**最优子结构的定义：**如果一个问题的最优解包含其子问题的最优解，则该问题具备最优子结构性质。

**具备最优子结构的条件：**分解性：可以将问题分解为多个子问题。无后效性：子问题的解不会受到其他决策的影响，即子问题的最优解不会因为其他部分的决策而改变。

相同点：递归子结构，将待求解的问题分解成若干个规模较小的相同类型的子问题，先求解子问题，然后从子问题中的出原问题的解。

不同点：重叠子问题，适用于动态规划求解的问题，经分解得到的子问题往往不是互相独立的。若用分治法来解决这类问题，则分解得到的子问题数目太多，有些子问题被重复计算。

不同点：求解问题顺序。分治自顶向下，动态规划自底向上。

分治法问题的特征

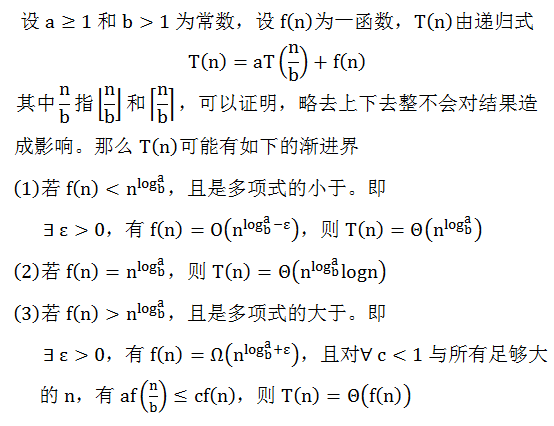
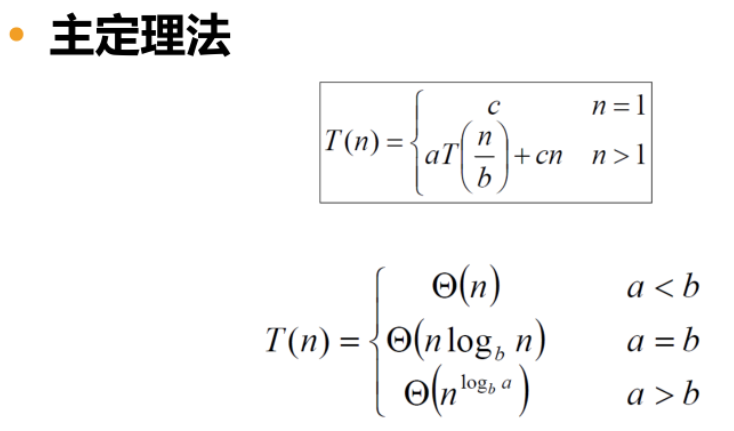
1.规模缩小到的一定程度可以解决

