**Question**

Social Network:

How would you design the data structures for a very large social network like Facebook or LinkedIn? Describe how you would design an algorithm to show the shortest path between two people

(e.g Me -> Bob -> Susan -> Jason -> You)

**Assumptions**

1. It has been mentioned as a very large social network which means that so many connections (friends) are possible and the order may vary and if we are going to use traditional comparing mechanism to find the shortest path it would be cumbersome.
2. We could pre-process while a new friend is being added and store the path to each of his friends but again it will involve huge memory space.
3. Here in order to approach a solution we need to design an class **Person** and it will have personal details and the Friend i.e., HashMap<AlphabeticalCharacter, DoublyLinkedList<Person>>
   1. HashMap – Since we need the data structure to look for a particular friends path, It would be easier it we have an big(1) or at least a nearer value for finding the specified Friend.
   2. DoublyLinkedList – Because when we are adding a new Friend if we are going to sort and insert them in alphabetical order. This will make it actually difficult since linked list means we cannot use the Binary Search or some technique to find more quickly. But if we are inserting in an order it will be easy for some other operations like get the list of Friends starting with a specific letter in order.
4. Tree concept can be used find the path to the friend.
5. Breadth First Search concept leveraging the Queue can be utilized.
6. Queued object should contain in the following format
   1. Person (Current Friend)
   2. ArrayList or LinkedList of all the previous graph vertex (path of friends like the one in example)
7. All the above procedures will be clearly explained below

**Queue**

**Algorithm**

1) Add the initial vertex to the Queue and start the process.

2) Visit each vertex of the tree and find using the hashing if the target friend is present. If so then take the path and break out of the loop.

3) If not check each of its friend if they have be added to the queue previously by checking some visited node flag value and ignore them add only the new vertex. This is to avoid infinite loop

|  |
| --- |
| Me |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | D | … | L |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| B | C | D | … | L | M | N |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| C | D | … | L | M | N | R |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D | … | L | M | N | R | S | T |

**Me**

A

B

C

D

L

B

Me

M

N

O

P

Q

R

S

T

You

W

V

M

R

O

P

|  |  |  |  |
| --- | --- | --- | --- |
| … | … | … | You |

4) Upon visiting a new vertex send the path (i.e. the previous friends of it), Let consider P who is a friend of M, so when we are visiting P we will have a list of its parent nodes (M, A, Me)

5) Continue the process until we reach the Target node. Here we are using the Breadth First Search since the shortest path is the one which is required. It might be cumbersome but the shortest path can be found in this way.