Лекция 5.1. Разреженные матрицы (продолжение)

В данной лекции описаны эффективные способы представления разреженных матриц.

Элемент при известных индексах легко найти, т.к. кол-во и положение элементов в массивах 1 - 5 соответствуют друг другу.

Т. о., можно найти все элементы столбца, например, столбца 2:

- `JC[2] = 1` - первый элемент
- `AN[1] = 3`. - сам элемент
- `I [1] = 1` - то есть - первая строка
- `NC[1] = 3` - следующий элемент `->` 3
- `AN[3] = 7. -> l[3] -> 2` - то есть - вторая строка
- `NC[3] = 5` - следующий элемент `->` 5
- `AN[5] = -5. -> l[5] -> 3` - т.е. - третья строка
- `NC[5] = 7` - следующий элемент `->` 7
- `AN[7] = 1 -> I[7] -> 4` - т.е. - четвертая строка
- `NC[7] = 0`, то есть, больше элементов нет
  Вообще, если `JC[I] = 0`, то i-ый столбец пуст.

Элемент a_{ij} можно найти, входя в список с первой строки. Обратная задача, если задано k, то `AN[k]`, `l[k]`, `J[k]`.

Достоинство - в любом месте можно включать или исключать элементы и можно эффективно сканировать строки и столбцы. Схема идеальна, когда матрица строится таким алгоритмом, где нельзя предсказать конечное число и позиции ненулевых элементов.

Недостаток: для хранения требуется достаточно большая накладная память

КРМ схема

Рейнболдт и Местеньи (1973г.) предложили модификацию схемы Кнута, уменьшающую накладную память. Она получила название КРМ (кольцевой) схемы.

В этой схеме связные списки строк и столбцов «закольцовываются», а начальные позиции списков включаются в указатели входа. Списки, ассоциированные со строками (столбцами), попарно не пересекаются, поэтому м. б. совместно хранимы одним массивом `NR` (т.е. `I` и `NR` хранится в `NR`) (для столбцов - массивом `NC`, т.е. `J+NC`).

Эта схема более плотная, чем сх. Кнута, но, если приходится просматривать элементы некоторой строки (или столбца), то мы не получим никакой информации о столбцовых (строчных) индексах этих элементов.

$$
A = \begin{array}{c} \\ \\ \\ \\ \end{array}
\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \end{array}
$$

$$
A= \begin{bmatrix} & 6. & & \\ 9. & 4. & & 7. \\ 5. & & & \\ & 2. & & 8. \end{bmatrix}
$$

Кольцевая КРМ схема:

	1	2	3	4	5	6	7
AN =	6.	9.	4.	7.	5.	2.	8.
I	1	2	2	2	3	4	4
NR =	0 1	3	4	0 2	0 3	7	0 4
J	2	1	2	4	1	2	4
NC =	3	5	6	7	0 1	0 2	0 4
JR =	1	2	5	6	- Номер элемента в AN, с которого начинается строка		
JC =	2	1	0	4	– Номер элемента в AN, с которого начинается столбец		

Рассмотрим, как можно найти a_{ij}: (допустим, $a_{2,4}$)

- сначала сканируется `i`-я строка, при этом определяется множество `Si` позиций элементов этой строки в массиве `AN`.
- Т. Е., Si =
- `S2 -> JR[2] -> 2; NR[2]->3, NR[3]->4, NR[4]->2` (кольцо), то есть, элементы:
  - `AN[2]->9`;
  - `AN[3]->4`;
  - `AN[4]->7`

Подразумевается, что повторения отсутствуют, т. е., каждой паре `p`, `q` строчного и столбцового индексов соответствует самое большее одна позиция в каждом из массивов AN, NR, NC. Далее сканируем j-ый столбец и получаем аналогично:

- `Sj = S4->JC[4]->4; NC[4]->7, NC[7]->4`, то есть,кольцо
- т.о. элементы: `AN[4]->7.`, `AN[7]->8.`

Пересечение `Sj` и `Si` даст сам элемент, то есть, `Sj ^ Si`

- `Si = AN[4]->7, a_2_4 = 7`.

Обратная задача, то есть, по заданной позиции `k` в `AN` найти соответствующие индексы `i` и `j` не имеет другого решения как просмотр всей матрицы.

Вариант КРМ схемы, для нахождения нужных индексов

Здесь указатели входа для строк и столбцов по существу включаются в соответствующие кольцевые списки. Один из способов такого включения: отрицательные числа для ссылок на указатели строк и столбцов. Тогда просматриваем ряд (строку или столбец) до тех пор, пока не будет найден отрицательный элемент (ссылка), абсолютная величина кот. есть номер ряда.

