Matrices

Matrix ➔ table of values

$$\begin{matrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{matrix}$$

Row 2

Column 4

4 x 5 matrix
4 rows
5 columns
20 elements
6 nonzero elements

Two Matrices

$$\begin{bmatrix} -27 & 3 & 4 \\ 6 & 82 & -2 \\ 109 & -64 & 11 \\ 12 & 8 & 9 \\ 48 & 27 & 47 \end{bmatrix} \qquad \begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

Sparse Matrices

Sparse Matrices

Sparse matrix ➔ #nonzero elements/#elements is small.

Examples:

- Diagonal $\begin{matrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{matrix}$
  - Only elements along diagonal may be nonzero
  - n x n matrix → ratio is $n/n^2 = 1/n$
- Tridiagonal
  - Only elements on 3 central diagonals may be nonzero
  - Ratio is $(3n-2)/n^2 = 3/n - 2/n^2$

Sparse Matrices

- Lower triangular $\begin{matrix} 1 & 0 & 0 \\ 2 & 2 & 0 \\ 3 & 3 & 3 \end{matrix}$
  - Only elements on or below diagonal may be nonzero
  - Ratio is $n(n+1)/(2n^2) \sim 0.5$

These are structured sparse matrices. Nonzero elements are in a well-defined portion of the matrix.

Sparse Matrices

An n x n matrix may be stored as an n x n array.

This takes $O(n^2)$ space.

The example structured sparse matrices may be mapped into a 1D array so that a mapping function can be used to locate an element quickly; the space required by the 1D array is less than that required by an n x n array (next lecture).

1	0	0
0	2	0
0	0	3

Diagonal matrix

→ 1 2 3

1-D array

Unstructured Sparse Matrices

Airline flight matrix.

- airports are numbered 1 through n
- flight(i,j) = list of nonstop flights from airport i to airport j
- n = 1000 (say)
- n x n array of list pointers → 4 mega bytes
  - Assume each pointer use 4 bytes.
- total number of nonempty flight lists = 20,000 (say)
- need at most 20,000 list pointers → at most 80,000 bytes

Unstructured Sparse Matrices

Web page matrix.

web pages are numbered 1 through n

web(i,j) = number of links from page i to page j

Web analysis.

authority page … page that has many links to it

Contain useful information about a topic.

hub page … links to many authority pages

The page is basically consisted of links.

Web Page Matrix

- n = 2 billion (and growing by 1 million a day)
- n x n array of ints → $16 * 10^{18}$ bytes ($=4*2*10^9*2*10^9$)$=16 * 10^9$ GB
- each page links to 10 (say) other pages on average
- on average there are 10 nonzero entries per row
- space needed for nonzero elements is approximately 20 billion x 4 bytes = 80 billion bytes (80 GB)

Representation Of Unstructured Sparse Matrices

Single linear list in row-major order.

- scan the nonzero elements of the sparse matrix in row-major order (i.e., scan the rows left to right beginning with row 1 and picking up the nonzero elements)
- each nonzero element is represented by a triple

  (row, column, value)
- the list of triples is stored in a 1D array

Single Linear List Example

```
0 0 3 0 4
0 0 5 7 0
0 0 0 0 0
0 2 6 0 0
```

list =

row	1	1	2	2	4	4
column	3	5	3	4	2	3
value	3	4	5	7	2	6

One Linear List Per Row

```
0 0 3 0 4
0 0 5 7 0
0 0 0 0 0
0 2 6 0 0
```

row1 = [(3, 3), (5,4)]

row2 = [(3,5), (4,7)]

row3 = []

row4 = [(2,2), (3,6)]

Single Linear List

- Class SparseMatrix
  - Array smArray of triples of type MatrixTerm
    - int row, col, value
  - int rows, // number of rows

    cols, // number of columns

    terms, // number of nonzero elements

    capacity; // size of smArray
- Size of smArray generally not predictable at time of initialization.
  - Start with some default capacity/size (say 10)
  - Increase capacity as needed

