一：贪心法概述

1）优化问题

- 优化问题的基本要素：**问题的解，约束条件，目标函数。**
- 很多优化问题均属于**NP-hard问题**，对其问题的求解大多只能求近似解，贪心法则是求近似解的主要途径。

注：

优化问题之所以被认为是NP难问题，是因为在一般情况下，找到一个问题的最优解需要遍历所有可能的解空间，这个解空间的规模通常是**指数**级别的。因此，对于大规模的优化问题，穷举搜索所有可能的解是不可行的。

虽然贪心算法是一种高效的求解优化问题的方法，但**并不保证能够得到全局最优解。**对于某些优化问题，可能需要使用其他算法或方法来求解，例如动态规划、回溯、分支界限等。这些算法通常具有更高的时间复杂度，但能够找到问题的最优解。因此，优化问题通常被认为是NP难问题。

2）贪心法是什么

- 贪心法是一种启发式算法，通过局部最优解得到全局最优解。求得最优解需要满足贪心选择性和最优子结构两个特征。
- **贪心选择性质（Greedy Choice Property）**是指在贪心算法中，每一步的选择都是当前看起来最优的选择。也就是说，在每一步中，做出的选择都是局部最优的，希望通过一系列局部最优选择达到全局最优解。

最优子结构性质（Optimal Substructure）是指问题的最优解包含了子问题的最优解。换句话说，问题的最优解可以通过一系列子问题的最优解逐步构建而得到。这种性质使得我们可以将原问题分解为若干个相互独立的子问题，并通过求解子问题的最优解来得到原问题的最优解。

- 贪心算法通过每一步选择当前看起来最优的解决方案，**逐步构建全局最优解**。它不需要对整个问题空间进行搜索，而是**基于局部最优**选择策略，通过**贪心选择性质**和**最优子结构**性质，得到最终的解。贪心算法的简单性和高效性使得它成为解决某些**优化问题**的有效方法。
- 优点：算法简单，时间和空间复杂性低。

3）贪心法算法思想

1. 贪心法适用于**组合优化问题**。
2. 求解过程是**多步判断过**程，最终的判断序列对应于问题的最优解。
3. 依据某种短视的**贪心选择性**质判断，性质好坏决定算法的成败。
4. 贪心法必须进行**正确性证明（精确解）**。
5. 证明贪心法不正确的技巧：**举反例**。

二：贪心算法的应用实例

1.货船装载问题

> 若有n个集装箱，集装箱大小一样，第i个集装箱大小为Wi，设船的载重量为c，设计一个装船的方法使得装入的集装箱数目做最多。

第一直觉：排个序，每次选择最小的。

这个也可以理解为一个**优化问题**。

问题的解为每个集装箱的选择和不选择。约束条件为总重量小于载重量。目标函数为装的集装箱数目最多。

所以尝试用**贪心法**求解。寻找贪心策略和是否满足最优子结构特征。

贪心策略为：轻者优先。且满足最优子结构。·

是最优解的一种， 不是唯一的最优解。

2.0/1背包问题

注：**NP-hard问题**，若有**多项式复杂度的算法产生的解可能是近似解。**

> 背包的容量为c。存在n个物品，每个物品的重量和价值分别为Wi和Pi（1<=i<=n），试给出一种装入物品的方法，使获得的总效益值最大。

第一眼：每个背包有装和不装两种状态，指数级的时间复杂度，为nph问题。如果使用贪心策略每次选择价值最大的或者重量最小的，都难以满足最优子结构的特点。

优化问题。问题的解，约束条件，目标函数都可以很容易看出来，不赘述了。

贪心策略：1）优先选择价值高；2）优先选择体积小；3）优先选择比值高。

这三种贪心策略都无法百分百得到最优解，不过放一下3）的伪代码。

```
for i<-1 to n do
    di=pi/wi #calculate density
    D<-D U {di}
D<-sort(D,P,W)
    #将物品按密度从大到小排序
for i<-1 to n do
    if C>=Wi xi=1 #put
        else xi=0
return X;
# 时间复杂度为O(nlgn)
```

这里贪心法不能保证一定能得到01背包的最优解，因此产生了一个贪心法和最优解误差的百分比的公式：（|优化值-贪心解值|）/优化值*100%。

接着产生了

0/1背包问题：k-优化算法

1.K优化算法是上述3）的算法策略的改进，将误差控制在1/（k+1）的范围内。

2.具体内容

对物品密度从大到小排序。

先将一些物品放入背包，然后其余物品使用贪心法

预先放入的物品不超过k

对所有预装物品数不超过k的剩余物品子集执行贪心过程，并从中找到有最大效益值的解作为k优化问题的解。

注：

- 也就是先选择满足大小小于k的子集，再采用密度贪心攻略选择剩下的。最后在先选择的所有子集对应的不同解法中，找到最优解。
- 需要测试的子集数目为排列组合问题，从n个中选择k个。

- 每个子集贪心法的时间为O(n)，k>0时，总的时间开销为O(n^(k+1))。

3.连续背包问题

这个与0/1背包的不同在于这里的物品能够拆开，也就是说如果按照上面的密度排序，是能够得到最优解的。

这是一个简单的思想，但没有学习这门课之前并不能意识到这是贪心法。贪心法确认的关键在于局部最优和全局最优是否等同。

4.拓扑排序问题

> 在一个有向无环图内，G=（V，E），找到一个顶点的线性序列，该序列满足以下两个条件：
>
> 1.每个顶点有且只出现一次，如果边包含（u，v），则结点u在拓扑序列中处于v的2前面。

