190905514 C SECTION 4^{th} SEMESTER MOHAMMAD TOFIK

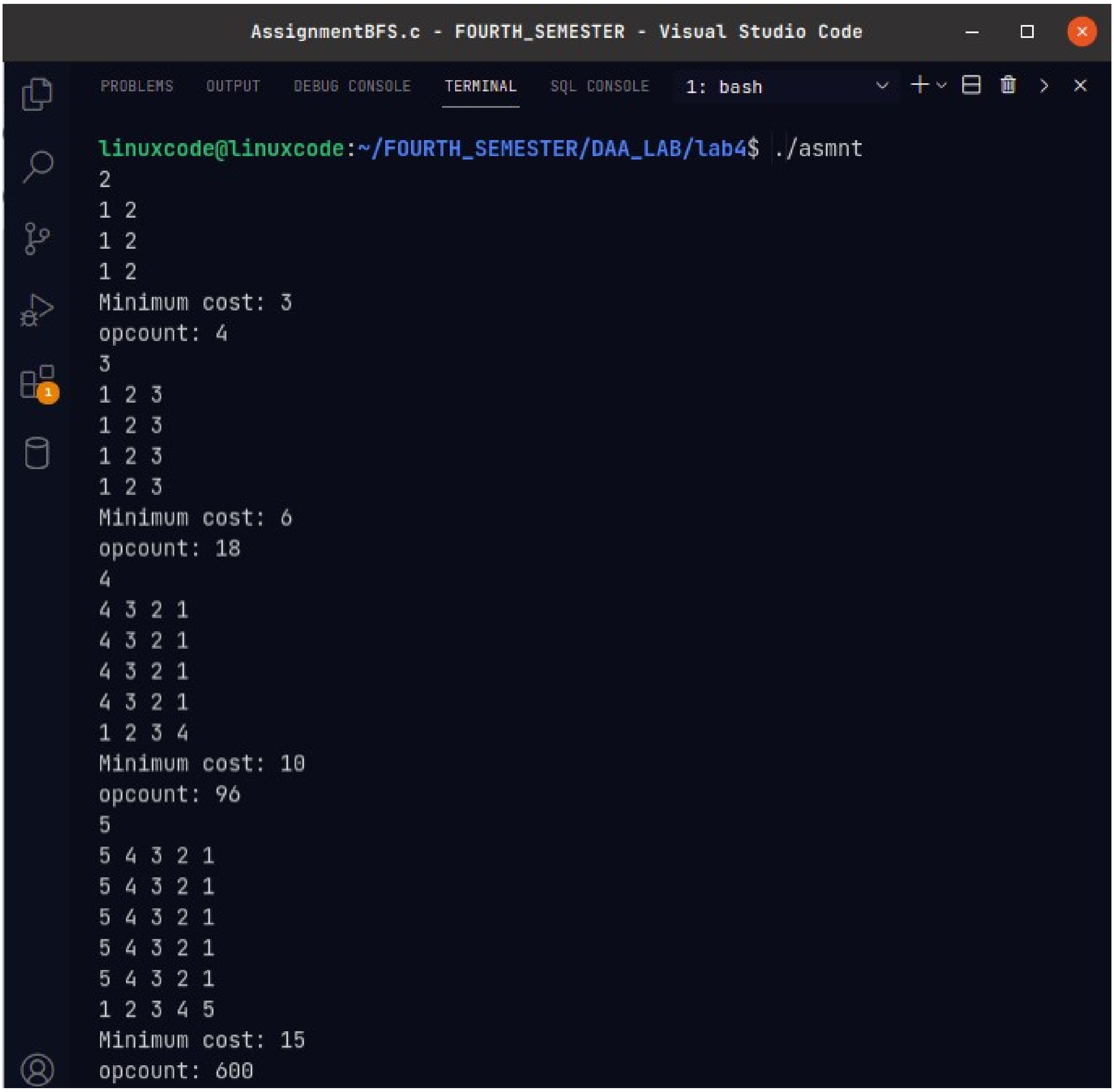
LAB 4 :

1.

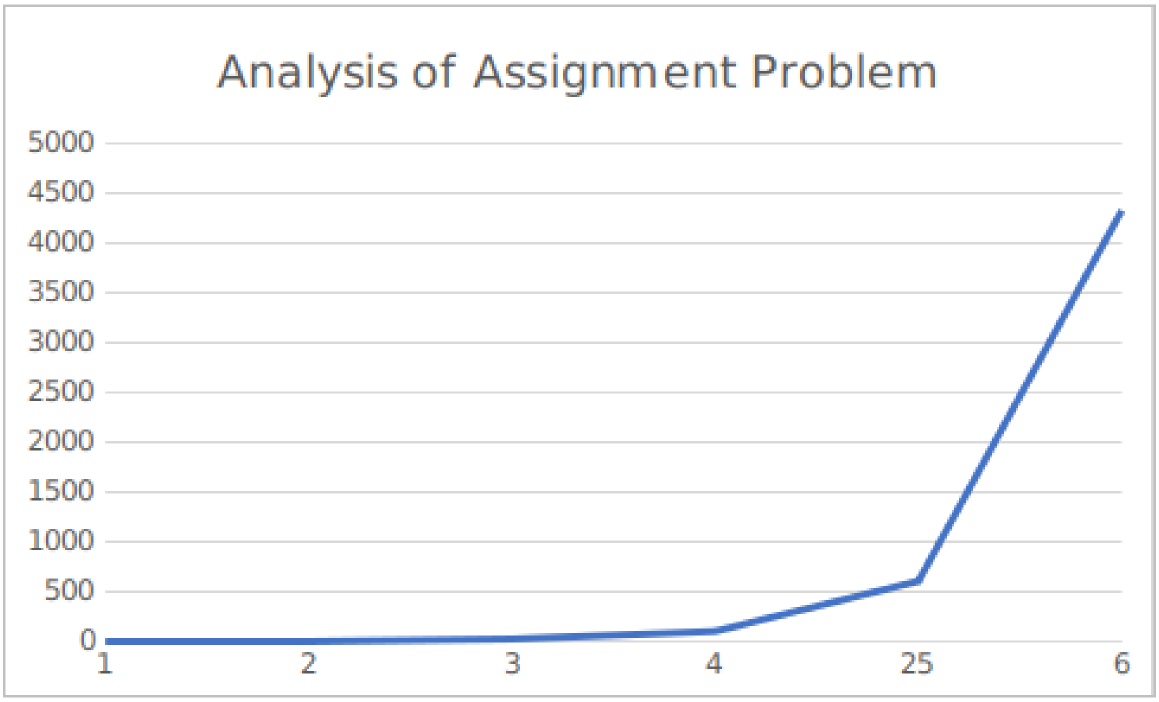
```c
#include <stdio.h>
#include <stdlib.h>
void swap(int arr[], int i, int j) {
int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
}
void genParmutation(int n,int idx,int arr[], int cost[][n],int* min,int* output,int* opcount) {
if(n == idx){
int count = 0;
for(int i=0;i<n;i++)
{ *opcount += 1;
count += cost[i][arr[i]];
}
if(*min>count){
*min = count;
for(int i=0;i<n;i++){
output[i] = arr[i];
}
}
return;
}
for(int i = idx;i<n;i++){
swap(arr,i,idx);
genParmutation(n,idx+1,arr,cost,min,output,opcount);
swap(arr,i,idx);
}
}

int main(int argc, char const *argv[])
{
```

```c
for(int r = 0;r<5;r++){
int n;
scanf("%d",&n);
int arr[n];
for (int i = 0; i < n; i++) {
arr[i] = i;
}
int cost[n][n];
for(int i=0;i<n;i++){
for(int j=0;j<n;j++){
scanf("%d",&cost[i][j]);
}
}
int* output = (int*)malloc(sizeof(int)*n);
int min = 1000;
int opcount = 0;
genParmutation(n,0,arr,cost,&min,output,&opcount);
for(int i=0;i<n;i++)
printf("%d ",output[i]+1 );
printf("\nMinimum cost: %d\n",min);
printf("opcount: %d\n",opcount );
}
return 0;
}
```

