



Articles » Development Lifecycle » Design and Architecture » Data Structures

Represent sparse matrices by some appropriate form of linked lists

By Tran Van Canh, 10 May 2003

★ ★ ★ ★ ★ 3.50 (19 votes)

[Download completed source files - 4.72 Kb](#)

Introduction

Sparse matrices are usually used to store and process data in many fields, for different purposes. Usually, we use 2D-dimension to store all elements of a matrix. It means that if size of the matrix is m rows and n columns, we have to use a 2D-dimension which has $m*n$ elements.

Consider a matrix in which most of the entries are zeros. This sometimes happens, for example when we process problems about graph, topological sort, etc... many elements of the data are zeros. In that case a large amount of memory is wasted to explicitly store all those zeros. Likewise when performing operations on the sparse matrices such as addition, subtraction and multiplication (especially), a large number of the operations have two operands of zero (0) which results in a large amount of wasted time. If a matrix is sparse time and space, we can store the elements that are non-zero only and implicitly assuming any element not stored is 0. This greatly reduces the number of times to be stored and the number of operations that must be performed for addition, subtraction and multiplication etc.. So instead of using 2D, we can use some other data structures to store elements of matrices and basing on new data structures, we perform the operations mentioned above more effectively.

We know that linked-lists are very popular and a useful kind of data store. There are several types of linked-lists, but some of them are singly linked-list type and doubly linked-list type which are always used because of their advantages. The problems of storing and processing data in sparse matrices mentioned above can be done well by using some kind of linked-lists. The most advantage of linked-list is manageability of memory and it is very important for large programs which are used to process a lot of data.

Problem

The purpose of the project is to find out some data structure to store data and implement operations on

sparse matrices by using linked-lists. As mentioned in the above part, using 2D to store elements of a sparse matrix in which an element has two index values (row index and column index), it is possible to access an element when we know the index values of it. But the disadvantages of 2D are waste of memory and time to store data and implement operations when most elements of sparse matrices are zeros.

Solution

Now, instead of using 2D, we will use doubly-linked lists to present sparse matrices. There are various methods of organizing doubly-linked-lists. We not only store elements of sparse matrices but also implement operations such as addition, subtraction, multiplication and so on. It means that we have to find out the suitable method of organization for storing and accessing data. The first type of organization I would like to show here is:

We can make up a linked list of linked lists. Here is a picture of the linked structure: Additional information would be needed to be stored for the size of the matrix. It could be larger than 3 rows and 4 columns, but any additional rows and columns are all zeros. We will use two types of node objects:

```
public class RowNode
{
    public RowNode nextRow;
    public int iRow;
    public CellNode firstCell;
}
public class CellNode
{
    public CellNode nextCell;
    public int iCol;
    public int iValue;
}
```

The second structure of data storing we can use to present a sparse matrix is that using the linked lists for each row and each column. So it needs two pointers: Next in this row and next in this column.

```
struct node {
    node *nextcol;
    node *nextrow;
    int row;
    int col;
    int value;
};
class matrix
{
    node **colhead; //the head element of columns
    node **rowhead; //the head element of rows
    int rows,cols; // number of rows and columns of a matrix
    operations of the class
}
```

Supplementing two attributes, rows and cols are needed to present number of rows and number of columns of a matrix. With these structures of data, we can implement the operations on matrices easier.

Matrix class detail definition

With the definition of a data structure as above, a sparse matrix is an object of class matrix. Some operations (methods) of class matrix can be defined as:

```
class matrix {
    node **colhead; //head pointer of cols
    node **rowhead; // head pointer of rows
    int rows,cols; //number of row and column of a matrix

public:
    matrix(char fn[LGMAX]); // init a matrix by reading data from a file
    matrix(int m, int n); //init a matrix with m rows and n cols
    matrix(int m); // init a unit marix

    // find a element which is located by row and col
    node *find(int row,int col);
    //get value of element at (row,col), return 0
    float get_element(int row,int col);

    if not exist
        // insert a element or remove
        void insert_remove(int row,int col,float value);
            //element if value==0 this operation
            //is needed when performing inverse
        void inverse(); // this operation is used to inverse a matrix
        void read_file(char fn[LGMAX+1]); //read data from file
        void write_file(char fn[LGMAX+1]); //write data to file
        void add(matrix b); //addition 2 matrices
        void multi(matrix b); //mutiply 2 matrices
        void display(); // display matrix to screen
        ~matrix();
        int getrows(); // get number of row of a matrix
        int getcols(); // get number of col of a matrix
};
```

