Google Tips Before Interview:

- **Euler path (Edge once) - O(N) algo. visit all nei of Node, erase it from graph, then add Node to ans.**

- Always first think of similar Standard problem explicitly (if this problem can be deduced from that.)

-Revise all graph algos.

-Revise string hashing

-Tree BFS/DFS == DAG DFS + DP or Topological Sort + DP

-Konig's theorem :- In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover.

- Shortest and Longest Path in a weighted DAG can be found by DP using topological ordering, in O(V+E)

- In any ques(including Graphs path), if finding the min/max value, just think if the answer is monotone function, apply Binary Search. (Alternative - Greedy Djikstra but it may not be very intuitive to think)

Must Do - https://leetcode.com/problems/swim-in-rising-water/

3 Methods -

1. **Djikstra (NlogN)**
2. **Binary Search + DFS (NlogN)**
3. **Union Find (NlogN)**

If stuck in a Graph ques, cannot think of memoization or anything, it is most probably solvable one of the above 3 methods (Second is easy to implement and prove).

- In Graph ques, if revisiting of nodes is allowed -> Top Down DP. (DP is also required if it is DAG, just check visited or not. Other method for DAG -> BFS topological)
else if exact order matters -> Bitmask DP. O(2^N)
else if only relative ordering of adjacent nodes in path matters -> DFS from every starting node O(N^2)

-To optmize DP, think of Binary search on answer + Greedy.

-In binary tree questions consisting of finding a node in a last lvl, use binary search. Left->0, Right->1

-In Graph/Grid questions, involving visiting some nodes cells, always try to see if you can use DP or topological sort(onion peel) if it is DAG to find longest path. **MUST DO**
https://leetcode.com/problems/longest-increasing-path-in-a-matrix/

- If memory issue in graph algos like dfs or want a live algo like running edges stream is there, then DSU is best.
- In Graph problems finding minimum cost (kindof MST), try to introduce a new vertex (or vertices) and reduce the problem to just MST.
- In Grid problems, If changing the values of grid is allowed :-
  1. Think of changing the value as weight of edge(1 or more) and apply (0-1)BFS or Djikstra respectively.
  2. Binary Search for minimum changes.
  3. Atmost K obstacle removals are allowed, then do a BFS having remainingK as a state in nodes.

-Try to include a state in nodes and thus make K nodes for every original node.

- Hamilton Path in DAG is just the unique Longest Topological Sort of DAG (Onion Peeling)

- Euler Path - Hierholzer's Algorithm for directed graph (works for both)
https://www.geeksforgeeks.org/hierholzers-algorithm-directed-graph/

- Questions about finding a subset can be viewed as finding a subsequence after sorting (Some conditions might get relaxed). Eg. https://leetcode.com/problems/largest-divisible-subset/

-For Subarray problems, try thinking about valid subarrays ending at i, with start[i] or freq[i].

Q- https://leetcode.com/problems/find-the-minimum-number-of-fibonacci-numbers-whose-sum-is-k/ (onsite)
Soltn - First find all subarrays ending at i, then pick greedily activity selection problem. Pick by increasing i.

5 103 7201 3551 5789

05/22
020

1 5 8 12 15
0 2 13 14 20

(5 - 2)

0 1 2 5 8 12 13 14 15 20

10 -> (5, 6) 4 elements before
9 -> (5th) 4 elements before median.

