(9) #973 K closest point to the Origin.

Given a list of coordinates return the k closest points.

Approach

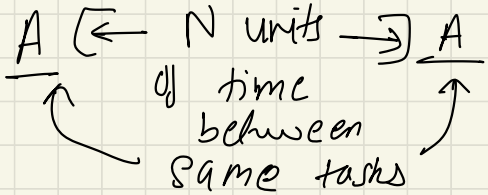
① We create a pq that is based on a min heap. (Returns the closest points first)

② The pq contains Point objects that have the property x, y, dist from origin.

```
class Solution {
    public int[][] kClosest(int[][] points, int k) {

        // let's create the pq
        PriorityQueue<Point> pq = new PriorityQueue<Point>((a,b) -> Double.compare(a.distance, b.distance) );

        // let's add all points to the PQ
        for(int[] point : points){
            pq.add(new Point(point[0], point[1]));
        }

        int[][] out = new int[k][];
        for(int i = 0; i < k; i++){
            out[i] = pq.poll().getCoord();
        }

        return out;


    }

    private class Point{
        public int x;
        public int y;
        public double distance;

        public Point(int x, int y){
            this.x = x;
            this.y = y;
            this.distance = Math.sqrt(Math.pow(x, 2)+Math.pow(y,2));
        }

        public int[] getCoord(){
            return new int[]{this.x, this.y};
        }
    }
}
```

(10) # 621 Task scheduler

- We have a task array [A, B, A, A, C, D, D, E...]
- Task can be done in any order.
- Each task :- 1 unit of time.
- For each unit CPU should do task or stay idle.
- N represents the cooldown period between 2 same tasks.



Return the least units of time the cpu can complete the task given N.

Eg tasks = [A, A, A, B, B, B]
n = 2.
⇒ we can't do 2 A's together.
or 2 B's together.

So we do [A → B → null → A → B → null → A, B]
minimized the nulls only 2

Eg 2 tasks = [A, B, A ....]

n = 0

⇒ No need to wait. Task can be done in any order.

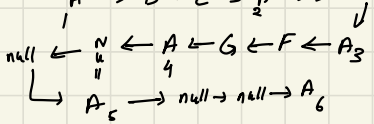
Eg 3:- [A, A, A, A, A, A, B, C, D, E, F, G]

n = 2

A	6	E	1
B	1	F	1
C	1	G	1
D	1		

We need a map to store all the different tasks.

If n was 0, we can take all tasks, but since it is 2, we skip it.



Iterate? have n in mind.

cycle through keys.

① Start cycle

if n=0, add same till available

else (n>0 & next available) go to next

else (n>0 & no next) add nulls = k

The above process is not that efficient.

Approach II

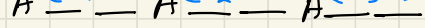
GREEDY Approach

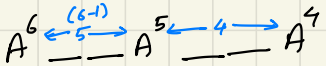
① Find the frequencies of all tasks

② Take the max frequency and find the max idle time using $\text{Idle Time}_{max} = (f_{max\,task} - 1) * \text{min idle time bet similar tasks.}$

Eg:- If A is the most frequent task

$f_A = 6$, idle min bet 2 same task = 2.





③ Now that we have spaced out the As, we can start filling the gaps with other tasks in descending freq order.

(we do this by sub tracting units of time from the max idle time calculated.

If i time count goes -ve, we make it 0.

④ Return (tasks + i time count)

Test case [A A A B B B]

n = 2

A _ _ A _ _ A

A B _ A B _ A

(fmax-1, f)

(2, 3)

min

In the prob if there are more than 1 tasks with the same frequency, then we need to cap the num of times we subtract from the idle time

Initially, we updated idle time as

$T_{idle} = (T_{idle} - freq_{Task_i})$

The problem here is that if we have another task that has same frequency as the most common task, then we will be forced to have 2 same tasks next to each other.

so we change the update function as

$T_{idle} = T_{idle} - \min[(f_{max} - 1), f_{task_i}]$

check the following test case for an example of why this works

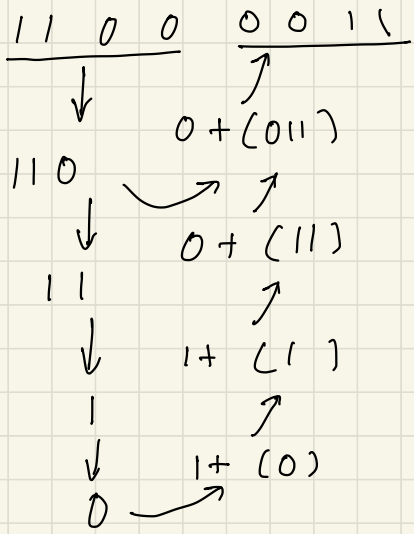
(11) Reversing the bits of an integer

(1) can be done recursively.

```
f(x) {
    1 2 3

    return 3 + f(12)
}
```

1 2 3

3 2 1



Clean implementation

① We need to return a 32 bit integer

Bit by bit Approach

- Iterate from right to left.
- Access the LSB using &.
- Shift it to the correct position.
- Since we need a 32 bit output but we perform left shift only 31 times start with Left shift 31 times and decrement this for every iteration.
- We keep accumulating this in an output number.

```
// you need treat n as an unsigned value
public int reverseBits(int n) {

    // 32 bit integer
    int result = 0;
    int leftShift = 31;

    // we start by left shifting the least significant bit 31 times and reduce as we encounter other digits.
    // when we encounter, the msb, we ideally should not left shift it

    while(n != 0){

        int lsb = n & 1;
        result = result | lsb << leftShift;
        leftShift--;

        n = n>>>1; // unsigned right shift // we don't want to add a 1 at the lsb when we perform the right shift
    }

    return result;
}
```

optimizing further using memoization

Note: As we loop through the bits, we need to perform an

Unsigned Right Shift

So that 1s dont get added to the left, which causes TLE

Optimization using Memoization.

Not very straight forward.

Approach #3 Using Bit masks and swapping the bits.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 30 31 32

Step 1

Break into 2 halves of 16 bits.

This is done by simple right and left shift.

1 2 3 .... 16 | 17 32

↳ >> 16 | ↳ << 16

17 18 ... 32 | 1 2 3 ... 16

Step 2

Break them into groups of 8 bits and perform left and right shift by 8 places

By end of step 2, the bits within the first and the second half would get swapped.

In hexadecimal

0 1 2 3 4 5 6
7 8 9 A B C D
E F

So to break it into chunks of 8 bits.

binary 1111 1111 0000 0000
hexa f f 0 0

∴ We use the mask 0Xff00ff00 >>8 || 0x00ff00ff <<8

|||ly

Step 3

0X f0 f0 f0 f0 >> 4
||
0x 0f 0f 0f 0f << 4

Step 4

Lets take the prev mask

Hexa 0X f0 f0 f0 f0

Binary 1111 0000 1111 0000 ....
f 0 f 0

We need. 8 4 2 1 00 11 00 11 00 11 00 ....

decimal 12

Hexa c c c c >> 2

Step 5

We need

8 2
1 0 1 0 1 0 1 0

10 10

a a >>1

and.

0 1 0 1 0 1 0 1

5 5 <<1

decimal & hexa

binary decimal Hexa

we need decimal

Hexa 0X 0f 0f 0f 0f

0000 1111
00 11 00 11

3 3 <<2

```java
public class Solution {
    // you need treat n as an unsigned value
    public int reverseBits(int n) {


        // Using masks switching
        // Since this is a 32 bit integer, we will use masks of 16 buts, 8 bits, 4 bits, 1 bits and 1 bit to sequentially swap the bits to their right spot
        // This is a divide and conquer technique

        // step 1 - moving across 16 bits at a time
        n = (n >>> 16) |  (n << 16);

        // step2 - moving 8 bits at a time
        n = ((n & 0xff00ff00) >>> 8) | ((n & 0x00ff00ff) << 8);

        // Step 3 - moving 4 bits at a time
        n =( (n & 0xf0f0f0f0) >>> 4) | ((n & 0x0f0f0f0f )<< 4);

        // Step 4 - moving 2 bits at a time
        n = ((n & 0xcccccccc) >>> 2) | ((n & 0x33333333) << 2) ;

        // Step 5 - moving 1 bit at a time
        n = ((n & 0xaaaaaaaa) >>> 1) | ((n & 0x55555555) << 1);

        return n;

    }


}
```

(12) #48 - Rotate image in place.

$$\begin{bmatrix}[1,2,3]\\[4,5,6]\\[7,8,9]\end{bmatrix} \Rightarrow \begin{bmatrix}[7,4,1]\\[8,5,2]\\[9,6,3]\end{bmatrix}$$

n = 3
m = 3

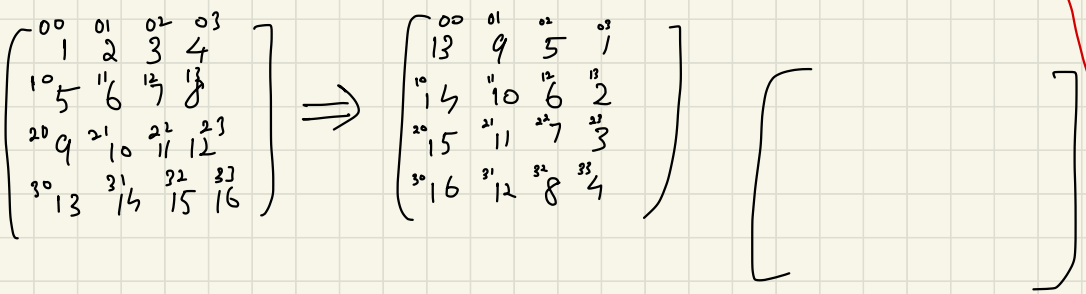
0,0	0,1	0,2
1	2	3
4	5	6
7	8	9

0,0	0,1	0,2
7	4	1
8	5	2
9	6	3

0,0 → 0,2
0,1 → 1,2
0,2 → 2,2

1,0 → 0,1
1,1 → 1,1
1,2 → 2,1

2,0 → 0,0
2,1 → 1,0
2,2 → 2,0



00 → 03
01 → 13
02 → 23
03 → 33

10 → 02
11 → 12
12 → 22
13 → 32

30 → 00
31 → 10
32 → 20
33 → 30

20 → 01
21 → 11
22 → 21
23 → 31

i, j → j, last index − i

3

3,3 → 3,0

2,1 → 1,1

Better approach is to follow the following steps:

① → Transpose the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

② Reverse each row

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \Longrightarrow \begin{bmatrix} 7 & 4 & 1 \\ 8 & 5 & 2 \\ 9 & 6 & 3 \end{bmatrix}$$

Rotated 90°.

