1. Two Sum

/\*\*

\* @param {number[]} nums

\* @param {number} target

\* @return {number[]}

\*/

var twoSum = function(nums, target) {

   var hash = {};

   var res = [];

   for(let i=0; i<nums.length; i++) {

       if(hash[target-nums[i]]!=undefined) {

           res.push(hash[target-nums[i]]);

           res.push(i);

           break;

       }

       hash[nums[i]] = i;

   }

   return res;

};

290. Word Pattern

var wordPattern = function(pattern, str) {

   if(str==null || str.length<pattern.length) return false;

   var arr = str.split(" ");

   if(arr.length!==pattern.length) return false;

   var hash\_p = {};

   var hash\_s = {};

   for(let i=0; i<pattern.length; i++) {

       if(hash\_p[pattern[i]]==undefined && hash\_s[arr[i]]==undefined) {

           hash\_p[pattern[i]] = i;

           hash\_s[arr[i]] = i;

       }

       else if(hash\_p[pattern[i]]==undefined || hash\_s[arr[i]]==undefined) {

           return false;

       }

       else if(hash\_p[pattern[i]]!=hash\_s[arr[i]]) {

           return false;

       }

   }

   return true;

};

291. Word Pattern II

var wordPatternMatch = function(pattern, str) {

   var map = new Map();

   var set = new Set();

   return helper(0,0,map,set);

   function helper(s\_idx, p\_idx, map, set) {

       if(s\_idx==str.length && p\_idx==pattern.length) {

           return true;

       }

       if(s\_idx==str.length || p\_idx==pattern.length) {

           return false;

       }

       //get current pattern character

       var c = pattern[p\_idx];

       //if pattern character exists

       if(map.has(c)) {

           var s = map.get(c);

           if(!str.startsWith(s,s\_idx)) {

               return false;

           }

           return helper(s\_idx+s.length, p\_idx+1, map, set);

       }

       for(let k=s\_idx; k<str.length; k++) {

           var p = str.substring(s\_idx, k+1);

           if(set.has(p)) {

               continue;

           }

           map.set(c, p);

           set.add(p);

           if(helper(k+1, p\_idx+1, map, set)) {

               return true;

           }

           map.delete(c);

           set.delete(p);

       }

       return false;

   }

};

609. Find Duplicate File in System

/\*\*

\* @param {string[]} paths

\* @return {string[][]}

\*/

var findDuplicate = function(paths) {

   var res = [];

   if(paths==null||paths.length==0) {

       res.push([]);

       return res;

   }

   var map = new Map();

   for(let path of paths) {

       var arr = path.split(" ");

       for(let i=1; i<arr.length; i++) {

           var content = arr[i].split('(')[1];

           var file = arr[0]+"/"+arr[i].split('(')[0];

           if(!map.has(content)) {

               map.set(content, []);

           }

           map.get(content).push(file);

       }

   }

   for(let files of map.values()) {

       if(files.length>1) {

           res.push(files);

       }

   }

   return res;

};

给了一个path之后，用stack来做DFS， 用两层的hashmap来验证duplicates, Map>>, 第一层key是file length, 第二层key是自己写的hash function 计算出来的结果, value是file path. 如果碰到的是file而不是directory, 要去首先验证length, 之后验证hash, 如果都一样，就放到list里。DFS之后因为hash并不是唯一的，要对duplicates进行filter, compare byte by byte, 这样才能找出最准确的duplicate. hash function 有各种方案，我用的是一个长度为6000的byte array, 在file beginning读2000， 在中间读2000， 在最后读2000，file小于6000B的话就全读。 python和java应该有native api支持file seek的。 然后SHA-512 来hash 这个 array. 把结果转化为hex string. Code我就不贴了，没存。

1. Imagine you are given a real file system, how will you search files? DFS or BFS ?

In general, BFS will use more memory than DFS. However **BFS** can take advantage of the locality of files in inside directories, and therefore will probably be faster

2. If the file content is very large (GB level), how will you modify your solution?

In a real life solution we will **not hash the entire file content,** since it's not practical. Instead we will first **map** all the **files** **according** **to** **size**. Files with different sizes are guaranteed to be different. We will then **hash a small part** of the files with equal sizes (using **MD5** for example). Only if the md5 is the same, we will **compare** the **files** **byte** **by** **byte**

3. If you can only read the file by 1kb each time, how will you modify your solution?

This won't change the solution. We can **create** the **hash** **from** the **1kb** chunks, and then read the entire file if a full byte by byte comparison is required.

4. What is the time complexity of your modified solution? What is the most time consuming part and memory consuming part of it? How to optimize?

Time complexity is **O(n^2 \* k)** since in worse case we might need to compare every file to all others. k is the file size

5. How to make sure the duplicated files you find are not false positive?

We will use several filters to compare: File **size**, **Hash** and **byte** **by** **byte** comparisons.

链接: <https://instant.1point3acres.com/thread/179277>

来源: 一亩三分地

17. Letter Combinations of a Phone Number

/\*\*

\* @param {string} digits

\* @return {string[]}

\*/

var letterCombinations = function(digits) {

   var res = [];

   if(digits==null||digits.length==0) {

       return res;

   }

   var abc = ['0','1','abc','def','ghi','jkl','mno','pqrs','tuv','wxyz'];

   res.push("");

   for(let digit of digits) {

       digit = parseInt(digit);

       var tmp = [];

       for(let choice of abc[digit]) {

           for(let r of res) {

               tmp.push(r+choice);

           }

       }

       res = [].concat(tmp);

   }

   return res;

};

289. Game of Life

/\*\*

\* @param {number[][]} board

\* @return {void} Do not return anything, modify board in-place instead.

\*/

var gameOfLife = function(board) {

   if(board==null || board.length==0 || board[0].length==0) return;

   var n = board.length;

   var m = board[0].length;

   for(let i=0; i<n; i++) {

       for(let j=0; j<m; j++) {

           let lives = liveNeighbors(i,j);

           if(board[i][j]==1 && lives>=2 && lives<=3) {

               board[i][j] = 3;

           }

           if(board[i][j]==0 && lives==3) {

               board[i][j] = 2;

           }

       }

   }

   for(let i=0; i<n; i++) {

       for(let j=0; j<m; j++) {

           board[i][j] >>= 1;

       }

   }

   function liveNeighbors(x, y) {

       var live = 0;

       for(let i=Math.max(x-1,0); i<=Math.min(x+1,n-1); i++) {

           for(let j=Math.max(y-1,0); j<=Math.min(y+1, m-1); j++) {

               live += board[i][j] & 1;

           }

       }

       live -= board[x][y] & 1;

       return live;

   }

};

362. Design Hit Counter

/\*\*

\* Initialize your data structure here.

\*/

var HitCounter = function() {

   this.times = new Array(300).fill(0);

   this.hits = new Array(300).fill(0);

};

/\*\*

\* Record a hit.

       @param timestamp - The current timestamp (in seconds granularity).

\* @param {number} timestamp

\* @return {void}

\*/

HitCounter.prototype.hit = function(timestamp) {

   let idx = timestamp%300;

   if(this.times[idx]!==timestamp) {

       this.times[idx] = timestamp;

       this.hits[idx] = 0;

   }

   this.hits[idx]++;

};

/\*\*

\* Return the number of hits in the past 5 minutes.

       @param timestamp - The current timestamp (in seconds granularity).

\* @param {number} timestamp

\* @return {number}

\*/

HitCounter.prototype.getHits = function(timestamp) {

   var hits = 0;

   for(let i=0; i<300; i++) {

       if(this.times[i]>timestamp-300) {

           hits += this.hits[i];

       }

   }

   return hits;

};

/\*\*

\* Your HitCounter object will be instantiated and called as such:

\* var obj = Object.create(HitCounter).createNew()

\* obj.hit(timestamp)

\* var param\_2 = obj.getHits(timestamp)

\*/

140. Word Break II

var wordBreak = function(s, wordDict) {

   var hash = new Map();

   return helper(s);

   function helper(str) {

       if(hash.has(str)) {

           return hash.get(str);

       }

       var res = [];

       if(str.length==0) {

           res.push("");

           return res;

       }

       for(let word of wordDict) {

           if(str.startsWith(word)) {

               var substr = str.substring(word.length);

               var sublist = helper(substr);

               for(let list of sublist) {

                   var sentence = word;

                   sentence += list.length==0? "":" ";

                   sentence += list;

                   res.push(sentence);

               }

           }

       }

       hash.set(str, res);

       return res;

   }

};

4. Median of Two Sorted Arrays

var findMedianSortedArrays = function(nums1, nums2) {

   var len1 = nums1.length;

   var len2 = nums2.length;

   if(len1<len2) {

       return findMedianSortedArrays(nums2, nums1);

   }

   //L is the number immediately left to the cut, and R the right

   let l = 0, r = len2\*2;

   while(l<=r) { //(L + R)/2 = (A[(N-1)/2] + A[N/2])/2

       let mid2 = Math.floor(l+(r-l)/2);

       let mid1 = len1+len2-mid2;

       let l1 = (mid1==0) ? Number.MIN\_VALUE:nums1[Math.floor((mid1-1)/2)];

       let l2 = (mid2==0) ? Number.MIN\_VALUE:nums2[Math.floor((mid2-1)/2)];

       let r1 = (mid1==len1\*2) ? Number.MAX\_VALUE:nums1[Math.floor(mid1/2)];

       let r2 = (mid2==len2\*2) ? Number.MAX\_VALUE:nums2[Math.floor(mid2/2)];

       if(l1>r2) l=mid2+1;

       else if(l2>r1) r=mid2-1;

       else return (Math.max(l1,l2)+Math.min(r1,r2))/2;

   }

   return -1;

};

Grid illumination

Given an NxN grid with an array of lamp coordinates. Each

lamp provides illumination to every square on their x axis, every square on

their y axis, and every square that lies in their diagonal (think of a Queen in

chess). Given an array of query coordinates, determine whether that point is

illuminated or not. The catch is when checking a query all lamps adjacent to, or

on, that query get turned off. The ranges for the variables/arrays were about:

10^3 < N < 10^9, 10^3 < lamps < 10^9, 10^3 < queries < 10^9.

function checkIllumination(n, lamps, queries){

 var rowMap = new Map(),

     colMap = new Map(),

     leftDiagMap = new Map(),

     rightDiagMap = new Map(),

     lampSet = new Set();

 var res = [];

 for(let lamp of lamps) {

   lampSet.add(lamp);

   let row = lamp[0];

   let col = lamp[1];

   if(!rowMap.has(row)) rowMap.set(row, 1);

   else rowMap.set(row, rowMap.get(row)+1);

   if(!colMap.has(col)) colMap.set(col, 1);

   else colMap.set(col, colMap.get(col)+1);

   if(!leftDiagMap.has(row-col)) leftDiagMap.set(row-col, 1);

   else leftDiagMap.set(row-col, leftDiagMap.get(row-col)+1);

   if(!rightDiagMap.has(row+col)) rightDiagMap.set(row+col, 1);

   else rightDiagMap.set(row+col, rightDiagMap.get(row+col)+1);

 }

 // Iterate each query and update the lamp count in each map (turn off the lamps)

 for(let query of queries) {

   let row = query[0];

   let col = query[1];

   // Iterate through the query itself and its neighbours to turn off the lamps

   for(let i=row-1; i<=row+1; i++) {

     for(let j=col-1; j<=col+1; j++) {

       if(i>=1 && i<=n && j>=1 && j<=n) {

         if(lampSet.has([i,j])) {

           rowMap.set(i, rowMap.get(i)-1);

           colMap.set(j, colMap.get(j)-1);

           leftDiagMap.set(i-j,leftDiagMap.get(i-j)-1);

rightDiagMap.set(i+j,rightDiagMap.get(i+j)-1);

         }

       }

     }

   }

   // Check whether the grid cell is light or dark after turning off all affected lights

   if((rowMap.has(row) && rowMap.get(row) != 0) ||

      colMap.has(col) && colMap.get(col) != 0 ||

      leftDiagMap.has(row-col) && leftDiagMap.get(row-col) != 0 ||

      rightDiagMap.has(row+col) && rightDiagMap.get(row+col) != 0)

     res.push("LIGHT");

   else

     res.push("DARK");

   // Turn on all the lights after examine the cell

   for(let i = row-1; i <= row+1; i++){

     for(let j = col-1; j <= col+1; j++){

       if(i >=1 && i <= n && j >= 1 && j <= n){

         if(lampSet.has([i,j])){

           rowMap.set(i,rowMap.get(i)+1);

           colMap.set(j,colMap.get(j)+1);

           leftDiagMap.set(i-j,leftDiagMap.get(i-j)+1);

           rightDiagMap.set(i+j,rightDiagMap.get(i+j)+1);

         }

       }

     }

   }

 }

   return res;

 }

 var lamps = [[0,3],[1,5],[4,5]];

 var queries = [[2,3],[3,2]];

 var res = checkIllumination(6, lamps, queries);

 console.log(res);

Count number of lamps in each horizontal, vertical and diagonal and store in map. On each query I **count the lamps in adjacent 8 cells** and **check if lamps in given vertical, horizontal or diagonal are more than these in adjacent cells**. If it is, I will return true, because there is at least one lamp except these in adjacent cells which illuminates the query point.

Photo Gallery

* DOM manipulation
* Event handling
* Application state
* Data structures
* Encapsulation

<https://codepen.io/smeng/pen/wqxPYw>

<https://codepen.io/smeng/pen/YxObbN>

Business Page

* CSS aptitude
* Code organization
* Semantic HTML
* Layout mechanics

<https://codepen.io/smeng/pen/wqXYJj>

Web Logger: Coding fundamentals

* Functional programming
* Control structure
* Array manipulation

<https://codepen.io/smeng/pen/ayazLj>