RAMAIAH
Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Data Structures

Course Code: CS32

Credits: 3:1:0

Term: September – December 2020

Faculty:
Mamatha Jadhav V
Vandana S Sardar

RAMAIAH
Institute of Technology

Polynomials $A(X)=3X^{20}+2X^5+4$, $B(X)=X^4+10X^3+3X^2+1$

Structure Polynomial is $p(x) = a_1 x^{e_1} + \ldots + a_n x^{e_n}$ **objects**: :a set of ordered pairs of $<e_i, a_i>$ where a_i in Coefficients and e_i in Exponents, e_i are integers >= 0

functions:

for all poly, poly1, poly2 □ Polynomial, coef □Coefficients, expon □□Exponents

```
Polynomial Zero( )                    ::= return the polynomial,
                   p(x) = 0
Boolean IsZero(poly)                  ::= if (poly) return FALSE
                                            else return TRUE
Coefficient Coef(poly, expon)         ::= if (expon □ poly) return its
                                            coefficient else return Zero
Exponent Lead_Exp(poly)               ::= return the largest exponent in
poly
Polynomial Attach(poly,coef, expon)   ::= if (expon □ poly) return error
                                            else return the polynomial poly
                                            with the term <coef, expon>
                                            inserted
```

```
Polynomial Remove(poly, expon)            ::= if (expon □ poly) return the
                    polynomial poly with the                    term
whose exponent is                            expon deleted
                           else return error
Polynomial SingleMult(poly, coef, expon) ::= return the polynomial
                                              poly • coef • x^expon
Polynomial Add(poly1, poly2)              ::= return the polynomial
                        poly1 +poly2
Polynomial Mult(poly1, poly2)             ::= return the polynomial
                                              poly1 • poly2
```

End *Polynomial*

Polynomial Addition

(1st Method)

```
#define MAX_DEGREE 101
typedef struct {
        int degree;
        float coef[MAX_DEGREE];
        } polynomial;

/* d =a + b, where a, b, and d are polynomials */
d = Zero( )
while (! IsZero(a) && ! IsZero(b)) do {
   switch COMPARE (Lead_Exp(a), Lead_Exp(b))
{
       case -1: d =
           Attach(d, Coef (b, Lead_Exp(b)), Lead_Exp(b));
           b = Remove(b, Lead_Exp(b));
           break;
```

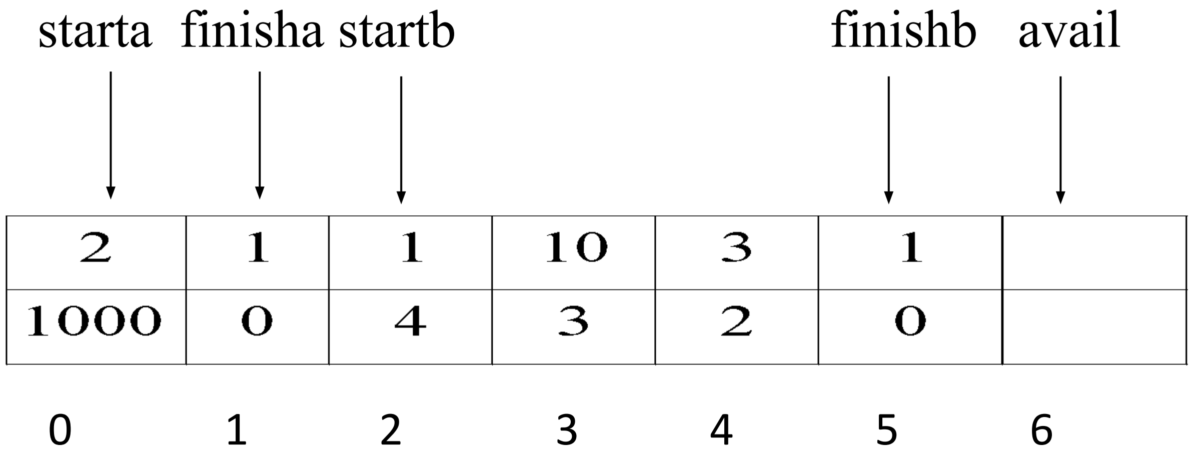
Polynomial Addition

```
case  0: sum = Coef (a, Lead_Exp (a)) + Coef ( b, Lead_Exp(b));
         if (sum) {
             Attach (d, sum, Lead_Exp(a));
             a = Remove(a , Lead_Exp(a));
             b = Remove(b , Lead_Exp(b));
             }
          break;
case 1: d =
          Attach(d, Coef (a, Lead_Exp(a)), Lead_Exp(a));
          a = Remove(a, Lead_Exp(a));
        }
   }
insert any remaining terms of a or b into d
```

