<http://www.1point3acres.com/bbs/thread-191081-1-1.html>

Meeting Time Slots

**Given a list of schedules, provide a list of times that are available for a meeting**

var time = [

 [[4,5],[6,10],[12,14]],

 [[4,5],[5,9],[13,16]],

 [[11,14]]

];

var res = findAvailableTime(time);

console.log(res);

function findAvailableTime(time) {

 var clock = [];

 for(let i=0; i<24; i++){

   clock[i] = true;

 }

 for(let i=0; i<time.length; i++) { // for each calendar

   for(let t of time[i]) { // for each time slot

     let start = t[0];

     let end = t[1];

     clock.fill(false, start, end); // [start, end)

   }

 }

 var res = [];

 for(let i=0; i<24; i++) {

   if(clock[i]) {

     var newFreeTime = [];

     newFreeTime[0] = i;

     while(clock[++i]) {

       newFreeTime[1] = i;

     }

     if(i<23){

       newFreeTime[1] = i++;

     }

     res.push(newFreeTime);

   }

 }

 return res;

}

Star Widget

Given **a star widget** embedded in a form write the code to select the stars and submit the correct value through a normal form action. Make reusable for multiple star widgets.

<p>Review Stars</p>

<form id="myform" name="myform" action="post">

 <fieldset class="rating">

   <input type="radio" id="star5" class="star" name="rating" value="5"/>

   <label for="star5"></label>

   <input type="radio" id="star4" class="star" name="rating" value="4"/>

   <label for="star4"></label>

   <input type="radio" id="star3" class="star" name="rating" value="3"/>

   <label for="star3"></label>

   <input type="radio" id="star2" class="star" name="rating" value="2"/>

   <label for="star2"></label>

   <input type="radio" id="star1" class="star" name="rating" value="1"/>

   <label for="star1"></label>

 </fieldset>

</form>

fieldset, label { margin: 0; padding: 0; }

body{ margin: 20px; }

/\*\*\*\*\*\* Style Star Rating Widget \*\*\*\*\*/

.rating {

 border: none;

 float: left;

}

.rating > input { display: none; }

.rating > label:before {

 margin: 5px;

 font-size: 1.5em;

 display: inline-block;

 content: "\*";

}

.rating > .half:before {

 content: "\f089";

 position: absolute;

}

.rating > label {

 color: #ddd;

float: right;

}

/\*\*\*\*\* CSS Magic to Highlight Stars on Hover \*\*\*\*\*/

.rating > input:checked ~ label, /\* show gold star when clicked \*/

.rating:not(:checked) > label:hover, /\* hover current star \*/

.rating:not(:checked) > label:hover ~ label { color: #FFD700;  } /\* hover previous stars in list \*/

Convert a snippet of html into a working widget

<fieldset class="star-rating" data-evaluated="wdw\_1" data-stars="5" data-value="2">

 <legend>Animal Kindom Lodge:</legend>

</fieldset>

(function( $ ){

 $.fn.starRating = function() {

   var i, html, max, value, evaluated, $this;

   return this.each(function() {

     $this = $(this);

     value = $this.data('value');

     max = $this.data('stars');

     evaluated = $this.data('evaluated');

     html = "";

     for (i = max; i > 0; i--) {

html += '<input id="'+evaluated+'\_'+i+'" name="' + evaluated +'" type="radio" value="' + i + '" /><label for="'+evaluated+'\_'+i+'">'+i+' out of '+max+' stars</label>';

    }

    this.innerHTML += html;

    if (value != null) { // check the max-value th radio box

 this.getElementsByTagName('input')[max-value].setAttribute('checked', 'checked');

    }

 });

 };

 $('.star-rating').starRating();

})( jQuery );

Autocomplete

Given an input and an endpoint which returns a JSON list, as a result, extend it to **autocomplete on change**, handle key navigation through the results

<h1>Autocomplete with JSON</h1>

<input id="myinput"/>

<div class="suggestion">

 <ul></ul>

</div>

.chosen {

 background-color: lightgray;

}

var suggestions = '{"0":"test", "1":"test1", "2":"test2"}';

var res=[], lastSearch, currentIdx=0;

function drawList() {

 var ulist = $('.suggestion ul');

 if(ulist) {

   ulist.remove();

 }

 res = [];

 ulist = $("<ul></ul>");

 var mysuggestions = $.parseJSON(suggestions);

 for(var i in mysuggestions) {

   var list = $("<li>"+mysuggestions[i]+"</li>");

   ulist.append(list);

   res.push(mysuggestions[i]);

