Федеральное государственное бюджетное

образовательное учреждение

высшего образования

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»

(национальный исследовательский университет)

Факультет № 8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Задание № 7
«Разреженные матрицы»

по дисциплине «Практикум на ЭВМ»

2 семестр

Студент	Мезенин О.А.
Группа	М8О-106Б-21
Преподаватель	Дубинин А.В.
Оценка	

Москва, 2022

Содержание

1 Введение

Разреженная матрица - это матрица, состоящая преимущественно из нулевых элементов. Как правило, такая матрица также является очень объёмной. Поэтому при её хранении может возникнуть проблема нехватки оперативной памяти компьютера. Хранение разреженных матриц можно оптимизировать, используя специальные методы и структуры данных.

Целью работы является ознакомление с методами хранения разреженных матриц, а также выполнение практического задания, используя один из алгоритмом хранения разреженных матриц.

2 Методы хранения разреженных матриц

Все методы основываются на хранении только ненулевых элементов матрицы. Методы отличаются друг от друга затратами памяти, а также скоростью доступа к элементам и добавления новых элементов.

2.1 Хранение в одном массиве

Ненулевому элементу соответствуют две ячейки: первая содержит номер столбца, вторая содержит значение элемента. Некорректное значение номера столбца (например, -1) означает конец строки, а вторая ячейка содержит в этом случае номер следующей хранимой строки.

-1	Номер строки	Номер столбца	Значение	Номер столбца	Значение	$\cdots$

$\ldots$

-1	Номер строки	Номер столбца	Значение	$\cdots$	-1	-1

2.2 Координатный формат хранения

При координатном формате хранятся только ненулевые элементы матрицы и их координаты (номера строк и столбцов).

Есть вариант хранения в виде трёх одномерных массивов. Пусть в массиве YE хранятся ненулевые значения исходной матрицы, в массиве LI - первый индекс (номер строки) каждого элемента из YE, в массиве LJ - второй индекс (номер столбца) каждого элемента из YE.

Но можно объединить массивы LI и LJ в один массив LB: пусть каждому элементу a из массива YE соответствует значение $\lambda_{ij} = i * n + j$, где n - количество столбцов исходной матрицы, i - элемент из LI, соответствующий элементу a, j - элемент из LJ, соответствующий элементу a.

LB:	λ_{ij}	λ_{ij}	λ_{ij}	$\cdots$	λ_{ij}
	$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$
YE:	Значение	Значение	Значение	$\cdots$	Значение

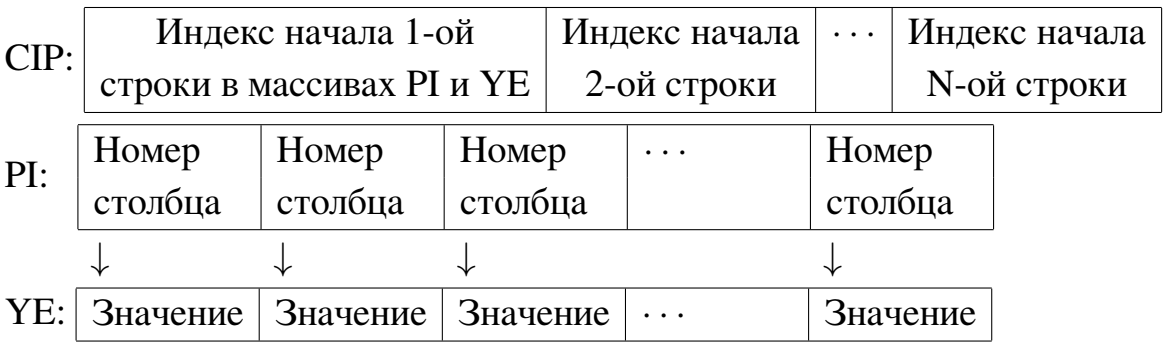
Координатный формат может быть:

1. Упорядоченный (по строкам/столбцам)
   (a) Быстрый доступ к строкам/столбцам
   (b) Необходимость перепаковки при вставке/удалении элементов
2. Неупорядоченный
   (a) «Переборный» доступ к элементам
   (b) Быстрая вставка/удаление элементов

2.3 Разреженный строчный формат

При разреженном строчном формате (Compressed Row Storage, CRS) матрица хранится в виде трёх одномерных массивов. В массиве YE хранятся ненулевые значения исходной матрицы, в массиве PI - первый индекс (номер строки) каждого элемента из YE.

