完全背包问题

- 简介：有一个容量大小为M的背包，每种物品有相对应的价值，每种物品可以在背包容量的前提下选择多次。问这个背包的可以得到的最大价值的方案是什么？

1.暴力方法解决

1.1 （逆着来）：

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e3 + 10;
int inf = -(1e9 + 10);
int n, m;
int f[maxn][maxn];
int v[maxn], w[maxn];
int dfs(int now, int have) //分别表示第几个，以及现在的背包重量是多少。
{
    if (have > m)
        return inf;
    if (now > n)
        return 0;
    int t = inf;
    for (int i = 0; have + have*w[now] <= m; i++)
        t = max(t, dfs(now + 1, have + w[now] * i) + v[now] * i);
    return t; //两个方向。
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i];
    cout << dfs(1, 0) << '\n';
}
```

1.11tips

- 迭代的结构有循环结构，递归的结构。将一个状态比喻为一个节点，那么循环结构之间的关系是一条线。而递归结构可以是多叉树，一条线。
- 用递归写暴力解法：
    - 1.递归的要点明白一个函数的作用并相信它能完成这个任务，千万不要跳进这个函数里面企图探究更多细节。
        - 典型问题： 深度优先搜索，综合一些量的贡献结果。

- 2.当前的暴力解法下，函数的作用就是解决每一个子问题。与参数相关，选择后now个物品，当前的背包容量为have的最优解。而这样一个过程，和其它函数之间的状态转移相关。就这样转移，最终完成这一个功能。
- 像上面这一个问题，用数学归纳法来理解。

1.2 正着来：

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e3 + 10;
int inf = -(1e9 + 10);
int n, m;
int f[maxn][maxn];
int v[maxn], w[maxn];
int dfs(int now, int have) //分别表示第几个，以及现在的背包重量是多少。
{
    if (now == 0)
        return 0;
    int t = inf;
    for (int i = 0; have - i * w[now] >= 0; i++)
        t = max(t, dfs(now - 1, have - w[now] * i) + v[now] * i);
    return t; //两个方向。
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i];
    cout << dfs(n, m) << '\n';
}
```

2 记录结果减少递归时间花销。

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e3 + 10;
int inf = -(1e9 + 10);
int n, m;
int f[maxn][maxn];
int v[maxn], w[maxn];
int dfs(int now, int have) //分别表示第几个，以及现在的背包重量是多少。
{
    if (f[now][have] > 0)
        return f[now][have];
    if (now == 0)
        return 0;
    int t = inf;
    return f[now][have] = t; //两个方向。
}
int main()
```

```
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i];
    cout << dfs(n, m) << '\n';
}
```

3.对记忆化搜索的剪枝。

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e3 + 10;
int inf = -(1e9 + 10);
int n, m;
int f[maxn][maxn];
int v[maxn], w[maxn];
int dfs(int now, int have) //分别表示第几个，以及现在的背包重量是多少。
{
    if (have < 0)
        return inf;
    if (f[now][have] > 0)
        return f[now][have];
    if (now == 0)
        return 0;
    int t = inf;
    t = max(dfs(now - 1, have), dfs(now, have - w[now]) + v[now]);
    return f[now][have] = t; //两个方向。
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i];
    cout << dfs(n, m) << '\n';
}
```

4抽象出二重循环

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1010;
int v[maxn], w[maxn];
int f[maxn][maxn];
int main()
{
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i];
    for (int i = 1; i <= n; i++)
```

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

5利用滚动数组来改进二重循环：

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1003;
int v[maxn], w[maxn], f[maxn];
int main()
{
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i];
        for (int i = 1; i <= n; i++)
            for (int j = w[i]; j <= m; j++)
                f[j] = max(f[j], f[j - w[i]] + v[i]);
    cout << f[m] << '\n';
}
```

证明：

- 1 数学归纳法证明上述转移的可行性：
- 2 假设，规模更小的子问题的指标函数值是正确的，证明按照上述迁移方式进行迁移之后，依然得到正确的结果：
  $f_j = max(f_j, f_{j-w[i]} + v[i])$
- 3 $first$ 假设$(k_0 - 1) \times w_i <= j < k =_0 \times w_i$
- 4 $f_{j-w_i} = max(f_{j-k*w_i} + (k-1)*v_i), j-k*w_i >= 0, k >= 1,$
- 5将3代入2之后可以发现，有(前提的k满足题意，即使关键字大于等于0)
  $f_j = max(f_j, f_{j-w[i]} + v[i])$
  $= max(f_j, max(f_{j-k*w_i} + (k-1)*v_i) + v_i)$
  $= max(f_j, max(f_{j-k*w_i} + k*v_i))$
  $= max(f_j, f_{j-k*w_i+} + k*v_i)$
- 综上，结论得证。
- 初始化$f_{0.....m} = 0$显然正确。第一个状态是正确的。由上述转移方式也可推出子问题的正确的解。由数学归纳法。

问题类型转化：

- 要求此时得合法解是，必须将背包装满。
- 有如下改变，像01背包问题也是如此。
  - 第一步初始化的，不合法解都初始化成inf.用到该类位置的都是不合法的解，这些解在max运算终会被淘汰。若被使用相当于标记了不合法解。