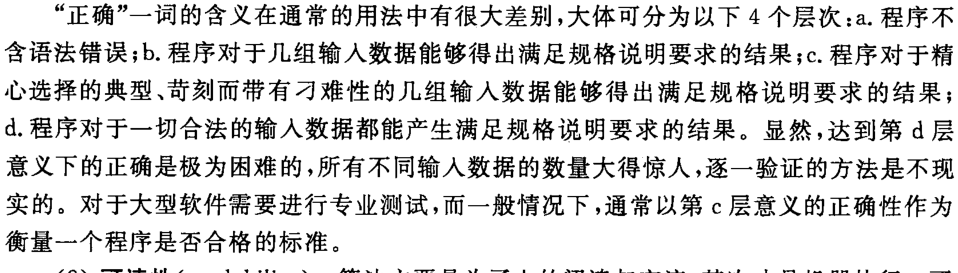


**×**

贪婪技术的核心：所做的每一步选择都必须满足：(1) 可行的，即它必须满足问题的约束； (2) 局部最优，即它是当前步骤中所有可行性选择中最佳的局部选择；(2) 不可取消，即一旦做出，在算法的后续步骤就无法改变了。

**×**



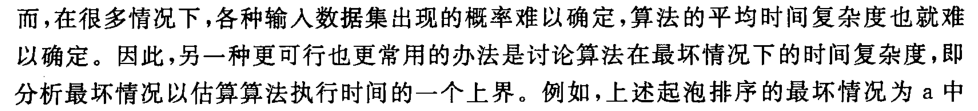


**×**

决策和策略：决策是指某阶段状态给定以后，从该状态演变到下一个阶段某状态的选择；由每段的决策组成的决策函数序列就称为全过程策略，简称策略。



**√**



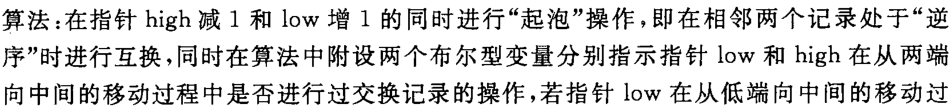


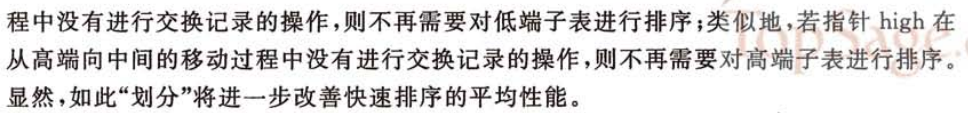
**√**

针对所要做的选择构造一棵所谓的状态空间树，树的每一层节点代表了对解的每一个分量所做的选择；用DFS（深度优先法）搜索状态空间树；在状态空间树中的任一个节点，满足一定条件的情况下，搜索回溯。



**×**





期望运行时间：Θ(nlgn)；但与输入无关，最坏情况出现在随机数生成结果恰好使数组有序----概率极小。



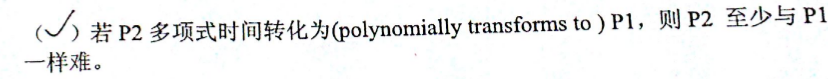
**×**

P中所有问题均属于NP。



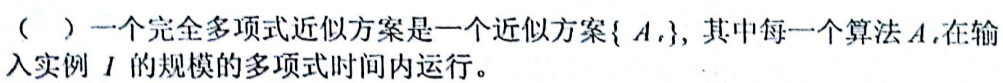
**×**

NP完全问题至少同其它所有NP问题一样难



**×**

P1多项式时间转化为P2，则P2至少与P1一样难；总是可以在可比时间内用P2之Algo2求解P1.



**×**

一个多项式近似方案(PAS)是一个近似方案{AƐ},其中每一个算法AƐ 在输入实例I的规模的多项式时间内运行;一个完全多项式近似方案(FPAS)是一个近似方案{AƐ},其中每个算法 AƐ

在以输入实例的规模和1/Ɛ两者的多项式时间内运行.

A description...

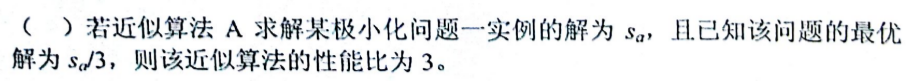
**×**

n-1

A description...

