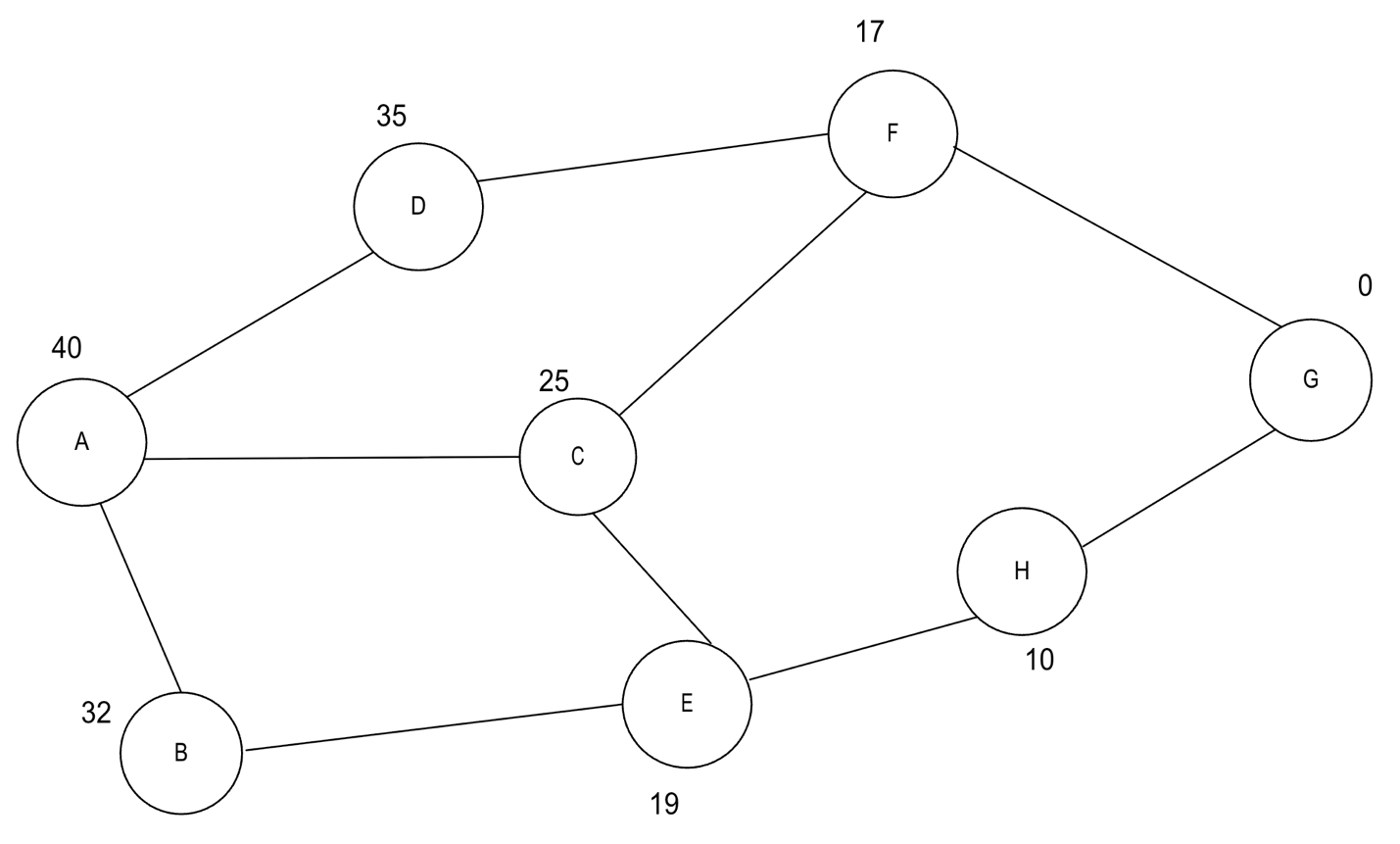
**BEST FIRST SEARCH ALGORITHM**

**Best First Search** is a **graph traversal and path finding algorithm** that explores a graph by **expanding the most promising node first**, based on a **heuristic function h(n)** that estimates the cost to reach the goal.

