Warm up: Q. n balls.

n = 7

Follow up ↓

Quant* companies

Painter → color them up. Red Blue

↳ not so good with eyes.

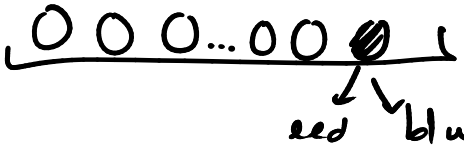
no of ways → color 'n' balls.

2 consecutive balls with red color.

i/p : n

o/p : —

Approach :



let r_n ⇒ no of n balls sequence where the last ball is red.

b_n ⇒ no. of n balls seq. where last ball is blue.

$$b_{n+1} = r_n + b_n$$

$$r_{n+1} = b_n \Rightarrow r_n = b_{n-1}$$

Recurrence relations

$r_1 = 1$

$b_1 = 1$

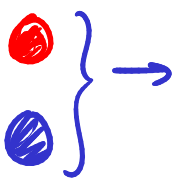
$$\boxed{b_{n+1} = b_n + b_{n-1}}$$

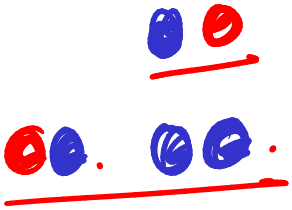
* Fibo.

$$ans = r_n + b_n = b_{n-1} + b_n = b_{n+1}$$

n = 10 return 11th fibonacci number

n=1 n=2





n=3

$b_3 = 1 + 2$

$r_3 = 2$

10 $b_{10} + r_1$

1 1 2 3 5 8 13 21 ....

Q. String of length 'n'. character → alphabet. (26 choices)

Variation. At least one consecutive vowels occurrence should be there in a string. #ways.

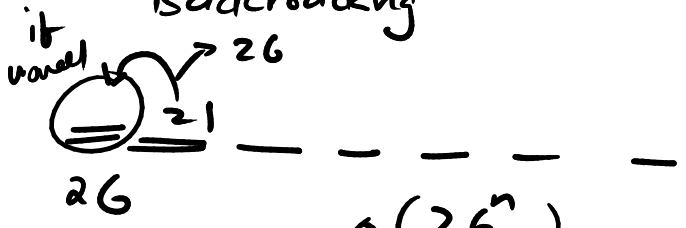
$n = 3$ **aex**

not a good approach / BF.

all possible — no vowels are consecutive

did not ask → actual strings | #count

Backtracking



$O(26^n)$ ↳ polynomial $O(n)$

$ans = 26^n - 21^n$??

$C = 21$ $V = 5$

$C_1 = 21$ $V_1 = 5$

$n \rightarrow ans = C_n + V_n = C_{n+1}$

$$C_{n+1} = C_n + C_{n-1}$$

$$\begin{cases} C_{n+1} = C_n + V_n \\ V_{n+1} = C_n \end{cases}$$

$\hookrightarrow V_n = C_{n-1}$

Base cond^n. $C_1 = 21$ $V_1 = 5$

Analogy: $C \rightarrow$ blue ball : 21 shade

$V \rightarrow$ red ball : 5 shades

$n = 3$ all possible strings $= 26^3 = 17576$ aaa ... zzz

Q. at least 1 consecutive should be present

$= 17576$ — all string no consecutive vowels.

$n=1$ $n=2$ $n=3$

$C_1 = 21$ $C_2 = V_1 + C_1 = 21+5 = 26 \times 21$ $C_3 = V_2 + C_2$

$V_1 = 5$ $V_2 = C_1 = 21 \times 5$ $= 26 + 21$

$= 47$

$V_3 = C_2 = 26$

$n=1$ $n=2$

$C_1 = 21 = 1 \times 21$ $C_2 = (C_1 + V_1) \times 21$

$V_1 = 5 = 1 \times 5$ $V_2 = C_1 \times 5$

$$\left.\begin{aligned} V_{n+1} &= C_n \times 5 \\ C_{n+1} &= (C_n + V_n) \times 21 \end{aligned}\right\} \quad \begin{aligned} V_1 &= 5 \\ C_1 &= 21 \end{aligned}$$