В то же время схема отсылает к соответствующему указателю входа, так что просмотр ряда при необходимости может продолжаться.

Поиск строчного индекса элемента AN(3)

	1	2	3	4	5	6	7
AN =	6.	9.	4.	7.	5.	2.	8.
						-индекс строки	
NR =	-1	3	4	-2	-3	7	-4
NC =	3	5	6	7	-1	-2	-4
JR =	1	2	5	6			
JC =	2	1	0	4			

Так как `NR[3] = 4` и `NR[4] = -2`, то индекс `i = 2`;

Кр. того, `JR[2]->2`, а `AN[2]->3`, т.е., опять же `i = 2`.

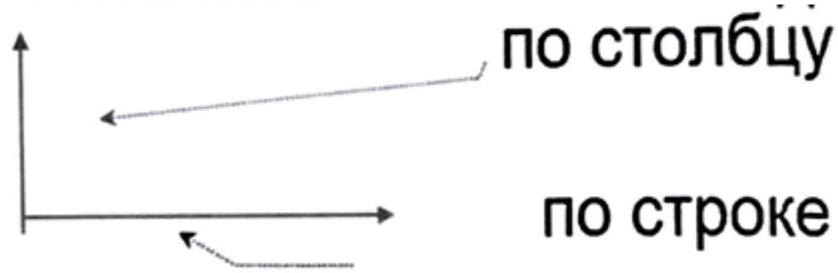
Чтобы найти в массиве AN элемент a_{ij}, можно сделать, как описано выше, а можно еще рассмотреть `i`-ю строку и найти столбцовый индекс каждого элемента, сканируя соответствующие столбцовые списки, пока не встретится `j`.

Вперед по строке, назад по столбцу

Лакрум в 1971г. использовал еще один вариант схемы Кнута для хранения симметричных положительно определенных матриц, содержавших числа или подматрицы.

Хранится только диагональ и верхний треугольник матрицы. Нужен только один массив указателей входа, которые отсылают нас к диагональным элементам.

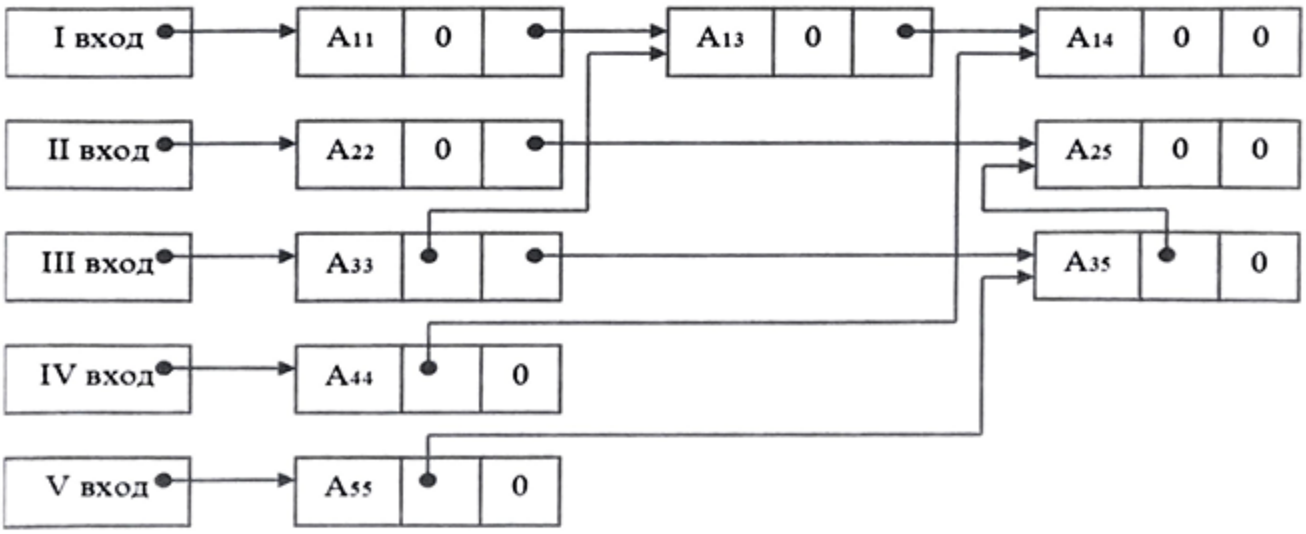
А от каждого диагонального элемента начинаются два списка: по строке и столбцу



