Java Collection

Monday, February 27, 2017 7:29 PM

ArrayList : add, contains, get, size()

HashMap: containsnKey, containsValue, put, get, size()

Queue: add, poll ,peek

Stack: push, pop, peek

Set: Set<Integer> s = new HashSet<Integer>(), add, contains, remove, size()

301. Remove Invalid Parentheses

Saturday, February 18, 2017 4:56 PM

Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

Note: The input string may contain letters other than the parentheses (and).

Examples:

```
"()())()" -> ["()()()", "(())()"]
"(a)())()" -> ["(a)()()", "(a())()"]
")(" -> [""]
```

变种，只需要返回一个的话

```
class Solution {
public:
    string removeInvalidParentheses(string s) {
        string result = delete_close(s);
        return delete_open(result);
    }
    string delete_close(string s){
        if(!s.length()) return s;
        string result = "";
        int count = 0;
        for(auto ch: s){
            if(ch == '('){
                ++count;
                result += s[i];
            }
            else if(ch == ')'){
                if(count > 0){
                    -- count;
                    result += ch;
                }
            }
        }
        return result;
    }
    string delete_open(string s){
        if(!s.length()) return s;
        string result = "";
        int count = 0;
        for(int i = s.length() - 1; i >= 0; --i){
            if(s[i] == ')'){
                ++count;
                result += s[i];
            }
            else if(s[i] == '('){
                if(count > 0){
                    -- count;
                    result += s[i];
                }
            }
```

```
                    result += s[i];
                }
            }
        }
        return result;
    }
};
```

Find same contacts in a list of contacts

Saturday, February 18, 2017 7:23 PM

Find same contacts in a list of contacts

Given a list of contacts containing username, email and phone number in any order. Identify the same contacts (i.e., same person having many different contacts) and output the same contacts together.

Notes:

1) A contact can store its three fields in any order, i.e., phone number can appear before username or username can appear before phone number.

2) Two contacts are same if they have either same username or email or phone number.

Example:

```
Input: contact[] =
     { {"Gaurav", "gaurav@gmail.com", "gaurav@gfgQA.com"},
       { "Lucky", "lucky@gmail.com", "+1234567"},
       { "gaurav123", "+5412312", "gaurav123@skype.com"}.
       { "gaurav1993", "+5412312", "gaurav@gfgQA.com"}
     }
Output:
   0 2 3
   1
contact[2] is same as contact[3] because they both have same
contact number.
contact[0] is same as contact[3] because they both have same
e-mail address.
Therefore, contact[0] and contact[2] are also same.
```

Java:

```
//Find same contacts in a list of contacts

class Contact{
    ArrayList<String> field;
}
//Check whether two Contact contains same emal address
public boolean isSameContact(Contact c1, Contact c2){
    int l1 = c1.field.size();
    int l2 = c2.field.size();
    for(int i=0; i<l1; i++){
        for(int j=0; j<l2; j++){
            if(c1.field.get(l1).equals(c2.field.get(l2))){
                return true;
            }
        }
    }
    return false;
}
//fill entries in adjancency matrix represent graph
public void buildGraph(Contact[] arr, int[][] mat){
    int n = arr.length;
    for(int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            if(isSameContact(arr[i],arr[j])){
                mat[i][j] = 1;
                mat[j][i] = 1;
                break;
            }
        }
    }
}
```

```
public void DFSvisit(int i, int[][] mat, boolean[] visited, ArrayList<Integer> sol, int n){
    visited[i] = true;
    sol.add(i);

    for(int j=0; j<n; j++){
        if(mat[i][j] && !visited[j]){
            DFSvisit(j, mat, visited, sol, n);
        }
    }
}

public void findResult(Contact[] arr){
    //contact length
    int n = arr.length;
    //Adjacent matrix
    int mat = new int[n][n];
    //Save solution
    ArrayList<Integer> sol = new ArrayList<>();
    //Keep track whether visted already
    boolean[] visited = new boolean[n];

    buildGraph(arr, mat);

    for(int i=0; i<n; i++){
        if(!visited[i]){
            DFSvisit(i, mat, visited, sol, n);
            sol.add(-1);
        }
    }

    return sol;

    /*
    To print the result
    for(int i=0; i<n; i++){
        if(sol[i] == -1) cout<<endl;
        else cout<<sol[i]<<" ";
    }
    */

}
```

C++:

Below is C++ implementation of this idea.

```
// A C++ program to find same contacts in a list of contacts
#include<bits/stdc++.h>
using namespace std;

// Structure for storing contact details.
struct contact
{
    string field1, field2, field3;
};

// A utility function to fill entries in adjacency matrix
// representation of graph
void buildGraph(contact arr[], int n, int *mat[])
{
    // Initialize the adjacency matrix
    for (int i=0; i<n; i++)
       for (int j=0; j<n; j++)
           mat[i][j] = 0;

    // Traverse through all contacts
    for (int i = 0; i < n; i++) {

        // Add mat from i to j and vice versa, if possible.
        // Since length of each contact field is at max some
        // constant. (say 30) so body execution of this for
        // loop takes constant time.
        for (int j = i+1; j < n; j++)
            if (arr[i].field1 == arr[j].field1 ||
                arr[i].field1 == arr[j].field2 ||
                arr[i].field1 == arr[j].field3 ||
                arr[i].field2 == arr[j].field1 ||
                arr[i].field2 == arr[j].field2 ||
                arr[i].field2 == arr[j].field3 ||
                arr[i].field3 == arr[j].field1 ||
                arr[i].field3 == arr[j].field2 ||
                arr[i].field3 == arr[j].field3)
            {
                mat[i][j] = 1;
                mat[j][i] = 1;
                break;
            }
    }
}
```

```
// A recuesive function to perform DFS with vertex i as source
void DFSvisit(int i, int *mat[], bool visited[], vector<int>& sol, int n)
{
    visited[i] = true;
    sol.push_back(i);

    for (int j = 0; j < n; j++)
        if (mat[i][j] && !visited[j])
            DFSvisit(j, mat, visited, sol, n);
}

// Finds similar contacrs in an array of contacts
void findSameContacts(contact arr[], int n)
{
    // vector for storing the solution
    vector<int> sol;

    // Declare 2D adjaceny matrix for mats
    int **mat = new int*[n];

    for (int i = 0; i < n; i++)
        mat[i] = new int[n];

    // visited array to keep track of visited nodes
    bool visited[n];
    memset(visited, 0, sizeof(visited));

    // Fill adjacency matrix
    buildGraph(arr, n, mat);

    // Since, we made a graph with contacts as nodes with fields as links.
    // two nodes are linked if they represent the same person.
    // so, total number of connected components and nodes in each component
    // will be our answer.
    for (int i = 0; i < n; i++)
    {
        if (!visited[i])
        {
            DFSvisit(i, mat, visited, sol, n);

            // Add delimeter to separate nodes of one component from other.
            sol.push_back(-1);
        }
    }

    // Print the solution
    for (int i = 0; i < sol.size(); i++)
        if (sol[i] == -1) cout << endl;
        else cout << sol[i] << " ";
}

// Drive program
int main()
{
    contact arr[] = {{"Gaurav", "gaurav@gmail.com", "gaurav@gfgQA.com"},
                     {"Lucky", "lucky@gmail.com", "+1234567"},
                     {"gaurav123", "+5412312", "gaurav123@skype.com"},
                     {"gaurav1993", "+5412312", "gaurav@gfgQA.com"},
                     {"raja", "+2231210", "raja@gfg.com"},
                     {"bahubali", "+878312", "raja"}
                    };

    int n = sizeof arr / sizeof arr[0];
    findSameContacts(arr, n);
    return 0;
}
```

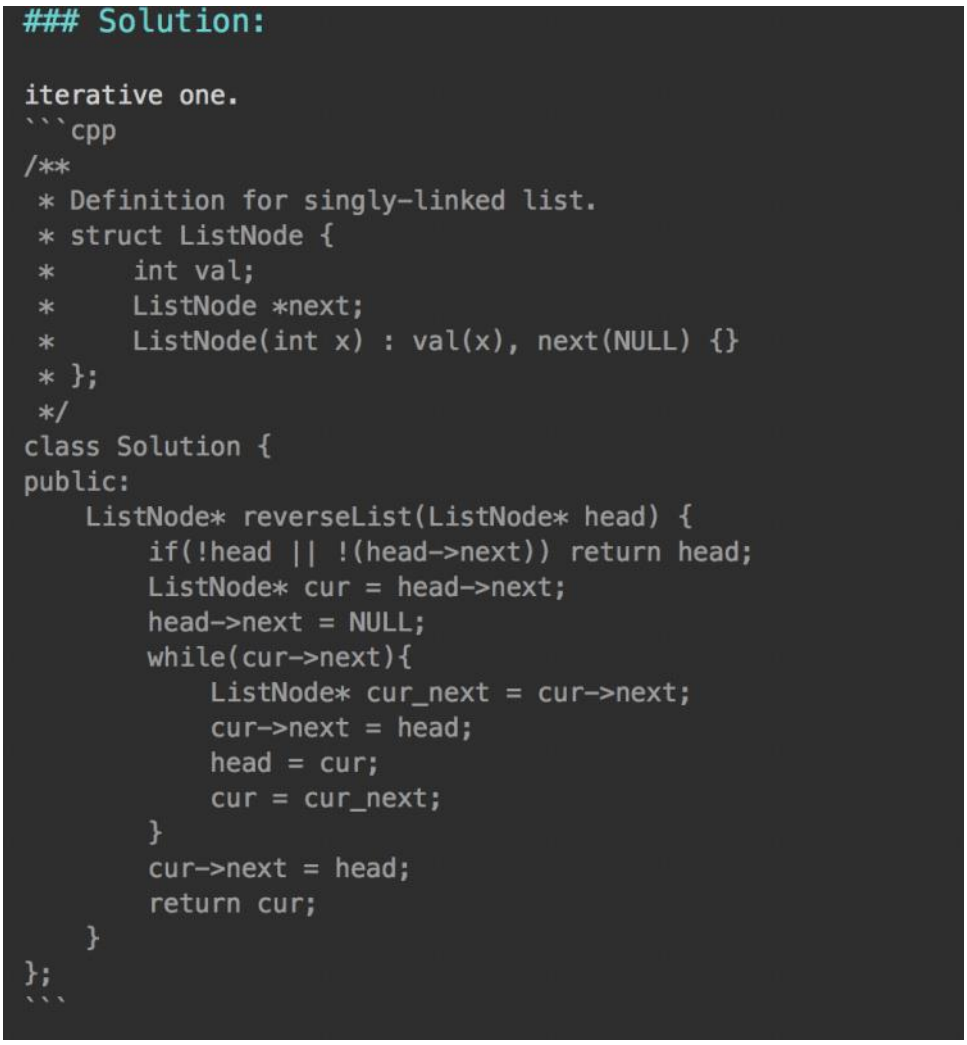
Print LinkedList in reverse order

Saturday, February 18, 2017 8:09 PM

递归

```
void printReverse(Node node) {
    if(node.next != null) { // we recurse every time unless we're on the last one
        printReverse(node.next);  // this says "do this to the next node first"
    }
    System.out.println(node.data); // we'll print out our node now
}
```

迭代



```
### Solution:

iterative one.
```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if(!head || !(head->next)) return head;
        ListNode* cur = head->next;
        head->next = NULL;
        while(cur->next){
            ListNode* cur_next = cur->next;
            cur->next = head;
            head = cur;
            cur = cur_next;
        }
        cur->next = head;
        return cur;
    }
};
```
```

```
//Print single linked list in reverse order
if(head == null || head.next == null) return head;

ListNode cur = head.next;
head.next = null;

while(cur.next != null){
    ListNode curNext = cur.next;
    cur.next = head;
    head = cur;
    cur = curNext;
}

cur.next = head;
return cur;
```

215. Kth Largest Element in an Array

Sunday, February 19, 2017 1:49 AM

Find the **k**th largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

For example,

Given `[3,2,1,5,6,4]` and k = 2, return 5.

Note:

You may assume k is always valid, 1 ≤ k ≤ array's length.

K selection

```java
public class Solution {
    public int findKthLargest(int[] nums, int k) {

        //initialize the process
        int first = 0;
        int last = nums.length - 1;

        boolean process = true;
        int result = -1;

        //find
        while(process){
            int pivot_index = partition(nums,first,last);
            if(pivot_index == k-1) {result = nums[pivot_index]; process = false;}
            else if(pivot_index < k-1) first = pivot_index + 1;
            else last = pivot_index - 1;
        }

        return result;
    }

    //partition process
    public int partition(int[] nums, int first, int end){
        int start = first+1;
        int pivot = nums[first];
        while(start<=end){
            if(nums[start]>pivot) ++start;
            else if(nums[end]<pivot) --end;
            else swap(nums, start++, end--);
        }
        //change the pivot to the middle;
        swap(nums, first, end);

        //return the index of the pivot
        return end;
    }

    public void swap(int[] nums, int a, int b){
        int c = nums[a];
        nums[a] = nums[b];
        nums[b] = c;
```

Heapsort

Well, this problem still has a tag "heap". If you are familiar with heapsort, you can solve this problem using the following idea:

1. Build a max-heap for `nums`, set `heap_size` to be `nums.size()`;
2. Swap `nums[0]` (after buding the max-heap, it will be the largest element) with `nums[heap_size - 1]` (currently the last element). Then decrease `heap_size` by 1 and max-heapify `nums` (recovering its max-heap property) at index `0`;
3. Repeat 2 for `k` times and the `k`-th largest element will be stored finally at `nums[heap_size]`.

