Recap: CRUD on graphs. 1. Create → adjacency matrix
→ adjacency list.

2. Read → BFS, DFS } both inspired by BFT, DFT

BFS

```
BFS (g, start):
  visited [n] = {false}
  levels [n] = {-1}
  parent [n] = {-1}
  queue<int> q ;
  i = 1 ;

  // process the start node
  visited [start] = true
  levels [start] = 0
  parent [start] = -1
   q. push (start)

while (!q.empty()):
  int u = q.top() ; q.pop();
  for all neighbours 'v' → g[u]:
    if !visited [v]:
      // process the neighbour
      visited [v] = true
      level [v] = i
      parent [v] = u
       q.push(v)
    i++
```

Usage of parent

```
dest = —
path = []
while (dest != -1):
  path.push(dest)
  dest = parent[dest]
path.reverse();
```

$O(n+e)$

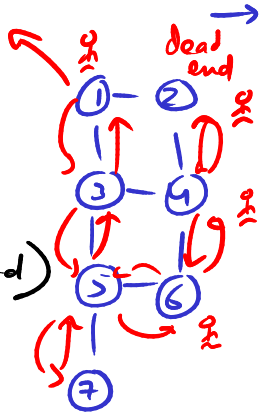
visit=[] **DFS** : disconnected graph. (use this the most).

```
DFS (g)
  n = g.size()
  visited [n] = {false}
  for i in range(n):
    if !visited [i]:
      DFSVisit (g, i, visited)
```

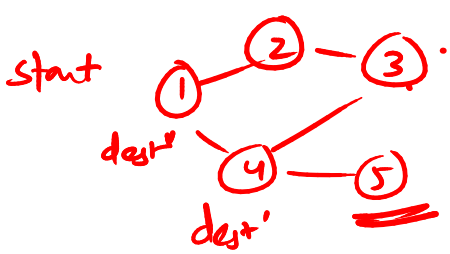
This will be called based on the number of components.

```
DFSVisit (g, u, visited):
  visited [u] = true
  cout << u << " ";
  for each neighbour v in
    g[u]:
      if ! visited [v]:
        DFSVisit (g, v,
        visited )
```

visit.push(u) →



1 3 5 7 6 4 2



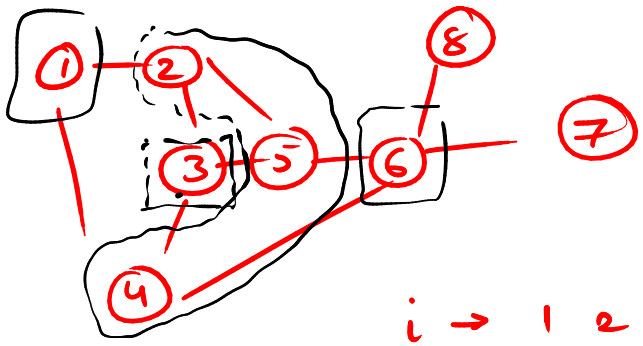
i	1	2	3	4	5
parent	-1	1	2	1	4

dest = parent[dest]

start=1 dest=5 path=[] 5 4 1

start = 3

dest^n:



i →	1	2	3	4	5	6	7	8
levels:	2	1	0	1	1	2	3	3
parent:	2	3	-1	3	3	5	6	6

Variations: (interview, X online assessment).

Q.1 Bombs → min no of bombs detonated → chain reactions | all N bombs burst.

x_i y_i r_i — n bombs — n^2 pairs

Q.2 Google Interview PRIME PATH

4 digit prime number : 2 of them.

P_1 : 1033

P_2 : 8179

one digit at a time and reach the dest^n.

Constraint: all intermediate numbers generated must be prime too!

P_1 1033 → 8033 → 8133 → 8173 → 8179

1033: P; 8033: ?; 8133: ?; 8173: ?; 8179: P

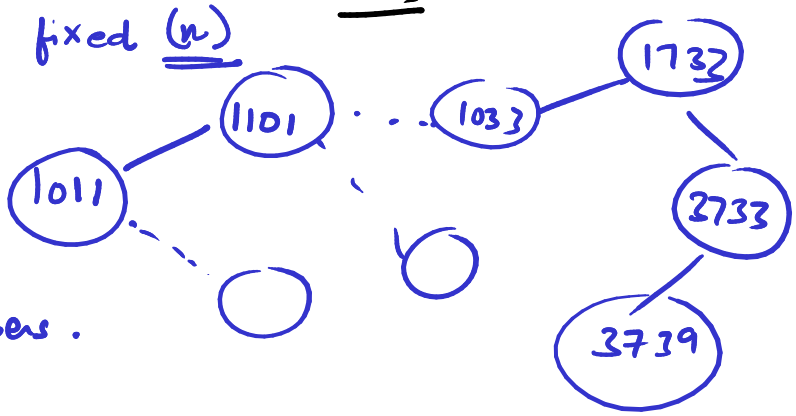
Difficult

Sol^n: 1033 → 1733 → 3733 → 3739 → 3779 → 8779 → 8179

Idea: 1: find all 4 digit prime numbers 1001 to 9999 ⇒ fixed (n)

graph of these nodes.

n^2 pairs



2: make a graph of n numbers.