2.具有最优子结构性质

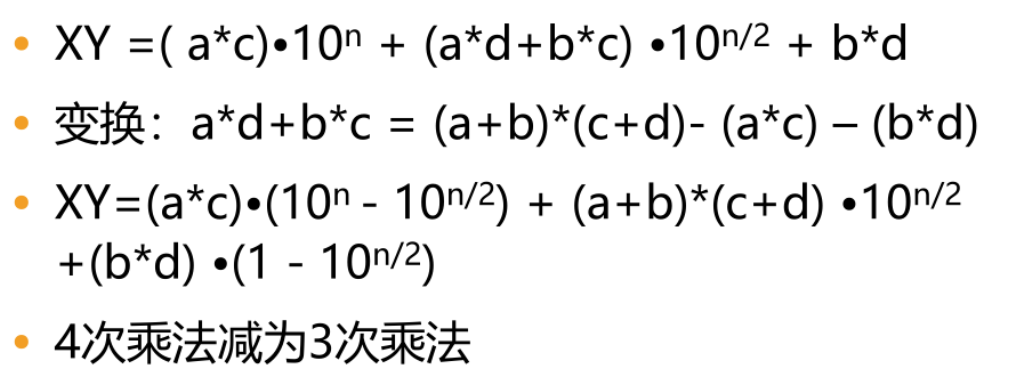
3.分解出的子问题可以合并为问题的解

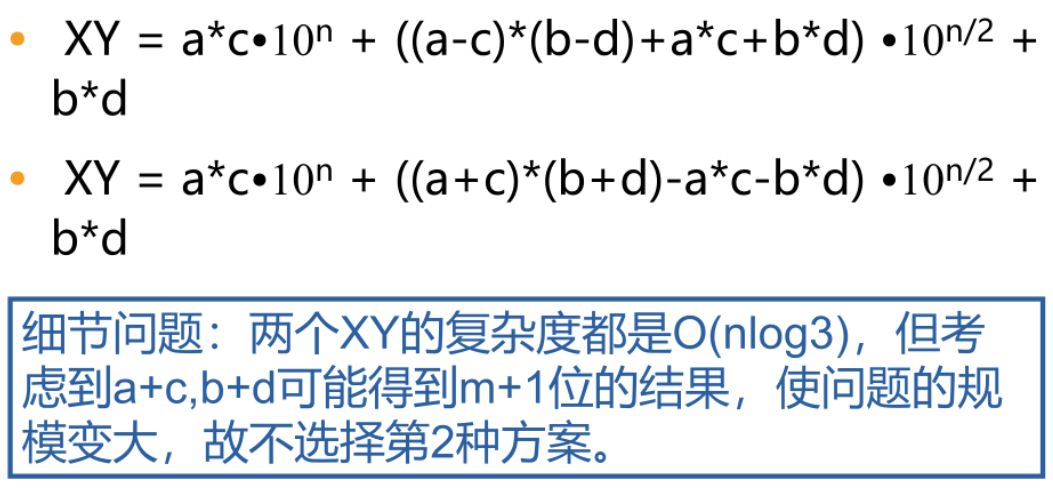
4.各个子问题相互独立

**主定理法**



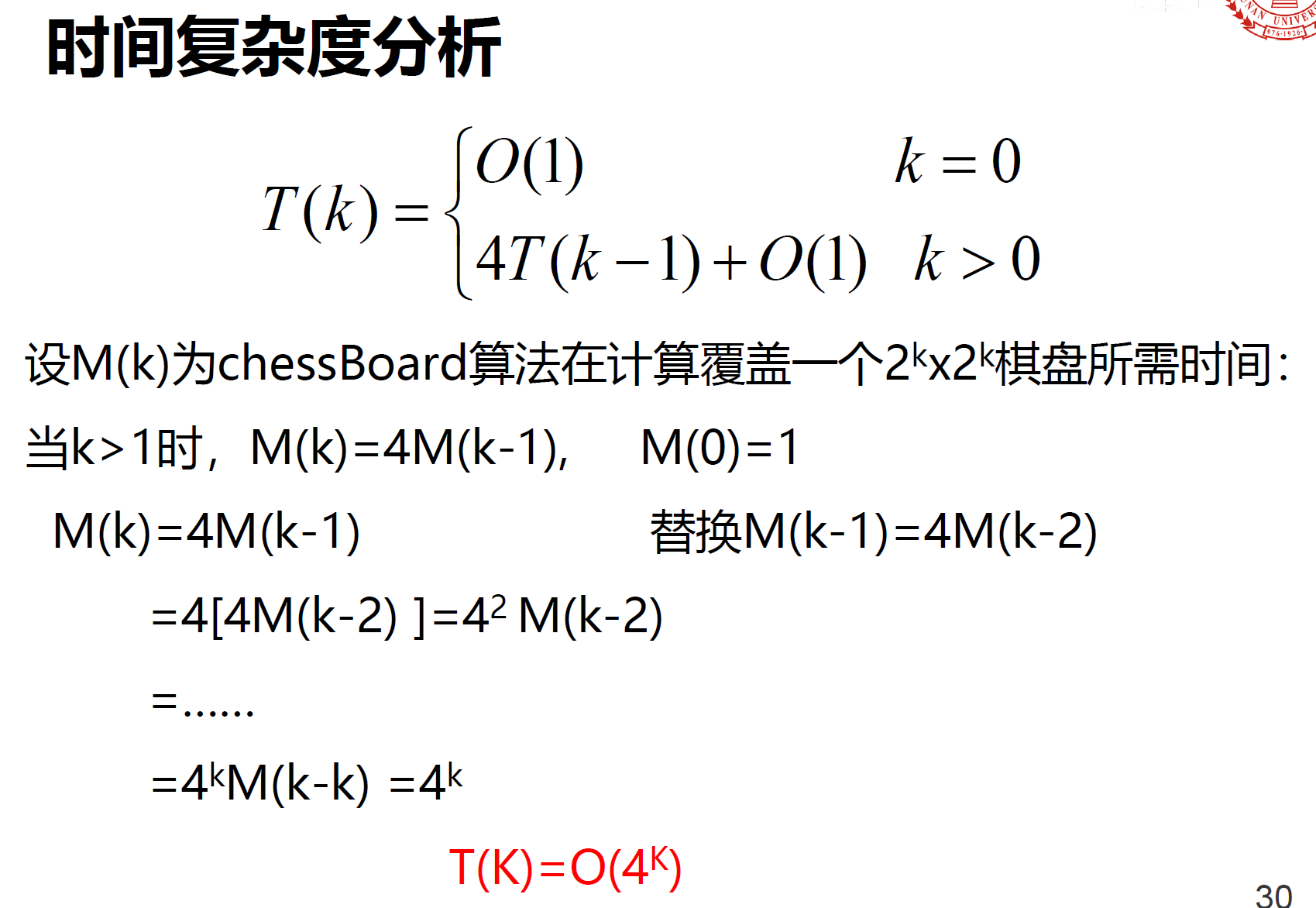
大整数乘法



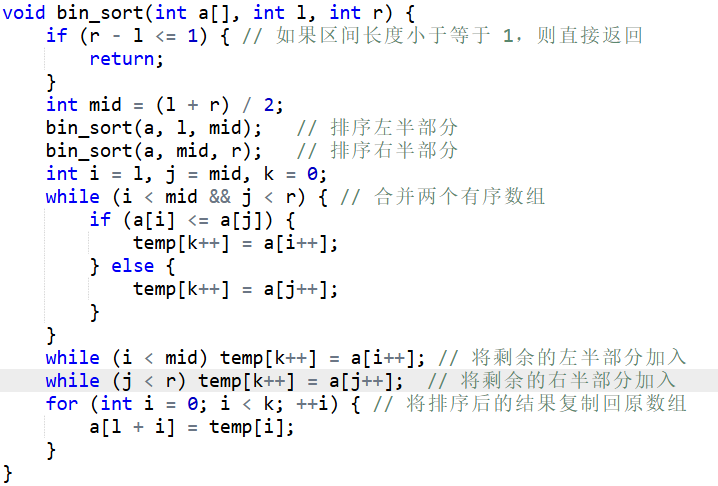


**棋盘覆盖**

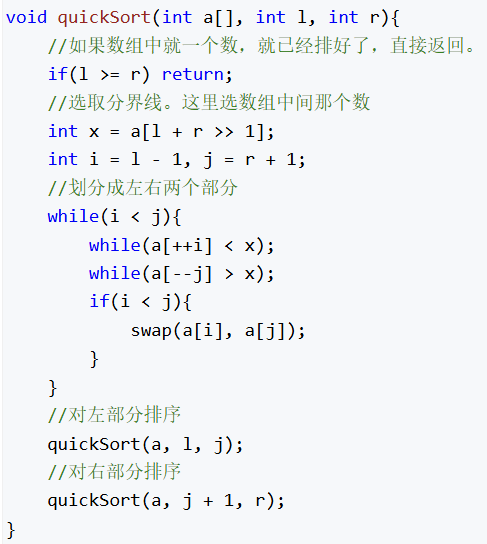
将这3个无特殊方格的子棋盘转化为特殊棋盘，可以用一个L型骨牌覆盖这3个较小棋盘的会合处。从而将原问题转化为4个较小规模的棋盘覆盖问题。



