算法

- 分类
  - 算法可大致分为基本算法、数据结构的算法、数论与代数算法、计算几何的算法、图论的算法、动态规划以及数值分析、加密算法、排序算法、检索算法、随机化算法、并行算法，厄米变形模型，随机森林算法。
  - 算法可以宏泛的分为三类：
  - 一，有限的，确定性算法 这类算法在有限的一段时间内终止。他们可能要花很长时间来执行指定的任务，但仍将在一定的时间内终止。这类算法得出的结果常取决于输入值。
  - 二，有限的，非确定算法 这类算法在有限的时间内终止。然而，对于一个（或一些）给定的数值，算法的结果并不是唯一的或确定的。
  - 三，无限的算法 是那些由于没有定义终止定义条件，或定义的条件无法由输入的数据满足而不终止运行的算法。通常，无限算法的产生是由于未能确定的定义终止条件。
- 递推法
  - 递推是序列计算机中的一种常用算法。它是按照一定的规律来计算序列中的每个项，通常是通过计算机前面的一些项来得出序列中的指定项的值。其思想是把一个复杂的庞大的计算过程转化为简单过程的多次重复，该算法利用了计算机速度快和不知疲倦的机器特点。
- 递归法
  - 程序调用自身的编程技巧称为递归（recursion）。一个过程或函数在其定义或说明中有直接或间接调用自身的一种方法，它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。递归的能力在于用有限的语句来定义对象的无限集合。一般来说，递归需要有边界条件、递归前进段和递归返回段。当边界条件不满足时，递归前进；当边界条件满足时，递归返回。
  - 注意：
  - (1) 递归就是在过程或函数里调用自身;
  - (2) 在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口。
- 穷举法
  - 穷举法，或称为暴力破解法，其基本思路是：对于要解决的问题，列举出它的所有可能的情况，逐个判断有哪些是符合问题所要求的条件，从而得到问题的解。它也常用于对于密码的破译，即将密码进行逐个推算直到找出真正的密码为止。例如一个已知是四位并且全部由数字组成的密码，其可能共有10000种组合，因此最多尝试10000次就能找到正确的密码。理论上利用这种方法可以破解任何一种密码，问题只在于如何缩短试误时间。因此有些人运用计算机来增加效率，有些人辅以字典来缩小密码组合的范围。
- 贪心算法
  - 贪心算法是一种对某些求最优解问题的更简单、更迅速的设计技术。
  - 用贪心法设计算法的特点是一步一步地进行，常以当前情况为基础根据某个优化测度作最优选择，而不考虑各种可能的整体情况，它省去了为找最优解要穷尽所有可能而必须耗费的大量时间，它采用自顶向下,以迭代的方法做出相继的贪心选择,每做一次贪心选择就将所求问题简化为一个规模更小的子问题, 通过每一步贪心选择,可得到问题的

一个最优解，虽然每一步上都要保证能获得局部最优解，但由此产生的全局解有时不一定是最优的，所以贪婪法不要回溯。

- 贪婪算法是一种改进了的分级处理方法，其核心是根据题意选取一种量度标准，然后将这多个输入排成这种量度标准所要求的顺序，按这种顺序一次输入一个量，如果这个输入和当前已构成在这种量度意义下的部分最佳解加在一起不能产生一个可行解，则不把此输入加到这部分解中。这种能够得到某种量度意义下最优解的分级处理方法称为贪婪算法。
- 对于一个给定的问题，往往可能有好几种量度标准。初看起来，这些量度标准似乎都是可取的，但实际上，用其中的大多数量度标准作贪婪处理所得到该量度意义下的最优解并不是问题的最优解，而是次优解。因此，选择能产生问题最优解的最优量度标准是使用贪婪算法的核心。
- 一般情况下，要选出最优量度标准并不是一件容易的事，但对某问题能选择出最优量度标准后，用贪婪算法求解则特别有效。

- 分治法
  - 分治法是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。
  - 分治法所能解决的问题一般具有以下几个特征：
  - (1) 该问题的规模缩小到一定的程度就可以容易地解决；
  - (2) 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质；
  - (3) 利用该问题分解出的子问题的解可以合并为该问题的解；
  - (4) 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子问题。
- 动态规划法
  - 动态规划是一种在数学和计算机科学中使用的，用于求解包含重叠子问题的最优化问题的方法。其基本思想是，将原问题分解为相似的子问题，在求解的过程中通过子问题的解求出原问题的解。动态规划的思想是多种算法的基础，被广泛应用于计算机科学和工程领域。
  - 动态规划程序设计是对解最优化问题的一种途径、一种方法，而不是一种特殊算法。不象前面所述的那些搜索或数值计算那样，具有一个标准的数学表达式和明确清晰的解题方法。动态规划程序设计往往是针对一种最优化问题，由于各种问题的性质不同，确定最优解的条件也互不相同，因而动态规划的设计方法对不同的问题，有各具特色的解题方法，而不存在一种万能的动态规划算法，可以解决各类最优化问题。因此读者在学习时，除了要对基本概念和方法正确理解外，必须具体问题具体分析处理，以丰富的想象力去建立模型，用创造性的技巧去求解。
  - 背包问题
    - void FindMax()//动态规划
    - {
    - int i,j;
    - //填表
    - for(i=1;i<=number;i++)
    - {
      - for(j=1;j<=capacity;j++)
      - {
        - if(j<w[i])//包装不进
        - {

