Министерство науки и высшего образования 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)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3 ПО ДИСЦИПЛИНЕ:
ТИПЫ И СТРУКТУРЫ ДАННЫХ

<Обработка разреженных матриц>

Студент ***<Ермаков И.Г>***

Группа ***<ИУ7-32Б>***

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **<Ермаков И.Г>**

Преподаватель _____ **<Фамилия ИО>**

2024

Оглавление

Цель работы

Реализация алгоритмов обработки разреженных матриц, сравнение эффективности применения этих алгоритмов со стандартными алгоритмами обработки матриц при различном размере матриц и степени их разреженности.

Описание условия задачи

Разреженная (содержащая много нулей) матрица хранится в форме 3-х объектов (CSR):

- вектор A содержит значения ненулевых элементов;
- вектор JA содержит номера столбцов для элементов вектора A;
- вектор IA, в элементе Nk которого находится номер компонент в A и JA, с которых начинается описание строки Nk матрицы A.

1. Смоделировать операцию сложения двух матриц, хранящихся в этой форме, с получением результата в той же форме.
2. Произвести операцию сложения, применяя стандартный алгоритм работы с матрицами.
3. Сравнить время выполнения операций и объем памяти при использовании этих 2-х алгоритмов при различном проценте заполнения матриц.

Описание технического задания

Входные данные

Изначально пользователю предагается набор опций

```
Выберите одну из доступных опций
1 - Ввод матриц
2 - Вывод входных матриц
3 - Сложение матриц стандартным алгоритмом
4 - Сложение матриц алгоритмом CSR
5 - Сравнение времени работы и памяти при разном проценте заполнения матриц для стандартного и сокращенного метода сложения
Введите режим работы (0 для выхода):
```

Приложение 1

Программа завершается только в том случае, если пользователь введет 0;
Рассмотрим каждую опцию и вариативность вариантов ответа после нее:

Опция 1 - Ввод матриц

После ввода этой опции пользователя встречает выбор заполнения матрицы

```
Введите первую матрицу
Выберите способ заполнения матрицы, введите 0 - стандартный, введите 1 - координатный
```

при заполнении матрицы стандартным методом, пользователь должен указать кол-во строк и столбцов матрицы, затем пользователь долен ввести все значения вручную. Так же доступен ввод координатным методом, где пользователь так же вводит кол-во строк и столбцов, затем предоставляется ввод кол-ва ненулевых элементов матрицы. Затем для каждого ненулевого элемента пользователь вводит его координаты и значение.

Опция – 5

Пользователю предоставляется посмотреть на замеры скорости и памяти для сравнения двух алгоритмов (обычный/упрощенный). После выбора опции 5, запрашивается кол-во матриц, и их размеры для замеров. Затем производятся замеры времени выполнения и расчет памяти.

Для всех остальных остальных опций их функции описаны на приложении выше.

Выходные данные
В зависимости от выбранной опции вывода матриц, пользователю предоставляется возможность выбрать формат выводимой матрицы (обычный формат/CSR формат). Вывод доступен как в обычной опции вывода входных матриц, так и в опциях сложения, вывод там встроен для большего удобства.

Действие программы

Программа работает до тех пор пока пользователь не введет 0, в противном случае программа будет выводить на экран сообщение об ошибке и приложение 1

Обращение к программе

Исполняемый файл программы запускается из корня проекта ./app.exe

Аварийные ситуации

Ошибка ввода – возникает при некорректном вводе данных.
Ошибка размера – возникает при указании недопустимого размера матрицы.
Ошибка выделения памяти – происходит при невозможности выделения необходимой памяти.
Ошибка неверного элемента – возникает при вводе недопустимого значения элемента.
Ошибка ненулевого элемента – ошибка при работе с ненулевыми элементами в матрице.
Ошибка столбцов – возникает, если количество столбцов указано неверно.
Ошибка координат – ошибка при вводе или работе с координатами в матрице.
Ошибка несовпадения размеров – возникает, если размеры матриц не равны при сложении.
Ошибка неверного выбора – ошибка при вводе некорректного режима работы программы.

Описание структуры данных

```
typedef struct {
    int *A;
    int *JA;
    int *IA;
    int rows;
    int cols;
    int nnz;
} CSRMatrix;
```

На приложении выше описана основная структура данных которая задействовалась в процессе выполнения лабораторной работы

Массив A – хранит значения всех ненулевых элементов
Массив JA – хранит столбцы в которых располагается i-тый элемент
Массив IA –хранит индексы начала каждой строки в массиве ненулевых элементов A
Поле rows хранит кол-во строк в матрице
Поле cols хранит кол-во столбцов в матрице
Поле nnz хранит кол-во ненулевых элементов в матрице

Описание основных функций программы

```
CSRMatrixResult add_csr_matrices(const CSRMatrix *matrix1, const
CSRMatrix *matrix2);
```

Данная функция складывает матрицы алгоритмом для сложения CSR матрицы (упрощенный алгоритм).

```
CSRMatrixResult add_std_matrices(const CSRMatrix *matrix1, const
CSRMatrix *matrix2);
```

Данная функция складывает матрицы обычным алгоритмом сложения матриц.

```
void compare_algorithms(int rows, int cols, int fill_percentage);
```

Данная функция выводит время работы и кол-во затраченной памяти для работы обычного или сокращенного метода сложения.

Описание алгоритма

Алгоритм для сложения двух матриц обычным способом:

Алгоритм складывает две матрицы в формате CSR. Сначала он проверяет, совпадают ли размеры матриц, и если нет — возвращает ошибку. Затем создается новая матрица для результата, инициализируются массивы для хранения ненулевых элементов и индексов. Алгоритм проходит по каждой строке матриц, и для каждого элемента строки ищет его положение в соответствующем столбце, суммируя значения из двух исходных матриц. Если сумма ненулевая, она добавляется в результирующую матрицу. В конце обновляется количество ненулевых элементов и завершается выполнение.

Алгоритм для сложения двух матриц упрощенным методом сложения

Алгоритм складывает две матрицы в формате CSR более эффективно, сравнивая их ненулевые элементы по столбцам в каждой строке. Сначала проверяется, совпадают ли размеры матриц, и если нет, возвращается ошибка. Затем создается новая матрица для результата, инициализируются массивы для хранения ненулевых элементов, индексов столбцов и начала каждой строки. В каждой строке одновременно перебираются ненулевые элементы обеих матриц, сравнивая индексы столбцов. Если элементы находятся в одном столбце, их значения складываются. Если элементы находятся в разных столбцах, добавляется ненулевое значение из одной из матриц. Если сумма ненулевая, она записывается в результирующую матрицу. В конце обновляется количество ненулевых элементов и возвращается итоговая матрица.

Оценка эффективности работы алгоритма

Процент заполнения 1

Размер	Время работы стандартного алгоритма (такты)	Время работы упрощенного алгоритма (такты)	Память для стандартного алгоритма (байты)	Память для упрощенного алгоритма (байты)
100 * 100	189286	7765	40000	1996
250 * 250	1020713	39419	250000	10980
500 * 500	6215271	135879	1000000	41820
1000 * 1000	45105070	510844	4000000	163044

Процент заполнения 10

Размер	Время работы стандартного алгоритма (такты)	Время работы упрощенного алгоритма (такты)	Память для стандартного алгоритма (байты)	Память для упрощенного алгоритма (байты)
100 * 100	672310	58367	40000	15524
250 * 250	11835493	316186	250000	96244
500 * 500	76318278	967411	1000000	382428
1000 * 1000	372344382	3989150	4000000	1524124

Процент заполнения 25

Размер	Время работы стандартного алгоритма (такты)	Время работы упрощенного алгоритма (такты)	Память для стандартного алгоритма (байты)	Память для упрощенного алгоритма (байты)
100 * 100	1243544	63596	40000	35596
250 * 250	16636510	597703	250000	219564
500 * 500	107783683	2415073	1000000	878644
1000 * 1000	842012974	10074907	4000000	3502004

