[编程题]: 背包问题九讲总结

代码可以参考网站 https://www.acwing.com

1. 01背包问题

问题描述

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 vi，价值是 wi。
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。
输出最大价值。

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 vi,wi，用空格隔开，分别表示第 i 件物品的体积和价值。

输出格式

输出一个整数，表示最大价值。

数据范围

0<N,V≤1000
0<vi,wi≤1000

输入样例

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

输出样例

```
8
```

状态转移方程

> dp[i][j] 表示前面 i 个物品，背包容量为 j 时，可以得到的最大利益
>
> dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - vi] + ci)
>
> vi为第 i 个物品的体积，ci为第 i 个物品的价值
>
> 边界：dp[0][j] = 0

优化dp数组，只使用一维数组就可以

> i 的状态只取决于 i - 1 的状态，状态转移方程变为了
>
> dp[j] = max(dp[j], dp[j - vi] + ci)
>
> 遍历时要从右至左，否则 i - 1的状态会被覆盖掉

代码

C++

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, v;
    cin >> n >> v;

    vector<int> dp(v + 1, 0);

    for (int i = 1; i <= n; i++) {
        int v, c;
```

```
        cin >> v >> c;
        for (int j = V; j >= v; j--) { // 背包容量要能放得下当前物品
            dp[j] = max(dp[j], dp[j - v] + c);
        }
    }
    cout << dp[V] << endl;
    return 0;
}
```

2. 完全背包问题

问题描述

有 N 种物品和一个容量是 V 的背包，每种物品都有无限件可用。

第 i 种物品的体积是 vi，价值是 wi。

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。
输出最大价值。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。

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

输出格式

输出一个整数，表示最大价值。

数据范围

0<N,V≤1000
0<vi,wi≤1000

输入样例

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

输出样例:

```
10
```

状态转移方程

> dp[i][j]表示前 i 个物品，背包容量为 j 时，可以得到的最大价值
>
> dp[i][j] = max(dp[i - 1][j], dp[i][j - vi] + wi)
>
> 边界dp[0][j] = 0, dp[i][0] = 0

优化dp数组，只使用一维数组就可以

> dp[j]表示容量为 j 的背包可以得到的最大价值
>
> dp[j] = max(dp[j], dp[j - vi] + wi)
>
> 计算dp数组时从左至右计算

代码

C++

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int N, V;
    cin >> N >> V;
    vector<int> v(N + 1, 0);
    vector<int> w(N + 1, 0);
    vector<int> dp(V + 1, 0);
    for (int i = 1; i <= N; i++) {
        cin >> v[i] >> w[i];
    }
    dp[0] = 0;
    for (int i = 1; i <= N; i++) {
        for (int j = v[i]; j <= V; j++) {
            dp[j] = max(dp[j], dp[j - v[i]] + w[i]);
        }
    }
    cout << dp[V] << endl;
    return 0;
}
```

3. 多重背包问题

问题描述

有 N 种物品和一个容量是 V 的背包。

第 i 种物品最多有 si 件，每件体积是 vi，价值是 wi。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。
输出最大价值。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 vi,wi,si 用空格隔开，分别表示第 i 种物品的体积、价值和数量。

输出格式

输出一个整数，表示最大价值。

数据范围

0<N,V≤100
0<vi,wi,si≤100

输入样例

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

输出样例:

```
10
```

状态转移方程

> dp[i][j]表示前 i 个物品中，背包容量为 j 时，可以得到的最大价值
>
> dp[i][j] = max(dp[i - 1][j - k * vi] + k * wi) (0 <= k <= si) 其中第 i 个物品的体积为 vi，价值为 wi，第 i 个物品有si个)
>
> 边界 dp[0][j] = 0, dp[i][0] = 0

优化dp数组，使用一维的dp数组就可以了

> dp[j]表示容量为 j 的背包可以得到的最大价值
>
> dp[j] = max(dp[j - k * vi] + wi) (其中0 <= k <= si)

