什么是决策树（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。

算法复杂度

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。

什么是分治法（Divide and Conquer）？什么是动态规划？二者区别？

分而治之方法与软件设计的模块化方法非常相似。为了解决一个大的问题，可以：

1) 把它分成两个或多个更小的问题；
2) 分别解决每个小问题，小问题通常与原问题相似，可以递归地使用分而治之策略来解决；
3) 把各小问题的解答组合起来，即可得到原问题的解答。

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。

动态规划也是一种分治思想（比如其状态转移方程就是一种分治），但与分治算法不同的是，分治算法是把原问题分解为若干个子问题，自顶向下求解子问题，合并子问题的解，从而得到原问题的解。动态规划也是把原始问题分解为若干个子问题，然后自底向上，先求解最小的子问题，把结果存在表格中，在求解大的子问题时，直接从表格中查询小的子问题的解，避免重复计算，从而提高算法效率。

什么是贪心算法 greedy algorithm？

答：贪心算法总是作出在当前看来最好的选择。也就是说贪心算法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。当然，希望贪心算法得到的最终结果也是整体最优的。

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。如单源最短路经问题，最小生成树问题等。在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似。

什么是回溯法？什么是分支限界法？二者有什么不同？（目标）

答：回溯法：一种选优搜索法，按选优条件向前搜索，以达到目标。但当搜索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回去再走的技术就是回溯法。大多使用深度遍历。
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。

备忘录法

备忘录方法是动态规划方法的变形，与动态规划算法不同的是，备忘录方法的递归方式是自顶向下的，一般的，当某个问题可以用动态规划法求解，但其中有相当一部分元素在整个计算中都不会被用到，我们就不需要以地推方式逐个计算，采用备忘录方法，其中的元素只是在需要计算时才去计算，计算采用递归方式，值计算出来之后将其保存起来以备用.

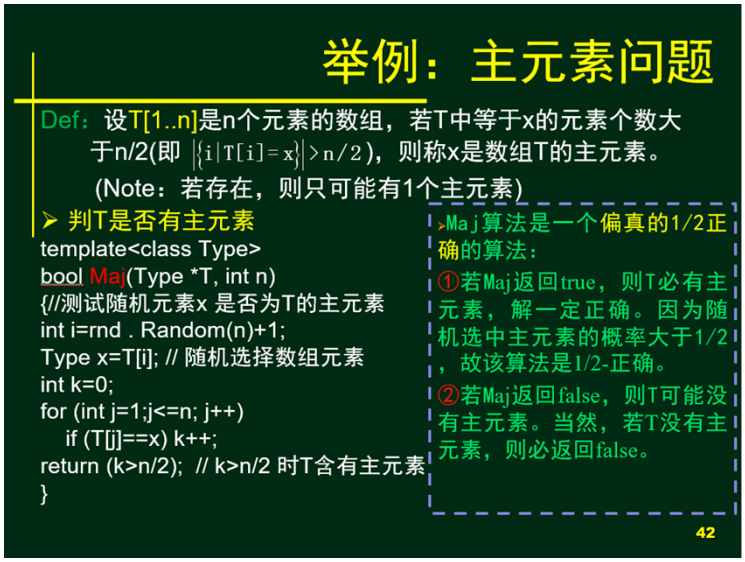
备忘录方法的控制结构与直接递归方法的控制结构相同，区别在于备忘录方法为每个解过的子问题建立了备忘录以备需要时查看，避免了相同子问题的重复求解。

```
int LookupChain(int i，int j)
{
    if (m[i][j] > 0) return m[i][j];
    if (i == j) return 0;
    int u = LookupChain(i,i) + LookupChain(i+1,j) + p[i-1]*p[i]*p[j];
    s[i][j] = i;
    for (int k = i+1; k < j; k++) {
        int t = LookupChain(i,k) + LookupChain(k+1,j) + p[i-1]*p[k]*p[j];
        if (t < u) { u = t; s[i][j] = k;}
    }
    m[i][j] = u;
    return u;
}
```

