参数搜索的应用

摘要

参数搜索法是解最优解问题中的常见的方法，它的应用十分广泛。本文通过几个例子说明了其在实际问题中的应用，并分析了它的优缺点。

关键字

参数搜索　上界　二分

正文：

1.　引言

参数搜索是解决最优解问题一种很常见的方法。其本质就是对问题加入参数，先解决有参数的问题，再不断调整参数，最终求得最优解，下面就例举出它在几个不同方面的应用。

二．应用

先来看一个例子，分石子问题：*有N个石子,每个石子重量Q_i；按顺序将它们装进K个筐中；求一种方案，使得最重的筐最轻。*

分析：本题乍一看很容易想到动态规划。事实上的确可以用动态规划解决，稍加分析我们很快得到一个简单的算法。用状态 f(i,k)表示将前 i 个石子装入 k 个筐最优方案，g(i,j)表示 i-j 中最重的石子，则可以写出状态转移方

程：

$$g(i,j)=\max\{g(i,j-1),Q_j\}$$
$$f(i,j)=\min\max\{f(k,j-1),g(k+1,i)\} \mid 1<=j<=k$$

边界条件：$g(i,i)=Q_i, f(1,1)=g(1,1)$

很明显，这个算法的时间复杂度为 $O(N^3)$,空间复杂度为 $O(N^2)$，并不十分理想。经过一些优化可以将复杂度降为 $O(N^2)$，不过这样思维复杂度骤然加大，且算法本身仍不够高效。

现在已经很难在原动态规划模型上做文章了，我们必须换一个思路。按一般的想法，顺序将石子装入筐，即先把石子放入第一筐，放到一定时候再改放第二个筐，第三个筐……但由于筐的重量没有上限，我们无法知道放到什么时候可以停止，转而将石子放入下一个筐。此时，问题的难点已经显露出来，是不是有方法可以化解呢？

我们不妨针对上面的难点，加入一个参数 P，改求一个判定可行解问题：

每个筐石子重量和不能超过 P，是否可以用 K 个筐装下所有石子。

首先经过分析不难发现，如果当前筐的石子总重量为 T_1，即将处理的石子重量为 T_2，若 $T_1+T_2 \leqslant P$，则我们仍将该石子放入当前框，这是显而易见的。由此可以得出贪心算法，按顺序把石子放进筐，若将石子放入当前筐后，筐的总重量不超过 P，则继续处理下一个石子；若重量和超过 P，则将该石子放入下一个筐，若此时筐的数目超过 K，则问题无解，否则处理完所有石子后就找到了一个可行解。

以上算法时间复杂度 O(N),空间复杂度 O(N)，这都是理论的下界。

现在我们已经解决了可行解问题，再回到原问题。是不是可以利用刚才的简化过的问题呢？答案是肯定的。一个最简单的想法是从小到大枚举 P，不断尝试找最优解为 P 的方案（这就是刚才的可行解问题），直到找到此方案。这就是题目的最优解。

估算一下上面算法的时间复杂度。令 $T=\sum Q_i$，则最坏情况下需要枚举 T 次才能找到解，而每次判断的时间复杂度为 O(N)，因此总的时间复杂度为 O(TN)，故需要做进一步优化。

下面考虑答案所在的区间。很明显，若我们可以找到一个总重量不超过 P’的解，则我们一定能找到一个总重量不超过 P’+1 的解，也就是说，可行答案必定可以位于区间[q,+∞]（其中 q 为本题最优解）。因此，我们自然而然的联想到了二分，具体方法为：在区间[1,T]内取中值 m，若可以找到不超过 m 的方案，则尝试区间[1,m-1];若不能，尝试区间[m+1,T]。不断重复以上步骤即可找到问题的最优解。

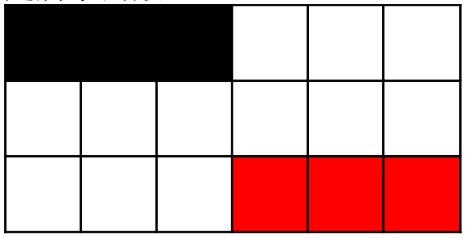
分析一下采用二分法后算法总的时间复杂度：由于每次除去一半的区间，

则一共只需判断 $\log_2 N$ 次，而每次判断的时间复杂度为 O(N)，因此这个算法总的时间复杂度为 O(NlgT)。

一般而言，这个复杂度可以令人满意的，并且实际使用中效果非常好。但该复杂度同权值有关，不完全属于多项式算法，我们可以继续求得一个多项式算法（该算法与本文核心内容无关，只作简单探讨）。

首先，此问题的答案必定为某一段连续石子的和，而段数不超过 n^2，因此，我们最多只要判断 $\log_2 N^2=2\log_2 N$ 次即可。现在新的问题是如何找到第 m 大的段。首先，以每个石子开头的所有段，例如：3 2 1 2，以 2 开头的所有段：2；2 1； 2 1 2， 由于每个石子的重量都大于 0，因此总和一定是递增的。

现在有 n 个递增段，如何快速的找到第 k 大的数呢？设各段长度为 $L_1,L_2,\ldots,L_N$,中点分别为 $M_1,M_2,\ldots,M_N$，我们每次询问各段中点的大小，若 $L_1/2+L_2/2+..+L_N/2>k$，则找到权值最大的中点，删去其后的数（下图中红色部分），否则找到权值最小的中点，删去其前面的部分（下图中黑色部分），可以证明删去的一定不是所求的数。



注：上图中每个各格子代表一个数。

由以上可知每次可以去掉某一段的 1/2，因为有 n 段，故总共需要去 $O(N\log_2 N)$次，每次找最小最大值可以用堆实现，复杂度仅为 O(lgN)，因此总的时间复杂度为 $O(\lg^2 N)$，而由上文我们知道找中值的操作总共有 $\log_2 N$ 次，故这个算法的时间复杂度为 $O(N\lg^3 N)$。

至此我们终于得到了一个多项式级别的算法，当然这个算法实现起来比较麻烦，实际效果也不甚理想，不过具有一定的理论价值，在此不做过多讨论。

回顾本题解题过程，首先我们得到了一个动态规划算法，由于很难再提

高其效率，因此我们另辟蹊径，寻求新的算法。在分析过程中我们发现由于筐的重量没有上限，因此很难确定将多少石子放入同一个筐内。为了解决此难点，我们加入了参数 P，改求可行解问题。参数的加入实际上就是强行给筐定了一个上界。正是由于这无形之中多出的条件，使得问题立刻简单化，于是我们很自然的得到了贪心算法。而最终使用二分法降低了算法的时间复杂度，成功地解决了问题。

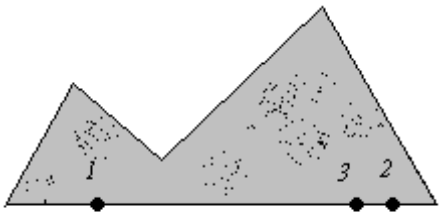
由上面的例子我们得到了此类解法的一般步骤，通常分为两步：

Ⅰ.首先引入参数 P，解决子问题：能否找到一个不优于 P 的方案

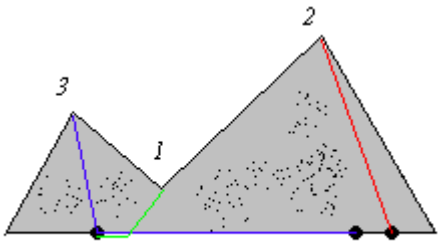
Ⅱ.从小到大枚举 P，判断是否有优于 P 的方案，直到找出原问题的最优解

一般地，参数搜索可以通过二分法或迭代降低时间复杂度，由于迭代法证明比较复杂，所以本文更多的讨论二分法。

这个方法的应用比较广泛，通常情况下和上例一样求最大（小）值尽量小（大）的题目都可以采用此方法，比如下面的例子。神秘的山：*有 n 个队员攀登 n 个山峰，每个队员需要选择一个山峰，队员们攀登的山峰各不相同，要求最后一个登顶的队员用的时间尽量短。*



Mountain T and 3 people



The solution to the example

分析：本问题分为两个部分解决：

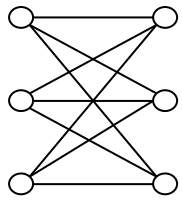
1. 求出每个队员攀登到各个山峰所需的时间；

2．找一个最优分配方案。

第一部分属于几何问题，我们可以枚举每个队员攀登山峰的位置再做简单的判断（题目规定攀登点必须为整点，这就为枚举提供了条件），这样就可求得队员们攀登上各个山峰所需的时间。
下面重点讨论第二部分。首先将我们的数据构图，很明显，我们得到的是一个二部图，假设有 3 个队员 3 个山峰，每个队员攀登的时间如下

	山峰 1	山峰 2	山峰 3
队员 1	1	3	2
队员 2	2	2	2
队员 3	1	3	3

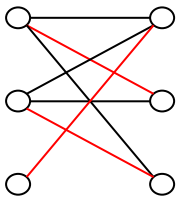
则我们可以构图，每条边附上相应的权值：



现在的任务就是在图中找一个匹配，匹配中权值最大的边尽量小。这个问题是否可以直接套用一些常见的模型呢？比如最大匹配或是最佳匹配？经过分析不难发现在这个问题中它们都是不足以胜任的，因此我们不得不做新的尝试。

