Lecture :- Hashing 2: Geometry and string problems

Agenda

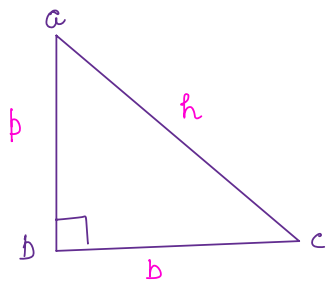
- count right Δs
- flip and find nearest
- Shortest substring of A having all characters of B.

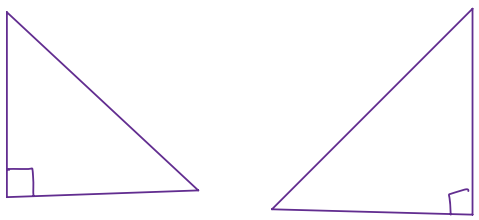
Qu1 Given n pts in a 2D plane, count no. of right ∠ Δs such that smaller sides are parallel to x and y axis.

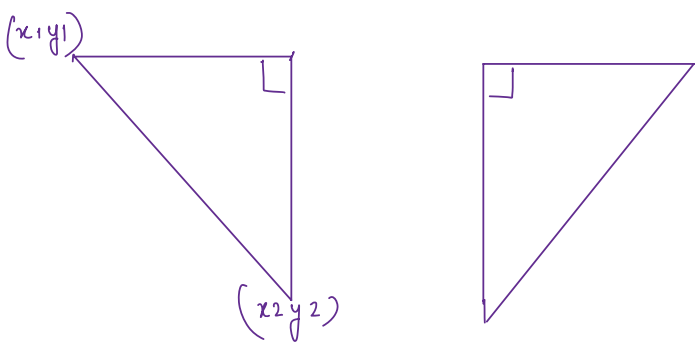
All 3 corners of Δ should be present in i/p.

2 Δs are diff if any one pt. is different.

→ How many right ∠ Δs can be || to x and y axis?

