Tech & Stack & Back-up

Generalization of Java:

Hash-Map self-defined class

Persistent data structures **Affirm**

Quick-Sort, Heap-Sort is not stable sorting algorithm.

Insertion Sort, Merge Sort is stable.

SQL Queries

JOIN: for each $item[i]$ of left table, connect $item[j]$ of right table as much as possible

ON: filter rules based on some attributes of both left & right table

619. Biggest Single Number (Based on Statistics)

Table number contains many numbers in column *num* including duplicated ones. write a SQL query to find the biggest number, which only appears once.

```
SELECT MAX(num) AS num FROM (
    SELECT num, COUNT(num) AS freq
    FROM number
    GROUP BY num
) AS T
WHERE T.freq = 1
```

612. Shortest Distance in a Plane

Given a list of unique $2D$ coordinates (x, y), write a query to find the shortest distance between these points rounded to 2 decimals.

```
SELECT ROUND(MIN(SQRT(POW(T.x1 - T.x2, 2) + POW(T.y1 - T.y2, 2))), 2) AS shortest FROM (
    SELECT A.x AS x1, A.y AS y1, B.x AS x2, B.y as y2
    FROM point_2d AS A
        JOIN point_2d AS B
    ON NOT(A.x = B.x and A.y = B.y)
) AS T
```

177. Nth Highest Salary

LIMIT (number, offset)

```
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
  DECLARE M INT;
  SET M = N - 1;
  RETURN (
      SELECT DISTINCT Salary FROM Employee ORDER BY Salary DESC LIMIT M, 1
  );
END
```

184. Department Highest Salary

```
SELECT Department.Name AS Department, Employee.name AS Employee, Employee.Salary
FROM Employee LEFT JOIN Department ON Employee.DepartmentId = Department.Id
WHERE (Department.Id, Employee.Salary) IN (
    SELECT DepartmentId, MAX(Salary)
    FROM Employee
    GROUP BY DepartmentId
)
```

181. Employees Earning More Than Their Managers

Intuition

For each entry E_1^i of E1 & each entry E_2^j, if $E_1^i.ManagerId = E_2^i.Id$ & $E_1.Salary > E_2.Salary$, connect them.

```
SELECT E1.Name AS Employee
FROM Employee AS E1 JOIN Employee AS E2 ON E1.ManagerId = E2.Id AND E1.Salary > E2.Salary
```

180. Consecutive Numbers

Find all numbers that appear at least three times consecutively.

```
SELECT DISTINCT L1.Num AS ConsecutiveNums
FROM Logs AS L1, Logs AS L2, Logs AS L3
WHERE L1.Id + 1 = L2.Id AND L2.Id + 1 = L3.Id AND L1.Num = L2.Num AND L2.Num = L3.Num
```

Join Version

```
SELECT DISTINCT L1.Num AS ConsecutiveNums
FROM Logs AS L1 JOIN Logs AS L2 JOIN Logs AS L3
ON L1.Id + 1 = L2.Id AND L2.Id + 1 = L3.Id AND L1.Num = L2.Num AND L2.Num = L3.Num
```

Brainteaser

292. Nim Game

Question: N stones, two players, each of them can only remove at-most m stone per time, check if player1 could win or not.

Idea: $if\ (mod(n, m+1) = 0)\ return\ false$

Data Structure & System Design

681. Next Closet Time Google

Solution: find closet point in a circle $(24 * 60)$

$for(i = 0 \rightarrow time.length - 1)$:

    $if(time[i] \neq ":")\ set.add(times[i] - '0')$

$curTimeInMs = Integer.parseInt(time_{0,1}) \times 60 + Integer.parseInt(time_{2,3})$

$minDif = 24 * 60$

$result = curTimeInMs$

$for((h_1, h_2, m_1, m_2)\ in\ set)$:

    $if(10h_1 + h_2 < 24\ \&\ 10_{m1} + m_2 < 60)$:

        $nextTime = (h_1 \star 10 + h_2) \star 60 + m_1 \star 10 + m_2$

        $difMod = mod((nextTime - cur + 24 \star 60), (24 \star 60))$

        $if(difMod > 0\ \&\ difMod < minDif)$:

            $minDif = difMod$

            $result = nextTime$

$return\ String.format('\%02d:\%02d', result/60, mod(result, 60))$

Follow Up: next time is current time's permutation

Solution: just change the last 5 lines of code (colored in blue) to the followings:

$set.add(all\ permutations\ of\ time\ digits)$

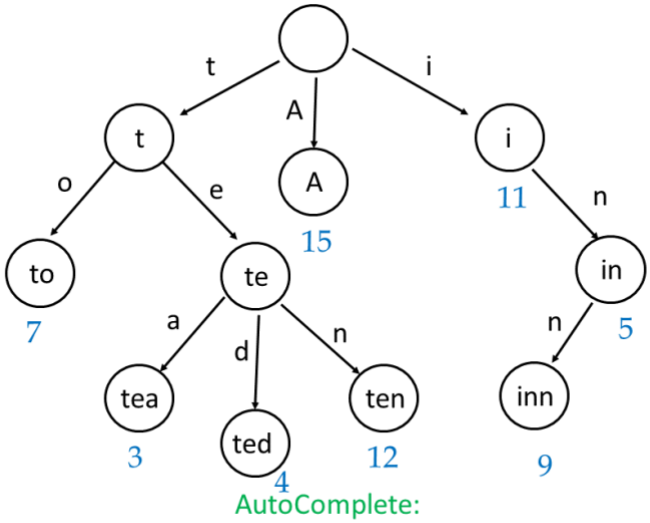
$curTimeInMs = int(time_{0,1}) \times 60 + int(time_{2,3})$

$minDif = 24 \star 60$

$result = curTimeInMs$

---

642. Design Search Autocomplete System Facebook Microsoft



Question: implements two functions: $AutocompleteSystem(sentences: [string],\ times: [int])$ This is the constructor. The input is historical data. Sentences is a string array consists of previously typed sentences. Times is the corresponding times a sentence has been typed, and $List\langle String\rangle\ input\ (char\ c)$. The input c is the next character typed by the user. The character will only be lower-case letters ($'a' \rightarrow 'z'$), blank space (' ') or a special character ('#'). Also, the previously typed sentence should be recorded in your system. The output will be the top 3 historical hot sentences that have prefix the same as the part of sentence already typed.

```
class AutocompleteSystem:
    class TrieNode: {next: [TrieNode, size = 27], time = int}
    class SNode: {str: String, time = int} ⋆ wrapper class
    TrieNode root
    String currPrefix = ""

    public AutocompleteSystem(sentences: [string], times[int])
        root = new TrieNode
        for(i = 0 → len(sentences) − 1): insert(sentences[i], times[i], root)

    public List⟨String⟩ input(char c):
        if(c = #):⋆ # suggests the input ended
            insert(currPrefix, 1, root)
            currPrefix = ""
            return [ϕ]
        else:
            currPrefix = currPrefix + c
```

```
        List⟨SNode⟩ ls = query(currPrefix, root)
        Collections.sort(ls, (a, b) → (a.time = b.time → a.str < b.str: b.time > a.time))
        List⟨String⟩ re = getTop3Strings(ls) ⋆ select top 3 strings with most frequency
        return re

public void insert(String s, int time, TrieNode r):
    for(i = 0 → len(s) − 1):
        index = s[i] = ' ' → 26: s[i]−'a'
        if(r.next[index] = nil): r.next[index] = new TrieNode
        r = r.next[index]
    r.time = r.time + time ⋆ update the frequency of s

public List⟨SNode⟩ query(String currPrefix, TrieNode r):
    for(i = 0 → len(currPrefix) − 1):
        index = s[i] = ' ' → 26: s[i]−'a'
        ⋆ if no such words starting with currPrefix, return empty list
        if(r.next[index] = nil): return [ϕ]
        r = r.next[index]
    List⟨SNode⟩ list = [ϕ]
    traverse(currPrefix, r, list)

public void traverse(String curr, TrieNode p, List⟨SNode⟩ list):
    if(p.time > 0): list.add(SNode(curr, p.time))
    for(char c = a → z):
        index = c = ' ' → 26: c−'a'
        if(p.next[index] ≠ nil): traverse(curr + c, p.next[index], list)
    if(p.next[26] ≠ nil): traverse(curr + ' ', p.next[26], list)
```

---

631. Design Excel Sum Formula Microsoft

The key idea behind the solution lies in some prerequisites, firstly, once a formula is applied to any cell in excel, say $\boldsymbol{C_1 = C_2 + C_3}$, then any changes made to either $\boldsymbol{C_2}$ or $\boldsymbol{C_3}$ will result in the change to the value of C_1. Further, suppose another cell like D_1 is also dependent on C_1, then any changes made to C_2 trigger an update chain $\boldsymbol{C_2 \rightarrow C_1 \rightarrow D_1}$.

Then, we discuss how to implement the above idea. We make use of a simple data structure (***Formula***), which contains two elements: ***value***, which represents the value of this cell, and a hash-map ***dependency***, which is a list of cells on which the current cell is dependent. The dependency map stores the data in the form of (***cellName***, ***count***) where ***cellName*** showed in the form of ***rolNum***: ***rowNum*** such like $A1$ *or* $B2$, and *count* stores the number of times the current cell comes in the current cell's formulas. For example, $C_1 = C_2 + C_3 + C_2$, in this case, the frequency of C_3 is 1, and the frequency of C_2 is 1.

The updating chain can be obtained through topological sort, starting from current cell, search all other non-empty cells whose ***dependency*** map contains current cell, and recursively continue searching from those satisfied cells. We make use of a stack which keeps track of cells in topological order($top \rightarrow bottom$).

```
class Excel
    class Cell: {dependency, value}
    stack = (r, c)
    cells = R^2d
    maxr, maxc

    public Excel(int H, char W): cells = new [H, W − A + 1], maxr = H, maxc = W − A + 1

    int get(int r, char c): return cells[r − 1, c − A] = nil → 0: cells[r − 1, c − A].value

    void set(int r, char c, int v):
        cells[r − 1, c − A] = {new HashMap<String, Integer>(), v}
        chain · update(r, c)

    int sum(int r, char c, strs: array):
        1 calculate the sum of all dependent cells specified in rules: strs
        2 set the cell(r, c) with the calculated sum
        3 chain · update all cells which are dependent on cell(r, c)
        dependency = {ϕ}
        for(s: strs):
            if(s.indexOf(:) < 0): dependency[s] = dependency[s] + 1 ⋆ single cell
            else:⋆⋆ rectangular area
                left · upper = s.split(:)[0], right · bottom = s.split(:)
                c1 = left · upper[0], r1 = int(left · upper[1:])
```

```
            c_2 = right · bottom[0], r_2 = int(right · bottom[1:])
            for(r: r_1 → r_2)
                for(c: c_1 → c_2):
                    dependency[c + r + ""] = dependency[c + r + ""] + 1
    sum = 0
    for(cell: dependency.keySet):
        r = int(cell[1:]), c = cell[0]
        sum := sum + dependency[c + r + ""] * (cells[r, c] = nil → 0: cells[r, c].value)

    ** update this cell(r, c) with new formulas and new value, namely the calculated sum
    if(cells[r − 1, c − A] = nil): cells[r − 1, c − A] = Cell{(dependency, sum)}
    else: cells[r − 1, c − A].dependency = dependency, cells[r − 1, c − A].sum = sum

    ** update all other cells that are dependent on this cell(r, c)
    update · chain(r, c)
    return sum


void chain · update(int r, char c)
    affects<String>: stack
    topological · sort(r, c, affects)
    affects.pop * the top of stack is the cell(r, c) itself
    while(affects.size > 0):
        cell = affects.pop
        c = cell[0], r = int(cell[1:])
        sum = 0
        for(dependentCell: cells[r − 1, c − A].keySet):
            x = int(dependentCell[1:]), y = dependentCell[0]
            value = cells[x − 1, y − A] = nil → 0: cells[x − 1, y − A].value
            times = cells[r − 1, c − A].dependency[dependentCell]
            sum = sum + value * times
        cells[r − 1, c − A].value = sum

* collect all cells which are directly or indirectly dependent on cell(r, c)
void topological · sort(int r, char c, affects<String>: stack):
```

$for(n_r = 1 \rightarrow maxr)$:

$for(n_c = A \rightarrow A + maxc - 1)$:

$if(cells[n_r - 1, n_c - A] \neq nil \ \& \ cells[n_r - 1, n_c - A].dependency[c + r] \neq nil)$:

$topological \cdot sort(r, c, affects)$

$affects.push(c + r + "")$

---

635. Design Log Storage System **Snapchat**

Design a log system supports two operations

$put(id_{int}, timestamp_{string})$

$retrieve(start_{timestamp}, end_{timestamp})$

359. Logger Rate Limiter **Google**

$map: [msg(String) \Rightarrow timestamp(int)]$

$bool \ shouldPrintMessage(int \ timestamp, String \ msg)$:

$if(map[msg] \neq nil)$:

$int \ prevTimeStamp = map[msg]$

$if(timestamp - prevTimeStamp \geq 10)$:

$map[msg] = timestamp$

$return \ true$

$else: return \ false$

$else$:

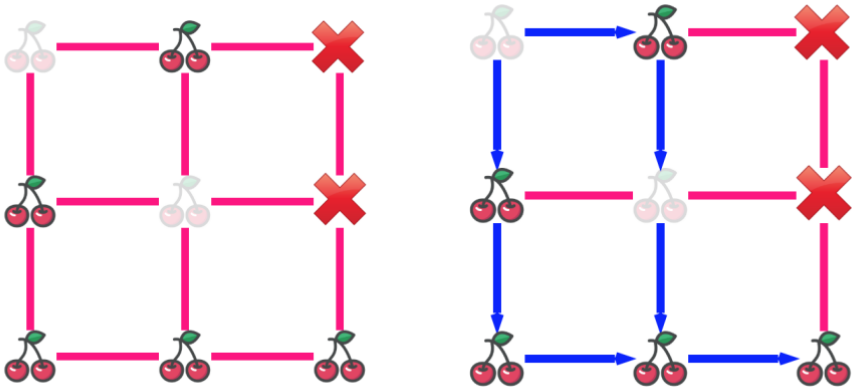
$map[msg] = timestamp$

$return \ true$

Dynamic Programming

741. Cherry Pickup

$$\begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{cases} 1: cherry; \\ -1: chorn; \\ 0: empty \ cell \end{cases} \rightarrow \max number \ of \ cherries \ collect: 5$$



one can start from $(0,0)$, collects as much cherries as possible, then reach $(n-1,n-1)$. Once it reaches $(n-1,n-1)$, he makes another journey from $(n-1,n-1)$ to $(0,0)$, and collect as much cherries as possible. Find out the max number of cherries he can collect.

Solution: the key to this problem is to imagine that starting from (0,0), 2 guys simultaneously make journey to $(n-1,n-1)$, and collect as much cherries as possible. DP method may be applied to this problem by defining $f_{3d}: f_{x_1,y_1,x_2}$ as the maximal amount of cherries when $player_1$ at the position of (x_1,y_1) and $player_2$ at the position of (x_2,y_2). Since two players move simultaneously, so their walking distance are the same meaning:

$$x_1 + y_1 = x_2 + y_2$$
$$y_2 = x_1 + y_1 - x_2$$

Also, since each player can either move up or left, so there are four movement combinations:

$$(x_1,y_1,x_2,y_2) \rightarrow \begin{cases} (x_1-1,y_1,x_2-1,y_2) \\ (x_1-1,y_1,x_2,y_2-1) \\ (x_1,y_1-1,x_2-1,y_2) \\ (x_1,y_1-1,x_2,y_2-1) \end{cases}$$

$$f_{x_1,y_1,x_2} = max \begin{cases} f_{x_1-1,y_1,x_2-1} \\ f_{x_1-1,y_1,x_2} \\ f_{x_1,y_1-1,x_2-1} \\ f_{x_1,y_1-1,x_2} \end{cases} + g_{x_1,y_1} + (x_1 \oplus x_2) * g_{x_2,x_1+y_1-x_2}$$

$dp(f_{3d}, g_{2d}, x_1, y_1, x_2)$:

$y_2 = x_1 + y_1 - x_2$

$if\big(x_1 < 0\ or\ x_2 < 0\ or\ x_2 < 0\ or\ y_2 < 0\ or\ g_{x_1,y_1} = -1\ or\ g_{x_2,y_2} = -1\ \big): return -1$

$if(x_1 = 0\ \&\ y_1 = 0): return\ g_{x_1,y_1}$

$if\big(f_{x_1,y_1,x_2} \neq nil\big): return\ f_{x_1,y_1,x_2}$

$re = \max\big(f_{x_1-1,y_1,x_2-1}, f_{x_1-1,y_1,x_2}, f_{x_1,y_1-1,x_2-1}, f_{x_1,y_1-1,x_2}\big)$

$if(re < 0): return\ f_{x_1,y_1,x_2} = -1$

$return\ f_{x_1,y_1,x_2} = re + g_{x_1,y_1} + (x_1 \oplus x_2) * g_{x_2,y_2}$

---

446. Arithmetic Slices II – Subsequence Baidu

A sequence of numbers is called arithmetic if it consists of at least three elements and if the difference between any two consecutive elements is the same.

$int\ scriptOfArithmeticSlices(int[\ ]\ A)$:

    $\mathcal{M}ap\langle int, int\rangle[\ ]\ map = [A.length]$

    $re \coloneqq 0$

    $for(i = 0 \to len(A) - 1)$:

        $map[i] \coloneqq new\ \mathcal{H}ashMap$

        $for(j \coloneqq i - 1 \to 0)$:

            $\ell ong\ dif \coloneqq A[i] - A[j]$

            $c_1 \coloneqq map[i][dif], c_2 \coloneqq map[j][dif]$

            $re \coloneqq re + c_2$

            $map[i][dif] \coloneqq c_1 + c_2 + 1$

    $return\ re$

---

334. Increasing Triplet Subsequence Facebook

Find such a triplet (i, j, k) s.t. $A[i] < A[j] < A[k]$. exists in a given array or not.

$l[i] \coloneqq (i = 0\ ||\ A[i] < l[i-1]) \to A[i]: l[i-1]$

$r[i] \coloneqq (i = n-1\ ||\ A[i] > r[i-1]) \to A[i]: r[i+1]$

$return\ l[i] < A[i] < r[i], \forall i \in [1, n-2]$

764. Largest Plus Sign Facebook

In a 2D grid from $(0,0) \to (N-1, N-1)$, every cell contains a 1, except those cells in the given list mines which are 0. What is the largest axis-aligned plus sign of 1s contained in the grid? Return the order of the plus sign. If there is none, return 0. An "axis-aligned plus sign of 1s of

order k" has some center $grid[x,y] = 1$ along with 4 arms of length k-1 going up, down, left, and right, and made of 1s. This is demonstrated in the diagrams below. Note that there could be 0s or 1s beyond the arms of the plus sign, only the relevant area of the plus sign is checked for 1s.

$left[i,j] \coloneqq left[i,j-1] + 1$

$right[i,j] \coloneqq left[i,j+1] + 1$

$top[i,j] \coloneqq top[i-1,j] + 1$

$bottom[i,j] \coloneqq bottom[i+1,j] + 1$

$re = \max(re, min(left[i,j], right[i,j], top[i,j], bottom[i,j]))$

---

718. Maximum Length of Repeated Subarray Citadel

Given two integer arrays A and B, return the maximum length of a subarray that appears in both arrays. $A = [1,2,3,2,1], B = [3,2,1,4,7] \Rightarrow [3,2,1]$

Algorithm #1

Longest Common Substring Problem. Define $f[i,j]$ as the maximal length of subarray $A[u,i]$ exactly match up with $B[v,j]$.

$$f[i,j] = f[i-1,j-1] + 1, if\ A[i] = B[j]$$

$$f[i,j] = 0, otherwise$$

$$result \coloneqq \max(f[i,j])$$

There is a little bit different from LCS(subsequence) Problem, where the DP formula goes like:

$$f[i,j] = f[i-1,j-1] + 1, if A[i] = B[j]$$

$$f[i,j] = \max(f[i-1,j], f[i,j-1])$$

The difference lies in the fact that subsequence just need to end up with either $A[i]$ or $B[j]$, while substring must end up with both $A[i]$ *or* $B[j] (A[i] = B[j])$

801. Minimum Swaps To Make Sequences Increasing Facebook Amazon

We have two integer sequences A and B of the same non-zero length. We are allowed to swap elements $A[i]$ and $B[i]$. Note that both elements are in the same index position in their respective sequences. At the end of some number of swaps, A and B are both strictly increasing. (A sequence is strictly increasing if and only if $A[0] < A[1] < \ldots < A[len(A) - 1]$.) Given A and B, return the minimum number of swaps to make both sequences strictly increasing. It is guaranteed that the given input always makes it possible.

Intuition & Algorithm

The cost of making both sequences increasing up to the first i columns can be expressed in terms of the cost of making both sequences increasing up to the first i-1 columns. This is because the only thing that matters to the ith column is whether the previous column was swapped or not. This makes dynamic programming an ideal choice.

Let's define $swap[i]$, the cost of making the first $(i-1)$ columns increasing and **not swapping** the $(i-1)^{th}$ column, and $noswap[i]$, the cost of making the first $(i-1)$ columns increasing and **swapping** the $(i-1)^{th}$ column.

Now, there are two cases to be considered,

Case #1

$$A[i-1] < A[i]$$
$$B[i-1] < B[i]$$

in this case, we have two options, one is doing nothing, neither swapping the $(i-1)^{th}$ column nor the i^{th} column. Another is swapping both the two columns. So we have:

$$noswap[i] = \min(noswap[i], swap[i-1])$$
$$swap[i] = \min(swap[i], swap[i-1]+1)$$

Case #2

$$A[i-1] < B[i]$$
$$B[i-1] < A[i]$$

in this case, we too have two options, one is only swapping the $(i-1)^{th}$ column. Another is swapping the i^{th} column. So we have:

$$noswap[i] = \min(noswap[i], swap[i-1])$$
$$swap[i] = min(swap[i], noswap[i-1]+1)$$

Finally, the result is: $re = \min(noswap[n-1], swap[n-1])$

Key Take-away: Case #1 & #2 can happen at the same time

$n = [N], s := [N]$

$n[0] := 0, s[0] := 1$

$for(i = 1 \rightarrow N-1)$:

$s[i] := n[i] := 2^{31}-1 \star very\ important$

$if(A[i-1] < A[i]\ \&\ B[i-1] < B[i])$:$\star$ $Case\ \#1$

$n[i] \coloneqq \min(n[i], n[i-1])$

$s[i] \coloneqq \min(s[i], s[i-1]+1)$

$if\,(A[i-1] < B[i]\ \&\ B[i-1] < A[i])$:$\star$ $Case\ \#2$

$n[i] \coloneqq \min(n[i], s[i-1])$

$s[i] \coloneqq \min(s[i], n[i-1]+1)$

$return\ \min(s[N-1], n[N-1])$

---

276. Paint Fence Google

Question: paint n fences with k colors, no 3 adjacent fences with same color are allowed.

Solution: two adjacent + single fence with the same color are allowed, so:

$f[i] = f[i-2] \star (k-1) + f[i-1] \star (k-1)$

[…… R R]

[…… R]

it is quite like Fibonacci, and can optimize the solution to constant space cost.

$a = k, b = k \star k$

$for(i = 3 \to n)$:

$c = (k-1) \star (a+b)$

$a = b$

$b = c$

---

256. Paint House

265. Paint House II Facebook

Given a row of n houses, each of which can be painted with one of k colors. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

Intuition & Algorithm #1

Let $f[i,j]$ as the min cost of painting first i houses, and the i^{th} house is painted with color j.

The DP formula goes like

$$f[i,j] = \min(f[i-1,k]) + costs[i-1,j]\,, \forall k \in [0, j-1] \cup [j+1, k-1]$$

Time Costs: $O(nk^2)$

$int\ mincost(int[\,][\,]\ costs)$:

$n \coloneqq len(costs), k \coloneqq len(costs[0])$

$re \coloneqq -1$

$int[\,][\,]\, f = [n][k]$

$for(j = 0 \to k-1)$: $f[0,j] \coloneqq costs[0,j]$

$for(i = 0 \to n-1)$:

    $for(j = 0 \to k-1)$:

        $f[i,j] = 2^{31} - 1$

        $for\left(j_{prev} = 0 \to k-1\right)$:

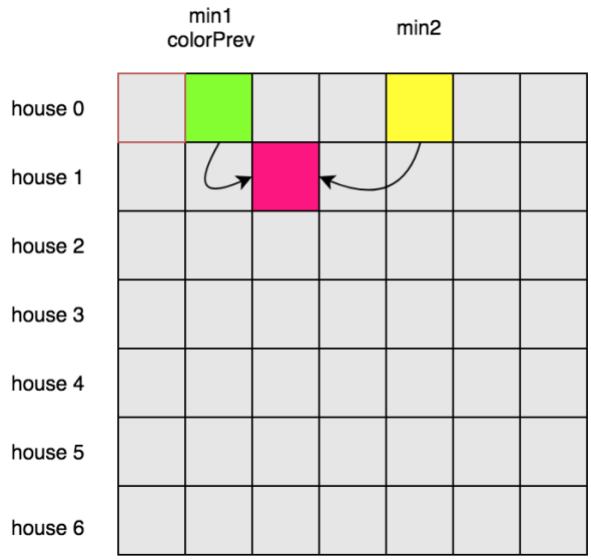
            $if\left(j_{prev} \neq j\right)$: $f[i,j] = \min\left(f[i,j], f\left[i-1, j_{prev}\right] + costs[i,j]\right)$

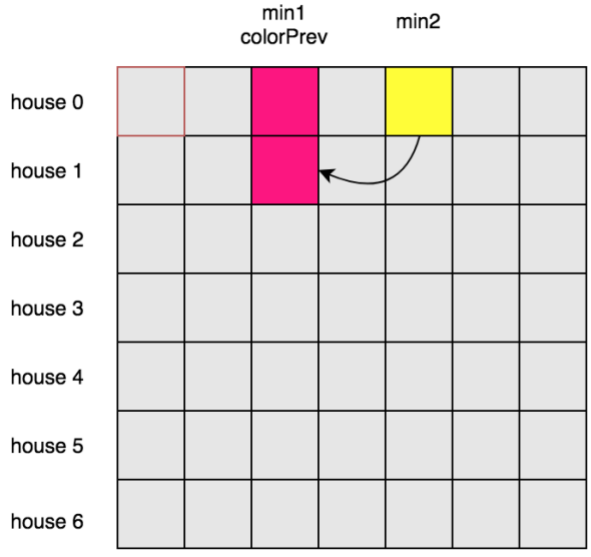
$return \min(f[n-1,j])\,, \forall j \in [0, k-1]$

Intuition & Algorithm #2

Optimization: for each of color j for house i, we don't need to compare k colors. Instead, we just need to maintain 2 minimal overall costs for painting until house $(i-1)$ as well as the best color correspond to the minimal cost.

Time Costs: $O(nk)$





$int\ mincost(int[\,][\,]\ costs)$:

$\star\ (min1, min2) \Rightarrow the\ 1^{st}, 2^{nd}\ minimum\ cost\ of\ painting\ houses\ so\ far$

$\star\ the$ "best" $color\ making\ the\ mimimum\ cost, namely, min1$

$min1 \coloneqq 0, min2 \coloneqq 0, color_{prev} \coloneqq -1$

$n \coloneqq len(costs), k \coloneqq len(costs[0])$

$for(i = 0 \to n - 1)$:

$curmin1 \coloneqq 2^{31} - 1, curmin2 \coloneqq 2^{31} - 1, mincolor \coloneqq -1$

$for(j = 0 \to k - 1)$:

$if\big(j \neq color_{prev}\big): costs[i,j] \coloneqq costs[i,j] + min1$

$else: costs[i,j] \coloneqq costs[i,j] + min2$

$if(costs[i,j] < curmin1)$:

$curmin2 \coloneqq curmin1, curmin1 \coloneqq costs[i,j]$

$mincolor \coloneqq color$

$elif(costs[i,j] < curmin2): curmin2 \coloneqq costs[i,j]$

$min1 \coloneqq curmin1, min2 \coloneqq curmin2, color_{prev} \coloneqq mincolor$

$return\ min1$

---

678. Valid Parenthesis String **Alibaba**

Solution: the short answer to this problem is applying DP by defining $f_{(l,r)}$ as true in cases $s[l: r]$ is valid or false otherwise.

1 $\star \cdots \cdots$

2 $\star \cdots \cdots)$

3 $\star \cdots)(\cdots) \to in\ this\ case, \star = ($

4 $(\cdots \cdots)$

5 $(\cdots \cdots \star \to in\ this\ case, \star =)$

6 $(\cdots \star (\cdots) \to in\ this\ case, \star =)$

7 $(\cdots)(\cdots)$

$boolean\ DP(s, l, r, f)$:

$if(l > r): return\ true \to$ *corner case 1*

$if(l = r): return\ s_l = \star \to$ *corner case 2*

$if(l + 1 = r): return\ \big(s_l = (\ or\ s_r = \star)\big)\ \&\ \big((s_r = \star\ or\ s_r =)\big) \to$ *corner case 3*

$if\big(f_{(l,r)} \neq nil\big): return\ f_{(l,r)}$

$boolean\ valid = false$

$if(s_l = \star)$:

$if\big(DP(s, l + 1, r, f)\big): valid = true$

$if(s_r =)\ \&\ DP(s, l + 1, r - 1, f)): valid = true$

$for(k = l + 1 \to r)$:

$if\big(s_k =) \ \&\ DP(s, l+1, k-1, f) \ \&\ DP(s, k+1, r, f)\big): valid = true, break$

$if(s_l = ( ):$

$if\big(s_r =) \ \&\ DP(s, l+1, r-1, f)\big): valid = true$

$if\ \Big(s_r =\star \ \&\ \big(DP(s, l+1, r-1, f)\ or\ DP(s, l, r-1, f)\big)\Big): valid = true$

$for(k = l+1 \rightarrow r):$

$if((s_k =)or\ s_k =\star) \ \&\ DP(s, l+1, k-1, f) \ \&\ DP(s, k+1, r, f)): valid = true, break$

$f_{(l,r)} = valid$

$return\ valid$

---

32. Longest Valid Parentheses Facebook Amazon Google Adobe Alibaba

Intuition & Algorithm #1

```
int longestValidParentheses(String s):
    le = len(s)
    if(le ≤ 1): return 0
    Stack[int] st = {ϕ}
    for(i = 0 → le − 1):
        if(st.empty || st.peek = ( ): st.push(i)
        else:
            if(st.peek =) ): st.pop
            else: st.push(i)
    i = len(s)
    while(~st.empty):
        int index ≔ st.pop
        int currLen ≔ i − index − 1
        re ≔ max(re, currLen)
        i ≔ index
    re ≔ max(re, i)
    return re
```

Intuition & Algorithm #2

In this approach, we make use of two counters $left$ & $right$, First, we start traversing the string from the left towards the right and for every '(' encountered, we increment the $left$ counter, and for every ')' encountered, we increment the right counter. Whenever $left$ becomes equal to $right$. we calculate the length of the current valid string and keep track of maximum length substring found so far. If $right$ becomes greater than $left$ we set $left$ & $right$ to zero.

Next, we start traversing the string from right to left and similar procedure is applied.

562. Longest Line of Consecutive One in Matrix Google

Given a 01 matrix M, find the longest line of consecutive one in the matrix. The line could be horizontal, vertical, diagonal or anti-diagonal.

Intuition

Define $f[i,j,0], f[i,j,1], f[i,j,2], f[i,j,3]$ storing the maximal number of continuous ones found so far along the horizontal, vertical, diagonal, x-diagonal line.

$$f[i,j,0] \coloneqq f[i,j-1,0] + 1$$
$$f[i,j,1] \coloneqq f[i-1,j,1] + 1$$
$$f[i,j,2] \coloneqq f[i-1,j-1,2] + 1$$
$$f[i,j,3] \coloneqq f[i-1,j+1,3] + 1$$

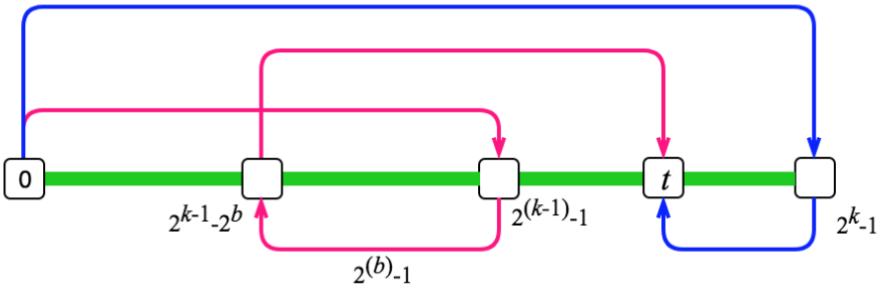
$int\ longest \cdot line(M: R^{2d})$:
    $f = [len(M)] \star [len(M[0])] \star 4, ones = 0$
    $for(i = 0 \to len(M))$:
        $for(j = 0 \to len(M[0]))$:
            $if(M_{(i,j)} = 1)$:
                $f[i,j,0] \coloneqq j > 0 \to f[i,j-1,0] + 1 : 1$
                $f[i,j,1] \coloneqq i > 0 \to f[i-1,j,1] + 1 : 1$
                $f[i,j,2] \coloneqq (i > 0\ \&\ j > 0) \to f[i-1,j-1,2] + 1 : 1$
                $f[i,j,3] \coloneqq (i > 0\ \&\ j < len(M[0]) - 1) \to f[i-1,j+1,3] + 1 : 1$
                $ones = \max(ones, f[i,j,k]\ \forall k \in [0,3])$

Interview: Given n, find the number of ways to fill a $2 \times n$ board with dominoes **Google**

Idea: $f[i] = f[i-1] \star f[i-2]$

818. Race Car **Google**

The car initially is at index 0, given a list of commands consisting of two characters: A & R,
When you get an instruction "A", your car does the following: $position += speed, speed \star= 2$.
When you get an instruction "R", your car does the following: if your speed is positive then $speed = -1, otherwise\ speed\ =\ 1$. (Your position stays the same.) For some target position, say the length of the shortest sequence of instructions to get there.



Intuition

The above diagram suggests two solutions: the blue & red line. As the blue line suggests, the car move forward from $0 \rightarrow\ 2^k - 1$, and then move backward from $(2^k - 1) \rightarrow t$. As for the red line, the car first move forward from $0 \rightarrow\ 2^k - 1$, then move backward from $2^{k-1} - 1 \Rightarrow 2^b - 1$, and finally move forward again from $2^b - 1 \rightarrow t$.

Since both two approaches has sub-problem in it, we can solve it with DP by defining $f[t]$ as the shortest sequence of instructions to get the target. For the blue line, its corresponding sequence is $A^k RA^{[DP(2^k-1-t)]}$, so we have $f[t] = k + 1 + f[2^k - 1 - t]$. As for the red line, the sequence is $A^{k-1}RA^bRA^{[DP(t-(2^{k-1}-2^b)]}$, so we have $f[t] = k + b + 1 + f[t - (2^{k-1} - 2^b)]$

Algorithm

$DP(t, int[\,][\,]\ f)$:

$if(t = 0): return\ 0$

$if(f[t] \neq nil): return\ f[t]$

$k = 32 - numberofLeadingZeros(t)$

$if(2^k - 1 = t): return\ f[t] \coloneqq k$

$f[t] = 2^{31} - 1$

$f[t] = \min\Big(f[t], k + 1 + DP(2^k - 1 - t, f)\Big)$

$for(b = 0 \rightarrow n - 2)$:

$pos = \left((2^{k-1} - 1) - (2^b - 1)\right) = 2^{k-1} - 2^b$

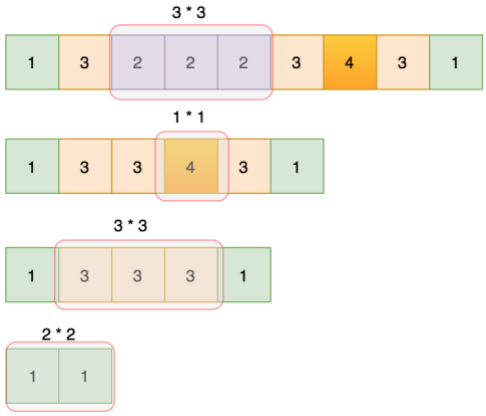
$f[t] = \min\left(f[t], k + b + 1 + DP(t - pos, f)\right)$

$return\ f[t]$

---

546. Remove Boxes Tencent

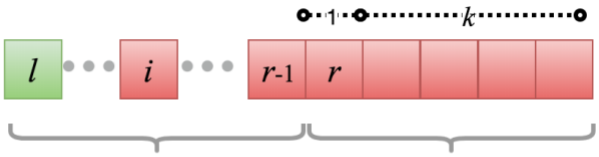
Given several boxes with different colors represented by different positive numbers. Each time you can choose some continuous boxes with the same color (composed of k boxes, $k \geq 1$), remove them and get $k * k$ points.



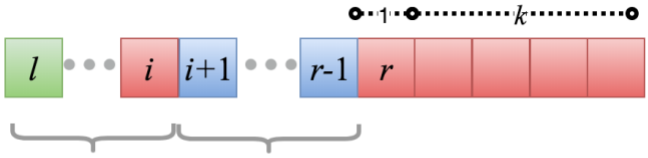
Intuition

For an entry $f[l, r, k]$ nas l represents the starting index of the subarray, r represents the ending index of the subarray and k represents the number of elements like the r^{th} element following it which can be combined to obtain the point information to be stored in $f[l, r, k]$.

Algorithm



$f[l, r, k] = f[l, r\text{-}1, 0] + (k+1)^2$



$f[l, r, k] = f[l, i, k+1] + f[i+1, r\text{-}1, 0]$

$$f[l,r,k] = \max(f[l,r,k], f[l,r-1,0] + (k+1)^2, f[l,i,k+1] + f[i+1,r-1,0])$$

$DP(l = 0, r = n-1, k = 0, int[\,][\,][\,]\, f, int[\,]\, boxes)$:

  $if(l > r)$: $return\ 0$

  $if(f[l,r,k] \neq nil)$: $return\ f[l,r,k]$

  $f[l,r,k] = f[l,r-1,0] + (k+1)^2$

  $for(i = l \to r-1)$:

    $if(boxes[i] = boxes[r])$:

      $f[l,r,k] = \max\big(f[l,r,k], DP(l,i,k+1,f,boxes) + DP(i+1,r-1,0,f,boxes)\big)$

  $return\ f[l,r,k]$

---

727. Minimum Window Subsequence Google

Find the shortest window in S such that T is one of its subsequence.

Solution I: Dynamic-Programming, quite like Longest Common Subsequence Problem.

$f[i,j] = index, T[0:i]\ is\ subsequence\ of\ S[index:j], f = (m+1) \star (n+1)$

$f[0,j] = j + 1 \star base\ case$

$f[i,j] = f[i-1][j-1]\ if\ T[i-1] == S[j-1]$

$f[i][j] = f[i][j-1]$

$result = S[j, \min(j - f[m][j] + 1)]\ \forall\ j \in [0,n]$

Solution II: Two Pointers

$for(i = 0 \to m-n)$:

$if(S[i] == T[0])$:

$p = i, q = 0$

$for(; p \to m-1, q \to n-1)$:

$if(S[p] = T[q]): q = q + 1$

$if(q = len(T)\ \&\ p - i + 1 < minLen)$:

$minLen = p - i + 1$

$minIndex = i$

$return\ minIndex = -1 \to "" : S[minIndex, minIndex + minLne - 1]$

---

551. Student Attendance Record I **Google**

552. Student Attendance Record II

568. Maximum Vacation Days **Google**

Intuition

Define $f[week = i, city = j]$ as the maximal days spent until this $i^{th}\ week, j^{th}\ city$

$Param: days[city, week], flights[city_a, city_b] = 1\ or\ 0$

$for(city: 0 \to cityNum - 1)$:

$if(city = 0\ or\ flights[0, city] > 0)$:

$f[0, city] = days[city][0]$

$canMake[0, city] = true$

$\star\ if\ at\ the\ (w-1)^{th}\ week, he\ can\ make\ to\ city_{prev}$

$\star\ and\ if\ there\ is\ a\ flight\ [city_{prev} \to city_{curr}]$

$\star\ thus\ he\ can\ make\ to\ city_{curr}, or\ he\ can\ choose\ to\ stay\ at\ the\ same\ city: city_{curr} = city_{prev}$

$for(w: 1 \to weekNum - 1)$:

$for(city_{curr}: 0 \to cityNum - 1)$:

$for(city_{prev}: 0 \to cityNum - 1)$:

$if\left(canMade\left[w-1, city_{prev}\right]\ \&\ \left(flights\left[city_{prev}, city_{curr}\right] > 0\ ||\ city_{prev} = city_{curr}\right)\right)$:

$f[w, city_{curr}] = \max\left(f\left[w-1, city_{prev}\right]\right) + days[city_{curr}, w]$

$$canMade[w, city_{curr}] = true$$

$re = \max(f[w-1, city_k]), \forall k \in [0, cityNum-1]$

---

746. Min Cost Climbing Stairs Google

Question: there are n steps, find the minimal cost to go to the top($n-1$).

$f[i] = cost[i] + \min(f[i-1], f[i-2])$

$f[i]$: *the minimal cost to have the ability to climb higher steps*

$result = \min(f[n-1], f[n-2])$

---

21 Points Game Google

Question: Given 10 cards (1 - 10), if the sum value of handcard do not exceed 17, you may need to fetch one more card, give out the probability that the sum value will exceed 21 points after having one more card.

Solution: define $f[i]$ as the probability of the sum value of hand-cards = i.

$i = 0 \Rightarrow f[0] = 1 \star$ *corner case*

$i = 1 \Rightarrow \{(1)\} \Rightarrow \frac{1}{10} = \frac{f[0]}{10}$

$i = 2 \Rightarrow \{(1,1),(2)\} \Rightarrow \frac{1}{10} \star \frac{1}{10} + \frac{1}{10} = \frac{1}{10} \star \left(\frac{1}{10} + 1\right) = \frac{f[1]+f[0]}{10}$

$i = 3 \Rightarrow \{(1,1,1),(1,2),(3)\} \Rightarrow \frac{1}{10} \star \frac{1}{10} \star \frac{1}{10} + \frac{1}{10} \star \frac{1}{10} + \frac{1}{10} = \frac{1}{10} \star \left(\frac{1}{10} \star \frac{1}{10} + \frac{1}{10} + 1\right) = \frac{f[2]+f[1]+f[0]}{10}$

$i \in [4,10] \Rightarrow f[i] = \frac{(f[i-1] + f[i-2] + \cdots + f[1] + f[0])}{10}$

$i = 11 \Rightarrow \{(10,1),(9,2),(8,3),(7,4),(6,5)\} \Rightarrow \frac{f[10]+f[9]+f[8]+f[7]+f[6]}{10}$

$i = 12 \Rightarrow \{(11,1),(10,2),(9,3),(8,4),(7,5),(6,6)\} \Rightarrow \frac{f[11]+f[10]+f[9]+f[8]+f[7]+f[6]}{10}$

$i \in [11,20] \Rightarrow f[i] = (f[i-10] + f[i-2])$

$$f[i] = \frac{\left(\sum_{k=0}^{k=i-1} f[k]\right)}{10} \ \forall i \in [1,10]$$

$$f[i] = \frac{\left(\sum_{k=i-\frac{i}{2}}^{k=i-1} f[k]\right)}{10}, \forall i \in [10, \infty)$$

$$p(> 21) = 1 - \sum_{i=17}^{i=21} f[i]$$

---

494. Target Sum Google Facebook

You are given a list of non-negative integers, $a_1, a_2, \ldots, a_n$, and a target, S. Now you have 2 symbols + & −. For each integer, you should choose one from + & − as its new symbol.
Find out how many ways to assign symbols to make sum of integers equal to target S.

Intuition

Define $f[i, v]$ as the number of combinations of first *i* numbers that sum up to *j*

```
findTargetSumWays(nums, S):
    f[i, v] = [len(nums) * 2001]
    return DP(nums, i = n − 1, v = 0, S, memo)

int calculate(nums, i, v, S, memo):
    if(i = −1):
        return sum = S → 1: 0
    else:
        if(f[i, v + 1k] ≠ nil): return f[i, v + 1k]
        add ≔ DP(nums, i − 1, v + nums[i], S, f)
        minus ≔ calculate(nums, i + 1, v − nums[i], S, f)
        f[i, v + 1k] ≔ add + minus
        return f[i, v + 1k]
```

---

361. Bomb Enemy Google

```
for(i = 0 → m − 1):
    for(j = 0 → n − 1):
        if(grid(i,j) == W): left(i,j) = top(i,j) = 0
        elif(grid(i,j) == E): left(i,j) = left(i,j−1) + 1, top(i,j) = top(i−1,j) + 1
```

$else: left_{(i,j)} = left_{(i,j-1)}, top_{(i,j)} = top_{(i-1,j)}$

$for(i = m-1 \to 0)$:

$for(j = n-1 \to 0)$:

$if\big(grid_{(i,j)} == W\big): right_{(i,j)} = bottom_{(i,j)} = 0$

$elif\big(grid_{(i,j)} == E\big): right_{(i,j)} = right_{(i,j+1)} + 1, bottom_{(i,j)} = bottom_{(i+1,j)} + 1$

$else: right_{(i,j)} = right_{(i,j+1)}, bottom_{(i,j)} = bottom_{(i+1,j)}$

$result = \max\big(left_{(i,j)} + top_{(i,j)} + right_{(i,j)} + bottom_{(i,j)}\big)\ \forall\, i \in [0, m-1], \forall\, j \in [0, n-1]$

---

486. Predict the Winner Google

Question: player 1 picks one of the numbers from either end of the array followed by the player 2 and then player 1 and so on.

Solution: $f[i][j] = \max(player_1'\, s\ scores - player_2'\, s\ scores)\ \ for\ nums[i:j]$, if $f[0:n-1] \geq 0$, then player1 wins, otherwise, player2 wins. **Time Complexity**: $O(n^2)$

$bool\ predict(nums)$

$f = R^2(type = int)$

$return\ DP(nums, 0, nums.length - 1, f)$

$DP(A, s, e, f)$:

$if(s = e): return\ A[s]$

$if(f[s,e] \neq nil): return\ f[s,e]$

$f[s,e] = \max\big(A[s] - DP(A, s+1, e, f), A[e] - DP(A, s, e-1, f)\big)$

$return\ f[s,e]$

The above solution can be improved to cost $O(n)$ space.

$bool\ predict(nums)$

$f = R^n$

$for(s = n \to 0)$:

$for(e = s+1 \to n-1)$:

$f[e] = \max(nums[s] - f[e], nums[e] - f[e-1])$

$return\ f[n-1] \geq 0$

---

689. Maximum Sum of 3 Non-Overlapping Subarrays Google Facebook

Question: find such 3 non-overlapping subarrays with length of k make up the largest sum.

$left[i] = x, means\ sum[x-k+1:x]\ is\ the\ largest\ 1^{st}\ subarray\ in\ range[0:i]$

$right[i] = x, means\ sum[x:x+k-1]\ is\ the\ largest\ 3^{rd}\ subarray\ in\ range[i:n-1]$

Since the starting index of 2^{nd} subarray is in the range of $[k:n-2\star k]$. So we just need to enumerate the starting index of 2^{nd} subarray, and find the "best" i which makes the maximal sum of 3 non-overlapping subarrays.

$leftMaxSum = sum[0:k-1]$

$left[k-1] = k-1$

$for(i = k \to n-1)$:

    $if(sum[i-k+1:i] > leftMaxSum)$:

        $leftMaxSum = sum[i-k+1:i]$

        $left[i] = i$

    $else: left[i] = left[i-1]$

$rightMaxSum = sum[n-k:n-1]$

$right[n-k] = n-k$

$for(i = n-k-1 \to 0)$:

    $if(sum[i:i+k-1] > rightMaxSum)$:

        $rightMaxSum = sum[i:i+k-1]$

        $right[i] = i$

    $else: right[i] = rigth[i+1]$

$for(i = k \to n-2\star k)$:

    $l = left[i-1], r = right[i+k]$

    $window_{curr} = sum[i:i+k-1]$

    $window_{left} = sum[l-k+1:l]$

    $window_{right} = sum[r:r+k-1]$

    $if\big(window_{curr} + window_{left} + window_{right} > max\big)$:

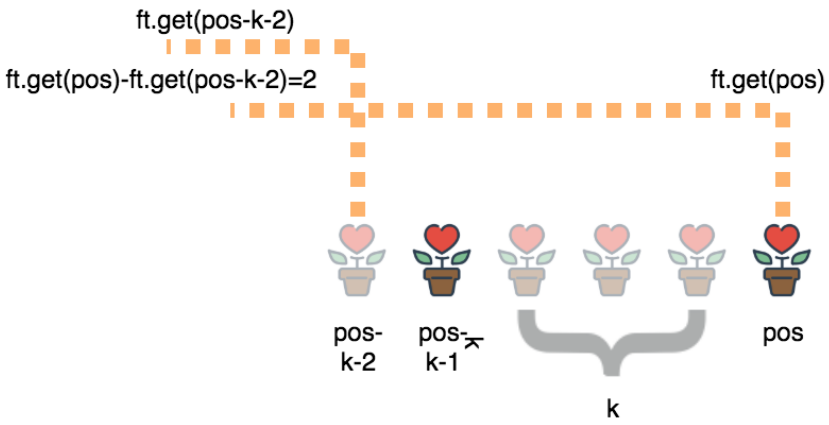
        $max = window_{curr} + window_{left} + window_{right}$

        $result = [l-k+1, i, r]$

$return\ result$

Tree Set & Tries & Linked List

683. K Empty Slots Google



Solution: Tree Set

$if(flowers.length = 1 \ \& \ k = 0): return\ 1$

$n = flowers.length$

$set = new\ TreeSet\langle int\rangle();$

$for(i = 0 \rightarrow n - 1):$

    $set.add(flowers[i])$

    $l = set.lower(flowers[i])$

    $r = set.higher(flowers[i])$

    $if(l \neq nil \ \& \ r \neq nil \ \& \ (r - flowers_i - 1 = k \ or \ flowers_i - l - 1 = k)):$

        $return\ i + 1$

$return - 1$

Solution II: Fenwick Tree(BIT)

$if(flowers.length == 1 \ \& \ k == 0): return\ 1$

$n = flowers.length$

$FenwickTree\ ft$

$array_{1d}\ has$

$for(i = 0 \rightarrow n - 1):$

    $pos = flowers[i]$

    $has \cdot flower[pos] = 1$

    $ft.insert(pos, 1)$

    $left = pos - k - 1$

$right = pos + k + 1$

$if(left \geq 0 \ \& \ has[left] == 1 \ \& \ ft.getSum(pos) - ft.getSum(left - 1) == 2)$:

$return\ i + 1$

$if(right \geq 0 \ \& \ has[right] == 1 \ \& \ ft.getSum(right) - ft.getSum(pos - 1) == 2)$:

$return\ i + 1$

$return\ -1$

$class\ FenwickTree$

$int\ n$

$array_{1d}\ sum$

$lowbit(x)$:

$return\ x\&(-x)$

$insert(pos, val)$:

$while(pos \leq n)$:

$sum[pos] = sum[pos] + \ val$

$pos = pos + lowbit(pos)$

$getSum(pos)$:

$result = 0$

$while(pos > 0)$:

$result = result + sum[pos]$

$pos = pos - lowbit(pos)$

$return\ result$

Follow Ups:

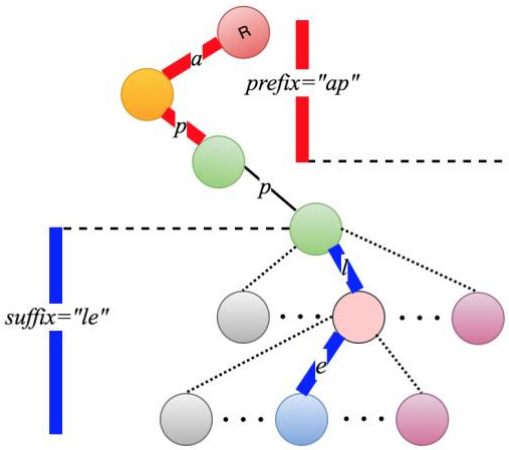
Last Day of K Empty Slots

Last Day of K Consecutive Flowers

745. Prefix and Suffix Search Trie

$$static\ class\ TrieNode: \{next^{TrieNode}[26], word^{string}\}$$

$$Find(prefix = 'ap', suffix = 'e')$$



When building the trie based on each word, attach each left node of trie with the word.

Then for given input of prefix & suffix, how to find the corresponding word?

Idea behind: firstly, search prefix and get the deepest node from built trie. Second, starting from that node and recursively search in trie until the word contained in the node exactly match the suffix.

$int\ f(prefix, suffix)$:

$Node\ p = findPrefix(prefix)$

$if(p = nil)$: $return -1$ ⋆ *means no such word that starts with* prefix *and end up with* suffix

$re = -1$

$DFS\left(p, s2w^{map(str \to int)}, suffix^{string}\right)$

$void$ DFS$(Node\ p, s2w, suffix)$

$if(p.word \neq nil)$:⋆ *in case p is a leaf node*

$if(p.word.$ *end up with* suffix *and* $s2w[p.word] > re)$: $re = s2w[p.word]$

$return$

$for(i = 0 \to 25)$:

$if(p.next[i] \neq nil)$:DFS$(p.next[i], s2w, suffix)$

---

143. Reorder List

Given a singly linked list $L = L_0 \to L_1 \to \cdots \to L_{n-1} \to L_n$

Reorder it to $L = L_0 \to L_{n-1} \to L_1 \to L_{n-1} \to \cdots$

Intuition:

Firstly, apply fast & slow pointer to halve the linked list. Second, merge the two halves.

237. Delete Node in a Linked List Microsoft Apple Adobe

Write a function to delete a node (except the tail) in a singly linked list, **given only access to that node**.

Intuition: Swap with Next Node

Since we do not have access to the node before the one we want to delete, we cannot modify the next pointer of that node in any way. Instead, we have to replace the value of the node we want to delete with the value in the node after it, and then delete the node after it.

Since in this problem, the given pointer **must not be the last one of linked list**, so this approach is applicable.

```
void deleteNode(ListNode node):
    node.val := node.next.val
    node.next := node.next.next
```

86. Partition List

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x. You should preserve the original relative order of the nodes in each of the two partitions.

Intuition & Algorithm

```
ListNode partition(ListNode head, int x):
    ListNode p := ListNode(−1)
    ListNode q := ListNode(−1)
    ListNode rp := p, rq := q
    while(head ≠ nil):
        if(head.val < x):
            rp.next := head
            rp := rp.next
        else: rq.next := head, rq := rq.next
    rp.next := q.next
    q.next := nil
    return p.next
```

234. Palindrome Linked List Facebook Amazon IXL

Determine if the given linked list is palindromic or not.

Intuition & Algorithm

First, apply slow & fast pointers to find the middle node(s). Second, reverse the first halved list. Third, check each node of two lists one by one.

2. Add Two Numbers Microsoft Amazon Bloomberg Airbnb Adobe

Given two non-empty linked lists representing two non-negative integers, Add the two numbers and return it as a linked list.

Algorithm

Reverse the two-linked list first, then sum them up one by one, finally reverse again.

$reverse(l_1), reverse(l_2)$

$p = l_1, q = l_2$

$while(p \neq nil \ \& \ q \neq nil)$:

    $p.val \coloneqq p.val + q.val$

    $p \coloneqq p.next, q \coloneqq q.next$

$while(q \neq nil): p.next \coloneqq q, q \coloneqq q.next, p \coloneqq p.next$

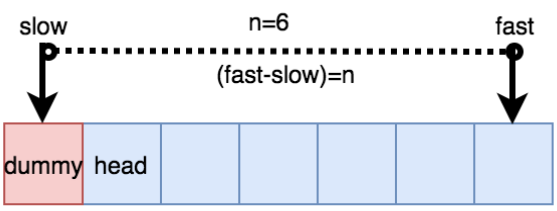
$return \ l_1$

---

19. Remove Nth Node from End of List Google

Algorithm

slow & fast pointers, let the “fast” pointers go n steps ahead, then let the “slow” pointer and “fast” pointer go simultaneously. Once the “fast” pointer reach the end(null), what the “slow” pointers pointing is the node we want to delete. Also, in order to delete a node, we need to keep track of its previous node.



$dummy = Node(-1), slow = dummy, fast = dummy$

$for(i = 0 \to n - 1): fast = fast.next$ ⋆ *fast pointer move ahead by n steps*

$if(fast = nil): return\ nil$

$prev = slow$

$while(fast \neq nil)$:

    $fast = fast.next$

    $prev = slow$

    $slow = slow.next$

$prev.next = slow.next$

$return\ dummy.next$

B-Search & Array & 2 Pts & Reservoir Sampling

Find a local minimum in an array Google

Whether a circular array of relative indices is composed of a single, complete cycle Google

For example, $[2,2,-1]$ forms a single, complete cycle. We start at index of 0, the next index is $(nums\ [0]\ +\ 0)\%3 = 2$, the next index is$(2 + nums[2])\%3 = 1$, the next index is$(1 + nums[1])\%3 = 0, which\ forms\ a\ cycle\ 0 \to 2 \to 1 \to 0$

424. Longest Repeating Character Replacement Pocket Gems

Given a string that consists of only uppercase English letters, you can replace any letter in the string with another letter **at most k times**. Find the length of a longest substring containing all repeating letters you can get after performing the above operations.

Intuition & Algorithm

Maintain a window $[\ell, r]$ s.t. $\mathcal{C}$ is the most frequent character and the corresponding frequency is $most$. Also, there are $(r - \ell + 1 - most) \leq k$

$\ell \coloneqq 0, r \coloneqq 0$

$for(r \to len(s) - 1)$:

    $most \coloneqq \max(most, count[s[r]] \coloneqq count[s[r]] + 1)$

    $while(\ell \leq r\ \&\ (r - \ell + 1 - most) > k)$

$count[s[\ell]] \coloneqq count[s[\ell]] - 1$

$\ell \coloneqq \ell + 1$

$re \coloneqq \max(re, r - \ell + 1)$

$return\ re$

---

215. Kth Largest Element in an Array

Facebook | Microsoft | Amazon | Bloomberg | Apple | Pocket Gems

Intuition & Algorithm

Quick-Select

$quickSelect(\mathcal{A}, \ell, r, k)$:

$if\ (l > r): return$

$pidx \coloneqq partition(\mathcal{A}, \ell, r)$

$if\ (pidx = \ell + k - 1): return\ arr_{index}$

$elif\ (pidx - k + 1 > \ell)$:

$return\ quickSelect(\mathcal{A}, pidx + 1, r, k - (pidx - l + 1))$

$else: return\ quickSelect(\mathcal{A}, l, pidx - 1, k)$

$partition(\mathcal{A}, l, r)$:

$pivot = \mathcal{A}[r], wall = l$

$for(i = l \to r - 1)$:

$if\ (\mathcal{A}[i] < pivot)$:

$swap(\mathcal{A}, i, wall)$

$wall \coloneqq wall + 1$

$swap(\mathcal{A}, wall, r)$

$return\ wall$

Google Interview Question (like Reconstruct Queue by Height)

Given a "left-side" order array, for example, the original array is $array = [5,3,1,2,4]$, and the "left-side" order goes like $left = [0,1,2,2,1]$, where $left[i]$ suggest how many numbers in range of $[0: i - 1]$ which are greater than $array[i]$.

462. Minimum Moves to Equal Array Elements II

Given a non-empty integer array, find the minimum number of moves required to make all array elements equal, where a move is incrementing a selected element by 1 or decrementing a selected element by 1.

Intuition & Algorithm

The best strategy is make all integers in the given array equal to median value among all integers. We can find the median value by quick-select, and sum all difference between each integer and median value.

$median \coloneqq quickSelect\left(A, \frac{len(A)}{2}\right)$

$for(i = 0 \to len(A) - 1): re \coloneqq re + |A[i] - median|$

$return\ re$

---

311. Sparse Matrix Multiplication Facebook LinkedIn

Given two sparse matrix A & B, return the product of them.

$result_{(i,j)} = A_{(i,k)} \star B_{(k,j)}, \forall i \in [0, A.row - 1], \forall k \in [0, A[0].col - 1], \forall j \in [0, B[0].col - 1]$

```
for(i = 0 → A.length − 1):
    for(k = 0 → A[0].length − 1):
        if(A_(i,k) ≠ 0):
            for(j = 0 → B[0].length − 1):
                result_(i,j) = result_(i,j) + A_(i,k) ⋆ B_(k,j)
```

---

278. First Bad Version Facebook

Key Take-away:

$while(\ell < r) \to \ell \coloneqq mid + 1, r \coloneqq mid$

$while(\ell \leq r): \to \ell \coloneqq mid + 1, r \coloneqq mid - 1$

$int\ 1^{st} badVersion(int\ n)$:

```
int l := 1, r := n, re = −1
while(l ≤ r):
    mid := (l+r)/2
    if(isBadVersion(mid)): re = mid, r := mid − 1
    else: l := mid + 1
return re
```

---

15. 3Sum Facebook Microsoft Amazon Bloomberg Adobe Works Application

Given an array $\mathcal{A}$ of n integers, are there elements a, b, c in $\mathcal{A}$ such that $a + b + c = 0$? Find all unique triplets in the array which gives the sum of zero.

Intuition & Algorithm

```
List[List[int]] 3sum(int[ ] 𝒜):
    𝒩 := len(𝒜)
    sort(𝒜)
    for(i = 0 → 𝒩 − 1):
        if (i ≥ 1 & 𝒜[i] = 𝒜[i − 1]): continue (skip duplicates)
        ℓ := i + 1, r := 𝒩 − 1
        while(ℓ < r):
            if (𝒜[ℓ] + 𝒜[r] + 𝒜[i] = 0):
                re.add([𝒜[i], 𝒜[ℓ], 𝒜[r])
                p := ℓ + 1
                while(p < r & 𝒜[p] = 𝒜[ℓ]): p := p + 1(skip duplicates)
                q := r − 1
                while(p < q & 𝒜[q] = 𝒜[r]): q := q − 1(skip duplicates)
                ℓ := p, r := q
            elif (𝒜[ℓ] + 𝒜[r] + 𝒜[i] > 0): r := r − 1
            else: ℓ := ℓ + 1
    return re
```

---

825. Friends of Appropriate Ages Facebook

Some people will make friend requests. The list of their ages is given and $ages[i]$ is the age of the ith person. Person A will NOT friend request person B $(B \neq A)$ if any of the following conditions are true:

$$age[B] \leq 0.5 \star age[A] + 7$$
$$age[B] > age[A]$$
$$age[B] > 100 \;\&\; age[A] < 100$$

Intuition & Algorithm

Instead of processing all 20000 people, we can process pairs of $(age, count)$ representing how many people are that age. Since there are only 120 possible ages, this is a much faster loop. For each pair $(ageA, countA)$, $(ageB, countB)$, if the conditions are satisfied with respect to age, then $countA \star countB$ pairs of people made friend requests.

If $ageA = ageB$, then we overcounted: we should have $countA \star (countA - 1)$ pairs of people making friend requests instead, as you cannot friend request yourself.

$int\ numFriendRequests(int[\,]\ ages)$:
    $int[\,]\ counts = [0]: size = 121$
    $for(int\ age: ages): counts[age] \coloneqq counts[age] + 1$
    $re = 0$
    $for(age_A = 0 \rightarrow 120)$:
        $for(age_B = 0 \rightarrow 120)$:
            $if\big(age_A \star 0.5 + 7 \geq age_B \;||\; age_A < age_B \;||\; \sim(age_A < 100 \;\&\; 100 < age_B)\big): continue$
            $re \coloneqq re + counts[age_A] \star counts[age_B]$
            $\star$ *since people cannot friend himself*
            $\star$ *so we have to deduce* $counts[age_A]$ *from result*
            $if(age_A = age_B): re \coloneqq re - counts[age_A]$
    $return\ re$

---

548. Split Array with Equal Sum Alibaba

Given an array with n integers, you need to find if there are triplets (i, j, k) which satisfies following conditions:

$$\sum_{0}^{i-1} a[w] = \sum_{i+1}^{j-1} a[x] = \sum_{j+1}^{k-1} a[y] = \sum_{k+1}^{n-1} a[z], \forall (i, j, k) \in [0, n-1]$$

$sum[i] = sum[i-1] + a[i]$

$for(m = 3 \to n-4)$:

  $for(l = 1 \to m-2)$:

    $\star\star\ block^1: [0 \to l-1], block^2: [l+1, m-1]$

    $block^2 = sum[m-1] - sum[l], block^1 = sum[l-1]$

    $if(block^1 = block^2): set.add(block^1)$

  $for(r = n-2 \to m-2)$:

    $\star\star\ block^3: [m+1 \to r-1], block^2: [r+1, n-1]$

    $block^4 = sum[n-1] - sum[r], block^3 = sum[r] - sum[m-1]$

    $if(block^3 = block^4\ and\ set[block^3] \neq null): return\ true$

744. Find Smallest Letter Greater Than Target LinkedIn

Given a list of sorted characters letters containing only lowercase letters, and given a target letter target, find the smallest element in the list that is larger than the given target. Letters also wrap around. For example, if the target is target = 'z' and letters = ['a', 'b'], the answer is 'a'.