Approximate Memory Requirements

500 x 500 matrix with 1994 nonzero elements, 4 bytes per element

2D array	500 x 500 x 4 = 1million bytes
Class SparseMatrix	3 x 1994 x 4 + 4 x 4
	= 23,944 bytes

Array Resizing

```
if (newSize < terms) throw “Error”;
MatrixTerm *temp = new MatrixTerm[newSize];
copy(smArray, smArray+terms, temp);
delete [] smArray;
smArray = temp;
capacity = newSize;
```


Array Resizing

- To avoid spending too much overall time resizing arrays, we generally set newSize = c * oldSize, where c >0 is some constant.
- Quite often, we use c = 2 (array doubling) or c = 1.5.
- Now, we can show that the total time spent in resizing is O(s), where s is the maximum number of elements added to smArray.

Matrix Transpose

```
0 0 3 0 4            0 0 0 0
0 0 5 7 0    ---->   0 0 0 2
0 0 0 0 0            3 5 0 6
0 2 6 0 0            0 7 0 0
                     4 0 0 0
```

Matrix Transpose

0	0	3	0	4
0	0	5	7	0
0	0	0	0	0
0	2	6	0	0

→

0	0	0	0
0	0	0	2
3	5	0	6
0	7	0	0
4	0	0	0

row	1	1	2	2	4	4
column	3	5	3	4	2	3
value	3	4	5	7	2	6

The order of Nonzero elements changes →

2	3	3	3	4	5
4	1	2	4	2	1
2	3	5	6	7	4

In Class Exercise: do the transposition and show the Single Linear List

0	2	0	0	4
0	0	0	0	0
4	1	0	0	0
0	0	0	9	5

Matrix Transpose

- Assume m*n matrix with t nonzero elments
- Two algorithms
  - Program 2.10
    - O(nt)
    - Easy to code
  - Program 2.11
    - O(n+t)
    - Hard to think & code

Matrix Transpose Program 2.10

Array

row	1	1	2	2	4	4
column	3	5	3	4	2	3
value	3	4	5	7	2	6

b.Array

2	3	3	3	4	5
4	1	2	4	2	1
2	3	5	6	7	4

```
 B=0;
for(c=0; c<cols;c++)
  {
    for(i=0;i<terms;i++)
     {
        if(Array[i].col==c)
         {
          b.Array[B].row=c;
          b.Array[B].col=Array[i].row;
          b.Array[B].v=Array[i].v;
           B=B+1;
         }
      }
   }
```

Scan n times, each time sacn t elements.

Fast Matrix Transpose
Program 2.11

```
0 0 3 0 4                0 0 0 0
0 0 5 7 0                0 0 0 2
0 0 0 0 0   ------->     3 5 0 6
0 2 6 0 0                0 7 0 0
                         4 0 0 0
```

row	1	1	2	2	4	4
column	3	5	3	4	2	3
value	3	4	5	7	2	6

Step 1: #nonzero in each row of transpose.
= #nonzero in each column of original matrix
= [0, 1, 3, 1, 1]

Step2: Start of each row of transpose
= sum of size of preceding rows of transpose
= [0, 0, 1, 4, 5]

Step 3: Move elements, left to right, from original list to transpose list.

Fast Matrix Transpose

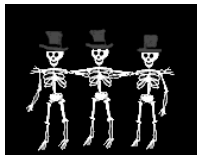
Step 1: #nonzero in each row of transpose.
= #nonzero in each column of original matrix
= [0, 1, 3, 1, 1]

Step2: Start of each row of transpose
= sum of size of preceding rows of transpose
= [0, 0, 1, 4, 5]

Step 3: Move elements, left to right, from original list to transpose list.

Complexity
m x n original matrix
t nonzero elements
Step 1: O(t)
Step 2: O(n)
Step 3: O(t)
Overall O(n+t)

Runtime Performance



Matrix Transpose
500 x 500 matrix with 1994 nonzero elements
Run time measured on a 300MHz Pentium II PC

2D array	210 ms
SparseMatrix (Fast)	6 ms

Homework

- 2.4 Exercise 4 Page 107