Лабораторная работа №5 по дисциплине «Типы и структуры данных»

Обработка разреженных матриц

Цель работы:

Реализация алгоритмов обработки разреженных матриц, сравнение этих алгоритмов со стандартными алгоритмами обработки матриц.

Задание (Вариант 4):

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.

Исходные данные:

Программа работает с матрицами размерности меньше 1000 на 1000. И значение каждого элемента матрицы на превосходит 1000, для предотвращения переполнения.

Интерфейс программы:

1) Главное меню

```
Умножение матрицы на вектор-столбец
Выберите одно из следующих действий:
0: Работа с матрицей и вектором
1: Сравнение скорости работы алгоритмов работы с матрицами
2: Exit
```

2) Работа с матрицей и вектором

```
Умножение матрицы на вектор-столбец
Выберите одно из следующих действий:
0: Работа с матрицей и вектором
1: Сравнение скорости работы алгоритмов работы с матрицами
2: Exit
0
Умножение матрицы на вектор-столбец:
Ввод матрицы
Введите кол-во строк [1, 1000]: 10
Введите кол-во столбцов [1, 1000]: 10
Выберите ввод:
     1- ручной
     2- автоматический
2
Введите процент заполнения [0, 100]: 80
Диапазон значений [-1000, 1000]:
Начало диапазона:
-100
Конец диапазона:
100
Ввод вектора-стобца
Выберите ввод:
     1- ручной
     2- автоматический
1
Введите по 3 числа для каждого ненулевого эл-та (индексация с нуля)
 (строка  столбц  значение)
0 0 6
Продолжить (1-нет)? 0
7 0 5
Продолжить (1-нет)? 0
9 0 4
Продолжить (1-нет)? 1
0: Вывести первую матрицу в разреженном формате
1: Вывести вектор-столбец в разреженном формате
2: Вывести результат умножения в разреженном формате
3: Вывести первую матрицу в стандартном формате
4: Вывести вектор-столбец в стандартном формате
5: Вывести результат работы стандартного алгоритма
6: Закончить работу
```

3) Сравнение скорости работы алгоритмов работы с матрицами

```
Умножение матрицы на вектор-столбец
Выберите одно из следующих действий:
0: Работа с матрицей и вектором
1: Сравнение скорости работы алгоритмов работы с матрицами
2: Exit
1
Введите кол-во строк [1, 1000]: 100
Введите кол-во столбцов [1, 1000]: 100
Введите процент заполнения [0, 100]: 80
Время умножения в стандартном виде: 248
Время умножения в разреженном виде: 1590
Память, для  стандартной матрицы 40816
Память, для  разряженной матрицы 45088
```

Возможные ошибки пользователя:

1) При некорректном вводе или выборе меню пользователь получит следующее сообщение об ошибке "error, попытайтесь еще раз: "

Тесты

Матрица	Вектор	Представление матрицы	Результат	Представление результата
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4	0 1 0 0 4	n = 5 m = 5 A : 1 4 IA: 0 4 JA: 0 -1 -1 -1 1	0 0 0 0 16	n = 5 m = 1 A : 16 IA: 5 JA: 0
5 0 -5 0 0 0 0 0 0	1 0 1	n = 3 m = 3 A : 5 -5 IA: 0 0 JA: 0 -1 1	0 0 0	n = 3 m = 1 A : IA: JA: 0
-9 0 0 2 0 0 0 0 0 -9 1 -6 -2 0 6 4 0 0 0 0 7 0 0 0 -9	0 0 -5 0 0	n = 5 m = 5 A : -9 1 4 7 -6 -2 2 -9 6 -9 IA: 0 2 3 4 2 2 0 1 2 4 JA: 0 4 5 6 7	0 0 10 0 0	n = 5 m = 1 A : 10 IA: 2 JA: 0
Представление матрицы			Представление вектора	
n = 1000 m = 1000 A: 200 400 500 300 500 IA:1 999 500 1 999 JA: -1 0 -1… 2 (500) -1… 3(999)			n = 1000 m = 1 A : 2555 400 333 IA: 1 500 999 JA: 0	