Процент заполнения 50

Размер	Время работы стандартного алгоритма (такты)	Время работы упрощенного алгоритма (такты)	Память для стандартного алгоритма (байты)	Память для упрощенного алгоритма (байты)
100 * 100	2000570	152636	40000	60324
250 * 250	32029185	1140463	250000	376140
500 * 500	173676180	5946650	1000000	1504332
1000 * 1000	1914491272	35431436	4000000	6003292

Процент заполнения 75

Размер	Время работы стандартного алгоритма (такты)	Время работы упрощенного алгоритма (такты)	Память для стандартного алгоритма (байты)	Память для упрощенного алгоритма (байты)
100 * 100	2210388	208464	40000	75388
250 * 250	27703356	1263977	250000	469908
500 * 500	288190501	10649713	1000000	1877140
1000 * 1000	2994027555	42351261	4000000	7502700

Процент заполнения 100

Размер	Время работы стандартного алгоритма (такты)	Время работы упрощенного алгоритма (такты)	Память для стандартного алгоритма (байты)	Память для упрощенного алгоритма (байты)
100 * 100	2160364	182331	40000	80404
250 * 250	57718924	798738	250000	501004
500 * 500	308516826	5072599	1000000	2002004
1000 * 1000	1862381687	11436720	4000000	8004004

Выводы на основе полученных данных
Сравнение по времени работы :

Оптимизированный алгоритм значительно выигрывает по времени на всех уровнях заполнения, особенно при низком проценте заполнения (1%, 10%). Например, при размере 1000x1000 и заполнении 1%, время работы оптимизированного алгоритма составляет 510844 такта, тогда как стандартный алгоритм требует 45105070 тактов.
Однако, при увеличении заполнения до 100%, прирост во времени у оптимизированного алгоритма все еще заметен, но разница становится менее значительной. Например, при размере 1000x1000 и заполнении 100% оптимизированный алгоритм работает за 11436720 тактов, а стандартный — за 186238167 тактов.
Сравнение использования памяти:

При низком проценте заполнения (например, 1% и 10%), оптимизированный алгоритм использует меньше памяти по сравнению со стандартным алгоритмом. Это связано с тем, что он сохраняет меньше ненулевых элементов и эффективнее работает с разреженными данными.
При более высоком проценте заполнения (например, 50% и выше), оптимизированный алгоритм начинает использовать больше памяти. Например, при заполнении 50% и размере 1000x1000, оптимизированный алгоритм использует 6003292 байт, тогда как стандартный требует только 4000000 байт. Это связано с тем, что оптимизированный алгоритм хранит больше промежуточных данных и требует больше ресурсов для динамического перераспределения памяти.
Теоретическая оценка по О-большому:

Стандартный алгоритм имеет сложность $O(n^2)$, так как он проходит через каждый элемент строки матрицы и выполняет два вложенных цикла для каждой строки и каждого столбца.
Оптимизированный алгоритм имеет улучшенную сложность O(nnz), где nnz — количество ненулевых элементов. Это связано с тем, что он оптимизирует процесс путем работы только с ненулевыми элементами, а не с полным множеством строк и столбцов.
Теоретический расчет памяти:
Для стандартного алгоритма, память оценивается как $O(n^2)$ для матриц с плотным заполнением, где n — размер матрицы. Это соответствует ситуации с максимальным заполнением, когда все элементы ненулевые.
Оптимизированный алгоритм, несмотря на преимущество по времени, при высоком проценте заполнения использует больше памяти, так как выделяет дополнительные ресурсы для промежуточных вычислений, что приводит к сложению ненулевых элементов по столбцам с большим объемом данных.

