背包方案问题

输出方案

一般而言，背包问题是要求一个最优值，如果要求输出这个最优值的方案，可以参照一般动态规划问题输出方案的方法：记录下每个状态的最优值是由状态转移方程的哪一项推出来的，换句话说，记录下它是由哪一个策略推出来的。便可根据这条策略找到上一个状态，从上一个状态接着向前推即可。还是以01背包为例，方程为

$$f[i][j] = max(f[i-1][j], f[i-1][j-w[i]]+v[i])$$

再用一个数组$g[i][j]$，设$g[i][j]=0$表示推出$f[i][j]$的值时是采用了方程的前一项（也即$f[i][j]=f[i-1][j]$），$g[i][j]$表示采用了方程的后一项。注意这两项分别表示了两种策略：未选第i个物品及选了第i个物品，最终状态为$f[N][V]$。

另外，采用方程的前一项或后一项也可以在输出方案的过程中根据$f[i][j]$的值实时地求出来，也即不须纪录g数组，将上述代码中的$g[i][j]==0$改成$f[i][j]==f[i-1][j]$，$g[i][j]==1$改成$f[i][j]==f[i-1][j-w[i]]+v[i]$也可。

输出字典序最小的最优方案

这里“字典序最小”的意思是$1\cdots N$号物品的选择方案排列出来以后字典序最小。以输出01背包最小字典序的方案为例。一般而言，求一个字典序最小的最优方案，只需要在转移时注意策略。首先，子问题的定义要略改一些。我们注意到，如果存在一个选了物品1的最优方案，那么答案一定包含物品1，原问题转化为一个背包容量为$j-w[1]$，物品为$2\cdots N$的子问题。反之，如果答案不包含物品1，则转化成背包容量仍为V，物品为$2\cdots N$的子问题。不管答案怎样，子问题的物品都是以$i\cdots N$而非前所述的$1\cdots i$的形式来定义的，所以状态的定义和转移方程都需要改一下。但也许更简易的方法是先把物品逆序排列一下，以下按物品已被逆序排列来叙述。在这种情况下，可以按照前面经典的状态转移方程来求值，只是输出方案的时候要注意：从N到1输入时，如果$f[i][j]==f[i-1][i-j]$及$f[i][j]==f[i-1][j-w[i]]+v[i]$同时成立，应该按照后者（即选择了物品$i$）来输出方案。

题目描述

有N件物品和一个容量是V的背包。每件物品只能使用一次。

第i件物品的体积是v_i，价值是w_i。

求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。

输出**字典序最小的方案。**这里的字典序是指：所选物品的编号所构成的序列。物品的编号范围是$1\cdots N$。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品数量和背包容积。

接下来有N行，每行两个整数v_i, w_i，用空格隔开，分别表示第i件物品的体积和价值。

输出格式

输出一行，包含若干个用空格隔开的整数，表示最优解中所选物品的编号序列，且该编号序列的字典序最小。

物品编号范围是$1\cdots N$。

数据范围

$0<N,V\le 1000$

$0<vi,wi\le 1000$

输入样例

```
4 5
1 2
2 4
3 4
4 6
```

输出样例：

```
1 4
```

In [1]:

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#ifdef __cplusplus  //曾经的C/C++，使用这个宏
#define Max(a, b) ((a > b) ? (a) : (b))
extern "C" {
    void fnpack(){
        freopen("dp04beibao08_01.in","r",stdin);
        const int N = 1010;
        int n, m;
        int f[N][N], w[N], v[N];
        memset(f,0,sizeof(f));
        memset(f,0,sizeof(w));
        memset(f,0,sizeof(N));
        scanf("%d %d",&n,&m);
        for(int i = 1; i <= n; i++) scanf("%d %d",&v[i],&w[i]);

        for(int i = n; i >= 1; i--)
            for(int j = 0; j <= m; j++){
                f[i][j] = f[i + 1][j];
                if(j >= v[i]) f[i][j] = Max(f[i][j], f[i + 1][j - v[i]] + w[i]);
            }

        int vol = m;

        for(int i = 1; i <= n; i++){
            if(i==n&&vol>=v[i]){
                printf("%d ",i);
                break;
            }
            if(f[i][vol] == f[i + 1][vol - v[i]] + w[i]){
                printf("%d ",i);
                vol -= v[i];
            }
            if(vol<0){
                break;
            }
        }
    }
}
#endif
```

Out[1]:

In [2]:

```
fnpack();
```

```
1 4
```

Out[2]:

```
(void) nullptr
```

求次优解、第K优解

对于求次优解、第K优解类的问题，如果相应的最优解问题能写出状态转移方程、用动态规划解决，那么求次优解往往可以相同的复杂度解决，第K优解则比求最优解的复杂度上多一个系数K。

其基本思想是将每个状态都表示成有序队列，将状态转移方程中的max/min转化成有序队列的合并。这里仍然以01背包为例讲解一下。

首先看01背包求最优解的状态转移方程：

$$f[i][j] = max(f[i-1][j], f[i-1][j-w[i]] + v[i])$$

如果要求第K优解，那么状态$f[i][j]$就应该是一个大小为K的数组$f[i][j][1 \cdots K]$。其中$f[i][j][k]$表示前i个物品、背包大小为j时，第k优解的值。“$f[i][j]$是一个大小为K的数组”这一句，熟悉C语言的同学可能比较好理解，或者也可以简单地理解为在原来的方程中加了一维。显然$f[i][j][1 \cdots K]$这K个数是由大到小排列的，所以我们把它认为是一个有序队列。然后原方程就可以解释为：$f[i][j]$这个有序队列是由$f[i-1][j]$和$f[i-1][j-w[i]] + v[i]$这两个有序队列合并得到的。有序队列$f[i-1][j]$即$f[i-1][j][1 \cdots K]$，$f[i-1][j-w[i]] + v[i]$则理解为在$f[i-1][j-w[i]][1 \cdots K]$的每个数上加上$v[i]$后得到的有序队列。合并这两个有序队列并将结果的前$K$项储存到$f[i][j][1 \cdots K]$中的复杂度是$O(K)$。最后的答案是$f[N][V][K]$。总的复杂度是$O(VNK)$。为什么这个方法正确呢？实际上，一个正确的状态转移方程的求解过程遍历了所有可用的策略，也就覆盖了问题的所有方案。只不过由于是求最优解，所以其它在任何一个策略上达不到最优的方案都被忽略了。如果把每个状态表示成一个大小为K KK的数组，并在这个数组中有序的保存该状态可取到的前K KK个最优值。那么，对于任两个状态的max maxmax运算等价于两个由大到小的有序队列的合并。另外还要注意题目对于“第K KK优解”的定义，将策略不同但权值相同的两个方案是看作同一个解还是不同的解。如果是前者，则维护有序队列时要保证队列里的数没有重复的。代码：

```
int kth(int n, int V, int k) {
    for (int i = 1; i <= n; i++) {
        for (int j = V; j >= w[i]; j--) {
            for (int l = 1; l <= k; l++) {
                a[l] = f[j][l];
                b[l] = f[j - w[i]][l] + v[i];
            }
            a[k + 1] = -1;
            b[k + 1] = -1;
            int x = 1, y = 1, o = 1;
            while (o != k + 1 and (a[x] != -1 or b[y] != -1)) {
                if (a[x] > b[y]) f[j][o] = a[x], x++;
                else f[j][o] = b[y], y++;
                if (f[j][o] != f[j][o - 1]) o++;
            }
        }
    }
    return f[V][k];
}
```

例题：

HDU 2639 Bone Collector II (http://acm.hdu.edu.cn/showproblem.php?pid=2639)

In [ ]: