**Problem 1**

**General Idea:**

As all nodes in a strongly connected component (SCC for short) have access to each other. My idea is take the problem as a graph, the set of tasks as nodes, and their direct dependencies as directed edge. Use DFS and **tarjan algorithm[1]** to find strongly connected components in this graph. If there is only one element in SCC, no extra effect C is needed, however if there are at least two elements in SCC, the extra effect will be **C\*k\*(k-1)/2** , where k is the number of elements in this SCC. Sum up all extra effect C needed for all SCC of this graph and add n\*D, where n is the number of task in this problem is the final result.

**Algorithm**

**Algorithm DFS[2]**

Input: Graph G where each vertex v has field v.visited initially set to false

and methods v.previsit() and v.postvisit(), and where each edge e has a

method e.traverse().

Procedure:

For each vertex v of G:

If not v.visited: explore(G,v)

**Algorithm explore[3]**

Input: Graph G as in DFS, vertex v of G

Output: u.visited is true for all nodes reachable from v

Procedure:

Set v.visited to true

v.previsit()

For each edge (v,u) in G:

if not u.visited: explore(G,u)

e.traverse()

v.postvisit()

**Algorithm FindSCC[4]**

Input: Graph G as in DFS and, in addition, each vertex has fields

v.isRemoved, v.lowlink initially set to false and undefined.

Output: The set of SCCs of G

Procedure:

Set stack S to empty

DFS(G)

previsit():

Set v.lowlink to the previsit time of v

Push v on S

traverse():

if u was visited: set v.lowlink to the minimum of v.lowlink and u.lowlink

else if u was not removed from S: set v.lowlink to the minimum of v.lowlink and the previsit time of u

postvisit():

if v is a root of an SCC (i.e., v.lowlink is the same as the previsit time of v):

Let C be a new empty set of vertices

loop:

Pop a vertex w from S and add it to C

Set w.isRemoved to true

if w is the same as v then break out of the loop

Add C to the set of SCCs.

**Prove algorithm has linear time[5]**

The Tarjan procedure is called once for each node; the forall statement considers each edge at most twice. The algorithm's running time is therefore linear in the number of edges and nodes in G, i.e. O(|V|+|E|)

**Reference**

[1][2][3][4] https://kenb.ccs.neu.edu:5800/tarjan.txt

[5] Tarjan’s strongly connected components algorithm <http://en.wikipedia.org/wiki/Tarjan's_strongly_connected_components_algorithm>

**Problem 2**

From node 1 to 9

parent list is [8, 5, 9, 5, 5, 5, 5, 5, 5]

rank list is [0, 0, 0, 1, 2, 0, 0, 1, 1]

Step by step

now union 5 and 7

5 parent is 5

7 parent is 7

union second to first

5 rank is now 1

now union 4 and 2

4 parent is 4

2 parent is 2

union second to first

4 rank is now 1

now union 4 and 6

4 parent is 4

6 parent is 6

union second to first

now union 6 and 2

6 parent is 4

2 parent is 4

same root

now union 8 and 1

8 parent is 8

1 parent is 1

union second to first

8 rank is now 1

now union 7 and 1

7 parent is 5

1 parent is 8

union second to first

5 rank is now 2

now union 2 and 8

2 parent is 4

8 parent is 5

union first to second

now union 7 and 6

7 parent is 5

6 parent is 5

same root

now union 9 and 3

9 parent is 9

3 parent is 3

union second to first

9 rank is now 1

now union 5 and 2

5 parent is 5

2 parent is 5

same root

now union 3 and 8

3 parent is 9

8 parent is 5

union first to second

**Problem 3**

**Problem 4**

Explore order:

9 1 2 4 6 10 7 5 11

Step by step:

*None equals Infinity below*

the present queue is

[9]

now distance from source node is

[None, None, None, None, None, None, None, None, 0, None, None]

now exploring 9

the present queue is

[1, 2, 4, 6, 10]

now distance from source node is

[1, 1, None, 1, None, 1, None, None, 0, 1, None]

now exploring 1

the present queue is

[2, 4, 6, 10, 7]

now distance from source node is

[1, 1, None, 1, None, 1, 2, None, 0, 1, None]

now exploring 2

the present queue is

[4, 6, 10, 7, 5]

now distance from source node is

[1, 1, None, 1, 2, 1, 2, None, 0, 1, None]

now exploring 4

the present queue is

[6, 10, 7, 5]

now distance from source node is

[1, 1, None, 1, 2, 1, 2, None, 0, 1, None]

now exploring 6

the present queue is

[10, 7, 5]

now distance from source node is

[1, 1, None, 1, 2, 1, 2, None, 0, 1, None]

now exploring 10

the present queue is

[7, 5]

now distance from source node is

[1, 1, None, 1, 2, 1, 2, None, 0, 1, None]

now exploring 7

the present queue is

[5]

now distance from source node is

[1, 1, None, 1, 2, 1, 2, None, 0, 1, None]

now exploring 5

the present queue is

[11]

now distance from source node is

[1, 1, None, 1, 2, 1, 2, None, 0, 1, 3]

now exploring 11

**Problem 5**

SCC:

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

Step by step:

current depth is :1

1 DFN is 1

1 LOW is 1

my stack is [1]

current depth is :2

5 DFN is 2

5 LOW is 2

my stack is [1, 5]

current depth is :3

4 DFN is 3

4 LOW is 3

my stack is [1, 5, 4]

current depth is :4

2 DFN is 4

2 LOW is 4

my stack is [1, 5, 4, 2]

a new SCC is

2

4 LOW now is 3

a new SCC is

4

5 LOW now is 2

current depth is :3

7 DFN is 5

7 LOW is 5

my stack is [1, 5, 7]

**7 LOW now is 2 (has back egde to 5)**

5 LOW now is 2

current depth is :3

8 DFN is 6

8 LOW is 6

my stack is [1, 5, 7, 8]

8 LOW noww is 5

current depth is :4

10 DFN is 7

10 LOW is 7

my stack is [1, 5, 7, 8, 10]

**10 LOW now is 2(back edge to 5)**

**8 LOW now is 2 (back edge to 7)**

5 LOW now is 2

5 LOW now is 2

a new SCC is

10

8

7

5

1 LOW now is 1

current depth is :2

6 DFN is 8

6 LOW is 8

my stack is [1, 6]

a new SCC is

6

1 LOW now is 1

a new SCC is

1

current depth is :1

3 DFN is 9

3 LOW is 9

my stack is [3]

a new SCC is

3

current depth is :1

9 DFN is 10

9 LOW is 10

my stack is [9]

a new SCC is

9