ASSIGNMENT- 3

DATE: 25.08.2023

//Q1.WAP to represent a given sparse matrix in 3-tuple/triplet format using 2-D array.

```
#include<stdio.h>
int main()
{
   int a[4][4]={{1, 1, 0, 0},
              {0 ,0, 2, 0},
              {0, 0, 0, 1},
              {9, 8, 0, 0}};
              printf("The given matrix is :-\n");
   int i,j;
   int nonzero=0;
   int sparse[3][3];
   for(i=0;i<=3;i++)
   {
      for(j=0;j<=3;j++)
      {
         printf("%d ",a[i][j]);
      }
      printf("\n");
   }
   for(i=0;i<=3;i++)
   {
      for(j=0;j<=3;j++)
      {
        if(a[i][j]!=0)
        {
         nonzero++;
        }
      }
   }
   sparse[0][2]=nonzero;
   sparse[0][1]=i;
   sparse[0][0]=j;
int s=1;
    for(i=0;i<=3;i++)
   {
      for(j=0;j<=3;j++)
      {
        if(a[i][j]!=0)
        {
         sparse[s][0]=i;
```

```
            sparse[s][1]=j;
            sparse[s][2]=a[i][j];
            s++;
           }
         }
     }
     printf("\n");
     printf("The sparse matrix is :- \n\n");
     printf ("Row Column  Nonzero-Element\n");
      for(i=0;i<=6;i++)
     {
        for(j=0;j<=2;j++)
        {
      printf("%d       ",sparse[i][j]);
        }
        printf("\n");
     }
     return 0;
}
```

//Q2 WAP to perform transpose of a given sparse matrix in 3-tuple/triplet format.

```
#include <stdio.h>
void transpose_sparse_matrix(int triplets[][3], int m, int n, int
transposed_triplets[2][3]) {
  int i, j;
  for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
      if (triplets[i][1] == j) {
        transposed_triplets[j][0] = i;
        transposed_triplets[j][1] = triplets[i][0];
        transposed_triplets[j][2] = triplets[i][2];
      }
    }
  }
}
int main() {
  int m = 3, n = 2;
  int triplets[][3] = {
    {0, 0, 1},
    {1, 1, 5},
    {2, 0, 2}
  };
  int transposed_triplets[2][3];
  transpose_sparse_matrix(triplets, m, n, transposed_triplets);
  printf("The transpose of the sparse matrix is: \n");
  for (int i = 0; i < n; i++) {
    printf("(%d, %d, %d)\n", transposed_triplets[i][0],
transposed_triplets[i][1], transposed_triplets[i][2]);
  }
  return 0;
}
```

//Q3 WAP to perform addition of two given sparse matrix in 3–tuple/triplet format.

```
#include <stdio.h>
#include <stdbool.h>
void add_sparse_matrices(int triplets1[][3], int m1, int n1, int
triplets2[][3], int m2, int n2, int triplets3[][3]) {
  int i, j;
  for (i = 0; i < m1; i++) {
    for (j = 0; j < n1; j++) {
      triplets3[i][0] = triplets1[i][0];
      triplets3[i][1] = triplets1[i][1];
      triplets3[i][2] = triplets1[i][2];
    }
  }
  for (i = 0; i < m2; i++) {
    for (j = 0; j < n2; j++) {
      if (triplets2[i][0] == i && triplets2[i][1] == j && triplets2[i][2] != 0){
        bool found = false;
        for (int k = 0; k < m1; k++) {
          if (triplets3[k][0] == i && triplets3[k][1] == j) {
            triplets3[k][2] += triplets2[i][2];
            found = true;
break;
          }
        }
        if (!found) {
          triplets3[i][0] = i;
          triplets3[i][1] = j;
          triplets3[i][2] = triplets2[i][2];
        }
      }
    }
  }
}
int main() {
  int m1 = 3, n1 = 2;
  int triplets1[][3] = {
    {0, 0, true},
    {1, 1, false},
    {2, 0, false}
  };
  int m2 = 2, n2 = 2;
  int triplets2[][3] = {
    {0, 1, true},
    {1, 0, false}
  };
  int m3 = m1, n3 = n2;
  int triplets3[3][3];
  add_sparse_matrices(triplets1, m1, n1, triplets2, m2, n2, triplets3);
int i,j;
  for (i = 0; i < m3; i++) {
    for (j = 0; j < n3; j++) {
      printf("%d, %d, %d\n", triplets3[i][0], triplets3[i][1], triplets3[i][2]);
    }
  }
  return 0;
}
```

//Q4 WAP to represent a polynomial of single variable using 1-D array and perform addition of two polynomial equations.

```
#include <stdio.h>
struct Term {
   int coefficient;
   int exponent;
};
void addPolynomials(struct Term poly1[], int size1, struct Term poly2[],
int size2, struct Term result[], int *size3) {
   int i = 0, j = 0, k = 0;

   while (i < size1 && j < size2) {
      if (poly1[i].exponent > poly2[j].exponent) {
         result[k++] = poly1[i++];
      } else if (poly1[i].exponent < poly2[j].exponent) {
         result[k++] = poly2[j++];
      } else {
         result[k].exponent = poly1[i].exponent;
         result[k].coefficient = poly1[i].coefficient + poly2[j].coefficient;
         i++;
         j++;
         k++;
      }
   }
   while (i < size1) {
      result[k++] = poly1[i++];
   }
   while (j < size2) {
      result[k++] = poly2[j++];
   }
   *size3 = k;
}
void displayPolynomial(struct Term poly[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%dx^%d", poly[i].coefficient, poly[i].exponent);
        if (i < size - 1) {
            printf(" + ");
        }
    }
    printf("\n");
}
int main() {
    struct Term poly1[10], poly2[10], result[20];
    int size1, size2, size3;

    printf("Enter the number of terms in the first polynomial: ");
    scanf("%d", &size1);
    printf("Enter the coefficients and exponents of the terms:\n");
    for (int i = 0; i < size1; i++) {
        scanf("%d %d", &poly1[i].coefficient, &poly1[i].exponent);
    }

    printf("Enter the number of terms in the second polynomial: ");
    scanf("%d", &size2);
    printf("Enter the coefficients and exponents of the terms:\n");
    for (int i = 0; i < size2; i++) {
        scanf("%d %d", &poly2[i].coefficient, &poly2[i].exponent);
    }
    addPolynomials(poly1, size1, poly2, size2, result, &size3);
    printf("First polynomial: ");
    displayPolynomial(poly1, size1);
    printf("Second polynomial: ");
    displayPolynomial(poly2, size2);
    printf("Sum of the polynomials: ");
    displayPolynomial(result, size3);
    return 0;
}
```