上文提到过要求极大（小）值尽量小（大）的题目往往先定出上界比较容易下手，那么这题也采用类似的思路。引入一个参数 T，先求一个判定可行解的子问题：找一个方案，要求最后登山的队员所用时间不超过 T。

改变后的题目有什么不同之处呢？如果队员 i 攀登上山峰 j 所需的时间超过 T，则可以认为他在规定时间内不能攀登上山峰 j，我们就可以把这条边从图中删去。例如在上例中找一个不超过 2 的方案，经过一次“筛选”，保留的图如下。



经过这次过滤保留下来的边都是“合法边”，我们只需要在这个新的二部图中找一个完备匹配即可，一个完备匹配唯一对应着一个可行方案。而找完备匹配可以借用最大匹配的算法，因为如果一个二部图的最大匹配数等于N，则找到了一个完备匹配，否则该图中将不存在完备匹配。（上图中的红色表示一个完备匹配），那么这一步的时间复杂度为 O(NM)，而在本题中 $M=N^2$，因此我们可以在 $O(N^3)$的时间内判断出是否存在一个方案满足最后等上山顶的队员时间不超过 T。

最后，再用二分法枚举参数 T，找到最优解。由于答案必定为某个队员攀登上其中一个山峰的时间，因此我们开始的时候可以将所有数据排序，这样最终的时间复杂度为 $O(N^3 lgN)$。

引入参数后求可行解的子问题与原问题最大的区别在于定下上界以后，我们很自然的排除了一些不可取条件，从而留下了“合法”条件，使得问题变的简单明了。

.

上面的例子在增加了上界之后，排除了一些无效条件，其实它的作用绝不仅局限于此，有些时候，它能将不确定条件变为确定条件，比如下面的例子，最大比率问题：*有N道题目,每道题目有不同的分值和难度,分别为Ai,Bi；要求从某一题开始,至少连续选K道题目,满足分值和与难度和的比最大。*

分析：最朴素的想法是枚举下标 i’,j’，得到每一个长度不小于 K 的连续段，通过累加 Ai’->Aj’，Bi’->Bj’求出比值，并找出比最大的一段。这样做的时间复杂度很高，由于总共有 N^2段，每次计算比值的时间是 O(N)，则总的时间复杂度到达 $O(N^3)$，不过计算比值时，可以采用部分和优化，这样能把

复杂度降至 $O(N^2)$，但仍然不够理想。

我们需要追求更出色的算法，先考虑一个简单的问题——不考虑难度（B_i），仅仅要求分值和最大（当然此时分值有正有负）。

这个简化后的问题可以直接扫描，开始时为一个长度为 K 的段，设为 $Q_1,Q_2,\ldots Q_k$，每次添加一个新数，若 $Q_1+Q_2+..+Q_{L-K}$ 小于 0，则把它们从数列中删去，不断更新最优解即可。

这样我们就能在 O(N)的时间内找到长度不小于 k 且和最大的连续段。

之所以能成功解决简化后的问题，是由于该问题中每个量对最终结果的影响是确定的，若其小于 0，则对结果产生不好的影响，反之则是有益的影响。那么原问题每个参数对最终结果的影响是不是确定的呢？很遗憾，并不是这样，因为每个题目有两个不同的参数，他们之间存在着某些的联系，而这些联系又具有不确定性，故我们很难知道它们对最终结果是否有帮助。想解决原问题，必须设法消除这个不确定因素！那么有没有办法将这些不确定的因素转化成确定的因素呢？此时，引入参数势在必行！那么我们引入参数 P，求一个新的问题：找一个比值不小于 P 的方案。

这个问题实际就是求两个下标 i',j'，满足下面两个不等式

$j'-i'+1\geq k$ ①

$(Ai'+Ai'+1....+Aj')/(Bi'+Bi'+1\ldots+Bj') \geq p$ ②

由不等式②=>$Ai'+Ai'+1....+Aj' \geq p(Bi'+Bi'+1\ldots+Bj')$

整理得$(Ai'-pBi')+(Ai'+1-pBi'+1)\ldots+(Aj'-pBj') \geq 0$

可以令 $Ci=Ai-pBi$，则根据上面不等式可知问题实际求一个长度不小于 k 且和大于 0 的连续段。由之前的分析可以知道我们能在 O(N)的时间内求出长度不小于 k 且和最大的连续段，那么如果该段的和大于等于 0，则我们找到了一个可行解，如果和小于 0，则问题无解。

