NAME:VAIBHAV JADHAV
PRN:202201040027
CLASS:A4
ROLL NO:63

INPUT:

```
 #include<iostream>
using namespace std;
class sparsematrix{
    private:
        int spp[20][3],len;
    public:
        sparsematrix(int r,int c,int l){
        spp[0][0]=r;
        spp[0][1]=c;
        spp[0][2]=l;
        len=1;
        }
        void insert(int,int,int);
        void print();
        void addition(sparsematrix &m);
        void transpose();
        void multiplication(sparsematrix &m2);
        void fast_transpose();
};
void sparsematrix::insert(int r,int c,int val){
    spp[len][0]=r;
    spp[len][1]=c;
    spp[len][2]=val;
    len++;
};
void sparsematrix::print(){
    int i;
    cout<<"Sparse Matrix triplet representation -"<<endl;
    cout<<"Dimension "<<spp[0][0]<<" by "<<spp[0][1]<<endl;
    for(i=0;i<len;i++){
        cout<<spp[i][0]<<"\t"<<spp[i][1]<<"\t"<<spp[i][2]<<endl;
    }
};
void sparsematrix::transpose(){
    int i,j,l1,len=1;
    sparsematrix t(spp[0][1],spp[0][0],spp[0][2]);
    for(i=0;i<spp[0][1];i++){
        for(j=1;j<=spp[0][2];j++){
            if(spp[j][1]==i){
                t.spp[len][0]=spp[j][1];
                t.spp[len][1]=spp[j][0];
                t.spp[len][2]=spp[j][2];
                len++;
            }
        }
    }
     cout<<"Transpose of Matrix -"<<endl;
     cout<<"Dimension "<<t.spp[0][0]<<" by "<<t.spp[0][1]<<endl;
     for(i=0;i<len;i++){
         cout<<t.spp[i][0]<<"\t"<<t.spp[i][1]<<"\t"<<t.spp[i][2]<<endl;
     }
```

```
};
void sparsematrix::fast_transpose() {
    int rows = spp[0][0];
    int cols = spp[0][1];
    int nonzero = spp[0][2];
    int result[20][3];
    int row[cols];
    int pos[cols];

    for (int i = 0; i < cols; i++) {
        row[i] = 0;
    }
    for (int i = 1; i <= nonzero; i++) {
        row[spp[i][1]]++;
    }
    pos[0] = 1;
    for (int i = 1; i < cols; i++) {
        pos[i] = pos[i - 1] + row[i - 1];
    }
    for (int i = 1; i <= nonzero; i++) {
        int j = pos[spp[i][1]];
        result[j][0] = spp[i][1];
        result[j][1] = spp[i][0];
        result[j][2] = spp[i][2];
        pos[spp[i][1]]++;
    }

    cout << "Fast transpose completed. Transposed matrix:" << endl;
    result[0][0] = spp[0][1];
    result[0][1] = spp[0][0];
    result[0][2] = spp[0][2];

    for (int i = 0; i <= nonzero; i++) {
        cout << result[i][0] << "\t" << result[i][1] << "\t" << result[i][2] << endl;
    }
};
void sparsematrix::multiplication(sparsematrix &m) {
    int totalarr[200],indexarr[201],i,j,k,z,len,loc,temp1,temp2,temp3;
    sparsematrix t(m.spp[0][1],m.spp[0][0],m.spp[0][2]);
    for(i=0;i<m.spp[0][1];i++){
        len=0;
        for(j=0;j<m.spp[0][2];j++){
            if(i==m.spp[j+1][1]){
                len++;
                totalarr[i]=len;
            }
        }
    }
    indexarr[0]=1;
    for(i=1;i<(m.spp[0][1]+1);i++){
        indexarr[i]=indexarr[i-1]+totalarr[i-1];
    }
    for(i=1;i<=(m.spp[0][2]);i++){
        loc=indexarr[m.spp[i][1]];
        t.spp[loc][0]=m.spp[i][1];
        t.spp[loc][1]=m.spp[i][0];
        t.spp[loc][2]=m.spp[i][2];
        indexarr[m.spp[i][1]]=indexarr[m.spp[i][1]]+1;
    }

    sparsematrix t1(spp[0][0],t.spp[0][0],spp[0][2]);
    k=1;
    for(i=1;i<(spp[0][2]+1);i++){
        for(j=1;j<(t.spp[0][2]+1);j++){
            if(spp[i][1]==t.spp[j][1]){
                t1.spp[k][0]=spp[i][0];;
                t1.spp[k][1]=t.spp[j][0];
                t1.spp[k][2]=(spp[i][2]*t.spp[j][2]);
                k++;
            }
        }
    }
    t1.spp[0][2]=k-1;
    z=t1.spp[0][2];
    for(i=1;i<=t1.spp[0][2];i++){
        for(j=i+1;j<=t1.spp[0][2];j++){
            if(t1.spp[i][0]==t1.spp[j][0] && t1.spp[i][1]==t1.spp[j][1]){
                t1.spp[i][2]=t1.spp[i][2]+t1.spp[j][2];
                for(k=j;k<=t1.spp[0][2];k++){
                    t1.spp[k][0]=t1.spp[k+1][0];
                    t1.spp[k][1]=t1.spp[k+1][1];