(II Method) use one global array to store all polynomials

$A(X)=2X^{1000}+1$

$B(X)=X^4+10X^3+3X^2+1$



2	1	1	10	3	1	
1000	0	4	3	2	0	
0	1	2	3	4	5	6

Polynomial Addition

```
MAX_TERMS 100 /* size of terms array */
typedef struct {
          float coef;
          int expon;
          } polynomial;
polynomial terms[MAX_TERMS];
int avail = 0;
```

Polynomial Addition

```
void padd (int starta, int finisha, int startb, int finishb, int * startd, int *finishd)
{
/* add A(x) and B(x) to obtain D(x) */
    float coefficient;
   *startd = avail;
   while (starta <= finisha && startb <= finishb)
     switch (COMPARE(terms[starta].expon,
                                  terms[startb].expon))
{
      case -1: /* a expon < b expon */
               attach(terms[startb].coef, terms[startb].expon);
               startb++
               break;
```

RAMAIAH
Institute of Technology

Polynomial Addition

```
case  0: /* equal exponents */
            coefficient = terms[starta].coef +
                           terms[startb].coef;
            if (coefficient)
              attach (coefficient, terms[starta].expon);
            starta++;
            startb++;
            break;
case 1: /* a expon > b expon */
        attach(terms[starta].coef, terms[starta].expon);
        starta++;
}
```

Polynomial Addition

```
/* add in remaining terms of  A(x) */
for( ; starta <= finisha; starta++)
     attach(terms[starta].coef, terms[starta].expon);
/* add in remaining terms of B(x) */
for( ; startb <= finishb; startb++)
    attach(terms[startb].coef, terms[startb].expon);
*finishd =avail -1;
}
```

```
void attach(float coefficient, int exponent)
{
/* add a new term to the polynomial */
    if (avail >= MAX_TERMS) {
       fprintf(stderr, “Too many terms in the polynomial\n”);
       exit(1);
      }
      terms[avail].coef  = coefficient;
      terms[avail++].expon = exponent;
}
```

RAMAIAH Institute of Technology

Sparse Matrix

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

(a) 15/15

$$\begin{array}{c|cccccc|}
 & \text{col1} & \text{col2} & \text{col3} & \text{col4} & \text{col5} & \text{col6} \\
\text{row0} & 15 & 0 & 0 & 22 & 0 & -15 \\
\text{row1} & 0 & 11 & 3 & 0 & 0 & 0 \\
\text{row2} & 0 & 0 & 0 & -6 & 0 & 0 \\
\text{row3} & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{row4} & 91 & 0 & 0 & 0 & 0 & 0 \\
\text{row5} & 0 & 0 & 28 & 0 & 0 & 0
\end{array}$$

(b) 8/36

Sparse Matrix data structure?

<u>Two matrices</u>

SPARSE MATRIX ABSTRACT DATA TYPE

Structure Sparse_Matrix is

objects: a set of triples, <row, column, value>, where row and column are integers and form a unique combination, and value comes from the set item.

functions:
for all a, b ∈ Sparse_Matrix, x □ item, i, j, max_col, max_row □ index

Sparse_Marix Create(max_row, max_col) ::=
return a Sparse_matrix that can hold up to max_items = max _row □ max_col and whose maximum row size is max_row and whose maximum column size is max_col.

SPARSE MATRIX ABSTRACT DATA TYPE

Sparse_Matrix Transpose(*a*) ::=
return the matrix produced by interchanging
the row and column value of every triple.