Класс для хранения записей разреженной матрицы

```
class SparseMatrix {

private:
  std::vector<int> A;  //ненулевые элементы
  std::vector<int> IA; // номера строк  для элементов вектора A
  std::list<int> JA;
  int n;// количество строк
  int m;// количество столбцов

public:
  SparseMatrix(); //Констуктор
```

```
  SparseMatrix(const SparseMatrix &obj);

  SparseMatrix(NormalMatrix matrA);//конструктор преобразующийся обычную матрицу в разряженную

  ~SparseMatrix();

  SparseMatrix &operator=(const SparseMatrix &obj);

  int getN() const;

  int getM() const;

  void show();// печать на экран матрицы в разряженном виде

  // Преобразование из обычной матрицы в разряженную
  void convert(NormalMatrix &matrA);

  void mult(SparseMatrix &obj1, SparseMatrix &obj2); //умножение

  void transposition();
//вспомогательная функция умножения
  int smallmult(std::vector<int> &A1, std::vector<int> &IA1, int st1, int end1,
          std::vector<int> &A2, std::vector<int> &IA2, int st2, int end2);
//занимаемая память
  int memory();

};
```

Сравнение эффективности при работе с квадратной матрицей и вектором столбцом

Размерность матрицы	Заполнение %	Время, мкс		Память,байт	
		Стандартный	Разряжен.	Стандартный	Разряжен.
10	0	8	21	496	72
	25	13	67		336
	50	14	134		520
	75	12	164		600
	100	12	196		632

100	0	399	109	40 816	72
	25	181	810		18624
	50	429	3035		32512
	75	317	2323		43 256
	100	246	2305		51 696
250	0	991	99	252 016	72
	25	2314	8321		112696
	50	1384	7166		198 896
500	0	2753	151	1004016	72
	6	3131	3306		12048
	9	2803	4802		176208
	13	4381	5485		247728
	25	3240	9054		446064
1000	1	11767	3491	4 008 016	87648
	2	13506	5232		166440
	4	11559	8447		321752
	6	11379	10899		474248
	100	12003	85598		5 062 768

Использование разреженной матрицы оправдано при большой размерности матрицы и ее небольшом заполнение. В таком случае мы можем получить выигрыш по памяти и значительно уменьшить потребление памяти.

Вывод:

Время выполнения стандартного алгоритма почти линейно зависит от количества элементов в матрице. Однако при заполнение менее 10% матрицы разраженный алгоритм позволяет добиться более высокой скорости работы. Низкая скорость разряженного алгоритма связана с необходимостью

транспонировать одну из матриц для умножения и невозможностью прямого доступа к элементу по индексам.

Стандартный алгоритм хранения матриц подразумевает постоянное хранение всей матрицы включая 0. Использование разреженного алгоритма работы с матрицей заполненной не более чем на 70% позволяет добиться снижения использования памяти.

При заполнении матрицы не более чем на 10%-15% и размерности матрицы более 250*250 можно добиться значительного выигрыша при использовании разреженного алгоритма. Это позволяет получить значительный выигрыш по памяти и увеличить скорость работы программы. При заполнении матрицы более чем на 15% или маленькой ее размерности целесообразнее всего использовать стандартные способы обработки и хранения матриц.

Контрольные вопросы

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

Разреженная – матрица, содержащая достаточно большое количество элементов, из которых лишь малая часть является ненулевыми (n^(1+g) для матрицы размерности n, g<1).

Простейшая схема хранения разреженной матрицы: хранить массив ненулевых элементов (AN), и два массива их «координат» (I, J).

Кнут предложил хранить дополнительно массивы NR (содержит номер из AN следующего ненулевого j элемента, расположенного в матрице по строке) и NC (номера –''- по столбцу), а также массивы JR и JC (указатели для входа в строку и столбец).

Чанг и Густавсон предложили схему разреженного строчного формата: значения ненулевых элементов хранятся в массиве AN, соответствующие им столбцовые индексы - в массиве JA. Кроме того, используется массив указателей, например IA, отмечающих позиции AN и JA, с которых начинаются описание очередной строки. Дополнительная компонента в IA содержит указатель первой свободной позиции в JA и AN.

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

Для хранение обычной матрицы: M * N * sizeof(elem_t).

Память под разреженную матрицу выделяется в зависимости от схемы хранения. Память выделяется по мере наполнения ненулевыми элементами.

3.Каков принцип обработки разреженной матрицы?

Предполагает работу только с ненулевыми элементами.

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

Чем больше ненулевых элементов в матрице тем менее эффективно использовать разряженные алгоритмы. При достижение определенного процента наполнения наблюдается значительная деградация разряженного алгоритма по времени работы.