**算法总共只有三种：**

Brute Force (DP all-path)

Divide and Conquer

Greedy (e.g. BFS)

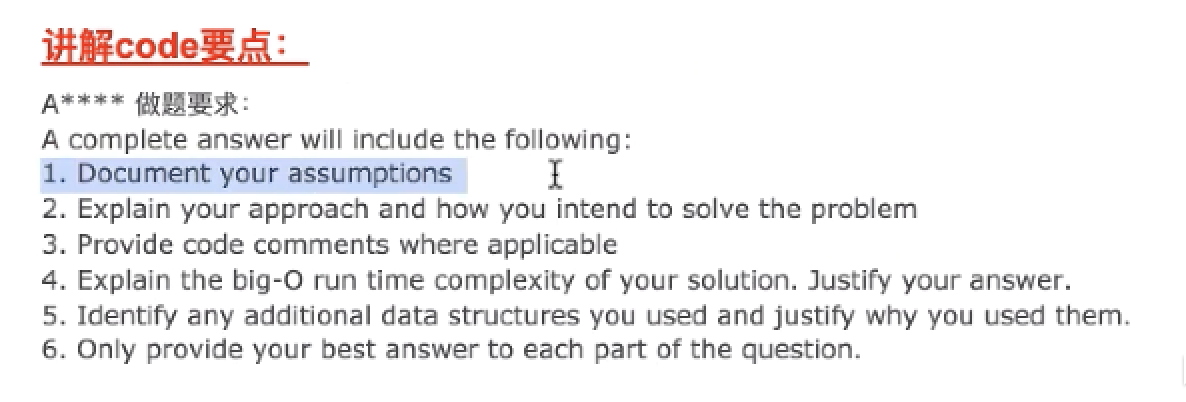
**Common Data Structures**

* Array
* Stack
* Queue
* Linked List
* Tree
* Heap
* Graph
* Hash Table

**Sorting Algorithms**

* Selection sort
* Insertion sort
* Merge sort
* Quicksort
* Bubble sort
* Bucket sort
* Patient sort
* Smooth sort
* Cocktail sort

...

****

**Class 1: Merge Sort and Quick Sort**

**Selection Sort**

基本原理：repeatedly (at each iteration) find the minimum element from the unsorted part and put it at the beginning. e.g. {-1, -3, 4, 7}

**Insertion Sort**

基本原理：

1) Compare the first two elements from the unsorted part, swap them if not in ascending order.

2) Keep swapping with the sorted part from right to left until it's in the right position.

e.g. {14, 33, 27, 10, 35, 19, 42}

**Merge Sort**

时间复杂度分析

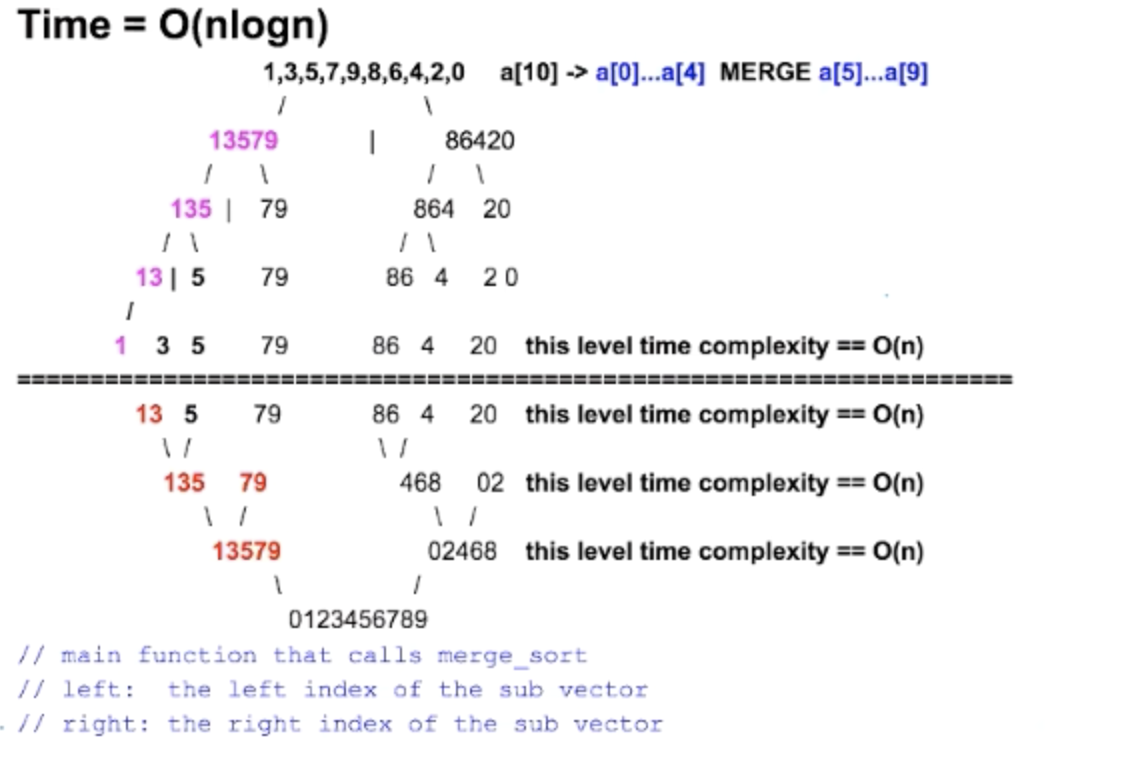
Divide (**log(N)**) 1 + 2 + 4 + … + N = 2N → above the line O(N) 一共分了N次 and

Conquer (**NlogN**) N + N + … = N \* depth = N\*logN → below the line O(NlogN)

改成 LinkedList 时间复杂度不变:

Divide (NlogN) and Conquer (NlogN)

space complexity: O(N)



**Quick Sort**

e.g. {1, 9, 8, 3, 5}

1. Partition (put pivot in the right position pivotPos)
   1. find pivot index, swap with the arr[end]
   2. i的左侧不包括i全部比pivot小
   3. j的右侧不包括j全部比pivot 大
2. quikSort left part (start, pivotPos-1)
3. quickSort right part (pivotPos+1, end)

**拓展：Rainbow Sort** aaaa...bbbbb..XXXX...cccc

**Class 2: Recursion and Binary Search**

**Recursion**

***基本思路：***

* base case （最小号问题）
* recursive rule: boil down a big problem to the same problem but smaller size

size N depends on size (N-1), size (N-2), …, etc.

**e.g. Fibonacci**

base case: F(0) = F(1) = 1;

recursive rule: F(N) = F(N-1) + F(N-2)

recursive tree:

**F(4)**  }

/ \ }

**F(3)**  F(2) }

/ \ / \ } **N 层**，每层数量double

**F(2)**  F(1) F(1) F(0) }

/ \ }

**F(0)**  F(1) }

time complexity: 1 + 2 + 4 + … + 2^(n-1) <= ***O(2^n)***

# of leaf nodes > # of all rest nodes combined

space complexity: maximum depth of ***call stack*** *(***记录recursion 每一层的local variable**）

**e.g. compute a^b**

base case: f(0) = 1

recursive rule: f(n) = f(n/2)\*f(n/2) if n is even, = f(n/2)\*f(n/2)\*a if n is odd

recursive tree:

b = 1000 }

| }

b = 500 }

| } **log(b) 层**

b = 250 }

| }

b = 125 }

…… }

time complexity: log(b), space complexity: log(b)

**Binary Search**

Classic version: find target in an array

left<=right, left = mid+1, right = mid -1;

2-D space: find target in a 2D matrix

binarySearch(0, m x n - 1);

check matrix[r][c] where r = n/col , c = n%col;

V1.1: find element in an array closest to the target. e.g. [1, 2, 3, 8, 9], target = 4

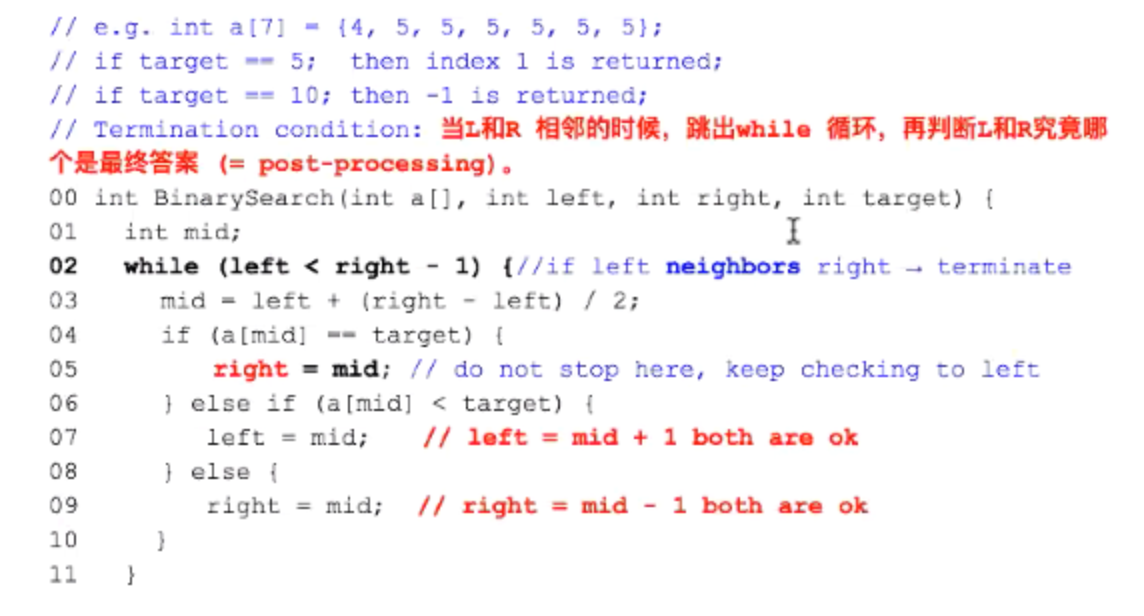
left+1<right, left=mid, right=mid

→ post-processing, check abs(nums[left]-target) and abs(nums[right]-target)

V1.2: return the index of the first occurance of an element.

left+1<right, left=mid or mid+1, right=mid or mid-1

→ pose-processing, if nums[left] == target return left else return right



V1.3: find the K elements closest to the target. e.g. [1, 2, 3, 8, 9], target = 4.

中心开花，先找到最近的两个，然后谁小移谁

**Binary Search V2.0**

Given a sorted dictionary with unkonwn size,

how to determine if a word is in the dictionary or not

Solution:

1. keep jumping 2^i steps until we jump out of the boundry
2. run binary search [0, 2^r] to find the solution

Follow up: How about jump 10 times each time?

**Class 3: Queue, Stack and Linked List**

描述一个复杂问题的时候，一定要成系统

**Queue**

典型问题：

* Tree level order traversal
* Sliding window
* Deque

**Stack**

Four popular interview questions

***Q1***: implement a queue using 2 stacks

Solution: push to stack1, pop from stack2. When stack2 is empty, move all elements from s1 to s2.

***Amortized*** time complexity: (2n+1+(n-1)\*1)/n = (3n)/n = O(1)

***Q2***: minStack

Sol: 同步加减

Follow-up: optimize s2 space assuming a lot of duplicates.

Sol: keep elements in S2 in decreasing order, store Pair<val, size>

s1|| **2** 2 2 2 **1** 1 1 2 1 3 4 1 1

s2|| {2, s1.size()==1}, {1, s1.size()==5}, …

***Q3***: simulate selection sort using two stacks

(unsorted) s1|| 1 3 2 4

(sorted) s2|| 1 2 3 4

Solution:

1. pop *all* elements from s1 and push to s2, record global\_min
2. pop elements **>=** global\_min from s2 and push elem *other than* global\_min back to s1
3. push global\_min to s2

*Follow-up* : what if there are duplicates ?

(unsorted) s1 || 3 1 2 2 2 4

(sorted) s2 || 1 2 2 2 3 4

Solution: keep a counter of global\_min and insert all of them into s2 at step (3)

***Q4***: implement deque using multiple stacks

Suppose after N pushes, S1 || 1, 2, 3, 4, 5, 6

1. pop 3 elements from S1 into S3, S1 || 1, 2, 3 S2 || S3 || 6, 5, 4
2. pop 3 elements from S1 into S2, S1 || S2 || 3, 2, 1 S3 || 6, 5, 4
3. pop 3 elements back from S3 to S1, S1 || 4, 5, 6 S2 || 3, 2, 1 S3 ||

pop() from S1 and popFront() from S2. Balance S1 and S2 whenever S2 is empty, which will happen after 0.5N times. Amortized time complexity: O(1)

**总结：什么时候用stack？**linear scan 回头看. e.g.

1) histogram 中最大正方形

2) reverse polish notation (a\*(b+c)) → abc+\*

3) string 的 repeately deduplication cabba → caa → c

**Linked List**

1. **check null pointer when de-referenced**
2. **never lose control of head pointer**

***reverse linked list*** (1:55:43)

Solution1: iterative using 3 pointers, prev, cur, next

Solution 2: recursive

base case: head == null || head.next==null, return head

除了subproblem 干了两件事：

1）head.next.next = head

2）head.next = null

注意最后return的是new\_head

常见考题：

***Q1***: find middle node of linked list 快慢指针(注意奇偶）

N1→ N2 → N3 → N4 → N5 → N6 → null

head tail head2

***Q2***: 判定linked list 是否有环？快慢指针相遇

***Q3***: insert a node in a sorted linked list

binary search: 用快慢指针找到mid需要O(n)时间, 不如linear scan

corner case: 一头一尾 1) insert 0 2) insert last

***Q4***: merge two sorted linked list

Sol: **dummy node**, 谁小移谁

When should we use dummy node ?

When we need to append new elements to an initially empty linked list without a head.

***Q5***: N1→ N2 → N3 → N4 → N5 → null 变成 N1→ N5 → N2 → N4 → N3 → null

Solution:

1. find the middle node of linked list
2. reverse the 2nd half
3. merge first half with the reversed second half

***Q6***: Partition list by target

1→ 6 → 3 → 2a → 5 →2b → null, target = 4

headSmall → 1 → 3→ 2a → 2b

tailSmall

headLarge → 6→ 5

tailLarge

tailSmall.next = headLarge.next;

tailLarge.next = **NULL**

**Class 4: Binary Tree & Binary Search Tree**

**Binary Tree**

Definition: at most two children

工业界应用：

social network analysis (Lowest Common Ancestor, 最早推荐人）

information indexing

information compression (serialization)

基本知识点1: tree traversal

(1)pre-order (2)in-order (3)post-order

trick: base case usually refers to null child nodes below the leaf node

基本概念：

1. balanced binary tree: depth of left subtree and right subtree differ by 1
2. complete binary tree: all levels except for the last is completely filled. All nodes are as far left as possible. Impl: heap
3. binary search tree (BST): for each nodes, all nodes in the left subtree are smaller and all nodes in the right subtree is larger than current node. Node values are *unique*.

Discussion:

* 每层node具备的性质，传递的值和下一层的性质，往往一致，容易定义 recursive rule
* Base case: null under leaf node
* Example 1: int getHeight(TreeNode node)
* Example 2: get total nodes

***Q1***: How to determine if a binary tree is balanced?

public boolean **isBalanced**(TreeNode root) {

if (root == null) return true;

if (!isBalanced(root.left) || !isBalanced(root.right)) {

return false;

}

int leftTreeHeight = depth(root.left);

int rightTreeHeight = depth(root.right);

return Math.abs(leftTreeHeight - rightTreeHeight) <= 1;

}

public int **depth**(TreeNode root) {

if (root == null) return 0;

int lDepth = depth(root.left);

int rDepth = depth(root.right);

return Math.max(lDepth, rDepth) + 1;

}

recursive method isBalanced() 嵌套了另一个 recursive method depth(). 时间复杂度分析：一共有log(n)层，每一层depth()所用时间是O(n), **总时间复杂度是 O(nlogn)**

isBalanced(root)

O(n/2) + O(n/2) = 2 x O(n/2) = O(n)

/ \

isBalanced(root.left) isBalanced(root.right)

O(n/4) + O(n/4) O(n/4) + O(n/4) = 4 x O(n/4) = O(n)

/ \ / \

… … = x O() = O(n)

***Q2***: How to determine if a binary tree is symmetric? 01:24

return helper(root.left, root.right);

boolean helper(TreeNode n1, TreeNode n2) {

if (n1 == null && n2 == null) return true; //base case 1

if (n1 == null || n2 == null) return false; //base case 2

if (n1.val != n2.val) return false; //base case 3

//如果当前层姑且满足条件，则调用下一层

return helper(n1.left, n2.right) && helper(n1.right, n2.left);

}

10

5a | 5b O(1) 一次比较

2a 3a | 3b 2b O(2) 两次比较

1a | 1b …...

