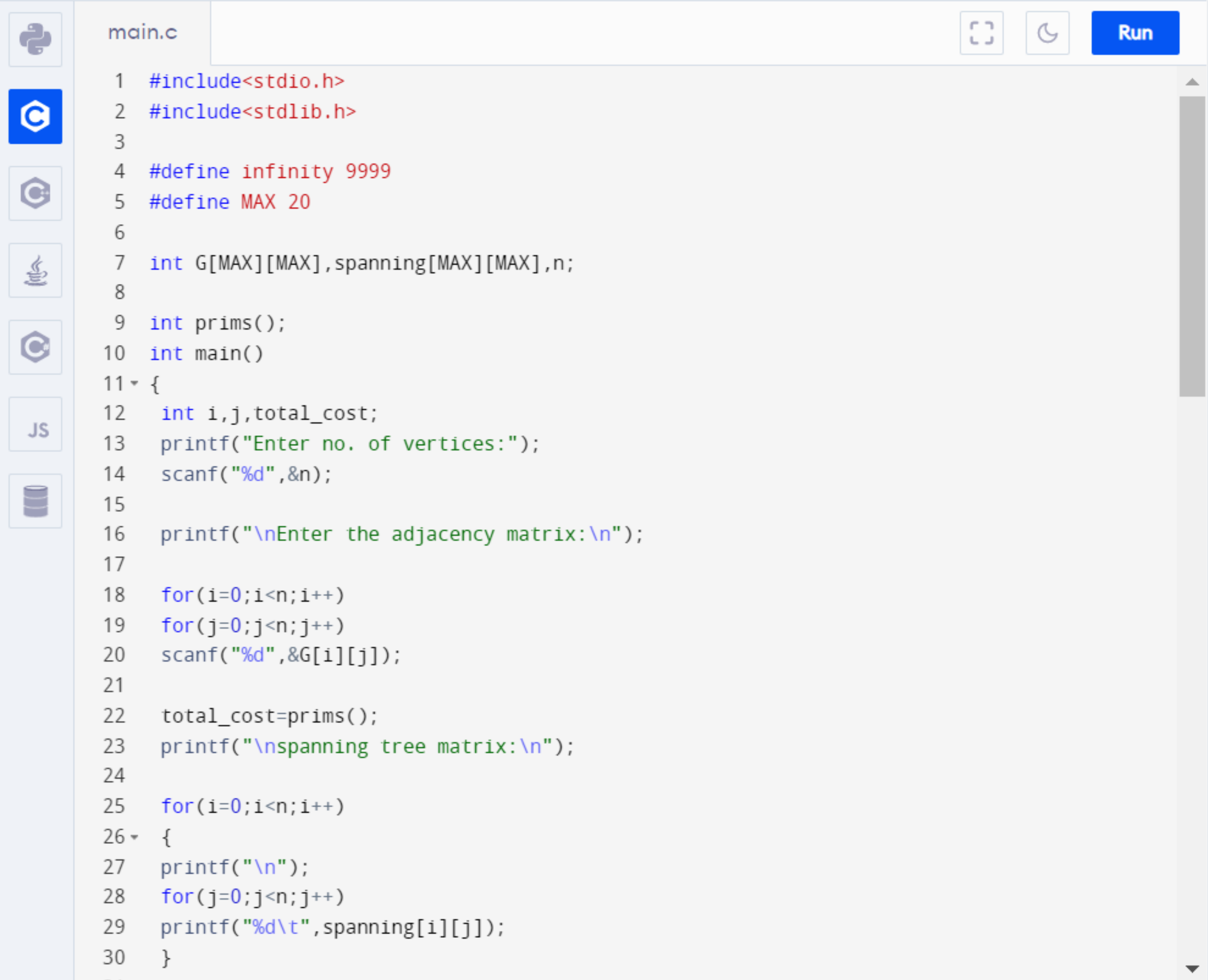
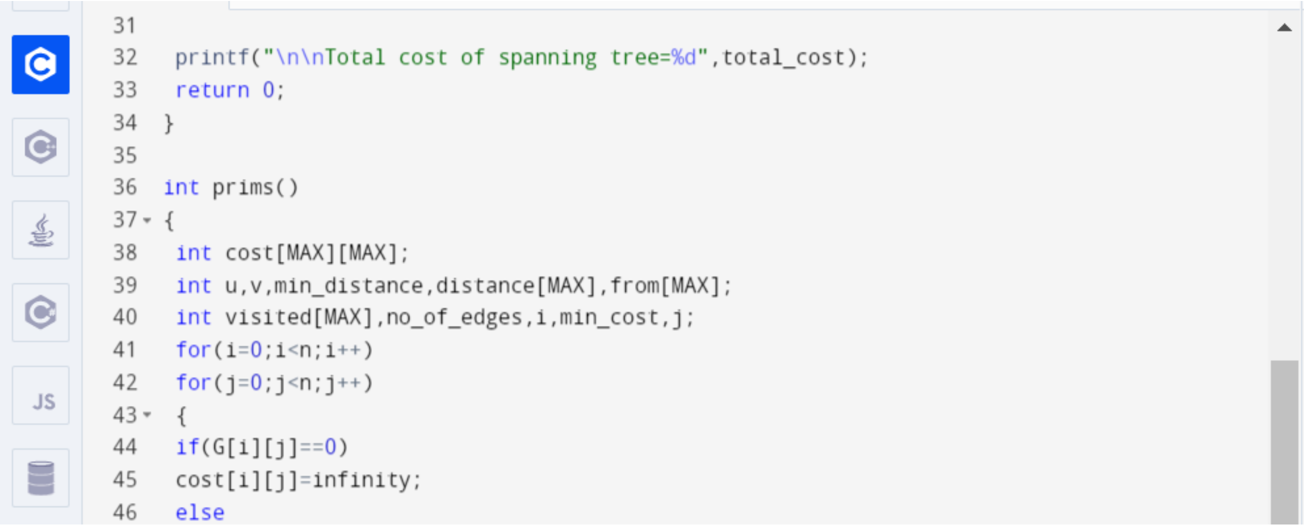
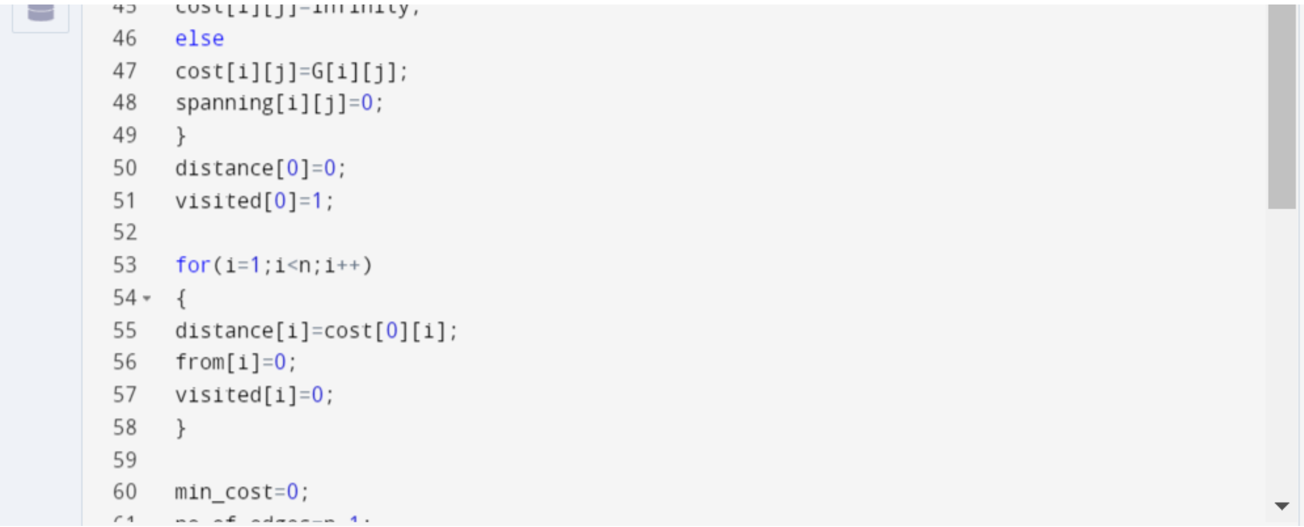


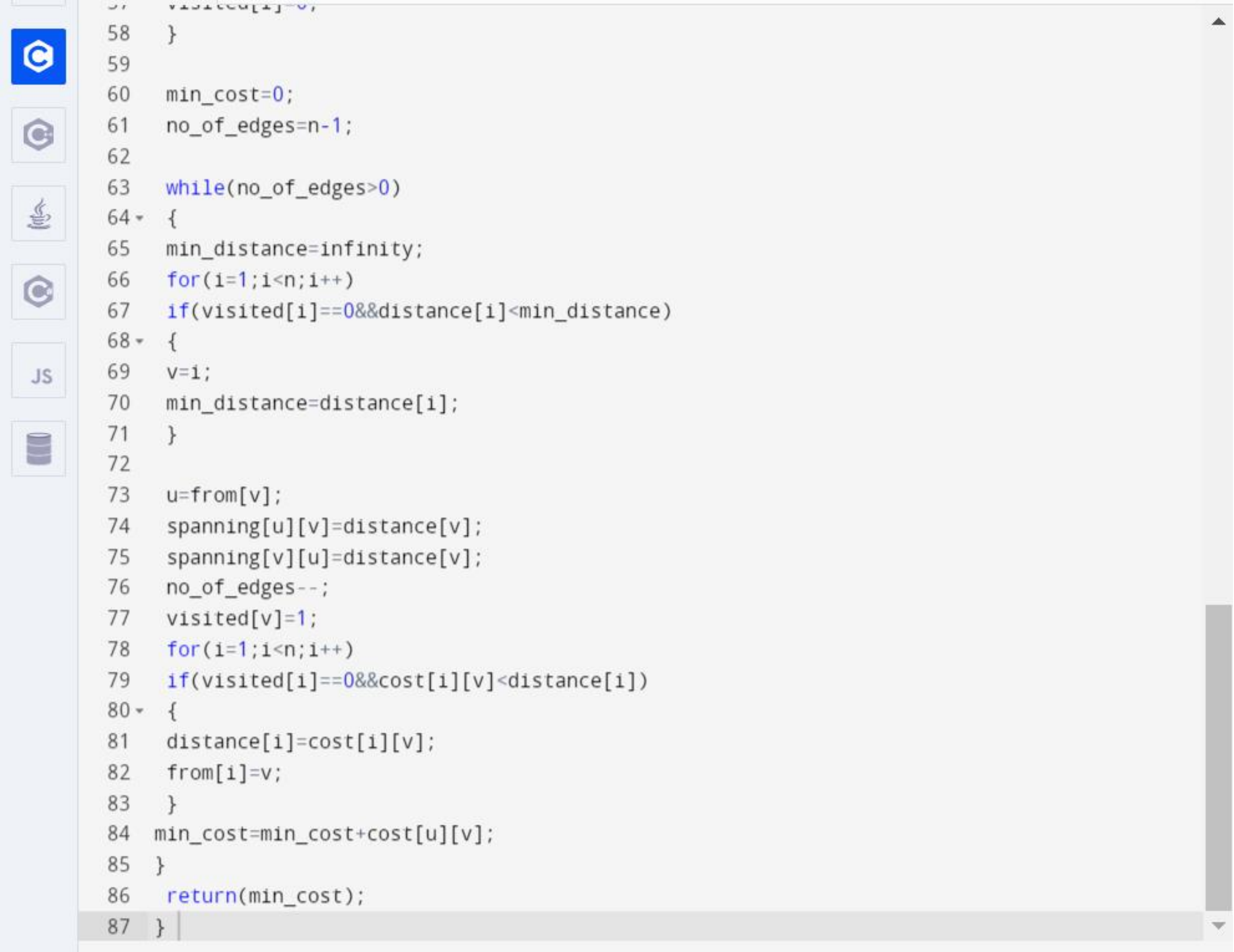
```c
#include<stdio.h>
#include<stdlib.h>

#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();
int main()
{
 int i,j,total_cost;
 printf("Enter no. of vertices:");
 scanf("%d",&n);

 printf("\nEnter the adjacency matrix:\n");

 for(i=0;i<n;i++)
 for(j=0;j<n;j++)
 scanf("%d",&G[i][j]);

 total_cost=prims();
 printf("\nspanning tree matrix:\n");

 for(i=0;i<n;i++)
 {
 printf("\n");
 for(j=0;j<n;j++)
 printf("%d\t",spanning[i][j]);
 }
```



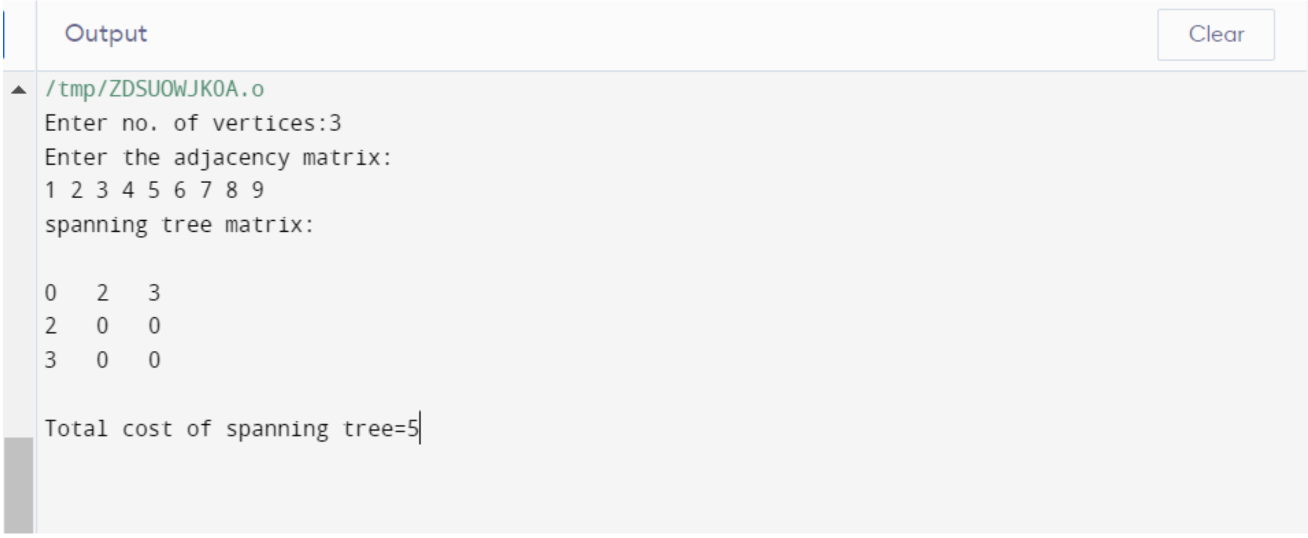
```c

 printf("\n\nTotal cost of spanning tree=%d",total_cost);
 return 0;
}

int prims()
{
 int cost[MAX][MAX];
 int u,v,min_distance,distance[MAX],from[MAX];
 int visited[MAX],no_of_edges,i,min_cost,j;
 for(i=0;i<n;i++)
 for(j=0;j<n;j++)
 {
 if(G[i][j]==0)
 cost[i][j]=infinity;
 else
```



