动态规划专题讲义

专题七：多重背包问题

/\*

Name: 动态规划专题之多重背包问题

Author: 巧若拙

Description: 多重背包问题：在n种物品中选取若干件（第i种物品最多选N[i]次）放在容量为c的背包里，分别用P[i]和W[i]存储第i种物品的价值和重量。

求解怎么装物品可使背包里物品总价值最大。

输入

第一行2个数n和c，表示共有n种物品，背包总容量为c

接下来n行，每行3个数，分别表示第i种物品的重量，价值和最大数量

输出

一个整数，表示背包里物品最大总价值

样例输入

5 1000

80 20 4

40 50 9

30 50 7

40 30 6

20 20 1

样例输出

1040

\*/

#include<iostream>

#include<cmath>

using namespace std;

const int MAXC = 6000; //背包最大容量

const int MAXN = 2000; //物品的个数

int W[MAXN+1];//物品的重量

int P[MAXN+1];//物品的价值

int N[MAXN+1];//物品的最大数量

int F1[MAXC+1]; //记录装入容量为c的背包的最大价值

int B1[MAXN+1][MAXC+1]; //备忘录，记录给定n个物品装入容量为c的背包的最大价值

int pre[MAXC+1]; //pre[j]相当于B1[i-1][j]

int cur[MAXC+1]; //cur[j]相当于B1[i][j]

int F2[MAXC+1]; //记录装入容量为c的背包的最大价值

int B2[MAXN+1][MAXC+1]; //备忘录，记录给定n个物品装入容量为c的背包的最大价值

int MultiPack\_1(int n, int c);//多重背包问题：二维数组+朴素的穷举思想

int MultiPack\_2(int n, int c);//多重背包问题：一维数组+朴素的穷举思想

int MultiPack\_3(int n, int c);//多重背包问题：2个一维数组记录最优解

int MultiPack\_4(int n, int c);//多重背包问题：一维数组+对第i种物品进行N[i]次0-1选择

int main()

{

int n, c;

cin >> n >> c;

for (int i=1; i<=n; i++)//不计下标为0的元素

{

cin >> W[i] >> P[i] >> N[i];

}

cout << MultiPack\_1(n, c) << endl;

cout << MultiPack\_2(n, c) << endl;

cout << MultiPack\_3(n, c) << endl;

cout << MultiPack\_4(n, c) << endl;

return 0;

}

算法1：二维数组+朴素的穷举思想，需要用到全局变量N[], W[], P[], 另有B1[MAXN+1][]初始化为0。

int MultiPack\_1(int n, int c)

{

int bestP;

for (int i=1; i<=n; i++)

{

for (int j=1; j<=c; j++)

{

bestP = 0;

for (int k=0; k<=N[i] && k\*W[i]<=j; k++) // 语句1

{

if (bestP < B1[i-1][j-k\*W[i]] + k\*P[i])

bestP = B1[i-1][j-k\*W[i]] + k\*P[i];

}

B1[i][j] = bestP;

}

}

return B1[n][c];

}

问题1：语句1能否改为：for (int k=min(N[i],j/W[i]); k>=0; k--)？为什么？

参考答案：

问题1：可以，因为语句1所在循环体的作用是穷举装入第i种物品的数量，并计算出能获得的最大价值，循环变量k值递增或递减均可，实际上修改后的语句1效率更高。

算法2：一维数组+朴素的穷举思想，需要用到全局变量N[], W[], P[], 另有F1[MAXC+1]初始化为0。

int MultiPack\_2(int n, int c)

{

int bestP;

for (int i=1; i<=n; i++)

{

for (int j=c; j>=W[i]; j--) //语句1

{

bestP = 0;

for (int k=0; k<=N[i] && k\*W[i]<=j; k++)

{

if (bestP < F1[j-k\*W[i]] + k\*P[i])

bestP = F1[j-k\*W[i]] + k\*P[i];

}

F1[j] = bestP;

}

}

return F1[c];

}

问题1：语句1能否改为：for (int j=W[i]; j<=c; j++)？为什么？

参考答案：

问题1：不可以。算法2是由算法1直接优化而得，与0-1背包问题类似，注意到第i行第j列的元素，由第i-1行的元素决定，且列坐标j大的元素由j小的元素决定，故在同一行中，必须先求出列坐标较大的元素，再求列坐标小的元素，即在第二层循环中，应该让循环变量j的值从大到小递减。

算法3：2个一维数组记录最优解，需要用到全局变量N[], W[], P[], 另有pre[]和cur[]均初始化为0。

int MultiPack\_3(int n, int c)

{

//cur[j]表示给定i个物品的情况下，背包容量为j时，对物品进行第k次选择时所能获得的最优解

//pre[j]表示给定i个物品的情况下，背包容量为j时，对物品进行第k-1次选择时所能获得的最优解

for (int i=1; i<=n; i++)

{

for (int k=0; k<N[i]; k++)//对第i种物品进行N[i]次选择

{

for (int j=1; j<=c; j++)