…… ……

O(n/2)

Time = O(n) = 1 + 2 + 4 + … + n/2

Space = O(height), worst case O(n), O(log(n)) if balanced

***Q3***: If we tweak the lchild and rchild of an arbitrary node. How to determine if this tree is identical to another tree?

Case 1: Case 2:

8 8 8 8

/ \ / \ / \ / \

4 5 4 5 5 4 4 5

| | | |

7 7 7 7

public boolean isIdentical(TreeNode n1, TreeNode n2) {

if (n1 == null && n2 == null) return true; //base case 1

if (n1 == null || n2 == null) return false; //base case 2

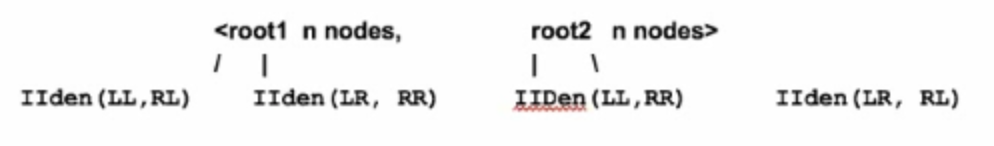
if (n1.val != n2.val) return false; //base case 3

return ((isIdentical(n1.left, n2.left) && isIdentical(n1.right, n2.right))

|| ((isIndentical(n1.left, n2.right) && isIdentical(n1.right, n2.left));

}

Time: O(n^2). The recursion tree is a quadral tree:



quadral height = original height = log\_2(n), since every node is only relevant to its next level.

*binary tree*: 1, 2, 4, 8… 1+2 < 4 = 2^h, 1 + 2 + 4 < 8 = 2^h, …

*quadral tree*: 1, 4, 16, 64, 256 ...

# of nodes on the last level = 4^(log2\_(n)) = 2^(2log\_2(n)) = O(n^2)

Space: O(height), worst case O(n), O(log(n)) if balanced

**Binary Search Tree**

***经典例题1***: How to determine a binary tree is BST?

Naive solution:

for each node, check left subtree and right subtree.

Time: each level takes O(n), total O(n)\*height, worse case O(n^2)

Better solution: O(n)

10 (-inf, +inf)

/ \

5 (-inf, 10) 15 (10, +inf)

/ \ / \

2 (-inf, 5) X(5, 10) 12(10,15) 20(15, +inf)

***经典例题2***: Print BST keys in the given range [k1, k2].

Naive Solution: inorder traversal

Better Solution: pruning

if (cur.val > k2) recurse(cur.left); //possible range

if (cur.val < k1) recurse(cur.right); //possible range

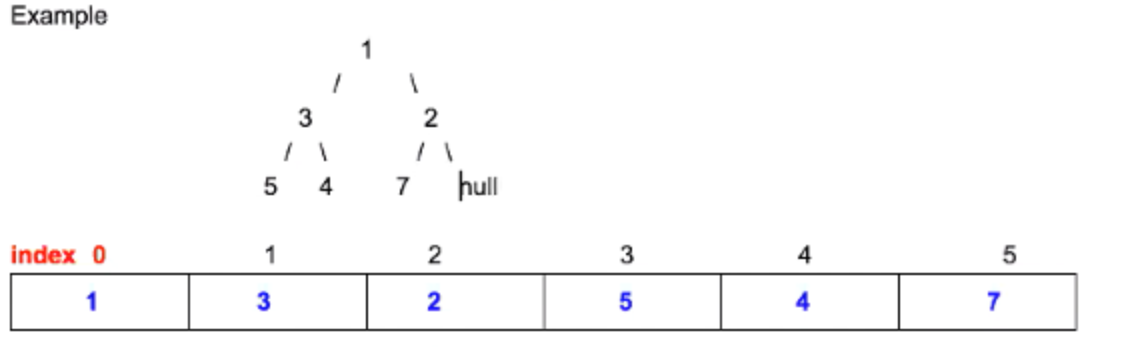
print node if (k1 <= cur.val <= k2)

Time: O(n) in worst case

**Class 5: Heap and Graph Search Algorithm I**

**Heap**

An unsorted array that follows special rules.



性质：堆的实现通过构造二叉堆(binary heap),

1. 任意节点小于所有它的孩子，最小元素为堆根（堆序性）
2. 堆是一课完全二叉树(Complete Tree), 只有最后一行有缺，且全部挤在左边
3. 根节点最大的堆叫 MAX HEAP, 根节点最小的堆叫 MIN HEAP
4. index of left child = index of parent \* 2 + 1
5. index of right child = index of parent \* 2 + 2
6. unsorted but follow rules above

基本操作: (N为heap中的元素个数）

1. insert: 先放最后，percolate up if smaller than parent O(logN)
2. update: percolate up/down (谁小和谁换) O(logN)
3. get/top: 获取当前堆顶元素值 O(1)
4. pop: 删除堆顶元素. 把最后一个换到根节点, percolate down O(logN)
5. heapify: turn unsorted array into a heap O(N) 不是 O(NlogN)

经典考题：

***Q1***: find the *smallest* k elements from an unsorted array of size N

k 和 N 的大小关系？如果 k = 6, N = 7, 直接 sort

Solution 1: sort and return first k elements (O(NlogN))

Solution 2:

1. heaptify in O(N) time
2. keep popping k elements O(klogN)

如果k比较小，N比较大的时候，Solution 2 好一些

Solution 3:

minHeap of size k

先把前k个元素进堆，不断更新那个最坏/大的那个元素

1. heaptify first k elements in O(k) time
2. iterate from k+1th to Nth element, if num(k+1) < heap.top(), update heap's top element in O((n-k)logk) time

如果 k <<< N，Solution 2 → O(c\*N), Solution 3 → O(Nlogk) it depends on c and logk

如果 k ~ N, Solution 2 → O(NlogN), Solution 3 → O(NlogN) when k = 0.5N in worse case

Solution 4:

**quick partition**

worst case: O(N^2), 每一轮都是 O(N)

average case : n + n/2 + n/4 + … + 1 < 2n, O(N)

**Graph**

Node

Edge

Directed vs. Undirected

Representation:

1. Adjacency matrix
2. Adjacency list
3. Adjacency set <key = node, value = a list of neighbors>

图里常用 Graph Search 算法：

**BFS-1 Breadth-First-Search**

**经典例题1**: print a binary tree level by level

**经典例题2**: check whether a graph is bipartite

从一个node开始expand，下一层必定和它不在一个set里，如果遇到conflict则不是bipartite

**经典例题3**: determine if a binary tree is a complete binary tree

没有泡泡. After detecting the first node that misses one child, if any of the following nodes has children, return false

**Discussion**:

1. when BFS-1? tree and graph-related and relationship on the same level

(同层亲戚关系）

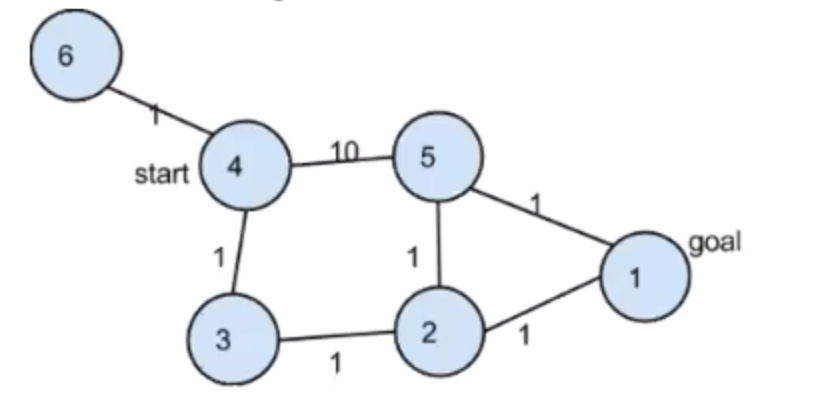
1. BFS-1 is NOT the algorithm to find the shortest path

