Keywords:

Array

String

Map  
Tree

DP

General：

#46 Permutations: give clear definition to your variable and method!!!!

## Min/Max Window

1. **Use** two pointers: **start** **and** **end** **to** represent a window.

2. **Move** **end** **to** find a valid window.

3. **When** a valid window **is** **found**, **move** **start** **to** find a smaller window.

## Dynamic Programming

**do back-up** to avoid duplicate computation!!!

**c++ pass vector by reference, and pass class by pointer**

pow(2, i) => 1 << i

use bit operation for times

## Brain Teasers

**Examplify, Simplify and Generalize(make x , y stand for specify number), Pattern Matching and Base Case and Build can be especially helpful.**

From high level to low level(examplify, base case and build), suppose and make assumption

From low level to high level(simplify and generalize)

suppose there are XXX

suppose it is like this XXX

# Array + DP:

nums[i] and nums[i-1]

or the answer is found among A[0], A[1], ….A[n-2],A[n-1]

# if you need to find several elements to satisfy a condition, maybe you can first set one of them and search for the others.

1\ use two pointers to dynamically find solutions.

Pointers could start from two endpoints or from the same side.

#16 3Sum closest #18 4Sum #209 Minimum Size Subarray Sum #15 3Sum #152 Max Product Array #11 Container With Most Water

2\ binary search

For unordered array, you may sort it first.

Or if solution is in a range, use binary search to find it.

#34 Search for a range

3\ stack

In general, hashtable and hashset is suitable for unordered search and insert. BST is good for ordered search and insert.

Especially, you can only search using int index in array. But in hash table, you can use string and other data structure as index.

Another useful usage of map and set is to store distinct element, or to find repeated element.

Big problem is consist of many similar subproblems. Solve the subproblems first and integrate their solutions to form result.

# Think about all the data structure and their APIs that you will use in your solution.

# Ignore the specific implementation of the data structure, only cares for its API.

# For the problem you met in MS, the data structure you should define include Point, Polygon. API of Point should include Point(x,y) first() second(). API for Polygon should include Polygon(Point[] arr), IsInside(Point x),etc

* Improve your code:
  + Make the repeated codes as a function.
* 没有思路时， 用recursion的思维想一想，不需要把细节弄得特别懂。
* 分解、整合：把大问题分解为平行的子问题，大问题的最优解由子问题的最优解整合得到的

例：#121 Best Time to Buy and Sell Stock #96 Unique Binary Search Tree #53 Maximum Subarray

* 动态规划：
* 定义状态和状态转移方程：一个状态变量（coin change，最长递增子序列），两个状态变量(背包问题，小偷问题，股票问题，二维矩阵，带条件的图的最短路径)
* 一个问题是由子问题的解得到，子问题和原问题在形式上相同，可以用数学表达式归纳。动态规划就是自顶向上或者自底向上，由前一个问题的解构建下一个问题的解。是一种思考问题的方法，不算是算法。
* 重要的优化： 做备忘录，以免重复计算同一个子问题的解。
* 许多最优化问题可以用dp得到。从最底层开始，某一个时刻有多个状态， 是由前一个时刻的多个状态得到的。重点是定义你的问题，并得到推导关系。如果推导关系得不到， 则换一种定义问题的方式试试。

推导n和n-1的关系，或者从1推导n（从最底层最简单的sample开始推导）。

例：#198 House Rob, #122 Best Time to buy and sell stocks, #337 House Robber III

#309 Best Time to buy and sell stocks with cooldown #62 Unique Paths #300 Longest Increasing Subsequence #313 Super Ugly Number #334 Increasing Triplet Subsequence #279 Perfect Sqaures #330 Patching Array #246 Ugly Number II #Jump Game #322 Coin Change

* 对于树的问题：bottom-up的方式想一想

例：#110 Balanced Binary Tree, # 337 House Robber III #124 Binary Tree Maximum Path Sum

* 对于树的问题：stack for iteration

例：#114 Flatten Binary Tree To Linked List #144 Binary Tree Preorder Traversal

* 抽象出数学问题，关键还是自己分析问题的能力。你要按照什么样的思路去分析这个问题。**要学会把大问题分解成小问题，或者找到问题解的结构的特点，或者发现递归关系**
* 树的遍历： 可分为DFS和BFS， DFS还可以分为recursion和iteration。recursion常常用helper function（可定义其它变量），iteration要借助stack（把一边的节点保存起来， 先处理另一边）或queue

例：#107 Binary Tree Level Order Traversal #144 Binary tree preorder traversal #199 Binary Tree Right Side View #170 Bianary Search Tree Iterator #114 Flatten Binary Tree to Linked List

#113 Path SumII

* Binary search
* Make use of what you already produced, build from the bottom

例：#89 Gray Code #Count Bits

* Backtracking:

例：#46 Permutations #216 Combination Sums III #77 Combinations #78 Subsets #39Combination Sum #131 Palindrome Partitioning

General Tips:

* use data structure: tuple dictionary
* think recursively;If possible, optimize recursion to iteration.
* Sort array first to simplify problems.
* Think out of the box, think from a different way.
* Two pointers in List problem.
* Do not fall into how to find a solution(process thinking), but focus on the structure and feature of the solutions. Separate the big problem into subproblems.
* What are the solution? How does it create? What are the structure and feature of this solution? What are all the possible solution? If they are in a range, then you can use binary search to find it.(#69 Sqrt(x))
* Dynamic Programming: DP问题的关键：**定义状态变量（函数）及状态（函数）间的推导关系。**找不到推导关系时， 换一种状态变量（函数）试试。
* **look for the local optimal solution from the very bottom, make use of what you already produce and get the next local optimal solution until global optimal solution is obtained.** For dynamic programming, if the opitimal solution is the sum of several state, using multi-dimension array to store the different states. For one-D problem, use [i,state], for two-D problem, use [i,j,state]. There are different ways to define the states of the problem. If one fails, change your definition and try another way.
* Backtracking: 在一个包含所有解的空间中， 按照某种规律系统地搜索满足问题条件的解， 逐一地把它们加入到解空间中。一旦遇到某个解不能满足问题的条件，则立即抛弃。
* Divide and Conquer: divide the original problems into subproblems. The global optimal solution is the summary of the subproblems’ optimal solutions.
* How to traverse a tree: you can use recursion which is trivial, and you can also use iteration with the help of queue or stack.
* For the tree problem, think about the bottom-up way to build your solution. Which is actually a recursive process, f(root) is connected to f(root.left) and f(root.right).
* Two dimension problem/ Two pointer problem: build a table for dynamic programming. The table should list all the potential solutions.
* 把问题做分解：如果一个问题的最优值只包含部分值， 则可以分解为包含i的最优值，包含j的最优值。。。在这些局部最优值中取最优就是全局最优值。
* **常见的解题思路：一个问题是基于前一时刻的问题得到的； 一个问题是多个子问题的整合得到的。**
* 换一个方向思考：反向的递归可以写为正向的迭代， 树的up-bottom也可以想成bottom-up。
* 抽象出问题，找到思路，尤其是最优化的问题（count change, perfect square）
* 定义很重要！！！
* for tree: 1\queue ***2\f(TreeNode, other variations)*** 3\stack
* Tree Traversal: use f(treenode, other variables) to control traversal order and process the node. Or push all the nodes in statck in your willing order first, then pop out and process.
* Do not fall into how to find a solution(process thinking), but focus on the structure and feature of the solutions.
* What are the variables in this problem? How to define the state of the problem? How to find the recursive relation between different state? How to divide a big problem into sub-problem and sum up the result of sub-problems will get the solution of big problem. There are different ways to define the state
* Think mathematically, then realize your solution using computer language.
* Partially determined, aggressively solve the whole problem.
* Reference to Permutation and CombinationIII
* Recursion to Iteration
* Different way to construct the same data. Think differently. Look from another way.
* When you do loop, optimize the stop condition.
* Recursion: 所消耗的内存是先增大后减少。
* Iteration: 会使用迭代变量， 在每次迭代中更新值， 最后一次迭代中变量所保存的值就是我们需要的。

#292 Nim Game

* deduction： from the simples example.
* Generilization: look for a must-win state

#258 Add Digits

* What are all the possible results?Is there any regulation in it? Just like alpha array, there are limited number of results.

#104 Maximum Depth of Binary Tree

#226 Invert Binary Tree

* Use queue to perform repeated actions for each node.

#283 Move zero

* Think out of the box: you don’t actually need to ‘move’ zeros.

#237 Delete Node in a Linked List

#100 Same Tree

* Use queue to operate on tree.
* Think recursively.

#242 Valid Anagram

* Learn how to use data structure like Dictionary\HashTable\HashSet\List\Array(when elements are limited) to solve problems.
* Difference between Dictionary & HashTable & HashSet

#172 Excel Sheet Column Number

#217 Contains Duplicate

#169 Majority Number

* Three possible ways to solve this problem.

**#206 Reverse Linked List**

* Recursive and iterative process.

**#235 Lowest Common Ancestor of a Binary Search Tree**

* Recursive and iterative process.

#191 Number of 1 Bits.

#83 Remove Duplicates from Sorted List

#70 Climbing Stairs

* In the iteration function f(a, others), if there is only one call of itself, you can use for-loop for replacement.

#263 Ugly Number

#202 Happy Number

#326 Power of Three

#231 Power of Two

**#121 Best Time to Buy and Sell Stock**

* Basic DP algorithm: analysis from the bottom, get the optimal solution of every single step(local optimal solution) and then get the global optimal solution.

**#21 Merge Two Sorted Lists**

* Recursive and iterative process.
* Careful when deal with linked list
* For list, consider the null reference situation in the beginning to avoid problems.
* Keep the head. Let other variables to tranversal.

#345 Reverse Vowels of a string

#24 Swap Nodes in Pairs

* recursive and iterative

#198 House Robber

* *For dynamic programming, if the opitimal solution is the sum of several state, using multi-dimension array to store the different states. For one-D problem, use [i,state], for two-D problem, use [i,j,state].*
* *Make problems concrete by listing examples, giving certain conditions(if…,so…) or specific situations. Think into details and detailed example.*
* If you want to opitimize the code, check if different states can be summed up.

#110 Balanced Binary Tree

* DFS: Depth First Search.
* *Bottom-Up search*, each node will be accessed only once. Once the subtree is not balanced, return the information.
* *Make problems concrete: for trees, think from the bottom leaves, then build all the way up.*

#232 Implement Queue using Stacks

#101 Symmetric Tree

* Think recursively.

**#107 Binary Tree Level Order Traversal II**

* BFS: using queue
* DFS: using helper function or stack
* You can insert new list in the front to avoid reversing in the end.

#27 Remove Element

#26 Remove Duplicates from Sorted Array

#66 Plus One

#118 Pascal’s Triangle

* Think recursively.
* Think iteratively.

#172 Factorial Trailing Zeros

**#103 Binary Tree Level Order Traversal**

* BFS: using queue
* DFS: using helper function.

#119 Pascal’s Triangle II

**#9 Palindrome Number**

* Corner case!

#112 Path Sum

#225 Implement Stack using Queues

**#111 Minimum Depth of Binary Tree**

* BFS: use queue. Dequeue all the nodes in the same level at one time.
* recursion.

#36 Valid Sudoku

* Limited situations: use array.

#160 Intersection of Two Linked Lists

#88 Merge Sorted Array

* Think out of the box.

#219 Contains Duplicate II

* Use set for distinct elements.

#223 Rectangle Area

* math

#205 Isomorphic Strings

#299 Bulls and Cows

#20 Valid Parentheses

#190 Reverse Bits

* Basic operation in bits:

n>>16 n<<16 n & 0xff00ff00 n|(n-1)

**#19 Remove Nth Node fom End of List**

* tow pointers.

#58 Length of Last Word

#290 Word Pattern

* hash table

#38 Count and Say

#203 Remove Linked List Elements

* Think recursively.

#257 Binary Tree Path

* Pre order traversal.

#234 Longest Common Prefix

#234 Palindrome Linked List

* Two pointers will be helpful in list.
* Don’t think too much.

#67 Add Binary

**#28 Implement strStr()**

* Take care of the stop condition.

#303 Range Sum Query-Immutable

**#204 Count Primes**

* Learn from the tutorial of how to improve this algorithm step by step.
* How to improve algorithm:

Elementary improvement

if any repeated computation.

#6 ZigZag Conversion

#125 Valid Palindrome

#7 Reverse Int

* How to judge if there are overflow?
* How to handle the overflow situation?
* Basci sense about the number expression in computer :
* 十六位二进制原码：带符号位---2^-16~2^16-1， 最高位符号位1为负0为正，其余位为数值位。011111…1111最大， 1111…1111最小。
* 补码：正数补码和原码相同， 负数补码除符号位以外的原码取反+1。10000..00为原码-1， 补码为100…00表示最小负数-2^16， 100…001表示-2^16+1。
* 注意计算机中数的表示范围：int 求绝对值或reverse以后可能超出了int表示的范围。应向上扩展。bool char/int/long/float/double。2^16=65536， 2^15=32768。

#1 Two Sum

#278 First Bad Version

**#155 Min Stack**

* keep track of local minimal.

#168 Excel Sheet Column Title

* Think clearly and do not mess.

#189 Rotate Array

* Do not only consider how to form a solution. Focus on the structure and feature of this solution and come up with different ways to create it.

#165 Compare Version Numbers

#8 String to Interger

#338 Counting Bits

* Dynamic programming: make use of what you already produced.

#136 Single Number

* Bit operation: XOR

#260 Single Number III

* Bit operation: XOR
* Be able to deduce step by step.

#238 Product of Array except Self

#122 Best Time to Buy and Sell Stock II

* Dynamic programming: Compare with #121, two different way to define the state of similar problems.

#319 Bulb Switcher

#268 Missing Number

#144 Binary Tree Preorder Traversal

* Recursion is trivial. Try iteration.
* When you traversal tree using iteration, queue and stack will be helpful.

#94 Binary Tree Inorder Traversal

* Use stack/queue for tree iterative traversal.

**#318 Maximum Product of Word Lengths**

* Learn to avoid duplicate calculation.
* Find key for your object just like #49 Group Anagrams.

**#328 Odd Even Linked List**

* More fluently.
* Write body first then consider list loop stop conditions.

#230 Kth Smallest Elements in a BST

* inorder traversal using recursion and iteration.
* **Binary search. Modify the structure of tree node.**

**#96 Unique Binary Search Trees**

* **Instantiate or make abstract math describtion.**
* Dynamic Programming.
* Tree

# 35 Search Insert Position

#108 Convert Sorted Array to BST.

