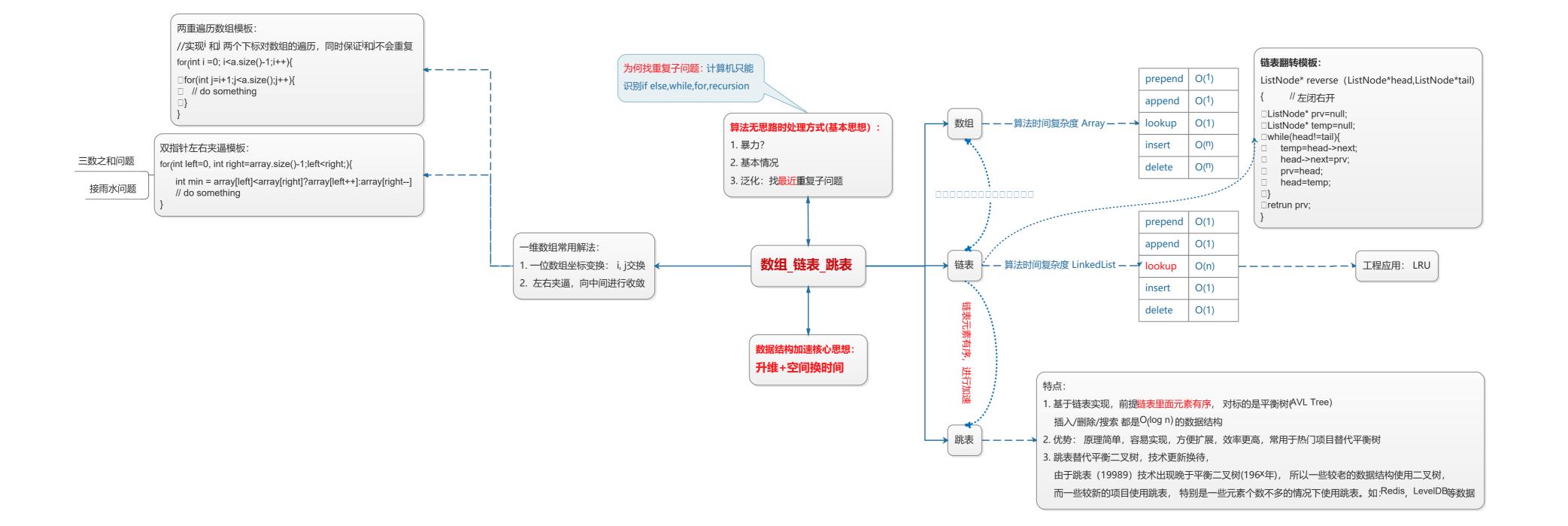
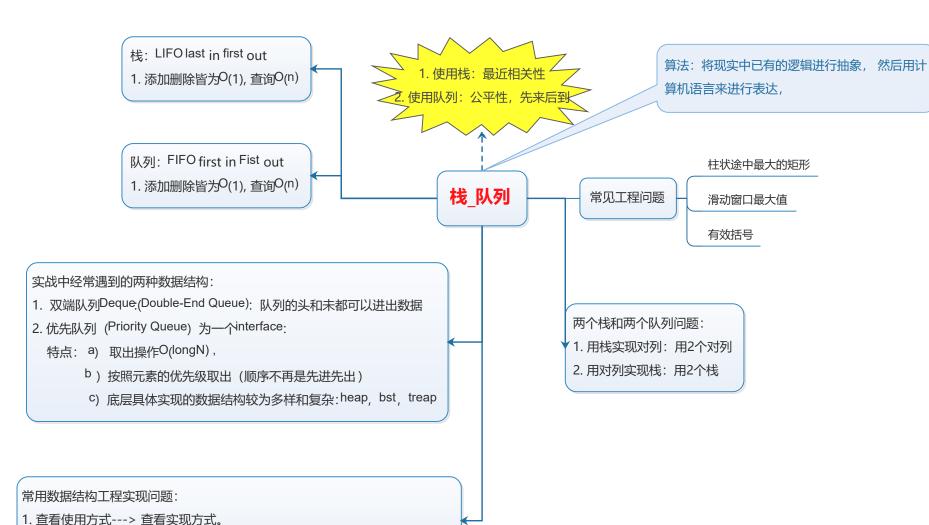


Algorithm 二分查找 (Binary search) Apply Master theorem case c=logba where T(n) = T(n/2) + O(1)O(logn) a=1,b=2,c=0,k=0 (本身有序) 二叉树遍历 (Binary tree Apply Master theorem case c<logba where T(n)=2T(n/2)+O(1)O(n) traversal) a=2,b=2,c=0 Apply the Akra-Bazzi theorem for p=1 and g 二维矩阵二分查找 (有序) T(n)=2T(n/2)+O(1)O(n) (u)=log(u) to get $\theta(2n$ -logn) Apply Master theorem case c=logba where 归并排序(Merge sort) T(n)=2T(n/2)+O(n)O(nlogn) a=2,b=2,c=1,k=0





2. 建议查看 java 实现是开源的,直接查看java 源代码实现即可,查看实现原理

```
二叉树的遍历 (广度优先遍历):
示例代码:
void levelOrder(TreeNode* root) {
   if(root == NULL){
      return;
   // 使用队列先进先出, 队列头为root
   queue<TreeNode*> queueTreeNode;
   queueTreeNode.push(root);
   while(!queueTreeNode.empty()){
  TreeNode* temp = queueTreeNode.front();
 // do process logic
       queueTreeNode.pop();
       if(temp->left !=NULL){
         queueTreeNode.push(temp->left);
        if(temp->right !=NULL){
         queueTreeNode.push(temp->right);
    return
```

```
树结构产生的原因
                  --→ 1. 人需要解决二维世界问题
                                                                                                 前序示例:
                      2.工程需求,斐波拉契,棋牌类游戏,生成对应状态树,叶子节点为最终状态,AlphaGo就是在树中找最优解
                                                                                                 void preorder(struct TreeNode *root){
                                                                                                  \Boxif(root ==null){
                                                                                                   return;
                                树的遍历基本都是基于递归的原因:
                                                                                                      int printvale= root_> value;
                               树的定义本身,它没法有效的进行所谓的循环,写循环比较麻烦,而写递归相对比较简单
                                                                                                  □preorder(root->left);
                                                                                                                  遍历左子树
                                                                                                  □preorder(right);//
                                                                                                                遍历右子树
                       二叉树的遍历 (深度优先遍历)
                       1. 前序(Pre-order): 根-左-右
                      2. 中序(In-order): 左-根-右
                                                                                                 中序示例:
                       3.后续(Post-order): 左-右-根
                                                                                                 void inorder(struct TreeNode *root){
                                                                                                  □if(root ==null){
                                                                                                   return;
                                                                                                      inorder(root->left);遍历左子树
                   处理当前逻辑:
                                                                                                      int printvale= root-> value;
                                                                                                  □inorder(right);//
                                                                                                               遍历右子树
有两个next的链表(树是特殊的链表)
                                        ---- 二叉搜索树demo
         链表
                                                                                                 后续示例:
                                                 二叉搜索树定义: (操作时间复杂度为q(log n))
                                                                                                 void postorder(struct TreeNode *root){
     树,二叉树,二叉搜索树
                                   二叉搜索树
                                                                                                  □if(root ==null){
                                                 1. —颗空树
                                                                                                    return;
                                                2. 有下面性质的二叉树
                                                                                                      postorde(root->left);遍历左子树
                ┏━━ 第一掌:前序遍历
                                                 a) 左子树上所有节点的值均小于它的根节点的值
                                                                                                      postorde(right);//遍历右子树
                                                 b) 右子树上所有节点的值均大于它的根节点的值
                 -▶ 第二掌:中序遍历
                                                                                                  □//c
                                                                                                       处理当前逻辑:
                                                 c) 以此类推: 左·右子树也分别为二叉树(这就是重复性!)
                                                                                                  □int printvale= root->value ;
               1- → 第三掌: 后续遍历
```

```
递归本质:循环,通过函数体进行的循环
  递归特点 (可以用盗梦空间来类比):
  1. 向下进入到不同的递归层,向上又回到原来一层,一般来讲不能直接跳跃,有一种对称性
  2. 使用参数来进行不同层之间的传递变量 (函数调用形参压栈过程)
  3. 每一层调用 (函数调用堆栈) 都是一份<sup>copy</sup>,只有函数参数,全局变量等会携带变化
电影盗梦空间来类比:
1. 向下进入不同梦境,向上又回到原来一层
2. 通过声音同步回到上一层
3. 每一层的环境和周围的人都是一份copy, 主角等几个人穿越不同层级的梦境(发生和携带变化)
               思维要点:
               1. 不要人肉进行递归(最大误区)
               2. 找到最近最简方法,将其拆解成可重复解决的问题(重复子问题)
               3. 数学归纳法的思维
                a. 最开始,最简单的条件是成立的
                b. 能够证明n成立时能推导出n+1条件成立
              Python 代码模板:
```

读《算法导论》扩展:

递归,泛型递归

```
#1. recusion terminator(递归终结条件)
   if level > MAX_LEVEL:
     process_result
   #2. process logic in current level (处理当前逻辑)
    process(level, data....)
   #3. drill down (下探到下一层)
   self.recusion(level+1,param1, param2...)
                                                                       递归代码模板:
   #4. reverse the current level status if needed(清理当前层)
                                                                       1. 递归终结条件
                                                                      ・ 2. 处理当前逻辑
                                                                      3. 下探到下一层
                                                                       4. 清理当前层 (如果有必要)
Java 代码模板:
public void recursion(level,param1,param2,...){
  #1. recusion terminator(递归终结条件)
   if (level > MAX_LEVEL) {
    //process_result
    return;
   #2. process logic in current level (处理当前逻辑)
   process(level, data....);
   #3. drill down (下探到下一层)
   recusion(level+1,param1, param2...);
   #4. reverse the current level status if needed(清理当前层)
```

def recursion(level,param1,param2,...):

```
五毒神掌: 代表经典问题: 爬楼梯问题, 括号生成问题, 二叉树深度(即中序遍历)验证二叉搜索树
问题分析: 我们用 f(x)f(x) 表示爬到第 xx 级台阶的方案数,考虑最后一步可能跨了一级台阶,也可能跨了两级台阶,所以我们可以列出如下式子:
f(x) = f(x-1)+f(x-2),即为: 斐不拉契数列
C++ 代码实现: (斐不拉契数列优化版,添加缓存,时间复杂度为O(n))
class Solution {
public:
 long climbStairs(long n) {
   // 递归终结者
   if(n <2) {
     return 1;
   //判断缓存是否存在
   if(mapcache.count(n) > 0){
     return mapcache[n];
   //处理当前逻辑
   long temp= climbStairs(n-1)+climbStairs(n-2);
   mapcache[n]=temp;
   return temp;
private:
 map<long,long> mapcache;
```