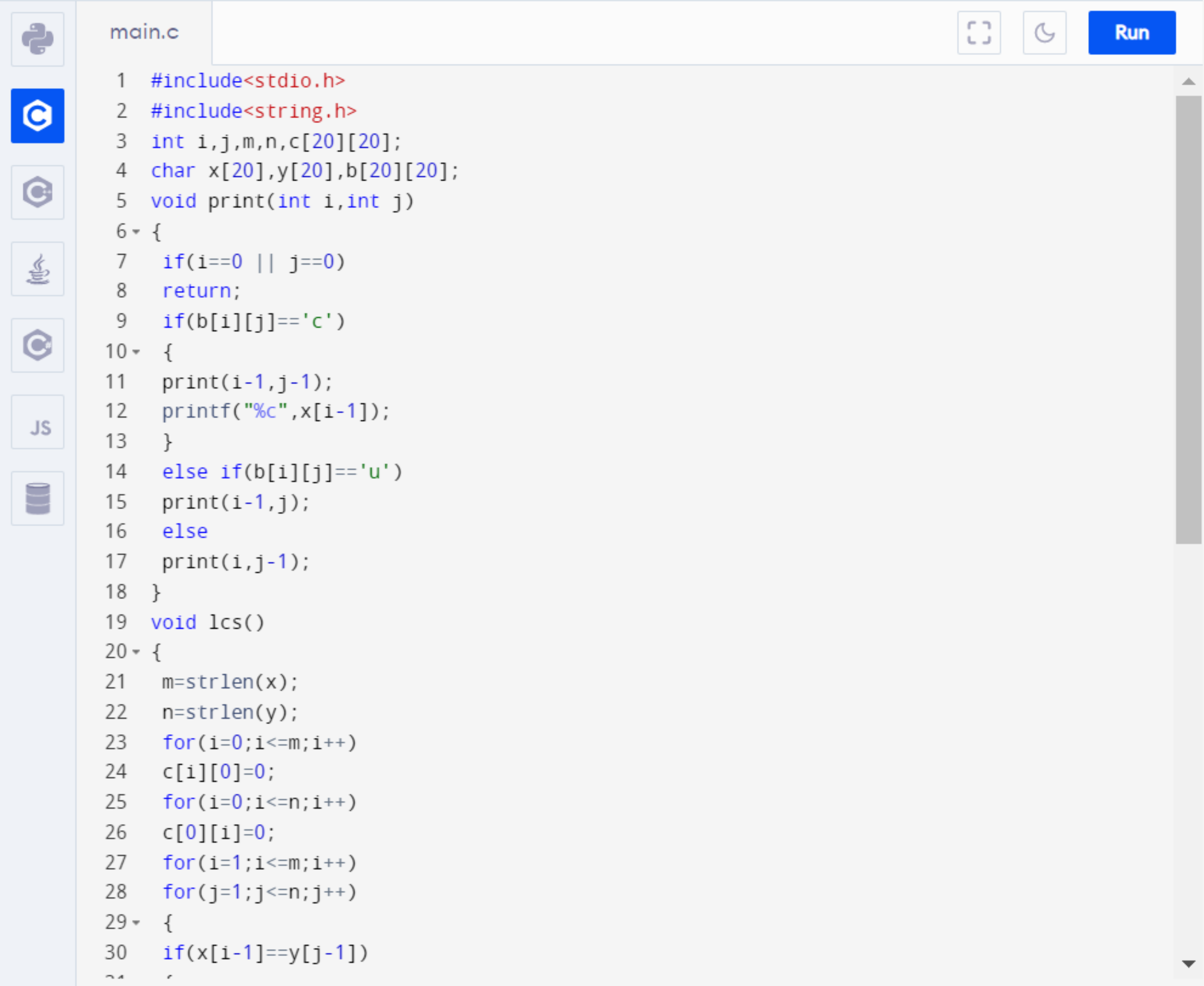
```
 cost[i][j]=infinity;
 else
 cost[i][j]=G[i][j];
 spanning[i][j]=0;
 }
 distance[0]=0;
 visited[0]=1;

 for(i=1;i<n;i++)
 {
 distance[i]=cost[0][i];
 from[i]=0;
 visited[i]=0;
 }

 min_cost=0;
 no_of_edges=n-1;
```



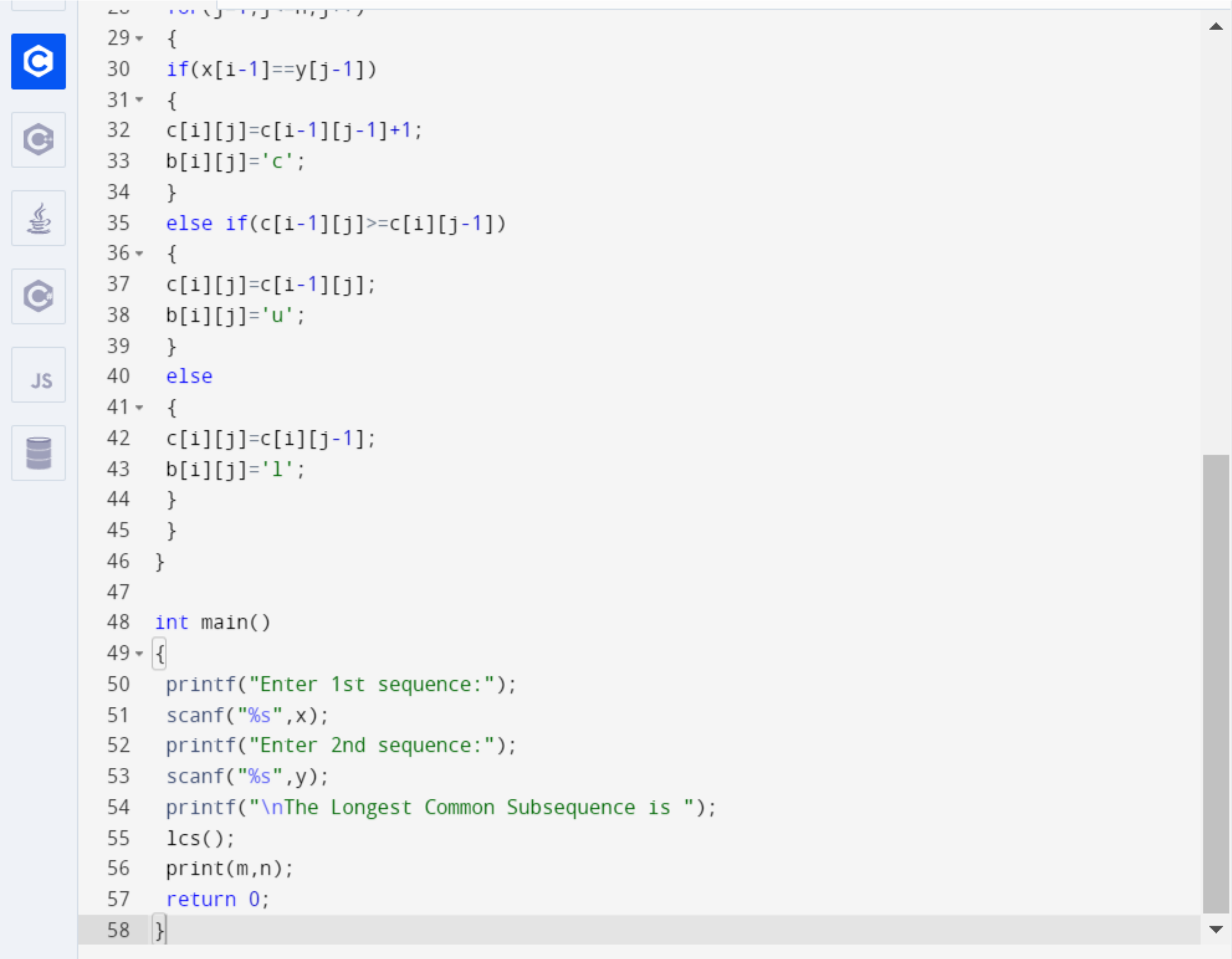
```
 visited[i]=0;
 }

 min_cost=0;
 no_of_edges=n-1;

 while(no_of_edges>0)
 {
 min_distance=infinity;
 for(i=1;i<n;i++)
 if(visited[i]==0&&distance[i]<min_distance)
 {
 v=i;
 min_distance=distance[i];
 }

 u=from[v];
 spanning[u][v]=distance[v];
 spanning[v][u]=distance[v];
 no_of_edges--;
 visited[v]=1;
 for(i=1;i<n;i++)
 if(visited[i]==0&&cost[i][v]<distance[i])
 {
 distance[i]=cost[i][v];
 from[i]=v;
 }
min_cost=min_cost+cost[u][v];
}
 return(min_cost);
}
```



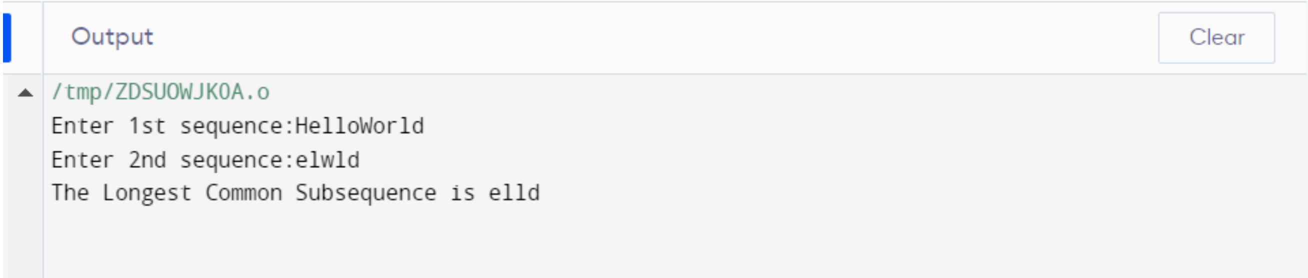
```
/tmp/ZDSUOWJKOA.o
Enter no. of vertices:3
Enter the adjacency matrix:
1 2 3 4 5 6 7 8 9
spanning tree matrix:

0   2   3
2   0   0
3   0   0

Total cost of spanning tree=5
```



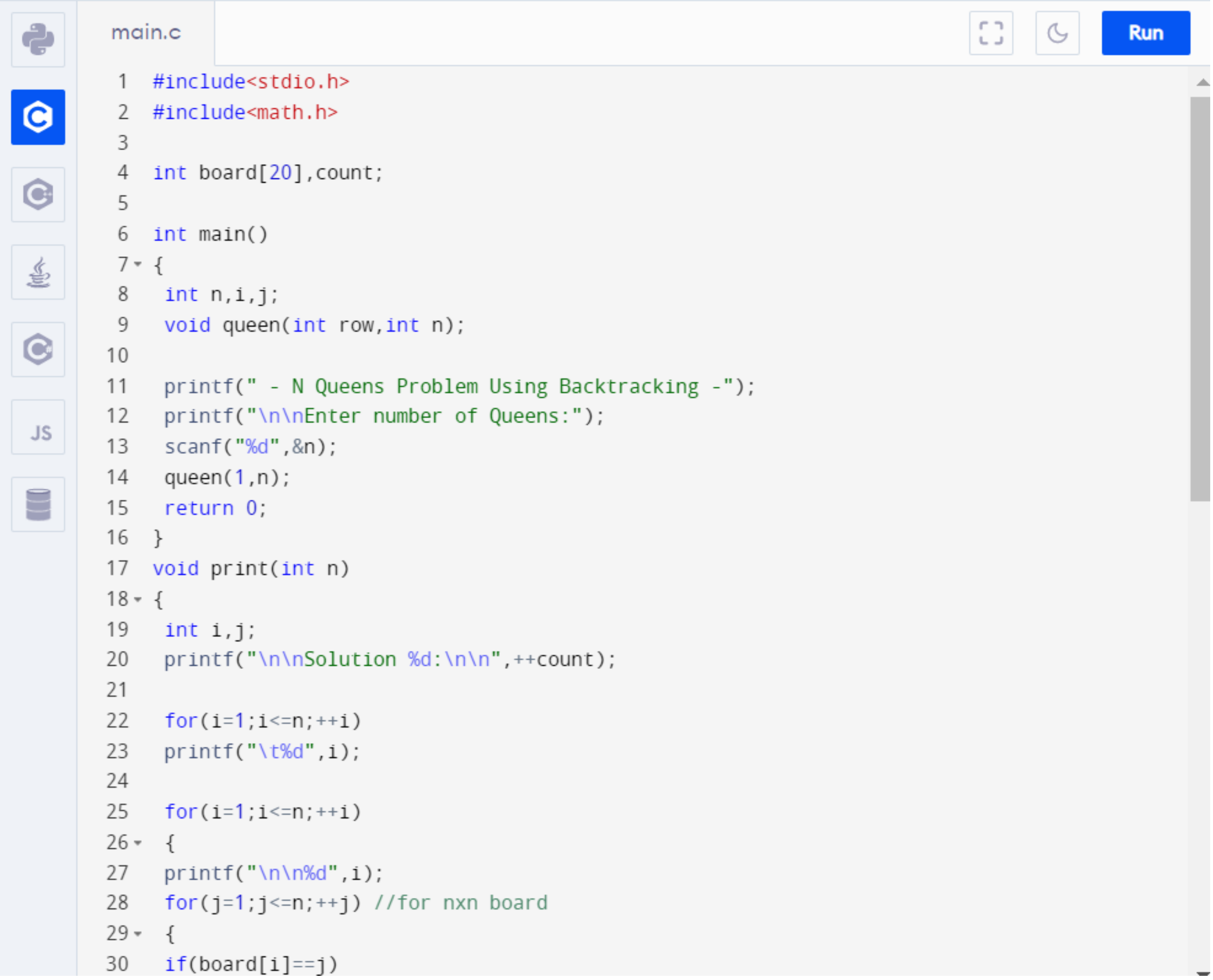
```c
#include<stdio.h>
#include<string.h>
int i,j,m,n,c[20][20];
char x[20],y[20],b[20][20];
void print(int i,int j)
{
 if(i==0 || j==0)
 return;
 if(b[i][j]=='c')
 {
 print(i-1,j-1);
 printf("%c",x[i-1]);
 }
 else if(b[i][j]=='u')
 print(i-1,j);
 else
 print(i,j-1);
}
void lcs()
{
 m=strlen(x);
 n=strlen(y);
 for(i=0;i<=m;i++)
 c[i][0]=0;
 for(i=0;i<=n;i++)
 c[0][i]=0;
 for(i=1;i<=m;i++)
 for(j=1;j<=n;j++)
 {
 if(x[i-1]==y[j-1])
```