3. BFS (g, start) dest^n. traverse back ⇒ path !!

Vector primes <int> = { } primes.push_back (p[i])

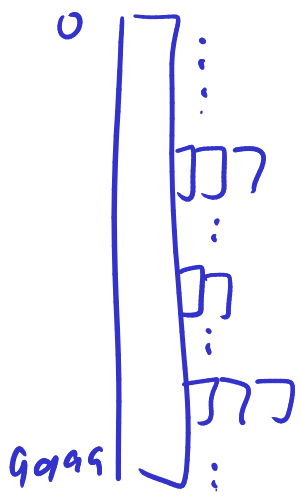
n = primes.size ( ) vector <vector <int>> g(n)

f^n. differ_by_one_digit (u, v)
return true / false

g[primes[i]] → x

g(9999)

if differ_by_one_digit (u,v):
g[u].push_back (v)
g[v] . push_back (u)



Q.3 chain the strings | google. word list

"sagar" "rashmi" "imran" "naresh"

"haris"

longest chain you can form.

sagar — rashmi — imran

haris — naresh

length of the longest cycle

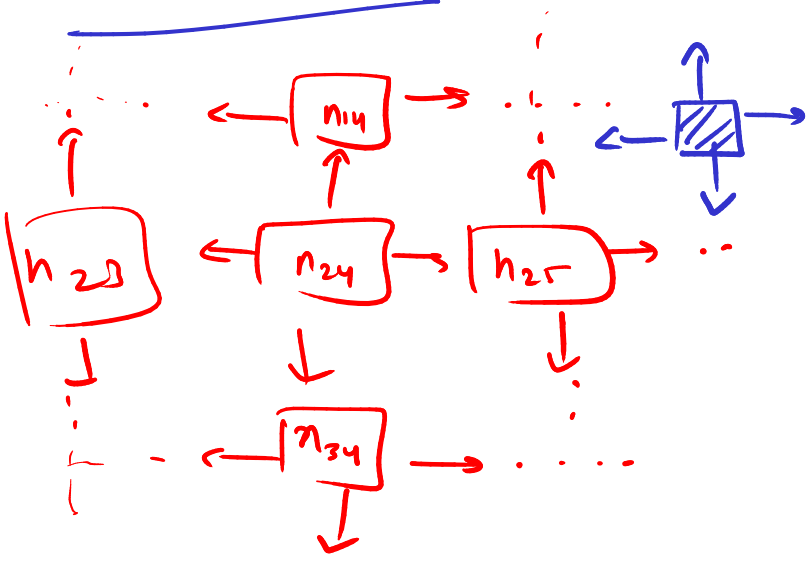
X online assessment

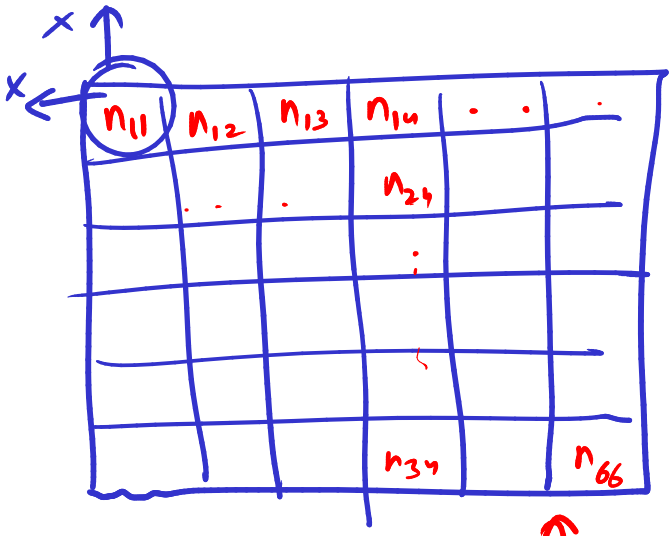
long challenges.

make a graph → x

how will I test BFS or DFS ?

"Grid as a graph"





graph g.