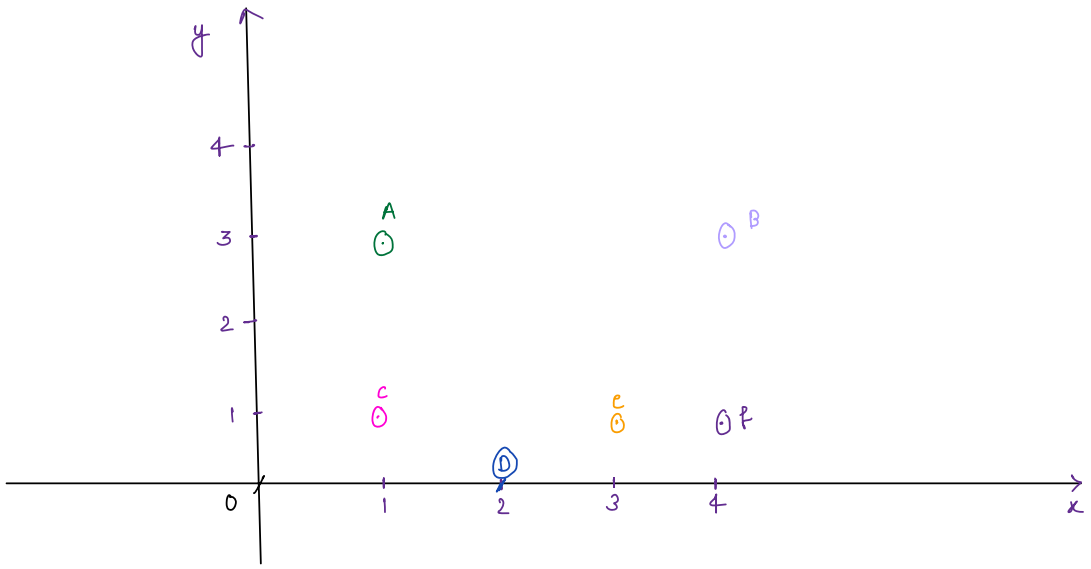


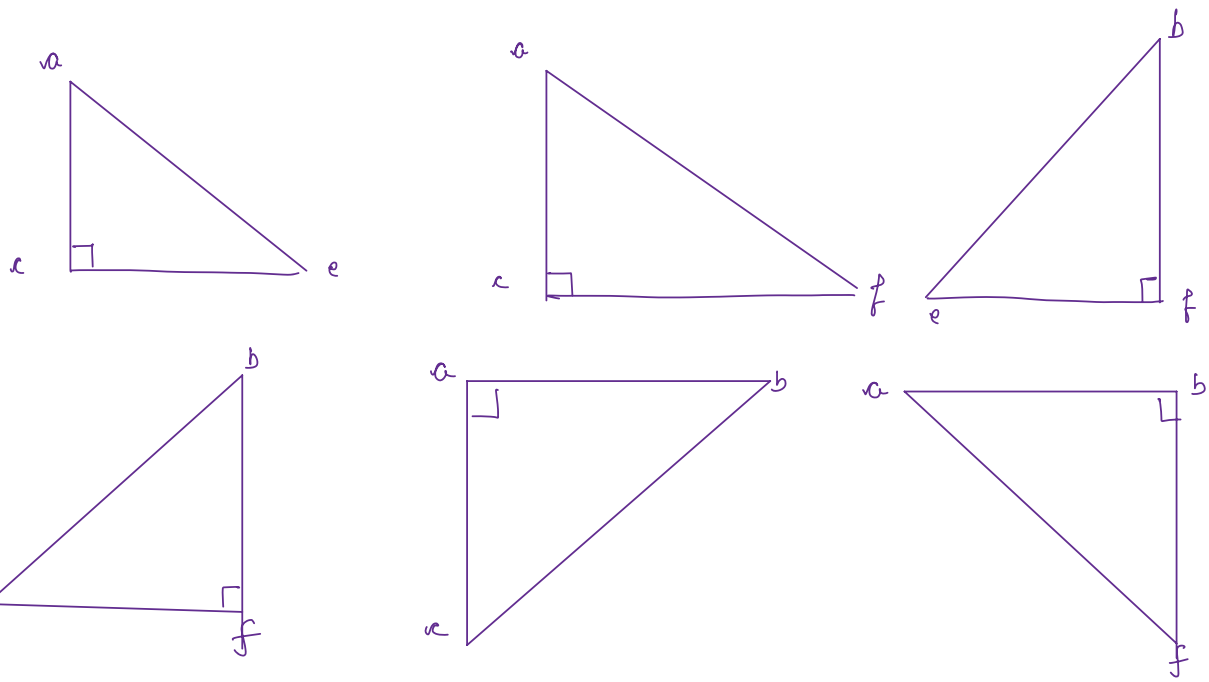




$x = [\ \underset{0}{1}\ \ \underset{1}{4}\ \ \underset{2}{1}\ \ \underset{3}{2}\ \ \underset{4}{3}\ \ \underset{5}{4}\ ]$

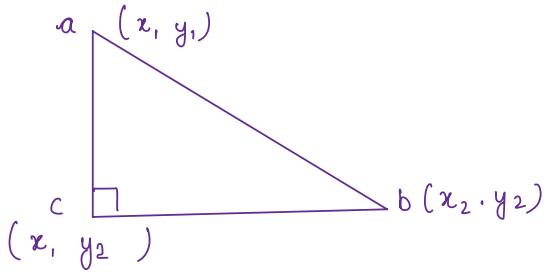
$y = [\ 3\ \ 3\ \ 1\ \ 0\ \ 1\ \ 1\ ]$





ans = 6.

Approach 1



for all pairs — $O(n^2)$

check if third pt is present inside arr or not? — $O(n)$

TC: $O(n^3)$

* Set < Class > set = new Hashset<>();

TC: O(n) — insert, search

x1 y1 ⟶ x1 @ y1

*

Set < String > set = new Hashset<>();

x = [1 4 1 2 3 4] (0 1 2 3 4 5)

y = [3 3 1 0 1 1]

set = ["1@3", "4@3", "1@1", "2@0" .....]

