Question 1. 2. 工作分配问题。设有 n 件工作需要分配给 n 个人去完成。将工作 i 分配给第 j 个人完成所需要的费用为 c_{ij}。试设计一个算法，为每一个人分配一件不同的工作，并使总费用达到最小。

My Answer 1. *1.* 要为每一个人分配一件不同的工作，解空间为一个 n 的全排列集，于是在搜索时可想到用深度优先搜索，同时设置上界的方式来剪枝，可得算法 *(*见该答案后*)* 实现效果如下：

```
价格矩阵为:
[[1 3 3]
 [3 1 2]
 [1 3 2]]
最优解为 [0, 1, 0]
最优解值为 3
```

代码说明：直接运行 *6_2.py* 即可得到上述结果；若需要改变费用矩阵，可用 *ide* 打开 *6_2.py*，修改 *cost* 矩阵的值

算法 1 工作分配问题

输入: *cost* 数组

输出: 最小分配费用 *bestFee* 以及对应的解 X

```
1: function MINFEE(cost)
2:     k ← 0, X[k] ← 0
3:     N, bestFee 初始化
4:     while X[k] < N and computFee(X) > bestFee do//当超出上界时剪枝
5:         X[k]+ = 1
6:     end while
7:     if 到达了叶子节点 then
8:         if k == N − 1 then
9:             记录，并更新最优解
10:        else
11:            k+=1
12:            X[k]-=1
13:        end if
14:    else
15:        k-=1
16:    end if
17: end function
```

Question 2. 3. 最佳调度问题。假设有 n 个任务要由 k 个可并行工作的机器来完成。完成任务 i 需要的时间为 t_i。试设计一个算法找出完成这 n 个任务的最佳调度，使得完成全部任务的结束时间最早。

My Answer 2. *2.* 由于每个机器完成一个任务的时间是相同的，因此该问题可以抽象为一个类似 n 皇后的问题，其中约束条件为 *0*，限界函数为已知解的下界，依照回溯法思想，若大于下界，则实行剪枝，可得算法 *(见该答案后)*

由于即使用了剪枝，解空间依然很大，考虑先用局部贪心找到一个较优解，得到一个可以接受的上界，提高剪枝效率

但是实际测试下来，对于一个 3^14 问题 *(14* 个任务，*3* 个机器*)*，用局部贪心只比回溯法少了一千左右个叶子节点 *(回溯法有 24000* 左右个叶子节点*)*

```
费用矩阵为:
 [5, 12, 7, 18, 9, 6, 3, 15, 9, 15, 3, 2, 3, 5]
最优解为 [0, 0, 0, 1, 0, 1, 0, 2, 1, 2, 1, 0, 2, 2]
最优解值为 38
```

代码说明：直接运行 *6_2.py* 即可得到上述结果；若需要改变费用矩阵，可用 *ide* 打开 *6_2.py*，修改 *cost* 矩阵的值

Question 3.

5. 最小重量机器设计问题。设某一机器由 n 个部件组成，每一种部件都可以从 m 个不同的供应商处购得。设 w_{ij} 是从供应商 j 处购得的部件 i 的重量，c_{ij} 是相应的价格。试设计一个算法，给出总价格不超过 c 的最小重量机器设计。

My Answer 3. 该问题可看作多维的 *0/1* 背包问题，解空间为 n^m，其中 n 是零件数，m 是厂家数。

由于过于复杂，考虑用递归实现回溯法，于是可得到算法 *(见后)*

效果如下：

```
质量矩阵为:
 [[2 4 1]
 [1 3 5]
 [3 4 1]
 [2 4 2]
 [8 4 3]]
价格矩阵为:
 [[10 10 20]
 [30 50 30]
 [20 10 30]
 [20 30 40]
 [30 40 50]]
价格上界为:
 100
最优解为:
 [2 0 2 0]
```

代码说明：直接运行 *6_5.py* 即可得到上述结果；若需要改变费用、质量矩阵，可用 *ide* 打开 *6_5.py*，修改 *W* 和 *C* 矩阵的值

算法 2 最佳调度问题

输入: t 数组

输出: 最小分配时间 $bestTime$ 以及对应的解 X

```
1: function minTime(cost)
2:     k ← 0, X[k] ← −1
3:     K, N, bPre = boundPre(()) 初始化, 并用局部贪心找到一个较优解
4:     while X[k] < K and computTime(X) > bestTime do//当超出上界时剪枝
5:         X[k]+ = 1//转到下一个机器
6:     end while
7:     if 到达了叶子节点 then
8:         if k == N − 1 then
9:             记录，并更新最优解
10:        else
11:            k+=1
12:            X[k]-=1
13:        end if
14:    else
15:        k-=1
16:    end if
17: end function
18: function boundPre(())
19:    Time[1, ..., K] = 0
20:    for i=1:N do
21:        找到当前 Time 最小对应的索引 idx
22:        Time[idx]=t[i]
23:    end for
24: end function
```

算法 3 最小重量机器设计问题

输入: *depth* 第几个零件

输出: 最小重量 *BW* 以及对应的解 *X*

```
1: function MINWEIGHT(depth)
2:     if 既 then 满足限界函数又满足约束函数
3:         更新并记录
4:     end if
5:     for i do=1:M
6:         cw+=W[depth,i]
7:         cc+=C[depth,i]
8:         X[depth]=i
9:         if cw<BW and cc <=CB then
10:            minWeight(depth+1)
11:        end if
12:        cw-=W[depth,i]
13:        cc-=C[depth,i]
14:    end for
15: end function
```