**一：判断题**

1. 一个正确的算法，对于每个合法输入，都会在有限的时间内输出一个满足要求的结果。（×）

2、NP完全问题比其他所有NP问题都要难。（错）

NP问题分为NP-hard问题和NP完全问题，NP完全问题是最难的，但是NP-hard问题又包含NP完全问题。

错：NP完全问题至少同其他所有NP问题一样难。

3、回溯法用深度优先法或广度优先法搜索状态空间树。（错，仅深度优先）

错：回溯法用深度优先法搜索状态空间树。

4、在动态规划中，各个阶段所确定的策略就构成一个策略序列，通常称为一个决策。（错）

错：在动态规划中，各个阶段所确定的决策构成一个决策序列，通常称为一个策略。

5、P类和NP类问题的关系用P⊂NP来表示是错误的。（错）

错：P中所有问题均属于NP。

P：存在求解判定问题P1的多项式时间算法。

NP：对P1的每一个肯定实例均存在一个多项式时间内的验证。（对一个实例为肯定实例的证明称为该实例的证书。）

NPcomplete:P1是一个NP问题，且NP中所有问题都可以多项式转化为P1.

NPhard：NP中所有问题都可以多项式转化为P1。

NPhard问题范围大于NPcomplete问题范围。

6、若近似算法A求解某极小化问题一实例的解为Sa，且已知该问题的最优解为Sa/3，则该近似算法的性能比为3。（错）

**错：**性能比是所有实例可能的精确率的上界。3只是这一实例的精确率，不是所有实例的。

7、通常来说，算法的最坏情况的时间复杂行比平均情况的时间复杂性容易计算。（对）

8、若P2多项式时间转化为(polynomial transforms to)P1，则P2至少与P1一样难。（错）

错：应该是p1至少和p2一样难，有可能p1更难。

总是可以在可比时间内用P1的算法解决P2，但不能说P1和P2一样难。事实上，有时候可能P2简单而P1更难。

9、快速排序算法的平均时间复杂度是O(nlogn)，使用随机化快速排序算法可以将平均时间复杂度降得更低。（错）

10、基于比较的寻找数组A[1,…,n]中最大元素的问题下届是Ω(n/3)。（错）

**对**：下界理论：问题的下界不唯一，越高越好。算法最优：下界=上界

几个问题的下界：

1. 排序问题：
2. 找最大：或
3. 找最大最小： 下界是用分治策略得到的结果：将数组均分成两部分，在每部分中找出最大值和最小值，在比较这两部分的最大值和最小值。
4. 找第二大元素： 找第一大元素需要比较n-1次，第一是通过两两配对比较中较大的元素中最大的元素，第二大元素师两两比较中第二大的元素，再找到第二大元素需要logn-1次比较，加在一起得到如上结果。

11、O(f(n))+O(g(n))=O(min{f(n),g(n)})（错）

错。应该是max，O代表的是算法的上界。

12、若f(n)=Ω(g(n))，g(n)=Ω(h(n))，则f(n)=Ω(h(n))（对）

对：相当于小于等于。

13、若f(n)=O(g(n))，则g(n)=Ω(f(n))（对）

对：O相当于大于等于。

14、贪婪技术所做的每一步选择所产生的部分解，不一定是可行性的。（错）

错：贪心算法每一步必须满足：可行的（即它必须满足问题的约束）、局部最优、不可撤销。

贪心算法通常包含一个用以寻找局部最优解的迭代过程，在某些实例中这些局部最优解转变成了全局最优解，而在另外一些实例中则无法找到全局最优解。

15、LasVegas算法只要给出解就是正确的。（对）

16、一个完全多项式近似方案是一个近似方案{Aε}，其中每一个算法Aε在输入实例I的规模的多项式时间内运行。（错）

错：题目中定义的是多项式近似方案。

完全多项式近似方案：是一个近似方案，其中每一个算法在输入实例的规模和两者的多项式时间内运行

**二：简答**

1. **二叉查找树属于减治策略的三个变种中的哪一个的应用？什么情况下二叉查找树表现出最差的效率？此时的查找和插入算法的复杂性如何？**

答：减治策略有3个主要的变种，包括减常量、减常数因子和减可变规模。(1) 二叉查找树属于减可变规模变种的应用。(2) 当先后插入的关键字有序时，构成的二叉查找树蜕变为单支树，树的深度等于n，此时二叉查找树表现出最差的效率，(3) 查找和插入算法的时间效率都属于Θ(n)。

