COMP90038 Algorithms and Complexity

Tutorial 5 Decrease and Conquer

Tutor: Chuang(Frank) Wang

THE UNIVERSITY OF MELBOURNE

1. Review

Decrease and Conquer

By a Constant
the size of an instance is reduced by the same constant on each iteration of the algorithm

Insertion Sort (Complexity - Quadratic)

```
function InsertionSort($A[\cdot], n$)
    for $i \leftarrow 1$ to $n - 1$ do
        $v \leftarrow A[i]$
        $j \leftarrow i - 1$
        while $j \geq 0$ and $v < A[j]$ do
            $A[j + 1] \leftarrow A[j]$
            $j \leftarrow j - 1$
        $A[j + 1] \leftarrow v$
```

By a Factor

By a constant factor
reduce a problem instance by the same constant factor on each iteration of the algorithm

Binary Search(Complexity - Logarithmic)

```
function BinSearch($A[\cdot], n, k$)
    $lo \leftarrow 0$
    $hi \leftarrow n - 1$
    while $lo \leq hi$ do
        $m \leftarrow \lfloor (lo + hi)/2 \rfloor$
        if $A[m] = k$ then
            return $m$
        if $A[m] > k$ then
            $hi \leftarrow m - 1$
        else
            $lo \leftarrow m + 1$
    return $-1$
```

QuickSelect

```
function QuickSelect($A[\cdot], lo, hi, k$)
    $s \leftarrow$ LomutoPartition($A, lo, hi$)
    if $s - lo = k - 1$ then
        return $A[s]$
    else
        if $s - lo > k - 1$ then
            QuickSelect($A, lo, s - 1, k$)
        else
            QuickSelect($A, s + 1, hi, (k - 1) - (s - lo)$)
```

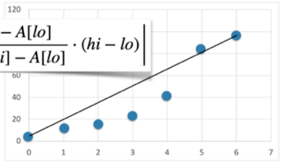
By a variable factor
the size reduction pattern varies from one iteration of the algorithm to another

Interpolation Search

A:	4	9	13	22	41	83	96
	0	1	2	3	4	5	6

Suppose we are searching for k = 83

$$mid = A[lo] + \left\lfloor \frac{k - A[lo]}{A[hi] - A[lo]} \cdot (hi - lo) \right\rfloor$$



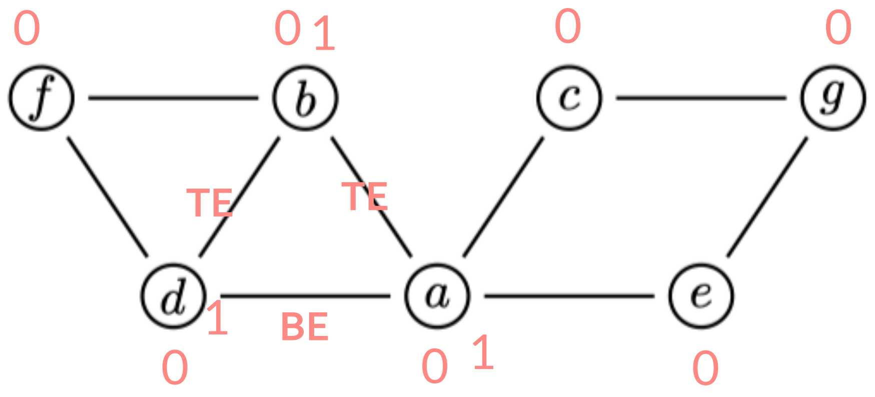
THE UNIVERSITY OF MELBOURNE

2. Tutorial Questions

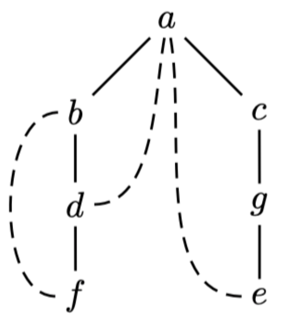
Question 33

33. Write an algorithm to classify all edges of an undirected graph, so that depth-first tree edges can be distinguished from back edges.

Tree edge & Back edge



a	$\to b \to c \to d \to e$
b	$\to a \to d \to f$
c	$\to a \to g$
d	$\to a \to b \to f$
e	$\to a \to g$
f	$\to b \to d$
g	$\to c \to e$



- Whenever a new unvisited vertex is reached for the first time, it is attached as a child to the vertex from which it is being reached. Such an edge is called *a tree edge.*
- The algorithm may also encounter an edge leading to a previously visited vertex other than its immediate predecessor (i.e., its parent in the tree). Such an edge is called a *back edge* because it connects a vertex to its ancestor, other than the parent, in the depth-first search forest.

