Jian Li

Problem 1

Explore order:

1 4 2 3 10 6 8 7 5 9

Value (pre, post):

vertex 1 :(1,20)

vertex 2 :(3,18)

vertex 3 :(4,17)

vertex 4 :(2,19)

vertex 5 :(9,12)

vertex 6 :(6,15)

vertex 7 :(8,13)

vertex 8 :(7,14)

vertex 9 :(10,11)

vertex 10 :(5,16)

edge classification:

tree :

[(1, 4), (2, 3), (3, 10), (4, 2), (5, 9), (6, 8), (7, 5), (8, 7), (10, 6)]

forward:

[(1, 5), (1, 6), (2, 7), (2, 9), (4, 6), (6, 9), (8, 9), (10, 8)]

back :

[(2, 4), (3, 2), (4, 1), (5, 1), (5, 7), (6, 1), (6, 4), (6, 10), (7, 2), (7, 8), (8, 6), (8, 10), (9, 2), (9, 5), (9, 6), (9, 8), (10, 3)]

cross

[]

Step by Step and Call Depth

now exploring 1

depth 1

visited[1] sets True

pre[1] sets 1

clock increase from 1 to 2

now exploring 4

depth 2

visited[4] sets True

pre[4] sets 2

clock increase from 2 to 3

4 is already explored

now exploring 2

depth 3

visited[2] sets True

pre[2] sets 3

clock increase from 3 to 4

now exploring 3

depth 4

visited[3] sets True

pre[3] sets 4

clock increase from 4 to 5

3 is already explored

now exploring 10

depth 5

visited[10] sets True

pre[10] sets 5

clock increase from 5 to 6

10 is already explored

now exploring 6

depth 6

visited[6] sets True

pre[6] sets 6

clock increase from 6 to 7

6 is already explored

6 is already explored

now exploring 8

depth 7

visited[8] sets True

pre[8] sets 7

clock increase from 7 to 8

8 is already explored

now exploring 7

depth 8

visited[7] sets True

pre[7] sets 8

clock increase from 8 to 9

7 is already explored

now exploring 5

depth 9

visited[5] sets True

pre[5] sets 9

clock increase from 9 to 10

5 is already explored

5 is already explored

now exploring 9

depth 10

visited[9] sets True

pre[9] sets 10

clock increase from 10 to 11

9 is already explored

9 is already explored

9 is already explored

9 is already explored

post[9] sets 11

clock increase from 11 to 12

post[5] sets 12

clock increase from 12 to 13

7 is already explored

post[7] sets 13

clock increase from 13 to 14

8 is already explored

8 is already explored

post[8] sets 14

clock increase from 14 to 15

6 is already explored

6 is already explored

post[6] sets 15

clock increase from 15 to 16

10 is already explored

post[10] sets 16

clock increase from 16 to 17

post[3] sets 17

clock increase from 17 to 18

2 is already explored

2 is already explored

2 is already explored

post[2] sets 18

clock increase from 18 to 19

4 is already explored

post[4] sets 19

clock increase from 19 to 20

1 is already explored

1 is already explored

post[1] sets 20

Problem 2

Explore order:

1 3 4 10 8 2 5 6 7 9

Topologically sort:

9 7 6 5 2 1 10 8 3 4 (the reverse sort of post number)

Value (pre, post):

vertex 1 :(1,10)

vertex 2 :(11,12)

vertex 3 :(2,5)

vertex 4 :(3,4)

vertex 5 :(13,14)

vertex 6 :(15,16)

vertex 7 :(17,18)

vertex 8 :(7,8)

vertex 9 :(19,20)

vertex 10 :(6,9)

edge classification:

tree :

[(1, 3), (1, 10), (3, 4), (10, 8)]

forward:

[(1,4)]

back:

[]

cross:

[(2, 8), (5, 2), (5, 10), (6, 10), (7, 3), (7, 4), (7, 10), (8, 3), (9, 2), (9, 3), (9, 4), (9, 5), (9, 8), (10, 3), (10, 4)]

Determine it is acyclic:

According to text book, there are no back edges in this graph, so it is acyclic.

