Announcements

1. Visualgo Quiz (12%) is this Friday (9 Nov) during Lab time slot.

2. Practical Exam is currently being marked. Might receive feedback after the end of this week.

CS2040C Tut 10

Harder Topics

DFS/BFS Applications

Depth First Search

(2) Complexity Analysis of DFS

(1) Cycle detection using DFS

Complexity Analysis of DFS

1. DFS is **O(V+E)**. [True/False?]

Complexity Analysis of DFS

1. DFS is **O(V+E)**. **[True]**
2. When finding connected components:

   We perform DFS once per connected component (c.c.). [True/False?]

Complexity Analysis of DFS

1. DFS is **O(V+E)**. **[True]**
2. When finding connected components:

   We perform DFS once per connected component (c.c.). **[True]**
3. There are up to **V** c.c. [True/False?]

Complexity Analysis of DFS

1. DFS is **O(V+E)**. **[True]**
2. When finding connected components:

   We perform DFS once per connected component (c.c.). **[True]**
3. There are up to **V** c.c. **[True]**
4. Total time complexity is **V * O(V+E)**

   = **O(V(V+E))** [True/False?]

Complexity Analysis of DFS

1. DFS is **O(V+E)**. **[True]**
2. When finding connected components:

   We perform DFS once per connected component (c.c.). **[True]**
3. There are up to **V** c.c. **[True]**
4. Total time complexity is **V * O(V+E)**

   = **O(V(V+E))** **[False!]**

Complexity Analysis of DFS

Why?

DFS is **O(V+E)** where ...

V is the number of vertices in the c.c.

E is the number of edges in the c.c.

Since the entire graph can be one c.c.

DFS is O(**V+E**)

Complexity Analysis of DFS

Why?

Assume there are **N** connected components:

Let $\mathbf{v_i}$ and $\mathbf{e_i}$ be number of vertices in the $\mathbf{i^{th}}$ c.c.

Then, total time complexity of the **N** DFS:

$$O(v_1 + e_1 + v_2 + e_2 + v_3 + e_3 + \dots + v_N + e_N)$$

Complexity Analysis of DFS

Why?

$$O(v_1 + e_1 + v_2 + e_2 + v_3 + e_3 + \dots + v_N + e_N)$$

$$= O([v_1 + v_2 + v_3 + \dots v_N] + [e_1 + e_2 + e_3 + \dots e_N])$$

$$= O (\mathbf{V} + \mathbf{E})$$

$$\mathbf{[v_1 + v_2 + v_3 + \dots v_N] = V}$$

$$\mathbf{[e_1 + e_2 + e_3 + \dots e_N] = E}$$

Complexity Analysis of DFS

Now you try:

I have **N** integers distributed across **K** arrays.

$1 \leq K \leq N$

For each array: I use **STL sort** to sort it.

What is the total time complexity?

Complexity Analysis of DFS

Now you try:

I have **N** integers distributed across **K** arrays.

$1 \leq K \leq N$

For each array: I use **STL sort** to sort it.

What is the total time complexity?

O(N log N)

Complexity Analysis of BFS

Quick Question

Can the same analysis be applied to **BFS** on different connected components?

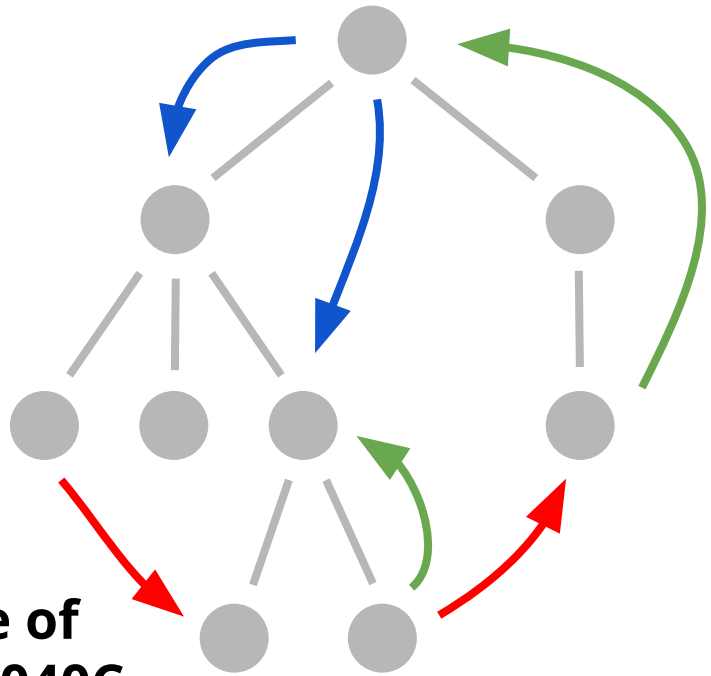
Spanning Trees [Credits: SG IOI Training Team]

Key Idea

- A set of **directed edges** that form a rooted tree **covering all vertices** in a graph.
- The remaining edges can be classified as:
  - **Forward** edges
  - **Back** edges
  - **Cross** edges