**BFS-2 Best-first-search**

*区别：BFS-1, 谁先进谁先出；BFS-2, 谁小把谁弹出来*

Dijkstra (runtime improvement A\* search) 1:45:37

1. Application: find the shortest path cost (assuming all costs are non-negative) from single node to any nodes in graph (点到面)
2. Example: 从北京到中国其他所有城市的最短距离是多少
3. Data Structure: Priority Queue (MIN\_HEAP)
4. ***BFS-2 解题思路***：
   1. initial state (start / source node)
   2. expansion / generation rule
   3. termination condition
      1. 所有点计算完 (expanded.size() == V), 用 Java built-in 的 PriorityQueue
      2. PriorityQueue 为空，自己实现 Heap.update()
5. 性质：
   1. each node is expanded/popped once and only once
   2. a node can be generated more than once(update if cost is smaller)
   3. costs of nodes expanded/popped are monotonically non-decreasing
   4. time complexity is O(nlogn) where n is # of vertices
   5. when a node is expanded/popped from priority queue, its value is fixed and is equal to the shortest distance from the source node



* 1. enqueue (4, 0) → expand (4, 0) → enqueue (6, 1), (3, 1), (5, 10) → expand (6, 1) → expand (3, 1) → enqueue (2, 2) → expand (2, 2) → generate (5, 3) → expand (5, 3) → enqueue (1, 4) → expand (1, 4) → queue is empty

经典考题

Given a matrix whose row and column are sorted. Find kth smallest element.

**1** **2a** 3a 4 5

**2b** 3b 4 5 6

3c 4 5 6 7 (红色是弹出，蓝色为p\_queue中的)

4 5 6 7 8

5 6 7 8 9

BFS-2 三步骤：(每次从p\_queue里弹出的元素是单调递增的)

1. initial state: m[0][0]
2. expansion rule: generate right and down, m[i][j+1], m[i+1][j]
3. termination condition: when the kth element is popped from p\_queue.

time complexity: O(klogk)

there are k iterations

each iteration has two operations:

1. pop an element from p\_queue, heap.pop() is O(logk))
2. generate two neighbors, heap.insert() is O(logk)

space complexity: O(k)

Note: both 2b and 3b will generate 3b, it might be added to queue twice：

1, 2a, 2b, 3a, 3b, 3b, 3c, 4 …

Solution: use hash table to keep track of nodes that are generated.

**Class 6: Graph Search Algorithm II**

**(DFS / Backtracking)**

recall using preorder to traverse a tree

backtracking is just a behavior

为什么不能用BFS？最后一层size会很大，queue会爆掉

实现方法：easy to use recursion

Discussion: Can DFS only be implemented using recursion?

No. iterative using stack.

***DFS 基本方法***：

1. **How many levels? (**多少层?) L
2. **How many states for each level**? (每层分出多少叉来) N
3. Time Complecity : N ^ L

常见考题：

***DFS例题1.1***: print all subsets of a set S = {'a', 'b', 'c'}

Recursion tree:

{ }

/ (add 'a') \ (add "") level 1: add 'a' or not

{a} { }

(add 'b') / \(add "") / \ level 2: add 'b' or not

{a, b} {a} {b} { }

(add 'c') / \(add "") level 3: add 'c' or not

{a, b, c}

**How many levels?** (多少层?)n elements → n layers

**How many states for each level**? (每层分出多少叉来)

* two, select or not select

Time Complexity: O(2^n)

***DFS例题1.2***: print all permutations of a set S = {'a', 'b', 'c'}

Time Complexity: there're n! permutations, each will take O(n) to build, total O(n x n!)

***DFS例题1.3***: Given two integers n and k, return all possible combinations of k numbers.

Time Complexity: there're Cnk combinations, each will take O(k) to build, total O(k x Cnk)

***DFS例题2***: find all *valid* permutation using n pairs of parentheses given () () () ...

**How many levels?** (多少层？) n-level

**How many states for each level**? (每层分出多少叉来)

* two, `(` or `)`

Restrictions:

1. when could we place a left parenthesis `(` ??? if only there're still `(` left
2. when could we place a right parenthesis `)` ??? if only `(` added so far is more than `)`

***DFS例题3***: print all combinations of coins that can sum up to a total value k.

E.g. total value k = 99 cents

coin values: 25, 10, 5, 1 cents

Solution 1: total of 99 levels

Pros and Cons: easy to think but if k value is large, stack overflow, time: O(4^99)

root (99 cents)

/ | | \

level1 25(74 rem) 10(89) 5(94) 1(98)

/ | | \

leval 2 25(49) 10(64)

Solution 2:

**How many levels?** (多少层? 每层代表什么意义？）

* four layers, each level considers one type of coin

**How many states for each level**? (每层分出多少叉来)

* depends on how many coins needed to make up the remaining value at that level

root (99 cents)

/ | | \

level25 0\*25(99) 1\*25(74) 2\*25(49) 3\*25(24)

/ | \

level10 0\*10(99) 1\*10(64) .... 9\*10(9)

level5

level1

Pros and Cons: no stack overflow. Time: O(99^4) will increase very quickly.

sol[0], sol[1], sol[2], sol[3] = how many 25, 10, 5, 1 cents

void FindCombination(int money\_left=99, int level=0, int sol[]) {

//base case:

if (level == 3) {

sol[level] = money\_left; // one cent left

//print solution here

}

// money value on this level == coin[level]

for (int i=0; i<=money\_left / coint[level]; i++) {

sol[level] = i;

FindCombination(money\_left-coin[level]\*i, level+1, sol);

}

}

***DFS例题4***: Given a string with *no duplicated* letters, how to print out all permutations.

**How many levels?** (多少层?) three levels

**How many states for each level**? (每层分出多少叉来)

* remaining unused letters

(Swap-Swap, in-place)

root = abc

/ | \

level0 **a**(bc) **b**(ac) **c**(ba) swap(a,a), (a,b), (a,c)

/ \ / \ / \

level1 **A**b(c) Ac(b) Ba(c) Bc(a) Ca(b) Cb(a) swap(b,b), (b,c), ...

**// index is the current layer, swap all letters *after* with input[index]**

**void Permutation(string& input, int index) {**

**if (index == input.length()) {**

**cout << input << endl;**

**return;**

**}**

**for (int i=index; i<input.size(); i++) { //no-swap is also a solution**

**swap(input, index, i);**

**permutation(input, index+1);**

**swap(input, index, i); //swap back**

**}**

**}**

When should we consider SWAP-SWAP? Whenever every single permutation contains ***all*** elements in the initial input.

**DISCUSSION**:

1. BFS1 vs BFS2
2. BFS1 vs DFS When to use one or the other ?
3. Can we use BFS for perumtation problem, why not ?

**Class 7: Midterm**

***Question 1***: reverse a singly linked list (iterative + recursive)

***Question 2***: How to check wehther a binary tree is a binary search tree?

***Question 3***: Given a string without duplicate letters, print out all permutations.

***Question 4***: QuickSort a given int array.

**Class 8: Hash Table & String I**

**Hash Table**

1. hash\_map
2. principle
3. syntax: declear → int m;
4. hash collision
   1. chaining
   2. open address(probe + rehash)

***Q1 Find the top k frequent words from an essay.***

Hash table + priority queue

Solution

step1: iterate over and count freq for each word, store in hash table

step2: use a **min\_heap** with size k (introduced in Class 5)

***Q2 Find the only missing number from 1 to n in an unsorted array of size n-1 in O(n) time***

Solution:

M1: math (sum up and compare with n(n+1)/2), time = O(n), space = O(1), downside is overflow

M2: hash set

M3: xor (1..n) xor (arr[0] .. arr[n-2]) = missing number

bit operation (相同为零，不同为1) 性质：自己和自己XOR结果为零

***Q3 Find the common numbers between two sorted arrays***

Solution:

M1: binary search. For each element in the shorter array, search in the longer array.

Time: mlog(n), m <<< n

M2: store elements of shorter array in a hash set, check if elem in longer table is in hash set.