1.（1）属于减可变规模的应用。（2）当关键字的个数等于二叉查找树的高度时表现出最差的效率。时间复杂度为O(n)。（3）此时查找和插入算法在最坏情况（查找当前序列的最大值或者插入最大值）的时间复杂度都是O(n)。

1. 二叉查找树属于减治策略中的减去的规模是可变的。当二叉查找树是严格歪斜的时候，效率最差，此时的查找和插入效率为。
2. 二叉查找树属于减治策略中减少可变规模变种中的应用；
3. 当树的高度=节点个数-1，即树是一颗严格歪树时表现出最差的效率；
4. 此时查找和插入算法的时间复杂度为O(n)。

减治法的三种变种：

1. 减去一个常量：

递归法求;

插入排序；最坏 平均

快速排序+插入排序

拓扑排序：减一

生成排序：减一 要求1-n的排序先求1-（n-1）的排序

生成子集：减一

1. 减去一个常数因子：

二分搜索

假币问题

俄式乘法

约瑟夫问题

1. 减去可变规模

欧几里得除法

插值查找：最差O(n)

二叉查找树:最差O(n) 平均O(logn)

变治的3种类型：

1. 实例化简：变换为同样问题的一个更简单或更方便的实例

预排序：如果列表是有序的，许多问题更容易求解。

-检验数组中元素的唯一性：先对数组排序，然后只检查连续元素，如果该数组有相等的元素则一定有一对元素是相互紧挨着的 复杂度：排序+查找紧邻元素是否相同即O（nlogn+n）=o(nlogn)。

-查找数字列表中出现频率最高的一个数：先对输入排序（之后有相等的数值都会邻接在一起），求出该有序数组中邻接次数最多的等值元素。

高斯消去：把n个线性方程构成的n元联立方程组变换为一个等价的方程组，该方程组有一个上三角系数矩阵，该矩阵主对角线下的元素都为0.

LU分解

AVL树：把集合变为二叉树

-AVL树是一棵二叉树，其中每个节点的平衡银子定义为该节点左子树和右子树的高度差，这个平衡因子要么是0要么是1要么是-1.

1. 改变表现：变换为同样实例的不同表现

2-3树：平衡查找树

堆和堆排序

霍纳法则

二进制幂

1. 问题化简：变换为另一个问题的实例，这种问题的算法是已知的

最小公倍数：通过两数乘积与其最大公约数之商求得

函数极值：已知最小求最大，目标函数加负号

线性规划

简化为图

1. **何谓伪多项式算法？如何将一Monte Carlo算法转化为Las Vegas算法？**

答：若一个数值算法的时间复杂度可以表示为输入数值N的多项式，但其运行时间与输入数值N的二进制位数呈指数增长关系，则称其时间复杂度为伪多项式时间。

Las Vegas算法不会得到不正确的解。一旦用拉斯维加斯算法找到一个解，这个解就一定是正确解。但有时用拉斯维加斯算法找不到解。

Monte Carlo算法每次都能得到问题的解，但不保证所得解的准确性

转化：可以在Monte Carlo算法给出的解上加一个验证算法，如果正确就得到解，如果错误就不能生成问题的解，这样Monte Carlo算法便转化为了Las Vegas算法。

2.（1）伪多项式时间算法是NPC的一种，存在复杂度是关于实例规模和实例所有参数中绝对值最大数是成多项式关系的算法。

1. 伪多项式时间算法是一种算法，它在L值的多项式时间内运行，其中L是输入实例中的最大数值。这是PPT里的定义（维基百科：伪多项式算法：若算法的时间复杂度可以表示成输入数值N的多项式，则称其为伪多项式算法。）
2. 伪多项式算法是一种在L值的多项式时间内运行的算法，其中L是输入实例中的最大数值；
3. Monte Carlo算法每次都能得到问题的解，但不保证所得的解的准确性；Las Vegas算法每一次不一定得到问题的解，只要得到解一定是正确的解。可以在Monte Carlo算法后加上一个验证算法，如果正确就得到解，如果错误就不能生成问题的解，这样Monte Carlo算法便转化成了Las Vegas算法。