#337 House Rob III

* *For bottom-up problems(DFS), you need recursion.*
* Depth-First Search.
* 定义问题： 即函数的签名 return-type f (x, other variations)。如果原问题不能很好的迭代， 则定义一个helper函数帮助解决问题。

#141 Linked List Cycle

#22 Generate Parentheses

* **Backtracking**: f(state, current answer, answer set)
* If(current answer good enough, add to answer set
* Add one more dimention to the current answer, change the state
* Recursively call f.
* Note: backtracking is building the answer dimension by dimension.

#309 Best Time to Buy and Sell Stock with Cooldown

#116 Populating Next Right Pointers in Each Node

**#53 Maximum Subarray**

**#89 Gray Code**

#62 Unique Paths

* Backtracking recursion VS Forward iteration

#153 Find Minimum in Rotated Sorted Array

#46 Permutations

#216 Combination Sum3

* For the above two, if you want to do it iteratively, you should find a way to generate them iteratively.

#59 Spiral Matrix II

#64 Minimum Path Sum

#75 Sort Colors

* Two pointers.

#48 Rotate Image

#11 Container With Most Water

* Draw a table for analysis.

#170 Bianary Search Tree Iterator

#240 Search a 2D Matrix II

**#199 Binary Tree Right Side View**

* DFS & BFS

#77 Combinations

* Backtracking: recursion
* *If you want to do it iteratively, think up a way to construct the final solutions.*

**#300 Longest Incereasing Subsequence**

* Dynamic Programming

#313 Super Ugly Number

#74 Search a 2D matrix

#73 Set Matrix Zeros

**#334 Increasing Triplet Subsequence**

* Pointers are helpful in array problem.

#215 Kth Largest Element in an Array

**#162 Find Peak Element**

* For search problem, trivial O(N) may be optimized to binary search O(lgN)

**#80 Remove Duplicates from Sorted Array II**

#129 Sum Root to Leaf Numbers

#279 Perfect Squares

**#81 Search in Rotated Sorted Array II**

#331 Verify Preorder Serialization of a Binary Tree

**#142 Linked List Cycle II**

#78 Subsets

#114 Flatten Binary Tree to Linked List

#39 Combination Sum

#90 Subsets II

**#213 House RobberII**

#109 Convert Sorted List to Binary Search Tree

#201 Bitwise AND of Numbers Range

#120 Triangle

#86 Partition List

#63 Unique Paths II

#147 Insertion Sort List

#34 Search for a Range

#106 Construct Binary Tree from Inorder and Postorder Traversal

#95 Unique Binary Search Trees II

#330 Patching Array

#236 Lowest Common Ancestor of a Binary Tree

#103 Binary Tree Zigzag Level Order Traversal

#105 Construct Binary Tree from Preorder and Inorder Traversal

#55 Jump Game

* Greedy algorithm: always find the best local optimal solution.

#113 Path Sum II

* Tree traversal: recursion or iteration using stack.

#47 Permutations II

#50 Pow(x, n)

* 考虑问题全面

**#264 Ugly Number II**

**#200 Number of Islands**

#131 Palindrome Partitioning

* Learn how to explain your code: give an example and draw picture to show what happens.

#40 Combination Sum II

#134 Gas Station

**#209 Minimum Size SubArray Sum**

* Two pointers for DP

#82 Remove Duplicates from Sorted ListII

#229 Majority Element II

#306 Additive Number

#69 Sqrt(x)

* When the solution is in a range, you can use binary search to find it.

#322 Coin Change

* Three methods:
* f(n) is based on the minimum of f(n-coin[0])…f(n-coin[1]). So you can do it iteratively or recursively.
* Backtracking: track all the possible coin changes and find the minimum one.

#148 Sort List

#228 Summary Range

#124 Binary Tree Maximum Path Sum

#91 Decode Ways

* **先定义好你的问题**， 考虑一下corner case的情况，不要着急

#343 Integer break

* 从最简单的情况推导，从中发现一些规律

#221 Maximal Square

* DP: 先定义好你的状态，再得出推广关系！

#96 Unique Binary Search Trees

#322 Coin Change & #264 Ugly Number II & #279 Perfect Squares

* 定义好你的问题，发现内部递推规律，从1推导到n

#Basic Calculator II

#345 Reverse Vowels of a string

* how to judge if an objects is in a set, for example, a char in a char array.
* Naïve solution is to look for this object in array(if not sorted, O(N), otherwise O(lgN))
* If the possible set elements is limited, build a boolean array for them and set relevant elements to true. So this object is as the index, this is O(1).

2016/07/01

#347 Top K Frequent Elements

* The basic task needed for this problem:
  + To count the frequency of an array. Map/dictionary is good for it.
  + To find the largest k elements from an array. Priority queue is good for it.
* Learn to use map and priority queue.
  + Map: insert, (count, find), erase, []
  + Queue: constructor

#357 Count Numbers with Unique Digits

* Probability method
* Backtracking for multi-dimension search.

#241 Different Ways to Add Parentheses

* DP solution: how to divide the original problem into subproblems.
* Optimization: take memo.

#367 Valid Perfect Square

* Binary Search: change the describtion of the original problem. From another view to see the problem, you may quickly get a breakpoint.

