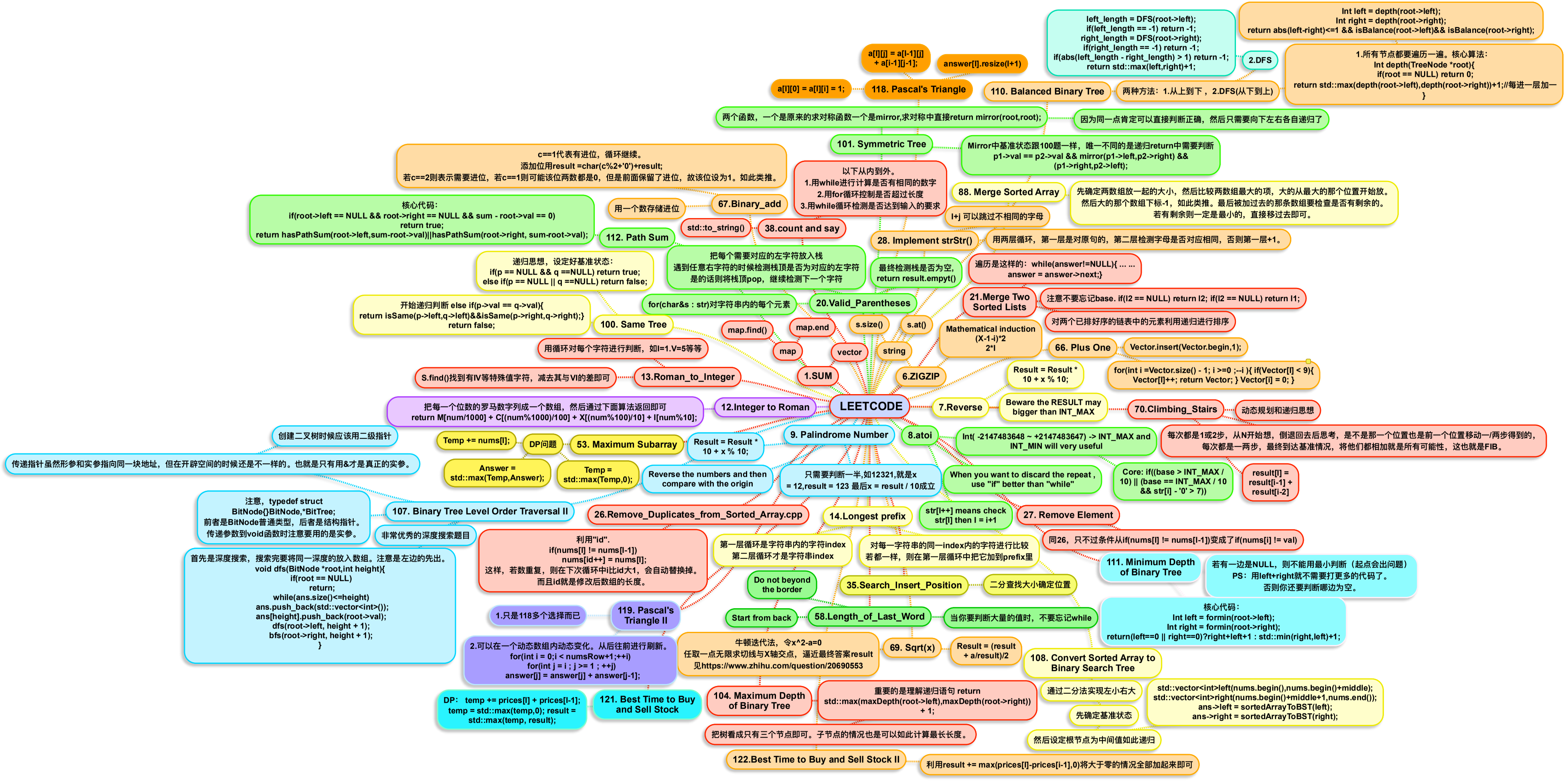
LEETCODE
1.SUM
2.DFS
6.ZIGZAG
7.Reverse
8.atoi
9. Palindrome Number
12.Integer to Roman
13.Roman_to_Integer
14.Longest prefix
20.Valid_Parentheses
21.Merge Two Sorted Lists
26.Remove_Duplicates_from_Sorted_Array.cpp
27. Remove Element
28. Implement strStr()
35.Search_Insert_Position
38.count and say
53. Maximum Subarray
58.Length_of_Last_Word
66. Plus One
67.Binary_add
69. Sqrt(x)
70.Climbing_Stairs
88. Merge Sorted Array
100. Same Tree
101. Symmetric Tree
104. Maximum Depth of Binary Tree
107. Binary Tree Level Order Traversal II
108. Convert Sorted Array to Binary Search Tree
110. Balanced Binary Tree
111. Minimum Depth of Binary Tree
112. Path Sum
118. Pascal's Triangle
119. Pascal's Triangle II
121. Best Time to Buy and Sell Stock
122.Best Time to Buy and Sell Stock II
map
vector
string
map.find()
map.end
s.size()
s.at()
Mathematical induction
(X-1 i)*2
2^I
Result = Result *
10 + x % 10;
Beware the RESULT may
bigger than INT_MAX
Int(-2147483648 ~ +2147483647) -> INT_MAX and
INT_MIN will very useful
When you want to discard the repeat ,
use "if" better than "while"
Core: if((base > INT_MAX /
10) || (base == INT_MAX / 10
&& str[i] - '0' > 7))
result[i] =
result[i-1] +
result[i-2]
Result = Result *
10 + x % 10;
Reverse the numbers and then
compare with the origin
只需要判断一半,如12321,就是x
= 12,result = 123 最后x = result / 10成立
Temp += nums[i];
DP问题
Answer =
std::max(Temp,Answer);
Temp =
std::max(Temp,0);
第一层循环是字符串内的字符index
第二层循环才是字符串index
对每一字符串的同一index内的字符进行比较
若都一样，则在第一层循环中把它加到prefix里
str[i++] means check
str[i] then i = i+1
同26，只不过条件从if(nums[i] != nums[i-1])变成了if(nums[i] != val)
利用"id".
if(nums[i] != nums[i-1])
nums[id++] = nums[i];
这样，若数重复，则在下次循环中比id大1，会自动替换掉。
而且id就是修改后数组的长度。
Do not beyond
the border
Start from back
二分查找大小确定位置
