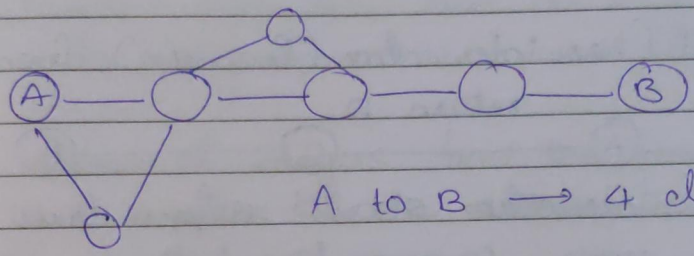
Page No. 1
Date 10 / 7 / 22

DARE TO DREAM

Problem :- Find the degree of seperation between two users/nodes on a graph of users on Facebook.
There is an edge between two nodes A & B iff A is friends with B. Each ~~node~~ edge represents a degree.

Note :- 1. The degree of seperation between two nodes A & B is the number of edges we have to traverse from A to B (shortest).



A to B ⟶ 4 degree seperation

2. The degree between two nodes is never more than 5. (observed using data science on facebook)

Thoughts :-

1. BFS — Order of BFS = O(V+E)
This will take a lot of time considering the billions of users on Facebook.

Date

Meet in the middle

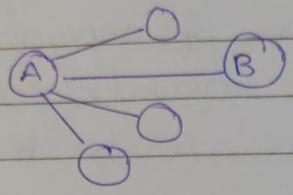
Solution

GetDegreeOfSeparation(A, B):

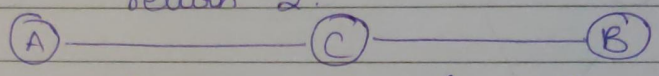
COMPUTATIONS Let's look at the cases we would encounter

$O(1)$ Case 0: The nodes A & B are one and the same. In this case, the degree of seperation is zero.

$O(1000)$ Case 1: Nodes (A) and (B) are directly connected.
if B in friends(A): return 1.

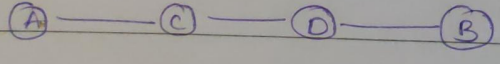


$O(1000^2)$ Case 2: if (has_intersection (friends(A), friends(B))):
return 2.

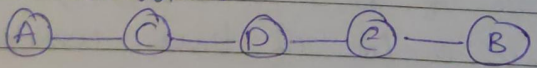


Here, has_intersection should return true if there exists a node common to both the sets.

$O(1000 \times 1000)$ Case 3: if (has_intersection (friends(friends(A)), ~~friends(friend~~ friends(B))):
return 3.



$O(10^6)$ Case 4 if (has_intersection (friends(friends(A)), friends(friends(B)))):
return 4



Case 5: else: return 5

This approach has a couple of names
1. Meet in the middle BFS
2. Two sided BFS

PROBLEM HOW CAN YOU FIND has_intersection(list1, list2) in the following CASES?

CASE 1 : YOU CAN USE EXTRA SPACE

CASE 2 : YOU CAN NOT USE EXTRA SPACE.

SOLUTION CASE 1 :- Use hashmap to store each & every element in the first list
Run a for-each loop on the second list and if the element is present in the set, ~~append it to the resultant list~~. OR RETURN TRUE

CASE 2 :- SORT the lists and do a binary search for every element in list 2. If found, return TRUE

PROBLEM Given a stream of integers, find the median at every step. What happens if the stream is spread across multiple machines?
NOTE :- We need median when the stream is sorted.

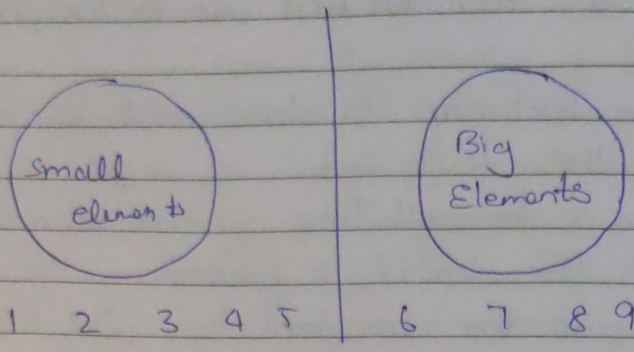
Thoughts In case of a single machine,
1. Have a ptr i pointing to the first number
Have a counter ready
2. If the counter %2 == 0, j = i+1 (arr[i] + arr[j])/2 is the answer
else, i += 1, arr[i] is the answer.
3. Go to step 2, after accepting the new number

Reality The approach mentioned above won't work as we missed the sorting part.

Other Approaches 1. Tree Sets → But this does not support finding middle element. We can put them in an array and get the middle. But expensive
2. BST → But this has a problem with duplicates.

[GOOGLE]

Solution Let's assume we are able to maintain two unique sets where one set has only smaller elements and the other only bigger as shown



- In this scenario, we only care about the largest & the smallest numbers on the left and right sets respectively.
- If both the sets / collections have the same no. of elements, then we take $\frac{\max(\text{first}) + \min(\text{second})}{2}$
- Else if size(first) > size(second), return max(first)
- Else, return max(second)

- Which data structure is perfect for this?
  MIN & MAX HEAPS

Min / Max heap

Returning min/max - O(1)
Insertion - O(logN)
Deletion of min/max - O(logN)

INCASE OF MULTIPLE MACHINES

- Given a 'k', we can get the number of values smaller than k & greater than k. Depending on the numbers, we can shift the binary search