Time: O(m+n), Space: O(min(m,n))

M3: 谁小移谁 (in-place)

**String I**

**5种String基本操作：**（和array的有些问题相似，往往需要2个index完成操作）

1. **Removal** 
   1. remove some particular char from a string
   2. remove all leading/trailing/duplicated empty spaces from a string
2. **De-duplication** aaaabbbb\_ccc -> ab\_c
3. **Substring Searching/Matching** → strstr
   1. regular method
   2. Robin-Carp (hash based string matching) & KMP(Kruth\_Morris\_Pratt)
4. **Reversal(swap)** e.g. i love yahoo → yahoo love i
5. **Replacement** e.g. replace empty spaces "" with "20%"

Popular representations of characters: 1) ASCII A = 65, a =97. 2) Unicode

***1. Char Removal***

***Q1.1*** (Char removal) remove a/some particular chars from a string.

e.g. string input = "student", remove "u" and "n" → output = "stdet"

i = ***slow pointer***, all letters that are not u or n (results to return) should be put to the left of i

j = ***fast pointer***, increments by 1 every time

swap (str, i, j) if str[j] != u and str[j] != n

Time = O(n), Space = O(1)

***Q1.2*** (Char removal) normalization. Remove all leading/trailing and duplicate empty spaces (only leave one empty space if duplicated spaces happen) from the input string.

input = "\_ \_ \_ a b c \_ \_ e d \_ \_ e f \_ \_"

output = "abc\_ed\_ef"

(2个挡板，3个区域，同向而行）

1. stripping all leading /duplicate empty space
2. add only one empty space in front of each word (except for the 1st word)

void RemoveSpaces(string &s) {

int slow=0, fast=0;

int word\_count = 0; //special case for the 1st word

while (1 /\*true\*/) {

**// 1. skip all leading ' ' in front of *each* word**

while (fast < s.size() && s[fast] == ' ') fast++;

if (fast == s.size()) break;

**// 2. add ' ' in front of (2nd+) word**

if (word\_count > 0) s[slow++] = ' ';

**// 3. copy a word**

while (fast < s.size() && s[fast] != ' ') {

s[slow++] = s[fast++];

}

word\_count++;

}

s.resize(slow);

}

***2. Char De-duplication***

***Q2.1*** Remove duplicated and adjacent letters, leaving only one letter in each duplicated section. E.g. "aabb\_ \_cc" → "ab\_c"

index 0 1 2 3 4 5 6

string s a b \_ c \_ c c

s →

f →

s = ***slow pointer***. all the letters to its left(inclusive) should be returned

f = ***fast pointer***, increments by 1 every time

***Q2.2*** **(Char de-duplication adjacent letters *repeatedly*)** abbbbaz → aaz → z

*Method1: explicitly maintain a stack using vector*

use stack to check every char, if duplicated, pop out from the stack

linear scan 回头看用 stack. 为什么用vector 而不是stack ？顺序打印结果

void removeDuplicate(string& s) {

if (s.length()<=1) return;

vector<char> stack;

int i = 0; //fast pointer

while (i < s.size()) {

char c = s[i];

if (stack.size() > 0 && c == stack.back()) {

while (i < s.size() && c == s[i]) i++;

st.pop\_back();

} else {

st.push\_back(s[i]);

i++;

}

}

s.clear();

for (int j=0; j< st.size(); j++) {

s+= st[j];

}

}

*Method2: implicitly maintain a stack using vector*

s = ***slow pointer***, top element of the stack (result = [0, slow]) inclusive

f = ***fast pointer***, increments by 1 every time

Time = O(n), Space = O(1)

"abbbbaxxxxw" → "w"

0 1 2 3 4 5 6 7 8 9 10

w b b b b a x x x x w

s →

f→

***3. Substring → String Searching and Matching***

***Q3:*** Substring problem: how to determine whether a string is a substring of another string. Example: 0 1 2 3 4

s1 = a b c d e s2 = "c d"

return 2; return -1 if s2 is not in s1.

Solution 1:

int strmatch(String s1, String s2) {

if (s1 == null || s2 == null) return -1;

int i, j;

for (i=0; i<=s1.length()-s2.length(); i++) { //try every start index in s1

for (j=0; j<s2.length(); j++) { //for every letter in s2

if (s1.charAt(i+j) != s2.charAt(j) break;

}

if (j == s2.length()) return i;

}

return -1;

}

Solution 2:[**Rabin-Karp**](https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm)

s1 = "a b c d e"; s2 = "c d"; = 82

a b = 28 = 1\*26^1 + 2\*26^0

b c = 55 = 2\*26^1 + 3\*26^0

**c d = 82 = 3\*26^1 + 4\*26^0**

if we can hash s2 to a unique value, then scan s1 once and compare each substring's hashed value with s2's hashed value. Slide the window over text one by one, remove leading digit (s[i]), multiply by the highest degree, and add trailing digit (s[i+M]).

Time: O(n+m), Space: O(1)

Things to worry about:

* overflow: hashed value is too big to be represented by 64 bit

application: to catch plagiaris

***4. String Reversal***

***Q4.1***: apple → elppa

iterative: swap i, j if not equal

recursion:

* base case: 1 or 0 letters remaining, i >= j
* recursive rule: reverse(i+1,..., j-1)

***Q4.2***: i love yahoo → yahoo love i

* Primitive way : stack
* Better way : 1) reverse each word; 2) reverse the whole sentence

***Q4.3***: abcdef → efabcd, shift the whole string to the right-hand side by k=2 positions

abcd | ef ef | abcd ⇒ Q4.2 Time : O(n)

**总结**: 如果只是word 之间顺序变化，word内部相对顺序不变，首选 **i love yahoo** trick

***5. Char Replacement***

Solution: student → stuXX (st1 = den → s2 = XX)

* *Case 1: if (s1.length() >= s2.length())*

s t u d e n t d e n t

*slow* pointer : all letters to the left (exclusive) are results to return

*fast* pointer : index to scan the string

* *Case 2 : if (s1.length() < s2.length())*

www.yahoo.com/?q=flower**\_**market#flower**\_**store\_ \_ \_ \_

← s

← f

1. scan input to find total occurances of "\_", k = 4
2. calculate the total length of resulting string = n + k\*(s2.length()-s1.length())
3. use 2 pointers in backward direction

**Class 9: String II**

**Advanced topics**

1. Shuffling e.g. ABCD1234 → A1B2C3D4
2. Permutation (use DFS)
3. Decoding/encoding aaaabcc → a4b1c2 (Run Length Encoding)
4. Sliding window using slow/fast pointers

4.1 Longest substring that contains only unique chars **abcd**a

1. Matching (\*.?)
2. etc.

***1. String Shuffling***

1.1 Forward direction: "A1B2C3D4E5" → "ABCDE12345"

A1B2C3 | D4E5

A1B2 | C3 | D4 | E5

A1 | B2 | C | 3 | D | 4 | E | 5

A | 1 | B | 2 | C | 3 | D | 4 | E | 5 ← base case

======================================

A1 | B2 | C3 D4 | E5

AB12 | C3 | DE45

ABC123 | DE45

ABCDE12345

1.2 Backward direction: "ABCDE12345" → "A1B2C3D4E5" (in place)

C1 C2 C3 C4

-----------------------------------------------------------------

AB | CD | 12 | 34

AB | 12 | CD | 34

A | B | 1 | 2 | C | D | 3 | 4

A | 1 | B | 2 | C | 3 | D| 4

***Critical details***: guarantee size of Chunk 1 == Chunk 3.

注意 n/2 == 7 (奇数情况）

index 0 1 2 3 4 5 6 7 8 9 0 1 2 3

**A B C** **D E F G** | **1 2 3** **4 5 6 7**

lm m rm

C1[0-2], C2[3-6] C3[7-9], C4[10-13]

size = 14, mid = left + size / 2 = 7

leftMid = left + ¼ \* size = 3

rightMid = left + ¾ \* size = 10

**void convert(char[] a, int left, int right) {**

**if (right - left <= 1) return;**

**int size = right - left + 1; // how many elements in the section**

**int mid = left + size / 2;**

**int leftMid = left + size / 4;**

**int rightMid = left + size \* 3 / 4;**

**// DE | 123 → 123 | DE**

**reverse(a, leftMid, mid-1); // i love yahoo trick is here**

**reverse(a, mid, rightMid-1);**

**reverse(a, leftMid, rightMid-1); // DE123 → 123DE**

**// A B C 1 2 3 | D E F G 4 5 6 7**

**// 2\*(leftMid- left) 是蓝色部分的长度，减一表示index**

**convert(a, left, left + 2\*(leftMid- left)-1); // A B C 1 2 3**

**convert(a, left + 2\*(leftMid-left), right); // D E F G 4 5 6 7**

**}**

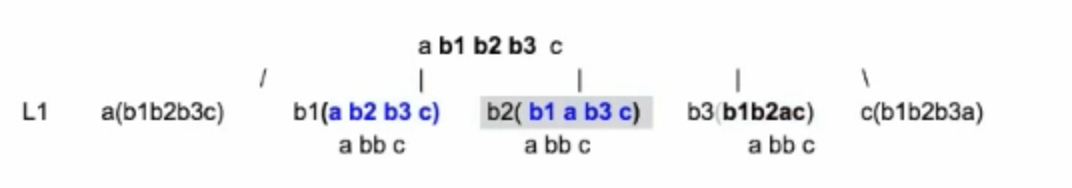
(这道题写出来了，对recursion的理解，对边界条件的选取，对数学的运算，向下取整，又到了一个更深层次的理解）

***2. String Permutation***

***Q2.1*** Have no duplicate letters in input string (see swap-swap)

E.g. input string "abc", output "aab", "aba", "baa"

***Q2.2***Have duplicate letters in input string



(去重, 保证同一个母节点下面，没有相同的letter 被swap到最左端）

**public void permutation(char[] c, int index) {**

**if (index == c.length) return;**

**// same letter may not necessarily be successive**

**Set<Character> dup = new HashSet<>();**

**for (int i = index; i < c.length; i++) {**

**if (!dup.contains(c[i]) {**

**dup.add(c[i]);**

**swap(c, index, i);**

**permutation(c, index+1);**

**swap(c, index, i);**

**}**

**}**

**}**

***3. String En/De-coding***

"aaaaazbbbwcc" → "a5z1b3w1c2" (in-place)

Run Length Encoding (压缩算法）

Step 1: 从左往右，处理变短的, leaving z, w unchanged.

统计只出现过一次的字母个数. K = 2. **a 5 z b 3 w c 2**

Step 2: 从 slow 开始，预留 K 个位置，从右往左，处理变长的.

**a 5 z b 3 w c 2** \_ \_ **a 5 z 1 b 3 w 1 c 2**

← s

← f

***4. Sliding Window in a string (slow + fast indices)***

***Q4.1*** Longest substring that contains only unique char

Given a string, returns the length of the longest substring without duplicate characters.

"BDEFGADE" → "BDEFGA"

***Follow-up*** : longest substring that contains letters that appear <= K times (e.g. K = 5)

Solution: maintain a hash\_table that reflects the real-time counts of each character.

***Q4.2*** Find all anagrams of a substring S2 in a long string S1 (fixed-size window)

e.g. s2 = "aabc", s1 = "zzzzcdebcaabcyywwww"

***Solution***: maintain a hash map and a counter types\_of\_letter\_to\_match = 3

HashMap Initialization, <a, 2>, <b, 1>, <c, 1>

Update types\_of\_letter\_to\_match when map value changes between zero/non-zero;

Update result when types\_of\_letter\_to\_match is zero.

***Q4.3*** Given a 0-1 array, you can flip *at most* K '0's to '1's.

Please find the longest subarray that consists of all '1's.

01010101000011101**0**1010101**0**101010000100101, K = 4

**Solution**: find a sliding window that contains at most K zeros.

* When to move the R border ? When the counter of zeros <= K
* When to move the L border ? When the counter of zeros > K

**Class 10: Bits Operation**

**Bit Representation**

"1" 0000 0000 0000 0000 … 0001

"7" 0000 0000 0000 … 1110

int a = 0

unsigned int a = 0 1111111111 (32 x 1) == 2^32 - 1

注意最左边一位是符号位， 0 : positive number; 1 : negative number

Two's Complement:

对于负数 "-1" 1 变反加1

1 == 0000 0000 0000 … 0001

-1 变反 1111 1111 1111 … 1110

加一 1111 1111 1111 … 1111

**Bit Operations**

* & AND

1100 1110

& 1001 1000

================

1000 1000

* | OR
* ~ NOT ( 1 → 0 and 0 → 1 for each bit)
* ^ XOR (相同为0，不同为1. xor 1 相当于反转其本身，0 → 1, 1→ 0)
* << left shift (右侧补充零, 相当于乘以2)
* >> right shift (左侧补充最高位，相当于除以2）

if we have negative number, after right-shift by 1 bit, the sign will fill in. 11111…

**Common Questions:**

1. Given a number x, how to set x's k-th bit to 1 ? e.g. x = xxx **0**xxx, and k = 3;

int bit\_setter = 1; // 00000… 0001

bit\_setter = bit\_setter << k; // 00000… 1000

x = x | bit\_setter;

1. Given a number x, how to set x's k-th bit to 0 ? x = xxx **1**xxx, and k = 3;

int bit\_setter = 1; // 00000… 0001

bit\_setter = bit\_setter << k; // 00000… 1000

bit\_setter = ~bit\_setter; // 11111… 0111

x = x & bit\_setter;

***Question 1***: Determine whether a number x is a power of 2 (x = 2^n), n > 0

10000000 x

& 011111111 x-1

=================

00000000 x & (x-1) != 0 && x != 0

***Question 2*** : How to determine the number of bits that are different between two positive integers? e.g. x = 0101, y = 0111, there is 1 bit difference. (XOR)

**public int checkTwoNumbersBits(int a, int b) {**

**int count = 0;**

**// 初始化 终止条件 每次操作**

**for (unsigned int c = a ^ b; c != 0; c = c >> 1) {**

**count += (c &1);**

**}**

**return count;**

**}**

***Question 3*** : What happens if we assign a negative number to an unsigned integer?

A: compiling error B: nothing happens / **correct answer** -1

C: runtime error D: very large integer number that is totally unexpected.

***Question 4*** : determine whether a word contains only unique letters (no duplicates).

input = student, output = false. ***Follow up***: assume all ASCII codes.

**public boolean isUnique (String word) {**

**int mask = 0;**

**for (int i = 0; i < word.length(); i++) {**

**int pos = word.charAt(i) - 'a'; // if d, pos = 3**

**if ((mask >> pos) & 1 == 1) return false;**

**else mask = mask | ( 1 << pos); // 0000 … 1000**

**}**

**return true;**

**}**

**Terminology**: **bit vector**

作用: 空间缩小8倍，用boolean表示需要一个 byte (8 bits)

108-th bit / 32 = 3.x (row)

108-th bit % 32 = 12 (col) 从右向左, 0 - 31



public boolean isUnique(String word) {

int[] dic = new int[8];

for (int i = 0; i < word.length(); i++) {

int pos = word.charAt(i);

int row = pos / 32;

int col = pos % 32;

if ( (dic[row] >> col) & 1 == 1) return false;

else { dic[row] = dic[row] | ( 1 << col ); }

}

return true;

}

***Question 5*** : How to reverse all bits of a number ? e.g. 1010 xxxxxxxx0010 → 0100 … 0101

L R

**1**010 xxxxxxxx 001**0**

XOR 1 … 1 ← bit mask

=======================

**0** 010 … 001 **1**

**unsigned int reverseBits(unsigned int num) {**

**unsigned int = sizeof(num) \* 8; // total bits = # Bytes \* 8**

**for (int i = 0; i < n/2; i++) {**

**num = swapAPairOfBits(num, i, n-i-1);**

**}**

**return num;**

**}**

**unsigned int swapAPairOfBits(unsigned int x, int i, int j) {**

**unsigned int right\_bit = ( (x >> i) & 1);**

**unsigned int left\_bit = ( (x >> j) & 1);**

**// if the i-th bit and j-th bit are different**

**if ( left\_bit ^ right\_bit == 1 )**

**x ^= ( (1U << i) | (1U << j)); // XOR with 1 both bit L and R**

**return x;**

**}**

***Question 6*** : Given a number x, how to get the hexadecimal representation of the number in string type ? E.g. 29 ⇒ 0X1D = 1x16^1 + D x 10^0 = 16 + 13 = 29 , 0X15 VS 15

10 进制 = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 int a = 11;

16 进制 = 0 1 2 3 4 5 6 7 8 9 A B C D E F int a = 0X11;

if ( remainder <= 9 ) cur = ‘0’ + remainder;

else cur = 'A' + remainder - 10;

**Class 12: Recursion II**

Recursion 三部曲：

1. What do you expect from left/right subtree ?
2. What to do in current layer ?
3. What do you want to report ?

***1. Recursion 与计算的结合***

***Q1.1*** a ^ b

1. 0 as the denominator 2) ⅓ as an integer ?? or float 3) 0 ^ 0

public double power (int a, int b) {

if (a == 0 && b <= 0) return -1;

else if (b < 0) return 1 / (double) pow(a, b);

else return (double) pow(a, b);

}

public int pow(int a, int b) {

if ( b == 0 ) return 1;

int half\_res = pow(a, b/2);

if ( b % 2 == 0 ) {

return half\_res \* half\_res;

} else {

return half\_res \* half\_res \* a;

}

}

***2. Recursion 与1D 或 2D 的 array 结合***

***Q1.1***: 1D array： 二分法比较常见

1.1 MergeSort

1.2 QuickSort

***Q2.1***: 2D array 逐层递归 : 8 Queens → n queens

**Recursive Rule**: for the i-th queue on the i-th row, we must make sure the Qi does not conflict with all previous queens that have been placed on the board.

