**Sparse Matrix:**

Problem Statement:

a) Write a c program to represent a sparse matrix taking the data from a file.

The file format is as follows:

6 5 6

2 2 -76

2 3 -12

3 1 -30

4 1 -6

4 3 -5

5 2 -10

The first line represent : number of rows, number of columns and number of non zero elements in spare matrix respectively.

Here, number of rows: 6, number of columns: 5, number of non zero elements: 6

The next 6 lines gives the position of non zero element in matrix with the non zero value. The rows and columns are given in ascending order in the file.

i.e, 2 2-76 means the non zero value -76 is stored in 2, 2 position.

Next, Write the transpose algorithm that runs in 0( n \* c) time, where n: total number of nonzero values in matrix and c: total number of columns. Note that, the transposed matrix must have its rows and columns within rows in ascending order.

b) Write another Program to transpose a sparse matrix in 0( n + c) times with extra space, where n and c are the number of non-zero values and number of columns in the matrix. Note that, the transposed matrix must have its rows and columns within rows in ascending order.

• Input example :

6 5 6

2 2 -76

2 3 -12

3 1 -30

4 1 -6

4 3 -5

5 2 -10

• Output example :

The O(n\*c) transpose

Row Column Value

1 3 -30

1 4 -6

2 2 -76

2 5 -10

3 2 -12

3 4 -5

The O(n+c) transpose

Row Column Value

1 3 -30

1 4 -6

2 2 -76

2 5 -10

3 2 -12

3 4 -5

Proposed C Code:

/\* ------- main.c ------- \*/

#include <stdio.h>

#include <stdlib.h>

typedef struct Point

{

int val;

int row;

int col;

} Point;

// This transpose alogorithm takes O(n\*c) time

Point \*transpose(Point \*arr, Point p)

{

Point \*ans = (Point \*)malloc(p.val \* sizeof(Point));

int current = 0;

for (int i = 1; i < p.col; i++)

{

for (int j = 0; j < p.val; j++)

{

if (arr[j].col == i)

{

ans[current].row = arr[j].col;

ans[current].col = arr[j].row;

ans[current].val = arr[j].val;

current++;

}

}

}

return ans;

}

// This transpose alogorithm takes O(n+c) time

Point \*quick\_transpose(Point \*arr, Point p)

{

Point \*ans = (Point \*)malloc(p.val \* sizeof(Point));

int \*count = (int \*)calloc(p.col + 1, sizeof(int));

for (int i = 0; i < p.val; i++)

{

count[arr[i].col]++;

}

int \*index = (int \*)calloc(p.col + 1, sizeof(int));

index[1] = 0;

for (int i = 2; i < p.col + 1; i++)

{

index[i] = index[i - 1] + count[i - 1];

}

for (int i = 0; i < p.val; i++)

{

int current = index[arr[i].col]++;

ans[current].col = arr[i].row;

ans[current].row = arr[i].col;

ans[current].val = arr[i].val;

}

return ans;

}

int main()

{

FILE \*fpr, \*fwr;

fpr = fopen("E:\\collage\\c\\matrix\\input.txt", "r");

fwr = fopen("E:\\collage\\c\\matrix\\output.txt", "w");

if (fpr == NULL)

{

printf("Not Opened");

}

Point p;

fscanf(fpr, "%d%d%d", &p.row, &p.col, &p.val);

Point \*arr = (Point \*)malloc(p.val \* sizeof(Point));

for (int i = 0; i < p.val; i++)

{

fscanf(fpr, "%d\n%d\n%d", &arr[i].row, &arr[i].col, &arr[i].val);

}

Point \*duplicate\_arr = arr; //Creating a duplicate array

arr = transpose(arr, p);

fprintf(fwr, "The O(n\*c) transpose\nRow Column Value\n");

for (int i = 0; i < p.val; i++)

{

fprintf(fwr, "%d\t%d\t%d\n", arr[i].row, arr[i].col, arr[i].val);

}

arr = duplicate\_arr; //Reset the array

fprintf(fwr, "\n\nThe O(n+c) transpose \nRow Column Value\n");

arr = quick\_transpose(arr, p);

for (int i = 0; i < p.val; i++)

{

fprintf(fwr, "%d\t%d\t%d\n", arr[i].row, arr[i].col, arr[i].val);

}

return 0;

}/\* ---------------------- \*/

Conclusion:

The first algorithm runs on O(n\*c) and the second algorithm runs on O(n+c) where  n is the total number of nonzero values in matrix and c is the total number of columns.

Limitations and assumptions for this algorithm include:

1. For this program “input.txt” must be present where the input information is stored.
2. Here we are using a duplicate array to reset the original array.
3. We are printing the answer on another file named “output.txt”.