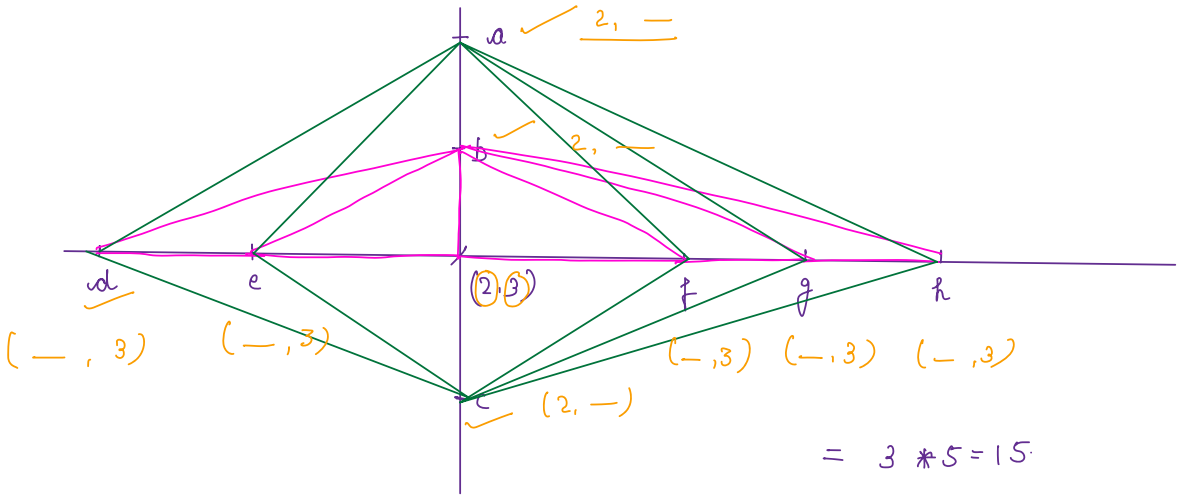
Ex: (1, 12) ⟶ ["112"] — wrong

(11, 2)

["1@12", "11@2"] — correct

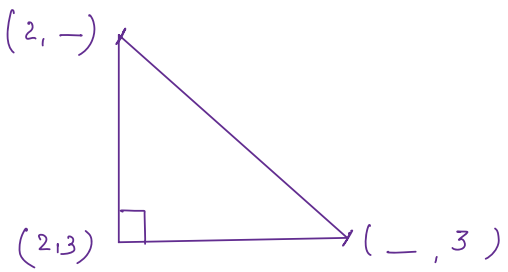
TC: $O(n^2)$

Approach 2



$= 3 * 5 = 15$

(2,3) as centre of right Δ



Ex:

	0	1	2	3	4	5
x =	1	4	1	2	3	4

y =	3	3	1	0	1	1

x map

1 : 2

2 : 1

3 : 1

4 : 2

y map

0 : 1

1 : 3

3 : 2

i=0 [1,3]	freq of 1 as x-coordinate = 2 - 1 = 1 apart from [1,3] freq of 3 as y-coordinate = 2-1 = 1 apart from [1,3] Total right Δ = 1 * 1 = 1
i=1 [4,3]	freq(4) x map : 2-1 = 1 freq(3) y map : 2-1 = 1 Total : 1 * 1 = 1
i=2 [1,1]	freq(1) x map : 2-1 = 1 freq(1) y map : 3-1 = 2 Total : 1 * 2 = 2
i=3 [2,0]	freq(2) x map : 1-1 = 0 (0) y map : 1-1 = 0 Total = 0
i=4 [3,1]	freq(3) x map = 1-1 = 0 Total = 0
i=5 [4,1]	freq(4) x map = 2-1 = 1 (1) y map = 3-1 = 2 Total = 1 * 2 = 2

Code:

```
int countRightTriangles(x[], y[]) {
    Map<Integer, Integer> x map;
    Map<  "       "  > y map;
```

→ update x map

→ update y map

```
    ans = 0;
    for(i=0; i<n; i++) {        // x.length (or) y.length
        freqx = xmap get(x[i]) - 1;
        freqy = y map. get(y[i]) - 1;
        ans += freqx * freqy;
    }
    return ans;
}
```

