贪心算法

- 贪心算法（又称贪婪算法）是指，在对问题求解时，总是做出在**当前看来是最好的选择。**也就是说，不从整体最优上加以考虑，他所做出的是在某种意义上的**局部最优解。**
- 贪心算法并不保证会得到最优解，但是在某些问题上贪心算法的解就是最优解。要会判断一个问题能否用贪心算法来计算。

找零问题

假设商店老板需要找零n元钱，钱币的面额有：100元、50元、20元、5元、1元，如何找零使得所需钱币的数量最少？

从最大面额开始找

```
t = [100, 50, 20, 5, 1]
def change(t, n):
    m = [0 for _ in range(len(t))]
    for i, money in enumerate(t):
        m[i] = n // money
        n = n % money
    return m, n
```

```
change(t, 376)
```

```
([3, 1, 1, 1, 1], 0)
```

背包问题

- 一个小偷在某个商店发现有n个商品，第i个商品价值v_i元，重w_i千克。他希望拿走的价值尽量高，但他的背包最多只能容纳W千克的东西。他应该拿走哪些商品？
  - 0-1背包：对于一个商品，小偷要么把它完整拿走，要么留下。不能只拿走一部分，或把一个商品拿走多次。（商品为金条）
  - 分数背包：对于一个商品，小偷可以拿走其中任意一部分。（商品为金砂）
- 举例：
  - 商品1：$v_1 = 60, w_1 = 10$
  - 商品2：$v_2 = 100, w_2 = 20$
  - 商品3：$v_3 = 120, w_3 = 30$
  - 背包容量：W=50
- 对于**0-1背包**和**分数背包**，贪心算法是否都能得到最优解？为什么？
  - 0-1背包不能用贪心算法，因为可能装不满

分数背包

```
goods = [(60,10), (120,30), (100,20)]
goods.sort(key=lambda x: x[0]/x[1], reverse=True) # 按照单位价值降序排列
```

```
def fractional_backpack(goods, w): # w是weight
    m = [0 for _ in range(len(goods))]
    total_v = 0
    for i,(price, weight) in enumerate(goods):
        if w >= weight:
            m[i] = 1
            w -= weight
            total_v += price
        else:
            m[i] = w / weight
            total_v += m[i]*price
            w = 0
            break
    return total_v, m
```

```
fractional_backpack(goods, 50)
```

```
(240.0, [1, 1, 0.6666666666666666])
```

拼接最大数字问题

- 有n个非负整数,将其按照字符串拼接的方式拼接为一个整数。如何拼接可以使得得到的整数最大?
    - 例：32,94,128,1286,6,71 可以拼接出的最大整数为94716321286128

```
from functools import cmp_to_key
```

```
li = [32, 94, 128, 1286, 6, 71]
```

```
def xy_cmp(x, y):
    if x+y < y+x:
        return 1 # x > y, 满足交换, y+x 放前面
    elif x+y > y+x:
        return -1
    else:
        return 0
```

```
def number_join(li):
    li = list(map(str, li))
    li.sort(key=cmp_to_key(xy_cmp))
    return "".join(li)
```

```
number_join(li)
```

```
'94716321286128'
```

活动选择问题

- 假设有n个活动，这些活动要占用同一片场地，而场地在某时刻只能供一个活动使用。
- 每个活动都有一个开始时间s_i和结束时间f_i（题目中时间以整数表示），表示活动在$[s_i, f_i)$区间占用场地。
- 问：安排哪些活动能够使改场地举办的活动的个数最多？

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- 贪心结论：最先结束的活动一定是最优解的一部分。
- 证明：假设a是所有活动中最先结束的活动，b是最优解重最先结束的活动。
  - 如果a=b，结论成立。
  - 如果$a \neq b$，则b的结束时间一定晚于a的结束时间，则此时a替换掉最优解中的b，a一定不与最优解中的其他活动时间重叠，因此替换后的解也是最优解。

```
activities = [(1,4),(3,5),(0,6),(5,7),(3,9),(5,9),(6,10),(8,11),(8,12),(2,14),
(12,16)]
activities.sort(key=lambda x: x[1]) # 按照结束时间排序
```

```
def activity_selection(a):
    res = [a[0]]
    for i in range(1, len(a)):
        if a[i][0] >= res[-1][1]: #当前活动的开始时间>=最后入选活动的结束时间
            # 不冲突
            res.append(a[i])
    return res
```

```
activity_selection(activities)
```

```
[(1, 4), (5, 7), (8, 11), (12, 16)]
```

贪心算法总结

1. 最优化问题
2. 不是所有最优化问题都能用贪心算法，如0-1背包，可以使用动态规划
3. 注意需要优化的重点