Местоположение первого ненулевого элемента в каждой строке записывается в одномерный массив CIP. Элементы i-ой строки хранятся в позициях массива YE с номера CIP[i] по CIP[$i+1$]-1. Если в строке i встречаются только нулевые элементы (строка является пустой), то значение CIP[i] == CIP[$i+1$]. Если массив YE состоит из n строк, то длина массива CIP будет $n+1$.



Разреженный строчный формат обеспечивает эффективный доступ к строчкам матрицы, но затруднен доступ к столбцам. Поэтому предпочтительно использовать этот способ хранения в тех алгоритмах, в которых преобладают строчные операции.

Если в решаемой задаче необходимо осуществлять доступ к элементам по столбцам, то схему хранения можно изменить. Хранить элементы можно не

по строкам, а по столбцам. Для этого существует аналогичный *разреженный столбцовый формат* (Compressed Column Storage, CCS).

3 Практика

Задание: составить программу на языке Си с функцией умножения вектора-строки на прямоугольную разреженную матрицу с элементами вещественного типа, которая:
1) вводит матрицу в обычном формате (по строкам), с одновременным размещением ненулевых элементов в разреженной матрице, используя координатный формат хранения.
2) печатает введённую матрицу во внутреннем представлении и в обычном виде.
3) выполняет умножение вектора-строки на матрицу.
4) выводит количество ненулевых элементов результата умножения.

3.1 Реализация

Для выполнения задания были подготовлены векторы svector_int и svector_dbl, которые подключаются в заголовке программы:

```
#include "svdbl.h"
#include "svint.h"
```

Структура разреженной матрицы будет выглядеть следующим образом:

```
typedef struct {
    svector_int lb;
    svector_dbl ye;
    int m;
    int n;
} matrix;
```

Векторы lb и ye служат для хранения координатных индексов k и ненулевых значений разреженной матрицы соответственно. Значения m и n - количество строк и столбцов разреженной матрицы.

Функция для ввода и вывода матрицы:

```
matrix read_matrix() {
    matrix res;
    int m, n;
    scanf("%d%d", &m, &n);
    res.m = m;
    res.n = n;
    svint_init(&res.lb);
```

```
    svdbl_init(&res.ye);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            double a;
            scanf("%lf", &a);
            if (a != 0) {
                svdbl_push_back(&res.ye, a);
                svint_push_back(&res.lb, i*n+j);
            }
        }
    }
    return res;
}

void print_matrix(matrix *matr) {
    int cur = 0;
    for (int i = 0; i < matr->m; i++) {
        for (int j = 0; j < matr->n; j++) {
            if (cur < svint_get_size(&matr->lb)
            && i*matr->n + j == svint_get(&matr->lb, cur)) {
                printf("%lf ", svdbl_get(&matr->ye, cur));
                cur++;
            } else {
                printf("%lf ", 0.);
            }
        }
        printf("\n");
    }
}
```

Реализация функции для умножения вектора-строки на разреженную матрицу:

```
svector_dbl dot(svector_dbl *vec, matrix *matr) {
    svector_dbl res;
    svdbl_init(&res);
    svdbl_set_size(&res, matr->n);
    for (int cur = 0; cur < svint_get_size(&matr->lb); cur++) {
        int k = svint_get(&matr->lb, cur);
        int i = k / matr->n;
        int j = k % matr->n;
        double val = svdbl_get(vec, i)*svdbl_get(&matr->ye, cur);
        svdbl_set(&res, j, svdbl_get(&res, j)+val);
```

```
    }
    return res;
}
```

Так как результатом умножения будет вектор-строка с размером n, то мы инициализируем вектор res размера n, заполненный нулями. Далее мы проходим счётчиком по вектору lb (и одновременно по ye) и вычисляем координату i и j, после чего умножаем i-ое значение вектора vec на текущее значение вектора ye и прибавляем результат к j-му значению результирующего вектора res. Временная сложность алгоритма будет O(max(n, u)), где u - количество ненулевых элементов в матрице.

