



## GRAPH

Name:	Wilson Vidyut Doloy
Reg No:	19BCE1603
Subject:	CSE2003 Data Structures and Algorithms Lab
Slot:	L67+L68 (Prof. Sandeep Kumar)
Date:	16 October 2020

**Q2.** You are given an undirected weighted graph of  $n$  nodes (0-indexed), represented by an edge list where `edges[i] = [a, b]` is an undirected edge connecting the nodes `a` and `b` with a probability of success of traversing that edge `succProb[i]`.

Given two nodes `start` and `end`, find the path with the maximum probability of success to go from `start` to `end` and return its success probability.

If there is no path from `start` to `end`, **return 0**. Your answer will be accepted if it differs from the correct answer by at most **1e-5**.

### CODE:

```
#include <stdio.h>

struct mygraph
{
    double next;
    double val;
}mygraph;
```

```
void help(int n, struct mygraph** graph, int index, int end, double *state, double current, double
*xx,int *col)

{
    if(current<=0.00005) return;

    if(state[index]>=current)
    {
        return;
    }

    state[index] = current;

    if(index == end)
    {
        if(*xx<current) *xx = current;

        return ;
    }
}

int i;
for(i=0;i<col[index];i++)
{
    help(n, graph, graph[index][i].next, end, state, current*graph[index][i].val, xx,col);
}
}

double maxProbability(int n, int** edges, int edgesSize, int* edgesColSize, double* succProb, int
succProbSize, int start, int end){

double *state = (double*)calloc(sizeof(double) , n);

struct mygraph **graph = ( struct mygraph**)calloc(sizeof( struct mygraph*), n);
```

```

int *col = (int*)calloc(sizeof(int),n);

int i;

for(i=0;i<edgesSize;i++)

{

    int temp0 = edges[i][0];

    int temp1 = edges[i][1];

    if(col[temp0]==0)

    {

        graph[temp0] = ( struct mygraph*)malloc(sizeof( struct mygraph ));

    }

    else

        graph[temp0] = ( struct mygraph*)realloc(graph[temp0],sizeof( struct mygraph )

*(1+col[temp0]));

    if(col[temp1]==0)

    {

        graph[temp1] = ( struct mygraph*)malloc(sizeof( struct mygraph));

    }

    else

        graph[temp1] = ( struct mygraph*)realloc(graph[temp1],sizeof( struct mygraph )

*(1+col[temp1]));

    graph[temp0][col[temp0]].next=temp1;

    graph[temp0][col[temp0]++].val = succProb[i];

    graph[temp1][col[temp1]].next=temp0;

    graph[temp1][col[temp1]++].val = succProb[i];

}

double max=-1;

help(n, graph, start, end, state, 1, &max,col);

return max== -1?0:max;

```

}

## OUTPUT:

The screenshot shows a C code editor interface with the following details:

- Code Area:** Displays a C program with line numbers 1 to 21. The code defines a struct `mygraph` and a function `help`. The `help` function takes parameters `n`, `graph`, `index`, `end`, `state`, `current`, `xx`, and `col`. It checks if `current` is less than or equal to 0.00005, or if `state[index]` is greater than or equal to `current`. If either condition is true, it returns. Otherwise, it sets `state[index] = current`, checks if `index == end`, and if so, updates `xx` to `current` if `*xx < current`. Finally, it returns.
- Toolbars:** Top toolbar includes icons for file operations (New, Open, Save, Print) and a dropdown for Autocomplete.
- Bottom Navigation:** Buttons for Testcase, Run Code Result, Debugger, and a dropdown menu.
- Result Area:** Shows the code is Accepted with a runtime of 0 ms.
- Input/Output Fields:** Your input: `3 [[0,1],[1,2],[0,2]]`; Output: `0.25000`; Expected: `0.25000`.
- Buttons:** Run Code, Submit, and Diff.
- Bottom Bar:** Includes a console link, a dropdown for "How to create a testcase", and system status indicators (signal strength, battery, time: 20:08, date: 16-10-2020).

```
i C Autocomplete
1 struct mygraph
2 {
3     double next;
4     double val;
5 }mygraph;
6
7 void help(int n, struct mygraph** graph, int index, int end, double *state, double current,
8 double *xx,int *col)
9 {
10     if(current<=0.00005) return;
11     if(state[index]>=current)
12     {
13         return;
14     }
15     state[index] = current;
16     if(index == end)
17     {
18         if(*xx<current) *xx = current;
19         return ;
20     }
21 }
```

Testcase Run Code Result Debugger

**Accepted** Runtime: 0 ms

Your input  
3  
[[0,1],[1,2],[0,2]]

Output 0.30000

Expected 0.30000

Console How to create a testcase ▾ Run Code Submit

1 T 1 20:09 ENG IN 16-10-2020

```
i C Autocomplete
1 struct mygraph
2 {
3     double next;
4     double val;
5 }mygraph;
6
7 void help(int n, struct mygraph** graph, int index, int end, double *state, double current,
8 double *xx,int *col)
9 {
10     if(current<=0.00005) return;
11     if(state[index]>=current)
12     {
13         return;
14     }
15     state[index] = current;
16     if(index == end)
17     {
18         if(*xx<current) *xx = current;
19         return ;
20     }
21 }
```

Testcase Run Code Result Debugger

**Accepted** Runtime: 0 ms

Your input  
3  
[[0,1]]

Output 0.00000

Expected 0.00000

Console How to create a testcase ▾ Run Code Submit

1 T 1 20:09 ENG IN 16-10-2020

**Q1.** Given a directed acyclic graph (**DAG**) of `n` nodes labelled from 0 to  $n - 1$ , find all possible paths from node `0` to node `n - 1`, and return them in any order.

The graph is given as follows: `graph[i]` is a list of all nodes you can visit from node `i` (i.e., there is a directed edge from node `i` to node `graph[i][j]`).

**CODE:**

```
#include <stdio.h>

int path[16];

int *ret[7000];

void helper(int** graph,int cur,int *idx,int graphSize,int i,int *graphColSize,int **returnColumnSizes)

{

    path[i]=cur;

    if(cur==graphSize-1)

    {

        ret[*idx]=calloc(16,sizeof(int));

        returnColumnSizes[0][*idx]=i+1;

        memcpy(ret[(*idx)++],path,sizeof(int)*(i+1));

    }

    else

        for(int j=0;j<graphColSize[cur];j++)

    {

        helper(graph,graph[cur][j],idx,graphSize,i+1,graphColSize,returnColumnSizes);

    }

}
```

```
int** allPathsSourceTarget(int** graph, int graphSize, int* graphColSize, int* returnSize, int**  
returnColumnSizes){  
  
    returnColumnSizes[0]=malloc(sizeof(int)*7000);  
  
    int idx=0;  
  
    helper(graph,0,&idx,graphSize,0,graphColSize,returnColumnSizes);  
  
    *returnSize=idx;  
  
    return ret;  
  
}
```

## OUTPUT:

```
1 int path[16];
2 int *ret[7000];
3 void helper(int** graph,int cur,int *idx,int graphSize,int i,int *graphColSize,int
4 **returnColumnSizes)
5 {
6     path[i]=cur;
7     if(cur==graphSize-1)
8     {
9         ret[*idx]=calloc(16,sizeof(int));
10        returnColumnSizes[0][*(idx)]=i+1;
11        memcpy(ret[(*idx)++],path,sizeof(int)*(i+1));
12    }
13    else
14        for(int j=0;j<graphColSize[cur];j++)
15        {
16            helper(graph,graph[cur][j],idx,graphSize,i+1,graphColSize,returnColumnSizes);
17        }
18    }
19
20 int** allPathsSourceTarget(int** graph, int graphSize, int* graphColSize, int* returnSize,
21 int** returnColumnSizes){
22     returnColumnSizes[0][0]=1;
23 }
```

Testcase Run Code Result Debugger

Accepted Runtime: 0 ms

Your input [[1,2], [3], [3], []]

Output [[0,1,3],[0,2,3]] Diff

Expected [[0,1,3],[0,2,3]]

