NP 问题是什么，画出 p，np，npc 的关系:

P Problem

如果一个问题可以找到一个能在多项式的时间里解决它的算法，那么这个问题就属于P问题，即算法的时间复杂度是多项式级的。比如n个数中间找到最大值，或者n个数排序之类的。

NP Problem

NP问题的另一个定义是可以在多项式的时间里猜到一个解的问题，例如求问图中起点到终点是否有一条小于100个单位长度的路线，随便选一条，如果算出来路径小于100，那么就猜到了一个解，也就是说如果你运气足够好的话就可以在多项式时间内解决这个问题。当然猜的前提是问题存在解。

再比如Hamilton问题，给定一幅图，是否能找到一条经过每个顶点一次且恰好一次最后又走回来的路，每条路都可以在多项式时间内判断这条路是否恰好经过了每个顶点，所以也是NP问题。

很显然，所有的P类问题都是NP问题，能在多项式时间内解决，必然能多项式地验证一个解。（NP是否等于P这个问题貌似还没有定论？）

NPC Problem

约化

为了说明NPC问题，我们先引入一个概念——约化(Reducibility，有的资料上叫"归约"（Reduction））。

约化的概念:
约化的标准概念：如果能找到这样一个变化法则，对任意一个程序A的输入，都能按这个法则变换成程序B的输入，使两程序的输出相同，那么我们说，问题A可约化为问题B，即可以用问题B的解法解决问题A，或者说，问题A可以"变成"问题B。 如：一元一次方程可以"归约"为一元二次方程。

约化的性质:
约化具有一项重要的性质：约化具有传递性。如果问题A可约化为问题B，问题B可约化为问题C，则问题A一定可约化为问题C。

约化的意义:
问题A可约化为问题B"有一个重要的直观意义：B的时间复杂度高于或者等于A的时间复杂度。也就是说，问题A不比问题B难。

约化的要求:
我们所说的"可约化"指的是可"多项式时间地"约化(Polynomial-time Reducible)，即变换输入的方法是能在多项式的时间里完成的。约化的过程只有用多项式的时间完成才有意义。

NPC问题

NPC问题是指满足下面两个条件的问题:
（1）它是一个NP问题;
（2）所有的NP问题都可以用多项式时间约化到它。

所以显然NP完全问题具有如下性质：它可以在多项式时间内求解，当且仅当所有的其他的NP完全问题也可以在多项式时间内求解。这样一来,只要我们找到一个NPC问题的多项式解，所有的NP问题都可以多项式时间内约化成这个NPC问题，再用多项式时间解决，这样NP就等于P了。