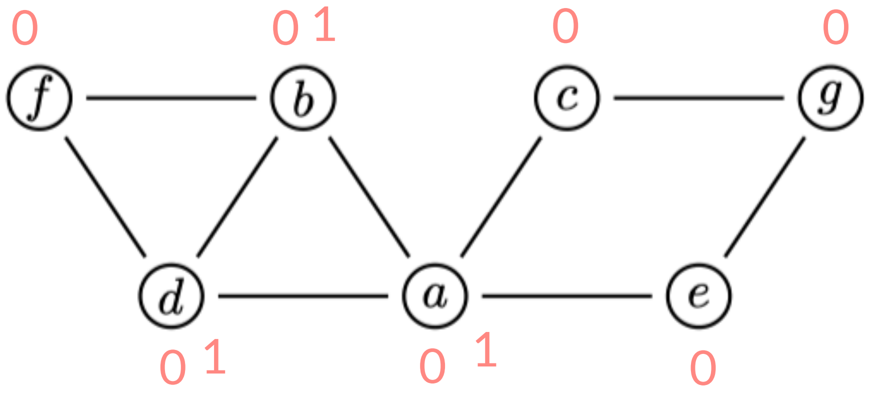
```
function ClassifyEdges(⟨$V, E$⟩)
    mark each node in $V$ with 0
    for each $v$ in $V$ do
        if $v$ is marked 0 then
            DfsExplore($v$)
```

```
function DfsExplore($v$)
    mark $v$ with 1
    for each edge $(v, w)$ do
        if $w$ is marked with 0 then
            classify $(v, w)$ (and thereby $(w, v)$) as "tree edge"
            DfsExplore($w$)
        else
            if $(v, w)$ (and thereby $(w, v)$) is not a "tree edge" then
                classify $(v, w)$ (and thereby $(w, v)$) as "back edge"
```

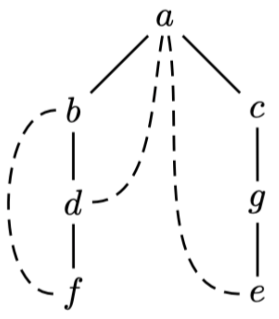
Question 34

34. Show how the algorithm from the previous question can be utilised to decide whether an undirected graph is cyclic.

Solution:



a	$\to b \to c \to d \to e$
b	$\to a \to d \to f$
c	$\to a \to g$
d	$\to a \to b \to f$
e	$\to a \to g$
f	$\to b \to d$
g	$\to c \to e$



v = a   w = b

v = b   w = a

v = b   w = d

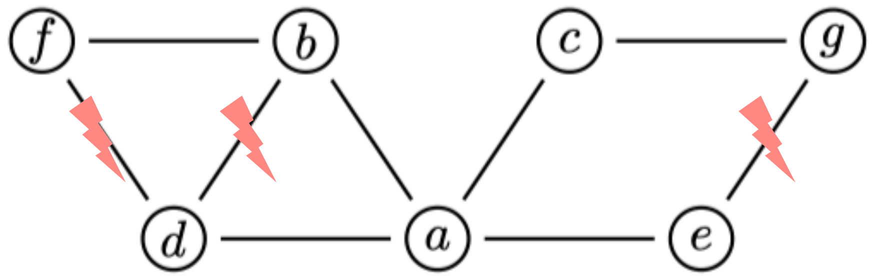
v = d   w = a

```
function CYCLIC($\langle V, E \rangle$)
    mark each node in $V$ with 0
    for each $v$ in $V$ do
        if $v$ is marked 0 then
            if DFSEXPLORE($v$) then
                return True
    return False

function DFSEXPLORE($v$)
    mark $v$ with 1
    for each edge $(v, w)$ do
        if $w$ is marked with 0 then
            if DFSEXPLORE($w$) then
                return True
        else
            if $(v, w)$ is a back edge then
                return True
    return False
```

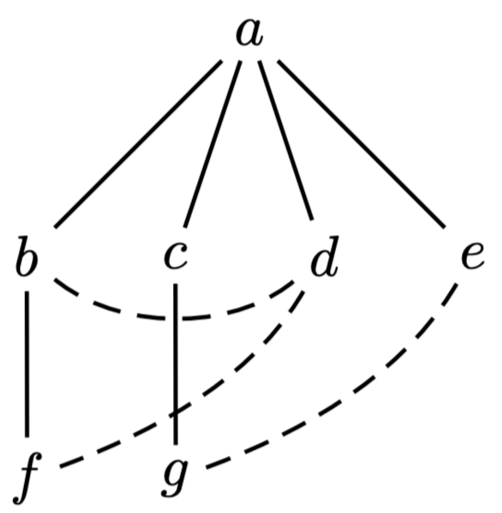
Question 35

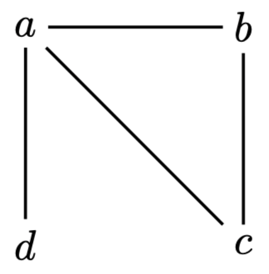
35. Explain how one could also use breadth-first search to see whether an undirected graph is cyclic. Which of the two traversals, depth-first and breadth-first, will be able to find cycles faster? (If there is no clear winner, give an example where one is better, and another example where the other is better.)