                    t1.spp[k][2]=t1.spp[k+1][2];
                }
                z--;
            }
        }
    }

    t1.spp[0][2]=z;

    cout<<"Final Multiply Matrix"<<endl;
    cout<<"Dimension "<<t1.spp[0][0]<<" by "<<t1.spp[0][1]<<endl;
    for(i=0;i<=t1.spp[0][2];i++){
        cout<<t1.spp[i][0]<<"\t"<<t1.spp[i][1]<<"\t"<<t1.spp[i][2]<<endl;
    }
}
void sparsematrix::addition(sparsematrix &m){
    int i=1,j=1,k=1,l1,l2;
    l1=spp[0][2];
    l2=m.spp[0][2];
    int result[20][3];
    if(spp[0][0]!=m.spp[0][0] || spp[0][1]!=m.spp[0][1]){
        cout<<"Cant add Matrix"<<endl;
    }
    else{
        while((i<=l1) && (j<=l2)){
            if(spp[i][0]==m.spp[j][0]){
                if(spp[i][1]==m.spp[j][1]){
                    result[k][0]=spp[i][0];
                    result[k][1]=spp[i][1];
                    result[k][2]=spp[i][2]+m.spp[j][2];
                    i++;
                    j++;
                    k++;
                }
                else{
                    if(spp[i][1]<m.spp[j][1]){
                        result[k][0]=spp[i][0];
                        result[k][1]=spp[i][1];
                        result[k][2]=spp[i][2];
                        i++;
                        k++;
                    }
                    else{
                        result[k][0]=m.spp[j][0];
                        result[k][1]=m.spp[j][1];
                        result[k][2]=m.spp[j][2];
                        j++;
                        k++;
                    }
                }
            }
            else{
                if(spp[i][0]<m.spp[j][0]){
                    result[k][0]=spp[i][0];
                    result[k][1]=spp[i][1];
                    result[k][2]=spp[i][2];
                    i++;
                    k++;
                }
                else{
                    result[k][0]=m.spp[j][0];
                    result[k][1]=m.spp[j][1];
                    result[k][2]=m.spp[j][2];
                    j++;
                    k++;
                }
            }
        }
        while(i<=l1){
            result[k][0]=spp[i][0];
            result[k][1]=spp[i][1];
            result[k][2]=spp[i][2];
            i++;
            k++;
        }
        while(j<=l2){
            result[k][0]=m.spp[j][0];
            result[k][1]=m.spp[j][1];
            result[k][2]=m.spp[j][2];
            j++;
            k++;
        }
    }
    result[0][0]=spp[0][0];
    result[0][1]=spp[0][1];
    result[0][2]=k-1;
    cout<<"Final Addition Matrix  -"<<endl;
    cout<<"Dimension "<<result[0][0]<<" by "<<result[0][1]<<endl;
    for(i=0;i<k;i++){
        cout<<result[i][0]<<"\t"<<result[i][1]<<"\t"<<result[i][2]<<endl;
```

```
   }
};
int main(){
   int r,c,l,i,r1,c1,value;
   cout<<"\n****** Sparse Matrix Operations ******"<<endl;
   cout<<"Enter the First Sparse Matrix Triplet Representation"<<endl;
   cout<<"Enter the Total Number Of Rows :";
   cin>>r;
   cout<<"Enter the Total Number Of Columns :";
   cin>>c;
   cout<<"Enter the Total Number Of Non-Zero Elements :";
   cin>>l;
   sparsematrix sp1(r,c,l);
   for(i=0;i<l;i++){
      cout<<"Enter Row Number, Column Number and Non-Zero Element Value"<<endl;
      cin>>r1>>c1>>value;
      sp1.insert(r1,c1,value);
   }
   sp1.print();

   cout<<"Enter the Second Sparse Matrix Triplet Representation"<<endl;
   cout<<"Enter the Total Number Of Rows :";
   cin>>r;
   cout<<"Enter the Total Number Of Columns :";
   cin>>c;
   cout<<"Enter the Total Number Of Non-Zero Elements :";
   cin>>l;
   sparsematrix sp2(r,c,l);
   for(i=0;i<l;i++){
      cout<<"Enter Row Number, Column Number and Non-Zero Element Value"<<endl;
      cin>>r1>>c1>>value;
      sp2.insert(r1,c1,value);
   }
   sp2.print();

while(true){
   int ch;
   cout<<"--------------------------------------------"<<endl;
   cout<<"Select Your Choice: \n1. Simple Traspose\n2. Fast Traspose\n3. Addition\n4.