abbdb ba
ab cba
abbdb bdbba
N + m
Ab ba **bdb**

```
L..r
L...L+i i+..r
N*N*M
0...........i j > i
```

```
Cost[0][0] = 0;
For relaxing cost[j][k]:
For i = 0 to j-1:
      Cost[j][k] = min(cost[i][k-1] + sum(i+1....j))
```

```
struct TrieNode {
 unordered_map<char, TrieNode*> children;
      vector<pair<string, int>> hotstrs;
      TrieNode* getNode() {
            TrieNode* node = new TrieNode();
            return node;
      }
};

class AutocompleteSystem {
      unordered_map<string, int> sentenceFreq;
      TrieNode* root;
      TrieNode* qroot;
      string querySentence;
      void initialize();
      bool compareHotstr(pair<string, int> s1, pair<string, int> s2);
      void updateHotstr(TieNode* node, string sentence);
      void insertTrie(string sentence);
public:
      AutocompleteSystem(vector<string> sentences, vector<int> times);
      vector<string> input(char c);
};

void initialize() {
      querySentence = "";
      qroot = root;
}

bool AutocompleteSystem:: compareHotstr(pair<string, int> s1, pair<string, int> s2) {
      if(s1.second != s2.second) {
            return s1.second < s2.second;
      }
      return s1.first > s2.first;
}
void AutocompleteSystem:: updateHotstr(TieNode* node, string sentence) {
      vector<pair<string, int>> &hotstrs = node->hotstrs;
      int countHotstr = hotstr.size();
      int freq = sentenceFreq[sentence];
      bool found = false;
      for(int i = 0; i < countHotstr; i++) {
            if(hotstrs[i].first == sentence) {
                  hotstrs[i].second = freq;
                  found = true;
                  break;
            }
      }
      if(found) return;
```

```
    if(countHotstr < 3) {
        hotstrs.push_back({sentence, freq});
    }
    else if(compareHotstr(hotstr[2], {sentence, freq}){
        hotstrs[2] = {sentence, freq};
    }
    sort(hostrs.begin(), hotstrs.end());
}
void insertTrie(string sentence) {
    TrieNode *curNode = root;
    for(char c: sentence) {
        if(curNode->chilren.find(c) == curNode->children.end()) {
            curNode->children[c] = getNode();
        }
        curNode = curNode->children[c];
        updateHotstr(curNode, sentence);
    }
}


AutocompleteSystem::AutocompleteSystem(vector<string> sentences, vector<int> times) {
    initialize();
    for(int i = 0; i < sentences.size(); i++) {
        sentenceFreq.insert({sentence, times});
        insertTrie(sentence);
    }
}

vector<string> AutocompleteSystem::input(char c) {
    querySentence.push_back(c);
    if(c == '#') {
        sentenceFreq[querySentence]++;
        insertTrie(querySentence);
        initialize();
        return vector<string>();
    }
    if(qroot->children.find(c) == qroot->children.end()) {
        qroot->children[c] = getNode();
    }
    qroot = qroot->children[c];
    return qroot->hotstrs;
}
```

```
int longestSubstringKDistinctChar(string s, int k) {
      int n = s.length();
      if(k == 0 || n == 0)
            return 0;
      int lastIndex[26], distinctCount = 0, start = 0, maxLen = 0;
      memset(lastIndex, -1, sizeof(lastIndex));
      for(int i = 0; i < n; i++) {
            if(lastIndex[s[i]-'a'] < start) {
                  distinctCount++;
            }
            while(distinctCount > k) {
                  if(lastIndex[s[start-'a']] == start) {
                        distinctCount--;
                  }
                  start++;
            }
            if(distinctCount == k) {
                  maxLen = max(maxLen, i - start + 1);
            }
            lastIndex[s[i]-'a'] = i;
      }
      return maxLen;
}

// abbacad 2
```

```
class Solution {
public:
      int numUniqueEmails(vector<string>& emails) {
            set<string> uniqueEmails;
            bool domainStarted = false;
            for(string &email: emails) {
                  int i = 0, j = 0;
                  while(j < email.length()) {
                        if(email[j] == '@') {
                              domainStarted = true;
                        }
                        if(!domainStarted && email[j]=='.') j++;
                        else if(!domainStarted && email[j]=='+') {
                              while(email[j] != '@') j++;
                        }
                        else email[i++] = email[j++];
                  }
                  uniqueEmails.insert(email.substr(i));
            }
            return uniqueEmails.size();
```

```
    }
};

class Solution {
      int findMaxLevel(TreeNode* root) {
            int maxLevel = 0;
            while(root) {
                  root = root->left;
                  maxLevel++;
            }
            return maxLevel-1;
      }

      bool nodePresent(int mid, int maxLevel, TreeNode* root) {
            TreeNode *cur = root;
            for(int i = maxLevel-1; i >= 0; i--) {
                  if((mid >> i)&1) {
                        cur = cur -> right;
                  }
                  else {
                        cur = cur -> left;
                  }
            }
            return cur != NULL;
      }
public:
      int countNodes(TreeNode* root) {
            int cntNodes = 0;
            int maxLevel = findMaxLevel(root);
            if(maxLevel == -1) {
                  return cntNodes;
            }
            cntNodes = (1<<maxLevel) - 1;
            int l = 0, r = (1<<maxLevel), mid;
            while(l < r) {
                  mid = l + (r - l)/2;
                  if(!nodePresent(mid, root)) {
                        r = mid;
                  }
                  else {
                        l = mid + 1;
                  }
            }
            return cntNodes + l;
    }
};
```

aaaazz

22 -> 4 times
4 -> 3 times

```
class Solution {
      vector<pair<int, int>> dir = {{0,-1}, {0,1}, {-1,0}, {1,0}};
public:
      int longestIncreasingPath(vector<vector<int>> &matrix) {
            int n = matrix.size(), m = matrix[0].size();
            int indegree[n][m];
            memset(indegree, 0, sizeof(indegree));
            auto isSafe = [&](int i, int j) {
                  if(i >= 0 && i < n && j >= 0 && j < m)
                        return true;
                  else
                        return false;
            };
            for(int i = 0; i < n; i++) {
                  for(int j = 0; j < m; j++) {
                        for(auto d: dir) {
                              int x = i + d.first, y = j + d.second;
                              if(isSafe(x, y) && matrix[x][y] < matrix[i][j]) {
                                    indegree[i][j]++;
                              }
                        }
                  }
            }
            queue<pair<int, int>> q;
            for(int i = 0; i < n; i++) {
                  for(int j = 0; j < m; j++) {
                        if(indegree[i][j] == 0) q.push({i, j});
                  }
            }
            int lis = 0;
            while(!q.empty()) {
                  int sz = q.size();
                  while(sz--) {
                        pair<int, int> u = q.front();
                        q.pop();
                        for(auto d: dir) {
                              int x = u.first + d.first, y = u.second + d.second;
                              if(isSafe(x, y)&& matrix[x][y]>matrix[u.first][u.second]){
                                    if(--indegre[x][y] == 0) q.push({x, y});
                        }
                  }
                  lis++;
            }
            return lis;
      }
};
```

`3[a2[c]]`

```
class Solution {
      string decodeStringRange(string &s, int left, int right) {
            string decodedString;
            int i = left;
            while(i <= right) {
                  if(isDigit(s[i]) {
                        int multiplier = 0;
                        while(isDigit[s[i]]) {
                              multiplier *= 10;
                              multiplier += (s[i]-'0');
                              i++;
                        }
                        decodedString += string(multiplier, decodeStringRange(i+1,
                        endingBrace[i]-1));
                        i = endingBrace[i]+1;
                  }
                  else {
                        decodedString.push_back(s[i++]);
                  }
            }
            return decodedString;
      }
public:
    string decodeString(string s) {
      int n = s.length();
      int endingBrace[n];
      memset(endingBrace, -1, sizeof(endingBrace));
      stack<int> st;
      for(int i = 0; i < n; i++) {
            if(s[i] =='[') {
                  st.push(i);
            }
            else {
                  endingBrace[st.top()] = i;
                  st.pop();
            }
      }

      string decodedString = decodeStringRange(0, n-1);
      return decodedString;
    }
};
```

```
a/b = x
b/c = y
c/d = z
a/d = p
a -> b
a -> d
a = b*x = cyx = dzyx => a/d = xyz
a/c = x*y

class Solution {
      double pathBFS(string src, string dest, unordered_map<string, vector<pair<string,
double>> &g) {
            if(src == dest) return 1.0;
            queue<pair<string, double>> q;
            set<string> vis;
            q.push({src, 1.0});
            vis.insert(src);
            while(!q.empty()) {
                  auto u = q.front();
                  q.pop();
                  if(u.first == dest) {
                        return u.second;
                  }
                  for(auto v: g[u]) {
                        if(vis.find(v) !=vis.end()) continue;
                        vis.insert(v.first);
                        q.push({v, u.second * v.second});
                  }
            }
            return -1.0;
      }
public:
    vector<double> calcEquation(vector<vector<string>>& equations, vector<double>&
values, vector<vector<string>>& queries) {
      int m = equations.size();
      unordered_map<string, vector<pair<string, double>> g;
      for(int i = 0; i < m; i++) {
            g[equations[0]].push_back({equations[1], values[i]};
            g[equations[1]].push_back({equations[0], 1.0/values[i]};
      }

      vector<double> ans;
      for(auto q: queries) {
            string u = q[0], v = q[1];
            double curAns = pathBFS(u, v, g, n);
            ans.push_back(curAns);
      }
    }
};

n = 2, k = 2
01
10
00
11

0->1 1->0 00 11

01


02

12

000..ntimes
1…..ntimes
.
.
.
k-1….ntimes
0222
n-1 times i -> n-1 times j
000111222333….k-1k-1….
k = 3
n = 2

t times i
00110220121


1 1 2 2 4

2 2 3 4


2 2 3 3 -5 -5
target = -3
l = 1, r = 5
{0, 2, 5}

1 2 3 4 7 5 8 6 0
1 2 3 4 7 6 8 5 0
1 2 3 4 7 6 0 5 8
```

Robot Room Cleaner

```
1110
1011
0010

1110 1210
0100 0100
0001 0000


      456
      123
      86310
      02190
      00654

      1368
      9120
     45600
class Solution {
      string addReverse(string multipliedNum, string curLayer) {
            if(multipliedNum.empty()) {
                  return curLayer;
            }
            int len1 = multipliedNum.length(), len2 = curLayer.length();
            int curDigit, carry = 0;
            string ans;
            for(int i = 0; i < max(len1, len2) || carry; i++) {
                  curDigit = i<len1?multipliedNum[i]-'0':0 + i<len2?curLayer[i]-'0':0 +
      carry;
                  carry = curDigit/10;
                  curDigit %= 10;
                  ans.push_back(curDigit + '0');
            }
            return ans;
      }
public:
    string multiply(string num1, string num2) {
      string multipliedNum;
      int len1 = num1.length(), len2 = num2.length();
      for(int i = len1-1; i>=0; i--) {
            string curLayer(len1-1-i, '0');
            int carry = 0, digit1 = num1[i] - '0', digit2, resDigit;
            for(int j = len2-1; j>=0; j--) {
                  digit2 = num2[j] - '0';
                  resDigit = digit1 * digit2 + carry;
                  carry = resDigit/10;
                  resDigit %= 10;
                  curLayer.push_back(resDigit + '0');
            }
            multipliedNum = addReverse(multipliedNum, curLayer);
      }

      reverse(multipliedNum.begin(), multipliedNum.end());
      return multipliedNum;
    }
};
```

```
a[i][j] -> a[j][i] -> a[j][m-i-1]
a[j][m-i-1] -> a[m-i-1][m-j-1]
a[m-i-1][m-j-1] -> a[m-j-1][i]
a[m-j-1][i] -> a[i][j]
```

```
  012
0 1230
1 4562
2 0789

  02

  741
  852
  963

1 30
10 20
23 24
25 40
```

find an event having start time less than cur start and having second maximum end time.
start[i] < end[cur] && end[i] > start[cur]

First give minimum wage to the minimum wage worker. Then,
Wage[i] = max((minWage[mnInd]/quality[minInd]) * qality[i], minWage[i])

wage -> high quality and less minimum wage

7 2.5 6
7 6 2.5
max wage/quality max wage with min quality
ith as first one, pick next k-1 having least quality.
sort the array by minWage/quality. If we pick Then subset of K-1 size with minimum sum(quality[i])   is the ans.

```
1 10, 2 1
1 <-> 2 cost = 6
3 10,   1 <-> 3 cost = 2
2 + 5 total cost
```

```
well <-> well
1     5     2
```

```
a -> b
b -> a
```

```
1 -> 1
6 -> 9
0 -> 0
8 -> 8
```

```
T F
First true - (l + r)/2
last true - (l + r + 1)/2
```

```
s 11341
g 23122
```

```
XLLL -> LXLL -> LLXL -> LLLX
XRXL
XLXR
XXLR
XLXR
```

```
0111111001
0110110111 -> 0110110001 2*2 = 4
0111110111
```

O(n^3)

```
   01234567
S: ssasbcdz
T: sbd
      0, 3, 6
      1, 3, 6
```

If we start at ith index of S, find minimum j such that S[i..j] contains T as subsequence.

```
O(lenS * lenT)

dp[i+1][j-1] = {startT, endT}
dp[i][j] = {startT - (S[i] == T[startT-1]), enT + (S[j] == T[endT+1])}
```

```
       0
      010
       0
```

Maximum of all subarray sums should be minimum. Binary search on Ans(maximum of sums of any subarray such that there are atmost k subarrays). Get the lowest such possible ans.