#289 Game of Life

#284 Peeking Iterator

#32 Longest Valid Parenthese

#368 Largest Divisible Subset

* Dynamic Programming:
  + 处理数组的最优化问题：由数组a[n]得到最优子序列
  + 由a[n-1]的最优子序列推导到a[n]的最优子序列，
  + 或者：由以a[i-1]结尾的最优解推导到以a[i]结尾的最优解

#341 Flatten Nested List Iterator

* Stack

#16 3Sum Closest

* Three problems: two+one(binary search), or one+two(two pointers, usually start from two endpoints)

#92 Reverse Linked List II

#310 Minimum Height Trees:

* Find some truth about the tree

#187 Repeated DNA Sequences:

* Repeated substring: find all substrings and sort

#183 Word Break:

* Redefine your problem.
* For array or string, go deep to each element. Define subproblem on each element.
* If string end with s[i] can be successfully broken, it means s[j:i] in set and s[0:j-1] can be broken.

#222 Count Complete Tree Nodes

#60 Permutation Sequence

#208 Implement Trie

* 初始化指针数组：TreeNode\* next[26], 此时next数组中的指针都没有被初始化，应手工赋值为nullptr。这一点和对象数组不同，对象数组会为每一个元素自动调用对象的默认构造函数

#332 Reconstruct Itinerary

#133 Clone Gragh

#18 4Sum

* 3sum 及以上的问题不能再用dictionary来做了，因为会出现重复的key

#43 Multiply Strings

* Keep a clear mind
* For big number, use string or int[] to store them

#150 Eval RPN

#93 Restore IP Address

* Keep a clean and organized mind. Do not make things too complicated

#2 Add Two Numbers

#5 Longest Palindromic Substring

* Do not complicate problems. Think from the base. Whether there exsits brute force solution.
* Expanded center: there are only 2n-1 centers.
* Dynamic programming, p(i,j) is false if substring (I,j ) is not palindromic, else is true.
* Longest common substring : substring can be contigous or not.
* 变量是什么！

#143 Reorder List

#61 Rotate List

#355 Design Twitter

* It’s all about1 getNewsFeed: using priority queue when popping out items according to an order.

#3 Longest Substrnig Without Repeating Characters

#304 Range Sum Query 2D-Immutable

#210 Course Schedule:

Bfs可以判断是否有cycle且同时给出topology order

Dfs 在确定没有cycle的情况下才可以给出topology order

#127 Word Ladder

* Bfs: try to model your problem using some known problems like dfs, bfs, binary search, prefix tree, etc!!!!

# Array

# Array + DP:

nums[i] and nums[i-1]

or the answer is found among A[0], A[1], ….A[n-2],A[n-1]

# if you need to find several elements to satisfy a condition, maybe you can first set one of them and search for the others.

1\ use two pointers to dynamically find solutions.

Pointers could start from two endpoints or from the same side.

#16 3Sum closest #18 4Sum #209 Minimum Size Subarray Sum #15 3Sum #152 Max Product Array #11 Container With Most Water

2\ binary search

For unordered array, you may sort it first.

Or if solution is in a range, use binary search to find it.

#34 Search for a range

3\ stack

#16 3Sum Closest

* Three problems:
  + two+one(binary search)
  + one+two(two pointers, usually start from two endpoints)

#216 Combination Sum III

* backtracking
  + build your solution dimension by dimension. Add all the possible element in current dimension and move to the next.

#209 Minimun Size Subarray Sum

* Two problem N:
  + Start from two endpoints and go to the middle.
  + Make one settled, and search another for optimal.
    - Start from index i, the optimal answer is …
    - End with index i, the optimal answer is ...
  + Dynamic modify two pointers.

#18 4Sum

* Settle two elements, and use two pointers to find another elements.

#31 Next Permutation

#162 Find Peak Element

* Two pointers

#153 Find Minimum in Rotated Sorted Array

* Binary search

#152 Maximum Product Subarray

* DP
* Math problem. Keep clear mind.

#268 Missing Number

* Bit operation: xor for duplicate.

#122 Best Time to Buy and Sell Stock II

#120 Triangle

#34 Search for A Range

* Two binary search

#90 Subsets II

* Backtracking

#289 Game of Life

#75 Sort Colors

* Pay attention to the quick sort method. It has many other application in array problems.

#74 Search a 2D Matrix

* Binary search

#73 Set Matrix Zeros

* Clear mind

#79 Word Search

* After finish your code, try to keep your code clean and elegant by defining function or something

#80 Remove Duplicates from Sorted Array II

* Keep good answer in a range and leave bad ones and ignore them

#81 Search in Rotated Sorted Array

* Clear mind

#64 Minimum Path Sum

* Basic DP

#54 Spiral Matrix

* Clear mind

#229 Majority Element II

* Clear Mind

Attention to good coding style, good naming, corner case caring, functionality

#15 3Sum

# Hash Table

Array: search is O(1), insert is O(N)

HashTable: search is O(1), insert is O(1)

HashSet: search is O(1), insert is O(1)

LinkList: search is O(N), insert is O(1)

BST: search is O(lgN), insert is O(lgN)

Hash table is very fast for lookup and search.