TC: O(n)

SC: O(n)

Qu2 Given 2 strings A and B [only lowercase characters]

$|A| = n$

$|B| = m$

Find length of smallest substring in B, which contains all characters of A. [if 2 x's are present in A, 2 x's should be present in B as well]

Return -1 if not possible

Ex1 A = ' x y z x '

B = ' x y z y x z x y ' ∴ ans = 4

Ex2 A = ' a b a c '

B = ' a b c a a c b b c b a c a b ' ans = 4

Ex3: a = a b c

b = a d o b e c o d e b a n c ans = 4

Approach

a = x y z (0 1 2)

b = x y z y x z x y (0 1 2 3 4 5 6 7)

l, r

mapA | freqA

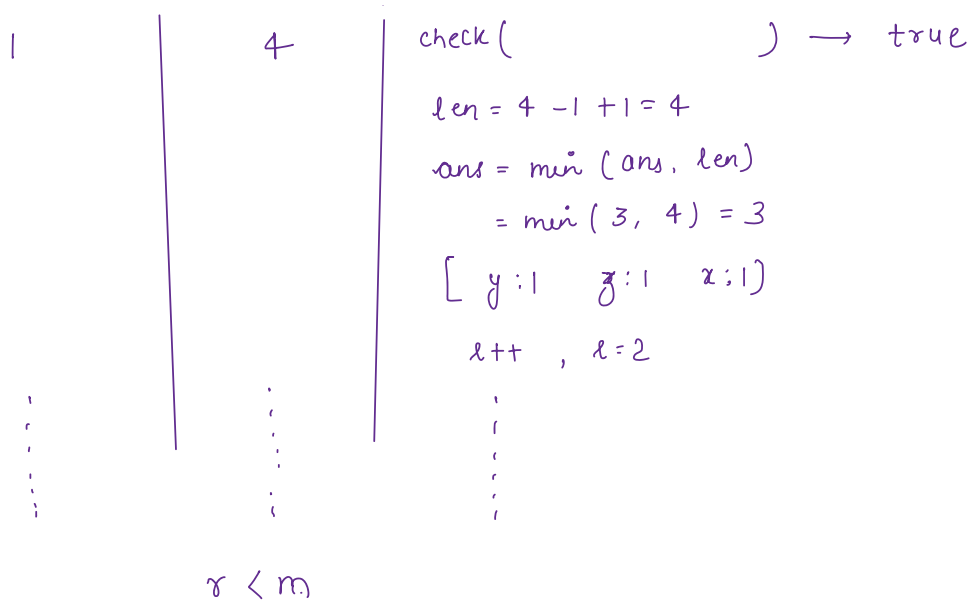
x : 1

y : 1

z : 1

mapB | freqB

l	r	freqA[x:1 y:1 z:1] freqB[x:1]
0	0	check (freqA, freqB) → false r++, r = 1 freqB [x:1 y:1]
0	1	check (freqA, freqB) → false r++, r = 2. freqB [x:1 y:1 z:1]
0	2	check (freqA, freqB) → true len = 2 - 0 + 1 = 3 r - l + 1 Try to search for smaller substring freqB[y:1 z:1] l++, l = 1.
1	2	check(freqA, freqB) → false r++, r=3 freqB [y:2, z:1)
1	3	check ( ) → false r++, r = 4 freqB [y:2 z:1 x:1]


1
4
check( ) → true
len = 4 - 1 + 1 = 4
ans = min (ans, len)
= min (3, 4) = 3
[y : 1 z : 1 x : 1)
l++ , l = 2
l < m

```
int minWindow( String A, String B) {
    n = A.length();
    m = B.length();

    freqA[26];
    freqB[26];

    for(i=0; i<n; i++) {
        freqA[ A.charAt(i) - 'a'] += 1;
    }

    ans = ∞;
    l = 0;
    r = 0;
    freqB[ B.charAt(0) - 'a'] += 1;

    while( r < m )
        if( check(freqA, freqB)) {
            ans = min(ans, r - l + 1);
            freqB[ B.charAt(l) - 'a'] -= 1;
            l++;
        } else {
            r += 1;
            if(r == m){
                break;
            }
            freqB[ B.charAt(r) - 'a'] += 1;
        }
    }
    return ans;
}
```

H/W

TC: O(m)
SC: O(1)

Break: 8:45 AM

Treemap — keys are sorted

Country → Population

Key	value
India	100
China	110
Aus	20
USA	40

Treemap

Aus : 20

China : 110

India : 100

USA : 40

All fun^{c} of treemap are similar to hashmap.