目前，NPC问题还没有找到一个多项式时间算法，因此我们就此可直观地理解，NPC问题目前没有多项式时间复杂度的有效算法，只能用指数级甚至阶乘级复杂度的搜索。

大多数人的观点对于 P类问题，NP类问题，NPC之间的关系可用下图表示（此图正确性有待验证）：

```
**P：能在多项式时间内解决的问题**
```

NP: 不能在多项式时间内解决或不确定能不能在多项式时间内解决，但能在多项式时间验证的问题

NPC: NP完全问题，所有NP问题在多项式时间内都能约化(Reducibility)到它的NP问题，即解决了此NPC问题，所有NP问题也都得到解决。

NP hard:NP难问题，所有NP问题在多项式时间内都能约化(Reducibility)到它的问题(不一定是NP问题)。

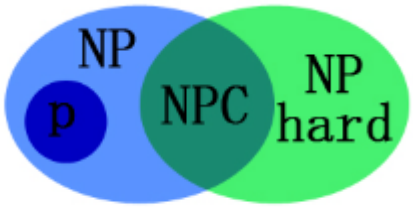
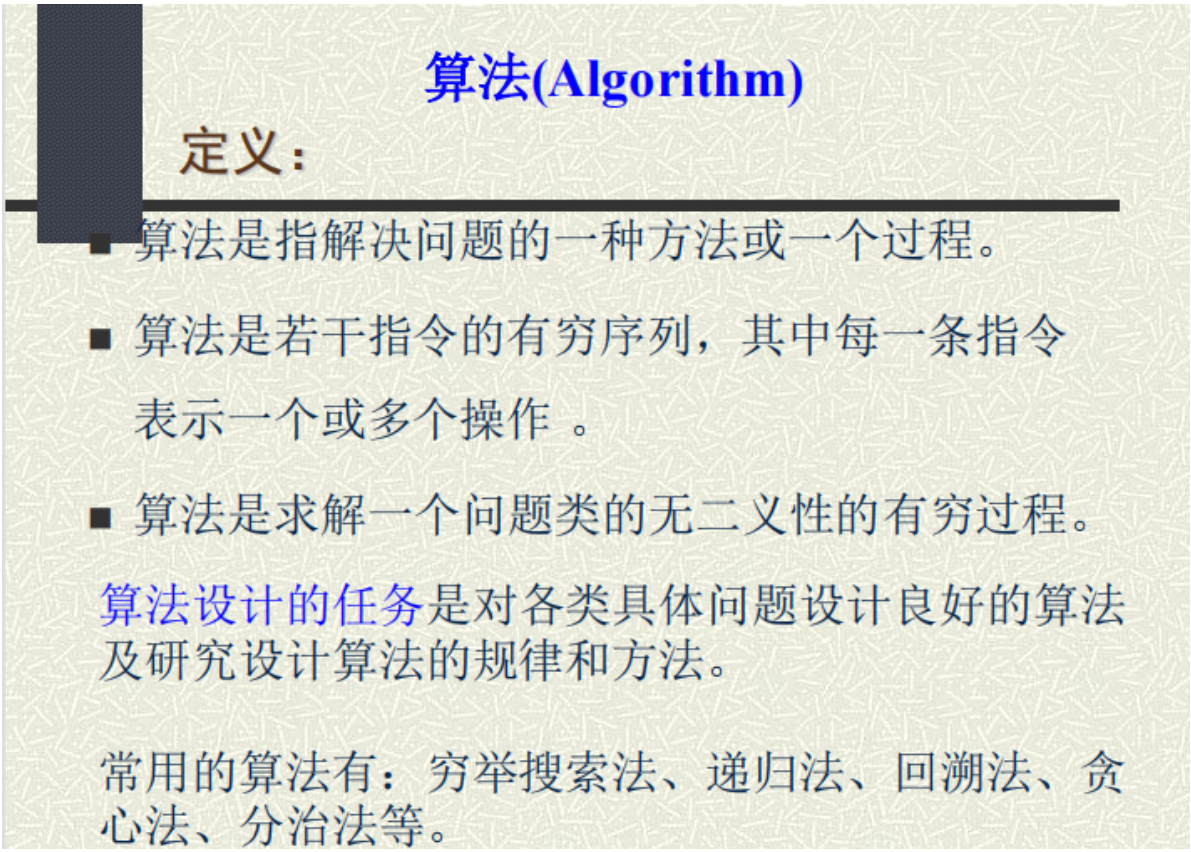


图1 P NP NPC NPhard关系的图形表示

算法的概念，什么是好的算法？算法的复杂度怎么计算？



算法(Algorithm)

算法的性质：

■(1)输入：有0个或多个外部提供的量作为算法的输入。

■(2)输出：算法产生至少一个量作为输出。

■(3)确定性：组成算法的每条指令是清晰，无歧义的。

■(4)有限性：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

■(5)可行性：算法中的所有运算都是基本的，原则上它们都能够精确地进行，而且进行有穷次即可完成。

通常设计一个“好”的算法应考虑达到以下目标：

① 正确性：4个层次

② 可读性：有助于对算法的理解（结构化、模块化、加注释等）

③ 健壮性：输入数据非法，作出适当反应或进行处理

④ 高效率与低存储要求：效率是指算法执行的时间

算法复杂性分析

- 算法复杂性是算法运行所需要的计算机资源的量
  - 需要时间资源的量称为时间复杂性
  - 需要空间资源的量称为空间复杂性
  - 这个量应该只依赖于算法要解的问题的规模、算法的输入和算法本身的函数。如果分别用N、I和A表示算法要解问题的规模、算法的输入和算法本身，而且用C表示复杂性，那么，应该有$C=F(N,I,A)$。
- 一般把时间复杂性和空间复杂性分开，并分别用T和S来表示，则有：$T=T(N,I)$和$S=S(N,I)$。（通常，让A隐含在复杂性函数名当中）

