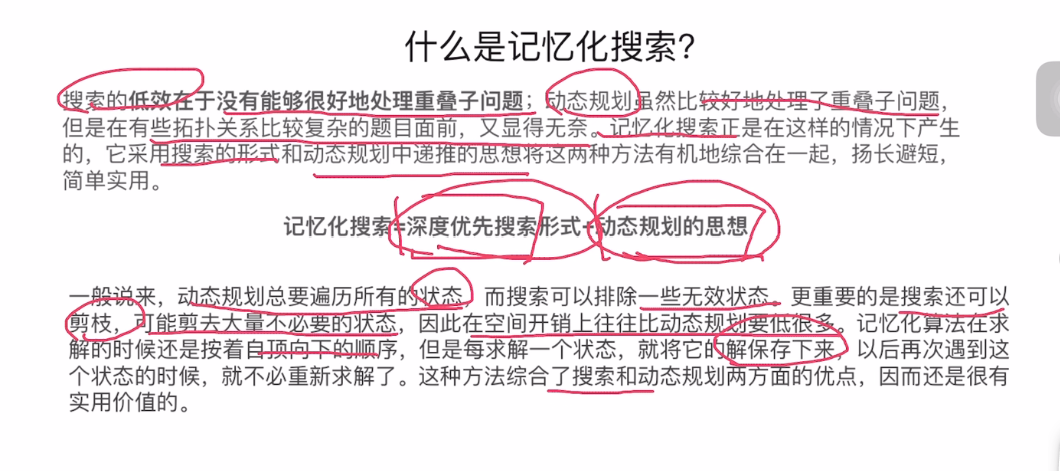
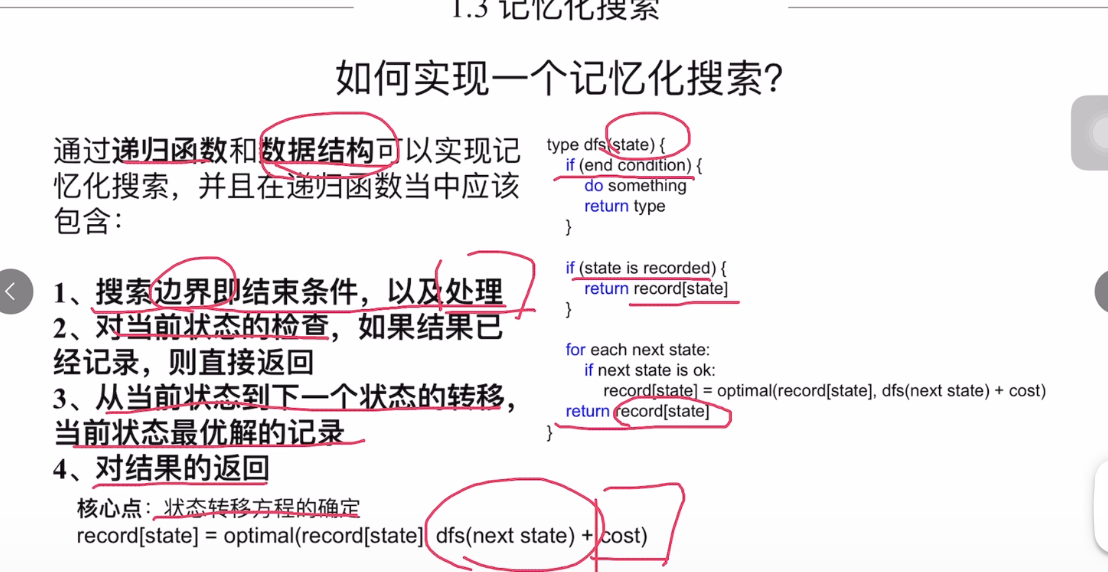
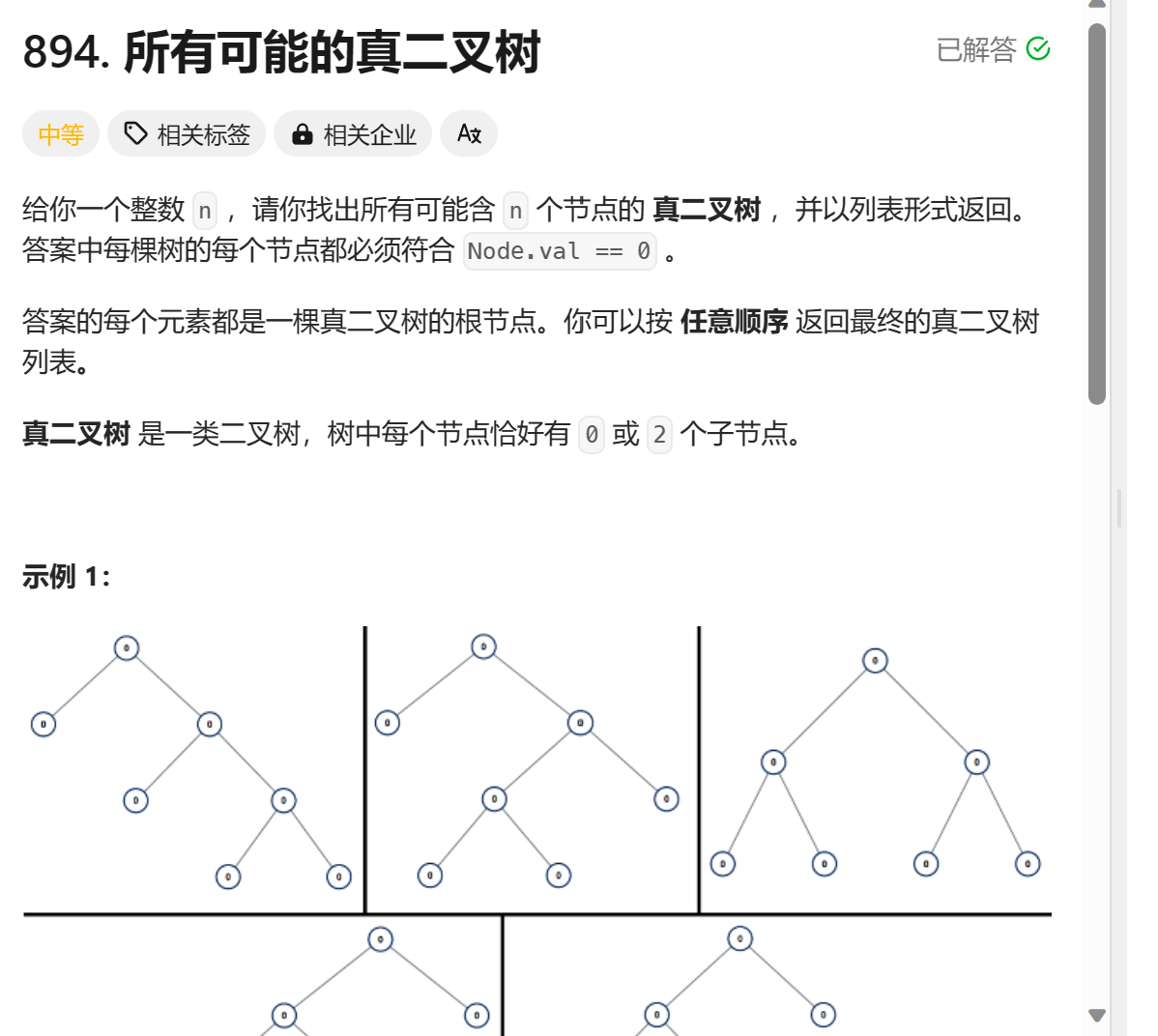
# 记忆化搜索