可以通过多次运行Monte Carlo，并且满足每次运行时的随机选择都相互独立，使产生非正确解的概率可以减到任意小，从而转化为Las Vegas。

1. **构造AVL树和2-3树的主要目的是什么？它们各自有什么样的查找和插入的效率？**

答：(1)当先后插入的关键字有序时，构成的二叉查找树蜕变为单支树，树的深度等于n，此时二叉查找树表现出最差的效率，为了解决这一问题，可以构造AVL树或2-3树，使树的深度减小。一棵AVL树要求它的每个节点的左右子树的高度差不能超过1。2-3树和2-3-4树允许一棵查找树的单个节点不止包含一个元素。(2) AVL树在最差情况下，查找和插入操作的效率属于Θ(lgn)。2-3树无论在最差还是平均情况下，查找和插入的效率都属于Θ(log n)。

3. 构建AVL树和2-3树能够平衡左右子树的高度，减少树的层数，使得平均搜索效率更高。

插入和查找的时间复杂度均为O(logn)。

1. 二叉查找树属于减治策略中的减去的规模是可变的。当二叉查找树是严格歪斜的时候，效率最差，此时的查找和插入效率为。
2. 构造AVL树和2-3树的目的是使左右子树更加平衡，减小树的层数，使平均搜索效率更高；

（避免出现2叉树中，树严格歪斜的情况，使以每个节点为根的左右子树的高度接近，增加查找和插入的效率。）

1. 他们在最坏情况下插入和查找的时间复杂度为O(log2n)。
2. **写出0/1背包问题的一个多项式等价(Polynomial Equivalent)的判定问题，并说明为什么它们是多项式等价的。**

答：0/1背包问题：从M件物品中，取出若干件放在空间为W的背包里，给出一个能获得最大价值的方案。每件物品的体积为W1，W2……Wn，与之相对应的价值为P1,P2……Pn。+

判定问题I：从M件物品中，取出若干件放在空间为W的背包里，是否存在一个方案，所获价值≥P\*？。每件物品的体积为W1，W2……Wn，与之相对应的价值为P1,P2……Pn。

若判定问题I存在多项式时间的解法，则反复调用该算法就可以在多项式时间内解决0/1背包的优化问题。因而这个判定问题与原问题多项式等价。

0/1背包问题的一个多项式等价判定问题是整数划分问题。

证明：1.背包优化问题可以多项式转化为背包判定问题。我可以利用背包优化问题的最优解K\*去判断K\*是否大于等于K。2.背包判定问题可以多项式转化为背包优化问题。我们可以对可能的背包物品价值进行二分查找。搜索区间的下限是（其中是所有物品中的最小值），上限是（其中是所有物品中的最小值）。在各搜索点上解其背包判定问题以确定背包优化问题的最优解K\*。

（1）0/1背包问题的一个多项式等价判定问题是：给定价值为V1,V2,..,Vn,重量为w1,w2,..,wn的n个项和两个整数v\*和w\*，是否存在一个子集S使得且。

（2）若存在0/1背包问题的多项式算法，则可用其在多项式时间内求解判定问题，令背包容量等于w\*，求出0/1背包问题的最优子集S，则可以通过判断S是否满足来确定判定问题的解。

若存在判定问题的多项式算法，则可以在可能的价值范围内进行二分搜索，在各搜索点上解判定问题以确定0/1背包问题的最优解，令，可在O(logV)时间内求解。

1. **下面问题是否属于NP问题？为什么？**

问题表述：给定图中的两个点、，整数和，图中每条边的长度及便利这条边的时间，问图中是否存在一条由到的路径，使得其长度大于，且遍历时间小于？

答：这个问题属于NP问题。因为若给出该问题的一个解，可以在多项式时间内检验这个解的正确性。如给出一条由p到q的路径，可以在多项式时间内计算出它的长度及遍历时间，然后分别与C和t进行比较，从而可以判断这个解的对错。