Minimum of all subarray sums should be maximum. Binary search on Ans(minimum of sums of any subarray such that there are atleast k subarrays). Get the largest such possible ans.

```
azypbazypcazypdazyp
a = 4, z = 4, y = 4, p = 4
b = 1, c = 1, d = 1
```

```
1 2 3 4 5
4 5 3 2 1

push 1 i = 1
push 2 i = 2
push 3 i = 3
push 4 pop 4
push 5
pop 5

0 1 2 3
4 7 9 10
0 3 5 6

0 1 2 3
Ai - A0 - i
Kth missing number = Find smallest i such that missing number before i are >=K.
Ans = A[i-1] + (K - (A[i-1] - A[0]))

1 2 4 5 6
1 2 3 4 5
1 2 3 4 7
1 2
4 5 6

1 2 3 3 4 4 4 5 5 5 5 6 6
5 -> 2
4 -> 1
End at 6. Reduce frequencies of 4 and 5 by frequency of 6.
End at next non zero frequency number.
```

```
4 5 6
4 5 6
2 3 4
```

```
update 1
update 2
update 3
snap 0
update 1
update 2
snap 1
```

```
0 1 2
1 3 1
6 0 3
3 3 3
```

```
0 1 1
1 0 1
1 1 0
```

In -1th virtual week, we are at 0th city.

ith city and jth week 2 Choices:-

i. Stay in this city for j+1th week, vacations += days[i][j+1] + vacations(i, j+1)

ii. Move to neighouring city c for j+1th week, vacations += days[c][j+1] + vacations(c, j+1)

```
dp[city][week]
```

max Flow of M

```
M = 1, X = 2
M = 2, X = 4
M = 4, X = 8
.
.
.
logN times
```

Min Flow of M

```
M = 1, X = 1
M = 1, X = 1
.
.
.
N times
```

i piles have been taken, so currently Alex/Lee is at ith pile. And value of M is curM.
So he has X = 1...2M options to take piles and pass on next M as max(M, X).
state of DP is currentIndex and curM.

```
1 -> 2 -> 3

1 -> 2 -> 3 -> 1

1 -> 2 -> 3

1 -> 2
1 -> 3

in[1] = 0
in[2] = 1
in[3] = 1
2 -> 3

abc def gh

abc
def
```

`[10,13,12,14,15]`

1. 10 -> 13
2. 13 -> 12
3. 12 -> 14
4. 14 -> __

1. 13 -> 15

1. 12 -> 14
2. 14 -> _

1. 14 -> 15

abcabc
2[abc]

abab
2[ab]

aaaa
4[a]

Do not encode strings of length <= 4

abcabcabcabca
4[abc]a
a4[bca]

aaaaabbbbbcccccaaaaabbbbbcccccaaaaabbbbbccccc

3[aaaaabbbbbccccc]
3[5[a]5[b]5[c]]

abcabc abcabcabcabc
ababab
aaaaaa

If pattern length is len, prefix function at every multiple of len(2len, 3len…) will be atleast len.
1
2, 4, 6, 8,

abpqrsab

I...J
choose k and split i..k and k+1...j or collapse the string i...j completely if possible.

1 0 1
1 1 1
1 1 1

fOR A ROW, N*N top corners, N bottom corners.
2 -> 1
3 -> +2

2 + 2 + 2 + 1 + 1 + 1

```
xyz
xzyxz
```

abcd

dcab

1+2+3+4+...K = n

K(K+1)/2 = n

5 1 2 3 4

0 6 7 8 9

ith index, we have two choices move ahead or swap. (If the operation is allowed)

To sort till ith index, :-
No swap: dp[i][0] = min(A[i] > A[i-1] && B[i] > B[i-1] + dp[i-1][0], A[i] > B[i-1] && B[i] > A[i-1] + dp[i-1][1])

Swap: dp[i][1] = min(B[i] > A[i-1] && A[i] > B[i-1] + dp[i-1][0], B[i] > B[i-1] && A[i] > A[i-1] + dp[i-1][1])

```
1 0 2 -> 1 0 0
0 1 1
```

```
// 7 -> 111
6       110
3       011
2       010
1       001
0       000
```

```
int numberOfSteps (int num) {
      int steps = 0;
      while(num) {
            if(num&1) num--;
            else num >>= 1;
            steps++;
      }
      return steps;
}
```

```
1st next to be NULL
2nd next to be NULL
O(N^2)
1 2 3 4
k = 4
1 NULL
2 3
3 4
4 NULL
```

```
ImmutableListNode* getKthNodeFromEnd(ImmutableListNode* head, int k) {
      ImmutableListNode *next = head, *prev = head;
      for(int i = 0; i < k; i++) {
            if(next == NULL) return NULL;
            next = next.getNext();
      }
      while(next) {
            next = next.getNext();
            prev = prev.getNext();
      }
      return prev;
}

void printLinkedListInReverse(ImmutableListNode* head) {
      int k = 1;
      while(true) {
            ImmutableListNode* curNode = getKthNodeFromEnd(head, k);
            if(curNode == NULL) break;
            cout << curNode.printValue();
            k++;
      }
}
```

Take lth pile and compute the sum for Lee for l+1...rth pile. Alex's score = pile[l] + (sum[l+1...r] - LeeScore(l+1….r))
Take lth pile and see if Lee can lose for pile[l+1...r].

```
class Solution {
      vector<int> prefixSums;
      int sum(int start, int end) {
            return prefixSums[end+1] - prefixSums[start];
      }
      int score(int start, int end, vector<int> &piles) {
            if(start == end) return piles[start];
            int curPlayerScore = max(pile[start] + sum(start+1, end) - score(start+1,
      end, piles), pile[end] + sum(start, end-1) - score(start, end-1, piles);
            return curPlayerScore;
      }
public:
    bool stoneGame(vector<int>& piles) {
      int n = piles.size();
      prefixSums.resize(n+1);
      for(int i = 0; i < n; i++) {
            prefixSums[i+1] = prefixSums[i] + piles[i];
      }
      int alexScore = willWin(0, n-1, piles);
      int leeScore = sum(0, n-1) - alexScore;
      return alexScore > leeScore;
    }
};
```

```
scoreDiff(start, end) :-
max Difference of Bob - Alice
Take 1 stone. stone[start] - (scoreDiff(start+1, end)
Take 2 stones. stone[start] + stone[start+1] - (scoreDiff(start+2, end)
Take 3 stone. stone[start] + stone[start+1] + stone[start+2]- (scoreDiff(start+3, end)
```

abcdeabe
ab
b

```
1 -> 2 -> 3 -> 4
|    |
---->|
```

aabaaaabaa
aaaaba
1- B is subtring of A - 1 repition
2- Rotate B such that it is a repition of A - k times K repitions with some prefix in the end or/ and some suffix in the beginning

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
(1,16) (2,15) (3,14) (4,13) (5,12) (6,11) (7,10) (8,9)
((1,16),(8,9)) ((2,15),(7,10)) ((3,14),(6,11)) ((4,13),(5,12))
(((1,16),(8,9)),((4,13),(5,12)))  (((2,15),(7,10)),((3,14),(6,11)))
((((1,16),(8,9)),((4,13),(5,12))),(((2,15),(7,10)),((3,14),(6,11))))
```

1. Block its parent. Then Blue nodes = Total nodes - Red node subtree size
2. Block its left child. Blue nodes = Left subtree size of red
3. Block its right child. Blue nodes = Right subtree size of red

r + l -> p
r + p -> l
p + l -> r

1 - 7
2 - 2

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

1 2
2
3
3 4
3 4 5

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

1
2*2 + 1
3*3
3*3+4*4
5*4 + 5*1 + 3*2

1 + 5 + 9 + 25 + 20 + 11

25
101
11001
1111101

```
x*5 = x*4 + 4

1010
1110
```

```
if(isSubtree(root1->left, root2) == 1) return 1;
if(isSubtree(root1->right, root2) == 1) return 1;
return -1;
```