这是数据结构与算法课程中的例题，不过多讲解，现在直接观察该题的解题思路：

贪心策略：从当前尚不在拓扑排序序列的顶点中选择一项顶点v，其所有的前驱节点u都在已产生的拓扑序列中，并将v加入到拓扑序列中。（简单来说，将入度为0的顶点加到拓扑排序的顶点中，然后在原图去掉）。

代码（自然语言）

```
/*
计算每个顶点的入度
从第一个开始遍历，将入读为0的首个结点入栈
while（栈不空）
{
任取一入度为0的顶点放入拓扑序列中；
将与其相邻的顶点的入读减1；
如有新的入度为0的结点出现，将其放入栈中；
}
如有剩余的顶点未被删除，说明该图有环路
*/
Kahn算法的时间复杂度为O(n+e);
```

5.分类二分覆盖问题

> 在二分图中寻找最小覆盖的问题，等价于集合覆盖问题，是NP难问题。
>
> 二分图是一个无向图，它的n个顶点分为两个不交叉集合A和B，且任一条边的两个顶点不在同一个集合。

> A的一个子集完全覆盖B，当且仅当B中每一个顶点至少与A的子集中的一个顶点相连。

如果使用贪心问题，贪心策略为每次寻找A中和B相连顶点最多的顶点，那么能够得到一个近似解，但是不一定得到最优解。

伪代码如下

```
for all i属于A,New[i]=degree[i];//A是A集合中的顶点
for all i属于B,covered[i]=false;
A1=空集;
while(for some i属于A,New[i]>0){
    选取v为A和A1中New[i]值最大的顶点
    A1=A1+{v};
    for 所有被v覆盖的B中的结点j{
        covered[j]=true;
        for 所有覆盖结点j中A的顶点k
            New[k]=New[k]-1
    }
}
if 有B中顶点没被覆盖 return false
    else 找到一个覆盖

#时间复杂度为：O(A^2+n^2)或者O(A^2+n+e)；
        n^2为邻接矩阵，n+e为邻接表。
```

6.最短路径问题

> 在一个加权有向图G=(V,E)中，它的每一条边都有一个非负的权重，每条路径的长度就是该条边上所有的权重之和。
>
> 从结点u到结点v之间的最短路径就是u到v中权重最小的路径。

直接搜索的复杂度为n的阶乘，复杂度较高，属于NPhard问题。

贪心策略：

dijkstra算法：确实初始节点为最优结点，然后更新最优结点到能到达结点的路径，然后选择初始节点到剩余结点的最优结点这样遍历。

也就是选择每次到达点路径最小且未标记的点作为一下个结点，这样子选的局部方法就能得到全局的最优解。

伪代码如下：

```
d[s]<-0;  #存放初始节点到每个结点的举例
for each v属于V - {s}
     do d[v]<-∞
S<-空集     #存放路径
Q<-V       #Q是维护V-S的优先队列
while Q不等于空集
    do u<- extrect-min{Q}
    S<-S U {u}
    for each v属于adj[u]
        do if d[v]>d[u]+w(u,v)
            then d[v]=d[u]+w(u,v)
# 时间复杂度为O(n^2);
```

7.最小生成树

> 给定无向连通带权图G=（V，E），E中每条边的权重w（u，v）。如果G的子图是一颗包含G的所有顶点的树，则称T为图G的生成树。生成树上各边权的总和称为该生成树的耗费，耗费最小的为最小生成树。

Prim算法：贪心策略：每次在已选边中选择和其它顶点连接最小权重的边的顶点

Prim伪代码

```
Q <- V
key[v] <- ∞ for all v属于V
key[s] <- 0 for some arbitrary s属于V
while Q不为空寂
    do u <- extract-min(Q)
        for each v属于adj[u]
            do if v属于Q and w(u,v)<key[v]
                then key[v]<- w(u,v)
                pai[v]<-u
// at last,{(v,pai[v])}forms the MST
```

Kruskal算法：贪心策略：每次在整个图中选择权重最小的边，不能构成环。

kruskal伪代码

```
//在一个具有n个顶点的网络中找到一颗最小生成树
令T为所选边的集合，初始化T为空
令E为网络中边的集合
while（E不为空&&T不等于n-1）{
    令(u,v)为E中代价最小的边
    E=E-{(u,v)}  //从E中删除边
    if(u,v)加入T中不会产生环路  将(u,v)加入T
}
if(T==n-1)T是最小耗费生成树
    else 网络不是互连的，不能找到生成树
```

8.哈夫曼编码问题

> 改变字符二进制的编码方式，进行缩短。

贪心选择性质：合并出现频率最低的两个字符。

算法以|C|个叶节点开始，执行|C|-1次合并运算后产生所需最终要求的树T。

基本流程：

- 初始：根据n个字符的频率{w1，w2，…，wn}构成n个根节点的集合F={T1,T2,…，Tn}，其中每颗二叉树Ti中只有一个带权为Wi的根节点，其左右子树均为空（叶子节点）。
- 在F中选取两颗根结点的权值最小的树作为左右子树构造一棵新的二叉树，且置新的二叉树的根节点的权值为其左右子树结点的根节点的权值之和。
- 在F中删除这两棵树，同时将新的二叉树加入F当中。
- 重复第二步和第三步，直到F中只含一棵树为止，称这棵树为最优二叉树。

附：

1.内容参考

佟鑫宇老师 天津大学智能与计算学部 2023秋 算法设计与分析 ppt

2.

考试考察重点：优化问题，贪心算法的设计要素，0/1背包问题的连续问题，最短路径问题，最小生成树，哈夫曼编码问题。

3.

全文同样包含个人的主观理解，如有错误，欢迎访问原文链接指正。