Now I paste my code below. If you find it tricky, I suggest you to read the Heapsort chapter of Introduction to Algorithms, which has a nice explanation of the algorithm. My code simply translates the pseudo code in that book :-)

```
public:
    inline int left(int idx) {
        return (idx << 1) + 1;
    }
    inline int right(int idx) {
        return (idx << 1) + 2;
    }
    void max_heapify(vector<int>& nums, int idx) {
        int largest = idx;
        int l = left(idx), r = right(idx);
        if (l < heap_size && nums[l] > nums[largest]) largest = l;
        if (r < heap_size && nums[r] > nums[largest]) largest = r;
        if (largest != idx) {
            swap(nums[idx], nums[largest]);
            max_heapify(nums, largest);
        }
    }
    void build_max_heap(vector<int>& nums) {
        heap_size = nums.size();
        for (int i = (heap_size >> 1) - 1; i >= 0; i--)
            max_heapify(nums, i);
    }
    int findKthLargest(vector<int>& nums, int k) {
        build_max_heap(nums);
        for (int i = 0; i < k; i++) {
            swap(nums[0], nums[heap_size - 1]);
            heap_size--;
            max_heapify(nums, 0);
        }
        return nums[heap_size];
    }
private:
    int heap_size;
}
```

300. Longest Increasing Subsequence

Sunday, February 19, 2017 3:01 PM

Given an unsorted array of integers, find the length of longest increasing subsequence.

For example,

Given `[10, 9, 2, 5, 3, 7, 101, 18]`,

The longest increasing subsequence is `[2, 3, 7, 101]`, therefore the length is `4`. Note that there may be more than one LIS combination, it is only necessary for you to return the length.

Your algorithm should run in O(n^2) complexity.

Follow up: Could you improve it to O(n log n) time complexity?

Credits:

Special thanks to @pbrother for adding this problem and creating all test cases.

```java
public class Solution {
    public int lengthOfLIS(int[] nums) {
        //Here we need 1 more extra space
        int[] dp = new int[nums.length];
        //The binary seacrh should start from index(0,0)
        int len = 0;

        for(int item : nums){
            //It is binarySearch not BS
            int index = Arrays.binarySearch(dp, 0, len, item);
            if(index < 0) index = -(index+1);
            dp[index] = item;
            //When it is at end, we increase len
            if(index == len) len++;
        }
        //We final return len not the real size of array nums,
        //As it will always be fixed length with 0 in unreached sapce
        return len;
    }
}
```

```
<terminated> leetTest [Java Ap
10 0 0 0 0 0 0 0
9 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0
2 5 0 0 0 0 0 0
2 3 0 0 0 0 0 0
2 3 7 0 0 0 0 0
2 3 7 101 0 0 0 0
2 3 7 18 0 0 0 0
```

9. Palindrome Number

Sunday, February 19, 2017 3:41 PM

Determine whether an integer is a palindrome. Do this without extra space.

Java

```java
public class Solution {
    public boolean isPalindrome(int x) {
        if(x<0||(x!=0&&x%10 == 0)) return false;
        int rev=0;
        while(rev<x){
            rev = rev*10 + x%10;
            x = x/10;
        }
        return (rev == x || x == rev/10);
    }
}
```

Longest palindrome substring

Tuesday, February 28, 2017 10:39 PM

substring说明结果必须是连续的，不能断开

```java
//Longest Palindrome Substring
public int longestPalindromSubstring(String str){
    int maxLength = 1;

    int start = 0;
    int length = str.length();

    int low,high;

    // One by one consider every character as center point of
    // even and length palindromes
    for(int i=1; i<len; i++){
        low = i-1;
        high = i;
        // Find the longest even length palindrome with center points
        // as i-1 and i.
        while(low >=0 && high < length && str.charAt(low) == str.charAt(high)){
            if(high-low+1>,maxLength){
                start = low;
                maxLength = high-low+1;
            }
            --low;
            ++high;
        }
        // Find the longest odd length palindrome with center
        // point as i
        low = i-1;
        high = i+1;
        while(low >= 0 && high < length && && str.charAt(low) == str.charAt(high)){
            if(high-low+1 >maxLength){
                start = low;
                maxLength = high-low+1;
            }
            --low;
            ++high;
        }
    }

}
```

时间复杂度O（n^2）

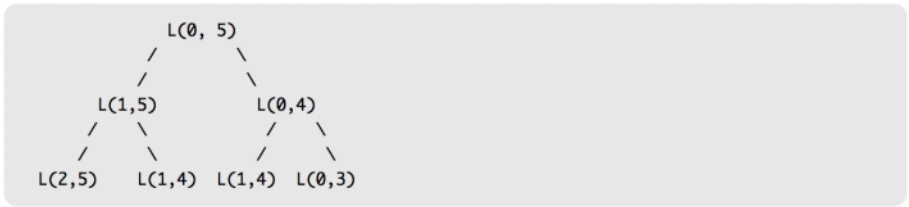
空间复杂度 O（1）

Longest Palindromic Subsequence

Wednesday, March 1, 2017 5:46 PM

subsequence说明序列可以是断开的

Considering the above implementation, following is a partial recursion tree for a sequence of length 6 with all different characters.



In the above partial recursion tree, L(1, 4) is being solved twice. If we draw the complete recursion tree, then we can see that there are many subproblems which are solved again and again. Since same suproblems are called again, this problem has Overlapping Subprolems property. So LPS problem has both properties (see this and this) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array L[][] in bottom up manner.

```
//A Dynamic Programming based Python Program for the Egg Dropping Puzzle
class LPS
{

    // A utility function to get max of two integers
    static int max (int x, int y) { return (x > y)? x : y; }

    // Returns the length of the longest palindromic subsequence in seq
    static int lps(String seq)
    {
       int n = seq.length();
       int i, j, cl;
       int L[][] = new int[n][n];  // Create a table to store results of subprc

       // Strings of length 1 are palindrome of lentgh 1
       for (i = 0; i < n; i++)
           L[i][i] = 1;

        // Build the table. Note that the lower diagonal values of table are
        // useless and not filled in the process. The values are filled in a
        // manner similar to Matrix Chain Multiplication DP solution (See
        // http://www.geeksforgeeks.org/archives/15553). cl is length of
        // substring
        for (cl=2; cl<=n; cl++)
        {
            for (i=0; i<n-cl+1; i++)
            {
                j = i+cl-1;
                if (seq.charAt(i) == seq.charAt(j) && cl == 2)
                   L[i][j] = 2;
                else if (seq.charAt(i) == seq.charAt(j))
                   L[i][j] = L[i+1][j-1] + 2;
                else
                   L[i][j] = max(L[i][j-1], L[i+1][j]);
            }
        }

        return L[0][n-1];
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        String seq = "GEEKSFORGEEKS";
        int n = seq.length();
        System.out.println("The lnegth of the lps is "+ lps(seq));
    }
}
/* This code is contributed by Rajat Mishra */
```

311. Sparse Matrix Multiplication

Monday, February 20, 2017 5:14 PM

Given two sparse matrices **A** and **B**, return the result of **AB**.

You may assume that **A**'s column number is equal to **B**'s row number.

Example:

```
A = [
  [ 1, 0, 0],
  [-1, 0, 3]
]

B = [
  [ 7, 0, 0 ],
  [ 0, 0, 0 ],
  [ 0, 0, 1 ]
]


     |  1 0 0 |   | 7 0 0 |   |  7 0 0 |
AB = | -1 0 3 | x | 0 0 0 | = | -7 0 3 |
                  | 0 0 1 |
```

```
public class Solution {
    public int[][] multiply(int[][] A, int[][] B) {
        int m = A.length, n = A[0].length, nB = B[0].length;
        int[][] C = new int[m][nB];

        for(int i = 0; i < m; i++) {
            for(int k = 0; k < n; k++) {
                if (A[i][k] != 0) {
                    for (int j = 0; j < nB; j++) {
                        if (B[k][j] != 0) C[i][j] += A[i][k] * B[k][j];
                    }
                }
            }
        }
        return C;
    }
}
```

```java
public int[][] multiply(int[][] A, int[][] B) {
    int m = A.length, n = A[0].length, nB = B[0].length;
    int[][] result = new int[m][nB];

    List[] indexA = new List[m];
    for(int i = 0; i < m; i++) {
        List<Integer> numsA = new ArrayList<>();
        for(int j = 0; j < n; j++) {
            if(A[i][j] != 0){
                numsA.add(j);
                numsA.add(A[i][j]);
            }
        }
        indexA[i] = numsA;
    }

    for(int i = 0; i < m; i++) {
        List<Integer> numsA = indexA[i];
        for(int p = 0; p < numsA.size() - 1; p += 2) {
            int colA = numsA.get(p);
            int valA = numsA.get(p + 1);
            for(int j = 0; j < nB; j ++) {
                int valB = B[colA][j];
                result[i][j] += valA * valB;
            }
        }
    }

    return result;
}
```

Sparse vector multiplication

Sunday, February 26, 2017 10:57 PM

```java
//Spase vector multiplication
public int[] multiplication(int[] a, int[] b){
    List<Integer> v1 = new ArrayList<Integer>();
    List<Integer> v2 = new ArrayList<Integer>();
    //Save index for vector a where a[index] != 0
    for(int i=0; i<a.size(); ++i){
        if(a[i] != 0) v1.add(i);
    }
    //Save index for vector b where a[index] != 0
    for(int i=0; i<b.size(); ++i){
        if(b[i] != 0) v2.add(i);

    }
    //Array to save the final result vector
    int result[] = new int[a.length];

    //Two point to go through two different index array
    int index1 = 0, index2 = 0;

    //We can stop loop once one of the index array ends
    //As in that case all the rest equal to 0
    while(index1 < vi.size()){
        int first = v1.get(index1);
        int second = v2.get(index2);
        //If they have same index, we need to save the new number to result
        if(first == second) {
            result[first] = a[first]*b[first];
            index1 ++;
            index2 ++;
        }
        else if(first<second) index1++;
        else index2++;
    }

    return result;
}
```

21. Merge Two Sorted Lists

Monday, February 20, 2017 5:25 PM

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
         if(l1 == null) return l2;
        if(l2 == null) return l1;
        if(l1.val < l2.val){
            l1.next = mergeTwoLists(l1.next, l2);
            return l1;
        } else{
            l2.next = mergeTwoLists(l1, l2.next);
            return l2;
        }
    }
}
```

88. Merge Sorted Array

Monday, February 20, 2017 6:26 PM

Given two sorted integer arrays *nums1* and *nums2*, merge *nums2* into *nums1* as one sorted array.

Note:

You may assume that *nums1* has enough space (size that is greater or equal to $m + n$) to hold additional elements from *nums2*. The number of elements initialized in *nums1* and *nums2* are m and n respectively.

```
1  public class Solution {
2      public void merge(int[] nums1, int m, int[] nums2, int n) {
3          int i=m-1, j=n-1, k=m+n-1;
4          while (i>-1 && j>-1) A[k--]= (A[i]>B[j]) ? A[i--] : B[j--];
5          while (j>-1)         A[k--]=B[j--];
6      }
7  }
```

23. Merge k Sorted Lists

Monday, February 20, 2017 7:06 PM

Merge *k* sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public static ListNode mergeKLists(ListNode[] lists){
    return partion(lists, 0, lists.length-1);
}

public static ListNode partion(ListNode[] lists,int s,int e){
    if(s==e)  return lists[s];
    if(s<e){
        int q=(s+e)/2;
        ListNode l1=partion(lists,s,q);
        ListNode l2=partion(lists,q+1,e);
        return merge(l1,l2);
    }else
        return null;
}

//This function is from Merge Two Sorted Lists.
public static ListNode merge(ListNode l1,ListNode l2){
    if(l1==null) return l2;
    if(l2==null) return l1;
    if(l1.val<l2.val){
        l1.next=merge(l1.next,l2);
        return l1;
    }else{
        l2.next=merge(l1,l2.next);
        return l2;
    }
}
}
```

340. Longest Substring with At Most K Distinct Characters

Monday, February 20, 2017 7:23 PM

Given a string, find the length of the longest substring T that contains at most *k* distinct characters.

For example, Given s = `“eceba”` and k = 2,

T is "ece" which its length is 3.

```
public int lengthOfLongestSubstringKDistinct(String s, int k) {
    int[] count = new int[256];     // there are 256 ASCII characters in the world

    int i = 0;  // i will be behind j
    int num = 0;
    int res = 0;

    for (int j = 0; j < s.length(); j++) {
        if (count[s.charAt(j)]++ == 0) {    // if count[s.charAt(j)] == 0, we know that it is a distinct character
            num++;
        }
        while (num > k && i < s.length()) {     // sliding window
            count[s.charAt(i)]--;
            if (count[s.charAt(i)] == 0){
                num--;
            }
            i++;
        }
        res = Math.max(res, j - i + 1);
    }
    return res;
}
```

360. Sort Transformed Array

Monday, February 20, 2017 7:58 PM

Given a **sorted** array of integers *nums* and integer values *a*, *b* and *c*. Apply a 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**.

