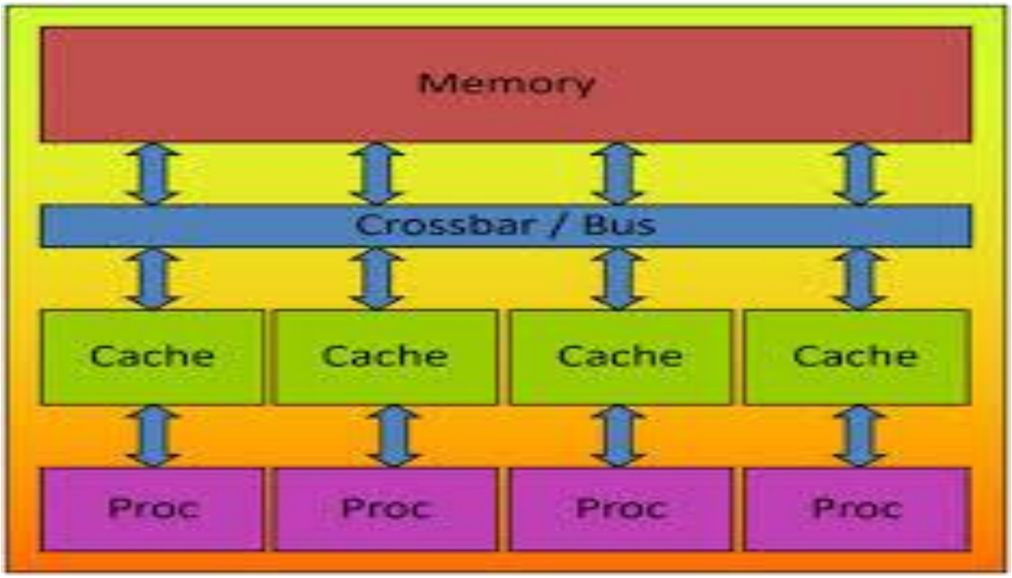
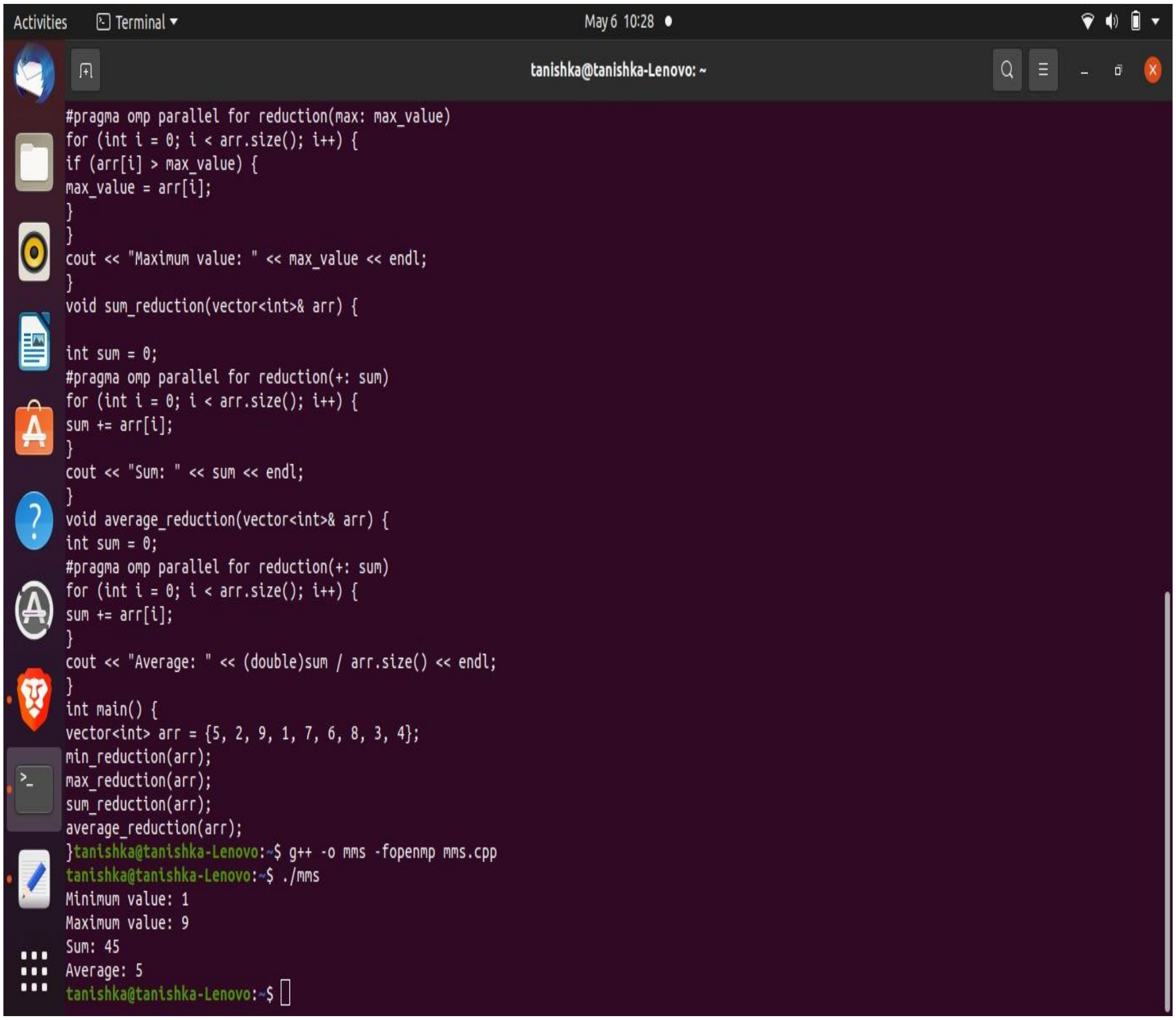
Aim: Implement Min, Max, Sum and Average operations using Parallel Reduction



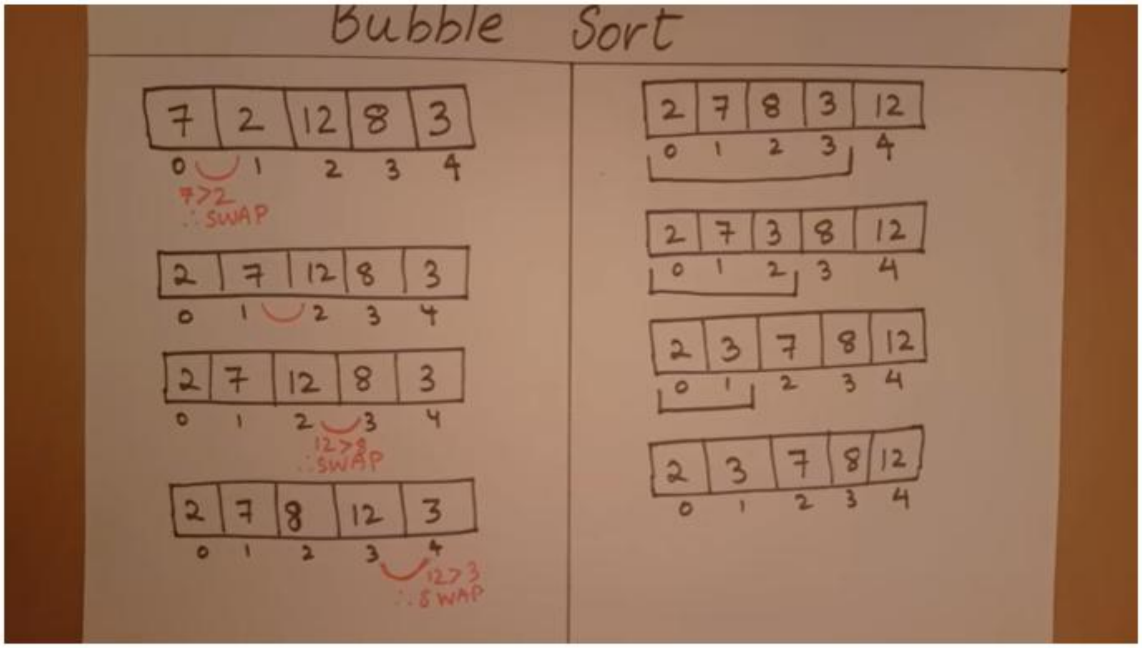

```
#include <iostream>
#include <vector>
#include <omp.h>
#include <climits>
using namespace std;
void min_reduction(vector<int>& arr) {
int min_value = INT_MAX;
#pragma omp parallel for reduction(min: min_value)
for (int i = 0; i < arr.size(); i++) {
if (arr[i] < min_value) {
min_value = arr[i];
}
}
cout << "Minimum value: " << min_value << endl;
}
void max_reduction(vector<int>& arr) {
int max_value = INT_MIN;
#pragma omp parallel for reduction(max: max_value)
for (int i = 0; i < arr.size(); i++) {
if (arr[i] > max_value) {
max_value = arr[i];
```

```
}
}
cout << "Maximum value: " << max_value << endl;
}
void sum_reduction(vector<int>& arr) {

int sum = 0;
#pragma omp parallel for reduction(+: sum)
for (int i = 0; i < arr.size(); i++) {
sum += arr[i];
}
cout << "Sum: " << sum << endl;
}
void average_reduction(vector<int>& arr) {
int sum = 0;
#pragma omp parallel for reduction(+: sum)
for (int i = 0; i < arr.size(); i++) {
sum += arr[i];
}
cout << "Average: " << (double)sum / arr.size() << endl;
}
int main() {
vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
min_reduction(arr);
max_reduction(arr);
sum_reduction(arr);
average_reduction(arr);
}
```



```
#pragma omp parallel for reduction(max: max_value)
for (int i = 0; i < arr.size(); i++) {
if (arr[i] > max_value) {
max_value = arr[i];
}
}
cout << "Maximum value: " << max_value << endl;
}
void sum_reduction(vector<int>& arr) {

int sum = 0;
#pragma omp parallel for reduction(+: sum)
for (int i = 0; i < arr.size(); i++) {
sum += arr[i];
}
cout << "Sum: " << sum << endl;
}
void average_reduction(vector<int>& arr) {
int sum = 0;
#pragma omp parallel for reduction(+: sum)
for (int i = 0; i < arr.size(); i++) {
sum += arr[i];
}
cout << "Average: " << (double)sum / arr.size() << endl;
}
int main() {
vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
min_reduction(arr);
max_reduction(arr);
sum_reduction(arr);
average_reduction(arr);
}tanishka@tanishka-Lenovo:~$ g++ -o mms -fopenmp mms.cpp
tanishka@tanishka-Lenovo:~$ ./mms
Minimum value: 1
Maximum value: 9
Sum: 45
Average: 5
tanishka@tanishka-Lenovo:~$
```

Aim: Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.