Time: O(8!)

***Q2.2*** : How to print 2D array in spiral order.

**1 2 3 4 5**  size1 = 5, offset = 0;

**16** **17 18 19**  **6** size2 = 5 - 2 = 3, offset = 1;

**15**  **24**  **25**  **20**  **7** size3 = 3 - 2 = 1, offset = 2;

**14**  **23 22**  **21**  **8**

**13 12 11 10**  **9**

**void sprialprint (int[][] a, int offset = 0, int size = 5, int val ) {**

**if (size <= 1) {**

**print matrix; //base case**

**return;**

**}**

**// upper-left corder : (0+offset, i+offset)**

**for (int i = 0; i < size - 1; i ++) { // upper row (1,2,3,4)**

**a[0 + offset][i + offset] = val ++;**

**}**

**for (int i = 0; i < size - 1; i++) // right column (5,6,7,8)**

**for ... // bottom row (from right to left)**

**for … // left column**

**// recursive rule**

**sprialprint(a, offset + 1, size - 2, val);**

**}**

***3. Recursion 与LinkedList 结合***

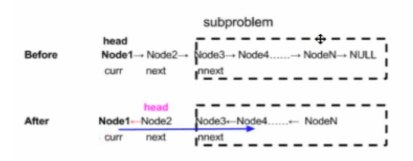
***Q3.1*** Reverse a linked list (Class 3 review)

***Q3.2*** Reverse a linked list (pair by pair).

Example: 1→ 2 → 3 → 4 → 5 → NULL

Output: 2 → 1 → 4 → 3 → 5 → NULL

要知道虚线框在谁脑袋/头顶上：



**public Node reverse (Node cur) {**

**if (cur == null || cur.next == null) return cur;**

**Node newHead = reverse(cur.next.next);**

**Node curHead = cur.next;**

**curHead.next = cur;**

**cur.next = newHead;**

**return curHead;**

**}**

***4. Recursion 与String 结合***

***Q4.1*** Reverse a string using recursion. abcd → dcba ( two pointers )

***Q4.2*** A word such as "book" can be abbreviated to 4, 1o1k, b3, b2k, etc. Given a string and an abbreviation, return if the string matches the abbreviation. e.g. "s11d" matches "sophisticated."

***5. Recursion 与 Tree 结合***

**5.1 第一类问题**：从下往上返值 (int, bool, etc.)

***Q5.1.1*** Review

int getHeight (Node root) {

if (root == null) return 0;

int left = getHeight(root.left); // breaking point

int right = getHeight(root.right);

return Math.max(left, right) + 1;

}

***Q5.1.2*** How to store how many nodes are in each node's left-subtree ?

class TreeNode {

TreeNode left;

TreeNode right;

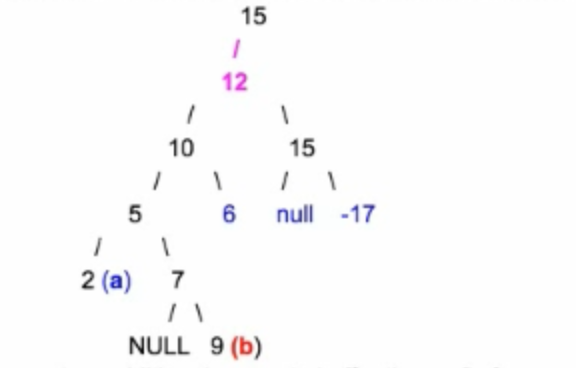
int val;

int total\_left;

}

***Q5.1.3*** Find the node with the max difference in the size of its left subtree and right subtree.

***Q5.1.4*** Lowest Common Ancestor (LCA)



Method 1 : use two additional arrays to buffer the prefix from root to a and b.

a : 15, 12, 10, **5**, 2

b : 15, 12, 10, **5**, 7, 9

Method 2 : recursion

*Follow-up* : What if a or b may not exist in the tree ?

**Class 13: Dynamic Programming**

DP 的核心思想类似于数学归纳法：

Recursion是从大到小解决问题(recursive rule)，DP是**从小到大**构造答案(induction rule).

DP 解决常用方法：

1. 一堆 original data(rope, word, a piece of wood), 求MAX或者MIN (cut, merge, …)
   1. if the weight of each smallest element in the original data is identical /similar
      1. identical : 1 meter of rope
      2. similar: a letter, a number

then this kind of problem is usually simple. E.g.

**linear scan 回头看 → 查表格**

**Longest Ascending Subarray** (when at i, look back at i-1)

Longest Ascending Subsequence (when at i, look back at 1, .., i-1)

**左大段 + 右小段**

**Maximal Product When Cutting Rope** (55:23)

**Dictionary Problem** (01:38:37)

**从右往左 scan 回头看**

**Jump Game** (01:44:43)

Cut Palindrome

* 1. if the weight are NOT the same
     1. DP1 课后题，沙子归并
     2. 强化练习题，切木头

从中心开花，[index = 0, 1, 2, 3, …, n-1], for each M[i, j],

we usually need to try out all possible k (i<k<j), s.t.

M[i,j] = max(M[i,k] +/-/\* M[k,j])

1. (TODO: 稍微复杂) 二维的 original data. E.g.
   1. two words 求 longest common substring
   2. 2D matrix 求最大 sub-matrix

***Examples***

***0. Fibonacci （linear scan 回头看2个）***

***1. Longest Ascending Subarray （linear scan 回头看1个）***

Given an unsorted array, find the length of the longest subarray in which the numbers are in ascending order. E.g. intput is {7, 2, 3, 1, 5, 8, 9, 6}, the expected output is 4. ({1, 5, 8, 9})

1. Base case M[0] = 1, one element
2. M[i] represents the value of the longest ascending subarray ending at the i-th element(inclusive). In the example, M[i] = 1, 1, 2, 1, 2, 3, **4**, 1
3. Induction Rule: M[i] = M[i-1] + 1 if nums[i] > nums[i-1]

= 1 if nums[i] >= nums[i-1]

***2. Maximal Product When Cutting Rope （linear scan 回头看所有）***

Given a rope with integer-length n, how to cut the rope into m integer-length parts with length p[0], p[1], … , p[m-1], in order to get the maximal product of p(0)\*p(1)\* … \* p(m-1) ? m is determnied by you and must be greater than 0 (at least one cut must be made).

**( 左大段 + 右大段 ) both 需要读表格**

**public int curRod (int n) {**

**int[] M = new int[n+1];**

**M[0] = 0; M[1] = 0;**

**for (int i = 1; i <=n; i++) { //possible rope len 1, 2,...,n**

**for (int j = 1; j <= i/2; j++) { //左大段所有可能性 j**

**int leftMax = Math.max(j, M[j]);**

**int rightMax = Math.max(i-j, M[i-j]); //右大段**

**M[i] = Math.max(M[i], leftMax \* rightMax);**

**}**

**}**

**return M[n];**

**}**

**( 左大段 + 右小段 ) 只有 left 需要读表格**

**for (int i = 1; i <=n; i++) { //possible rope len 1, 2,...,n**

**for (int j = 1; j < i; j++) { //左大段所有可能性 j**

**int leftMax = Math.max(j, M[j]);**

**int rightMax = Math.max(i-j); //右小段**

**M[i] = Math.max(M[i], leftMax \* rightMax);**

**}**

**}**

**Time** : O(N^2), for each length, loop though (n-1) cases.

**Space** : O(N)

***3. Dictionary Problem***

Given a word, check if it can be composed by concatenating words from a given dictionary ?

