Handling Special Matrices

Special Matrices

- Square – same number of rows and columns.
- Some special forms of square matrices are
  - Diagonal: $M(i,j) = 0$ for $i \neq j$
  - Tridiagonal: $M(i,j) = 0$ for $|i-j| > 1$
  - Lower triangular: $M(i,j) = 0$ for $i >= j$
  - Upper triangular: $M(i,j) = 0$ for $i <= j$
  - Symmetric $M(i,j) = M(j,i)$ for all i and j

Contd…

M(i,j) = 0 for i ≠ j

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}$$

Diagonal

M(i,j) = 0 for |i-j| > 1

$$\begin{bmatrix} 2 & 1 & 0 & 0 \\ 3 & 1 & 3 & 0 \\ 0 & 5 & 2 & 7 \\ 0 & 0 & 9 & 0 \end{bmatrix}$$

Tri-Diagonal

M(i,j) = 0 for i < j

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 4 & 2 & 7 & 0 \end{bmatrix}$$

Lower Triangular

$$\begin{bmatrix} 2 & 1 & 3 & 0 \\ 0 & 1 & 3 & 8 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Upper Triangular

$$\begin{bmatrix} 2 & 4 & 6 & 0 \\ 4 & 1 & 9 & 5 \\ 6 & 9 & 4 & 7 \\ 0 & 5 & 7 & 0 \end{bmatrix}$$

Symmetric

M(i,j) = 0 for i > j

M(i,j) = M(j,i) for all i and j

Contd…

- Why are we interested in these "special" matrices?
  - We can provide more efficient implementations for specific special matrices.
  - Rather than having a space complexity of $O(n^2)$, we can find an implementation that is O(n).
  - We need to be clever about the "store" and "retrieve" operations to reduce time.

Diagonal Matrix

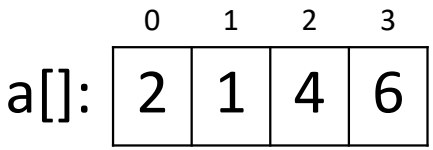
- Naive way to represent n x n diagonal matrix
  - <datatype> d[n][n]
  - d[i][j] for D(i,j)
  - Requires n^2 x sizeof(<datatype>) bytes of memory.

- Better way
  - <datatype> d[n]
  - d[i] for D(i,j) where i = j
    0 for D(i,j) where i ≠ j
  - Requires n x sizeof(<datatype>) bytes of memory.

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}$$

Example

```
1.  #include<stdio.h>
2.  #define MAX 4
3.  int main()
4.  {  int i,j, a[MAX];
5.     printf("\nEnter elements (row major):\n");
6.     for(i = 0; i < MAX; i++)
7.       scanf("%d",&a[i]);
8.     printf("\nThe matrix is...\n");
9.     for(i = 0; i < MAX; i++)
10.    {  for(j = 0; j < MAX; j++)
11.       {    if(i==j)
12.              printf("%d ", a[i]);
13.           else
14.              printf("0 ");    }
15.       printf("\n");  }
16.    return 0;    }
```



Contd…

```
1.  #include<stdio.h>
2.  #define MAX 4
3.  int main()
4.  {  int i,j, a[MAX];
5.     printf("\nEnter elements (row major):\n");
6.     for(i = 0; i < MAX; i++)
7.       scanf("%d",&a[i]);
8.     printf("\nThe matrix is...\n");
9.     for(i = 0; i < MAX; i++)
10.    {  for(j = 0; j < MAX; j++)
11.       {    if(i==j)
12.              printf("%d ", a[i]);
13.            else
14.              printf("0 ");    }
15.       printf("\n");  }
16.    return 0;    }
```

a[]:

0	1	2	3
2	1	4	6

	0	1	2	3
0	2	0	0	0
1	0	1	0	0
2	0	0	4	0
3	0	0	0	6

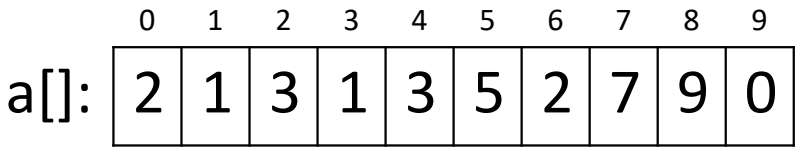
Tridiagonal Matrix

- Nonzero elements lie on one of three diagonals:
  - main diagonal: i = j
  - diagonal below main diagonal: i = j+1
  - diagonal above main diagonal: i = j-1
- Total elements are 3n – 2: <datatype> d[3n-2]
- Mappings
  - by row [2,1,3,1,3,5,2,7,9,0]
  - by column [2,3,1,1,5,3,2,9,7,0]
  - by diagonal [3,5,9,2,1,2,0,1,3,7]

$$\begin{bmatrix} 2 & 1 & 0 & 0 \\ 3 & 1 & 3 & 0 \\ 0 & 5 & 2 & 7 \\ 0 & 0 & 9 & 0 \end{bmatrix}$$

Example

```
1.  #include<stdio.h>
2.  #define MAX 4
3.  int main()
4.  {  int i,j,k=0, size = 3*MAX-2, a[size];
5.     printf("\nEnter elements (row major):\n");
6.     for(i = 0; i < size; i++)
7.       scanf("%d",&a[i]);
8.     printf("\nThe matrix is...\n");
9.     for(i = 0; i < MAX; i++)
10.    {  for(j = 0; j < MAX; j++)
11.       {    if(i-j == -1 || i-j == 0 || i-j == 1)
12.             {  printf("%d ", a[k]);  k++;  }
13.           else
14.              printf("0 ");}
15.       printf("\n");  }
16.    return 0;   }
```



Example

```
1.  #include<stdio.h>
2.  #define MAX 4
3.  int main()
4.  {  int i,j,k=0, size = 3*MAX-2, a[size];
5.     printf("\nEnter elements (row major):\n");
6.     for(i = 0; i < size; i++)
7.       scanf("%d",&a[i]);
8.     printf("\nThe matrix is...\n");
9.     for(i = 0; i < MAX; i++)
10.    {  for(j = 0; j < MAX; j++)
11.      {    if(i-j == -1 || i-j == 0 || i-j == 1)
12.             {  printf("%d ", a[k]);  k++;  }
13.           else
14.              printf("0 ");}
15.      printf("\n");  }
16.    return 0;   }
```

a[]:

0	1	2	3	4	5	6	7	8	9
2	1	3	1	3	5	2	7	9	0

	0	1	2	3
0	2	1	0	0
1	3	1	3	0
2	0	5	2	7
3	0	0	9	0

Triangular Matrix

- Nonzero elements lie in the upper triangular or lower triangular region.
- Total elements are 1 + 2 + … + n = n(n+1)/2:

  <datatype> d[(n(n+1)/2)]
- Mappings
  - by row
  - by column

[2,5,3,1,1,2,4,7,0]

$$\begin{bmatrix} 2 & 5 & 3 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 4 & 7 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Upper Triangular

[2,5,0,1,3,1,4,2,7,0]

[2,5,0,4,1,3,2,1,7,0]

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 4 & 2 & 7 & 0 \end{bmatrix}$$

Lower Triangular

Example

```
1.  #include<stdio.h>
2.  #define MAX 4
3.  int main()
4.  {  int i,j,k=0, size = (MAX*(MAX+1))/2, a[size];
5.     printf("\nEnter elements (row major):\n");
6.     for(i = 0; i < size; i++)
7.       scanf("%d",&a[i]);
8.     printf("\nThe upper triangular matrix is...\n");
9.     for(i = 0; i < MAX; i++)
10.    {  for(j = 0; j < MAX; j++)
11.       {    if(i <= j)
12.             {  printf("%d ", a[k]);  k++;  }
13.           else
14.             printf("0 ");       }
15.       printf("\n");     }
16.    k = 0;
17.    printf("\nThe lower triangular matrix is...\n");
18.    for(i = 0; i < MAX; i++)
19.      {  for(j = 0; j < MAX; j++)
20.         {    if(i >= j)
21.               {  printf("%d ", a[k]);  k+
       +;  }
22.            else
23.              printf("0 ");       }
24.         printf("\n");     }
25.      return 0;      }
```

