```
#include<stdio.h>
#include "def.h"
#include "forw.h"
#include "backw.h"
#include "Bw.h"
#include "pv.h"
#include "noram.h"

int main()
{

    forward();
    backward();
    Bw_algo();
    P_V();              // probabability of being visited ……..
    noramal();
    getch();
    return 0;



}
```

```
    void forward()
{
     //Forward Calculation
    for(i=0;i<N;i++)
      Alpha[0][i]=Pi[i] * b[i][Ob[0]];
    for(t=0;t<T-1;t++)
    {
      for(j=0;j<N;j++)
      {
        sum=0.0;
        for(i=0;i<N;i++)
            sum=sum+ Alpha[t][i]*a[i][j];
        Alpha[t+1][j]=sum * b[j][Ob[t+1]];
      }
    }
    printf("Forward matrix is \n");// printing Forward matrix ....
    for(t=0;t<T;t++)
    {
       for(i=0;i<N;i++)
         printf("%lf\t",Alpha[t][i]);
       printf("\n");
    }
}
```

```
void backward()
{
    // Back ward calculation .....
    for(i=0;i<N;i++)
      Beta[T-1][i]=1;
    for(t=T-2;t>=0;t--)
    {
        for(i=0;i<N;i++)
        {
            sum=0.0;
            for(j=0;j<N;j++)
              sum=sum+a[i][j]*Beta[t+1][j]*b[j][Ob[t+1]];
            Beta[t][i]=sum;
        }
    }
    printf("Backward matrix is \n");              //Printing Backward matrix ...
    for(t=0;t<T;t++)
    {
       for(i=0;i<N;i++)
         printf("%lf\t",Beta[t][i]);
       printf("\n");
    }
}
```

```
void Bw_algo()
{     int kk=0;
       double sum2=0.0;

       // Calculation of ZI values .
    for(t=0;t<T-1;t++)
    {
        for(i=0;i<N;i++)
        {
            for(j=0;j<N;j++)
             {
                    nu=Alpha[t][i]*b[j][Ob[t+1]]*Beta[t+1][j]*a[i][j];
                    sum=0.0;
                    for( m=0;m<N;m++)
                    {
                      for( n=0;n<N;n++)
                      {
                        sum+=Alpha[t][m] *a[m][n]*b[n][Ob[t+1]] *Beta[t+1][n];
                      }
                    }
                    ZI[t][i][j] =nu/sum;
             }
         }
    }
```

```
printf("\n ZI matrix is\n ");
for(t=0;t<T-1;t++)
{
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
              printf("%lf \t",ZI[t][i][j]);
        }
        printf("\n");
    }
    printf("\n");
 }
  // Gamma computation ..
  printf("GAMMA is");
  for(t=0;t<T;t++)
  {
    for(i=0;i<N;i++)
    {  sum=0.0;
       for(j=0;j<N;j++)
          sum=sum+ZI[t][i][j];
       printf("%lf\t",Gamma[t][i]=sum);
    }
    printf("\n");
  }
```

```
 // Expected number of transistions from state i
 for(i=0;i<N;i++)
 {   sum=0.0;
     for(t=0;t<T-1;t++)
        sum+=Gamma[t][i];
     E_T[i]=sum;
 }
 // Expected number of transitions from node i to node j
/*  for(i=0;i<N;i++)
 {
    for(j=0;j<N;j++)
    {   sum=0.0;
        for(t=0;t<T-1;t++)
           sum=sum+ZI[t][i][j];
        E_I_J[i]=sum;                  // may be a mistake.....……..
    }
 }  */
   // Computing estimated values for Pi ,A and B.
   printf("matrix pi is \n");
   for(i=0;i<N;i++)  //  for Pi
        printf("%lf\t ",E_Pi[i]=Gamma[0][i]);
    printf("\n");
    for(i=0;i<N;i++)
    {
         for(j=0;j<N;j++)
```

```
    {
        sum=0.0;nu=0.0;
        for(t=0;t<T-1;t++)
        {
            sum=sum+ZI[t][i][j];
            nu=nu+Gamma[t][i];
        }
        E_A[i][j]=sum/nu ;
    }
}
printf("The estimated transistion matrix is \n");
for(i=0;i<N;i++)
{
    for(j=0;j<N;j++)
    {
       printf("\t%lf",E_A[i][j]);
    }
    printf("\n");
}
// Computing the matrix B.....
printf("Matrix B is \n");
for(j=0;j<N;j++)    // number of states
{  sum2=0.0;
   for(kk=0;kk<K;kk++)
      sum1[kk]=0.0;
```

```
        for(t=0;t<T;t++)   // to traverse the observation sequence…
        {
            for(kk=0;kk<K;kk++)
            {
              if(Ob[t] == kk)  // here one for loop will come
              {    sum2+=Gamma[t][j]; // overall sum ……..