Intuition & Algorithm

$char\ nextGreatestLetter(char[\ ]\ letters, char\ target)$:

  $n \coloneqq len(letters)$

  $l \coloneqq 0, r \coloneqq n-1, t \coloneqq target - a + 1$

  $while(l < r)$:

    $int\ of = \left(letters\left[\frac{l+r}{2}\right] - a + 1\right) mod\ 26$

    $if(of \leq t): l \coloneqq m+1$

    $else: r \coloneqq m$

  $if(lettes[l] > target): return\ letters[l]$

  $else: return\ letters[(l+1)\ mod\ n]$

41. First Missing Positive

Given an unsorted integer array, find the smallest missing positive integer.

$wall = 0$

$for(i = 0 \to n-1)$:

$if(A[i] > 0): swap(A, wall, i), wall := wall + 1$

$for(i = 0 \to wall - 1):$

$index = |A[i]| - 1$

$if(index < wall \ \& \ A[index] > 0): A[index] \star= -1$

$for(i = 0 \to wall - 1):$

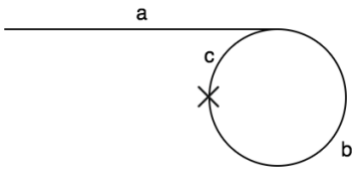
$if(A[index] > 0): return\ index + 1$

$return\ wall + 1$

---

141. Linked List Cycle Microsoft Amazon Bloomberg Yahoo

Solution: "slow & fast" pointers.



Theorem: the fast & slow pointer will eventually meet at the cross point.

$boolean\ hasCycle(ListNode\ head):$

$if(head = nil): return\ false$

$slow = head, fast = head$

$while(fast \neq nil):$

$slow = slow.next, fast = fast.next$

$if(fast \neq nil): fast = fast.next$

$if(fast \neq nil \ \& \ fast = slow): return\ true$

$return\ false$

---

849. Maximize Distance to Closest Person Google

In a row of seats, 1 represents a person sitting in that seat, and 0 represents that the seat is empty. There is at least one empty seat, and at least one person sitting. Alex wants to sit in the seat such that the distance between him and the closest person to him is maximized. Return that maximum distance to closest person.

Intuition & Algorithm

$int[\,]\ left, int[\,]\ right, re = -2^{31}$

$one = -1$

$for(i = 0 \to n - 1)$:

    $if(seats[i] = 0): left[i] = one$

    $else\ one = i$

$one = -1$

$for(i = n - 1 \to 0)$:

    $if(seats[i] = 0): left[i] = one$

    $else\ one = i$

$for(i = 0 \to n - 1)$:

    $if(seats[i] = 0)$:

        $if(i = 0): re = \max(re, right[i] - i)$

        $elif(i = n - 1): re = \max(re, i - left[i])$

        $else: re = \max(re, min(right[i] - i, i - left[i]))$

$return\ re$

---

396. Rotate Function **Amazon**

Given an array of integers A and let n to be its length. Assume B_k to be an array obtained by rotating the array A_k positions clock-wise, we define a "rotation function" F on A as follow:

$$F_k = 0 \star B_k[0] + 1 \star B_k[1] + \cdots + (n-1) \star B_k[n-1]$$

Calculate the maximum value of $F_0, F_1, \cdots, F_{n-1}$, we can find that

$$F_0 = 0 \star A_0 + 1 \star A_1 + 2 \star A_2 + 3 \star A_3$$

$$F_0 = 0 \star A_3 + 1 \star A_0 + 2 \star A_1 + 3 \star A_2 = F_0 + sum(A) - n \star A[n-i]$$

$$F_i = F_{i-1} + sum(A) - n \star A[n-i]$$

$sum = 0, result = 0$

$for(i = 0 \to n - 1)$:

    $sum \coloneqq sum + a[i], re = re + i \star a[i]$

$RF = re$

$for(i = 1 \to n)$:

    $RF \coloneqq RF + sum - n \star a[n-i]$

    $re = \max(re, RF)$

$return\ re$

---

845. Longest Mountain in Array **Google**

Let's call any (contiguous) subarray B (of A) a mountain if the following properties hold:

$$len(A) \geq 3$$

$$A[i] < A[i+1] < \cdots < A[k-1] < A[k] > A[k+1] > \cdots > A[j], \exists k \in [1, len(B) - 2]$$

$n = len(A)$

$int[\,]\ left, int[\,]\ right, re = -2^{31}$

$for(i = 0 \to n - 1)$:

    $if(i = 0\ \&\ A[i-1] \geq A[i])$: $left[i] = 0$

    $else$: $left[i] = left[i-1] + 1$

$for(i = n - 1 \to 0)$:

    $if(i = n - 1\ \&\ A[i] \leq A[i+1])$: $right[i] = 0$

    $else$: $right[i] = right[i+1] + 1$

$for(i = 1 \to n - 2)$:

    $if(left[i] > 0\ \&\ right[i] > 0)$: $re = \max(re, left[i] + right[i] + 1)$

$return\ re = -2^{31} \to 0 : re$

---

768. Max Chunks to Make Sorted II Google

769. Max Chunks to Make Sorted Google

Intuition & Algorithm

clone the array to another array "sorted" and then sort the copied array, in case of $\sum_i^j sorted[k] = \sum_i^j A[k]$, then $[i: j]$ forms a good chunk.

$int[\,]\ cp = A.clone$

$sort(cp)$

$chunks \coloneqq 0$

$for(i = 0 \to len(A) - 1)$:

    $dif \coloneqq dif + A[i] - cp[i]$

    $if(dif = 0)$: $chunks \coloneqq chunks + 1$

$return\ chunks$

---

360. Sort Transformed Array Google

Given a sorted array of integers *nums* and integer values a, b & c. Apply a quadratic function of the form $f(x) = ax^2 + bx + c$ to each element x in the array. The returned array must be in sorted order.

Intuition & Algorithm

Apply math formula, axis of symmetry $= -\frac{b}{2\star a}$

$$if\ a > 0, \forall x \in \left[-\infty, \frac{-b}{2 \star a}\right], x \downarrow \rightarrow f(x) \uparrow$$

$$\forall x \in \left[\frac{-b}{2 \star a}, +\infty\right], x \uparrow \rightarrow f(x) \uparrow$$

$x = \left\lceil -\frac{b}{2 \star a} \right\rceil, index = bsearch(A, x)$

$index = index < 0 ? (-index - 1) : index$

$q = index, p = q - 1, e = 0$

$while(p \geq 0\ and\ q < n)$:

    $v_p = a \star (A[p])^2 + b \star A[p] + c$

    $v_q = a \star (A[q])^2 + b \star A[q] + c$

    $if(a > 0)$:

        $if\left(v_p \leq v_q\right): re[e^{++}] = v_p, p = p - 1$

        $else: re[e^{++}] = v_q, q = q - 1$

    $else$:

        $if\left(v_p \geq v_q\right): result[e^{++}] = v_p, p = p - 1$

        $else: result[e^{++}] = v_q, q = q - 1$

    $while(p \geq 0): re[e^{++}] = a \star (A[p])^2 + b \star A[p] + c, p = p - 1$

    $while(q < n): re[e^{++}] = a \star (A[q])^2 + b \star A[q] + c, q = q + 1$

    $if(a < 0): reverse(re)$

    $return\ re$

---

487. Max Consecutive Ones II Google

Given a binary array, find the maximum number of consecutive 1s if you can flip at most one 0.

Solution: two pointers, keep track of number of zeros within the sliding window, if the number of zeros exceeds 1, reduce the size of window by increasing the left boundary.

$l = 0, r = 0, maxlen = 0, zeros = 0$

$for(r = 0 \to n - 1)$:

    $if(a[r] = 0): zeros = zeros + 1$

    $while(l \le r \ \&\ zeros > 1)$:

        $if(a[l++] == 0): zeros = zeros - 1$

    $if(maxlen < r - l + 1): maxlen = r - l + 1$

$return\ maxlen$

Follow Up: if the input numbers come in one by one as an infinite stream

Solution: store the length of previous and current consecutive 1's (separated by the last 0) as $pre(default: -1)$ & $curr(default: 0)$, respectively.$11 \cdot pre \cdot 110 11 \cdot curr \cdot 11$

$pre = -1, curr = 0, maxlen = 0$

$for(instream.hasNext)$:

    $x = instream.next$

    $if(x = 0): pre, curr = curr, 0$

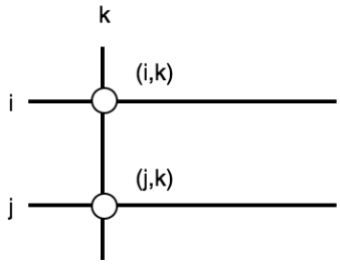
    $else: curr = curr + 1$

    $maxlen = \max(maxlen, pre + 1 + curr)$

---

750. Number of Corner Rectangles Google Facebook

Corner Rectangle: four corners are filled with ones, and both depth & width are greater than 1.



$result = 0$

$for(i = 0 \to m - 2)$:

    $for(j = 0 \to m - 1)$:

        $counter = 0$

        $for(k = 0 \to n - 1)$:

            $if\big(grid_{(i,k)} = 1\ and\ grid_{(j,k)} = 1\big): counter = counter + 1$

        $result = result + \frac{\big(counter \star (counter - 1)\big)}{2}$

Convulsion Word **Google**

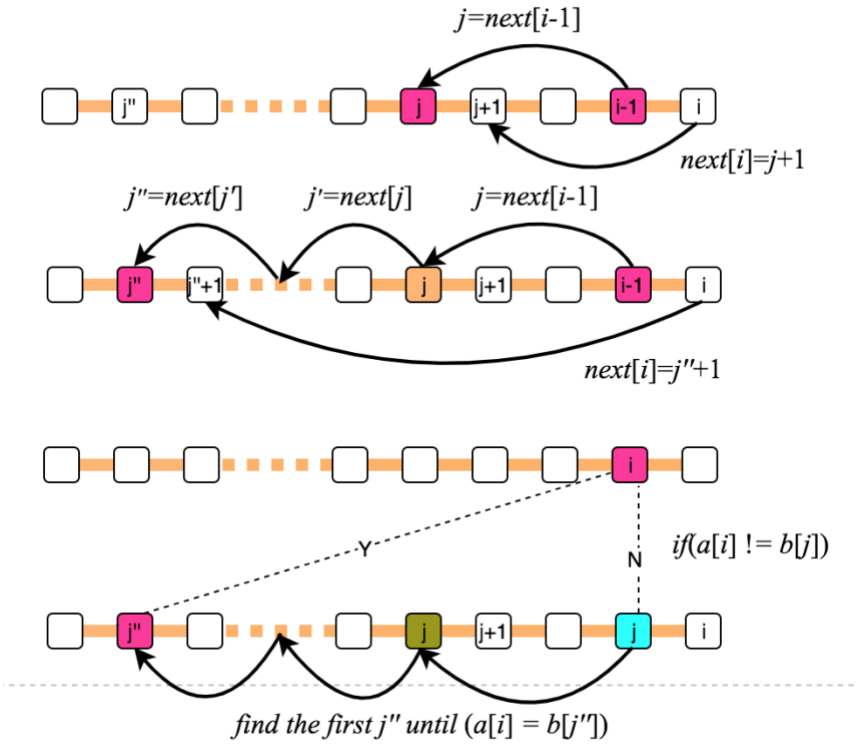
For example, hello ($normal\ word$) ⇒ heeellooo ($convulsion\ word$)

Longest Matched Subfix and Prefix/ String Match - KMP **Google**

28. Implement strStr **Facebook** **Microsoft** **Apple** **Pocket Gems**

Question: given two arrays, find the longest subfix from first array that matched the prefix of second array. Ex.$A = [1,2,3,4,2,3,4,5], B = [2,3,4,5,7]$, the "red" part is the result.

Solution: KMP



$next[0] = next[1] = 0$

$for(i = 2 \rightarrow B.length - 1)$:

$j = next[i - 1]$

$while(j \neq 0\ and\ B[i - 1] \neq B[j]): j = next[j]$

$next[i] = B[i - 1] = B[j] \rightarrow (j + 1): 0$

$i = 0, j = 0$

$for\big(i < len(A)\big)$:

$while(j \neq 0\ \&\ A[i] \neq B[j]): j = next[j]$

$if\,(A[i] = B[j]): j = j + 1$

$maxLen = \max(maxLen, j)$

Sorted Matrix Series

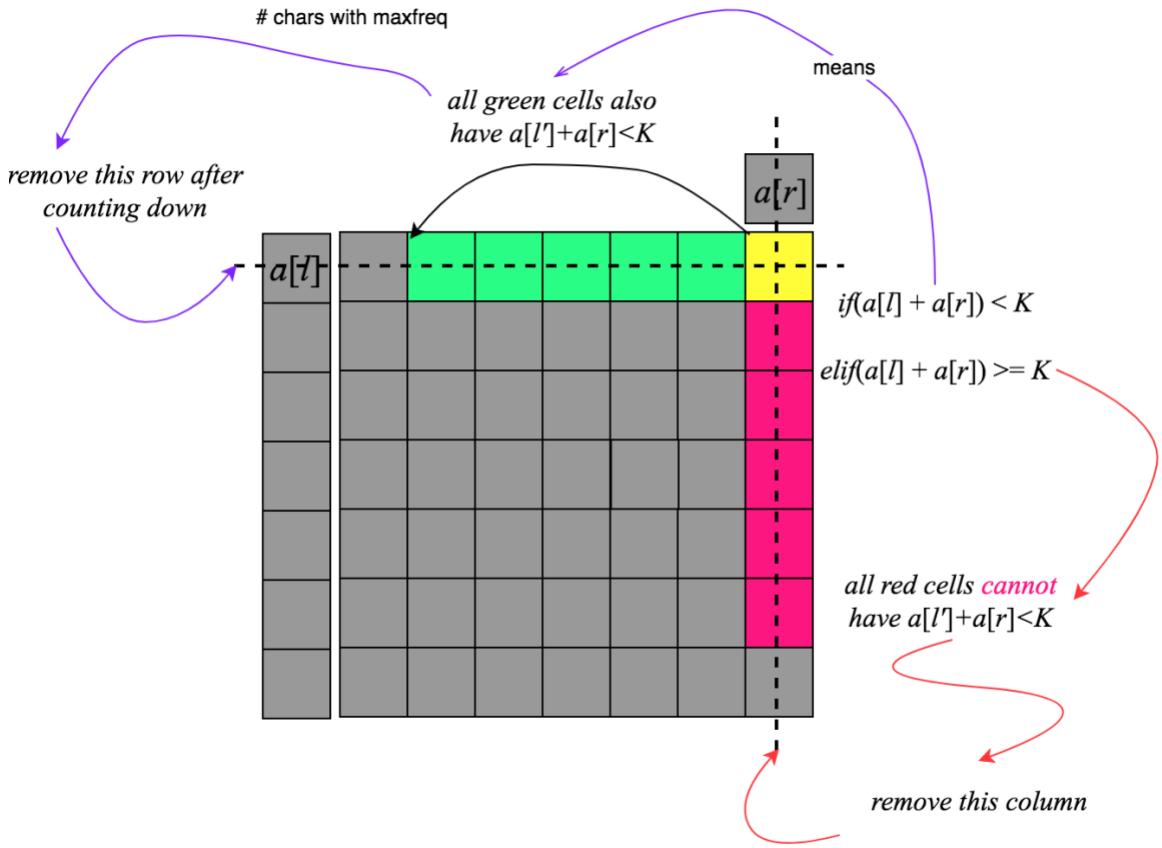
减除左上 加法右上

259. 3Sum Smaller Google

Find t such triplets (i, j, k) that $A[i] + A[j] + A[k] < target$.

Intuition

Sort the array first then we can fix $A[i]$, then this problem can be converted to finding $\#(j, k)$ s.t. $A[l] + A[r] < K (K = target - A[i])$, there are some tricks for counting such combinations. Image all combinations of $(A[l_i] + A[r_i])$ forms a sorted matrix, based on this special matrix, we always start search from the right-up corner $(0, n-1)$.



Algorithm

$re = 0$

$for(i = 0 \to n - 3)$:

    $\ell \coloneqq i + 1, r \coloneqq n - 1$

    $while(\ell < r)$:

        $if(\mathcal{A}[l] + \mathcal{A}[r] < \mathcal{T} - \mathcal{A}[i])$:

            $re = re + (r - \ell)$

            $\ell \coloneqq \ell + 1$

        $else: r \coloneqq r - 1$

$return\ re$

---

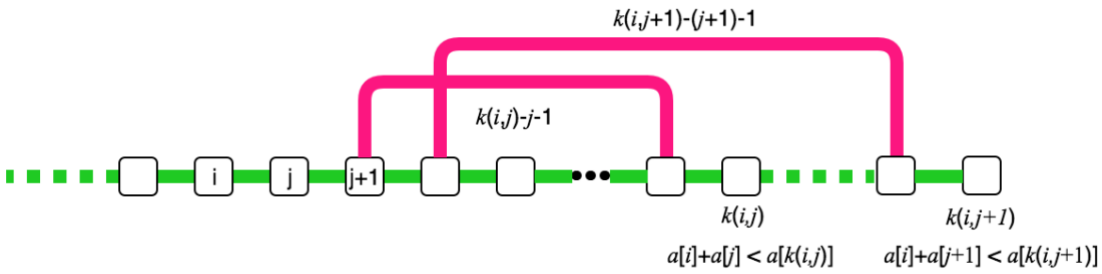
611. Valid Triangle Number Expedia

Find the number of triplets (i, j, k) such that those three numbers can form a valid triangle.

Intuition & Algorithm

Solution: sort the array first, then apply "non-backtrack" two pointers method, meaning once we find the right limit index $k_{i,j}$ for a particular pair (i, j) chosen which dissatisfies the inequality $(a_i + a_j > a_{k_{(i,j)}})$, when we choose a higher value of j for the same value of i, we do not need to start searching for the right limit $k_{i,j+1}$ from the index of $(j + 2)$. Instead, we can start off from the index $k_{i,j+1}$ directly where we left off for the last j choosen.

This holds correct because when we choose a higher value of j, all the a_k, such that $k < k_{(i,j)}$ will obviously satisfy the $(a_i + a_j > a_k)$ for the new value of j chosen.



$re = 0$

$for(i = 0 \to n - 3)$:

    $l \coloneqq i + 1, r \coloneqq i + 2$

    $for(l \to n - 2)$:

        $while(a[r] - a[l] < a[i]): r \coloneqq r + 1$

        $re \coloneqq re + (r - l - 1)$

$return\ re$

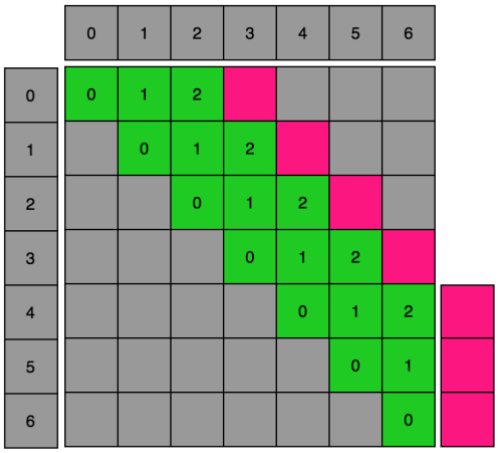
---

719. Find K^{th} Smallest Pair Distance Google

Pair distance is defined as: $pd(x,y) = |x-y|$,

this problem is to find the $k^{th}\ pd(arr_i, arr_j)\ \forall(i,j) \in [0, n-1]$ in given array

Solution: binary-search



$sort(nums)$

$l = \min(nums[i] - nums[i-1])\ , r = nums[n-1] - nums[0], i \in [1, n-1]$

$while(l < r)$:

$mid = \frac{(l+r)}{2}$

$counts = countNoLarger(mid, nums)$

$if\,(counts < k): l := mid + 1$

$else: r := mid$

$return\ l$

$int\ countNoLarger(guess, A)$

$n = len(A)$

$i = 0, j = 1$

$while(i < n\ \&\ j < n)$:

$while(A[j] - A[i] \leq guess): j := j + 1$

$re := re + (j - i - 1)$

$i := i + 1$

---

786. K^{th} Smallest Prime Fraction Pony.ai

both nominator & denominator are chosen from primes array.

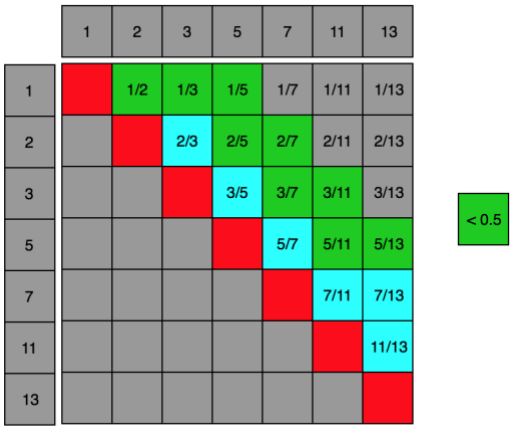
For example, $primes = [1,7,13,19], ugly\ decimals\ are: \frac{1}{19}, \frac{1}{13}, \frac{1}{7}, \frac{7}{19}, \frac{7}{13}, \frac{13}{19}, \cdots$

Algorithm #1

Define a class $Node\ (int\ primes \cdot idx, int\ ne, int\ de)\ +\ PriorityQueue(type = Node)$

Time Cost: $O(k \star log(n))$, where n is the size of array, the heap has up to n elements, which uses $O(\log(n))$ work to perform a pop operation on the heap, we perform $O(k)$ such operations.

Algorithm #2



Apply binary search, initially the searching range is [0,1], and each time we pick up a middle value, and count how many $(primes[i], primes[j])$ $(i < j)$ such that $primes[i]\ /\ primes[j] < mid$. Based on the counting value, we can narrow down the searching range.

Time Cost: $O(n \star \log(w))$, where n is the size of array and w is the is the width (in quantized units) of our binary search, $\frac{(hi - lo)}{1e^{-9}}$ which is 10^9.

$int[]\ k^{th}\ smallest \cdot prime \cdot fraction(primes^{int} = [n], k^{int})$:

$n = len(primes)$

$lo = 0, hi = 1$

$result^{int} = [2]$

$while(hi - lo > e^{-9})$:

$guess = \frac{lo+hi}{2}$

$re = countNoLarger(guess, primes)$

$if(re[0] < k): lo = guess$

$else: hi = guess, result = [re[1], re[2]]$

$return\ result$

$int[]\ countNoLarger(guess^{double}, primes^{int} = [n])$:

$n = len(primes), i = 0, counts = 0, nu = 0, de = 1$

$for(j = 1 \to n - 1)$:

$while\left(i \in [0, j-1]\ \&\frac{primes[i]}{primes[j]} < guess\right): i \coloneqq i + 1$

$counts \coloneqq counts + i$

$if\left(i > 0\ \&\frac{nu}{de} < \frac{primes[i-1]}{primes[j]}\right): nu = primes[i-1], de = primes[j]$

$return\ \{counts, nu, de\}$

---

378. Kth Smallest Element in a Sorted Matrix Google Twitter

for example, $\begin{bmatrix} 1 & 5 & 9 \\ 10 & 11 & 13 \\ 12 & 13 & 15 \end{bmatrix}$ $k = 8$, return 13

Algorithm #1

Apply min-heap, $(x, y) \to push\ (x+1, y)\ \&\ (x, y+1)$

Time Cost #1: $O\big(k \star log(k)\big) = O(mn \star \log(mn)) = O(n^2 \star \log(n^2)) = O(n^2 \star \log(n))$

Algorithm #2

Binary-search, since $k^{th}\ element \in [matrix[0,0], matrix[n-1, n-1]]$, each time we guess a number from that range, and count how many numbers smaller than this guess number. Based on the counting number, we can halve the search range per time.

Time Cost #2: $O\big((m+n) \star \log(\text{max-min})\big)$, if max-min is very small, then Algo #2 is more efficient.

$int\ k^{th}\ smallest(matrix = [n, n]: int)$:

$n = len(matrix)$

$if(k = 1): return\ matrix[0,0]$

$if(k = n^2): return\ matrix[n-1, n-1]$

$l = matrix[0,0], r = matrix[n-1, n-1]$

$while(l < r)$:

$guess = \frac{l+r}{2}$

$counts = countNoLarger(guess, matrix)$

$if(counts < k): l = guess + 1$

$else: r = guess$

$int\ countNoLarger(int\ g, int[\,][\,]\ mat)$: ⋆ *time cost* $= O(m+n)$

$n = len(matrix)$

$i = 0, j = n-1, counts = 0$

$while(i < n\ and\ j \geq 0)$:

$if(matrix[i,j] \leq guess): i := i+1, counts := counts + (j+1)$

$else: j := j-1$

$return\ counts$

---

668. Kth Smallest Number in Multiplication Table

Solution I: Priority-Queue $O(k \cdot logk) \approx O(mn \cdot \log(mn))$

$$pq: \left(PriorityQueue, type = Node \begin{Bmatrix} row \\ col \\ val \end{Bmatrix}, default = (1,1,1) \right)$$

$visited: (type = boolean_{2d}, default = false)$

$counter = 0$

$while(pq.size > 0\ \&\ counter \leq k)$:

$Node\ top = pq.poll()$

$r = top.row, c = top.col, v = top.val$

$if(r+1 < m\ \&\ visited[r+1, c] = false)$:

$pq.offer(new\ Node(r+1, c, (r+1) * c));$

$if(c+1 < n\ \&\ visited[r, c+1] = false)$:

$pq.offer(new\ Node(r, c+1, r * (c+1)));$

$counter = counter + 1$

Binary Search, $O(\log(mn) * m)$

$l = 1, r = m * n$

$while(l < r)$:

$guess = \frac{(l+r)}{2}$

$co = count(guess, l, r)$ ★ *count # numbers in matrix* ≤ *mid*

$if(co < k): l = mid + 1$

$else: r = mid$

$return\ r$

$int\ count(val, m, n)$:

$counter = 0$

$for(i = 1 \rightarrow m)$:

$c = \min\left(\frac{val}{i}, n\right)$ ★ $\left[(i \star 1), (i \star 2), \cdots \left(i \star \frac{val}{i}\right) \leq val\right]$

$counter = counter + c$

$return\ counter$

---

697. Degree of an Array **GE Digital**

degree of this array is defined as the maximum frequency of any one of its elements.

Solution: two Pointers. Firstly, calculate the degree of array, then shrink the subarray with such degree until the shortest length.

$degree = calculateDegree(array)$

$l = 0, r = 0, d = 0, set = \{\phi\}$

$for(; r \rightarrow n - 1;)$:

$count[array[r]] := count[array[r]] + 1$

$if(count[array[r]] > d)$:

$d = count[array[r]], set.clear(), set.add(array[r])$

$elif(count[array[r]] = d): set.add(array[r])$

$if(d = degree)$:

$while(l \leq r\ and\ (\sim set.contains(array[l])\ or\ set.size > 1))$:

$if(set.size > 1): set.remove(array[l])$

$counts[array[l]] := counts[array[l]] - 1$

$l := l + 1$

$if(r - l + 1 < minl): minl := r - l + 1$

$return\ minl$

---

624. Maximum Distance in Arrays Yahoo

Given m arrays, and each array is sorted in ascending order. Now you can pick up two integers from two different arrays, find out the maximum distance.

$min = arrays[0][0], max = arrays[0][arrays[0].length - 1]$

$maxd = 0$

$for(i = 1 \rightarrow arrays.length - 1)$

    $line = arrays[i]$

    $currMin = line[0], currMax = line[line.length - 1]$

    $maxd = \max(|currMin - max|, |currMax - min|, maxd)$

    $min = min(min, currMin)$

    $max = \max(max, currMax)$

$return\ maxd$

---

414. Third Maximum Number Amazon

Algorithm

$first, second, third = null$

$for(i = 0 \rightarrow n - 1)$

    $if\big((first\ or\ second\ or\ thrid) = nums[i]\big): continue$ ⋆ *avoid duplicate*

    $if(first = null\ ||\ nums[i] > first)$:

        $third = second, second = first, first = nums[i]$

    $elif(second = null\ ||\ nums[i] > second)$:

        $third = second, second = nums[i]$

    $elif(third = null\ ||\ nums[i] > third): third = nums[i]$

$return\ third = null \rightarrow first: third$

---

713. Subarray Product Less Than K Yatra

Algorithm

Maintain the window of $a[l: r]$ where $\prod_l^r a[i] < k$

$l = 0, r = 0$

$for(; r < n;)$

$product \coloneqq prodcut \star a_r$

$while(l \leq r \;\&\; product \geq k): product = \frac{product}{a_{l++}}$

$* \; if\,(product[l:r] < k), then\; product[l:r], product[l+1:r], \cdots, product[r] < k$

$result = r - l + 1$

159. Longest Substring with At Most Two Distinct Characters

Given a string, find the length of the longest substring T that contains at most 2 distinct characters. For example, given s = "*eceba*", answer = "*ece*"

340. Longest Substring with At Most K Distinct Characters Google

Algorithm

Maintain two variables *counts* and *distincts*, *counts* used to count occurrence times of each character in range of $[l: r]$, and *distincts* used to keep track of distinct character in the range. Always maintain the window of $a[l:r]$ s.t. $distincts[l:r] = k$

$counts = [256]: int$

$l = r = 0, distincts = 0$

$for(; r < n; r = r + 1)$

$counts[s[r]] \coloneqq counts[s[r]] + 1$

$if\,(counts[s[r]] = 1): distinct \coloneqq distincts + 1$

$while(l \leq r \;\&\; distincts \geq k):$

$counts[s[l]] \coloneqq counts[s[l]] - 1$

$if\,(counts[s[l]] == 0): distinct \coloneqq distincts - 1$

$l \coloneqq l + 1$

$result = \max(result, r - l + 1)$

$result = r - l + 1$

243. Shortest Word Distance I

244. Shortest Word Distance II

245. Shortest Word Distance III LinkedIn

Algorithm #1

$for(i = 0 \to n - 1)$

  $if(words[i] = word_1 \,\&\, map[word_2] \neq nil)$: $dist = \min(dist, i - map[word_2])$

  $if(words[i] = word_2 \,\&\, map[word_1] \neq nil)$: $dist = \min(dist, i - map[word_1])$

  $map[word[i]] = i$

$return\ dist$

Algorithm #2 Constant Space

$p_1 = nil, p_2 = nil$

$for(i = 0 \to n - 1)$

  $if(words[i] = word_1)$: $p_1 = i$

  $if(words[i] = word_2)$: $p_2 = i$

  $if(p_1 \neq nil \,\&\, p_2 \neq nil)$: $dist = |p_1 - p_2|$

$return\ dist$

II

$for(i = 0 \to n - 1)$

  $map[words[i]] = i$

$shortest(word_1, word_2)$

  $l_1 = map[word_1], l_2 = map[word_2]$

  $i = 0, j = 0, dist = 2^{31} - 1$

  $while(i \leq l_1.size \,\&\, j < l_2.size)$:

    $dist = \min(dist, |l_1[i] - l_2[j]|)$

    $if(l_1[i] \leq l_2[j])$: $i = i + 1$

    $else$: $j = j + 1$

$return\ dist$

BT & BST & Iterator

Count Inversions of size three in the given array Affirm

https://www.geeksforgeeks.org/count-inversions-of-size-three-in-a-give-array/

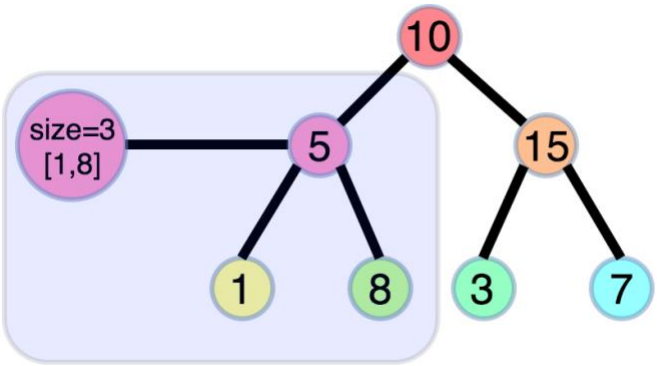
Solution: Fenwick Tree

```
array_1d getRank(nums):
    array_1d temp ← {}
    copy nums ⇒ temp
    sort(temp)
    for(i = 0 → n − 1):
        nums[i] = bsearch(temp, nums[i]) + 1
    return nums

count · triple · inversion(nums):
    rank = getRank(nums)
    left · large = []
    right · small = []
    FenwickTree ft = new FenwickTree
    for(i = 0 → n − 1):
        ⋆⋆ ft. getSum(x) → the number of items smaller than or equals to x so far.
        ⋆⋆ i − ft. getSum(x) → the number of items greater than x so far
        left · large[i] = i − ft. getSum(rank[i])
        ft. insert(rank[i], 1)
    FenwickTree ft = new FenwickTree
    for(i = n − 1 → 0):
        ⋆⋆ ft. getSum(x) → the number of items smaller than or equals to x so far.
        ⋆⋆ ft. getSum(x − 1) → the number of items strictly smaller than x so far.
        right · small[i] = ft. getSum(rank[i] − 1)
        ft. insert(rank[i], 1)
    for(i = 0 → n − 1):
        result = result + left · large[i] ⋆ right · small[i]
```

333. Largest BST Subtree Microsoft

Given a binary tree, find the largest subtree which is a Binary Search Tree (BST), where largest means subtree with largest number of nodes in it.