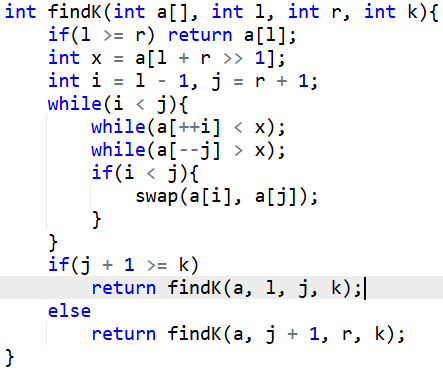
**归并排序**



快速排序

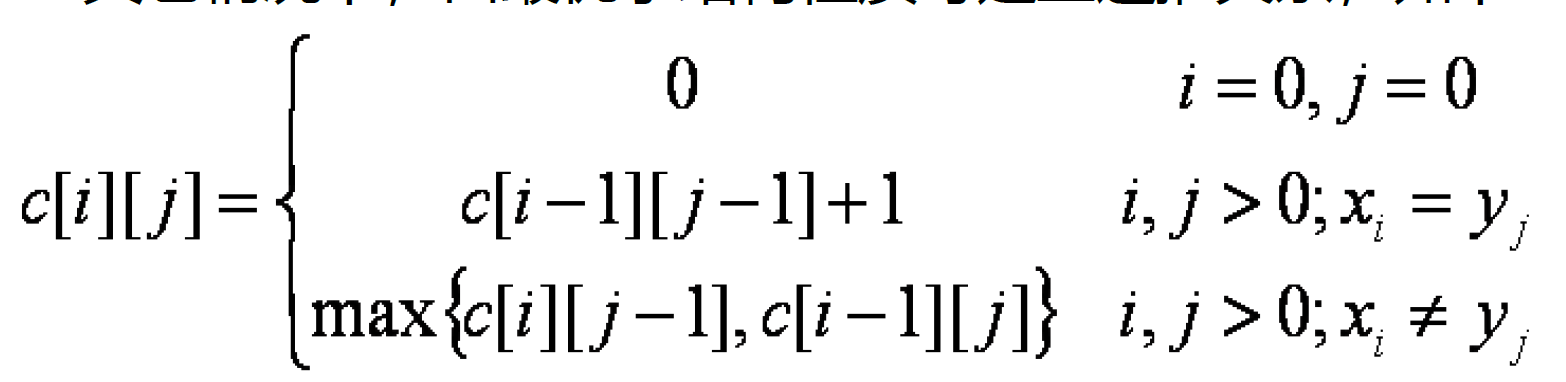


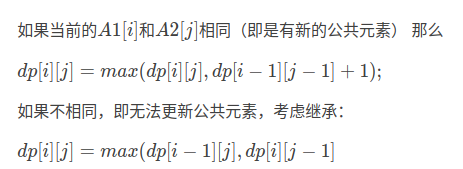
**线性时间元素选择**



**循环赛日程表**

最长公共子序列





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

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

{

dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

if(a1[i]==a2[j])

dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);

//因为更新，所以++；

}

最长上升子序列

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

{

dp[i]=1;//初始化

for(int j=1;j<i;j++)//枚举i之前的每一个j

if(data[j]<data[i] && dp[i]<dp[j]+1)

