# 1、名词解释

**动态规划（Dynamic planning）**

通过组合子问题的解而解决整个问题。

将待求解问题分解成若干个子问题，先求解子问题的解，然后从子问题的解得到原问题的解。适用于子问题不独立的情况，即各子问题包含公共的子问题。保存已求解子问题的答案，在需要时再找出以求得的答案，可以避免大量重复计算。**步骤**：1、找出最优解的性质，刻画其结构特征2、递归的定义最优值3、以自底向上的方式计算最优值4、根据计算最优值时得到的信息，构造最优解

**最优子结构（Optimal substructure）**

如果问题的最优解包含子问题的最优解，则问题展示了最优子结构。

重叠子问题：每次产生的子问题并不总是新问题，有些子问题被反复计算多次。

运用贪心策略在每一次转化时都取得了最优解。问题的最优子结构性质是该问题可用贪心算法或动态规划算法求解的关键特征。贪心算法的每一次操作都对结果产生直接影响，而动态规划则不是。贪心算法对每个子问题的解决方案都做出选择，不能回退；动态规划则会根据以前的选择结果对当前进行选择，有回退功能。动态规划主要运用于二维或三维问题，而贪心一般是一维问题

**随机扰动（Random disturbance）**

**随机算法**

分为两类，第一类称为**Las Vegas**算法，它建立的那些随机算法总是或者给出正确的解，或者无解。而另一类**Monte Carlo**算法与之相反，总是给出解，但是偶尔可能会产生非正确的解。然而，可以通过多次运行原算法，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。

**决策树（decision tree）**

决策树(Decision Tree）是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法，是直观运用概率分析的一种图解法。由于这种决策分支画成图形很像一棵树的枝干，故称决策树。

**贪心算法（greedy algorithm）**

贪心算法总是作出在当前看来最好的选择。并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优选择。虽然贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解。

**分治法(**Divide and Conquer**)**

将一个规模为n的问题分解为k个规模较小的子问题，这些子问题互相独立且与原问题相同。对这k个子问题分别求解。如果子问题的规模仍然不够小，则再划分为k个子问题，如此递归的进行下去，直到问题规模足够小，很容易求出其解为止。将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解。分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。

**时间、空间复杂性**

当给出合法输入时，为了得到输出，该算法所需要的时间和空间

算法复杂度分为时间复杂度和空间复杂度。其作用： 时间复杂度是度量算法执行的时间长短；而空间复杂度是度量算法所需存储空间的大小。

**近似算法评价标准**

近似算法的性能分析包括时间复杂度分析、空间复杂度分析和近似精度分析，其中时间（空间）复杂度的分析同精确复杂度相同。近似精度分析是近似算法特有的，它主要用于刻画近似算法给出的近似解相比于问题优化解的优劣程度。目前，存在三种刻画近似精度的度量，即近似比、相对误差界和1+ε近似。

**时间、空间复杂度衡量标准**

算法复杂度分为时间复杂度和空间复杂度。其作用： 时间复杂度是度量算法执行的时间长短；而空间复杂度是度量算法所需存储空间的大小。

**分治法思想：二分搜索**

将一个规模为n的问题分解为k个规模较小的子问题，这些子问题互相独立且与原问题相同。递归的求解这些子问题，然后将子问题的解合并得到原问题的解。

将一个给定元素x与一个已排序的数组A[low….high]的中间元素作比较，如果x< A[mid],而这里mid= ，则不考虑A[mid….high]，而对A[low….mid-1]重复实施同样的方法。类似的，如果x> A[mid],并对A[mid+1….high] 重复实施同样的方法.

**分治法求合并排序的思想与算法**

合并排序法是将两个（或两个以上）有序表合并成一个新的有序表，即把待排序序列分为若干个子序列，每个子序列是有序的。然后再把有序子序列合并为整体有序序列。即将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序，

**子串与子序列比较**

子序列是在整个字符串中只要按照顺序可以不用连续的，但是子串是指必须连续的字符串

**动态规划算法思想，可能与分治算法思想比较**

动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干子问题，先求解子问题，然后从这些子问题的解得到原问题的解。与分治法**不同**的是，适合于用动态规划法求解的问题，经分解得到的子问题往往不是相互独立的。若用分治法来解这类问题，则分解得到的子问题数目太多，以至于最后解决原问题需要耗费指数时间。然而，不同子问题的数目常常有多项式量级。在用分治法求解时，有些子问题被重复计算了许多次。如果我们能够保存已经解决的子问题的答案，而在需要 时再找出已求解的答案，这样就可以避免大量重复计算，从而得到多项式的时间算法。

**算法**

算法是指解题方案的准确而完整的描述，是一系列解决问题的清晰[指令](https://baike.baidu.com/item/%E6%8C%87%E4%BB%A4/3225201" \t "_blank)，算法代表着用系统的方法描述解决问题的策略机制。也就是说，能够对一定规范的[输入](https://baike.baidu.com/item/%E8%BE%93%E5%85%A5/32696" \t "_blank)，在有限时间内获得所要求的输出。不同的算法可能用不同的时间、空间或效率来完成同样的任务