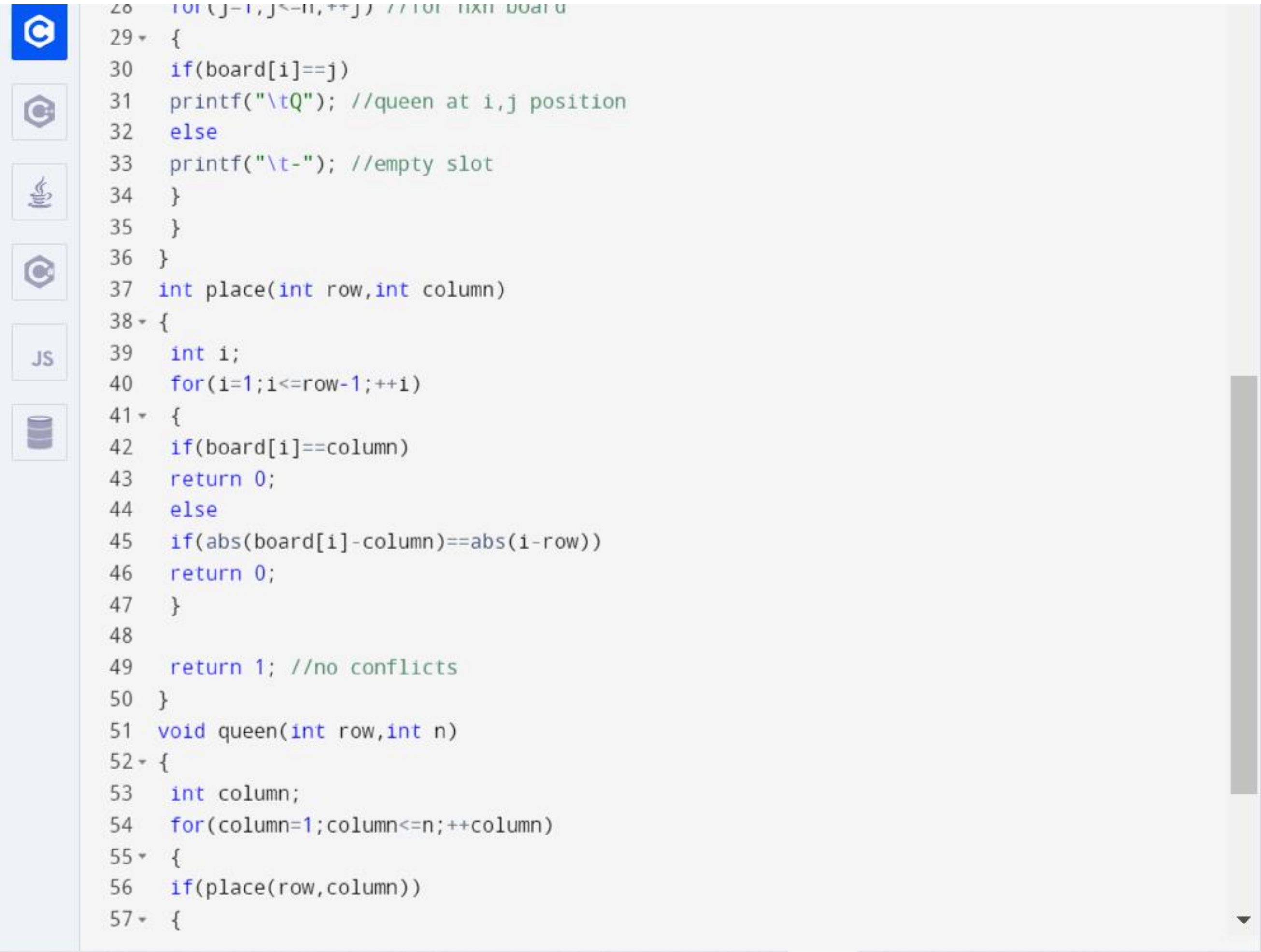
```c
 for(j=1;j<=n;j++)
 {
 if(x[i-1]==y[j-1])
 {
 c[i][j]=c[i-1][j-1]+1;
 b[i][j]='c';
 }
 else if(c[i-1][j]>=c[i][j-1])
 {
 c[i][j]=c[i-1][j];
 b[i][j]='u';
 }
 else
 {
 c[i][j]=c[i][j-1];
 b[i][j]='l';
 }
 }
}

int main()
{
 printf("Enter 1st sequence:");
 scanf("%s",x);
 printf("Enter 2nd sequence:");
 scanf("%s",y);
 printf("\nThe Longest Common Subsequence is ");
 lcs();
 print(m,n);
 return 0;
}
```



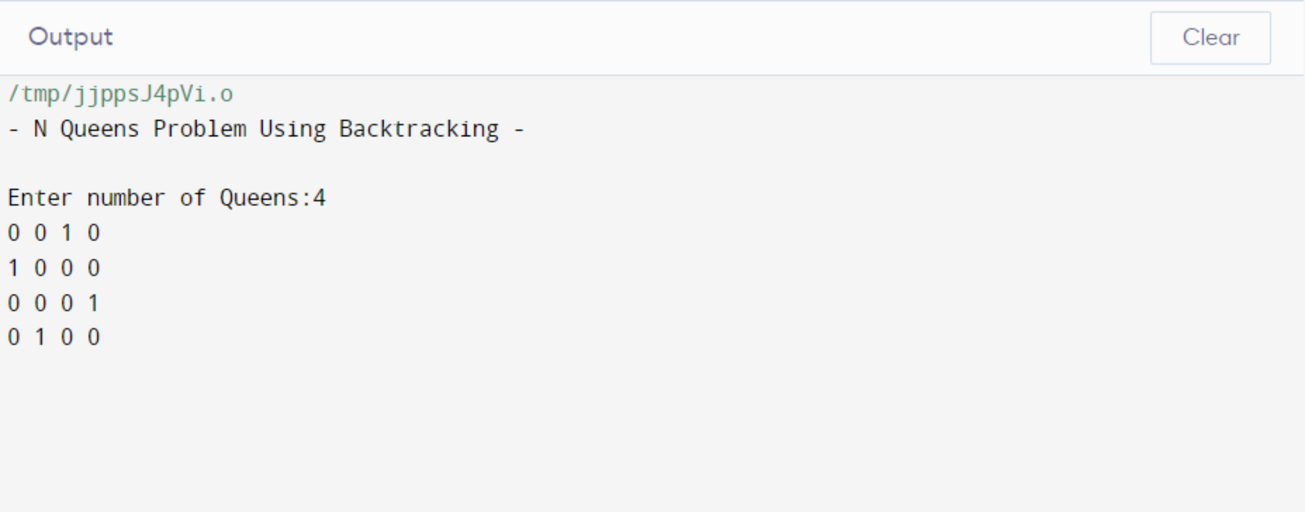
```
/tmp/ZDSUOWJKOA.o
Enter 1st sequence:HelloWorld
Enter 2nd sequence:elwld
The Longest Common Subsequence is elld
```



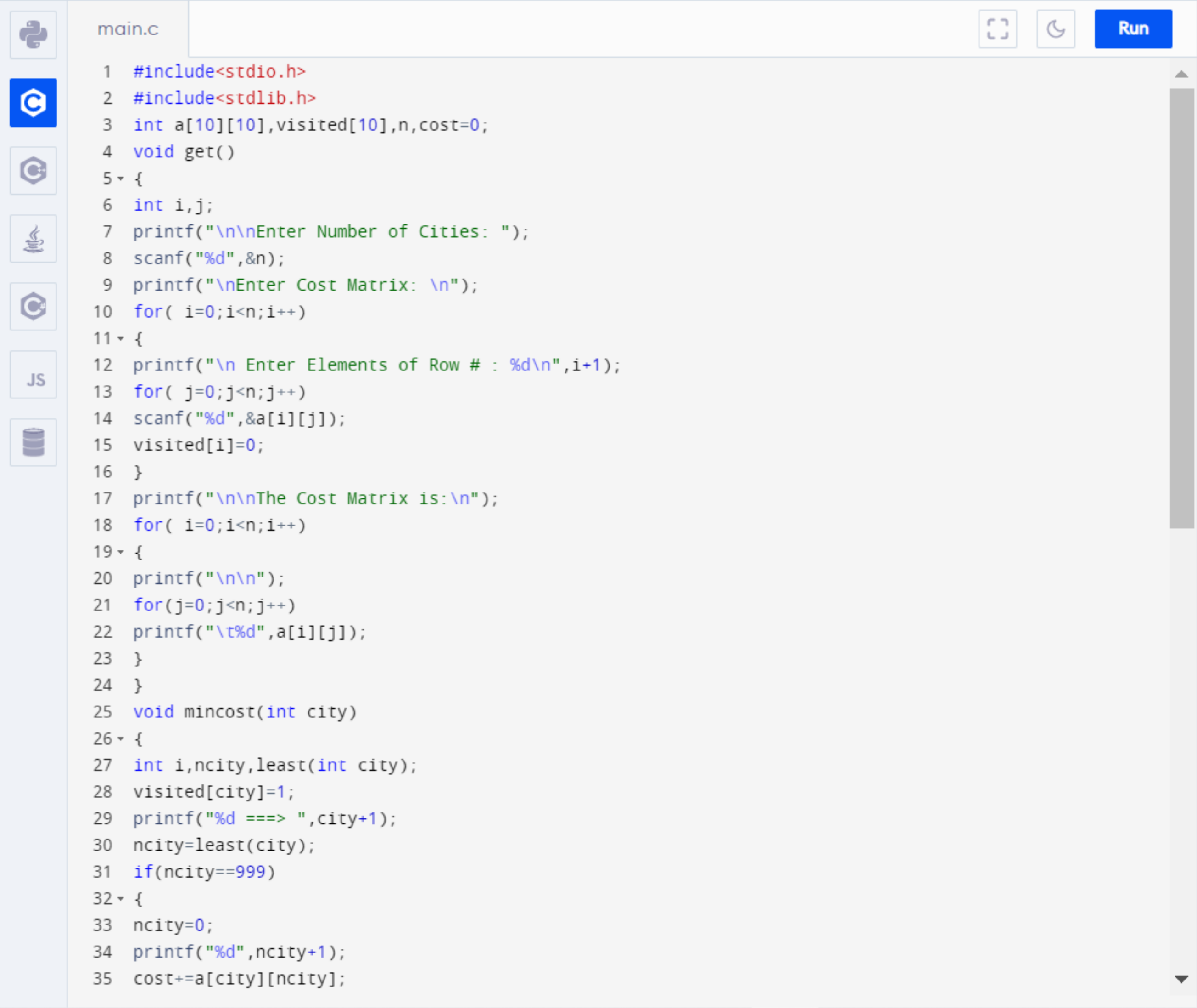
```c
#include<stdio.h>
#include<math.h>

int board[20],count;

int main()
{
 int n,i,j;
 void queen(int row,int n);

 printf(" - N Queens Problem Using Backtracking -");
 printf("\n\nEnter number of Queens:");
 scanf("%d",&n);
 queen(1,n);
 return 0;
}
void print(int n)
{
 int i,j;
 printf("\n\nSolution %d:\n\n",++count);

 for(i=1;i<=n;++i)
 printf("\t%d",i);

 for(i=1;i<=n;++i)
 {
 printf("\n\n%d",i);
 for(j=1;j<=n;++j) //for nxn board
 {
 if(board[i]==j)
```



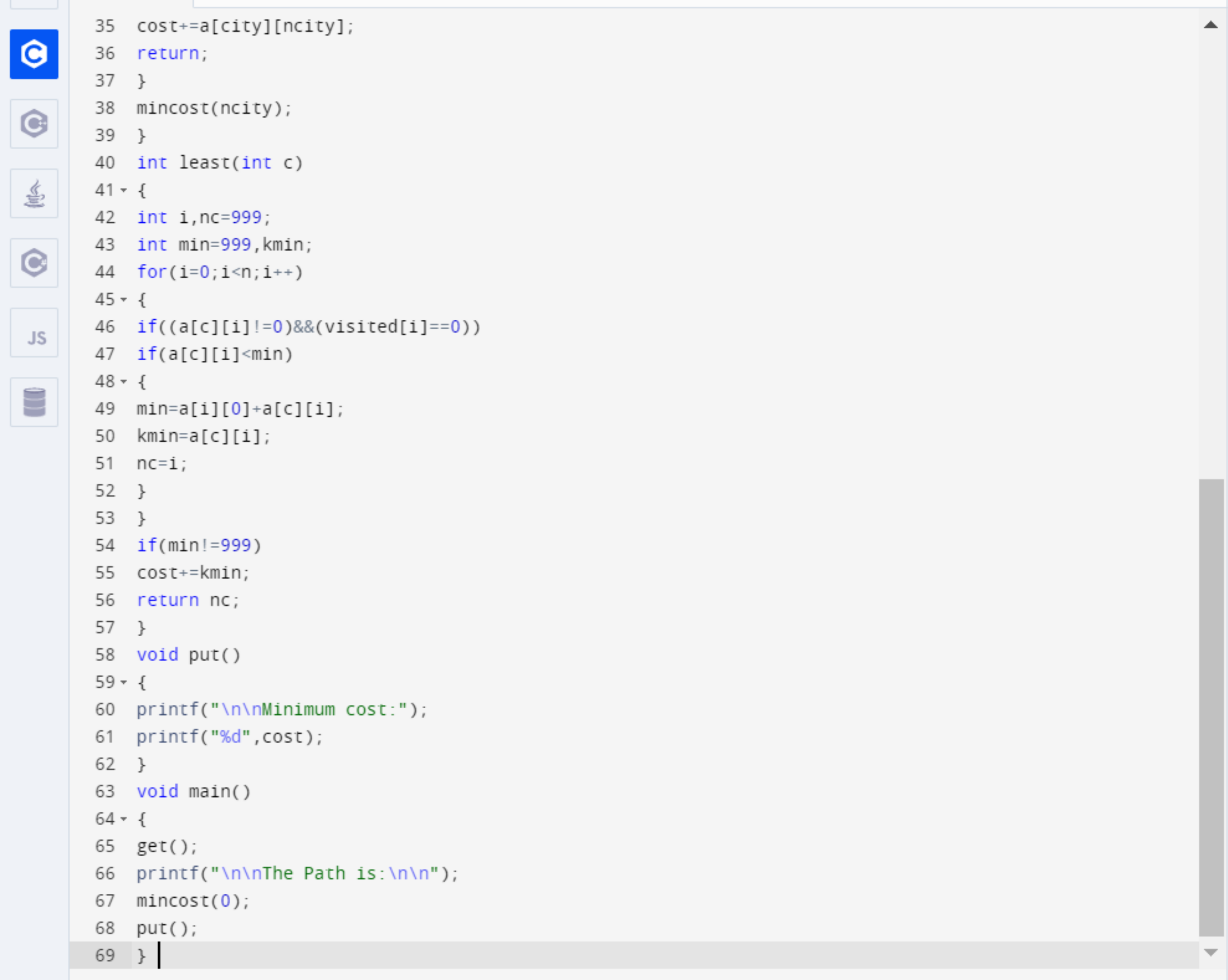
```
 for(j=1;j<=n;++j) //for nxn board
 {
 if(board[i]==j)
 printf("\tQ"); //queen at i,j position
 else
 printf("\t-"); //empty slot
 }
 }
}
int place(int row,int column)
{
 int i;
 for(i=1;i<=row-1;++i)
 {
 if(board[i]==column)
 return 0;
 else
 if(abs(board[i]-column)==abs(i-row))
 return 0;
 }

 return 1; //no conflicts
}
void queen(int row,int n)
{
 int column;
 for(column=1;column<=n;++column)
 {
 if(place(row,column))
 {
```