```
pair<int,int> locationOfTargetValue(int rowCount, int columnCount, vector<vector<int>>&
matrix, int targetValue) {
        if (matrix.size() == 0)
            return false;
        int col = matrix[0].size();
        int row = matrix.size();
        bool result = false;
        pair<int, int> res = {-1,-1};
        for (int i = 0, j = col - 1; i < row && j >= 0 ; ) {
            if (matrix[i][j] == target) {
               result = true;
               res = {i, j};
               break;
            }
            else if (matrix[i][j] > target) {
                j = j - 1;
            } else {
                i = i + 1;
            }

        }
        return result;
    }
```

https://github.com/Nimishkhurana

Backend - Python Flask/Django REST, Node.js, MongoDb, PostgreSQL
Frontend - React.js
And some experience in Machine Learning and Deep Learning using Python Libraries like Numpy, Pandas, Matplotlib, and PyTorch, TensorFlow for Deep Learning.

```
10 7 6 9 4 5
10     9
7      6    5
         4
```

```
1 2 4 5     O(MN)
0 3 2 1
4 5 6 2
2 4 1 9

1 2 3 4 5 6
1 4 5 6 7
1 6 7


1 10 100 1000

1 10 -> 10
100 2 -> 100

10 100 -> 100
1 2 -> 102

1 100 -> 100
10 2 -> 110

100 200
100 -> 150

1 3 2
4 5 6

20C1 + 20C2 + 20c3 +

 0  1  2  3  4  5  6  7  8
[0, 0, 1, 1, 1, 1, 0, 1, 0]

[10, 20]

10 -> [20, 30]
20 -> [30, 40]
[0, 0] -> [2, 5]
[2, 5] -> [4, 10]
[4, 10] -> [6, 15]

14 16
8 + 4 + 2
16
5 -> 4
4 -> 3
3 -> 1
2 -> 5
```

```
p -> x
q -> y

100*100*(1<<10)

(p1 + 1)(p2 + 1)…(pn + 1)

0 2 3 1
0 1 2 3 4 5

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

0 1 2 3
0 3 1 2

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

5 5 5 5 10 20 5 2 3

0,-1
05,0
10,1
15,2
20,3
30,4  [0,4]
50,5  [4,5]
55,6
57,7
60,8  [5,8]

start[i] = index of starting position of subarray ending at i, with sum = K
[0,4][4,5][5,8]
Greedy activity selection problem.

11011
11101
10111
30213

N^3

0011100110
```

```
3 + 4 + 3 + 2 + 2
```

```
 0  1  2  3  4 5
[2, 1, 3, 4, 6, 5]
```

001
010
011
100
101
110
111

001
010
001
100
110
111
111

```
1      2      4      3      8      9
{0,1}  {2,3}  {6,7}  {10,9}  {18,17}{26,27}
```

3, 4, 1, 5, 6 and k=3
1 2 0 1 2 -> 1 3 3 4 6
2 1 0 2 1 -> 6 4 3 3 1

4 + 6 + 1

```
  b
 bwb
bwewb
 bwb
  b
```

Stack using queues
Reverse a string without special character.
Linkedlist
Circular queue

rear++
front++

```
bool isReachable(vector<vector<int>> &mat) {
      if(mat.empty()) return true;
      int n = mat.size(), m = mat[0].size();
      vector<vector<bool>> vis(n, vector<bool>(m, false));
      queue<pair<int, int>> q;
      q.push(make_pair(0, 0));
      vis[0][0] = true;
      int dx[4] = {0, 0, 1, -1};
      int dy[4] = {1, -1, 0, 0};
      while(!q.empty()) {
            auto cell = q.front();
            q.pop();
            int r = cell.first, c = cell.second;
            if(r == n-1 && c == m-1) return true;
            for(int i = 0; i < 4; i++) {
                  int newR = r + dx[i], newC = c + dy[i];
                  if(newR >= 0 && newR < n && newC >= 0 && newC < m && !vis[newR][newC]){
                        q.push(make_pair(newR, newC});
                        vis[newR][newC] = true;
                  }
            }
      }
      return false;
```

