Somesh Pratap Singh
19110206

23/11/21

Page No.
Date

DSAI Theory Endsem

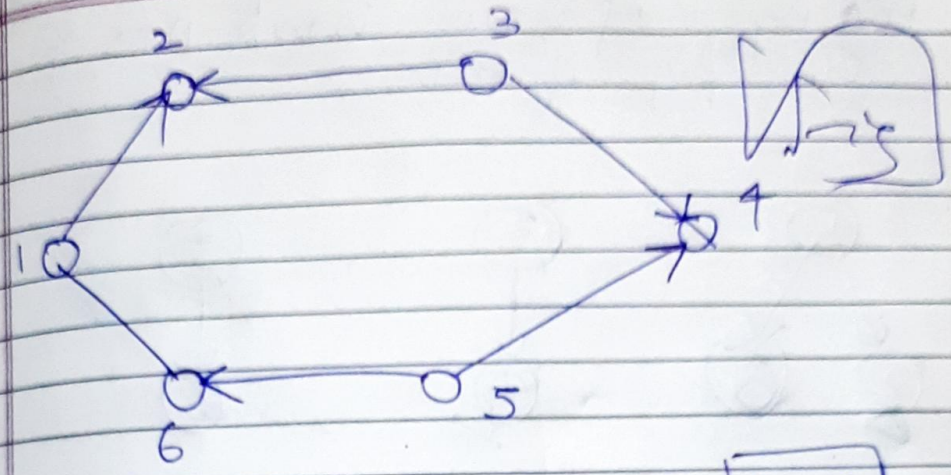
(1)

3

(a)

```
Finding Imp. vertex {
  for every vertex v ∈ V {
      Do BFS (v)
      if (the BFS tree of v
        contains all other
        vertices) {
          graph has Important
            vertex v
          return v;
      }
      [illegible]
  }
  graph has no Important vertex.
}
```

3



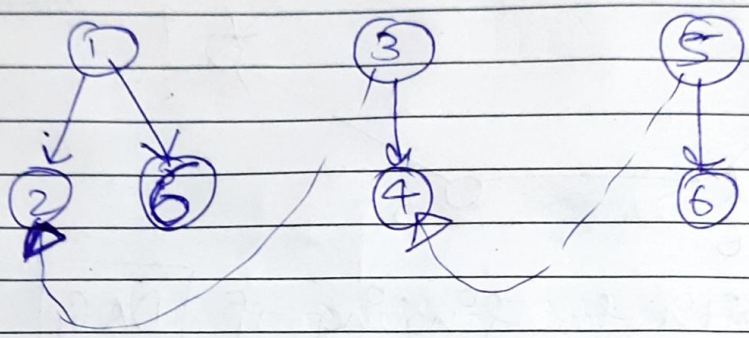
Algorith for finding if DAG has an imp. vertex or not.

Step1. Make the DFS tree of the DAG

Step2. If there are no components in the DFS tree i.e. it is a simply connected acyclic graph (tree) then the DAG has an imp. vertex.

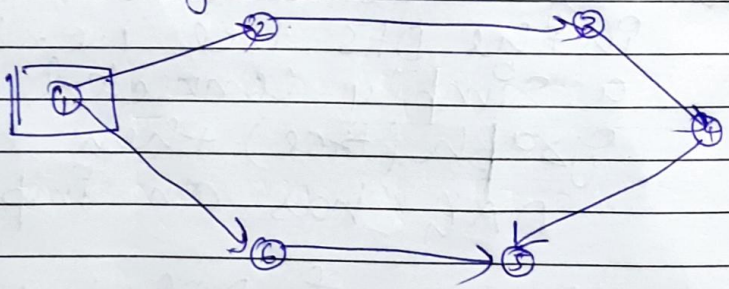
Step3: However if DAG has more than one components then for an important vertex to exist
(i) there should be a back edge b/w components
(ii) and the backedge b/w within a component should end at a vertex that has a indeed has a backedge to prev component.

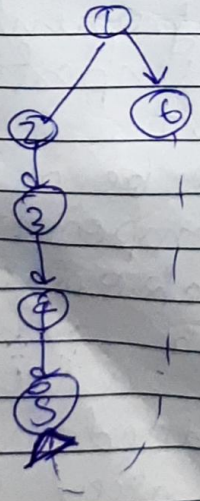
For eg. If you make DFS of the fig given.



This graph does not have an imp. vertex.

However if





this has an imp. vertex.

Note: Running time = O(m+n)

of DFS and even if we check all non tree edges

~~hasImpverter (root)~~

hasImpverter( )

Do DFS(v)

if (DFS do not have any components)
There exist an Impverter

else
~~(If there are no back edges)~~
there does not exist Impverter.

}
If there are no back edges within components {
~~there does not~~
exist Impverter

}

else if(back edges b/w every component and backed with component end at vertex which has backedge to prev compat)
~~there exist imp vert~~

}

}

Important

→ pass head by reference (not by value)

Page No.
Date

4

1. Put (head, int x) {
create a new node v with value x;
temp = head;
tempPrev = head;

*

while (temp → value < x) {
temp = temp → next;
}

while (tempPrev → next → value < x) {
tempPrev = tempPrev → next;
}

tempPrev → next = v;
v → next = temp;

* if (head == NULL) {
head = v
v → next = NULL
}

if (temp → next == NULL) {
temp → next = v;
v → next = NULL;
}
}

2.

0	1	2	3	4	5	6	7	8	9	10	11	12
-1	-2	0	2	3	4	5	6	7	9	12	13	14

↑
S

~~if iden~~

```
identical index (arr, n) {
int s=0;
int e = n-1;
int m;

~~else~~ while (true) {
        if (s>e)
            return false // or return -1
        if (a[m] > m)
            e = m-1
        else if (a[m] < m)
            s = m+1
        else if (a[m] = m)
            return true; // return m;
    }
}
```

O[logn] → we can make a BST for this

4. maxdepth (root) {

~~if (root → left~~

if (root == NULL) {
return 0;
}

if (root → left == NULL OR
root → right = NULL) {
return 0;
}

ansL = maxdepth(root → left);
ansR = maxdepth(root → right);

~~return max~~

return max{ansL, ansR} + 1;

}

The algorith goes over every vertex its runtiy time = O(n)