也就是说，我们已经能在 O(N)的时间内判断出是否存在比值不小于 P 的方案，那么接下来的步骤也就顺理成章了。我们需要通过二分法调整参数 P，不断逼近最优解。

计算一下以上算法的时间复杂度，设答案为 T，则该算法的时间复杂度

为 O(NlgT)，虽然这并不是多项式级别的算法，但在实际使用中的效果非常好。

引入参数后，由于增加了一个条件，我们就可以将不确定的量变为确定的量，从而解决了问题。

3. 总结

本文主要通过几个例子说明了参数搜索法在信息学中的应用，从上文的例子可以看出加入参数一方面能大大降低思维复杂度，另一方面也能得到效率相当高的算法，这使得我们解最解问题又多了一中有力的武器。当然，任何事物都是具有两面性的。参数搜索在具有多种优点的同时也有着消极的一面。由于需要不断调整参数逼近最优解，其时间复杂度往往略高于最优算法，且通常依赖于某个权值的大小，使得我们得到的有时不是严格意义上的多项式算法。

文章中还总结了使用此方法解题的一般步骤，在实际应用中，我们不能拘泥于形式化的东西，必须灵活应用，大胆创新，这样才能游刃有余的解决问题。

附录

文中例题的原题

问题一

Copying Books

Before the invention of book-printing, it was very hard to make a copy of a book. All the contents had to be re-written by hand by so called *scribers*. The scriber had been given a book and after several months he finished its copy. One of the most famous scribers lived in the 15th century and his name was Xaverius Endricus Remius Ontius Xendrianus (*Xerox*). Anyway, the work was very annoying and boring. And the only way to speed it up was to hire more scribers.

Once upon a time, there was a theater ensemble that wanted to play famous Antique Tragedies. The scripts of these plays were divided into many books and actors needed more copies of them, of course. So they hired many scribers to make copies of these books. Imagine you have *m* books (numbered *1, 2 ... m*) that may have different

number of pages (p_1, p_2 ... p_m) and you want to make one copy of each of them. Your task is to divide these books among k scribes, $k <= m$. Each book can be assigned to a single scriber only, and every scriber must get a continuous sequence of books. That means, there exists an increasing succession of numbers $0 = b_0 < b_1 < b_2, \ldots < b_{k-1} <= b_k = m$ such that i-th scriber gets a sequence of books with numbers between $b_{i-1}+1$ and b_i. The time needed to make a copy of all the books is determined by the scriber who was assigned the most work. Therefore, our goal is to minimize the maximum number of pages assigned to a single scriber. Your task is to find the optimal assignment.

Input Specification

The input consists of N cases. The first line of the input contains only positive integer N. Then follow the cases. Each case consists of exactly two lines. At the first line, there are two integers m and k, $1 <= k <= m <= 500$. At the second line, there are integers p_1, p_2, ... p_m separated by spaces. All these values are positive and less than 10000000.

Output Specification

For each case, print exactly one line. The line must contain the input succession p_1, p_2, ... p_m divided into exactly k parts such that the maximum sum of a single part should be as small as possible. Use the slash character ('/') to separate the parts. There must be exactly one space character between any two successive numbers and between the number and the slash.

If there is more than one solution, print the one that minimizes the work assigned to the first scriber, then to the second scriber etc. But each scriber must be assigned at least one book.

Sample Input

```
2
9 3
100 200 300 400 500 600 700 800 900
5 4
100 100 100 100 100
```

问题二

Mysterious Mountain

A group of M people is chasing a very strange animal. They believe that it will stay on a myterious mountain T, so they decided to climb on it and have a loot.

That is, the outline of the moutain consists of N+1 segments. The endpoints of them are numbered 0..N+1 from left to right. That is to say, x[i] < x[i+1] for all 0<=i<=n. And also, y[0]=y[n+1]=0, 1<=y[i]<=1000 for all 1<=y<=n.

According to their experience, the animal is most likely to stay at one of the N endpoits numbered 1..N . And... funny enough, they soon discover that M=N, so each of them can choose a different endpoint to seek for the animal.

Initially, they are all at the foot of the mountain. (i.e at (si,0)) For every person i, he is planing to go left/right to some place (x,0) (where x is an integer - they do not want to take time to work out an accurate place) at the speed of wi, then climb directly to the destination along a straight line(obviously, no part of the path that he follows can be OVER the mountain - they can't fly) at the speed of ci. They don't want to miss it this time, so the teamleader wants the latest person to be as early as possible. How fast can this be done?

The input will contain no more than 10 test cases. Each test case begins with a line containing a single integer N(1<=N<=100). In the following N+2 lines, each line contains two integers xi and yi(0<=xi,yi<=1000) indicating the coordinate of the ith endpoints. in the following N lines, each line contains three intergers ci,wi and si describing a person(1<=ci < wi<=100, 0<=si<=1000) - the climbing speed, walking speed and initial position. The test case containing N=0 will terminate the input and should not be regarded as a test case.

For each test case, output a single line containing the least time that these people must take to complete the mission, print the answer with two decimal places.

Sample Input

```
3
0 0
3 4
6 1
12 6
16 0
2 4 4
8 10 15
4 25 14
0
```

Sample Output

```
1.43
```