Intuition

do “check” & “get-size” in one shot.

Algorithm

$class\ Result: \{int\ size, lower, upper\}$

$int\ largestBSTSubtree(root)$:

$maxsize = 1$

$Result\ re = traverse(root)$

$return\ maxsize$

$Result\ traverse(root)$:

$if(root.left = nil\ \&\ root.right = nil): return\ Result(1, root.val, root.val)$

$if(root.left = nil)$:

$Result\ right = traverse(root.right)$

$\star\ if\ root's\ right\ subtree\ is\ not\ bst, then\ root\ isn't\ bst\ anymore$

$if\big(right.size < 0\ or\ \sim(root.val < right.lb)\big): return\ Result(-1,0,0)$

$else$:

$maxsize = \max(maxsize, right.size + 1)$

$return\ Result(right.size + 1, root.val, right.upper)$

$elif(root.right = nil)$:

$Result\ left = traverse(root.left)$

$if\big(left.size < 0\ or\ \sim(root.val > left.ub)\big): return\ Result(-1,0,0)$

$else$:

$maxsize = \max(maxsize, left.size + 1)$

$return\ Result(left.size + 1, left.lower, root.val)$

$else$:

$Result\ left = traverse(root.left)$

$Result\ right = traverse(root.right)$

$if\big(left.size < 0\ or\ right.size < 0\ or \sim(root.val \in [left.lower, right.upper])\big)$:

$return\ Result(-1,0,0)$

$else$:

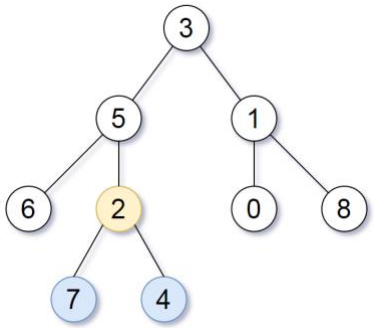
$maxsize = \max(maxsize, left.size + 1 + right.size)$

$return\ Result(left.size + 1 + right.size, left.lower, right.upper)$

---

865. Smallest Subtree with all the Deepest Nodes Facebook

Given a binary tree rooted at root, the depth of each node is the shortest distance to the root. A node is deepest if it has the largest depth possible among any node in the entire tree. The subtree of a node is that node, plus the set of all descendants of that node. Return the node with the largest depth such that it contains all the deepest nodes in its subtree.



Intuition & Algorithm

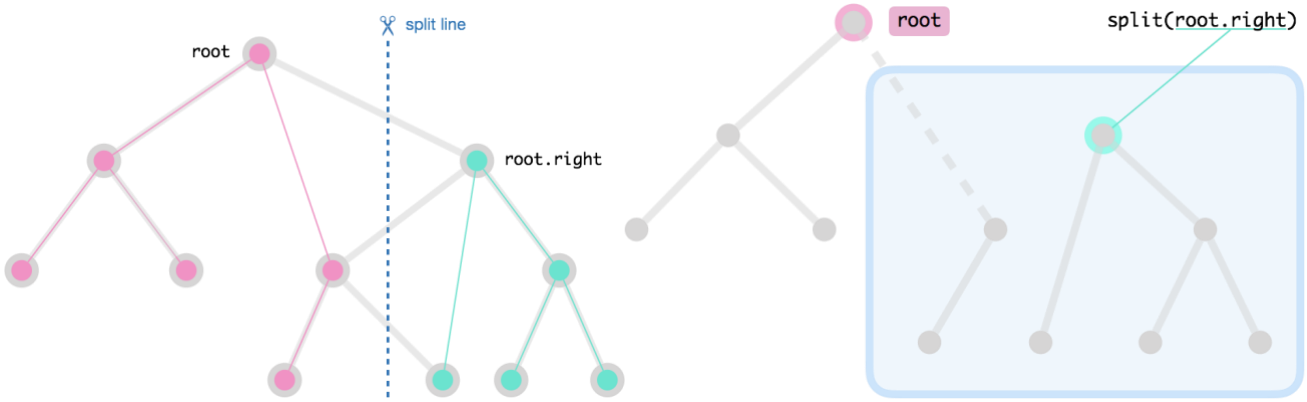
Calculate depth for each node in the tree first, then similar to "LCA Problem", we find the minimum size subtree which contains all "deepest" nodes.

776. Split BST Amazon Coupang

Given a Binary Search Tree (BST) with root node root, and a target value V, split the tree into two subtrees where one subtree has nodes that are all smaller or equal to the target value, while the other subtree has all nodes that are greater than the target value. It's not necessarily the case that the tree contains a node with value V.

Intuition

The root node either belongs to the first half or the second half. Let's say it belongs to the first half. Then, because the given tree is a binary search tree (BST), the entire subtree at root. left must be in the first half. However, the subtree at root. right may have nodes in either halves, so it needs to be split.



In the diagram above, the thick lines represent the main child relationships between the nodes, while the thinner colored lines represent the subtrees after the split. Lets say our secondary answer $bns = split(root.right)$ is the result of such a split. Recall that $bns[0]$ and $bns[1]$ will both be BSTs on either side of the split. The left half of bns must be in the first half, and it must be to the right of root for the first half to remain a BST. The right half of bns is the right half in the final answer.

The diagram above explains how we merge the two halves of $split(root.right)$ with the main tree, and illustrates the line of code $root.right = bns[0]$ in the implementations.

Algorithm

$TreeNode[\,]\ splitBST(root, V)$:

  $if\,(root = nil)$: $return\ [nil, nil]$

  $if\,(root.val \leq V)$:

    $TreeNode[\,]\ bns = splitBST(root.right, V)$

    $root.right = bns[0]$

    $bns[0] = root$

    $return\ bns$

  $else$:

    $TreeNode[\,]\ bns = splitBST(root.left, V)$

    $root.left = bns[1]$

    $bns[1] = root$

```
    return bns
```

637. Average of Levels in Binary Tree Facebook

Given a non-empty binary tree, return the average value of the nodes on each level in the form of an array.

Intuition & Algorithm

Level-BFS

250. Count Univalue Subtrees

Given a binary tree, count the number of uni-value subtrees. A Uni-value subtree means all nodes of the subtree have the same value.

Intuition & Algorithm

Do counting and check in one call

```
public int re = 0
int countUniqueSubtrees(TreeNode root):
    helper(root)
    return re

bool helper(r):
    if (r = nil): return true
    if (r.left ≠ nil & r.right ≠ nil):
        re := re + 1
        reutrn true
    bool lb = helper(root.left)
    bool rb = helper(root.right)
    if (lb & rb):
        if (r.left ≠ nil & r.right = nil & r.left.val = r.val): re := re + 1, return true
        if (r.right ≠ nil & r.left = nil & r.right.val = r.val): re := re + 1, return true
        if (r.left ≠ nil & r.right ≠ nil & r.left.val = r.right.val & r.left.val = r.val):
            re := re + 1, return true
    return false
```

742. Closest Leaf in a Binary Tree Amazon Databricks

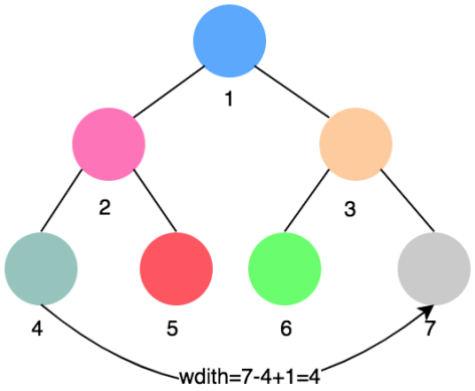
Given a binary tree where every node has a unique value, and a target key k, find the value of the nearest leaf node to target k in the tree.

Intuition & Algorithm

Construct graph based on the binary tree, then apply level-bfs

662. Maximum Width of Binary Tree Amazon

Given a binary tree, write a function to get the maximum width of the given tree. The width of a tree is the maximum width among all levels. The binary tree has the same structure as a full binary tree, but some nodes are null. The width of one level is defined as the length between the end-nodes (the leftmost and right most non-null nodes in the level, where the null nodes between the end-nodes are also counted into the length calculation.



$Queue[Node]\ Q = \{\phi\}, re = 0$

$Q.offer(Node(root, id = 1))$

$while(Q.size > 0)$:

    $sz \coloneqq Q.size, mi = 2^{31} - 1, mx = -2^{31}$

    $for(i = 0 \to sz - 1)$:

        $Node\ curr \coloneqq Q.poll$

        $mi \coloneqq \min(mi, curr.id), mx \coloneqq \max(mx, curr.id)$

        $if(curr.node.left \neq nil): Q.offer(Node(curr.node.left, curr.id \star 2))$

        $if(curr.node.right \neq nil): Q.offer(Node(curr.node.right, curr.id \star 2 + 1))$

    $re \coloneqq \max(re, mx - mi + 1)$

$return\ re$

---

604. Design Compressed String Iterator **Google**

[StringIterator,next,next,next,next,next,next,hasNext,next,hasNext]

[[L1e2t1C1o1d1e1], [*null*,L,e,e,t,C,o, *true*,d, *true*]]

```
String s, int ptr, char ch, int remains
char next():
    if (~hasNext): return ' '
    if (remains > 0):
        remains := remains − 1, return ch
    else:
        ch = s[ptr]
        i := ptr + 1
        while(i < len(s) & s[i] ∈ [0,9]) i := i + 1
        remains := int(s[ptr: i − 1])
        ptr = i, remains := remains − 1
        return ch

char hasNext():
    return ptr < len(s) or remains > 0
```

---

285. In-order Successor in BST **Facebook** **Microsoft** **Pocket Gems**

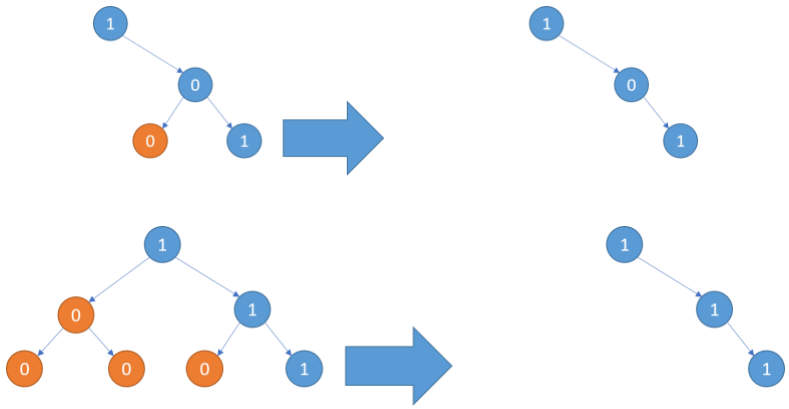
Idea: each time compare node's(p) value with root's value and halve the search range.

```
TreeNode inorderSuccessor(TreeNode root, TreeNode p):
    TreeNode suc = nil
    while(root ≠ nil):
        if (p.val = root.val): root = root.right
        elif (p.val < root.val): suc = root, root = root.left
        else: root = root.right
    return suc
```

---

814. Binary Tree Pruning **Hulu**

We are given the head node root of a binary tree, where additionally every node's value is either a 0 or a 1. Return the same tree where every subtree (of the given tree) not containing a 1 has been removed.



$pruntTreeUtil(root)$:

$if(root = nil)$: $return\ nil$

$root.left = pruneTreeUtil(root.left)$

$root.right = pruneTreeUtil(root.right)$

$if(root.val = 0\ \&\ root.left = nil\ \&\ root.right = nil)$: $return\ nil$

$else\ return\ root$

---

270. Closest Binary Search Tree Value Google Microsoft Snapchat

Algorithm

Each time compare root's value with target, if $root.value$ equals to target, just return the root, else if $target > root.value$, then the results must reside in root or root's right-subtree, else if $target < root.value$, search the closet node at root and root's left-subtree.

Time Cost $O(h)$

$int\ closetValue(root, target)$:

$closet = nil, mindif = 2^{31}$

$while(root \neq nil)$:

$lv = (long)root.val$

$dif = abs(lv - target)$

$if(dif < 1e^{-6})$: $return\ root.val$

$elif(root.val < target)$:

```
            if(target − root.val < mindif):
                mindif = target − lv
                closet = root
            root = root.right
        else:
            if(root.val − target < mindif):
                mindif = lv − target
                closet = root
            root = root.left
    return closet.val
```

---

272. Closest Binary Search Tree Value II Google

Idea: flatten the b-tree first, then find k closet values from "sorted" flatten btree.

Time Complexity: $O(n + k)$

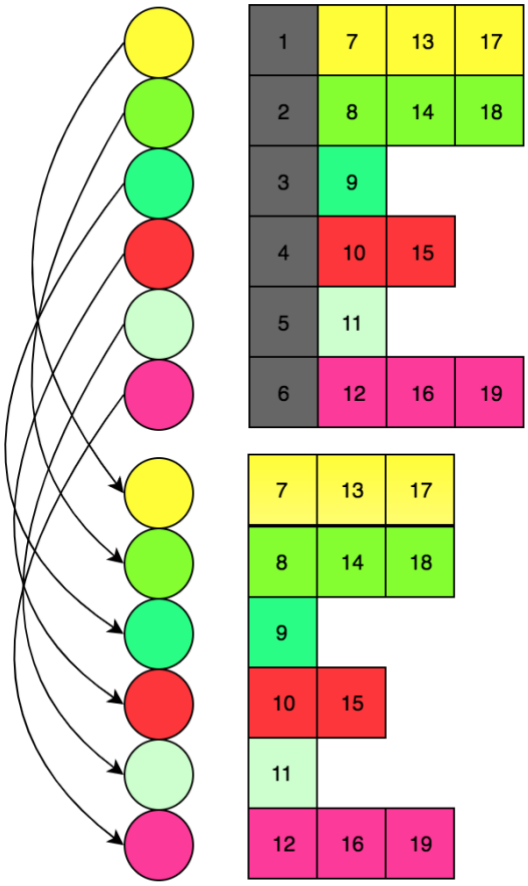
```
int closetValue(root, target):
    closet = nil, mindif = 2^31
    while(root ≠ nil):
        lv = (long)root.val
        dif = abs(lv − target)
        if(dif < 1e^−6): return root.val
```

---

251. Flatten 2D vector & 281 (Zigzag Iterator) Google Twitter Airbnb Zenefits

Intuition & Algorithm

Push all list's iterator into queue, each time take one iterator out of the queue, then let the iterator move forward and push it back to the queue in case this iterator is not empty. Keeping doing such steps until queue is empty.



```
public class MultiIteratorVertical<E> implements Iterator
    Queue<Iterator<E>> iterators = new LinkedList<>()
    Iterator<E> current = null

public boolean hasNext():
    boolean exist = false
    if(iterators.isEmpty() & (current == null or !current.hasNext())):return false
    if(current == null):current = iterators.poll()
    while(!current.hasNext() & !iterators.isEmpty()):current = iterators.poll()
    if(current.hasNext()):return true
    return false

public E next():
    if(current == null):
        try:
            current = iterator.poll()
        catch(IndexOutOfBoundsException ex):throw new NoSuchEelementException()
```

```
E result = current.next()
iterators.offer(current)
current = iterators.poll()
return result
```

---

669. Trim a Binary Search Tree Bloomberg

```
TreeNode trimBST(root, L, R):
    if (root = null): return null
    if (root.val < L): return trimBST(root.right, L, R)
    elif (root.val > R): return trimBST(root.left, L, R)
    else:
        root.left = trimBST(root.left, L, R)
        root.right = trimBST(root.right, L, R)
        return root
```

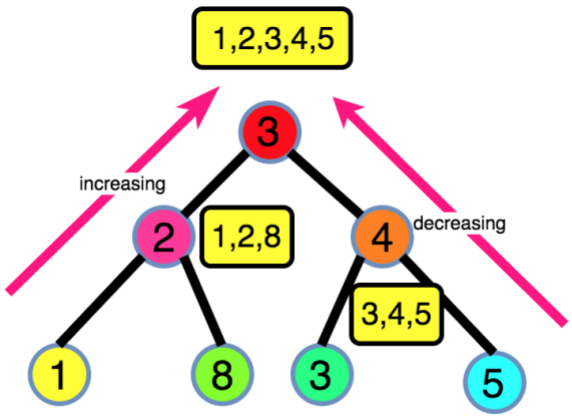
---

298. Binary Tree Longest Consecutive Sequence Google

Solution: just calculate decreasing depth for each node, and keep updating the length of longest decreasing sequence so far, (no need for hash-map)

549. Binary Tree Longest Consecutive Sequence II Google

Solution: calculate decreasing & increasing depth for each node, and store it in two hash-maps respectively. Then the node whose sum of corresponding two hash-map values the largest, is exactly what we want to find.



```
int maxLen = 0
int longest · consecutive(TreeNode root):
    find · longest · consecutive(root)
    return maxLen

⋆ return an 2 items array A,
⋆ A[0] suggests the increasing order,
⋆ A[1] suggests the decreasing order
int[ ] find · longest · consecutive(root)
    if(root = nil): return [0,0]
    int[ ] left := find · longest · consecutive(root.left)
    int[ ] right := find · longest · consecutive(root.right)
    incr = decr = 1
    if(root.left ≠ nil):
        if(root.left.val + 1 = root.val): incr = left[0] + 1
        elif(root.left.val − 1 = root.val): decr = left[1] + 1

    if(root.right ≠ nil):
        if(root.right.val − 1 = root.val): decr = max(right[1] + 1, decr)
        elif(root.left.val + 1 = root.val): incr = max(right[0] + 1, incr)

    maxLen = max(maxLen, incr + decr − 1)
    return [incr, decr]
```

314. Binary Tree Vertical Order Traversal Google Facebook Snapchat

Algorithm

```
que^Queue[int[]] = {(root, 0: horizontal offset)}
minof = maxof = 0
while(que.size > 0):
    [node, of] = que.poll
    minof := min(minof, of), maxof := max(maxof, of)
    of2vals[of].add(node.val)
```

$if(node.left \neq nil)$: $que.offer([node.left, of-1])$

$if(node.right \neq nil)$: $que.offer([node.right, of+1])$

$for(of: minof \rightarrow maxof)$:

$re.add(of2vals[of])$

Queue & Stack & Greedy & Heap

Minimum Queue ForUsAll

Solution: first, implement $class\ MinStack$, then:

$class\ MinQueue$

$MinStack\ in = \{\phi\}, out = \{\phi\}$

$getMin$:

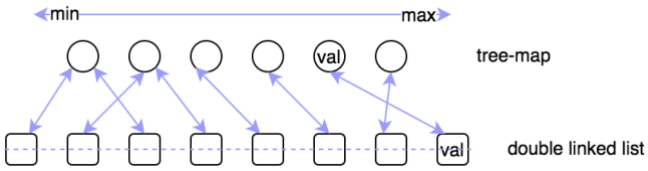
$if(in.empty)$: $return\ out.getMin$

$if(out.empty)$: $return\ in.getMin$

---

716. Max Stack LinkedIn DataVisor

Design a stack which supports $push, pop, top, popMax, peekMax$



$push(x)$: $map[x].add(Node\{x\})$

pop: $map[tail.val].remove(map[tail.val].size - 1)$

top: $return\ tail.val$

$popMax$: $map[map.lastKey].remove(map[map.lastKey].size - 1)$

$peekMax$: $return\ map.lastKey$

---

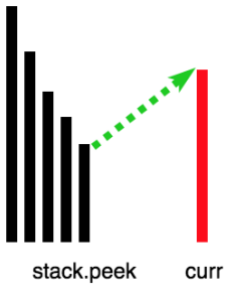
739. Daily Temperatures Google

Given a list of daily temperatures, produce a list that, for each day in the input, tells you how many days you would have to wait until a warmer temperature. If there is no future day for which this is possible, put 0 instead.

$temperatures = [73, 74, 75, 71, 69, 72, 76, 73]$, $result = [1, 1, 4, 2, 1, 1, 0, 0]$.

Idea: for each $temperatures[i]$, find its closet larger temperatures to the right of it

Solution: use stack and maintain all temperature values are in decreasing order.



$stack = \{\phi\}$

$for(i = 0 \to n - 1)$:

    $while(stack.size > 0\ \&T[stack.peek] < T[i])$:

        $index = stack.pop$

        $result[index] = i - stack.pop$

    $stack.push(i)$

---

844. Backspace String Compare Google Facebook

Given two strings $\mathcal{S}$ and $\mathcal{T}$, return if they are equal when both are typed into empty text editors. # means a backspace character.

Intuition & Algorithm #1

Let's individually build the result of each string ($build(\mathcal{S})$ and $build(\mathcal{T})$), then compare if they are equal. To build the result of a string $build(\mathcal{S})$, we'll use a stack based approach, simulating the result of each keystroke.

Time Cost: $O(\mathcal{M} + \mathcal{N})$, **Space Cost**: $O(\mathcal{M} + \mathcal{N})$

$bool\ backspaceCompare(string\ s, string\ t)$:

    $return\ build(s).equals(t)$

```
string build(string s):
    stack[char] st := [φ]
    for(char c: s):
        if(c ≠ #): st.push(c)
        elif(st.empty = false): st.pop
    return string.valueOf(st)
```

Intuition & Algorithm #2

When writing a character, it may or may not be part of the final string depending on how many backspace keystrokes occur in the future. If instead we iterate through the string in reverse, then we will know how many backspace characters we have seen, and therefore whether the result includes our character.

Iterate through the string in reverse. If we see a backspace character, the next non-backspace character is skipped. If a character isn't skipped, it is part of the final answer.

Time Cost: $O(M + N)$, **Space Cost**: $O(1)$

```
bool backspaceCompare(string s, string t):
    i := len(s) − 1, j := len(t) − 1
    skips := 0, skipt := 0

    for(; i ≥ 0 || j ≥ 0; ):
        for(; i ≥ 0; ):
            if(s[i] = #): skips := skips + 1, i := i − 1
            elif(skips > 0): skips := skips − 1, i := i − 1
            else: break
        for(; j ≥ 0; ):
            if(s[j] = #): skipt := skipt + 1, j := j − 1
            elif(skips > 0): skipt := skipt − 1, j := j − 1
            else: break
        if(i ≥ 0 & j ≥ 0 & s[i] ≠ t[j]): return false
        if((i ≥ 0) ≠ (j ≥ 0)): return false
```

```
        i := i − 1, j := j − 1
    return true
```

---

503. Next Greater Element II Google

Given a circular array (the next element of the last element is the first element of the array), print the Next Greater Number for every element.

```
int nextGreaterElement(int[ ] nums):
    n := len(nums)
    if(n = 0): return [ϕ]
    int[ ] A := [2n − 1]
    for(i := 0 → 2n − 2): A[i] := nums[i % n]
    int[ ] R := [2n − 1], R[i] := −1
    st := Stack
    for(i := 0 → 2n − 2):
        while(st.size > 0 & A[st.peek] < A[i]):
            top := st.pop
            if(i − top ≤ n − 1): R[top] := A[i]
        st.push(i)
    int[ ] re := [n]
    for(i := 0 → 2n − 2): re[i] := R[i]
    return re
```

---

71. Simplify Path Facebook Microsoft

Given an absolute path for a file (Unix-style), simplify it. For example,

path = "/home/", => "/home"

path = "/a/./b/../../c/", => "/c"

Intuition & Algorithm

```
string simplyPath(String path):
    if(path = ""): return""
    string[ ] sa := path.split(/)
    for(i = 0 → len(sa) − 1):
        if(sa[i] = "" || sa[i] = .): continue
```

```
        if(sa[i] = ..):
            if(~st.empty): st.pollFirst
        else: st.push(sa[i])
    if(st.empty): return "/"
    re := "/"
    while(~st.empty):
        re := re + st.pop
        if(~st.empty): re := re + "/"
    return re
```

---

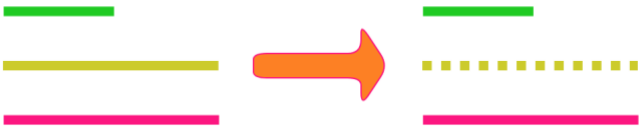
388. Longest Absolute File Path Google

The string "dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext" represents:

```
dir
    subdir1
    subdir2
        file.ext
```

Intuition & Algorithm

Apply stack to keep track of single-chain file path.



```
length · longest · path(input):
    tokens = input.split(\n)
    pathLen → stack
    result = 0
    for(s: token):
        level = s.lastIndexOf(\t) + 2 * root's level = −1 + 2 = 1
        * if current path is not subdirectoy of its previous directory, pop it
        while(~pathLen.empty & pathLen.size ≥ level): pathLen.pop
        * +2: means add two slashes
        currPathLen = (s.len − level + 2) + pathLen.empty → 0: pathLen.peek
        pathLen.push(currPathLen)
```

$if(s.indexOf(.) \geq 0): result = \max(result, currPathLen - 1)$ ⋆ *remove last "/"*

$return\ result$

---

582. Kill Process Bloomberg

List<int> $killProcess(pid^{list,int}, ppid^{list,int}, kill)$:

$if(pid.size = 0\ \&\ ppid.size = 0\ \&\ ppid.size \neq pid.size): return\ [\phi]$

List<int>$re = \phi, map = \{int \Rightarrow set^{int}\}$

$for(i = 0 \rightarrow pid.size - 1)$:

$child = pid[i], father = ppid[i]$

$map[father].add(child)$

Queue<int> $Q = \{kill\}$

$while(Q.size > 0)$:

$curr = Q.poll$

$re.add(curr)$

$Q.addAll(map[curr])$

$return\ re$

---

690. Employee Importance Uber

385. Mini Parser (Deserialization) Airbnb

$NestedInteger\ deserialize(s: string)$:

$if(\sim(s[0] = [\ ): return\ NestedIntger()$

$stack = \{\phi\}, i = 0$

$while(i < len(s) - 1)$:

$if(s[i] = [\ )$:

$NestedInteger\ ni = \{\}$

$if(stack.size > 0): stack.peek.add(ni)$ ⋆ *add current node to father node*

$stack.push(ni)$

$elif(s[i] =])$:

$stack.pop, i = i + 1$

$elif(s[i] =,): i = i + 1$

```
        else:
            sign = 1
            if(c[i] = −): i = i + 1, sign = −1
            value = 0
            while(i < len(s) − 1 & c[i] ∈ [0 − 9]): value = value ⋆ 10 + c[i]−'0', i = i + 1
            stack.peek.add(NestedInteger(sign ⋆ value))
    return result
```

---

56. Merge Intervals Google Facebook Microsoft Bloomberg LinkedIn Twitter Yelp

---

```
merge(List[Interval] list):
    list.sort((a, b) → (a.start = b.start) → (a.end − b.end): (a.start − b.start))
    LinkedList[Interval] re = [ϕ]
    for(Interval it: list):
        if(re.empty or re.last.end < it.start): re.add(it)
        else re.last.end = max(re.last.end, it.end)
    return re
```

---

502. IPO

You are given several projects. For each project ***i***, it has a pure profit P_i and a minimum capital of C_i is needed to start the corresponding project. Initially, you have ***W*** capital. When you finish a project, you will obtain its pure profit and the profit will be added to your total capital. To sum up, pick a list of at most ***k*** distinct projects from given projects to maximize your final capital, and output your final maximized capital.

Solution: the short answer to this problem is using heap(*PriorityQueue*), firstly store the projects (*profit, capital*) in a heap in the order of capital, and store the projects (*profit, capital*) in another heap in the order of profit.

```
findMaximizedCapital(k, W)
    candidate(heap): ((a, b) → (a.cap − b.cap))
    profit(heap): ((a, b) → (b.profit − a.profit))
    for(i = 0 → Profits.length − 1): candidate.offer(Capital[i], profits[i])
    t = 0, cap = W
```

$while(t < k)$:

$\star$ *select all satisfied project into profit*

$\star$ *the* profit *heap will sort project by profit in ascending order*

$while(candidate.size > 0 \ \& \ candidate.peek.cap \leq cap)$:

$profit.offer(candidate.pop)$

$if(profit.size == 0): break$

$cap = cap + profit.poll.profit$

$t = t + 1$

$return\ cap$

---

358. Rearrange String k Distance Apart Google

Given a non-empty string s and an integer k, rearrange the string such that the same characters are at least distance k from each other. All input strings are given in lowercase letters. If it is not possible to rearrange the string, return an empty string "".

Intuition & Algorithm

$rearrangeString(s, k)$:

$counts = [26]^{int}, pqueue\ pq = \{\phi\}$

$for\big(i = 0 \rightarrow len(s)\big): counts[c - a] \coloneqq counts[c - a] + 1$

$for(c = a \rightarrow z)$:

$if(counts[c] > 0): pq.offer\big(Item(c, counts[c])\big)$

$re = "", index = 0$

$while(pq.size > 0)$:

$n = k, queue\ saved = \{\phi\}$

$\star$ *each time arrange k chars with most frequency*

$while(n > 0 \ \& \ pq.size > 0)$:

$item = pq.poll$

$if(item.previndex = -1): item.previndex = index$

$elif(index - item.previdex < k): return\ ""$

$else: item.previndex = index$

$index \coloneqq index + 1$

$item.freq \coloneqq item.freq - 1$

$re := re + item.key$

$n := n - 1$

$saved.offer(item)$

$while(saved.size > 0)$:

$item = saved.poll$

$if(item.freq > 0): pq.offer(item)$

$$Item\begin{cases} char\ key \\ int\ freq \\ int\ previndex \end{cases}$$

---

134. Gas Station Google

There are N gas stations along a circular route, where the amount of gas at station i is $gas[i]$. You have a car with an unlimited gas tank and it costs $cost[i]$ of gas to travel from station i to its next station $(i + 1)$. You begin the journey with an empty tank at one of the gas stations.

Algorithm

There is a trick, if the car can make to i^{th} gas station, and has no more gas, it not only means the car cannot makes to the $i + 1^{th}$ gas station, but also means the car cannot makes to the $i + 1^{th}$ gas station if it starts its journey from any stations from$[0: i]$, the possible "best" station may reside in the range of $[i + 1: n - 1]$. Also, to be noticed, there may not exist such a best station if the whole amount of gas is smaller than that of costs.

$result = -1, total \cdot gas = 0, total \cdot costs = 0, tank = 0$

$for(i = 0 \rightarrow n - 1)$:

$tank = tank + gas[i] - cost[i]$

$total \cdot gas = total \cdot gas + gas[i]$

$tot \cdot costs = total \cdot costs + cost[i]$

$if(tank \leq 0)$:

$tank = 0$

$result = i + 1$

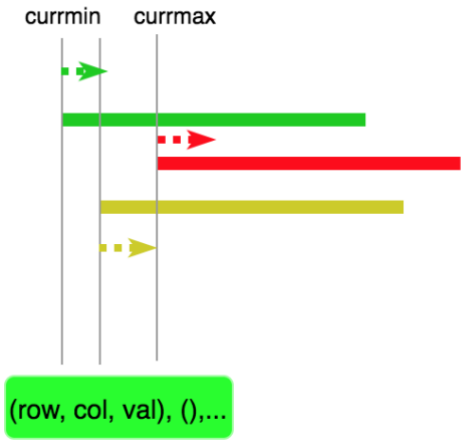
$return\ (total \cdot gas \geq total \cdot costs) \rightarrow result: -1$

---

632. Smallest Range Lyft

You have k lists of sorted integers in ascending order. Find the smallest range that includes at least one number from each of the k lists.

Input: $[[4,10,15,24,26],[0,9,12,20],[5,18,22,30]]$, Output: [20,24]



Since $\min(A[i,0], A[i,len(A[i])-1])$, $\max(A[i,0], A[i,len(A[i])-1])$ includes at-least one number from each of k lists.

$currMin = 2^{31}-1, currMax = -2^{31}$

$for(i = 0 \to len(A)-1)$:

    $pq.offer(Item(A[i,0], row = i, col = 0))$

    $currMax := \max(currMax, A[i,0])$

$range := 2^{31}-1, start = -1, end = -1$

$while(pq.size = len(A))$:

    $Item\ item := pq.poll$

    $currMin := item.val$

    $if(currMax - currMin + 1 < range)$:

        $range := currMax - currMin + 1$

        $start := currMin, end := currMax$

    $if(item.col + 1 < len(A[item.row]))$:

        $pq.offer(Item(A[item.row, item.col+1], item.row, item.col+1))$

        $currMax := \max(currMax, A[item.row, item.col+1])$

    $return\ range$

---

406. Queue Reconstruction by Height Google

People with different height are standing in a queue, given a list of pairs (h, k), where the h stands for the height of people, and the k suggests how many people who are taller than or equal to him standing in front him. Asked to reconstruct the queue.

Algorithm

The key of solving this problem is "sort & insertion sort", the first step is sorting the pairs array. If two pairs share the same "h-value", then the one with smaller "k-value" should take higher priority. If "h-value" is different, then the one with smaller "h-value" takes higher priority.

$$if(h_1 = h_2), return\ k_1 < k_2,$$
$$otherwise, return\ h_2 < h_1$$

The second step is "insert-sort", based on the index of each pair after previous sorting, we insert each pair to the place which is the same as the index

$$for(i = 0 \to n - 1): list.add\big(k_i, (h_i, k_i)\big)$$

$int[\,][\,]\ reconstructQueue(people: int[\,][\,] = R^{n*2})$:

$n = len(people)$

$\Rightarrow sort\ by\ height\ (desc)\ first, then\ \#\ people$

$Arrays.sort(people, (a[0] = b[0]) \to (a[1] - b[1]): b[0] - a[0])$

$list: \{int[\,]\}$

$for(i = 0 \to n - 1)$:

$\Rightarrow people[i, 1]\ means\ where\ this\ guy\ people[i]\ should\ be$

$list.add\big(people[i, 1], [people[i, 0], people[i, 1]]\big)$

DFS & BFS

399. Evaluate Division Landing.ai

$$\begin{bmatrix} USD & EUR & 0.81 \\ EUR & RMB & 7.83 \\ RMB & GBP & 0.11 \end{bmatrix} \to \frac{USD}{GBP} = 0.81 * 7.83 * 0.11 = 0.69$$

To calculate $\frac{x}{y}$, it just needs to find a path between x and y, let's say there is a path $\{x, a_0, a_1, \cdots, a_n, y\}$, then $\frac{x}{y} = \frac{x}{a_0} \times \frac{a_0}{a_1} \times \cdots \times \frac{a_n}{y} = (x, a_0).w \times (a_0, a_1) \cdots \times (a_n, y).$

Time Complexity: $O(E + Q \star E)$, **Space Cost**: $O(E)$

$mp[equations[i, 0]].add(\{equations[i, 0], values[i]\})$

$mp[equations[i, 1]].add\left(\left\{equations[i, 0], \frac{1}{values[i]}\right\}\right)$

$used \leftarrow \{\}, type = set$

$for(qe\ in\ queries)$:

    $used.clear()$

    $ne = qe[0], de = qe[1]$

    $tmp = dfs(ne, de, used, mp, 1.0)$

    $results[i] = tmp = 0 \rightarrow -1.0 : tmp$

$dfs(start, end, used, mp, value)$:

    $\star\ if\ there\ is\ no\ way\ from\ start \rightarrow end$

    $if(used.contains(start)\ ||\ mp[start] \neq nil)\ return\ 0.0;$

    $if(start = end) : return\ value$

    $used.add(start)$

    $for(denominator :\ mp.keySet)$:

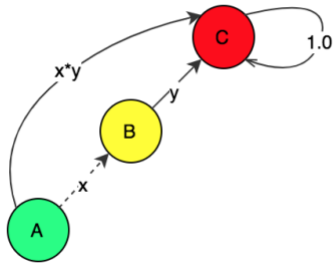
        $rs = dfs(denominator\ , end, used, mp, value \star mp[de])$

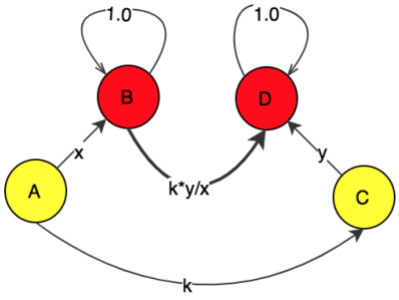
        $if(rs \neq 0) : return\ rs$

    $used.remove(start)$

Follow Up: Time Cost reduced to $O(E + Q)$, means each query just takes constant time.

Solution: Union-Find.





$fa: map(string \Rightarrow string)$

$val: map(string \Rightarrow double)$

$for(i = 0 \rightarrow len(equations) - 1)$:

$n_a = equations[i][0], n_b = equations[i][1], k = values[i]$

$if(fa[n_a] \neq nil): fa[n_a] = n_a, val[n_a] = 1$

$if(fa[n_b] \neq nil): fa[n_b] = n_b, val[n_b] = 1$

$r_a = find(n_a, fa, val), r_b = find(n_b, fa, val)$

$if(r_a \neq r_b)$:★ *if (na,nb) are not belong to the same group*

$fa[r_a] = r_b$ ★ *then union it*

$val[ra] = \frac{k \star val[n_b]}{val[n_a]}$,★ *and also update the relative weight of* (r_a, r_b)

$for(i = 0 \rightarrow queries.length - 1)$:

$n_a, n_b = queries[i, 0], queries[i, 1]$

$r_a, r_b = find(n_a, fa, val), find(n_b, fa, val)$

$if(r_a = nil \ or \ r_b = nil \ or \ r_a \neq r_b): results[i] = -1.0$

$else: results[i] = \frac{val[n_a]}{val[n_b]}$

$find(x)$:⋆ *return the root of string s*

$if(fa[x] = nil)$: return nil

$if(fa[x] \neq x)$:★★ $fa[root] = root$

$currFather = fa[x]$

$fa[x] = find(currFather)$

$val[x] \star= val[currFather]$ ⋆ *multiply the value when compressing path*

return $fa[s]$

---

301. Remove Invalid Parentheses Facebook

Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

Intuition & Algorithm #1 (Not Efficient)

Level-bfs

```
List[string] removeInvalidParentheses(String s):
    List[String] re := [ϕ]
    if(len(s) = 0): return re
    Set[String] used = {ϕ}
    Queue[String] Q = {ϕ}
    Q.offer(s)
    used.add(s)
    if(valid(s)): return re.add(s)

    while(Q.size > 0):
        sz := Q.size
        bool found := false
        for(k = 0 → sz − 1):
            p := Q.poll
            for(i = 0 → len(p) − 1):
                if(p[i] ≠ ( || p[i] ≠ )): continue
                candidate := p[0, i − 1] + p[i + 1: ]
                if(used[candidate] ≠ nil):
                    used[candidate] = true
                    Q.offer(candidate)
                    if(valid(candidate)):
                        re.add(candidate)
                        if(! found): found = true
        if(found = True): break
    return re
```

Intuition & Algorithm #2

Optimized DFS by cutting unnecessary branches.

```
List[string] removeInvalidParentheses(String s):
    n := len(s)
    int rmL := 0, rmR := 0 ⇒ the min# removal for left & right bracket
    for(i = 0 → n − 1):
        if(s[i] = ( ): rmL := rmL + 1
        elif(s[i] =)):
            if(rmL > 0): rmL := rmL − 1
            else: rmR := rmR + 1

    List[string] re := {ϕ}
    Set[string] set := {ϕ} ⇒ used to remove duplicates
    String load = ""
    int index := 0
    int open = 0
    dfs(s, index, load, rmL, rmR, open, set)

void dfs(String s, int index, String load, int rmL, int rmR, int open, Set[string] set):
    if(index = len(s)):
        if(rmL = 0 & rmR = 0 & open = 0): set.add(load)
        return
    if(s[index] = ( ):
        dfs(s, index + 1, load + "(", rmL, rmR, open + 1, set)
        if(rmL > 0): dfs(s, index + 1, load, rmL − 1, rmR, open, set)
    elif(s[index] = ) ):
        if(open > 0): dfs(s, index + 1, load + ")", rmL, rmR, open − 1, set)
        if(rmR > 0): dfs(s, index + 1, load, rmL, rmR − 1, open, set)
    else:
        dfs(s, index + 1, load + s[index], rmL, rmR, open + 1, set)
```

841. Keys and Rooms Google

There are N rooms and you start in room 0. Each room has a distinct number in $0, 1, 2, \ldots, N - 1$, and each room may have some keys to access the next room. Formally, each room i has a list

of keys $rooms[i]$, and each key $rooms[i,j]$ is an integer in $[0,1,\ldots,N-1]$ where $N = len(room)$. A key $rooms[i,j] = v$ opens the room with number v. Initially, all the rooms start locked (except for room 0).

Intuition & Algorithm

Simple DFS, marked each of visited node. At the end, check if the number of visited node equals to the size of graph.

266. Palindrome Permutation

267. Palindrome Permutation II **Google** **Bloomberg** **Uber**

```
List[String] generatePalindromes(s):
    int[ ] counts = 128 * [0]
    for(i = 0 → len(s) − 1): counts[s[i]] := counts[s[i]] + 1
    int cntOddChar = 0, char oddChar = null, int totChars = 0
    for(i = 0 → 127):
        if(cntOddChar > 1): return [ϕ]
        if(mod(counts[i], 2) = 1):
            cntOddChar := cntOddChar + 1
            oddChar = (char)i
        totChars := totChars + 1

    List[String] results = [ϕ]
    * if one char shows up in odd times, then place one of it at the middle
    load = oddChar = nil → "": oddChar
    remains = oddChar = nil → tot: tot − 1
    if(oddChar ≠ nil): counts[oddChar] := counts[oddChar] − 1
    dfs(counts, load = oddChar, remains, results)

dfs(counts, load, remains, results):
    if(remains = 0): results.add(load), return
    for(i = 0 → 127):
        char c = (char)i
        if(counts[i] > 0 and mod(counts[i], 2) = 0):
            counts[i] := counts[i] − 2
            dfs(counts, c + load + c, remains − 2, results)
```

$counts[i] \coloneqq counts[i] + 2 \star$ *for backtrack*

---

773. Sliding Puzzle Airbnb

On a $2 \star 3$ board, there are 5 tiles represented by the integers 1 through 5, and an empty square represented by 0. A move consists of choosing 0 and a 4-directionally adjacent number and swapping it. The state of the board is solved if and only if the board is $[[1,2,3],[4,5,0]]$. Given a puzzle board, return the least number of moves required so that the state of the board is solved. If it is impossible for the state of the board to be solved, return -1.

Intuition & Algorithm

"level-bfs"

```
int slidingPuzzle(int[ ][ ] board):
    Queue[String] Q = {ϕ}
    Set[String] used = {ϕ}
    String start = board[0,0] + ⋯ + board[1,2]
    Q.offer(start)
    step = 0
    found = false

    while(Q.size > 0):
        sz = Q.size
        used.addAll(Q)
        for(k = 0 → sz − 1):
            curr ≔ Q.poll
            if(curr = 123450): found = true, break
            index ≔ curr.indexOf(0), r = mod(index, 3), c = index/3
            for(i = 0 → len(dirs) − 1):
                n_r, n_c = r + dirs[i, 0], c + dirs[i, 1]
                if(n_r ∈ [0,1] & n_c ∈ [0,2]):
                    nidx ≔ n_r ⋆ 3 + n_c
                    char[ ] chars = curr.toCharArray
```

$swap(chars, index, nidx)$

$nstr \coloneqq String(chars)$

$if(used[nstr] \neq nil)$: $Q.offer(nstr)$

$if(found)\ break$

$step \coloneqq step + 1$

$return\ found \rightarrow step$: -1

---

694. Number of Distinct Islands Amazon

Solution: flood-fill + relative path

Idea: because identical island has the same "dfs-relative" path.

11000

11000

00011

00011

For example, the left-top island's dfs path (0,0) → (0,1) → (1,1) → (1,0). Also, the right-bottom island's dfs path (0,0) → (0,1) → (1,1) → (1,0).

$for(i = 0 \rightarrow grid.length - 1)$

$for(j = 0 \rightarrow grid[0].length - 1)$

$if(grid[i,j] = 1)$:

$grid[i,j] = 0$

$path = ""$

$floodFill(grid, i, j, 0,0, path)$

$if(set[path] \neq nil)$:

$counts = counts + 1$

$set.add(path)$

$return\ counts$

---

488. Zuma Game Baidu

You have a row of balls on the table, colored red(R), yellow(Y), blue(B), green(G), and white(W). You also have several balls in your hand. Each time, you may choose a ball in your hand, and insert it into the row (including the leftmost place and rightmost place). Then, if there is a group of 3 or more balls in the same color touching, remove these balls. Keep doing

this until no more balls can be removed. Find the minimal balls you have to insert to remove all the balls on the table. If you cannot remove all the balls, output -

Intuition

DFS, in each of DFS call, find consecutive balls with the same color, and check if there are enough number of balls in that color in hands. If there are, use that and delete 3 consecutive balls. Otherwise, keep finding.

Algorithm

```
int findMinStep(board: string, hand: string):
    balls = [256]: int
    for(char c: hand): balls[c] = balls[c] + 1
    re = dfs(board, hand)

int dfs(board, balls):
    if(len(board) = 0): return 0 ⋆ if no more colored balls in hand
    i = 0, minstep = 2^31 − 1
    while(i < len(board)):
        j ≔ i
        while(j < len(board) & board[i] = board[j]) j ≔ j + 1
        char color = board[i]
        consecBallNum = j − i ⋆ balls[i: j − 1] are in the same color
        cancelBallNum = 3 − consecBallNum ⋆ min # of balls need for deleting balls[i: j − 1]
        if(balls[color] ≥ cancelBallNum):
            ⋆ delete all consecutive same colored balls after concatenating
            nextBoard = update(board[0: i − 1] + board[j: ])
            balls[color] = balls[color] − cancelBallNum
            substeps = dfs(nextBoard, balls)
            if(substeps ≥ 0): minstep = min(minstep, substeps + cancelBallNum)
            balls[color] = balls[color] + cancelBallNum
        i ≔ j
    return minstep = 2^31 − 1 → −1: minstep

int update(String s):
    i ≔ 0
```

$while(i < len(s))$:

$j = i$

$while(j < len(s)\ \&\ s[i] = s[j]): j = j + 1$

$if(j - i \geq 3): s = s[0: i - 1] + s[j:]$

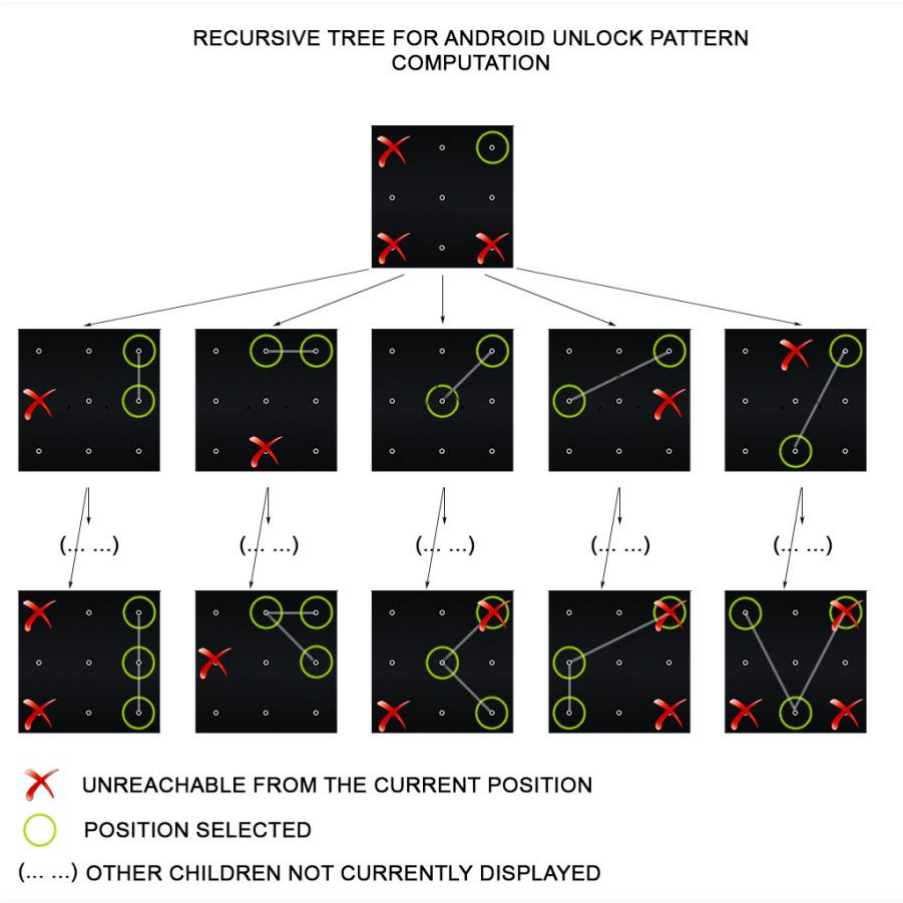
$else: i = i + 1$

$return\ s$

---

351. Android Unlock Patterns Google

Given an Android $3x3$ key lock screen and two integers m and n, where $1 \leq m \leq n \leq 9$, count the total number of unlock patterns of the Android lock screen, which consist of minimum of m keys and maximum n keys.



$number \cdot of \cdot patterns(m, n)$:

$for(le = m \rightarrow n)$:

$used: [false]$

$re = re + count \cdot patterns(-1, le, used)$

$count \cdot patterns(prev, remainedLen, used)$:

$if(remainedLen == 0): return\ 1$

$counts = 0$

$for(curr = 0 \rightarrow 8)$:

$if\big(valid(prev, curr, used)\big)$:

$used[curr] = true$

$count \cdot patterns(curr, remainedLen - 1, used)$

$used[curr] = false$

$return\ counts$

$boolean\ valid(prev, curr, used)$:

$if(prev = -1): return\ true$

$if(used[curr]): return\ false$

$\star$ *if prev & curr are adjacent to each other*

$if\big((prev + curr)\ \&\ 1 == 1\big): return\ true$

$\star$ *if both prev & curr are at the end of diagonal or x · dignonal line*

$mid = \frac{(prev+curr)}{2}$

$if(mid = 4): return\ used[mid]$

$\star$ *if prev & curr are neither in the same row nor column*

$if\left((prev\%3 \neq curr\%3)\ \&\ \left(\frac{prev}{3} \neq \frac{curr}{3}\right)\right): return\ true$

$return\ used[mid]$

---

542. 01 Matrix Google

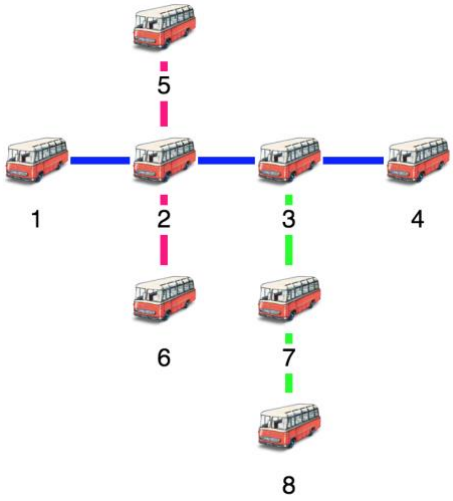
Given a matrix consists of 0 and 1, find the distance of the nearest 0 for each cell.

Solution: level-bfs. Push all cells contains zero into queue, do "level-bfs", and for each cell one, update it with the level value.

815. Bus Routes Google

We have a list of bus routes. Each $routes[i]$ is a bus route that the i^{th} bus repeats forever. For example, if $routes\ [0]\ =\ [1, 5, 7]$, this means that the first bus (0^{th} indexed) travels in the sequence $1 \to 5 \to 7 \to 1 \to 5 \to 7 \to 1 \to \cdots$ forever. We start at bus stop S (initially not on a bus), and we want to go to bus stop T. Travelling by buses only, what is the least number of buses we must take to reach our destination? Return -1 if it is not possible.

Solution: "level-bfs", maintain two variables, $usedLines$ & $usedStops$, each time at a stop, select a line which has not been taken before, and after select that line, select a bus-stop which has not been arrived before.



$int\ numBusesToDestination(routes = R^{2d}, S, T)$:

$if(S = T): return\ 0$

Map<int, Set<int>> $stop2Lines$

$for(i = 0 \to len(routes) - 1)$:

$for(j = 0 \to len(routes[0]) - 1)$:

$int\ bus \cdot stop = routes[i, j]$

$int\ line \cdot id = i$

$stop2Lines[bus \cdot stop] = stop2Lines[bus \cdot stop] + line \cdot id$

$Queue\langle int\rangle\ Q = \{S\}$

$level = 0$

$usedLines: set = \{\phi\}$

$usedStops: set = \{\phi\}$

$while(Q.size > 0)$:

$size = Q.size$

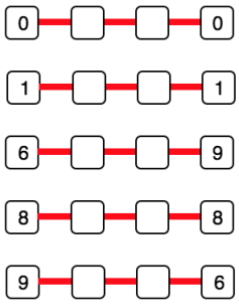
```
used · stops.addAll(Q)
level = level + 1

for(i = 0 → size − 1):
    curr · stop = Q.poll
    for(line · id: stop2Lines[curr · stop]):
        if(~usedLines[line · id]):
            for(stop: routes[line · id]):
                if(~usedStops[stop]):
                    if(stop = T): return level
                    usedLines.add(line · id)
                    usedStops.add(stop)
                    Q.offer(stop)
return − 1
```

---

247. Strobogrammatic Number II Google

Solution: DFS, $0 + [DFS] + 0, 1 + [DFS] + 1, 6 + [DFS] + 9, 8 + [DFS] + 8, 9 + [DFS] + 6$



```
create(n, m):
    if(m = 0): return {""}
    if(m = 1): return {"1","8","0"};
    list⟨string⟩ subs = create(n, m − 2)
    for(String sub: subs):
        if(n ≠ m): results.add("0 + sub + 0")
        results.add("1 + sub + 1")
        results.add("6 + sub + 9")
        results.add("8 + sub + 8")
```

$results.add("9 + sub + 6")$

$return\ results$

---

248. Strobogrammatic Number III Google

Given a range$[low, high]$, count the number of strobogrammatic numbers fall into this range.

Solution: Backtrack

$find(low, high, curr)$:

$if(len(curr) \in [len(low), len(high)])$:

$if(curr.len = low.len\ and\ curr < low): return$

$if(curr.len = high.len\ and\ curr > high): return$

$if(!(curr.len \geq 2\ and\ curr[0] == '0')): counts := counts + 1$

$if(len(curr) + 2 \in [len(low), len(high)])$:

$find(low, high, 0 + curr + 0)$

$find(low, high, 1 + curr + 1)$

$find(low, high, 6 + curr + 9)$

$find(low, high, 8 + curr + 8)$

$find(low, high, 9 + curr + 6)$

$return\ counts$

---

417. Pacific Atlantic Water Flow Google

Solution: add left & top cells to queue, and do flood-fill (bfs), also add right & bottom cells to queue, and do another flood-fill.

$List\langle[\ ]\rangle\ pacificAtlantic(matrix = [m][n])$

$pacific \cdot queue = \{\phi\}$

$atlantic \cdot queue = \{\phi\}$

$pacific \cdot canreach = [len(mat) \star len(mat[0])]$

$atlantic \cdot canreach = [len(mat) \star len(mat[0])]$

$m = len(matrix), n = len(matrix[0])$

$for(i = 0 \rightarrow m - 1)$:

$pacific \cdot canreach[i][0] = atlantic \cdot canreach[i][n - 1] = true$

$pacific \cdot queue.offer([i, 0]), atlantic \cdot queue.offer([i, n - 1])$

$for(i = 0 \to n - 1)$:

$pacific \cdot canreach[0][i] = \ atlantic \cdot canreach[m - 1][i] = true$

$pacific \cdot queue.offer([0, i]), atlantic \cdot queue.offer([m - 1, i])$

$bfs(matrix, pacific \cdot queue, pacific \cdot canreach = boolean[m][n])$

$bfs(matrix, atlantic \cdot queue, atlantic \cdot canreach = boolean[m][n])$

$bfs(mat, pacific \cdot queue, canreach)$:

$while(pacific \cdot queue.size > 0)$:

$curr = pacific \cdot queue.poll$

$x = curr.x, y = curr.y$

$for(i = 0 \to 4)$:

$nx = x + dirs[i, 0]$

$ny = y + dirs[i, 1]$

$if\ \left(n_x \in [0, m - 1]\ \&\ n_y \in [0, n - 1]\ \&\ {\sim}canreach_{(n_x, n_y)}\ \&\ mat_{(n_x, n_y)} > mat_{(x,y)}\right)$:

$canreach[n_x, n_y] = true$

$pacific \cdot queue.offer([n_x, n_y])$

---

679. 24 Game Google

Question: Given four cards, each of which contains a value from $0 - 9$. Return true if insert operators between four cards can make 24, otherwise, return false.

Algorithm

Each time take two cards, compute all possible values between the two and store them in sort of new cards. Then, replace the two cards with the each of new cards, and recursively judge if the new hand-cards can make 24 or not.

$bool\ judgePoint24Util(handCards: list)$:

$if(handCards.size = 0): return\ false$

$if(handCards.size = 1): return\ |handCards[0] - 24.0| < 10^{-6}$

$for(i = 0 \to handCards.size - 1)$:

$for(j = 0 \to i - 1)$:

$v_i := handCards[i], v_j := handCards[j]$

$vals = [v_i + v, v_i - v_j, v_j - v_i, v_i * v_j]$

$if(v_i \neq 0): vals.add\left(\frac{v_j}{v_i}\right)$

$if(v_j \neq 0): vals.add\left(\frac{v_i}{v_j}\right)$

$handCards.remove(i)$

$handCards.remove(j)$

$for(v: vals)$:

$handCards.add(v)$

$if(judgePoint24Util(handCards)): return\ true$

$handCards.remove(handCards.size - 1)$

$handCards.add(j, v_j)$

$handCards.add(i, v_i)$

$return\ false$

---

778. Swim in Rising Water Google

Given an $N\ x\ N$ grid, each square $grid[i,j]$ represents a time which means at that time, $cell(i,j)$ can be reached.

Solution: use priority-queue + bfs, select the adjacent cell which has the smallest timestamp, and always keep track of the max timestamp visited so far.

286. Walls and Gates Google Facebook

You are given a m x n 2D grid initialized with these three possible values.

-1:Wall, 0:gate, INF: empty room. Fill each empty room with the distance to its nearest gate. If it is impossible to reach a gate, it should be filled with INF.

Intuition & Algorithm

Push all "gates" into queue and do level-bfs

$void\ wallsAndGates(int[\,][\,]\ rooms)$:

  $m \coloneqq len(room), n \coloneqq len(rooms[0])$

  $Q.offer([i,j], \forall (i,j)\ s.t. room[i,j] = 0)$

  $level \coloneqq 1$

  $while(Q.size > 0)$:

    $sz \coloneqq Q.size$

    $for(k = 0 \to sz - 1)$:

      $[i,j] \coloneqq Q.poll$

      $for([ni,nj] \in (i,j).next)$:

        $if(rooms[ni,nj] = 2^{31} - 1)$:

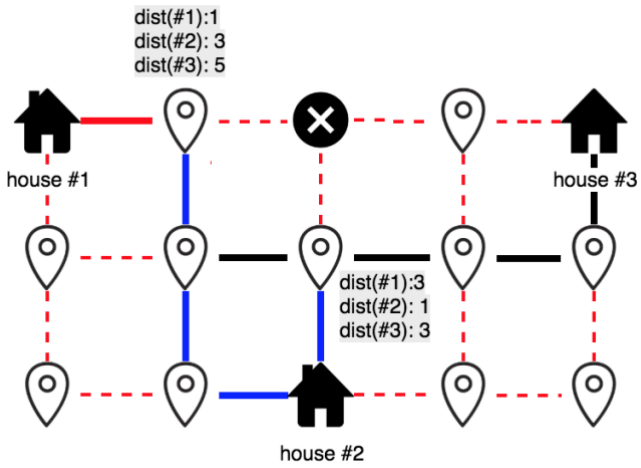
          $Q.offer([ni,nj]), rooms[ni,nj] \coloneqq level$

    $level \coloneqq level + 1$

---

317. Shortest Distance from All Buildings Google Zenefits

Question: given a 2d grid which filled with $0(empty\ space), 1(building), 2(obstacle)$



Intuition

Starting from each house, do "level-bfs" to calculate the shortest distance for each empty space from this house, and sum all the shortest distance up. The sum of distance $dist[i,j]$ marks the cost from this empty space to all houses. Also, using another variable$access[i,j]$ marks the number of houses can be reached from $[i,j]$.

Algorithm

$access = R^2, access[i][j]: number\ of\ buildings\ can\ be\ reached\ from\ (i,j)$

$dist = R^2, dist[i][j]: sum\ value\ of\ shortest\ distance\ from\ (i,j)\ to\ all\ buildings(grid[i,j] = 1)$

$for(i = 0 \to m-1)$:

    $for(j = 0 \to n-1)$:

        $if(grid[i][j] == 1)$:

            $buildings \coloneqq buildings + 1$

            $bfs(i,j,m,n,access,dist,grid)$

$(i,j)\ is\ the\ best\ position\ if\ \&\ only\ if$:

1 $grid[i][j] == 0$

2 $access[i][j] == buildings$

3 $dist[i][j]\ is\ the\ smallest\ among\ all\ candiates$

$for(i = 0 \to m-1)$:

    $for(j = 0 \to n-1)$:

        $if(grid[i][j] = 0\ and\ access[i][j] = buildings\ and\ dist[i][j] < mind)$:

            $mind = dist[i][j]$

$bfs(i,j,m,n,access,dist,grid)$:

    $Queue\langle int[]\rangle\ que = \{\phi\}$

    $que.offer([i,j])$

    $used = R^2, used[i][j]\ marks\ the\ position(i,j)\ has\ been\ visited$

    $level = 0$

    $while(que.size > 0)$:

        $size = que.size$

        $for(i = 0 \to size - 1)$

            $curr = que.poll$

            $x = curr[0], y = curr[1]$

            $access[x,y] \coloneqq access[x,y] + 1$

$dist[x,y] \coloneqq dist[x,y] + level$ ⋆ *the distance between* (x,y) *and* $(i,j) = level$

$for(i = 0 \to dirs.length)$:

$nx = x + dirs[i][0], ny = y + dirs[i][1]$

$if(nx \in [0, m-1]\ \&\ ny \in [0, n-1]\ \&\ !used[nx][ny]\ \&\ grid[nx][ny] == 0)$:

$used[nx][ny] = true$

$que.offer([nx, ny])$

$level = level + 1$

---

296. Best Meeting Point Google

Definition, Manhattan Distance, $MD(p,q) = |p.x - q.x| + |p.y - q.y|$. A group of two or more people wants to meet and minimize the total travel distance. You are given a 2D grid of values 0 or 1, where each 1 marks the home of someone in the group.

Intuition & Algorithm #1 (TLE)

For like what the 317. Shortest Distance from All Buildings did, starting from each of building ($cell[i,j] = 1$), apply "Level BFS" to find how many buildings can be accessed from each of empty space and sum of shortest distance from empty space to all buildings.

Intuition & Algorithm #2 (AC)

Record all 1's x & y coordinates, and sort them respectively, the middle (x,y) is such best point.

$for(i = 0 \to m-1)$:

$for(j = 0 \to n-1)$:

$if(grid[i][j] == 1): X.add(i), Y.add(j)$

$$median_x, median_y = quick \cdot select\left(X, \frac{len(X)}{2} + 1\right), quick \cdot select(Y, \frac{Y.length}{2} + 1)$$

$for(i = 0 \to len(X) - 1)$:

$md \coloneqq md + |X[median_x] - X[i]| + \left|Y\left[median_y\right] - Y[i]\right|$

Intuition & Algorithm #3 (AC)

⋆ *the best meeting point is the middle element of array, for example,* $[x_1, x_2, x_3, x_4]$,

⋆ *the best meeting point is* x_2, *the manhattan distance starting from* x_2 *is*:

⋆ $md(x_2) = |x_2 - x_1| + |x_3 - x_2| + |x_4 - x_2| = (x_4 - x_1) + (x_3 - x_2)$

⋆ *in general, given a sorted array* $[x_1, x_2, \cdots, x_n]$, *suppose the best point is* x_b

$\star\ md(x_b) = (x_n - x_1) + (x_{n-1} - x_2) + (x_{n-2} - x_3) + \cdots$

$for(i = 0 \to m - 1)$:

    $for(j = 0 \to n - 1)$:

        $if(grid[i][j] == 1)$: $X.add(i), Y.add(j)$

$return\ getMinManhattanDistance(X) + getMinManhattanDistance(Y)$

$getMinManhattanDistance(list)$

    $Collections.sort(list)$

    $l = 0, r = list.length - 1$

    $while(l < r)$: $md = md + list[r--] - list[l++]$

    $return\ md$

Maze Trilogy Google

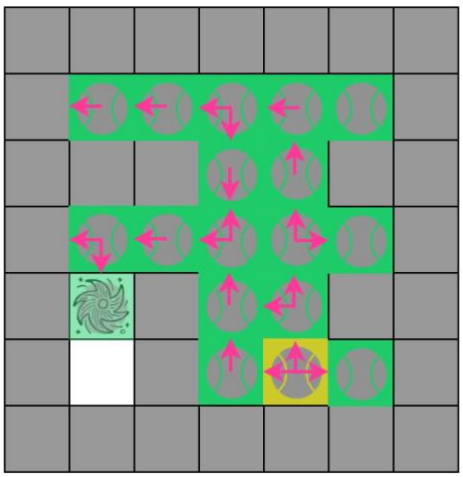
490. The Maze I (check if ball can make to the target)

505. The Maze II (find shortest distance from source to target)

499. The Maze III (find the lowest alphabetical order of move from source to target)

I, II: the ball won't stop until hits wall

III: the ball will fall into a hole, no wall condition



Intuition

To be noticed, unlike other 2D matrix where one cell (i, j) is supposed to relate to other four directional adjacent cells $[i-1, j], [i, j-1], [i+1, j], [i, j+1]$. In this problem, since ball won't stop rolling until hits a wall, the ball won't "smartly" change its ongoing direction at-will. So when mark one cell as visited, we need to add one more dimension to visited array, $visited[x, y, dir], dir \in [left, right, up, down]$.

Algorithm

$String\ findShortestWay(maze = R^2, ball = [x, y], hole = [x, y])$

$m = len(maze)$

$if(m = 0): return$ impossible

$n = len(maze[0])$

$if(ball_0 = hole_0\ \&\ ball_1 = hole_1): return$ ""

Queue<Cell> $Q = \{\phi\}$

$used = R^3: bool[x, y, dir]$

$for(i = 0 \rightarrow len(dirs) - 1):$

$if\big(can \cdot roll(ball[0], ball[1], d, maze, used)\big):$

$Q.offer\big(Cell(ball_0, ball_1, d, dirmap[d] + "")\big)$

$used[ball_0, ball_1, d] = true$

$while(Q.size > 0):$

$size = Q.size$

$for(k = 0 \rightarrow size - 1):$

$curr = Q.poll$

$n_x = curr.x + dirs[curr.dir, 0], n_y = curr.y + dirs[curr.dir, 1]$

$if\big(n_x = hole_0\ \&\ n_y = hole_1\big): return\ curr.route$

$if\ \Big(can \cdot roll\big(n_x, n_y, curr.dir, maze, used\big)\Big):$

$used\big[n_x, n_y, curr.dir\big] = true$

$Q.offer\ \Big(Cell\big(n_x, n_y, curr.dir, curr.route\big)\Big)$

$else:$

$if\ \Big(can \cdot roll\big(n_x, n_y, mod(curr.dir + 1,4), maze, used\big)\Big):$

$used\big[n_x, n_y, mod(curr.dir + 1,4)\big] = true$

$route = curr.route + dirmap[mod(curr.dir + 1,4)]$

$$Q.offer\left(Cell\left(n_x, n_y, mod(curr.dir + 1,4), route\right)\right)$$

$$if\left(can \cdot roll\left(n_x, n_y, mod(curr.dir + 2,4), maze, used\right)\right):$$

$$used\left[n_x, n_y, mod(curr.dir + 2,4)\right] = true$$
$$route = curr.route + dirmap[mod(curr.dir + 2,4)]$$

$$Q.offer\left(Cell\left(n_x, n_y, mod(curr.dir + 2,4), route\right)\right)$$

$$if\left(can \cdot roll\left(n_x, n_y, mod(curr.dir + 3,4), maze, used\right)\right):$$

$$used\left[n_x, n_y, mod(curr.dir + 3,4)\right] = true$$
$$route = curr.route + dirmap[mod(curr.dir + 3,4)]$$

$$Q.offer\left(Cell\left(n_x, n_y, mod(curr.dir + 3,4), route\right)\right)$$

$return$ impossible

$bool\ can \cdot roll(x, y, d, maze, used)$:
$n_x = x + dirs[d, 0]$
$n_y = y + dirs[d, 1]$
$m = len(maze), n = len(maze[0])$
$if\left(n_x \in [0, m-1]\ \&\ n_y \in [0, n-1]\ \&\ maze\left[n_x, n_y\right] = 0\ \&\ {\sim}used\left[n_x, n_y, d\right]\right)$:
$return\ true$
$return\ false$

---

749. Contain Virus **Bloomberg**

https://leetcode.com/submissions/detail/141727342/

$$\begin{bmatrix} 0^1 & |1| & 0^1 & 0 & 0 & 0 & 0 & 1 \\ 0^1 & |1| & 0^1 & 0 & 0 & 0 & 0 & 1 \\ 0 & \overline{0^1} & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} : day1$$

Because the left virus zone $(0,1),(1,1)$ has larger affect range, 5 cells $(0,0),(1,0),(0,2),(2,1),(1,2)$ will be affected next day without wall, while the right virus zone

$(0,7),(1,7),(2,7)$ just affected 4 cells. So, at day 1, it is better to choose the left virus zone as the defending area. The key is that the number of walls need for quarantine:

$$walls = number\ of\ times\ that\ 1\ enters\ 0$$

$$\begin{bmatrix} 0 & |2| & 0 & 0 & 0 & 0^1 & |1 & 1 \\ 0 & |2| & 0 & 0 & 0 & 0^1 & |1 & 1 \\ 0 & \overline{0} & 0 & 0 & 0 & 0^1 & |\_1\_ & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0^{1+1} & |1 \end{bmatrix} : day2$$

At day2, the cells of $(0,6),(1,6),(2,6),(3,7)$ haves already been affected, and the cells of $(0,5),(1,5),(2,5),(3,6)$, to be noticed, the cell of $(3,6)$ will be affected twice, the wall need for quarantine is 5.

```
m = grid.length, n = grid_0.length
total · walls = 0
while(true):
    * virus · area denotes the virus zone which affects the max number of cell 0
    used_set = {φ}, virus · area_list = {φ}, nexts_list⟨set⟩ = {φ}
    block · index = −1, block · walls = −1
    for(i = 0 → m − 1):
        for(j = 0 → n − 1):
            id = x * n + y
            if(grid_i,j ≠ 1 or used[id] ≠ null): continue
            curr_list = {φ}, next_set = {φ}, walls = 0
            get · area(i, j, m, n, grid, used, curr, next, walls):
            if(next.size = 0): continue
            if(nexts.size = 0 or next.size > nexts[block · index].size):
                swap(virus · area, curr)
                block · index = nexts.size
                block · walls = walls
            nexts.add(next)
    total · walls+= block · walls
    if(nexts.size == 0): break

    for(i = 0 → nexts.size − 1):
        if(i == block · index): * for blocking virus area
```

$for(k: virus \cdot area): grid(\frac{k}{n}, k - \frac{k}{n} * n) = 2$

$else: * for\ on - blocking\ virus\ area$

$for(k: virus \cdot area): grid(\frac{k}{n}, k - \frac{k}{n} * n) = 1$

==========

$get \cdot area(x, y, m, n, g_{2d}, used_{set}, curr_{list}, next_{set}, walls)$:

$id = x * n + y$

$if\big(x < 0\ or\ x \geq m\ or\ y < 0\ or\ y \geq n\ or\ g_{x,y} == 2\big): return$

$if(g_{x,y} == 0)$:

$walls = walls + 1$

$next.add(id)$

$return$

$if(used[id] \neq nil): return$

$used.add(id)$

$curr.add(id)$

$for(i = 0 \rightarrow dirs.length - 1)$:

$nx = x + dirs_{i,0}, ny = y + dirs_{i,1}$

$get \cdot area(nx, ny, m, n, g_{2d}, used_{set}, curr_{list}, next_{set}, walls)$

---

489. Robot Room Cleaner

Given a robot cleaner in a room modeled as a grid. Each cell in the grid can be empty or blocked. The robot cleaner can move forward, turn left or turn right. When it tries to move into a blocked cell, its bumper sensor detects the obstacle and it stays on the current cell.

$interface\ Robot$:

$boolean\ move()$:

$void\ turnLeft(k)$:

$void\ turnRight(k)$:

$void\ clean()$:

$boolean\ move(Direction\ d)$:

Intuition & Algorithm

$void\ cleanRoom(Robot\ robot)$:

$int\ x \coloneqq 0, y \coloneqq 0, direction \coloneqq 0 * initially, robot\ face\ up$

$seen \coloneqq \{\phi\}$

```
    dfs(robot, x, y, d)

void dfs(robot, x, y, d):
    robot.clean()
    for(i = 0 → 3):
        nd := mod((d + i), 4)
        n_x := x + dirs[nd, 0], n_y := y + dirs[nd, 1]
        string location := n_x + "," + n_y
        if(seen[location] = false):
            seen.add(location)
            if(robot.move()):
                dfs(robot, n_x, n_y, nd)
                backoff(robot)
        robot.turnLeft()

void backoff(robot):
    robot.turnLeft()
    robot.turnLeft()
    robot.move()
    robot.turnLeft()
    robot.turnLeft()
```

Maze Treasure Problem Google

Solution: multiple BFS, for reach round of BFS, if it finds the treasure, just return true, in other cases, if the types and number of keys undergo some changes, it is possible to find out a path to treasure. Otherwise, if case nothing has happen to the keys we have, meaning we are entering a dead end and never have the chance to get the treasure.

726. Number of Atoms Google

Given a chemical formula (for example, "$K_4(ON(SO_3)_2)_2$"), return the count of each atom.

Algorithm

Use dfs to parse the whole formula, scan the formula letter by letter, in case meeting a left round bracket, making a recursive call, if meeting a right round bracket, terminate the "dfs" and return partial results, otherwise, do "$getName$" and "$getCount$".

```
TreeMap⟨String, Integer⟩ countAtoms(String formula, int index)
    TreeMap⟨String, Integer⟩ counts = {ϕ}
    while(index < len(formula))
        if(formula[index] = ( ):
            index = index + 1
            ⋆ After this call, index will point to a digit
            sub · counts = countAtoms(formula, index)
            times = getCount(formula, index)
            for(String atom: sub · counts. keySet):
                counts[atom] = counts[atom] + sub · count[atom] ⋆ times
        else if(formula[index] =) ):
            index = index + 1
            return counts
        else:
            String atom = getAtoms(formula, index)
            int times = getCount(formula, index)
            counts[atom] = counts[atom] + times
    return counts

String getAtoms(String formula, int index)
    s = formula[index]
    index = index + 1
    while(index < len(formula) & formula[index]. isLowerCase):
        s = s + formula[index]
        index = index + 1
    return s

int getCount(String formula, int index)
    times = 0
    while(index < len(formula) & formula[index]. isDigit):
        times = times ⋆ 10 + (formula[index] − 0)
        index = index + 1
```

$return\ times = 0 \rightarrow 1 : times$

---

471. Shortest Encode String Google

Intuition & Algorithm

$f[i, j]$ as the shortest encoded string of string "s".

459. Repeated Substring Pattern Google Amazon

Given a non-empty string check if it can be constructed by taking a substring of it and appending multiple copies of the substring together.

$$bool\ repeat := (s + s).indexOf(s, 1) < len(s)$$

$org := s.substr(i, j + 1)$

$f[i, j] := org \Rightarrow CASE\ \#1$

$f[i, j] := \min_{\text{len}}(f[i, j], f[i, k] + f[k + 1, j])\ , \forall k \in [i, j - 1]. \Rightarrow CASE\ \#2$

$index = \big(org + org.indexOf(org, 1)\big)$:

$if\big(index < len(org)\big) : \Rightarrow CASE\ \#3$

$unitLen := index$

$encodedUnit := f[i, i + len - 1]$

$times = \frac{len(org)}{unitLen}$

$if\ (len(encodedUnit) < f[i, j].length) : f[i, j] := encodedUnit$

---

394. Decode String Google Facebook Coupang Yelp

Given an encoded string, return it's decoded string. i.e.$3[a]2[bc] \Rightarrow aaabcbc$

Intuition & Algorithm

$String\ decode(s)$:

$sb = ""$

$T := 0$

$while(index \leq len(s) - 1)$:

$if\ (s[index] \in [0,9])$:

$i := index$

```
            while(i < len(s) & s[i] ∈ [0,9]): i := i + 1
            T := int(s[index: i − 1])
            index := i
        elif(s[index] = []):
            index := index + 1
            sub := decode(s)
            for(i = 0 → T − 1): sb := sb + sub
            T := 0
        elif(s[index] = ]):
            index := index + 1
            return sb.tostring
        else: sb := sb + s[index]
    return sb.toString()
```

---

796. Rotate String

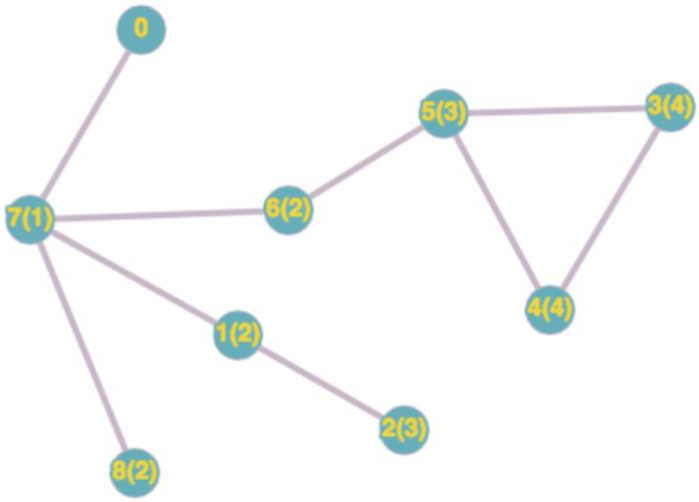
Question: given two strings A and B, check B is one rotation of A.

Idea: $(A + A).indexOf(B) \neq -1$

797. All Paths from Source to Target

Solution: dfs +backtrack

Similar Question: Find all paths from source to target



Hash, Tree Map, Binary Index Tree

Count Triple Inversion in an Array **Affirm**

Q: count such triplets of a_i, a_j, a_k s.t. $a_i > a_j > a_k$ & $i < j < k$.

Solution: Brute-Force, $O(n^3)$, TreeMap, $O(n^2)$, Binary Index Tree, $O(n^2)$

- Binary Index Tree is one of the simplified version of Segment Tree

Binary Tree structure itself in a way quite like Binary Tree, where some indices has parent indices which exercise controls over the child indices. The key of child & parent indices transition goes like: $index$ & $(-index)$. if walking from parent indices down to child indices, $index -= index$ & $(-index)$, while from child indices to parent indices: $index += index$ & $(-index)$. By convention, all indices used in BIT are starting from Zero(0).

The followings are the key operations of BIT, namely update & range-Query

---

$*$ *update operation goes from down to top*

$update(BIT_{1d}, n, index, val)$:

    $if\,(index \le n)$:

        $BIT[index] \mathrel{+}= val$

        $index \mathrel{+}= index$ & $(-index)$

$\star$ *update operation goes from down to top*

$query(BIT_{1d}, index)$:

    $sum = 0$

    $if\,(index \ge 1)$:

        $sum = sum + BIT[index]$

        $index := index - index$ & $(-index)$

    $return\ sum$

BIT is commonly used in counting how many elements are smaller than current element. Like

Count All Subarrays Sum to K **Hulu**

$countSubarraySumToTarget(int[\,]\ A, int\ target)$:

    $s := 0$

    $map: \{sum \Rightarrow times\}$

    $map[0] := 1$

$re \coloneqq 0$

$for(i = 0 \to len(A) - 1)$:

    $s \coloneqq s + A[i]$

    $if(map[s - target] \neq nil): re \coloneqq re + map[s - target]$

    $map[s] \coloneqq map[s] + 1$

$return\ re$

---

523. Continuous Subarray Sum Facebook

Given a list of non-negative numbers and a target integer k, write a function to check if the array has a continuous subarray of size at least 2 that sums up to the multiple of k, that is, sums up to n*k where n is also an integer.

Intuition & Algorithm

$map: \{presum \Rightarrow index\}$

$map[0] \coloneqq -1$

$s \coloneqq 0$

$for(i = 0 \to len(A) - 1)$:

    $s \coloneqq s + A[i]$

    $if(k \neq 0): s \coloneqq s\ \%\ k$

    $if(map[s] \neq nil)$:

        $if(i - map[s] > 1): return\ true$

    $if(map[s] = nil): map[s] \coloneqq i$

$return\ false$

Count All Subarrays Sum to Multiple of K Hulu

Intuition & Algorithm

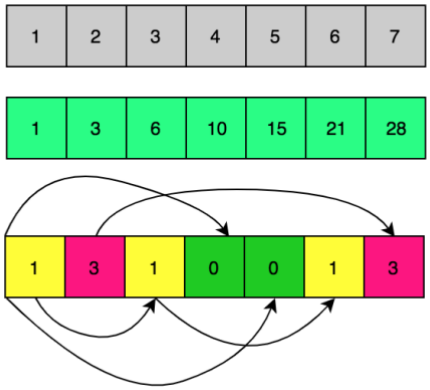
If there exists such a subarray $A[i: j]$ that:

$$\sum_{i}^{(j)} A[p] = c \star k$$

$$\sum_{0}^{i-1} A[p] + c \star k = \sum_{0}^{j} A[p]$$

Thus,

$$\left(\sum_{0}^{i-1} A[p]\right) \% \, k = \left(\sum_{i}^{j} A[p]\right) \% \, k$$



$countSubarraySumToTarget(int[\,]\, A, int\ k)$:

$s \coloneqq 0$

$int[\,]\, mod \coloneqq [k]$

$for(i = 0 \to len(A) - 1)$:

$s \coloneqq s + A[i]$

$mod[(s \,\%\, k + k) \,\%\, k] \coloneqq mod[(s \,\%\, k + k) \,\%\, k] + 1$

$re \coloneqq 0$

$\star$ count those subarrays which do not start at index of 0

$for(i = 0 \to k - 1)$:

$re \coloneqq re + \frac{mod[i] \star (mod[i]-1)}{2}$

$\star$ mod[0] means the number of prefix sum that are multiple of $\mathcal{K}$

$re \coloneqq re + mod[0]$

$return\ re$

Find All Substrings that are:

1: *exactly the same as* $\mathcal{S}$

2: $any\ rotation\ of\ S$

3: $shifted\ string\ of\ S$, i.e. $abc = shifted(def)$

249. Group Shifted Strings Google Uber

Given a string, we can "shift" each of its letter to its successive letter, for example: "abc" -> "bcd". We can keep "shifting" which forms the sequence:

$List[List[string]]\ groupStrings(String[\ ]\ strings)$:

    $map: encoded\ string \Rightarrow [words]$
    $for(string\ s: strings)$:
        $enc \coloneqq a$
        $first \coloneqq s[0]$
        $for(i = 1 \rightarrow len(s) - 1)$:
            $of \coloneqq ((s[i] - first)\ \%\ 26 + 26)\ \%\ 26$
            $enc \coloneqq enc + (of + a)$
        $map[enc].add(s)$
    $List[List[string]]\ re \coloneqq [\phi]$
    $for(List[string]\ line: map.values): re.add(line)$
    $return\ re$

---

315. Count of Smaller Numbers After Self

There is a kind of pattern for solving this type of problem.

$we\ need\ to\ preprocess\ array\ such\ like [7, -90, 100, 1] \rightarrow [3,1,4,2]$

$countSmallerAfterself(A)$:

    $for(i = n - 1 \rightarrow 0)$:
        $int\ count \cdot smaller = query(BIT, A[i] - 1)$
        $smaller[i] = count \cdot smaller$
        $update(BIT, n, A[i],1)$
    $return\ sum$

$countLargerBeforeself(array_{1d})$:

    $for(i = 0 \rightarrow n - 1)$:
        $int\ count \cdot larger = i - query(BIT, A[i])$
        $larger[i] = count \cdot larger$

$update(BIT, n, A[i], 1)$

Thus, for the problem of counting triple inversions:

$re \coloneqq 0$

$for(i = 0 \to n - 1)$:

$re \coloneqq re + smaller[i] \star larger[i]$

$return\ re$

---

325. Maximum Size Subarray Sum Equals k Facebook Palantir

Given an array *nums* and a target value k, find the maximum length of a subarray that sums to k. For example, *nums*: $[1, -1, 5, -2, 3], k = 3$, return 4. (because the subarray $[1, -1, 5, -2]$ sums to 3 and is the longest).

Solution: hash-map, keep track of the current sum as well as its ending index.

$maxSubarrayLen(nums[], k)$:

$n = nums.length, s = 0, result = 0$

$map = \{sum \Rightarrow index\}$

$map[0] = -1$

$for(i = 0 \to n - 1)$:

$s = s + nums[i]$

$if(map[s - k] \neq nil): result = \max(result, i - map[s - k])$

$if(map[s] = null): map[s] = i$

$return\ result$

---

734. Sentence Similarity I

737. Sentence Similarity II Google

I: similarities relationship is not transmittable

II: similarities relationship is transmittable

$areSentencesSimilar(words_1, words_2, pairs(string \to string))$:

$m = words_1.length, n = words_2.length$

$if(m \neq n): return\ false$

```
    for(pair in pairs):
        mp[pair.first] = pair.second
        mp[pair.second] = pair.first
    for(i = 0 → words1.length):
        if(words1[i] = words2[i]):continue
        if(mp[words1[i]] ≠ words2[i] & mp[words2[i]] ≠ words1[i]):return false
    return true

areSentencesSimilarII(words1, words2, pairs(string → string)):
    m = words1.length, n = words2.length
    if(m ≠ n):return false
    int e = 0
    * map each word with a unique id
    for(p in pairs):
        if(mp[p0] = null):mp[p0] = e++
        if(mp[p1] = null):mp[p1] = e++
        union(mp[p0], mp[p1])
    for(i = 0 → words1.length):
        if(words1[i] == words2[i]):continue
        r1 = find(mp[words1[i]]), r2 = find(mp[words2[i]])
        if(r1 ≠ r2):return false
    return true
```

My Calendar Trilogy

729. My Calendar I

731. My Calendar II

732. My Calendar III **Google**

Question: return false if double overbooking happens

```
bool book(start, end):
    if(map[start] ≠ null):return false
    l ← map.lowerKey(start), r ← map.higherKey(start)
    if((l ≠ null & start < map[l].end) or (r ≠ null & end < map[r].end)):
```

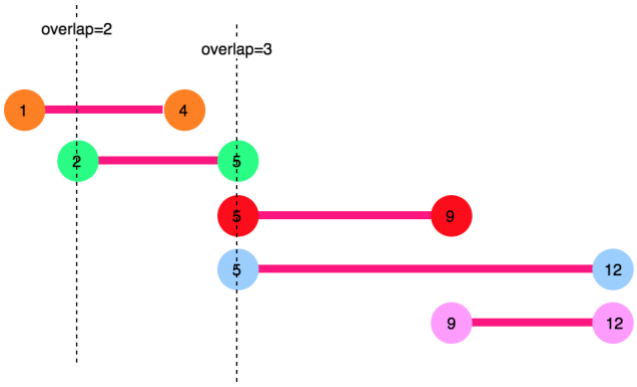
```
        return false
    map.put(start, end)
    return true
```

II. Question: if triple overbooking happens, cancel the triple booked event and return false, otherwise, return true.

```
timeLine = TreeMap⟨int, int⟩
bool book(start, end):
    timeLine[start] := timeLine[start] + 1
    timeLine[end] := timeLine[end] − 1
    int overLapNum = 0
    for(v: timeLine.values):
        overLapNum = overLapNum + v
        if(overLapNum ≥ 3):
            ⋆ cancel this trip
            timeLine[start] = timeLine[start] − 1
            timeLine[end] = timeLine[end] + 1
            return false
    return true
```

K-booking happens when K events have some non-empty intersection (ie., there is some time that is common to all K events.) return the value of K.

Idea: this is to find the maximum number of concurrent ongoing event at any time.




```
timeLine = TreeMap⟨int, int⟩ ⋆ track at time i, # of people come in, # of people exit
bool book(start, end):
    timeLine[start] = timeLine[start] + 1
```

$timeLine[end] = timeLine[end] - 1$

$int\ overLapNum = 0, result = 0$

$for(v: timeLine.values())$:⋆ *scan the whole timeline*

$\quad overLapNum = overLapNum + v$

$\quad result = \max(result, overLapNum)$

$return\ result$

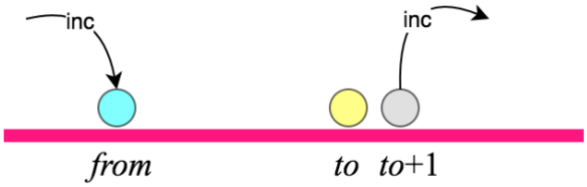
---

370. Range Addition Google

Assume you have an array of length n initialized with all 0's and are given k update operations. Each operation is represented as a triplet: $[startIndex, endIndex, inc]$ which increments each element of subarray $A[startIndex \dots endIndex]$ (startIndex and endIndex inclusive) with inc. Return the modified array after all k operations were executed.

Intuition & Algorithm

This problem is the same as the scenario of "get on or off the bus", at one triplet $[s, e, inc]$, it describes at the time of s, inc people get on the bus, and the time of $(e + 1)$. inc people get off the bus. So for this problem, we first keep track of the number of people changed at each time slot $t \in [0, n - 1]$ using an tool array $timeLine: \mathcal{R}^n$. Second, scan each slot on timeline, and update the number of people in the bus.



$getModifiedArray(int\ length, int[\ ][\ ]\ updates)$:

$\quad int[\ ]\ timeLine = [length]$

$\quad for(int[\ ]\ update: updates)$:

$\quad\quad from, to, inc = update[0], update[1], update[2]$

$\quad\quad timeLine[from] := timeLine[from] + inc$

$\quad\quad if(to + 1 < length): timeLine[to + 1] := timeLine[to + 1] - inc$

$\quad int\ v = 0, int[\ ]\ re = [length]$

$\quad for(i = 0 \to length - 1)$:

$v := v + timeLine[i]$

$re[i] = v$

---

170. Two Sum III - Data structure design Facebook Samsung

Implement two functions: add(value) - add the number to an internal data structure. & find(sum)- Find if there exists any pair of numbers which sum is equal to the value.

There are two types of solutions: if we do care more about the efficiency of "find" function.

$sum = \phi, num = \phi$

$void\ add(v)$:

    $if(num[v] \neq nil)$: $sum.add(v \star 2)$

    $else$:

        $for(int\ x: num)$: $sum.add(x + v)$

        $num.add(v)$

$bool\ find(s)$

    $return\ sum[s] \neq nil$

---

if we do care more about the efficiency of the "add" function.

$map: v \Rightarrow occurence\ times$

$void\ add(v): map[v] := map[v] + 1$

$bool\ find(s)$

    $for(x: map.keySet)$:

        $if(map[x] \neq nil\ \&\ map[x] \geq 2)$: $return\ true$

        $elif(map[s - x] \neq nil)$: $return\ true$

    $return\ false$

---

804. Unique Morse Code Words Google

Given a map which maps character to Morse-Coded string, and a list of normal words. Since different words may share the same morse-code, find out the most frequent morse-code.

Solution: create morse-code for each word then count morse-code using hash-map

Follow Up: Find the most frequent morse-code as well as corresponding word list

653. Two Sum IV - Input is a BST Facebook Samsung

Given a Binary Search Tree and a target number, return true if there exist two elements in the BST such that their sum is equal to the given target.

Intuition & Algorithm

Flatten BST to double linked list.

$TreeNode\ head \coloneqq nil, prev \coloneqq nil$

$bool\ findTarget(TreeNode\ \mathcal{R}, int\ k)$:

    $if(\mathcal{R} = nil)$: $return\ false$

    $flatten(\mathcal{R})$

    $l = head, r = prev$

    $while(l \neq r)$:

        $if(l.val + r.val = k)$: $return\ true$

        $elif(l.val + r.val < k)$: $l \coloneqq l.right$

        $else$: $r \coloneqq r.left$

    $return\ false$

$void\ flatten(TreeNode\ \mathcal{R})$:

    $if(\mathcal{R} = nil)$: $return$

    $flatten(\mathcal{R}.left)$

    $if(head = nil)$: $head \coloneqq \mathcal{R}, prev \coloneqq \mathcal{R}$

    $else$: $prev.right \coloneqq \mathcal{R}, \mathcal{R}.left \coloneqq prev, prev \coloneqq \mathcal{R}$

    $flatten(\mathcal{R}.right)$

Find a Number exists in a List of Intervals **Google**

Given a list of intervals, check if one number lies on one of them.

Solution: put those intervals into tree-map, for example, $[a, b] \rightarrow map[a]$: $[a, b]$, it is supposed to take $O(n \log n)\ time$. Next, for any given number x, check if $map[x]\ exists\ not\ not,$ if not, continue check $lb = map.lowerKey(x)$ & $rb = map.higherKey(x)$

$$if(lb \neq null\ \&\ map[lb].a \leq x \leq map[lb].b)$$

$$if(rb \neq null\ \&\ map[rb].a \leq x \leq map[rb].b)$$

Follow Up: if there exists overlaps within the given intervals

Solution: use stack to merge them first.

Vote Problem **Google**

$\mathcal{G}$iven a list of votes $\{candidate_{string}, timestamp_{int}\}$, design a function which takes the timestamp T as input, find the winner at time T.

$\mathcal{A}$pply a hash-map to keep track of number of votes per candidate, and maintain two variables, one is $maxVoteNum$ to keep track of maximal number of votes, another is $leaders(set)$ which record current winner(s) per timestamp. Then, we wrap this two variables into a self-defined class objects and store it in the list($timeline$). At the end, since the objects reside in the list are arranged by increasing order of timestamp, so we can apply binary-search to find one object whose timestamp is the most closet to given timestamp(T), then the candidates associated with the object is the answer.

Time Cost: $O(n)$ for building the timeline, $O(\log(n))$ for searching.

$for(Vote\ vote: list)$:

    $counts[vote.cand] := counts[vote.cand] + 1$
    $if(counts[vote.cand] = maxVotes)$:
        $leaders.add(vote.cand)$
    $elif(counts[vote.cand] > maxVotes)$:
        $maxVotes = counts[vote.cand]$
        $leaders.clear$
        $leaders.add(cand)$
    $timeline.add\big(Node(leaders, vote.timestamp)\big)$

$findLeadersSofar(T)$:

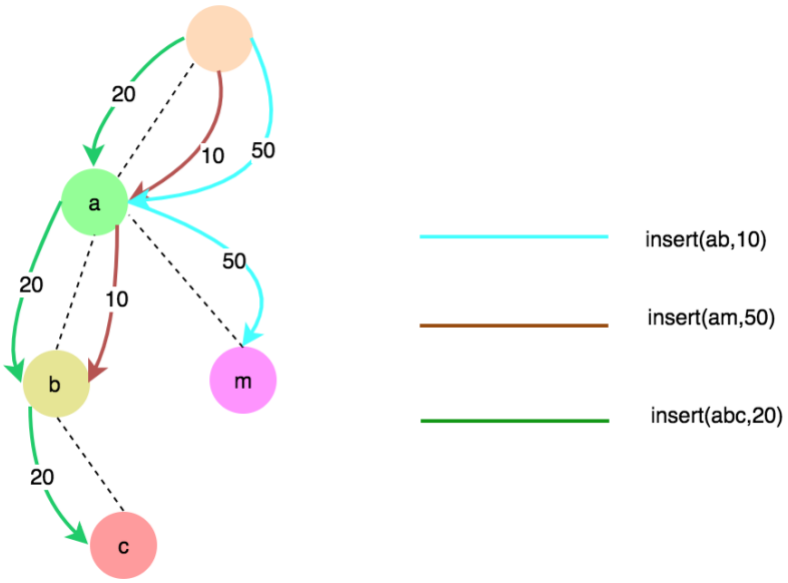
    $l = 0, r = n - 1, rid = -1$
    $while(l \leq r)$:
        $ptr_{mid} = \frac{(l+r)}{2}$
        $Node\ mid = timeline[ptr_{mid}]$
        $if(mid.ts > T): r = ptr_{mid} - 1$
        $elif(mid.ts < T): rid = ptr_{mid}, l = ptr_{mid} + 1$
        $else: rid = ptr_{mid}, l = l + 1$
    $return\ timeline[rid].cands$

---

677. Map Sum Pairs **Akuna Capital**

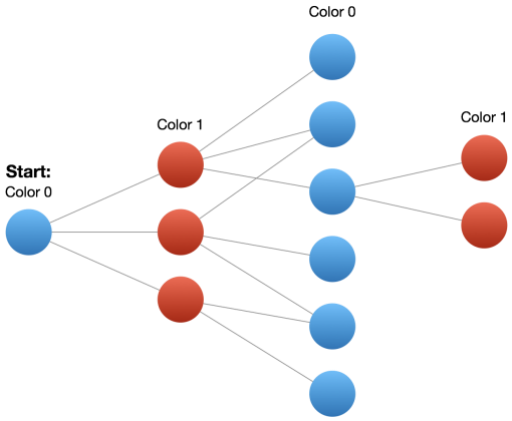
Algorithm

Let TrieNode has two attributes: *next*[26] & *sum*, when inserting a new word



Graph & Topo-sort & Union-Find

785. Is Graph Bipartite **Facebook**



Intuition & Algorithm

Start from each gray node, do BFS and coloring each adjacent node of current node with different color, in case adjacent node has been colored in the same color, return false.

---

bool isBipatrite(int[ ][ ] g):

$\mathcal{N} \coloneqq len(g)$

$color = [\mathcal{N}], color[i] \coloneqq \mathcal{GRAY}$

$for(i = 0 \to n - 1)$:

    $if(color[i] = \mathcal{GRAY})$:

        $color[i] \coloneqq \mathcal{RED}$

        $Q.offer(i)$

    $while(Q.size > 0)$:

        $v = Q.poll$

        $for(nv : g[v])$:

            $if(color[nv] = color[v]) : return\ false$

            $elif(color[nv] = \mathcal{GRAY}) : color[nv] = 1 - color[v]) : Q.offer(nv)$

$return\ true$

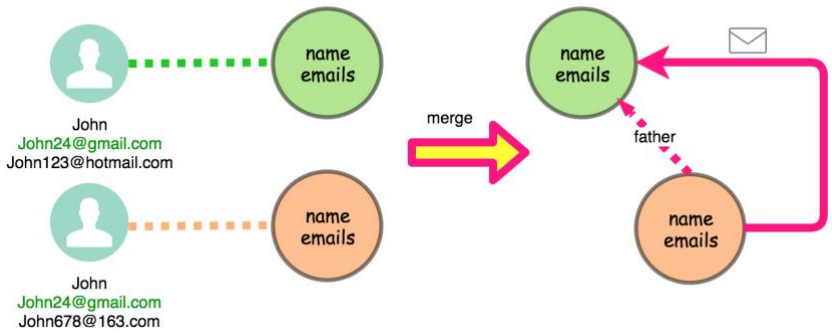
---

721. Accounts Merge **Facebook**

Given a list accounts, each element $accounts[i]$ is a list of strings, where the first element $accounts[i][0]$ is a name, and the rest of the elements are emails representing emails of the account. Now, we would like to merge these accounts. Two accounts definitely belong to the same person if there is some email that is common to both accounts. Note that even if two accounts have the same name, they may belong to different people as people could have the same name. A person can have any number of accounts initially, but all of their accounts definitely have the same name. After merging the accounts, return the accounts in the following format: the first element of each account is the name, and the rest of the elements are emails in sorted order.

Source Code

Solution: Union-Find



261. Graph Valid Tree Google Facebook Zenefits

Given a graph (number of vertex: n and a list of edges), determine if it is a tree.

Intuition & Algorithm

Union-find for each edge, if two end points of one edge belongs to the same group before union-find, then **return false** because a cycle is detected ($case$ #1). To be noticed, even case 1 doesn't happen, it does not mean the edges can form a tree, because they could form a "forest" or multiple trees. Thus, we need to check each vertex if they are all fall into only one group.

130. Surrounded Regions

Given a 2D board containing 'X' and 'O' (the letter O), capture all regions surrounded by 'X'. A region is captured by flipping all 'O's into 'X's in that surrounded region.

Intuition & Algorithm

This problem is similar to Number of Islands. In this problem, only the cells on the boarders cannot be surrounded. So we can first merge those O's on the boarders like in Number of Islands and replace O's with '#', and then scan the board and replace all O's left (if any).

$for\big((i,j) \in [m,n]\big)$:
    $if(board[i,j] = O)$:
        $if(i = 0\ or\ i = m-1\ or\ j = 0\ or\ j = n-1): uf.union(i \star n + j, dummy)$
        $else$:
            $if(i \geq 1\ \&\ board[i-1,j] = O): uf.union\big((i-1) \star n + j, dummy\big)$
            $if(i \leq m-2\ \&board[i+1,j] = O): uf.union\big((i+1) \star n + j, dummy\big)$
            $if(j \geq 1\ board[i,j-1] = O): uf.union(i \star n + j - 1, dummy)$
            $if(j \leq n-2\ \&\ board[i,j+1] = O): uf.union(i \star n + j + 1, dummy)$

$for\big((i,j) \in [m,n]\big)$:
    $if(board[i,j] = O)$:
        $if\big(uf.find(i \star n + j) \neq uf.find(dummy)\big): board[i,j] = X$

---

839. Similar String Groups Google

$X^{string} \approx Y^{string}\ if\ X.swap(i,j) = Y$

A group is such that a word is in the group if and only if it is similar to at least one other word in the group. We are given a list A of unique strings. Every string in A is an anagram of every other string in A. How many groups are there.

Algorithm

There are two solutions approaching this problem, one attempt is a standard kind of brute force: for each pair of words, let's draw an edge between these words if they are similar. We can do this in $O(N^2W)$ time. After, finding the connected components can be done in $O(N^2)$ time naively (each node may have up to $(N-1)$ edges). The total complexity is $O(N^2 \star W)$.

Another attempt is to enumerate all neighbors of a word. A word has up to $\binom{W}{2}$ neighbors, and if that neighbor itself is a given word, we know that word and neighbor are connected by an edge. In this way, we can build the graph in $O(NW^3)$ time, and again take $O(N^2)$ or $O(N)$ time to analyze the number of components.

Thus, in case $N^2W < NW^3$, namely $N < W^2$, we try the first attempt. Otherwise, we try the second attempt.

684. Redundant Connection Google

The given input is an **un-directed graph** that started as a tree with N nodes (with distinct values $1, 2, \ldots, N$), with one additional edge added, find the additional edge.

Algorithm

Union-find, if two vertices of current edge are found belong to the same group before union this edge, then return this edge.

685. Redundant Connection II Google

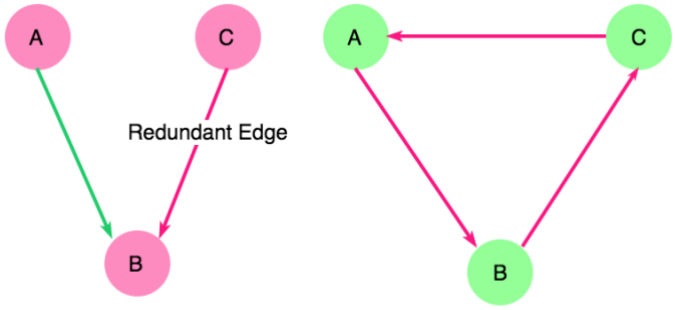
The given input is a **directed** graph that started as a tree with N nodes (with distinct values $1, 2, \ldots, N$), with one additional edge added, find the additional edge.

Algorithm

Idea: either of 2 properties may prevent a list of edges from becoming a tree:

1 $two\ parents\ node(a\ node\ whose\ indegree \geq 2)$

2 $there\ is\ loop$



Based on the idea, we firstly loop over the edges and find such a "two parents" node as well as associated two edges ($redundant_1, redundant_2$), one of them may be the redundant edge we want to find). Second step, loop over the edges array again and do "union-find" on each edge, once a loop is discovered, we check if $redundant_1$ is null or not, in case $redundant_1 \neq nil$, return it, otherwise, return the currently scanned edge. If in the second step, no loops are detected, return $redundant_2$ instead.

$N = getNumberOfVertices(edges)$

$parent: map(edge.to \Rightarrow edge.from)$

$redundant_1 = nil, redundant_2 = nil$

$for(e[]: edges)$:

    $from = e[0], to = e[1]$

    $if(parent[to] \neq nil)$: $\star$ *if two parents node detected*

        $redundant_1 = \{parent[to], to\}$

        $redundant_2 = \{from, to\}$

        $e[0] = e[1] = -1$ $\star$ *delete the edge*

    $parent[to] = from$

$for(e[]: edges)$:

    $if(e[0] = -1): continue$

    $boolean\ b = UnionFind.union(e[0] - 1, e[1] - 1)$

    $if(!b): return\ redundant_1 \neq nil: redundant_1: e$

$return\ redundant_2$

---

827. Making A Large Island Uber

In a 2D grid of 0s and 1s, we change at most one 0 to a 1. After, what is the size of the largest island? (An island is a 4-directionally connected group of 1s).

Algorithm

Apply Union-Find to union all islands, and for each of 0, find 4-directionally group of 1s.

$int\ largestIsland(int[\,][\,]\ grid)$:

$m = len(grid), n = len(grid[0]), N = m \star n$

$boolean[m][n]\ used, re = 0, ids: HashSet = null$

$for(i = 0 \to m - 1)$:

$for(j = 0 \to n - 1)$:

$if(grid[i,j] = 1\ \&\ {\sim}used[i,j])$:

$rid = i \star n + j$

$used[i,j] = true$

$\star\ union\ all\ islands\ that\ connected\ to\ island(i,j)$

$dfs(i, j, grid, used, rid, uf)$

$for(i = 0 \to m - 1)$:

$for(j = 0 \to n - 1)$:

$if(grid[i,j] = 0)$:

$area = 0$

$ids = new\ HashSet$

$if(i \geq 1\ \&\ grid[i-1,j] = 1): ids.add\big((i-1) \star n + j\big)$

$if(i < m - 1\ \&\ grid[i+1,j] = 1): ids.add\big((i+1) \star n + j\big)$

$if(j \geq 1\ \&\ grid[i,j-1] = 1): ids.add\big(i \star n + (j-1)\big)$

$if(j < n - 1\ \&\ grid[i,j+1] = 1): ids.add\big(i \star n + (j+1)\big)$

$for(rid: ids)\ area \coloneqq area + uf.clusterSize[rid]$

$re = \max(re, area + 1)$

$if(ids = nil): return\ 1$

$return\ re$

---

332. Reconstruct Itinerary(Euler Path) Google

Given a list of airline tickets represented by pairs of departure and arrival airports [from, to], reconstruct the itinerary in order. All of the tickets belong to a man who departs from JFK. Thus, the itinerary must begin with JFK.

Solution: find an Euler-Path (an Euler path is a path that uses every edge of a graph exactly once) with the smallest lexical order.

---

map: $(String \Rightarrow priorityqueue(String))$

$result = \{\phi\}$

$for(edge\ e: edges)$:

    $map[e.from].offer(e.to)$

dfs(JFK)

$dfs(curr, map)$:

    $if(map[curr] \neq nil\ \&\ map[curr].size > 0)$:

        $dfs(map[curr].poll, map)$

    $result.addFirst(curr)$

---

737. Sentence Similarity II Google

Given two sentences $words1, words2$ (each represented as an array of strings), and a list of similar word pairs $pairs$, determine if two sentences are similar. Note that the similarity relation is **transitive**. For example, if "great" and "good" are similar, and "fine" and "good" are similar, then "great" and "fine" are similar.

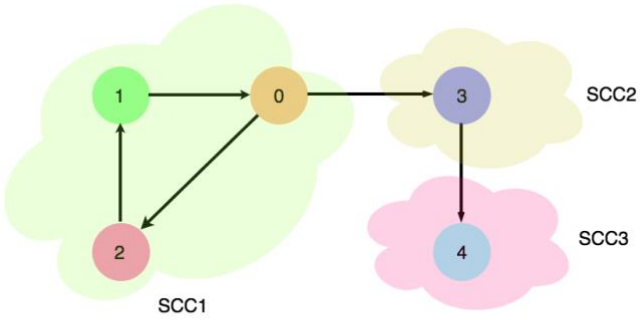
Idea: wrap each string with class Node object, then apply union-find

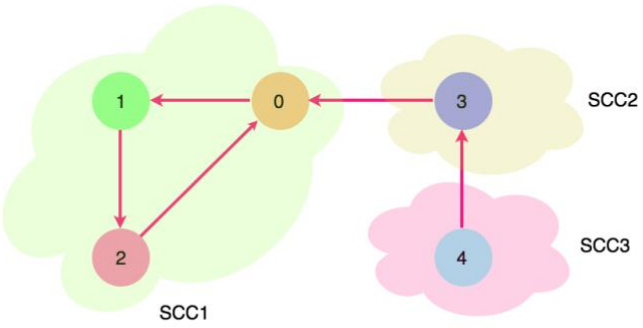
323. Number of Connected Components in an Undirected Graph Google Twitter

Solution: union-find, initially, $CC = n$, each time we union two groups, $cc = cc - 1$

Follow Up: Strong Connected Components in directed Graph

We can find all strongly connected components in $O(V + E)$ time using Kosaraju's algorithm. Following is detailed Kosaraju's algorithm.





Intuition & Algorithm

The above diagram suggests a fact that the number of SCC and each vertex contained in each SCC remains the same after we reverse each edge of graph.

Math & Geometry & Number

754. Reach a Number

You are standing at position 0 on an infinite number line. There is a goal at position target. On each move, you can either go left or right. During the n^{th} move (starting from 1), you take n steps. Return the minimum number of steps required to reach the destination.

To be noticed, $min \cdot steps(target) = min \cdot steps(-target)$. The strategy to walk to the target is to make the minimal number of trials to reach a position which is greater or equals to target.

$$1 + 2 + 3 + \cdots + k = target - dif$$

There are three cases to be considered:

$if(if = 0)$:

$result = k, (1 + 2 + \cdots + i + \cdots + k = target - 0)$

$elif\big(dif = 2i\ (even)\big)$:

$result = k, (1 + 2 + \cdots - i + \cdots + k = target - 2i, i = \frac{dif}{2} = Z^*)$

$elif\big(dif = 2i + 1\ (odd)\big)$:

$case_{31}: k = 2y\ (even)$,

$result = k + 1, (1 + \cdots - i + \cdots + k + k + 1 = target + k + 1 - 2i, i = \frac{k+1-dif}{2} = Z^*)$

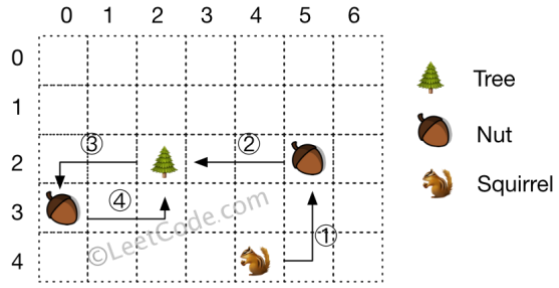
$case_{32}: k = 2y + 1\ (odd)$,

$result = k + 1, (1 + \cdots - i + \cdots + k + 1 + k + 2 = target + 2k + 3 - 2i, i = \frac{2k+3-dif}{2} = Z^*)$

---

573. Squirrel Simulation Square

There's a tree, a squirrel, and several nuts. Positions are represented by the cells in a 2D grid. Your goal is to find the minimal distance for the squirrel to collect all the nuts and put them under the tree one by one. The squirrel can only take at most one nut at one time and can move in four directions - up, down, left and right, to the adjacent cell. The distance is represented by the number of moves.



Intuition & Algorithm

$minDistance(height, width, int[\,]\ tree, int[\,]\ squirrel, int[\,][\,]\ nuts)$:

$int\ totDist = 0, maxDif = -2^{31}$

$for(int[\,]\ nut : nuts)$:

$totDist := totDist + 2 \star calcDistance(nut, tree)$

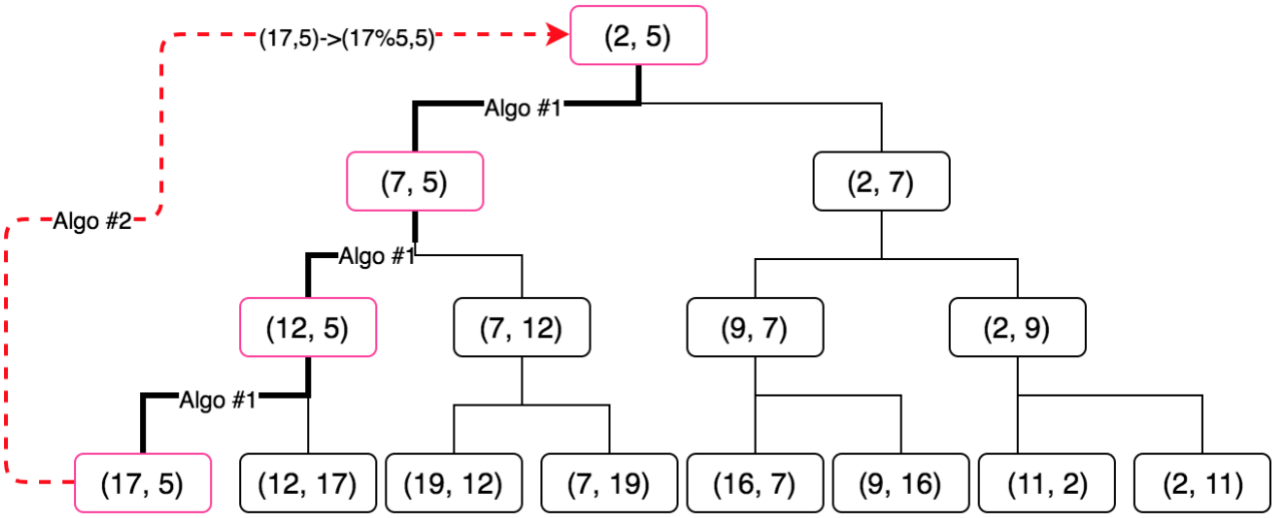
$maxDif = \max\big(maxDif, calcDistance(nut, tree) - calcDistance(nut, squirrel)\big)$

$return\ totDist - maxDif$

---

780. Reaching Points Google Coursera Facebook

A move consists of taking a point (x, y) and transforming it to either $(x, x + y)$ or $(x + y, y)$. Given a starting point (s_x, s_y) and a target point (t_x, t_y). Determine if (s_x, s_y) can make to (t_x, t_y).



Intuition

Every parent point (x, y) has two children, $(x, x + y)$ and $(x + y, y)$. However, every point (x, y) only has one parent candidate $(x - y, y)$ if $x \geq y$, else $(x, y - x)$. This is because we never have points with negative coordinates. Looking at previous successive parents of the target point, we can find whether the starting point was an ancestor. For example, if the target point is $(19, 12)$, the successive parents must have been $(7, 12)$, $(7, 5)$, and $(2, 5)$; *so* $(2, 5)$ is a starting point of $(19, 12)$.

Algorithm #1

Starting from (t_x, t_y), if $t_x > t_y$, replacing t_x with $t_x = t_x - t_y$. Otherwise, replacing t_y with $t_y = t_y - t_x$. Time Cost: $O(\max(t_x, t_y))$

Algorithm #2

We can speed up the above transformation by bundling together parent operations, if $t_x > t_y$, replacing $t_x = mod(t_x, t_y)$. Otherwise, replacing t_y with $t_y = mod(t_y, t_x)$. Time Cost: $O(\log(max(t_x, t_y)))$, As the diagram suggests, at the point of $(17,5)$ instead of jump to its parent point of $(12,5)$, directly jumps to its ascendant point of $(2,5)$.

$reaching \cdot points(s_x, s_y, t_x, t_y)$:

$while(t_x \geq s_x \ \& \ t_y \geq s_y)$:

$if(t_x > t_y): t_x = mod(t_x, t_y)$

$else: t_y = mod(t_y, t_x)$

$if(t_x = s_x): return\ mod(t_y - s_y, s_x) = 0$

$if(t_y = s_y): return\ mod(t_x - s_x, s_y) = 0$

$return\ false$

---

625. Minimum Factorization Tencent

Given a positive integer a, find the smallest positive integer b whose multiplication of each digit equals to a. If such an integer doesn't exist, return 0 instead. For example, $48 \Rightarrow 68(6 \star 8 = 48)$

Idea: divide integer a using divisors from $9 \rightarrow 2$ until a cannot be divided or equals to 1. The idea behind dividing bigger divisors at-first is making divisors set smaller, hence making the multiplication of divisors set smaller. For example, $48 = (68)8 \star 6 = (344)4 \star 4 \star 3$. Obviously, dividing a by bigger divisor can reduce the size of final divisors set. The second step is to traverse the divisors list backwards or from smaller to larger one, and calculate the number formed by those divisors or digits.

Magic Number Sequence Google

The number sequence is built based on two formulas,

$n = \frac{n}{2}, (n \,\&\, 1 = 0)$

$n = 3 \star n + 1, (n \,\&\, 1 = 1)$

And no matter what the n is, the sequence will reach 1, for example,

$n = 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

$n = 15 \rightarrow 46 \rightarrow 23 \rightarrow 70 \rightarrow 35 \rightarrow 106 \rightarrow 53 \rightarrow 160 \rightarrow 80 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow \cdots \rightarrow 1$

Used $steps(n)$ to denote the number of step to reach 1, for example,

$steps(16) = 7$

$steps(15) = 20$

So we got a table mapping from n to steps(n):

n	1	2	3	4	5	6	7
$steps(n)$	0	1	7	2	5	8	15
$records(n)$	T	T	T	F	F	T	T
	(1)	(1,2)	(1,2,3)	(ϕ)	ϕ	(1,2,3,6)	(1,2,3,6,7)

```
steps(n):
    if(n == 1): return 0
    if(map[n] ≠ nil): return map[n]
    int v = (n&1) = 0? steps(n/2) + 1: steps(3 * n + 1) + 1
    map[n] = v
    return map[n]

records(n):
    currmax = 0
    for(i = 1 → n):
        si = steps(i)
        if(si > currmax):
            currmax = si
            results.add(i)
        currmax = max(currmax, si)
    return results
```

386. Lexicographical Numbers Bloomberg

440. K^{th} Smallest in Lexicographical Order Hulu

Given an integer n, return $1 - n$ in lexicographical order

For example, given 13, return: [1,10,11,12,13,2,3,4,5,6,7,8,9]

Solution: this problem is about how to find $x's$ next lexicographically larger element, and the key to problem is to find where the most-important-digit (msd) of 9 is.

```
int next(x)
    if(10 * x ≤ n): return 10 * x (the closet larger one is to append 0 after x)
    elif(mod(x, 10) = 9):
        while(mod(x, 10) = 9): x = x/10 (i.e. x = 199, n = 199, the msd = 1, next(1) = 1 + 1)
        return x + 1
    elif(mod(x, 10) ≠ 9):
        if(x + 1 ≤ n): return x + 1(i.e. x = 123, n = 200, next closet larger one is 124)
        else:
            x = x/10 (i.e. x = 192, n = 192, next closet larger one is 2)
```

$while(mod(x, 10) \neq 9): x = \frac{x}{10}$

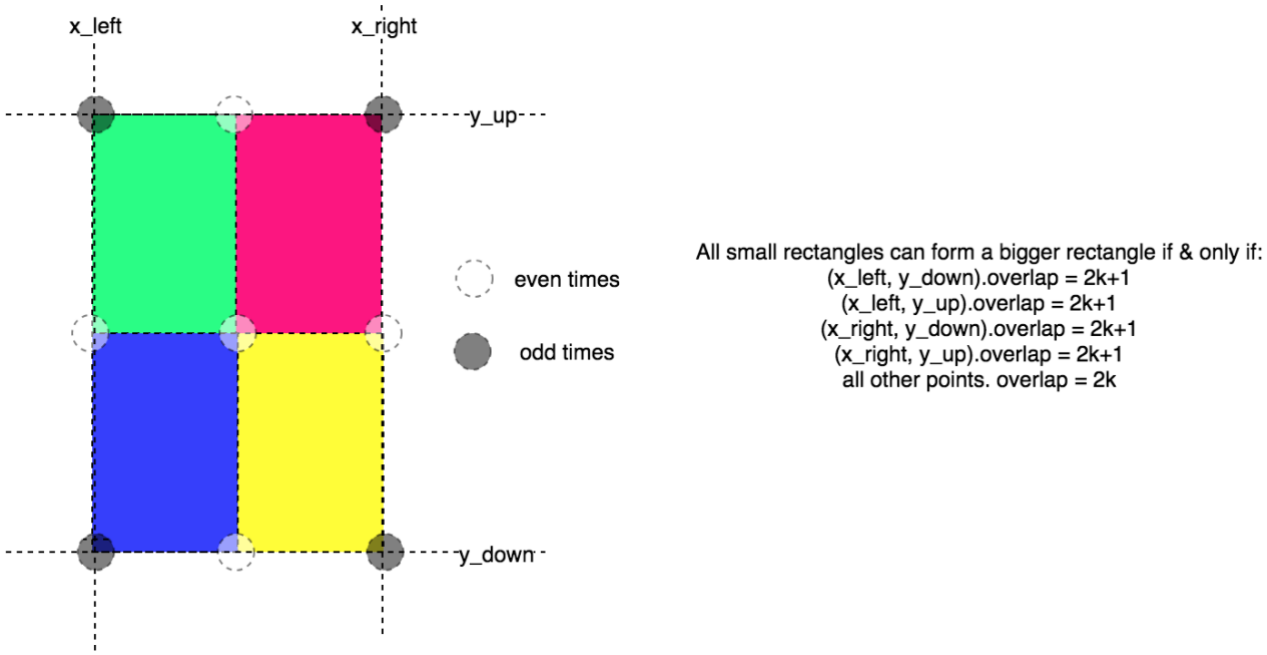
$return\ x + 1$

---

391. Perfect Rectangle Google

Given N axis-aligned rectangles where N > 0, determine if they all together form an exact cover of a rectangular region. (If there are overlaps, the rule is 偶消奇不消).

Idea: two conditions make true.



Intuition

There are two rules should be followed if the overlapping rectangles can form a rectangle.

Rule #1, the area of rectangular boundary = areas of all rectangles.

Rule #2, based on the rule of **even counteract while odd doesn't**, use hash-set to keep track of the 4 points of each rectangle, if one point shows even times, remove, otherwise, add it into set. Finally, if there are exactly only four points saved, and those four points are the same as rectangular boundary points

---

$bool\ isRectangleCover(int[\,][\,]\ rectangles)$:

$mi \coloneqq 2^{31} - 1, mx \coloneqq -2^{31}$

$x\ell, xr, yd, yu \coloneqq (mi, mx, mi, mx)$

$Set[String]\ used = \{\phi\}$

$for(i = 0 \to len(rectangles) - 1)$:

$x_1, y_1, x_2, y_2 \coloneqq rectangles[0,1,2,3]$

$x\ell := \min(x\ell, x_1), xr := \max(xr, x_2), yd := \min(y_1, yd), yu := \max(yu, y_2)$

$currArea := (x_2 - x_1) \star (y_2 - y_1)$

$totArea := totArea + currArea$

$if(used.add(x_1 + y_1) = \mathcal{F}alse): used.remove(x_1 + y_1)$

$if(used.add(x_1 + y_2) = \mathcal{F}alse): used.remove(x_1 + y_1)$

$if(used.add(x_2 + y_1) = \mathcal{F}alse): used.remove(x_1 + y_1)$

$if(used.add(x_2 + y_2) = \mathcal{F}alse): used.remove(x_1 + y_1)$

$if(used.size \neq 4 \;||\; used[x\ell] \neq nil \;||\; used[xr] \neq nil \;||\; used[yd] \neq nil \;||\; used[yu] \neq nil):$

$return\ \mathcal{F}alse$

$return\ totArea = (xr - x\ell) \star (yu - yd)$

---

850. Rectangle Area II

Follow Up: Select a Random Point from Multiple Rectangles **Google**

Follow Up: Rectangles may intersect each other

Solution: use fenwick-tree on y-axis, like the following 2 diagrams suggest. In specific, each rectangle can be suggested with left & right boundary vertical lines. Each line object is described as $\{x, y_{down}, y_{up}, in/out\}$. As for fenwick-tree, each tree node is described as $\{x, y_{down}, y_{up}, covers, isLeaf, leftchild, rightchild\}$.

Firstly, projecting rectangles with a list of vertical line objects. Each rectangle corresponds to two vertical lines, the incoming edge and exit edge, and two y values (y_{up}, y_{down}). Thus, for n rectangles, we end up with getting $2n$ values: $(y_0, y_1, \cdots, y_{2n-1})$

$function \Rightarrow (List\langle Rectangle\rangle\ rects):$

$lines = [\phi], y = [\phi], i = 0$

$for(Rectangle\ rect: rects):$

$lines[i] = \big(x = rect.x_1, y_{up} = rect.y_2, y_{down} = rect.y_1, in = 1\big)$

$y[i] = rect.y_1$

$lines[i] = \big(x = rect.x_2, y_{up} = rect.y_2, y_{down} = rect.y_1, in = -1\big)$

$y[i^{++}] = rect.y_2$

$Arrays.sort(y)$

$Arrays.sort\big(lines, (la, lb) \rightarrow (la.x - lb.x)\big)$

Second, build fenwick-tree based on the vertical lines.

$Node\ buildFenwickTree(y, l, r)$:

$if(l \geq r)$: $return\ nil$

$root = Node(y[l], y[r], isLeaf = false)$

$if(l + 1 = r)$: $return\ root.isLeaf = true$

$root.left = buildFenwickTree\left(y, l, \frac{(l+r)}{2}\right)$

$root.right = buildFenwickTree\left(y, \frac{(l+r)}{2}, r\right)$

$return\ root$

Third, insert each line into fenwick-tree & find the segmented rectangles. What this function does is to break down the segment (y_{down}, y_{up}), into smaller pieces until the small piece exactly match the leaf node$(y_{down}, y_{up}) = (leaf.y_{down}, leat.y_{up})$. once it matches up, it may form a small rectangle if this leaf node has been hosted by one or multiple y in-coming edges. At the end, we need to update the covers of this leaf node based on whether current edge in coming edge or exit one.