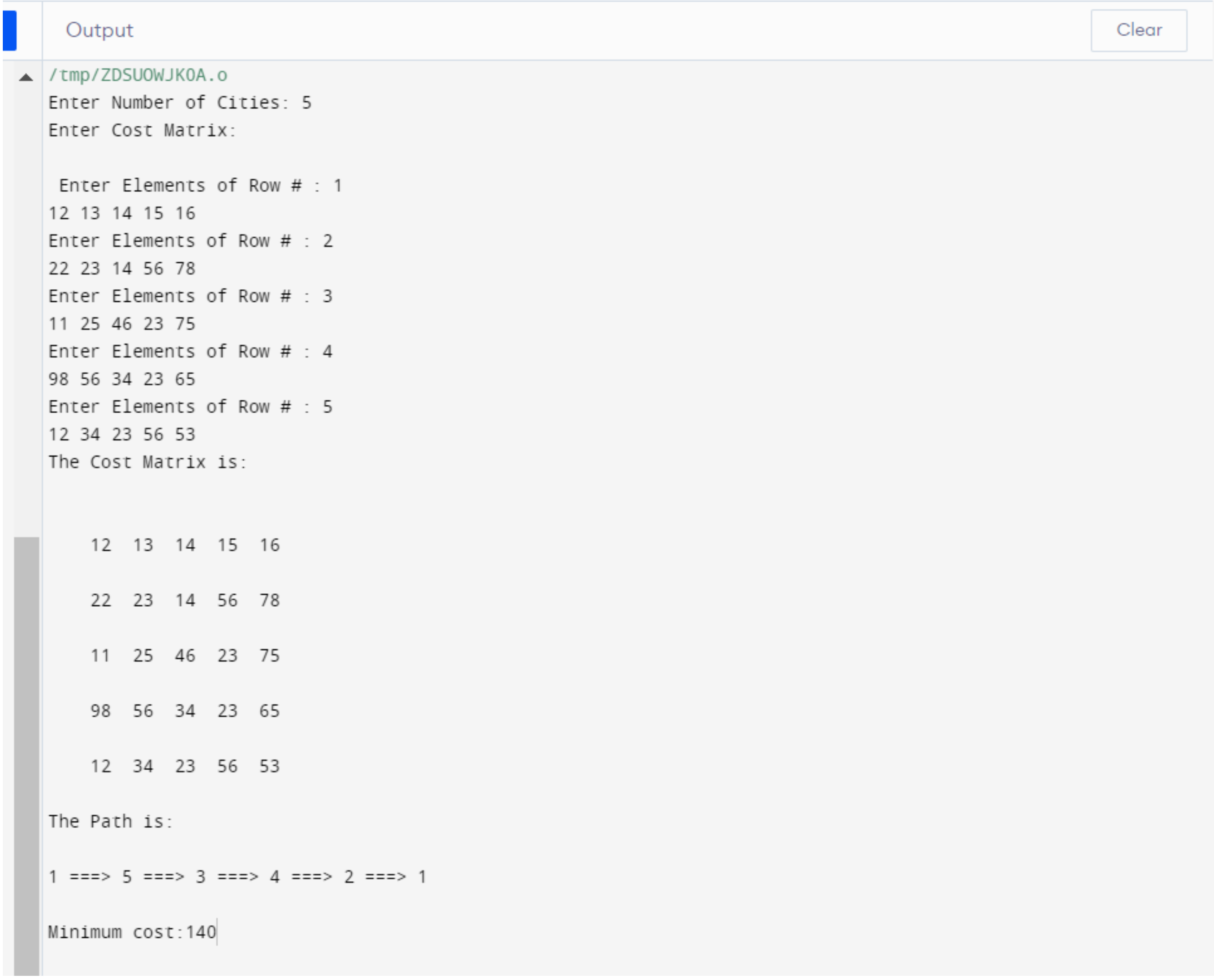
```
/tmp/jjppsJ4pVi.o
- N Queens Problem Using Backtracking -

Enter number of Queens:4
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```



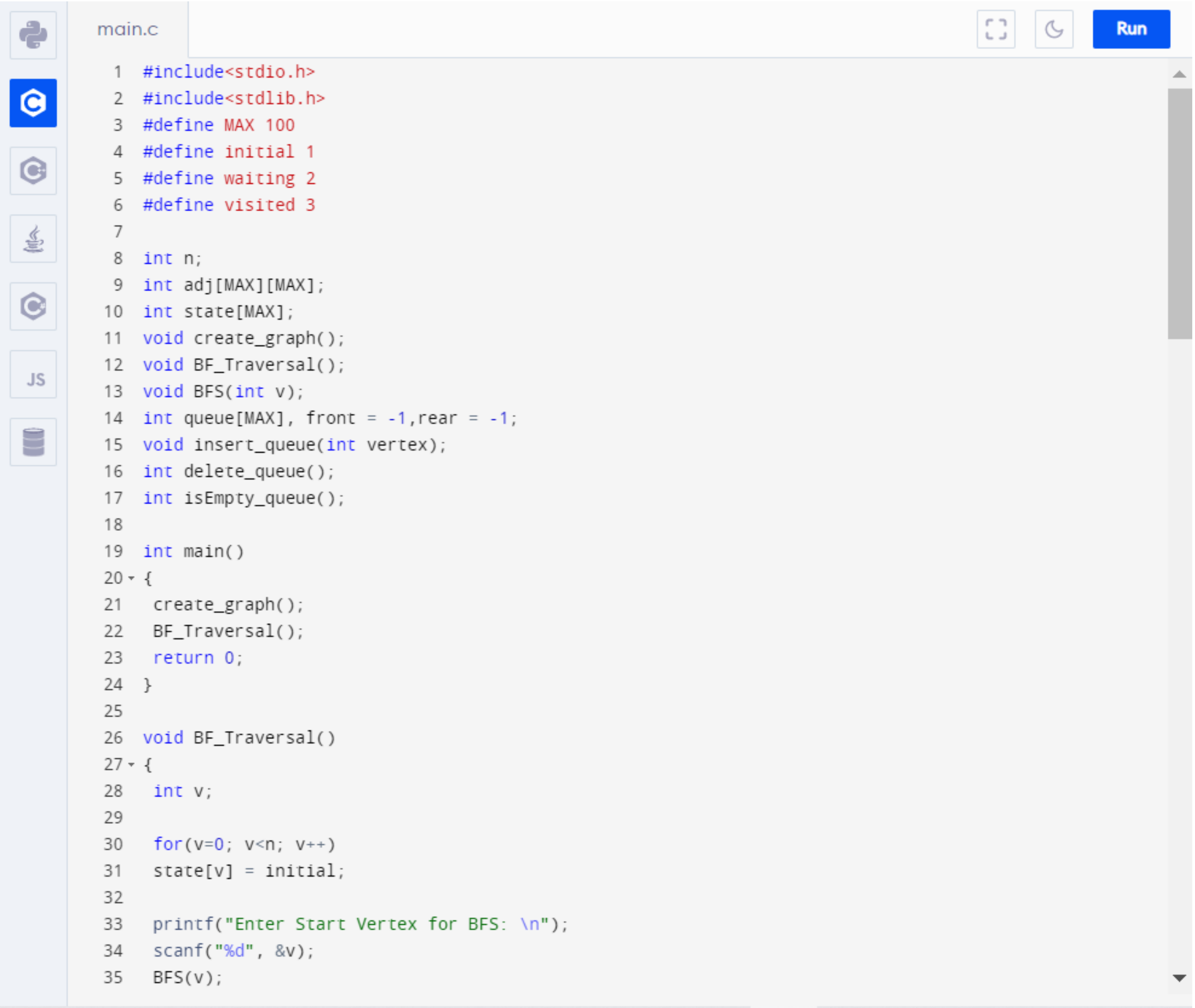
```c
#include<stdio.h>
#include<stdlib.h>
int a[10][10],visited[10],n,cost=0;
void get()
{
int i,j;
printf("\n\nEnter Number of Cities: ");
scanf("%d",&n);
printf("\nEnter Cost Matrix: \n");
for( i=0;i<n;i++)
{
printf("\n Enter Elements of Row # : %d\n",i+1);
for( j=0;j<n;j++)
scanf("%d",&a[i][j]);
visited[i]=0;
}
printf("\n\nThe Cost Matrix is:\n");
for( i=0;i<n;i++)
{
printf("\n\n");
for(j=0;j<n;j++)
printf("\t%d",a[i][j]);
}
}
void mincost(int city)
{
int i,ncity,least(int city);
visited[city]=1;
printf("%d ===> ",city+1);
ncity=least(city);
if(ncity==999)
{
ncity=0;
printf("%d",ncity+1);
cost+=a[city][ncity];
```



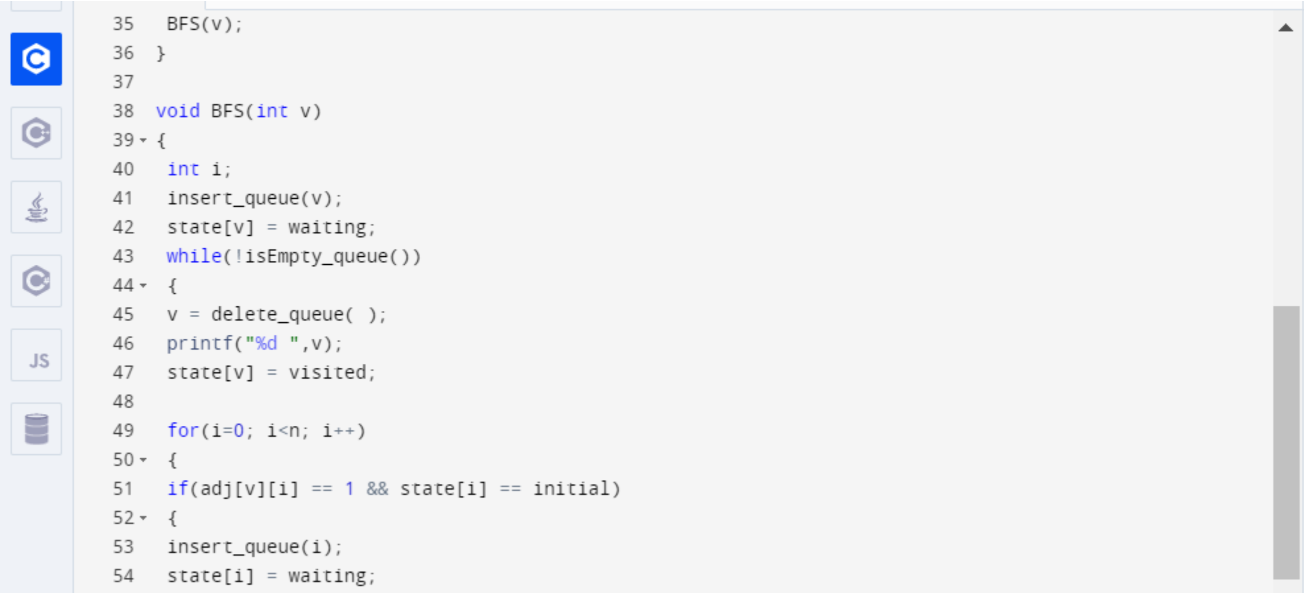
```
cost+=a[city][ncity];
return;
}
mincost(ncity);
}
int least(int c)
{
int i,nc=999;
int min=999,kmin;
for(i=0;i<n;i++)
{
if((a[c][i]!=0)&&(visited[i]==0))
if(a[c][i]<min)
{
min=a[i][0]+a[c][i];
kmin=a[c][i];
nc=i;
}
}
if(min!=999)
cost+=kmin;
return nc;
}
void put()
{
printf("\n\nMinimum cost:");
printf("%d",cost);
}
void main()
{
get();
printf("\n\nThe Path is:\n\n");
mincost(0);
put();
}
```



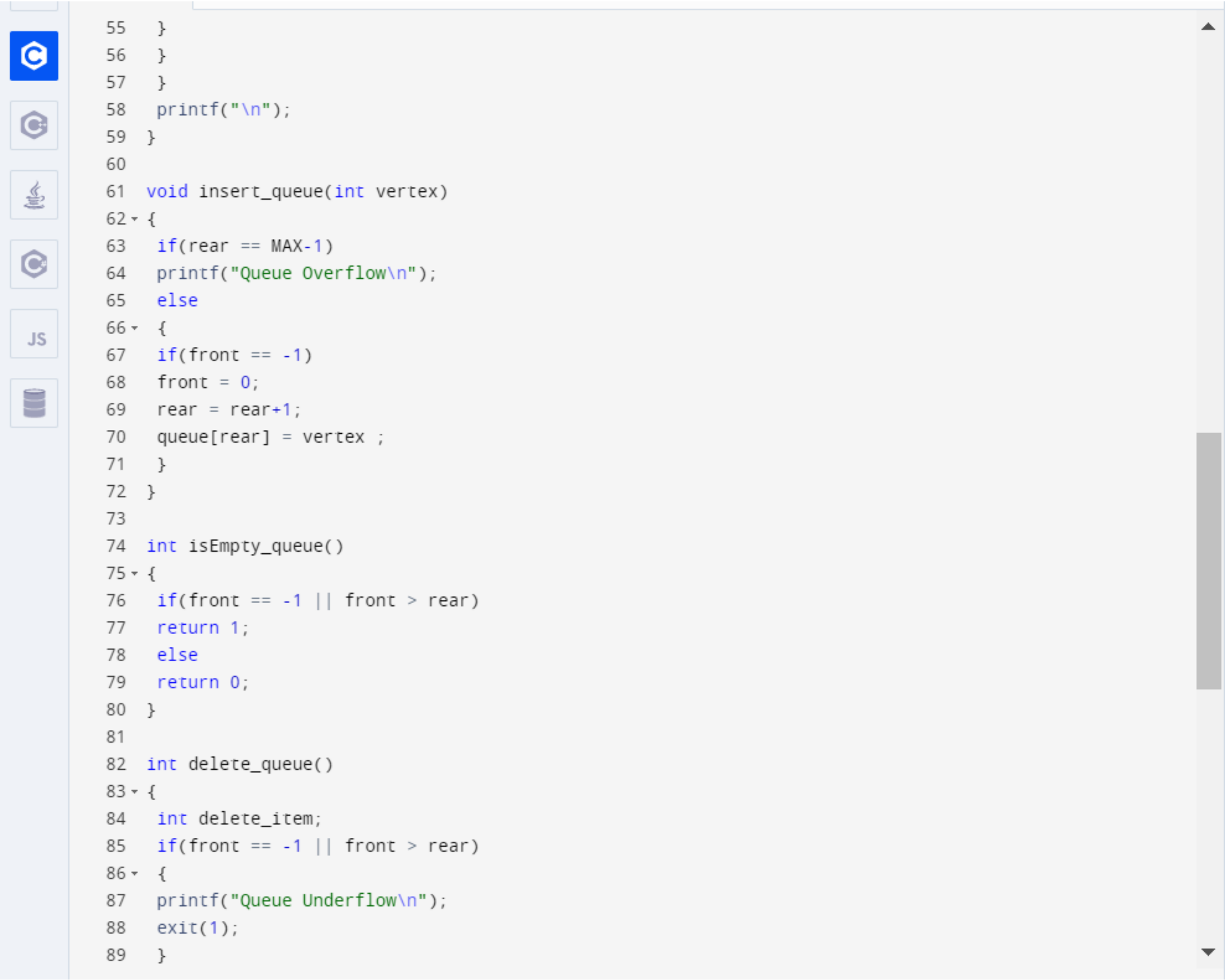
```
/tmp/ZDSUOWJKOA.o
Enter Number of Cities: 5
Enter Cost Matrix:

 Enter Elements of Row # : 1
12 13 14 15 16
Enter Elements of Row # : 2
22 23 14 56 78
Enter Elements of Row # : 3
11 25 46 23 75
Enter Elements of Row # : 4
98 56 34 23 65
Enter Elements of Row # : 5
12 34 23 56 53
The Cost Matrix is:


    12   13   14   15   16

    22   23   14   56   78

    11   25   46   23   75

    98   56   34   23   65

    12   34   23   56   53

The Path is:

1 ===> 5 ===> 3 ===> 4 ===> 2 ===> 1

Minimum cost:140
```



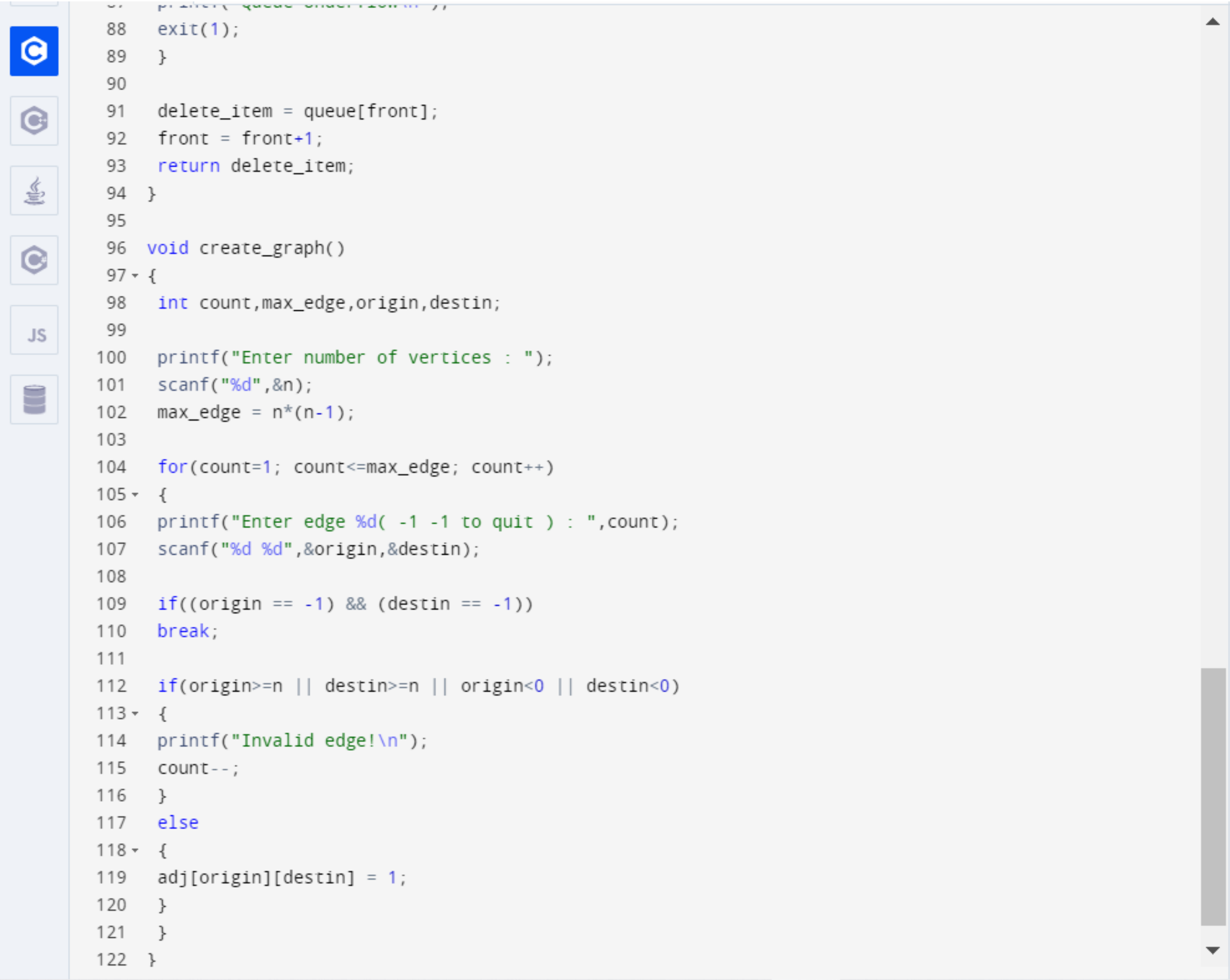
```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);
int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
{
 create_graph();
 BF_Traversal();
 return 0;
}

void BF_Traversal()
{
 int v;

 for(v=0; v<n; v++)
 state[v] = initial;

 printf("Enter Start Vertex for BFS: \n");
 scanf("%d", &v);
 BFS(v);
```



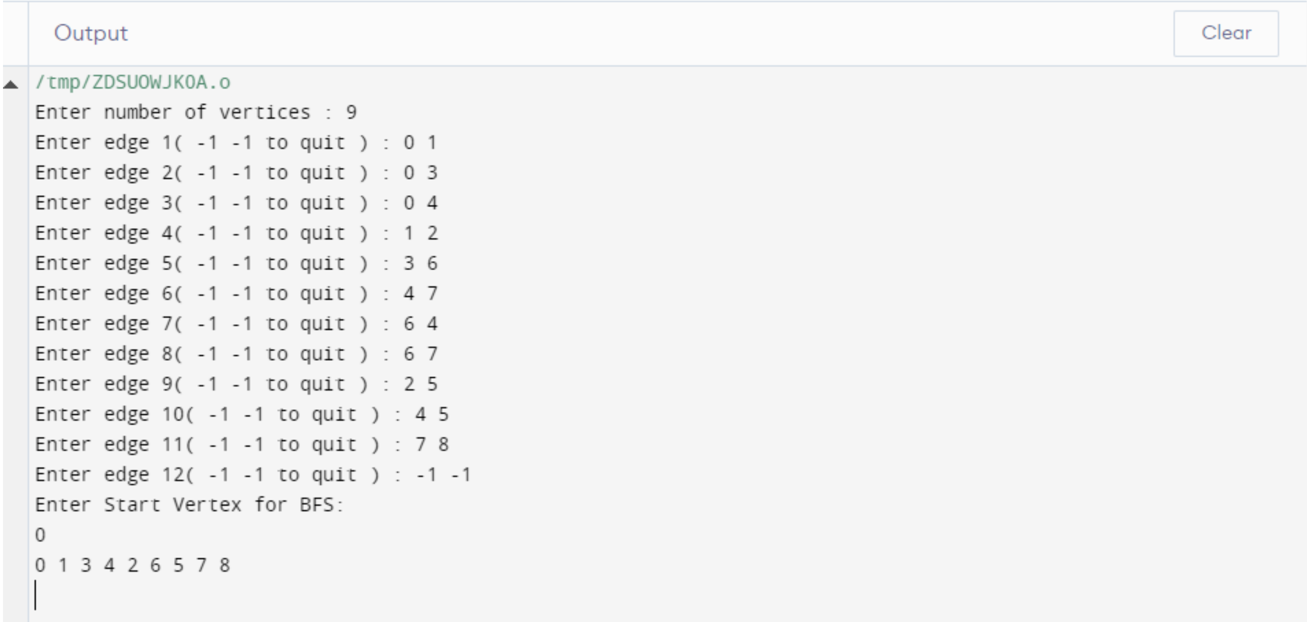
```c
 BFS(v);
}

void BFS(int v)
{
 int i;
 insert_queue(v);
 state[v] = waiting;
 while(!isEmpty_queue())
 {
 v = delete_queue( );
 printf("%d ",v);
 state[v] = visited;

 for(i=0; i<n; i++)
 {
 if(adj[v][i] == 1 && state[i] == initial)
 {
 insert_queue(i);
 state[i] = waiting;
```



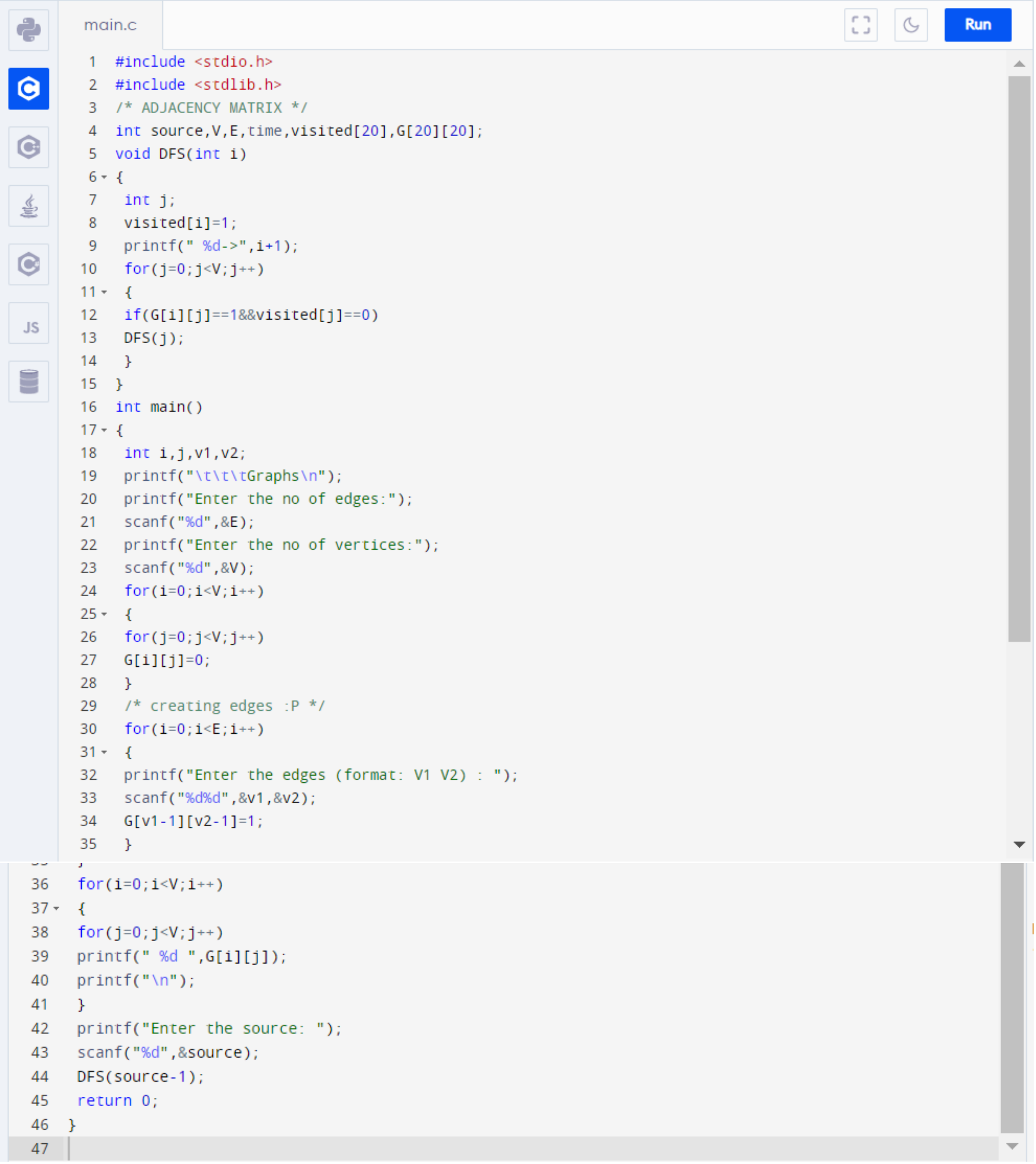
```
 }
 }
 }
 printf("\n");
}

void insert_queue(int vertex)
{
 if(rear == MAX-1)
 printf("Queue Overflow\n");
 else
 {
 if(front == -1)
 front = 0;
 rear = rear+1;
 queue[rear] = vertex ;
 }
}

int isEmpty_queue()
{
 if(front == -1 || front > rear)
 return 1;
 else
 return 0;
}

