CS206A Data Structure

HW6

Student ID: 20160253

name: 박예기 (Park ye gi)

Problem1(a)

The program for Problem1(a) is Problem1(a).py in Project Folder.

The program gets input so that it makes an adjacency matrix for the graph. First it checks the graph is connected or not. (This checking is done by using the code for Problem2.) Then it makes a list, which is a collection of lists that contains connected nodes for each node.

It uses two lists, stack and stack2. Stack is used for depth first search. When an element is added to stack, a tuple is added to stack2. That tuple consists of the last element of stack and the element that is added to stack.

With the stack2, the program makes adjacency matrix of depth first spanning tree.

Finally, it prints out the adjacency matrix of depth first spanning tree.

The below example is an example on the lecture note 5-3.

```
Enter the number of nodes: 8
The name of node is 0,1,2,3,... in order.
Enter the row of adjacency matrix(0 or 1): 01100000
Enter the row of adjacency matrix(0 or 1): 10011000
Enter the row of adjacency matrix(0 or 1): 10000110
Enter the row of adjacency matrix(0 or 1): 01000001
Enter the row of adjacency matrix(0 or 1): 01000001
Enter the row of adjacency matrix(0 or 1): 00100001
Enter the row of adjacency matrix(0 or 1): 00100001
Enter the row of adjacency matrix(0 or 1): 00011110
The adjacency matrix for depth first spanning tree is
[0, 1, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 1, 1, 0, 0]
```

Problem1(b)

The program for Problem1(b) is Problem1(b).py in Project Folder.

The program gets input so that it makes an adjacency matrix for the graph. First it checks the graph is connected or not. (This checking is done by using the code for Problem2.) Then it makes a list, which is a collection of lists

that contains connected nodes for each node.

It uses two lists, queue and queue2. Queue is used for breadth first search. When an element is added to queue, a tuple is added to queue2. That tuple consists of the first element of queue and the element that is added to queue.

With the queue2, the program makes adjacency matrix of breadth first spanning tree.

Finally, it prints out the adjacency matrix of breadth first spanning tree.

The below example is an example on the lecture note 5-3.

```
Enter the number of nodes: 8
The name of node is 0,1,2,3,... in order.
Enter the row of adjacency matrix(0 or 1): 01100000
Enter the row of adjacency matrix(0 or 1): 10011000
Enter the row of adjacency matrix(0 or 1): 10000110
Enter the row of adjacency matrix(0 or 1): 01000001
Enter the row of adjacency matrix(0 or 1): 01000001
Enter the row of adjacency matrix(0 or 1): 00100001
Enter the row of adjacency matrix(0 or 1): 00100001
Enter the row of adjacency matrix(0 or 1): 00011110
The adjacency matrix for breadth first spanning tree is
[0, 1, 1, 0, 0, 0, 0, 0]
[1, 0, 0, 1, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 1, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
```

Problem2

The program for Problem2 is Problem2.py in Project Folder.

The program gets input so that it makes an adjacency matrix for the graph. Then it makes a list named ‘a’, which is a collection of lists that contains the connected nodes of each node and that node. Next, it makes another list named ‘connect’, which is a collection of sets that contains connected components of the graph.

Finally, the program prints out the list ‘connect’.

```
Enter the number of nodes: 5
The name of node is 0,1,2,3,... in order.
Enter the row of adjacency matrix(0 or 1): 01100
Enter the row of adjacency matrix(0 or 1): 10000
Enter the row of adjacency matrix(0 or 1): 10000
Enter the row of adjacency matrix(0 or 1): 00001
Enter the row of adjacency matrix(0 or 1): 00010

All connected components of the graph is
[{1, 2, 3}, {4, 5}]
```