Example: Dictionary = { bob, rob, cat }

word1 = bcoabt → return false

word2 = bobcatrob → return true

word3 = bobbob → return true

input = b o b c a t r o b

size = 1 b M[1] = false

size = 2 b o M[2] = false

size = 3 b o b M[3] = true

size = 4 b o b c M[4] = false

size = 5 b o b c a

…

(左大段，右小段） **b** | **obca** (无法读表格)

***4. Jump Game （从右往左 scan 回头看）***

Given an array of non-negative integers, initially positioned at first index. Each element of the array represents the maximum jump reachable from that position. Determine if one is able to reach the last index.

e.g. 0 1 2 3 4

A = [ 2, 3, 1, 1, 4]

B = [ 3, 2, 1, 0, 4]

A = [2, 3, 2, 1, 4]

←T

Solution (DP)

1. Base case M[n-1] = true
2. Induction rule: M[i] represents whether we can jump to the target from the i-th index

M[i] = true iff there is an element j where j <= i + input[i], and M[j] = true

false o / w

Follow up: Return the minimum number of jumps needed to reach the end.

M[i] represents the min steps to jump from i-th index to the end.

**Class 14: Dynamic Programming II**

**Q1: Largest sum of subarray (15:17) most popular**

Given an unsorted array, find the subarray that has the greatest sum. Return the sum.

e.g. [1, 2, 4, -1, -20, 10, -1], the largest sum is achieved by [1, 2, 4, -1, -20, 10]

M[i] represents the largest subarray including i-th element between input[0, i]

M[i] = M[i-1] + nums[i] if M[i-1] > 0

= nums[i] if M[i-1] < 0

*follow-up 1*: can we do this in O(1) space ? use prev variable

*follow-up 2*: what if we want to return starting and ending index of the subarray ? use two pointers:

* when to move start: when M[i-1] is negative and we set M[i] = input[i]
* when to move end: end is always == i

**Q2: Dictionary word problem (47:47)**

**Q3: Edit Distance (01:17:45)** 二维 DP

Given the two strings of alphanumeric characters, determine the minimum number of ***Replace***, ***Delete*** or ***Insert*** operations to transform one string into another. E.g. s1 = "asdf", s2 = "sghj"

M[i][j] represents #min operations to transform s1[0..i] into s2[0..j]

M[i][j] = M[i-1][j-1] if s1[i-1] == s2[j-1]

= min( M[i-1][j-1] + 1, // replace

M[i-1][j] + 1, // delete

M[i][j-1] + 1 ) // insert

**Q4: Largest square of 1's in a binary matrix (01:52:41)**

What is the edge length of the largest square of 1's in a given binary matrix ?

In the following case, should return 3 x 3

0 0 0 0 0

1 1 1 1 0

1 1 1 1 0

1 1 1 0 0

1 1 ***1*** 0 0

bottom corner

M[i][j] represents the largest square of 1's with [i, j] as the bottom corner.

In order for the bottom corner `1` to form a square of 1's , it needs ***ALL*** of its up, left and diagonal 1's to form squares of 1's:

M[i][j] = 1 + min(M[i-1][j-1], M[i-1][j], M[i][j-1])

**Class 16: Dynamic Programming III**

**Question 1**: 一个unsorted 一维数组最长连续1的问题 (24 : 04)

01100111011101001010011111111

Base case: M[0] = input[0] only 1 element in the input

Induction rule: M[i] represents the longest contiguous 1s, including the i-th element

**Question 2a** (28:11): Given a matrix of 0's and 1's, how to find the *largest* cross with the same arm length and the two arms join at the central point of each other

0 ***1*** 0 0

***1 1 1*** 1

0 ***1*** 0 0

0 1 0 0

Essentually, solving the "longest contiguous 1's" (Question 1) for all four directions O(n2)

Cross 取决于最小的那支arm, min(Direction1[x][y], Direction2[x][y], …)

|  |  |  |  |
| --- | --- | --- | --- |
| Direction 1  (left to right) | Direction 2  (right to left) | Direction 3  (top down) | Direction 4  (bottom up) |
| 0 1 0 0  1 2 3 4  0 1 0 0  0 1 0 0  D1[x][y]表示从(x,y)向左延伸最长连续1的长度 | 0 1 0 0  4 3 2 1  0 1 0 0  0 1 0 0  D2[x][y]表示从(x,y)向右延伸最长连续1的长度 |  |  |

**Follow-up:** find the largest "X", that consists of all 1's. Solution: almost the same as Question 2a

**Question 3a** (39:17): Given a matrix with 1's and 0's, find the largest subsquare surrounded by '1'. (LC1139. Largest 1-Bordered Square) . E.g.

原矩阵 连续最长1矩阵(right to left)

[ 1 0 1 1 1

1 **1 1 1** 1 ← 5 4 3 2 1

1 **1** 0 **1** 0

1 **1 1 1** 1

1 1 1 0 0 ]

DP solution:

1. pre-processing for "longest contiguous 1's" for all directions
2. for each (x,y) as the top-left corner,

for all possible edge length(n, n-1, …)

O(1) check all four edges

Total Time: O(N3)

**Question 3b** (55:34) : Given a bunch of matches, find the largest square.

p00\_\_ p01\_\_ p02 \_\_p03 \_\_ \_\_ \_\_ \_\_

| |

p10 p11

| \_\_ \_\_ \_\_ |

| |

| \_\_ \_\_

class Point {

int right\_to\_left; //右边是否存在一个火柴棍

int bottom\_up; //下面是否存在一个火柴棍

}

**p00 p01 p02 p03** ...

(1,0) (1,0) (1,1)

**p10 p11 ...**

(0,1) (0,0)

**Question 4** (01:10:43) : Given an integer array A[N], there're repeated queries asking for the sum between A[i] and A[j], what should we do to speed up the query ? (i <= j)

index: 0 1 2 3 4 5 6 7 8

A[8]: 3 2 1 4 5 3 2 6

prefixSum: 0 3 5 6 10 15 18 20 26

Sum of A[i] to A[j] = prefixSum[j+1] - prefixSum[i]

Sum of A[2] to A[3] = prefixSum[4] - prefixSum[2] = 10 - 5 = 5

**int[] P = new int[N + 1];**

**for (int i = 0; i < N; ++i)**

**P[i+1] = P[i] + A[i];**

**Question 5** (01:16:52): **Given a matrix of integers (positive & negative numbers & 0's), how to find the submatrix with the largest sum ?**

* Solution 0: how many submatrix are there in the input matrix of size nxn ?

Cn2 X Cn2 = O(n4), each matrix sum takes O(n2), total time = O(n6)

* Solution 1 (1D prefix sum): compute the prefixSum in 1D. Then we can get the sum on each row in O(1) time. Each matrix sum takes O(n). Total time O(n5)
* Solution 2 (2D prefix sum): O(n4)

pre-processing prefix sum from (0, 0) to (i, j).

[0][0]

0 x x x x x x x

x x x x x x x x

[i][j] [i][k]

x x **X**  x **Z** x x x

x x **W** x **Y** x x x

[t][j] [t][k]

x x x x x x x x

sum of square XWYZ = sum[t][k] - sum[i-1][k] - sum[t][j-1] + sum[i-1][j-1]

**How to build 2D prefix sum** ? add prefix sum row by row

* [Optimal Solution](https://www.geeksforgeeks.org/maximum-sum-rectangle-in-a-2d-matrix-dp-27/): Kadane’s algorithm O(n3)

**Class 18: Probability, Sampling, Randomization, etc.**

**Class 20: Midterm II**

**Class 21: 加强练习 1**

**Class 23: 加强练习 2**

**Class 25: 加强练习 3 (Recursion III)**

**Class 26: 加强练习 4**

**Class 28: 加强练习 5**

**Class 31: 加强练习 6**

**Class 32: 加强练习 7**

**Class 34: 加强练习 8**

**Class 35: 加强练习 9**

**Class 36: Final Exam**