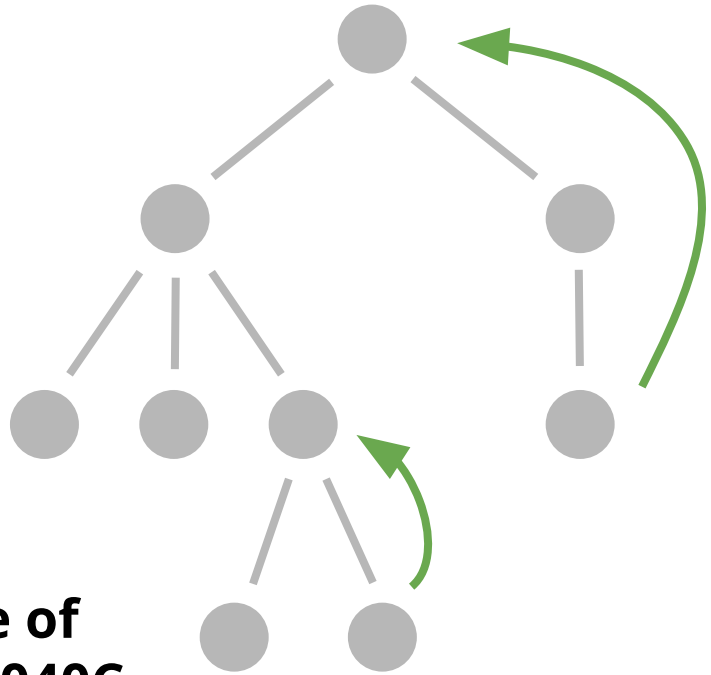
Spanning Trees [Credits: SG IOI Training Team]

- **Tree** edges
- **Forward** edges
- **Cross** edges
- **Back** edges



Don’t need to know significance of Forward and Cross edges in CS2040C.

Spanning Trees [Credits: SG IOI Training Team]



Don’t need to know significance of Forward and Cross edges in CS2040C.

DFS Spanning Trees [Credits: SG IOI Training Team]

- The only edges that can create cycles are **back** edges.
- **Back** edges always point from a node to one of its ancestors.

Need to know in CS2040C.

Cycle finding using DFS

- Run DFS, identify the **back** edges
  - **back** edges point to ancestors

- Cycle detection
  - 0 **back** edges → no cycles
  - Have **back** edges → cycles

Cycle finding using DFS

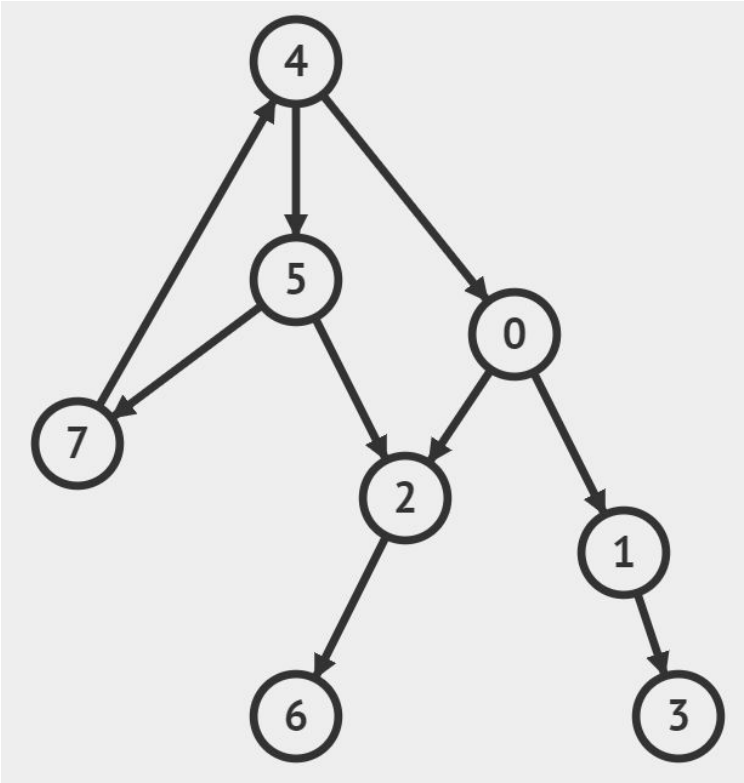
- We need to keep 3 states in the DFS
  - **Unvisited**
  - **Visiting**
  - **Visited**
- **Back** edges go from **Visiting**→ **Visiting**.
  - Ancestors are still in **Visiting** state.
  - Non-ancestors will be in **Visited** state.

Cycle finding using DFS

Why do we need **Visiting**?

Or else:

- **Back** edges go from **Visited**→ **Visited**.



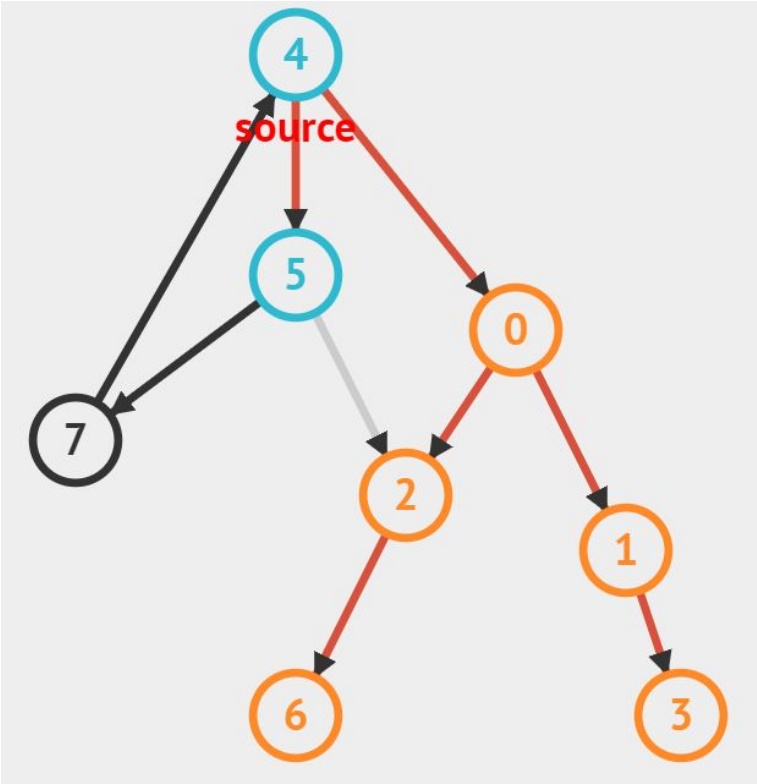
Cycle finding using DFS

If no **Visiting** state:

Vertex 5: **Current Vertex**

Vertex 2: **Visited**

Not a **back** edge



Cycle finding using DFS

If no **Visiting** state:

Vertex 5: **Current Vertex**

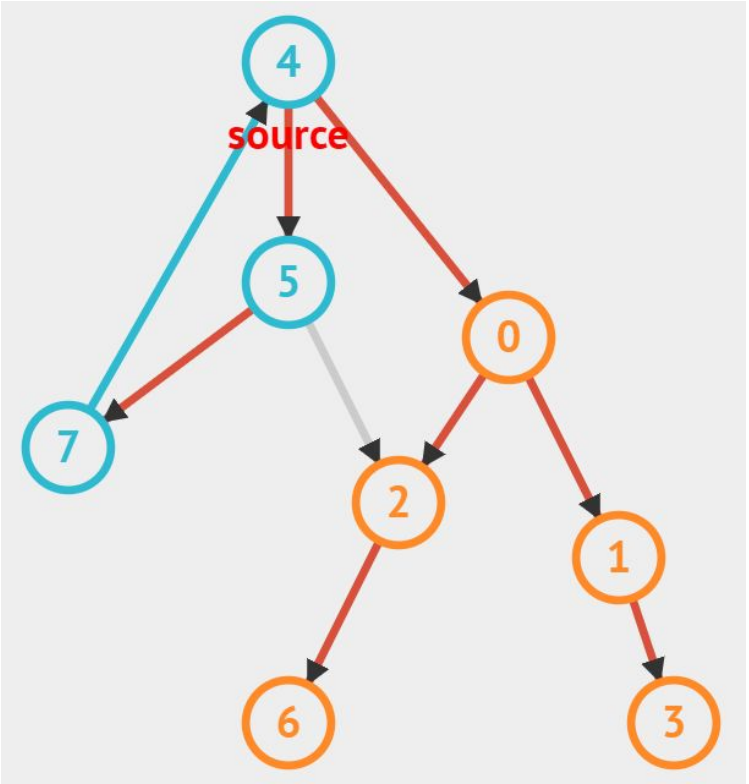
Vertex 2: **Visited**

Not a **back** edge

Vertex 7: **Current Vertex**

Vertex 4: **Visited**

Back edge



Cycle finding using DFS

If we have **Visiting** state:

Vertex 5: **Current Vertex**

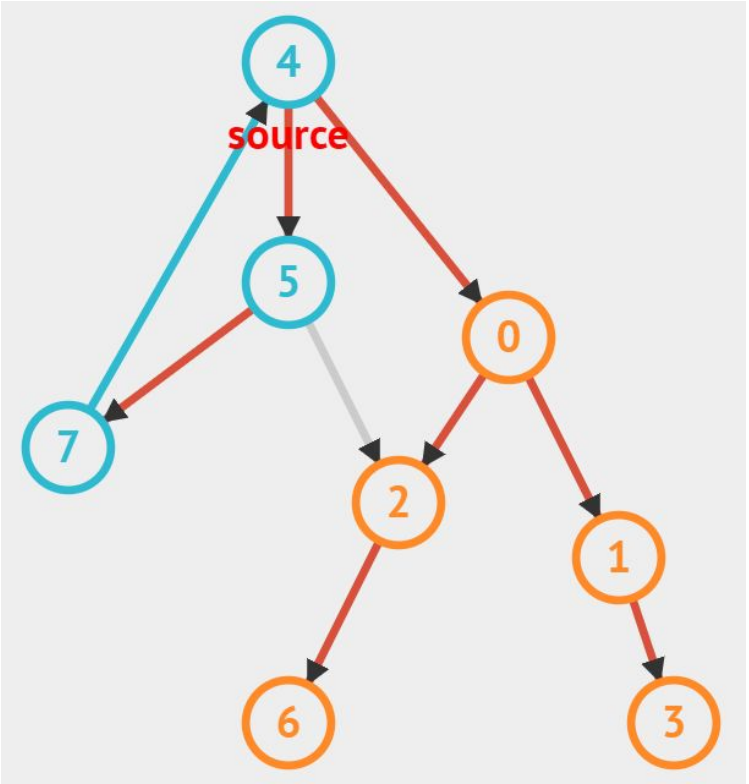
Vertex 2: **Visited**

Not a **back** edge

Vertex 7: **Current Vertex**

Vertex 4: **Visiting**

Back edge



Flood Fill

Example Problem 1 - countingstars

https://open.kattis.com/problems/countingstars

Counting Stars

Summary

You are given a **N** by **M** grid of characters.

Each cell is either ‘#’ or ‘-’.

‘#’ cells are black pixels, ‘-’ cells are white pixels.

White pixels that are adjacent vertically or horizontally are part of the same star.

Counting Stars

Summary

You need to count how many stars there.

Approach

When we count a star, lets remove it from the picture.

(All the ‘-’ that correspond to the same star)

Counting Stars

Approach

When we count a star, lets remove it from the picture.

(All the ‘-’ that correspond to the same star)

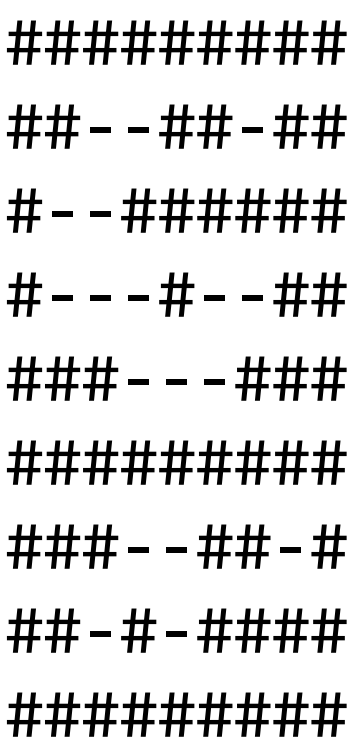
Repeat this for every ‘-’ that has not been removed yet.