```
bool isReachable(vector<vector<int>> &mat) {
    if(mat.empty()) return true;
    int n = mat.size(), m = mat[0].size();
    vector<vector<bool>> vis(n, vector<bool>(m, false));
    queue<pair<int, int>> q;
    q.push(make_pair(0, 0));
    vis[0][0] = true;
    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};
    while(!q.empty()) {
        auto cell = q.front();
        q.pop();
        int r = cell.first, c = cell.second;
        if(r == n-1 && c == m-1) return true;
        for(int i = 0; i < 4; i++) {
            int newR = r + dx[i], newC = c + dy[i];
            if(newR >= 0 && newR < n && newC >= 0 && newC < m && !vis[newR][newC]){
                q.push(make_pair(newR, newC});
                vis[newR][newC] = true;
            }
        }
    }
    return false;
}
```

```
[1, 10] [5, 15] [20, 30] [25, 35] [26, 40]
```

```
  5  9
5     45
4     20
```

```
1.Left should be present in row
2.Top should be present in col.
3.Product of row = right.
```

```
abab
abab
```

```
1 2 3 First
1 3 2 First
2 1 3 First
2 3 1 First
3 1 2 First
3 2 1 First
```

```
10 20 350 10 20 350
```

```
absze
eztba
[0, N/2+1]
O(N^2)

1, 2, 3, 4
A - [4]
B - [1]
C - [2]
D - [3]

                     K
                  /     \
                  K/2   K/2
                  /\    /\
                  K/4
1 + 2 + ...logK times
1 + 2 + ...K times
K(K+1)/2 splits
A should be there K times.
1 should be there K-1 times.
2 should be there K-2 times.
4

ab
abcd
```

```
struct TrieNode {
      bool endOfWord;
      unordered_map<char, TrieNode*> children;

      TrieNode() {
            endOfWord = false;
            children = unordered_map<char, TrieNode*>();
      }
};

class StringPrefixCheck {
      TrieNode* root;

      StringPrefixCheck() {
            root = new TrieNode();
      }

      bool insertAndCheckPrefix(string word) {
            TrieNode* curNode = root;
            bool prefixFound = false, isPrefix = true;

            for(char c: word) {
                  if(!cur->children[c]) {
                        cur->children[c] = new TrieNode();
                        isPrefix = false;
                  }
                  cur = cur->children[c];
                  if(cur->endOfWord) {
                        prefixFound = true;
                  }
            }
            return prefixFound || isPrefix;
      }
};
```

For first non zero digit, x -> x-1 zeroes.
One at x-1th place.

For second non-zero digit, x ->

For every 0 added, len++, sum++
For every 1 added, len++, sum++

5
0,1,2,2

q 70 -> 70
2q 130 -> 140

Optimal -> Pick k-1 workers having less ratio of W/Q having minimum W at current workers wage.

```
0 -> 2      2*2
1 -> 3      3*3
2 -> 1
3 -> 2
```

degree - {1, 2, 3, 5, 5, 6, 6}
edges [
6 + max(degree[i] - isEdge(i, n))
2 vertices not having edge bw them with maximum degree sum

```
pair<Node*, Node*> roots = {NULL, NULL;
if(!root) return roots;
if(root->val > V) {
      auto lroots = dfs(root->left);
      root->left = lroots.right;
      roots.second = root;
      roots.first = lroots.first;
}
else {
      auto rroots = dfs(root->right);
      root->right = rroots.first;
      roots.second = rroot.second;
      roots.first = root;
}
return roots;
```

```
1 4 3 2

1->4->3->2

1 3 2 4

1 2 3

0101
00001111
01110001
01001101
01010101

00001111
00010111
00101011
01010101

4 4 6
2 2 4 4 4 5
2 3 4 5 6 7


3 3 3 3
1 2 3 4

9 9 9 9 9 10 11 12
7 8 9 10 11 12 13 14
5 6 7 8 9 10 11 12

2 1 0 1 2 2 2 2   11
4 3 2 1 0 0 0 0   10


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


1 1 2 3 3

1->1
1->2
2->3
3->4
4->5
```

```
1     2
1     1
1     1
1     1
1111111


10    40    50
5     27    51
15    35    20
18    53    25
30    65    10

5     27    51
15    35    20
10    40    50
18    53    25
30    65    10
```

Dunzo

1. Study manchester algorithm (for 3 palindromic substring ques)
2. Merge all the words into one, just store the frequency of every character at idx i
3. Now question deduces to a leetcode dp O(M*N) problem Ways to form target string

123
c = 3
m = 4

123
124
134
234

cnt, prev, idx
50. 50.100.50.100