Step by Step and Call Depth

now exploring 1

depth 1

visited[1] sets True

pre[1] sets 1

clock increase from 1 to 2

now exploring 3

depth 2

visited[3] sets True

pre[3] sets 2

clock increase from 2 to 3

now exploring 4

depth 3

visited[4] sets True

pre[4] sets 3

clock increase from 3 to 4

post[4] sets 4

clock increase from 4 to 5

post[3] sets 5

clock increase from 5 to 6

1 is already explored

now exploring 10

depth 2

visited[10] sets True

pre[10] sets 6

clock increase from 6 to 7

10 is already explored

10 is already explored

now exploring 8

depth 3

visited[8] sets True

pre[8] sets 7

clock increase from 7 to 8

8 is already explored

post[8] sets 8

clock increase from 8 to 9

post[10] sets 9

clock increase from 9 to 10

post[1] sets 10

clock increase from 10 to 11

now exploring 2

depth 1

visited[2] sets True

pre[2] sets 11

clock increase from 11 to 12

2 is already explored

post[2] sets 12

clock increase from 12 to 13

now exploring 5

depth 1

visited[5] sets True

pre[5] sets 13

clock increase from 13 to 14

5 is already explored

5 is already explored

post[5] sets 14

clock increase from 14 to 15

now exploring 6

depth 1

visited[6] sets True

pre[6] sets 15

clock increase from 15 to 16

6 is already explored

post[6] sets 16

clock increase from 16 to 17

now exploring 7

depth 1

visited[7] sets True

pre[7] sets 17

clock increase from 17 to 18

7 is already explored

7 is already explored

7 is already explored

post[7] sets 18

clock increase from 18 to 19

now exploring 9

depth 1

visited[9] sets True

pre[9] sets 19

clock increase from 19 to 20

9 is already explored

9 is already explored

9 is already explored

9 is already explored

9 is already explored

post[9] sets 20

Problem 3

a. First, Prove a is empty, the algorithm terminate.
If a is empty, will move b one by one until b empty,
so algorithm return C, terminated.
The same goes with b is empty, the algorithm terminate.
Second, assume length of $a \le n_1$, length of $b \le n_2$, the algorithm terminate
Third, we need prove length of $a = n_1+1$, length of $b = n_2+1$, the algorithm terminate
① if the first element of a and b be moved alternately during the algorithm, there must be a memert when both a and b are moved at least one element. Thus, length of $a \le n_1$, length of $b \le n_2$, according to Second assumption. Proved.
② if the first element of a always less than b, the algorithm will move a one by one until a is empty, according to first prove, algorithm will terminate.
③ if the first element of b always less than a, same reason with ②.
In conclusion, the merge algorithm will terminate.

b. Notation: A, B is the set of elements in array a, b, C is the set of elements in output.
Assume, exist one element X in A that C doesn't contain X, which means the algorithm does not move X to C.
However, in execution of the algorithm, when the length of a is not empty (at least X exist). ① If b is empty, A will be moved one by one, so eventually C will contain X. Contradict with assumption.
② if b is not empty, if exist $b_i > X$, X will be moved to C before b_i. So C will contain X. if all $b < X$, it will be the same situation in ①.
In conclusion, doesn't exist one element in A that not in C. We can prove this in B same way. So The output array contains union of elements of input array.

c. Same Notation with b.

First prove, if randomly picked two elements from C, they are in correct sorted order. If this is proved, we can have $C_1 \le C_2$, $C_2 \le C_3$, $\cdots$ $C_{n-2} \le C_{n-1}$, $C_{n-1} \le C_n$.

So $\because$ $C_1 \le C_2 \le C_3 \le \cdots \le C_{n-1} \le C_n$. The problem is proved.

So assume randomly pick C_i, C_j from C. $(1 \le i < j \le n.)$

① if C_i, C_j both from A or B. as A or B is sorted, if $i<j$. $C_i \le C_j$. they are in correct order.

② if One of C_i, C_j is from A, the other from B.

because $i<j$ $\therefore$ there exist a moment C_i just moved to C while C_j still in array B. ① if C_j is the first element of B now, $C_i < C_j$ (according to algorithm) ② if C_j is not the first of B, must exist C_k $(k<j)$ and C_k is the first element of B. So $C_i < C_k$, while $C_k < C_j$ as B is sorted, thus $C_i < C_j$. they are in correct order.

In conclusion, we randomly pick two elements from C, they must be in correct order.