Идею схемы легко распространить на матрицу общего вида, если существует диагональ ненулевых элементов (так называемая трансверсаль), то от каждого элемента может начинаться уже не 2, а 4 списка с соответствующими значениями строк и столбцов.
перед по строке, назад по столбцу

Например:

$$\begin{bmatrix} a_{11} & & a_{13} & a_{14} & \\ & a_{22} & & & a_{25} \\ & & a_{33} & & a_{35} \\ & & & a_{44} & \\ & & & & a_{55} \end{bmatrix}$$



Разреженный строчный формат

Схема хранения разреженных матриц,(Чанг (1969г.) и Густавсон (1972г.)) - это разреженный строчный формат Compressed Sparse Row (CSR).

Она предъявляет минимальные требования к памяти и очень удобна при выполнении операций сложения, умножения матриц, перестановок строк и столбцов, транспонировании, решении систем линейных уравнений при хранении коэффициентов в разреженных матрицах как прямыми, так и итерационными методами.

Значения ненулевых элементов хранятся в массиве `AN`, соответствующие им стобцовые индексы - в массиве `JA`. Кроме того, используется массив указателей (номеров), например, `IA`, отмечающих позиции `AN` и `JA`. с которых начинаются описание очередной строки. Дополнительная компонента в `IA` содержит указатель (номер) первой свободной позиции в `JA` и `AN`.

Существует несколько модификаций данной схемы хранения, наиболее очевидной является разреженная столбцовая схема или **Compressed Sparse Column (CSC)**.

Например, портрет:

$$A = \begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline 1 & & & 1. & 4. & & & & 5. & & \\ 2 & & & & & & & & & & \\ 3 & & & & & & 7. & & 11. & & \end{array}$$

представляется так:

	1	2	3	4	5	6	
AN	**1.**	**4.**	**5.**	**7.**	**11.**		- элементы;
JA	3	4	8	6	8		- соответств.номера столбцов;
IA	1	4	4	6			- с какого элемента начинается описание строки;

например, `4` `4` - указывает на то, что во второй строке элементов нет,

6 - свободная позиция в `JA` и `AN`, т. е., элементов в них `5`.

Фактически, `JA` и `AN` дают портрет матрицы. Это представление матриц называют полным, т. к. представлены все ненулевые элементы, и упорядоченным, т. к. элементы каждой строки упорядочены по возрастанию индексов столбцов.

(Row-Wise Representation complete and Ordered `->` RR(C)O).

Неупорядоченное строчное представление, например такое:

```
1  2  3  4  5
AN 5.  1.  4.  11.  7.
JA 8  3  4  8  6
IA 1  4  4  6
```

Это **Row-Wise Representation complete and Unordered** `->` **RR(C)U** - представление. (№№ строк- упорядочены, описание элементов строки - нет)

Удобно *(конечно, удобно!)* , т.к. результаты большинства матричных операций - неупорядочены, а упорядочивание увеличивает затраты машинного времени.

Аналогично можно хранить номера строк и список (номера) вхождений в столбец.

Сложение разреженных векторов

Разреженный вектор - это разреженная матрица-строка или матрица-столбец.

Предыдущая схема - упакована, т. к. хранятся только ненулевые элементы. Для алгебраических операций удобно во время работы хранить не только полный (расширенный) вектор элементов, но еще хранить и массив JA, чтобы работать только с ненулевыми элементами, таким образом, сокращая перебор всех элементов вектора. Для сложение разреженных векторов используют расширенный накопитель.

Например, есть два разреженных вектора: a и b размером N (у нас -12)

J индекс	1	2	3	4	5	6	7	8	9	10	11	12
a			0.3	-0.7			0.4			0.2		
b				0.7	0.6					0.5		

Их представление будет:

$$\left.\begin{array}{lllll} JA & 10 & 3 & 7 & 4 \\ AN & 0.2 & 0.3 & 0.4 & -0.7 \end{array}\right\} a$$

$$\left.\begin{array}{llll} JB & 5 & 4 & 10 \\ BN & 0.6 & 0.7 & 0.5 \end{array}\right\} b$$

При обычном сложении необходим просмотр всех элементов `JA` и `JB`, сложность - $Q(n)$, где n = количеству элементов обоих массивов - неэффективно, при разреженности.

Для эффективной процедуры сложения разбиваем алгоритм на две части:

- т. н. **символическая часть**, где осуществляется распределение памяти под результат;
- **численная часть**, где осуществляется численная обработка, т. е., реальные вычисления.

