия для теста, в котором первые $N - 1$ строк линейно независимые, после чего следует $M - N$ строк линейно зависимы от первых строк, и последняя строка также линейно независима.

По неизвестным причинам, количество вызовов операторов уравнений изменяется примерно в два раза между $N = 252$ и $N = 253$ на графике (b) (причем остальные вызовы изменяются незначительно, в том числе вызовы *malloc*).

Выводы

Некоторые виды задач с огромной на первой взгляд сложностью могут приведены к простому решению из-за того, что в любой появляющейся подзадаче можно выбрать наиболее оптимальное решение без пересмотра всех остальных.

На этой идее и основаны жадные алгоритмы (несмотря на слова "алгоритмы в названии **это метод**) – в любой состоянии задачи делается наиболее оптимальный на данных момент выбор, который в итоге приводит к оптимальному решению для всей задачи.

Жадные алгоритмы очень схожи с динамическим программирование, но, в то время как в динамическом программировании решение делается после просмотра всего или какой-то части состояния системы, жадные алгоритмы делают только один наилучший выбор. Значит, динамическое программирование можно использовать вместо жадных алгоритмов, но они будут значительнее медленнее их. Но наоборот сделать нельзя, так как там, где используется по умолчанию динамическое программирование, требуется просмотр всей системы, чего жадные алгоритмы не делают.

Примеры использования:

- Алгоритм Дейкстры – в каждый момент времени примыкает к достигнутым вершинам на графе наименьшую грань (оптимальный выбор) пока конечная вершина не будет достигнута.
- Минимальное остовное дерево – множество граней, минимально соединяющих все вершины на графе. Строится за счет соединения множеств уже соединенных вершин (могут состоять из одной вершины) гранями с наименьшим весом (оптимальный выбор).
- Планирование заданий без приоритетов в операционных системах.