Data Structures (CS 21001)

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

School Of Computer Engineering

Dr. Pradeep Kumar Mallick
Associate Professor [II]
School of Computer Engineering,
Kalinga Institute of Industrial Technology (KIIT),
Deemed to be University,Odisha

4 Credit

Lecture Note 01

Chapter Contents

- Sparse Matrix
- Polynomial

Sparse Matrix

- A matrix is a two-dimensional data object made of *m* rows and *n* columns having total m × n values.
- If most of the elements of the matrix have 0 value, then it is called a sparse matrix.

- Why to use Sparse Matrix instead of simple matrix?
  – Storage: less memory used to store only those non-zero elements.
  – reduces scanning time.
  – Computing time: Computing time can be reduced by logically designing a data structure traversing only non-zero elements.

Sparse Matrix

- **Problem**

  Check whether a matrix is a sparse matrix or not.

- **Solution**
  - Let us assume ZERO in the matrix is greater than (row * column)/2.
  - Then, the matrix is a sparse matrix otherwise not.

[Link: https://www.youtube.com/watch?v=iqZKBptJV2U](https://www.youtube.com/watch?v=iqZKBptJV2U)

Sparse Matrix

$$\begin{array}{c} \\ \text{row 1} \\ \text{row 2} \\ \text{row 3} \\ \text{row 4} \\ \text{row 5} \end{array} \begin{array}{c} \begin{array}{ccc} \text{col 1} & \text{col 2} & \text{col 3} \end{array} \\ \left[\begin{array}{ccc} -27 & 3 & 4 \\ 6 & 82 & -2 \\ 109 & -64 & 11 \\ 12 & 8 & 9 \\ 48 & 27 & 47 \end{array}\right] \end{array} \quad 5\times 3$$

(a) 15/15

$$\begin{array}{c} \\ \text{row0} \\ \text{row1} \\ \text{row2} \\ \text{row3} \\ \text{row4} \\ \text{row5} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{col0} & \text{col1} & \text{col2} & \text{col3} & \text{col4} & \text{col5} \end{array} \\ \left[\begin{array}{cccccc} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{array}\right] \end{array} \quad 6\times 6$$

(b) 8/36

Two matrices

sparse matrix

data structure?

Sparse Matrix: Array Representation

- Represented by a two-dimensional array.
- Sparse matrix wastes space
- Each element is characterized by <row, col, value>.

	row	col	value
	# of rows (columns)		# of nonzero terms
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8]	5	2	28

row, column in ascending order

Whether the given matrix is sparse matrix or not

Checking Sparse Matrix

```
#include<stdio.h>
#include<stdlib.h>
int main()
{   int row,col,i,j,a[10][10],count = 0;
 printf("Enter row ");
 scanf("%d",&row);
 printf("Enter Column ");
 scanf("%d",&col);
printf("Enter Element of Matrix1");
  for(i = 0; i < row; i++)
{
        for(j = 0; j < col; j++)
         {

               scanf("%d",&a[i][j]);
        }
  }
// add code to display the elements
```

```
/*checking sparse of matrix*/
for(i = 0; i < row; i++)
{
  for(j = 0; j < col; j++)
   {
         if(a[i][j] == 0)
         count++;
     }
}
if(count > ((row * col)/2))
     printf("Matrix is a sparse matrix ");

else
     printf("Matrix is not sparse matrix ");
}
```