Note:- While transposing using 2 for loops use the following.

```
for (i=0; i<len; i++):
    for (j=i+1; j<len; j++):
        temp = mat[i][j]
        mat[i][i] = mat[i][i]
        mat[j][i] = temp
```

Tik tok OA

(1) Re order from shortest to tallest

List < Integer >

Eg:- 11 5 4 21

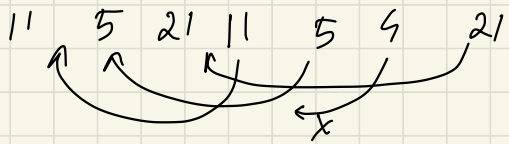
Logic Till we reach end of list.

(1) Iterate from start to find min.

11 5 (4) 21 — end i

Check for min till end

(2) Remove elements from end and move to start.

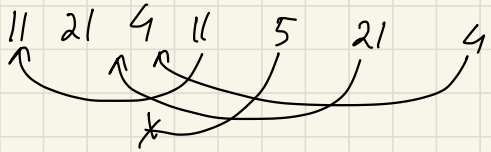


∴ ↳

11 (5) 21 4. — end i

check min till end - 1

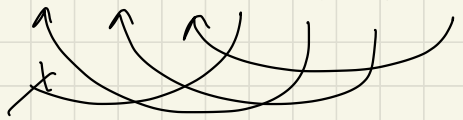
(3) except min move.



∴ ↳ (11) 21 4 5

check min till end - 2.

④ 21 4 5 11 21 4 5



∴ ↳

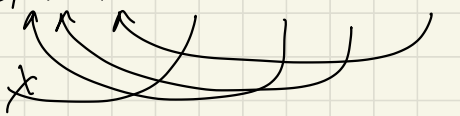
21 | 4 5 | 11

check for min till end-3

end i

(21) 4 5 11

⑤ 4 5 11 21 4 5 11



0 1 2 3

∴ ⟹ 4 5 11 21

end − 4

end − 4

3 − 4 = −1

i = 2

i = 3

5 10 12 20 22

0 1 2 3 4

22 5 10 12 20

i = 4

20 22 5 10 12

5 10 12 20 22

(5 10 20 22 12)

1 4 2 5 2

1 4 2 5 3 4 5

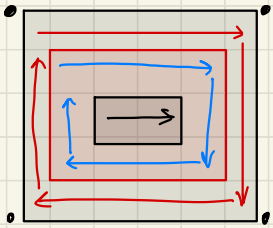
2

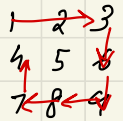
1 5 4 3 2

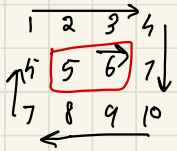
6 8 7 1 2 5 3 9 4

1 5 4 3 2

(13) Spiral matrix







① We set the bounds

top = 0

right = mat[0].len - 1

bottom = mat.len - 1

left = 0

② We set a variable visited. that has to be <= Count of elements.

③ While visited < total, keep visiting.

Use the bounds in the for loops.

moving → right.

① for col = left; col <= right; col ++

visit matrix [top] [col]

② moving down ↓ // we already visited top right corner

∴ Start from top++

for (row = top+1; r <= bottom; r ++

visit: matrix (row) [right];

④ Just like how we had a check for moving left. we check if visited < total while moving up ↑

we have already visited bottom left and top left.

for (row = bottom - 1; row > top; row --)

Visit mat [row] [left];

③ Moving left ←

// we already visited bottom right

// here we don't know if we have already visited all cells.

Eg:- This happens in the case when the spiral ends at the center row, it tries to go back left.

∴ ⊗ Add a check.

if (visited < total)

⇒ this row is still pending.

// we already visited bottom right.

∴ for. (col = right - 1; col >= left; col --)

visit matrix [bottom] [col]

⑤ top ++

right --

bottom --

right ++

after every loop.

Spiral matrix variation - Jump n cells

So instead of printing every cell, write a function that prints out every nth cell. BUT visit every cell.

---

(14) #57 Insert interval.

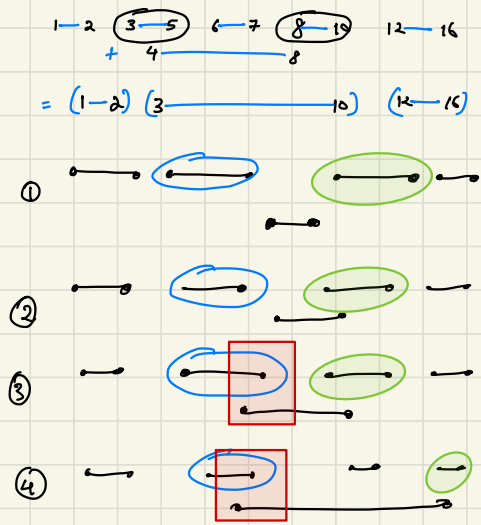
Q:- We are given intervals in ascending order based on start time.

∴ There is no mention if intervals overlap.

We are given another interval that we have to insert.

If it happens to overlap with the other intervals, merge it.

Eg:- 1,2 3,5 6,7 8,10 12,16 — New int 4,8.



S① Find start int ie. start$_i$ <= start$_{new}$.

S② See if start overlaps. ie:- start$_{new}$ ≤ end$_i$

then start$_{new}$ = start$_i$.

S② find end$_i$ ≥ end$_{new}$.

See if start$_i$ ≤ end$_{new}$

then end$_{new}$ = end$_i$.

Cleaner implementation

① Add all intervals till new interval.
② When adding new interval check if it has to be merged, if so merge it.
③ Keep adding other intervals and merge if required.

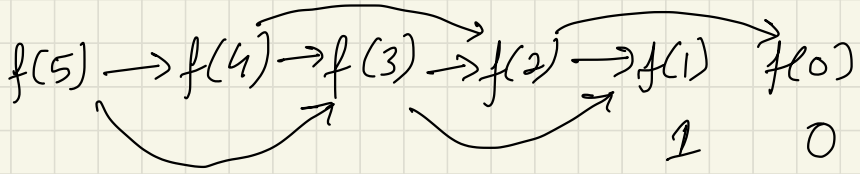
Use Linked List

It has methods like
get Last
remove last
that help with this implementation.

---

(15) # 509.

Fibonacci number with mem



① Create an array of size n. ie:- if fib(5) is asked, create an array with size 5+1 because we have fib series starting from f(0).

f(x):	f(0)	f(1)	f(2)	f(3)	f(4)	f(5)	f(6)	f(7)	...
value:	0	1	1	2	3	5	8	13	....
arr. len	1	2	3	4	5	6	7	8	....

② assign f(0) = 0 and f(1) = 1.
Illy return 0 and 1 when n is 0 or 1.

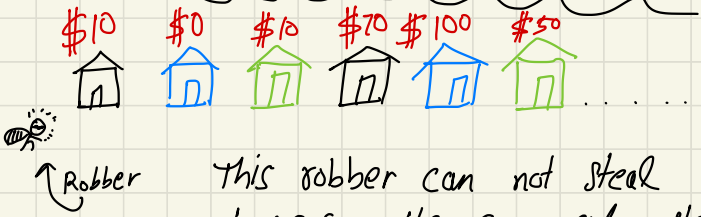
③ In every step, check if arr[n] has a value, if so return it, else calculate

arr[n] as fib(n-1, arr) + fib(n-2, arr);

④ Return arr[n] in every iteration.

---

⑯ House robber problem #198 LC.

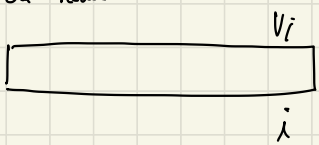


This robber can not steal from 2 adjacent houses. He can only steal from alternative houses.

Q:- Find the max money the robber can make.

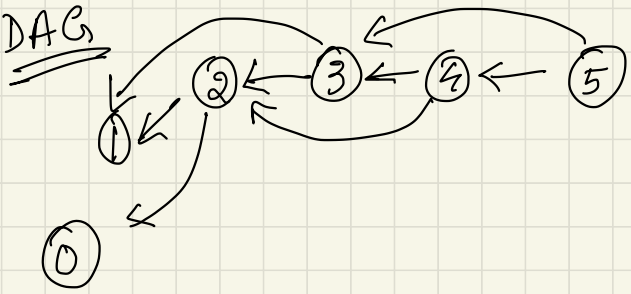
We know that if he steals from a given house he then cannot steal from the house next to it.

To convert this into a directed acyclic graph, we start seeing from the end house.



$$\text{Max Val} = \text{Max}\begin{cases} V_i + \max(i-2) \\ \text{or} \\ \max(i-1) \end{cases}$$

DAG



Eg:- [1, 2, 3, 1]

i

num	a	b
	0	0
	0	1
1	1	2

Code :-

a = 0 (holds i-2)

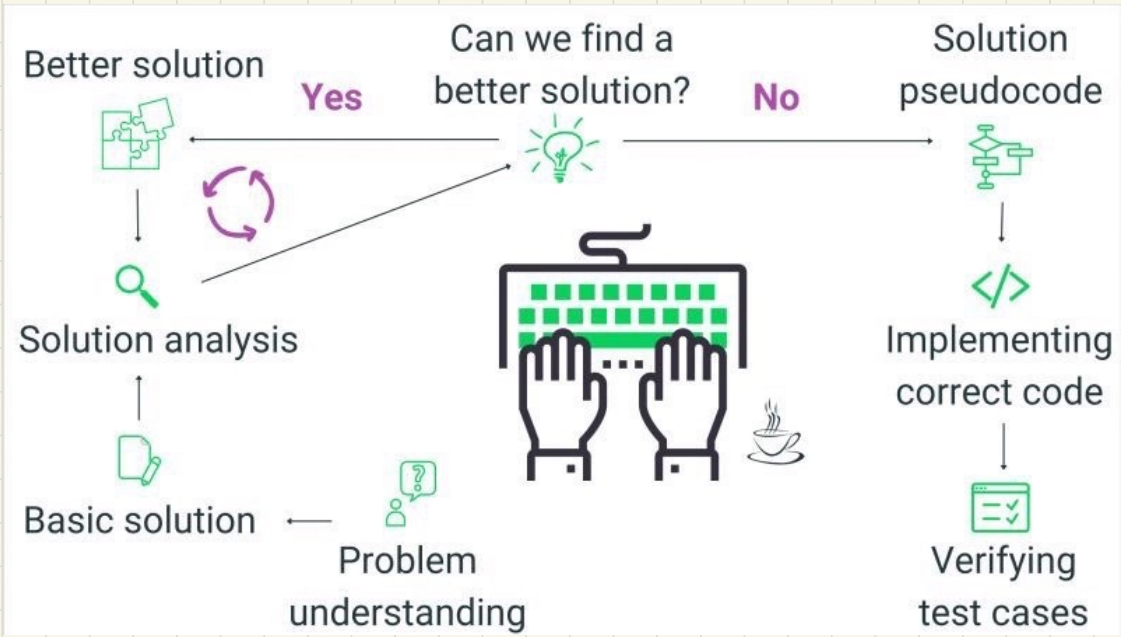
b = 0 (holds i-1)

```
for (num : nums) {
    temp = b
    b = max(num + a, b)
    a = temp
}
return b;
```

At every step we do the following

Max step = Max(this + step-2
or
step-1)

In python we can directly swap, in JAVA we will need a temp variable to solve this.