В этом случае общее число элементарных операций будет линейной функцией от общего числа ненулевых элементов.

Символический этап определяет позиции ненулевых элементов путем слияния списков (индексов) `JA` и `JB` с помощью расширенного массива переключателей, кот. используется для формирования столбцовых индексов в разреженных матрицах и осуществляет процесс слияния списков.

Массив переключателей `IX` указывает, был ли элемент задействован в списках `JA` и `JB`. Вначале `IX` нулевой.

Т.О. есть 2 списка:

- Список (массив) `JA`: `10` `3` `7` `4`
- Список (массив) `JB` : `5` `4` `10`

Позиции ненулевых эл-тов:

```
- Позиции      1 2 3 4 5 6 7 8 9 10 11 12...N
- значения IX 0 0 1 1 1 0 1 0 0 1        до max в (JA JB)
```

из списка (массива) `JA` - в список `JC`:

Список (массив) `JC` : `10 3 7 4`

Проверяем теперь список `JB`, предварительно сверив по `IX` нужно ли вписывать число в `JC`.

В результате `JC` : `10 3 7 4 5`

Т. е. - это множество значений в списках (объединение списков).

Теперь используем расширенный вещественный накопитель X размерности N, во все ненулевые позиции которого (используя `JC`) зашлем 0:

```
позиция     1 2 3  4  5  6 7  8  9 10 ...
Значение X  x x 0. 0. 0. x 0. x  x  0. x x...
```

Загружаем `AN` в `X`:

```
1  2  3     4  5  6    7  8  9  10 ...
X  x  x  0.3 -0.7  0. x  0.4  x  x 0.2 x x...
```

Теперь прибавляем `BN`:

```
1  2  3  4   5  6    7  8  9  10 ...
X   x  x  0.3 0. 0.6  x  0.4  x  x  0.7 xx...
```

Теперь выбираем из X нужные числа для формирования `CN`:

```
JC    10    3    7  4     5
CN  0.7  0.3  0.4  0.  0.6
```

Количество операций пропорционально числу ненулевых элементов, не считая засылки `N` (`maxN` из `Ja` и `Jb`) нулей в массив `IX`.

Скалярное умножение двух разреженных векторов с использованием массива указателей (номеров):

- Допустим, два разреженных вектора размером N хранятся в массивах `JA`, `AN` и `JB`, `BN`.
- Оба представления компактные и неупорядоченные. Надо вычислить скалярное произведение этих векторов:

$$h = \sum_{i=1}^{N} a_i \cdot b_i$$

Алгоритм следующий:

Для хранения указателей позиций ненулевых элементов в `AN` используется расширенный массив указателей `IP`, который заполняется путем одного просмотра массива `JA` (его начальное состояние - нулевое).

На том же примере: номер индекса в `JA`

```
1  2  3  4  5  6  7  8  9  10  11
IP   0  0  2  4  0  0  3  0  0   1   0
```

номер элемента в `AN`

Отсюда видно, что `a3` хранится в `AN (2)`.

Далее просматриваются массивы `JB` и `BN` и если позиции ненулевык элементов совпадают, то вычисляем `ai * bi` и накапливаем в `h` до тех пор, пока не будет исчерпан `JB`.

У нас:

- `JB(1) = 5`, `IP(5) = 0` - действий нет;
- `JB(2) = 4`, `IP(4) = 4`,

- следоват. смотрим `->` `AN(4) = -0.7`, а `BN(2) = 0.7`,
- и умножаем `->` `BN(2) * AN(4) = -0.49`

Применение этого алгоритма особенно интересно в ситуации, когда вектор `a` нужно скалярно умножить на несколько векторов. В этом случае массив `IP` заполняется только один раз и затем используется для вычисления всех требуемых скалярных произведений. Эта ситуация возникает при необходимости перемножить разреженную матрицу и разреженный вектор.

Задача №3 Обработка разреженных матриц

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

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

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

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

При различной обработке матриц при хранении в разреженном строчном формате Compressed Sparse Row (CSR) или при хранении в разреженном столбцовом формате Compressed Sparse Column (CSC) может возникнуть проблемы к доступу элементов.

Возможность удобного доступа состоит в транспонировании матриц. Сам алгоритм транспонирования работает достаточно быстро.

Другой вариант-для каждой CRS *(CSR?)* матрицы, которая может понадобиться в столбцовом представлении, дополнительно хранить транспонированный портрет. Сэкономив время на транспонировании, придется смириться с расходами времени на поддержание дополнительного портрета в актуальном состоянии.