Expected time complexity: **O(*n*)**

Example:

```
nums = [-4, -2, 2, 4], a = 1, b = 3, c = 5,

Result: [3, 9, 15, 33]

nums = [-4, -2, 2, 4], a = -1, b = 3, c = 5

Result: [-23, -5, 1, 7]
```

```java
public class Solution {
    public int[] sortTransformedArray(int[] nums, int a, int b, int c) {
        int n = nums.length;
        int[] sorted = new int[n];
        int i = 0, j = n - 1;
        int index = a >= 0 ? n - 1 : 0;
        while (i <= j) {
            if (a >= 0) {
                sorted[index--] = quad(nums[i], a, b, c) >= quad(nums[j], a, b, c) ? quad(nums[i++], a, b, c) : quad(nums[j--],
                    a, b, c);
            } else {
                sorted[index++] = quad(nums[i], a, b, c) >= quad(nums[j], a, b, c) ? quad(nums[j--], a, b, c) : quad(nums[i++],
                    a, b, c);
            }
        }
        return sorted;
    }

    private int quad(int x, int a, int b, int c) {
        return a * x * x + b * x + c;
    }
}
```

200. Number of Islands

Monday, February 20, 2017 8:16 PM

Given a 2d grid map of `'1'` s (land) and `'0'` s (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
11110
11010
11000
00000
```

Answer: 1

Example 2:

```
11000
11000
00100
00011
```

Answer: 3

DFS:

```java
public class Solution {

private int n;
private int m;

public int numIslands(char[][] grid) {
    int count = 0;
    n = grid.length;
    if (n == 0) return 0;
    m = grid[0].length;
    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++)
            if (grid[i][j] == '1') {
                DFSMarking(grid, i, j);
                ++count;
            }
    }
    return count;
}

private void DFSMarking(char[][] grid, int i, int j) {
    if (i < 0 || j < 0 || i >= n || j >= m || grid[i][j] != '1') return;
    grid[i][j] = '0';
    DFSMarking(grid, i + 1, j);
    DFSMarking(grid, i - 1, j);
    DFSMarking(grid, i, j + 1);
    DFSMarking(grid, i, j - 1);
}
```

BFS:

```java
public int numIslands(char[][] grid) {
    int count=0;
    for(int i=0;i<grid.length;i++)
        for(int j=0;j<grid[0].length;j++){
            if(grid[i][j]=='1'){
                bfsFill(grid,i,j);
                count++;
            }
        }
    return count;
}
```

```java
private void bfsFill(char[][] grid,int x, int y){
    grid[x][y]='0';
    int n = grid.length;
    int m = grid[0].length;
    LinkedList<Integer> queue = new LinkedList<Integer>();
    int code = x*m+y;
    queue.offer(code);
    while(!queue.isEmpty())
    {
        code = queue.poll();
        int i = code/m;
        int j = code%m;
        if(i>0 && grid[i-1][j]=='1')    //search upward and mark adjacent '1's as '0'.
        {
            queue.offer((i-1)*m+j);
            grid[i-1][j]='0';
        }
        if(i<n-1 && grid[i+1][j]=='1')  //down
        {
            queue.offer((i+1)*m+j);
            grid[i+1][j]='0';
        }
        if(j>0 && grid[i][j-1]=='1')  //left
        {
            queue.offer(i*m+j-1);
            grid[i][j-1]='0';
        }
        if(j<m-1 && grid[i][j+1]=='1')  //right
        {
            queue.offer(i*m+j+1);
            grid[i][j+1]='0';
        }
    }
}
```

Task cool down

Monday, February 20, 2017 8:42 PM

二面电话一接通，听口音感觉是印度人。。。（其实后来发现是俄罗斯人），一上来没问简历，直接做题，但是因为口音问题（囧），coderpad的链接就给我念了五分钟。。。他念的C和Z我实在分不清楚，最后就问他是Cat还是Zippo，小哥哈哈一下笑，Cat。之后上题（运气挺好的这次还是只有一道）。给了一串task，不同的task可能属于不同type。这些task要放到CPU里运行，运行同一种type是要考虑一个**冷却时间**。。。弄了半天，过了好几个例子才搞明白，就类似于一个OS。给你一个单线程的scheduler，和eg. 4种thread：1，2，3，4, 冷却时间: 3，在multithreading的时候同类型的thread要等上一个thread跑完**冷却时间**之后才能运行，求最后scheduler用了多少time slot。 举个例子，thread: 1, 2, 1, 1, 3, 4; 冷却时间: 2 time slot，scheduler应该是这样的：1, 2, _, 1, _, _, 3, 4，最后返回8. 想了一下就用个map存了不同type最近的time slot，每碰到相同的type就check一下冷却时间过了没，没过就等待。写到等待的地方，在考虑时间要不要加一的时候，脑子有点绕晕了。。。最后觉得不用加一，就告诉小哥我写完了，跑了一个test case发现撒比。。加上一，过了。面完之后感觉没发挥出正常水平，这么简单的地方还出问题。。。不过和小哥聊的挺开心的，两人一直哈哈＝。＝结束的时候小哥给我发了他Facebook链接，我们加了好友聊了几句，让我觉得可能还有戏?

```
mission order , same task cannot be called in a period
(missions, task, cd)

### Solution

Complexity:
- time: O(n)
- space: O(n)
```cpp
class solution{
    int mission_schedule(vector<int>& mission, int cooldown){
        int len = mission.size();
        if(!len) return 0;
        unordered_map<int, int> mis_pre;
        int result = 0;
        for(auto mis:mission){
            ++ result;
            if(mis_pre.find(mis) == mis_pre.end())
                mis_pre.insert(make_pair(mis, result));
            else{
                int time_pre = mis_pre[mis];
                if(result - time_pre <= cooldown){
                    result = cooldown + time_pre + 1;
                }
                mis_pre[mis] = result;
            }
        }
        return result;
    }
```
```

```java
//Task cooldown
public int taskCoolDown(int[] nums, int cooldown){
    int len = nums.length;
    if(len == 0) return 0;
    HashMap<Integer, Integer> map = new HashMap<>();
    int result = 0;
    for(int num : nums){
        ++result;
        if(!map.containsKey(num)){
            map.put(num, result);
        }else{
            int time_pre = map.get(num);
            if(result - time_pre < cooldown){
                result = cooldown + time_pre + 1;
            }
            map.put(num, result);
        }
    }
    return result;
}
```

17. Letter Combinations of a Phone Number

Friday, February 3, 2017 4:17 PM

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.

Input: Digit string "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

Note:

Although the above answer is in lexicographical order, your answer could be in any order you want.

Subscribe to see which companies asked this question

```java
public class Solution {
    public List<String> letterCombinations(String digits) {
    LinkedList<String> ans = new LinkedList<String>();
    String[] mapping = new String[] {"0", "1", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
    for(int i =0; i<digits.length();i++){
        int x = digits.charAt(i)-'0';
        if(i!=0){
            while(ans.peek().length()==i){
                String t = ans.remove();
                for(char s : mapping[x].toCharArray())
                    ans.add(t+s);
            }
        }else{
            for(char s : mapping[x].toCharArray())
                    ans.add(Character.toString(s));
        }
    }
    return ans;
    }
}
```

The **java.util.LinkedList.peek()** method retrieves, but does not remove, the head (first element) of this list.

211. Add and Search Word - Data structure design

Tuesday, February 21, 2017 10:18 AM

Design a data structure that supports the following two operations:

```
void addWord(word)
bool search(word)
```

search(word) can search a literal word or a regular expression string containing only letters `a-z` or `.`. A `.` means it can represent any one letter.

For example:

```
addWord("bad")
addWord("dad")
addWord("mad")
search("pad") -> false
search("bad") -> true
search(".ad") -> true
search("b..") -> true
```

Note:

You may assume that all words are consist of lowercase letters `a-z`.

```java
public class WordDictionary {
    public class TrieNode {
        public TrieNode[] children = new TrieNode[26];
        public String item = "";
    }

    private TrieNode root = new TrieNode();

    public void addWord(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                node.children[c - 'a'] = new TrieNode();
            }
            node = node.children[c - 'a'];
        }
        node.item = word;
    }

    public boolean search(String word) {
        return match(word.toCharArray(), 0, root);
    }
```

```
    private boolean match(char[] chs, int k, TrieNode node) {
        if (k == chs.length) return !node.item.equals("");
        if (chs[k] != '.') {
            return node.children[chs[k] - 'a'] != null && match(chs, k + 1, node.children[chs[k] - 'a']);
        } else {
            for (int i = 0; i < node.children.length; i++) {
                if (node.children[i] != null) {
                    if (match(chs, k + 1, node.children[i])) {
                        return true;
                    }
                }
            }
        }
        return false;
    }
}
```

```
//Word dictionary
public class TrieNode{
    public class TrieNode{
        public TrieNode[] children = new TrieNode[26];
        public String item = "";
    }

    private TrieNode root = new TrieNode();

    public void addWord(String word){
        TreeNode node = root;
        for(char c: word.toCharArray()){
            if(node.children[c-'a'] == null){
                node.children[c-'a'] = new TrieNode();
            }
            node = node.children[c-'a'];
        }
        node.item = word;
    }

    public boolean search(String word){
        return match(word.toCharArray(), 0, root);
    }

    public boolean match(char[] chs, int k, TrieNode node){
        if(k == chs.length) return !node.item.equals("");
        if(chs[k] != '.'){
            return node.children[ch[k]-'a'] != null && match(chs, k+1, node.children[ch[k]-'a']);
        }
        else{
            for(int i=0; i<node.children.length; ++i){
                if(node.children[i] != null){
                    if(match(chs, k+1, node.children[i])){
                        return true;
                    }
                }
            }
        }
        return false;
    }
}
```

76. Minimum Window Substring

Tuesday, February 21, 2017 10:29 AM

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

For example,

S = `"ADOBECODEBANC"`

T = `"ABC"`

Minimum window is `"BANC"`.

Note:

If there is no such window in S that covers all characters in T, return the empty string `""`.

If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.

```
public String minWindow(String S, String T) {
    if(S==null||S.isEmpty()||T==null||T.isEmpty()) return "";
    int i=0, j=0;
    int[] Tmap=new int[256];
    int[] Smap=new int[256];
    for(int k=0; k< T.length(); k++){
        Tmap[T.charAt(k)]++;
    }
    int found=0;
    int length=Integer.MAX_VALUE;
    String res="";
    while(j<S.length()){
        if(found<T.length()){
            if(Tmap[S.charAt(j)]>0){
                Smap[S.charAt(j)]++;
                if(Smap[S.charAt(j)]<=Tmap[S.charAt(j)]){
                    found++;
                }
            }
            j++;
        }
        while(found==T.length()){
            if(j-i<length){
                length=j-i; res=S.substring(i,j);
            }
            if(Tmap[S.charAt(i)]>0){
                Smap[S.charAt(i)]--;
                if(Smap[S.charAt(i)]<Tmap[S.charAt(i)]){
                    found--;
                }
            }
            i++;
        }
    }
    return res;
```

Maximum length in matrix

Tuesday, February 21, 2017 10:53 AM

题目：矩阵中由1构成的一横一竖的连续的1，并且这一横一竖有一个交叉点的， 是一条十字路
总的来说对每一个1： 和他在同一列上，与他相连的的连续的1的数量 + 和同他在同一行上，与他相连的连续的1的数量 的和 就是以当前1为交叉点的十字路的长度
找出矩阵中最长的十字路。

举几个例子吧

0 0 0
1 1 1
1 0 0 中最长的十字路长度是4

0 0 1 0 0 0
0 0 1 1 1 1
1 1 1 0 1 0
0 0 1 0 0 1
中最长的十字路长度是7

```
01. public class Solution {
02.     public int maxXing(int[][] nums) {
03.         if (nums.length == 0 || nums[0].length == 0)
04.             return 0;
05.         int rowHit = 0;
06.         int[] colHit = new int[nums[0].length];
07.         int max = 0;
08.         for (int i = 0; i < nums.length; i++) {
09.             for (int j = 0; j < nums[0].length; j++) {
10.                 if (nums[i][j] == 0) continue;
11.                 // get rowHit
12.                 if (j == 0 || nums[i][j - 1] == 0) {
13.                     rowHit = 0;
14.                     while (j + rowHit < nums[0].length && nums[i][j + rowHit] == 1)
15.                         rowHit ++;
16.                 }
17.                 // get colHit
18.                 if (i == 0 || nums[i - 1][j] == 0) {
19.                     colHit[j] = 0;
20.                     while (i + colHit[j] < nums.length && nums[i + colHit[j]][j] == 1)
21.                         colHit[j] ++;
22.                 }
23.                 max = Integer.max(max, colHit[j] + rowHit - 1);
24.             }
25.         }
26.         return max;
27.     }
28. }
```

复制代码

361. Bomb Enemy

Tuesday, February 28, 2017 4:16 PM

Given a 2D grid, each cell is either a wall `'W'`, an enemy `'E'` or empty `'0'` (the number zero), return the maximum enemies you can kill using one bomb.

The bomb kills all the enemies in the same row and column from the planted point until it hits the wall since the wall is too strong to be destroyed.

Note that you can only put the bomb at an empty cell.

Example:

```
For the given grid

0 E 0 0
E 0 W E
0 E 0 0

return 3. (Placing a bomb at (1,1) kills 3 enemies)
```

Credits:

Special thanks to @memoryless for adding this problem and creating all test cases.

only need to store one killed enemies value for a row and an array of each column;
if current grid value is W, this means previous stored value becomes invalid, you need to recalculate.

