Knapsack Problems

Prerequisite modules

- Greedy
- Dynamic Programming
- Recursive Descent

Sample Problem: Tape Recording

Farmer John's favorite hobby is creating a tape containing some of Bessie's favorite music to listen to while she's being milked. The amount of milk that Bessie produces is dependent on the songs that Bessie listens to while being milked.

Given a collection of songs, each represented by a pair of integers, the length of the song (in seconds), the amount of milk Bessie produces while she's listening to that song, and the total amount of time that it takes to milk Bessie, find the set of songs such that their total length is no more than the time it takes to milk Bessie and they maximize Bessie's milk production.

The Abstraction

Given, A collection of objects, each which a size, a value (i.e., weight), and the total `space' available, find the set of objects which maximizes the sum of the value of the set, but whose sum of size is constrained by some limit. The total number/size of any particular item used in the set cannot exceed its availability.

Problem Viewpoint

The general way to view a knapsack problem is that of a bag of limited capacity, which is to be filled while maximizing the value of the objects in it.

For the problem above, the tape which Bessie will listen to while being milked is the ``knapsack,'' while the songs are the ``objects to be placed within the knapsack.''

Three Knapsack Problems

The knapsack problem style has three forms:

- Fractional knapsack problem
  A fractional knapsack problem is one in which you are allowed to place fractional objects in the knapsack. For example, if the objects were crude oil, airplane fuel, and kerosene and your knapsack a bucket, it might make sense to take 0.473 liter of the crude oil, 0.263 liter of the airplane fuel, and 0.264 liter of the kerosene. This is the easiest form of the knapsack problem to solve.
- Integer Knapsack problem
  In integer knapsack problems, only complete objects can be inserted into the knapsack. The example problem is of this form: partial songs aren't allowed.
- Multiple knapsack problem
  In the multiple knapsack problem, more than one knapsack is to be filled. If fractional objects are allowed, this is the same as having one large knapsack with capacity equal to the sum of all the available knapsacks, so this term will only be used to refer to the case of multiple integer knapsacks.

Fractional knapsack problem

The fractional knapsack problem is the easiest of the three to solve, as the greedy solution works:

- Find the object which has the highest ``value density'' (value of object / size).
- If the total amount of capacity remaining exceeds the availability of that object, put all of it in the knapsack and iterate.
- If the amount of capacity is less than the availability of the object, use as much as possible and terminate.
- This algorithm runs in $N \log N$ since it must sort the objects first based on value density and then put them into the knapsack in decreasing order until the capacity is used. It's normally easier to not sort them but rather just keep finding the highest value density not used each time, which gives a $O(N^2)$ algorithm.

Side note: For problems of this class, it's rare to have both size and availability, as you can do a trivial transformation to have all the objects of size 1, and the availability be the product of the original size and availability (dividing the value by the original size, of course).

Extensions: The value and availability of the objects can be real numbers without a problem in this case. The fractional size issue is also trivial to handle by this algorithm.

Integer knapsack problem

This is slightly more difficult, but is solvable using dynamic programming if the knapsack is small enough.

- Do dynamic programming on the maximum value that a knapsack of each size can have in it.
- Update this array for an object of size S by traversing the array in reverse order (of capacity), and seeing if placing the current object into the knapsack of size K yields a better set than the current best knapsack of size K+S.
- This algorithm runs in *K x N* time, where *K* is the size of the knapsack, and *N* is the sum of availability of objects.
- If the knapsack is too large to allocate this array, recursive descent is the method of choice, as this problem is NP-complete. Of course, recursive descent can run for a very long time in a large knapsack being filled with small objects.

Extensions:

- Fractional values are not a problem; the array just becomes an array of real numbers instead of integers. Fractional availability doesn't affect things, as you can, without loss of generality, truncate the number (if you have 3.5 objects, you can only use 3).
- Fractional size is a pain, as it makes the problem recursive descent.
- If the sizes are all the same, the problem can be solved greedily, picking the objects in decreasing value order until the knapsack is full.
- If the values are all 1.0, then again greedy works, selecting the objects in increasing size order until the knapsack is full.

Multiple knapsack problem

With multiple knapsacks of any size, the state space is too large to use the DP solution from the integer knapsack algorithm. Thus, recursive descent is the method to solve this problem. Extensions:

- With recursive descent, extensions are generally easy. Fractional sizes and values are no problem, nor is another evaluation function.
- If the values are all one, then if the maximum number of objects that can be placed in all the knapsacks is n, then there is such a solution which uses the n smallest objects. This can greatly reduce the search time.