In general, hashtable and hashset is suitable for unordered search and insert. BST is good for ordered search and insert.

Especially, you can only search using int index in array. But in hash table, you can use string and other data structure as index.

Another useful usage of map and set is to store distinct element, or to find repeated element.

#290 Word Pattern

* Two hash tables

#49 Group Anagrams

#187 Repeated DNA Sequences

* Hash set for duplicate situations

#3 Longest Substring Without Repeating Characters

#355 Design Twitter

* Use hashtable or hashset to store information so that you can query them later.
* Use hash set for distict element, no duplicate case

#166 Fraction to Recurring Decimal

* Clear mind
* Hash table

# Linked List

Fake head to make problems simpler

runner two pointers

recursive way

#92 Reverse Linked List II

* Clear mind
* Coner case for link list problems.

#328 Odd Even Linked List

* Loop condition can be filled later.

#109 Convert Sorted List to Binary Search Tree

#2 Add Two Numbers

* Learn from it about how to handle list null situation.

#61 Rotate List

* More observation and thinking to find the best way to handle problems.

#82 Remove Duplicates from Sorted List II

* Try both recursion and iteration again
* The key is to keep in mind what you are trying to do, what all the variables mean
* 分情况讨论，不要先想把各种情况融合到一起，分开讨论再合并

#147 Insertion Sort List

#148 Sort List

#86 Partition List

# Math

* 发现一些原问题的数学性质！

#43 Multiply Strings

* Math: figure out the property of your model

#357 Count Numbers with Unique Digits

* Backtracking
* Probability problem

#368 Largest Divisible Subset

#60 Permutation Sequence

#313 Super Ugly Number

* If you know 1st-(n-1)th ugly number, how could you find the nth one?
* Deduce from the bottom, from some specified examples.

#367 Valid Perfect Square

* Overflow problems

#327 Super Pow

* Basic knowledge

# Two Pointers

#Two pointers for binary search

#Two pointers for array problem: build a 2D map, move two pointers from the same side or opposite side.

#16 3Sum Closest

#75 Sort Colors

#3 Lonegest Substring Without Repeating Characters

#18 4Sum

#11 Container With Most Water

#209 Minimum Size Subarray Sum

# String

#165 Compare Version Numbers

* Learn how to deal with string with different length, just like list/array with different length

#28 Implement strStr()

* Take care of the corner case, empty string, just like empty list/ array.
* Take care of when to stop loop: i < m-n not i < m
* More efficient algorithm

#14 Longest Common Prefix

#227 Basic Calculator II

#93 Restore IP Addresses

* Iterate all the possible combinations to find the correct answers.
* Use internal function to simplify

#91 Decode Ways

* DP: start from the simple situation! Do not make things too complicated!!

#5 Longest Palindromic Substring

* Stop condition in loop can be ahead than the last element of array/string
* Two way: center or dp

#151 Reverse Words in a String

# Binary Search

# if the answer is in a range [x, y], use binary search to find. Take case of overflow and boundary case. Or you can make range narrower first.

# if the answer is in [x, +inf], then start from x and times 2 each time to find the answer.

#34 Search for a range

* Good for binary search practise because you need to do two binary search and check the boundary.

#35 Search Insert Position

* Deal with corner case first will make your solution more clear.

#29 Divide Two Integers

* Another type of binary search.
* Take care of INT\_MIN INT\_MAX

# Tree

# The result of node depends on the result of left node and right node:::: use helper function and make the return value helpful. Examples include:

#110 Balanced Binary Tree

#337 House Robber III

#236 Lowest Common Ancestor of a Binary Tree

#98 Validate Binary Search Tree

#210 Course Schedule

* DFS and BFS solution for topological sort considering the cycle condition.

#133 Clone Graph

#114 Flatten Binary Tree to Linked List

* Stack for preorder traversal

#113 Path SumII

* In c# or python, list in the called function will change in caller function. Make copy if necessary.
* Lst1.CopyTo(Lst2, 0) a = b.copy() a = list(b)

#106 Construct Binary Tree from Inorder and Postorder Traversal

* If for each recusion only part of list needed, do not create a sub list, just use two pointers to indicate the valid data range.

#332 Reconstruct Itinerary

* Similar to topological order

#310 Minimum Heigth Trees

#230 Kth Smallest Element in a BST

* If you query some information many times, use constructor to build up necessary information.

#95 Unique Binary Search Trees II

* What’s your recursion part is ?

# Divide and Conquer

Big problem is consist of many similar subproblems. Solve the subproblems first and integrate their solutions to form result.

#215 Kth Largest Element in an Array

#53 Maximum Subarray

#241 Different Ways to Add Parentheses

* Use map for repeated subproblems.