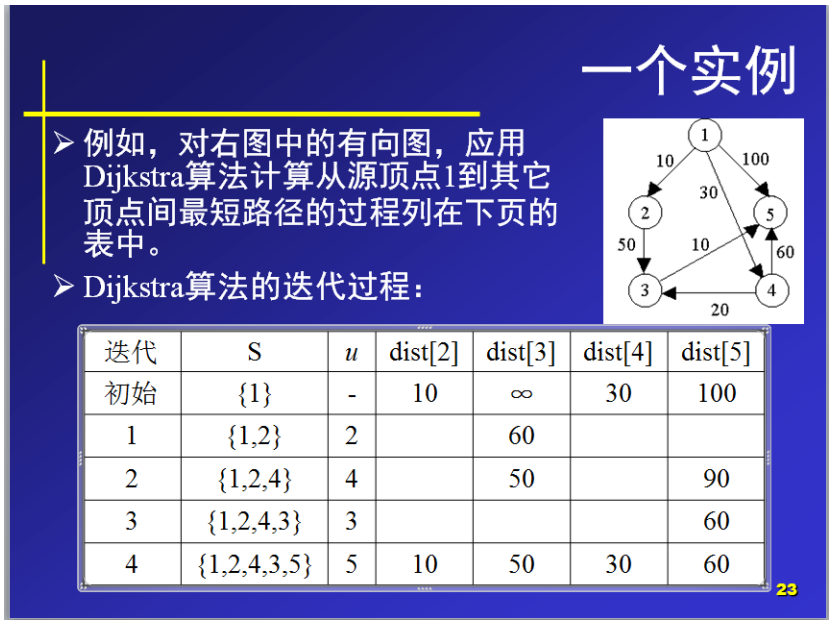
近似算法评判最重要的标准是什么？

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。

主元素问题:



Dijkstra



迭代	S	*u*	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	∞	30	100
1	{1,2}	2		60		
2	{1,2,4}	4		50		90
3	{1,2,4,3}	3				60
4	{1,2,4,3,5}	5	10	50	30	60

应用题(作业)

矩阵链相乘：

- M_1：2×3， M_2：3×6，M_3：6×4，M_4：4×2， M_5：2×7，
- 求出这 5 个矩阵相乘需要的最少乘法的次数。

14 章 13，14 课后习题：

13. 判断 AB=C

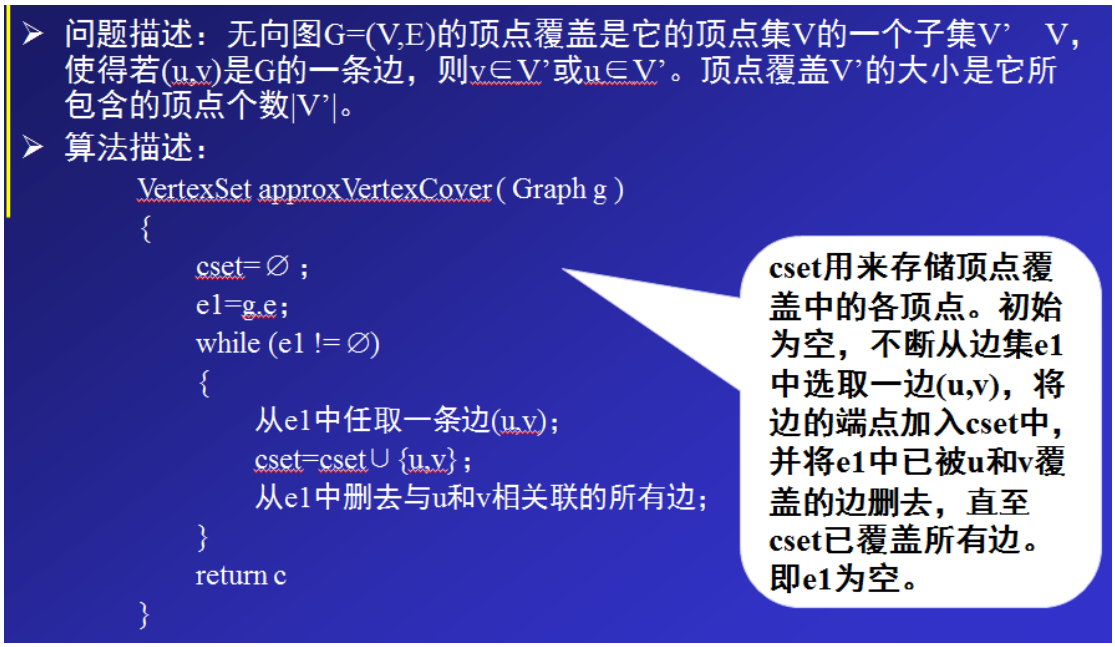
```
Public static boolean product(double [][]a,double [][]b, double [][]c,int n){
cnd=new Random();
double []x=new double [n+1];
double []y=new double [n+1];
double []z=new double [n+1];
for(int i=1;i<=n;i++){
x[i]=cnd.random(2);
if(x[i]==0) x[i]=-1;
}
mult(b,x,y,n);
mult(a,y,z,n);
mult(c,x,y,n);
for(int i=1;i<=n;i++){
if(y[i]!=z[i]) return false;
else
return ture;
}
}
```