OutPUT :



```
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab4$ ./asmnt
2
1 2
1 2
1 2
Minimum cost: 3
opcount: 4
3
1 2 3
1 2 3
1 2 3
1 2 3
Minimum cost: 6
opcount: 18
4
4 3 2 1
4 3 2 1
4 3 2 1
4 3 2 1
1 2 3 4
Minimum cost: 10
opcount: 96
5
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
1 2 3 4 5
Minimum cost: 15
opcount: 600
```



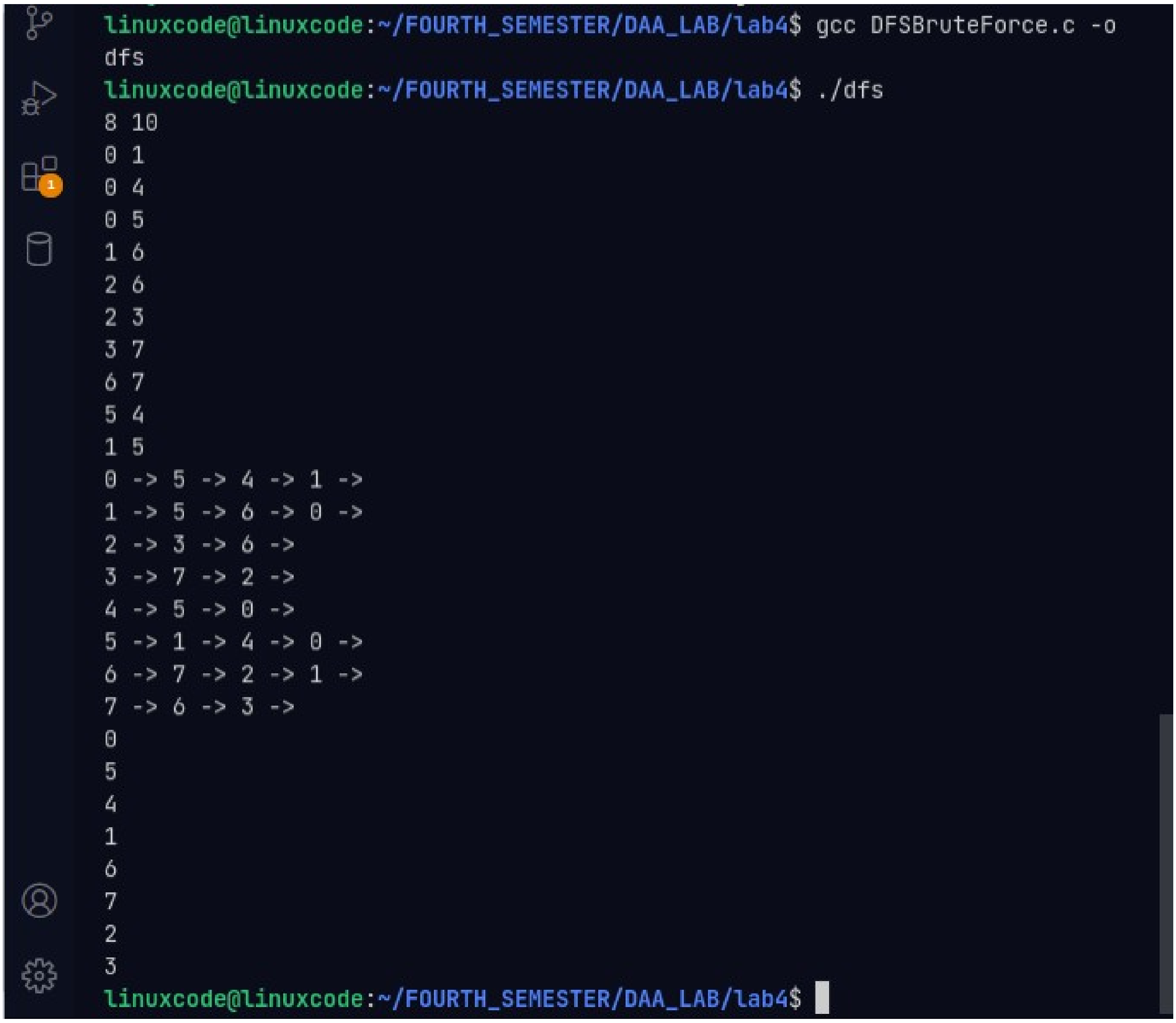
2.