 }

 $('.suggestion').append(ulist);

}

function changeSearch() {

 var currentSearch = $('ul li:nth-child('+currentIdx+')');

 $('li').removeClass('chosen');

 currentSearch.addClass('chosen');

 lastSearch = currentSearch.text();

 $('#myinput').val(lastSearch);

}

$('#myinput').on('keyup', function(e) {

 // console.log('onchage');

 var searchValue = e.target.value;

 switch(e.key) {

   case "ArrowUp" :

     currentIdx == 1 ? currentIdx = res.length : currentIdx--;

     changeSearch();

     break;

   case "ArrowDown":

     currentIdx == res.length ? currentIdx = 1 : currentIdx++;

     changeSearch();

     break;

   default:

     if(searchValue == lastSearch) break;

     lastSearch = searchValue;

     currentIdx = 0;

     drawList();

     break;

 }

});

Class create

Given: **var thing = new Thing();**

How would you implement the following functionality:

thing.set('x', val);

thing.set('age', val2);

thing.unset(‘x’);

thing.get(‘age’);

thing.on(‘change’, fn(x1,x2,x3) {});

thing.on(‘change:age’, fn1(){});

thing.on(‘change:age’, fn2(){});

console.log(thing.get('x')); // val

console.log(thing.get('y')); // val2

function Thing() {

 let list = {};

 let function\_list = [];

 let keyfunction\_list = {}; // key: fn[]

 this.set = function(key,val) {

   list[key] = val;

   for(var fn of function\_list) {

     fn();

   }

   if(keyfunction\_list[key]!==undefined) {

      for(var fn of keyfunction\_list[key]) {

        fn();

      }

   }

 }

 this.get = function(key) {

   return list[key];

 }

this.on = function(action, fn) {

   switch(action) {

     case ‘change’:

        function\_list.push(fn);

        break;

     default:

        var key = action.split(‘:’)[1];

        if(keyfunction\_list[key]===undefined) {

          keyfunction\_list[key]=[];

        }

        keyfunction\_list[key].push(fn);

        break;

   }

 }

}

var thing = new Thing();

thing.set('mom','cate');

thing.set('dad','bill');

var v1 = thing.get('mom');

var v2 = thing.get('dad');

console.log(v1);

console.log(v2);

Palindrome Pair

Check if word **is Palindrome**

var word = "adbsda";

function isPalindrome(word) {

 return word.split("").reverse().join("") == word;

}

console.log(isPalindrome(word));

**Palindrome pairs**, best solution for them

var list1 = ["abcb", "bcba", "sdf", "fds", "asd"]; // only consider reverse

var res = findPair(list1);

console.log(res);

// pair is palindrome nomatter which one comes first

function findPair(arr) {

 var res = [];

 var hash = {};

 for(let i=0; i<arr.length; i++) {

   hash[arr[i]]=true;

 }

 for(let i=0; i<arr.length; i++) {

   var rev = arr[i].split('').reverse().join('');

   if(hash[rev]!==undefined) {

     if(res.indexOf(rev)==-1) {

       res.push(rev,arr[i]);

     }

   }

 }

 return res;

}

var list2 = ["abcb", "bcba", "a", "sdf", "fds", "asd"]; // consider itself

var res2 = findPair2(list2);

console.log(res2);

// consider two string of a pair have different length

function findPair2(arr) {

 var res = [];

 var hash = {};

 for(let i=0; i<arr.length; i++) {

   hash[arr[i]]=true;

 }

 for(let i=0; i<arr.length; i++) {

   var rev\_list = findPosibleRes(arr[i]);

   for(var rev of rev\_list) {

     if(hash[rev]!==undefined) {

       if(res.indexOf(rev)==-1) {

         res.push(rev);

       }

       if(res.indexOf(arr[i])==-1) {

         res.push(arr[i]);

       }

     }

   }

 }

 return res;

}

function findPosibleRes(str) {

 var res = [];

 var reverse\_even = str.split('').reverse().join('');

 var reverse\_odd = reverse\_even.slice(1);

 res.push(reverse\_even,reverse\_odd);

 for(let i=str.length-1; i>=parseInt(str.length/2); i--) {

   var curr = str.slice(0,i);

   var reverse = curr.split('').reverse().join('');

   var newstring\_even = curr+reverse;

   var newstring\_odd = curr+reverse.slice(1);

   var add\_even = newstring\_even.slice(str.length);

   var add\_odd = newstring\_odd.slice(str.length);

   if(str+add\_even==newstring\_even) {

     if(res.indexOf(add\_even)==-1) {

       res.push(add\_even);

     }

   }

   if(str+add\_odd==newstring\_odd) {

     if(res.indexOf(add\_odd)==-1) {

       res.push(add\_odd);

     }

   }

 }

 return res;

}

Alien Alphabet

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of words from the dictionary, where words are sorted lexicographically by the rules of this new language. Derive the order of letters in this language.