BFS. (grid)

```
BFS(g, x_s, y_s, x_d, y_d)
    visited[n][n] = {false}
    level[n][n] = {-1}
    parent[n][n] = {-1}
    i = 1
    queue < pair<int, int> > q;

// process of node.
visited[x_s][y_s] = true
level[x_s][y_s] = 0
parent[x_s][y_s] = -1
q.push({x_s, y_s})
```

all diag.

{00, 1-1, 1-1, 00}

```
// all neighbours
int dx[] = {1, 1, -1, -1}
int dy[] = {-1, 1, -1, 1}

while (!q.empty()):
    int u = q.front();
    q.pop();
cur_x = u.first
cur_y = u.second

for i = 0; i < 4; i++
    new_x = cur_x + dx[i]
    new_y = cur_y + dy[i]

if (!visited[new_x][new_y]
```

new_x and new_y are within boundaries) :

process the node

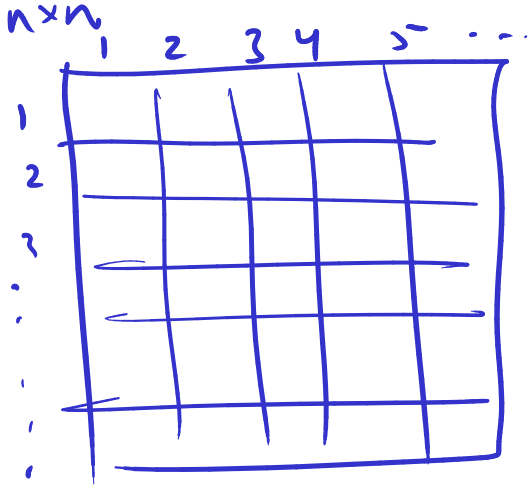
Q. Knight

```
int dx[] = {2, 2, -2, -2, 1, 1, -1, -1}
int dy[] = {1, -1, 1, -1, 2, -2, 2, -2}
```

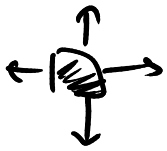
Amazon R2 Q.2

(1,2) → (5,8) 1-8 1-8

shortest moves to reach the destn.



DFS (grid). majority



```
DFS (grid, xs, ys, xd, yd, visited, path):
    if (xs and ys is !valid): return
    if (visited[xs][yc]): return
    if (xs ys == xd yd): print path  return
    visited[xs][ys] = true
    path.insert(xs ys)
```

Backtracking

if(g[i][j]==0) return

Flood Fill algorithm

Way 1

```
dx = [1, 1, -1, -1]
dy = [1, -1, 1, -1]
for i = 0; i < 4; i++:
    DFS(grid, xs+dx[i], ys+dy[i],
        xd, yd, visited, path)
    path.pop()
```

Way 2

```
DFS (g, xs+1, ys+1, ...
DFS (g, xs+1, ys-1 ...
DFS (g, xs-1, ys+1 ...
DFS (g, xs-1, ys-1 ...
```

2.

no. of islands
rat in a maze
word finder

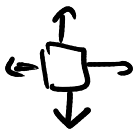
} classic

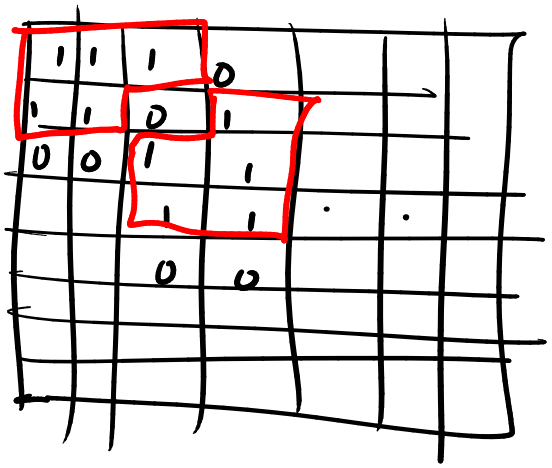
DFS

g[i][j] == 1 and !visited[i][j] :

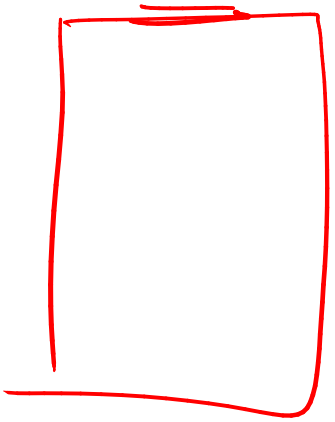
DFSVisit(g, i, j, visited)