14.14.A =B^((- - 1)即 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;
}
}
```

最小顶点覆盖：

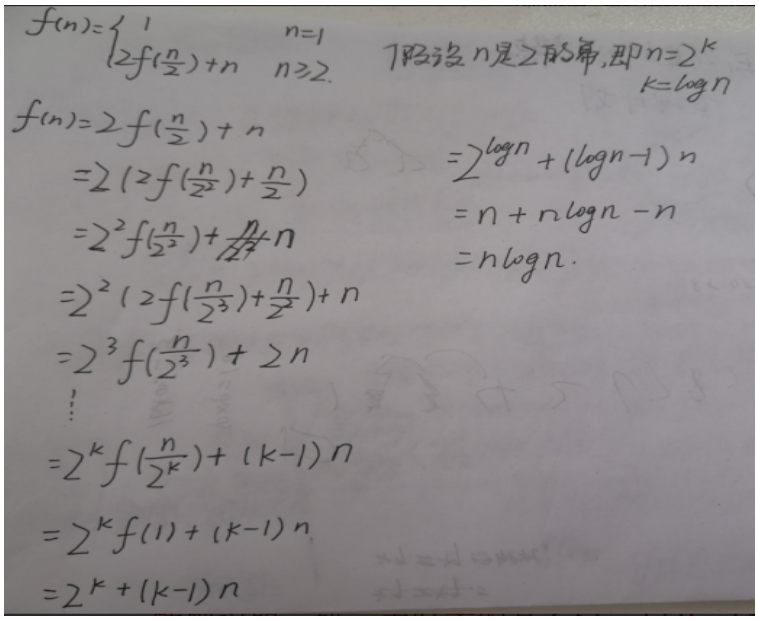


背包问题（贪心算法）：

1、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

研二当时算法公式推导题为：

$$f(n)=\begin{cases}1 & n=1\\ 2f(n/2)+n & n\geq 2\end{cases}$$



算法设计（递归、8 皇后的解空间、Monte Carlo）

二叉树高度：

设计一个分治算法，求给定二叉树的高度

```
void Btdepth(bitreptr t, int k, int &h){
//指针 t 指向二叉树的根结点，k, h 的初值均设为 0
if (t! = NULL) {
k++; //表示结点的层次
if (k>h)
h = k;
Btdepth(t->Lchild,k,h);
Btdepth(t->Rchild,k,h);
}
}
```

叶子节点总数

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);
}
```

3 着色问题

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

8 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

- 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。
- 动态规划：与分治算法的情况不一样，直接实现递推结果，导致了不止一次的递归调用，因此这种技术采取自底向上的方式递推求值，并把中间结果存储起来以便以后用来计算所需要求的解。广泛用于求解组合最优化问题。
- 最优子结构：当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。
- 随机算法：分治、动态规划、贪心法等每一个计算步骤是确定的，而随机算法每一步都是随机的。对于相同的输入每次随机算法产生的结果可能不同。
- 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。
- 贪心算法：对问题求解时，不从整体最优上加以考虑，而是做出一个看上去最优的决策（即局部最优解），并希望通过每次所做的局部最优解产生全局最优解。需要具有最优子结构和贪心选择性质。
- 分治法：一个分治算法把问题实例划分为成若干子问题（通常是两个），并分别递归地解决每个子实例，然后把这些子实例的解组合起来，得到原问题实例的解。
- 回溯法：一种选优搜索法，按选优条件向前搜索，以达到目标。但当搜

索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回去再走的技术就是回溯法。大多使用深度遍历。

2、时间、空间复杂度衡量

时间复杂度：时间复杂度实际上是一个函数，代表基本操作重复执行的次数，进而分析函数虽变量的变化来确定数量级，数量级用 O 表示，所以算法的时间复杂度为： T（n）=O（f（n））；在一个算法存在最好、平均、最坏三种情况，我们一般关注的是最坏情况。

空间复杂度： 是对一个算法在运行过程中临时占用存储空间的度量

3、分治法思想：二分搜索

给定排好序的 n 个元素，在这 n 个元素找出一特定元素 x。把有序序列分成大致相同的两部分，然后取中间元素与特定元素 x 进行比较，如果 x 等于中间元素，算法停止，如果 x 小于中间元素，则在序列的左半部分继续查找，然后在序列的左半部分重复分解和治理操作；否则，在序列的右半部分继续查找，然后在序列的右半部分重复分解和治理操作

4、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。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。

5、子串与子序列比较

子串一定要连续，子序列可以不连续，子序列就好比把子串一个个拆开，按顺序插入任意串里。

6 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。

动态规划也是一种分治思想（比如其状态转移方程就是一种分治），但与分治算法不同的是，分治算法是把原问题分解为若干个子问题，自顶向下求解子问题，合并子问题的解，从而得到原问题的解。动态规划也是把原始问题分解为若干个子问题，然后自底向上，先求解最小的子问题，把结果存在表格中，在求解大的子问题时，直接从表格中查询小的子问题的解，避免重复计算，从而提高算法效率。