For example,

Given the following words in dictionary,

[  
 "wrt",  
 "wrf",  
 "er",  
 "ett",  
 "rftt"  
]

The correct order is: "wertf".

words = [

 "wrt","wrf", "er","ett", "rftt"

];

var res = alienOrder(words);

console.log(res);

function alienOrder(words) {

 var hash = {}; // character : set{character}

 var degree = {}; // character : degree

 var res = "";

 if(words==null || words.length==0) return res;

 for(var word of words) {

   for(var c of word) {

     degree[c] = 0;

   }

 }

 for(let i=0; i<words.length-1; i++) {

   var cur = words[i];

   var next = words[i+1];

   var len = Math.min(cur.length, next.length);

   for(let j=0; j<len; j++) {

     var c1 = cur[j];

     var c2 = next[j];

     if(c1!=c2) {

       var set = {};

       if(hash[c1]!==undefined) {

         set = hash[c1];

       }

       if(set[c2]==undefined) {

         set[c2] = true;

         hash[c1] = set;

         degree[c2]++;

       }

       break;

     }

   }

 }

 var queue = [];

 for(var key in degree) {

   if(degree[key]==0) {

     queue.push(key);

   }

 }

 while(queue.length!==0) {

   var c = queue.shift();

   res += c;

   if(hash[c]!==undefined) {

     console.log(hash[c]);

     for(var c3 in hash[c]) {

       degree[c3]--;

       if(degree[c3]==0) {

         queue.push(c3);

       }

     }

   }

 }

 if(res.length!==Object.keys(degree).length) return "";

 return res;

}

Max Sum Booking

Provide a set of positive integers (an array of integers). Each integer represent number of nights userrequest on [Airbnb.com](http://airbnb.com/). If you are a host, you need to design and implement an algorithm to find out the maximum number a nights you can accommodate. The constraint is that you have to reserve at least one day between each request, so that you have time to clean the room.

Example:

1) Input: [1, 2, 3]===&gt; output: 4, because you will pick 1 and 3

2) input: [5,1, 2, 6] ===&gt; output: 11, because you will pick 5 and 6

3) input: [5,1, 2, 6, 20, 2] ===&gt; output: 27, because you will pick 5, 2, 20

function findMaxSum(bookings) {

 var dp = [];

 dp[0] = bookings[0];

 dp[1] = Math.max(dp[0], bookings[1]);

 for(let i=2; i<bookings.length; i++) {

   dp[i] = Math.max(dp[i-1], dp[i-2]+bookings[i]);

 }

 return dp[bookings.length-1];

}

Pagelist

第一轮实现分页显示。给了以下一些输入数据，要求将以下行分页显示，每页12行，其中每行已经按score排好序，分页显示的时候如果有相同host id的行，则将后面同host id的行移到下一页

|  |
| --- |
| [ |
| "host\_id,listing\_id,score,city", |
| "1,28,300.1,SanFrancisco", |
| "4,5,209.1,SanFrancisco", |
| "20,7,208.1,SanFrancisco", |
| "23,8,207.1,SanFrancisco", |
| "16,10,206.1,Oakland", |
| "1,16,205.1,SanFrancisco", |
| "6,29,204.1,SanFrancisco", |
| "7,20,203.1,SanFrancisco", |
| "8,21,202.1,SanFrancisco", |
| "2,18,201.1,SanFrancisco", |
| "2,30,200.1,SanFrancisco", |
| "15,27,109.1,Oakland", |
| "10,13,108.1,Oakland", |
| "11,26,107.1,Oakland", |
| "12,9,106.1,Oakland", |
| "13,1,105.1,Oakland", |
| "22,17,104.1,Oakland", |
| "1,2,103.1,Oakland", |
| "28,24,102.1,Oakland", |
| "18,14,11.1,SanJose", |
| "6,25,10.1,Oakland", |
| "19,15,9.1,SanJose", |
| "3,19,8.1,SanJose", |
| "3,11,7.1,Oakland", |
| "27,12,6.1,Oakland", |
| "1,3,5.1,Oakland", |
| "25,4,4.1,SanJose", |
| "5,6,3.1,SanJose", |
| "29,22,2.1,SanJose", |
| "30,23,1.1,SanJose" |
| ] |