Counting Stars

Approach

Lets start off with this grid:

We loop through to find the first ‘-’ cell.

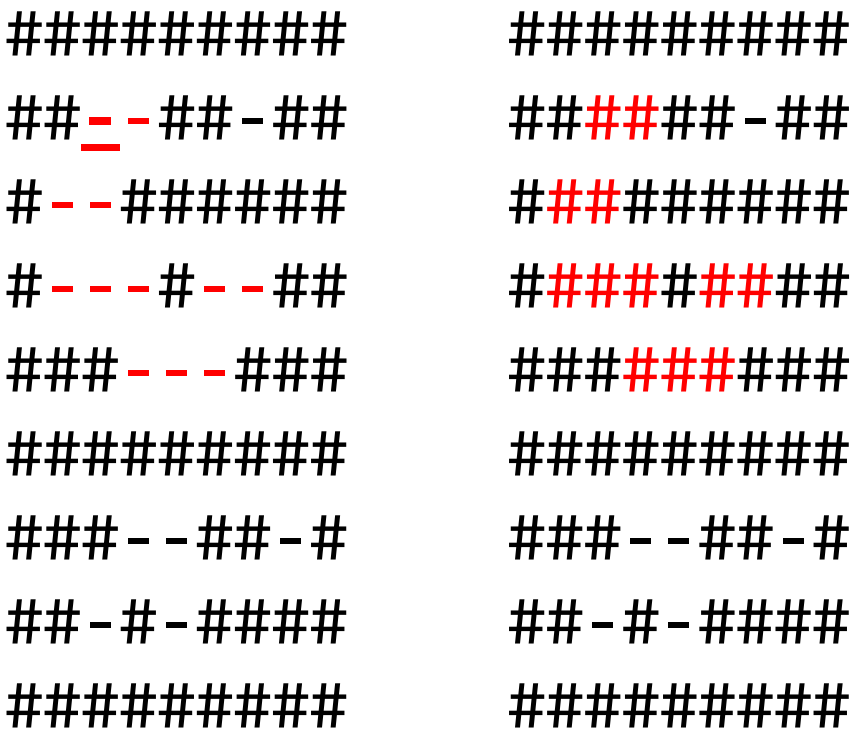


Counting Stars

Approach

First star →

Then, remove all the '-'

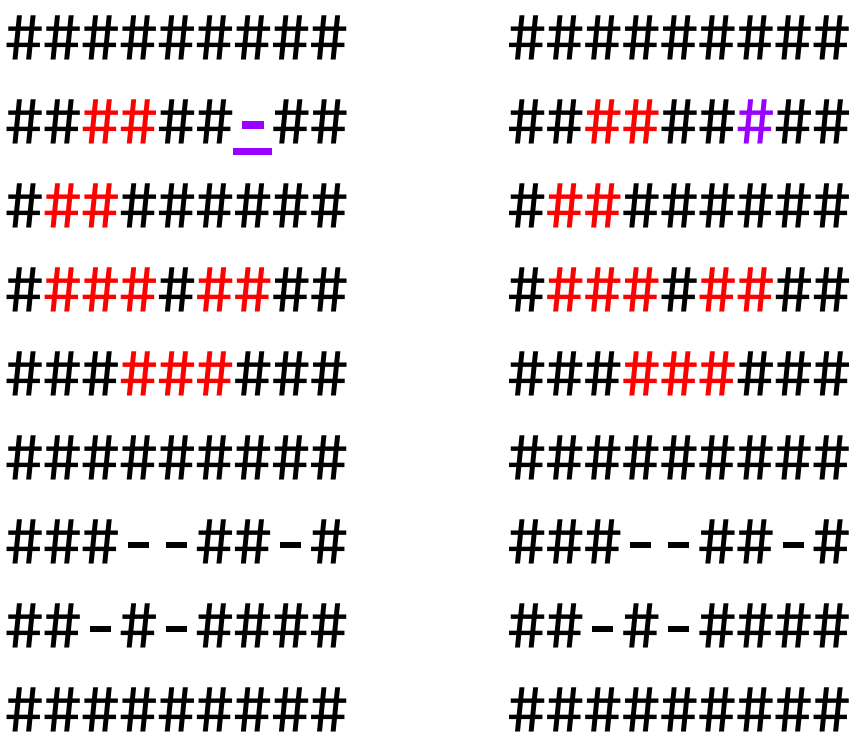


Counting Stars

Approach

Second star →

Then, remove all the '-'

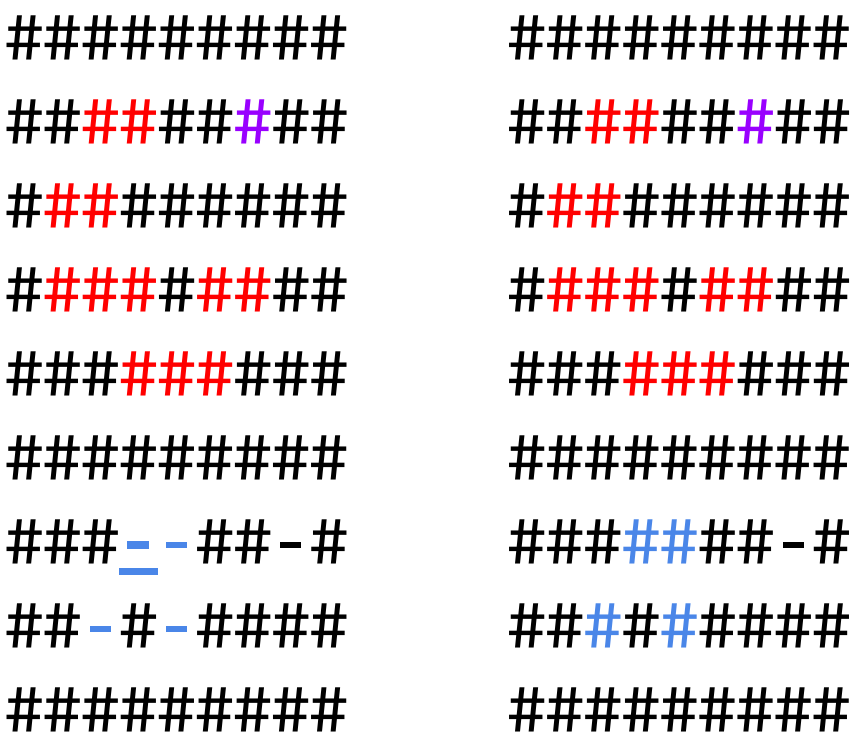


Counting Stars

Approach

Third star →

Then, remove all the '-'