Addition algorithm

To compute value of element at (row,col) of the addition matrix, at first we find the elements in the two source matrices respectively. If there is no existence of at least one element at (row,col), it means that in the result matrix we have no element at this cell (row,col). But if there is existence of at least one element at cell (row,col) then we have to compute the value of result matrix element at cell (row,col). After computing value, we add this value (of course this value is non zero) to result matrix at location of (row,col) by using **add_item** operation.

```
void matrix::add(matrix b) {
    node *lead1, *lead2;
    int m,n,m1,n1,i,j,kq;
    //rows1 and cols1 are stored size of matrix 1 when call init_matrix1
    m=rows;
    n=cols;
    m1=b.rows;
    n1=b.cols;

    if ((m!=m1) || (n!=n1))
    {
        box_str("Size of matrices is not the same !!");
        return;
    }
}
```

```

matrix c(rows,cols);
for (i=0;i for (j=0;j
{
    kq=0;
    if (rowhead[i]->nextcol!=NULL)
    {
        lead1=rowhead[i]->nextcol;
        while (lead1!=NULL)
        {
            if ((lead1->row==(i+1)) && (lead1->col==(j+1)))
                kq=kq+lead1->value;
            lead1=lead1->nextcol;
        }
    }
    if (b.rowhead[i]->nextcol!=NULL)
    {
        lead2=b.rowhead[i]->nextcol;
        while (lead2!=NULL)
        {
            if ((lead2->row==(i+1)) && (lead2->col==(j+1)))
                kq=kq+lead2->value;
            lead2=lead2->nextcol;
        }
    }
    if (kq!=0)
        c.insert_remove(i+1,j+1,kq);
}
char fn[LGMAX+1];
cout<<"Write to file:";gotoxy(7,25);cin>>fn;
c.write_file(fn);
cout<<"FIRST MATRIX";
display();
getch();
cout<<"SECOND MATRIX";
b.display();
getch();
cout<<"THE RESULT OF ADDITION MATRICE";
c.display();
}

```

Multiply algorithm

Similar to adding two matrices, the result matrix of multiplying two matrices is computed as follows:

We will use the original algorithm that uses loop commands.

```

for (i=0;i for(j=0;j {
    c[i][j]=0;
    for(k=0;k c[i][j]=c[i][j]+a[i][k]*b[k][j];
}

```

But because we do not store zero elements, the loop commands will be transformed to new form as follows:

```

void matrix:: multi(matrix b)
{
    node *lead1, *lead2;

```

```

int m,n,i,j,kq,k,m1,n1;
//rows1 and cols1 are stored size of matrix 1 when call init_matrix1
m=rows;
n=b.cols;
if (cols!=b.rows)
{
    box_str("Sizes of matrices are not allow for multiplication...");
    return;
}
matrix c(rows,b.cols);
for (i=0;i {
    if (rowhead[i]->nextcol!=NULL) //
    for(k=0;k {
        kq=0;
        lead1=rowhead[i]->nextcol;
        while (lead1!=NULL)
        {
            lead2=b.colhead[k]->nextrow;
            while (lead2!=NULL)
            {
                if (lead1->col==lead2->row)
                    kq=kq+lead1->value*lead2->value;
                lead2=lead2->nextrow;
            }
            lead1=lead1->nextcol;
        }
        if (kq!=0)
            c.insert_remove(i+1,k+1,kq);
    }
}
char fn[LGMAX+1];
gotoxy(5,24);cout<<"Write to file:";gotoxy(7,25);cin>>fn;
c.write_file(fn);
gotoxy(22,2);cout<<"FIRST MATRIX";
display();
getch();
gotoxy(22,2);cout<<"SECOND MATRIX";
b.display();
getch();
gotoxy(22,2);cout<<"THE RESULT OF MULTIPLY MATRICE";
c.display();
}