```
public int maxKilledEnemies(char[][] grid) {
    if(grid == null || grid.length == 0 ||  grid[0].length == 0) return 0;
    int max = 0;
    int row = 0;
    int[] col = new int[grid[0].length];
    for(int i = 0; i<grid.length; i++){
        for(int j = 0; j<grid[0].length;j++){
            if(grid[i][j] == 'W') continue;
            if(j == 0 || grid[i][j-1] == 'W'){
                 row = killedEnemiesRow(grid, i, j);
            }
            if(i == 0 || grid[i-1][j] == 'W'){
                 col[j] = killedEnemiesCol(grid,i,j);
            }
            if(grid[i][j] == '0'){
                max = (row + col[j] > max) ? row + col[j] : max;
            }
        }

    }

    return max;
}
```

```java
//calculate killed enemies for row i from column j
private int killedEnemiesRow(char[][] grid, int i, int j){
    int num = 0;
    while(j <= grid[0].length-1 && grid[i][j] != 'W'){
        if(grid[i][j] == 'E') num++;
        j++;
    }
    return num;
}
//calculate killed enemies for  column j from row i
private int killedEnemiesCol(char[][] grid, int i, int j){
    int num = 0;
    while(i <= grid.length -1 && grid[i][j] != 'W'){
        if(grid[i][j] == 'E') num++;
        i++;
    }
    return num;
}
```

133. Clone Graph

Tuesday, February 21, 2017 11:15 AM

Clone an undirected graph. Each node in the graph contains a `label` and a list of its `neighbors`.

OJ's undirected graph serialization:

Nodes are labeled uniquely.

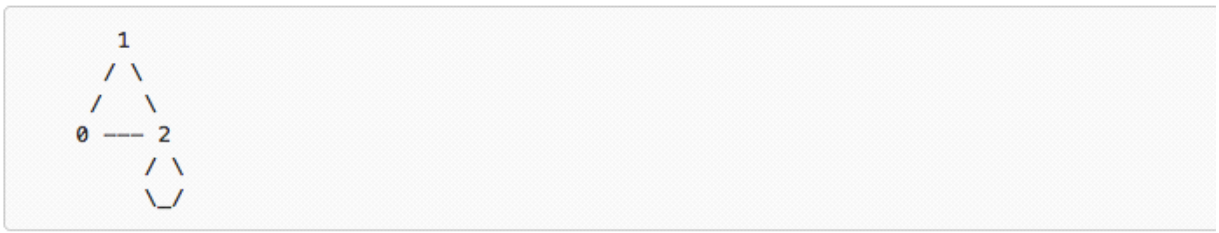
We use `#` as a separator for each node, and `,` as a separator for node label and each neighbor of the node.

As an example, consider the serialized graph `{0,1,2#1,2#2,2}`.

The graph has a total of three nodes, and therefore contains three parts as separated by `#`.

1. First node is labeled as `0`. Connect node `0` to both nodes `1` and `2`.
2. Second node is labeled as `1`. Connect node `1` to node `2`.
3. Third node is labeled as `2`. Connect node `2` to node `2` (itself), thus forming a self-cycle.

Visually, the graph looks like the following:




```
/**
 * Definition for undirected graph.
 * class UndirectedGraphNode {
 *     int label;
 *     List<UndirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) { label = x; neighbors = new ArrayList<UndirectedGraphNode>(); }
 * };
 */
```

```java
public class Solution {
    private HashMap<Integer, UndirectedGraphNode> map = new HashMap<>();
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        return clone(node);
    }

    private UndirectedGraphNode clone(UndirectedGraphNode node) {
        if (node == null) return null;

        if (map.containsKey(node.label)) {
            return map.get(node.label);
        }
        UndirectedGraphNode clone = new UndirectedGraphNode(node.label);
        map.put(clone.label, clone);
        for (UndirectedGraphNode neighbor : node.neighbors) {
            clone.neighbors.add(clone(neighbor));
        }
        return clone;
    }
}
```

273. Integer to English Words

Tuesday, February 21, 2017 11:24 AM

Convert a non-negative integer to its english words representation. Given input is guaranteed to be less than $2^{31} - 1$.

For example,

```
123 -> "One Hundred Twenty Three"
12345 -> "Twelve Thousand Three Hundred Forty Five"
1234567 -> "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"
```

```java
private final String[] LESS_THAN_20 = {"", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "
private final String[] TENS = {"", "Ten", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninet
private final String[] THOUSANDS = {"", "Thousand", "Million", "Billion"};

public String numberToWords(int num) {
    if (num == 0) return "Zero";

    int i = 0;
    String words = "";

    while (num > 0) {
        if (num % 1000 != 0)
            words = helper(num % 1000) +THOUSANDS[i] + " " + words;
        num /= 1000;
        i++;
    }

    return words.trim();
}

private String helper(int num) {
    if (num == 0)
        return "";
    else if (num < 20)
        return LESS_THAN_20[num] + " ";
    else if (num < 100)
        return TENS[num / 10] + " " + helper(num % 10);
    else
        return LESS_THAN_20[num / 100] + " Hundred " + helper(num % 100);
}
```

31. Next Permutation

Tuesday, February 21, 2017 12:07 PM

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

`1,2,3` → `1,3,2`

`3,2,1` → `1,2,3`

`1,1,5` → `1,5,1`

```java
public class Solution {
    public void nextPermutation(int[] nums) {
        if(nums.length < 2) return ;
        //Do not initialize pos to 0
        //Because in that way you can not distinguish
        //Whether index 0 is the one that satisfy nums[i]<nums[i+1]
        //Or the array just sorted in decrease order
        int pos = -1;
        //Here we must go from end to begin,
        //Because we need to search from right to left to find the first one
        //that decrease
        for(int i=nums.length-1; i>0; --i){
            if(nums[i]>nums[i-1]){
                pos = i-1;
                break;
            }
        }
        if(pos == -1){
            reverse(nums, 0, nums.length-1);
        }
        else{
            //Here should be i>pos, not i>0, as we stop when i=pos
            for(int i=nums.length-1; i>pos; i--){
                if(nums[i]>nums[pos]){
                    swap(nums,i,pos);
                    break;
                }
            }
            reverse(nums, pos+1, nums.length-1);
        }

    }
```

```
34
35      public void swap(int[] nums, int i, int j){
36          int temp = nums[i];
37          nums[i] = nums[j];
38          nums[j] = temp;
39      }
40
41      public void reverse(int[] nums, int i, int j){
42          int p = j;
43          for(int k = i; k<=(i+j)/2; ++k){
44              swap(nums, k, p);
45              p--;
46          }
47      }
48  }
```

471. Encode String with Shortest Length

Sunday, February 26, 2017 4:28 PM

Given a **non-empty** string, encode the string such that its encoded length is the shortest.

The encoding rule is: `k[encoded_string]`, where the *encoded_string* inside the square brackets is being repeated exactly *k* times.

Note:

1. *k* will be a positive integer and encoded string will not be empty or have extra space.
2. You may assume that the input string contains only lowercase English letters. The string's length is at most 160.
3. If an encoding process does not make the string shorter, then do not encode it. If there are several solutions, return any of them is fine.

Example 1:

```
Input: "aaa"
Output: "aaa"
Explanation: There is no way to encode it such that it is shorter than the input string, so we do not encode it.
```

Example 2:

```
Input: "aaaaa"
Output: "5[a]"
Explanation: "5[a]" is shorter than "aaaaa" by 1 character.
```

Example 3:

```
Input: "aaaaaaaaaa"
Output: "10[a]"
Explanation: "a9[a]" or "9[a]a" are also valid solutions, both of them have the same length = 5, which is the same as "10[a]".
```

Example 4:

```
Input: "aabcaabcd"
Output: "2[aabc]d"
Explanation: "aabc" occurs twice, so one answer can be "2[aabc]d".
```

Example 5:

```
Input: "abbbabbbcabbbabbbc"
Output: "2[2[abbb]c]"
Explanation: "abbbabbbc" occurs twice, but "abbbabbbc" can also be encoded to "2[abbb]c", so one answer can be "2[2[abbb]c]".
```

用DP的思想来做

This is the first question I have answered in Leetcode. I hope you guys will like my solution. The approach here is simple. We will form 2-D array of Strings.
dp[i][j] = string from index i to index j in encoded form.

We can write the following formula as:-
dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j]) or if we can find some pattern in string from i to j which will result in more less length.

Time Complexity = O(n^3)

时间复杂度存疑。

```
public String encode(String s) {
   String[][] dp = new String[s.length()][s.length()];

   for(int l=0;l<s.length();l++) {
       for(int i=0;i<s.length()-l;i++) {
           int j = i+l;
           String substr = s.substring(i, j+1);
           // Checking if string length < 5. In that case, we know that encoding will not help.
           if(j - i < 4) {
               dp[i][j] = substr;
           } else {
               dp[i][j] = substr;
               // Loop for trying all results that we get after dividing the strings into 2 and combine the   resul
               for(int k = i; k<j;k++) {
                   if((dp[i][k] + dp[k+1][j]).length() < dp[i][j].length()){
                       dp[i][j] = dp[i][k] + dp[k+1][j];
                   }
               }


                 // Loop for checking if string can itself found some pattern in it which could be repeated.
                 for(int k=0;k<substr.length();k++) {
                     String repeatStr = substr.substring(0, k+1);
                     if(repeatStr != null
                        && substr.length()%repeatStr.length() == 0
                        && substr.replaceAll(repeatStr, "").length() == 0) {
                           String ss = substr.length()/repeatStr.length() + "[" + dp[i][i+k] + "]";
                           if(ss.length() < dp[i][j].length()) {
                             dp[i][j] = ss;
                           }
                      }
                 }
             }
         }
     }

     return dp[0][s.length()-1];
 }
 }
```

最后寻找重复子串的地方效率不高，而且用了replaceAll这个函数，时间复杂度可能是O(n). 我们可以用KMP来代替这一部分。

下面说一下KMP这个算法：
比如，在ABCABABCAB中寻找ABCAB并且输出index. 对于这个例子来说输出0和5.

我们对于ABCAB进行预处理，LPS[]数组存储相应的数值。那么LPS中的数就是[0,0,0,1,2].因为直到第二次A出现，才开始重复。

类似的，如果是AAAA，那么就是[0,1,2,3]，一定是从0开始
AABAA , [0,1,0,1,2]
AAABAAAAA, [0,1,2,0,1,2,3,3,3]

这样我们每次找匹配的时候，如果发现在某个位置不匹配，就让指向匹配子串（较短的那个）的index跳到LPS[j-1]去（假设当前在j）

预处理的代码如下：

```
void computeLPSArray(String pat, int M, int lps[])
{
    // length of the previous longest prefix suffix
    int len = 0;
    int i = 1;
    lps[0] = 0;  // lps[0] is always 0

    // the loop calculates lps[i] for i = 1 to M-1
    while (i < M)
    {
        if (pat.charAt(i) == pat.charAt(len))
        {
            len++;
            lps[i] = len;
            i++;
        }
        else  // (pat[i] != pat[len])
        {
            // This is tricky. Consider the example.
            // AAACAAAA and i = 7. The idea is similar
            // to search step.
            if (len != 0)
            {
                len = lps[len-1];

                // Also, note that we do not increment
                // i here
            }
            else  // if (len == 0)
            {
                lps[i] = len;
                i++;
            }
        }
    }
}
```

匹配部分代码如下：

```
class KMP_String_Matching
{
    void KMPSearch(String pat, String txt)
    {
        int M = pat.length();
        int N = txt.length();

        // create lps[] that will hold the longest
        // prefix suffix values for pattern
        int lps[] = new int[M];
        int j = 0;  // index for pat[]

        // Preprocess the pattern (calculate lps[]
        // array)
        computeLPSArray(pat,M,lps);

        int i = 0;  // index for txt[]
        while (i < N)
        {
            if (pat.charAt(j) == txt.charAt(i))
            {
                j++;
                i++;
            }
            if (j == M)
            {
                System.out.println("Found pattern "+
                                   "at index " + (i-j));
                j = lps[j-1];
            }

            // mismatch after j matches
            else if (i < N && pat.charAt(j) != txt.charAt(i))
            {
                // Do not match lps[0..lps[j-1]] characters,
                // they will match anyway
                if (j != 0)
                    j = lps[j-1];
                else
                    i = i+1;
            }
        }
    }
```

下面改进最初的答案，用KMP来算。

```java
public class Solution {
    public String encode(String s) {
        if(s == null || s.length() <= 4) return s;

        int len = s.length();

        String[][] dp = new String[len][len];

        // iterate all the length, stay on the disgnose of the dp matrix
        for(int l = 0; l < len; l ++) {
            for(int i = 0; i < len - l; i ++) {
                int j = i + l;
                String substr = s.substring(i, j + 1);
                dp[i][j] = substr;
                if(l < 4) continue;

                for(int k = i; k < j; k ++) {
                    if(dp[i][k].length() + dp[k + 1][j].length() < dp[i][j].length()) {
                        dp[i][j] = dp[i][k] + dp[k + 1][j];
                    }
                }

