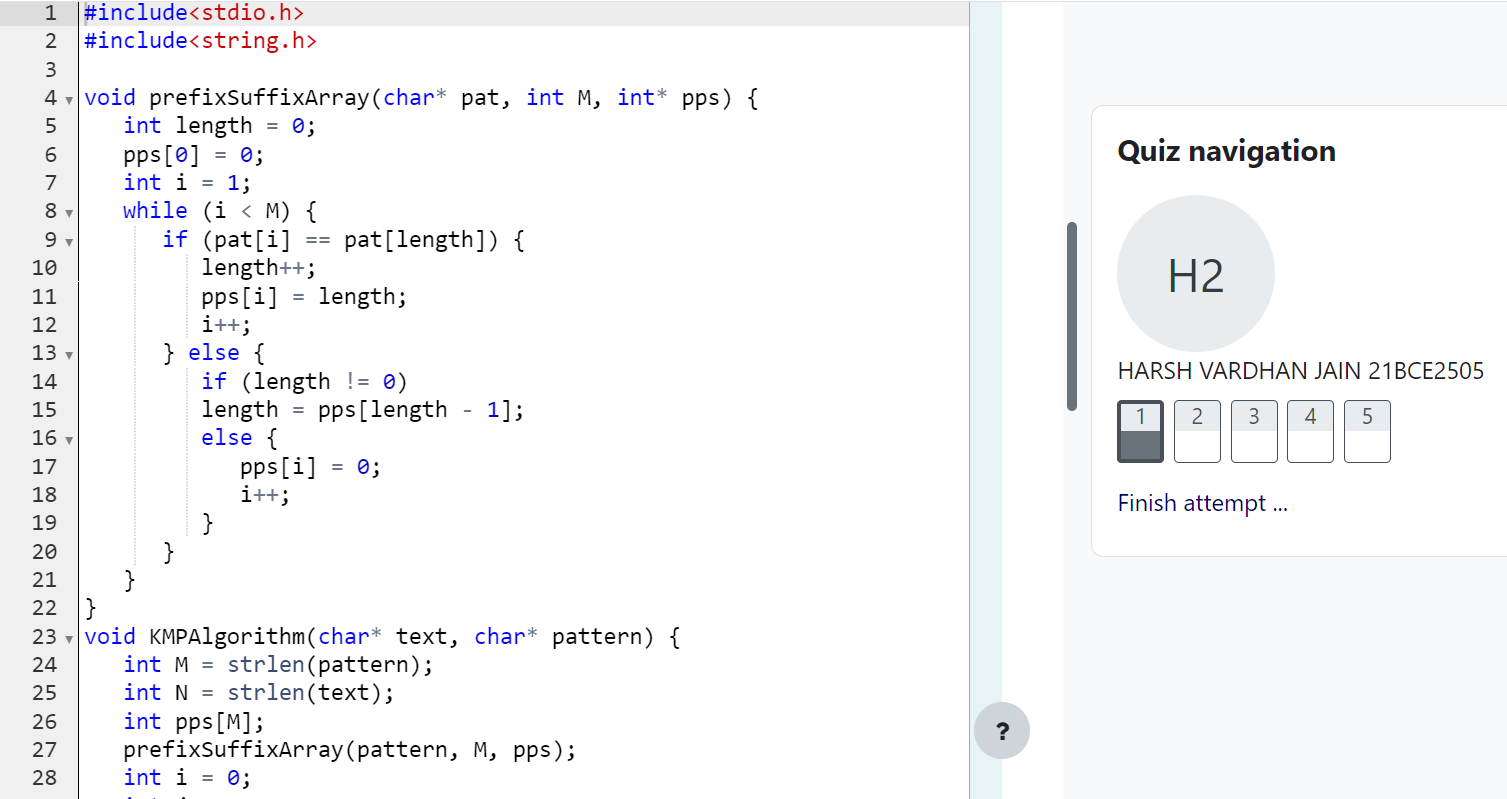
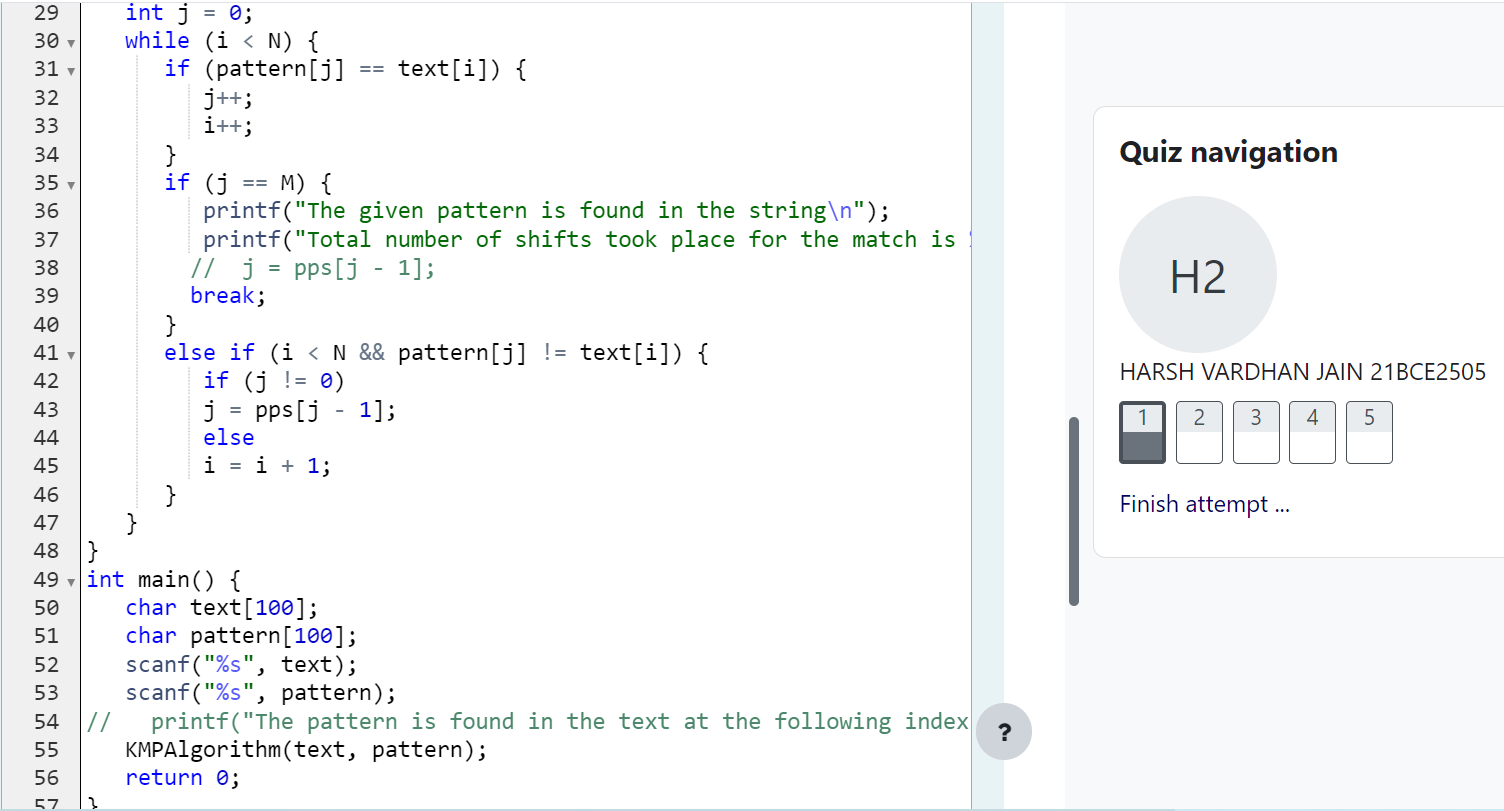
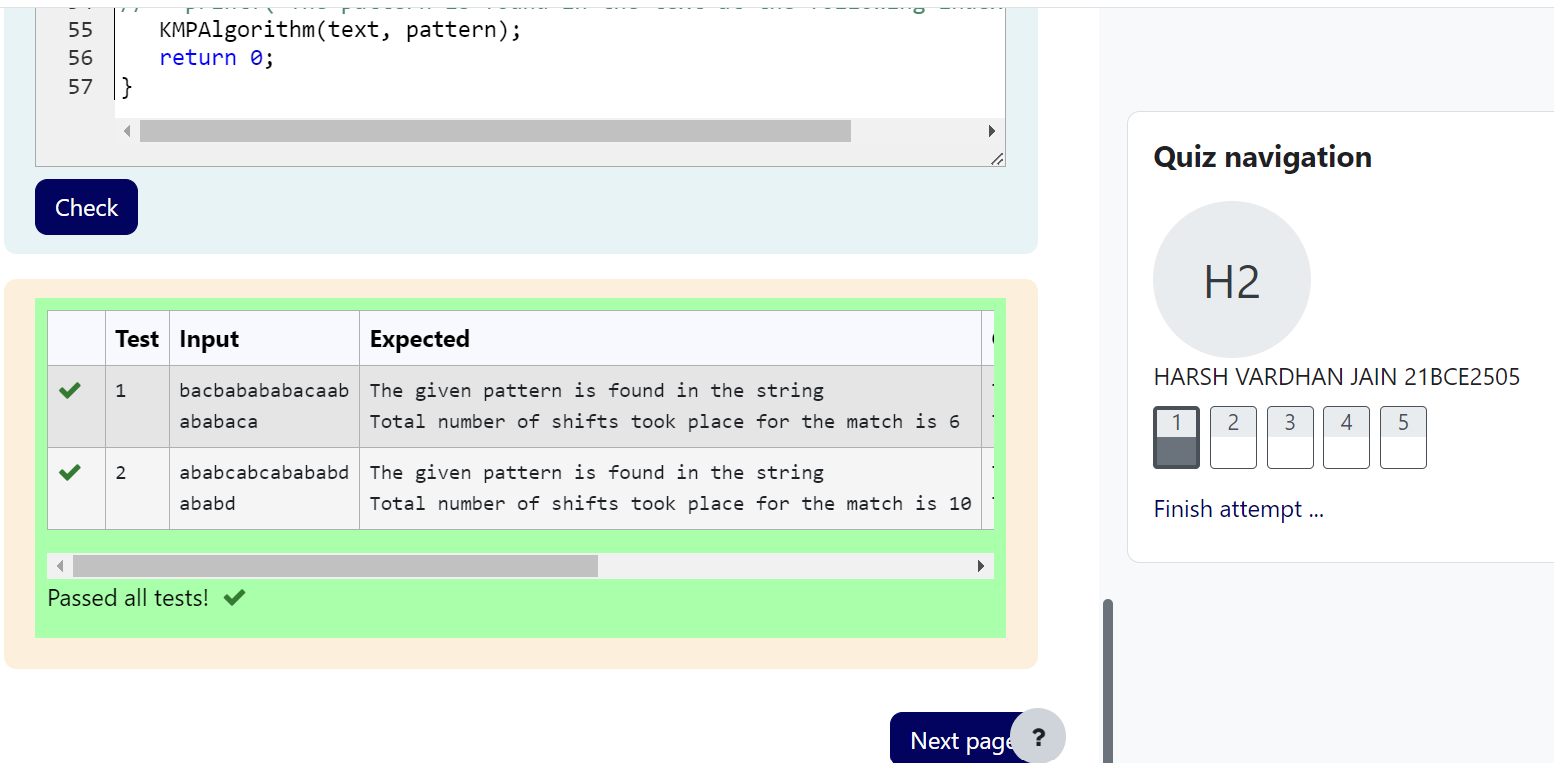
Name: Harsh Vardhan Jain