```

Inverse algorithm

In this project, we use the GAUSS algorithm to find the inverse matrix of a square matrix. We know that there are square matrices which do not have an inverse matrix because the determinant of these matrices is zero. The method that we perform, does a transition to the source matrix through steps to compute the value of elements of inverse matrix by using a unit matrix and a supplement matrix.

```

void matrix::inverse() //inverse matrix using unit matrix
{
    int i=1, j, done=0, m, k;
    double max, c;
    int n=rows;
    matrix b(rows);
    matrix x(rows,rows);
    while (!done)

```

```

{
    if (get_element(i,i) == 0)
    {
        max = 0;
        m = i;
        for (k=i+1; k<=n; k++)
            if (max < fabs(get_element(k,i)))
            {
                m = k;
                max = fabs(get_element(k,i)); //
            }
        if (m != i)
        {
            for (j=i; j<=n; j++)
            {
                c = get_element(i,j);
                insert_remove(i,j,get_element(m,j));
                insert_remove(m,j,c);
            }
            for (j=1; j<=n; j++)
            {
                c=b.get_element(i,j);
                b.insert_remove(i,j,b.get_element(m,j));
                b.insert_remove(m,j,c);
            }
        }
        if (m == i)
            done = 1;
    }
    if (get_element(i,i)!=0)
    {
        c= 1/get_element(i,i);
        for (j=i; j<=n; j++) //
            insert_remove(i,j,get_element(i,j)*c);
        for (j=1; j<=n; j++)
            b.insert_remove(i,j,b.get_element(i,j)*c);
        for (k=i+1; k<=n; k++)
        {
            for (j=i+1; j<=n; j++) //
                insert_remove(k,j,
                    (get_element(k,j)-get_element(i,j)*get_element(k,i)));
            for (j=1; j<=n; j++)
                b.insert_remove(k,j,
                    b.get_element(k,j)-b.get_element(i,j)*get_element(k,i));
            insert_remove(k,i,0);
        }
    }
    i ++;
    if (i>n)
        done = 1;
}
if (i> n)
{
    for (k=1; k<=n; k++)
    {
        x.insert_remove(n,k,b.get_element(n,k)/get_element(n,n));
        for (i=n-1; i>0; i--)
        {
            x.insert_remove(i,k,0);
            for (j=n; j>i; j--)
                b.insert_remove(i,k,
                    b.get_element(i,k)-get_element(i,j)*x.get_element(j,k));
        }
    }
}

```

```
        x.insert_remove(i,k,b.get_element(i,k)/get_element(i,j));
    }
}
char fn[LGMAX+1];
gotoxy(5,24);cout<<"Write to file:";gotoxy(7,25);gets(fn);
write_file(fn);
x.display();
}
else
{
    box_str("Can not inverse this matrix. OK");
    return;
}
}
```

Conclusion

This project is intended to solve the problem of storing, and implementing operations, in sparse matrices, by using linked lists. This is the way to save time and resources of the computer by reducing the number of operations which have zero elements. I know that this solution is not absolutely good because the data structure I use may not be yet logical. So please kindly tell me the way to resolve this problem more effectively. Thank you very much.

License

This article has no explicit license attached to it but may contain usage terms in the article text or the download files themselves. If in doubt please contact the author via the discussion board below.


A list of licenses authors might use can be found [here](#)

About the Author



Tran Van Canh

Web Developer

Anonymous Proxy 


Mr. Tran Van Canh finished bachelor degree of Computer Science at Vinh University-VietNam in 2000. He got master degree of Computer Science from Asian Institute of Technology (AIT) in August - 2004.

He has experiences in Visual Basic,.NET (C#, VB.NET), Java, C++, ASP, PHP, JSP, SQL,

Network administration, Portal Web Development.

Contact him: tranvan_canh@yahoo.com, canhtv@google.com

Comments and Discussions

 **6 messages** have been posted for this article Visit <http://www.codeproject.com/Articles/3276/Represent-sparse-matrices-by-some-appropriate-form> to post and view comments on this article, or click [here](#) to get a print view with messages.

[Permalink](#) | [Advertise](#) | [Privacy](#) | [Mobile](#)
Web02 | 2.6.130903.1 | Last Updated 11 May 2003

Article Copyright 2002 by Tran Van Canh
Everything else Copyright © [CodeProject](#), 1999-2013
[Terms of Use](#)