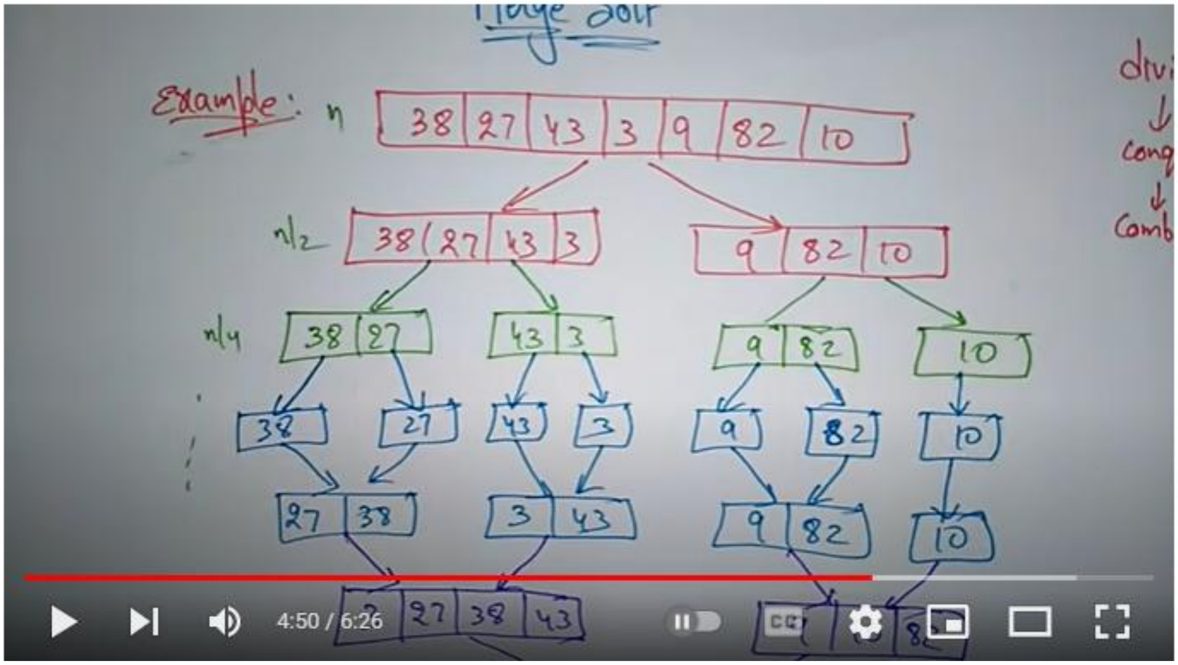
Bubble Sort Code-

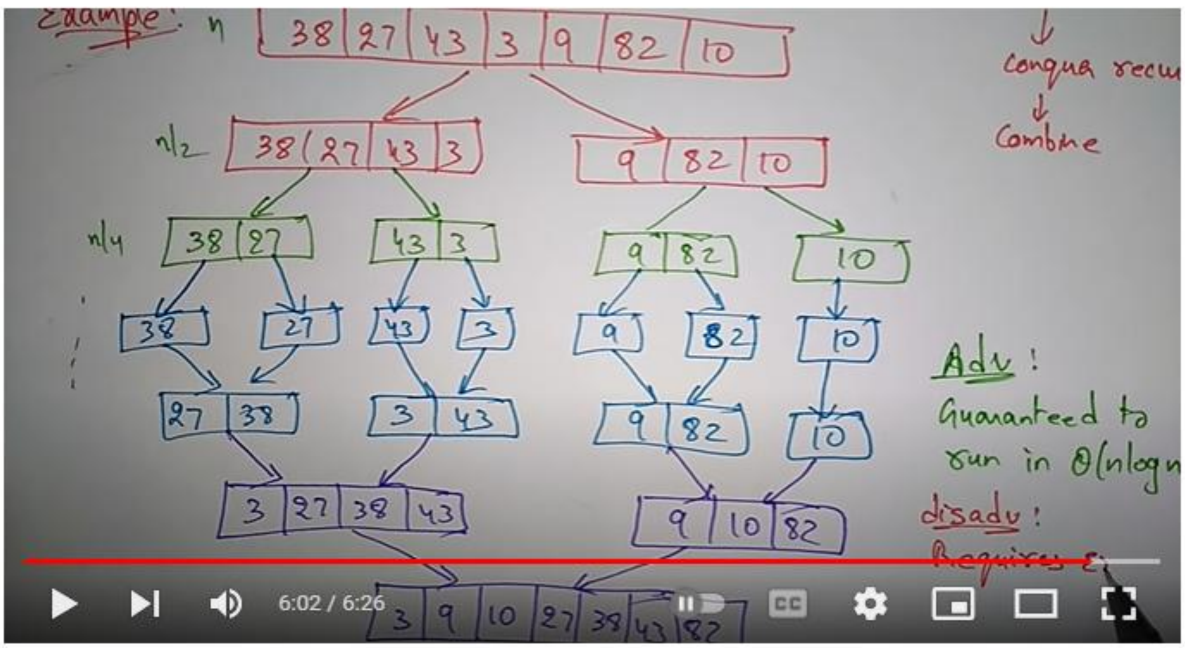
```
#include <iostream>
#include <vector>
#include <omp.h>
using namespace std;
void bubble_sort_odd_even(vector<int>& arr) {
bool isSorted = false;
while (!isSorted) {
isSorted = true;
#pragma omp parallel for
for (int i = 0; i < arr.size() - 1; i += 2) {
if (arr[i] > arr[i + 1]) {
swap(arr[i], arr[i + 1]);
isSorted = false;
}   }
#pragma omp parallel for
```

```
for (int i = 1; i < arr.size() - 1; i += 2) {
if (arr[i] > arr[i + 1]) {
swap(arr[i], arr[i + 1]);
isSorted = false;
}   }   }   }
int main() {
vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
double start, end;
// Measure performance of parallel bubble sort using odd-
//even transposition
start = omp_get_wtime();
bubble_sort_odd_even(arr);
end = omp_get_wtime();
cout << "Parallel bubble sort using odd-even transposition time: " << end - start << endl;
}
```

```
#include <iostream>
#include <vector>
#include <omp.h>
using namespace std;
void bubble_sort_odd_even(vector<int>& arr) {
bool isSorted = false;
while (!isSorted) {
isSorted = true;
#pragma omp parallel for
for (int i = 0; i < arr.size() - 1; i += 2) {
if (arr[i] > arr[i + 1]) {
swap(arr[i], arr[i + 1]);
isSorted = false;
}
}
#pragma omp parallel for
for (int i = 1; i < arr.size() - 1; i += 2) {
if (arr[i] > arr[i + 1]) {
swap(arr[i], arr[i + 1]);
isSorted = false;
}
}
}
}

int main() {
vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
double start, end;
// Measure performance of parallel bubble sort using odd-
//even transposition
start = omp_get_wtime();
bubble_sort_odd_even(arr);
end = omp_get_wtime();
cout << "Parallel bubble sort using odd-even transposition time: " << end - start << endl;
}tanishka@tanishka-Lenovo:~$ g++ -o bbsort -fopenmp bbsort.cpp
tanishka@tanishka-Lenovo:~$ ./bbsort
Parallel bubble sort using odd-even transposition time: 0.000706607
tanishka@tanishka-Lenovo:~$
```

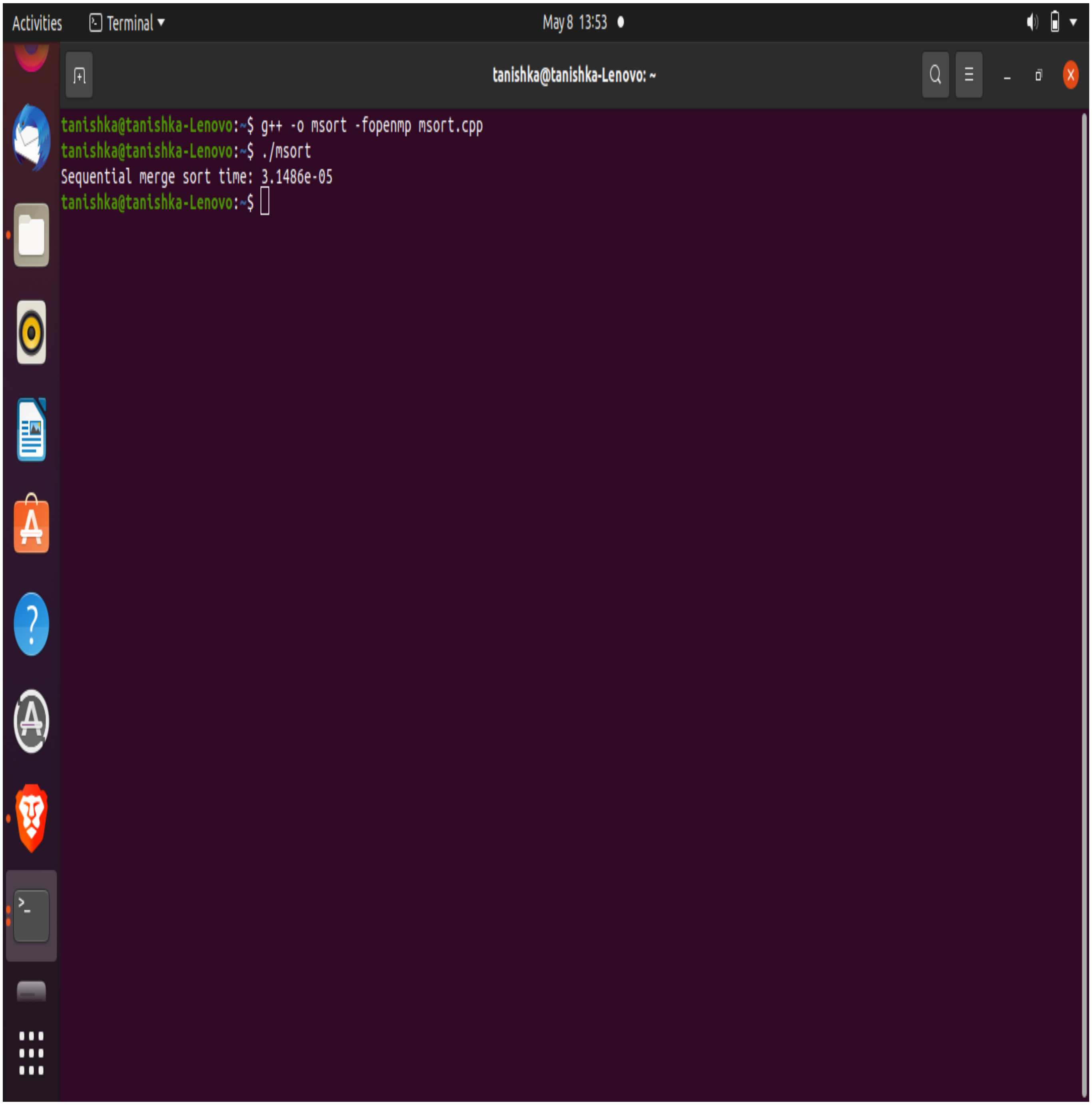
Merge Sort


Example:
n
38 27 43 3 9 82 10
n/2
38 27 43 3
9 82 10
n/4
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
4:50 / 6:26


Example:
n
38 27 43 3 9 82 10
Combine
n/2
38 27 43 3
9 82 10
n/4
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
Adv:
Guaranteed to
run in
3 27 38 43
9 10 82
disadv:
3 9 10 27 38 43 82
6:02 / 6:26

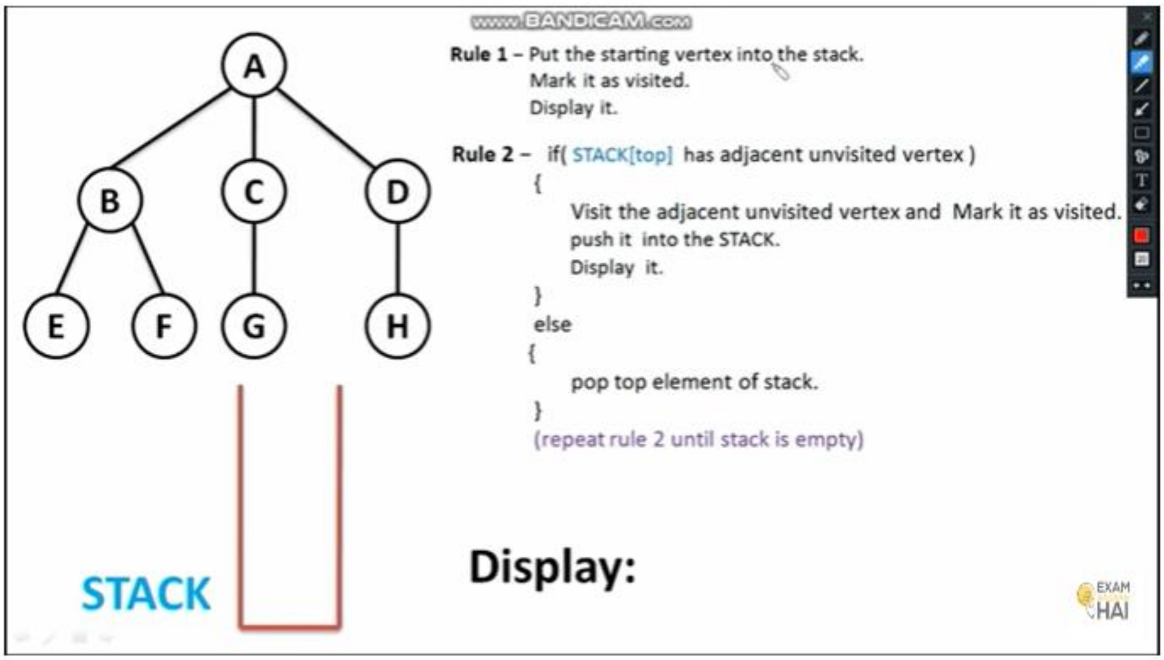
```
#include <iostream>
#include <vector>
#include <omp.h>
using namespace std;
void merge(vector<int>& arr, int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    vector<int> L(n1), R(n2);
    for (i = 0; i < n1; i++) {
        L[i] = arr[l + i];
    }
    for (j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
    }
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }
}
void merge_sort(vector<int>& arr, int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
#pragma omp task
        merge_sort(arr, l, m);
#pragma omp task
        merge_sort(arr, m + 1, r);
        merge(arr, l, m, r);
```

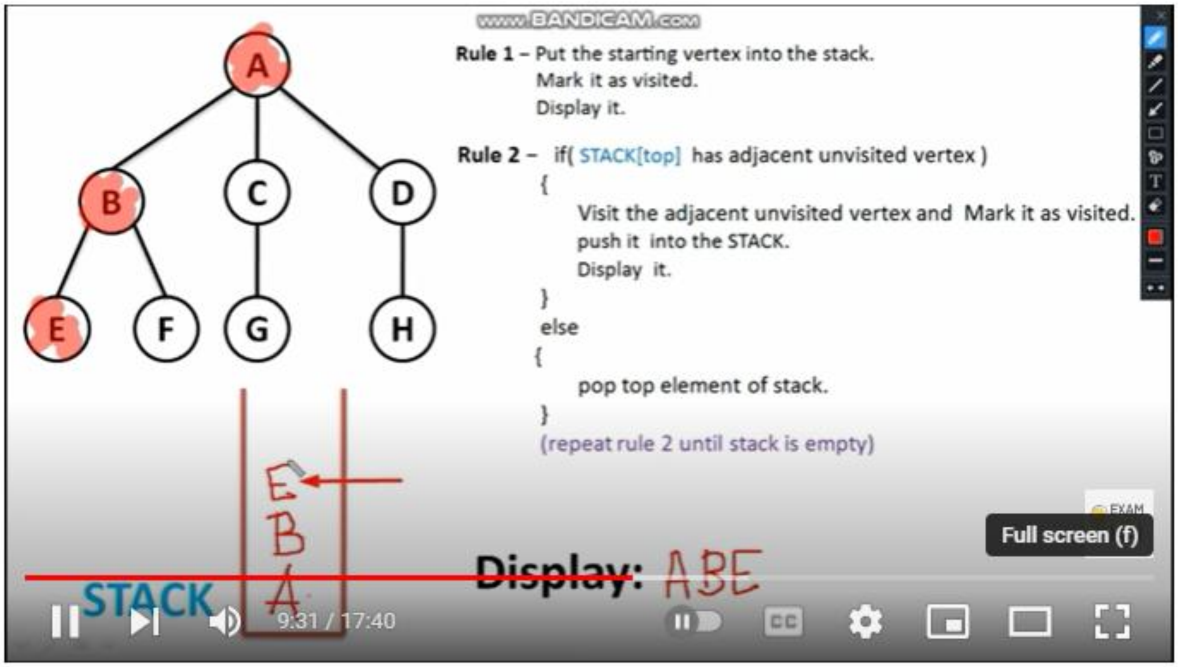
```
    }
}
void parallel_merge_sort(vector<int>& arr) {
#pragma omp parallel
    {
#pragma omp single
        merge_sort(arr, 0, arr.size() - 1);
    }
}
int main() {
    vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
    double start, end;
    // Measure performance of sequential merge sort
    start = omp_get_wtime();
    merge_sort(arr, 0, arr.size() - 1);
    end = omp_get_wtime();
    cout << "Sequential merge sort time: " << end - start <<endl;
    // Measure performance of parallel merge sort
    arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
    start = omp_get_wtime();
    parallel_merge_sort(arr);
    end = omp_get_wtime();
    return 0;
    }
```


Activities
Terminal
May 8 13:53
tanishka@tanishka-Lenovo: ~
tanishka@tanishka-Lenovo:~$ g++ -o msort -fopenmp msort.cpp
tanishka@tanishka-Lenovo:~$ ./msort
Sequential merge sort time: 3.1486e-05
tanishka@tanishka-Lenovo:~$

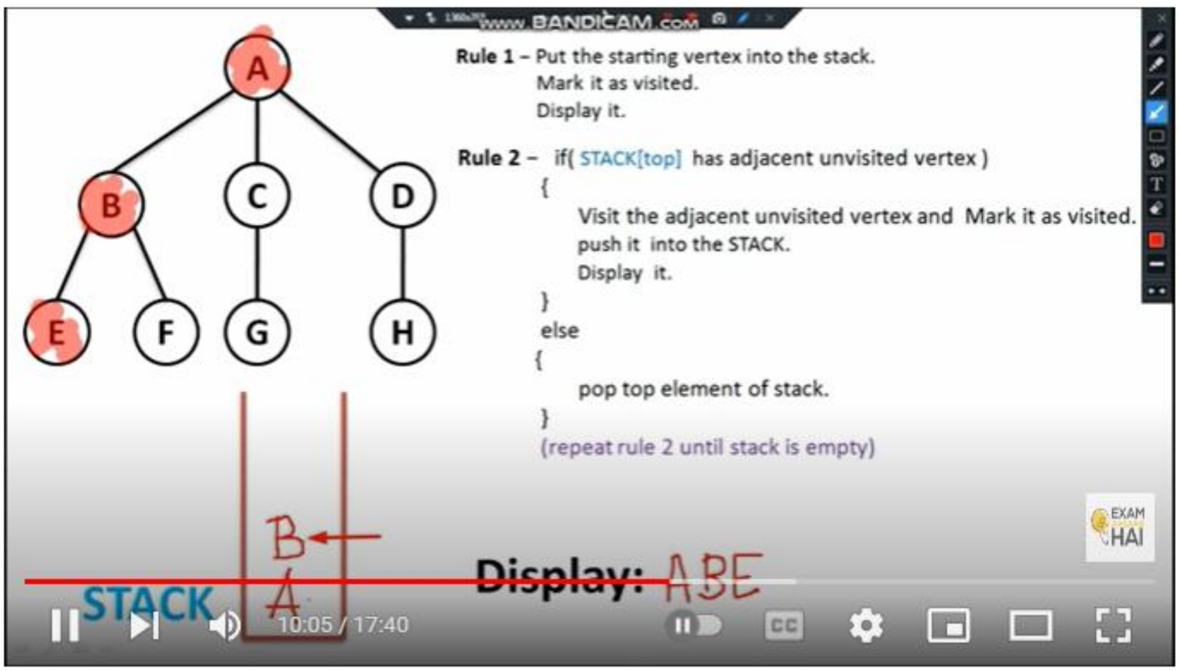
Aim: Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS.