A graph has a cycle iff its breadth-first forest contains a cross edge.

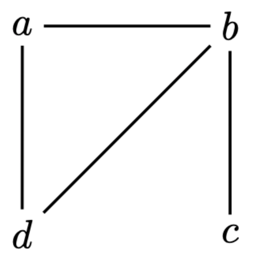
- Whenever a new unvisited vertex is reached for the first time, the vertex is attached as a child to the vertex it is being reached from with an edge called a **tree edge**.
- If an edge leading to a previously visited vertex other than its immediate predecessor (i.e., its parent in the tree) is encountered, the edge is noted as a **cross edge**.





DFS:
a - b
b - a: tree edge
b - c
c - a: back edge

BFS:
a - b
a - c
a - d
b - a: tree edge
b - c: cross edge



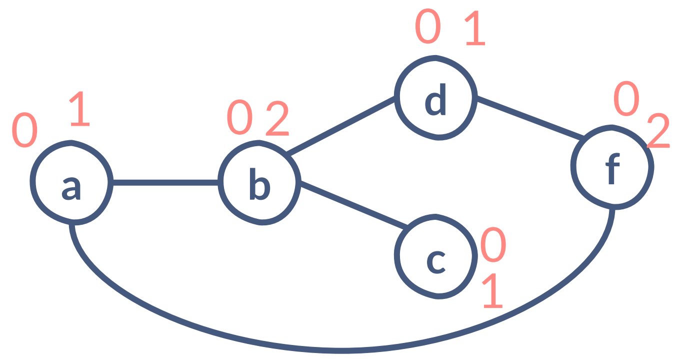
BFS:
a - b
a - d
b - a: tree edge
b - c
b - d: cross edge

DFS:
a - b
b - a: tree edge
b - c
c - b: tree edge
b - d
d - a: back edge

Question 36

36. Design an algorithm to check whether an undirected graph is 2-colourable, that is, whether its nodes can be coloured with just two colours in such a way that no edge connects two nodes of the same colour. Hint: Adapt one of the graph traversal algorithms.

Solution:



v = a	u = b
v = b	u = c
v = b	u = d
v = d	u = f
v = f	u = a

```
function IsTwoColourable($G$)
    let $\langle V, E \rangle = G$
    for each $v$ in $V$ do
        $colour[v] \leftarrow 0$
    for each $v$ in $V$ do
        if $colour[v] = 0$ then
            if Dfs($v, 1$) = False then
                return False
    return True

function Dfs($v, currentColour$)
    $colour[v] \leftarrow currentColour$
    for each node $u$ in $V$ adjacent to $v$ do
        if $colour[u] = currentColour$ then
            return False
        if $colour[u] = 0$ then
            if Dfs($u, 3 - currentColour$) = False then
                return False
    return True
```

Question 37

Topological Sorting Algorithm 1

We can solve the top-sort problem with depth-first search:

1. Perform DFS and note the order in which nodes are popped off the stack.
2. List the nodes in the reverse of that order.

This works because of the stack discipline.

If (u, v) is an edge then it is possible (for some way of deciding ties) to arrive at a DFS stack with u sitting below v.

Taking the "reverse popping order" ensures that u is listed before v.

a

b $\longrightarrow$ a $\longrightarrow$ c

c

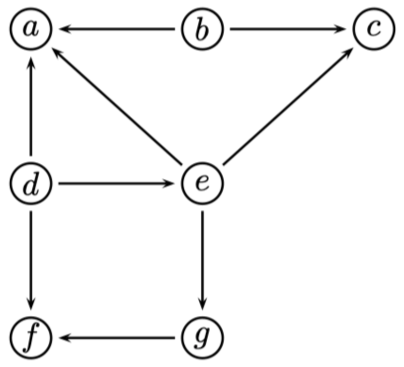
d $\longrightarrow$ a $\longrightarrow$ e $\longrightarrow$ f

e $\longrightarrow$ a $\longrightarrow$ c $\longrightarrow$ g

f

g $\longrightarrow$ f

37. Apply the DFS-based topsort algorithm to linearize the following graph:



				f7	4
				g6	5
		c3	2	e5	6
a1	1	b2	3	d4	7

d, e, g, f, b, c, a.

Topological Sorting Algorithm 2

An alternative method would be to repeatedly select a random source in the graph (that is, a node with no incoming edges), list it, and remove it from the graph.

This is a very natural approach, but it has the drawback that we repeatedly need to scan the graph for a source.

However, it exemplifies the general principle of decrease-and-conquer.

Next Week:

Divide and conquer

Thank you !