```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <omp.h>
#define min(x, y) (((x) < (y)) ? (x) : (y))
//  Using the MONOTONIC clock
#define CLK CLOCK_MONOTONIC

/* Function to compute the difference between two points in time */
struct timespec diff(struct timespec start, struct timespec end);

/*
   Function to computes the difference between two time instances

   Taken from - http://www.guyrutenberg.com/2007/09/22/profiling-code-using-clock_gettime/

   Further reading:
http://stackoverflow.com/questions/6749621/how-to-create-a-high-resolution-timer-in-linux-to-measure-program-performance
http://stackoverflow.com/questions/3523442/difference-between-clock-realtime-and-clock-monotonic
 */
struct timespec diff(struct timespec start, struct timespec end){
        struct timespec temp;
        if((end.tv_nsec-start.tv_nsec)<0){
                temp.tv_sec = end.tv_sec-start.tv_sec-1;
                temp.tv_nsec = 1000000000+end.tv_nsec-start.tv_nsec;
        }
        else{
                temp.tv_sec = end.tv_sec-start.tv_sec;
                temp.tv_nsec = end.tv_nsec-start.tv_nsec;
        }
        return temp;
}

double absolute(double a)
{
   if(a<0)
   {
      return -a;
   }
   else
   {
```

```
        return a;
    }
}

double* compute(double* mat1, double* mat2,double d, int n)
{
    int i,j;
    double* ans = (double*) malloc(n * sizeof(double));
    #pragma omp parallel for schedule(guided,4)
    for(i=0;i<n;i++)
    {
        ans[i]=(1-d)/n;
    }

//    for(i=0;i<n;i++)
//    {
//        for(j=0;j<n;j++)
//        {
//            ans[i]=ans[i]+mat1[j]mat2[j * n + i](d);
//        }
//    }

    for(i=0;i<n;i++)
    {
        double ans1=ans[i];
       #pragma omp parallel for reduction(+:ans1) schedule(guided,4)
        for(j=0;j<n;j++)
        {

            ans1=ans1+mat1[j]mat2[j * n + i](d);
        }
        ans[i]=ans1;
    }
    return ans;
}


int main()
{
  struct timespec start_alg, end_alg, alg, arr[4];
    int n;
    int b,z,az;
    scanf("%d %d",&n,&b);
    //FILE *fpt;
```

```
// fpt = fopen("RunTotalTime.csv", "w+");
 clock_t start, end;

    int i,j,a1,a2;
    double* h = (double*) malloc(n*n * sizeof(double));
    double* pagerank = (double*) malloc(n * sizeof(double));
    int* num_outlinks = (int*) malloc(n * sizeof(int));
    double* pagerank_prev = (double*) malloc(n * sizeof(double));



    //data input----------------------------------
    double total_time=0;
//for(z=0;z<5;z++)
//{
for(az=1;az<=4;az++)
 {
    omp_set_num_threads(az);
    #pragma omp parallel for schedule(guided,4)
    for(i=0;i<n;i++)
    {
        pagerank[i]=1.0/n;
        pagerank_prev[i]=1.0/n;
        num_outlinks[i]=0;
        for(j=0;j<n;j++)
        {
            h[i*n+j]=0;
        }
    }
    double e=0.00001;
    double d=0.85;
    // data input-------------------------------------
    printf("enter input:");

    //printf("enter b");

    for (i=0;i<b;i++)
    {
    //printf("enter karo a1:");
    scanf("%d",&a1);
   // printf("enter karo a2:");
    scanf("%d",&a2);

    h[a1 * n + a2]=1;
```

```
  }
  clock_gettime(CLK, &start_alg);

// #pragma omp parallel for  schedule(auto)
 for (i=0;i<n;i++)
  {
  int num_outlinks1=0;
 #pragma omp parallel for reduction(+:num_outlinks1) schedule(guided,4)
     for(j=0;j<n;j++)
     {
        if(h[i * n + j]==1)
           num_outlinks[i]++;
     }
  num_outlinks[i]+=num_outlinks1;
  }

  #pragma omp parallel for schedule(guided,4) collapse(2)
  for(i=0;i<n;i++)
  {
     for(j=0;j<n;j++)
     {
        if(h[i * n + j]==1)
        {
           h[i * n + j]=1.0/num_outlinks[i];
        }
     }
  }

  int itr=0;

  while(1)
  {
     itr++;
     printf("thread: %d\n",az);
     for(i=0;i<10;i++)
     {
        printf("%lf ",pagerank[i]);
     }
     printf("\n");
     pagerank = compute(pagerank_prev,h,d,n);
     int i = 0;
     int flag=0;
     #pragma omp parallel for schedule(guided,4)
     for(i=0;i<n;i++)
```

```
    {
        if (absolute(pagerank[i]-pagerank_prev[i])>e)                {
            flag=1;

        }
    }
    if(flag==0)
    {
        break;
    }
    #pragma omp parallel for schedule(guided,4)
    for(i=0;i<n;i++)
    {
        pagerank_prev[i]=pagerank[i];
    }
 }

 printf("itr: %d\n",itr);

 clock_gettime(CLK, &end_alg);
 alg = diff(start_alg, end_alg);
 arr[az-1]=alg;
}

 //total_time += (end - start)/((double)CLOCKS_PER_SEC);
//}
//total_time=alg/5;
 for(i=0;i<10;i++)
 {
    printf("%lf ",pagerank[i]);
 }
 printf("\n \n time: %d\n",alg.tv_sec);

 printf("\n \n");

 for(i=0;i<4;i++)
 {
    printf("%d\n",arr[i].tv_sec);
 }

 free(h);
 free(pagerank);
 free(pagerank_prev);
 free(num_outlinks);
```

```
    //fclose(fpt);
    return 0;

}
```