在遍历的时候需要维护一个LinkedHashMap作为page并且完成去重。用LinkedHashMap的好处是可以保证所有的entry是按插入的顺序排序的，所以仍然可以保证按score排序的性质。另外，一旦遇到相同的host\_id，则将其对应的行存到另一个buffer里。由于需要变遍历边增减容器里的数据，需要用ListIterator，并调用remove和add方法。之前只用过remove，从来没用过add。

var hash = {}; //{host\_id:[idx,idx,...]}

var hash\_nb = 0;

//create hash table

for(var i=1; i<pagelist.length; i++) {

 var curr\_id = pagelist[i].split(',')[0];

 if(hash[curr\_id]===undefined) {

   hash[curr\_id] = [];

 }

 hash[curr\_id].push(i);

 hash\_nb++;

}

// console.log(hash);

var page = Math.ceil(hash\_nb/12);

var page\_nb = 0;

var idPerPage = {};

for(var curr\_id in hash) {

 idPerPage[curr\_id] = [];

 // console.log(curr\_id);

 if(hash[curr\_id].length%page!==0) {

     var rest = hash[curr\_id].length%page;

     console.log(rest);

     for(let i=0; i<page; i++) { // same id for curr\_id per page

       idPerPage[curr\_id][i] = Math.floor(hash[curr\_id].length/page);

       if(i>=page-rest) {

         idPerPage[curr\_id][i]++;

       }

     }

  }else {

     for(let i=0; i<page; i++) { // same id for curr\_id per page

       idPerPage[curr\_id][i] = Math.floor(hash[curr\_id].length/page);

     }

  }

}

var printlist = []; //[page\_buffer, page\_buffer, page\_buffer...]

var reslist = []; // [[record, record,...],[],...]

for(var i=1; i<pagelist.length; i++) {

 var curr\_id = pagelist[i].split(',')[0];

 if(printlist[page\_nb]===undefined) {

   printlist[page\_nb] = {};

   printlist[page\_nb]['record\_len'] = 0;

   reslist[page\_nb] = [];

 }

 var page\_buffer = printlist[page\_nb]; // {record\_len:0, host\_id:[idx,idx,...],host\_id:[idx,idx,...],...}

 if(page\_buffer[curr\_id]==undefined) {

   page\_buffer[curr\_id] = [];

   page\_buffer[curr\_id].push(i);

   page\_buffer['record\_len']++;

   reslist[page\_nb].push(pagelist[i]);

 }

 else { //page has the id print, check how many should be there, if less than then print

   var min = idPerPage[curr\_id][page\_nb];

   var num = page\_buffer[curr\_id].length;

   // every page is not full 12 record

   if(num < min) { // current same id # < limitation - > number of the same id shown in the same page decreasing order

     page\_buffer[curr\_id].push(i);

     page\_buffer['record\_len']++;

     reslist[page\_nb].push(pagelist[i]);

   }

   else { // check the next page

     var next\_page\_nb = page\_nb + 1;

     while(next\_page\_nb<page) {

       if(printlist[next\_page\_nb]===undefined) {

         printlist[next\_page\_nb] = {};

         printlist[next\_page\_nb]['record\_len'] = 0;

         reslist[next\_page\_nb] = [];

       }

       var next\_page\_buffer = printlist[next\_page\_nb];

       if(next\_page\_buffer['record\_len']<12) {

         if(next\_page\_buffer[curr\_id]==undefined) {

           next\_page\_buffer[curr\_id] = [];

           next\_page\_buffer[curr\_id].push(i);

           next\_page\_buffer['record\_len']++;

           reslist[next\_page\_nb].push(pagelist[i]);

           break;

         }else {

           var min = idPerPage[curr\_id][next\_page\_nb]; // same id average per page

           var num = next\_page\_buffer[curr\_id].length;

           if(num < min) { // current same id # < limitation

             next\_page\_buffer[curr\_id].push(i);

             next\_page\_buffer['record\_len']++;

             reslist[next\_page\_nb].push(pagelist[i]);

             break;

           }

         }

         next\_page\_nb++;

       }

     }

   }

 }

 if(page\_buffer['record\_len']==12) {

   page\_nb++;

 }

}

console.log(printlist);

console.log(reslist);

Sorting CSV

given **csv file**, sort it with first column, remove duplicate column value rows.