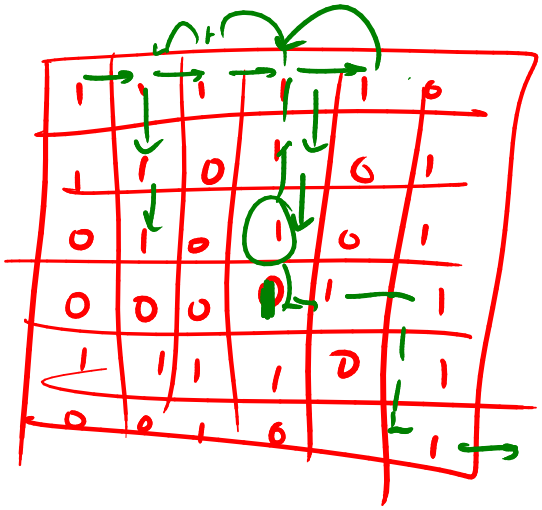
1 → land
0 → water



① 

visited



② 

Online assessment Amazon

1 → step on cell
0 → can't step

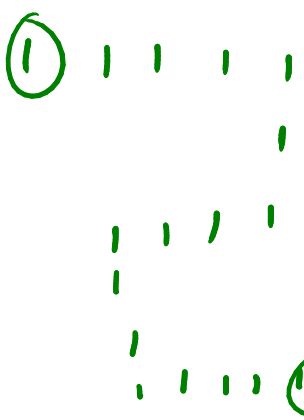
x_s y_s → 0, 0

x_d y_d → m, n

DP works

Graph

0,

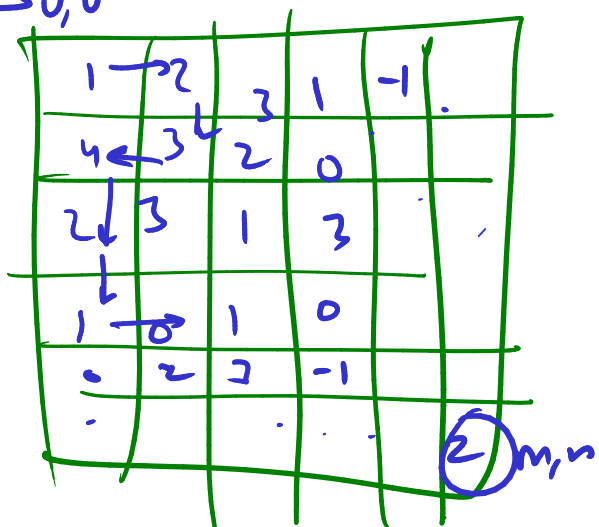


m, n

Online assessment Amazon Q.2

10

0, 0



m, n

g[i][j] → number | cost to step on the cell

-1 → can't step at all

initial money

3. Word finder

grid of chars.

c L e e t c o d e
o d e c h e f a
b g r e m a t
c x y b c d a

classmate

word if the word is present or not.

edocteel

DFS

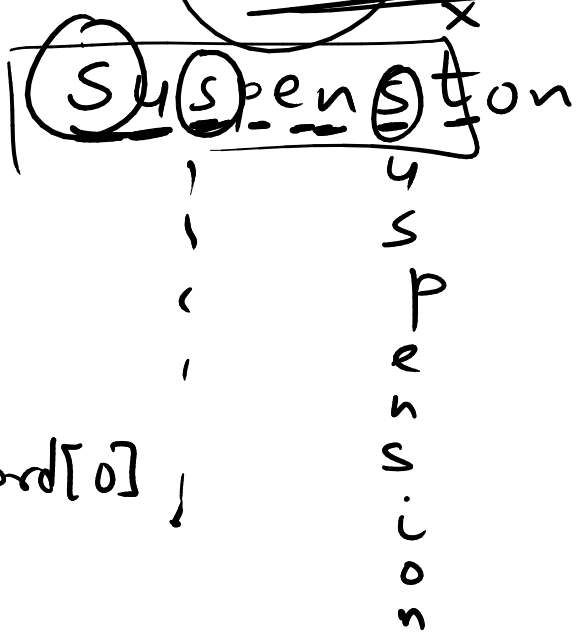
DFSvisit

g[i][j] = word[0]

DfsVisit(grd, i, j, ...)

reset visited[].

visited = true



O(nm)

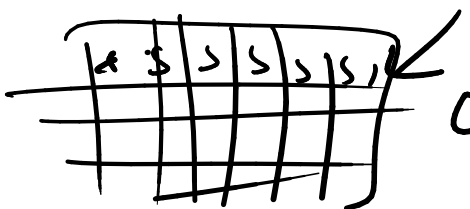
```
for i=0; i<n; i++
    j=0; j<m; j++
        if g[i][j] = word[0]
            DFS Visit
            reset visited
```

Worst

$O(n^2 m^2)$ → finite

nxm

DFSVisit



O(n·m)