**衡量近似算法性能最重要的标准有两个**

1. 、**算法的时间复杂性**：近似算法的时间复杂性必须是多项式阶的，这是设计近似算法的基本目标；（2）、**解的近似程度**：近似最优解的近似程度也是设计近似算法的重要目标。近似程度可能与近似算法本身、问题的规模，乃至不同的输入实例都有关。

**归纳法**

这种方法基于数学归纳法证明技术。给出一个带有参数n的问题，用归纳法设计一个算法，如果我们知道如何求解带有参数小于n的问题，那么我们的任务就化为如何把解法扩展到带有参数n的实例。

**分支限界法**

采用广度优先遍历产生状态空间树的结点，并使用剪枝函数的方法称为分支限界法。

**回溯法**

具有限界函数的深度优先搜索方法。回溯法在问题的解空间树中，按深度优先策略，从根结点出发搜索解空间树。算法搜索至解空间树的任意一点时，先判断该结点是否包含问题的解：如果肯定不包含，则跳过对该结点为根的子树的搜索，逐层向其祖先结点回溯；否则，进入该子树，继续按深度优先策略搜索。

**子集树：**

当所给的问题是从n个元素的集合S中找出满足某种性质的子集时，相应的解空间即是一棵子集树。

**排列树：**

当所给的问题确定n个元素满足某种性质的排列时，相应的解空间树成为排列树。

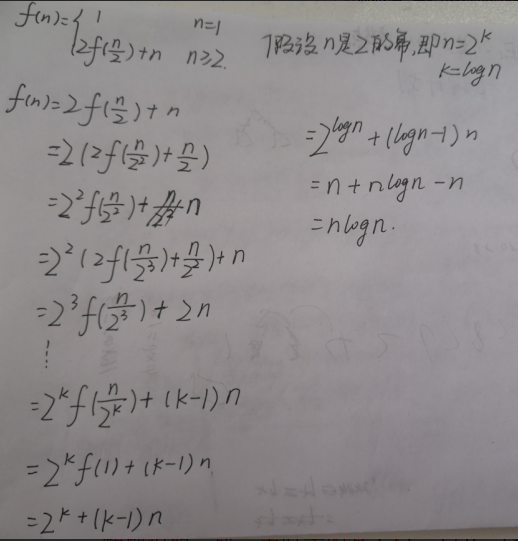
**备忘录方法**

# 公式推导

**P107，王涛春老师以107页公式推导举例说明，不知今年是不是推导这公式，仔细看看推导过程**

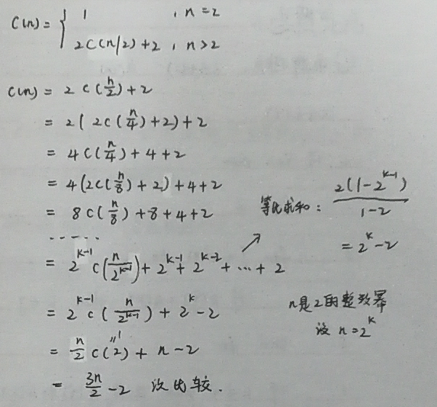
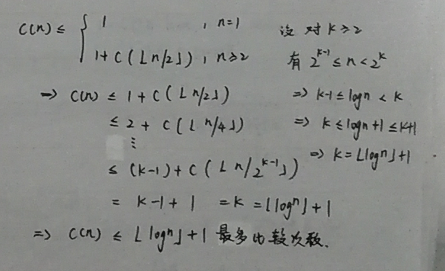
**1、递推关系，求时间空间复杂度公式推导：**

****

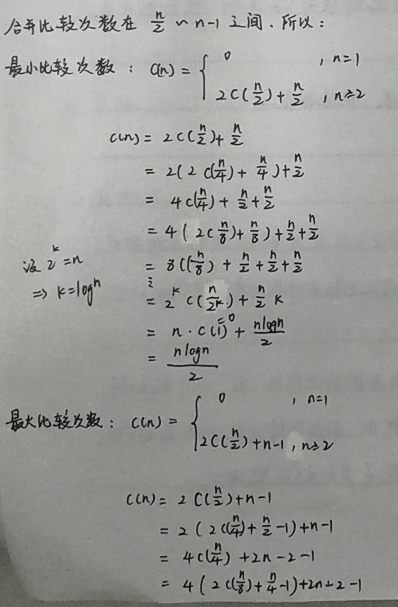


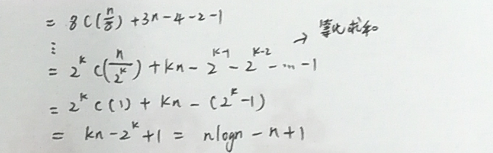
**2、找最大最小值比较次数：**

**3、二分搜索比较次数：**



**4、合并算法比较次数：**





**1、动态规划算法：矩阵链乘法**

**2、0-1背包问题：与139页例7.6相似**

**3、最短路径问题（迪杰斯特拉）**

**4、最小耗费生成树（克鲁斯卡尔，普里姆）**

# 算法设计

**1、P128：6.50、6.51**

**2、P227：13.9、13.10**

**3、P242：14.13、14.14**

**特别说明：研二当时考的是6.51、13.9、14.13**

**6.50、设计一个分治算法，判定两棵给定的二叉树T1和T2是否相同**

int EqualBtr(bitreptr t1, bitreptr t2)