Problem3

i	j	A[i,j]	i	k	j	A[i,j]	i	k	j	A[i,j]	i	k	j	A[i,j]	i	k	j	A[i,j]	i	k	j	A[i,j]
0	0	**F**	0	0	0	**F**	0	1	0	**F**	0	2	0	**F**	0	3	0	**F**	0	4	0	**F**
0	1	**T**	0	0	1	**T**	0	1	1	**T**	0	2	1	**T**	0	3	1	**T**	0	4	1	**T**
0	2	**F**	0	0	2	**F**	0	1	2	**T**	0	2	2	**T**	0	3	2	**T**	0	4	2	**T**
0	3	**F**	0	0	3	**F**	0	1	3	**F**	0	2	3	**T**	0	3	3	**T**	0	4	3	**T**
0	4	**F**	0	0	4	**F**	0	1	4	**F**	0	2	4	**F**	0	3	4	**T**	0	4	4	**T**
1	0	**F**	1	0	0	**F**	1	1	0	**F**	1	2	0	**F**	1	3	0	**F**	1	4	0	**F**
1	1	**F**	1	0	1	**F**	1	1	1	**F**	1	2	1	**F**	1	3	1	**F**	1	4	1	**F**
1	2	**T**	1	0	2	**T**	1	1	2	**T**	1	2	2	**T**	1	3	2	**T**	1	4	2	**T**
1	3	**F**	1	0	3	**F**	1	1	3	**F**	1	2	3	**T**	1	3	3	**T**	1	4	3	**T**
1	4	**F**	1	0	4	**F**	1	1	4	**F**	1	2	4	**F**	1	3	4	**T**	1	4	4	**T**
2	0	**F**	2	0	0	**F**	2	1	0	**F**	2	2	0	**F**	2	3	0	**F**	2	4	0	**F**
2	1	**F**	2	0	1	**F**	2	1	1	**F**	2	2	1	**F**	2	3	1	**F**	2	4	1	**F**
2	2	**F**	2	0	2	**F**	2	1	2	**F**	2	2	2	**F**	2	3	2	**F**	2	4	2	**T**
2	3	**T**	2	0	3	**T**	2	1	3	**T**	2	2	3	**T**	2	3	3	**T**	2	4	3	**T**
2	4	**F**	2	0	4	**F**	2	1	4	**F**	2	2	4	**F**	2	3	4	**T**	2	4	4	**T**
3	0	**F**	3	0	0	**F**	3	1	0	**F**	3	2	0	**F**	3	3	0	**F**	3	4	0	**F**
3	1	**F**	3	0	1	**F**	3	1	1	**F**	3	2	1	**F**	3	3	1	**F**	3	4	1	**F**
3	2	**F**	3	0	2	**F**	3	1	2	**F**	3	2	2	**F**	3	3	2	**F**	3	4	2	**T**
3	3	**F**	3	0	3	**F**	3	1	3	**F**	3	2	3	**F**	3	3	3	**F**	3	4	3	**T**
3	4	**T**	3	0	4	**T**	3	1	4	**T**	3	2	4	**T**	3	3	4	**T**	3	4	4	**T**
4	0	**F**	4	0	0	**F**	4	1	0	**F**	4	2	0	**F**	4	3	0	**F**	4	4	0	**F**
4	1	**F**	4	0	1	**F**	4	1	1	**F**	4	2	1	**F**	4	3	1	**F**	4	4	1	**F**
4	2	**T**	4	0	2	**T**	4	1	2	**T**	4	2	2	**T**	4	3	2	**T**	4	4	2	**T**
4	3	**F**	4	0	3	**F**	4	1	3	**F**	4	2	3	**T**	4	3	3	**T**	4	4	3	**T**
4	4	**F**	4	0	4	**F**	4	1	4	**F**	4	2	4	**F**	4	3	4	**T**	4	4	4	**T**

Problem4

The list will look like

embed, other, refer, class, leaks, every, array

after the first iteration of outer loop.

Problem5

The array will look like

15 26 35 50 53 57 2 11 14 21 22 25

right before the last merge.

Problem6

After the first level of recursion,

25 2 11 26 14 15 22 21 35 50 53 57

After the second level of recursion,

22 2 11 21 14 15 25 26 35 50 53 57

After the third level of recursion,

15 2 11 21 14 22 25 26 35 50 53 57

After the fourth level of recursion,

14 2 11 15 21 22 25 26 35 50 53 57

After the fifth level of recursion,

11 2 14 15 21 22 25 26 35 50 53 57

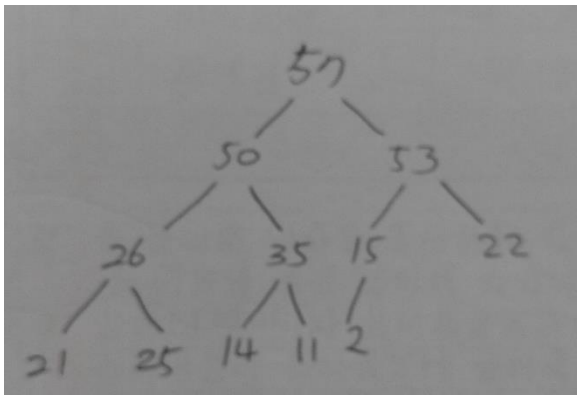
After the sixth level of recursion,

2 11 14 15 21 22 25 26 35 50 53 57

This is the final result of Quicksort.

Problem7

The heap will look like the below picture.



The list will be **57 50 53 26 35 15 22 21 25 14 11 2**

Problem8

After one iteration (one iteration means putting max on the end of heap and then re-heapify),

53 50 22 26 35 15 2 21 25 14 11 57

After two iterations,

50 35 22 26 14 15 2 21 25 11 53 57

After three iterations,

35 26 22 25 14 15 2 21 11 50 53 57

After four iterations,

26 25 22 21 14 15 2 11 35 50 53 57