Reg. No.: 21BCE2505

Q1) 





Code

#include<stdio.h>

#include<string.h>

void prefixSuffixArray(char\* pat, int M, int\* pps) {

int length = 0;

pps[0] = 0;

int i = 1;

while (i < M) {

if (pat[i] == pat[length]) {

length++;

pps[i] = length;

i++;

} else {

if (length != 0)

length = pps[length - 1];

else {

pps[i] = 0;

i++;

}

}

}

}

void KMPAlgorithm(char\* text, char\* pattern) {

int M = strlen(pattern);

int N = strlen(text);

int pps[M];

prefixSuffixArray(pattern, M, pps);

int i = 0;

int j = 0;

while (i < N) {

if (pattern[j] == text[i]) {

j++;

i++;

}

if (j == M) {

printf("The given pattern is found in the string\n");

printf("Total number of shifts took place for the match is %d", i - j);

// j = pps[j - 1];

break;

}

else if (i < N && pattern[j] != text[i]) {

if (j != 0)

j = pps[j - 1];

else

i = i + 1;

}

}

}

int main() {

char text[100];

char pattern[100];

scanf("%s", text);

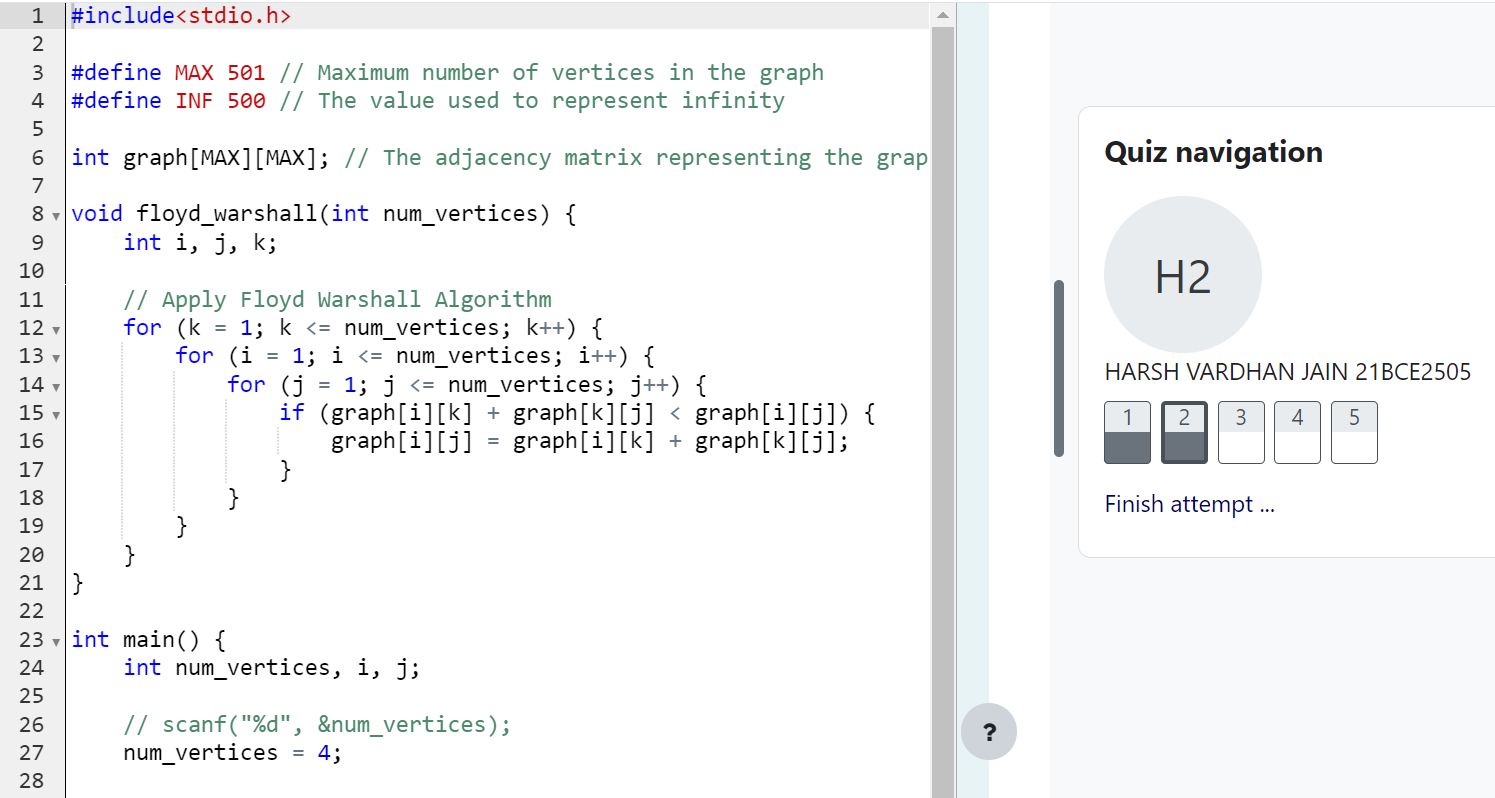
scanf("%s", pattern);