(17) # 53 MAXIMUM SUBARRAY

Q:- Given an array of nums, find the subarray with max sum and return the sum.

Eg:- [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Output: 6.

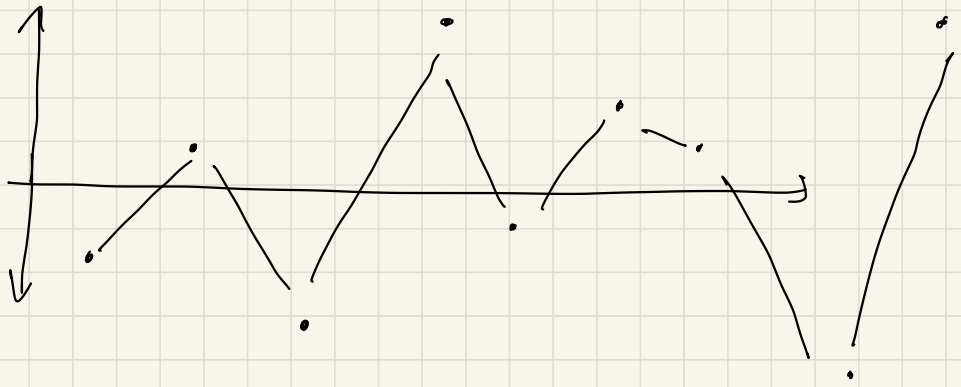
max sub array.

Basic solution

Using window method.

- Using 2 pointers.
- Start at 0 and keep expanding. ... till?

Window method does not seem to work.



① Brute force

$O(n)$

① Start with each element and find all sub arrays for that element ($O(n)$)

$\therefore O(n^2)$

② Dynamic programming approach.

Kadane's algorithm

- When asked for max or min of something, consider d/p.

KEY Problem

To find if a given -ve number is worthy enough to be kept in the sub array!!

Algorithm $O(n)$ $[-2, 1, -3, 4, -1, 2, 1, -5, 4]$

① Initialize 2 variables both at 0th index.

② 2 variables current sub array sum and max sub array sum

③ Start with 2nd element and iterate through the sub array.

For every number check

max (num or num + current_sum)

After updating current sum

max = max(max or current_sum)

	current	max	num
	-2	-2	-
① or -2+1	1	1	1
-3 or 1-3	-2	1	-3
4 or 2	4	4	4
-1 or 4-1	3	4	-1
2 or 3+2	5	5	2
1 or 5+1	6	6	1
-5 or 1	1	6	-5
4 or 5	5	6	4

---

(18) # 121

Best time to buy and sell stock

Q:- We are given an array of stock prices. We need to find the buy and sell time such that we get max profit.

→ All +ve numbers. → Min 1 price point.

Solution

① We need to keep track of max profit and current profit.

② This is similar to kadane's algo.

③ We keep a variable to track min value

④ We reset current profit when we find a new min.

122 Buy Sell Stock 2. Variation I

- Here we can buy and sell a stock multible times.
- Whenever we hit a new mining, we can sell and buy again
  - But we check if we are selling at a profit.

Note :- We can buy and sell on the same day. Interesting.!!

Basic Solution

① Keep track of min.
② Re compute profit every time
③ Have a variable for max profit.
④ Whenever we encounter a nother min, sell and buy.
⑤ Increment max profit when we sell.

```java
class Solution {
    public int maxProfit(int[] prices) {

    // this is similar to the stock problem, but here we can buy and sell stock multiple times
    // and we can hold one stock at a times
    // find max profit you can achieve this way

    // we try extracting the positive sums from the graph

    int buyPrice = prices[0];

    int currentProfit = 0;

    int totalProfit = 0;

    int prevSell = prices[0]; // previous selling price

    for(int i = 1; i < prices.length; i++){

        int price = prices[i];

       // if we encounter a prie point that is lesser than the previous sell point,
       // we sell and buy again
       if(price < prevSell){
           totalProfit += currentProfit; // update the total
           currentProfit = 0; // reset current profit
           buyPrice = price; // reset buy price
           prevSell = Integer.MIN_VALUE;
       } else if(price > buyPrice) {
           // update the current profit only if price is larger than the buy price
           currentProfit = price - buyPrice;
           prevSell = price;
       } else {
           // the subsequent price is lesser than the buy price, so we sell at the same price and buy again
           prevSell = price;
           buyPrice = price;

       }

    }

    return Math.max(totalProfit + currentProfit, 0);
    }
}
```

Variation 2 :- Simbler Approach

Since we need to get hold of all the upward trends, let's just capture those intervals where the arr[i+1] > arr[i]

This way we can accumulate all profits.

```
class Solution {
    public int maxProfit(int[] prices) {

        int profit  = 0;

        // we iterate throug the prices,
        for(int i = 0; i < prices.length -1; i ++){
            if(prices[i+1] > prices[i]){
                profit+= prices[i+1] - prices[i];
            }
        }

        return profit;
    }
}
```

Variation (5) #123

Key Question :- How do you find if a given transaction must be counted or not.?

You can at most perform 'n' transactions. Find max profit that can be obtained by doing at most n transactions.

The transactions should not be overlapping.

Examples :-

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

[1,2,3,4,5,4,2,0,10]

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

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

Let n = 1

[3,2,6,5,0,5]

4 (5)

Looks like ill need a pq. for this, to store the top k transaction-s.

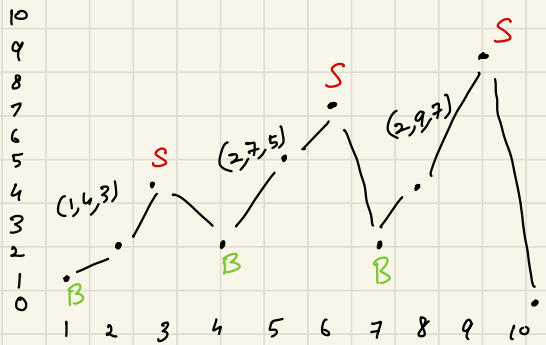
Trying to solve the above problem, we face a dilemma do we split the trades whenever we encounter a smaller next number? we can only do a limited amount of trades, so how do we decide if we want to exit the trade or hold it.

Logic to merge trades

To be visited later

Assume we can only return 2 trades out of 3, which have individual profit of 3, 5 and 7.

data :- [1, 2, 4, 2, 5, 7, 2, 4, 9, 0]

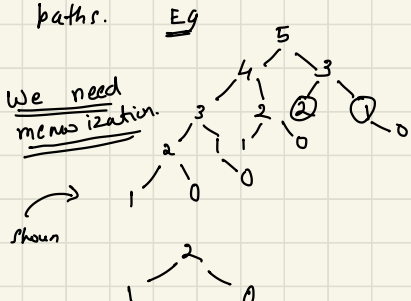


---

19 # 70 Climbing Stairs

Q:- We have n steps and at each step we can climb either 1 or 2 steps. Ways to reach to the top.?

Till we reach 0, we have to keep checking the various paths. Eg



We need memoization.

TC

① Brute force :- ① Recursion (without memo)

TC = $O(2^n)$

as we form a tree as shown

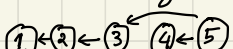
② Recursion with memoization

As we recurse from n all the way down to 1, 0 we will be storing the values in the memo array. ∴ TC is O(n) and SC is O(n)

③ Using DP

We know that this problem can be converted to a dag.

Eg



Ways to climb n steps = f(n-1) + f(n-2)

fibonacci series.

Ways	1	1	2	3	5	8	13
Steps	0	1	2	3	4	5	6

20 #746 Min cost of climbing stairs

- We have n stairs
- We have a cost of standing on each stair
- We can start at stair 1 or 2
- At each step we can take 1 or 2 steps.

Eg:- [10] [15] [20] | R

This is 111th to flower bot problem.

* Min 2 steps will be given.

Before we reach the top, we either will land on 15 or on 20.

- Cost to reach 0th step is 0
- Cost to reach 1st step is 0.