Функция для подсчёта ненулевых значений вектора:

```
int get_number_nonull(svector_dbl *vec) {
    int res = 0;
    for (int i = 0; i < svdbl_get_size(vec); i++) {
        if (svdbl_get(vec, i) != 0) res++;
    }
    return res;
}
```

Функция для вывода матрицы во внутреннем представлении:

```
void print_matrix_r(matrix *matr) {
    printf("LB: ");
    for (int i = 0; i < svint_get_size(&matr->lb); i++) {
        printf("%d ", svint_get(&matr->lb, i));
    }
    printf("\nYE: ");
    for (int i = 0; i < svdbl_get_size(&matr->ye); i++) {
        printf("%lf ", svdbl_get(&matr->ye, i));
    }
}
```

Функция $main$ будет выглядеть следующим образом:

```
int main() {
    matrix matr = read_matrix();
    svector_dbl vec = read_vector(matr.m);
    svector_dbl new_vec = dot(&vec, &matr);
    printf("Матрица во внутреннем представлении:\n");
    print_matrix_r(&matr);
    printf("\nМатрица в обычном виде:\n");
```

```
    print_matrix(&matr);
    printf("Результат умножения:\n");
    print_vector(&new_vec);
    printf("Количество ненулевых элементов результата умножения: %d",
           get_number_nonull(&new_vec));
    return 0;
}
```

3.2 Тесты

```
3 3
0 1 0
0 0 2
3 0 0
1 2 3
Матрица во внутреннем представлении:
LB: 1 5 6
YE: 1.000000 2.000000 3.000000
Матрица в обычном виде:
0.000000 1.000000 0.000000
0.000000 0.000000 2.000000
3.000000 0.000000 0.000000
Результат умножения:
9.000000 1.000000 4.000000
Количество ненулевых элементов результата умножения: 3

4 3
1 0 0
0 0 2
3 0 0
0 0 5
0 1 2 3
Матрица во внутреннем представлении:
LB: 0 5 6 11
YE: 1.000000 2.000000 3.000000 5.000000
Матрица в обычном виде:
1.000000 0.000000 0.000000
0.000000 0.000000 2.000000
3.000000 0.000000 0.000000
0.000000 0.000000 5.000000
Результат умножения:
```

```
6.000000 0.000000 17.000000
Количество ненулевых элементов результата умножения: 2

2 4
1 0 0 2
0 3 0 0
0 3
Матрица во внутреннем представлении:
LB: 0 3 5
YE: 1.000000 2.000000 3.000000
Матрица в обычном виде:
1.000000 0.000000 0.000000 2.000000
0.000000 3.000000 0.000000 0.000000
Результат умножения:
0.000000 9.000000 0.000000 0.000000
Количество ненулевых элементов результата умножения: 1

2 4
-1 0 0 0
2 0 0 3
2 1
Матрица во внутреннем представлении:
LB: 0 4 7
YE: -1.000000 2.000000 3.000000
Матрица в обычном виде:
-1.000000 0.000000 0.000000 0.000000
2.000000 0.000000 0.000000 3.000000
Результат умножения:
0.000000 0.000000 0.000000 3.000000
Количество ненулевых элементов результата умножения: 1
```

4 Выводы

В ходе работы были рассмотрены три метода хранения разреженных матриц. Для экономии памяти при всех вариантах хранятся только ненулевые элементы, но методы все равно различаются по затратам оперативной памяти, а также скоростью доступа к элементам. Поэтому выбор метода хранения зависит от поставленной задачи.

5 Список источников

1. Хранение разреженной матрицы общего вида - https://docplayer.com/41553268-Hranenie-razrezhennoy-matricy-obshchego-vida.html

2. Разреженная матрица - https://ru.wikipedia.org/wiki/Разреженная_матрица