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, 4, 9, 9, 4, 4, 4, 4, 9]

Step by step

now union 5 and 7

5 parent is 5

7 parent is 7

union 7 to 5

now union 4 and 2

4 parent is 4

2 parent is 2

union 2 to 4

now union 4 and 6

4 parent is 4

6 parent is 6

union 6 to 4

now union 6 and 2

6 parent is 4

2 parent is 4

union 2 to 6

now union 8 and 1

8 parent is 8

1 parent is 1

union 1 to 8

now union 7 and 1

7 parent is 5

1 parent is 8

union 1 to 7

now union 2 and 8

2 parent is 4

8 parent is 5

union 8 to 2

now union 7 and 6

7 parent is 4

6 parent is 4

union 6 to 7

now union 9 and 3

9 parent is 9

3 parent is 3

union 3 to 9

now union 5 and 2

5 parent is 4

2 parent is 4

union 2 to 5

now union 3 and 8

3 parent is 9

8 parent is 4

union 8 to 3

Problem 3

parent list is [1, 1, 1, 4, 4, 1, 4, 8, 1, 1]

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

(from 1 to 10)

Step by step:

Depth Operation

1 visited[1] sets True

pre[1] sets 1

now union 3 1

1 parent is 1

3 parent is 3

union second to first

1 rank is now 1

now exploring 3

2 visited[3] sets True

pre[3] sets 2

now union 9 3

3 parent is 1

9 parent is 9

union second to first

now exploring 9

3 visited[9] sets True

pre[9] sets 3

post[9] sets 4

post[3] sets 5

now union 6 1

1 parent is 1

6 parent is 6

union second to first

now exploring 6

2 visited[6] sets True

pre[6] sets 6

now union 2 6

6 parent is 1

2 parent is 2

union second to first

now exploring 2

3 visited[2] sets True

pre[2] sets 7

2 is already explored

2 is already explored

post[2] sets 8

6 is already explored

now union 10 6

6 parent is 1

10 parent is 10

union second to first

now exploring 10

3 visited[10] sets True
pre[10] sets 9
10 is already explored
10 is already explored
post[10] sets 10
post[6] sets 11
post[1] sets 12
1 visited[4] sets True
pre[4] sets 13
4 is already explored
4 is already explored
4 is already explored
4 is already explored
now union 7 4
4 parent is 4
7 parent is 7
union second to first
4 rank is now 1
now exploring 7
2 visited[7] sets True
pre[7] sets 14
now union 5 7
7 parent is 4
5 parent is 5
union second to first
now exploring 5
3 visited[5] sets True
pre[5] sets 15
5 is already explored
5 is already explored
5 is already explored
post[5] sets 16
7 is already explored

7 is already explored

7 is already explored

post[7] sets 17

4 is already explored

post[4] sets 18

1 visited[8] sets True

pre[8] sets 19

8 is already explored

8 is already explored

8 is already explored

8 is already explored

8 is already explored

post[8] sets 20

Problem 4

Explore order:

9 1 2 4 6 10 7 5 11

Step by step:

(None represents 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

Algorithm offered by Professor using lowlink.

1 visted[1] sets true

pre[1] sets 1

LOW[1] sets 1

push 1 to stack

isRemoved[1] set False

my stack is [1]

2 visted[5] sets true

pre[5] sets 2

LOW[5] sets 2

push 5 to stack

isRemoved[5] set False

my stack is [1, 5]

3 visted[4] sets true

pre[4] sets 3

LOW[4] sets 3

push 4 to stack

isRemoved[4] set False

my stack is [1, 5, 4]

4 visted[2] sets true

pre[2] sets 4

LOW[2] sets 4

push 2 to stack

isRemoved[2] set False

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

stack pop

isRemoved[2] sets True

---a new SCC is [2] **(happens because pre[2] = low[2])**

now traversing edge(4,2)

4 LOW now is 3

stack pop

isRemoved[4] sets True

---a new SCC is [4] **(happens because pre[4] = low[4])**

now traversing edge(5,4)

5 LOW now is 2

3 visted[7] sets true

pre[7] sets 5

LOW[7] sets 5

push 7 to stack

isRemoved[7] set False

my stack is [1, 5, 7]

7 LOW now is 2

now traversing edge(5,7)

5 LOW now is 2

3 visted[8] sets true

pre[8] sets 6

LOW[8] sets 6

push 8 to stack

isRemoved[8] set False

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

8 LOW now is 5

4 visted[10] sets true

pre[10] sets 7

LOW[10] sets 7

push 10 to stack

isRemoved[10] set False

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

10 LOW now is 2

now traversing edge(8,10)

8 LOW now is 2

now traversing edge(5,8)

5 LOW now is 2

5 LOW now is 2

stack pop

isRemoved[10] sets True

stack pop

isRemoved[8] sets True

stack pop

isRemoved[7] sets True

stack pop

isRemoved[5] sets True

---a new SCC is [10, 8, 7, 5] **(happens because 10 has edge to 5,8 has edge to 7 , 7 has edge to 5, so there are low all equals to low[5] = 2)**

now traversing edge(1,5)

1 LOW now is 1

2 visted[6] sets true

pre[6] sets 8

LOW[6] sets 8

push 6 to stack

isRemoved[6] set False

my stack is [1, 6]

stack pop

isRemoved[6] sets True

---a new SCC is [6] **(happens because pre[6] = low[6])**

now traversing edge(1,6)

1 LOW now is 1

stack pop

isRemoved[1] sets True

---a new SCC is [1] **(happens because pre[1] = low[1])**

1 visted[3] sets true

pre[3] sets 9

LOW[3] sets 9

push 3 to stack

isRemoved[3] set False

my stack is [3]

stack pop

isRemoved[3] sets True

---a new SCC is [3] **(happens because pre[3] = low[3])**

1 visted[9] sets true

pre[9] sets 10

LOW[9] sets 10

push 9 to stack

isRemoved[9] set False

my stack is [9]

stack pop

isRemoved[9] sets True

---a new SCC is [9] **(happens because pre[9] = low[9])**