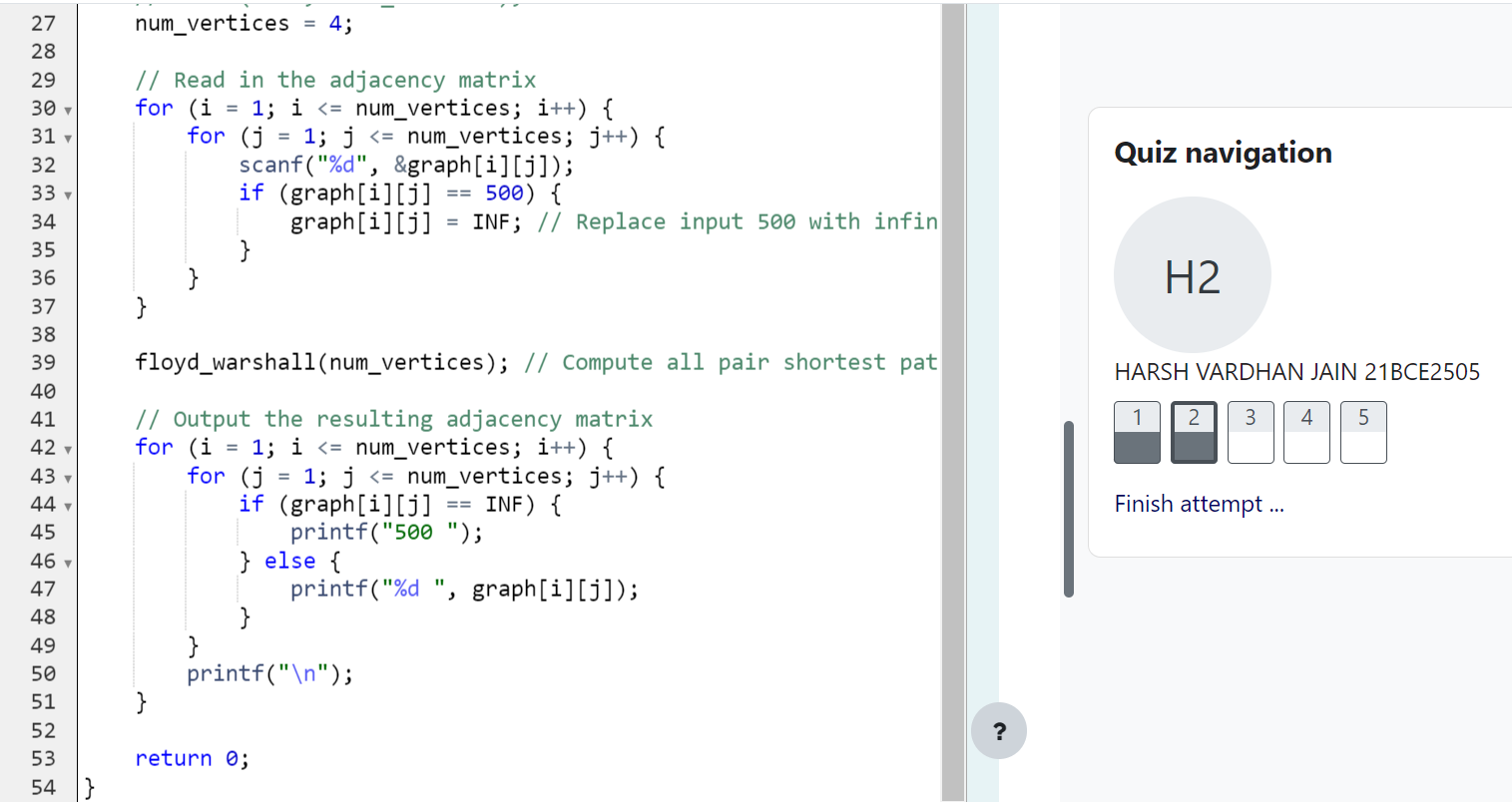
// printf("The pattern is found in the text at the following index : ");

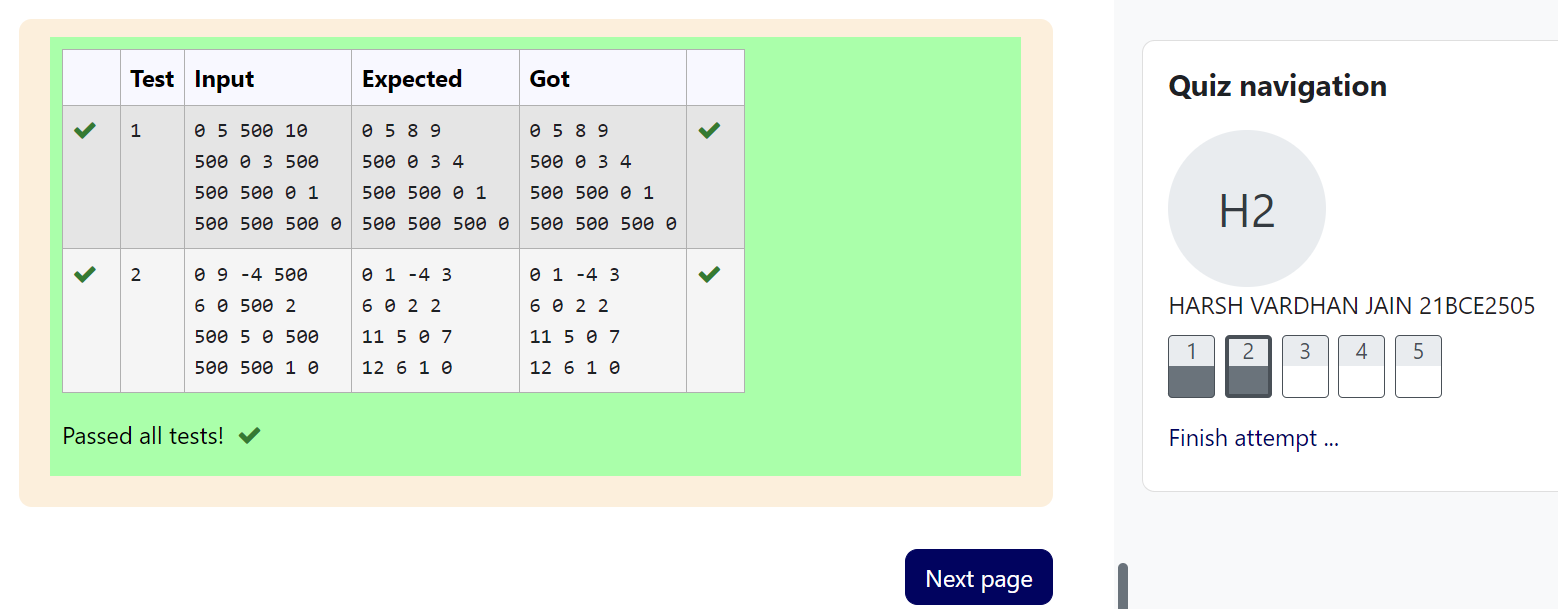
KMPAlgorithm(text, pattern);

return 0;

}

Q2) 





Code

#include<stdio.h>

#define MAX 501 // Maximum number of vertices in the graph

#define INF 500 // The value used to represent infinity

int graph[MAX][MAX]; // The adjacency matrix representing the graph

void floyd\_warshall(int num\_vertices) {

int i, j, k;

// Apply Floyd Warshall Algorithm

for (k = 1; k <= num\_vertices; k++) {

for (i = 1; i <= num\_vertices; i++) {

for (j = 1; j <= num\_vertices; j++) {

if (graph[i][k] + graph[k][j] < graph[i][j]) {

graph[i][j] = graph[i][k] + graph[k][j];

}

}

}

}

}

int main() {

int num\_vertices, i, j;

// scanf("%d", &num\_vertices);

num\_vertices = 4;

// Read in the adjacency matrix

for (i = 1; i <= num\_vertices; i++) {

for (j = 1; j <= num\_vertices; j++) {

scanf("%d", &graph[i][j]);

if (graph[i][j] == 500) {

graph[i][j] = INF; // Replace input 500 with infinity

}

}

}

floyd\_warshall(num\_vertices); // Compute all pair shortest path

// Output the resulting adjacency matrix

for (i = 1; i <= num\_vertices; i++) {

for (j = 1; j <= num\_vertices; j++) {

if (graph[i][j] == INF) {

printf("500 ");