1. 属于NP问题。因为给定一个肯定的实例（存在一条由p到q的路径经过顶点p,v1,v2,...vn,q，该路径的长度大于C，遍历时间小于t）。我们可以在多项式时间内验证该实例是否正确，只需要求出这条路径上的每条边的长度之和S和遍历时间T，判断S>C,T<t是否成立即可。以上验证算法是可以在多项式时间内完成的。

属于NP问题，因为给定一个肯定实例能够在多项式时间O(n)内验证。

分治题

1. 写出一个求解下列问题的分治算法，推导其时间复杂性并与蛮力法相比较。

给定互不相等的n个数的一个序列，若其中某两个数和的关系为：且，则称和是逆序的。要求计算该序列中的逆序个数，即具有逆序关系的元素对的总数目。

**解：这一份代码正规点**

int count=0;

void merge(int arr[],int first,int mid,int last)

{

int tmpArr[100];

int i=first,j=mid+1;

int cur=0;

while(i<=mid && j<=last)

{

if(arr[i]<arr[j])

{

tmpArr[cur++]=arr[i++];

}

else

{

tmpArr[cur++]=arr[j++];

count += mid-i+1;//只增加这一句便可求逆序数

}

}

if(i<=mid)

{

while(i<=mid)

tmpArr[cur++]=arr[i++];

}

else

{

while(j<=last)

tmpArr[cur++]=arr[j++];

}

for(int k=0;k<cur;k++)

{

arr[first++]=tmpArr[k];

}

}

void mergeSort(int arr[],int first,int last)

{

if(first==last)

return;

int mid=(first+last)/2;

mergeSort(arr,first,mid);

mergeSort(arr,mid+1,last);

merge(arr,first,mid,last);

}

算法思路：以归并排序为基础，在两两集合合并的时候如果前一个集合的元素a[i]>a[j]，那么说明需要调整次序，逆序数num=num+mid-i。

时间复杂度的迭代公式为 因此算法的时间复杂度为T(n)=O(nlogn)；

蛮力法的时间复杂度为O(n2)，当n数目较大时，分治法计算规模远小于蛮力法。

**解：**/\*\* \*求解n个数的一个序列，具有逆序关系的元素对的总数目\*/

count = 0; //逆序元素对的全局计数变量

mergeInvertedPairs(A,low,mid,high) {

i = low;

j = mid+1;

k = low;

tmp[n]; //用于归并排序的辅助数组

while i <= mid && j <= high {

if (A[i] > A[j]) {

tmp[k] = A[j++];

count += (mid-i+1); //相比归并排序，就多了这一条语句

} else {

tmp[k] = A[i++];

}

k++;

}

while i <= mid {

tmp[k++] = A[i++];

}

while j <= high {

tmp[k++] = A[j++];

}

for (j = low; j <= high; j++) {

A[j] = tmp[j];

}

}

findInvertedPairs(A[], low, high) {

if (low < high) {

mid = (low+high) / 2;

findInvertedPairs(A,low,mid);

findInvertedPairs(A,mid+1,high);

mergeInvertedPairs(A,low,mid,high);

}

算法思路：以归并排序为基础，在两两集合合并的时候如果前一个集合的元素a[i]>a[j]，那么说明需要调整次序，逆序数num=num+mid-i。

时间复杂度的迭代公式为 因此算法的时间复杂度为T(n)=O(nlogn)；

蛮力法的时间复杂度为O(n2)，当n数目较大时，分治法计算规模远小于蛮力法。

1. 为一个整数序列，中的整数如果在中出现次数多余，那么称为多数元素。例如，在序列中3是多数元素，因为出现了4次，大于。求A的多数元素问题的蛮力算法复杂性如何？设计一个具有变治思想的算法，提高蛮力算法的效率，写出伪代码并分析其事件复杂性。

2. num **<-** src**[**0**];**

count **<-** 0**;**

for i **<-** 0 to n**-**1 do

**{**

if**(**num **==** src**[**i**])**

**{**

count**++;**

**}**

else

**{**

count**--;**

if**(**count **<** 0**)**

**{**

num **<-** src**[**i**];**

**count = 0;**

**}**

**}**

**}**

采用减治的思想每一个减去一个元素，时间复杂度为O(n)，蛮力法的时间复杂度为O(n2)。

1. **蛮力算法**：对数组进行扫描，在另外两个数组中存储已经遇到的值和它的频率，每次迭代时，通过遍历储值辅助数组，使原始数组中第i个元素和已遇到的数值进行比较，如果碰到一个匹配值，该数值出现的频率加1，否则，将当前元素添加到储值数组中，并把它出现的频率置为1.最后检查频率数组中是否有元素大于，若有则对应的储值数组中的元素为多数元素。蛮力算法的时间复杂度为O(n2)。
2. **变治思想的算法**：先将数组中的元素按非递减顺序排序，检查该有序数组中邻接次数最多的等值元素个数是否大于。

