HM String : Freq
K V

a
- I love leetcode — 7
- I love you — 5
- I love brijani — 2

a
- Island — 3
- I love India — 10

b
- Ironman — 3
- ICE Cream — 2
- India — 2
- Srivarn — 4
- Swathy — 3
- Rohit — 2
- RD — 1

I	#
I love LC	
I love U	
Ironman	

Search String:

I

StartsWith(Search String)

TRIES

Ironman < Island
b a

a.lt(b)

a < b

def __lt__(self, other):
    if self.val == other.val:
        return self.key > other.key
    return self.val < other.val

a (S) b (R) (True)

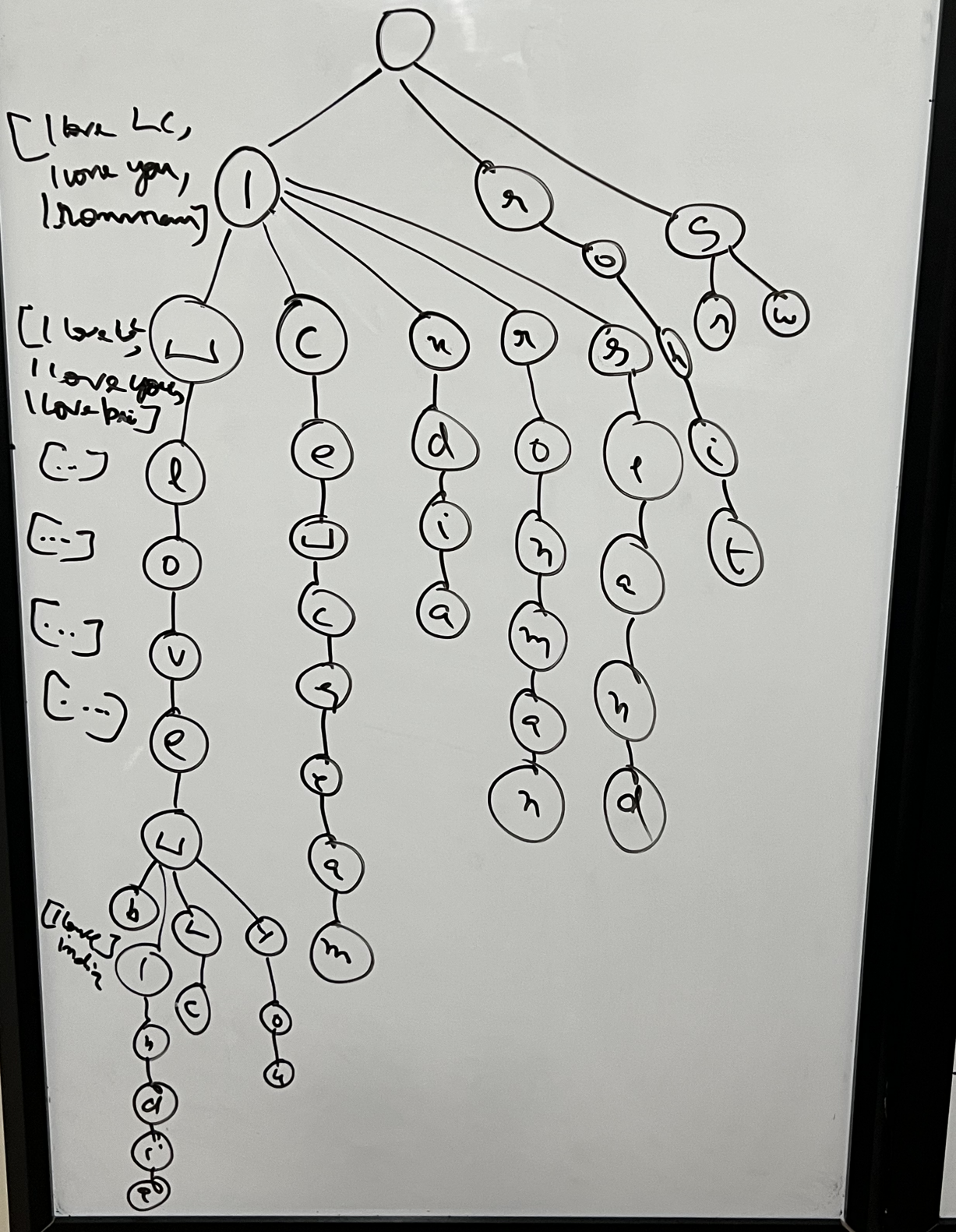
Island a
Ironman b

b | Ironman - Top
a | I love Leetcode - bottom

3 (False)

