Типы и структуры данных — Сиденко Анастасия
Лабораторная работа 3
Вариант 4 — ИУ7-33Б
Обработка разреженных матриц.

Техническое задание:

Задача:

Реализовать алгоритмы обработки разреженных матриц, сравнить эффективность использования этих алгоритмов (по времени выполнения и по требуемой памяти) со стандартными алгоритмами обработки матриц при различном процентном заполнении матриц ненулевыми значениями и при различных размерах матриц.

Разреженная (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.

Входные данные: матрица и вектор-столбец.

Выходные данные: вектор-столбец полученный при умножении данной матрицы на вектор или время необходимое на расчет.

Взаимодействие с программой

Взаимодействие через консольное меню.

Вводится матрица и вектор-столбец (из файла, вручную или заполняется рандомно).

Аварийные ситуации:

1. Файл невозможно открыть. Вывод сообщения и аварийное завершение программы.

2. Некорректные вводимые данные. Вывод сообщения.

Используемые структуры:

Структура односвязный список.

```
struct list {
    int data;
    struct list *next;
};
```

Структура динамический вектор.

```
struct vector {
    int *data;
    int allocate;
    int size;
};
```

Алгоритмы:

1. Добавление элемента в список

```
struct list *push_list(int data)
```

2. Изменение элемента на значение

```
void list_add_plus(struct list **list, int val)
```

3. Изменение элемента на позиции

```
void list_plus(struct list **list, int j)
```

4. Добавление элемента вектора

void push_vector(struct vector *a, int data)

5. Печать списка

void print_list(const struct list *list)

6. Печать вектора

void print_vector(const struct vector vector)

7. Освобождение списка

void list_free(struct list *list)

8. Значение элемента по номеру в списке

int number_list(struct list *list, int n)

9. Умножение матрицы на столбец в разреженном виде

void multiplication(struct list *ja, struct vector ia, struct vector a, struct vector column, struct vector row, int m, int n, struct vector *res, struct vector *res_col)

10. Умножение матрицы на столбец в обычном виде

void multiplication_matrix(struct list *ja, struct vector ia, struct vector a, struct vector column, struct vector row, int m, int n)

Тесты:

1. Некорректные данные

```
Использовать заранее приготовленные данные?
да - 1, нет - 2
2
Введите количество строк и стобцов
й
Некорректный ввод%
```

```
Использовать заранее приготовленные данные?
да - 1, нет - 2
й
Некорректный ввод%
```

```
Использовать заранее приготовленные данные?
да – 1, нет – 2
1
Введите номер файла
55
Нет файла
```

```
Использовать заранее приготовленные данные?
да – 1, нет – 2
2
Введите количество строк и стобцов
3 3
1. Ручной ввод
2. Автозаполение
2
Введите количество ненулевых элементов
10
Некорректный ввод%
```

2. Вычисление произведения

```
Что вы хотите сделать
1. Вычислить произведение
2. Сравнить время работы
1
Время умножения матрицы на столбец в разреженном виде 5

Результат
0 0 0
```

3. Сравнение времени

```
Что вы хотите сделать
1. Вычислить произведение
2. Сравнить время работы
2
Время умножения матрицы на столбец в разреженном виде 6
Время умножения матрицы на столбец в обычном виде 11
```

Сравнение времени работы и заполнения матрицы

	Размер матрицы	**Заполнение**	**Время в тактах**	**Память**
В разреженном виде	5 на 5	100 %	10	448
В обычном виде			2	200

Использовать обычную матрицу эффективнее на 80% по времени и на 51% по памяти.

	Размер матрицы	**Заполнение**	**Время в тактах**	**Память**
В разреженном виде	100 на 100	50 %	601	88008
В обычном виде			57	80000

Использование обычной матрицы эффективнее на 90% по времени, и на 10% по памяти.

	Размер матрицы	**Заполнение**	**Время в тактах**	**Память**
В разреженном виде	100 на 100	25 %	209	40808
В обычном виде			76	80000

Использовать обычную матрицу эффективнее на 60% по времени, но менее эффективно на 49% по памяти.

	Размер матрицы	**Заполнение**	**Время в тактах**	**Память**
В разреженном виде	100 на 100	15 %	79	24808
В обычном виде			65	80000

Использовать обычную матрицу эффективнее на 15% по времени, но менее эффективно на 68% по памяти.

	Размер матрицы	Заполнение	Время в тактах	Память
В разреженном виде	100 на 100	10%	37	16808
В обычном виде			57	80000

Использовать разреженную матрицу эффективнее на 35% по времени, на 79% по памяти.

	Размер матрицы	Заполнение	Время в тактах	Память
В разреженном виде	100 на 100	100 %	2641	160808
В обычном виде			50	80000

Использовать обычную матрицу эффективнее на 98% по времени и на 51% по памяти.

	Размер матрицы	Заполнение	Время в тактах	Память
В разреженном виде	1000 на 1000	2 %	1119	8328
В обычном виде			2539	8000000

Использовать разреженную матрицу эффективнее на 45% по времени и на 99% по памяти.

Вывод:

Использовать вычисления с разреженными матрицами эффективнее чем с обычными при заполнении матрицы менее чем на 10-15%.

Контрольные вопросы:

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

Разреженная матрица - запись матрицы, в которой хранятся значения ненулевых элементов и их позиции.

Хранение матрицы:

1) Значение, номер строки и столбца
2) Значение, номер строки, указатели на начало и конец столбцов в массиве номеров строк
3) Значение, номер столбца, указатели на начало и конец строк в массиве номеров столбцов

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

Обычная матрица хранится целиком, разреженная матрица хранится в двух векторах и одном списке, хранятся только ненулевые элементы.

3. Каков принцип обработки разреженной матрицы?

Работа с ненулевыми элементами.

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

Эффективнее применять стандартные алгоритмы при большом заполнении матриц, больше 10-15%. Зависит соответсвенно от количества нулевых элементов.