伪代码：

findMostNum(A[1…n])

{

对数组A排序；

I=1;

While(i<=n)

{

Length=n/2;

While(i+length<=n&&A[i]=A[i+length])

Length=length+1;

If(length>=n/2)

输出A[i];

I=i+length;

}

}

该算法的时间复杂度为排序的时间复杂度和线性时间复杂度之和即O(nlogn)。

动态规划题

1. 某工厂调查了解市场情况，估计在今后四个月内，市场对其产品的需求量如下表所示。

|  |  |
| --- | --- |
| 时期（月） | 需要量（产品单位） |
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |
| 4 | 4 |

已知：对每个月来讲，生产一批产品的固定成本费为3（千元），若不生成，则为零。每生产单位产品的成本费为1（千元）。同时，在任何一个月内，生产能力所允许的最大生产批量为不超过6个单位。

又知：每单位产品的库存费用为每月0.5（千元），同时要求在第一个月开始之初，及在第四个月末，均无产品库存。

问：在满足上述条件下，该厂应如何安排各个时期的生产与库存，使所花的总成本费用最低？写出你所设的状态变量、决策变量、状态转移方程与递推关系式，和手工求解的详细步骤及结果。

**解：**设阶段序数k表示月份，状态变量xk为第k个月初拥有的单位产品数量，亦为第k-1月末时的单位产品数量，决策变量uk为第k个月生产的单位产品数量，ck为第k月份需要的产品数量，这里xk和uk均取离散变量。

状态转移方程为： , k =1,2,3,4; 且x1=0。

k段允许决策集合为: k = 1,2,3;

当k=4时，。

设为第k月的成本费,单位为(千元)，则 ,

故指标函数为

令表示为由出发采用最优生产方案到第4个月结束这段期间的产品成本。

根据最优化原理，则有递推公式：

其中：

逆序计算的详细步骤如下：

1. 当k=4时，
2. 当k=3时，因为 ，且所以有：

当处取得最小值。

当,此时处取得最小值。

当,此时处取得最小值。

当,此时,在处取得最小值。

当,此时,在处取得最小值。

当,此时,在处取得最小值。

当,此时,在处取得最小值。

1. 当k=2时，因为，且所以有：

当时，,在处取得最小值。

当时，,且在处取得最小值。

当时，,在处取得最小值。

当时，，且在处取得最小值。

当时，,且在处取得最小值。

当时，,且在处取得最小值。

当时，,且在处取得最小值。

1. 当k=1时，因为,所以有：

当,,且在处取得最小值。

综上所述，最优的库存方案为：第一月生产5单位产品，第二月和第四月不生产，第三月生产6单位产品。

1. 用动态规划方法手工求解以下问题

有8万元的投资可以投给3个过目，每个项目在不同筒子数额下（以万元为单位）的利润如下表

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 投资额  盈利  项目 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 项目1 | 0 | 5 | 15 | 40 | 80 | 90 | 95 | 98 | 100 |
| 项目2 | 0 | 5 | 15 | 40 | 60 | 70 | 73 | 74 | 75 |
| 项目3 | 0 | 4 | 26 | 40 | 45 | 50 | 51 | 52 | 53 |

请安排投资计划，使总的利润最大。

写出你所设的状态变量、决策变量、状态转移方程与递推关系式和手工求解的详细步骤及结构。

**解：**状态变量：xk 表示留给项目k..n的投资额，其中n为项目总个数，k=1..n.

决策变量：uk 表示投给项目k的投资额.

允许决策集合：

状态转移方程：

递推关系式：

其中，表示项目k的投资额为uk时的盈利.