{

if (j < W[i] || pre[j] > pre[j-W[i]] + P[i])

cur[j] = //语句1

else

cur[j] = //语句2

}

for (int j=1; j<=c; j++)//更新后的pre[j]有可能是选择了第i种物品

{

pre[j] = cur[j];

}

}

}

return pre[c];

}

问题1：将语句1和语句2补充完整。

问题2：本算法看上去与完全背包问题的算法3差不多，请问二者的pre和cur数组的有何异同？

参考答案：

问题1：语句1：cur[j] = pre[j];

语句2：cur[j] = pre[j-W[i]] + P[i];

问题2：完全背包算法3中pre[j]相当于B2[i-1][j]，cur[j]相当于B2[i][j]，即分别表示给定(i-1)个和i个物品的情况下，背包容量为j时的最优解；而本算法中的cur和pre分别表示给定i个物品的情况下，背包容量为j时，对物品进行第(k-1)次和k次选择时所能获得的最优解。本算法比完全背包算法3多了第二层循环，而且语句2也不一样。

算法4：1个一维数组记录最优解，需要用到全局变量N[], W[], P[]，另有F2[MAXC+1]初始化为0。

int MultiPack\_4(int n, int c)

{

for (int i=1; i<=n; i++)

{

for (int k=0; k<N[i]; k++)//对第i种物品进行N[i]次选择

{

for (int j=c; j>=W[i]; j--) //语句1

{

if (F2[j] < F2[j-W[i]] + P[i])

F2[j] = F2[j-W[i]] + P[i];

}

}

}

return F2[c];

}

问题1：语句1能否改为：for (int j=W[i]; j<=c; j++)？为什么？

问题2：本算法看上去与0-1背包问题的算法5一样（就是多了第二层循环），请问二者有何关系？

参考答案：

问题1：不能。算法4是对算法3的降维优化。虽然算法4没有对应的二维数组算法，但是我们仍然可以将算法3看作一个只有2行的二维数组，cur和pre分别表示当前行和上一行。

因为cur[j] 由pre[j]或pre[j-W[i]]决定，即第i行的某个元素可能由上一行中列坐标j较小的元素来决定，而现在我们用1个一维数组F2 []来代替cur[]和pre[]，则只记录了列坐标j，未记录行坐标i，在同一行中，必须先求出列坐标j较大的元素，再求j较小的元素。这样先改变的是下标j较大的元素，且其不会影响j小的元素。故在内层循环中，应该让循环变量j的值从大到小递减。

问题2：0-1背包问题是多重背包问题的一个特例（N[i]=1），因此他们的代码结构必然是相同的，如果多重背包问题的第2层循环for (int k=0; k<N[i]; k++)只执行一次的话，就变成了一个纯粹的0-1背包问题。

因为每轮循环过后，F2[j]的值都有可能更新，则对第i个物品进行第k次选择的时候，背包中可能已经装载了第i个物品了，这是本算法0-1背包中算法5 不同的地方。

拓展练习：通过算法1的MultiPack\_1 ()函数，我们用二维数组B1[][]记录了各种解的信息，现在请你根据B1[][]记录的信息，设计一个递归函数void Show(int i, int j);//i和j分别表示正在处理的第i个物品和此时背包的剩余容量。

输出物品装载情况，按照编号顺序，递归输出装入背包的物品信息（编号，数量，重量，价值）。

参考答案：

void Show(int i, int j) //i和j分别表示正在处理的第i个物品和此时背包的剩余容量

{

if (j == 0 || i == 0)

return;

if (B1[i][j] == B1[i-1][j])

{

Show(i-1, j); //未装载物品i

}

else

{

for (int k=min(N[i],j/W[i]); k>=0; k--)

{

if (B1[i][j] == B1[i-1][j-k\*W[i]] + k\*P[i]) //装载了k个物品i

{

Show(i-1, j-k\*W[i]);

cout << i << ": " << k << " " << W[i] << " " << P[i] << endl;

return;

}

}

}

}

课后练习：

练习1： 购买大米的最多重量

描述： 为了挽救灾区同胞的生命，心系灾区同胞的你准备自己采购一些粮食支援灾区，现在假设你一共有资金n元，而市场有m种大米，每种大米都是袋装产品，其价格不等，并且只能整袋购买。

请问：你用有限的资金最多能采购多少公斤粮食呢？

Input：输入数据首先包含一个正整数C，表示有C组测试用例，每组测试用例的第一行是两个整数n和m(1<=n<=100, 1<=m<=100),分别表示经费的金额和大米的种类，然后是m行数据，每行包含3个数p，h和c(1<=p<=20,1<=h<=200,1<=c<=20)，分别表示每袋的价格、每袋的重量以及对应种类大米的袋数。