Sparse Matrix Representation


```
#include<stdio.h>
int main() {
    // Assume 4x5 sparse matrix
    int smat[4][5] =
    {  {0 , 0 , 3 , 0 , 4 },
        {0 , 0 , 5 , 7 , 0 },
        {0 , 0 , 0 , 0 , 0 },
        {0 , 2 , 6 , 0 , 0 }
    };
    int i, j, k, size = 0;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 5; j++)
            if (smat[i][j] != 0)
                size++;
    int sm[size+1][3];
      k = 0;
   sm[k][0] = 4; sm[k][1] = 5; sm[k][2] = size;
   k++;
```

```
for (i = 0; i < 4; i++)
{
        for (j = 0; j < 5; j++)
        {
              if (smat[i][j] != 0)
              {
                sm[k][0] = i;  sm[k][1] = j;
                sm[k][2] = smat[i][j];   k++;
                }
          }
     }
     for (int i=0; i<=size; i++) {
              for (int j=0; j<3; j++)
               {
                      printf("%d ", sm[i][j]);
               }
            printf("\n");
          }

    return 0;
}
```

Output

```
4 5 6
0 2 3
0 4 4
1 2 5
1 3 7
3 1 2
3 2 6
```

Sparse Matrix Addition

Step-1 : Obtain the triplet form of both sparce matrices.
Steep-2: Create a new triplet to store result as result matrix .
Step-3 : Copy number of rows and columns from any sparce matrix to result matrix (because both size is equal)
Step-4: Let i, j, and k be the indices of sparce matriceces-1,2 ,3 respectively .
Step-5: Intialize i, j, k to 1
Step-6: Travrse both matrices from second row .

```
if (row number of matrix-1 == row number of matrix-2)
{
    if(column number of matrix-1 == columnnumber of matrix-2)
    {
        Make the addtion of non zero values and store in to result matrix by
    incrementing all indices.
    }
    Else
    {
        Whichever has less column value copy that to result matrix by
    incrementing respective indices.
    }
}
```

Sparse Matrix Addition

Else
{
Compare row of both sparce matrices and which ever has less row value copy that to result matrix by incrementing respective indices.
}

Step-7: Repeat step-6 till the end of any matrix triplet.
Step-8 : Copy the remaining term of sparce matrix (if any) to result matrix.

Sparse Matrix Addition

Matrix-1

Row	Column	Value
2	3	4
0	0	5
0	1	6
1	0	8
1	1	5

Matrix-2

Row	Column	Value
2	3	4
0	1	4
0	2	3
1	1	3
1	2	6

Result Matrix

Row	Column	Value
2	3	6
0	0	5
0	1	10
0	2	3
1	0	8
1	1	8
1	2	6

Sparse Matrix Transpose

12

Step-1 : Obtain the triplet form of sparse matrices.
Steep-2: Traverse the triplet from second row and consider column elements.
Stpe-3: Check if column number is zero(0) then swap its row and column and add it into transpose matrix.
Step-4: Repeat above step for all rows.
Step-5: Repeat step-3 and Step-4 column values for 1,2,3…. (total number of columns.)

Triplet Form

Row	Column	Value
3	2	4
0	1	3
1	0	2
1	1	5
2	0	8

Transpose Matrix Form

Row	Column	Value
2	3	4
0	1	2
0	2	8
1	0	3
1	1	5

Sparse Matrix Transpose


```
transpose()
{
    int transpose_m[10][3], k=1;
    for(int z=0; z<sparse[0][1]; z++)
    {
        for(int i=1; i<=sparse[0][2];i++)
        {
            if(sparse[i][1]==z)
            {
                transpose_m[k][0]= sparse[i][1];
                transpose_m[k][1]= sparse[i][0];
                transpose_m[k][2]= sparse[i][2];

                k++;
            }
        }
}
```

Polynomials

A single variable polynomial p(x) = $4x^6 + 10x^4 - 5x + 3$

Remark: order of this polynomial is 6 (highest exponent)

- Representing Polynomials
  - In general, the polynomial are represented as:

- $A(x) = a_{m-1}x^{e_{m-1}} + \cdots + a_0 x^{e_0}$ coefficients and the e_i are nonnegative integer exponents such that

$$e_{m-1} > e_{m-2} > \ldots > e_1 > e_0 \geqq 0 \ .$$

Polynomial

How to implement this?

