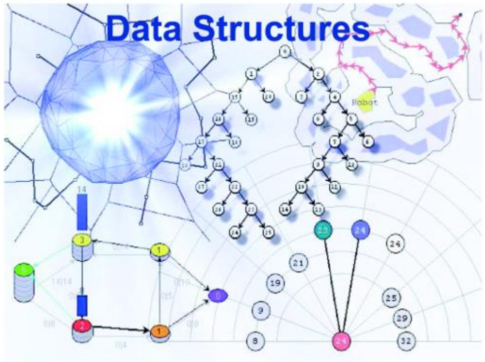
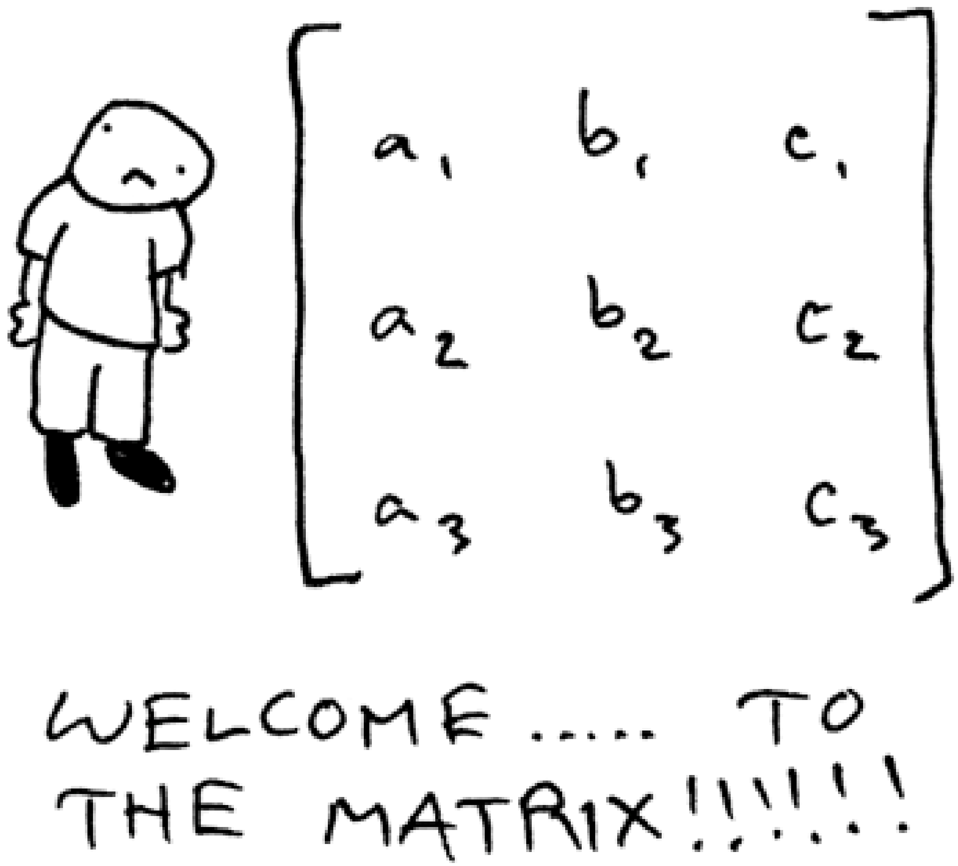
BBM 201
DATA STRUCTURES

Lecture 4:
Lower/Upper Triangular Matrix
Band Matrix
Sparse Matrix



2018-2019 Fall


a_1 b_1 c_1
a_2 b_2 c_2
a_3 b_3 c_3
WELCOME ..... TO THE MATRIX!!!!!!

Lower Triangular Matrix

Triangular matrix

Upper triangular matrix

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & . & . & . & u_{1n} \\ 0 & u_{22} & u_{23} & & & & . \\ 0 & 0 & u_{33} & & & & . \\ . & & & . & & & . \\ . & & & & . & & . \\ . & & & & & . & . \\ 0 & . & . & . & . & 0 & u_{nn} \end{bmatrix}$$

Lower triangular matrix

$$L = \begin{bmatrix} l_{11} & 0 & 0 & . & . & . & 0 \\ l_{21} & l_{22} & 0 & & & & . \\ l_{31} & l_{32} & l_{33} & & & & . \\ . & & & . & & & . \\ . & & & & . & & . \\ . & & & & & . & 0 \\ l_{n1} & . & . & . & . & . & l_{nn} \end{bmatrix}$$

Lower Triangular Matrix

- Does the definition of a special data structure for triangular matrix provide any benefits over a typical matrix in terms of memory and processing time?
- We can insert the items in a single dimensional array:
- ***ALT***

a_{00}	a_{10}	a_{11}	a_{20}	a_{21}	a_{22}	a_{30}	a_{31}	a_{32}	a_{33}
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

- Number of items in the array becomes:

$$1+2+\ldots+(n-1)+n=\frac{n(n+1)}{2}$$

$$a=\begin{bmatrix} a_{00} & 0 & 0 & 0 \\ a_{10} & a_{11} & 0 & 0 \\ a_{20} & a_{21} & a_{22} & 0 \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Lower Triangular Matrix

- How can we find the position of u[i][j] in the array?
  - Answer: i=1, there is one item in the 0th row, 2 items in the 1st row.
  - i=2, there is one item in the 0th row, 2 items in the 1st row, 3 items in the 2nd row.

  - Therefore the address of u[i][j] in the array is calculated as below:

$$k = \sum_{t=0}^{i} (t) + (j) = (0 + 1 + 2 + \ldots + i) + (j)$$

$$= \frac{i(i+1)}{2} + (j)$$

Lower Triangular Matrix

```
void main(void){
   int alt[MAX_SIZE];
   int i, n;
   scanf(“%d“, &n); //matrix size
   readtriangularmatrix(alt,n);
   for(i=0; i<=n*(n+1)/2-1; i++)
      printf(“ %d“, alt[i]);

   i=gettriangularmatrix(3,0,n);
   if(i==-2)
      printf(“\n invalid index\n“);
   else if(i==-1)
      printf(“\n access to the upper triangular\n“);
   else
      printf(“\n  the position in ‘alt’ matrix: %d value: %d \n“, i, alt[i]);
```

Lower Triangular Matrix

```
void readtriangularmatrix(int alt[], int n)
{
     int i, j, k;
     if(n*(n+1)/2 > MAX_SIZE){
        printf("\n invalid array size \n");
        exit(-1);
     }
     else
       for(i=0; i<=n-1; i++){
          k=(i+1)*i/2;
          for(j=0; j<=i; j++)
               scanf("%d", &alt[k+j]);
       }
}
```

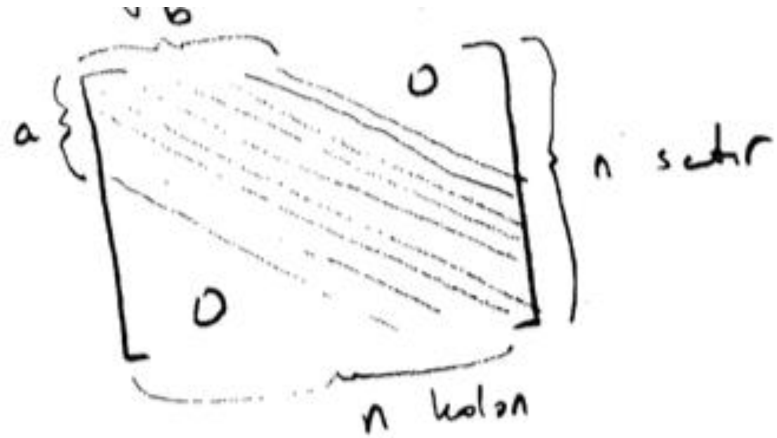
```
int gettriangularmatrix(int i, int j, int n){

   if(i<0 || i>=n || j<0 || j>=n){
         printf("\n invalid index\n");
         exit(-2);
    }
    else if(i>=j) //valid index
         return (i+1)*i/2+j;
    else return -1; //outside of the triangular
                          //value is zero
```

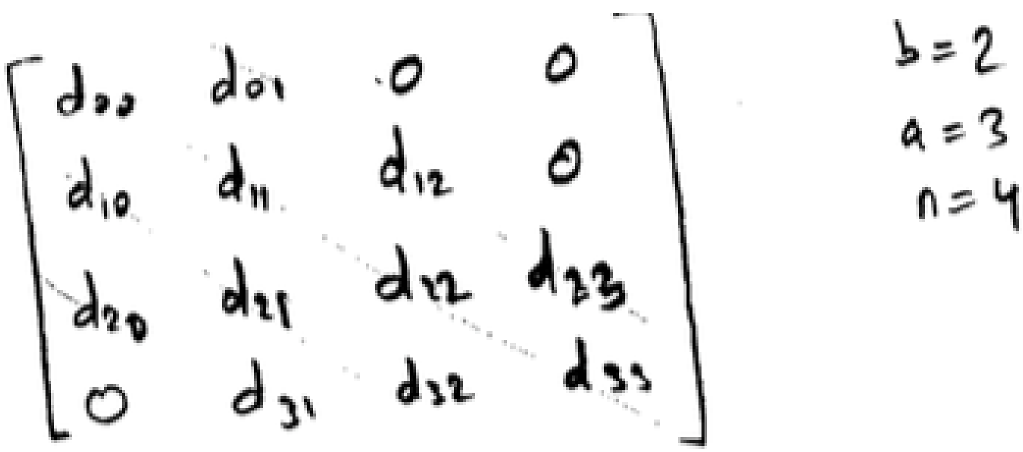
Band Matrix

$$\begin{bmatrix} B_{11} & B_{12} & 0 & \cdots & \cdots & 0 \\ B_{21} & B_{22} & B_{23} & \ddots & \ddots & \vdots \\ 0 & B_{32} & B_{33} & B_{34} & \ddots & \vdots \\ \vdots & \ddots & B_{43} & B_{44} & B_{45} & 0 \\ \vdots & \ddots & \ddots & B_{54} & B_{55} & B_{56} \\ 0 & \cdots & \cdots & 0 & B_{65} & B_{66} \end{bmatrix}$$

Matrix (n, a) : n by n matrix, non-zero entries are confined to a diagonal band, comprising the main diagonal and zero or more diagonals (a-1) on either side.

Band Matrix





Band Matrix

- What kind of a data structure can we use?

- We can insert the items in a single dimensional array:

- ***ALT***

a_{20}	a_{31}	a_{10}	a_{21}	a_{32}	a_{00}	a_{11}	a_{22}	a_{32}	a_{33}	a_{01}	a_{12}	a_{23}

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]

- What is the number of items in the array?

Band Matrix

- What is the number of items in the array?
  - Number of items **on** and **below** the diagonal:

$$n+(n-1)+(n-2)+...+n-(a-1)$$

  - Number of items **above** the diagonal:

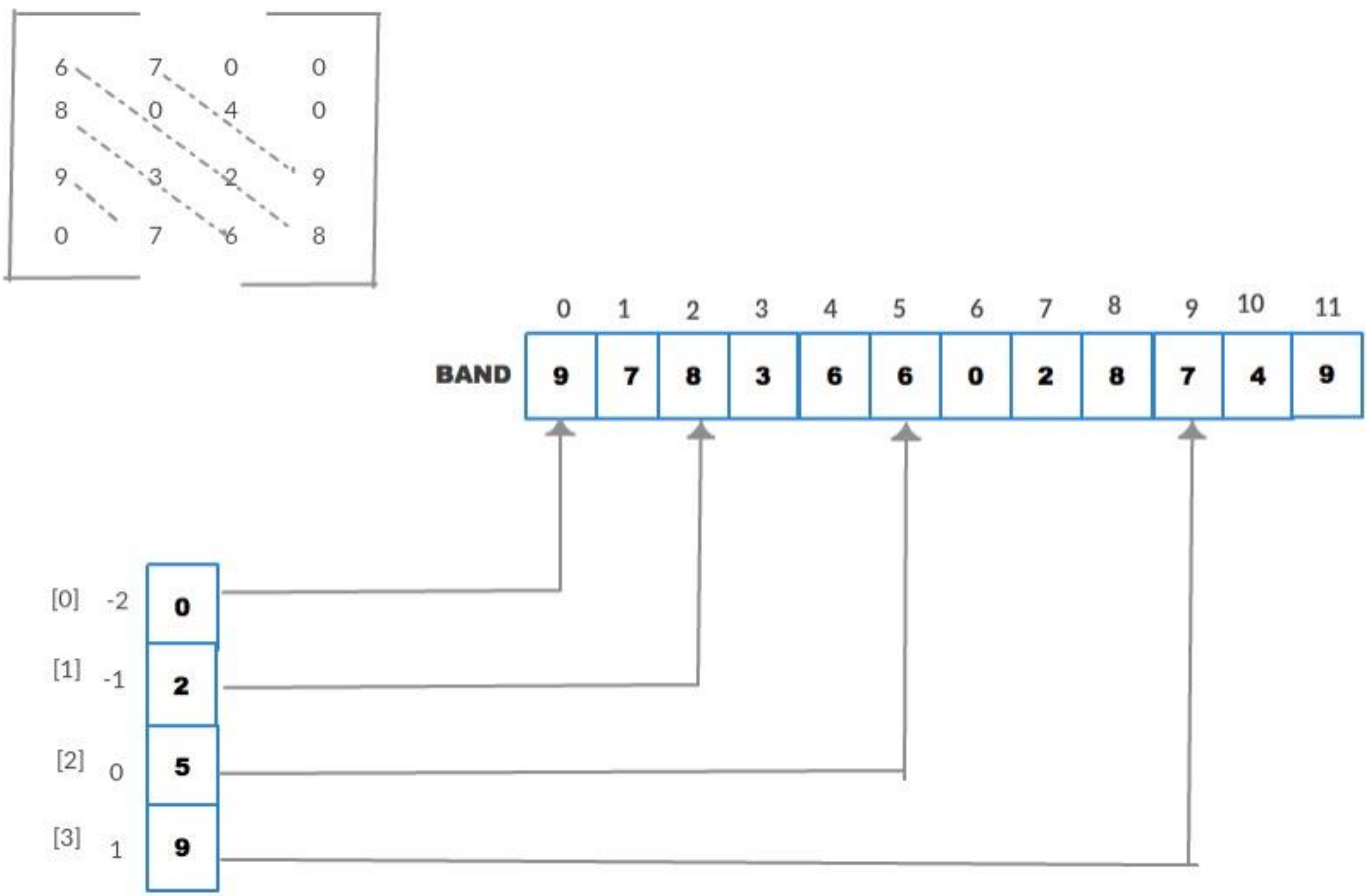
$$(n-1)+(n-2)+...+n-(b-1)$$

  - Sum of these becomes:

$$Sum=n+(n-1)+(n-2)+...+n-(a-1)+(n-1)+(n-2)+...+n-(b-1)$$

$$=n(a+b-1)-\frac{(a-1)a}{2}-\frac{(b-1)b}{2}$$

Band Matrix



6 7 0 0
8 0 4 0
9 3 2 9
0 7 6 8
0 1 2 3 4 5 6 7 8 9 10 11
BAND
9 7 8 3 6 6 0 2 8 7 4 9
[0] -2 0
[1] -1 2
[2] 0 5
[3] 1 9

Band Matrix

```
void main(void){
    int band[MAX_SIZE];
    int search[MAX_SIZE];

    int i, n, a, b;
    printf(" n:", &n); scanf("%d", &n);
    printf(" a:", &a); scanf("%d", &a);
    printf(" b:", &b); scanf("%d", &b);

    buildbandmatrix(band,search,a,b);

    for(i=0; i<=n*(a+b-1)-a*(a-1)/2-b*(b-1)/2-1; i++)
        printf(" %d ",band[i]);

    printf("\n");
    for(i=0; i<=a+b-2; i++)
        printf(" %d", search[i]);

    i=getbandmatrix(3,3,n,a,b,search);

    if(i==2)
        printf("\n invalid index");
    else if(i==1)
        printf("\n item to be searched: 0");
    else
        printf("\n item to be searched: %d→%d", i, band[i]);
```

Band Matrix

```
void buildbandmatrix(int band[], int search[], int n, int a, int b){

    int i, k, itemnum;

    if(n*(a+b-1)-a*(a-1)/2-b*(b-1)/2 > MAX_SIZE){

        printf(“\n not enough memory“);
        exit(-1);
    }
    else{
        itemnum=0;
        for(i=-a+1; i<=b-1; i++){ //for each diagonal
            search[i+a-1]=itemnum;

            for(k=0; k<= n-abs(i)-1; k++)   //for the current diagonal
                scanf(“%d“, &band[search[i+a-1]+k]);
            itemnum = itemnum+(n-abs(i));
        }
    }
}
```

Band Matrix

```
void getbandmatrix(int i, int j, int n, int a, int b, int search[]){

    if(i>=n || i<0 || j>=n || j<0){  //index overflow
        printf("\n invalid index\n");
        return -2;
    }
    else{
        if(j>i)   //above the diagonal
            if(j-i<b)  //above the upper band
                return(search[a-1+j-i]+i);  //yes
            else  //no
                return -1;
        else if(i-j<a) //below or on the diagonal
                return(search[j-i+a-1]+j);
        else //not on the band
                return -1;
}
```

Sparse Matrix

- Most of the elements are zero.
- It wastes space.

Sparsity: the fraction of zero elements.

<u>Basic matrix operations:</u>

1. Creation
2. Addition
3. Multiplication
4. Transpose

A	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

Sparse Matrix
Data Structure

```
#define MAX_TERMS 101
typedef struct{
            int col;
            int row;
            int value;
           }term;
term a[MAX_TERMS];
```

- a[0].row: row index
- a[0].col: column index
- a[0].value: number of items in the sparse matrix

Rows and columns are in ascending order!

Sparse Matrix

A	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	Column	Value
A[0]	6	6	8
A[1]	0	0	15
A[2]	0	3	22
A[3]	0	5	-15
A[4]	1	1	11
A[5]	1	2	3
...			
A[8]	5	2	28

Matrix Transpose

- Replacement of rows and columns in a matrix is called the transpose of the matrix:

$$A = \begin{bmatrix} 1 & 3 \\ 0 & 4 \end{bmatrix} \qquad A' = \begin{bmatrix} 1 & 0 \\ 3 & 4 \end{bmatrix}$$

- The item a[i][j] becomes a[j][i].

Matrix Transpose

```
void transpose(term a[],term b[])
{
   int n,i,j,currentb;
   n=a[0].value;  //number of items
   b[0].row=a[0].col;   //number of rows
   b[0].col=a[0].row;   //number of columns
   b[0].value=n;

   if(n>0){
      currentb=1;
      for(i=0; i<a[0].col; i++)
          for(j=1; j<=n; j++) //find the ones with col i in a
              if(a[j].col==i){
                     b[currentb].row=a[j].col;
                     b[currentb].col=a[j].row;
                     b[currentb].value=a[j].value;
                     currentb++;
               }
    }
}
```

Question: What is the complexity of this method?

Fast Transpose

```
#define MAX_TERM 101
typedef struct{
            int row;
            int col;
            int value;
    } term;
term a[MAX_TERM];


void fastTranspose(term a[], term b[])
{

   int ItemNum[MAX_COL], StartPos[MAX_COL];
   int i,j,ColNum=a[0].col,TermNum=a[0].value;
   b[0].value=TermNum;
   if(TermNum>0){ //does the item exist?
         for(i=0;i<ColNum;i++)
              ItemNum[i]=0;
         for(i=1;i<=TermNum;i++)
              ItemNum[a[i].col]++;
         StartPos[0]=1;
         for(i=1;i<ColNum;i++)
               StartPos[i]=StartPos[i-1]+ItemNum[i-1];
         for(i=1;i<=TermNum;i++){
              j=StartPos[a[i].col]++;
              b[j].row=a[i].col; b[j].col=a[i].row;
              b[j].value=a[i].value;
         }
     }
}
```

Fast Transpose

- Execute the fastTranspose method.
- **Question:** What is the complexity of the method?
- Compare its complexity with the previous transpose method.