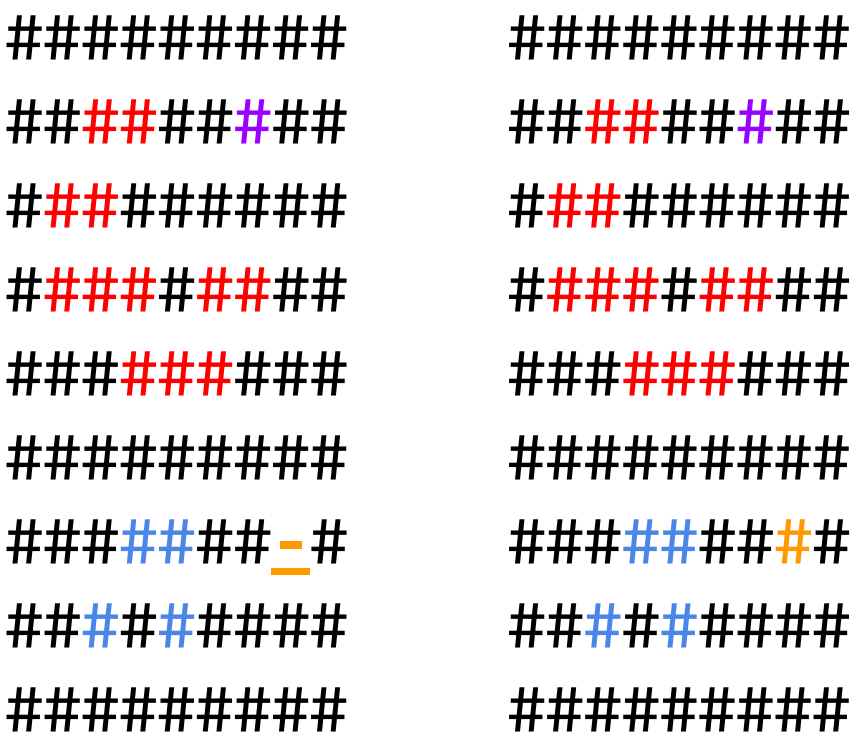


Counting Stars

Approach

Fourth star →

Then, remove all the '-'



Counting Stars

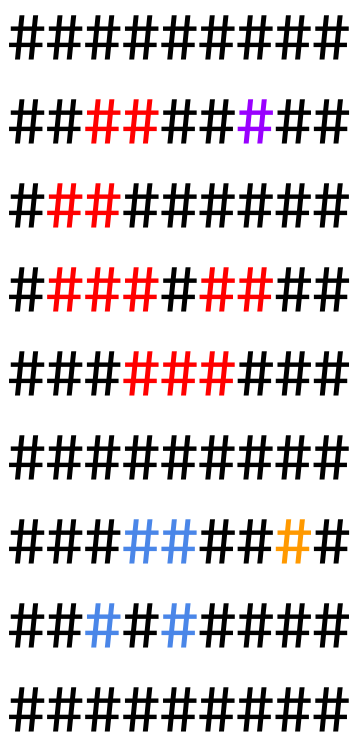
Approach

In total, there are 4 stars.

In graph terms, this counting the number of *connected components*.

Cells → vertices

Edges → between adjacent cells that are '-'



‘Facebook Privacy Setting’

CS2010 FE AY2013/2014 Sem 1

O(V+E) solution

O(k) solution

‘Facebook Privacy Setting’