∴ Cost $_{min}$ Top = min(cost to reach t-1 + value of t-1 , cost to reach t-2 + value of t-2.)

```
// we know that the min cost to reach the top  is
// costMinTop = min(costMin[t-1] + valueT-1 , costMin[t-2] + valueT-2);

// let's construct an array that would hold these values
// and let's try the BOTTOM UP APPROACH

int[] dp = new int[cost.length];

// we know that the cost to reach the 0th and 1st index is always 0,
// as we can directly land there
for(int i = 2; i < cost.length; i++){
    dp[i] = Math.min(dp[i-1] + cost[i-1], dp[i-2] + cost[i-2]);
}

int top = cost.length;
return Math.min(dp[top-1] + cost[top-1], dp[top-2] + cost[top-2]);
```

Bottom UP Iteration

```
    // TOP DOWN APPROACH
    // we need a cost function that would give us the cost of reaching a particular step
    // we will need a memory array
    int[] memo = new int[cost.length+1]; // we need the cost to reach all steps saved

    Arrays.fill(memo, -1);

    // starting from the end
    int price = findCost(cost.length, cost, memo);

    return price;
}

public int findCost(int toReach,int[] val, int[] memArr){

    // if we reach 0 or 1 index, we return 0
    if(toReach == 0 || toReach == 1){
        return 0;
    }

    // check if we have it in the memo memo
    if(memArr[toReach] == -1){

        int nMinus1Cost = findCost(toReach-1, val, memArr) + val[toReach-1];
        int nMinus2Cost = findCost(toReach-2, val, memArr) + val[toReach-2];
        memArr[toReach] = Math.min(nMinus1Cost, nMinus2Cost);
    }

    return memArr[toReach];
}
```

Top down RECURSION

Optimization

Instead of using a dp array, we can use 2 variables. To store the cost to reach t-1 and cost to reach t-2.

```
a = 0, b = 0;          O(1) space
for( i : 2 → len+1 ):
    a, b = b, (min logic)      O(n) time
return b;
```

21 #198 House Robber I

Q: Array of houses. Cannot rob sub sequent houses. Find max loot.

Max loot = max → val t-1 + max (t-1) till
→ max till (t-2)

Eg:- 1 2 3 1 Loot

max till 1 ✓

max till 2 ✗

1 2 3 1

1	2	4	4

1 + 3 or
2 + 1 or

Loot

22 #253 Meeting Rooms II

Given a set of intervals, return the number of unique meeting rooms required to host all the meetings.

A given room can only have one meeting at any given instant.

Solution :- Rather than thinking from an interval stand point. We need to think from the stand point of rooms available.

We need a DS that would hold info about each room and the last meeting in that room. Also here we do not have a given set of rooms, we need to add rooms as and when needed.

We can use an Array List or a PQ for this purpose.

Every new interval that we encounter, we check which room can hold this meeting at the time when it is happening. Here we realize that the ability to access rooms information in sorted order will help. Hence we use a PQ to store the rooms data.

Every element in the PQ will be a single room with details about the last meeting that happened in that room.

In case the last meeting in that room overlaps with

the new meeting, then we add a new room to the pq; else we replace that meeting with this latest one.

Having a PQ data structure helps here because we can maintain the rooms based on which room had the meeting that ended earliest.

We have this because if we cant accomodate this new meet in that room, we can't do it in any other room. This way we can skip checking all rooms for the meeting end times.

---

(23) #45 Jump game II.

Given array of integers.
Return min num of jumps to reach last index.

Minimize the number of jumps.

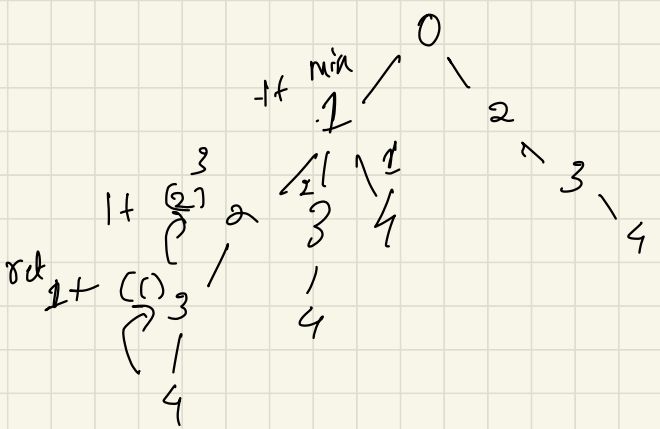
Brute force approach :- back tracking.

Eg:-

0	1	2	3	4
2	3	1	1	4
i				R

In the prev problem, we just had to reach end.

Now, we have to reach in min steps.



Optimization

1) Memoization to reduce re computation

2) Pruning the tree to reduce number of branches.
(Maintain a global minima. If any branch goes above the global minima, stop checking further).

Greedy

- At each step we know how much forward we can go.
- We check if any of the subsequent steps helps us go further.
  - if so we jump on the one that takes us the farthest.
- If any path can take us till the end, we are done with the problem.

Else : keep checking for other paths.

---

(24) #208. Implement Trie: Implemented using ideas mentioned in JAVA cheat sheet.

---

(25) Given an array of strings, find the number pairs of strings where both strings are equal or one string starts with the other.

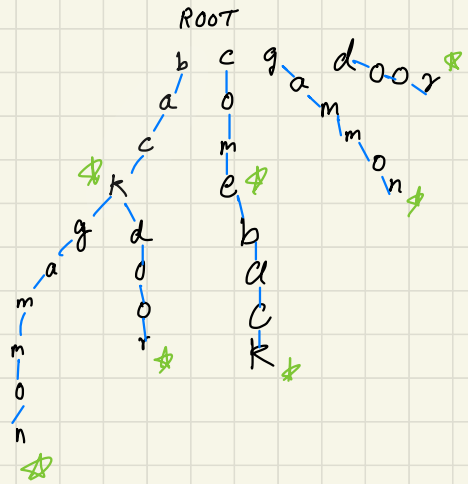
Brute force approach $O(n^2)$ TC

For each string, we iterate through the other strings and see if S_1 starts with S_2 or vice versa.

Optimization. O(n) TC

Using the trie Data Structure we can solve this in O(n) TC.

Eg:- ["back", "back door", "gammon", "back_gammon", "come back", "come", "door"]



For the above example, if we notice, we see that while adding the words to the trie 2 things happen.

- we traverse a path already taken
  or
- we form a new path.

Walk through

1) We add the word back to the trie.
   - we notice that we have not passed any node while adding this

2) We add the word "backgammon".
   - we notice that we pass through the word "back" in the process.
   ∴ We have found 1 pair.

3) We add the word "gammon":
   We did not pass through any end word.

4) We added the word "back door"
   - We passed through the word back.
   ∴ A new pair found!

5) We add the word "Come back". But we have not passed through any word yet.

6) "Door" → new word.

7) "Come" → We notice that we are passing through Come back and we are updating the node 'e' as the end node.
   ∴ New pair found !!!

abc a a b ab ac

1 abc, a
2 abc, a
3 abc; ab
4 a, a
5 a, ab
6 a, ac
8 a, ab
8 a, ac

abc a
abc ab
a ab
a ac
+
a ab
a ac
a a
a abc

double

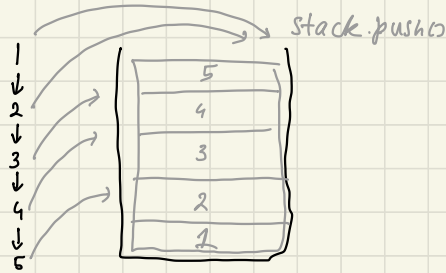
26 #206 Reversing a Linked List

Approach 1 : O(n) space
O(n) Time.

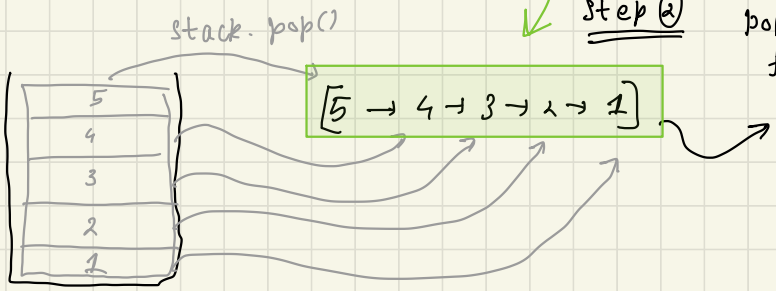
- Use a stack to store the nodes
- From the top of the stack we can access the nodes in reverse order.
- Create a new node and chain the nodes and return the head.

Eg:- [1 → 2 → 3 → 4 → 5]

Step ① Add to a stack.



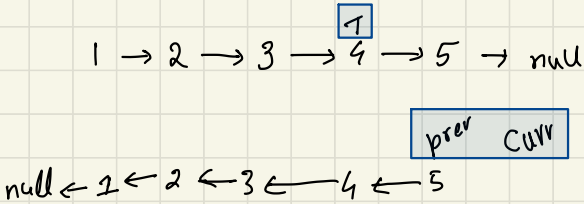
Step ②



pop from top and form a new list to return.

Approach 2 O(n) Time , O(1) space.

Using variables



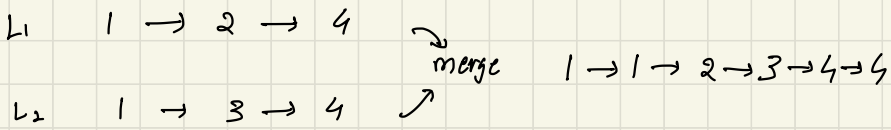
Logic

As we iterate through the nodes we save the next node in a temp variable and moving the next pointer to point to the node pointed by the prev variable.

At each step we move current and previous forward by a step.

Need to practice the recursive approach.

(27) #21 Merge 2 Sorted linked list



Concepts to practice

- If you want to attach the rest of a linked list to an existing list, we don't have to iterate over all the remaining nodes. Just do

  existing node . next = pointer to the start of rest of the list.

TC: O(n) linear

SC: O(1)

1 reference variable pointing to the start of the new list.

(28) #141 Check if the Linked list has a cycle

Approach 1 :- Using a hash map to keep track of visited nodes

- In every iteration, move to the next node and check if it is in the hash map. If so we are at the node that is already visited. ∴ we have a cycle.
- Else add this node to the hash map.

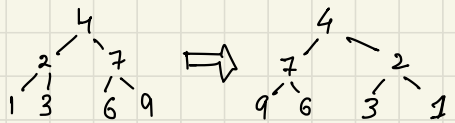
How are we sure that both the slow and fast pointer will coincide.
How does it work for different number of nodes in the list.

Approach 2 :- Using a slow and fast pointer Floyd's hare and tortise algo.

- Make the slow pointer move 1 step at a time and the fast 2 steps at a time
- When the slow p == fast p return.
- if fast P == null or fast . next == null, no cycle present.