//用if判断是否可以拼凑成上升子序列，

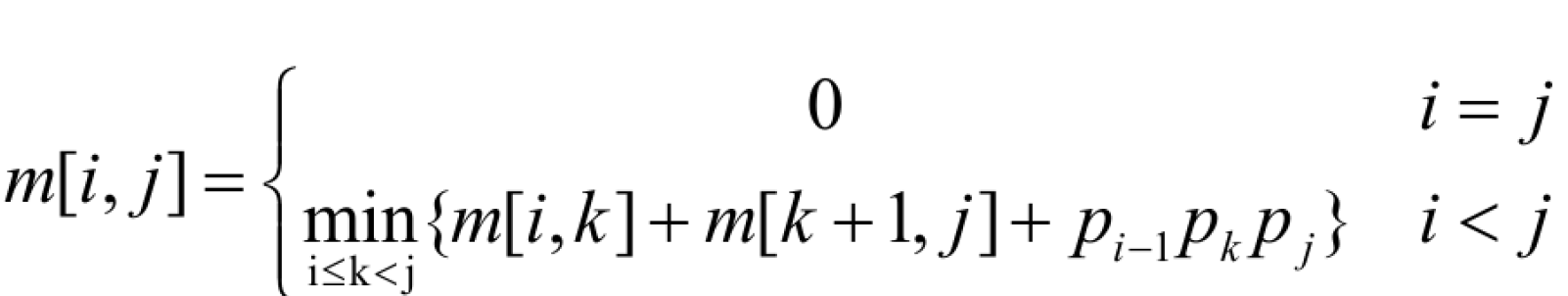
//并且判断当前状态是否优于之前枚举

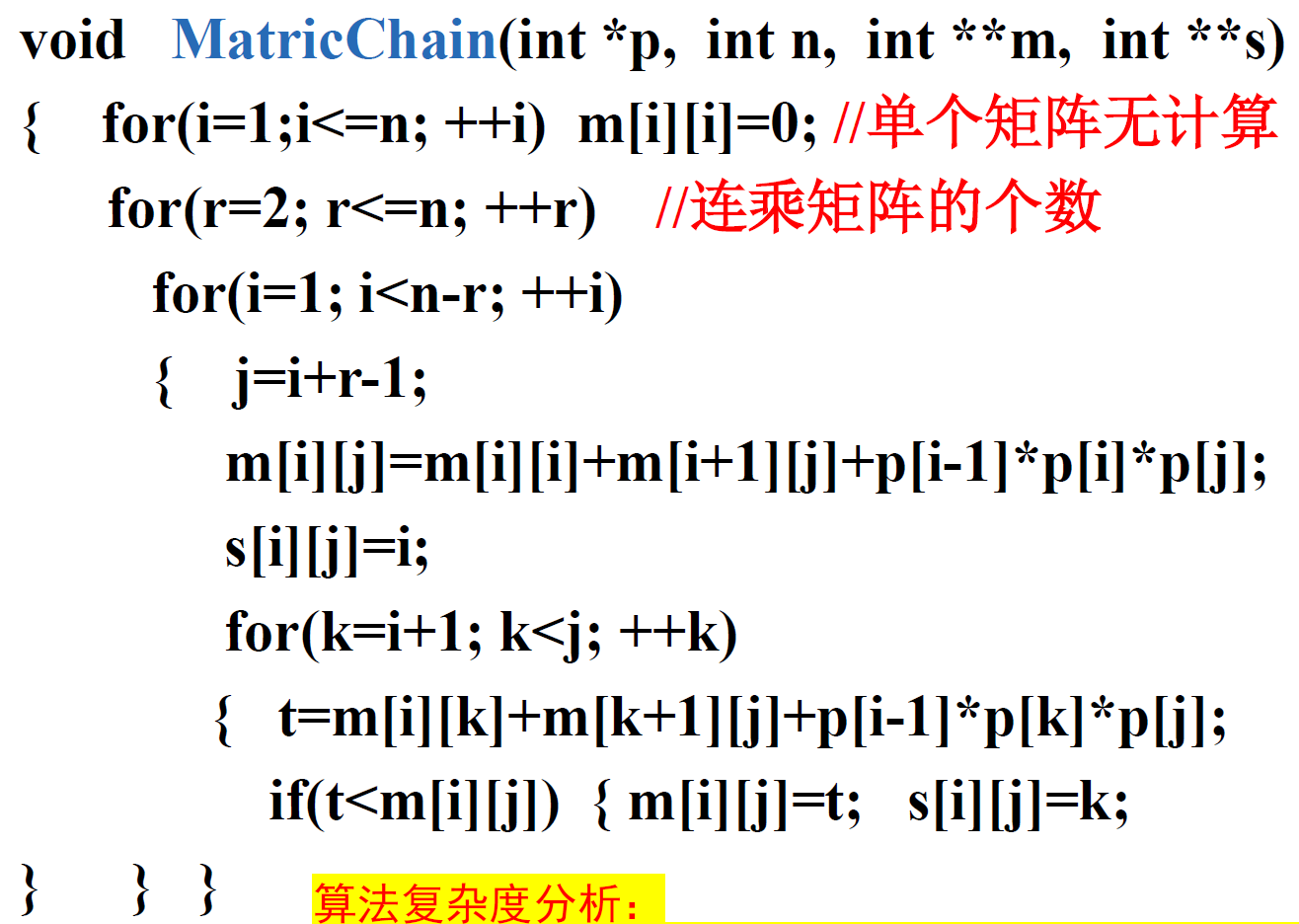
//过的所有状态,如果是，则↓

dp[i]=dp[j]+1;//更新最优状态

}

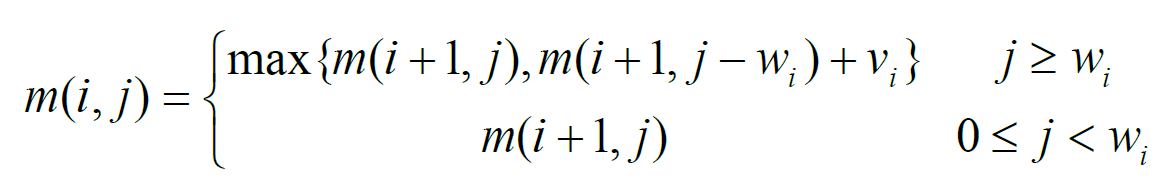
矩阵连乘





标价函数s，记录从哪个开始分割的

**01背包**



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

for (int j = W; j >= w[i]; j--)

f[j] = max(f[j], f[j - w[i]] + v[i]);

**逆序对**

void msort(int b,int e)//归并排序

{

if(b==e)

return;

int mid=(b+e)/2,i=b,j=mid+1,k=b;

msort(b,mid),msort(mid+1,e);

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

if(a[i]<=a[j])

c[k++]=a[i++];

else

c[k++]=a[j++],ans+=mid-i+1;//统计答案

while(i<=mid)

c[k++]=a[i++];

while(j<=e)

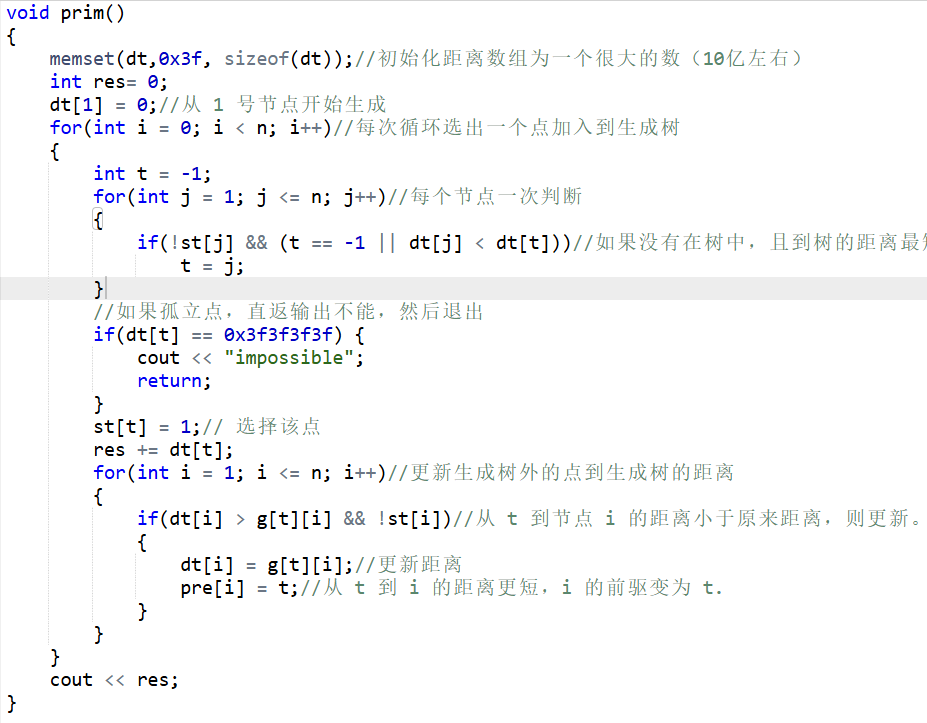
c[k++]=a[j++];

for(int l=b;l<=e;l++)

a[l]=c[l];