Search Autocomplete System

K = 3

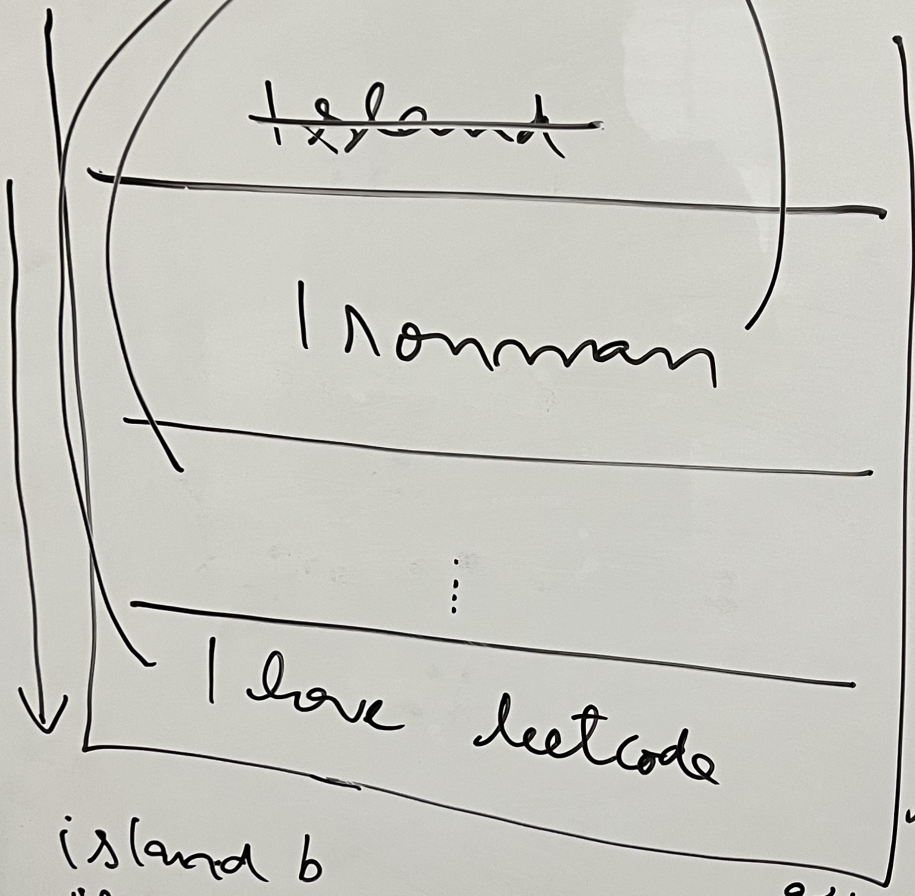


Search Autocomplete System

K = 3

a, b
b - a
b < a

Result = [, ,]



island b
ironman a

a < b
False

TC: N(1 + log k)

```
if s.val == o.val:
    return s.key > o.key
```

(False)

HM String : Freq
K V

a | I love leetcode – 7
| love you – 5
| love brijani – 2
a | island – 3
b | ironman – 3
| ice cream – 2
| india – 2
| Srivani – 4
| Swathy – 3
| Rohit – 2
| Rv – 1

| I ↳ #
| love LC
| love U
| ironman

Search String:
I

startswith(Search String)
TRIES

ironman < island
b a

a.lt(b)
a < b

def __lt__(self, other):
a b
if self.val == other.val:
return self.key > other.key
island a
ironman b

return self.val < other.val
a b
(S) (R) (True)

b | ironman - Top
a | I love leetcode - bottom

3 (False)