} else {

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

}

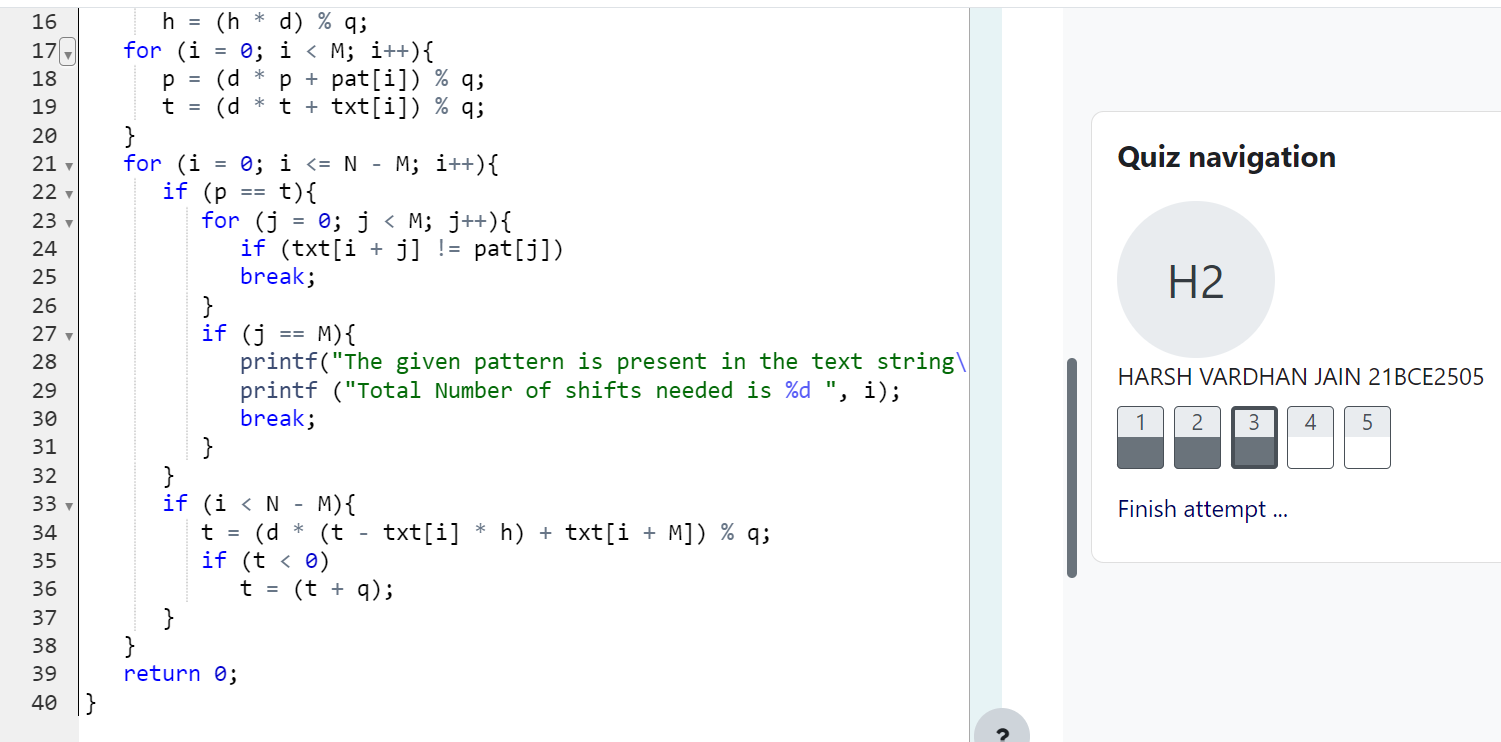
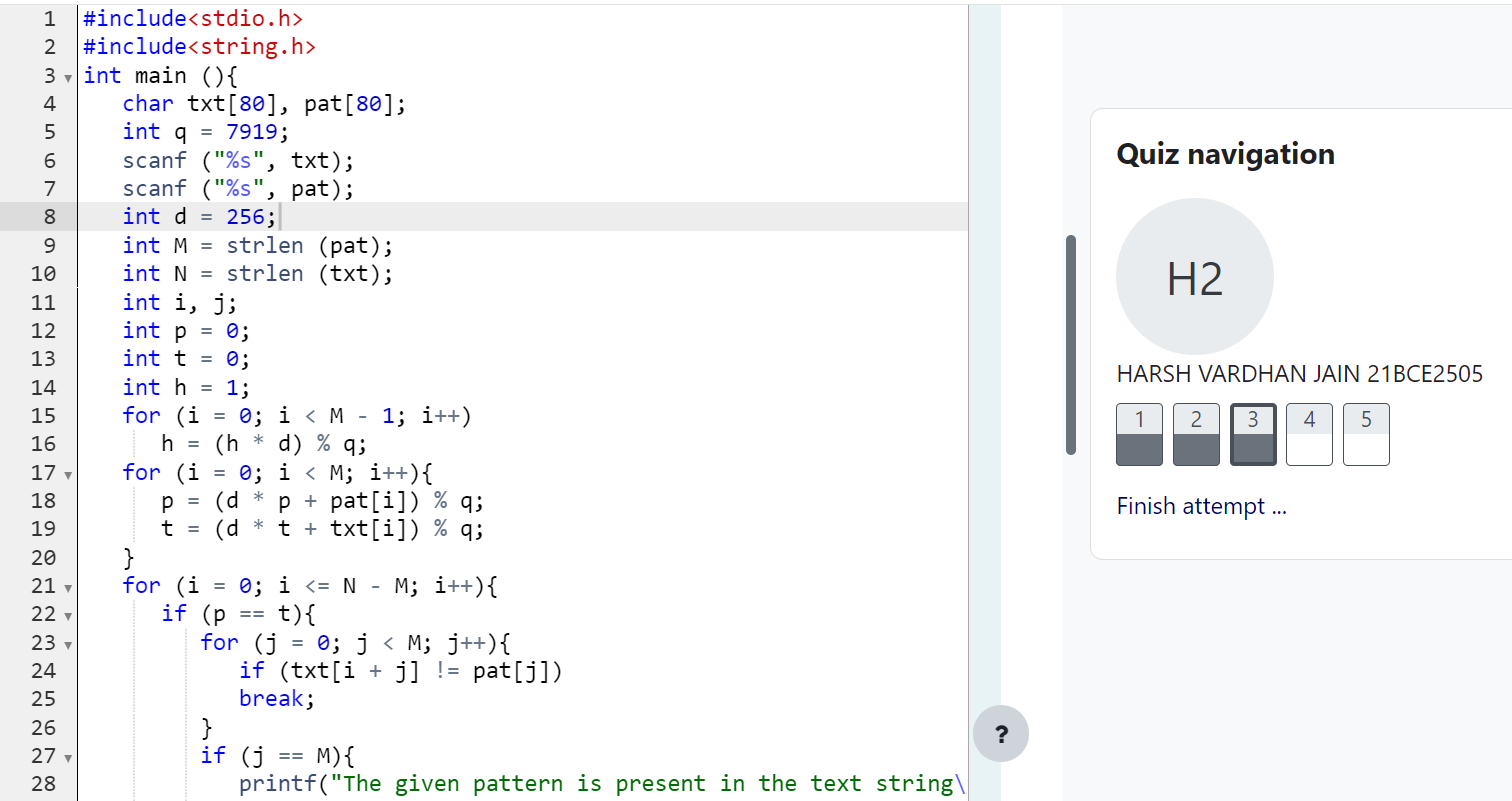
}

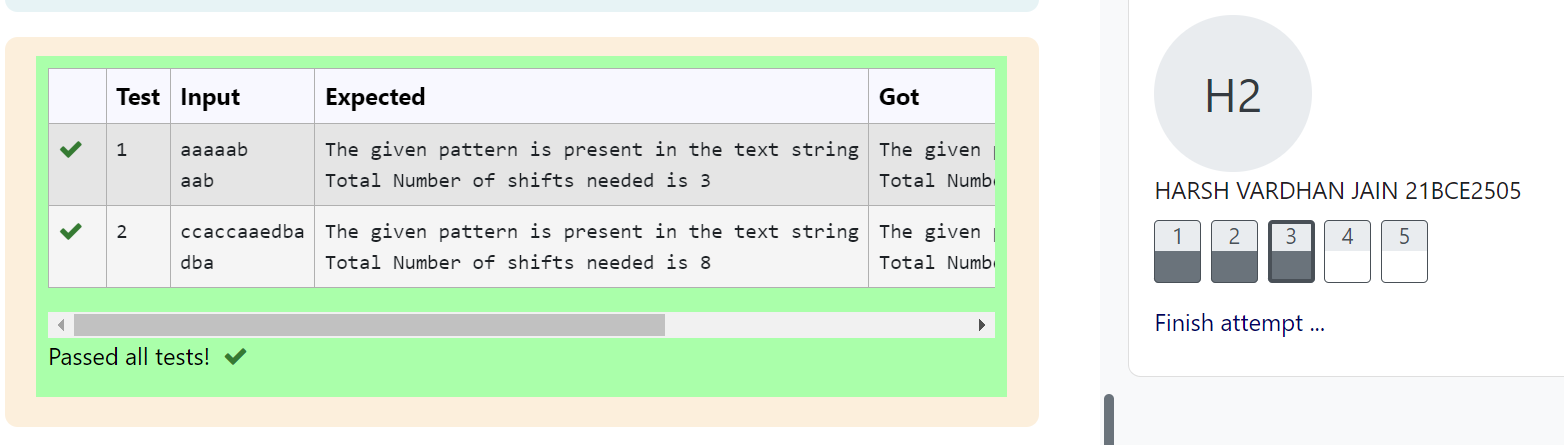
printf("\n");

}

return 0;

}

Q3) 