针对本题，n = 3，xk最大取8

手工详解过程：

1. **初始化k = 3**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x3** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **f3(x3)** | 0 | 4 | 26 | 40 | 45 | 50 | 51 | 52 | 53 |

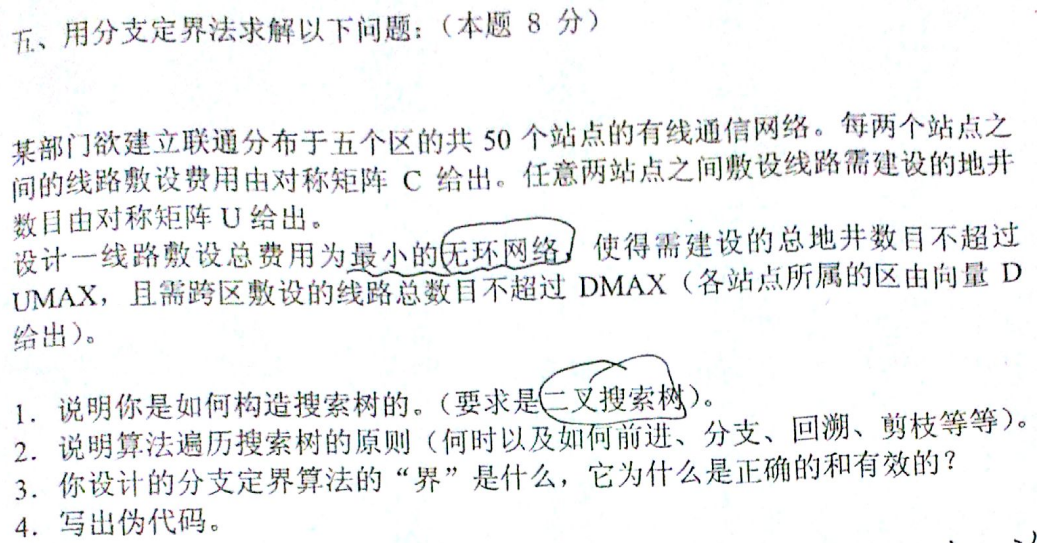
1. **k = 2**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x2** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **f2(x2)** | 0 | 5 | 26 | 40 | 60 | 70 | 86 | 100 | 110 |

1. **k = 1**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x1** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **f1(x1)** | 0 | 5 | 26 | 40 | 80 | 90 | 106 | 120 | **140** |

**最终结果：**给项目1投资4万元，项目2投资4万元，项目3不投资，将获得最大利润140万元.



线路题的某种深搜解法：

1)可以根据线路(l1,l2,...,lm)的取舍构建一棵m层二叉搜索树。第i层的所有左分支表示铺设线路li，右分支则表示不铺设。如果存在可行解，遍历此二叉搜索树即可找到最优解。

2)

前进：当前节点未被剪枝并且仍有子节点即可继续前进。

分支：先遍历左分支，后遍历右分支。

回溯：左右分支都被遍历时返回父节点。

剪枝：

剪枝条件如下：

1。有环路

2。当前地井数 + 地井数下界 > UMAX

3。当前跨区铺设线路数 + 跨区铺设线路数下界 > DMAX

4。当前费用 + 费用下界 >= 已知最优方案的费用

3)

子问题的下界为费用下界、地井数下界、跨区线路数下界。费用下界是根据剩余站点数量定义的，累计最小的路线花费即可得到。由于限制被极度弱化，所以非常粗糙，但是正确有效。另外两个下界也类似。父问题的上界是已知最优方案的费用，显然正确有效。

4)

按费用从小到大排序所有路线l1,l2,...,lm

计算子问题下界：

1。费用下界：剩余站点数量->最小花费 #累计最小的线路花费即可得到，下同

2。地井数下界：剩余站点数量->最小地井数

3。跨区线路数下界：剩余站点数量->最小跨区线路数

search(空集, l1)

返回最优结果

def search(线路集合S，当前线路l):

判断线路集合S是否合格，条件如下：

1。无环路

2。当前地井数 + 地井数下界 <= UMAX

3。当前跨区铺设线路数 + 跨区铺设线路数下界 <= DMAX

4。当前费用 + 费用下界 < 已知最优方案的费用

如果合格：

当前网络已经覆盖所有站点:

记S为已知最优

否则若剩下的线路数有可能使所有站点构成网络:

search(S ∪ {l}, l的下一条路线)

search(S, l的下一条路线)

void LL()

{

int feeLowBound=INF;//初始化最小费用的界

A[]=init();//初始化矩阵C，D，为边标序号,A为边集合

Tree=buildtree(A);//以A建立二叉搜索树

while(not over) //循环到遍历结束

{

if( //符合以下的界

countminU()<UMAX //计算当前路径满足条件的最小数目

&& countminD()<DMAX //计算当前路径满足条件的最小跨区数目

&& countminfee() <feeLowBound //计算当前路径的最小花费

&& hasnocircle() //没有出现环

)

{

if(findscheme) //找到一个方案，则这个方案必然小于lowbound

{

feeLowBound=countminfee();

record(route);//更新界，记录路径

backward();//回溯

}

else //前进

{

movetonextchild();

continue;

}

}

else {

backward();//回溯&剪枝

}

}

}

解：

1. 构造一棵二叉搜索树，根节点表示起始站点，叶节点表示其余站点，每层的分支对应于是否把某条边添加到解集中的决策。沿左分支前进表示选择该边，沿右分支前进表示不选该边。例如，若ai左子树为aj，则表示选择边aij到解集中，若ai的右子树为aj，表示不添加aij到解集。
2. 前进：总是从树的根节点开始，当线路费用小于DMAX并且地井数目小于UMAX且图中无环时前进，当下行时总是先沿着左分支进行。

分支：当一个节点下一步有多个选择时分支。

回溯：当有如下情况之一时进行回溯：

1. 当前挑选的边使得解不可行；
2. 已经找到一个解

当从左分支回溯到某顶点时，接着沿其右分支向下进行；

当从右分支回溯到某顶点时，接着回溯到其父顶点。

剪枝：当前路径不满足可行性要求或超出界时。

1. 题目要求设计一个无环网络使得线路总数不超过DMAX、地井总数不超过UMAX。所以问题初始时的界为：线路总数=DMAX且地井总数=UMAX。

当扩展出的节点满足：1.线路无环；2.线路总数不大于DMAX；3.地井总数不大于UMAX；4.线路覆盖所有站点时，更新问题的上届，此时获得了一个比原来更紧的界。

分支（线路e，是左子树还是右子树）

begin

if (是左子树)

begin

add 线路e to 选择的线路集合；

if（选择的线路集合中线路条数＞49 or 选择的线路集合会形成环 or 当前敷设费用＞最小费用 or 当前总地井数＞UMAX or 当前跨区线路数＞DMAX） return；