DFS Concept -

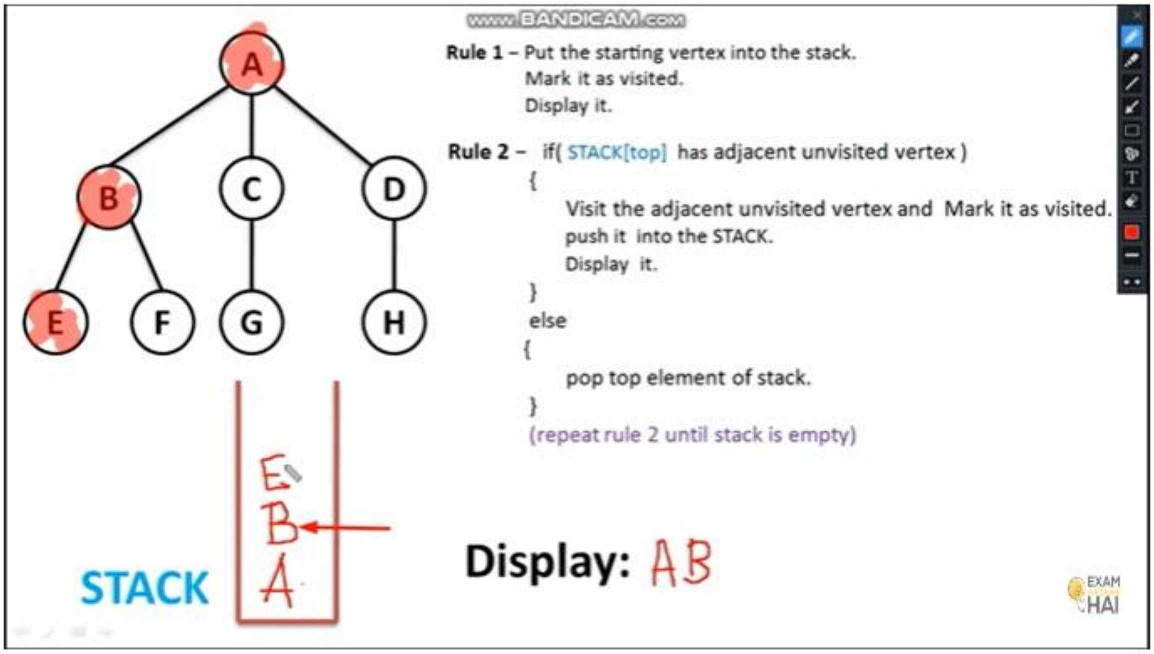


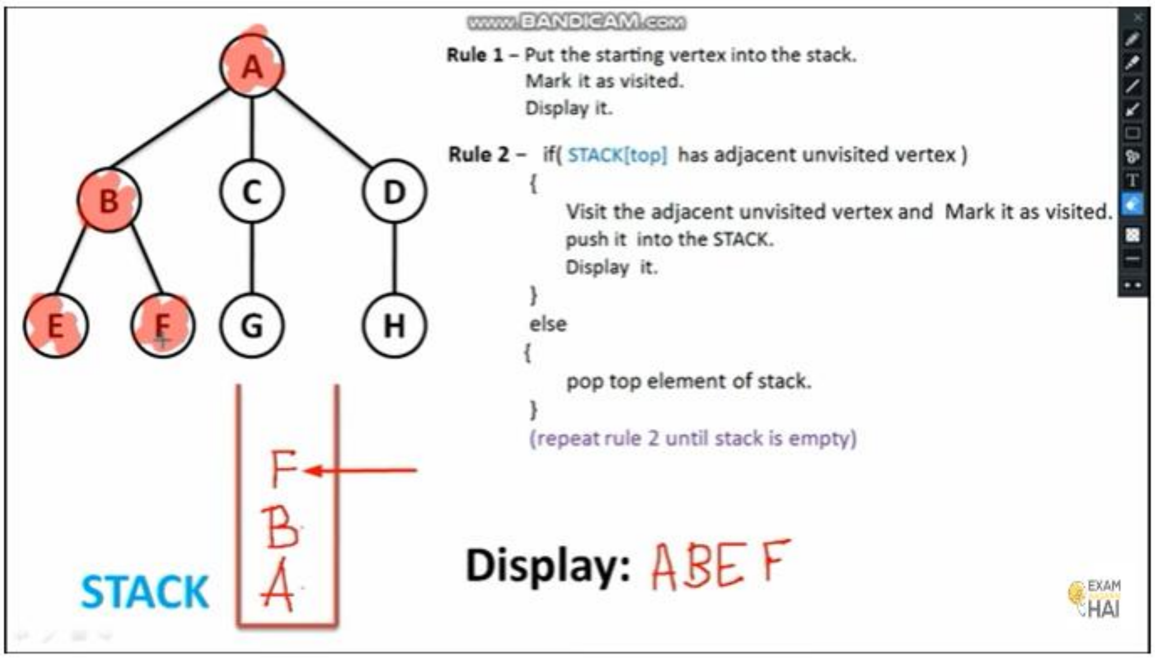


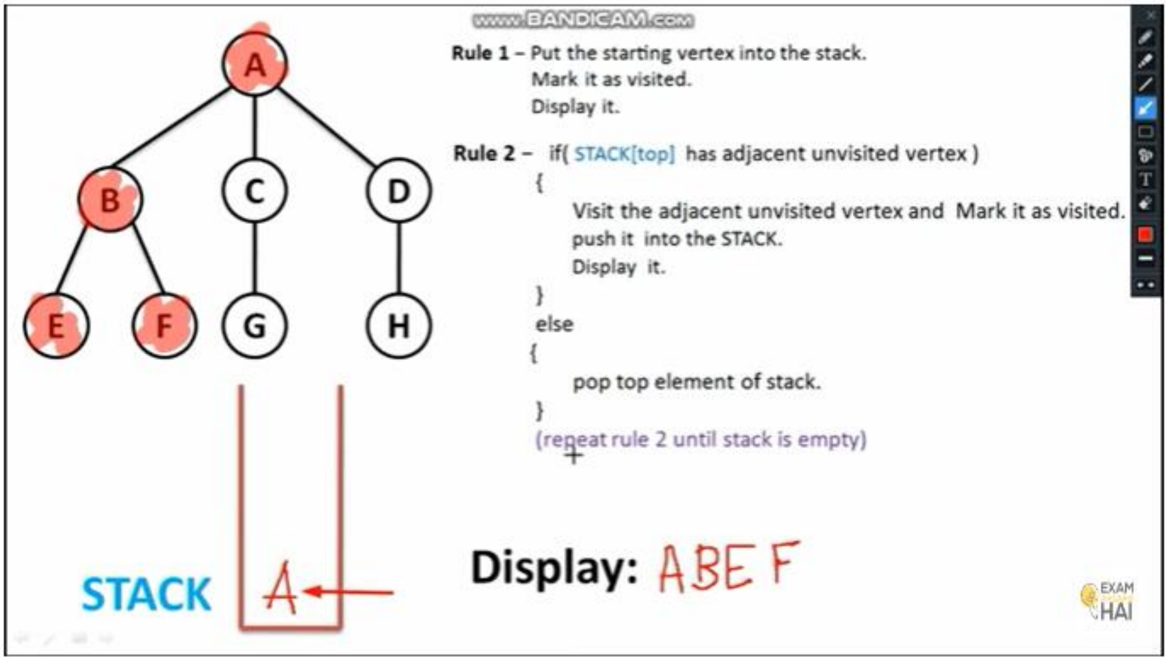
www.BANDICAM.com
Rule 1 – Put the starting vertex into the stack.
Mark it as visited.
Display it.
Rule 2 – if(STACK[top] has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
push it into the STACK.
Display it.
}
else
{
pop top element of stack.
}
(repeat rule 2 until stack is empty)
A
B
C
D
E
F
G
H
E
B
A
STACK
Display: ABE
Full screen (f)
9:31 / 17:40