Code

#include<stdio.h>

#include<string.h>

int main (){

char txt[80], pat[80];

int q = 7919;

scanf ("%s", txt);

scanf ("%s", pat);

int d = 256;

int M = strlen (pat);

int N = strlen (txt);

int i, j;

int p = 0;

int t = 0;

int h = 1;

for (i = 0; i < M - 1; i++)

h = (h \* d) % q;

for (i = 0; i < M; i++){

p = (d \* p + pat[i]) % q;

t = (d \* t + txt[i]) % q;

}

for (i = 0; i <= N - M; i++){

if (p == t){

for (j = 0; j < M; j++){

if (txt[i + j] != pat[j])

break;

}

if (j == M){

printf("The given pattern is present in the text string\n");

printf ("Total Number of shifts needed is %d ", i);

break;

}

}

if (i < N - M){

t = (d \* (t - txt[i] \* h) + txt[i + M]) % q;

if (t < 0)

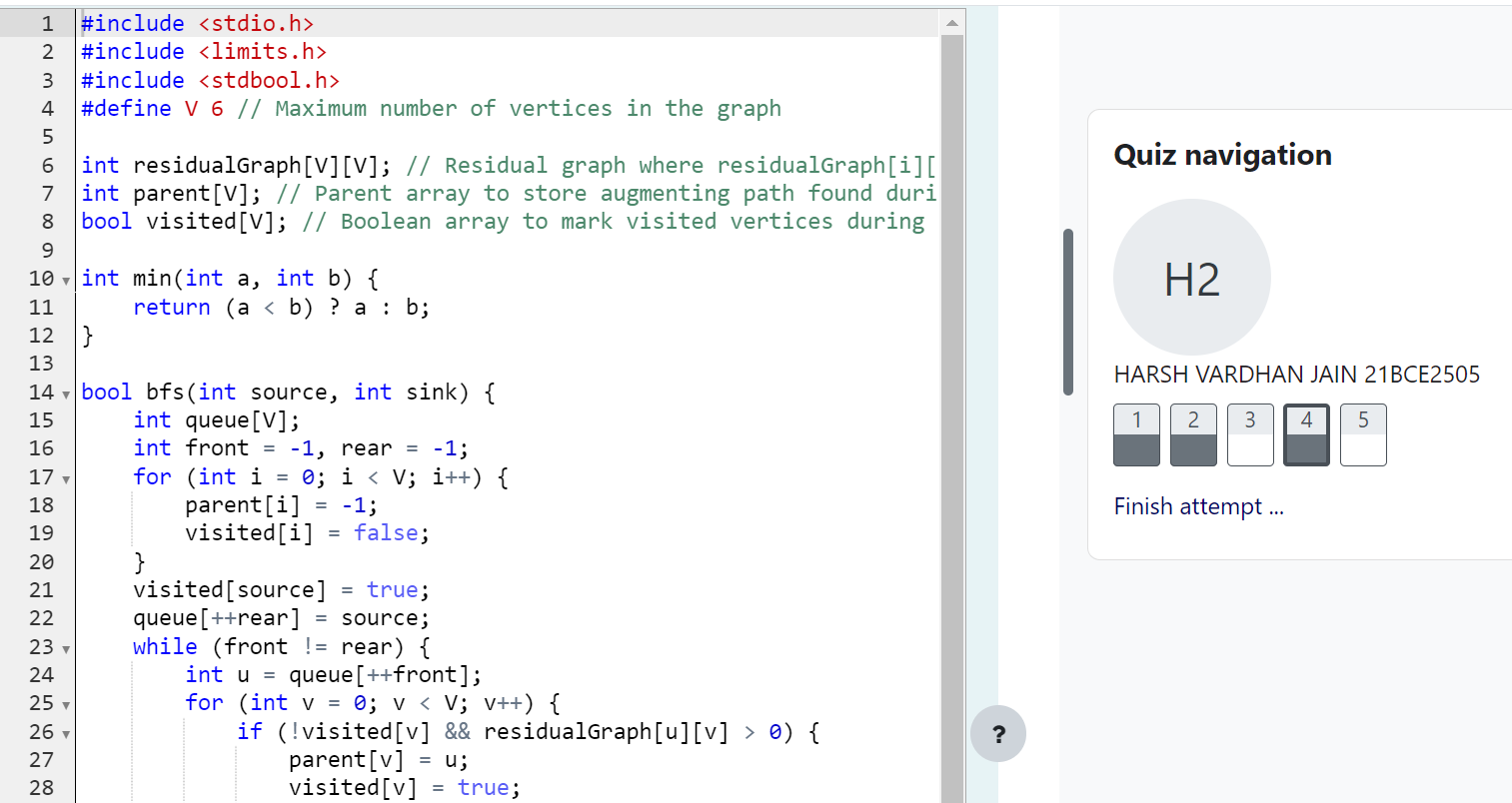
t = (t + q);