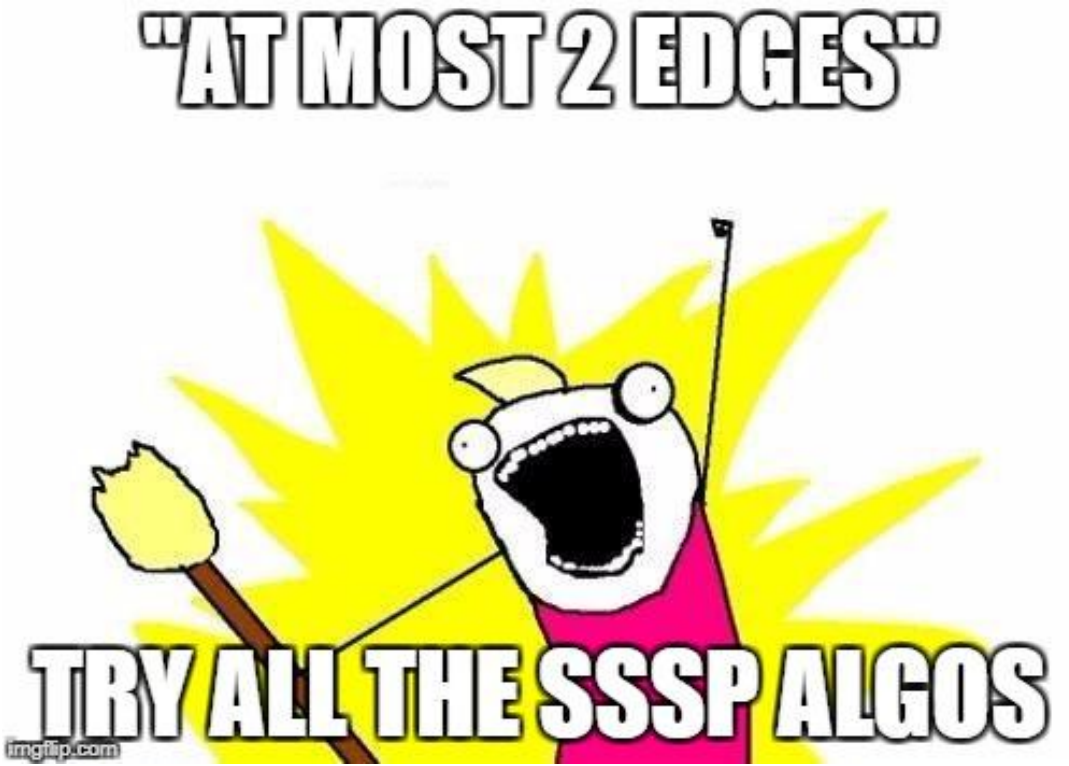
Summary

- You have a graph of **V** vertices, **E** edges.
- You are given **one** pair of vertices, **i** and **j**.
  - Compute whether vertex **i** and **j** are *at most* 2 edges apart.
- O(**V**+**E**) for 7 marks
- O(**k**) for 19 marks
  - **k** is sum of number of adjacent vertices of **i** and **j**

‘Facebook Privacy Setting’

O(V+E) solution

- Bellman Ford?
  - O(VE)
- Dijkstra?
  - O((V+E) log V)
- … :(



‘Facebook Privacy Setting’

O(V+E) solution

Breadth First Search

- Start from vertex **i**, compute shortest path from **i** to every other vertex.
- Check if **dist(i, j) ≤ 2**

‘Facebook Privacy Setting’

Observation

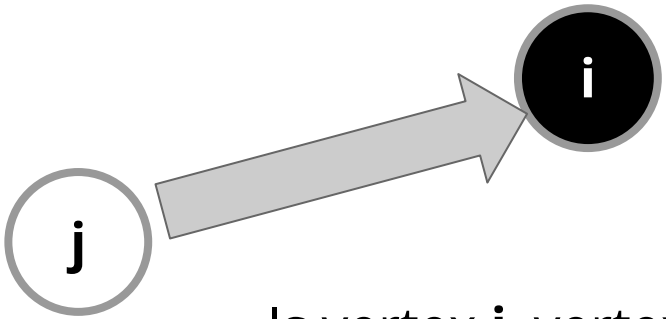
We only need **distance ≤ 2**.

3 cases:

- distance = 0 (**i** == **j**)
- distance = 1 (check neighbours of **i** or **j**)
- distance = 2 (???)

‘Facebook Privacy Setting’

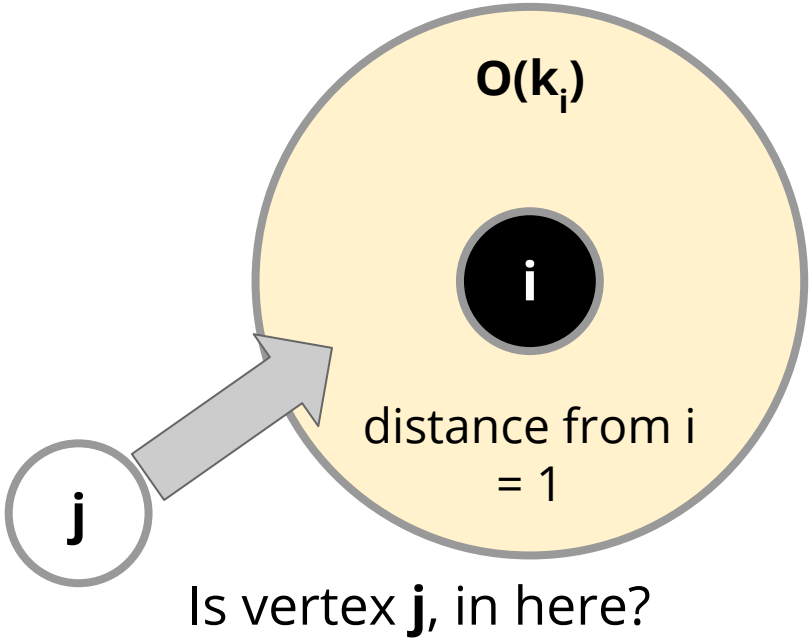
Distance = 0



Is vertex **j**, vertex **i**?
O(1)

‘Facebook Privacy Setting’