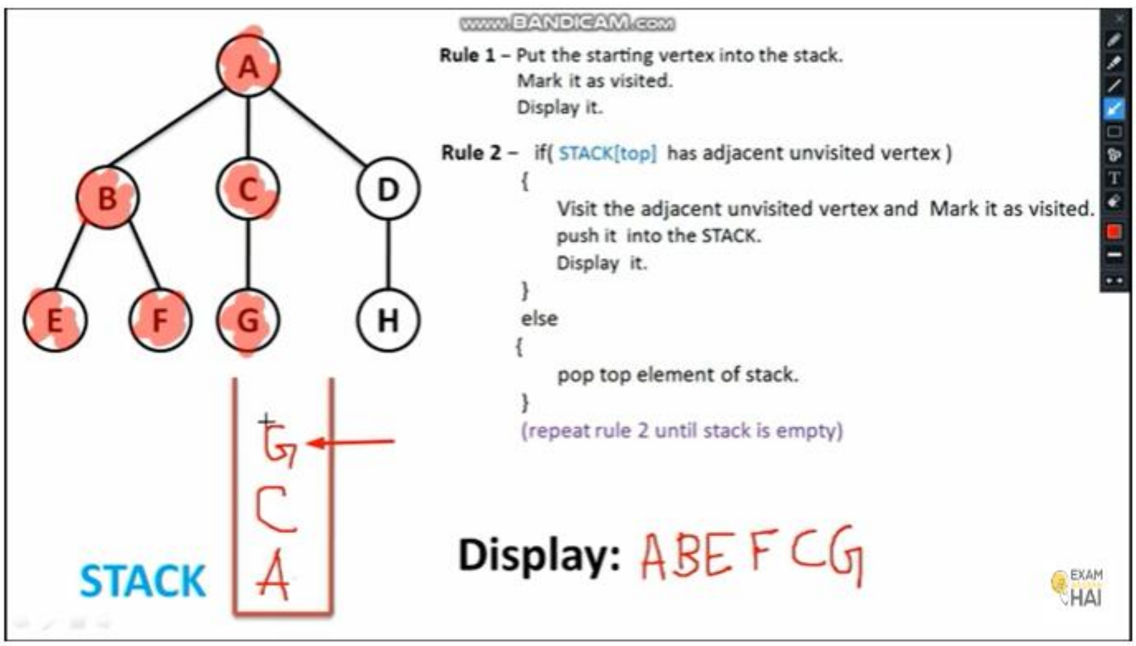


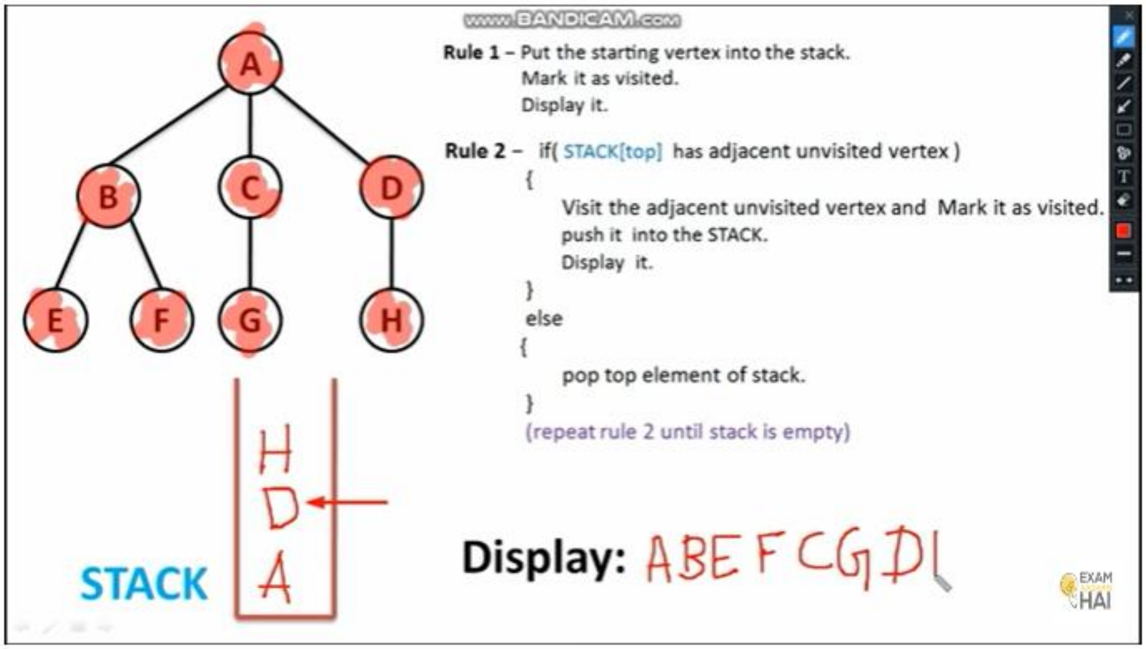
www.BANDICAM.com
Rule 1 – Put the starting vertex into the stack.
Mark it as visited.
Display it.
Rule 2 – if(STACK[top] has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
push it into the STACK.
Display it.
}
else
{
pop top element of stack.
}
(repeat rule 2 until stack is empty)
A
B
C
D
E
F
G
H
B
A
STACK
Display: ABE
EXAM HAI
10:05 / 17:40

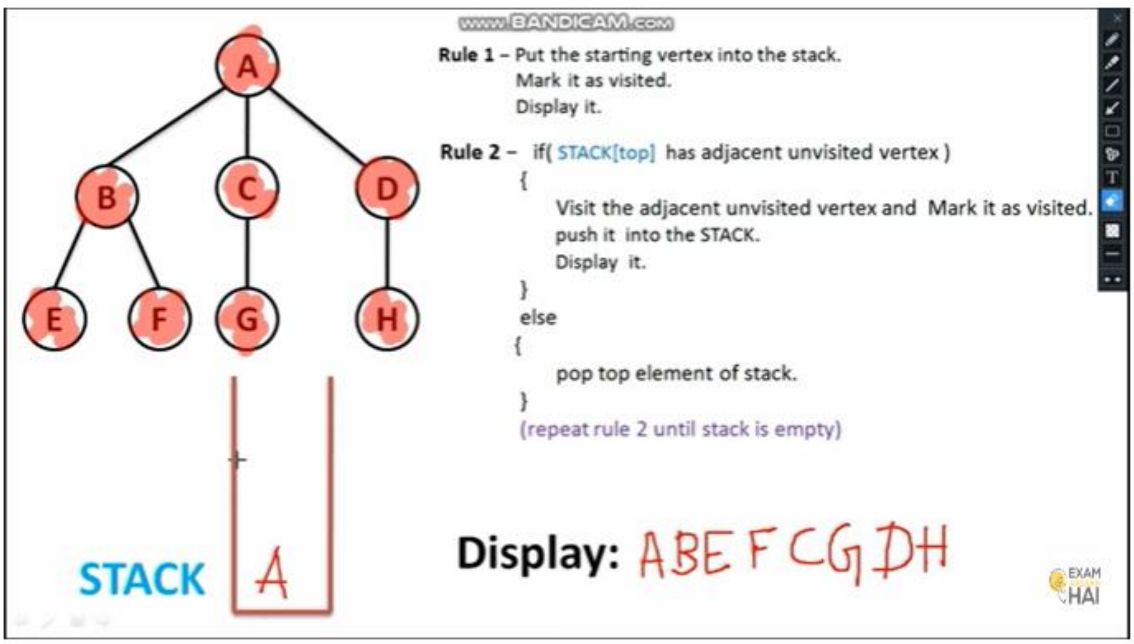

www.BANDICAM.com
A
B
C
D
E
F
G
H
Rule 1 – Put the starting vertex into the stack.
Mark it as visited.
Display it.
Rule 2 – if(STACK[top] has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
push it into the STACK.
Display it.
}
else
{
pop top element of stack.
}
(repeat rule 2 until stack is empty)
E
B
A
STACK
Display: AB
EXAM HAI

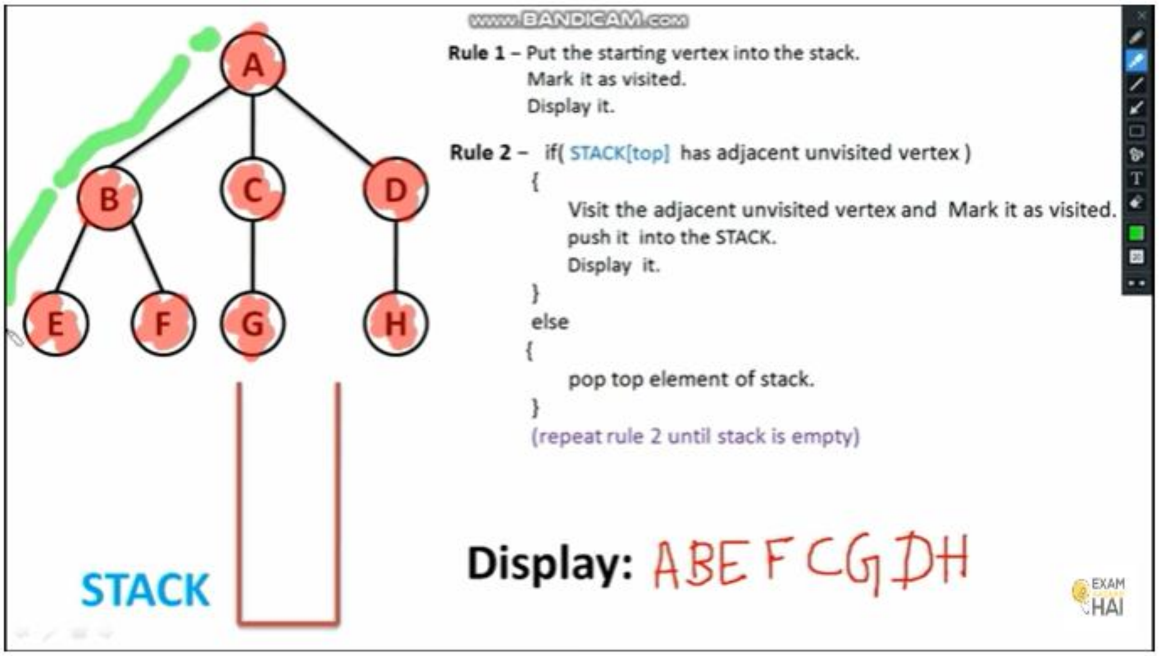

www.BANDICAM.com
A
B
C
D
E
F
G
H
Rule 1 – Put the starting vertex into the stack.
Mark it as visited.
Display it.
Rule 2 – if(STACK[top] has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
push it into the STACK.
Display it.
}
else
{
pop top element of stack.
}
(repeat rule 2 until stack is empty)
F
B
A
STACK
Display: ABEF
EXAM HAI


www.BANDICAM.com
Rule 1 – Put the starting vertex into the stack.
Mark it as visited.
Display it.
Rule 2 – if(STACK[top] has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
push it into the STACK.
Display it.
}
else
{
pop top element of stack.
}
(repeat rule 2 until stack is empty)
A
B
C
D
E
F
G
H
STACK
A
Display: ABEF
EXAM HAI


www.BANDICAM.com
Rule 1 – Put the starting vertex into the stack.
Mark it as visited.
Display it.
Rule 2 – if(STACK[top] has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
push it into the STACK.
Display it.
}
else
{
pop top element of stack.
}
(repeat rule 2 until stack is empty)
A
B
C
D
E
F
G
H
STACK
G
C
A
Display: ABEFCG
EXAM HAI


www.BANDICAM.com
A
B
C
D
E
F
G
H
Rule 1 – Put the starting vertex into the stack.
Mark it as visited.
Display it.
Rule 2 – if(STACK[top] has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
push it into the STACK.
Display it.
}
else
{
pop top element of stack.
}
(repeat rule 2 until stack is empty)
H
D
A
STACK
Display: ABEFCGD
EXAM HAI


www.BANDICAM.com
A
B
C
D
E
F
G
H
Rule 1 – Put the starting vertex into the stack.
Mark it as visited.
Display it.
Rule 2 – if(STACK[top] has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
push it into the STACK.
Display it.
}
else
{
pop top element of stack.
}
(repeat rule 2 until stack is empty)
A
STACK
Display: ABEFCGDH
EXAM HAI