代码

C++

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int N, V;
    cin >> N >> V;
    vector<int> dp(V + 1, 0);
    int v, w, s;
    for (int i = 0; i < N; i++) {
        cin >> v >> w >> s;
        for (int j = V; j >= 0; j--) {
            for (int k = 1; k <= s && k * v <= j; k++) {
                dp[j] = max(dp[j], dp[j - k * v] + k * w);
            }
```

```
        }
    }
    cout << dp[V] << endl;
    return 0;
}
```

多重背包问题的优化 I ——二进制拆分优化

有 N 种物品和一个容量是 V 的背包。

第 i 种物品最多有 si 件，每件体积是 vi，价值是 wi。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。
输出最大价值。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 vi,wi,si 用空格隔开，分别表示第 i 种物品的体积、价值和数量。

输出格式

输出一个整数，表示最大价值。

数据范围

0<N≤1000
0<V≤2000
0<vi,wi,si≤2000

提示:

本题考查多重背包的二进制优化方法。

输入样例

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

输出样例:

```
10
```

> 将背包拆分，构成一个新的01背包问题，根据01背包的转移方程解决问题

代码

C++

```
#include <iostream>
```

```
#include <vector>
#include <algorithm>

using namespace std;

struct Good{
    int v, w;
};

int main() {
    int N, V;
    cin >> N >> V;
    vector<Good> goods;
    vector<int> dp(V + 1, 0);
    for (int i = 0; i < N; i++) {
        int v, w, s;
        cin >> v >> w >> s;
        for (int k = 1; k <= s; k *= 2) {
            s -= k;
            goods.push_back({k * v, k * w});
        }
        if (s > 0) goods.push_back({s * v, s * w});
    }
    for (auto& good: goods) {
        for (int j = V; j >= good.v; j--) {
            dp[j] = max(dp[j], dp[j - good.v] + good.w);
        }
    }
    cout << dp[V] << endl;
    return 0;
}
```

多重背包问题的优化II——单调队列优化

有 N 种物品和一个容量是 V 的背包。

第 i 种物品最多有 si 件，每件体积是 vi，价值是 wi。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。
输出最大价值。

输入格式

第一行两个整数，N，V (0<N≤1000, 0<V≤20000)，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 vi,wi,si，用空格隔开，分别表示第 i 种物品的体积、价值和数量。

输出格式

输出一个整数，表示最大价值。

数据范围

0<N≤1000
0<V≤20000
0<vi,wi,si≤20000

本题考查多重背包的单调队列优化方法。

输入样例

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

输出样例：

```
10
```

代码

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cstring>

using namespace std;

const int N = 20010;
int n, m;
int f[N], g[N], q[N];

int main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        int c, w, s;
        cin >> c >> w >> s;
        memcpy(g, f, sizeof f);
        for (int j = 0; j < c; j++) {
            int hh = 0, tt = -1;
            for (int k = j; k <= m; k += c) {
                f[k] = g[k];
                if (hh <= tt && k - s * c > q[hh]) hh++;
                if (hh <= tt) f[k] = max(f[k], g[q[hh]] + (k - q[hh]) / c * w);
                while (hh <= tt && g[q[tt]] - (q[tt] - j) / c * w <= g[k] - (k -
j) / c * w) tt--;
                q[++tt] = k;
            }
        }
    }
    cout << f[m] << endl;
    return 0;
}
```

4. 混合三种背包问题

问题描述

有 N 种物品和一个容量是 V 的背包。

物品一共有三类：

- 第一类物品只能用1次（01背包）；
- 第二类物品可以用无限次（完全背包）；
- 第三类物品最多只能用 si 次（多重背包）；

每种体积是 vi，价值是 wi。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。
输出最大价值。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 vi,wi,si 用空格隔开，分别表示第 i 种物品的体积、价值和数量。

- si=−1 表示第 i 种物品只能用1次；
- si=0 表示第 i 种物品可以用无限次；
- si>0 表示第 i 种物品可以使用 si 次；

输出格式

输出一个整数，表示最大价值。

数据范围

$0<N,V\le 1000$
$0<vi,wi\le 1000$
$-1\le si\le 1000$

输入样例

```
4 5
1 2 -1
2 4 1
3 4 0
4 5 2
```

输出样例：

```
8
```

状态转移方程

> 将多重背包问题拆分为01背包问题，这样原问题就变成为01背包问题和完全背包问题
>
> 根据不同的背包种类，使用不同的转移方程
>
> dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - vi] + wi)（01背包问题时采用此转移方程)
>
> dp[i][j] = max(dp[i - 1][j], dp[i][j - vi] + wi) （完全背包问题时采用此背包问题）