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

typedef struct List
{
int data;
int visit;
struct List *right;
} List;

List *createNode(int data)
{
List *temp = (List *)malloc(sizeof(List));
temp->data = data;
temp->visit = 0;
temp->right = NULL;
return temp;
}
```

```c
void displayList(List **list, int v)
{
for (int i = 0; i < v; i++)
{
printf("%d -> ", i);
List *a = list[i]->right;
int j = 10;
while (a != NULL && j--)
{
printf("%d -> ", a->data);
a = a->right;
}
printf("\n");
}
}

List **createGraphAdjacencyList(int v, int e, int arr[]
[2])
{
List **node = (List **)malloc(sizeof(List *) * v);
for (int i = 0; i < v; i++)
{
node[i] = createNode(i);
}
for (int i = 0; i < e; i++)
{
int j = 0;
List **temp = node;
while (j < v)
{
if (temp[j]->data == arr[i][0])
{
List *temp1 = createNode(arr[i][1]);
temp1->right = temp[j]->right;
temp[j]->right = temp1;
temp1 = createNode(arr[i][0]);
temp1->right = temp[arr[i][1]]->right;
temp[arr[i][1]]->right = temp1;
break;
}
else
{
j++;
}
}
}
return node;
}

void breadthFirstSearch(List **list, int v)
{
List *current = list[0];
current->visit = 1;
printf("%d\n", current->data);
int count = 1;
int q[v];
int front = -1;
int rear = -1;
q[++front] = current->data;
while (front != rear)
{
current = list[q[front]];
rear++;

while (current != NULL)
{
current = current->right;

if (current == NULL)
{
break;
}
if (list[current->data]->visit != 0)
```

```
{
continue;
}
list[current->data]->visit = ++count;
printf("%d\n", current->data);
q[++front] = current->data;
}
}
}

int main()
{
int v, e;
scanf("%d %d", &v, &e);
int arr[e][2];
for (int i = 0; i < e; i++)
{
scanf("%d %d", &arr[i][0], &arr[i][1]);
}
List **list = createGraphAdjacencyList(v, e, arr);
displayList(list, v);
breadthFirstSearch(list, v);
return 0;
}
```

OUTPUT :



```
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab4$ gcc DFSBruteForce.c -o
dfs
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab4$ ./dfs
8 10
0 1
0 4
0 5
1 6
2 6
2 3
3 7
6 7
5 4
1 5
0 -> 5 -> 4 -> 1 ->
1 -> 5 -> 6 -> 0 ->
2 -> 3 -> 6 ->
3 -> 7 -> 2 ->
4 -> 5 -> 0 ->
5 -> 1 -> 4 -> 0 ->
6 -> 7 -> 2 -> 1 ->
7 -> 6 -> 3 ->
0
5
4
1
6
7
2
3
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab4$
```

3.

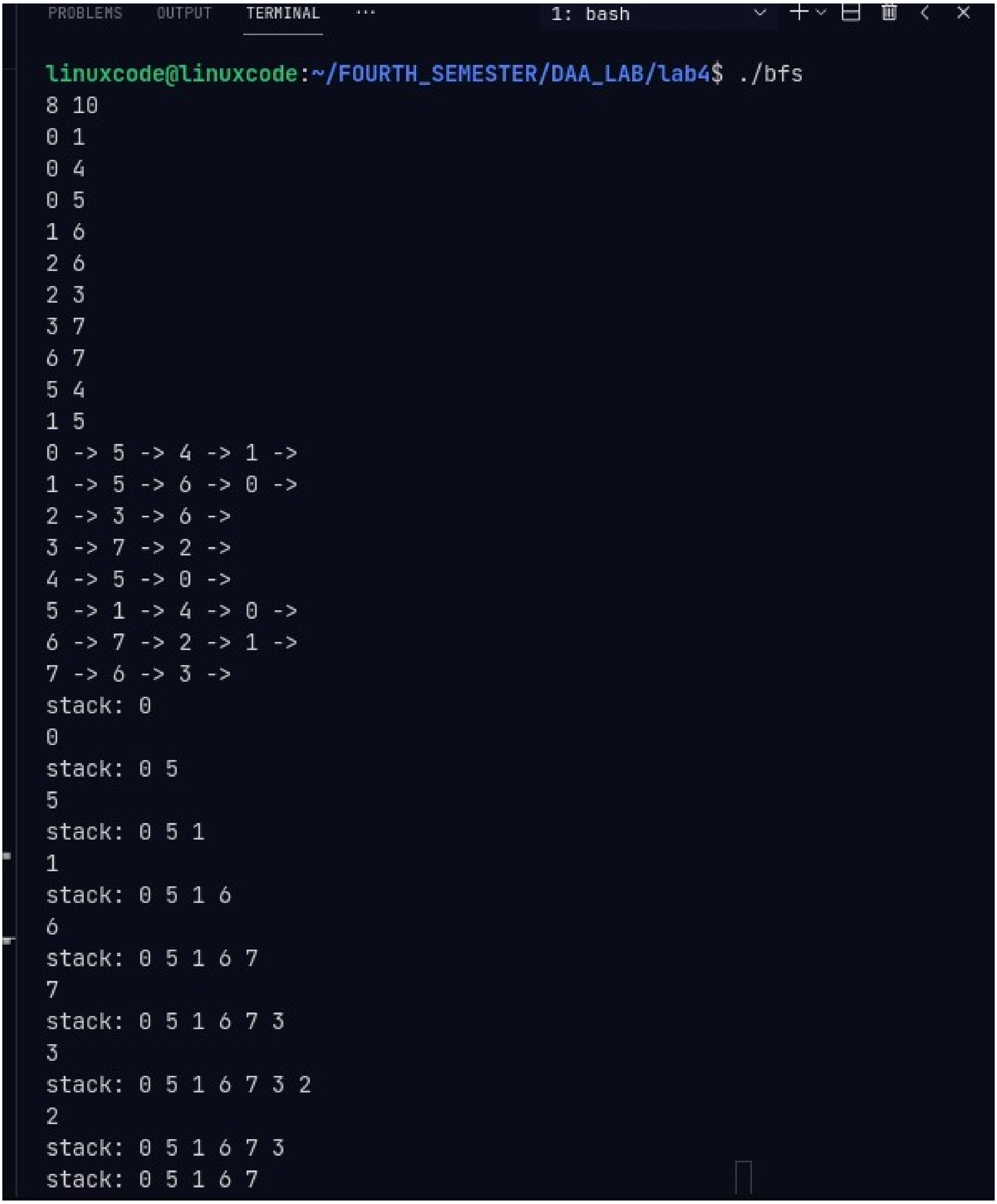
```
#include <stdio.h>
#include <stdlib.h>

typedef struct adjacency_list
{
int data;
int visit;
struct adjacency_list *right;
} adjacency_list;
```