**Heuristic Function**: This is a function that estimates the **minimum cost** or distance from a given node N to the goal node.

**Example:**

Suppose we have that graph as below where A is the starting node and G is the Goal node or ending node.



Here the value written with every node is the estimated cost to reach the goal from that particular node.

First start with the node “A” and watch out its Childs -> D (35), B( 2), C (25)

Choose the closest path which is closest to the goal “G”.

Select C (25) which is closer to the goal “G”. [ D has cost 35, B has 32]

Now from C, watch out its childs -> E (19), F (17)

Select F (17) which is closer to the goal “G”. [ E has cost 19]

Now from F Watch out childs -> D (35), G (0)

As one of the childs is our goal node, we stop iterating here.

Now Write down selected edges serially to find out the actual path according to the best first search algorithm.

A->C->F->G

**CODE:**

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

struct node{

int id;

int heuristic\_value;

vector<node\*> neighbors;

node(int id,int heuristic\_value){

this->id = id;

this->heuristic\_value = heuristic\_value;

}

};

bool compareByHeuristicValue(node\* node1, node\* node2){return node1->heuristic\_value < node2->heuristic\_value;}

void bfs(node\* start,node\* goal,int total\_nodes){

vector<node\*> nodes;

vector<bool> isVisited(total\_nodes+1 , false); // isVisited name er array jetar by default all values false

nodes.push\_back(start);

while(!nodes.empty()){

auto closest = min\_element(nodes.begin(),nodes.end(),compareByHeuristicValue);

node\* current = \*closest;

nodes.erase(closest);// remove kora hoise jate bar bar ei same node ke niye experiment na kora hoy

cout<<"Now visiting : "<<current->id<<endl;

cout<<"Heuristic Value : "<<current->heuristic\_value;

if(current == goal){

cout<<"\nGoal Found!"<<endl;

cout<<goal->id<<endl;

return;

}

isVisited[current->id+1] = true;

for(int i=0;i< current->neighbors.size();i++){

node\* t = current->neighbors[i];

if(!isVisited[t->id+1]){ // checking if the neighbor is already visited or not

nodes.push\_back(t);

}

}

cout<<endl;

//cout<<"\tGoal Not Found"<<endl;

cout<<" "<<current->id<<"-> ";// to keep track of paths

}

}

int main(){

node a(1,40);

node b(2,32);

node c(3,25);

node d(4,35);

node e(5,19);

node f(6,17);

node g(7,0);

node h(8,10);

a.neighbors = {&d,&c,&b};

b.neighbors = {&a,&e};

c.neighbors = {&a,&e,&f};

d.neighbors = {&a,&f};

e.neighbors = {&c,&b,&h};

f.neighbors = {&d,&g,&c};

g.neighbors = {&h,&f};

h.neighbors = {&e,&g};

bfs(&a,&g,8);

}

Code Explanation:

1. First create a structure named “node” to hold the id and heuristic values.
2. Create vector named neighbors to keep track of the neighbors of a node.
3. Inside the bfs function create 2 vectors named “nodes” type of \*node and “isVisited” type of boolean. Set the by default all values of isVisited function = false.

Set false because we havent visited any of the nodes nor the neighbors.

1. Bfs fucntion will receive 3 parameters node\* start and node\* goal, int total\_nodes.
2. Insert the start node into the “nodes” vector.
3. Start loop with closing condition of !nodes=empty().
4. Use auto iterator to sort the nodes vector according to the heuristic values.
5. Store the location which node having minimum heuristic value and remove that actual node to avoid repeated checking of that same node. Set the isVisited vector`s index true.
6. Check if that pointer pointing to the goal node or not. If true print goal found and return.
7. Using loop check if the neighbors of that current node is visited or not. If unvisited, insert that into nodes vector to repeat the same whole process.

OUTPUT:

