

Introduction to Computers and Programming LAB-12^{2017/01/04}

- ✧ The output must be in our sample output format.
- ✧ You can raise your hand to demo once you finish a program.
- ✧ **There is no second demo this time, we will announce the answer tomorrow.**
- ✧ TAs will update lab records every Monday after the lab hours in the link: <http://goo.gl/ZVJu2Y>

1. Header and Source Files Practice: BFS (Breadth-First Search)

You *cannot* implement any function in “main.c”. Besides, you *cannot* implement more than 5 line of code in “int main()”.

Design a program to do BFS with at least three header files (“load.h”, “bfs.h”, “queue.h”) and their corresponding source files (“load.c”, “bfs.c”, “queue.c”).

-“load.c”: deal with the inputs and make an adjacency matrix then pass the adjacency matrix (and the total number of node **n**) to the BFS function in “bfs.c”.

-“bfs.c”: do the Breadth-First Search start from node **0** with the adjacency matrix (and **n**).

-“queue.c”: a queue that you can make by yourself or you can use the one we provided.

-Format of input:

1st line: the number of node **n**.

The rest of lines: $a_0, a_1, a_2, \dots, a_k$ (which means that a_1, a_2, \dots, a_k are linked with a_0).

The input is ended by an empty line.

-Output:

The result of BFS

-Note:

The nodes are named by $0, 1, 2, \dots, n-1$.

-Example:

```
Input n:5
Input the relation:
0 1 3 4
4 3

The result of BFS:0 1 3 4 2
```

```
Input n:6
Input the relation:
0 1 2 4
2 3
4 5

The result of BFS:0 1 2 4 3 5
```

If you want to know more details, you can see the appendix.

2. Game Modifier

In this lab, TA give you a simple game names “Kill the Boss”, however because the low attack value of hero and high HP of boss, it is impossible to win this game. But you notice that each time the game run, it will create a save file “save.bin”. And with some research you found it is in binary form and has 7 integers inside. The first 3 integers are HP, Att, Def of hero, and the next 3 integers are HP, Att, Def of boss. The last integer is the valid code of this file, which can be create by calculate the result of

$$((\text{sum of all hero stat values}) * (\text{sum of all boss stat values})) \% 1234$$

Please write a simple game modifier to modify the game save file and then win the game.

Your program:

```
Please input game file path: save.bin
Hero: HP: 100, Att: 32, Def 10
Boss: HP:10000, Att: 100, Def 30

Please input new value of hero:
10000 10000 10000
Please input new value of boss:
1 1 1
```

Run game.exe for first time: **(This is not your program!!)**

```
Cannot find save.bin, create a new one.

Hero: HP: 100, Att: 32, Def 10
Boss: HP:10000, Att: 100, Def 30
Please any key to attack boss

Hero attack Boss, boss loss 2 HP
Boss attack Hero, hero loss 90 HP

Hero: HP: 10, Att: 32, Def 10
Boss: HP: 9998, Att: 100, Def 30
Please any key to attack boss
```

Run game.exe for second time: **(This is not your program!!)**

```
Load save.bin success!!

Hero: HP: 10, Att: 32, Def 10
Boss: HP: 9998, Att: 100, Def 30
Please any key to attack boss
```

If the valid code is invalid: **(This is not your program!!)**

```
save.bin is corrupt, replace with a new one.
```

If you successfully modify the file: **(This is not your program!!)**

```
Load save.bin success!!

Hero: HP:10000, Att:10000, Def 10000
Boss: HP: 1, Att: 1, Def 1
Please any key to attack boss
```

Note: If you messed up “save.bin”, just delete it and then run game.exe again.

3. Share Cookies!

Do you remember the generous Myanmar kid Mike who would like to share his cookies to his friends in Lab 7? HE IS BACK! However, this time he wants to share cookies in a different way. **He shares only one or two cookies to a friend.** For example, if he has 4 cookies, there are five ways to share them: (1,1,1,1), (2,1,1), (1,2,1), (1,1,2), (2,2). **Given the number of cookies Mike has, the program outputs the number of ways to share cookies.** Note that your program should be able to let user to input repeatedly.

Note : the order of the output doesn't matter.