TC: $O(\log_2 n)$

Additional functions

1. floor (k) : greatest key <= k
2. ceil (k) : smallest key >= k

arr[] = [3 2 10 6 8]

↓ treeset

set [treeset] = [2 3 6 8 10]

floor(7) = 6

floor(5) = 3

floor (0) = null

ceil (7) = 8

ceil(11) = null

Qu Given arr[n] with all el as 0. and Q queries.

2 type of queries:

type1: flip data at ith idx : 0 ⟺ 1

Type2: Get nearest idx from i which has 1 -

- if (arr[i] == 1) return i.
- if 2 indices are present, print min idx
- if no idx exist, print -1.

n = 10 ; ip

arr[] = [0 0 0 0 0 0 0 1 0 1]

(idx: 0 1 2 3 4 5 6 7 8 9)

Queries = 8 (ip) Q

type	idx	
1	2	
1	8	
1	7	
2	4	[print 2]
1	8	
2	9	[print 7]
1	9	
2	8	[print 7]
2	2	[print 2]

Approach 1

create arr[n]

for each query — O(Q)

type1: flip [1 → 0, 0 → 1] O(1)

arr[idx] = 1 − arr[idx]

- arr[idx] = 0 ⟶ 1 − 0 = 1
- arr[idx] = 1 ⟶ 1 − 1 = 0

Type2:-

```
if (arr[idx] == 1) {
    print (idx);
}
```

O(n)

Iterate on left & right and find nearest idx

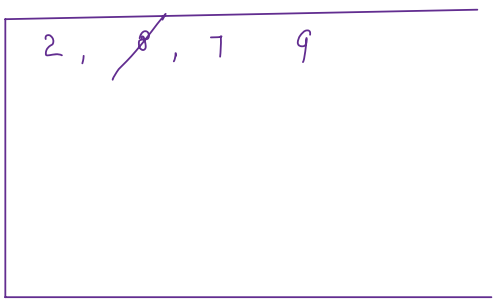
TC: O(Q * n)
SC: O(n)

Approach2

n = 10

arr[] = [0 0 ~~0~~ 1 0 0 0 0 ~~0~~ 1 ~~0~~ 0 ~~0~~ 1]
(indices 0 1 2 3 4 5 6 7 8 9)

treeset



Queries = 8 (i/p) Q

type	idx
1	2
1	8
1	7
2	4 [print 2]

l = floor(4) = 2
r = ceil(4) = 7
print (min(l, r)) // 2.

1	8
2	9 [print 7]

l = floor(9) = 7
r = ceil(9) = null
print (7)

1	9
2	8 [print 7]

l = floor(8) = 7
r = ceil(8) = 9
print (min(7, 9)) = 7.

2 : 2 [print 2]

Code:

```
void flipAndFindNearest(int n, int[][] queries) {
    Set<Integer> set = new TreeSet<>();
    for(i=0; i<queries.length; i++) {                    — O(Q)
        type = queries[i][0];
        idx = queries[i][1];
        if (type==1) {                                   — O(log2n)
            if(set.contains(idx)) {
                set.remove(idx);
            } else {
                set.add(idx);
            }
        } else {                                         — O(log2n)
            l = floor(idx);
            r = ceil(idx);
            if(l==null && r==null) {
                print(-1);
            } else if(l==null) {
                print(r);
            } else if(r==null) {
                print(l);
            } else {
                // distance b/w l and idx ⟶ x
                // ..  ..  r  ..  ..  ⟶ y
                HW    if(x<=y) { l }
                      else     { r }
            }
        }
    }
```

TC: O(Q * log2n)
SC: O(Q)

Thankyou ☺