# DP

#198 House Robber

#121 Best Time to Buy and Sell Stock

#300 Longest Increasing Subsequence

#91 Decode Ways

#95 Unique Binary Search Trees II

#309 Best Time to Buy and Sell Stock with Cooldown

#279 Perfect Squares

#357 Count Numbers with Unique Digits

#265 Ugly Number II

#221 Maximal Square

#213 House Robber II

#53 Maximum Subarray

#343 Integer Break

#152 Maximum product Subarray

#338 Counting Bits

#322 Coin Change

#368 Largest Divisible Subset

# Backtracking

#17 Letter Combinations of a Phone Number

* Use stack for combination problems OR recursion method
* Recursion method, no helper function

#22 Generate Parenthese

#39 Combination Sum

#40 Combination SumII

#216 Combination SumIII

#46 Permutations

* Permutations can be solved by swap

#131 Palindrome Partitioning

* Palindrome Problem: use dp to build a palindrome map isPalindrome[i][j]

#93 Restore IP Address

* Backtracking problem: add the possible answer of current dimension to the result and move to the next one.

#79 Word Search

#77 Combinations

* Some bp problems can be solved by iteration.

#78 Subset

#60 Permutations Sequence

# Stack

#Binary Tree Traversal

#Flatten Nested List Iterator

#Min Stack

# Heap

The usage of priority heap in finding the top k elements in a group

#Find top k smallest pairs

Two pointers in one or two arrays could be solved by building a matrix board.

# Bit Operation

#187 Repeated DNA Sequence

* Use bit operation and hashmap/hashset for distinct/repeated problems.
* Find a unique bit representation for each data. Store in map/set, and later you are able to find quickly whether there is a repeated one.
* When you find bit representation for original data, think about the data structure and compress it as much as possible.
* XOR operation for distinct element.

#Bit And Range

# Design

# Think about all the data structure and their APIs that you will use in your solution.

# Ignore the specific implementation of the data structure, only cares for its API.

# For the problem you met in MS, the data structure you should define include Point, Polygon. API of Point should include Point(x,y) first() second(). API for Polygon should include Polygon(Point[] arr), IsInside(Point x),etc

**# Tower of Hanoi**

* there are two data structure in this problem: Tower, Disk
* try to model them and use their API to solve problems.

#341 Flatten Nested List Iterator

* You should only care about two data structure:
  + Nested list and its API
  + Iterator and its API

# **Overflow**

long int judgement whether overflow before computation

INT\_MIN INT\_MAX 0

#367 Valid Perfect Square

* Overflow when you compute on ints like + \* -

#50 Pow(x, n)

* Overflow when abs(INT\_MIN)

#69 Sqrt(x)11

#7 Reverse Int

#8 String to Integer

# Hard

General solution:

1\ think about the process that you generate an answer, iterate the whole input, use data structure to help(map, stack, set, tree, etc), especially the tree: binary search tree, dictionary tree, prefix tree, etc

process thinking and run through all the data structures:

array? quick for query with index

linked list? quick for insert and delete

heap? goor for keep tracking of max/min

stack? queue?

tree? binary search

hash table? quick for insert and query

hash set?

2\if no process-thinking method, try dp, use recursive thinking

3\ divide and conquer. Split array into two parts, see if you can get ans for left part, right part and form answers.

Well Design the API first: think about data structure and most efficient way to represent data

0\ we have XXX, YYY, ZZZ in this problem. First let’s decide how to represent them: base data structure or design class? Tradeoff?

1\ what is data structure in this problem? Any need to create new class to represent data structure?

Point, Interval

2\ what’s the best data structure to represent data?

If not duplicate, set is better than vector.

If order does not matter, unordered\_set if better than set.

If possibility is limited, vector is better than map.

If the integer is only 0-9, use char to represent is more efficient than int. char is 1 byte.

The structure of solving back-tracking problem:

Vector<vector<T>> bp(T, some\_criteria)

Transform T to meet some criteria, return all possible transformation.

Usually use bp to solve this problem

Helper(T, current\_trans, idx, some\_criteria, ret)

If(idx comes to last and current\_trans meets criteria) ret.push\_back(current\_trans)

Add the next possible part into current\_trans(maybe additional info needed and you should collect info ahead.)

Move idx forward and continue to call helper.

If you find some situations are hard to satisfy, then discuss with your interviewee and see if you can ignore it first and concentrate on other parts.

Simplify it.

Examplify: if there are n …. suppose there are n….

make detailed example and see if you can derive a general rule from there.

or for a problem exsits many states, try to analyze a detailed situation instead of merge all the situations together and get a mess.

Pattern Match: learn from your past experience

Good code looks like:

1\ use data structure generously. Care about object-oriented.

2\ appropriate code reuse. Define general helper function for repeated ops.

3\ Modular: helps to outline your thoughts.

4\ flexible: work with more general situation if possible.

1\ ask questions:

problem size

**when deal with number** **: what type**, int/double? what range, 32bits? will it be **negative or zero**? will it have **overflow** so long int needed? Any other extreme\edges cases should be considered?

Do not assume int yourself.

when deal with character: how it is encode?