Hint: Define a recursive function properly.

```
Input the number of cookies: 3
(1,1,1)
(1,2)
(2,1)
Total number of sharing is 3

Input the number of cookies: 6
(1,1,1,1,1,1)
(1,1,1,1,2)
(1,1,1,2,1)
(1,1,2,1,1)
(1,1,2,2)
(1,2,1,1,1)
(1,2,1,2)
(1,2,2,1)
(2,1,1,1,1)
(2,1,1,2)
(2,1,2,1)
(2,2,1,1)
(2,2,2)
Total number of sharing is 13

Input the number of cookies:
```

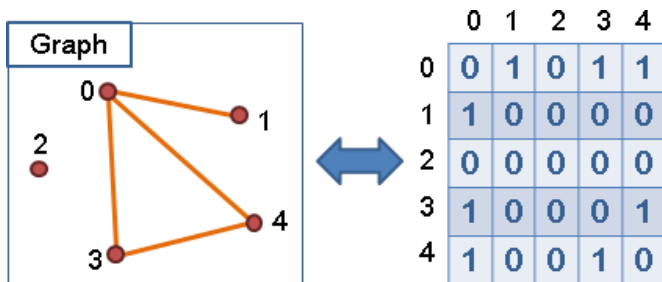
Appendix.

Breadth-first search is a strategy for searching in a graph. The BFS begins at a root node and inspects all the neighboring nodes. Then for each of those neighbor nodes in turn, it inspects their neighbor nodes which were unvisited, and so on.

In this question all the nodes are named by numbers from 0 to $n-1$, which n is an input number that represents the total number of nodes. In order to record the relation between each node, we need to build an adjacency matrix.

An adjacency matrix is a square matrix, and the elements of the matrix indicate whether pairs of vertices are adjacent(linked) or not in the graph.

Example:

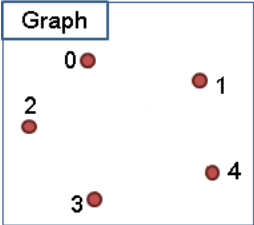
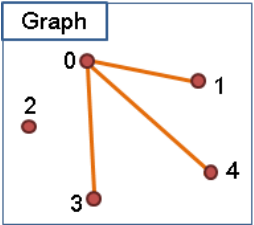


The first input is n which is the total number of node. Then the rest of the inputs are the relation of each node (ended by inputting an empty line).

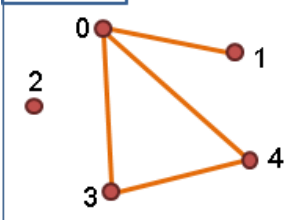
Format of the inputting relation:

$n_0 n_1 n_2 \dots n_k$: which means that n_1, n_2, \dots, n_k are linked with n_0 . (The link is bidirectional)

Example:

Original:																																					
<div><div>Graph</div></div>	<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>3</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>		0	1	2	3	4	0	0	0	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	3	0	0	0	0	0	4	0	0	0	0	0
	0	1	2	3	4																																
0	0	0	0	0	0																																
1	0	0	0	0	0																																
2	0	0	0	0	0																																
3	0	0	0	0	0																																
4	0	0	0	0	0																																
After input: 0 1 3 4																																					
<div><div>Graph</div></div>	<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>3</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>4</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>		0	1	2	3	4	0	0	1	0	1	1	1	1	0	0	0	0	2	0	0	0	0	0	3	1	0	0	0	0	4	1	0	0	0	0
	0	1	2	3	4																																
0	0	1	0	1	1																																
1	1	0	0	0	0																																
2	0	0	0	0	0																																
3	1	0	0	0	0																																
4	1	0	0	0	0																																
After input: 4 3																																					

Graph



```
graph LR; 0 --- 1; 0 --- 3; 1 --- 3; 0 --- 4; 3 --- 4; 2;
```

	0	1	2	3	4
0	0	1	0	1	1
1	1	0	0	0	0
2	0	0	0	0	0
3	1	0	0	0	1
4	1	0	0	1	0

We also provide bfs1.gif and bfs2.gif that represent the process of BFS.

The result of bfs1.gif is: A B C D E F G

The result of bfs2.gif is: 1 2 3 5 4 6