There are different ways of implementing the polynomial ADT:

- Array (not recommended)
- Double Array (inefficient)
- Array of Structure (inefficient)
- Linked List (preferred and recommended)

Polynomial

Array Implementation:

$p_1(x) = 8x^3 + 3x^2 + 2x + 6$

$p_2(x) = 23x^4 + 18x - 3$

$p_1(x)$

0	1	2	3
6	2	3	8

$p_2(x)$

0	1	2	3	4
-3	18	0	0	23

0 2 0 2 4

Index represents exponents

Polynomial

This is why arrays are not good to represent polynomials:

$p_3(x) = 16x^{21} - 3x^5 + 2x + 6$

0	1	2	3	4	5	. . .	20	21
6	2	0	0	0	-3	. . .	0	16

WASTE OF SPACE!

Polynomial

- Advantages of using an Array
  - good for non-sparse polynomials.
  - easy to store and retrieve.

- Disadvantages of using an Array:
  - Allocate array size ahead of time.
  - huge array size required for sparse polynomials. Waste of space and runtime.

Polynomial Addition

polynomial 1 = 2.0+ 3.0x^1+ 5.0x^2+ 1.0x^3
polynomial 2 = 7.0+ 8.0x^1+ 5.0x^2

```
#include<stdio.h>
#include<math.h>

float a[50], b[50], c[50];
int main() {
   int i;
   int deg1, deg2;
   int m=0;
   printf("Enter the highest degree of polynomial1: ");
   scanf("%d", &deg1);
   for(i=0; i<=deg1; i++)  {
       printf("\nEnter the coeff of x^%d :", i);
       scanf("%f", &a[i]);
  }
```

Polynomial Addition


```
printf("\nEnter the highest degree of polynomial2: ");
 scanf("%d",  &deg2);
 for(i=0; i<=deg2; i++)  {
     printf("\nEnter the coeff of x^%d : ", i);
     scanf("%f", &b[i]);
 }
 printf("\nPolynomial 1 = %.1f", a[0]);
 for(i=1; i<=deg1; i++)
      printf("+ %.1fx^%d", a[i], i);
 printf("\nPolynomial 2 = %.1f", b[0]);
 for(i=1; i<=deg2; i++)
      printf("+ %.1fx^%d",b[i], i);
```

Polynomial Addition


```
if(deg1>deg2) {
  for(i=0; i<=deg2; i++) {
      c[m] = a[i] + b[i];
      m++;
  }
  for(i=deg2+1; i<=deg1; i++) {
      c[m] = a[i];
      m++;
  }
}
```

```
else   {
    for(i=0; i<=deg1; i++)  {
        c[m] = a[i] + b[i];
        m++;
    }
    for(i=deg1+1; i<=deg2; i++)  {
        c[m] = b[i];
        m++;
    }
}
```

Polynomial Addition


```
  printf("\npolynomial after addition  = %.1f",
c[0]);
  for(i=1; i<m; i++)
     printf("+ %.1fx^%d", c[i], i);
   return 0;
}
```

Output

Enter the highest degree of polynomial1: 3
Enter the coeff of x^0 :2
Enter the coeff of x^1 :3
Enter the coeff of x^2 :5
Enter the coeff of x^3 :1
Enter the highest degree of polynomial2: 2
Enter the coeff of x^0 :7
Enter the coeff of x^1 :8
Enter the coeff of x^2 :5
polynomial 1 = 2.0+ 3.0x^1+ 5.0x^2+ 1.0x^3
polynomial 2 = 7.0+ 8.0x^1+ 5.0x^2
polynomial after addition = 9.0+ 11.0x^1+ 10.0x^2+ 1.0x^3

Polynomial

Double Array Implementation:

Represent the following two polynomials:

$p_1(x) = 23x^{9} + 18x^{7} - 41x^{6} + 163x^{4} - 5x + 3$
$p_2(x) = 4x^{6} + 10x^{4} + 12x + 8$