}

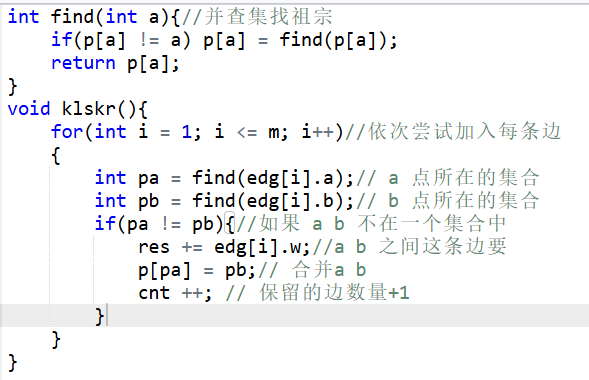
**Prim算法**

****

**Kruskal算法**

1.所有边按权值排序

2.若不形成回路，则取最小的边加入图中

****

**活动安排问题**

1.将各个活动按照活动结束时间fi排序

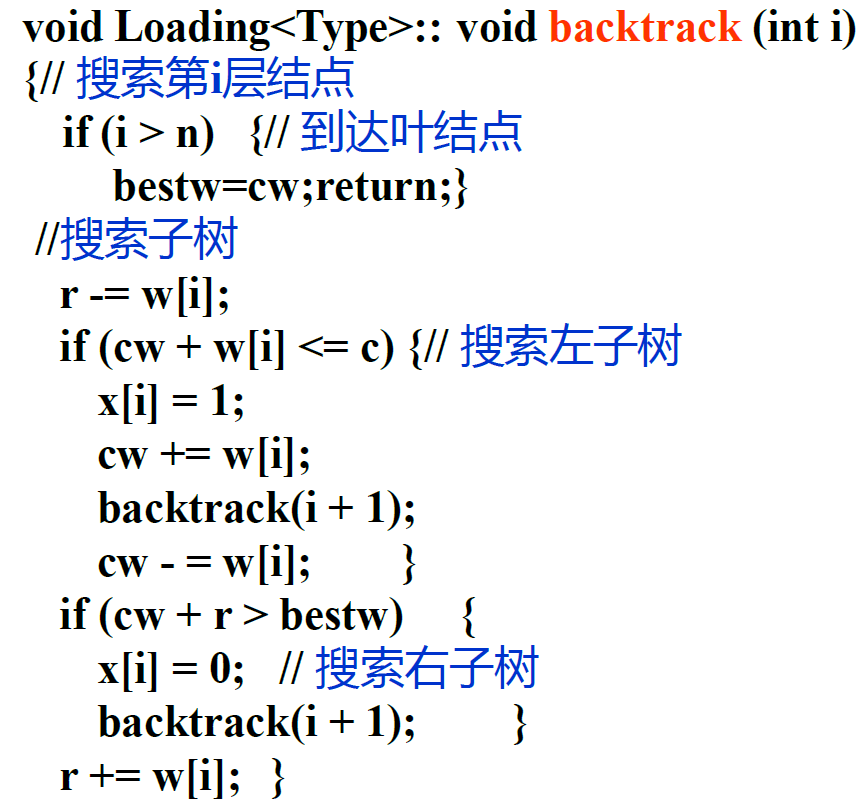
2.选择结束时间最早的为第一个活动

3.遍历剩下的n个活动寻找相容的活动

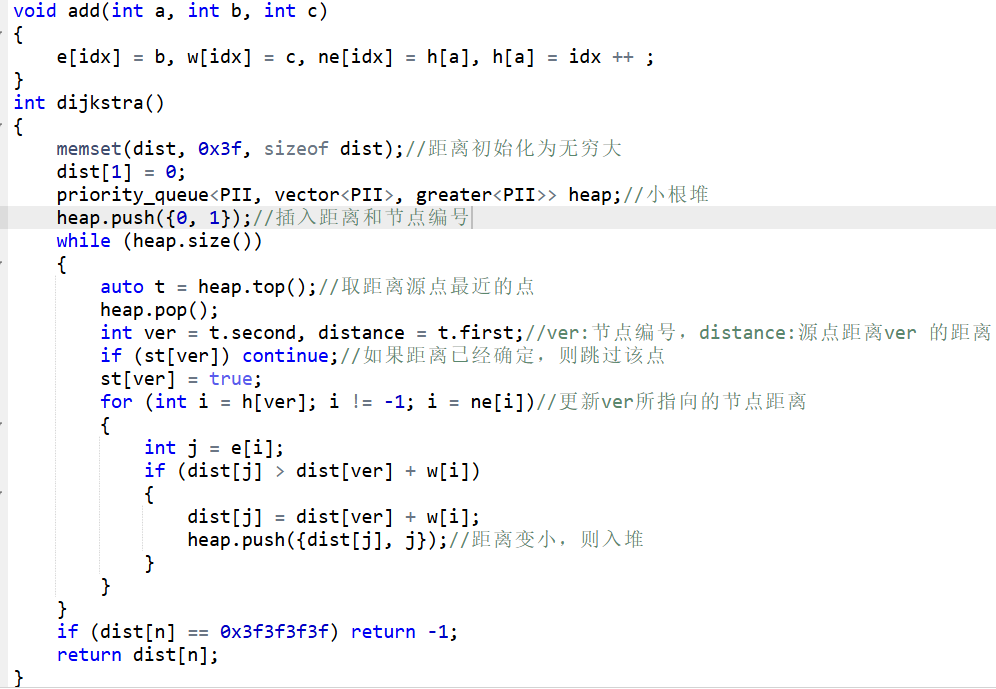
**最优前缀编码问题**

循环地选择具有最低频率的两个结点，生成一棵子树，直至形成树

**最优装载问题回溯**



**Dijkstra算法**



**背包问题回溯法**

1.当前正在处理的物品索引i；

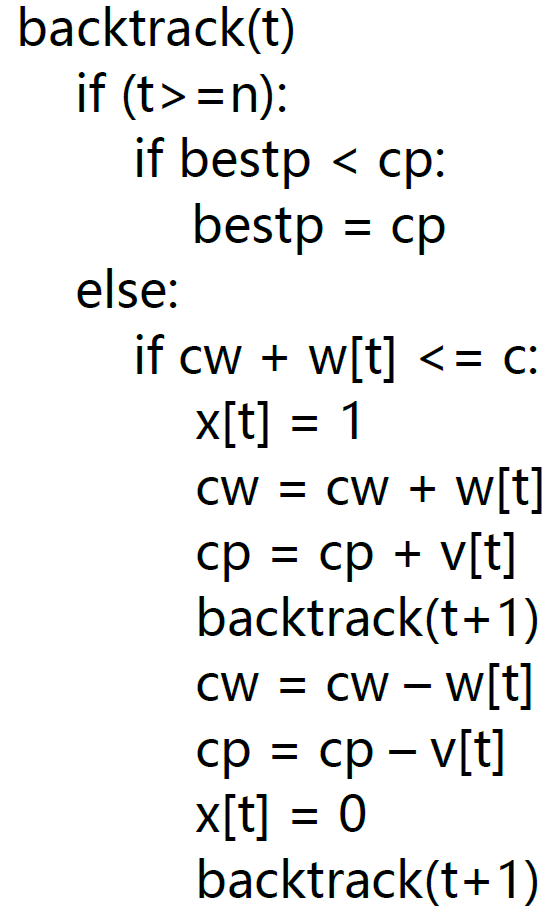