                   sum1[kk]+=Gamma[t][j];
                   break;
              }
            }
        }                       //here two more for loops will come

        for(kk=0;kk<K;kk++)
           E_B[j][kk]= sum1[kk] /(sum2);
        for(kk=0;kk<K;kk++)
           printf("\t%lf \t",E_B[j][kk]);
         printf("\n");
      }

}
```

```
#define T 19

#define N 6

#define K 3

        //for matrix B

double
a[N][N]={{0.2,0.2,0.15,0.15,0.1,0.1},{0,0.2,0.1,0.25,0.25,0.1},{0,0,0.15,0.15,0.2,0.2},{0,0,0,0.25,0.3,0.45},{
0,0,0,0,0.62,0.38},{0,0,0,0,0,1}};   // forward computation .

double b[N][K]={{0.4,0.4,0.2},{0.25,0.45,0.3},{0.2,0.35,0.45},{0.2,0.3,0.5},{0.6,0.2,0.2},{0.1,0.4,0.5}};

double Pi[N]={0.4,0.3,0.3}, sum;

int Ob[T]={1,2,1,2,1,0,1,1,1,0,2,1,0,0,1,2,2,1,0};   // possible values in the sequence .

double Beta[T][N]={0.0};  // Backward computation .

double Alpha[T][N]={0.0};

int i,j,t;//  T is for sequence length ;; N is for number of states

double ZI[T][N][N]={0.0},nu=0.0;

double Gamma[T][N]={0.0};

double E_T[N]={0.0};

double E_I_J[N]={0.0};

double E_Pi[N]={0.0};

double E_A[N][N]={0.0};

double N_E_A[N][N]={0.0};

double E_B[N][N]={0.0};

double sum1[K]={0};

double p_v[N]={0.0};

int status[N]={0};

int m,n,tt;

void forward();
```

```
void backward();

void Bw_algo();

void P_V();
```

```
void P_V()
{
        int i,j;
        double sum=0;
        // skip train .....

        printf("The probability of the node being \n\tvisited during the training phase %d\n",N);

        for(i=0;i<N;i++)  // loop for each node
        {
              if(i==0)
              { p_v[i]=E_Pi[i];
                continue;
              }
```

```
        else
        {    sum=0;
             printf("\n");
             for(j=0;j<=i-1;j++)
                sum=sum+ p_v[j]*(E_A[j][i]/(1-E_A[j][j]) );
        }
        p_v[i]= sum + (E_Pi[i]);

}
printf("\n");
for(i=0;i<N;i++)
  printf("%lf\t",p_v[i]*100);
tt=0;
for(i=0;i<N;i++)
{
   if(p_v[i]*100>=40.0)
   {
     status[tt]=i;
     tt++;
   }
}
   //else
    // status[i]=0;
printf("\nnstatus matrix is \n");
for(i=0;i<tt;i++)
```

```
        printf("\t%d",status[i]);
}




void noramal()
{   double sum2=0.0,sum3=0.0;
    int pp=0,pp1=0;
    for(i=0;i<N;i++)  // for each row ....
    {
       if(i==status[pp]) // row is not normalised
       {    pp=pp+1;
            for(j=0;j<N;j++)
             N_E_A[i][j]=E_A[i][j];    // row copied suc
            continue ;
       }
       else           // row is  normalised
       {
          sum3=0.0;sum2=0.0;pp1=0;
          for(j=0;j<N;j++)     // for each column
          {
             if(j==status[pp1]) // get the sum of edge weights keeping as it is ..
              { pp1=pp1+1;
                sum3+=a[i][j];
```

```
          }
          else
            sum2+=E_A[i][j]; //get the sum of edge weights to be normalised w.r.t sum3...
      }
      pp1=0;
      for(j=0;j<N;j++)
      {
         if(j!=status[pp1])
         {
             N_E_A[i][j]=(1-sum3)*(E_A[i][j]/sum2);
         }
         else
         {
              pp1=pp1+1;
              N_E_A[i][j]=a[i][j];
         }
      }
   }
}
printf("\nBEFORE NORMALISATION\n");
for(i=0;i<N;i++)
{
  for(j=0;j<N;j++)
     printf("%lf\t",E_A[i][j]);
  printf("\n");
```

```
    }
    printf("\nAFTER NORMALISATION\n");
    for(i=0;i<N;i++)
    {
      for(j=0;j<N;j++)
          printf("%lf\t",N_E_A[i][j]);
      printf("\n");
    }


}
```