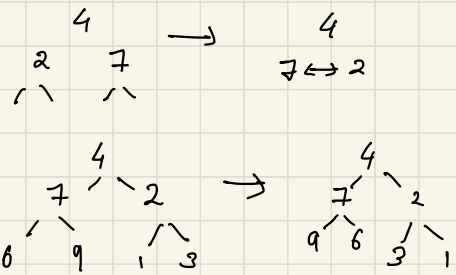
(29) # 226 Invert a binary tree

Eg:-

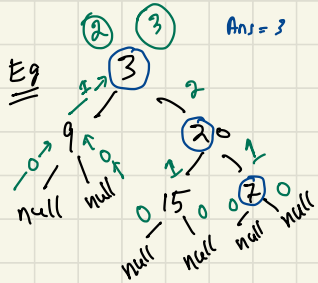


Flip the left and right sub trees.

Steps



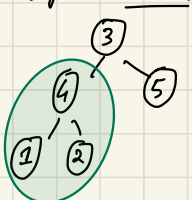
(30) # 104 Find max depth of a BT



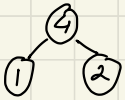
- Height of the tree is defined as the number of nodes that lie along the longest path from root to the leaf nodes.
- We solve it in an recursive way.
- If we encounter null nodes, we return 0.
- Once we find the max on the left and right we add 1 and return the value.
- We add 1 because we include the current node that is in focus.

(31) # 572 Subtree of another tree

Eg:- Tree is



sub Root is



We see that starting at node 4 on the main tree, we have it matching all nodes on the given sub root tree.

Steps ① if given node is null then we dont have anything more to check. ∴ Return null

② If root node has same value as sub root, check if subtrees are similar starting from root.

③ If found similar, return true, else recurse into the left and right subtrees of the tree and repeat from step 1.

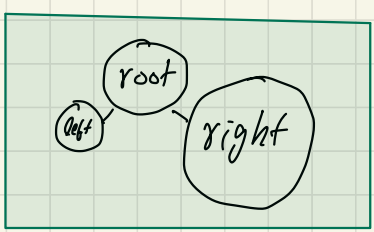
TC :- We might end up checking all trees. ∴ TC is O(n)
SC :- Since we use recursion we use O(n) space to store the stack.

---

(32) #235 Lowest common ancestor in a BST of 2 given nodes p & q.

In a BST we have the following properties

- Nodes on the left are lesser than root
- Nodes on the right are greater than root.

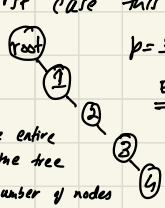


Using the mentioned properties, we can formulate the solution.

- Check is both p and q are < root traverse left.
- Check if both p & q are > right then traverse right
- if root == p or root == q return root
- if p < root > q or q < root > p, return root.

Time complexity : Since at every stage we will be moving either left or right, we are halving the number of nodes. Maximum we will have traversed is the height of the tree.

In the worst case this can be O(n)



p=3, q=4
Eg:-

Here we iterate the entire height of the tree ≈ total number of nodes

33 # 98 Validate a binary search tree

Return true at leaf node.

Constraint

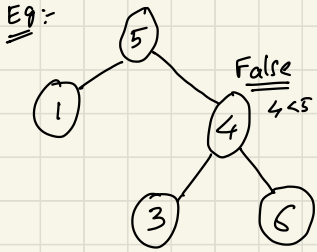
- we definitely have 1 node

To be a valid BST,

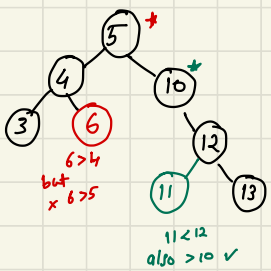
- Check nodes on the left sub tree,
  - left nodes < their parent
  - right nodes > their parent
  - but both left and right < parent's parent.

vice versa for right tree.

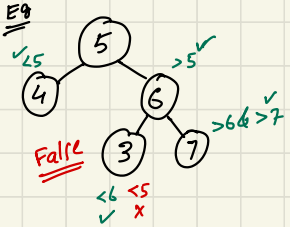
Eg:



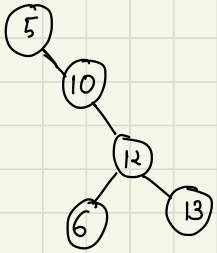
Eg



Eg



Eg It validates the 2 cases but violates 3rd case.



To validate a BST, we need to ensure that the tree is valid at all levels.

A valid BST has the following property

Left Bound	<	Left node	<	Root	<	Right node	<	Right bound.

Invariant ↗

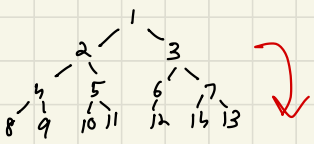
We update these bounds at every call in the tree and return true if any given node is null.

34 # 105 Construct a tree from given order

① Inorder
:- Left (ST) node right (ST).

② Post order.
node.
left right

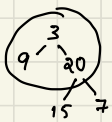
③ Pre order.
node, left, right



Eg Pre order

3 9 20, 15, 7

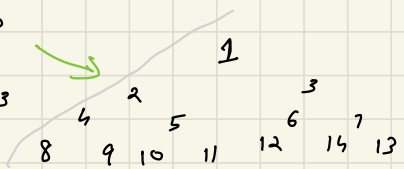
node, left, right.



Start with node and pass right down.

Inorder 8 4 9 2 10 5 11 1 12 6 3 14 7 13

Pre 1 2 4 8 9 5 10 11 3 6 12 7 14 13

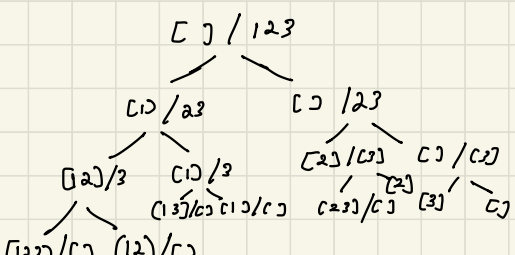


- The nodes in the pre order traversal are in the order of root nodes.
- The nodes in the in order traversal can be thought of as the top view of a BST.
  - The the in order traversal array can be split at the indices of the elements that appear in the pre order list in the same order.
  - The root node will be the node in the pre order traversal and while moving left and right, we update the left and right bounds.
  - These bounds tell us when we reach the leaf nodes and return null.

35 # 78. Sub sets

Q:- given an int array, return all possible subsets.

Eg:- [1,2,3] : [[1], [2], [3], [1,2], [1,3], [2,3], [1,2,3], []]



Function Signature :-

Initially an [] empty list

Index of next element to process.

```
genSubs ( List<Integer> list, int[] nums, int i) :
```

① `List<List<Integer>> out = new ArrayList<>()` // Format of solution for problem A (A*)

- Should match function return data structure
- We update this state of every iteration. Reducing entropy
- state of the model/ contains the final list of lists to be returned.

② `if (i == nums.length) return new ArrayList(list);` // Terminating condition for recursive c . Also (B*)

- i indicates which node to process next. if i == len ⇒ processed all indices already.

∴ Time to return back to parent.

③ Providing direction. (B)

include Node.

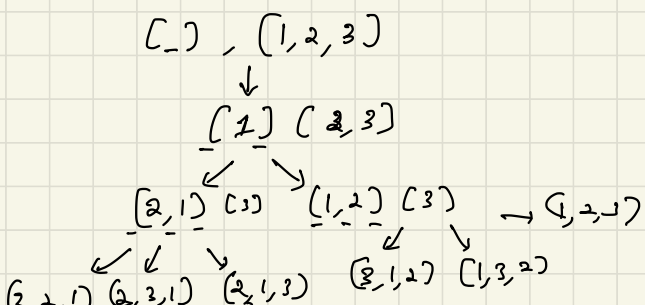
Processing subarrays without including current node

```
copy = new ArrayList(list); out.addAll( func( list, nums, i+1));
copy.add(nums[i]);
out.addAll( f( copy, nums, i+1));
```

④ By using out.addAll in the prev step, we perform the operation of combining answers from sub problems B back to an answer for problem A.

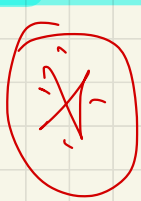
---

36 #46 Permutations



---

37 #90 Return all subsets without duplicate subsets.

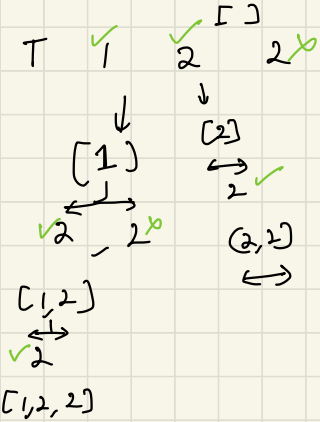


Eg:- [5,5,5,5]

[ ], [5], [5,5] [5,5,5], [5,5,5,5], [5,5,5,5]

[2], [1,2] [2,2], [1,2,2]

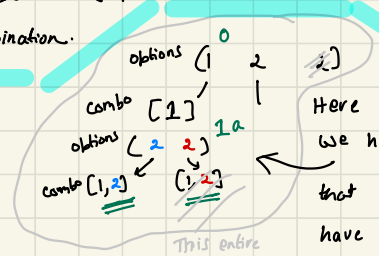
1, 2, 2 → [ ], [1], [2], [1,2], [2,2], [1,2,2]



[ ], [1], [1,2], [1,2,2], [2], [2,2]

- This is a bit tricky problem given that we need to avoid duplicates.
- One key here is to sort the array before we make use of it.

- We run this in a recursive manner where at every level we start with the start index pointing at the element to start adding to the existing combination.
- We also maintain an array of traversed elements so that we dont re compute the same set of combinations.



Here we see that at level 1a, since we had already calculated the combination that involved 1 and 2, we do not have to re compute it.

Illy at level 0, the second 2 is not required. If we use it, we will generate duplicate elements in the combo.

(38) #40 Combination Sum

Illy concept.

(39) #79

WORD SEARCH

- Backtracking over matrix traversal.
- Mark visited and non visited cells.