HM String : Freq
K V

| love leetcode - 7
| love you - 5
| love biryani - 2
a | island - 3
b | ironman - 3
| ice cream - 2
| india - 2
| srivarn - 4
| swathy - 3
| Rohit - 2
| RV - 1

| I #
| love LC
| love U
| ironman

Search String:
I

startswith(search string)
TRIES

ironman < island
b a

a.lt(b)
a < b

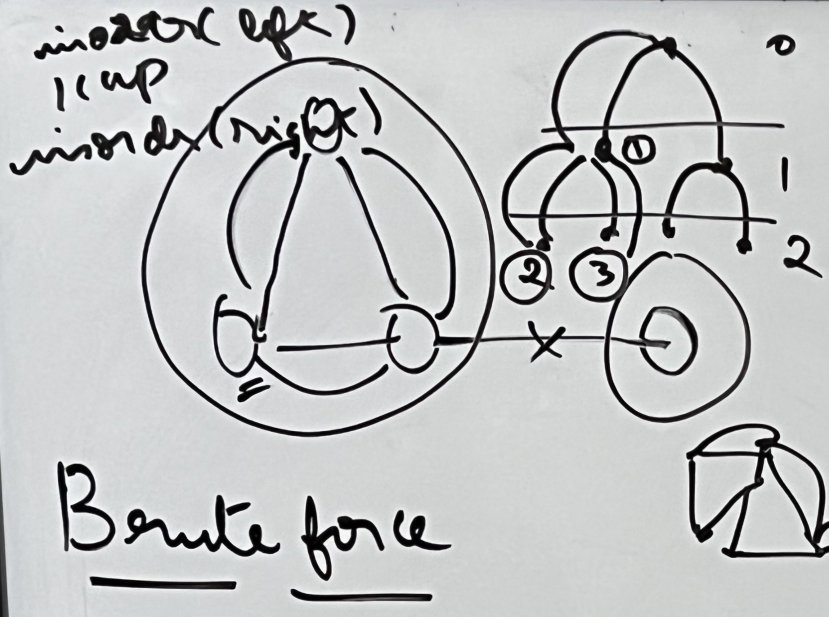
def __lt__(self, other):
a b
if self.val == other.val:
return self.key > other.key

island a
ironman b

Return self.val < other.val
a (S) b (R) (True)

b | ironman - Top
a | love leetcode - bottom

3 (False)




```
{ 0 : [1,2]
  1 : [0,2,3]
  2 : [0,1]
  3 : [1,4,5]
  4 : [3,5]
  5 : [3,4] }
```

Brute force

- Remove every edge one by one
- Do DFS
- Check if you can traverse all n nodes

TC: O(E(V+E))
SC: O(V+E)

if yes:

edge is not critical

else:

edge is Critical connection

---

arr

0	1	2	2	3	4
0	1	2	3	4	5

min

0	0	0	2	2	2
0	1	2	3	4	5

Critical Connections (TARJAN)

[[1,2],[0,2,3],[0,1],[1,4,5],[3,5],[3,4]]

0, 0 adj

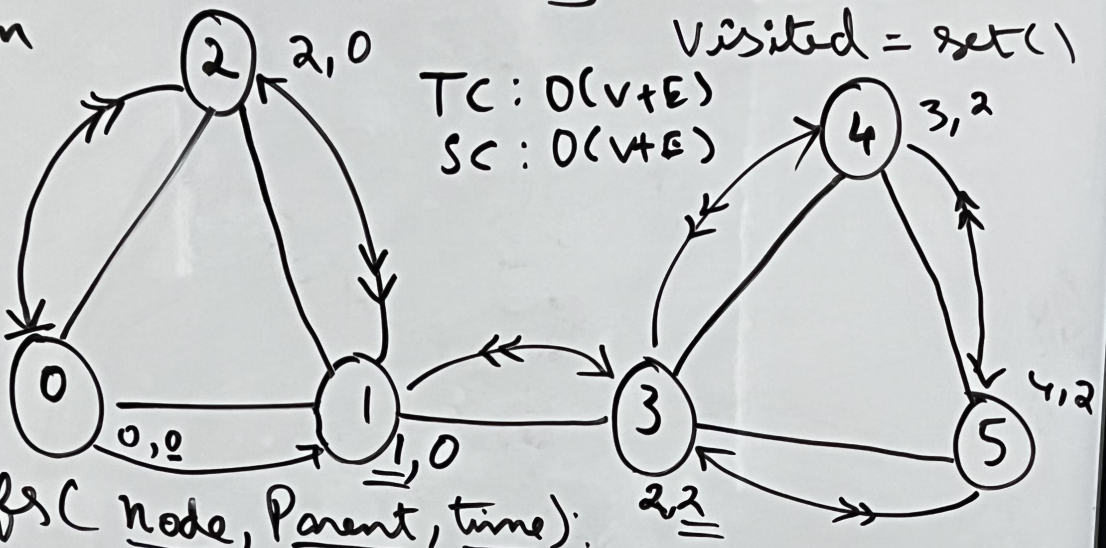
a m

- discovered / arrival time
- lowest / Minimum time

Visited = set()

