贪心和分治

一、贪心算法

贪婪算法(贪心算法)是指在对问题进行求解时，在每一步选择中都采取最好或者最优(即最有利)的选择，从而希望能够导致结果是最好或者最优的算法。
贪婪算法所得到的结果往往不是最优的结果(有时候会是最优解)，但是都是相对近似(接近)最优解的结果。
贪婪算法并没有固定的算法解决框架，算法的关键是贪婪策略的选择，根据不同的问题选择不同的策略。
必须注意的是策略的选择**必须具备无后效性**，即某个状态的选择不会影响到之前的状态，只与当前状态有关，所以对采用的贪婪的策略一定要仔细分析其是否满足无后效性。

基本思路

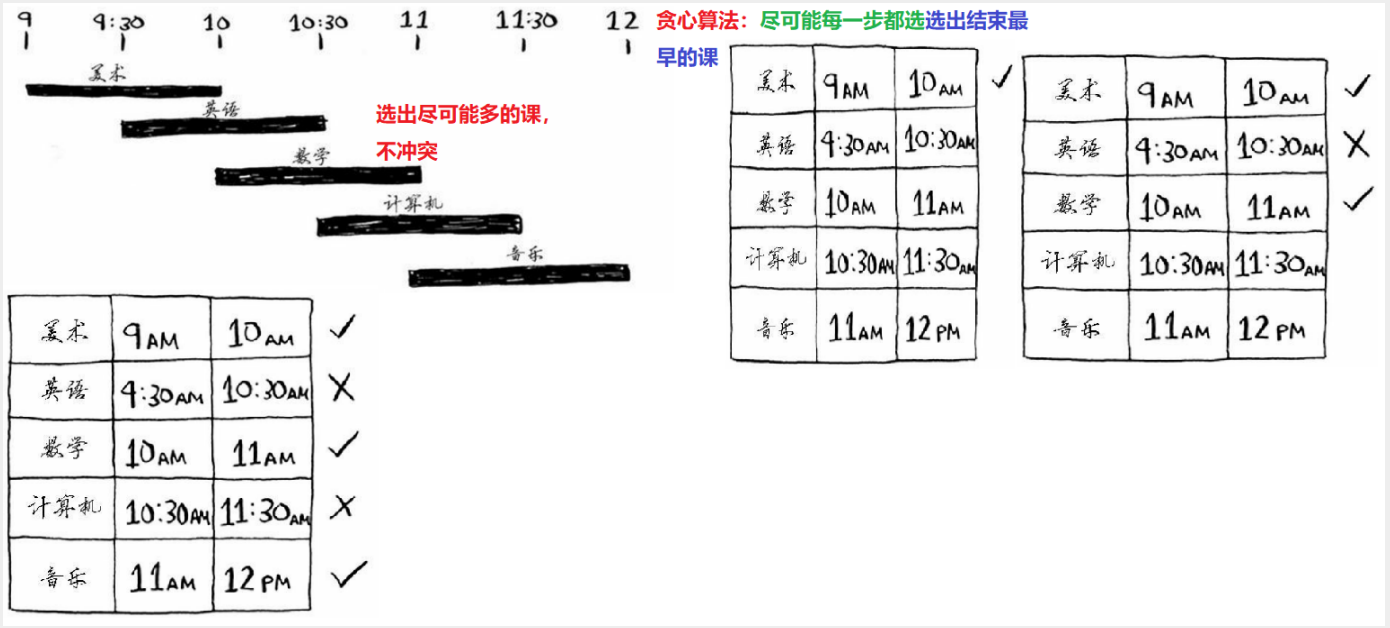
其基本的解题思路为：

1.建立数学模型来描述问题

2.把求解的问题分成若干个子问题

3.对每一子问题求解，得到子问题的局部最优解

4.把子问题对应的局部最优解合成原来整个问题的一个近似最优解



1.1 背包问题

```java
public class GreedyPackage {

    private int  MAX_WEIGHT = 120;
    private int[] weights = new int[]{35,30,60,50,40,10,25};
    private int[] values = new int[]{10,40,30,50,35,40,30};
```

```java
private void packageGreedy(int capacity,int weights[],int[] values){
    int n = weights.length;
    double[] r = new double[n];//性价比数组
    int [] index = new int[n];//按性价比排序物品的下标
    //1.计算性价比
    for(int i = 0;i<n;i++){
        r[i] = (double)values[i]/weights[i];
        index[i] = i;//默认排序
    }
```

贪心算法：当然是选最贵的先，所以先搞性价比

```java
    //2.性价比排序，并注意要交换下标，下标对应的是原来重量和价值的索引
    double temp = 0;//对性价比进行从大到小排序
    for(int i = 0;i<n-1;i++){
        for(int j = i+1;j<n;j++){
            if(r[i]<r[j]){
                temp = r[i];
                r[i] = r[j];
                r[j] = temp;
                int x = index[i];//性价比排序后，下标需要对应交换保存对应信息
                index[i] = index[j];
                index[j] = x;
            }
        }
    }
```

```java
    //3.根据上面排好序的性价比数组及其对应索引建立相对应的重量和价值数组
    //排序好的重量和价值分别存到数组
    int[] weightSorted = new int[n];
    int[] valueSorted = new int[n];
    for(int i = 0;i<n;i++){
        weightSorted[i] = weights[index[i]];
        valueSorted[i] = values[index[i]];
    }
```

```java
    //4.按照顺序装包
    int[] x = new int[n];
    int maxValue = 0;
    for(int i = 0;i<n;i++){
        if(weightSorted[i]<capacity){
            //还可以装得下
            x[i] = 1;//表示该物品被装了
            maxValue+=valueSorted[i];
            System.out.println("物品"+weightSorted[i]+", 价值为"+valueSorted[i]+"被放进包包");
            capacity = capacity - weightSorted[i];
        }
    }
    System.out.println("总共放下的物品数量："+Arrays.toString(x));
    System.out.println("最大价值："+maxValue);
```

二、分治算法

分治法的设计思想是：

分–将问题分解为规模更小的子问题；

治–将这些规模更小的子问题逐个击破；
合–将已解决的子问题合并，最终得出“母”问题的解；
一个先自顶向下，再自底向上的过程。

2.1分治经典例题

2.1.1 二分搜索技术

2.1.2 大整数的乘法

2.1.3 Strassen矩阵乘法

2.1.4 棋盘覆盖

2.1.5 合并排序

2.1.6 线性时间选择

2.1.7 最接近点对问题

2.1.8 快速排序

2.1.9 循环赛日程表