动态规划

维基百科，自由的百科全书

动态规划（英语：Dynamic programming，简称DP）是一种在数学、管理科学、计算机科学、经济学和生物信息学中使用的，通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法。

动态规划常常适用于有重叠子问题[1]和最优子结构性质的问题，动态规划方法所耗时间往往远少于朴素解法。

动态规划背后的基本思想非常简单。大致上，若要解一个给定问题，我们需要解其不同部分（即子问题），再合并子问题的解以得出原问题的解。

通常许多子问题非常相似，为此动态规划法试图仅仅解决每个子问题一次，从而减少计算量：一旦某个给定子问题的解已经算出，则将其记忆化存储，以便下次需要同一个子问题解之时直接查表。这种做法在重复子问题的数目关于输入的规模呈指數增長时特别有用。

目录

概述

动态规划在查找有很多**重叠子问题**的情况的最优解时有效。它将问题重新组合成子问题。为了避免多次解决这些子问题，它们的结果都逐渐被计算并被保存，从简单的问题直到整个问题都被解决。因此，动态规划保存递归时的结果，因而不会在解决同样的问题时花费时间。

动态规划只能应用于有**最优子结构**的问题。最优子结构的意思是局部最优解能决定全局最优解（对有些问题这个要求并不能完全满足，故有时需要引入一定的近似）。简单地说，问题能够分解成子问题来解决。

适用情况

1. 最优子结构性质。如果问题的最优解所包含的子问题的解也是最优的，我们就称该问题具有最优子结构性质（即满足最优化原理）。最优子结构性质为动态規劃算法解决问题提供了重要线索。
2. 无后效性。即子问题的解一旦确定，就不再改变，不受在这之后、包含它的更大的问题的求解决策影响。
3. 子问题重叠性质。子问题重叠性质是指在用递归算法自顶向下对问题进行求解时，每次产生的子问题并不总是新问题，有些子问题会被重复计算多次。动态規劃算法正是利用了这种子问题的重叠性质，对每一个子问题只计算一次，然后将其计算结果保存在一个表格中，当再次需要计算已经计算过的子问题时，只是在表格中简单地查看一下结果，从而获得较高的效率。

实例

斐波那契数列（Fibonacci polynomial）

计算斐波那契数列（Fibonacci polynomial）的一个最基础的算法是，直接按照定义计算（函数递归）：

```
  function fib(n)
      if n = 0 or n = 1
          return n
      return fib(n − 1) + fib (n − 2)
```

当n=5时，fib(5)的计算过程如下:

1. fib(5)
2. fib(4) + fib(3)
3. (fib(3) + fib(2)) + (fib(2) + fib(1))
4. ((fib(2) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))
5. (((fib(1) + fib(0)) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))

由上面可以看出，这种算法对于相似的子问题进行了重复的计算，因此不是一种高效的算法。实际上，该算法的运算时间是指数级增长的。 改进的方法是，我们可以通过保存已经算出的子问题的解来避免重复计算：

```
array map [0...n] = { 0 => 0, 1 => 1 }
fib (n)
    if (map m does not contain key n )
        m[n] := fib(n − 1) + fib (n − 2)
    return m[n]
```

将前n个已经算出的数保存在数组map中，这样在后面的计算中可以直接应用前面的结果，从而避免了重复计算。算法的运算时间变为$O(n)$

背包问题

背包问题作为NP完全问题，暂时不存在多项式时间算法。动态规划属于背包问题求解最优解的可行方法之一。此外，求解背包问题最优解还有搜索法等，近似解还有贪心法等，分数背包问题有最优贪心解等。 背包问题具有最优子结构和重叠子问题。动态规划一般用于求解背包问题中的整数背包问题（即每种物品所选的个数必须是整数）。 解整数背包问题： 设有n件物品，每件价值记为Pi，每件体积记为Vi，用一个最大容积为Vmax的背包，求装入物品的最大价值。 用一个数组f[i,j]表示取i件商品填充一个容积为j的背包的最大价值，显然问题的解就是f[n,Vmax].

f[i,j]=

```
     f[i-1,j] {j<Vi}
     max{f[i-1,j],f[i,j-Vi]+Pi} {j>=Vi}
     0 {i=0 OR j=0}
```

对于特例01背包问题（即每件物品最多放1件，否则不放入）的问题，状态转移方程：

f[i,j]=

```
     f[i-1,j] {j<Vi}
     max{f[i-1,j],f[i-1,j-Vi]+Pi} {j>=Vi}
     0 {i=0 OR j=0}
```

参考Pascal代码

```
for i:=1 to n do
  for j:=totv downto v[i] do
    f[j]:=max(f[j],f[j-v[i]]+p[i]);
writeln(f[totv]);
```

参考C++代码（不含include和数组声明）

```
#define max(x,y) x>y?x:y  //max宏函数，也可以自己写或者调取algorithm
for(int i=1;i<=n;i++)
  for (j=totv;j>=v[i];j--)
    f[j]=max(f[j],f[j-v[i]]+p[i]);
printf("%d",f[totv]); //或std:cout<<f[totv];
```

使用动态规划的算法

- 最长公共子序列
- Floyd-Warshall算法
- Viterbi算法

参考

1. S. Dasgupta, C.H. Papadimitriou, and U.V ．Vazirani, '**Algorithms**' , p 173, available at http://www .cs.berkeley .edu/~vazirani/algorithms.html

外部链接