代码模板：

1. 首先需要思考在dfs过程中需要记录的对象
2. 确定特殊的返回边界
3. 每一层的问题的拆分与记录
4. 遍历记录过的过程将解决的问题记录下来
5. 返回记录的结果



public:

    // 记忆化搜索的本质是在dfs过程中，维护一个dp数组来记录之前遇到的状态遇到过的状态用该数组记录下来不再去计算。

    //其代码步骤和回溯差不多

    unordered\_map<int,vector<TreeNode \*>> dp;

    vector<TreeNode\*> allPossibleFBT(int n) {

        if(dp.find(n)==dp.end()){//这是没遇到过的状态//边界思考

         vector<TreeNode\*> root;

         if(n==1){

             dp[1].push\_back(new TreeNode(0));

             return dp[1];

         }

         else if(n%2==1){

             for(int i=1;i<n;i+=2){//因为是真二叉树所有以节点个数是两个两个加的

                 int j=n-i-1;//右子树节点个数

                    vector<TreeNode\*> le=allPossibleFBT(i);//得到左子树的所有可能

                    vector<TreeNode\*> ri=allPossibleFBT(j);//得到右子树的所有可能

                    for(auto &l:le){

                        for(auto &r:ri){

                            TreeNode \* key =new TreeNode(0,l,r);

                            root.emplace\_back(key);

                        }

                    }

             }

             dp[n]=root;

         }

        }

        return dp[n];

    }



 unordered\_map<string ,vector<int>> dp;

     bool juage(string s){//判断是否全为数字

         for(auto key :s){

             if(key=='+'||key=='-'||key=='\*'){

                 return false;

             }

         }

         return true;

     }

     vector<int> dfs\_recall(string s){

         if(dp.find(s)==dp.end()){

             if(juage(s)){

                 dp[s].emplace\_back(stoi(s));

                 return dp[s];//边界的返回条件

             }

             int lf = 0;

             int rig=s.size()-1;

             for(int i=lf;i<rig;i++){

                 if(s[i]=='+'){

                     string l = s.substr(lf,i);//分界

                     string r = s.substr(i+1,s.size()-i-1);

                     dp[l]=dfs\_recall(l);

                     dp[r]=dfs\_recall(r);

                     for(auto &lt :dp[l]){//枚举所有情况

                         for(auto &rg :dp[r]){

                             dp[s].emplace\_back(lt + rg);

                         }

                     }

                 }

                 else if(s[i]=='-'){

                      string l = s.substr(lf,i);

                     string r = s.substr(i+1,s.size()-i-1);

                     dp[l]=dfs\_recall(l);

                     dp[r]=dfs\_recall(r);

                     for(auto &lt :dp[l]){

                         for(auto &rg :dp[r]){

                             dp[s].emplace\_back(lt - rg);//记录所有情况

                         }

                     }

                 }

                  else if(s[i]=='\*'){

                      string l = s.substr(lf,i);

                     string r = s.substr(i+1,s.size()-i-1);

                     dp[l]=dfs\_recall(l);

                     dp[r]=dfs\_recall(r);

                     for(auto &lt :dp[l]){

                         for(auto &rg :dp[r]){

                             dp[s].emplace\_back(lt \* rg);

                         }

                     }

                 }

             }

         }

         return dp[s];

     }

    vector<int> diffWaysToCompute(string expression) {

              dfs\_recall(expression);

              return dp[expression];

    }