int delete_queue()
{
 int delete_item;
 if(front == -1 || front > rear)
 {
 printf("Queue Underflow\n");
 exit(1);
 }
```



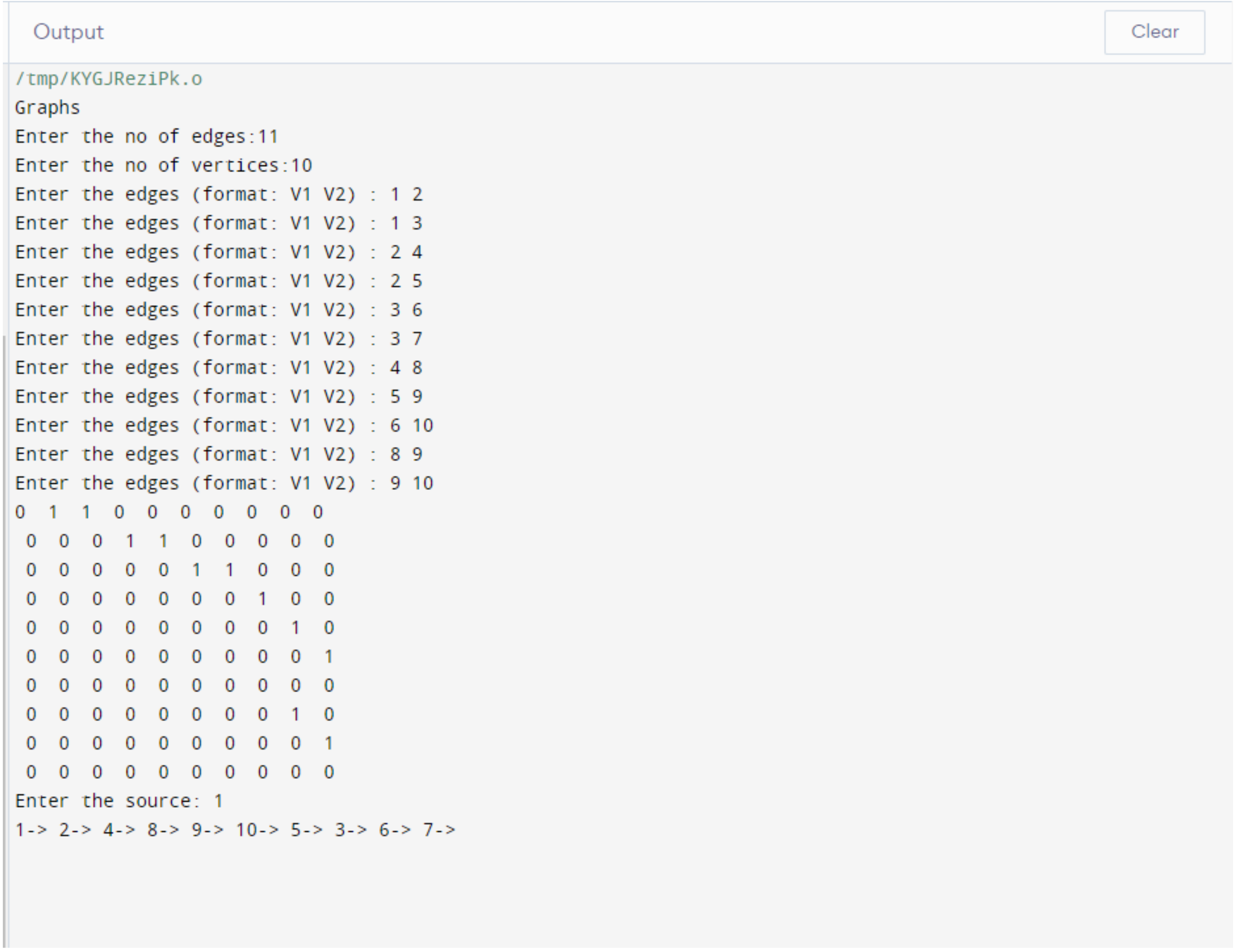
```
 printf("Queue Underflow\n");
 exit(1);
 }

 delete_item = queue[front];
 front = front+1;
 return delete_item;
}

void create_graph()
{
 int count,max_edge,origin,destin;

 printf("Enter number of vertices : ");
 scanf("%d",&n);
 max_edge = n*(n-1);

 for(count=1; count<=max_edge; count++)
 {
 printf("Enter edge %d( -1 -1 to quit ) : ",count);
 scanf("%d %d",&origin,&destin);

 if((origin == -1) && (destin == -1))
 break;

 if(origin>=n || destin>=n || origin<0 || destin<0)
 {
 printf("Invalid edge!\n");
 count--;
 }
 else
 {
 adj[origin][destin] = 1;
 }
 }
}
```



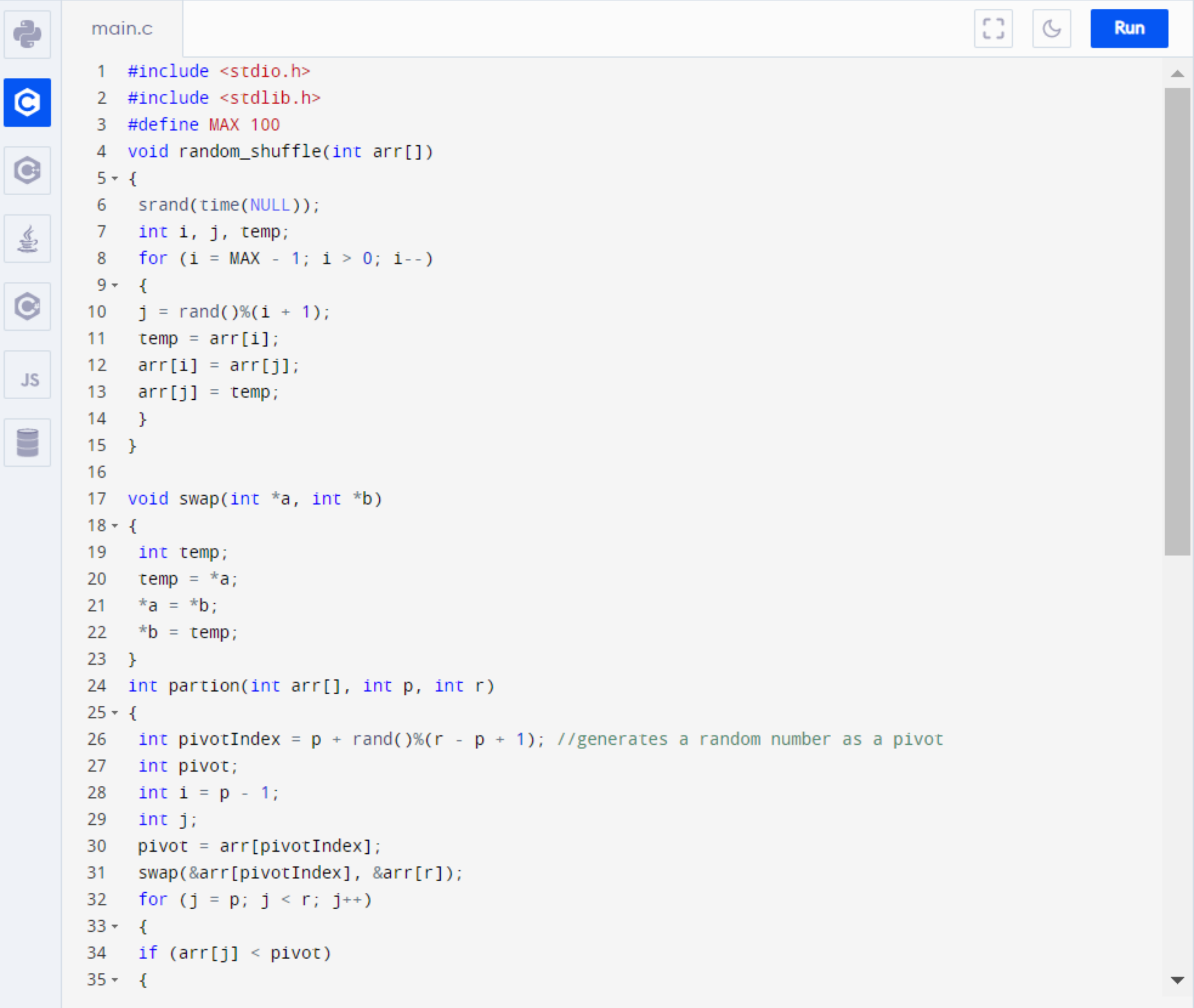
```
/tmp/ZDSUOWJK0A.o
Enter number of vertices : 9
Enter edge 1( -1 -1 to quit ) : 0 1
Enter edge 2( -1 -1 to quit ) : 0 3
Enter edge 3( -1 -1 to quit ) : 0 4
Enter edge 4( -1 -1 to quit ) : 1 2
Enter edge 5( -1 -1 to quit ) : 3 6
Enter edge 6( -1 -1 to quit ) : 4 7
Enter edge 7( -1 -1 to quit ) : 6 4
Enter edge 8( -1 -1 to quit ) : 6 7
Enter edge 9( -1 -1 to quit ) : 2 5
Enter edge 10( -1 -1 to quit ) : 4 5
Enter edge 11( -1 -1 to quit ) : 7 8
Enter edge 12( -1 -1 to quit ) : -1 -1
Enter Start Vertex for BFS:
0
0 1 3 4 2 6 5 7 8
```



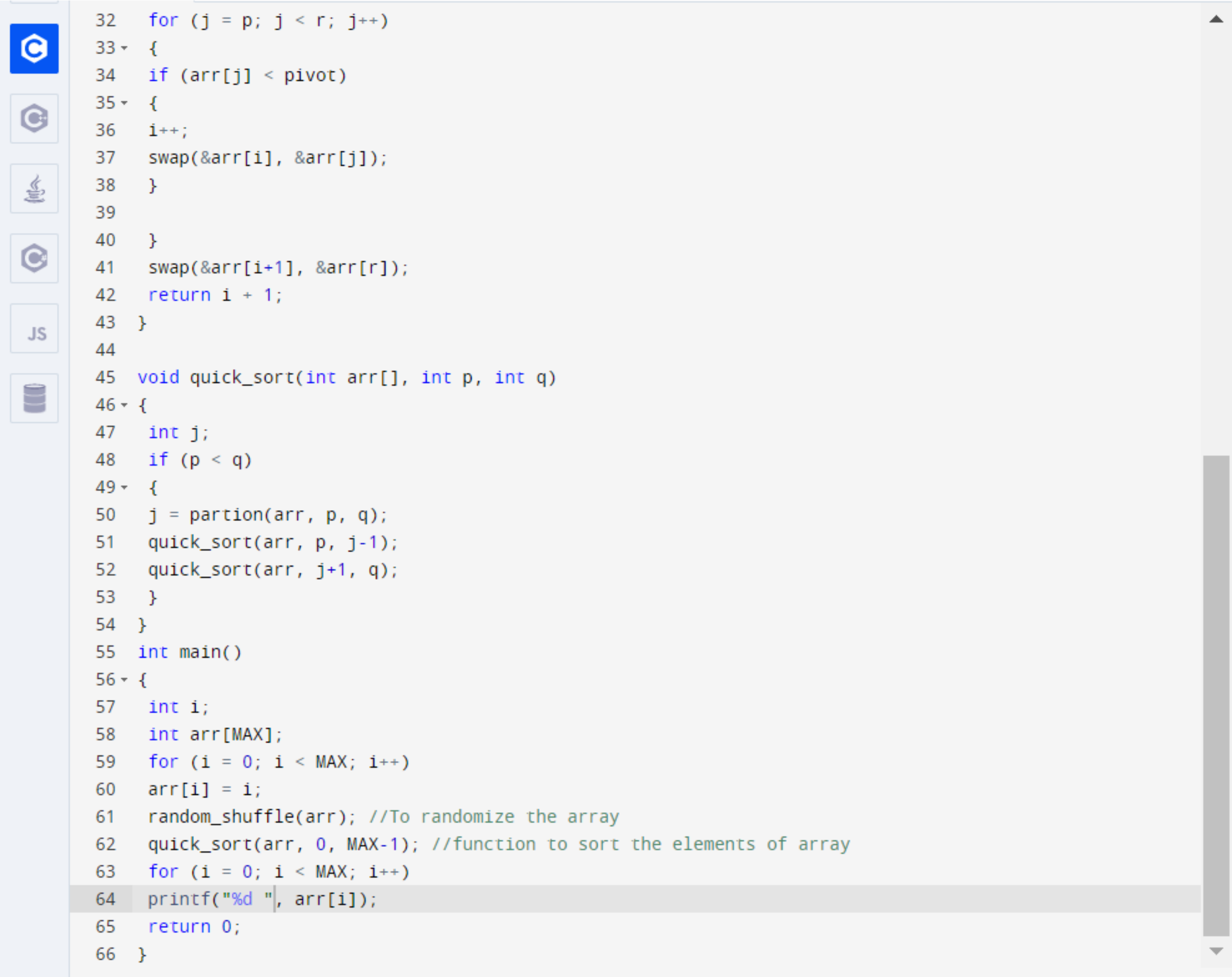
```c
#include <stdio.h>
#include <stdlib.h>
/* ADJACENCY MATRIX */
int source,V,E,time,visited[20],G[20][20];
void DFS(int i)
{
 int j;
 visited[i]=1;
 printf(" %d->",i+1);
 for(j=0;j<V;j++)
 {
 if(G[i][j]==1&&visited[j]==0)
 DFS(j);
 }
}
int main()
{
 int i,j,v1,v2;
 printf("\t\t\tGraphs\n");
 printf("Enter the no of edges:");
 scanf("%d",&E);
 printf("Enter the no of vertices:");
 scanf("%d",&V);
 for(i=0;i<V;i++)
 {
 for(j=0;j<V;j++)
 G[i][j]=0;
 }
 /* creating edges :P */
 for(i=0;i<E;i++)
 {
 printf("Enter the edges (format: V1 V2) : ");
 scanf("%d%d",&v1,&v2);
 G[v1-1][v2-1]=1;
 }
 for(i=0;i<V;i++)
 {
 for(j=0;j<V;j++)
 printf(" %d ",G[i][j]);
 printf("\n");
 }
 printf("Enter the source: ");
 scanf("%d",&source);
 DFS(source-1);
 return 0;
}
```



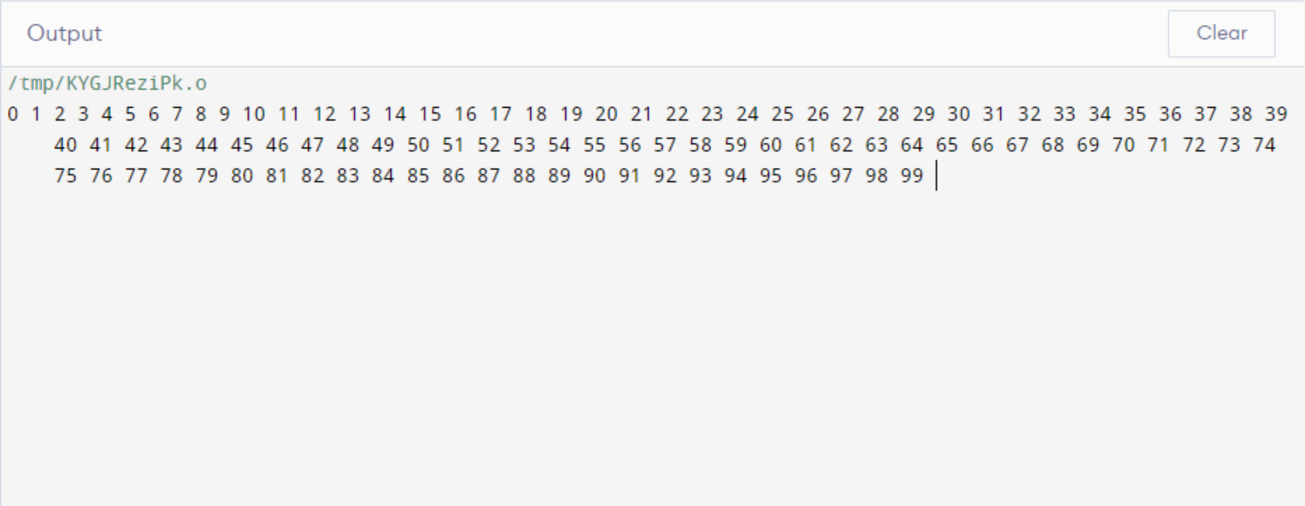
```
/tmp/KYGJReziPk.o
Graphs
Enter the no of edges:11
Enter the no of vertices:10
Enter the edges (format: V1 V2) : 1 2
Enter the edges (format: V1 V2) : 1 3
Enter the edges (format: V1 V2) : 2 4
Enter the edges (format: V1 V2) : 2 5
Enter the edges (format: V1 V2) : 3 6
Enter the edges (format: V1 V2) : 3 7
Enter the edges (format: V1 V2) : 4 8
Enter the edges (format: V1 V2) : 5 9
Enter the edges (format: V1 V2) : 6 10
Enter the edges (format: V1 V2) : 8 9
Enter the edges (format: V1 V2) : 9 10
0  1  1  0  0  0  0  0  0  0
 0  0  0  1  1  0  0  0  0  0
 0  0  0  0  0  1  1  0  0  0
 0  0  0  0  0  0  0  1  0  0
 0  0  0  0  0  0  0  0  1  0
 0  0  0  0  0  0  0  0  0  1
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  0
 0  0  0  0  0  0  0  0  0  1
 0  0  0  0  0  0  0  0  0  0
Enter the source: 1
1-> 2-> 4-> 8-> 9-> 10-> 5-> 3-> 6-> 7->
```



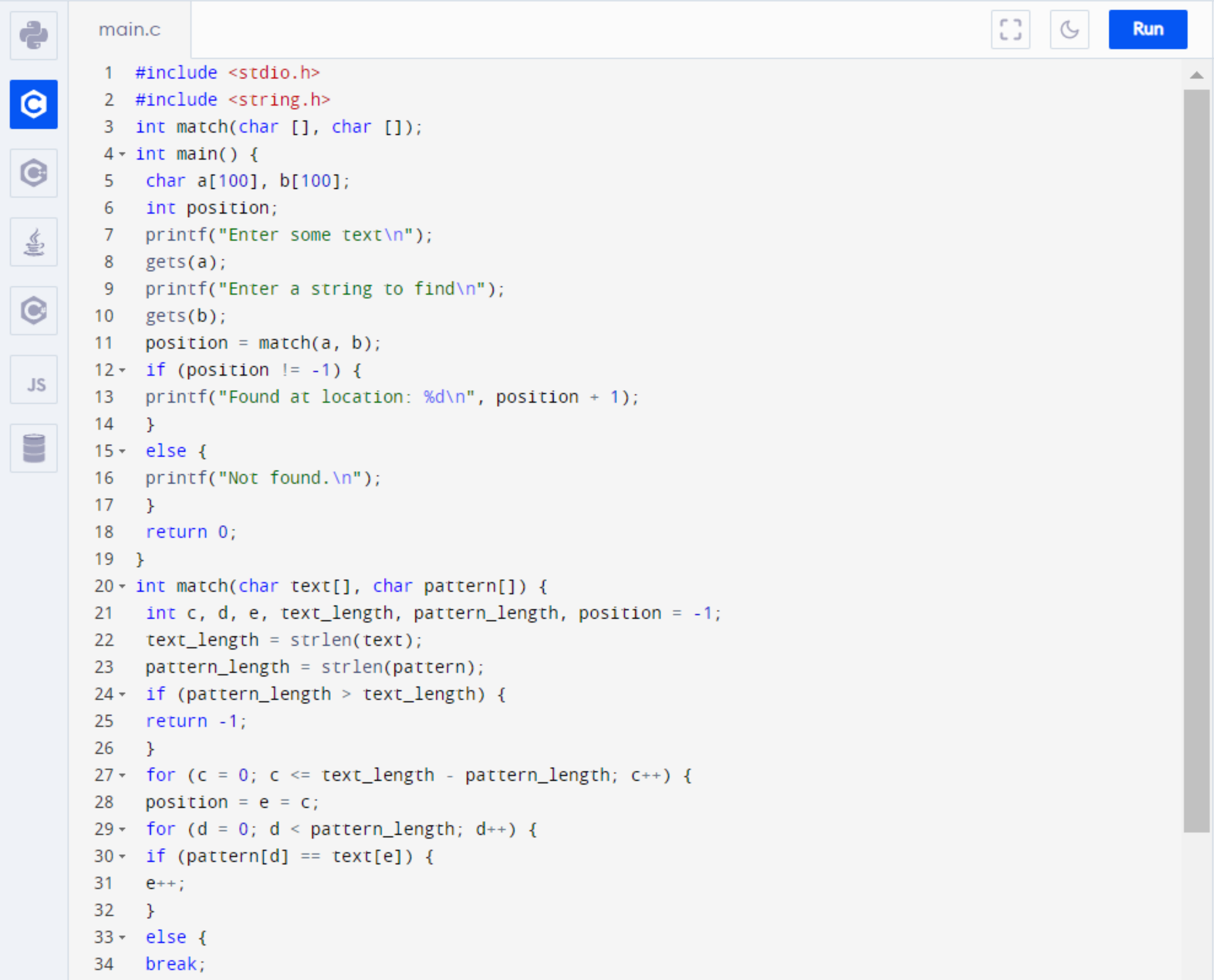
```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
void random_shuffle(int arr[])
{
 srand(time(NULL));
 int i, j, temp;
 for (i = MAX - 1; i > 0; i--)
 {
 j = rand()%(i + 1);
 temp = arr[i];
 arr[i] = arr[j];
 arr[j] = temp;
 }
}

void swap(int *a, int *b)
{
 int temp;
 temp = *a;
 *a = *b;
 *b = temp;
}
int partion(int arr[], int p, int r)
{
 int pivotIndex = p + rand()%(r - p + 1); //generates a random number as a pivot
 int pivot;
 int i = p - 1;
 int j;
 pivot = arr[pivotIndex];
 swap(&arr[pivotIndex], &arr[r]);
 for (j = p; j < r; j++)
 {
 if (arr[j] < pivot)
 {
```



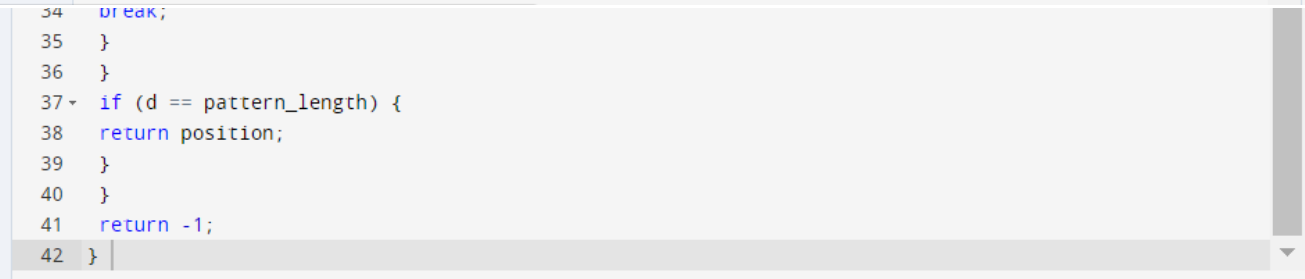
```
 for (j = p; j < r; j++)
 {
 if (arr[j] < pivot)
 {
 i++;
 swap(&arr[i], &arr[j]);
 }

 }
 swap(&arr[i+1], &arr[r]);
 return i + 1;
}

void quick_sort(int arr[], int p, int q)
{
 int j;
 if (p < q)
 {
 j = partion(arr, p, q);
 quick_sort(arr, p, j-1);
 quick_sort(arr, j+1, q);
 }
}
int main()
{
 int i;
 int arr[MAX];
 for (i = 0; i < MAX; i++)
 arr[i] = i;
 random_shuffle(arr); //To randomize the array
 quick_sort(arr, 0, MAX-1); //function to sort the elements of array
 for (i = 0; i < MAX; i++)
 printf("%d ", arr[i]);
 return 0;
}
```



```
/tmp/KYGJReziPk.o
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
    40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
    75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
```

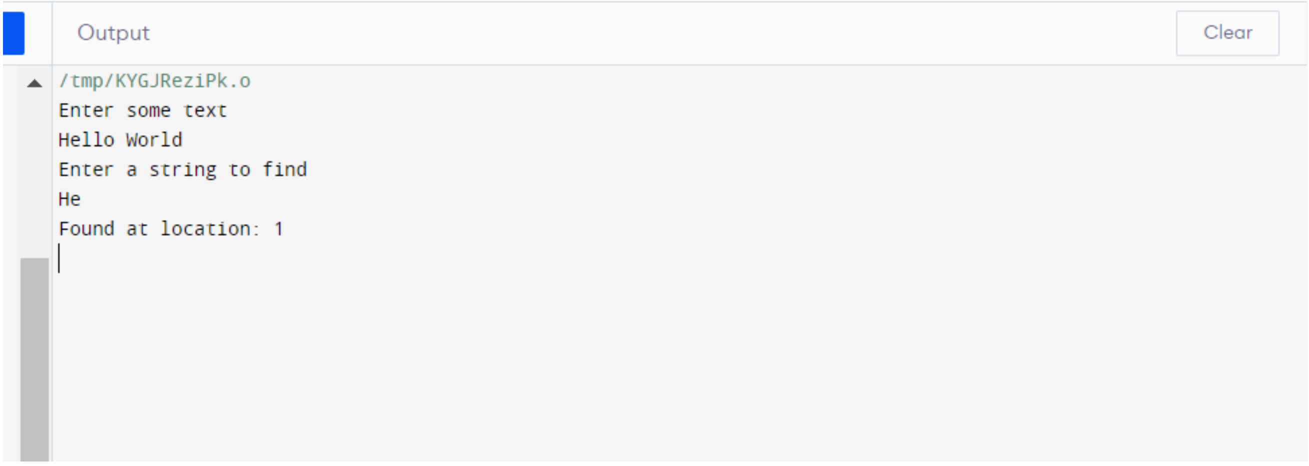


```c
#include <stdio.h>
#include <string.h>
int match(char [], char []);
int main() {
 char a[100], b[100];
 int position;
 printf("Enter some text\n");
 gets(a);
 printf("Enter a string to find\n");
 gets(b);
 position = match(a, b);
 if (position != -1) {
 printf("Found at location: %d\n", position + 1);
 }
 else {
 printf("Not found.\n");
 }
 return 0;
}
int match(char text[], char pattern[]) {
 int c, d, e, text_length, pattern_length, position = -1;
 text_length = strlen(text);
 pattern_length = strlen(pattern);
 if (pattern_length > text_length) {
 return -1;
 }
 for (c = 0; c <= text_length - pattern_length; c++) {
 position = e = c;
 for (d = 0; d < pattern_length; d++) {
 if (pattern[d] == text[e]) {
 e++;
 }
 else {
 break;
```

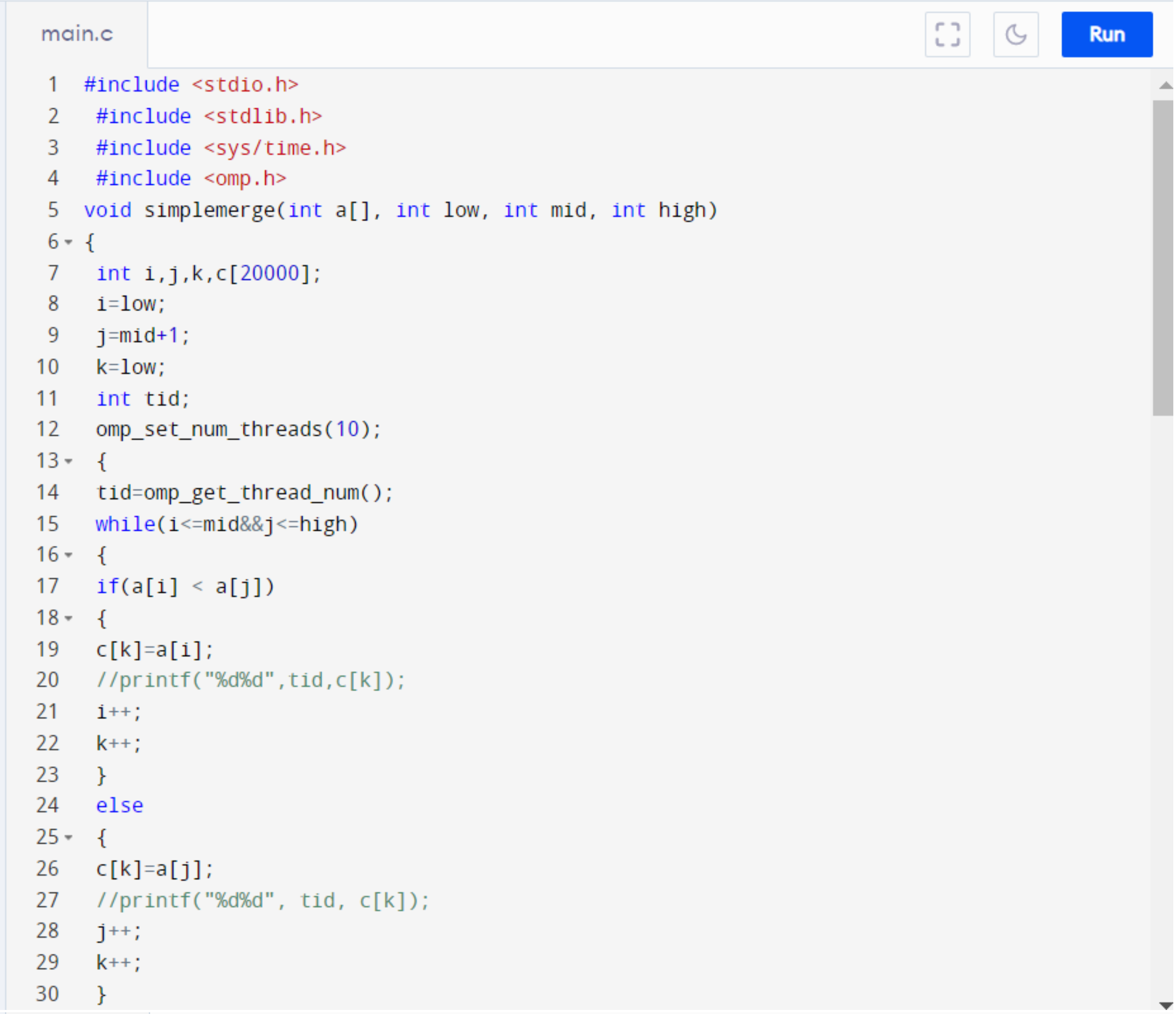


break;

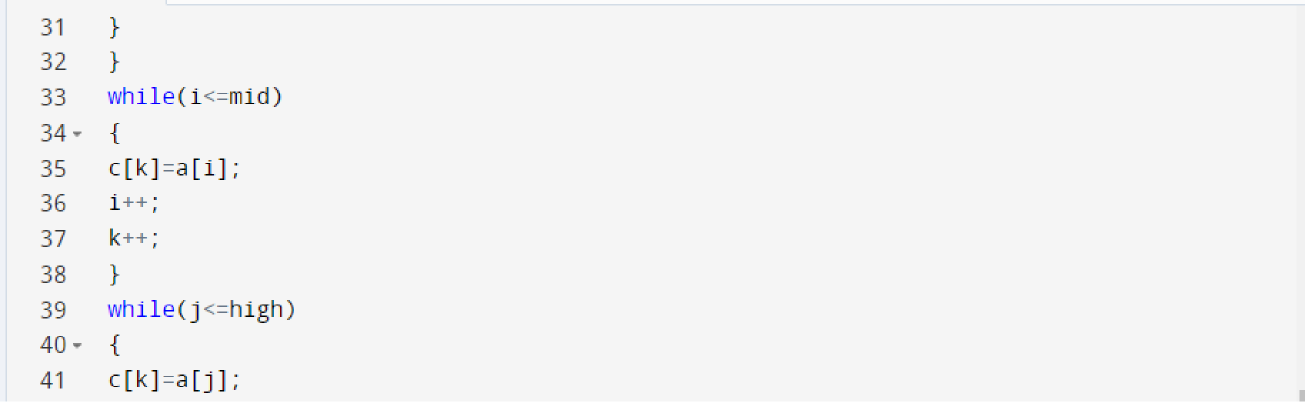
```c
 }
 }
 if (d == pattern_length) {
 return position;
 }
 }
 return -1;
}
```



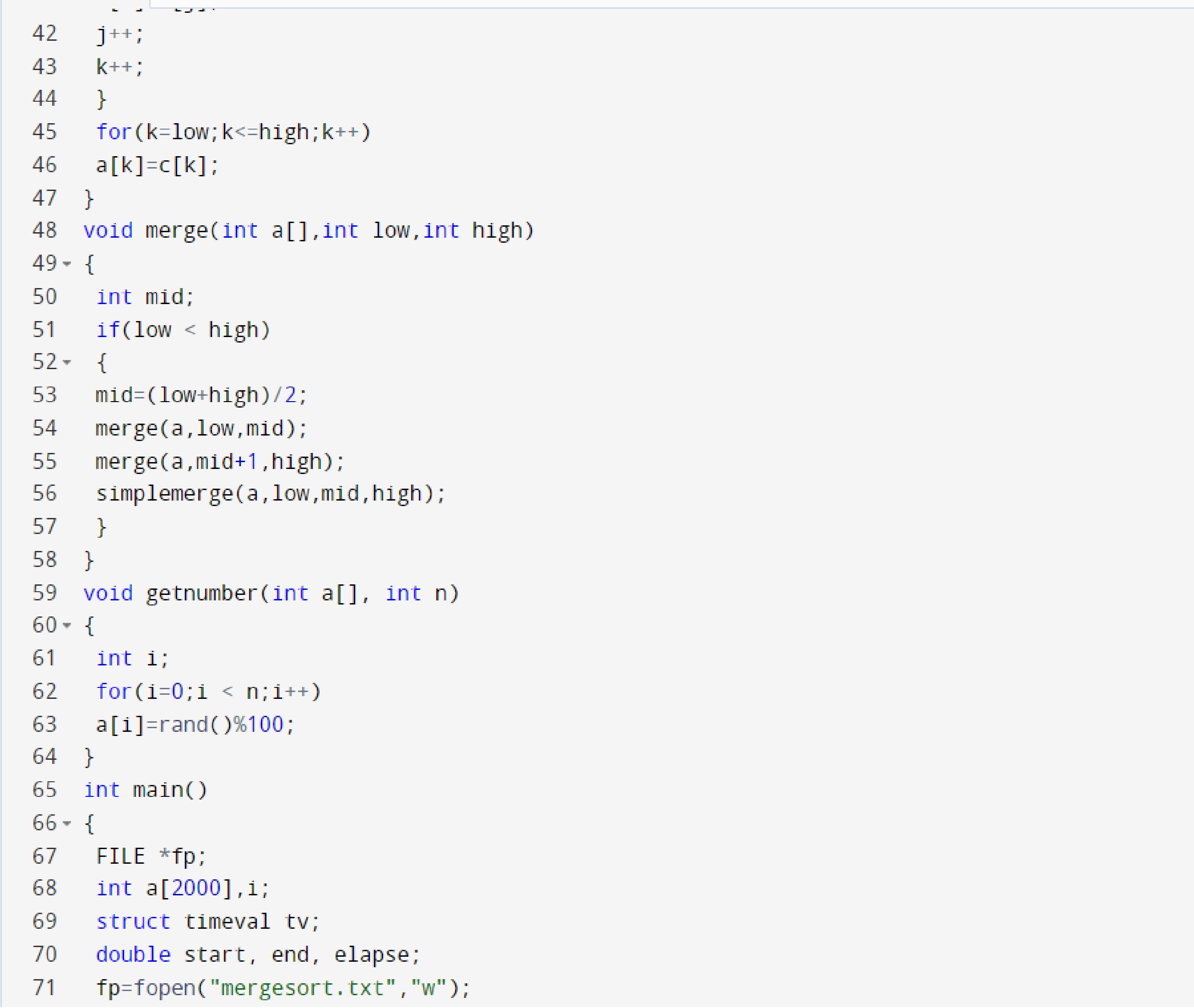
```
/tmp/KYGJReziPk.o
Enter some text
Hello World
Enter a string to find
He
Found at location: 1
```



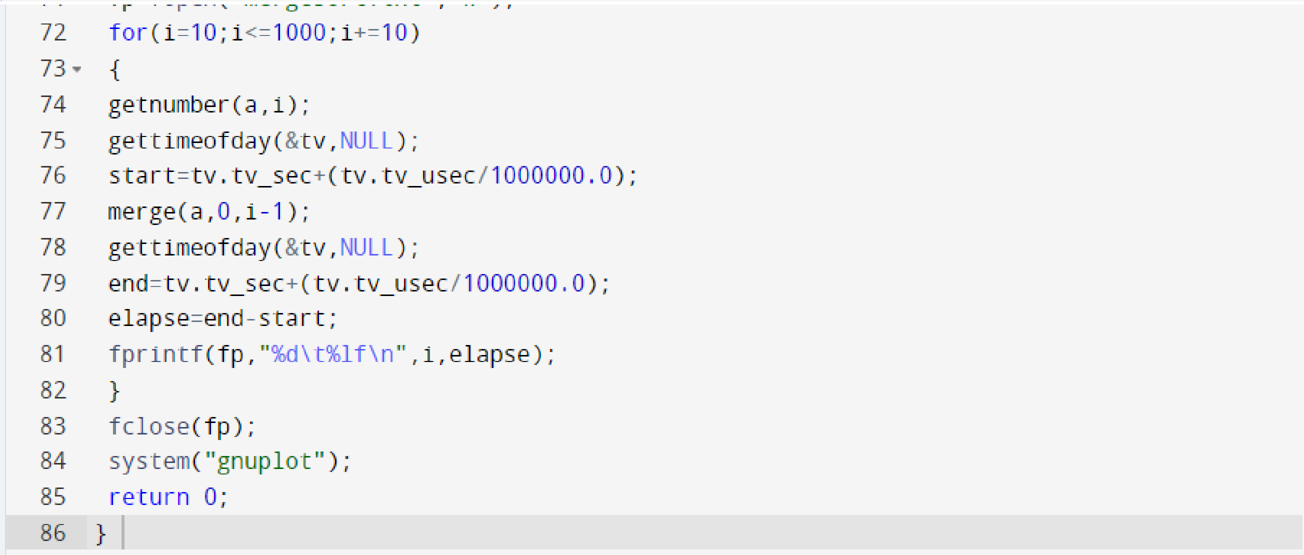
```c
#include <stdio.h>
 #include <stdlib.h>
 #include <sys/time.h>
 #include <omp.h>
void simplemerge(int a[], int low, int mid, int high)
{
 int i,j,k,c[20000];
 i=low;
 j=mid+1;
 k=low;
 int tid;
 omp_set_num_threads(10);
 {
 tid=omp_get_thread_num();
 while(i<=mid&&j<=high)
 {
 if(a[i] < a[j])
 {
 c[k]=a[i];
 //printf("%d%d",tid,c[k]);
 i++;
 k++;
 }
 else
 {
 c[k]=a[j];
 //printf("%d%d", tid, c[k]);
 j++;
 k++;
 }
```



```c
 }
 }
 while(i<=mid)
 {
 c[k]=a[i];
 i++;
 k++;
 }
 while(j<=high)
 {
 c[k]=a[j];
```

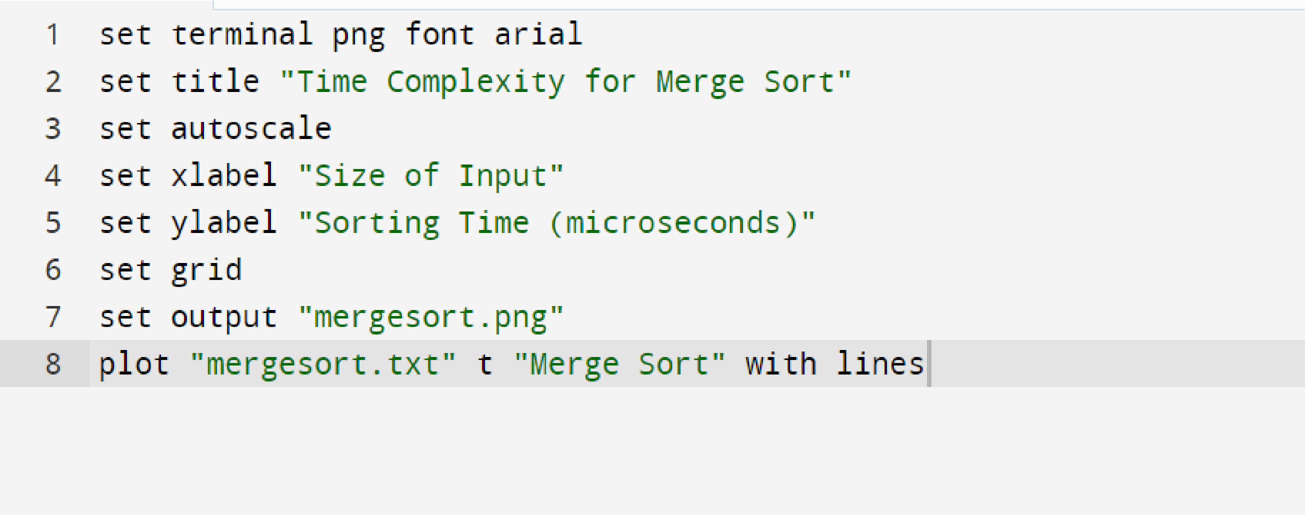


```
 j++;
 k++;
 }
 for(k=low;k<=high;k++)
 a[k]=c[k];
}
void merge(int a[],int low,int high)
{
 int mid;
 if(low < high)
 {
 mid=(low+high)/2;
 merge(a,low,mid);
 merge(a,mid+1,high);
 simplemerge(a,low,mid,high);
 }
}
void getnumber(int a[], int n)
{
 int i;
 for(i=0;i < n;i++)
 a[i]=rand()%100;
}
int main()
{
 FILE *fp;
 int a[2000],i;
 struct timeval tv;
 double start, end, elapse;
 fp=fopen("mergesort.txt","w");
```

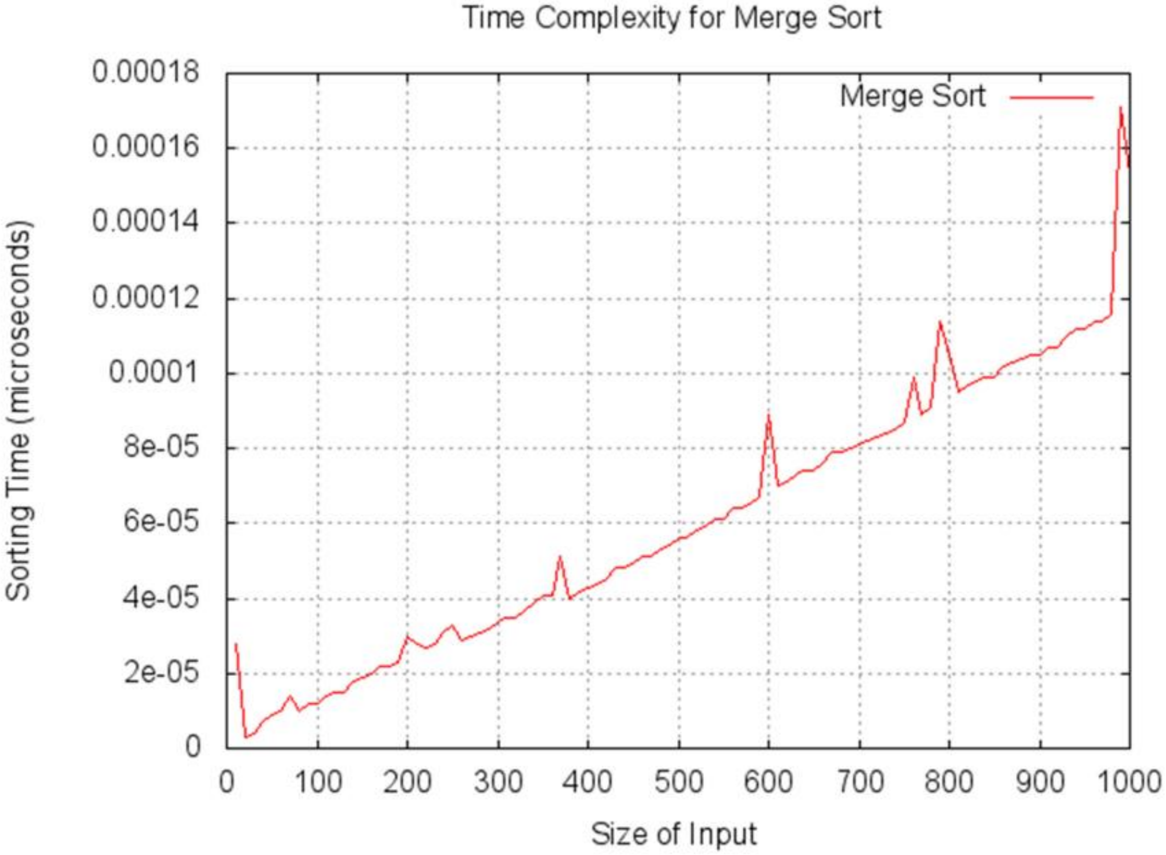


```
 for(i=10;i<=1000;i+=10)
 {
 getnumber(a,i);
 gettimeofday(&tv,NULL);
 start=tv.tv_sec+(tv.tv_usec/1000000.0);
 merge(a,0,i-1);
 gettimeofday(&tv,NULL);
 end=tv.tv_sec+(tv.tv_usec/1000000.0);
 elapse=end-start;
 fprintf(fp,"%d\t%lf\n",i,elapse);
 }
 fclose(fp);
 system("gnuplot");
 return 0;
}
```

Mergesort.gpl



```
set terminal png font arial
set title "Time Complexity for Merge Sort"
set autoscale
set xlabel "Size of Input"
set ylabel "Sorting Time (microseconds)"
set grid
set output "mergesort.png"
plot "mergesort.txt" t "Merge Sort" with lines
```


Time Complexity for Merge Sort
Merge Sort
Sorting Time (microseconds)
0.00018
0.00016
0.00014
0.00012
0.0001
8e-05
6e-05
4e-05
2e-05
0
0
100
200
300
400
500
600
700
800
900
1000
Size of Input