**√**

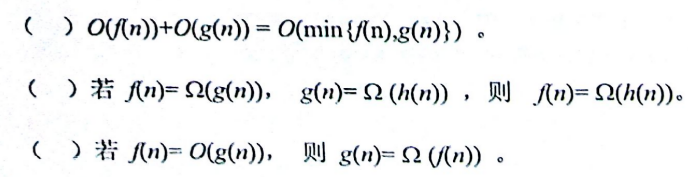
Las Vegas总是要么给出正确的解，要么告知无解。



**×**

对于一个f最小化的问题，可以用, 作为sa的精确度量。对于问题的所有实例，它们可能的r(sa)的最佳（也就是最低）上界，被称为该算法的性能比，计作RA。

性能比是一个来指出近似算法质量的主要指标，我们需要那些RA尽量接近1的近似算法。



**× √ √**

***O*(*f*(*n*))+*O*(*g*(*n*)) = *O*(max{*f*(n),*g*(*n*)})**

*O*(*f*(*n*))+*O*(*g*(*n*)) = *O*(*f*(n)+*g*(*n*))

*O*(*f*(*n*))\**O*(*g*(*n*)) = *O*(*f*(n)\**g*(*n*))

*O*(*cf*(*n*)) = *O*(*f*(n))

*f*(*n*)= *O*(*g*(*n*))， *g*(*n*)= *O* (*h*(*n*)) ⇒ *f*(*n*)= *O* (*h*(*n*))；

***f*(*n*)= Ω(*g*(*n*))， *g*(*n*)= Ω (*h*(*n*)) ⇒ *f*(*n*)= Ω(*h*(*n*))；**

*f*(*n*)= Θ(*g*(*n*))， *g*(*n*)= Θ(*h*(*n*)) ⇒ *f*(*n*)= Θ(*h*(*n*))；

*f*(*n*)= Θ(*g*(*n*)) ⇔ *g*(*n*)= Θ (*f*(*n*)) .

***f*(*n*)= *O*(*g*(*n*)) ⇔ *g*(*n*)= Ω (*f*(*n*)) ；**

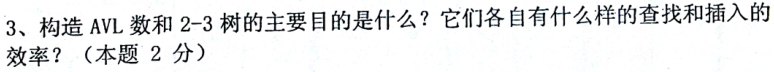
*f*(*n*)= Θ(*f*(*n*))；

*f*(*n*)= *O*(*f*(*n*))；

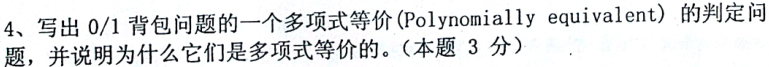
*f*(*n*)= Ω(*f*(*n*)).



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)。



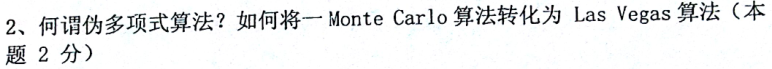
(1)当先后插入的关键字有序时，构成的二叉查找树蜕变为单支树，树的深度等于n，此时二叉查找树表现出最差的效率，为了解决这一问题，可以构造AVL树或2-3树，使树的深度减小。一棵AVL树要求它的每个节点的左右子树的高度差不能超过1。2-3树和2-3-4树允许一棵查找树的单个节点不止包含一个元素。(2) AVL树在最差情况下，查找和插入操作的效率属于Θ(lgn)。2-3树无论在最差还是平均情况下，查找和插入的效率都属于Θ(log n)。



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。

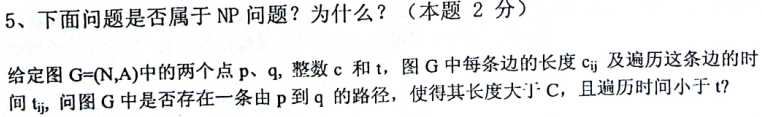


伪多项式时间算法是一种算法，它在L值的多项式时间内运行，其中L是输入实例中的最大数值。

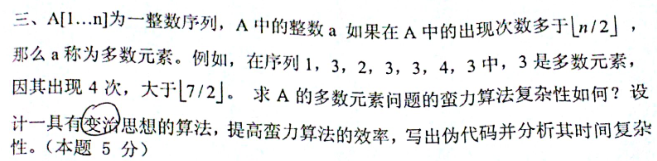
Las Vegas算法不会得到不正确的解。一旦用拉斯维加斯算法找到一个解，这个解就一定是正确解。但有时用拉斯维加斯算法找不到解。

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。



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。



蛮力算法时间复杂度：O(n2)；空间复杂度：O(1)

/\*\*

\*对长度为len的数组A，找出它的多数元素

\*/

majority(A[], len) {

HeapSort(A); // 对数组A中元素按从小到大顺序进行堆排序

i = 0;

flag = false;

while i < len {

tmp = A[i]

count = 1;

while i + count < len && A[i+count] == tmp {

count++;

}

if count > len/2 {

print(tmp+“是多数元素”)

flag = true;

break;

}

i + = count;

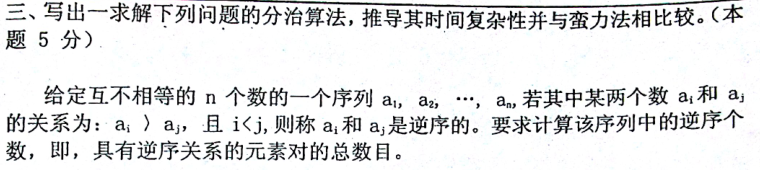
}

if flag == false

print(“没有多数元素”)

}

该算法的时间复杂性：O(nlgn)



/\*\*

\*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);

}

}

[**动态规划**](动态规划考题.docx)