                String pattern = kmp (substr);
                if(pattern.length() == substr.length()) continue; // no repeat pattern found
                String patternEncode = substr.length() / pattern.length() + "[" + dp[i][i + pattern.length() - 1] + "]";
                if(patternEncode.length() < dp[i][j].length()) {
                    dp[i][j] = patternEncode;
                }
            }
        }

        return dp[0][len - 1];
    }


    private String kmp (String s) {
        int len = s.length();
        int[] LPS = new int[len];

        int i = 1, j = 0;
        LPS[0] = 0;
        while(i < len) {
            if(s.charAt(i) == s.charAt(j)) {
                LPS[i ++] = ++ j;
            } else if(j == 0) {
                LPS[i ++] = 0;
            } else {
                j = LPS[j - 1];
            }
        }

        int patternLen = len - LPS[len - 1];
        //Here we do not need to check substr.replaceAll(repeatStr, "").length() == 0
        //As we are using patternLen = len - LPS[len - 1] and len % patternLen == 0
        if(patternLen != len && len % patternLen == 0) {
            return s.substring(0, patternLen);
        } else {
            return s;
        }
    }
}
```

394. Decode String

Tuesday, February 28, 2017 4:49 PM

- Total Accepted: **23034**
- Total Submissions: **56586**
- Difficulty: **Medium**
- Contributors: **Admin**

Given an encoded string, return it's decoded string.

The encoding rule is: `k[encoded_string]`, where the *encoded_string* inside the square brackets is being repeated exactly *k* times. Note that *k* is guaranteed to be a positive integer.

You may assume that the input string is always valid; No extra white spaces, square brackets are well-formed, etc.

Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, *k*. For example, there won't be input like `3a` or `2[4]`.

Examples:

```
s = "3[a]2[bc]", return "aaabcbc".
s = "3[a2[c]]", return "accaccacc".
s = "2[abc]3[cd]ef", return "abcabccdcdcdef".
```

```java
public class Solution {
    public String decodeString(String s) {
        String res = "";
        Stack<Integer> countStack = new Stack<>();
        Stack<String> resStack = new Stack<>();
        int idx = 0;
        while (idx < s.length()) {
            if (Character.isDigit(s.charAt(idx))) {
                int count = 0;
                while (Character.isDigit(s.charAt(idx))) {
                    count = 10 * count + (s.charAt(idx) - '0');
                    idx++;
                }
                countStack.push(count);
            }
            else if (s.charAt(idx) == '[') {
                resStack.push(res);
                res = "";
                idx++;
            }
            else if (s.charAt(idx) == ']') {
                StringBuilder temp = new StringBuilder (resStack.pop());
                int repeatTimes = countStack.pop();
                for (int i = 0; i < repeatTimes; i++) {
                    temp.append(res);
                }
                res = temp.toString();
                idx++;
            }
            else {
                res += s.charAt(idx++);
            }
        }
        return res;
    }
```

10. Regular Expression Matching

Sunday, February 26, 2017 8:58 PM

Implement regular expression matching with support for `'.'` and `'*'`.

```
'.' Matches any single character.
'*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

The function prototype should be:
bool isMatch(const char *s, const char *p)

Some examples:
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "a*") → true
isMatch("aa", ".*") → true
isMatch("ab", ".*") → true
isMatch("aab", "c*a*b") → true
```

递归解法

```java
public class Solution {
    public boolean isMatch(String s, String p) {
       if (p.isEmpty()) {
        return s.isEmpty();
    }

    if (p.length() == 1 || p.charAt(1) != '*') {
        //not match
        if (s.isEmpty() || (p.charAt(0) != '.' && p.charAt(0) != s.charAt(0))) {
            return false;
        }
        //match at postion 0
        else {
            return isMatch(s.substring(1), p.substring(1));
        }
    }

    //p.length() >=2 && p.charAt(1) == '*'
    while (!s.isEmpty() && (s.charAt(0) == p.charAt(0) || p.charAt(0) == '.')) {
        //in case * cancel the char before it
        if (isMatch(s, p.substring(2))) {
            return true;
        }
        //in case * repeat preceding char
        s = s.substring(1);
    }

    //Now we get rid of the * in string p, and we continue match process
    return isMatch(s, p.substring(2));
    }
}
```

DP解法

```java
public boolean isMatch(String s, String p) {

    if (s == null || p == null) {
        return false;
    }
    boolean[][] dp = new boolean[s.length()+1][p.length()+1];
    dp[0][0] = true;
    for (int i = 0; i < p.length(); i++) {
        if (p.charAt(i) == '*' && dp[0][i-1]) {
            dp[0][i+1] = true;
        }
    }
    for (int i = 0 ; i < s.length(); i++) {
        for (int j = 0; j < p.length(); j++) {
            if (p.charAt(j) == '.') {
                dp[i+1][j+1] = dp[i][j];
            }
            if (p.charAt(j) == s.charAt(i)) {
                dp[i+1][j+1] = dp[i][j];
            }
            if (p.charAt(j) == '*') {
                if (p.charAt(j-1) != s.charAt(i) && p.charAt(j-1) != '.') {
                    dp[i+1][j+1] = dp[i+1][j-1];
                } else {
                    dp[i+1][j+1] = (dp[i+1][j] || dp[i][j+1] || dp[i+1][j-1]);
                }
            }
        }
    }
    return dp[s.length()][p.length()];
}
```

44. Wildcard Matching

Sunday, February 26, 2017 9:22 PM

Implement wildcard pattern matching with support for `'?'` and `'*'`.

```
'?' Matches any single character.
'*' Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).

The function prototype should be:
bool isMatch(const char *s, const char *p)

Some examples:
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "*") → true
isMatch("aa", "a*") → true
isMatch("ab", "?*") → true
isMatch("aab", "c*a*b") → false
```

DP解法

```
boolean comparison(String str, String pattern) {
        int s = 0, p = 0, match = 0, starIdx = -1;
        while (s < str.length()){
            // advancing both pointers
            if (p < pattern.length()  && (pattern.charAt(p) == '?' || str.charAt(s) == pattern.charAt(p))){
                s++;
                p++;
            }
            // * found, only advancing pattern pointer
            else if (p < pattern.length() && pattern.charAt(p) == '*'){
                starIdx = p;
                match = s;
                p++;
            }
           // last pattern pointer was *, advancing string pointer
            else if (starIdx != -1){
                p = starIdx + 1;
                match++;
                s = match;
            }
           //current pattern pointer is not star, last patter pointer was not *
          //characters do not match
            else return false;
        }

        //check for remaining characters in pattern
        while (p < pattern.length() && pattern.charAt(p) == '*')
            p++;

        return p == pattern.length();
}
```

282. Expression Add Operators

Sunday, February 19, 2017 2:02 PM

Given a string that contains only digits `0-9` and a target value, return all possibilities to add **binary** operators (not unary) `+`, `-`, or `*` between the digits so they evaluate to the target value.

Examples:

```
"123", 6 -> ["1+2+3", "1*2*3"]
"232", 8 -> ["2*3+2", "2+3*2"]
"105", 5 -> ["1*0+5","10-5"]
"00", 0 -> ["0+0", "0-0", "0*0"]
"3456237490", 9191 -> []
```

```java
public class Solution {
    public List<String> addOperators(String num, int target) {
        List<String> result = new LinkedList<String>();
        if(num.length() == 0) return result;
        char[] path = new char[num.length()*2 - 1];
        //How to change string to char array
        char[] nums = num.toCharArray();
        long n = 0;
        for(int i=0; i<nums.length; ++i){
            n = n*10 + nums[i] - '0';
            path[i] = nums[i];
            dfs(result, target, path, nums, i+1, 0, n, i+1);
            if(n == 0) break;
        }
        return result;
```

```java
//result is the final list of all valid string
//target is the sum value given by the problem
//path is current string
//nums is a copy of num but in char array, not string
//len is current index of the end of the current string
//before is the number on the left of current symbol
//cur is the number on the right of current symbol
//rightPos is the end index of the string on the right side of the symbol
void dfs(List<String> result, int target, char[] path, char[] nums, int len, long before, long cur, int rightPos){
    if(rightPos == nums.length){
        if(before + cur == target) result.add(new String(path, 0, len));
        return;
    }

    long n = 0;
    int j = len+1;
    for(int i=rightPos; i<nums.length; ++i){
        n = n*10+nums[i]-'0';
        path[j++] = nums[i];
        path[len] = '+';
        dfs(result, target, path, nums, j, before+cur, n, i+1);
        path[len] = '-';
        dfs(result, target, path, nums, j, before+cur, -n, i+1);
        path[len] = '*';
        dfs(result, target, path, nums, j, before, cur*n, i+1);
        if(nums[rightPos] == '0') break;
    }

}
```

class mutex, lock(), unlock()

Sunday, February 26, 2017 9:36 PM

```
public class ReadWriteLock{

  private int readers       = 0;
  private int writers       = 0;
  private int writeRequests = 0;

  public void lockRead() throws InterruptedException{
    lock();
    while(writers > 0 || writeRequests > 0){
        await();
    readers++;
    unlock();

  }

  public void unlockRead(){
    lock();
    readers--;
    signal();
    unlock();
  }

  public void lockWrite() throws InterruptedException{
      lock();
    writeRequests++;

    while(readers > 0 || writers > 0){
      await();
    }
    writeRequests--;
    writers++;
      unlock();
  }


   public void unlockWrite() throws InterruptedException{
       lock();
     writers--;
     notifyAll();
       unlock();
   }
}
```

Writer process:

1. Writer requests the entry to critical section.
2. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
3. It exits the critical section.

```
do {
    // writer requests for critical section
    wait(wrt);

    // performs the write

    // leaves the critical section
    signal(wrt);

} while(true);
```

Reader process:

1. Reader requests the entry to critical section.
2. If allowed:
    - it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the **wrt** semaphore to restrict the entry of writers if any reader is inside.
    - It then, signals mutex as any other reader is allowed to enter while others are already reading.
    - After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore "wrt" as now, writer can enter the critical section.
3. If not allowed, it keeps on waiting.

```
do {

   // Reader wants to enter the critical section
   wait(mutex);

   // The number of readers has now increased by 1
   readcnt++;

   // there is atleast one reader in the critical section
   // this ensure no writer can enter if there is even one reader
   // thus we give preference to readers here
   if (readcnt==1)
      wait(wrt);

   // other readers can enter while this current reader is inside
   // the critical section
   signal(mutex);

   // current reader performs reading here
   wait(mutex);   // a reader wants to leave

   readcnt--;

   // that is, no reader is left in the critical section,
   if (readcnt == 0)
       signal(wrt);         // writers can enter

   signal(mutex); // reader leaves

} while(true);
```

277. Find the Celebrity

Monday, February 27, 2017 6:28 PM

Suppose you are at a party with `n` people (labeled from `0` to `n - 1`) and among them, there may exist one celebrity. The definition of a celebrity is that all the other `n - 1` people know him/her but he/she does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. The only thing you are allowed to do is to ask questions like: "Hi, A. Do you know B?" to get information of whether A knows B. You need to find out the celebrity (or verify there is not one) by asking as few questions as possible (in the asymptotic sense).

You are given a helper function `bool knows(a, b)` which tells you whether A knows B. Implement a function `int findCelebrity(n)`, your function should minimize the number of calls to `knows`.

Note: There will be exactly one celebrity if he/she is in the party. Return the celebrity's label if there is a celebrity in the party. If there is no celebrity, return `-1`.

- Total Accepted: **23660**
- Total Submissions: **66609**
- Difficulty: **Medium**
- Contributors: **Admin**

Hide Company Tags LinkedIn Facebook

Show Tags

Have you met this question in a real interview? Yes No

```java
/* The knows API is defined in the parent class Relation.
      boolean knows(int a, int b); */

public class Solution extends Relation {
    public int findCelebrity(int n) {
        int candidate = 0;
        for(int i = 1; i < n; i++){
            if(knows(candidate, i))
                candidate = i;
        }
        for(int i = 0; i < n; i++){
            if(i != candidate && (knows(candidate, i) || !knows(i, candidate))) return -1;
        }
        return candidate;
    }
}
```

第一遍循环找到可能的candidate

第二遍循环确实这个candidate是真的

假设a是candidate，那么只要candidate不认识i，就说明candidate暂时是真的，而且i不可能是candidate,因为candidate必须被所有人知道，而且不认识所有人。如果candidate认识i，那就说明candidate是假的，因为他必须不认识所有人。这个时候就变成 i可能是新的candidate。因为最多只能有1个candidate，也可能没有，所以我们需要再循环一遍确认这个candidate是真的。

Bipartite check

Monday, February 27, 2017 8:42 PM

```
//Check if a graph is bipartite
Class Bipartite{
    //First we need to know the number of vertices
    //So that we can later color all the vertices
    final static int V = 4;

    //If a graph is bipartite, then it can be colored with
    //two colors, and no adjacent nodes have same color
    //We use 0 and 1 represent two colors
    //and -1 mean uncolored vertices

    boolean isBipartite(int G[][], int src){
        //At beginning all nodes are not colored
        int colorArr[] = new int[V];
        for(int i=0; i<V; i++){
            colorArr[i] = -1
        }
        //color the source node
        colorArr[src] = 1;

        Queue<Integer> q = new LinkedList<Integer>();
        q.add(arc);

        //Run all the maps
        while(q.size() != 0){
            int u = q.poll();

            for(int v=0 ;v<V; v++){
                //Set all the adjacent nodes to opposite color
                if(G[u][v] == 1 && colorArr[v] == -1){
                    colorArr[v] = 1 - colorArr[u]
                    q.add(v);
                }
                //If we cannot color adjacent node, then return false
                else if(G[u][v] == 1 && colorArr[u] == colorArr[v]){
                    return false;
                }
            }

        }

        return true;
    }
}
```

139. Word Break

Monday, February 27, 2017 9:02 PM

Given a **non-empty** string *s* and a dictionary *wordDict* containing a list of **non-empty** words, determine if *s* can be segmented into a space-separated sequence of one or more dictionary words. You may assume the dictionary does not contain duplicate words.

For example, given

s = `"leetcode"`,

dict = `["leet", "code"]`.

Return true because `"leetcode"` can be segmented as `"leet code"`.

UPDATE (2017/1/4):

The *wordDict* parameter had been changed to a list of strings (instead of a set of strings). Please reload the code definition to get the latest changes.