{

if(t1 == NULL && t2 == NULL)

return 1;

if((t1==NULL&&t2!=NULL)||t1!=NULL&&t2==NULL)||

(t1->data!=t2->data))

return 0;

hl = EqualBtr (t1->Lchild, t2->Lchild);

hr = EqualBtr (t1->Rchild, t2->Rchild);

if(hl==1&&hr==1)

return 1;

else

return 0;

}

**6.51、设计一个分治算法，求给定二叉树的高度**

int heigh(BiTree t){

if (t== NULL) return 0;

else{

int m=heigh(t->lchild);

int n=heigh(t->rchild);

return m>n?(m+1): (n+1)

}

}

**13.9、设计一个回溯算法，生成1,2,3,…,n的全排列（着色问题）**

输入：数字1,2,3,...,n

输出：全排列

1.for k=1 to n

2. c[k] 0

3.end for

4.k 1

5.while k>=1

6. While c[k]<= n-1

7 c[k] c[k]+1

8 if c为合法着色 then output c and goto 11

9 else if c为部分解 then k k+1

10 end while

11.c[k] 0

12.k k-1 {回溯}

13.end while

**14.13、判断AB=C**

Public static boolean product(double [][]a,double [][]b,

double [][]c,int n){

cnd=new Random();

double []x=new double [n+1];

double []y=new double [n+1];

double []z=new double [n+1];

for(int i=1;i<=n;i++){

x[i]=cnd.random(2);

if(x[i]==0) x[i]=-1;

}

mult(b,x,y,n);

mult(a,y,z,n);

mult(c,x,y,n);

for(int i=1;i<=n;i++){

if(y[i]!=z[i]) return false;

else

return ture;

}

}

**14.14、AB=E**

Public static boolean product(double [][]a,double [][]b,int n){

cnd=new Random();

double []x=new double [n+1];

double []y=new double [n+1];

double []z=new double [n+1];

Double []e=new double [n+1];

for(int i=1;i<=n;i++){

x[i]=cnd.random(2);

if(x[i]==0) x[i]=-1;

}

for(int i=1;i<=n:i++){

for(int j=1;j<=n;j++)

if(i==j) e[i][j]=1

else e[i][j]=0;

}

mult(b,x,y,n);

mult(a,y,z,n);

mult(e,x,y,n);

for(int i=1;i<=n;i++){

if(y[i]!=z[i]) return false;

else

return ture;

}

}

**皇后问题**

1.for k=1 to 8

2. c[k] 0

3.end for

4.k 1

5.while k>=1

6. While c[k]<= 7

7 c[k] c[k]+1

8 if c为合法布局 then output c 退出所有循环

9 else if c为部分解 then k k+1

10 end while

11. c[k] 0

12. k k-1 {回溯}

13.end while

**着色问题(迭代法)**

n个顶点

1.for k=1 to n

2. c[k] 0

3.end for

4.k 1

5.while k>=1

6. While c[k]<= 2

7 c[k] c[k]+1

8 if c为合法着色 then output c 退出所有循环

9 else if c为部分解 then k k+1

10 end while

11. c[k] 0

12. k k-1 {回溯}

13.end while

**着色问题（递归法）**

1.for k<- 1 to n

C[k]<- 0

2.end for

3.flag = false

4.graphcolor(1)

5.if flag then output c

6.else output “no solution”

过程：graphcolor(k)

1. for color = 1 to 3
2. c[k]<-color
3. if c为合法着色 then set flag<-true and exit
4. else if c为部分的 then graphcolor(k+1)
5. end for

**14.20 最小割问题**

1.min=MAXINT，固定一个顶点P

2.从点P用类似prim的s算法扩展出“最大生成树”，记录最后扩展的顶点和最后扩展的边

3.计算最后扩展到的顶点的切割值（即与此顶点相连的所有边权和），若比min小更新min

4.合并最后扩展的那条边的两个端点为一个顶点

5.转到2，合并N-1次后结束

6.min即为所求，输出min

**二叉树叶子节点个数**

int leaf(BitTree T){

if(T==null)

return 0;

else if(T->lchild==null && T->rchild==null)

return 1;

else return leaf(T->lchild)+leaf(T->rchild);

}

void LeafCountBinaryCount(Node \*node){

int leafCount = 0;

if (node == NULL)

 {

 return;

 }

  if (node ->lChild == NULL && node ->rChild == NULL)

 {

 leafCount++;

 }

  LeafCountBinaryCount(node->lChild);

 LeafCountBinaryCount(node->rChild);

  return;

}