	0	1	2	3	4	5	6	7	8	9
a[]:	2	5	1	0	3	1	4	2	7	0

Example

```
1.  #include<stdio.h>
2.  #define MAX 4
3.  int main()
4.  {  int i,j,k=0, size = (MAX*(MAX+1))/2, a[size];
5.     printf("\nEnter elements (row major):\n");
6.     for(i = 0; i < size; i++)
7.       scanf("%d",&a[i]);
8.     printf("\nThe upper triangular matrix is...\n");
9.     for(i = 0; i < MAX; i++)
10.    {  for(j = 0; j < MAX; j++)
11.      {   if(i <= j)
12.           {  printf("%d ", a[k]);  k++;  }
13.         else
14.           printf("0 ");       }
15.     printf("\n");     }
16.    k = 0;
17.    printf("\nThe lower triangular matrix is...\n");
18.    for(i = 0; i < MAX; i++)
19.      {  for(j = 0; j < MAX; j++)
20.        {   if(i >= j)
21.             {  printf("%d ", a[k]);  k++;  }
22.           else
23.             printf("0 ");       }
24.       printf("\n");     }
25.    return 0;      }
```

	0	1	2	3	4	5	6	7	8	9
a[]:	2	5	1	0	3	1	4	2	7	0

	0	1	2	3
0	2	5	1	0
1	0	3	1	4
2	0	0	2	7
3	0	0	0	0

Example

```
1.  #include<stdio.h>
2.  #define MAX 4
3.  int main()
4.  {  int i,j,k=0, size = (MAX*(MAX+1))/2, a[size];
5.     printf("\nEnter elements (row major):\n");
6.     for(i = 0; i < size; i++)
7.       scanf("%d",&a[i]);
8.     printf("\nThe upper triangular matrix is...\n");
9.     for(i = 0; i < MAX; i++)
10.    {  for(j = 0; j < MAX; j++)
11.       {    if(i <= j)
12.            {  printf("%d ", a[k]);  k++;  }
13.          else
14.            printf("0 ");       }
15.       printf("\n");     }
16.    k = 0;
17.    printf("\nThe lower triangular matrix is...\n");
18.    for(i = 0; i < MAX; i++)
19.      {  for(j = 0; j < MAX; j++)
20.         {    if(i >= j)
21.              {  printf("%d ", a[k]);  k++; }
22.            else
23.              printf("0 ");       }
24.         printf("\n");     }
25.    return 0;      }
```

	0	1	2	3	4	5	6	7	8	9
a[]:	2	5	1	0	3	1	4	2	7	0

	0	1	2	3
0	2	0	0	0
1	5	1	0	0
2	0	3	1	0
3	4	2	7	0

Symmetric Matrix

- An n x n matrix can be represented using 1-D array of size n(n+1)/2 by storing either the lower or upper triangle of the matrix.

$$\begin{bmatrix} 2 & 4 & 6 & 0 \\ 4 & 1 & 9 & 5 \\ 6 & 9 & 4 & 7 \\ 0 & 5 & 7 & 0 \end{bmatrix}$$

- Use one of the methods for a triangular matrix.
- The elements that are not explicitly stored may be computed from those that are stored.

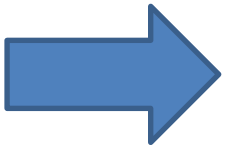
Sparse Matrix

- A matrix is sparse if many of its elements are zero.
- A matrix that is not sparse is dense.
- Two possible representations
  - Array (also known as triplet)
  - Linked list

$$\begin{bmatrix} 0 & 0 & 0 & 2 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 9 \\ 0 & 5 & 4 & 0 \end{bmatrix}$$

Array representation

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	15	0	0	22	0	-15
[1]	0	11	3	0	0	0
[2]	0	0	0	-6	0	0
[3]	0	0	0	0	0	0
[4]	91	0	0	0	0	0
[5]	0	0	28	0	0	0



Row	Col	Value
6	6	8
0	0	15
0	3	22
0	5	-15
1	1	11
1	2	3
2	3	-6
4	0	91
5	2	28

Operations

- Transpose
- Addition
- Multiplication

Transpose

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	15	0	0	22	0	-15
[1]	0	11	3	0	0	0
[2]	0	0	0	-6	0	0
[3]	0	0	0	0	0	0
[4]	91	0	0	0	0	0
[5]	0	0	28	0	0	0

Row	Col	Value
6	6	8
0	0	15
0	3	22
0	5	-15
1	1	11
1	2	3
2	3	-6
4	0	91
5	2	28

Original

Row	Col	Value
6	6	8
0	0	15
3	0	22
5	0	-15
1	1	11
2	1	3
3	2	-6
0	4	91
2	5	28

Column Major

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	15	0	0	0	91	0
[1]	0	11	0	0	0	0
[2]	0	3	0	0	0	28
[3]	22	0	-6	0	0	0
[4]	0	0	0	0	0	0
[5]	-15	0	0	0	0	0

Row	Col	Value
6	6	8
0	0	15
0	4	91
1	1	11
2	1	3
2	5	28
3	0	22
3	2	-6
5	0	-15

Row Major

Addition

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	15	0	0	22	0	-15
[1]	0	11	3	0	0	0
[2]	0	0	0	-6	0	0
[3]	0	0	0	0	0	0
[4]	91	0	0	0	0	0
[5]	0	0	28	0	0	0

+

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	15	0	0	0	91	0
[1]	0	11	0	0	0	0
[2]	0	3	0	0	0	28
[3]	22	0	-6	0	0	0
[4]	0	0	0	0	0	0
[5]	-15	0	0	0	0	0

=

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	30	0	0	22	91	-15
[1]	0	22	3	0	0	0
[2]	0	3	0	-6	0	28
[3]	22	0	-6	0	0	0
[4]	91	0	0	0	0	0
[5]	-15	0	28	0	0	0

Addition

Counter = 14

Row	Col	Value
6	6	8
0	0	15
0	3	22
0	5	-15
1	1	11
1	2	3
2	3	-6
4	0	91
5	2	28

Row	Col	Value
6	6	8
0	0	15
0	4	91
1	1	11
2	1	3
2	5	28
3	0	22
3	2	-6
5	0	-15

Row	Col	Value
6	6	14
0	0	30
0	3	22
0	4	91
0	5	-15
1	1	22
1	2	3
2	1	3
2	3	-6
2	5	28
3	0	22
3	2	-6
4	0	91
5	0	-15
5	2	28

Multiplication

- Compute A x B
- First take transpose of B.
- Multiply only if the corresponding elements are present and add them for each position in the resultant matrix.

Multiplication

	[0]	[1]	[2]	[3]
[0]	0	10	0	12
[1]	0	0	0	0
[2]	0	0	5	0
[3]	15	12	0	0

×

	[0]	[1]	[2]	[3]
[0]	0	0	8	0
[1]	0	0	0	23
[2]	0	0	9	0
[3]	20	25	0	0

=

	[0]	[1]	[2]	[3]
[0]	240	300	0	230
[1]	0	0	0	0
[2]	0	0	45	0
[3]	0	0	120	276

	[0]	[1]	[2]	[3]
[0]	0	10	0	12
[1]	0	0	0	0
[2]	0	0	5	0
[3]	15	12	0	0

×

	[0]	[1]	[2]	[3]
[0]	0	0	0	20
[1]	0	0	0	25
[2]	8	0	9	0
[3]	0	23	0	0

=

	[0]	[1]	[2]	[3]
[0]	240	300	0	230
[1]	0	0	0	0
[2]	0	0	45	0
[3]	0	0	120	276

Multiplication

B'

Counter = 0

Row	Col	Value
4	4	5
0	1	10
0	3	12
2	2	5
3	0	15
3	1	12

A

Row	Col	Value
4	4	5
0	3	20
1	3	25
2	0	8
2	2	9
3	1	23

B

Row	Col	Value
4	4	5
0	2	8
1	3	23
2	2	9
3	0	20
3	1	25

Row	Col	Value
4	4	

Multiplication

Counter = 6

Row	Col	Value
4	4	5
0	1	10
0	3	12
2	2	5
3	0	15
3	1	12

Row	Col	Value
4	4	5
0	3	20
1	3	25
2	0	8
2	2	9
3	1	23

Row	Col	Value
4	4	6
0	0	240
0	1	300
0	3	230
2	2	45
3	2	120
3	3	276

	[0]	[1]	[2]	[3]
[0]	240	300	0	230
[1]	0	0	0	0
[2]	0	0	45	0
[3]	0	0	120	276

- **Worst case time complexity:** Addition operation traverses the matrices linearly, hence, has a time complexity of O(n), where n is the number of non-zero elements in the larger matrix amongst the two.
- Transpose has a time complexity of O(n+m), where n is the number of columns and m is the number of non-zero elements in the matrix.
- Multiplication, however, has a time complexity of O(x*n + y*m), where (x, m) is number of columns and terms in the second matrix; and (y, n) is number of rows and terms in the first matrix.