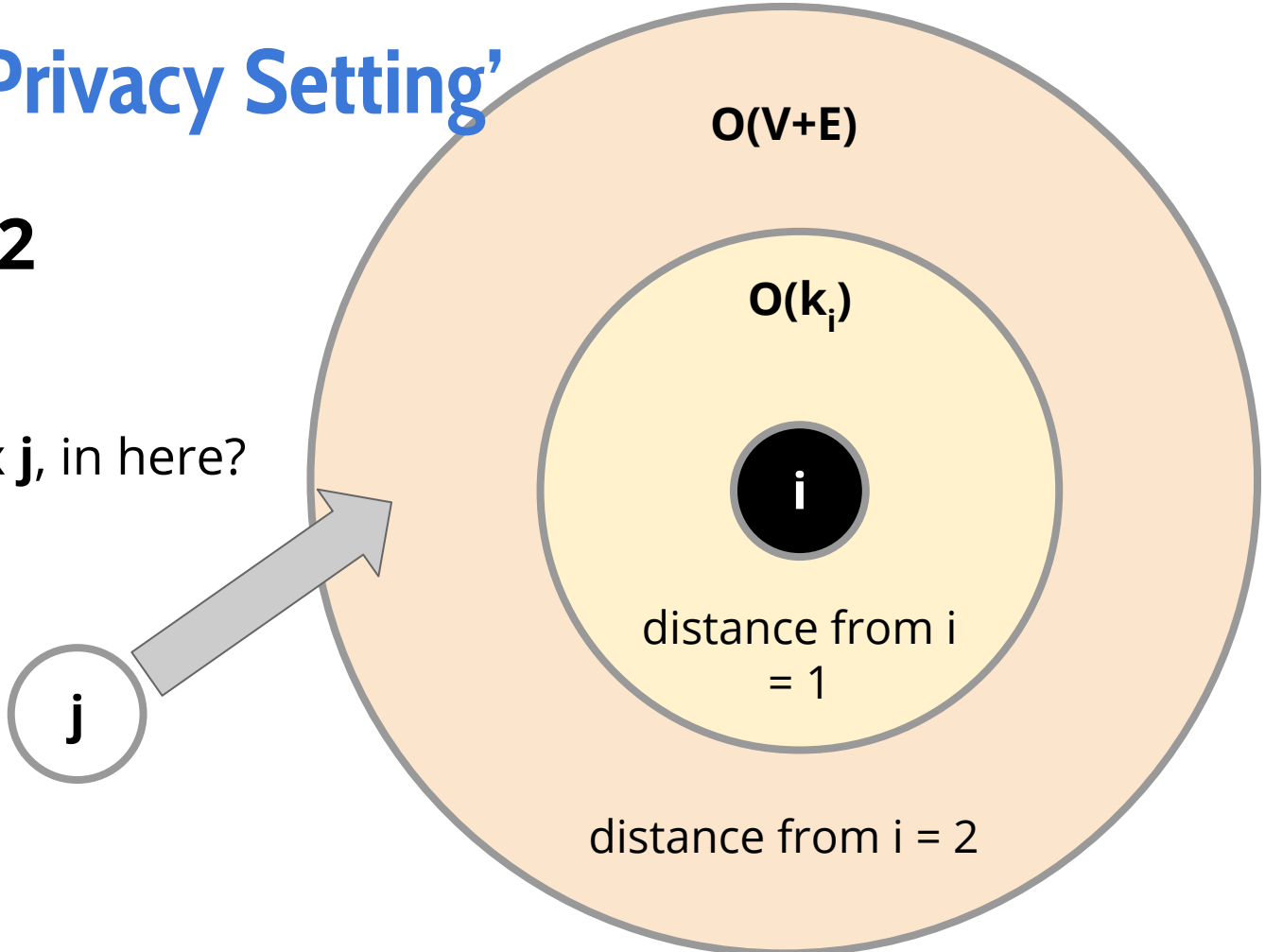
Distance = 1



‘Facebook Privacy Setting’

Distance = 2

Is vertex **j**, in here?



‘Facebook Privacy Setting’

Random question

You are in **NUS** now. (West)

Your girlfriend/boyfriend just arrived at **Changi Airport**. (East)

What should your do in order to be able to meet each other in the shortest possible time?

‘Facebook Privacy Setting’

Random question

a) Meet at NUS.
b) Meet at Changi.
c) Meet somewhere in the middle.
d) Use video call.
e) This is a scam.

I don’t have a girlfriend/boyfriend.

‘Facebook Privacy Setting’