Sample Problems

Score Inflation [1998 USACO National Championship]

You are trying to design a contest which has the maximum number of points (<10,000). Given the length of the contest, a group of problems, the problem lengths, and the point value of each problem, find the contest which has the maximum number of points (which satisfies the length constraint).

Analysis: This is an integer knapsack problem, where the knapsack is the contest, the sizes are the length of problems, and the values are the point values. The limit on knapsack (contest) size is small enough that the DP solution will work in memory.

Fence Rails [1999 USACO Spring Open]

Farmer John is trying to construct a fence around his field. He has installed the posts already, so he knows the length requirement of the rails. The local lumber store has dropped off some boards (up to 50) of varying length. Given the length of the boards from the lumber store, and the lengths of rails that Farmer John needs, calculate the maximum numbers of rails that Farmer John can create.

Analysis: This is a multiple knapsack problem, where the knapsacks are the boards from the store, and the objects are the rails that Farmer John needs. The size of the objects are just the length, and the value is just one.

Since the values are all one, you know that if there is a solution using any K rails, there is a solution using the K smallest rails, which helps the recursive descent solver quite a bit.

Filling your Tank

You're in the middle of Beaver County, at the only city within 100 miles with a gas station, trying to fill up your tank so you can get to

Rita Blanca. Fortunately, this town has a couple of gas stations, but they all seem to be almost out of gas. Given the price of gasoline at each station, and the amount of gas each one has, calculate how much gasoline to buy from each station in order to minimize the total cost.

Analysis: This is an fractional knapsack problem, where your knapsack is your gas tank, and the objects are gasoline.

背包问题

译 by 铭

（在中国，背包问题一般是这样描述的：设 n 个重量为（W1,W2,...Wn）的物品和一个载重为 S 的背包，将物品的一部分 xi 放进背包中的利润是 Pixi，问如何选择物品的种类和数量，使得背包装满而获得最大的利润？另有一简化版本说：设有一个背包可以放入的物品重量为S,现有 n 件物品,重量分别为W1,W2,...Wn。问能否从这 n 件物品中选择若干件放入此背包，使得放入的重量之和正好为 S。--译者加，不知道有没有班门弄斧之嫌）

前提

●贪心法（它是一种多步决策法，它总是作出在当前看来是最好的选择，它的考虑不是从整体出发，而只是某种意义上的局部最优，这样贪心法不能对所有问题达到整体最优解，但是对相当范围的许多问题都能够产生整体最优解。--译者）
●动态规划（它是将问题进行逐步的划分来缩小问题的规模，直到可以求出子问题的解为止。分划子问题后，对应的子问题中含有大量的重复，这样就将重复地求解；在第一次遇到重复时把它解决，并将解保存起来，以备后面引用。动态规划法常用来求一个问题在某种意义下的最优解。--译者）
●递归下降

示例问题：用录音带录音

农场主约翰最喜欢的爱好是制作一个 Bessie 喜欢的音乐合集磁带以便它在产奶时听。Bessie 的产奶量取决于它产奶时所听的歌曲。
已知一组歌曲（每首歌都由一对整数－－此曲的长度（以秒计），听该首歌时的产奶量来表示）以及给挤奶的总时间。找到这样一组歌曲的集合，使得歌曲的总长度不超过给 Bessie 挤奶的总时间且使 Bessie 的产奶量达到最大。

抽象描述

已知一组物品--每个都有其尺寸和值（比如，重量），以及可用的总空间。找到这样一个集合，使得该集合的值的和最大，且其尺寸的和受某些限制所约束。集合中任何一个特定的项目的总数目/尺寸不能超过它的可利用率。

解题想法

视其为背包问题的一般方法是一个容量受限的背包使得放入其中的物品的值达到最大。
以上述问题为例，Bessie 产奶时听的音乐带就是“背包”，而那些歌就是“放入背包中的物品”。

三个背包问题

背包问题有三种形式：
●小数背包问题
允许将小数表示的物品放入背包中的是小数背包问题。举例来说，如果物品是原油、飞机燃料、煤油而你的背包是一只水桶，取 0.473 升的原油，0,263 升的飞机燃料和 0,264 升的煤油就是有意义的。这是形式最简单的要解决的背包问题。
●整数背包问题
在整数背包问题中，只有完整的物品能放入背包里。此形式的一个例子就是：部分的曲子不允许放入包中。
●多重背包问题
在多重背包问题中，需被填充的背包多于一个。如果允许有小数的物品放入，也就等于有一个大的背包，其容量相当于所有可用背包的和。因此，此术语只用来指多重整数背包的情况。

