Московский авиационный институт
(Национальный Исследовательский Институт)

Кафедра вычислительной математики и программирования.

Курсовая работа
по курсу «практикум на ЭВМ»
Задание №7:
Разреженные матрицы

Студент:	Суханов Е. А.
Группа:	М80-106Б
Преподаватель:	Дубинин А. В.
Оценка:	
Дата:	

Москва
2020

Оглавление

Введение

В некоторых задачах приходится использовать матрицы больших размеров. Зачастую они являются разреженными. Хранить и обрабатывать такие матрицы в виде простого двумерного массива не получится (уже при размерах 10^4 x 10^4 массив типа double будет занимать ≈762 МБ). Поэтому для разреженных матриц были придуманы более оптимальные форматы хранения.

Цель. Изучить некоторые форматы хранения разреженных матриц, их плюсы и недостатки. Опробовать заданный способ хранения на практике.

Задание. Составить программу на языке Си с функциями для обработки прямоугольных разреженных матриц с элементами вещественного (double) типа. Программа должна иметь следующий функционал:

- Ввод матрицы различного размера. Матрица вводится в виде двумерного массива;
- Вывод матрицы в «человеческом» и «внутреннем» форматах;
- Выполнение заданного преобразования;

Заданный способ хранения: Три вектора.

Заданное преобразование: Найти элемент матрицы, ближайший к заданному значению. Разделить на него элементы строки и столбца, на пересечении которых он расположен. Если таких элементов несколько, обработать все.

Теория

Определение 1. Разреженной называют матрицу, имеющую малый процент ненулевых элементов.
Рассмотрим некоторые форматы хранения разреженных матриц.

Цепочка ненулевых элементов в векторе с строчным индексированием
Каждому ненулевому элементу a_{ij} соответствует в памяти запись. Запись состоит из трех полей (j, a, p), где j – номер столбца элемента, a – само значение элемента, p – адрес следующего ненулевого элемента i-й сроки. Значение p равно нулю для последнего элемента сроки. Кроме списка записей A в памяти хранится массив M размера n, содержащий адреса записей первых ненулевых элементов соответствующих строчек. Если у i-й строки нет не нулевых элементов, то M[i] = 0. Данный формат изображен на рис. 1

Рис. 1

Замечание 1. Если массив не будет изменяться, то вместо списка можно использовать простой массив, а адреса заменить на индексы (тогда упрощается формат хранения, но вставка нового не нулевого становится не эффективной).
Замечание 2. Строчки и столбцы можно поменять местами (соответствующие преимущества и недостатки тоже изменятся). Данное замечание применимо ко всем рассматриваемым форматам.

Эффективность по памяти:
Пусть n – кол-во строчек; m – кол-во столбцов; t – кол-во не нулевых элементов;
Тогда для хранения матрицы в таком формате потребуется $\boldsymbol{n + 3t}$ ячеек памяти (для простоты будем считать, что адрес, значение и номер столбца занимают одинаковое количество памяти).

Преимущества:

1. Главным преимуществом этого способа хранения является быстрая вставка/удаление не нулевого элемента (Так как A – односвязный список);
2. Можно за **O(1)** поменять местами строки (swap(M[i_1], M[i_2]));

Недостатки:

1. Для хранения требуется больше места, чем для других форматов.;
2. Медленный доступ к элементам определенного столбца;

Один вектор

Каждому ненулевому элементу соответствует запись (*j*, *a*), где *j* – номер столбца элемента, *a* — его значение. Начало новой (одновременно конец предыдущей) строки обозначается следующим способом: (0, *i*), то есть *j* = 0; *a* равен номеру новой строки. Конец матрицы обозначается записью (0, 0). Данный формат изображен на рис. 2.

Рис. 2

Замечание 1. Для обозначения начала новой строки я предлагаю использовать немного другую запись: (*i*, 0). Так как тип поля a может не точно отображать номер строки (или добавлять дополнительные сложности в обработке матрицы).

Эффективность по памяти:

В итоге массив будет занимать ***2(n + t + 1)*** ячеек памяти.