So if intput are in order, the output is also in order.

(~~I only proved the situation where A and B are in an increasing~~ Order, if they are reversed, the conclusion cannot be proved).

Problem 4

General idea:

Doing a DFS using stack, push the next node and the explore history to get to this node(the path to this node) together in the stack. Explore the graph, if the present pop node has a edge to the starting node, the present explore history add the starting point is the cycle. If the stack is empty, return false.

Prove it is liner time:

The algorithm is just doing DFS, using extra memory space to keep track of explore history to each node, so the time complexity is still O(E).

Algorithm:

```
Findcycle(graph,start):
    Stack.push( (start, None) );
    While true:
    {
        If stack is empty:
            Return false;
        now_node, now_path = stack.pop
        now_path append now_node
        for node in next_nodes:
            If node is start:
                Return (now_path append start)
            else:
                Stack.push( next_node, now_path)

    }
```

My algorithm return false with start point is 1.

However, as professor required, we need to provide Step by Step and Call Depth. I can provide the explore order using a stack, but cannot provide other information. So I conduct a DFS with recursion same with the first and second problem on the graph.

I think my algorithm can work according to the problem, but cannot provide some information.

In the perspective of finishing homework itself, I can modify my solution to this:

```
1.Perform DFS on graph
2.Get pre and post number
3.Classify the edges
4.If back edges not exisit:
5.    return false
6.    else if start node in them:
7.        get another node of the edge
8.        Doing DFS using stack:
9.            If next_node == another node:
10.               Return now_path append start
```

time complexity for step 1 and 2 is O(E)
time complexity for step 3 is O(E)
time complexity for step 4-6 is O(1)
time complexity for step 8-10 is O(E)
so the total is O(E+E+1+E) still O(E)

Explore order:
1 3 4 10 8 2 5 6 7 9

Value (pre, post):
vertex 1 :(1,16)
vertex 2 :(2,13)
vertex 3 :(4,7)
vertex 4 :(14,15)
vertex 5 :(17,20)
vertex 6 :(9,10)
vertex 7 :(3,12)
vertex 8 :(5,6)
vertex 9 :(18,19)

vertex 10 :(8,11)

edge classification:

forward is :

[(1, 2), (1, 4), (2, 7), (2, 8), (3, 8), (5, 9), (7, 3), (7, 10), (10, 6)]

back is :

[(6, 2), (8, 7)]

cross is

[(4, 7), (5, 10), (6, 8), (9, 1), (10, 3)]

Step by Step and Call Depth

now exploring 1

depth 1

visited[1] sets True

pre[1] sets 1

clock increase from 1 to 2

now exploring 2

depth 2

visited[2] sets True

pre[2] sets 2

clock increase from 2 to 3

now exploring 7

depth 3

visited[7] sets True

pre[7] sets 3

clock increase from 3 to 4

now exploring 3

depth 4

visited[3] sets True

pre[3] sets 4

clock increase from 4 to 5

now exploring 8

depth 5

```
visited[8] sets True
pre[8] sets 5
clock increase from 5 to 6
8 is already explored
post[8] sets 6
clock increase from 6 to 7
post[3] sets 7
clock increase from 7 to 8
now exploring 10
depth 4
visited[10] sets True
pre[10] sets 8
clock increase from 8 to 9
10 is already explored
now exploring 6
depth 5
visited[6] sets True
pre[6] sets 9
clock increase from 9 to 10
6 is already explored
6 is already explored
post[6] sets 10
clock increase from 10 to 11
post[10] sets 11
clock increase from 11 to 12
post[7] sets 12
clock increase from 12 to 13
2 is already explored
post[2] sets 13
clock increase from 13 to 14
now exploring 4
depth 2
visited[4] sets True
```

```
pre[4] sets 14
clock increase from 14 to 15
4 is already explored
post[4] sets 15
clock increase from 15 to 16
post[1] sets 16
clock increase from 16 to 17
now exploring 5
depth 1
visited[5] sets True
pre[5] sets 17
clock increase from 17 to 18
now exploring 9
depth 2
visited[9] sets True
pre[9] sets 18
clock increase from 18 to 19
9 is already explored
post[9] sets 19
clock increase from 19 to 20
5 is already explored
post[5] sets 20
```