如何分析算法复杂性

1、分析算法的工作量—忽略（细节）次要工作量，留下主要工作量（主要运算），从而分析其趋势

2、基本运算—主要工作是哪方面

寻找规则：基本运算的次数和算法总运算次数成比例关系；基本运算以外的其他运算可以忽略

3、算法复杂度还和问题规模n有关，如顺序查找 T(n)=n

度量标准分为五个：

① **时间复杂度**：指算法运行效率高，即算法运行所消耗的时间短。

② **空间复杂度**：指算法所需的存储空间少。

③ **难易程度**：指算法遵循标识符命名规则，简洁，易懂，注释语句恰当、适量，方便自己和他人阅读，便于后期调试和修改。

④ **健壮性**：指算法对非法数据及操作有较好的反应和处理。

⑤ **正确性**:指算法能够满足具体问题的需求，程序运行正常，无语法错误。

算法设计与分析的步骤—分析算法的基本原则

⑤ 最优性

- 含义：
  - 指求解某类问题中效率最高的算法
- 两种最优性（设A是解某个问题的算法）
  a) 最坏情况：如果在解这个问题的算法类中没有其它算法在最坏情况下的时间复杂性比A在最坏情况下的时间复杂性低，则称A是解这个问题在最坏情况下的最优算法。
  b) 平均情况：如果在解这个问题的算法类中没有其它算法在平均情况下的时间复杂性比A在平均情况下的时间复杂性低，则称A是解这个问题在平均情况下的最优算法。

什么是分治法，说说你的研究中哪里用到了分治法?

算法总体思想

- 分治法的设计思想：
  - 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

深度学习的优化算法，说白了就是梯度下降。每次的参数更新有两种方式。

第一种，遍历全部数据集算一次损失函数，然后算函数对各个参数的梯度，更新梯度。这种方法每更新一次参数都要把数据集里的所有样本都看一遍，计算量开销大，计算速度慢，不支持在线学习，这称为Batch gradient descent，批梯度下降。

另一种，每看一个数据就算一下损失函数，然后求梯度更新参数，这个称为随机梯度下降，stochastic gradient descent。这个方法速度比较快，但是收敛性能不太好，可能在最优点附近晃来晃去，hit不到最优点。两次参数的更新也有可能互相抵消掉，造成目标函数震荡的比较剧烈。

为了克服两种方法的缺点，现在一般采用的是一种折中手段，mini-batch gradient decent，小批的梯度下降，这种方法把数据分为若干个批，按批来更新参数，这样，一个批中的一组数据共同决定了本次梯度的方向，下降起来就不容易跑偏，减少了随机性。另一方面因为批的样本数与整个数据集相比小了很多，计算量也不是很大。

2.1 递归的概念

- 定义：
  - 直接或间接地调用自身的算法称为**递归算法**。
  - 用函数自身给出定义的函数称为**递归函数**。
- 由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便：
  - 在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。
  - 这自然导致递归过程的产生。
- 分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

蛮力和回溯

9、 蛮力法和回溯法的定义

蛮力法：蛮力法是一种简单直接地解决问题的方法(暴力求解)，常常直接基于问题的描述和所涉及的概念定义。

回溯法：回溯法（探索与回溯法）是一种选优搜索法，又称为试探法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。

10、 蛮力法和回溯法的基本思想和求解步骤

蛮力法：蛮力法是指采用**遍历**（扫描）技术，即采用一定的策略将待求解问题的所有元素依次处理一次，从而找出问题的解。**依次处理**所有元素是蛮力法的关键，为了避免陷入重复试探，应保证处理过的元素不再被处理。蛮力法本质是先有策略地穷举，然后验证。简单来说，就是列举问题所有可能的解，然后去看看是否满足题目要求，是一种逆向解题方式。

回溯法：在回溯法中，每次扩大当前部分解时，都面临一个可选的状态集合，新的部分解就通过在该集合中选择构造而成。这样的状态集合，其结构是一棵**多叉树**，

每个树结点代表一个可能的部分解，它的儿子是在它的基础上生成的其他部分解。树根为初始状态，这样的状态集合称为**状态空间树**。回溯法对任一解的生成，一般都采用逐步扩大解的方式。每前进一步，都试图在当前部分解的基础上扩大该部分解。它在问题的状态空间树中，从开始结点（根结点）出发，以深度优先搜索整个状态空间。