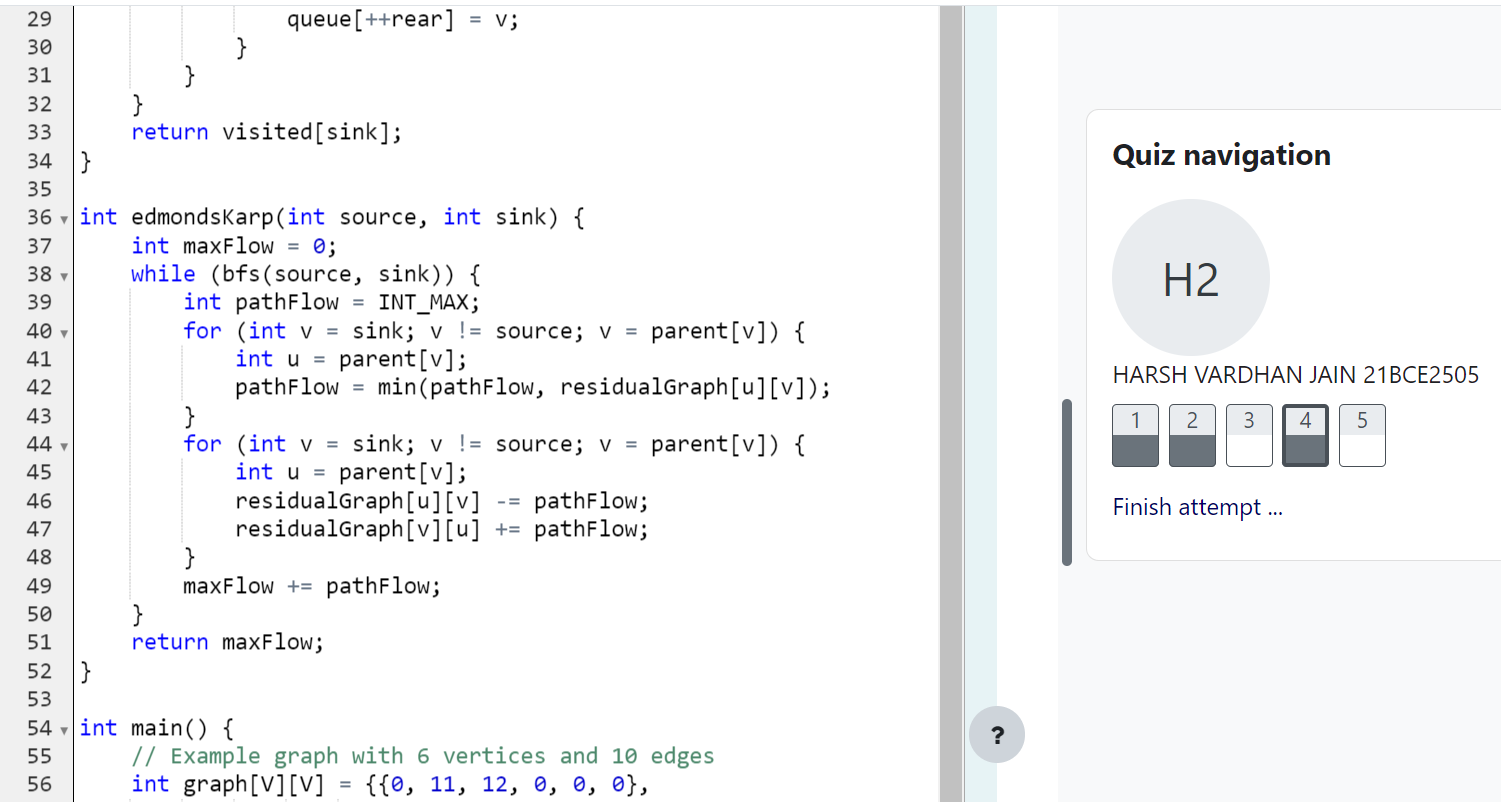
}

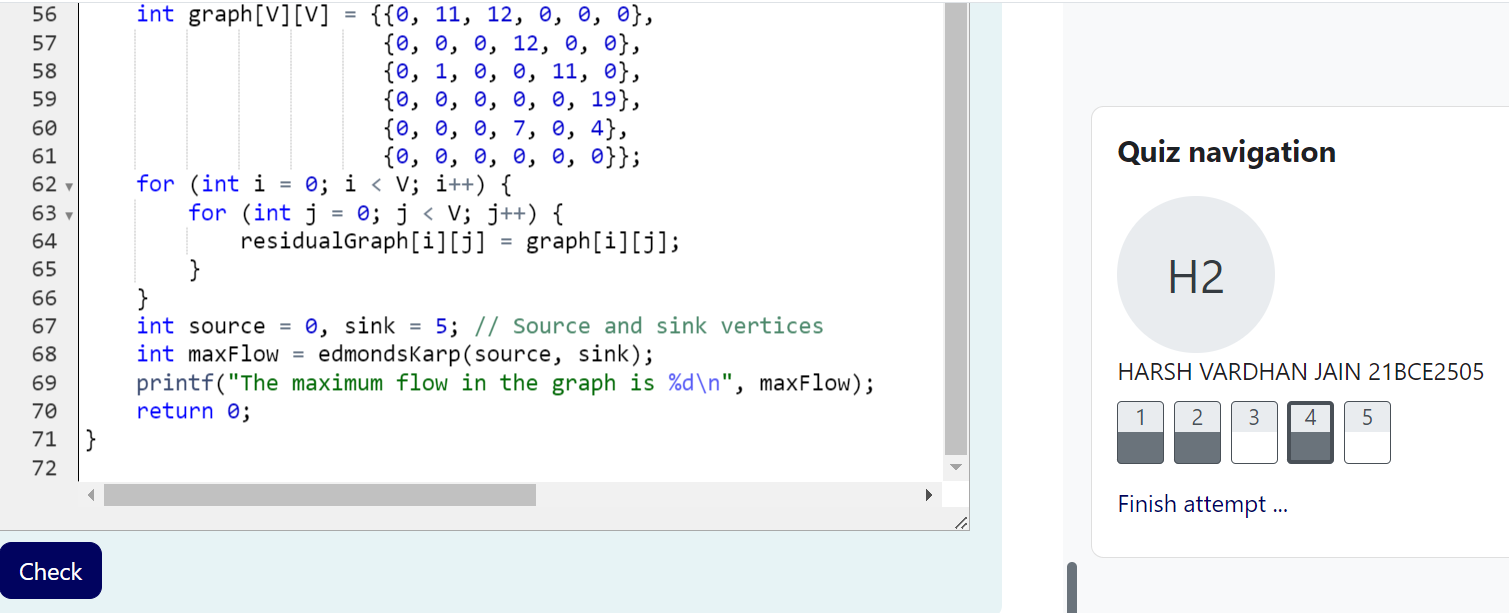
}

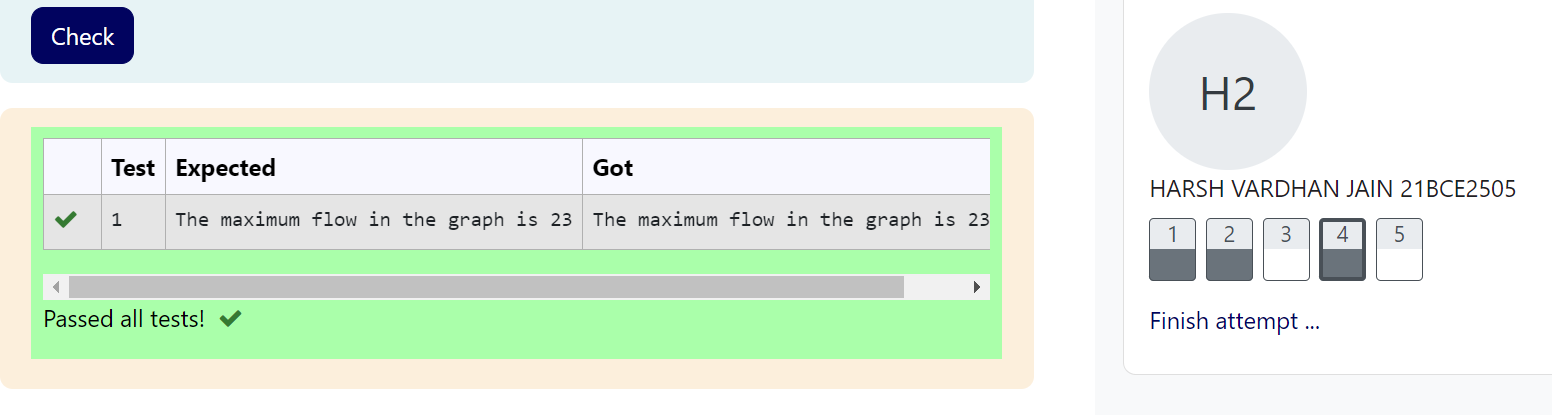
return 0;

}

Q4) 







Code

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 6 // Maximum number of vertices in the graph

int residualGraph[V][V]; // Residual graph where residualGraph[i][j] indicates remaining capacity of edge from vertex i to j

int parent[V]; // Parent array to store augmenting path found during BFS

bool visited[V]; // Boolean array to mark visited vertices during BFS

int min(int a, int b) {

return (a < b) ? a : b;

}

bool bfs(int source, int sink) {

int queue[V];

int front = -1, rear = -1;

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

parent[i] = -1;

visited[i] = false;

}

visited[source] = true;

queue[++rear] = source;

while (front != rear) {

int u = queue[++front];

for (int v = 0; v < V; v++) {

if (!visited[v] && residualGraph[u][v] > 0) {

parent[v] = u;

visited[v] = true;

queue[++rear] = v;

}

}

}

return visited[sink];

}

int edmondsKarp(int source, int sink) {

int maxFlow = 0;

while (bfs(source, sink)) {

int pathFlow = INT\_MAX;

for (int v = sink; v != source; v = parent[v]) {

int u = parent[v];

pathFlow = min(pathFlow, residualGraph[u][v]);

}

for (int v = sink; v != source; v = parent[v]) {

int u = parent[v];

residualGraph[u][v] -= pathFlow;

residualGraph[v][u] += pathFlow;

}

maxFlow += pathFlow;

}

return maxFlow;

}

int main() {

// Example graph with 6 vertices and 10 edges

int graph[V][V] = {{0, 11, 12, 0, 0, 0},

{0, 0, 0, 12, 0, 0},

{0, 1, 0, 0, 11, 0},

{0, 0, 0, 0, 0, 19},

{0, 0, 0, 7, 0, 4},

{0, 0, 0, 0, 0, 0}};

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

for (int j = 0; j < V; j++) {

residualGraph[i][j] = graph[i][j];

}

}

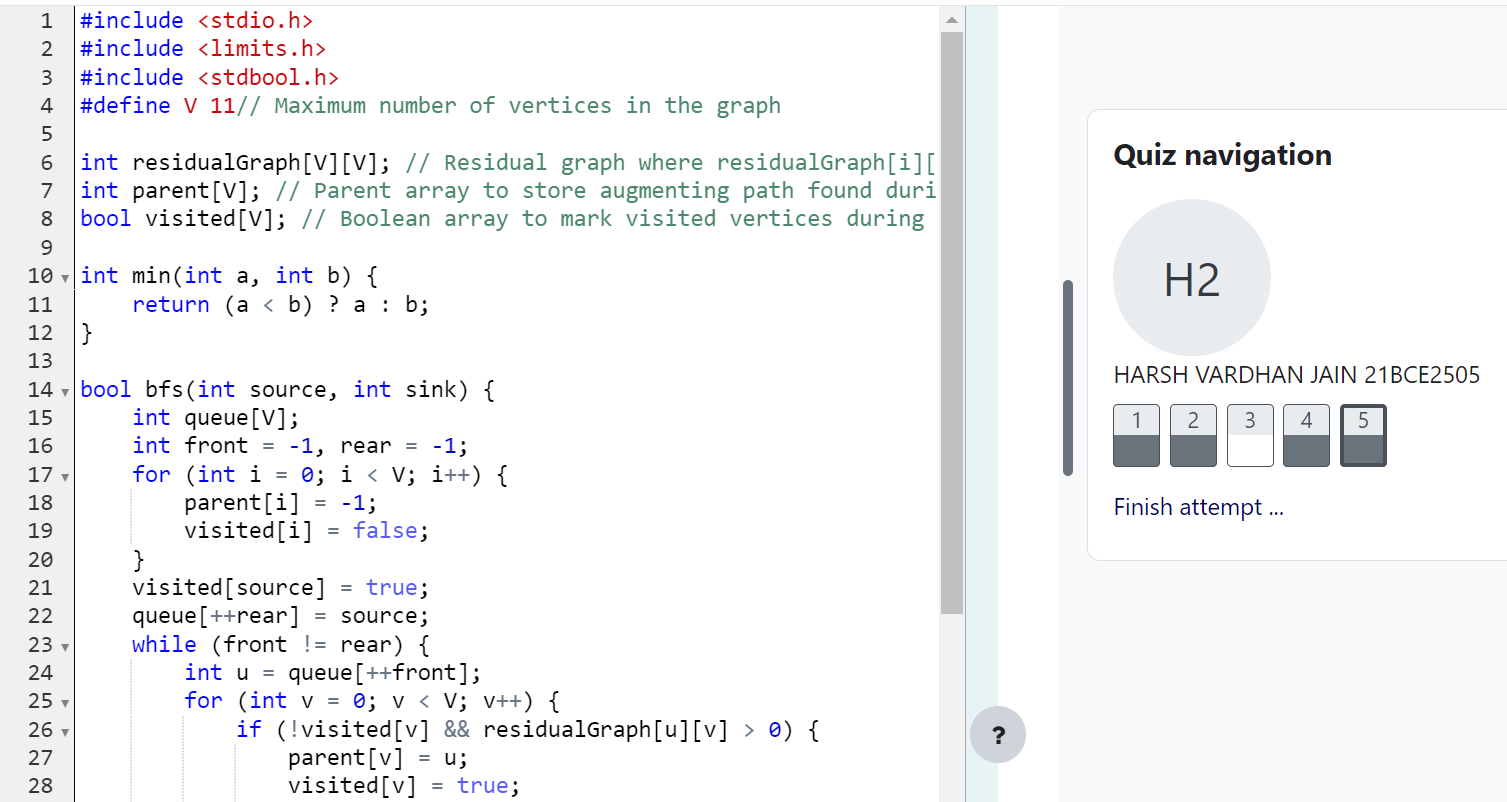
int source = 0, sink = 5; // Source and sink vertices

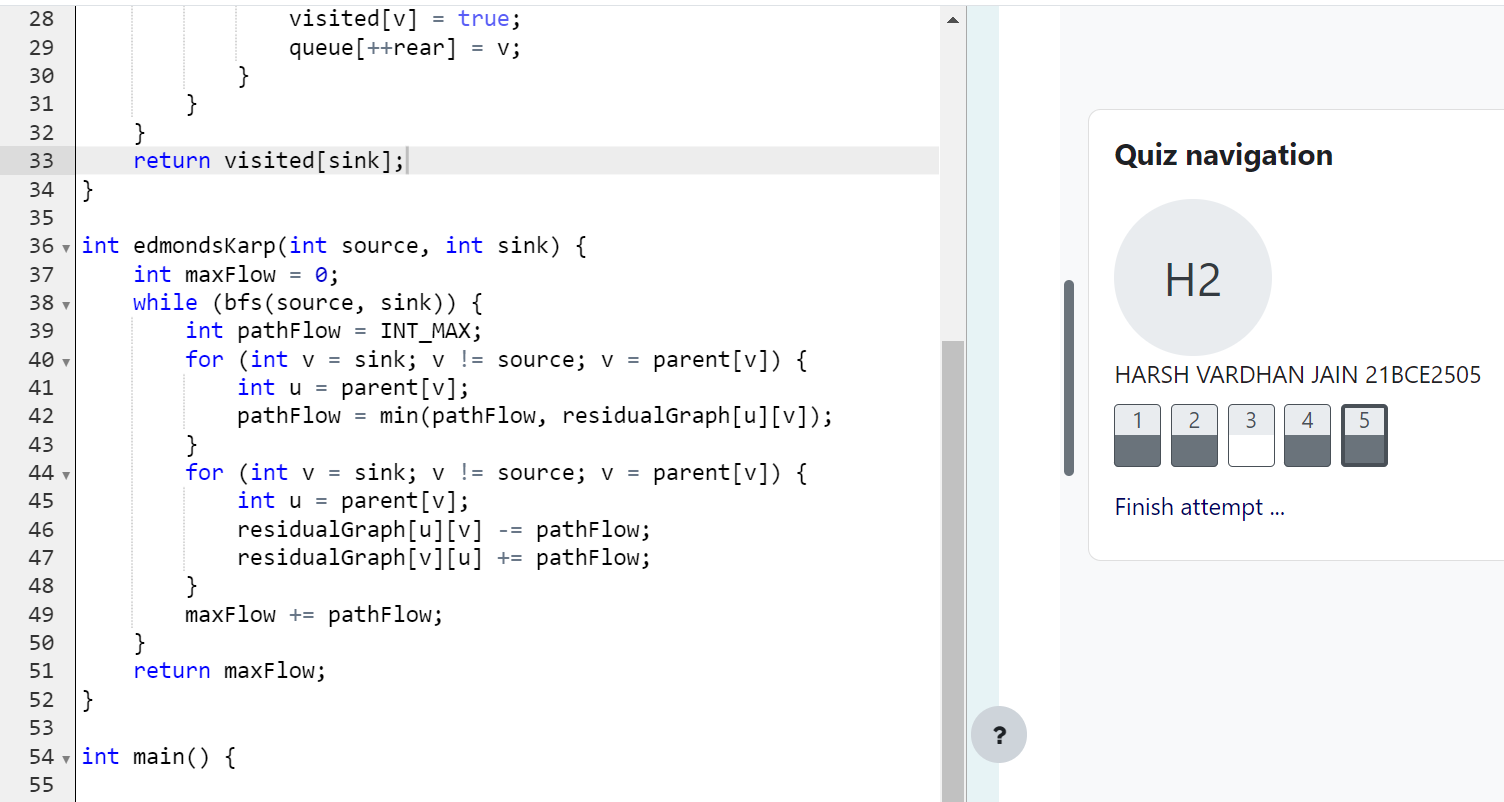
int maxFlow = edmondsKarp(source, sink);

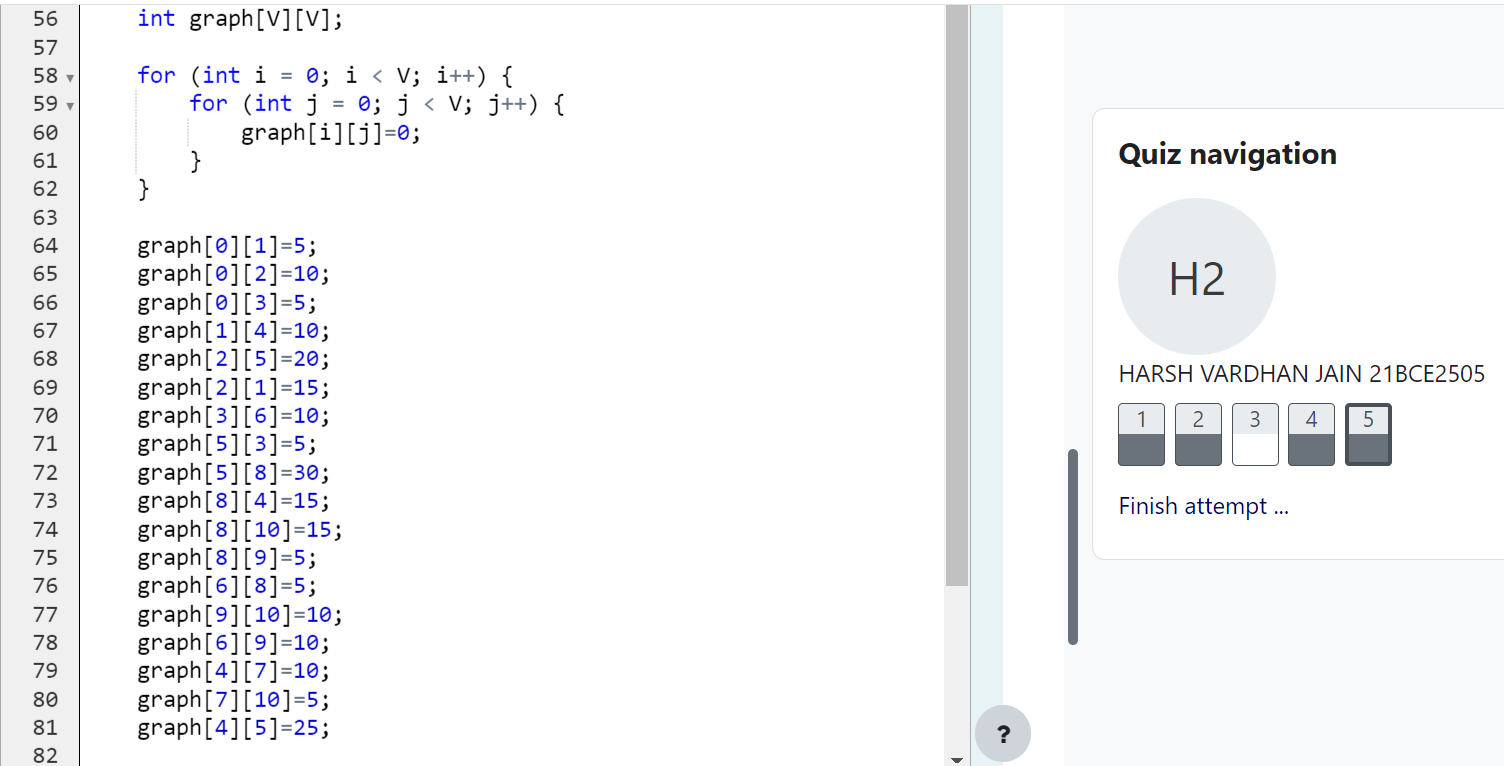
printf("The maximum flow in the graph is %d\n", maxFlow);