```
public class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        Set<String> set = new HashSet<String>();
        for(int i=0; i<wordDict.size(); ++i){
            set.add(wordDict.get(i));
        }

        //DP to track all the substring
        boolean[] f = new boolean[s.length() + 1];
        f[0] = true;

        //here is <=, and we start from 1
        for(int i=1; i<=s.length(); ++i){
            for(int j=0; j<i; j++){
                if(f[j] && set.contains(s.substring(j,i))){
                    f[i] = true;
                    break;
                }
            }
        }
        return f[s.length()];
    }
}
```

140. Word Break II

Monday, February 27, 2017 9:09 PM

Given a **non-empty** string *s* and a dictionary *wordDict* containing a list of **non-empty** words, add spaces in *s* to construct a sentence where each word is a valid dictionary word. You may assume the dictionary does not contain duplicate words.

Return all such possible sentences.

For example, given

s = `"catsanddog"`,

dict = `["cat", "cats", "and", "sand", "dog"]`.

A solution is `["cats and dog", "cat sand dog"]`.

UPDATE (2017/1/4):

The *wordDict* parameter had been changed to a list of strings (instead of a set of strings). Please reload the code definition to get the latest changes.

- To
- To
- Dif
- Co

```
public class Solution {
    public List<String> wordBreak(String s, List<String> wordDict) {
        Set<String> set = new HashSet<String>();
        for(int i=0; i<wordDict.size(); ++i){
            set.add(wordDict.get(i));
        }
        return DFS(s, set, new HashMap<String, LinkedList<String>>());
    }

    // DFS function returns an array including all substrings derived from s.
List<String> DFS(String s, Set<String> wordDict, HashMap<String, LinkedList<String>>map) {
    if (map.containsKey(s))
        return map.get(s);

    LinkedList<String>res = new LinkedList<String>();
    if (s.length() == 0) {
        res.add("");
        return res;
    }
    for (String word : wordDict) {
        if (s.startsWith(word)) {
            List<String>sublist = DFS(s.substring(word.length()), wordDict, map);
            for (String sub : sublist)
                res.add(word + (sub.isEmpty() ? "" : " ") + sub);
        }
    }
    map.put(s, res);
    return res;
}
}
```

K continuous 1 to 0 in M*N matrix

Monday, February 27, 2017 9:18 PM

一个m x n 的 array 只有 0 和 1 给一个 int k
需要把 小于 k 数量 连续的 1 变成 0
连续： 上下左右和四个斜线方向
面试官是个中国女孩

DFS:

```java
//K conitnuous 1 to 0 in M*N matrix
class Solution{

    public static void merge(int[][] points, int k){
        int m = points.length;
        int n = points[0].length;
        boolean[][] visited = new boolean[m][n];

        int[] dx = {-1, -1, -1, 0, 0, 1, 1, 1}
        int[] dy = {-1, 0, 1, -1, 1, -1, 0 ,1}

        for(int i=0; i<m; i++){
            for(int j=0; j<n; j++){
                if(points[i][j] == 1 && !visited[i][j]){
                    Queue<int[]> queue = new LinkedList<>();
                    queue.add(new int[]{i,j});

                    ArrayList<int[]> record = new ArrayList<int[]>();
                    record.add(new int[]{i,j});

                    while(!queue.isEmpty()){
                        int[][] point = queue.poll();
                        int x = point[0];
                        int y = point[1];
                        visited[x][y] = true;

                        for(int k1=0; k1<8; k1++){
                            int xx = x+dx[k1];
                            int yy = y+dy[k1];
                            if(xx>=0 && xx<m && yy>=0 && yy<n && points[xx][yy] == 1 && ! visited[xx][yy]){
                                queue.add(new int[]{xx,yy});
                                record.add(new int[]{xx,yy});
                            }
                        }
                    }

                    if(record.size()<k){
                        for(int[] p: record){
                            points[p[0]][p[1]] = 0;
                        }
                    }

                    record.clear();
                }
            }
        }

    }

}
```

221. Maximal Square

Tuesday, February 28, 2017 11:38 PM

Given a 2D binary matrix filled with 0's and 1's, find the largest square containing only 1's and return its area.

For example, given the following matrix:

```
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

Return 4.

Credits:

Special thanks to @Freezen for adding this problem and creating all test cases.

```java
public int maximalSquare(char[][] a) {
    if(a.length == 0) return 0;
    int m = a.length, n = a[0].length, result = 0;
    int[][] b = new int[m+1][n+1];
    for (int i = 1 ; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if(a[i-1][j-1] == '1') {
                b[i][j] = Math.min(Math.min(b[i][j-1] , b[i-1][j-1]), b[i-1][j]) + 1;
                result = Math.max(b[i][j], result); // update result
            }
        }
    }
    return result*result;
}
```

29. Divide Two Integers

Saturday, February 4, 2017 8:29 PM

Description | Submission | Discussion | Add to List

Divide two integers without using multiplication, division and mod operator.

If it is overflow, return MAX_INT.

Subscribe to see which companies asked this question.

This seems to be a very difficult question with whole lot of restriction, but if you break down the question and tackle every sub-problem individually, it might not seem as hard as it is.

First, we need to check whether the end result is positive or negative. Two cases will lead to negative case which is when dividend and divisor has different signs. Alright, first part is done.

Second part is to deal with overflow. You know, there are so many corner cases which will lead to overflow. So it is better to convert them to long first, and convert it back to integer when returning the value.

Alright, now we get to the main part. We know that division is actually the backward of multiplication, for example , `20 / 5 = 4` can be seen as `4 * 5 = 20` . Here what we are going to do is to find the multiplication. We set tmp as divisor `(5)` and set count to `1` . As long as the tmp is less than or equal to dividend `(20)` , we left shift `<<` **which is same as multiply 2 but without using multiplication**.

```
1st loop --- tmp = 10 , count = 2
2nd loop --- tmp = 20, count = 4
3rd loop --- tmp = 40, count = 8 (exit the loop)
```

Now we right shift both tmp and count by 1, which gives us result of `4` . After subtraction of `20` from dividend, which gives us dividend = `0` and that we break out the outer loop and get to the last part.

Finally, we gotta check if the sign is positive or negative. If it is negative, then we apply negation `~result + 1` (two's complement) to get the negative result (why not just `result * -1` ? Well, critics might say you use multiplication -_-!!! (lol jk). Also make sure to check if result is overflow, because you know, leetcode is pretty strict to corner cases as well.

Another example : `10 / 3 = 3`

```
1st outer loop
--------------

1st inner loop --- tmp = 6 , count = 2
2nd inner loop --- tmp = 12, count = 4 (exit the inner loop, result = 0 + (4 >> 1) = 2)

dividend = 10 - (12 >> 1) = 10 - 6 = 4 (4 >  divisor, so here we go second outer loop)

2nd outer loop
--------------

1st inner loop --- tmp = 6, count = 2 (exit the inner loop, result = 2 + (2 >> 1) = 3)

dividend = 4 - (6 >> 1) = 4 - 3 = 1( divisor > 1, exit outer loop, return result)
```

Credits to @HelloWorld123456. Here is the simplified code:

```java
public int divide(int dividend, int divisor) {
    boolean isNegative = (dividend < 0 && divisor > 0) || (dividend > 0 && divisor < 0) ? true : false;
    long absDividend = Math.abs((long) dividend);
    long absDivisor = Math.abs((long) divisor);
    long result = 0;
    while(absDividend >= absDivisor){
        long tmp = absDivisor, count = 1;
        while(tmp <= absDividend){
            tmp <<= 1;
            count <<= 1;
        }
        result += count >> 1;
        absDividend -= tmp >> 1;
    }
    return  isNegative ? (int) ~result + 1 : result > Integer.MAX_VALUE ? Integer.MAX_VALUE : (int) result;
}
```

~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.

```java
public class Solution {
    public int divide(int dividend, int divisor) {
        boolean isNegative = (dividend < 0 && divisor > 0) || (dividend > 0 && divisor < 0) ? true : false;
        long absDividend = Math.abs((long) dividend);
        long absDivisor = Math.abs((long) divisor);
        long result = 0;
        while(absDividend >= absDivisor){
            long tmp = absDivisor, count = 1;
            while(tmp <= absDividend){
                tmp <<= 1;
                count <<= 1;
            }
            result += count >> 1;
            absDividend -= tmp >> 1;
        }
        return  isNegative ? (int) -result : result > Integer.MAX_VALUE ? Integer.MAX_VALUE : (int) result;
    }
}
```

Longest arithmetic subsequence

Wednesday, March 1, 2017 2:55 PM

```java
//For unsorted array
class Solution{
    public int longestArithematicSubseq(int[] nums){
        int max_len = 2;
        int len = nums.length;

        HashMap<Integer, ArrayList<Integer>> num_idx = new HashMap<>();
        int[][] result = new int[len][len];
        for(int i=0; i<len; ++i){
            if(num_idx.containsKey(num[i])){
                num_idx.put(nums[i], new ArrayList<Integer>());
            }
            num_idx.get(nums[i]).add(i);
        }
        for(int i=1; i<len; ++i){
            int end = nums[i];
            for(int j=i-1; j>=0; --j){
                int begin = nums[j];
                int diff = end-begin;
                int target = begin - diff;
                if(num_idx.containsKey(target)){
                    int next = -1;

                    for(int k=num_idx.get(target).size()-1; k>=0; --k){
                        if(num_idx.get(target).get(k) < j){
                            next = k;
                            break;
                        }
                    }

                    if(next != -1){
                        result[j][i] = result[next][j] + 1;
                        max_len = Math.max(max_len, result[j][i]);
                    }
                }

                if(!result[j][i])
                    result[j][i] = 2;
            }
        }
        return max_len;
    }
}
```

Small typo in first loop

325. Maximum Size Subarray Sum Equals k

Monday, February 27, 2017 5:37 PM

Given an array *nums* and a target value *k*, find the maximum length of a subarray that sums to *k*. If there isn't one, return 0 instead.

Note:

The sum of the entire *nums* array is guaranteed to fit within the 32-bit signed integer range.

Example 1:

Given *nums* = `[1, -1, 5, -2, 3]`, *k* = `3`,

return `4`. (because the subarray `[1, -1, 5, -2]` sums to 3 and is the longest)

Example 2:

Given *nums* = `[-2, -1, 2, 1]`, *k* = `1`,

return `2`. (because the subarray `[-1, 2]` sums to 1 and is the longest)

Follow Up:

Can you do it in O(*n*) time?

```java
public class Solution {
    public int maxSubArrayLen(int[] nums, int k) {
        int sum = 0, max = 0;
        HashMap<Integer, Integer> map = new HashMap<>();
        for(int i=0; i<nums.length; ++i){
            sum += nums[i];
            if(sum == k) max = i+1;
            else if(map.containsKey(sum-k)){
                //Here it is i-map.get(sum-k) not i-map.get(sum-k)+1
                //Because i start from 0 so we do not need to +1
                max = Math.max(max, i-map.get(sum-k));
            }
            //Here the idea is if sum(0,i) - k = sum(0,j)
            //Then sum(j,i) = k
            if(!map.containsKey(sum)) map.put(sum, i);
        }
        return max;
    }
}
```

523. Continuous Subarray Sum

Tuesday, February 28, 2017 11:40 AM

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**.

Example 1:

```
Input: [23, 2, 4, 6, 7],  k=6
Output: True
Explanation: Because [2, 4] is a continuous subarray of size 2 and sums up to 6.
```

Example 2:

```
Input: [23, 2, 6, 4, 7],  k=6
Output: True
Explanation: Because [23, 2, 6, 4, 7] is an continuous subarray of size 5 and sums up to 42.
```

Note:

1. The length of the array won't exceed 10,000.
2. You may assume the sum of all the numbers is in the range of a signed 32-bit integer.

```java
public class Solution {
    public boolean checkSubarraySum(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<Integer, Integer>();
        map.put(0, -1);
        int runningSum = 0;
        for (int i=0;i<nums.length;i++) {
            runningSum += nums[i];
            if (k != 0) runningSum %= k;
            Integer prev = map.get(runningSum);
            if (prev != null) {
                if (i - prev > 1) return true;
            }
            else map.put(runningSum, i);
        }
        return false;
    }
}
```

525. Contiguous Array

Tuesday, February 28, 2017 2:07 PM

Given a binary array, find the maximum length of a contiguous subarray with equal number of 0 and 1.

Example 1:

```
Input: [0,1]
Output: 2
Explanation: [0, 1] is the longest contiguous subarray with equal number of 0 and 1.
```

Example 2:

```
Input: [0,1,0]
Output: 2
Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.
```

Note: The length of the given binary array will not exceed 50,000.