Polynomial

$p_1(x) = 23x^9 + 18x^7 - 41x^6 + 163x^4 - 5x + 3$
$p_2(x) = 4x^6 + 10x^4 + 12x + 8$

Start $p_1(x)$ Start $p_2(x)$

Coefficient

23	18	-41	163	-5	3	4	10	12	8
9	7	6	4	1	0	6	4	1	0

0 2 8

Exponent

End $p_1(x)$ End $p_2(x)$

Polynomial

Advantages of using two array:

- save space (compact)

Disadvantages of using two Array:

- difficult to maintain
- have to allocate array size ahead of time
- more code required for misc. operations.

Polynomial using structure

Structure Implementation:

```
struct poly  {
    float coeff;
    int exp;   };
struct poly p[50];
```

Polynomial Addition


```
#include<stdio.h>
#include<math.h>
struct poly  {
    float coeff;
    int exp;   };
struct poly a[50], b[50], c[50], d[50];
int main() {
   int nterm1, nterm2, nterm3;
   int i, k=0, l=0, m=0;
   printf("Enter the number of non-zero terms in Polynomial1: ");
   scanf("%d", &nterm1);
   for(i=0; i<nterm1; i++)  {
       printf("\nEnter the coeff of %d th term: ", i);
       scanf("%f", &a[i].coeff);
       printf("\nEnter the exp of %d th term: ", i);
       scanf("%f", &a[i].exp);
   }
```

Polynomial Addition


```
printf("\nEnter the number of non-zero terms in Polynomial2: ");
 scanf("%d",  &nterm2);
 for(i=0; i<nterm2; i++)  {
    printf("\nEnter the coeff of %d th term: ", i);
    scanf("%f", &b[i].coeff);
    printf("\nEnter the exp of %d th term: ", i);
    scanf("%f", &b[i].exp);
 }
 printf("\nPolynomial 1 = %.1f", a[0].coeff);
 for(i=1; i<nterm1; i++)
     printf("+ %.1fx^%d", a[i].coeff, a[i].exp);
 printf("\nPolynomial 2 = %.1f", b[0].coeff);
 for(i=1; i<nterm2; i++)
     printf("+ %.1fx^%d", b[i].coeff, b[i].exp);
```

Polynomial Addition


```
while(k<nterm1 && l<nterm2)  {
    if(a[k].exp < b[l].exp) {
        c[m].coeff = a[k].coeff;
        c[m].exp = a[k].exp;
        k++; m++;        }
     else if(a[k].exp > b[l].exp) {
        c[m].coeff = b[l].coeff;
        c[m].exp = b[l].exp;
        l++; m++;        }
      else {
        c[m].coeff = a[k].coeff + b[l].coeff;
        c[m].exp = a[k].exp;
        k++; l++; m++;  }
 }
```

Polynomial Addition


```
while(k<nterm1)  {
    c[m].coeff = a[k].coeff;
    c[m].exp = a[k].exp;
    k++; m++;
}
 while(l<nterm2)  {
     c[m].coeff = b[l].coeff;
     c[m].exp = b[l].exp;
     l++; m++;
 }
 nterm3 = m-1;
 printf("\npolynomial after addition  = %.1f", c[0].coeff);
 for(i=1; i<nterm3; i++)
      printf("+ %.1fx^%d", c[i].coeff, c[i].exp);
 return 0;
}
```

Polynomial Multiplication


```
#include<stdio.h>
#include<math.h>

float a[50], b[50], c[50], d[50];
int main() {
   int i;
   int deg1,deg2;
   int k=0,l=0,m=0;
   printf("Enter the highest degree of polynomial1: ");
   scanf("%d", &deg1);
   for(i=0; i<=deg1; i++)  {
       printf("\nEnter the coeff of x^%d :", i);
       scanf("%f", &a[i]);
  }
```