var csv = [{1:2, 2:"fun", 3:"love"},{1:3, 2:"cute", 3:"you"},{1:4, 2:"fun", 3:"me"},{1:1, 2:"love", 3:"dad"}];

function compare(a, b) {

 return a['1']-b['1'];

}

function filter(arr, key) {

 var res = [];

 var unique = {};

 for(var i=0; i<arr.length; i++) {

   var curr = arr[i];

   if(unique[curr[key]]===undefined) {

     unique[curr[key]]=true;

     res.push(curr);

   }

 }

 return res;

}

csv.sort(compare);

var keys = Object.getOwnPropertyNames(csv[0]);

console.log(keys);

for(var key of keys) {

 csv = filter(csv, key);

}

console.log(csv);

Price Sum

Given a **menu** (list of items prices), **find all possible combinations of items that sum a particular value K**. (A variation of the typical 2sum/Nsum questions).

// Integers

var menu = [12, 5, 2, 1, 4, 2, 3];

var target = 6;

console.log(findSum(menu,target));

//version 1 : hash save all candidates and the posible sums

function findSum(menu, target) {

 var hash = {}; //{price sum: [[idx1],[idx2, idx3],[idx5]]}

 menu.sort(function(a, b) {

   return a-b;

 });

 if(menu[0]>target) return [];

 for(let i = 0; i<menu.length; i++) {

   if(menu[i]>target) {

     break;

   }

   // if we can't order same thing put it here, then put it after hash[menu[i]]

   for(var key in hash) {

     if (hash.hasOwnProperty(key)) {

       var sum = parseInt(key)+menu[i]; // add current chosen price to every already chosen ones

       if(sum<=target) {

         for(let arr of hash[key]) { // loop through the price sum combination array

           var choice = arr.concat(i);

           if(hash[sum]===undefined) {

             hash[sum] = [];

           }

           hash[sum].push(choice);

         }

       }

     }

   }

   if(hash[menu[i]]==undefined) {

     hash[menu[i]] = [];

   }

   var choice\_idx = [];

   choice\_idx.push(i);

   hash[menu[i]].push(choice\_idx);

 }

 var res = [];

 for(var arr of hash[target.toString()]) {

   var choice = [];

   for(let i=0; i<arr.length; i++) {

       choice.push(menu[arr[i]]);

   }

   res.push(choice);

 }

 return res;

}

//varsion 2 : recurse

var combinationSum = function(candidates, target) {

   candidates.sort(function(a,b) {

       return a-b;

   });

   var res = [];

   if(candidates[0]>target) {

       return res;

   }

   var choice = [];

   helper(res, choice, candidates, target, 0);

   // console.log(res);

   return res;

   function helper(res, choice, candidates, target, start) {

       if(target>0) {

           for(var i=start; i<candidates.length && target>=candidates[i]; i++) {

               choice.push(candidates[i]);

               helper(res, choice, candidates, target-candidates[i], i);

               choice.pop();

           }

       }

       if(target == 0) {

         var copy = choice.concat();

           res.push(copy);

           this.res = res;

       }

   }

};

var candidates = [2,3,6,7];

var target = 7;

var res = combinationSum(candidates, target);

console.log(res);

// Precision

**toFixed(n) provides n length after the decimal point; toPrecision(x) provides x total length.**

var combinationSum = function(candidates, target) {

   candidates.sort(function(a,b) {

       return a-b;

   });

   var res = [];

   if(candidates[0]>target) {

       return res;

   }

   var choice = [];

   helper(res, choice, candidates, target, 0);

   // console.log(res);

   return res;

   function helper(res, choice, candidates, target, start) {

       if(target>0.1) {

           for(var i=start; i<candidates.length && target>=candidates[i]; i++) {

               choice.push(candidates[i]);

               var diff = (target-candidates[i]).toFixed(3);

               helper(res, choice, candidates, diff, i+1); // the last parameter is i, then we can choose duplicate item

               choice.pop();

           }

       }

       if(target<0.1) {

           var copy = choice.concat();

           res.push(copy);

           this.res = res;

       }

   }

};

var candidates = [2.2, 1.1, 3.2, 2.1];

var target = 4.3;

var res = combinationSum(candidates, target);

console.log(res);

2D Array Iteractor

给一个2d array，要求写⼀个顺序访问这个2d array的Iterator，包括hasNext()与

next()。注意2d array的每⾏行中元素的个数可能不⼀一样，也可能为空。followup是写⼀一个

remove()，注意是remove当前item，不是下⼀一个item。remove是需要同时删除原来数组⾥里的元素，也能在Iterator调⽤用时体现出来。

var a = ['w', 'y', 'k', 'o', 'p'];

var array = [['w', 'y'],[],['k', 'o', 'p'],['e', 'f', 'g']];