- V[i][j]=V[i-1][j];
- }
- else//能装
- {
- if(V[i-1][j]>V[i-1][j-w[i]]+v[i])//不装价值大
- {
- V[i][j]=V[i-1][j];
- }
- else//前i-1个物品的最优解与第i个物品的价值之和更大
- {
- V[i][j]=V[i-1][j-w[i]]+v[i];
- }
- }
- }
- }
- }
-
- 找零钱
- #include <iostream>
- using namespace std;
- int main() {
- //硬币面值的数组
- int values[100];
- //要找零的数
- int money,size;
- //保存每个面值对应的最小值 因为0号位置要舍弃 因此要加1
- //输入总共有几种面值 和从小到大输入每种面值的大小
- cout<<"输入要找零的金额";
- cin>>money;
- cout<<"输入要面值的总数和种类";
- cin>>size;
- for(int i = 0; i < size; i++)
- cin >> values[i];
- int *coinsUsed = new int[money + 1];
- coinsUsed[0] = 0;
- //总共的钱数的遍历
- int mincoin = 0;
- for(int i = 1; i <= money; i++){
- mincoin = i;
- //总共有几种面值的遍历
- for (int j = 0; j < size; j++) {
- //如果当前第j中面值小于总共的钱数

```
if(values[j] <= i){
//用了这一面值 前提是用这张纸币的数量比不用这张纸币的数量小采用
mincoin = min(mincoin,coinsUsed[i - values[j]] + 1);
}
}
//当前金额所需的最小纸张数
coinsUsed[i] = mincoin;
}
cout<<"总共需要面币张数为"<<coinsUsed[money]<<endl;
return 0;
}
```

- 迭代法
  - 迭代法也称辗转法，是一种不断用变量的旧值递推新值的过程，跟迭代法相对应的是直接法（或者称为一次解法），即一次性解决问题。迭代法又分为精确迭代和近似迭代。“二分法”和“牛顿迭代法”属于近似迭代法。迭代算法是用计算机解决问题的一种基本方法。它利用计算机运算速度快、适合做重复性操作的特点，让计算机对一组指令（或一定步骤）进行重复执行，在每次执行这组指令（或这些步骤）时，都从变量的原值推出它的一个新值。
- 分支界限法
  - 分枝界限法是一个用途十分广泛的算法，运用这种算法的技巧性很强，不同类型的问题解法也各不相同。
  - 分支定界法的基本思想是对有约束条件的最优化问题的所有可行解（数目有限）空间进行搜索。该算法在具体执行时，把全部可行的解空间不断分割为越来越小的子集（称为分支），并为每个子集内的解的值计算一个下界或上界（称为定界）。在每次分支后，对凡是界限超出已知可行解值那些子集不再做进一步分支，这样，解的许多子集（即搜索树上的许多结点）就可以不予考虑了，从而缩小了搜索范围。这一过程一直进行到找出可行解为止，该可行解的值不大于任何子集的界限。因此这种算法一般可以求得最优解。
  - 与贪心算法一样，这种方法也是用来为组合优化问题设计求解算法的，所不同的是它在问题的整个可能解空间搜索，所设计出来的算法虽其时间复杂度比贪婪算法高，但它的优点是与穷举法类似，都能保证求出问题的最佳解，而且这种方法不是盲目的穷举搜索，而是在搜索过程中通过限界，可以中途停止对某些不可能得到最优解的子空间进一步搜索（类似于人工智能中的剪枝），故它比穷举法效率更高。
- 回溯法
  - 回溯法（探索与回溯法）是一种选优搜索法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。
  - 其基本思想是，在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根结点出发深度探索解空间树。当探索到某一结点时，要先判断该结点是否包含问题的解，如果包含，就从该结点出发继续探索下去，如果该结点不包含问题的解，则逐层向其祖先结点回溯。（其实回溯法就是对隐式图的深度优先搜索算法）。 若用回溯法求问题的所有解时，要回溯到根，且根结点的所有可行的子树都要已被搜索遍才结束。 而若使用回溯法求任一个解时，只要搜索到问题的一个解就可以结束。

幕布 - 思维概要整理工具