The idea is to change `0` in the original array to `-1`. Thus, if we find `SUM[i, j] == 0` then we know there are even number of `-1` and `1` between index `i` and `j`. Also put the `sum` to `index` mapping to a HashMap to make search faster.

```
public class Solution {
    public int findMaxLength(int[] nums) {
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == 0) nums[i] = -1;
        }

        Map<Integer, Integer> sumToIndex = new HashMap<>();
        sumToIndex.put(0, -1);
        int sum = 0, max = 0;

        for (int i = 0; i < nums.length; i++) {
            sum += nums[i];
            if (sumToIndex.containsKey(sum)) {
                max = Math.max(max, i - sumToIndex.get(sum));
            }
            else {
                sumToIndex.put(sum, i);
            }
        }

        return max;
    }
}
```

477. Total Hamming Distance

Tuesday, February 28, 2017 2:27 PM

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Now your job is to find the total Hamming distance between all pairs of the given numbers.

Example:

```
Input: 4, 14, 2

Output: 6

Explanation: In binary representation, the 4 is 0100, 14 is 1110, and 2 is 0010 (just
showing the four bits relevant in this case). So the answer will be:
HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6.
```

Note:

1. Elements of the given array are in the range of `0` to `10^9`
2. Length of the array will not exceed `10^4`.

For each bit position 1-32 in a 32-bit integer, we count the number of integers in the array which have that bit set. Then, if there are n integers in the array and k of them have a particular bit set and (n-k) do not, then that bit contributes k*(n-k) hamming distance to the total.

```
public int totalHammingDistance(int[] nums) {
    int total = 0, n = nums.length;
    for (int j=0;j<32;j++) {
        int bitCount = 0;
        for (int i=0;i<n;i++)
            bitCount += (nums[i] >> j) & 1;
        total += bitCount*(n - bitCount);
    }
    return total;
}
```

```
public class Test {

   public static void main(String args[]) {
      int a = 60;       /* 60 = 0011 1100 */
      int b = 13;       /* 13 = 0000 1101 */
      int c = 0;

      c = a & b;        /* 12 = 0000 1100 */
      System.out.println("a & b = " + c );

      c = a | b;        /* 61 = 0011 1101 */
      System.out.println("a | b = " + c );

      c = a ^ b;        /* 49 = 0011 0001 */
      System.out.println("a ^ b = " + c );

      c = ~a;           /*-61 = 1100 0011 */
```

```
      System.out.println("a ^ b = " + c );

      c = ~a;           /*-61 = 1100 0011 */
      System.out.println("~a = " + c );

      c = a << 2;       /* 240 = 1111 0000 */
      System.out.println("a << 2 = " + c );

      c = a >> 2;       /* 15 = 1111 */
      System.out.println("a >> 2  = " + c );

      c = a >>> 2;      /* 15 = 0000 1111 */
      System.out.println("a >>> 2 = " + c );
   }
}
```

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

208. Implement Trie (Prefix Tree)

Tuesday, February 28, 2017 2:54 PM

Implement a trie with `insert`, `search`, and `startsWith` methods.

Note:

You may assume that all inputs are consist of lowercase letters `a-z`.

Hide Company Tags | Google | Uber | Facebook | Twitter | Microsoft | Bloomberg

Show Tags

Show Similar Problems

```
class TrieNode {
    public boolean isWord;
    public TrieNode[] children = new TrieNode[26];
    public TrieNode() {}
}

public class Trie {
    private TrieNode root;
    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode ws = root;
        for(int i = 0; i < word.length(); i++){
            char c = word.charAt(i);
            if(ws.children[c - 'a'] == null){
                ws.children[c - 'a'] = new TrieNode();
            }
            ws = ws.children[c - 'a'];
        }
        ws.isWord = true;
    }
```

```
    public boolean search(String word) {
        TrieNode ws = root;
        for(int i = 0; i < word.length(); i++){
            char c = word.charAt(i);
            if(ws.children[c - 'a'] == null) return false;
            ws = ws.children[c - 'a'];
        }
        return ws.isWord;
    }

    public boolean startsWith(String prefix) {
        TrieNode ws = root;
        for(int i = 0; i < prefix.length(); i++){
            char c = prefix.charAt(i);
            if(ws.children[c - 'a'] == null) return false;
            ws = ws.children[c - 'a'];
        }
        return true;
    }
}
```

398. Random Pick Index

Tuesday, February 28, 2017 3:07 PM

Given an array of integers with possible duplicates, randomly output the index of a given target number. You can assume that the given target number must exist in the array.

Note:

The array size can be very large. Solution that uses too much extra space will not pass the judge.

Example:

```
int[] nums = new int[] {1,2,3,3,3};
Solution solution = new Solution(nums);

// pick(3) should return either index 2, 3, or 4 randomly. Each index should have equal probability of returning.
solution.pick(3);

// pick(1) should return 0. Since in the array only nums[0] is equal to 1.
solution.pick(1);
```

- Total Accepted: **14961**
- Total Submissions: **36426**
- Difficulty: **Medium**
- Contributors: **Admin**

```
public class Solution {

    int[] nums;
    Random rnd;

    public Solution(int[] nums) {
        this.nums = nums;
        this.rnd = new Random();
    }

    public int pick(int target) {
        int result = -1;
        int count = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] != target)
                continue;
            if (rnd.nextInt(++count) == 0)
                result = i;
        }

        return result;
    }
}```
```

@yidongwang Actually it does.

```
public int pick(int target) {
int result = -1;
int count = 0; // to record how many targets in the array
for (int i = 0; i < nums.length; i++) {
if (nums[i] != target)
continue;
/*
For the nth target, ++count is n. Then the probability that rnd.nextInt(++count)==0 is 1/n. Thus, the probability that return nth target is 1/n.
For (n-1)th target, the probability of returning it is (n-1)/n * 1/(n-1)= 1/n.
*/
if (rnd.nextInt(++count) == 0)
result = i;
}
return result;
}
```

50. Pow(x, n)

Tuesday, February 28, 2017 3:32 PM

Implement pow(*x*, *n*).

Hide Company Tags | LinkedIn | Google | Bloomberg | Facebook

```java
public class Solution {
    public double myPow(double x, int n) {
        if (n == 0)
            return 1.;
        if (n < 0) {
            n = -n;
            x = 1. / x;
        }
        if (Double.isInfinite(x))
            return 0.;
        return (n % 2 == 0) ? myPow(x * x, n / 2) : x * myPow(x * x, n / 2);
    }
}
```

372. Super Pow

Tuesday, February 28, 2017 3:43 PM

Your task is to calculate a^b mod 1337 where *a* is a positive integer and *b* is an extremely large positive integer given in the form of an array.

Example1:

```
a = 2
b = [3]

Result: 8
```

Example2:

```
a = 2
b = [1,0]

Result: 1024
```

The main idea is cashed on the repeated pattern of the remainder of **a^b**.
As long as we know the length of the pattern **m**, we just have to find an index point of this pattern based on **b mod m**.
In addition, if **a > 1337**, we can let **a = a mod 1337**.
Because if we let a = (1337*x* + *c) where c = a mod 1337,*
*(1337*x + c)(1337*x + c)(1337*x + c)...(1337*x + c) mod 1337 == ccc*...c mod 1337.

```
public class Solution {
    int DIV = 1337;

    List<Integer> findLoop(int a){
        List<Integer> index = new ArrayList<>();
        boolean[] set = new boolean[DIV];
        int rem = a % DIV;
        while ( ! set[rem] ) {
            set[rem]=true;
            index.add(rem);
            rem = (rem*a) % DIV;
        }
        return index;
    }

    int modBy(int[] b, int m){
        int rem = 0;
        for (int i=0; i < b.length; i++) {
            rem = (rem*10+b[i]) % m;
        }
        return rem;
    }

    public int superPow(int a, int[] b) {
        if (a==0 || a==DIV || b==null || b.length == 0) return 0;
        if (a==1) return 1;
        if (a > DIV) return superPow( a % DIV, b);
        List<Integer> index = findLoop(a);
        int loopsize = index.size();
        int rem = modBy(b, loopsize);
        rem = rem==0? loopsize: rem;
        return index.get(rem-1);
    }
}
```

There are multiple ways:

- `String.valueOf(number)` (my preference)
- `"" + number` (I don't know how the compiler handles it, perhaps it is as efficient as the above)
- `Integer.toString(number)`

380. Insert Delete GetRandom O(1)

Tuesday, February 28, 2017 5:06 PM

Design a data structure that supports all following operations in *average* **O(1)** time.

1. `insert(val)` : Inserts an item val to the set if not already present.
2. `remove(val)` : Removes an item val from the set if present.
3. `getRandom` : Returns a random element from current set of elements. Each element must have the **same probability** of being returned.

Example:

```
// Init an empty set.
RandomizedSet randomSet = new RandomizedSet();

// Inserts 1 to the set. Returns true as 1 was inserted successfully.
randomSet.insert(1);

// Returns false as 2 does not exist in the set.
randomSet.remove(2);

// Inserts 2 to the set, returns true. Set now contains [1,2].
randomSet.insert(2);

// getRandom should return either 1 or 2 randomly.
randomSet.getRandom();

// Removes 1 from the set, returns true. Set now contains [2].
randomSet.remove(1);

// 2 was already in the set, so return false.
randomSet.insert(2);

// Since 2 is the only number in the set, getRandom always return 2.
randomSet.getRandom();
```

```java
public class RandomizedSet {
    ArrayList<Integer> nums;
    HashMap<Integer, Integer> locs;
    java.util.Random rand = new java.util.Random();
    /** Initialize your data structure here. */
    public RandomizedSet() {
        nums = new ArrayList<Integer>();
        locs = new HashMap<Integer, Integer>();
    }

    /** Inserts a value to the set. Returns true if the set did not already contain the specified element. */
    public boolean insert(int val) {
        boolean contain = locs.containsKey(val);
        if ( contain ) return false;
        locs.put( val, nums.size());
        nums.add(val);
        return true;
    }

    /** Removes a value from the set. Returns true if the set contained the specified element. */
    public boolean remove(int val) {
        boolean contain = locs.containsKey(val);
        if ( ! contain ) return false;
        int loc = locs.get(val);
        if (loc < nums.size() - 1 ) { // not the last one than swap the last one with this val
            int lastone = nums.get(nums.size() - 1 );
            nums.set( loc , lastone );
            locs.put(lastone, loc);
```

```
    if (loc < nums.size() - 1 ) { // not the last one than swap the last one with this val
        int lastone = nums.get(nums.size() - 1 );
        nums.set( loc , lastone );
        locs.put(lastone, loc);
    }
    locs.remove(val);
    nums.remove(nums.size() - 1);
    return true;
}
```

```
/** Get a random element from the set. */
public int getRandom() {
    return nums.get( rand.nextInt(nums.size()) );
}
```

381. Insert Delete GetRandom O(1) - Duplicates allowed

Tuesday, February 28, 2017 5:18 PM

Design a data structure that supports all following operations in *average* **O(1)** time.

Note: Duplicate elements are allowed.

1. `insert(val)` : Inserts an item val to the collection.
2. `remove(val)` : Removes an item val from the collection if present.
3. `getRandom` : Returns a random element from current collection of elements. The probability of each element being returned is **linearly related** to the number of same value the collection contains.

- Total Accepted: **11178**
- Total Submissions: **39505**
- Difficulty: **Hard**
- Contributors: **Admin**

Example:

```
// Init an empty collection.
RandomizedCollection collection = new RandomizedCollection();

// Inserts 1 to the collection. Returns true as the collection did not contain 1.
collection.insert(1);

// Inserts another 1 to the collection. Returns false as the collection contained 1. Collection now contains [1,1].
collection.insert(1);

// Inserts 2 to the collection, returns true. Collection now contains [1,1,2].
collection.insert(2);

// getRandom should return 1 with the probability 2/3, and returns 2 with the probability 1/3.
collection.getRandom();

// Removes 1 from the collection, returns true. Collection now contains [1,2].
collection.remove(1);

// getRandom should return 1 and 2 both equally likely.
collection.getRandom();
```

The follow-up: allowing duplications.
For example, after insert(1), insert(1), insert(2), getRandom() should have 2/3 chance return 1 and 1/3 chance return 2.
Then, remove(1), 1 and 2 should have an equal chance of being selected by getRandom().

