Отчёт

Лабораторная работа 3 по предмету «Типы и структуры данных».
Керимов Ахмед, ИУ7-34Б, Вариант 4.

Цель работы: реализация алгоритмов обработки разрежённых матриц, сравнение этих алгоритмов со стандартными алгоритмами обработки матриц при различном размере матриц и степени их разрежённости.

Техническое задание

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

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

1. Смоделировать операцию умножения матрицы и вектора-столбца, хранящихся в этой форме, с получением результата в той же форме.
2. Произвести операцию умножения, 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.
4. Обеспечить возможность заполнения разрежённых матриц вручную.
5. Обеспечить вывод разрежённой матрицы на экран. При малых размерах матрицы (не больше 20 в высоту, 40 в ширину) вывести матрицу в стандартном виде.
6. Обеспечить хранение разрежённых матриц в файлах.

Аварийные ситуации

1. Некорректный ввод операции. Будет выведено сообщение об ошибке.
2. Некорректный путь до матрицы. Будет выведено сообщение об ошибке.
3. Ввод некорректной матрицы. Будет выведено сообщение об ошибке.
4. Умножение матриц некорректного размера. Будет выведено сообщение об ошибке.
5. Ошибка выделения памяти. Будет выведено сообщение об ошибке.
6. Добавление нулевого элемента или выход за пределы разрежённой матрицы. Будет выведено сообщение об ошибке и предложение повторить ввод элемента.

Структуры данных

```
typedef struct node node_t;

struct node {
    void *data;
    node_t *next;
};
```

```
typedef float T;
typedef size_t U;

#pragma pack(push, 1)
typedef struct {
    T *a;       // T
    U *ia;      // U
    node_t *ja; // U
    size_t reserved;
    size_t size;
    size_t height;
    size_t width;
} sparse_matrix_t;
#pragma pack(pop)
```

```
typedef struct {
    double **data;
    size_t height, width;
    size_t nonzeros;
} matrix_t;
```

Алгоритмы

Умножение разреженной матрицы на разреженный вектор-столбец

Сначала проверяются размеры матриц, затем, в случае успеха, создаётся результирующая матрица нужного размера. Для удобства объяснения, разобьём матрицу на классы эквивалентности по номеру строки. Для каждой строки матрицы пробежимся по вектору-столбцу и найдём элементы, которые можно умножить. Если такие есть, то, складывая все такие произведения, добавляем их в результирующую матрицу. Понятно, что так как матрица и вектор-столбец индексно упорядочены, то сложность алгоритма в худшем случае описывается как $O(M + N^2)$, где M и N — количество ненулевых элементов в матрице и векторе-столбце соответственно (память — константа). Это, конечно, не совсем так, ведь для столбцов используется односвязный список, и в алгоритме много ветвлений. Матрица итерируется от начала и до конца, вектор-столбец — каждый раз от начала и пока не встретится элемент, строка которого больше номера столбца последнего ненулевого элемента в строке матрицы.

См. mult_sm (в файле `sparse_matrix.c`).

Вопросы

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

Разрежённая матрица — это матрица с преимущественно нулевыми элементами. В противном случае, если бóльшая часть элементов матрицы ненулевые, матрица считается плотной.

Схемы хранения разрежённых матриц: координатный формат, разрежённый строчный формат (CSR), разрежённый столбцовый формат (CSC), модифицированная строчная схема (MSR).

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

Под хранение обычной матрицы выделяется двумерный массив размера $M \times N$ типа T, под разрежённый — массив размера K типа T и два массива размера K типа U, где M, N — количество строк и столбцов матрицы, K — количество ненулевых элементов, тип T для хранения элементов матрицы, тип U для хранения индексов матрицы

3. Каков принцип обработки разрежённой матрицы?

Разрежённость матрицы следует учитывать только в том случае, когда из этого можно извлечь выгоду за счёт игнорирования нулевых элементов.

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

Стандартные алгоритмы обработки матриц выгоднее применять на матрицах малого размера или же на плотных матрицах.

Таблицы

При тестировании не были использованы матрицы больших размеров, так как стоит программное ограничение на 2500 ненулевых элементов. Тем более что тенденция измеряемых величин видна и на приведённых ниже данных.

Время

sparse | simple, (sec). В процентах указана заполненность матрицы.

	100 %	**50 %**	**25 %**	**10 %**	**5 %**	**1 %**
50	0.00286 | 0.00010	0.00124 | 0.00007	0.00044 | 0.00017	0.00035 | 0.00026	0.00006 | 0.00010	0.00004 | 0.00027
100			0.00282 | 0.00050	0.00127 | 0.00035	0.00087 | 0.00186	0.00025 | 0.00158
200					0.00130 | 0.00157	0.00038 | 0.00176

500						0.00155 | 0.00671

Память

sparse | simple, (bytes)

	100 %	**50 %**	**25 %**	**10 %**	**5 %**	**1 %**
50	50016 | 20008	25016 | 20008	12516 | 20008	5016 | 20008	2516 | 20008	516 | 20008
100			50016 | 80008	20016 | 80008	10016 | 80008	2016 | 80008
200					40016 | 320008	8016 | 320008
500						50016 | 2000008

Вывод

В результате работы реализованы алгоритмы обработки разрежённых матриц, стандартные алгоритмы обработки матриц, которые были протестированы при различных размерах матриц и степенях их разрежённости. Выяснено, что разрежённые матрицы выгоднее использовать по времени при степени заполненности не более 5%, а по памяти — не более 40%.