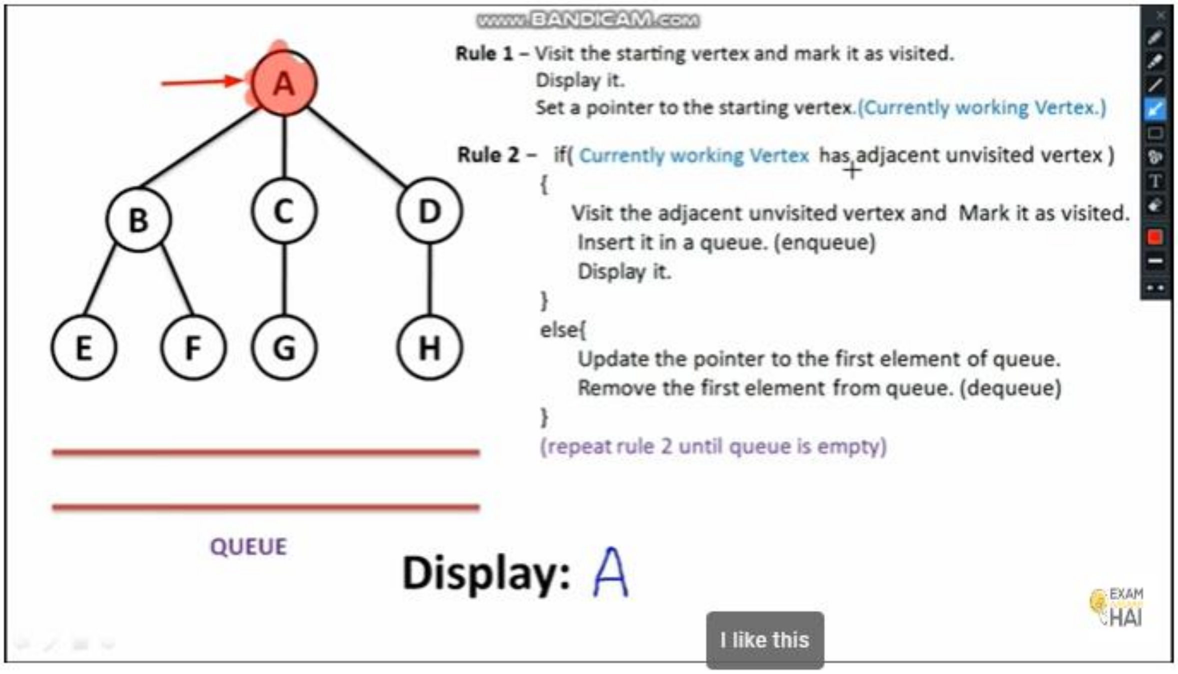


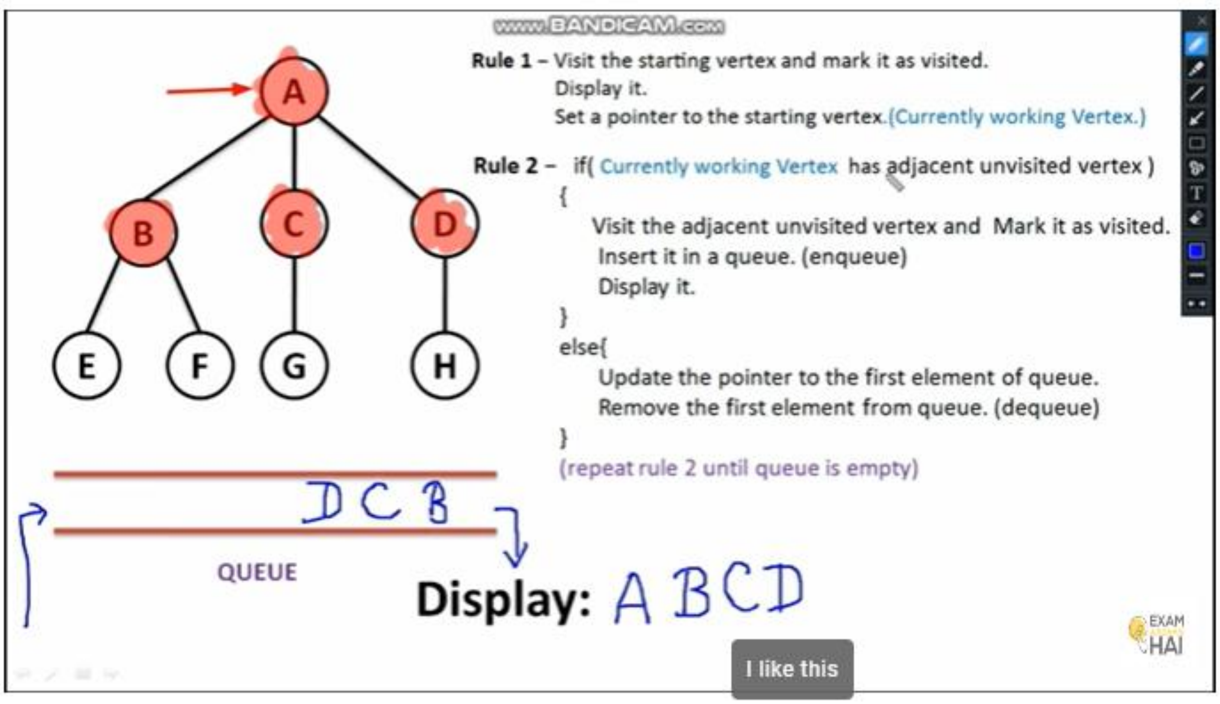
DFS Code-

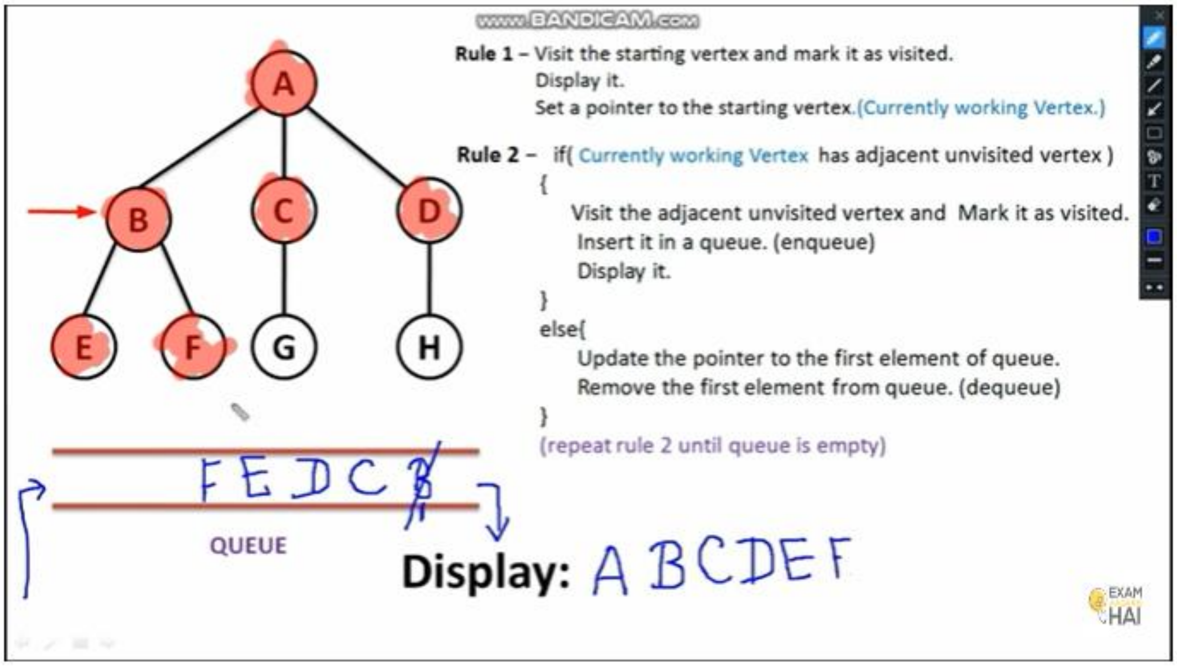
```
#include <iostream>
#include <vector>
#include <omp.h>
using namespace std;
const int MAXN = 1e5;
vector<int> adj[MAXN+5]; // adjacency list
bool visited[MAXN+5]; // mark visited nodes
void dfs(int node) {
    visited[node] = true;
    #pragma omp parallel for
    for (int i = 0; i < adj[node].size(); i++) {
        int next_node = adj[node][i];
        if (!visited[next_node]) {
            dfs(next_node);
        }
    }
}
int main() {
    cout << "Please enter nodes and edges";
    int n, m; // number of nodes and edges
    cin >> n >> m;
```

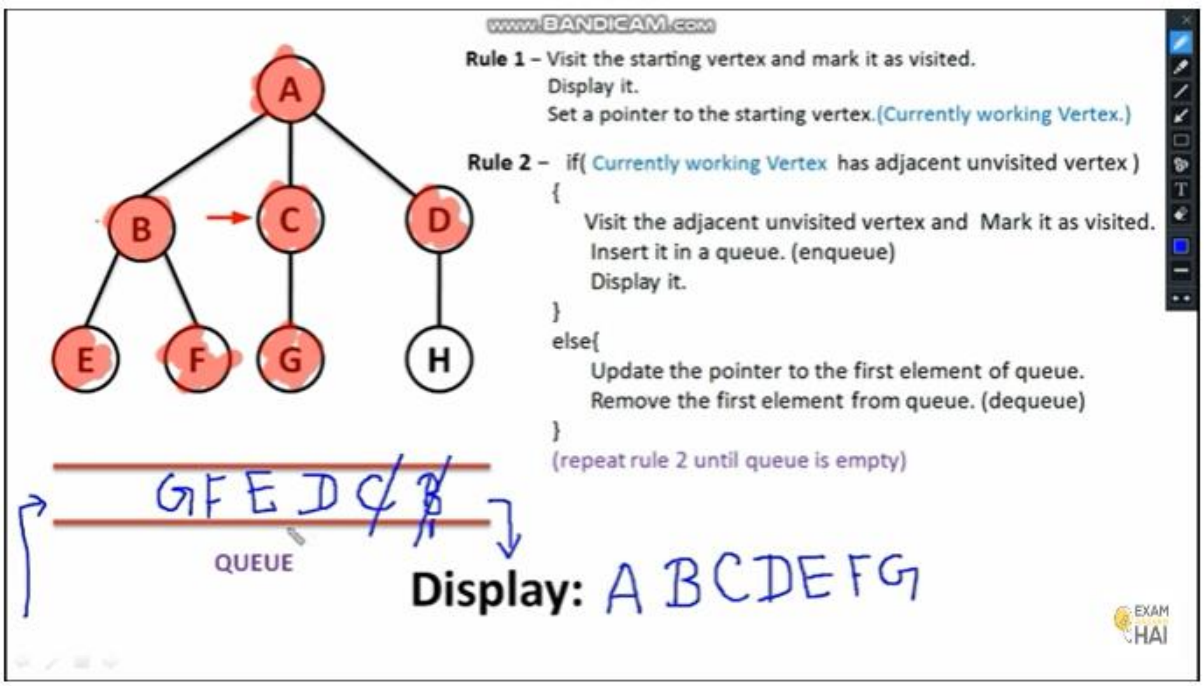
```
    for (int i = 1; i <= m; i++) {
        int u, v; // edge between u and v
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    int start_node; // start node of DFS
    cin >> start_node;

    dfs(start_node);

    // Print visited nodes
    for (int i = 1; i <= n; i++) {
        if (visited[i]) {
            cout << i << " ";
        }
    }
    cout << endl;

    return 0;
}
```