当你要判断大量的值时，不要忘记while
核心代码：
Int left = formin(root->left);
Int right = formin(root->right);
return(left==0 || right==0)?right+left+1 : std::min(right,left)+1;
牛顿迭代法，令x^2-a=0
任取一点无限求切线与X轴交点，逼近最终答案result
见https://www.zhihu.com/question/20690553
Result = (result
+ a/result)/2
重要的是理解递归语句 return
std::max(maxDepth(root->left),maxDepth(root->right))
+ 1;
把树看成只有三个节点即可。子节点的情况也是可以如此计算最长长度。
利用result += max(prices[i]-prices[i-1],0)将大于零的情况全部加起来即可
通过二分法实现左小右大
先确定基准状态
然后设定根节点为中间值如此递归
std::vector<int>left(nums.begin(),nums.begin()+middle);
std::vector<int>right(nums.begin()()+middle+1,nums.end());
ans->left = sortedArrayToBST(left);
ans->right = sortedArrayToBST(right);
DP： temp += prices[i] + prices[i-1];
temp = std::max(temp,0); result =
std::max(temp, result);
2.可以在一个动态数组内动态变化。从后往前进行刷新。
for(int i = 0;i < numsRow+1;++i)
for(int j = i ; j >= 1 ; ++j)
answer[j] = answer[j] + answer[j-1];
1.只是118多个选择而已
首先是深度搜索，搜索完要将同一深度的放入数组。注意是左边的先出。
void dfs(BitNode *root,int height){
if(root == NULL)
return;
while(ans.size()<=height)
ans.push_back(std::vector<int>());
ans[height].push_back(root->val);
dfs(root->left, height + 1);
bfs(root->right, height + 1);
}
非常优秀的深度搜索题目
注意，typedef struct
BitNode{}BitNode,*BitTree;
前者是BitNode普通类型，后者是结构指针。
传递参数到void函数时注意要用的是实参。
传递指针虽然形参和实参指向同一块地址，但在开辟空间的时候还是不一样的。也就是只有用&才是真正的实参。
创建二叉树时候应该用二级指针
Int left = depth(root->left);
Int right = depth(root->right);
return abs(left-right)<=1 && isBalance(root->left)&& isBalance(root->right);
left_length = DFS(root->left);
if(left_length == -1) return -1;
right_length = DFS(root->right);
if(right_length == -1) return -1;
if(abs(left_length - right_length) > 1) return -1;
return std::max(left,right)+1;
1.所有节点都要遍历一遍。核心算法：
Int depth(TreeNode *root){
if(root == NULL) return 0;
return std::max(depth(root->left),depth(root->right))+1;//每进一层加一