Преимущества:

1. Более эффективен по памяти, чем первый способ хранения;
2. Достаточно просто приписать новую строчку в конец массива;

Недостатки:

1. Вставка/удаление ненулевого элемента за **O(*n*);**
2. Доступ к определенному элементу за **O(*n*);**

Два вектора

Каждому элементу a_{ij} ставится в соответствие целое число λ_{ij} :

$$\lambda_{ij} = (i-1)n + (j-1),\ a_{ij} \neq 0$$

Другими словами λ_{ij} – номер элемента a_{ij} в его матрице, отображенной на одномерный массив. Хранение ненулевых элементов обеспечивается двумя массивами LB и YE. При этом LB[a] ($0 \le a \le t$) хранит в себе значение λ_{ij}, а YE[a] – a_{ij} . Конец матрицы обозначается значением -1 в LB. Данный формат изображен на рис. 3.

Рис. 3

Номер строки и столбца для элемента a соответственно равны $i = \lambda_{ij}$ div n и $j = \lambda_{ij}$ mod m.

Замечание 1. Так как в задании используются векторы, то можно не обозначать конец матрицы.

Эффективность по памяти:

Так как длина LB = t+1, а – YE = t, то в итоге получается $\mathbf{2t + 1}$ ячеек памяти.

Преимущества:

1. Самый эффективный формат по памяти из рассматриваемых;
2. Если добавить условие монотонности массива LB (то есть для $a_1 < a_2$ => LB[a_1] < LB[a_2]). То искать определенный элемент можно за **O(log*n*)**, используя бинарный поиск;

Недостатки:

1. Вставка/Удаление за **O(*n*)**;

Три вектора

Хранение элементов обеспечивается с помощью 3-х массивов. CIP – хранит в себе индексы (для PI и YE) начала *i*-й строки. PI и YE хранят в себе номер столбца и значение соответственно. Данный формат изображен на рис. 4.

Рис. 4

Замечание 1. Конец не нулевых элементов в PI можно опустить, если используется вектор. (При этом немного усложнится обработка)

Эффективность по памяти:

Размер CIP равен n, размер PI и YE – t+1 и t соответственно. В итоге имеем $\boldsymbol{n + 2t + 1}$ ячеек памяти.

Преимущества:

1. За $\mathbf{O(\log n)}$ можно найти определенный элемент;
2. Просто добавить новую строку в конец;

Недостатки:

1. Сложно найти элементы определенного столбца ($\mathbf{O(t+n)}$ либо $\mathbf{O(n\log(t/n))}$);
2. Вставка/Удаление элемента за $\mathbf{O(n)}$ (при этом надо менять значения во всех массивах);

Описание программы

Разобьём задание на подзадачи:

1. Формат хранения матрицы. Для формата «три вектора» нужен вектор типа int (CIP и PY) и типа double (YE). Матрица должна обладать следующим функционалом:
    - Инцилизация/Деинцилизация
    - Чтение матрицы из потока
    - Вывод матрицы в поток. В «человеческом» и «внутреннем» форматах
    - Возможность получить и задать определенный элемент
    - Передвигаться по матрице. Получить следующий/предыдущий ненулевой элемент в этой строке/строке
2. Шаблон вектора на макросах.
    - Инцилизация/Деинцилизация
    - Получить/Изменить элемент
    - Получить/Изменить размер
    - Проверка на пустоту
    - Добавление элемента в конец
3. Преобразование.

Реализация программы:

Для проверки работоспособности некоторых компонентов программы были написаны соответствующие тесты. Тестирование отдельных компонентов выполнялось в виде отдельной программы, код которой находится в файлах test.c, а так же в директории test.

Для проверки работоспособности программы в целом был написан скрипт на bash, который запускал программу на тестовых примерах из test_files/main/ и

проверял её результат работы с заведомо правильным ответом на соответствующий тест.

«Шаблонный» вектор. Структура вектора состоит из 3-х полей:

- buf – указатель на выделенную память, где хранятся элементы вектора.
- cap – настоящий размер buf
- size – размер используемой части buf

При увеличении размера вектора выделенная память изменяется только в случае, если size сравняется с cap. В противном случае будет произведено выделение дополнительной памяти. При этом cap увеличится в 1.5 раза.

При уменьшении размера вектора выделенная память изменяется только в случае, если cap стал достаточно большим по сравнению с size. Cap уменьшается, если size меньше его в 4/9 раза. При этом cap становится равен 1.5 * size.

Такая оптимизации позволяет приблизить сложность небольших изменений размера вектора к **O(1)**.