// var iterator = a.entries(); // get array iterator

var it = makeIterator(array);

console.log(it.next());

console.log(it.next());

console.log(it.next());

console.log(it.next());

console.log(it.next());

console.log(it.next());

console.log(it.next());

it.remove();

console.log(array);

console.log(it.hasNext());

function makeIterator(array) {

   var rowIndex = 0;

   var colIndex = 0;

   return {

        next: function() {

            var res = null;

            while(rowIndex < array.length) {

               let arr = array[rowIndex];

               if(arr.length == 0) { // check if the inner array is null

                  rowIndex++;

                  break;

               }

               if(colIndex < arr.length) {

                 res = arr[colIndex++];

                 break;

               }

               if(colIndex === arr.length) { // get to the last item in current array sub

                 rowIndex++;

                 colIndex = 0;

               }

            }

            return res;

        },

        hasNext: function() {

            return  rowIndex < array.length && colIndex < array[rowIndex].length;

        },

        remove: function() { // remove current item

            if(colIndex == 0) { // current item is the last one in the current sub array

               let currentIndex = rowIndex-1;

               let arr = array[currentIndex];

               arr.splice(arr.length-1, 1);

            }

            else {

               let arr = array[rowIndex];

               arr.splice(--colIndex, 1);

            }

        }

   };

}

Text Justification

Given an array of words and a length L, format the text such that each line has exactly L characters and is fully (left and right) justified.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

如果除最后一行某一行只有一个单词怎么办？左对齐

words: ["This", "is", "an", "example", "of", "text", "justification."]

L: 16.

[  
  "This      is      an",  
  "example  of text",  
  "justification.      "  
]

var fullJustify = function(words, maxWidth) {

   var res = [];

   var index = 0;

   while(index<words.length) {

     var count = words[index].length;

     var last = index+1;

     while(last<words.length) {

       if(words[last].length+count+1>maxWidth) {

         break;

       }

       count = words[last].length+count+1;

       last++;

     }

     var line = "";

     var diff = last-1-index;

     if(last == words.length || diff == 0) { // if last line or one word then left justified

        for(let i=index; i<last; i++) {

          line += words[i] + " ";

        }

        line = line.slice(0, -1);

        for(let i=line.length; i<maxWidth; i++) {

          line += " ";

        }

     }

     else { // middle justified

        var spaces = parseInt((maxWidth-count) / diff);

        var rest = (maxWidth-count) % diff;

        for(let i=index; i<last; i++) {

          line += words[i];

          if(i<last-1) {

            for (let j = 0; j <= (spaces + ((i - index) < rest ? 1 : 0)); j++) {

              line += " ";

            }

          }

        }

     }

     res.push(line);

     index = last;

   }

   return res;

};

var words = ["This", "is", "an", "example", "of", "text", "justification."];

var maxWidth = 16;

var res = fullJustify(words, maxWidth);

console.log(res);

water land

比如terrian是[3,2,1,2] print出来就是

\*. 鐣欏鐢宠璁哄潧-涓€浜╀笁鍒嗗湴

\* \*    \*

\* \* \* \*

\* \* \* \*

然后给你一个dumpPoint，一个waterAmount，比如dumpPoint 1, waterAmount 2，因为有重力，所以是从index 2开始加水.鐣欏璁哄潧-涓€浜�-涓夊垎鍦�

\* w-涓€浜╀笁鍒嗗湴

\* \* w\*

\* \* \* \*

\* \* \* \*

var water = [3,2,1,2,4,3];

var printWater = function(array) {

 var max = Math.max(...array);

 var res = [];

 for(let row=0; row<=max; row++) {

   var stars = "";

   for(let col=0; col<array.length; col++) {

     if(array[col] >= max-row || row==max) {

       stars = stars.concat('\*');

     }

     else {

       stars = stars.concat(' ');

     }

   }

   res.push(stars);

 }

 console.log(res);

}

printWater(water);

一个二维vector named "people" :

{6,7}//people 0 knows people 6 7

{4}//people 1 knows people 4

...

...

一共10个数组。然后output出people 0到people 9的一个path，并且每一步的平方值加起来的cost 最小。比如0-1-2-3-4-5-6-7-8-9 比0-4-9的cost要少，(4-0)^2+(9-4)^2明显大。