优化dp数组，只使用一维数组就可以

> dp[j] = max(dp[j], dp[j - vi] + wi)（01背包问题时采用此转移方程)

> dp[j] = max(dp[j - k * vi] + wi)（完全背包问题时采用此背包问题）

代码

C++

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

const int N = 1010;

struct Node{
    int kind;
    int c, w;
};

int n, m;
vector<Node> nodes;
int f[N];

int main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        int c, w, s;
        cin >> c >> w >> s;
        if (s == -1) nodes.push_back({-1, c, w});
        else if (s == 0) nodes.push_back({0, c, w});
        else {
            for (int k = 1; k <= s; k <<= 1) {
                s -= k;
                nodes.push_back({-1, c * k, w * k});
            }
            if (s > 0) nodes.push_back({-1, c * s, w * s});
        }
    }
    for (auto& node: nodes) {
        if (node.kind == -1) {
            for (int i = m; i >= node.c; i--) {
                f[i] = max(f[i], f[i - node.c] + node.w);
            }
        }
        if (node.kind == 0) {
            for (int i = node.c; i <= m; i++) {
                f[i] = max(f[i], f[i - node.c] + node.w);
            }
        }
    }
    cout << f[m] << endl;
    return 0;
}
```

5. 二维费用的背包问题

问题描述

有 N 件物品和一个容量是 V 的背包，背包能承受的最大重量是 M。

每件物品只能用一次。体积是 vi，重量是 mi，价值是 wi。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，总重量不超过背包可承受的最大重量，且价值总和最大。
输出最大价值。

输入格式

第一行两个整数，N, M，用空格隔开，分别表示物品件数、背包容积和背包可承受的最大重量。

接下来有 N 行，每行三个整数 vi,mi,wi 用空格隔开，分别表示第 i 件物品的体积、重量和价值。

输出格式

输出一个整数，表示最大价值。

数据范围

0<N≤1000
0<V,M≤100
0<vi,mi≤100
0<wi≤1000

输入样例

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

输出样例:

```
8
```

状态转移方程

> dp[i][j][k]表示前 i 个物品中，容量为 j 的背包，最多可以装质量为 k 的物品时，可以得到的最大价值
>
> dp[i][j][k] = max(dp[i - 1][j][k], dp[i - 1][j - vi][k - mi] + wi)（vi为第 i 个物品的体积，mi为第 i 个物品的质量，wi为第 i 个物品的质量）
>
> 边界 dp[0][j][k] = dp[i][0][k] = dp[i][j][0] = 0;

简化dp数组，使用二维数组解决问题

> dp[j][k] = max(dp[j][k], dp[j - vi][k - mi] + wi)

代码

C++

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 110;

int n, v, m;
int f[N][N];

int main() {
    cin >> n >> v >> m;
    for (int i = 0; i < n; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        for (int j = v; j >= a; j--) {
            for (int k = m; k >= b; k--) {
                f[j][k] = max(f[j][k], f[j - a][k - b] + c);
            }
        }
    }

    cout << f[v][m] << endl;
    return 0;
}
```

6. 分组的背包问题

问题描述

有 NN 组物品和一个容量是 VV 的背包。

每组物品有若干个，同一组内的物品最多只能选一个。
每件物品的体积是 vijvij，价值是 wijwij，其中 ii 是组号，jj 是组内编号。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

输出最大价值。

输入格式

第一行有两个整数 N，V，用空格隔开，分别表示物品组数和背包容量。

接下来有 NN 组数据：

- 每组数据第一行有一个整数 Si，表示第 i 个物品组的物品数量；
- 每组数据接下来有 Si 行，每行有两个整数 vij,wij，用空格隔开，分别表示第 i 个物品组的第 j 个物品的体积和价值；