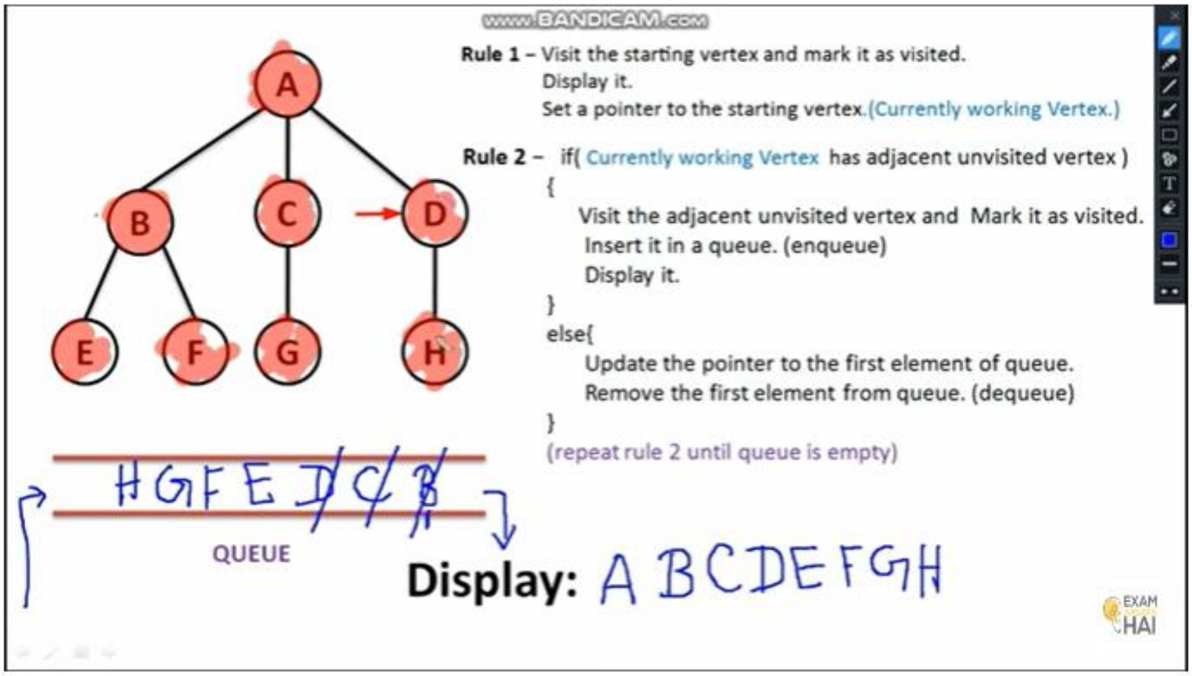
BFS Concept:

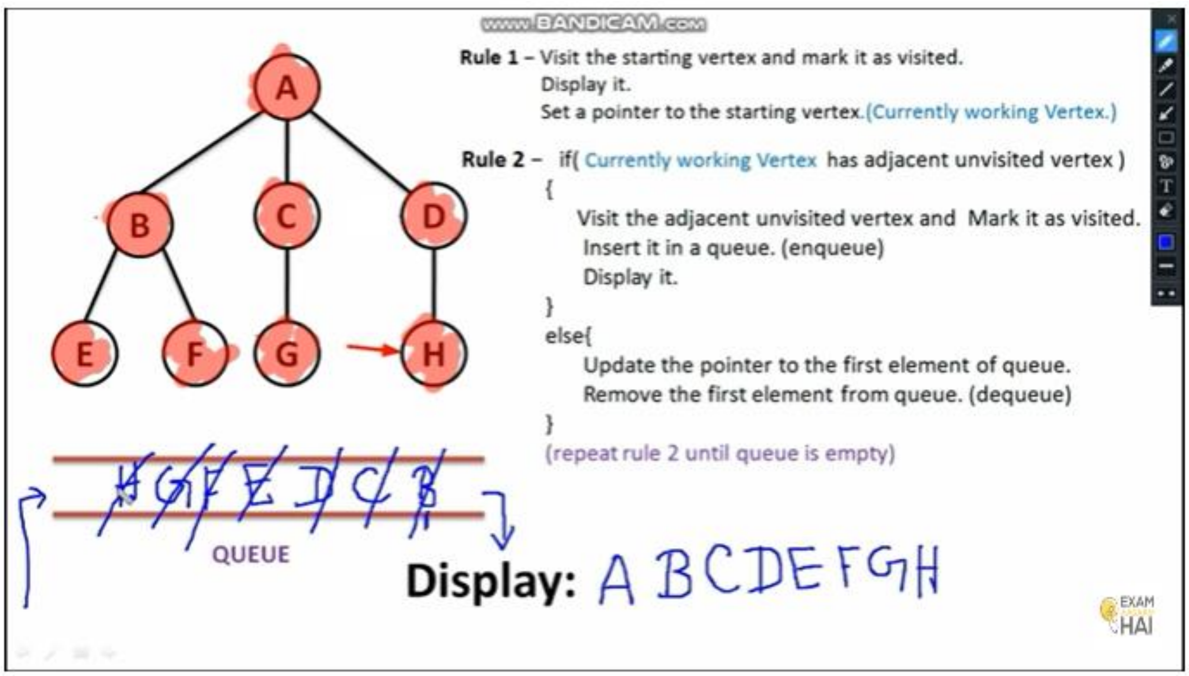

www.BANDICAM.com
Rule 1 – Visit the starting vertex and mark it as visited.
Display it.
Set a pointer to the starting vertex.(Currently working Vertex.)
Rule 2 – if(Currently working Vertex has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
Insert it in a queue. (enqueue)
Display it.
}
else{
Update the pointer to the first element of queue.
Remove the first element from queue. (dequeue)
}
(repeat rule 2 until queue is empty)
A
B
C
D
E
F
G
H
QUEUE
Display: A
I like this
EXAM HAI

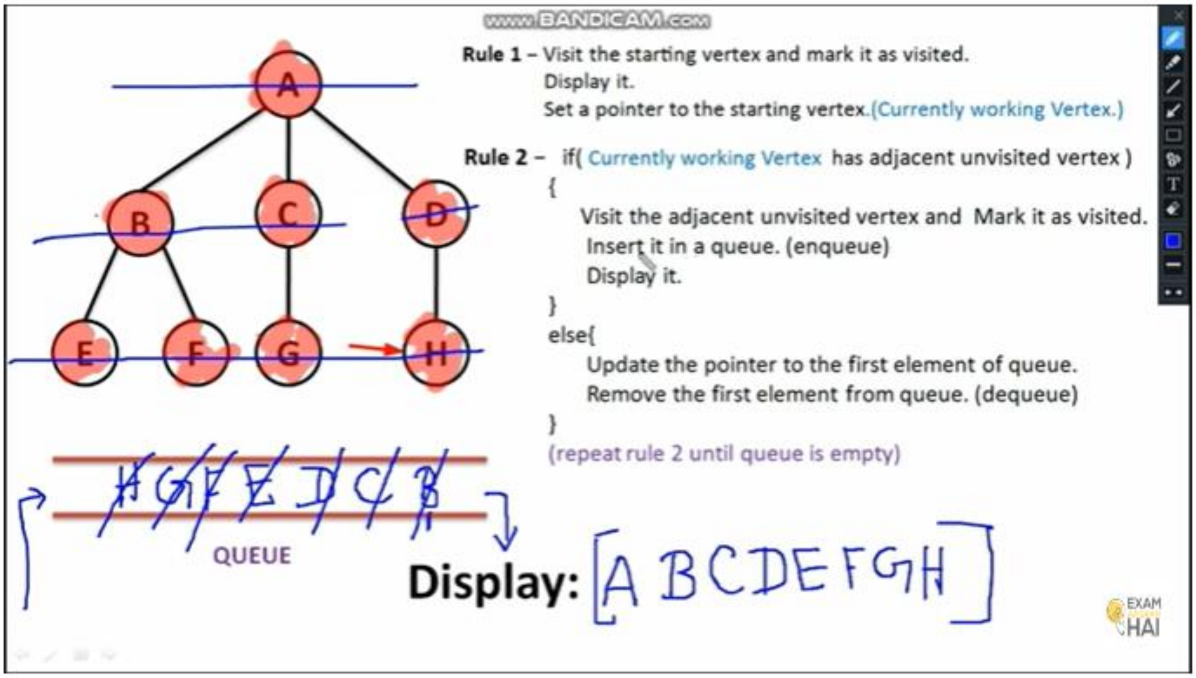

www.BANDICAM.com
Rule 1 – Visit the starting vertex and mark it as visited.
Display it.
Set a pointer to the starting vertex.(Currently working Vertex.)
Rule 2 – if(Currently working Vertex has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
Insert it in a queue. (enqueue)
Display it.
}
else{
Update the pointer to the first element of queue.
Remove the first element from queue. (dequeue)
}
(repeat rule 2 until queue is empty)
A
B
C
D
E
F
G
H
D C B
QUEUE
Display: A B C D
I like this
EXAM HAI


