# MATRIX MULTIPLICATION:

#include <stdio.h>

#include <conio.h>

#include <omp.h>

int main()

{

int m, n, p, q, c, d, k, sum = 0;

int first[10][10], second[10][10], multiply[10][10];

printf("Enter no of rows and columns of first matrix\n");

scanf("%d%d", &m, &n);

printf("Enter elements of first matrix\n");

for (c = 0; c < m; c++)

for (d = 0; d < n; d++)

scanf("%d", &first[c][d]);

printf("Enter no of rows and columns of second matrix\n");

scanf("%d%d", &p, &q);

if (n != p)

printf("The matrices can't be multiplied with each other.\n");

else

{

printf("Enter elements of second matrix\n");

#pragma omp parallel num threads()

{

#pragma omp for

for (c = 0; c < p; c++)

for (d = 0; d < q; d++)

scanf("%d", &second[c][d]);

for (c = 0; c < m; c++) {

for (d = 0; d < q; d++) {

for (k = 0; k < p; k++) {

sum = sum + first[c][k]\*second[k][d];

}

multiply[c][d] = sum;

sum = 0;

}

}

printf("Product of the matrices:\n");

for (c = 0; c < m; c++) {

for (d = 0; d < q; d++)

printf("%d\t", multiply[c][d]);

printf("\n");

}

}

}

return 0;

}

# ADJACENCY MATRIX:

#include<stdio.h>

#include<conio.h>

#include<omp.h>

#define max 20

int adj[max][max];

int n;

int main() {

int choice;

int node, origin, destin;

create\_graph();

while (1) {

printf("1.Insert a node\n");

printf("2.Delete a node\n");

printf("3.Dispaly\n");

printf("4.Exit\n");

printf("Enter your choice : ");

scanf("%d", &choice);

switch (choice) {

case 1:

insert\_node();

break;

case 2:

printf("Enter a node to be deleted : ");

fflush(stdin);

scanf("%d", &node);

delete\_node(node);

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("Wrong choice\n");

break;

}

}

getch();

}

create\_graph() {

int i, max\_edges, origin, destin;

printf("Enter number of nodes : ");

scanf("%d", &n);

max\_edges = n \* (n - 1);

#pragma omp parallel num threads()

{

for (i = 1; i <= max\_edges; i++) {

printf("Enter edge %d( 0 0 ) to quit : ", i);

scanf("%d %d", &origin, &destin);

if ((origin == 0) && (destin == 0))

break;

if (origin > n || destin > n || origin <= 0 || destin <= 0) {

printf("Invalid edge!\n");

i--;

} else

adj[origin][destin] = 1;

}

}

}

display() {

#pragma omp parallel num threads()

{

int i, j;

for (i = 1; i <= n; i++) {

for (j = 1; j <= n; j++)

printf("%4d", adj[i][j]);

printf("\n");

}

}

}

insert\_node() {

#pragma omp parallel num threads()

{

int i;

n++;

printf("The inserted node is %d \n", n);

for (i = 1; i <= n; i++) {

adj[i][n] = 0;

adj[n][i] = 0;

}

}

}

void delete\_node(char u) {

int i, j;

if (n == 0) {

printf("Graph is empty\n");

return;

}

if (u > n) {

printf("This node is not present in the graph\n");

return;

}

for (i = u; i <= n - 1; i++)

for (j = 1; j <= n; j++) {

adj[j][i] = adj[j][i + 1];

adj[i][j] = adj[i + 1][j];

}

n--;

}