```
adjacency_list *createNode(int data)
{
adjacency_list *temp = (adjacency_list
*)malloc(sizeof(adjacency_list));
temp->data = data;
temp->visit = 0;
temp->right = NULL;
return temp;
}

void display_adjacency_list(adjacency_list **list, int
v)
{
for (int i = 0; i < v; i++)
{
printf("%d -> ", i);
adjacency_list *a = list[i]->right;
int j = 10;
while (a != NULL && j--)
{
printf("%d -> ", a->data);
a = a->right;
}
printf("\n");
}
}

adjacency_list **createGraphAdjacencyList(int v, int e,
int arr[][2])
{
adjacency_list **node = (adjacency_list
**)malloc(sizeof(adjacency_list *) * v);
for (int i = 0; i < v; i++)
{
node[i] = createNode(i);
}
```

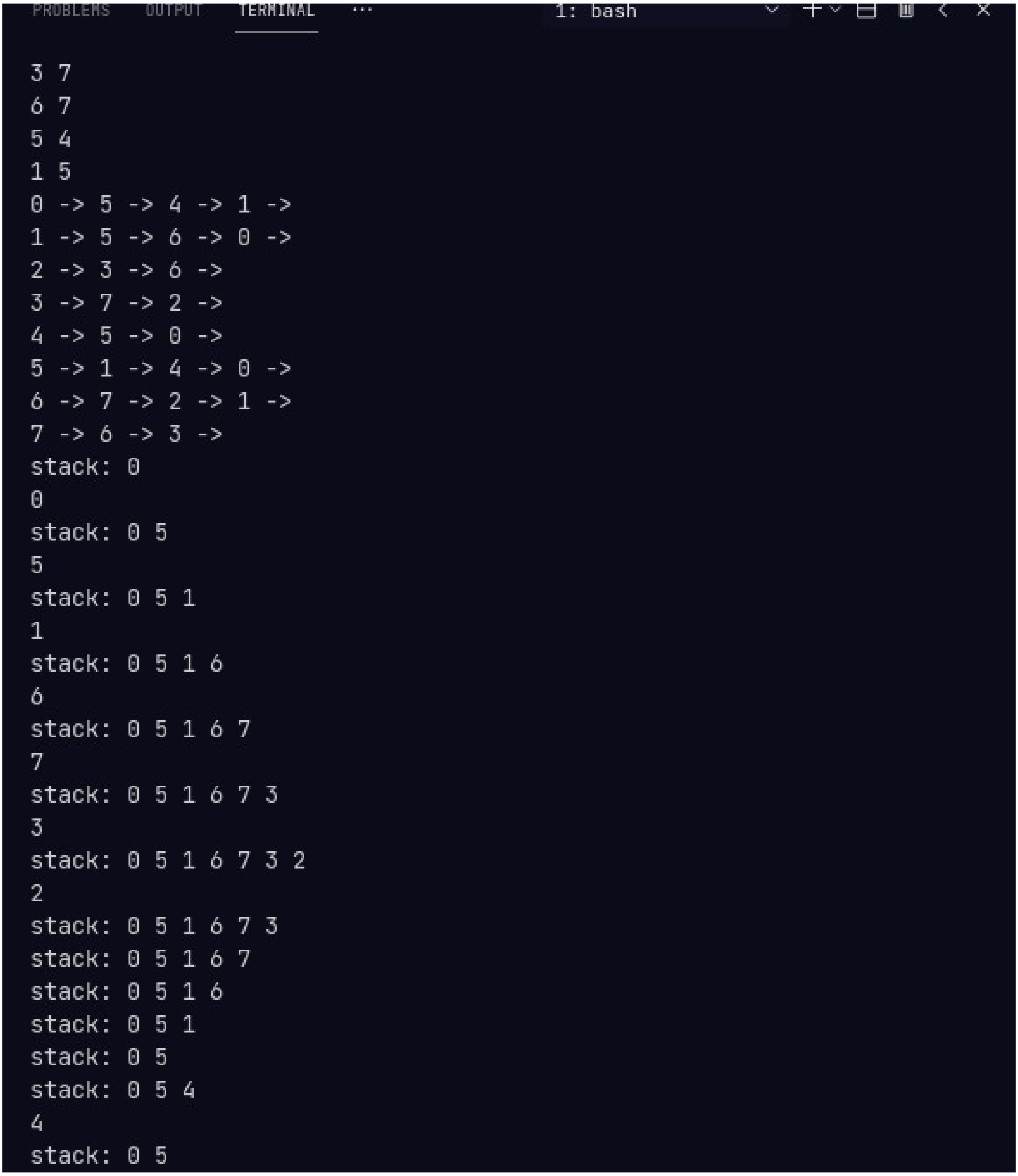
```c
for (int i = 0; i < e; i++)
{
int j = 0;
adjacency_list **temp = node;
while (j < v)
{
if (temp[j]->data == arr[i][0])
{
adjacency_list *temp1 = createNode(arr[i][1]);
temp1->right = temp[j]->right;
temp[j]->right = temp1;
temp1 = createNode(arr[i][0]);
temp1->right = temp[arr[i][1]]->right;
temp[arr[i][1]]->right = temp1;
break;
}
else
{
j++;
}
}
}
return node;
}