Output：对于每组测试数据，请输出能够购买大米的最多重量，你可以假设经费买不光所有的大米，并且经费你可以不用完。每个实例的输出占一行。

Sample Input

1

8 2

2 100 4

4 100 2

Sample Output

400

练习2：混合背包问题

描述：在n种物品中选取若干件放在容量为c的背包里，分别用P[i]和W[i]存储第i种物品的价值和重量。

有的物品只可以取一次（01背包），有的物品可以取无限次（完全背包），有的物品可以取的次数有一个上限（多重背包）

求解怎么装物品可使背包里物品总价值最大。

输入 ：第一行2个数n和c，表示共有n种物品，背包总容量为c

接下来n行，每行3个数Wi，Pi和Ni，分别表示第i种物品的重量，价值和最大数量（若Ni=0，则表示此物品可取无限次）

输出 ：一个整数，表示背包里物品最大总价值

样例输入

3 10

2 1 0

3 3 1

4 5 4

样例输出

11

练习3：机器分配

描述：总公司拥有高效设备M台，准备分给下属的N个分公司。各分公司若获得这些设备，可以为国家提供一定的盈利。问：如何分配这M台设备才能使国家得到的盈利最大？求出最大盈利值。其中M≤15，N≤10。分配原则：每个公司有权获得任意数目的设备，但总台数不超过设备数M。

Input ：输入数据文件格式为：第一行有两个数，第一个数是分公司数N，第二个数是设备台数M。

接下来是一个N\*M的矩阵，表明了第 I个公司分配 J台机器的盈利。

Output ：输出第一行为最大盈利值；

接下来有n行，分别为各分公司分配的机器数。

Sample Input

3 3 {三个分公司分3台机器}

30 40 50

20 30 50

20 25 30

Sample Output

70 {最大盈利值为70}

1 1 {第一分公司分1台}

2 1 {第二分公司分1台}

3 1 {第三分公司分1台}

练习4：复制书稿

Problem Description：现在要把m本有顺序的书分给k个人复制(抄写)，每一个人的抄写速度都一样，一本书不允许给两个(或以上)的人抄写，分给每一个人的书，必须是连续的，比如不能把第一、第三和第四本书给同一个人抄写。

现在请你设计一种方案，使得复制时间最短。复制时间为抄写页数最多的人用去的时间。

Input ：输入有多组数据，每组数据第1行两个整数m,k(k<=m<=500)；

第2行m个整数，第i个整数表示第i本书的页数。

Output ：对于每组数据输出k行，每行两个整数，第i行表示第i个人抄写的书的起始编号和终止编号。k行的起始编号应该从小到大排列，如果有多解，则尽可能让前面的人少抄写。

Sample Input

9 3

1 2 3 4 5 6 7 8 9

Sample Output

1 5

6 7

8 9

练习5：花店橱窗布置(flower.cpp)

描述： 假设以最美观的方式布置花店的橱窗，有F束花，每束花的品种都不一样，同时，至少有 同样数量的花瓶，被按顺序摆成一行，花瓶的位置是固定的，并从左到右，从1到V顺序编 号，V 是花瓶的数目，编号为1的花瓶在最左边，编号为V的花瓶在最右边，花束可以移动， 并且每束花用1到F 的整数惟一标识，标识花束的整数决定了花束在花瓶中列的顺序即如果 i < j，则花束i 必须放在花束j左边的花瓶中。

例如，假设杜鹃花的标识数为1，秋海棠的标识数为2，康乃馨的标识数为3，所有的花束在 放入花瓶时必须保持其标识数的顺序，即：杜鹃花必须放在秋海棠左边的花瓶中，秋海棠 必须放在康乃馨左边的花瓶中。如果花瓶的数目大于花束的数目，则多余的花瓶必须空，即每个花瓶中只能放一束花。每一个花瓶的形状和颜色也不相同，因此，当各个花瓶中放入不同的花束时会产生不同的美学效果，并以美学值(一个整数)来表示，空置花瓶的美学值为0。在上述例子中，花瓶与花束的不 同搭配所具有的美学值，可以用如下表格表示。

花瓶 1 2 3 4 5

1 (杜鹃花) 7 23 -5 -24 16

2 (秋海棠) 5 21 -4 10 23

3 (康乃馨) -21 5 -4 -20 20

根据表格，杜鹃花放在花瓶2中，会显得非常好看，但若放在花瓶4中则显得很难看。 为取得最佳美学效果，必须在保持花束顺序的前提下，使花的摆放取得最大的美学值，如果具有最大美学值的摆放方式不止一种，则输出任何一种方案即可。

题中数据满足下面条件：1≤F≤100，F≤V≤100，－50≤Aij≤50，其中Aij是花束i摆放在花瓶j中的美学值。输入整数F，V 和矩阵(Aij)，输出最大美学值和每束花摆放在各个花瓶中的花瓶编号。

输入：第一行包含两个数：F，V。 随后的F行中，每行包含V个整数，Aij 即为输入文件中第（i+1）行中的第j个数

输出：包含两行，第一行是程序所产生摆放方式的美学值。第二行必须用F个数表示摆放方式，即该行的第K个数表示花束K所在的花瓶的编号。

输入样例

3 5

7 23 -5 -24 16

5 21 -4 10 23

-21 5 -4 -20 20

输出样例

53

2 4 5