2.当前已放入背包的总重量curWeight和3.总价值 curVal；

3.记录最优解（最大价值）及对应的选法。

剪枝策略

1.当前的价值+剩余的所有价值<最优解，回退

2.加上第i个物品后超重，回退。

****

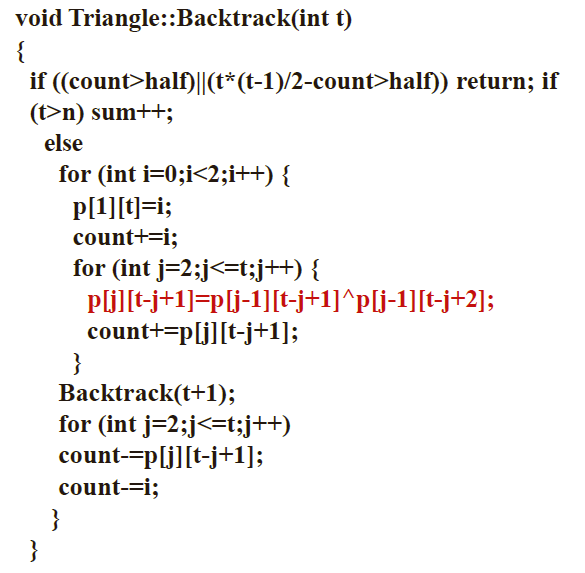
**符号三角形问题**

第一行在排列数搜索，然后生成三角形，统计对应的结果

剪枝函数：

可行性约束函数：当前符号三角形所包含的“+”个数与“-”个数均不超过n\*(n+1)/4

无解的判断：n\*(n+1)/2为奇数

****

**装载问题分支限界**

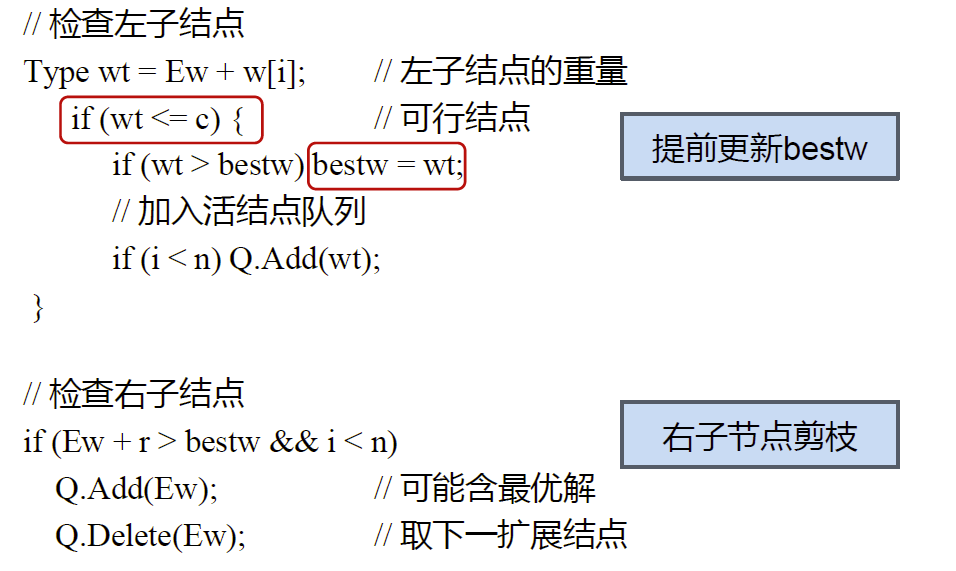
策略：

1.尽可能将第一艘船装满，再将剩余的放到第二艘船

2.如何装满第一艘船是一个子集树（01背包问题）

剪枝函数：

1. 如果加上当前货物超重，就剪枝
2. 如果当前重量+剩余的重量<最优解，就剪枝

