



**Assignment 1**

#include <bits/stdc++.h>

#include <omp.h>

using namespace std;

const int N = 1e5 + 5;

vector<int> g[N];

bool vis[N];

void bfs(int s) {

queue<int> q;

q.push(s);

vis[s] = true;

while (!q.empty()) {

int u = q.front();

q.pop();

#pragma omp parallel for

for (int i = 0; i < g[u].size(); i++) {

int v = g[u][i];

if (!vis[v]) {

vis[v] = true;

q.push(v);

}

}

}

}

void dfs(int u) {

vis[u] = true;

#pragma omp parallel for

for (int i = 0; i < g[u].size(); i++) {

int v = g[u][i];

if (!vis[v])

dfs(v);

}

}

int main() {

int n, m, s, choice;

cin >> n >> m >> s >> choice;

for (int i = 0; i < m; i++) {

int x, y;

cin >> x >> y;

g[x].push\_back(y);

g[y].push\_back(x);

}

if (choice == 1)

bfs(s);

else if (choice == 2)

dfs(s);

else

cout << "Invalid choice\n";

cout << "The result of traversal:\n";

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

if (vis[i])

cout << i << " ";

}

cout << endl;

return 0;

}

**Assignment 2**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <omp.h>

using namespace std;

void parallelBubbleSort(int \*array, int n) {

int i, j;

#pragma omp parallel for private(i, j) shared(array)

for (i = 0; i < n-1; i++) {

for (j = 0; j < n-i-1; j++) {

if (array[j] > array[j+1]) {

// Swap elements

int temp = array[j];

array[j] = array[j+1];

array[j+1] = temp;

}

}

}

}

void merge(int \*array, int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int \*L = new int[n1];

int \*R = new int[n2];

for (i = 0; i < n1; i++)

L[i] = array[l + i];

for (j = 0; j < n2; j++)

R[j] = array[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

array[k] = L[i];

i++;

} else {

array[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

array[k] = L[i];

i++;

k++;

}

while (j < n2) {

array[k] = R[j];

j++;

k++;

}

delete [] L;

delete [] R;

}

void parallelMergeSort(int \*array, int l, int r) {

if (l < r) {

int m = l+(r-l)/2;

#pragma omp parallel sections

{

#pragma omp section

parallelMergeSort(array, l, m);

#pragma omp section

parallelMergeSort(array, m+1, r);

}

merge(array, l, m, r);

}

}

int main() {

int n;

cout << "Enter the size of the array: ";

cin >> n;

int \*array = new int[n];

srand(time(0));

for (int i = 0; i < n; i++) {

array[i] = rand() % 100;

}

cout << "Original Array: ";

for (int i = 0; i < n; i++) {

cout << array[i] << " ";

}

cout << endl;

int choice;

cout << "Enter 1 for Parallel Bubble Sort or 2 for Parallel Merge Sort: ";

cin >> choice;

if (choice == 1) {

parallelBubbleSort(array, n);

} else if (choice == 2) {

parallelMergeSort(array, 0, n-1);

} else {

cout << "Invalid choice. Exiting program." << endl;

return 0;

}

cout << "Sorted Array: ";

for (int i = 0; i < n; i++) {

cout << array[i] << " ";

}

cout << endl;

delete [] array;

return 0;

}