小数背包问题

小数背包问题是三者中最简单的，其贪婪解法如下：
●找到“值密度”（物品值/尺寸）最大的物品
●如果总容量仍就超过物品的可利用率，把所有满足条件的物品放入背包中，然后反复执行。
●如果总容量少于物品的可利用率，尽可能多的使用可用空间，然后终止。
●由于这个算法必须先按照值密度把物品分类，然后以降序将它们放入背包，直至容量用完，该算法以 N log N 级运行。通常简单些的方法不是将它们分类，而是不停地找每次不用的最大值密度，这种算法的时间复杂度是 O(N 2) 。
注意：对于这类问题，因为你可以做一个微小的变换使得所有的物品尺寸大小为一，且原始尺寸大小和可利用率(当然用原始尺寸大小除值)的乘积就是总容量，同时有尺寸和可利用率是很少见的。
延伸：在这种情况下，物品的值和可利用率可以是实数。用这种算法处理有小数的尺寸大小也不是问题。

整数背包问题

这个问题有点难度，但是如果背包足够小，使用动态规划，它还是可解的。
●依据背包大小的最大值设计动态的程序。
●刷新用来表示大小为 S 的物品的数组，颠倒其次序，看将当前物品放入大小为

K 的背包中所产生的集合是否比当前最好的大小为 K+S 的背包更符合条件。
●这个算法运行 K*N 次，其中 K 是背包的大小，N 是物品的可利用率义之和。
●如果背包太大了以至于无法分配此数组，递归下降是一种选择，即这个问题是 NP 完全的（给定 I 上的一个语言 L，如果有一架非确定图灵机 M 和一个多项式 P(n)，对任何 I 上的长度为 n 的串 w，M 都可以在 P(n)步内确定是否接受 w，则称 L 是非确定图灵机下多项式时间复杂性问题，简记为 NP 问题/语言。若 L 是属于 NP 的，且对 NP 中的每一个语言 L'，都存在一个从 L'至 L 的多项式时间转化，我们说 L 是 NP 完整的。--译者）。当然，递归下降在以小的物品填充的大背包情况下可以运行相当长的一段时间。
延伸：
●小数的值不是问题；数组可以用实数数组来代替整数数组。小数的可利用率并不影响什么，在没有大量损失的条件下，缩短数字（如果你有 3,5 个物品，你可以仅用 3）。
●小数的尺寸是个讨厌的东西，它使得问题递归下降。
●如果尺寸都相同，问题就能贪婪地解开，在下降的值排序中选择物品，直到背包满为止。
●如果值都是 1.0，同样地使用贪心法，在上升的尺寸大小排序中选择物品，直到背包满为止。

多重背包问题

对于任何大小的多重背包，状态空间太大了以至于无法使用从整数背包算法中来的 DP 解法。于是递归下降是解决这个问题的方法。
延伸：
●用递归下降，通常扩展就简单了。小数的尺寸和值就不是问题了，同样地值的计算功能也不是问题。
●如果值都是同一个，那么如果能被放入所有背包中的物品的最大值是 n，则存在使用 n 个最小物品的解法。它能大大减少查找时间。

示例问题

分数膨胀[1998 USACO National Championship]

你正试图设计一个有最高分数(<10,000)的比赛。已知比赛长度，一组问题，问题的长度以及每个问题的分值，计算满足长度约束的最高分数的比赛。
分析：这是一个整数背包问题，比赛是背包，尺寸是问题的长度，值是分数值。背包（比赛）尺寸的限制是其足够小使得解法在存储器中运行。

篱笆栏[1999 USACO Spring Open]

农场主约翰准备在他的领地建一圈篱笆。他已装好了柱子，所以他知道所要的围栏长度。当地的木材店有各种长度的木板（至多 50 个）。已知木材店木板的长度，约翰要的围栏长度，计算约翰建篱笆所用的围栏最大值。
分析：这是个多重背包问题，木材店的木板是背包，物品是约翰用的围栏。物品

的尺寸就是长度，值是一。
由于值都是一，如果存在用任意 K 个围栏的解法，则有用 K 个最小围栏的解法。

装满你的油箱

你在 Beaver 郡中部一百英里有一个加油站的城市中，想将你的油箱装满好能到达 Rita Blanca。幸运地是，这个小镇有两三个加油站，但它们的油都好像要用光了。已知每个加油站的油价，每个加油站的油量，计算为了花最少的钱，应该从每个加油站买多少汽油。
分析：这是一个小数背包问题，背包是油箱，物品是汽油。