Матрица. Исходные файлы матрицы находятся в файлах matrix.c и matrix.h. Структуру было добавлено поле columns, которое хранит в себе кол-во столбцов матрицы, так как в оригинальной реализации не было эффективного способа узнать кол-во столбцов. Итоговая структура:

- Вектор CIP (в исходном коде имеет имя a);
- Вектор PI (в исходном коде имеет имя b);
- Вектор YE (в исходном коде имеет имя c);
- Число columns;

Чтение матрицы из потока. Реализовано в функции m_read(matrix* m, FILE* file). Ввод осуществляется в следующем формате:

- Первая строчка содержит два целых числа n и m – размеры матрицы;
- Следующие n строчек содержут m вещественных чисел — значения элементов a_{ij} ;

Вывод матрицы в поток. Реализован в функции m_print(matrix*m, FILE* file, matrix_io_mode mode), где mode – формат вывода. Mode может принимать два значения HUMAN и INTERNAL – «человеческое» и «внутреннее» представления соответственно. «Человеческий» формат аналогичен формату ввода, а «внутренний» формат выводит три вектора CIP; PI и YE.

Чтение и вывод матрицы работают за **O(*n***m*)**.

Возможность получить и задать определенный элемент реализована в двух видах — непосредственно функции m_get и m_set, могут переменятся для единичного применения. Если требуется изменить большее кол-во элементов, то лучше использовать итераторы. Так как для поиска столбца в строчке использован бинарный поиск, то эти функции работают за **O(log*t*)**.

Навигация по матрице. Для решения этой задачи были разработаны итераторы. Которые позволяют эффективно получить следующий/предыдущий не нулевой элемент в этой строке/столбце. Стоит обратить внимание, что перемещение итератора по строке может перевести его на другую строку, а перемещение итератора по столбцу – нет. Функции mi_next_col и mi_prev_c в худшем случае работают **O(*n*)**. Функции mi_next_row и mi_prev_row работают за **O(*n*log*t*)**

Так как массив PI на полуинтервале [CIP[*i*], CIP[*i*+1]) является монотонно возрастающим, используется бинарный поиск для нахождения определенного столбца в строке.

Преобразование. Алгоритм преобразования:

1. Поиск подходящих элементов. Состоит в линейном прохождении всех ненулевых элементов матрицы. И добавлении всех итераторов, указывающих на одинаковые элементы в вектор. При этом, если был найден более близкий элемент, то вектор очищается. И в него добавляется новый более близкий элемент.
   Итоговая сложность составляет **O(t)**;
2. Для каждого элемента вектора найденных элементов выполняем функцию деления столбца и строчки, в которых находится этот элемент, на этот же элемент. Сложность деления строчки состовляет **O(*t*)**. Сложность деления столбца состовляет **O(*n*log*t*)**, так как в каждой строчке нужно найти столбец.

Итоговая сложность составляет $\mathbf{O(k(n\log t + t))}$, где k – кол-во одинаковых элементов;

Имеем сложность $\mathbf{O(kn\log t + kt)}$.

Структура проекта.
Include – здесь находится шаблонный вектор и модуль логирования. Модуль логирования упрощает логирование ошибок и информации для отладки, а так же тестирование.

Src – Директория с исходным кодом.

- Vector – реализация векторов различных типов;
- Test – исходный код тестов для отдельных компонентов;

Test_files – Текстовые файлы для тестирования самой программы и некоторые файлы для тестирования отдельных компонентов

Полное тестирование программы осуществляется с помощью скрипта testing_program.sh

Сборка проекта происходит с помощью простого Makefile.

Заключение

Форматы хранения разреженных матриц позволяют экономить память. Существует много различных вариантов хранения разреженных матриц, каждый из которых имеет свои достоинства и недостатки. Например, большинство приведенных выше форматов хранения имеют медленную скорость поиска столбцов, тогда как строчки ищутся быстро. А Для некоторых форматов хранения матрицы вставка нового ненулевого элемента является не тривиальной задачей. А если матрица на самом деле является не разреженной, то данные способы теряют свою эффективность. Поэтому формат хранения следует выбирать исходя из задачи.

Список источников

1. Тьюарсон Р.П. Разреженные матрицы. (Sparse Matrices, 1973). Перевод с английского Э.М. Пейсаховича под редакцией Х.Д. Икрамова. Художник К. Сиротов. Москва: Издательство «Мир»: Редакция литературы по математическим наукам, 1977.
2. Практикум по циклу дисциплин “Информатика”. Ч. II. 2012/13 уч. года