The idea is to add a set to the hashMap to remember all the locations of a duplicated number.

```
public class RandomizedCollection {
    ArrayList<Integer> nums;
        HashMap<Integer, Set<Integer>> locs;
        java.util.Random rand = new java.util.Random();
    /** Initialize your data structure here. */
    public RandomizedCollection() {
        nums = new ArrayList<Integer>();
            locs = new HashMap<Integer, Set<Integer>>();
    }

    /** Inserts a value to the collection. Returns true if the collection did not already contain the specified elem
    public boolean insert(int val) {
        boolean contain = locs.containsKey(val);
            if ( ! contain ) locs.put( val, new LinkedHashSet<Integer>() );
            locs.get(val).add(nums.size());
            nums.add(val);
            return ! contain ;
    }
```

```
    /** Removes a value from the collection. Returns true if the collection contained the specified element. */
    public boolean remove(int val) {
        boolean contain = locs.containsKey(val);
            if ( ! contain ) return false;
            int loc = locs.get(val).iterator().next();
            locs.get(val).remove(loc);
            if (loc < nums.size() - 1 ) {
               int lastone = nums.get( nums.size()-1 );
               nums.set( loc , lastone );
               locs.get(lastone).remove( nums.size()-1);
               locs.get(lastone).add(loc);
            }
            nums.remove(nums.size() - 1);

            if (locs.get(val).isEmpty()) locs.remove(val);
            return true;
    }

    /** Get a random element from the collection. */
    public int getRandom() {
        return nums.get( rand.nextInt(nums.size()) );
    }
}
```

Mine sweeping with resovir sampling

Tuesday, February 28, 2017 7:14 PM

刚刚面完，感觉不是太好。
题目很简单，可惜不是面经。reservior sampling的题目
windows里面的扫雷，给一个h,w和m. 生成一个高度h，宽度w，总共m颗雷的矩阵。要求m颗雷随机分布。

第一个想法是把雷都放在前m个位置，从m+1的位置开始产生一个index小于m+1的位置，然后交换雷的位置。这一问写的磕磕绊绊，然后结果居然swap function 写错了，有个地方写太快把i写成了j。被指出来了之后很尴尬。

然后小哥又问了运行时间。说了是O(hw)。然后问能不能在O(m)时间内搞定。才想起来正确的reservior sampling的写法。

哎，就免了这一道题，感觉要挂了，求大米和祝福，谢谢！

```java
//Mine sweep reservior sampling
//h*m matrix with count mines
private int[][] bomb(int h, int w, int count){
    Random rmd = new Random();
    int[] nums = new int[h*w];

    int size = h*w;
    //Initialize the index arrray to
    //set each of the element become different
    for(int i=0; i<size; ++i){
        nums[i] = i;
    }

    //To save the position of mines
    int[] locations = new int[count];
    for(int i=0; i<count; ++i){
        locations[i] = nums[i];
    }

    //Reservior Sampling
    for(int i = count; i<size; ++i){
        int j=rmd.nextInt(i+1);
        //change the number in reservior
        if(j<count) locations[j] = nums[i];
    }
    //result matrix array
    int[][]res = new int[h][w];
    for(int i=0; i<locations.length; i++){
        int x = locations[i]/h;
        int y = locations[i]%w;
        res[x][y] = 1;
    }
    return res;
}
```

295. Find Median from Data Stream

Tuesday, February 28, 2017 7:53 PM

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

Examples:

`[2,3,4]` , the median is `3`

`[2,3]` , the median is `(2 + 3) / 2 = 2.5`

Design a data structure that supports the following two operations:

- void addNum(int num) - Add a integer number from the data stream to the data structure.
- double findMedian() - Return the median of all elements so far.

For example:

```
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
```

Java

```java
class MedianFinder {

    private Queue<Long> small = new PriorityQueue(),
                        large = new PriorityQueue();

    public void addNum(int num) {
        large.add((long) num);
        small.add(-large.poll());
        if (large.size() < small.size())
            large.add(-small.poll());
    }

    public double findMedian() {
        return large.size() > small.size()
                ? large.peek()
                : (large.peek() - small.peek()) / 2.0;
    }
};
```

152. Maximum Product Subarray

Tuesday, February 28, 2017 9:03 PM

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

For example, given the array `[2,3,-2,4]`,

the contiguous subarray `[2,3]` has the largest product = `6`.

```java
public class Solution {
    public int maxProduct(int[] A) {
        // store the result that is the max we have found so far
        int n = A.length;
        int r = A[0];

        // imax/imin stores the max/min product of
        // subarray that ends with the current number A[i]
        for (int i = 1, imax = r, imin = r; i < n; i++) {
            // multiplied by a negative makes big number smaller, small number bigger
            // so we redefine the extremums by swapping them
            if (A[i] < 0){
                int temp = imax;
                imax = imin;
                imin = temp;
            }

            // max/min product for the current number is either the current number itself
            // or the max/min by the previous number times the current one
            imax = Math.max(A[i], imax * A[i]);
            imin = Math.min(A[i], imin * A[i]);

            // the newly computed max value is a candidate for our global result
            r = Math.max(r, imax);
        }
        return r;
    }
}
```

209. Minimum Size Subarray Sum >= S

Tuesday, February 28, 2017 11:27 PM

Given an array of **n** positive integers and a positive integer **s**, find the minimal length of a **contiguous** subarray of which the sum ≥ **s**. If there isn't one, return 0 instead.

For example, given the array `[2,3,1,2,4,3]` and `s = 7`,

the subarray `[4,3]` has the minimal length under the problem constraint.

click to show more practice.

More practice:

If you have figured out the *O*(*n*) solution, try coding another solution of which the time complexity is *O*(*n* log *n*).

Credits:

Special thanks to @Freezen for adding this problem and creating all test cases.

```java
public class Solution {
    public int minSubArrayLen(int s, int[] nums) {
        if(nums.length == 0) return 0;
        int pos=0;
        int sum = 0;
        int length = Integer.MAX_VALUE;
        for(int i=0; i<nums.length; ++i){
            sum += nums[i];
            while(sum >= s){
                length = Math.min(length, i-pos+1);
                sum -= nums[pos];
                ++pos;
            }
        }
        return length == Integer.MAX_VALUE ? 0 : length;
    }
}
```

347. Top K Frequent Elements

Wednesday, March 1, 2017 12:21 AM

Given a non-empty array of integers, return the ***k*** most frequent elements.

For example,

Given `[1,1,1,2,2,3]` and k = 2, return `[1,2]` .

Note:

- You may assume *k* is always valid, 1 ≤ *k* ≤ number of unique elements.
- Your algorithm's time complexity **must be** better than O(*n* log *n*), where *n* is the array's size.

```
public List<Integer> topKFrequent(int[] nums, int k) {

        List<Integer>[] bucket = new List[nums.length + 1];
        Map<Integer, Integer> frequencyMap = new HashMap<Integer, Integer>();

        for (int n : nums) {
                frequencyMap.put(n, frequencyMap.getOrDefault(n, 0) + 1);
        }

        for (int key : frequencyMap.keySet()) {
                int frequency = frequencyMap.get(key);
                if (bucket[frequency] == null) {
                        bucket[frequency] = new ArrayList<>();
                }
                bucket[frequency].add(key);
        }

        List<Integer> res = new ArrayList<>();

        for (int pos = bucket.length - 1; pos >= 0 && res.size() < k; pos--) {
                if (bucket[pos] != null) {
                        res.addAll(bucket[pos]);
                }
        }
        return res;
}
```

The addAll function add list with no duplication.

28. Implement strStr()

Wednesday, March 1, 2017 12:26 AM

Implement strStr().

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

```
public int strStr(String haystack, String needle) {
  for (int i = 0; ; i++) {
    for (int j = 0; ; j++) {
      if (j == needle.length()) return i;
      if (i + j == haystack.length()) return -1;
      if (needle.charAt(j) != haystack.charAt(i + j)) break;
    }
  }
}
```

286. Walls and Gates

Wednesday, March 1, 2017 12:33 PM

You are given a *m x n* 2D grid initialized with these three possible values.

1. `-1` - A wall or an obstacle.
2. `0` - A gate.
3. `INF` - Infinity means an empty room. We use the value $2^{31} - 1 = 2147483647$ to represent `INF` as you may assume that the distance to a gate is less than `2147483647`.

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`.

For example, given the 2D grid:

```
INF  -1  0  INF
INF INF INF  -1
INF  -1 INF  -1
  0  -1 INF INF
```

After running your function, the 2D grid should be:

```
  3  -1   0   1
  2   2   1  -1
  1  -1   2  -1
  0  -1   3   4
```

```java
public class Solution {
    public void wallsAndGates(int[][] rooms) {
        if (rooms.length == 0 || rooms[0].length == 0) return;
        Queue<int[]> queue = new LinkedList<>();
        for (int i = 0; i < rooms.length; i++) {
            for (int j = 0; j < rooms[0].length; j++) {
                if (rooms[i][j] == 0) queue.add(new int[]{i, j});
            }
        }
        while (!queue.isEmpty()) {
            int[] top = queue.remove();
            int row = top[0], col = top[1];
            if (row > 0 && rooms[row - 1][col] == Integer.MAX_VALUE) {
                rooms[row - 1][col] = rooms[row][col] + 1;
                queue.add(new int[]{row - 1, col});
            }
            if (row < rooms.length - 1 && rooms[row + 1][col] == Integer.MAX_VALUE) {
                rooms[row + 1][col] = rooms[row][col] + 1;
                queue.add(new int[]{row + 1, col});
            }
            if (col > 0 && rooms[row][col - 1] == Integer.MAX_VALUE) {
                rooms[row][col - 1] = rooms[row][col] + 1;
                queue.add(new int[]{row, col - 1});
            }
            if (col < rooms[0].length - 1 && rooms[row][col + 1] == Integer.MAX_VALUE) {
                rooms[row][col + 1] = rooms[row][col] + 1;
                queue.add(new int[]{row, col + 1});
            }
        }
    }
}
```

348. Design Tic-Tac-Toe

Wednesday, March 1, 2017 3:20 PM

Design a Tic-tac-toe game that is played between two players on a *n* x *n* grid.

You may assume the following rules:

1. A move is guaranteed to be valid and is placed on an empty block.
2. Once a winning condition is reached, no more moves is allowed.
3. A player who succeeds in placing *n* of their marks in a horizontal, vertical, or diagonal row wins the game.

```
Given n = 3, assume that player 1 is "X" and player 2 is "O" in the board.

TicTacToe toe = new TicTacToe(3);

toe.move(0, 0, 1); -> Returns 0 (no one wins)
|X| | |
| | | |    // Player 1 makes a move at (0, 0).
| | | |

toe.move(0, 2, 2); -> Returns 0 (no one wins)
|X| |O|
| | | |    // Player 2 makes a move at (0, 2).
| | | |

toe.move(2, 2, 1); -> Returns 0 (no one wins)
|X| |O|
| | | |    // Player 1 makes a move at (2, 2).
| | |X|

toe.move(1, 1, 2); -> Returns 0 (no one wins)
|X| |O|
| |O| |    // Player 2 makes a move at (1, 1).
| | |X|

toe.move(2, 0, 1); -> Returns 0 (no one wins)
|X| |O|
| |O| |    // Player 1 makes a move at (2, 0).
|X| |X|

toe.move(1, 0, 2); -> Returns 0 (no one wins)
|X| |O|
|O|O| |    // Player 2 makes a move at (1, 0).
|X| |X|

toe.move(2, 1, 1); -> Returns 1 (player 1 wins)
|X| |O|
|O|O| |    // Player 1 makes a move at (2, 1).
|X|X|X|
```

Initially, I had not read the Hint in the question and came up with an O(n) solution. After reading the extremely helpful hint; a much easier approach became apparent. The key observation is that in order to win Tic-Tac-Toe you must have the entire row or column. Thus, we don't need to keep track of an entire n^2 board. We only need to keep a count for each row and column. If at any time a row or column matches the size of the board then that player has won.

To keep track of which player, I add one for Player1 and -1 for Player2. There are two additional variables to keep track of the count of the diagonals. Each time a player places a piece we just need to check the count of that row, column, diagonal and anti-diagonal.

Also see a very similar answer that I believe had beaten me to the punch. We came up with our solutions independently but they are very similar in principle.
Aeonaxx's soln

```
public class TicTacToe {
private int[] rows;
private int[] cols;
private int diagonal;
private int antiDiagonal;

/** Initialize your data structure here. */
public TicTacToe(int n) {
    rows = new int[n];
    cols = new int[n];
}

/** Player {player} makes a move at ({row}, {col}).
    @param row The row of the board.
    @param col The column of the board.
    @param player The player, can be either 1 or 2.
    @return The current winning condition, can be either:
            0: No one wins.
            1: Player 1 wins.
            2: Player 2 wins. */
```

```java
public int move(int row, int col, int player) {
    int toAdd = player == 1 ? 1 : -1;

    rows[row] += toAdd;
    cols[col] += toAdd;
    if (row == col)
    {
        diagonal += toAdd;
    }

    if (col == (cols.length - row - 1))
    {
        antiDiagonal += toAdd;
    }

    int size = rows.length;
    if (Math.abs(rows[row]) == size ||
        Math.abs(cols[col]) == size ||
        Math.abs(diagonal) == size  ||
        Math.abs(antiDiagonal) == size)
    {
        return player;
    }

    return 0;
}
```

158. Read N Characters Given Read4 II - Call multiple times

Wednesday, March 1, 2017 7:03 PM

The API: `int read4(char *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 *buf, int n)` that reads *n* characters from the file.

Note:

The `read` function may be called multiple times.

```
private int buffPtr = 0;
private int buffCnt = 0;
private char[] buff = new char[4];
public int read(char[] buf, int n) {
    int ptr = 0;
    while (ptr < n) {
        if (buffPtr == 0) {
            buffCnt = read4(buff);
        }
        if (buffCnt == 0) break;
        while (ptr < n && buffPtr < buffCnt) {
            buf[ptr++] = buff[buffPtr++];
        }
        if (buffPtr >= buffCnt) buffPtr = 0;
    }
    return ptr;
}
```

I used buffer pointer (buffPtr) and buffer Counter (buffCnt) to store the data received in previous calls. In the while loop, if buffPtr reaches current buffCnt, it will be set as zero to be ready to read new data.