求解方法：

回溯法将解空间看作树形结构(状态空间树)，进行深度优先遍历+跳跃式搜索

跳跃式：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，则跳过该节点为根的子树的搜索，逐层向其祖先节点回溯；否则进入该子树，继续深度优先策略搜索

回溯法与暴力查找是一样的吗？

可以把回溯法和分支界限法看成是穷举法的一个改进。该方法至少可以对某些组合难题的较大实例求解。

不同点：

每次只构造候选解的一部分，然后评估这个部分的候选解，如果加上剩余的分量也不可能求得一个解，就绝对不会生成剩下的分量。

11、 蛮力法和回溯法的经典求解问题

蛮力法：选择排序、冒泡排序、顺序查找

回溯法：n 皇后问题、0-1 背包问题

➢ 0-1背包问题

问题描述：给定n种物品和一个背包。物品i的重量是w(i)，其价值为v(i)，背包的容量为c。问应该如何选择装入背包的物品，使得装入背包中的物品的总价值最大？

注：每个物品只能使用一次。

➢ 八皇后问题

问题描述：在一个N*N的棋盘上放置N个皇后，使其不能互相攻击。(同一行、同一列、同一斜线上的皇后都会自动攻击)那么问，有多少种摆法？

分支界限法

第四章 分支界限

12、 分支界限的定义

所谓分支就是采用**广度优先**的策略，一次搜索活结点的所有分支。为了有效的选择下一扩展节点，以加速搜索的进程。在每一处活节点处，计算一个函数值（限界函数），在当前的活结点中选择一个可行最优解作为下一拓展节点。

分支界限法按广度优先策略遍历问题的解空间树，在遍历过程中，对已经处理的每一个结点根据限界函数估算目标函数的可能取值，从中选取使目标函数取得极值的结点优先进行广度优先搜索，从而不断调整搜索方向，尽快找到问题的解。

13、 分支界限的基本思想和求解步骤

分支限界法常以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树。

在分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中。

此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止。

分支界限法是一种搜索方法，用于求解最优化问题。其基础是回溯法，基本思路在于解节点的生成，扩展，剪枝。

14、 分支界限的经典求解问题

0-1 背包问题、旅行商问题、单元最短路径问题。

动态规划算法

矩阵连乘计算次序问题的最优解包含着其子问题的最优解。这种性质称为**最优子结构性质**。**问题的最优子结构性质是该问题可用动态规划算法求解的显著特征。**

证明0/1 背包问题具有最优子结构性质

即若背包容量为c时，$(y_1, y_2, \cdots y_n)$为待选物品为1到n时knap(1, n, c)的最优解，则$(y_2, \cdots y_n)$将是物品1作出选择后的子问题 knap(2, n, c- w1*y1) 的最优解

如果问题的一个最优解$x_1, x_2, \ldots, x_n$包含物品n，即$x_n = 1$，那么其余决策$x_1, x_2, \ldots, x_{n-1}$一定构成子问题1, 2, ..., n-1在容量为W-w_n时的最优解。我们可以利用“切割–粘贴”方法证明：如果存在子问题1, 2, ...,n-1在容量为W-w_n时的更大价值的解x_1', x_2', ..., x_{n-1}'，我们可以构造问题的一个新解$x_1', x_2', \ldots, x_{n-1}', x_n$。这个解比$x_1, x_2, \ldots, x_n$的总价值更大，这与$x_1, x_2, \ldots, x_n$是问题最优解相矛盾。如果这个最优解不包含物品$n$，即$x_n = 0$，那么这个最优解一定包含子问题1, 2, ..., n-1在容量为W时的最优解。证明过程类似上述情形。

```
class solution{
        public int knapsackProblem(int[] wt,int[] val,int size){
                //定义dp数组
                int[][] dp = new int[wt.length][size];
                //对于装入前0个物品而言，dp数组储存的总价值初始化为0
                for(int i = 0;i < size;i++){
                    int[0][i] = 0;
                }
                //对于背包容量W=0时，装入背包的总价值初始化为0
                for(int j = 0;j < size;j++){
                    int[j][0] = 0;
                }
                //外层循环遍历物品
                for(int i = 1;i <= N;i++){
                    //内层循环遍历1~W(背包容量)
                    for(int j = 0;j <= W;j++){
                        //外层循环i,如果第i个物品质量大于当前背包容量
                        if (wt[i] > W) {
                            dp[i][W] = dp[i-1][W];  //继承上一个结果
                        } else {
                            //在“上一个结果价值”和“把当前第i个物品装入背包里所得到价值”二者里选价值较大的
                            dp[i][W] = Math.max(dp[i-1][W],dp[i-1][W-wt[i]] + val[i])
                        }
                    }
                }
        }
```