****

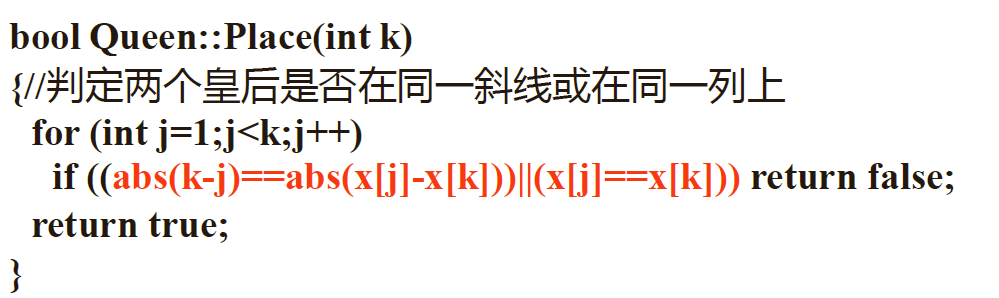
N后问题

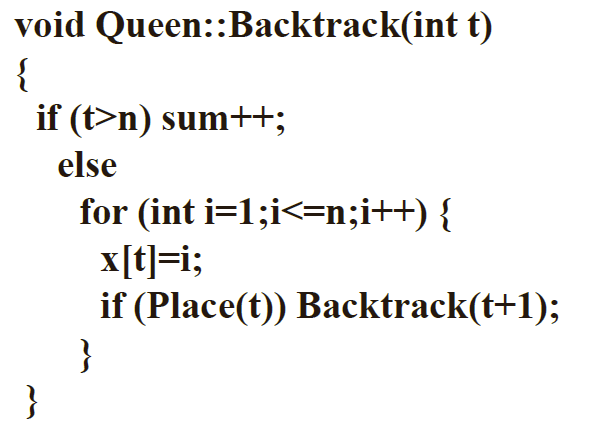
约束条件：

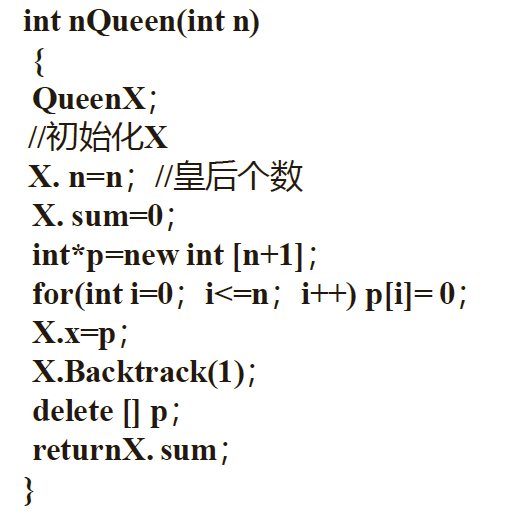
皇后不能在同一列xi≠xj

皇后不能在同一主对角线

皇后不能在同一副对角线

****

****

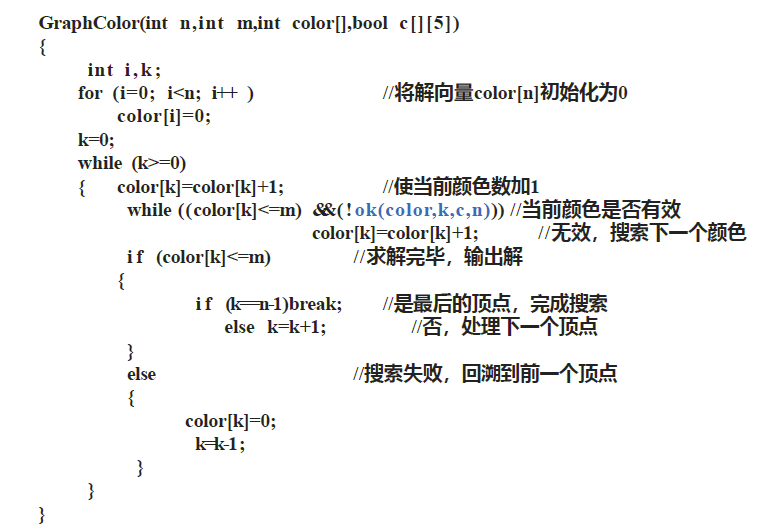
****

图的M着色问题

用排列数求解，每个子节点有m个孩子节点

约束函数：

顶点i与已着色的相邻顶点颜色不重复

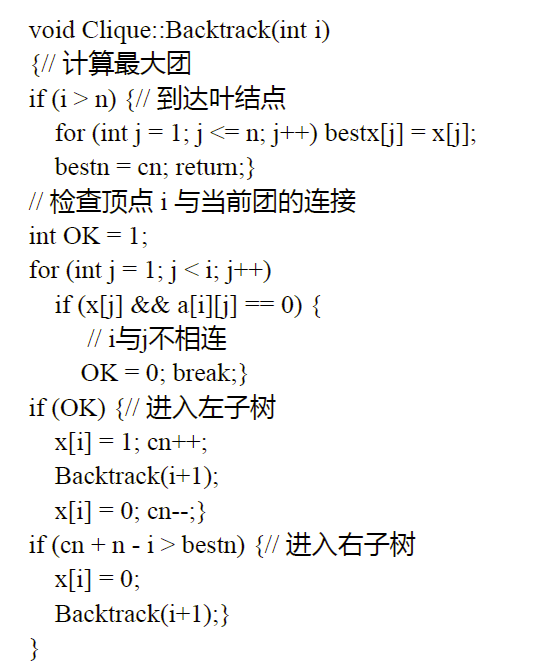
****

最大团问题

首先设最大团为一个空团，往其中加入一个顶点，然后依次考虑每个顶点，查看该顶点加入团之后仍然构成一个团，如果可以，考虑将该顶点加入团或者舍弃两种情况，如果不行，直接舍弃，然后递归判断下一顶点。