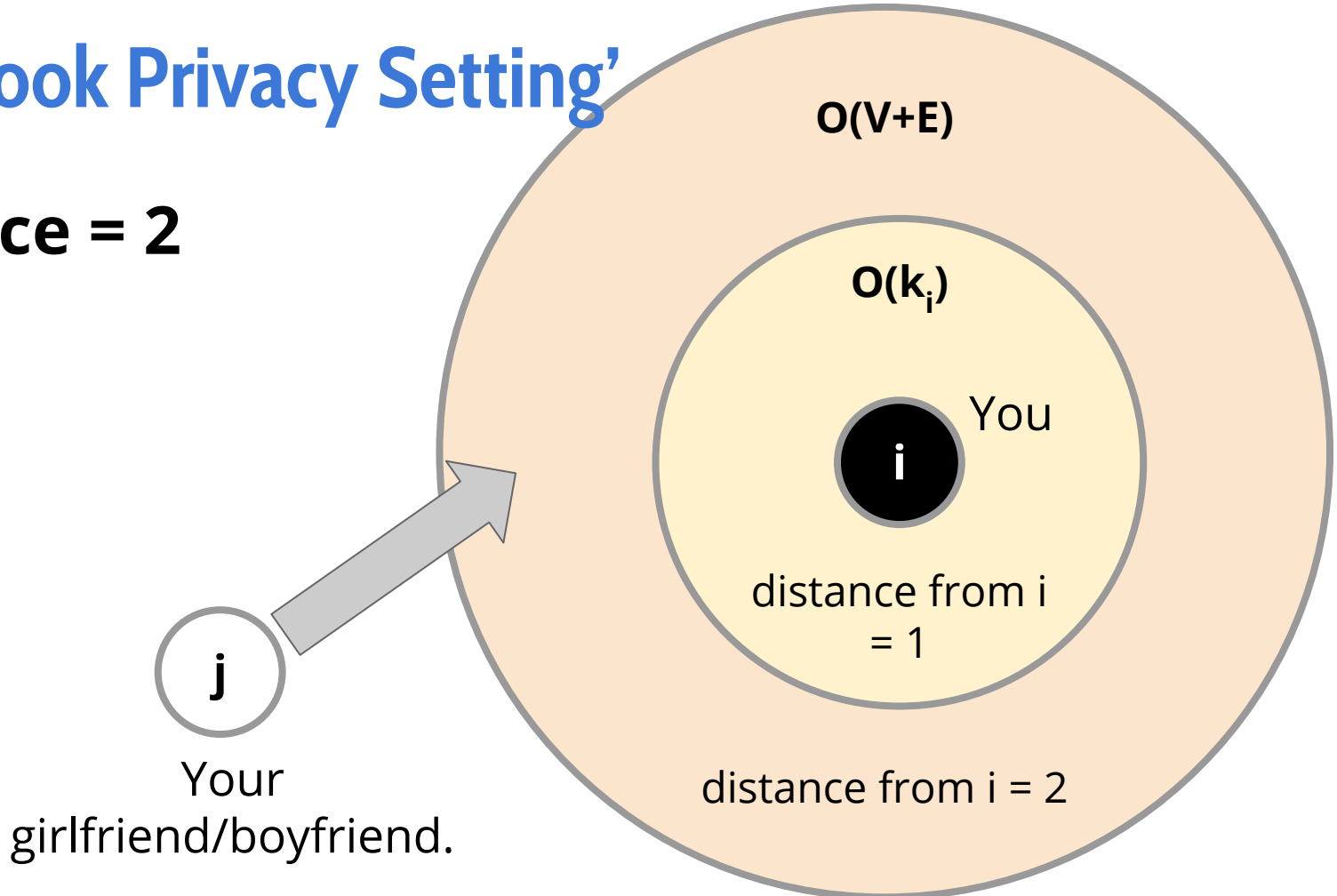
Meet in the Middle

a) Meet at NUS.
b) Meet at Changi.
c) Meet somewhere in the middle.
d) Use video call.
e) This is a scam.

I don’t have a girlfriend/boyfriend.

‘Facebook Privacy Setting’

Distance = 2

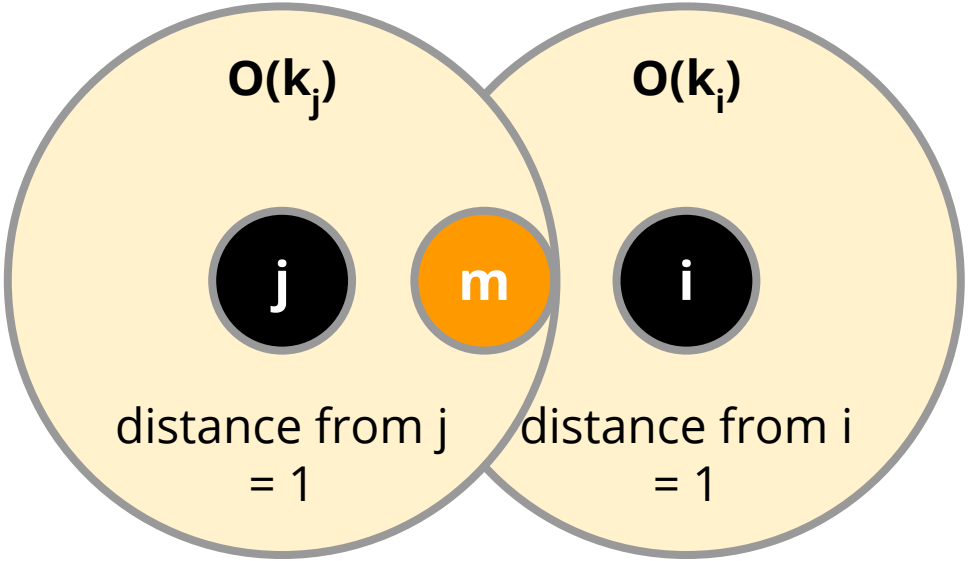


‘Facebook Privacy Setting’

If vertex **i** and **j** are distance 2 away, what should happen?

‘Facebook Privacy Setting’

If vertex **i** and **j** are distance 2 away, there must be a ‘mutual friend’.



‘Facebook Privacy Setting’

Transformed Problem

Given 2 integer arrays **A** and **B**, find whether there exist an integer that is in both arrays.

$O(N^2)$?

O(N log N) [sort + 2 pointers?]

‘Facebook Privacy Setting’

Transformed Problem

Given 2 **sorted** integer arrays **A** and **B**, find whether there exist an integer that is in both arrays.

profile is given a unique integer index i. The friend list of profile i is stored in `AdjList[i]` and **sorted in ascending order**.

O(N) using **2 pointers** → O(k) in original problem

Moral of the Story

Don’t judge a book by its cover

What might appear like a Graph question, might not actually be a Graph question.

What might appear not a Graph question, might actually be a Graph question.

PS5

The Onset of Labour

PS5A & PS5B & PS5C

Conditions

- How large can **V** get?
- How large can **E** get?
- How large can **Q** get?

PS5A & PS5B

Conditions

Guaranteed that **K = V**

- Effectively no restriction on number of junctions
- Standard shortest path algorithm

PS5A

Conditions

Guaranteed an *undirected,* weighted *tree*.

- **E = V-1 (inferred)**
- One DFS/BFS is O(V+E)
  - O(V) for tree.
- How many times can you run DFS/BFS without exceeding Time Limit?

PS5A

Conditions

Guaranteed an *undirected,* weighted *tree*.

- **E = V-1 (inferred)**
- Bellman Ford is $O(V^2)$
- Dijkstra's Algorithm is O(V log V)

PS5A

Single Source Shortest Path

- Shortest Path from the *source* vertex to every other vertex
- If you need to calculate the distance between **every pair** of vertices, how many SSSP algorithms do you need to run?

PS5A

Tree Property

- There is only one simple path between two vertices of a tree.
- That same path is the longest/shortest/*whatever* path :)

PS5B

Conditions

Guaranteed a *directed,* weighted *graph*.

- **$0 \leq s \leq 9$**
- What does this *mean*?
- Do you need to run a shortest path algorithm for **every** of the **Q** queries?

PS5C

Conditions

No guarantee that **K = V**.

- **$1 \le k \le \min(20, V)$**
- **$1 \le Q \le 20$**

Hint:

- Why are **k** and **Q** *small*? Can we exploit it?

Questions?