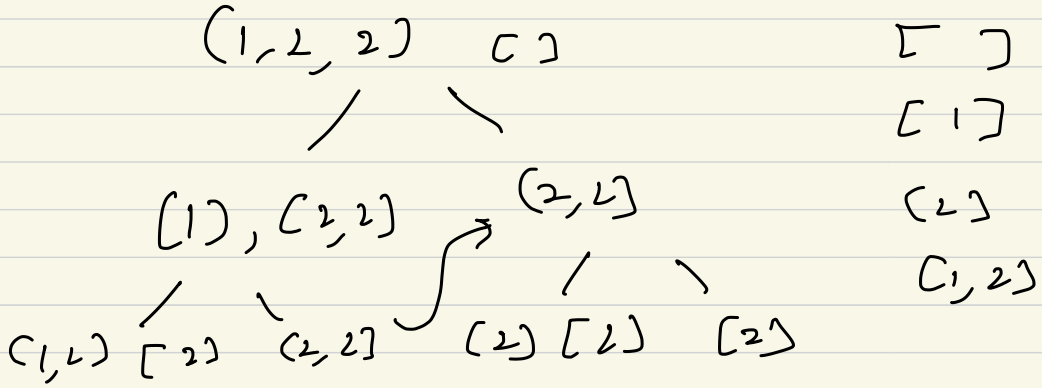
Approach :-

- Have a matrix to track visited cell positions
- Run search across matrix to find start of word.
- Call function to recursively check if we are able to continuously find subsequent characters of the word.
- mark each cell as visited as we traverse
- un mark as we return from any recursive call.

38 Extension. Return power set.

Eg:- [1,2,2] → ([ ], [1], [2], [1,2], [2,2], [1,2,2])

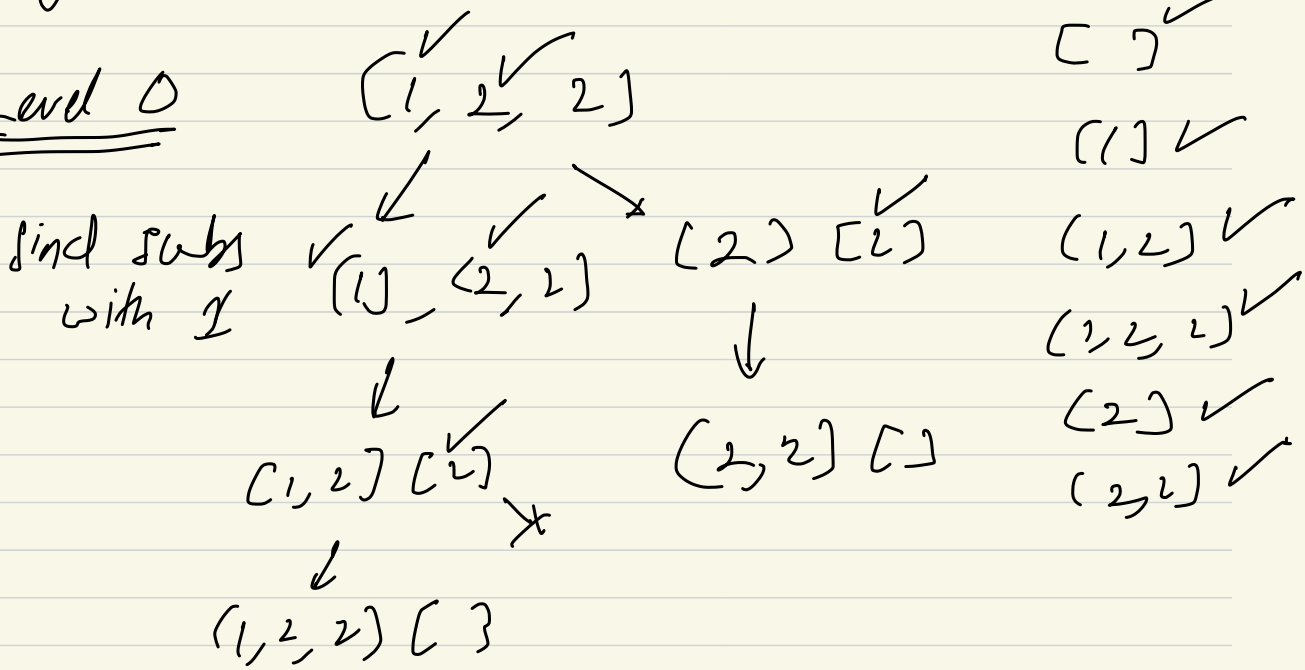
Check diagram and explanation on the right of #38 above.



[ ]
[1]
[2]
[1,2]

(X) Here we need to make sure that we do not perform an operation on an already tested number.

Level 0



[ ] ✓
[1] ✓
[1,2] ✓
[1,2,2] ✓
[2] ✓
[2,2] ✓

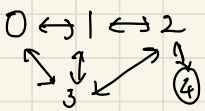
Logic is that at every level of the tree, maintain a traversed list that prevents us from recreating combos.

1615 (40) Maximal network rank.

G → Array of roads [[0,1], [0,3], [1,2], [1,3]]

Network rank of 2 different cities is the total number of directly connected roads to either city.

H (0,1) (0,3) (1,2) (1,3) (2,3) (2,4)



(0,1) (0,3) (1,2) (1,3) (2,3) (2,4)

num roads to city

0	1	2	3	4
2	3	3	3	1

(0,1) 4 ✓

road Exists

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

Note

- While filling the adjacency matrix for this problem mark both (c_1, c_2) and (c_2, c_1) as 1.

Solution :- 1. This involves using an adjacency matrix.

2. We use an array to keep an individual track of how many roads connect each city
3. Adjacency matrix helps us prevent double counting when we sum up the count of roads for each pair.

Solution pt 1

```
//number of road connected to city
int[] numRoadsConnectedCity = new int[100 + 1];

//road exist between two two cities
boolean[][] raadExist = new boolean[n][n];

for(int[] cities : roads){

    //increment the count of numbers of connected
    numRoadsConnectedCity[cities[0]]++;
    numRoadsConnectedCity[cities[1]]++;

    //mark road exist, between two cities
    raadExist[cities[0]][cities[1]] = true;
    raadExist[cities[1]][cities[0]] = true;
}
```

Solution Pt. 2.

```
int maxRank = 0;
for(int city1 = 0; city1 < n - 1; city1++){
    for(int city2 = city1 + 1; city2 < n; city2++){

        //count total number of road connected to both city
        int rank = numRoadsConnectedCity[city1] + numRoadsConnectedCity[city2];

        //just decrement the rank, if both city connected
        if(raadExist[city1][city2]) rank--;

        maxRank = Math.max(maxRank, rank);
    }
}

return maxRank;
```

(41) Consecutive letter deletion Minimum Cost.

Example

Input of size n

[a, a, a, a, b, b, b, ... d, d, d, d...]

[c1 c2 c3 c4 ..... cost n.

Cost array of size n.

You are given a string S. Deletion of the K-th letter of S costs C[K]. After deleting a letter, the costs of deleting other letters do not change. For example, for S = "ab" and C = [1, 3], after deleting 'a', deletion of 'b' will still cost 3.

You want to delete some letters from S to obtain a string without two identical letters next to each other. What is the minimum total cost of deletions to achieve such a string?

Write a function:

```
class Solution { public int solution(String S, int[] C); }
```

that, given string S and array C of integers, both of length N, returns the minimum cost of all necessary deletions.

Examples:

1. Given S = "abccbd" and C = [0, 1, 2, 3, 4, 5], the function should return 2. You can delete the first occurrence of 'c' to achieve "abcbd".
2. Given S = "aabbcc" and C = [1, 2, 1, 2, 1, 2], the function should return 3. By deleting all letters with a cost of 1, you can achieve string "abc".
3. Given S = "aaaa" and C = [3, 4, 5, 6], the function should return 12. You need to delete all but one letter 'a', and the lowest cost of deletions is 3+4+5=12.
4. Given S = "ababa" and C = [10, 5, 10, 5, 10], the function should return 0. There is no need to delete any letter.

Write an efficient algorithm for the following assumptions:

- string S and array C have length equal to N;
- N is an integer within the range [1..100,000];
- string S is made only of lowercase letters (a-z);
- each element of array C is an integer within the range [0..1,000].

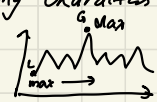
* Deleting any letter does not affect cost of deletion of any other letter.
* We need a letter array without 2 identical letters next to each other.

Approach O(n)

S1
- We break down the input into groups of repeating characters.

Step2
- Now the problem is among each group how do you identify which single character to keep.

Step3 If we look at each group. We have repeating characters with different cost.



- keep 2 pointers, global and local max.
- Initialize both to first char.
- If another char <= local max delete another else another > local max, delete local, set another to max.

This way we eliminate duplicate characters within groups.

(42) Find the intersection of 2 linked lists.

Brute force :- For each node of list 1, traverse entire list 2 to check if node is there $O(n^2)$

Optimized 1 :- Save all nodes of list 1 in hash map.
- Iterate list 2 and check if that in map.

Optimized 2. :- Iterate both lists and find length.
- Traverse the longer list for the Δ in length.
- Now compare both list node by node.

(43) #5 Longest Palindromic Substring

Brute force :-
- Find all sub strings,
- Find which all substrings are a palindrome
- Find the longest.

Optimized :- Expand around the center.
- Find all centers O(n)
- Expand around each center O(n)
- save current largest start ↔ end range
- Return substring in that range.

(44) #989 Add to array form of the integer.

Q:- Given an integer in array form Eg:- [1, 3, 2, 1]

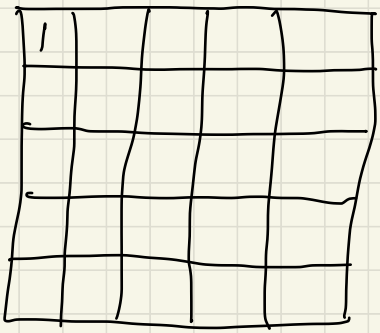
Given a value to add eg k = 31.
Return the output 1321 + 31 = 1352 in a list
ie :- [1, 3, 5, 2].