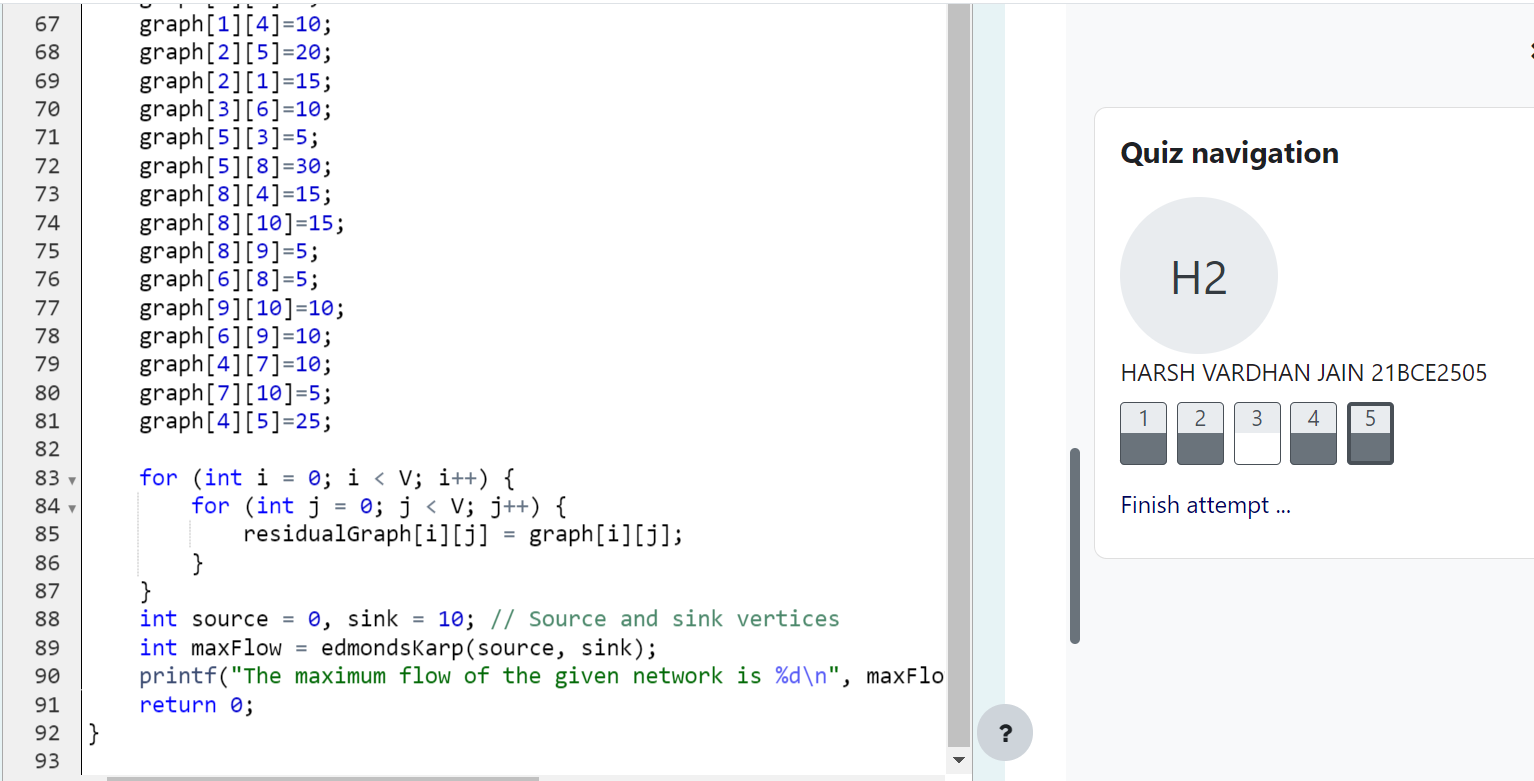
return 0;

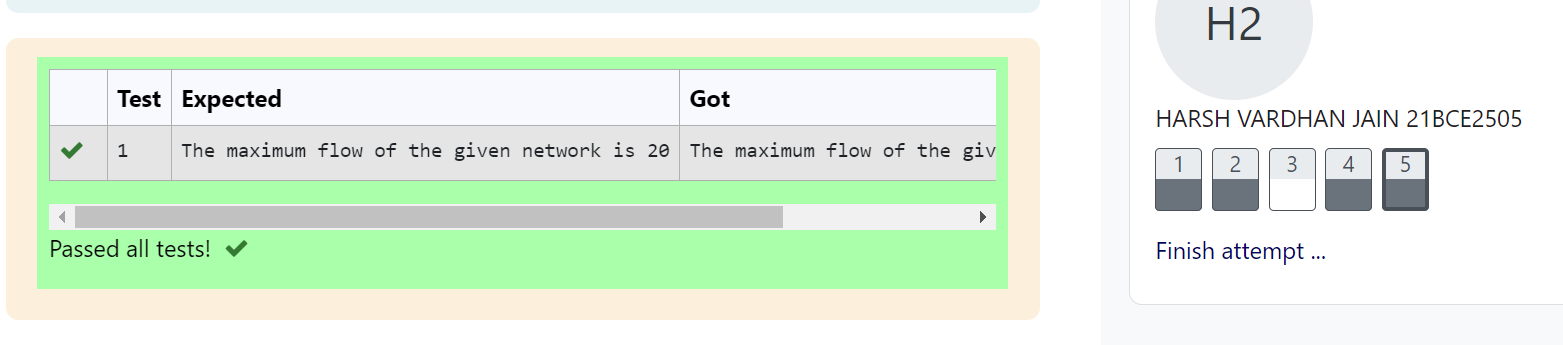
}

Q5) 









Code

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 11// Maximum number of vertices in the graph

int residualGraph[V][V]; // Residual graph where residualGraph[i][j] indicates remaining capacity of edge from vertex i to j

int parent[V]; // Parent array to store augmenting path found during BFS

bool visited[V]; // Boolean array to mark visited vertices during BFS

int min(int a, int b) {

return (a < b) ? a : b;

}

bool bfs(int source, int sink) {

int queue[V];

int front = -1, rear = -1;

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

parent[i] = -1;

visited[i] = false;

}

visited[source] = true;

queue[++rear] = source;

while (front != rear) {

int u = queue[++front];

for (int v = 0; v < V; v++) {

if (!visited[v] && residualGraph[u][v] > 0) {

parent[v] = u;

visited[v] = true;

queue[++rear] = v;

}

}

}

return visited[sink];

}

int edmondsKarp(int source, int sink) {

int maxFlow = 0;

while (bfs(source, sink)) {

int pathFlow = INT\_MAX;

for (int v = sink; v != source; v = parent[v]) {

int u = parent[v];

pathFlow = min(pathFlow, residualGraph[u][v]);

}

for (int v = sink; v != source; v = parent[v]) {

int u = parent[v];

residualGraph[u][v] -= pathFlow;

residualGraph[v][u] += pathFlow;

}

maxFlow += pathFlow;

}

return maxFlow;

}

int main() {

int graph[V][V];

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

for (int j = 0; j < V; j++) {

graph[i][j]=0;

}

}

graph[0][1]=5;

graph[0][2]=10;

graph[0][3]=5;

graph[1][4]=10;

graph[2][5]=20;

graph[2][1]=15;

graph[3][6]=10;

graph[5][3]=5;

graph[5][8]=30;

graph[8][4]=15;

graph[8][10]=15;

graph[8][9]=5;

graph[6][8]=5;

graph[9][10]=10;

graph[6][9]=10;

graph[4][7]=10;

graph[7][10]=5;

graph[4][5]=25;

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

for (int j = 0; j < V; j++) {

residualGraph[i][j] = graph[i][j];

}

}

int source = 0, sink = 10; // Source and sink vertices

int maxFlow = edmondsKarp(source, sink);

printf("The maximum flow of the given network is %d\n", maxFlow);

return 0;

}