Multiplication\n5. Exit"<<endl;
   cout<<"Enter your choice here: ";
   cin>>ch;

   switch (ch) {
      case 1: {// Simple Transpose.
            cout<<"--------------------------------------------"<<endl;
            cout<<"Select Your Choice: \n1. Traspose of First Matrix \n2. Traspose of First Matrix.\n3.
Exit."<<endl;
            cout<<"Enter your choice here: ";
            cin>>ch;
            switch (ch) {
               case 1: {// Simple Transpose of first matrix.
                     sp1.transpose();
                     break;}

               case 2: {// Simple Transpose of second matrix.
                     sp2.transpose();
               break;}

          case 3: {// Exit.
              cout << "**  Thank You! **" << endl;
              exit;
              break;}

          default: cout<<"Enter the right choice.";
        }
        break;}

    case 2: {// Fast Transpose.
        cout<<"------------------------------------------"<<endl;
        cout<<"Select Your Choice: \n1. Fast Traspose of First Matrix \n2. Fast Traspose of First
Matrix.\n3. Exit."<<endl;
        cout<<"Enter your choice here: ";
        cin>>ch;
        switch (ch) {
          case 1: {
              sp1.fast_transpose();
              break;}

          case 2: {
              sp2.fast_transpose();
              break;}

          case 3: {// Exit.
              cout << "**  Thank You! **" << endl;
              exit;
              break;}

          default: cout<<"Enter the right choice!";
        }
        break;}

    case 3: {
        cout<<"------------------------------------------"<<endl;
        sp1.addition(sp2);
        break;}

    case 4: {
        cout<<"------------------------------------------"<<endl;
        sp1.multiplication(sp2);
        break;}

    case 5: {
        cout << "**  Thank You! **" << endl;
        return 0;
        break;}

    default :       cout<<"Enter the Right choice!";
        }
}
  return 0;
}
```

OUTPUT:

```
/tmp/H43bB1fV8W.o
****** Sparse Matrix Operations ******
Enter the First Sparse Matrix Triplet Representation
Enter the Total Number Of Rows :3
Enter the Total Number Of Columns :3
Enter the Total Number Of Non-Zero Elements :4
Enter Row Number, Column Number and Non-Zero Element Value
1
1
5
Enter Row Number, Column Number and Non-Zero Element Value
3
2
4
Enter Row Number, Column Number and Non-Zero Element Value
1
2
8
Enter Row Number, Column Number and Non-Zero Element Value
3
3
9
Sparse Matrix triplet representation -
Dimension 3 by 3
3       3       4
1       1       5
3       2       4
1       2       8
3       3       9
Enter the Second Sparse Matrix Triplet Representation
Enter the Total Number Of Rows :3
Enter the Total Number Of Columns :3
Enter the Total Number Of Non-Zero Elements :4
Enter Row Number, Column Number and Non-Zero Element Value
1
1
5
Enter Row Number, Column Number and Non-Zero Element Value
2
3
6
Enter Row Number, Column Number and Non-Zero Element Value
2
2
5
Enter Row Number, Column Number and Non-Zero Element Value
1
1
6
Sparse Matrix triplet representation -
Dimension 3 by 3
3       3       4
1       1       5
2       3       6
2       2       5
1       1       6
```

```
---------------------------------------------
Select Your Choice:
1. Simple Traspose
2. Fast Traspose
3. Addition
4. Multiplication
5. Exit
Enter your choice here: 1
---------------------------------------------
Select Your Choice:
1. Traspose of First Matrix
2. Traspose of First Matrix.
3. Exit.
Enter your choice here: 1
Transpose of Matrix -
Dimension 3 by 3
3           3           4
1           1           5
2           3           4
2           1           8
---------------------------------------------
Select Your Choice:
1. Simple Traspose
2. Fast Traspose
3. Addition
4. Multiplication
5. Exit
Enter your choice here: 5
**  Thank You! **
```