}
两种方法：1.从上到下，2.DFS(从下到上)
因为同一点肯定可以直接判断正确，然后只需要向下左右各自递归了
两个函数，一个是原来的求对称函数一个是mirror,求对称中直接return mirror(root,root);
Mirror中基准状态跟100题一样，唯一不同的是递归return中需要判断
p1->val == p2->val && mirror(p1->left,p2->right) &&
(p1->right,p2->left);
先确定两数组放一起的大小，然后比较两数组最大的项，大的从最大的那个位置开始放。
然后大的那个数组下标-1，如此类推。最后被加过去的那条数组要检查是否有剩余的。
若有剩余则一定是最小的，直接移过去即可。
l+j 可以跳过不相同的字母
用两层循环，第一层是对原句的，第二层检测字母是否对应相同，否则第一层+1。
遍历是这样的：while(answer!=NULL){ ... ...
answer = answer->next;}
注意不要忘记base. if(l2 == NULL) return l2; if(l2 == NULL) return l1;
对两个已排好序的链表中的元素利用递归进行排序
Vector.insert(Vector.begin,1);
for(int i =Vector.size() - 1; i >=0 ;--i){ if(Vector[i] < 9){
Vector[i]++; return Vector; } Vector[i] = 0; }
动态规划和递归思想
每次都是1或2步，从N开始想，倒退回去后思考，是不是那一个位置也是前一个位置移动一/两步得到的，
每次都是一两步，最终到达基准情况，将他们都相加就是所有可能性，这也就是FIB。
若有一边是NULL，则不能用最小判断（起点会出问题）
PS：用left+right就不需要打更多的代码了。
否则你还要判断哪边为空。
a[i][j] = a[i-1][j]
+ a[i-1][j-1];
answer[i].resize(i+1)
a[i][0] = a[i][i] = 1;
以下从内到外。
1.用while进行计算是否有相同的数字
2.用for循环控制是否超过长度
3.用while循环检测是否达到输入的要求
c==1代表有进位，循环继续。
添加位用result =char(c%2+'0')+result;
若c==2则表示需要进位，若c==1则可能该位两数都是0，但是前面保留了进位，故该位设为1。如此类推。
用一个数存储进位
std::to_string()
核心代码：
if(root->left == NULL && root->right == NULL && sum - root->val == 0)
return true;
return hasPathSum(root->left,sum-root->val)||hasPathSum(root->right, sum-root->val);
递归思想，设定好基准状态：
if(p == NULL && q ==NULL) return true;
else if(p == NULL || q ==NULL) return false;
把每个需要对应的左字符放入栈
遇到任意右字符的时候检测栈顶是否为对应的左字符
是的话则将栈顶pop，继续检测下一个字符
最终检测栈是否为空,
return result.empyt()
for(char&s : str)对字符串内的每个元素
开始递归判断 else if(p->val == q->val){
return isSame(p->left,q->left)&&isSame(p->right,q->right);}
return false;
用循环对每个字符进行判断，如I=1.V=5等等
S.find()找到有IV等特殊值字符，减去其与VI的差即可
把每一个位数的罗马数字列成一个数组，然后通过下面算法返回即可
return M[num/1000] + C[(num%1000)/100] + X[(num%100)/10] + I[num%10];