Polynomial Multiplication


```
printf("\nEnter the highest degree of polynomial2: ");
 scanf("%d",  &deg2);
 for(i=0; i<=deg2; i++)  {
     printf("\nEnter the coeff of x^%d : ", i);
     scanf("%f", &b[i]);
 }
 printf("\nPolynomial 1 = %.1f", a[0]);
 for(i=1; i<=deg1; i++)
      printf("+ %.1fx^%d", a[i], i);
 printf("\nPolynomial 2 = %.1f", b[0]);
 for(i=1; i<=deg2; i++)
      printf("+ %.1fx^%d", b[i], i);
```

Polynomial Multiplication


```
deg3 = deg1+deg2;
for (int i = 0; i<=deg3; i++)
      c[i] = 0;
for (int i=0; i<=deg1; i++) {
       for (int j=0; j<=deg2; j++)
             c[i+j] += a[i] * b[j];
}
printf("\nPolynomial after multiplication = %.1f", c[0]);
for(i=1; i<=deg3; i++)
     printf("+ %.1fx^%d", c[i], i);
return 0;
}
```

Polynomial Multiplication

Output

Enter the highest degree of polynomial1:2
Enter the coeff of x^0 :2
Enter the coeff of x^1 :3
Enter the coeff of x^2 :4
Enter the highest degree of polynomial2:3
Enter the coeff of x^0 :5
Enter the coeff of x^1 :6
Enter the coeff of x^2 :7
Enter the coeff of x^3 :2
Polynomial 1 = 2.0+ 3.0x^1+ 4.0x^2
Polynomial 2 = 5.0+ 6.0x^1+ 7.0x^2+ 2.0x^3
Polynomial after multiplication = 10.0+ 27.0x^1+ 52.0x^2+ 49.0x^3+ 34.0x^4+ 8.0x^5

Polynomial Multiplication


```
#include<stdio.h>
#include<math.h>
struct poly  {
    float coeff;
    int exp;   };
struct poly a[50], b[50], c[50], d[50];
int main() {
   int nterm1, nterm2, nterm3;
   int i, j, k, l=0, m=0;
   float prod;
   printf("Enter the number of non-zero terms in Polynomial1: ");
   scanf("%d", &nterm1);
   for(i=0; i<nterm1; i++)  {
       printf("\nEnter the coeff of %d th term: ", i);
       scanf("%f", &a[i].coeff);
       printf("\nEnter the exp of %d th term: ", i);
       scanf("%f", &a[i].exp);       }
```

Polynomial Multiplication


```
printf("\nEnter the number of non-zero terms in Polynomial2: ");
scanf("%d",  &nterm2);
for(i=0; i<nterm2; i++)  {
     printf("\nEnter the coeff of %d th term: ", i);
     scanf("%f", &b[i].coeff);
     printf("\nEnter the exp of %d th term: ", i);
     scanf("%f", &b[i].exp);
 }
 printf("\nPolynomial 1 = %.1f", a[0].coeff);
 for(i=1; i<nterm1; i++)
      printf("+ %.1fx^%d", a[i].coeff, a[i].exp);
 printf("\nPolynomial 2 = %.1f", b[0].coeff);
 for(i=1; i<nterm2; i++)
      printf("+ %.1fx^%d", b[i].coeff, b[i].exp);
```

Polynomial Multiplication


```
for (int i=0; i<nterm1; i++) {
        for (int j=0; j<ntern2; j++) {
               prod = a[i].coeff*b[j].coeff;
               for (int k=0; k<m; k++) {
                     if(a[i].exp+b[j].exp == c[k].exp)  {
                           c[k].coeff += prod;
                           break;  }
               }
               c[m].exp=a[i].exp+b[j].exp;
               c[m++].coeff = prod;
        }
}
nterm3 = m-1;
printf("\nPolynomial after multiplication = %.1f", c[0].coeff);
for(i=1; i<nterm3; i++)
       printf("+ %.1fx^%d", c[i].coeff, c[i].exp);
return 0;  }
```

THANK YOU!