**简单描述**

0-1背包问题描述如下：

有一个容量为V的背包，和一些物品。这些物品分别有两个属性，体积w和价值v，每种物品只有一个。要求用这个背包装下价值尽可能多的物品，求该最大价值，背包可以不被装满。因为最优解中，每个物品都有两种可能的情况，即在背包中或者不存在（背 包中有0个该物品或者 1个），所以我们把这个问题称为0-1背包问题。

**0-1背包问题状态转移方程**

用dp[i][j]表示前i个物品在总体积不超过j的情况下，放到背包里的最大价值。由此可以推出状态转移方程：

dp[0][j] = 0;

dp[i][j] = max{dp[i-1][j-v[i]] + w[i],dp[i-1][j]};

上面的式子应该很好理解，当第i物品的体积小于当前剩余的体积，则说明可以装入背包，那么dp[i][j] = dp[i-1][j-v[i]]+w[i]。反之就是不能转入背包，dp[i][j] = dp[i-1][j]。

**0-1背包问题实现算法1**

[复制代码](javascript:void(0);)

#include <iostream>

using namespace std;

#define MAXSIZE 100

int w[MAXSIZE];

int v[MAXSIZE];

int maxv;

int n;

int dp[MAXSIZE][MAXSIZE];

int max(int a, int b)

{

if (a > b)

return a;

else

return b;

}

int main()

{

cin >> n >> maxv;

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

{

cin >> w[i] >> v[i];

}

for (int i = 0; i <= maxv; i++)

dp[0][i] = 0;

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

{

//只有当j >= w[i],dp[i][j]才能进行选取最大值

for (int j = maxv; j >= w[i]; j--)

{

dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);

}

//当j < w[i]，说明第i个物品是不能转入背包的，故dp[i][j] = dp[i-1][j]

for (int j = w[i] - 1; j >= 0; j--)

dp[i][j] = dp[i - 1][j];

}

cout << dp[n][maxv] << endl;

return 0;

}

[复制代码](javascript:void(0);)

 看到这里，还没完，哈哈！下面介绍一下对dp的优化，我们发现这里的dp是一个二维数组，其实dp完全可以用一维数组表示。为啥子？？？

**0-1背包问题实现算法2**

看这里：可以发现0-1背包的状态转移方程 dp[i][j] = max{dp[i-1][j-w[i]]+v[i]，dp[i-1][j]}的特点，当前状态仅依赖前一状态的剩余体积与当前物品体积v[i]的关系。根据这个特点，我们可以将dp降到一维即dp[j] = max{dp[j]，dp[j-w[i]]+v[i]}。从这个方程中我们可以发现，有两个dp[j]，但是要区分开。等号左边的dp[j]是当前i的状态，右边中括号内的dp[j]是第i-1状态下的值。

所以为了保证状态的正确转移，我们需要先更新等号左边中的dp[j]（当前状态的dp[j]）。

[复制代码](javascript:void(0);)

#include <iostream>

using namespace std;

#define MAXSIZE 100

int w[MAXSIZE];

int v[MAXSIZE];

int maxv;

int n;

int dp[MAXSIZE];

int max(int a, int b)

{

if (a > b)

return a;

else

return b;

}

int main()

{

cin >> n >> maxv;

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

{

cin >> w[i] >> v[i];

}

for (int i = 0; i <= maxv; i++)

dp[i] = 0;

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

{

//只有当j >= w[i],dp[j]才能进行选取最大值,否则dp[j]将不作更新，等于dp[i-1][j]。

for (int j = maxv; j >= w[i]; j--)

{

dp[j] = max(dp[j], dp[j - w[i]] + v[i]);

}

}

cout << dp[maxv] << endl;

return 0;

}

[复制代码](javascript:void(0);)

比较上面的两个算法可以发现，它们的时间复杂度都是O(n\*maxv)，只有空间复杂度发生变化。后者的空间复杂度得到了优化。

**拓展0-1背包问题**

哈哈，还没完，继续0-1背包问题，如果在上面的问题加上一个限制条件，所选择的物品必须恰好装满背包，否则输出-1。

同样的给出两种算法，它们的时间复杂度都是一样的，只不过是空间复杂度不同。

**空间复杂度为O(n\*maxv)的算法**

[复制代码](javascript:void(0);)

#include <iostream>

using namespace std;

#define MAXSIZE 100

int w[MAXSIZE];

int v[MAXSIZE];

int maxv;

int n;

int dp[MAXSIZE][MAXSIZE];

int max(int a, int b)

{

if (a > b)

return a;

else

return b;

}

int main()

{

cin >> n >> maxv;

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

{

cin >> w[i] >> v[i];

}

//初始化，当容积为0时，即不能装入，最大价值即为0

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

{

dp[i][0] = 0;

}

//初始化为-1，表示没有装满

for (int i = 0; i <= n; i++)

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

dp[i][j] = -1;

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

{

for (int j = maxv; j >= w[i]; j--)

{

//dp[i - 1][j - w[i]] != -1表示容积为j - w[i]时没有装满，所以当容积为j，装w[i]时一定不能装满

//dp[i - 1][j - w[i]] + v[i] > dp[i-1][j]表示装入物品i时签好装满并且总价值比前i-1个物品的总价值要大

if (dp[i - 1][j - w[i]] != -1 && dp[i - 1][j - w[i]] + v[i] >= dp[i - 1][j])

dp[i][j] = dp[i - 1][j - w[i]] + v[i];

}

for (int j = w[i] - 1; j >= 1; j--)

dp[i][j] = dp[i - 1][j];

}

cout << dp[n][maxv] << endl;

return 0;

}

[复制代码](javascript:void(0);)

**空间复杂度为O(maxv)的算法**

[复制代码](javascript:void(0);)

#include "stdafx.h"

#include <iostream>

using namespace std;

#define MAXSIZE 100

int w[MAXSIZE];

int v[MAXSIZE];

int maxv;

int n;

int dp[MAXSIZE];

int max(int a, int b)

{

if (a > b)

return a;

else

return b;

}

int main()

{

cin >> n >> maxv;

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

{

cin >> w[i] >> v[i];

}

//初始化，当容积为0时，即不能装入，最大价值即为0

dp[0] = 0;

//初始化为-1，表示没有装满

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

dp[j] = -1;

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

for (int j = maxv; j >= w[i]; j--)

{

if (dp[j - w[i]] != -1 && dp[j - w[i]] + v[i] >= dp[j])

dp[j] = dp[j - w[i]] + v[i];

}

cout << dp[maxv] << endl;

return 0;

}

[复制代码](javascript:void(0);)

从上面的算法我们发现，这里的状态转移方程和0-1背包问题的状态转移方程是一样一样滴，只不过是初试状态发生了一点改变。

呵呵，到这里0-1背包问题先结束了，后面会继续介绍更加复杂的背包问题。

ps:走一步，学一步，总结一步，路就不会太远，目标也不会遥不可及。

写代码是一种艺术，甚于蒙娜丽莎的微笑。