- 顾名思义，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。

1. 贪心选择性质

- 所谓贪心选择性质是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。
- 这是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。

- 对于一个具体问题，要确定它是否具有贪心选择性质，必须证明每一步所作的贪心选择最终导致问题的整体最优解。

贪心算法的基本要素—最优子结构性质

2. 最优子结构性质

- 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。

- 问题的最优子结构性质是该问题可用动态规划算法或贪心算法求解的关键特征。

0-1背包和一般背包问题

- 贪心选择的策略有哪些呢？

1. 每次从剩余的物品中，选出可以装入背包的价值最大的物品。
2. 从剩下的物品中选择可装入背包的重量最小的物品。
3. 从剩余物品中选择可装入包的v_i / w_i值（性价比—效益值）最大的物品。

- 用贪心算法解背包问题的基本步骤：

① 首先，计算每种物品单位重量的价值V_i/W_i；

② 然后，依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包。

a) 若将这种物品全部装入背包后，背包内的物品总重量未超过C，则选择单位重量价值次高的物品并尽可能多地装入背包。

③ 依此策略一直地进行下去，直到背包装满为止。

第七章 随机算法与近似算法

23、 随机算法的定义

一个随机算法是一种算法，它采用了一定程度的随机性作为其逻辑的一部分。该算法通常使用均匀随机位作为辅助输入来指导自己的行为，超过随机位的所有可能的选择实现了"平均情况下的"良好业绩的希望。从形式上看，该算法的性能将会是一个随机变量，由随机位决定;因此无论是运行时间，或输出(或两者）是随机变量。

在常见的实践中，随机化算法是使用近似的伪随机数发生器代替随机比特的真实来源的;这样的实施可以从预期的理论行为偏离。

24、 近似算法的定义

近似算法是一种处理优化问题 NP 完全性的方式，它无法确保最优解。近似算法的目标是在多项式一时间内尽可能地接近最优值。

它虽然无法给出精确最优解，但可以将问题收敛到最终解的近似值。其目标满足以下三个关键特性:①能够在多项式时间内高效运行;②能够给出最优解;③对于每个问题实例均有效。

25、 启发式算法的定义

启发式算法是相对于最优化算法提出的。一个问题的最优算法求得该问题每个实例的最优解。启发式算法可以这样定义:一个基于直观或经验构造的算法，在可接受的花费(指计算时间和空间)下给出待解决组合优化问题每一个实例的一个可行解，该可行解与最优解的偏离程度一般不能被预计。

26、 随机算法、近似算法、启发式算法的应用场景和具体应用

随机算法：场景有单位年会抽奖、随机挑观众互动、学生上课随机挑选回答问题。
常用应用：传统的快排序算法、随机快排序算法、模式匹配

近似算法：目前大规模的 NPC 问题我们无法通过计算得到，因此我们需要通过损失一部分精度的做法来找到多项式的近似算法。常用应用：顶点覆盖问题的近似算法、旅行售货员问题近似算法、集合覆盖问题的近似算法、子集和问题的近似算法

启发式算法：一般用于解决 NPhard 问题。常用算法有：模拟退火算法(SA)、遗传算法（GA)、蚁群算法(ACO)、人工神经网络(ANN)

什么是旅行商问题？在回溯法，分支限界法，贪心算法，动态规划算法其中任选两种，求解旅行商问题,写出求解思路,伪代码

https://www.cnblogs.com/dddyyy/p/10084673.html

https://blog.csdn.net/sgsx11/article/details/121482958