Console ▾ How to create a testcase ▾ Run Code Submit ENG IN 20:06 16-10-2020

```
1 int path[16];
2 int *ret[7000];
3 void helper(int** graph,int cur,int *idx,int graphSize,int i,int *graphColSize,int
4 **returnColumnSizes)
5 {
6     path[i]=cur;
7     if(cur==graphSize-1)
8     {
9         ret[*idx]=calloc(16,sizeof(int));
10        returnColumnSizes[0][*(idx)]=i+1;
11        memcpy(ret[(*idx)++],path,sizeof(int)*(i+1));
12    }
13    else
14        for(int j=0;j<graphColSize[cur];j++)
15        {
16            helper(graph,graph[cur][j],idx,graphSize,i+1,graphColSize,returnColumnSizes);
17        }
18    }
19
20 int** allPathsSourceTarget(int** graph, int graphSize, int* graphColSize, int* returnSize,
21 int** returnColumnSizes){
22     returnColumnSizes[0][0]=1;
23 }
```

Testcase Run Code Result Debugger

Accepted Runtime: 0 ms

Your input [[4,3,1], [3,2,4], [3], [4], []]

Output [[0,4],[0,3,4],[0,1,3,4],[0,1,2,3,4],[0,1,4]] Diff

Expected [[0,4],[0,3,4],[0,1,3,4],[0,1,2,3,4],[0,1,4]]

Console ▾ How to create a testcase ▾ Run Code Submit ENG IN 20:07 16-10-2020

```
i C Autocomplete
1 int path[16];
2 int *ret[7000];
3 void helper(int** graph,int cur,int *idx,int graphSize,int i,int *graphColSize,int
4 **returnColumnSizes)
5 {
6     path[i]=cur;
7     if(cur==graphSize-1)
8     {
9         ret[*idx]=calloc(16,sizeof(int));
10        returnColumnSizes[0][*idx]=i+1;
11        memcpy(ret[(*idx)++],path,sizeof(int)*(i+1));
12    }
13    else
14        for(int j=0;j<graphColSize[cur];j++)
15        {
16            helper(graph,graph[cur][j],idx,graphSize,i+1,graphColSize,returnColumnSizes);
17        }
18 }
19
20 int** allPathsSourceTarget(int** graph, int graphSize, int* graphColSize, int* returnSize,
21 int** returnColumnSizes){
    returnColumnSizes[0][*idx]=i+1;
    memcpy(ret[(*idx)++],path,sizeof(int)*(i+1));
}
Testcase Run Code Result Debugger 🔒
Accepted Runtime: 0 ms
Your input [[1], []]
Output [[0,1]]
Expected [[0,1]]
Console ▾ How to create a testcase ▾ Run Code Submit
T 1 D
ENG IN 20:07 16-10-2020
```

```
i C Autocomplete
1 int path[16];
2 int *ret[7000];
3 void helper(int** graph,int cur,int *idx,int graphSize,int i,int *graphColSize,int
4 **returnColumnSizes)
5 {
6     path[i]=cur;
7     if(cur==graphSize-1)
8     {
9         ret[*idx]=calloc(16,sizeof(int));
10        returnColumnSizes[0][*idx]=i+1;
11        memcpy(ret[(*idx)++],path,sizeof(int)*(i+1));
12    }
13    else
14        for(int j=0;j<graphColSize[cur];j++)
15        {
16            helper(graph,graph[cur][j],idx,graphSize,i+1,graphColSize,returnColumnSizes);
17        }
18 }
19
20 int** allPathsSourceTarget(int** graph, int graphSize, int* graphColSize, int* returnSize,
21 int** returnColumnSizes){
    returnColumnSizes[0][*idx]=i+1;
    memcpy(ret[(*idx)++],path,sizeof(int)*(i+1));
}
Testcase Run Code Result Debugger 🔒
Accepted Runtime: 4 ms
Your input [[1,2,3], [2], [3], []]
Output [[0,1,2,3],[0,2,3],[0,3]]
Expected [[0,1,2,3],[0,2,3],[0,3]]
Console ▾ How to create a testcase ▾ Run Code Submit
T 1 D
ENG IN 20:07 16-10-2020
```