输出格式

输出一个整数，表示最大价值。

数据范围

$0<N,V\le 1000$
$0<S_i\le 1000$
$0<v_{ij},w_{ij}\le 1000$

输入样例

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

输出样例：

```
8
```

状态转移方程

> dp[i][j]表示前 i 组物品中，背包容量为 j 时可以得到的最大价值
>
> dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - vk] + wk)（vk为第 i 组物品中第 k 个的体积，wk为第 i 组中第 k 个物品的价值。即对于第 i 组的物品，一共有 s + 1种选：不选、选第1个，选第2个，..., 选第s个）
>
> 边界dp[0][j] = 0, dp[i][0] = 0

简化dp数组，抵用一维数组解决问题

> dp[j] = max(dp[j], dp[j - vk] + wk)

代码

C++

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 110;

int n, m;
int f[N], v[N], w[N];

int main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        int s;
        cin >> s;
        for (int j = 0; j < s; j++) cin >> v[j] >> w[j];
        for (int j = m; j >= 0; j--) {
            for (int k = 0; k < s; k++) {
                if (j >= v[k]) {
                    f[j] = max(f[j], f[j - v[k]] + w[k]);
                }
```

```
            }
        }
    }
    cout << f[m] << endl;
    return 0;
}
```

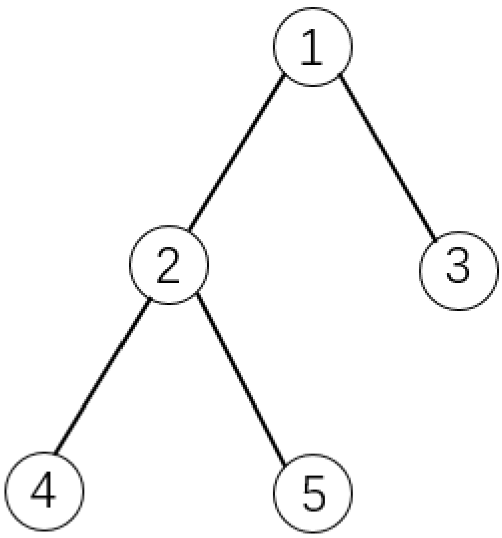
7. 有依赖的背包问题

问题描述

有 N 个物品和一个容量是 V 的背包。

物品之间具有依赖关系，且依赖关系组成一棵树的形状。如果选择一个物品，则必须选择它的父节点。

如下图所示：



如果选择物品5，则必须选择物品1和2。这是因为2是5的父节点，1是2的父节点。

每件物品的编号是 i，体积是 vi，价值是 wi，依赖的父节点编号是 pi。物品的下标范围是 1…N。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

输出最大价值。

输入格式

第一行有两个整数 N，V，用空格隔开，分别表示物品个数和背包容量。

接下来有 N 行数据，每行数据表示一个物品。
第 i 行有三个整数 vi,wi,pi 用空格隔开，分别表示物品的体积、价值和依赖的物品编号。
如果 pi=−1，表示根节点。 **数据保证所有物品构成一棵树。**

输出格式

输出一个整数，表示最大价值。

数据范围

1≤N,V≤100
1≤vi,wi≤100

父节点编号范围：

- 内部结点：1≤pi≤N
- 根节点 pi=−1

输入样例

```
5 7
2 3 -1
2 2 1
3 5 1
4 7 2
3 6 2
```

输出样例：

```
11
```

状态转移方程

树形动态规划 Dynamic Programming

8. 背包问题求方案数

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 vi，价值是 wi。

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 **最优选法的方案数**。注意答案可能很大，请输出答案模 10^9 + 7 的结果。

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 vi, wi，用空格隔开，分别表示第 i 件物品的体积和价值。

输出格式

输出一个整数，表示 **方案数** 模 10^9 + 7 的结果。

数据范围

0<N,V≤1000
0<vi,wi≤1000

输入样例

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

输出样例：

```
2
```

状态转移方程

代码

9. 背包问题求具体方案

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 vi，价值是 wi。

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…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 vi,wi 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…N。

数据范围

$0<N,V\le 1000$
$0<v_i,w_i\le 1000$

输入样例

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

输出样例:

```
1 4
```

状态转移方程

代码