www.BANDICAM.com
A
B
C
D
E
F
G
H
Rule 1 – Visit the starting vertex and mark it as visited.
Display it.
Set a pointer to the starting vertex.(Currently working Vertex.)
Rule 2 – if(Currently working Vertex has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
Insert it in a queue. (enqueue)
Display it.
}
else{
Update the pointer to the first element of queue.
Remove the first element from queue. (dequeue)
}
(repeat rule 2 until queue is empty)
F E D C ~~B~~
QUEUE
Display: A B C D E F
EXAM HAI


www.BANDICAM.com
A
B
C
D
E
F
G
H
Rule 1 – Visit the starting vertex and mark it as visited.
Display it.
Set a pointer to the starting vertex.(Currently working Vertex.)
Rule 2 – if(Currently working Vertex has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
Insert it in a queue. (enqueue)
Display it.
}
else{
Update the pointer to the first element of queue.
Remove the first element from queue. (dequeue)
}
(repeat rule 2 until queue is empty)
G F E D ~~C~~ ~~B~~
QUEUE
Display: A B C D E F G
EXAM HAI


www.BANDICAM.com
Rule 1 – Visit the starting vertex and mark it as visited.
Display it.
Set a pointer to the starting vertex.(Currently working Vertex.)
Rule 2 – if(Currently working Vertex has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
Insert it in a queue. (enqueue)
Display it.
}
else{
Update the pointer to the first element of queue.
Remove the first element from queue. (dequeue)
}
(repeat rule 2 until queue is empty)
A
B
C
D
E
F
G
H
H G F E D C B
QUEUE
Display: A B C D E F G H
EXAM HAI


www.BANDICAM.com
Rule 1 – Visit the starting vertex and mark it as visited.
Display it.
Set a pointer to the starting vertex.(Currently working Vertex.)
Rule 2 – if(Currently working Vertex has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
Insert it in a queue. (enqueue)
Display it.
}
else{
Update the pointer to the first element of queue.
Remove the first element from queue. (dequeue)
}
(repeat rule 2 until queue is empty)
A
B
C
D
E
F
G
H
H G F E D C B
QUEUE
Display: A B C D E F G H
EXAM HAI



BFS Code-

```
#include <iostream>
#include <queue>
#include <vector>
#include <omp.h>

using namespace std;

int main() {
    int num_vertices, num_edges, source;
    cin >> num_vertices >> num_edges >> source;

    vector<vector<int>> adj_list(num_vertices + 1);
    for (int i = 0; i < num_edges; i++) {
        int u, v;
        cin >> u >> v;
        adj_list[u].push_back(v);
        adj_list[v].push_back(u);
    }

    queue<int> q;
```

```
    vector<bool> visited(num_vertices + 1, false);

    q.push(source);
    visited[source] = true;

    while (!q.empty()) {
        int curr_vertex = q.front();
        q.pop();

        cout << curr_vertex << " ";

        #pragma omp parallel for shared(adj_list, visited, q) schedule(dynamic)
        for (int i = 0; i < adj_list[curr_vertex].size(); i++) {
            int neighbour = adj_list[curr_vertex][i];
            if (!visited[neighbour]) {
                visited[neighbour] = true;
                q.push(neighbour);
            }
        }
    }

    return 0;
}
```

```
tanishka@tanishka-Lenovo:~$ cat >bfs8.cpp
#include <iostream>
#include <queue>
#include <vector>
#include <omp.h>
#include <cstring>

using namespace std;

const int MAXN = 100000;

vector<int> graph[MAXN+1];
int dist[MAXN+1];

void bfs(int start)
{
    queue<int> q;
    q.push(start);
    dist[start] = 0;

    while(!q.empty())
    {
        int u = q.front();
        q.pop();

        #pragma omp parallel for
        for(int i=0; i<graph[u].size(); i++)
        {
            int v = graph[u][i];
            if(dist[v] == -1)
            {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
```

```
tanishka@tanishka-Lenovo:~$ ./bfs9
6 8
0
0 1
0 2
1 2
2 0
2 3
3 3
1 4
4 5
0 2 0 3 tanishka@tanishka-Lenovo:~$ ./bfs9
5 7
0 1
0 2
1 2
2 0
2 3
3 3
4 4
0
0 4 0 1 3 2 tanishka@tanishka-Lenovo:~$ ./bfs9
4 4
1
1 2
2 4
1 3
3 4
1 3 0 4 tanishka@tanishka-Lenovo:~$ ./bfs9
4 4
0
0 1
1 3
0 2
2 3
0 2 1 3 tanishka@tanishka-Lenovo:~$
```

Aim: Write a CUDA Program for :

1. Addition of two large vectors

2. Matrix Multiplication using CUDA C

1. Vector Addition Program

```
!nvcc --version
!pip install git+https://github.com/afnan47/cuda.git
%load_ext nvcc_plugin

%%cu
#include <iostream>
using namespace std;
__global__
void add(int* A, int* B, int* C, int size) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    if (tid < size) {
        C[tid] = A[tid] + B[tid];
    }
}
void initialize(int* vector, int size) {
    for (int i = 0; i < size; i++) {
        vector[i] = rand() % 10;
    }
}
void print(int* vector, int size) {
    for (int i = 0; i < size; i++) {
        cout << vector[i] << " ";
    }
    cout << endl;
}
int main() {
    int N = 4;
```

```
    int* A, * B, * C;
    int vectorSize = N;
    size_t vectorBytes = vectorSize * sizeof(int);
    A = new int[vectorSize];
    B = new int[vectorSize];
    C = new int[vectorSize];
  initialize(A, vectorSize);
    initialize(B, vectorSize);
    cout << "Vector A: ";
    print(A, N);
    cout << "Vector B: ";
    print(B, N);
    int* X, * Y, * Z;
    cudaMalloc(&X, vectorBytes);
    cudaMalloc(&Y, vectorBytes);
    cudaMalloc(&Z, vectorBytes);
    cudaMemcpy(X, A, vectorBytes, cudaMemcpyHostToDevice);
    cudaMemcpy(Y, B, vectorBytes, cudaMemcpyHostToDevice);
    int threadsPerBlock = 256;
    int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;
    add<<<blocksPerGrid, threadsPerBlock>>>(X, Y, Z, N);
    cudaMemcpy(C, Z, vectorBytes, cudaMemcpyDeviceToHost);
    cout << "Addition: ";
    print(C, N);
    delete[] A;
    delete[] B;
    delete[] C;
    cudaFree(X);
    cudaFree(Y);
    cudaFree(Z);
    return 0;
}
```

Output:

Vector A: 3 6 7 5

Vector B: 3 5 6 2

Addition: 6 11 13 7

2. Matrix multiplication

```
!nvcc --version

!pip install git+https://github.com/afnan47/cuda.git
%load_ext nvcc_plugin




%%cu
#include <iostream>
#include <cuda.h>

using namespace std;

#define BLOCK_SIZE 2

__global__ void gpuMM(float *A, float *B, float *C, int N)
{
    // Matrix multiplication for NxN matrices C=A*B
    // Each thread computes a single element of C
    int row = blockIdx.y*blockDim.y + threadIdx.y;
    int col = blockIdx.x*blockDim.x + threadIdx.x;

    float sum = 0.f;
```

```
    for (int n = 0; n < N; ++n)
        sum += A[row*N+n]*B[n*N+col];

    C[row*N+col] = sum;
}

int main(int argc, char *argv[])
{int N;float K;
    // Perform matrix multiplication C = A*B
    // where A, B and C are NxN matrices
    // Restricted to matrices where N = K*BLOCK_SIZE;
  cout<<"Enter a Value for Size/2 of matrix";
  cin>>K;

    K = 1;
    N = K*BLOCK_SIZE;

    cout << "\n Executing Matrix Multiplcation" << endl;
    cout << "\n Matrix size: " << N << "x" << N << endl;

    // Allocate memory on the host
    float *hA,*hB,*hC;
    hA = new float[N*N];
    hB = new float[N*N];
    hC = new float[N*N];

    // Initialize matrices on the host
    for (int j=0; j<N; j++){
        for (int i=0; i<N; i++){
            hA[j*N+i] = 2;
            hB[j*N+i] = 4;


        }
    }
```

```
// Allocate memory on the device
int size = N*N*sizeof(float);    // Size of the memory in bytes
float *dA,*dB,*dC;
cudaMalloc(&dA,size);
cudaMalloc(&dB,size);
cudaMalloc(&dC,size);

dim3 threadBlock(BLOCK_SIZE,BLOCK_SIZE);
dim3 grid(K,K);
cout<<"\n Input Matrix 1 \n";
for (int row=0; row<N; row++){
        for (int col=0; col<N; col++){

                cout<<hA[row*col]<<" ";

        }
        cout<<endl;
    }
cout<<"\n Input Matrix 2 \n";
for (int row=0; row<N; row++){
        for (int col=0; col<N; col++){

                cout<<hB[row*col]<<" ";

        }
        cout<<endl;
    }
// Copy matrices from the host to device
cudaMemcpy(dA,hA,size,cudaMemcpyHostToDevice);
cudaMemcpy(dB,hB,size,cudaMemcpyHostToDevice);

//Execute the matrix multiplication kernel

gpuMM<<<grid,threadBlock>>>(dA,dB,dC,N);
```

```
    // Now do the matrix multiplication on the CPU
   /*float sum;
    for (int row=0; row<N; row++){
        for (int col=0; col<N; col++){
            sum = 0.f;
            for (int n=0; n<N; n++){
                sum += hA[row*N+n]*hB[n*N+col];
            }
            hC[row*N+col] = sum;
            cout << sum <<" ";

        }
        cout<<endl;
    }*/

 // Allocate memory to store the GPU answer on the host
    float *C;
    C = new float[N*N];

    // Now copy the GPU result back to CPU
    cudaMemcpy(C,dC,size,cudaMemcpyDeviceToHost);

    // Check the result and make sure it is correct
    cout <<"\n\n\n\n\n Resultant matrix\n\n";
    for (int row=0; row<N; row++){
        for (int col=0; col<N; col++){

               cout<<C[row*col]<<"  ";

        }
        cout<<endl;
    }

    cout << "Finished." << endl;
}
```

Output-

Matrix size: 2x2

Input Matrix 1

2 2

2 2

Input Matrix 2

4 4

4 4

Resultant matrix

16 16

16 16

Finished.