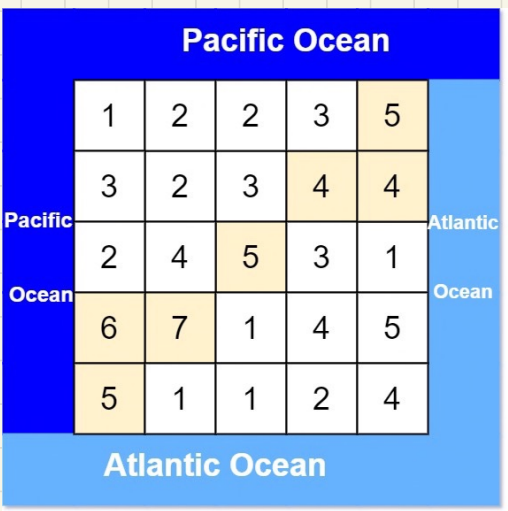
Approach :-
- Perform the operations as a list from the end index to start index.
- Return a reversed list at the end.
- Check all conditions in while loop.

```
class Solution {
    public List<Integer> addToArrayForm(int[] num, int k) {

        // we are going to add the digits in reverse order and then reverse it before
        // we submit the output
        // if we do not follow this method we will have to resort to using a linked List
        // which is too much work
        List<Integer> out = new ArrayList<Integer>();
        int carry = 0;
        // int the linked list we will be adding from the ones place
        int end = num.length - 1;
        // while we have numbers to add or carry present or digits left we proceed
        while(end>= 0 || carry > 0 || k > 0){
            int digit = end >= 0 ? num[end--] : 0; // if we still have unprocessed numbers in the array
            int add = k % 10; // from the k value
            k /= 10;
            int sum = digit + add + carry;
            out.add(sum%10); // we can only add the ones place of the number
            carry = sum / 10;
        }
        Collections.reverse(out);
        return out;
```

45 # 417 Pacific atlantic water flow.

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





Find cells from which water can flow to both oceans.

Brute force approach

Iterate through all cells and recurse to all directions and see for which cell we reach both oceans.

Optimized Approach

- We use a queue and a BFS algorithm to solve it in a cleaner way.
- Problems with our previous approach
  :- Messy logic - too much conditions to handle with respect to traversing through the matrix.

- This approach :- The problem :- Find nodes that can connect to both oceans.
- Entropy :- max.
- What do we already know? :- Nodes

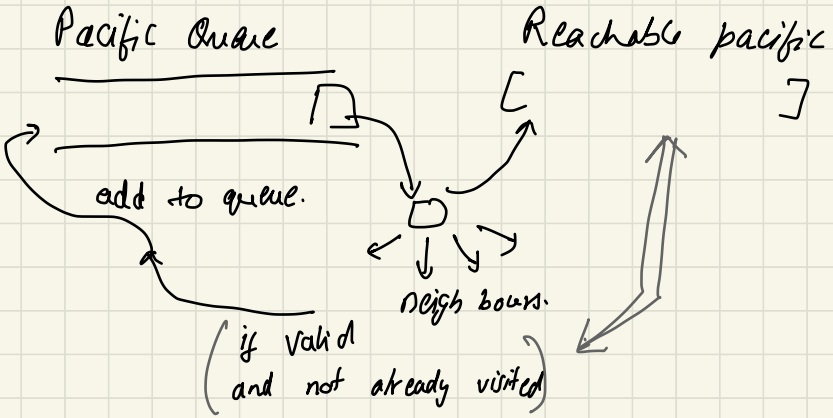
(1) that are touching either ocean. (Entropy ↓)

- From these nodes find connected nodes

(2) - Save this info in a 2D matrix. (E ↓)
- Can serve as check later.

(3) From each node in the queue, we find other connected nodes.

→ We keep populating the reachable array. (E ↓)



(4) Program ends when no more elements to check from queue.

Note We have transformed A to

B :- Find nodes connected to each ocean.

B* :- We have a 2D Matrix of nodes connected to each ocean.

A* :- Find nodes common in both matrices.

That is our answer.

Time complexity

	Time	Space
① Generating queue	$O(m+n)$	$O(m+n)$
② Performing BFS	$O(m \times n)$	
③ Performing final check	$O(m \times n)$	

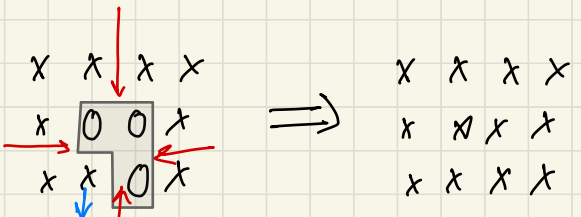
∴ TC is $O(m \times n)$ and SC is $O(m \times n)$.

---

46 #130 Surrounded regions

Q: Given a board of Xs and Os capture all regions of Os surrounded by X on all 4 sides.

Eg :-



X O X X X O X X

- The red region is surrounded on all 4 sides by X.
- While the blue region is only surrounded by 3 sides.

Point to note :- Any region that lies along the edges need not be captured.

1) →

(2) Similar to find number of islands.

- find islands that are not having an edge cell.

(3) Further breaking down the problem.

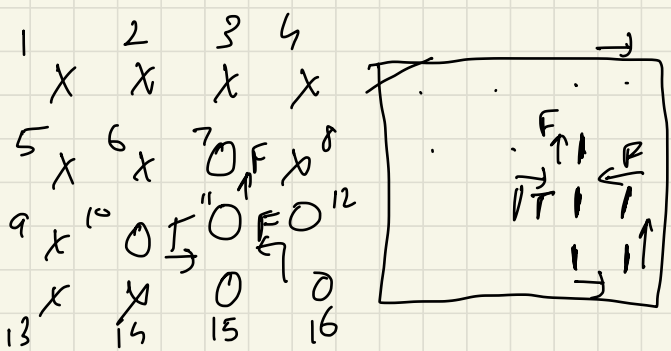
(B) Find islands ↓

how? → BFS

→ have visited cells for each BFS.

→ unmark while back tracking.

(B1) Mark islands ↓



We are facing in efficiencies in marking visited cells.

Let's revisit the finding islands problem.

(47) #200 Number of Islands

Q: Given a matrix of '1' and '0's return the number of Islands.

An Island is a connected set of 1s.

→ Getting back to our previous approach of first marking the border cells.

→ We then identify those cells that are 0 in the border and perform a search to see which other cell is connected that is also '0'.

→ We mark them as 'E'.

→ So now, we have ① 'X' cells as they were before.

② We have E cells, the ones that were touching the border, so we set them back to 'O' as they are not a valid island.

③ Then we have 'O' cells, that were not connected to any border. ∴ They now are changed to 'X'.

Mark border connected cells as E.

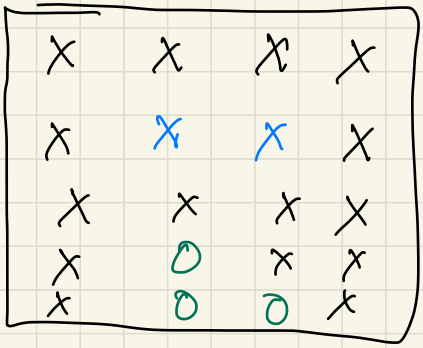
X	X	X	X
X	O	O	X
X	X	X	X
X	O	X	X
X	O	O	X

⟹

X	X	X	X
X	O	O	X
X	X	X	X
X	E	X	X
X	E	E	X

⟸

Mark O's as X
E's back to O.



⇒ Output

(47) Bounded array :- Pending

We have a min and a max value, find all sub arrays where their min and max is that value.

- All values in that sub array min <= val <= max.

[4 - 10]

p1 pr

[12, 13, 11, 1, 10, 6, 8, 9, 4, 6, 5, 10, 12, 58]

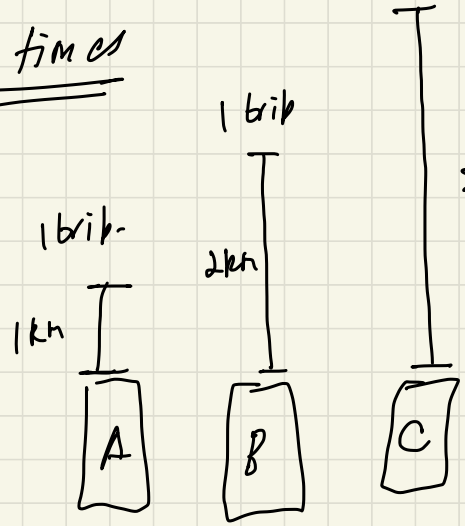
i j

min : 4

max = 10

(48) #2187 Min time to complete trips.

BW times



We need to make 5 trips.

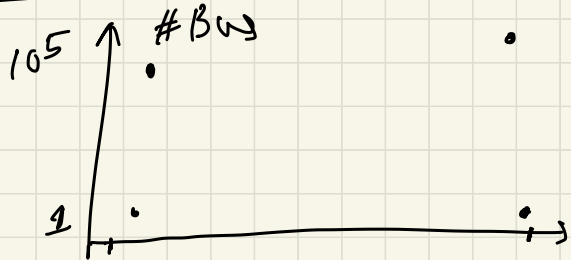
min time to complete 5 trips.

Use binary search to reduce search space.

Max total trips = 10^7.

min time to complete 10^2 trips.

What are the time values.



1 hr. Trip distance. Trips 10^7

Time taken

#	time/trip	Trips.	min
1	1	10^7	10^7
10^5	1	10^7	10^7
1	10^7	10^7	$10^7 \times 10^7$
10^5	10^7	10^7	$10^7 \times 10^7$

cont call

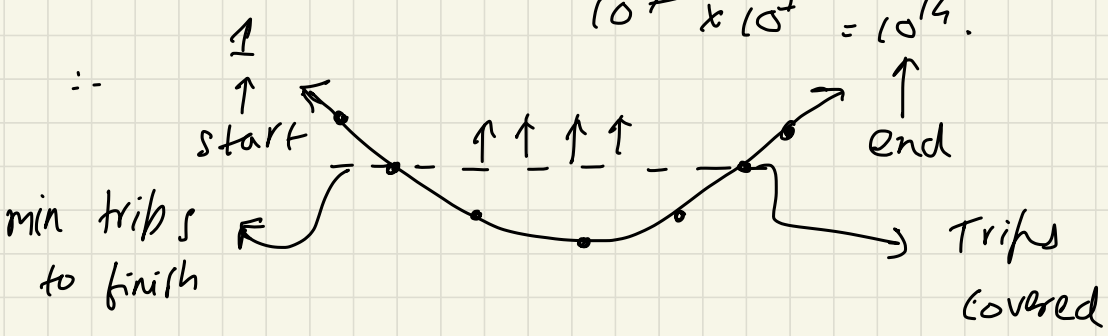
Best case 1 trip.

#	time/trip	Trips.	min time.
1	1	1	1
10^5	1	1	1
1	10^7	1	10^7
10^5	10^7	1	10^7

To solve the above problem, we employ binary search.