剪枝策略：

剩余未考虑的顶点数加上团中顶点数不大于当前解的顶点数

****

**回溯法效率分析**

回溯算法的效率在很大程度上依赖于以下因素：

（1）产生x[k]的时间；

（2）满足显约束的x[k]值的个数；

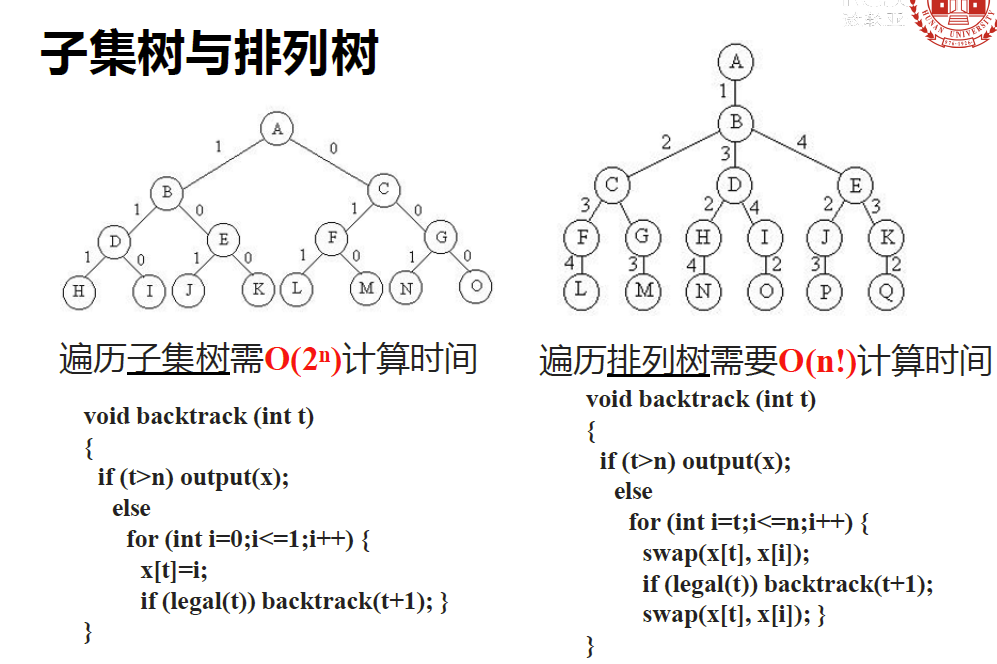
（3）计算约束函数constraint的时间；

（4）计算上界函数bound的时间；

（5）满足约束函数和上界函数约束的所有x[k]的个数。

好的约束函数能显著地减少所生成的结点数。但这样的约束函数往往计算量较大。因此，在选择约束函数时通常存在生成结点数与约束函数计算量之间的折中。

**回溯法框架**

****

**TSP问题回溯**



**随机数值算法**

1. 主要用于数值问题求解
2. 算法的输出往往是近似解
3. 近似解的精确度与算法执行时间成正比

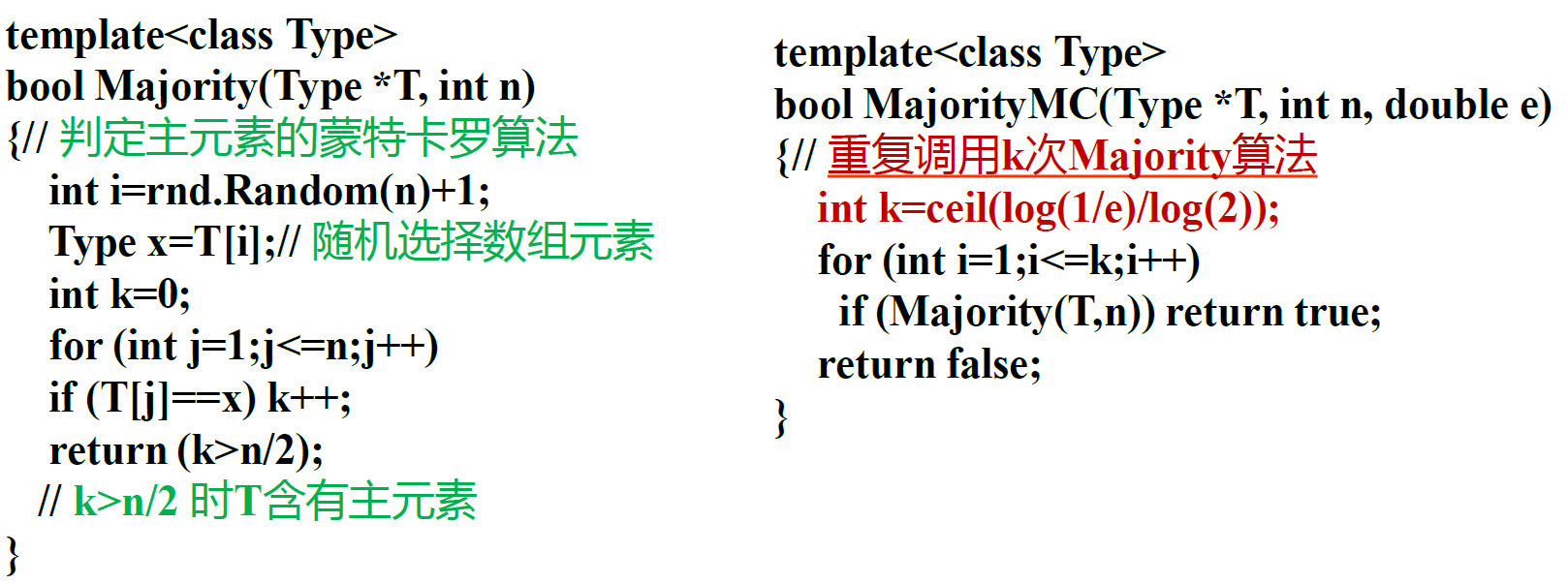
**蒙特卡罗算法**

1. 主要用于求解需要准确解的问题
2. 算法可能给出错误解
3. 获得精确解概率与算法执行时间成正比

对于一个解所给问题的蒙特卡罗算法MC(x)，如果存在问题实例的子集X使得：

1.当x属于X时，MC(x)返回的解是正确的；

2.当x属于X时，正确解是y0，但MC(x)返回的解未必是y0。称上述算法MC(x)是偏y0的算法。

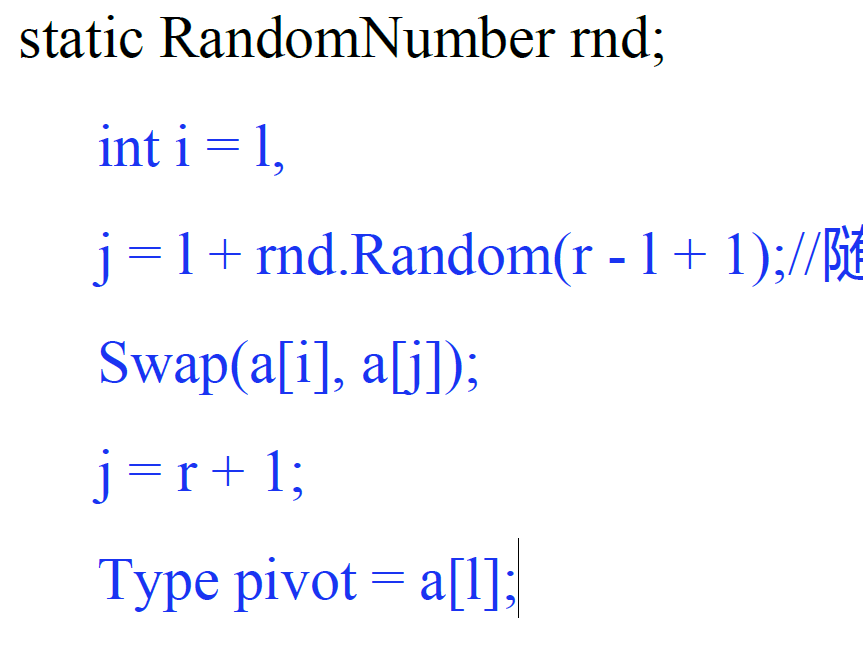


**舍伍德算法**

1. 一定能够求得一个正确解
2. 确定算法的最坏与平均复杂性差别大时, 加入随机性, 即得到Sherwood算法
3. 消除最坏行为与特定实例的联系

基本思想：获得一个随机化算法B，使得对问题的输入规模为n的每一个实例均有

舍伍德算法可获得很好的平均性



**拉斯维加斯算法**

求得的解总是正确的，但有时拉斯维加斯算法可能 始终找不到解.

1. 一旦找到一个解, 该解一定是正确的
2. 找到解的概率与算法执行时间成正比
3. 增加对问题反复求解次数, 可使求解无效的概率任意小