if （选择的线路集合中线路条数为49 && 这些线路是联通的）

begin

最小费用 = 当前敷设费用；

最优方案 = 选择的线路集合；

return；

end；

end；

线路next = 线路e的下一条未访问的线路；

线路next访问标记 = 已访问；

当前敷设费用 += 线路next的敷设费用；

当前总地井数 += 敷设线路next需要建设的地井数；

if（线路next是跨区线路） 当前跨区线路数++；

分支（线路next，左子树）；

当前敷设费用 -= 线路next的敷设费用；

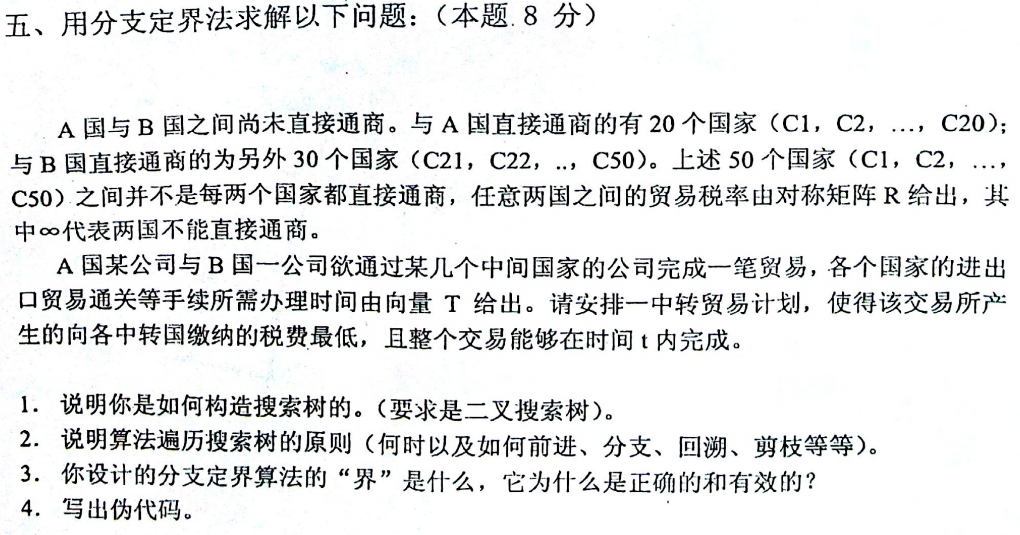
当前总地井数 -= 敷设线路next需要建设的地井数；

if（线路next是跨区线路） 当前跨区线路数--；

分支（线路next， 右子树）；

线路next访问标记 = 未访问；

end；



税费题的某种深搜解法：

1)

可以根据除A外的51个国家定义一棵若干层二叉搜索树。每个节点的左分支表示选择其代表的国家为下一个贸易顺序上的国家，右分支则表示不选择。构造搜索树需要两个辅助变量，

之前的贸易顺序S（s为S的最后一个国家）和这一轮否决的国家V。任取可以和s国贸易的国家c（不属于S和V）置于树的当前生成位置，然后用(S' = <S, c>和V' = 空集)生成左子树，用(S' = S和V' = V ∪ {c})生成右子树。如果c不存在或者s = B则终止当前子树的生成。如此反复可以建立一棵二叉搜索树。

2)

前进：当前节点未被剪枝并且仍有子节点即可继续前进。

分支：先遍历左分支，后遍历右分支。

回溯：左右分支都被遍历时返回父节点。

剪枝：

剪枝条件如下：

1。当前税费 + s国与B国贸易的最小税费 >= 已知最优方案的税费

2。当前时间 + s国与B国贸易的最短时间 > t

3)

子问题的下界为税费下界和时间下界，均由dijkstra算法算法得到，表示某国与B国贸易的最小税费和最短时间。两个结果均由弱化限制的方法得到，所以是正确的，计算复杂度也不高，当然有效。

父问题的上界是已知最优方案的税费，显然正确有效。

4)

使用Dijkstra算法得到子问题下界：

     1、税费下界：某国与B国贸易的最小税费，顺便记录对应的事件和贸易顺序

     2、时间下界：某国与B国贸易的最短时间

search(<A>, <V>)

返回最优结果

def search(贸易顺序S, 否决的国家V):

     令s为S的最后一个国家

     判断S是否合格，条件如下：

          1.当前税费 + s国与B国贸易的最小税费 < 已知最优方案的税费

          2.当前时间 + s国与B国贸易的最短时间 <= t

     如果合格：

          当前时间 + s国与B国贸易的最小税费对应的时间 <= t:

               记<S,s国与B国贸易的最小税费对应贸易顺序(不包括s)>为已知最优

          否则对与s国贸易的不属于S和不属于V的国家c：

               search(<S, c>, V)

               search(<S>, <V, c>)

解：

1. 构造二叉搜索树，根节点为A国，其余节点代表剩余国家。每层分支对应于是否在两国之间中转贸易。沿左分支前进表示在两国之间中转贸易，沿右分支前进表示不在两国之间中转贸易。例如，若ai的左子树为aj，则表示在ai和aj两国之间中转贸易；若ai的右子树为aj则表示不在ai和aj两国之间中转贸易。
2. 前进：总是从树的根节点开始，当交易税费小于目前得到解的税费值且办理手续时间小于T时前进，当下行时总是先沿着左分支前进。

分支：当一个节点下一步有多个选择时分支。

回溯：当满足下列情况之一时回溯：

1. 当前挑选的边使得解不可行；
2. 已经找到一个解。

当从左分支回溯到某顶点时，接着沿其右分支向下进行；

当从右分支回溯到某顶点时，接着回溯到其父顶点。

剪枝：当前路径不满足可行性要求或超出界时。

（3）问题的上界是目前已经得到的一个可行解的总税费，当产生更好的界的时候更新界。