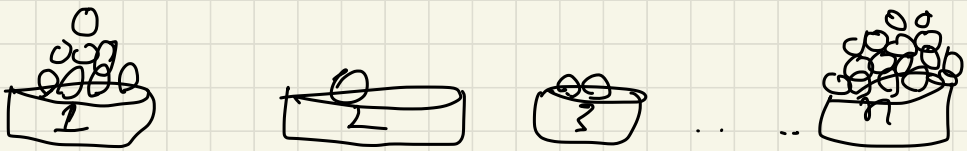
- We know that the best case is we have 1 bus and it takes 1 hr/trip and we have to complete 1 trip ∴ Time req = 1 unit.
- Worst case : we have 10^7 buses, each take 10^7 hours/trip. we need to complete 10^7 trips.

∴ Worst case min time req is $10^7 \times 10^7 = 10^{14}$.



∴ Using binary search to find which is the min value of time to cover the min # trips to be completed.

(49). #875 Koko eating bananas.



We have n piles of bananas.

Guard goes out for 'h' hours.

- Koko = eats k bananas from one pile every hour.

Eg 3 6 7 11 8 hours.

k: b/hr

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

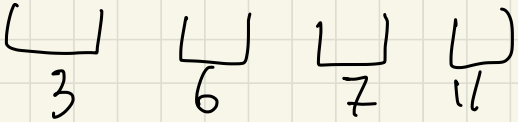
n
+7
18
± 6
24
+ 3
27

0 0 3 3

0 0 0 3 6

0 0 0 0 7

What happens if 4 cows k=3



8 C 3 = 24 but we need to finish

3 B/hr

0 6 7 11 1

0 3 7 11 2

0 0 7 11 3

0 0 4 11 4

0 0 1 11 5

0 0 0 11 6

0 0 0 8 7

8

So basically when we select a k value, we should have all bananas gone by the end of h^{th} hour.

So basically we need to calculate k such that $k \times h >=$ total bananas.

Best case — 1 pile, 1 banana.

1 hr. $\therefore k = 1$

$h = 1$

Worst case — 10^9 piles.

10^9 bananas each

$\therefore$ min time required :

total banana.

$1 \times k = (10^9 \times 10^9)$.

Lets assume $k = 3$

piles = [3 6 7 11] $h = 8$

hour 1 = 0 6 7 11

h2 = 0 3 7 11

h4 = 0 0 7 11

h5 = 0 0 4 11

h6 = 0 0 1 11

h7 = 0 0 0 8

h8 = 0 0 0 5 → pending

Total = 11 + 7 = 18 + 6 = 24 + 3 = 27.

∴ 3 is bad.

If speed is 3/hr and we have 8 hrs, we can only eat 24.

If they are unevenly distributed, if they are evenly distributed.

we can distribute the hours.

k = 3

3 6 7 11

1 2 3 4 ⇒ we need 10 hrs.

3 6 7 11 $k = 4$

1 2 2 8 ⟶ we need 8 hrs.

$\therefore$ 4 is min.

$3 < 4 \therefore 1$

$6 > 4$

$4 \times x > 6$

$x > 6/4$

$x > \lceil 1.5 \rceil$

$x = \lceil \frac{11}{4} \rceil = \lceil 2.7 \rceil$

(50) # 994 Rotting Oranges

m x n matrix

0 → empty cell

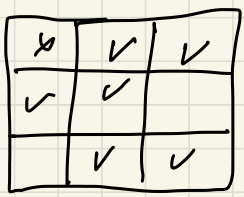
1 → fresh orange

2 → Rotten orange.

Every min, any fresh orange in contact with a rotten orange is going to become rotten.

Return min sec that there are no fresh oranges. Return -1 if there will still be fresh oranges at the end.

Eg



→ here we see that the (✓)s are connected to (x)s. Hence they all will get rotten.

→ Here we see that the green (✓) is not connected to the (X) hence it wont rot.

(2) Finding min time.

We start with the rotten orange at t_0.

- At t_1 → we get all its neighbours rotten.
- t_2 → we get all the rotten neighbours of the prev neighbours.

- So we have a queue of fresh oranges that are going to get rotten in this tick.
- When we no longer have any more neighbours to child, we check if there are any fresh oranges left.
- If there are, return -1, else return value of tick.

$$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

---

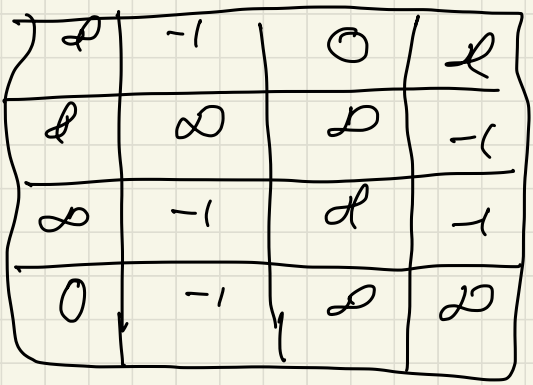
(51) #286 Walls and gates

We have a matrix with walls, gates and rooms.

Wall = -1 empty room = ∞.

gate = 0

Eg:-



- We need to replace all the ∞ cells with the value that represents the distance to the closest gate.
- A cell might be accessible from multiple gates.

1) Approach 1

- Get a queue of all gates.
- Do a bfs and see which are the cells connected to it. Keep track of distance from gate.
- If it is < current distance already in that path, update it and

add it to queue.

---

(52) #207 Course Schedule

- We have n courses.
- We have a 2d array of pre requisites.

eg [1, 0] ⇒ To do course 1, we need to complete 0.

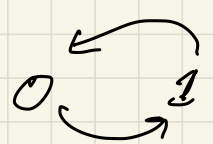
eg :- [(1, 0), (0, 1)]

↳ to do course 1, we need 0 and for 0, we need 1 :. Not possible.

Course numbers are from 0 → (numCourses - 1)

Eg :- 2 → 0, 1

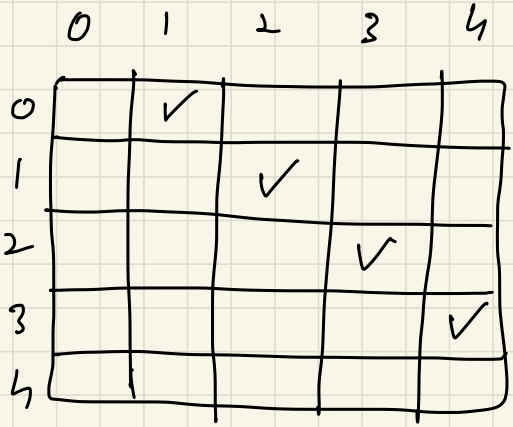
3 → 0, 1, 2.



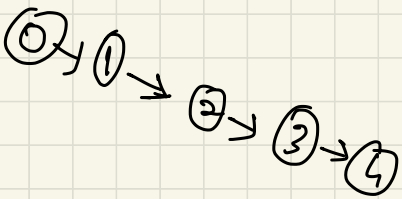
X.

- We can have upto 2000 courses

- The bare repo array can go upto 5000 elems.
  - Assume we have 5 courses.



This is a valid matrix.

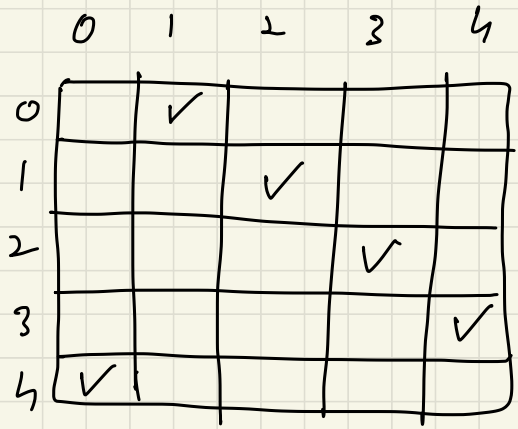


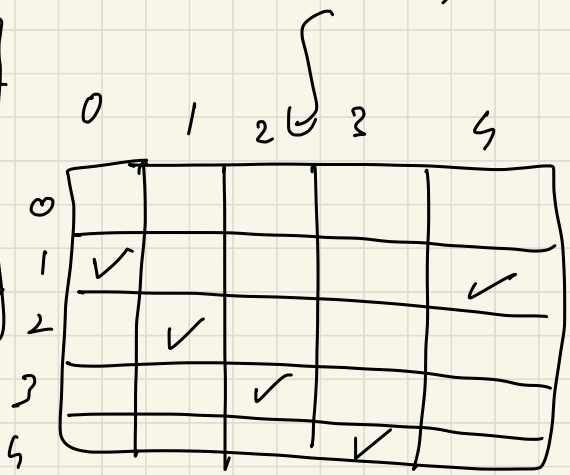
  - We can maybe try floyd's tortoise hare algo to check if we find any loop.

- What if there are multiple loops and branches?

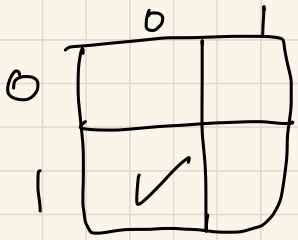
  - How will a matrix look like if there is a loop?

(1,0), (2,1), (3,2), (4,3), (0,4)

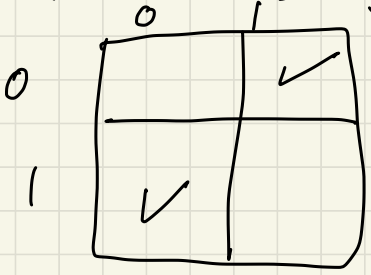


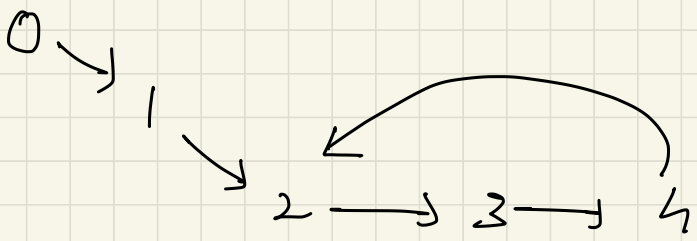


(2) :- (1,0)



[(1,0) (0,1)]





- The problem here is that we don't get the elements in order.
- So we need a more generic

approach.