$int\ insert(Node\ p, x, y_{down}, y_{up}, in/out)$

$if(p = nil\ or\ p.y_{down} \geq y_{up}\ or\ p.y_{up} \leq y_{down})$: $return\ 0$

$if(p.isLeaf)$

$area = 0$

$if(p.covers > 0)$:

$prev = p.x$

$area = (x - prev) \star (p.y_{up} - p.y_{down})$

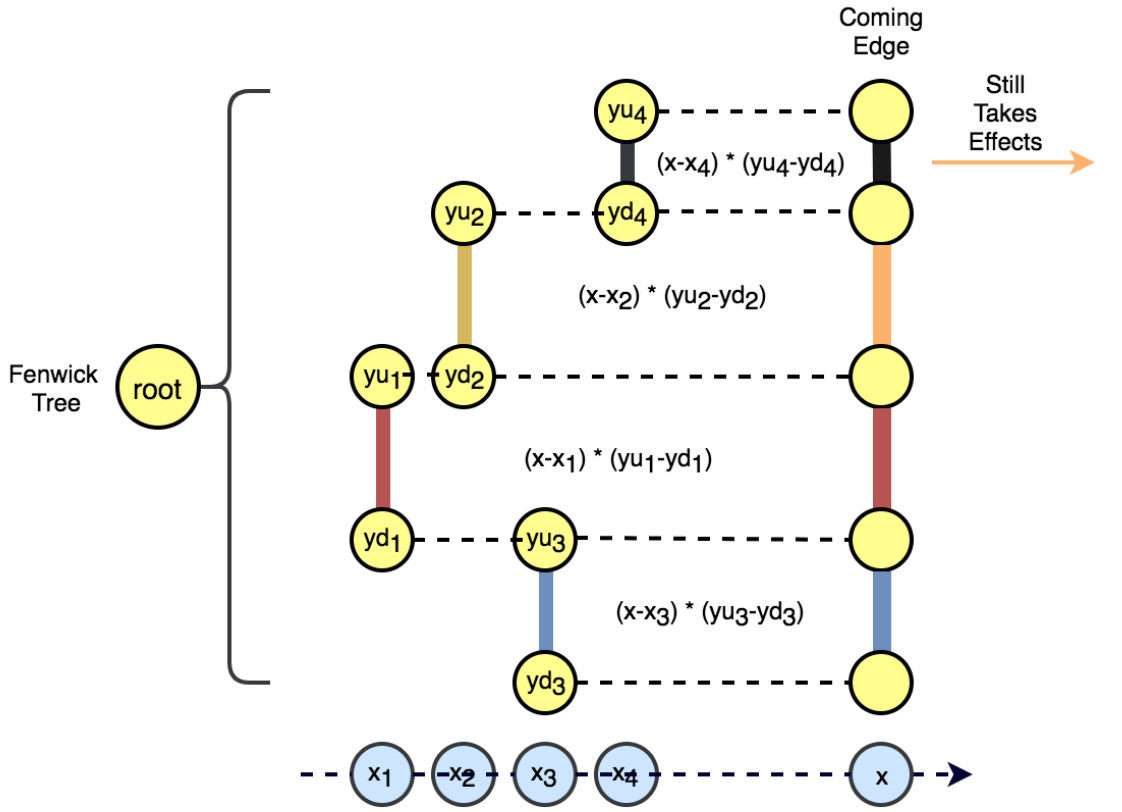
$blocks.add\left(Rectangle(prev, y_{down}, x, y_{up})\right)$

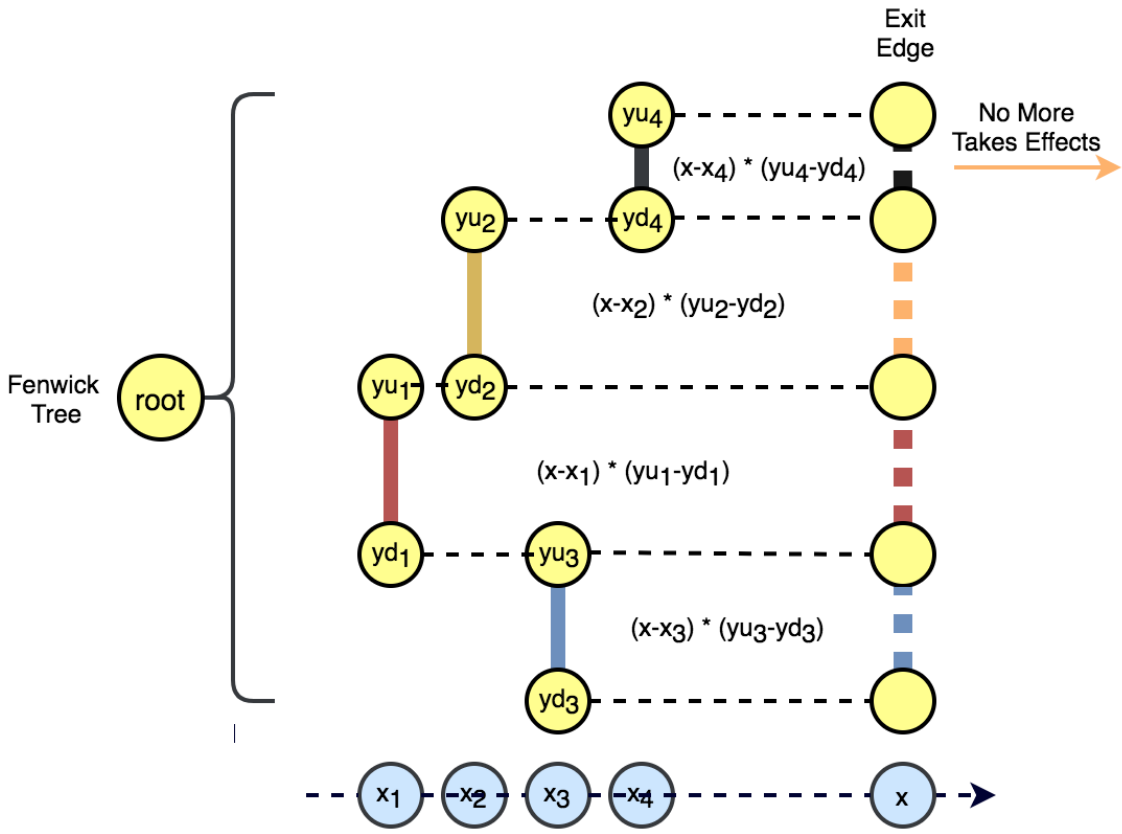
$p.cover = p.cover + in(1)/out(-1)$

$p.x = x$

$return\ area$

$return\ insert(p.left, y_{down}, y_{up}, in/out) + insert(p.right, y_{down}, y_{up}, in/out)$


Coming
Edge
Still
Takes
Effects
Fenwick
Tree
root
yu4
yd4
yu2
yd2
yu1
yd1
yu3
yd3
(x-x4) * (yu4-yd4)
(x-x2) * (yu2-yd2)
(x-x1) * (yu1-yd1)
(x-x3) * (yu3-yd3)
x1
x2
x3
x4
x


Exit
Edge
No More
Takes Effects
Fenwick
Tree
root
yu4
yd4
yu2
yd2
yu1
yd1
yu3
yd3
(x-x4) * (yu4-yd4)
(x-x2) * (yu2-yd2)
(x-x1) * (yu1-yd1)
(x-x3) * (yu3-yd3)
x1
x2
x3
x4
x

$selectFromSingleRectangle(Recntangle\ rect)$:

$x_1 = rect.x_1, x_2 = rect.x_2, y_1 = rect.y_1, y_2 = rect.y_2$

$return\ \left(x_1 + rand.nextInt(x_2 - x_1), y_1 + rand.nextInt(y_2 - y_1)\right)$

$selectFromMultipleRectangle(Recntangle[]\ rects)$:

$selected = nil$

$for(rect : rects)$:

$x_1 = rect.x_1, x_2 = rect.x_2, y_1 = rect.y_1, y_2 = rect.y_2$

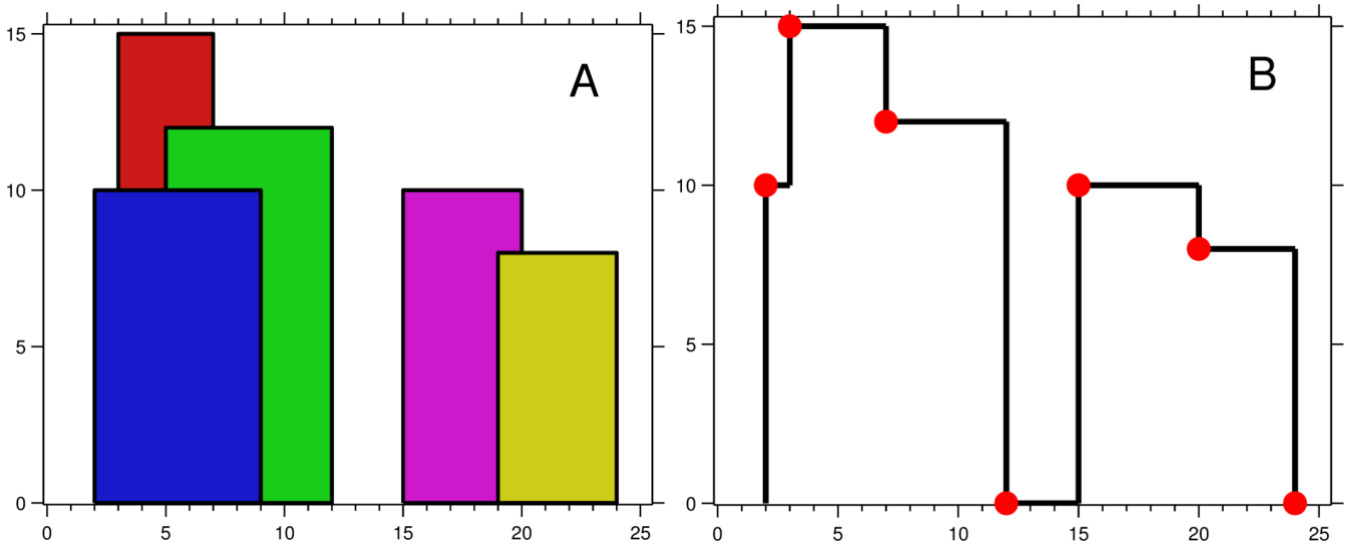
$area = (x_2 - x_1) * (y_2 - y_1)$

$if(rand.nextInt(total + area) \geq total) : selected = rect$

$total = total + area$

$return\ selectFromSingleRectangle(selected)$

218. The Skyline Problem Google Facebook Microsoft Twitter Yelp



Solution: borrow partial idea from "select random points from multiple rectangles" and presenting each rectangle with two vertical lines: incoming & exit edge. Let the incoming edge has negative y value and positive y value for exit edge. Then, scan sort those lines based on x value. Next, scan sorted vertical lines from left to right and use priority-queue to keep track of updated temporal max height.

```
for(b: buildings):
    height.add([b[0], −b[2]) ⋆ (x, −y)
    height.add([b[1], b[2])
Collections.sort(height, (a, b) → (a[0] = b[0]? a[1] − b[1]: a[0] − b[0]))
pqueue sorted = {ϕ}
sorted.offer(0)
prev = 0
for(h: heights):
    if(h[1] < 0): sorted.offer(−h[1])
    else: sorted.remove(h[1])
    currMax = sorted.peek
    if(prev ≠ currMax):
        re.add([h[0], currMax])
        prev = currMax
return re
```

String & Easy & Game Theory

243. Shortest Word Distance & II & III

I: $words$ = [practice, makes, perfect, coding, makes],

$dist(makes, coding) = 1, dist(practice, coding) = 3$

```
dist(word_a, word_b):
    p_a = −1, p_b = −1
    for(i = 0 → words.length − 1):
        if(words[i] = word_a): p_a = i
        if(words[i] = word_b): p_b = i
        result = min(result, abs(p_a − p_b))
```

If the *dist* function is executed many of times, then how to optimize the above solution

Solution: use a $Map\langle word, List\langle indice\rangle\rangle$ to keep track each word's occurrence positions.

Time Complexity: *dist* function takes $O(m + n)$, where m, n are the occurrence times of $word_a$ & $word_b$ respectively.

```
for(i = 0 → words.length − 1): mp[words_i].add(i)
```

$dist(word_a, word_b)$:

$la = mp[word_a], lb = mp[word_b]$

$i = 0, j = 0, minDif = 1 \ll 31 - 1$

$while(i < la.size \ \& \ j < lb.size)$

$if(abs(la[i] - lb[j]) < minDif): minDif = abs(la[i] - lb[j])$

$if(la[i] \leq lb[j]): i = i + 1 * if\ la[i] < lb[j], la[i+1]\ may\ be\ more\ closer\ to\ lb[j]$

$else: j = j + 1$

III: $word_a\ may\ equals\ word_b$

$dist(word_a, word_b)$:

$p_a = -1, p_b = -1$

$for(i = 0 \rightarrow words.length - 1)$:

$if(words[i] = word_a)$:

$if(p_a \neq -1 \ \& \ word_a = word_b): minDif = \min(minDif, i - p_a)$

$p_a = i$

$elif(words[i] = word_b): p_b = i$

$if(p_a \neq -1 \ \& \ p_b \neq -1): result = min(result, abs(p_a - p_b))$

---

161. One Edit Distance **Facebook** **Uber** **Twitter** **Snapchat**

Given two strings s and t, determine if they are both one edit distance apart. There are 3 possibilities to satisfy one edit distance apart: Insert a character into s to get t; Delete a character from s to get t; Replace a character of s to get t.

$if(len(s) = 0): return\ len(t) = 1$

$if(len(t) = 0): return\ len(s) = 1$

$m = len(s), n = len(t)$

$if(m = n)$:

$index = 1^{st} missMatchChar(s, t)$

$if(index = m): return\ false$

$if(index = m - 1): return\ true$

$return\ s[i+1:] = t[i+1:]$

$elif(m = n + 1)$:

$index = 1^{st} missMatchChar(s, t)$

$if\,(index = n)$: $return\ true$

$return\ s[i+1:] = t[i:]$

$elif\,(n = m + 1)$:

$index = 1^{st} missMatchChar(s, t)$

$if\,(index = m)$: $return\ true$

$return\ s[i:] = t[i+1:]$

$return\ false$

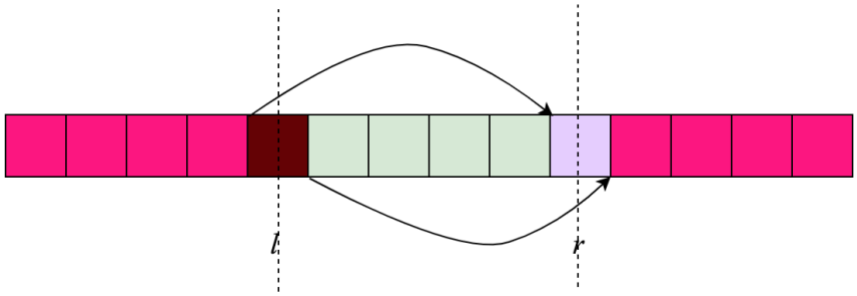
---

125. Valid Palindrome

Intuition: Apply Two Pointers

680. Valid Palindrome II Facebook

Given a non-empty string s, you may delete at most one character. Judge whether you can make it a palindrome.



Intuition & Algorithm

Find such (ℓ, r) s.t. $s[0:\ell-1] = s[r+1:n-1]$ and $s[\ell] \neq s[r]$. That means we can make s palindromic by deleting either $s[\ell]$ or $s[r]$. Thus, if we delete $s[\ell]$ and $s[\ell+1, r]$ is palindromic, then return true. If we delete $s[r]$ and $s[\ell, r-1]$ is palindromic, then return true also. Otherwise, return false.

$bool\ validPalindrome(String\ s)$:

$if\,(len(s) \leq 1)$: $return\ True$

$int\ l \coloneqq 0, r \coloneqq len(s) - 1$

$while(l \leq r)$:

$if\,(s[l] = s[r])$: $l \coloneqq l + 1, r \coloneqq r - 1$

```
        elif(check(s, l, r − 1) || check(s, l + 1, r)): return true
        else: return false
    return true
```

647. Palindromic Substrings Facebook LinkedIn

```
countSubstrings(String s):
    if(len(s) ≤ 1): return len(s)
    bool[ ][ ] f ≔ [len(s)][len(s)]
    for(i = 0 → len(s) − 1):
        for(j = 0 → i − 1):
            f[i, j] ≔ true
    for(len = 2 → n):
        for(i = 0 → n − len)
            j ≔ i + len − 1
            if(s[i] = s[j]): f[i, j] ≔ f[i, j] || f[i + 1, j − 1]
    for(i = 0 → n − 1):
        for(j = i → n − 1):
            re ≔ re + (f[i, j] = True) → 1: 0
    return re
```

Constant Space Solution

```
int countSubstrings(String s):
    for(i = 0 → len(s) − 1):
        c1 ≔ centerAt(s, i, i), c2 ≔ centerAt(s, i, i + 1)
        re ≔ re + (c1/2 + 1) + (c2/2)

int centerAt(s, i, j):
    ℓ ≔ i, r ≔ j
    while(ℓ ≥ 0 & r < len(s) & s[i] = s[j]): ℓ ≔ ℓ + 1, r ≔ r − 1
    return (r − ℓ + 1)
```

722. Remove Comments Microsoft

```
List⟨String⟩ remove · comments(source: string array):
```

```
re = {ϕ}
inblock = false
newline = StringBuilder
for(i = 0 → len(source) − 1):
    line = source[i]
    if(~inblock): newline = ""
    k = 0, n = len(line)
    while(k < n):
        if(~inblock & k + 1 < n & line[k] =/& line[k + 1] = ⋆):
            inblock = true, k = k + 2
        elif(inblock & k + 1 < n & line[k] =⋆ & line[k + 1] = /):
            inblock = false, k = k + 2
        elif(~inblock & k + 1 < n & line[k] =/ & line[k + 1] = /):
            break
        elif(~inblock): newline.append(line[k + +])
        else: k = k + 1
    if(~inblock & len(newline) > 0): re.add(newline)
return re
```

---

157. Read N Characters Given Read4 Facebook

The API: $int\ read4(char \star buf)$ reads 4 characters from a file per time.

```
int read(buf[char], n):
    cnt ≔ 0, counts = 0, tmp = [char]
    while ((cnt = read4(tmp)) > 0 & counts < n):
        i = 0
        while(i < cnt & counts < n): buf[counts++] = tmp[i++]
```

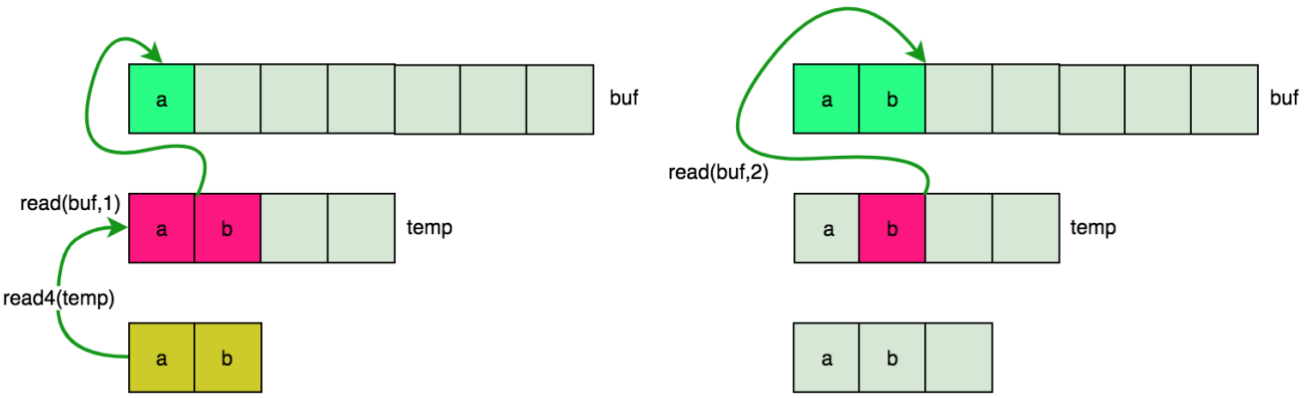
---

158. Read N Characters Given Read4 II - Call multiple times

Google Facebook Bloomberg

The API: $int\ read4(char \star buf)$ reads 4 characters at a time from a file. The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in

the file. By using the read4 API, implement the function $int\ read(char \star buf, int\ n)$ that reads n characters from the file.



Intuition & Algorithm

Use buffer pointer (ptr) and buffer Counter (cnt) to store the data received in previous calls. In the while loop, if ptr reaches current cnt, it will be set as zero to be ready to read new data.

$ptr \coloneqq 0 \Rightarrow \#\ of\ chars\ has\ been\ read$

$cnt \coloneqq 0 \Rightarrow \#\ of\ chars\ stored\ in\ warehouse$

$char[\,]\ warehouse = [4] \Rightarrow warehouse\ ,global\ variable\ for\ multiple\ call$

$int\ read(char[\,]\ buf, int\ n)$:

  $index \coloneqq 0 \Rightarrow number\ of\ chars\ read\ into\ buf$

  $while(index < n)$:

    $if(cnt = 0)$: $cnt \coloneqq \mathbf{read4}(warehouse) \Rightarrow if\ warehouse\ is\ empty, reload..$

    $if(cnt = 0)$: $break \Rightarrow if\ nothing\ stored\ in\ warehouse\ after\ reloading, break$

    $while(index < n\ \&\ ptr < cnt)$: $\Rightarrow copy\ data\ from\ warehouse \Rightarrow buf$

      $buf[index{+}{+}] \coloneqq warehouse[ptr{+}{+}]$

    $if(ptr = cnt)$: $\Rightarrow clear\ the\ warehouse$

      $ptr \coloneqq 0$

      $cnt \coloneqq 0$

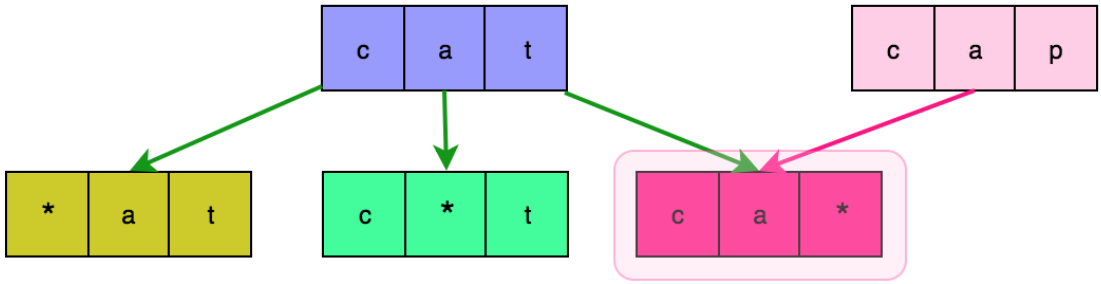
Key Take-away

$if(ptr = cnt)$: $ptr \coloneqq 0$

In case no more letter left during the previous call, clear the temp array

676. Implement Magic Dictionary Google

Given a word, determine if there is a word whose distance to it is exactly one. For example, the distance between “apply” and “apple” is 1.



Intuition & Algorithm

for each of the word in the dictionary, compute neighbor words.

$words = \phi, set$

$counts: map\ word \Rightarrow frequency$

$List[String]\ getNeighbors(word):$

    $neighbors = [\phi]$

    $for(i = 0 \rightarrow len(word) - 1):\star\ O\big(len(word[i])\big)$

        $temp = word[i]$

        $word[i] = \star$

        $neighbor = String(word)\ \star\ O\big(len(word[i])\big)$

        $neighbors.add(neighbor)$

        $word[i] = temp$

    $return\ neighbors$

$void\ build(String[\,]\ dict):\star\ \sum\big(len(words[i])\big)^2$

    $for(word: dict)$

        $words.add(word)$

        $neighbors = getNeighbors(word)$

        $for(nei: neighbors):\ counts[nei] \coloneqq count[nei] + 1$

$search(word):\star\ O(\mathcal{K}^2), \mathcal{K} = len(word)$

    $neighbors = getNeighbors(word)$

    $for(nei: neighbors):$

        $if\big(counts[nei] > 1\ or\ (counts[nei] = 1\ and\ words[word] \neq nil)\big): return\ true$

    $return\ false$

---

293. Flip Game Google

294. Flip Game II Google

Given a string that contains only these two characters: + and -, two players are playing a game where each player can only flip two consecutive "++" into "--". Determine if player one can win the game or not.

Solution: backtrack + **dp/hash-map** (map string to true/false, suggest player one will win based on current string or not).

$can \cdot win(s)$:

    $map(string \Rightarrow boolean)$

    $return\ can \cdot win \cdot util(s, map)$:

$can \cdot win \cdot util(s, map)$:

    $if(map[s] \neq nil)$: $return\ map[s]$

    $for(i = 0 \rightarrow s.length - 2)$:

        $if\big(s.startsWith(++, i)\big)$:

        $replace = s[0{:}\, i-1] + " --" + s[i+2{:}\, end]$

        $if(can \cdot win \cdot util(replace, map) == false)$:

            $map[s] = true$

            $return\ true$

    $map[s] = false$

    $return\ false$

Prefix to Postfix Affirm

792. Number of Matching Subsequences Google

Given string S and a dictionary of words $words$, find the number of $words[i]$ that is a subsequence of S.

Intuition & Algorithm

Check each word if it is subsequence of S, time cost is: $O\big(len(S) \star n \star len(word_i)\big)$.

Follow Up: if there are many of duplicates in the dictionary.

Solution: use two String hashsets:$subseq$ & $notSubseq$, if the word exists in either $subseq\ or\ notSubseq$, just return true or false respectively without checking again.

Meta Strings Google

Given two strings, check if the two are meta string, Meta strings are the strings which can be made equal by exactly one swap in any of the strings.

Intuition

go through the indices of string, and use a variable count the number of mismatches, if it is greater than 2, return false. Also, use two variables to denote the two indices $first, second$ s where mismatch happened. If $str1[first] = str2[second]\ and\ str1[second] = str2[first]$ return true, other-wise, return false.

506. Relative Ranks Google

Solution: sort the array first, then apply binary search.

$copy\ array \rightarrow temp$

$sort(temp)$

$for(i = 0 \rightarrow n - 1)$:

    $rank = Arrays.binarySearch(temp, nums[i])$

    $if(rank = n - 1): result[i] =$ gold Medal

    $elif(rank = n - 2): result[i] =$ silver Medal

    $elif(rank = n - 3): result[i] =$ Bronze Medal

    $else: result[i] = n - rank$

---

527. Word Abbreviation Google

Given a list of words, return a map which maps word to its abbreviation.

Idea: make abbreviation for each word. Then, check each word, if there are some strings which have same abbreviation with it, increase the prefix.

$wordsAbbrevation(dict_{string})$

    $prelen[i] = 1, for\ \forall i \in [0: dict.size - 1]$

    $abbr[i] = makeabbr(dict[i], prelen[i]), for\ \forall i \in [0: dict.size - 1]$

    $for(i = 0 \rightarrow n - 1)$:

        $set = \phi$

```
        while(true):
            for(j = i + 1 → dict.size − 1):
                if(abbr[i] == abbr[j]): set.add(j)
            if(set.empty): break
            set.add(i)
            for(x: set): abbr[x] = makeabbr(dict[x], + + prelen[x])
            set = ϕ
    return (dict[i] ⇒ abbr[i])

makeabbr(s, plen): ⋆ plen stands for the length of prefix
    if(plen + 2 ≥ s.length): if len(s.abbrevation) ≥ len(word), abbrevation lose its meaning
    abv = ""
    abv = abv + s[0: plen)
    abv = abv + len(s) − plen − 1
    abv = abv + s[s.length − 1]
    return abv
```

---

728. Self-Dividing Numbers Epic Systems

A self-dividing number is a number that is divisible by every digit it contains.

443. String Compression Microsoft Bloomberg Snapchat Yelp Expedia GoDaddy Lyft

824. Goat Latin **Facebook**

Rule #1, If a word begins with a vowel (a, e, i, o, or u), append "ma" to the end of the word.
Rule #2, If a word begins with a consonant (i.e. not a vowel), remove the first letter and append it to the end, then add "ma".
Rule #3, Add one letter 'a' to the end of each word per its word index in the sentence, starting with 1.

812. Largest Triangle Area **Google**

Area of Triangle = $\sqrt{p \star (p-a) \star (p-b) \star (p-c)}$, where $p = \frac{a+b+c}{2.0}$

```
largestTriangleArea(int[ ][ ] points):
```

$n = len(points)$

$re = 0.0$

$for\big((i, j, k) \in [0, n-1]\big)$:

$area = calcArea(points[i], points[j], points[k])$

$re = \max(re, area)$

$return\ re$

---

733. Flood Fill Uber

An image is represented by a 2-D array of integers. Given a coordinate (sr, sc) representing the starting pixel (row and column) of the flood fill, and a pixel value $newColor$, "flood fill" the image by changing all cells connected 4-directionally to the starting pixel with the same color.

$int[\,][\,]\ floodFill(int[\,][\,]\ image, int\ sr, int\ sc, int\ newColor)$:

$m = len(image), n = len(image[0])$

$if\,(image[sr, sc] = newColor)$: $return\ image$

$int\ color = image[sr, sc]$

$image[sr, sc] = newColor$

$DFS(sr, sc, image, color, newColor)$

$return\ image$

$void\ DFS(int\ r, int\ c, int[\,][\,]\ image, int\ color, int\ newColor)$:

$for\big((nr, nc) = getNext(r, c)\big)$:

$if\,(nr \in [0, len(image) - 1]\ \&\ nc \in [0, len(image[0]) - 1]\ \&\ image[nr, nc] = color)$

$image[nr, nc] = newColor$

$DFS(nr, nc, image, color, newColor)$

---

7. Reverse Integer Bloomberg Apple

$int\ reverse(x)$:

$sign = x > 0 \rightarrow 1 : -1$

$mx = 2^{31} - 1, x = |x|, re = 0$

$while(x > 0)$:

$d \coloneqq mod(x, 10)$

$if\left(re > \frac{mx-d}{10}\right)$: $Return\ 0$

$re \coloneqq re \star 10 + d$

$x \coloneqq \frac{x}{10}$

$return\ sign \star x$

---

8. String to Integer Microsoft Amazon Bloomberg Uber

$int\ myAtoi(String\ str)$:

$if(str = nil\ or\ len(str) = 0): return\ 0$

$str = str.trim$

$sign = str[0] = + \rightarrow 1: -1, index = 1, re = 0$

$while(index < len(str)\ and\ str[index] \in [0,9])$:

$if\left(re > \frac{mx-d}{10}\ and\ sign = -1\right): return\ -2^{31}$

$elif\left(re > \frac{mx-d}{10}\ and\ sign = 1\right): return\ 2^{31} - 1$

$else: re = re \star 10 + (str[index] - 0), index \coloneqq index + 1$

$return\ re$

---

598. Range Addition II IXL

Given an $m \star n$ matrix M initialized with all 0's and several update operations. Operations are represented by a 2D array, and each operation is represented by an array with two positive integers a and b, which means $M[i, j]$ should be added by one for all $0 \leq i < a$ and $0 \leq j < b$. You need to count and return the number of maximum integers in the matrix after performing all the operations.

$int\ maxCount(m, n, int[\ ][\ ]\ ops)$:

$minr, minc = -1, -1$

$for\big(i = 0 \rightarrow len(ops)\big)$:

$r = ops[i, 0], c = ops[i, 1]$

$if(minr = -1): minr = r$

$else\ minr = \min(r, minr)$

$if(minc = -1): minc = -1$

$else\ minc = \min(c, minc)$

$if(minr = -1): return\ m \star n$

$else: return\ minr \star minc$

---

791. Custom Sort String Amazon

S and T are strings composed of lowercase letters. In S, no letter occurs more than once.
S was sorted in some custom order previously. We want to permute the characters of T so that they match the order that S was sorted. More specifically, if x occurs before y in S, then x should occur before y in the returned string. Return any permutation of T (as a string) that satisfies this property.

Intuition & Algorithm

Firstly, count each char of T. Second, scan the char of S, in case such char exists in the T, append such char with corresponding frequency. Third, scan the char of S again, append chars which do not exist in T with corresponding frequency.

$String\ customSortString(String\ S, String\ T)$:

    $int[\,]\ counts = [26]$
    $for(char\ c: T): counts[c-a] \coloneqq counts[c-a] + 1$
    $re =$ ""
    $for(char\ c: S)$:
        $if(counts[c-a] > 0)$:
            $for(i = 0 \rightarrow counts[c-a] - 1): re \coloneqq re + c$
            $counts[c-a] \coloneqq 0 \star$ *mark this vahr as visited*
    $for(char\ c = a \rightarrow z)$:
        $if(counts[c-a] > 0)$:$\star$ *scan those unvisited chars*
            $for(i = 0 \rightarrow counts[c-a] - 1): re \coloneqq re + c$
    $return\ re$