2\ design an algorithm

even if cannot design an algorithm , it is helpful to find some truth and facts

maybe a brute force way first

use function and pseudocode first to outline your thoughts clearly.

3\ code:

at a nice and slow pace, no need to rush

use data structures generously: find the minium age of people->design Person class first.

which shows you care about good object-oriented design.

4\ test:

extreme condition: 0, negative, null, min, max

user error: what happens if the user passes in null or a invalid value

general cases: test the normal case

Especially for tree:

1\ segment tree: to deal with interval/segment problem. In each index, you can store either sum, max, or min, etc.

2\ binary search tree and variants: organize data in a tree. fast to query some data in a range, like whether there is a target value, how many vals are in a range, how many vals are smaller than a number.

316 Remove duplicate number:

your analysis is good. But when you are stuck with some unknown situation, suppose you know it first, then continue to solve the problem. The unknown issue may become known through some preprocessing. It may be another subproblem.

#37 Sudoku Solve: solve bp problem like this.

Row by row, for each empty cell, try possible answer and move to the next cell. If return false, try the next possible answer.

bool Helper(i, j, board, additional\_info)

{

for k = 1:9

if(k is not ok) continue

fill board[i][j] with k

if(Helper(nexti, nextj, board)==true) return true;

unfill board[i][j] try next possible answer.

}

**#327 Count of Range Sum**

* divide and conquer
* when deal with number : what type ? what range ? overflow ? Do not assume int yourself.
* functionize: make more small functions to show your thinking path.

#87 Scramble String

* do not think about the process, think about its structure and use dp/recursive thinking to generate the answer.
* Make **memo, pre-checking** to speed up!

#99 Recover Binary Search Tree

* take time to analysis the problem!! maybe from different aspects!!!! carefully thinking!!

#164 Maximum Gap

* string and int sort:
  + First, ask how the character is encoded or what is the range of int.
  + radix sort: from lsb to msb
* bucket sort: put items into different buckets

#25 Reverse Nodes in k-Group

linkedlist常见的操作：reverse， switch等等

#239 Sliding Window Maximum

analyze the process. use specific examples to analyze how data should be dealt with, what should be done under different situations.

#115 Distinct Subsequences

string s, t ----- 2D dp

if only one string, try 1D dp.

memo = new int[s.Count(), t.Count()];

memo[i, j] indicates a subproblem.

memo[m-1,n-1] is the answer.

String:

1\ scan the string, use map/set/stack to store information and solve problems.

like

#43 Multiply strings

#17 Letter Combinations of a Phone Number

#227 Basic calculator

#385 Mini Parser

#93 Restore IP Address

#3 Longest Substring Without Repeating Characters

#71 Simplify path

#76 Minimum Window Substring

#32 Longest Valid Parenthese

2\ use dp to solve problem: if there are two string, 2D dp, use a matrix to store information. otherwise use an array.

Give clear definition to your matrix/array

#115 Distinct Subsequences

#5 Longest Palindromic substring

#91 Decode ways

#315 count smaller numbers in the right

when you are supposed to find some qualifed data in a collection, try to save this collection as binary search tree!!!!!

Store information in a tree(binary search) instead of an array(linear search)

when the problem is about the range(like how many number bigger than, or smaller than), you should think about sorted array and binary search.

Or you should think about BST.

#330 Patching Array

Analysis the process.

#301 Remove Invalid Parenthese:

when you get confused, may try to implement what you already have in you mind first. Implement a sub problem first.

**#312 Burst Balloons:**

**if you deal with array using dp, it seems like two pointers pointing to endpoint of the array, try to move the pointers and find if the subproblem can be contributed to the original problem.**

#375 Guess Number Higher or Lower II

use detailed example to analyse, find the repeated model in your problem(dp,similar subproblem)

use array or ND array to replace map

#287 Find the Duplicate Number

* Two problem NlgN:
  + Set one, and binary search for another one.
  + Or binary search for one, then scan all the array.

#85 Maximal Rectangle

* Find a solution first and then try to optimize it.

#42 Trapping Rain Water

#84 Largest Rectangle in Histogram

* The above two problems are not easy to see clearly.
* Stack to solve array problems.
* 从简单的情况开始，先抓住重点，再考虑一些边际情况。什么时候情况开始make sense and need some computation.
* Some thing is not that clear to see. Make assumption and detailed examples. Discussion under different situations. If…so, if…so,

#56 Merge Intervals

* sort(intervals.begin(), intervals.end(), [](Interval a, Interval b) { return a.start < b.start; });
* Pay attention to define struct and sort function.
* C++与C#的匿名函数, 委托与函数指针

#57 Insert Interval

#41 First Missing Positive

* Find some truth about original problem

#123 Best Time to Buy and Sell Stock III

* Very similar to #309 With Cool down
* Implement several correlated action in an array requires DP. To define the state, the i th index end with state j

#138 Copy List with Random Pointer

* Hash table

#76 Minimum Window Substring

* Repeat especially learn from the post

#23 Merge K Sorted Lists

* Using priority queue
* Learn from merge sort

#25 Reverse Nodes in K-Group