贪心（Greedy）

@M了个J

码拉松

https://github.com/CoderMJLee

http://cnblogs.com/mjios

贪心（Greedy）

- 贪心策略，也称为贪婪策略
  - 每一步都采取当前状态下最优的选择（局部最优解），从而希望推导出全局最优解

- 贪心的应用
  - 哈夫曼树
  - 最小生成树算法：Prim、Kruskal
  - 最短路径算法：Dijkstra

练习1 – 最优装载问题（加勒比海盗）

- 在北美洲东南部，有一片神秘的海域，是海盗最活跃的加勒比海
- 有一天，海盗们截获了一艘装满各种各样古董的货船，每一件古董都价值连城，一旦打碎就失去了它的价值
- 海盗船的载重量为 W，每件古董的重量为 w_i，海盗们该如何把尽可能多数量的古董装上海盗船?
- 比如 W 为 30，w_i 分别为 3、5、4、10、7、14、2、11

- 贪心策略：每一次都优先选择重量最小的古董

① 选择重量为 2 的古董，剩重量 28
② 选择重量为 3 的古董，剩重量 25
③ 选择重量为 4 的古董，剩重量 21
④ 选择重量为 5 的古董，剩重量 16
⑤ 选择重量为 7 的古董，剩重量 9

- 最多能装载 5 个古董

```java
int capacity = 30;
int[] weights = {3, 5, 4, 10, 7, 14, 2, 11};
int count = 0, weight = 0;
Arrays.sort(weights);

for (int i = 0; i < weights.length && weight < capacity; i++) {
    int newWeight = weight + weights[i];
    if (newWeight <= capacity) {
        weight = newWeight;
        count++;
    }
}
```

练习2 – 零钱兑换

■ 假设有 25 分、10 分、5 分、1 分的硬币，现要找给客户 41 分的零钱，如何办到硬币个数最少？

■ 贪心策略：每一次都优先选择面值最大的硬币

① 选择 25 分的硬币，剩 16 分

② 选择 10 分的硬币，剩 6 分

③ 选择 5 分的硬币，剩 1 分

④ 选择 1 分的硬币

◘ 最终的解是共 4 枚硬币

✓ 25 分、10 分、5 分、1 分硬币各一枚

```
Integer[] faces = {25, 10, 5, 1};
Arrays.sort(faces);
int coins = 0, money = 41;
int idx = faces.length - 1;
while (idx >= 0) {
    while (money >= faces[idx]) {
        money -= faces[idx];
        coins++;
    }
    idx--;
}
```

零钱兑换的另一个例子

■ 假设有 25 分、20 分、5 分、1 分的硬币，现要找给客户 41 分的零钱，如何办到硬币个数最少？

■ 贪心策略：每一步都优先选择面值最大的硬币

① 选择 25 分的硬币，剩 16 分

② 选择 5 分的硬币，剩 11 分

③ 选择 5 分的硬币，剩 6 分

④ 选择 5 分的硬币，剩 1 分

⑤ 选择 1 分的硬币

□ 最终的解是 1 枚 25 分、3 枚 5 分、1 枚 1 分的硬币，共 5 枚硬币

■ 实际上本题的最优解是：2 枚 20 分、1 枚 1 分的硬币，共 3 枚硬币

小码哥教育 SEEMYGO

注意

- 贪心策略并不一定能得到全局最优解
  - 因为一般没有测试所有可能的解，容易过早做决定，所以没法达到最佳解
  - 贪图眼前局部的利益最大化，看不到长远未来，走一步看一步

- 优点：简单、高效、不需要穷举所有可能，通常作为其他算法的辅助算法来使用

- 缺点：鼠目寸光，不从整体上考虑其他可能，每次采取局部最优解，不会再回溯，因此很少情况会得到最优解

练习3 – 0-1背包

- 有 n 件物品和一个最大承重为 W 的背包，每件物品的重量是 w_i、价值是 v_i
- 在保证总重量不超过 W 的前提下，将哪几件物品装入背包，可以使得背包的总价值最大?
- 注意：每个物品只有 1 件，也就是每个物品只能选择 0 件或者 1 件，因此称为 0-1背包问题

- 如果采取贪心策略，有3个方案

① 价值主导：优先选择价值最高的物品放进背包

② 重量主导：优先选择重量最轻的物品放进背包

③ 价值密度主导：优先选择价值密度最高的物品放进背包（价值密度 = 价值 ÷ 重量）

0-1背包 – 实例

■ 假设背包最大承重150，7个物品如表格所示

编号	1	2	3	4	5	6	7
重量	35	30	60	50	40	10	25
价值	10	40	30	50	35	40	30
价值密度	0.29	1.33	0.5	1.0	0.88	4.0	1.2

① 价值主导：放入背包的物品编号是 4、2、6、5，总重量 130，总价值 165

② 重量主导：放入背包的物品编号是 6、7、2、1、5，总重量 140，总价值 155

③ 价值密度主导：放入背包的物品编号是 6、2、7、4、1，总重量 150，总价值 170

小码哥教育 SEEMYGO

0-1背包 – 实现

```
void run(String titile, Comparator<Article> comparator) {
    Article[] articles = new Article[] {
        new Article(35, 10), new Article(30, 40),
        new Article(60, 30), new Article(50, 50),
        new Article(40, 35), new Article(10, 40),
        new Article(25, 30)
    };
    Arrays.sort(articles, comparator);
    int capacity = 150, weight = 0, value = 0;
    List<Article> selectedArticles = new ArrayList<>();
    for (int i = 0; i < articles.length && weight < capacity; i++) {
        int newWeight = articles[i].weight + weight;
        if (newWeight <= capacity) {
            selectedArticles.add(articles[i]);
            weight = newWeight;
            value += articles[i].value;
        }
    }
    System.out.println("------------" + titile + "------------");
    System.out.println("总价值: " + value);
    for (Article article : selectedArticles) {
        System.out.println(article);
    }
}
```

0-1背包 – 实现

```
run("重量主导", (Article a1, Article a2) -> {
    return a1.weight - a2.weight;
});

run("价值主导", (Article a1, Article a2) -> {
    return a2.value - a1.value;
});

run("价值密度主导", (Article a1, Article a2) -> {
    return Double.compare(a2.valueDensity, a1.valueDensity);
});
```

```
------------重量主导------------
总价值：155
[weight=10, value=40]
[weight=25, value=30]
[weight=30, value=40]
[weight=35, value=10]
[weight=40, value=35]
------------价值主导------------
总价值：165
[weight=50, value=50]
[weight=30, value=40]
[weight=10, value=40]
[weight=40, value=35]
------------价值密度主导------------
总价值：170
[weight=10, value=40]
[weight=30, value=40]
[weight=25, value=30]
[weight=50, value=50]
[weight=35, value=10]
```

小码哥教育 SEEMYGO

0-1背包 – 实现

```
public class Article {
    int weight;
    int value;
    double valueDensity;
    public Article(int weight, int value) {
        this.weight = weight;
        this.value = value;
        valueDensity = value * 1.0 / weight;
    }
    @Override
    public String toString() {
        return "[weight=" + weight + ", value=" + value + "]";
    }
}
```

小码哥教育 SEEMYGO

作业

- 分发饼干
  - https://leetcode-cn.com/problems/assign-cookies/

- 用最少数量的箭引爆气球
  - https://leetcode-cn.com/problems/minimum-number-of-arrows-to-burst-balloons/

- 买卖股票的最佳时机 II
  - https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/

- 种花问题
  - https://leetcode-cn.com/problems/can-place-flowers/

- 分发糖果
  - https://leetcode-cn.com/problems/candy/