void depthFirstSearch(adjacency_list **list, int v)
{
int s[v];
int top = -1;
s[++top] = list[0]->data;
int count = 0;
while (count != v)
{
printf("stack: ");
for (int i = 0; i <= top; i++)
{
printf("%d ", s[i]);
}
printf("\n");
adjacency_list *a = list[s[top]];
adjacency_list *b = a;
if (list[b->data]->visit == 0)
{
list[b->data]->visit = ++count;
printf("%d\n", b->data);
}
while (a != NULL && list[a->data]->visit != 0)
{
a = a->right;
}
if (a != NULL)
{
s[++top] = a->data;
}
else
{
top--;
}
}
printf("stack: ");
for (int i = 0; i <= top; i++)
{
printf("%d ", s[i]);
}
printf("\n");
}

int main()
{
int v, e;
scanf("%d %d", &v, &e);
int arr[e][2];
for (int i = 0; i < e; i++)
{
```

```
scanf("%d %d", &arr[i][0], &arr[i][1]);
}
adjacency_list **list = createGraphAdjacencyList(v, e,
arr);
display_adjacency_list(list, v);
depthFirstSearch(list, v);
return 0;
}
```

OUTPUT :



```
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab4$ ./bfs
8 10
0 1
0 4
0 5
1 6
2 6
2 3
3 7
6 7
5 4
1 5
0 -> 5 -> 4 -> 1 ->
1 -> 5 -> 6 -> 0 ->
2 -> 3 -> 6 ->
3 -> 7 -> 2 ->
4 -> 5 -> 0 ->
5 -> 1 -> 4 -> 0 ->
6 -> 7 -> 2 -> 1 ->
7 -> 6 -> 3 ->
stack: 0
0
stack: 0 5
5
stack: 0 5 1
1
stack: 0 5 1 6
6
stack: 0 5 1 6 7
7
stack: 0 5 1 6 7 3
3
stack: 0 5 1 6 7 3 2
2
stack: 0 5 1 6 7 3
stack: 0 5 1 6 7
```



```
3 7
6 7
5 4
1 5
0 -> 5 -> 4 -> 1 ->
1 -> 5 -> 6 -> 0 ->
2 -> 3 -> 6 ->
3 -> 7 -> 2 ->
4 -> 5 -> 0 ->
5 -> 1 -> 4 -> 0 ->
6 -> 7 -> 2 -> 1 ->
7 -> 6 -> 3 ->
stack: 0
0
stack: 0 5
5
stack: 0 5 1
1
stack: 0 5 1 6
6
stack: 0 5 1 6 7
7
stack: 0 5 1 6 7 3
3
stack: 0 5 1 6 7 3 2
2
stack: 0 5 1 6 7 3
stack: 0 5 1 6 7
stack: 0 5 1 6
stack: 0 5 1
stack: 0 5
stack: 0 5 4
4
stack: 0 5
```

EFFICIENCY :

The time complexity for BFS with adjacency matrix is O(V2) where V is the number of vertices. This is because for every vertex v, we are checking if each vertex from 0 to v-1 is adjacent and if it is, then we are checking if it has been visited and if not, adding it to the queue. This takes O(V) time. Since there are V vertices and each of them takes O(V) time, the time complexity is O(V2).