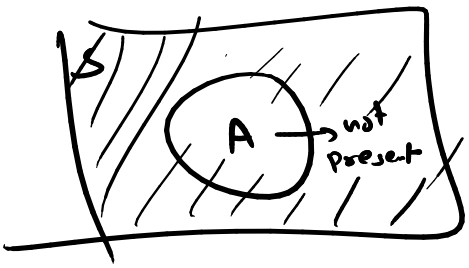
generic ans ⇒

$ans = C_n + V_n$

at least 2 vowels together once ↑

ans ⇒ = (S) − (A) ← everything, no vowels together

$26^n - C_n - V_n$



$n=2$ {RB BB BR} ~~RR~~

$n=1$ B $r_1 = 1$ $n=2$ $b_2 = r_1 + b_1 = 1+1 = 2$

R $b_1 = 1$ $r_2 = b_1 = 1$

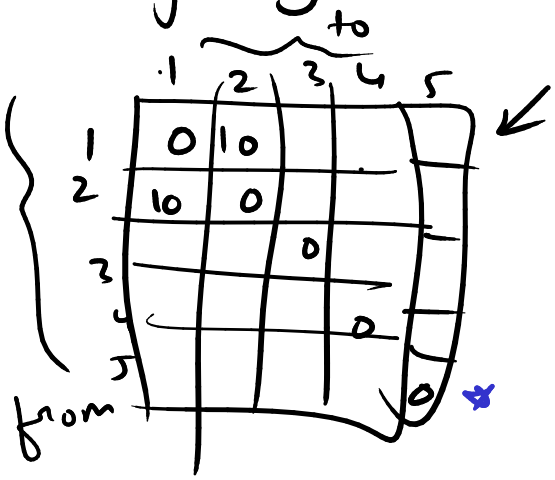
$ans = b_2 + r_2 = 2+1 = 3$

callback

pp.

Graphs. Weighted graphs. Representation. u: (v, w)

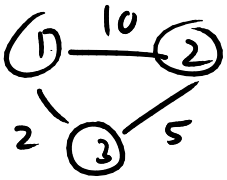
adjacency matrix



sense

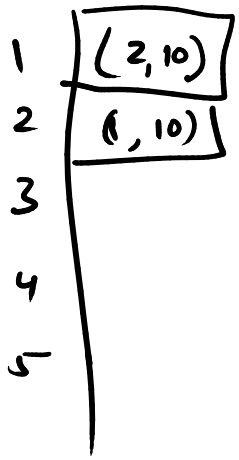
5 nodes.

airline

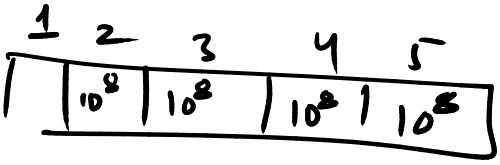


fine.

adjacent list.



BFS and DFS.



Dijkstra

Shortest path.

$$cost[v] = \min(cost[u] + g[u][v],\ cost[v])$$

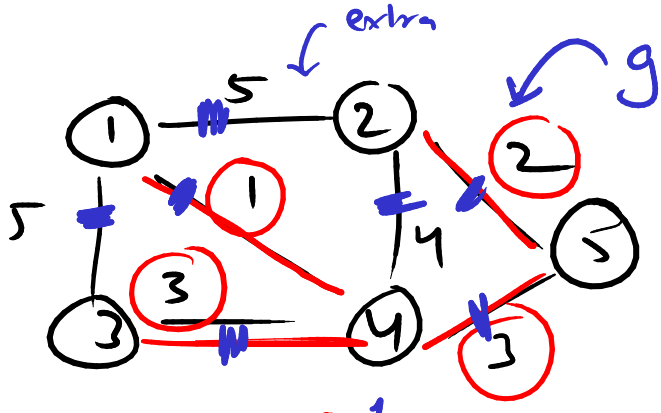
Task: edges | priority to edges with min weight.

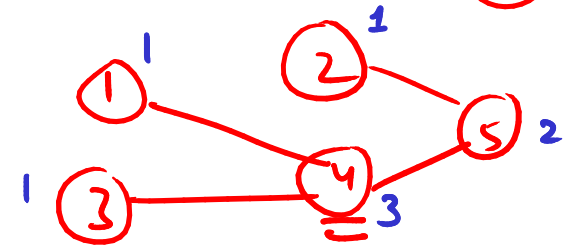
pick the cheapest edge first.

1-4
2-5
4-5
3-4

} edges

connected.





degree

① weighted undirected graph.

↳ min. connected graph | n nodes, n-1 edges → priority on the edge

MINIMUM SPANNING TREE

Σ edge weights ↓ edges sorted as per weight.

Naive represented : edge { int src
int dest
int weight
}

n vertices
e edges

only when ordering of edges is imp.

edge[e] edges ;

7 [[...]]

BFs and DFs ✗ 'u' → tell me 'v's ??

Fine: Read ?? Tricky.

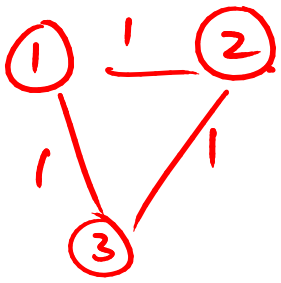
MST vs. Dijkstra

avg.

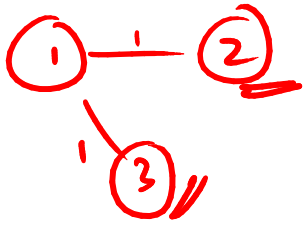
1. MST → greedy | minimize graph as a whole.

Dijkstra → shortest/min cost
A → B

DP



~



Q. n-1 edges in MST. Rest are discarded.

edges are imp shortest path

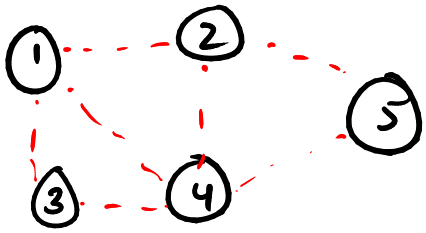
2 to 3 ⇒ cost 2

edges[ ] ⇒ all edges are considered discrete.

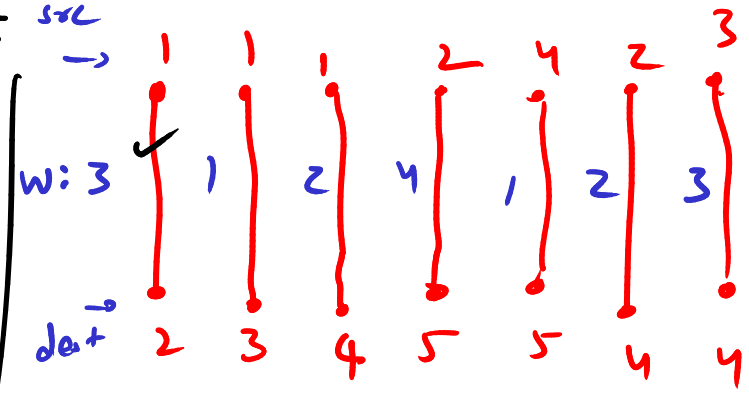
n nodes ⇒ nodes are discrete.

no neighborhood concept.

Image



set of nodes



Set of edges.

Connectivity.

sets of nodes. : {1} {2} {3} {4} {5}

parent/head of the set. {1,2} {3} {4} {5}

root.

node

index : 1 2 3 4 5 ← nodes

root : 2 2 2 3 5 ← head of the set

Initial cond^: every node belongs to its set.

add_edge (root, u, v)

root_of_u = root(u)

root_of_v = root(v)

root [root_of_v] = root_of_u

(3,4) (1,2) (5)

(2, 1)

2 1

root[1] = 2

1-3 Somehow ensure: 4 & 2 are automatically connected.

how ⇒ 2 and 4 are connected!

root[u] != u → its not a parent.

(1, 3)

1 3

root[3] = 2

(2, 4) → connected!

2 3

2

DISJOINT SET UNION

DSU

How to utilize DSU ?

Kruskal's algo.

1. edges → sort them based on edge weight.
2. root array → 1 to n ⇒ all set their own parent.
3. Traverse each edge in sequence:

```
if disconnected(u, v):
    root of u = root(u)
    root of v = root(v)
    root[root_of_u] = root_of_v
else: continue
```

n-1 times.