Выводы:

Оптимизированный алгоритм выигрывает по времени на всех уровнях заполнения, особенно при низком проценте ненулевых элементов, но начинает проигрывать по памяти при более плотных заполнениях (50% и выше).
При низком заполнении, оптимизированный алгоритм эффективнее как по времени, так и по памяти, но при заполнении 50% и более стандартный алгоритм начинает выигрывать по использованию памяти, хотя и проигрывает по времени.
Если важен баланс между временем и памятью, оптимизированный алгоритм лучше использовать при низком заполнении матрицы (до 25%). При высоком заполнении может потребоваться пересмотр подхода в зависимости от требований к ресурсам.

Тесты

Описание теста	Входные данные	Вывод
Неверный пункт меню	6	Неправильный режим работы. Пожалуйста, выберите корректный режим.
Неверный пункт меню	-1	Неправильный режим работы. Пожалуйста, выберите корректный режим.
Неверный пункт меню	ww	Ошибка: неверная опция! Пожалуйста, введите корректный режим..
Символ в матрице при обычном вводе (не координатном)	3 3 - размеры 1 2 a	Ошибка ввода первой матрицы!
При вводе координатным методом, выбор столбца куда нужно вставить элемент больше размера	4 4 0 5 12	Ошибка ввода первой матрицы!
В размерах матрицы символ	4 a	Ошибка ввода первой матрицы!
При вводе координатным методом, выбор строки куда нужно вставить элемент больше размера	4 4 5 2 12	Ошибка ввода первой матрицы!
При вводе матрицы имеют разные размеры для сложения стандартным алгоритмом	4 4 3 3	Ошибка: матрицы должны быть одинакового размера для сложения!
При вводе матрицы имеют разные размеры для сложения упрощенным алгоритмом	4 4 3 3	Ошибка при сложении матриц в формате CSR!

Вывод

На основе проведенных экспериментов и измерений, можно сделать следующие выводы. Применение стандартного или оптимизированного алгоритма сложения разреженных матриц в формате CSR зависит от степени заполнения матрицы.

Оптимизированный алгоритм значительно выигрывает по времени при низком проценте заполнения матриц (до 25%), что делает его целесообразным для работы с действительно разреженными матрицами, где ненулевые элементы встречаются редко. В таких случаях он демонстрирует высокую производительность как по времени, так и по памяти.

Стандартный алгоритм уступает по времени, но выигрывает по использованию памяти при высокой плотности матриц (заполнение более 50%). Это связано с тем, что оптимизированный алгоритм использует больше памяти для временных вычислений, что делает стандартный подход более эффективным с точки зрения памяти при плотных матрицах.

В целом, оптимизированный алгоритм предпочтителен для работы с сильно разреженными матрицами, где важна скорость. Стандартный алгоритм может быть целесообразен при обработке матриц с высоким уровнем заполнения, где критичен объем используемой памяти.

Ответы на контрольные вопросы

1. **Что такое разреженная матрица, какие схемы хранения таких матриц Вы знаете?**

Разреженная матрица — это матрица, содержащая много нулей. Схемы хранения таких матриц включают формат CSR (Compressed Sparse Row), COO (Coordinate Format), и DIA (Diagonal Format). Эти схемы хранят только ненулевые элементы, что экономит память.

2. **Каким образом и сколько памяти выделяется под хранение разреженной и обычной матрицы?**

Для разреженной матрицы выделяется память только под ненулевые элементы и их индексы: массивы значений, столбцов и строк (в CSR). Память пропорциональна количеству ненулевых элементов. Обычная матрица требует память для всех элементов, независимо от их значений, то есть $O(n^2)$ для матрицы размером $n \times n$.

3. **Каков принцип обработки разреженной матрицы?**

Принцип обработки разреженной матрицы заключается в работе только с ненулевыми элементами, что ускоряет вычисления и снижает использование памяти. Например, в CSR хранятся только значения ненулевых элементов и их индексы.

4. **В каком случае для матриц эффективнее применять стандартные алгоритмы обработки матриц? От чего это зависит?**

Стандартные алгоритмы эффективнее для плотных матриц, где большинство элементов ненулевые. Это зависит от степени заполнения: если матрица плотная, разреженные схемы хранения будут неэффективны из-за накладных расходов на хранение индексов.