TC : O(V+E)

SC : O(V+E)



dfs(node, Parent, time):

Visited[node] = True

time += 1

ans = time

mint = time

- node (1)
- Parent (0)
- nchild [0,2,3]

for nchild in adj[node]:

if nchild != Parent

if not visited[nchild]:

dfs(nchild, node, time)

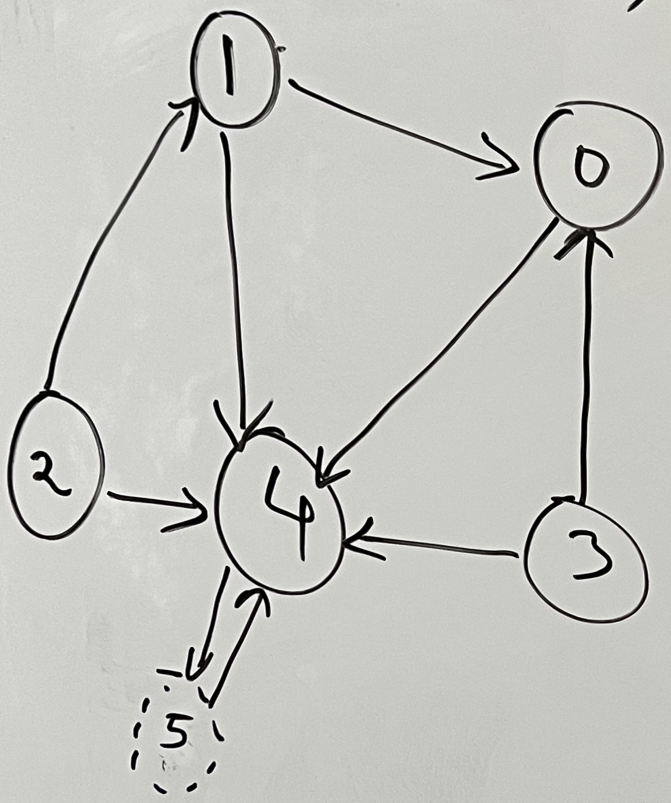
Visited 0, 1, 2, 3, 4, 5

if min_time[nchild] > arrival time[node]:

CRITICAL CONNECTION BETWEEN (NCHILD, NODE)

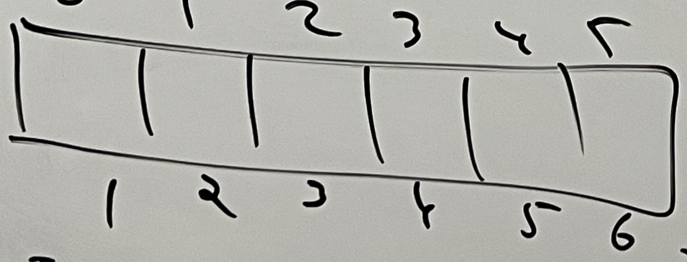
min time[node] = min(mintime[node], mintime[nchild])

[1,0], [1,4], [2,1], [3,4], [2,4]
[3,0], [0,4]]



---

1 - n



n=6

Trust=[0]* n

if T[i]==n-1

Town Judge

1, Everyone needs to trust Judge

2, Town Judge doesn't trust anyone

---

return -1, if no TJ exists

Person	Trusted by
0 :	1, 3
1 :	2
2 :	
3 :	
4 :	1, 2, 3, 0

Person	Trusts
0 :	4
1 :	0, 4, 5
2 :	1, 4
3 :	0, 4
4 :	

1	-1	-2	-2	4
0	1	2	3	4

Trust

4

-1

if trust[i] == n

return i

return -1

TC: O(E)

SC: O(n)