Sparse_Matrix Add(*a, b*) ::=
if the dimensions of a and b are the same
return the matrix produced by adding
corresponding items, namely those with
identical *row* and *column* values.
else return error

Sparse_Matrix Multiply(*a, b*) ::=
if number of columns in a equals number of
rows in **b**
return the matrix *d* produced by multiplying
a by *b* according to the formula: *d* [*i*] [*j*] =
□(a[i][k]•b[k][j]) where *d (i, j)* is the *(i,j)*th
element
else return error.

Transpose of a Sparse Matrix

(1) If represented by a two-dimensional array.

Sparse matrix wastes space.

(2) Each element is characterized by <row, col, value>.

	row	col	value
[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	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]	2	1	3
[5]	2	5	28
[6]	3	0	22
[7]	3	2	-6
[8]	5	0	-15

row, column in ascending order

Sparse matrix and its transpose stored as triples

Transpose of a Sparse Matrix

Sparse_matrix Create(max_row, max_col) ::=

```
#define MAX_TERMS 101 /* maximum number of terms +1*/
    typedef struct {
                int col;
                int row;
                int value;
                } term;
    term a[MAX_TERMS]
```

# of rows (columns)

# **of nonzero** terms

Transpose of a Sparse Matrix

(1) for each row i

  take element <i, j, value> and store it
  in element <j, i, value> of the transpose.

difficulty: where to put <j, i, value>

(0, 0, 15) ====> (0, 0, 15)
(0, 3, 22) ====> (3, 0, 22)
(0, 5, -15) ====> (5, 0, -15)
(1, 1, 11) ====> (1, 1, 11)

Move elements down very often.

(2) For all elements in column j,

  place element <i, j, value> in element <j, i, value>

Transpose of a Sparse matrix

```
void transpose (term a[], term b[])
/* b is set to the transpose of a */
{
   int n, i, j, currentb;
   n = a[0].value;  /* total number of elements */
   b[0].row = a[0].col;  /* rows in b = columns in a */
   b[0].col = a[0].row;  /*columns in b = rows in a */
   b[0].value = n;
   if (n > 0) {                  /*non zero matrix */
      currentb = 1;
      for (i = 0; i < a[0].col; i++)
      /* transpose by columns in a */
           for( j = 1; j <=  n; j++)
           /*  find elements from the current column */
           if (a[j].col == i) {
          /* element is in current column, add it to b */
    b[currentb].row = a[j].col;
             b[currentb].col  = a[j].row;
             b[currentb].value = a[j].value;
             currentb++
          }
     }
  }
```

Scan the array "columns" times.
The array has "elements" elements.

O(columns*elements)

Discussion: compared with 2-D array representation

O(columns*elements) vs. O(columns*rows)

elements --> columns * rows when nonsparse

O(columns*columns*rows)

Problem: Scan the array “columns” times.

Solution:

Determine the number of elements in each column of the original matrix.

==>

Determine the starting positions of each row in the transpose matrix.

FAST TRANSPOSE

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[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28

	[0]	[1]	[2]	[3]	[4]	[5]
row_terms =	2	1	2	2	0	1
starting_pos =	1	3	4	6	8	8

FAST TRANSPOSE

```
void fast_transpose(term a[ ], term b[ ])
 {
   int row_terms[MAX_COL], starting_pos[MAX_COL];
   int i, j, num_cols = a[0].col, num_terms = a[0].value;
   b[0].row = num_cols; b[0].col = a[0].row;
   b[0].value = num_terms;
   if (num_terms > 0){ /*nonzero matrix*/
     for (i = 0; i < num_cols; i++)
          row_terms[i] = 0;
     for (i = 1; i  <= num_terms; i++)
          row_term [a[i].col]++
     starting_pos[0] = 1;
     for (i =1; i < num_cols; i++)
          starting_pos[i]=starting_pos[i-1] +row_terms [i-1];
```

FAST TRANSPOSE

```
for (i=1; i <= num_terms, i++) {
            j = starting_pos[a[i].col]++;
            b[j].row = a[i].col;
            b[j].col = a[i].row;
            b[j].value = a[i].value;
      }
   }
}
```

Compared with 2-D array representation

O(columns+elements) vs. O(columns*rows)

elements --> columns * rows

O(columns+elements) --> O(columns*rows)

Cost: Additional row_terms and starting_pos arrays are required.

Let the two arrays row_terms and starting_pos be shared

RAMAIAH
Institute of Technology

Thank you