

# 动态规划 背包

```
#include <iostream>
#include <vector>
#include <string>
#include <cstring>
#include <set>
#include <map>
#include <queue>
#include <ctime>
#include <random>
#include <sstream>
#include <numeric>
#include <stack>
#include <stdio.h>
#include <algorithm>
using namespace std;

#define Multiple_groups_of_examples
#define rep(i,x,n) for(int i = x; i <= n; i++)
#define vf first
#define vs second

typedef long long LL;
typedef pair<int,int> PII;

const int INF = 0x3f3f3f3f;
const int N = 1e2 + 21;

namespace golitter {
namespace bag {

    // knapsack problem
    // 具有背包性质的:
    // 选    or    不选

/**
 *       $F(i, j)$  表示前  $i$  件物品恰放入一个容量为  $v$  的背包可以获得的最大价值。
 * 状态转移方程:
 *       $F(i, j) = \max( F(i-1, j) , F(i-1, j - C_i) + W_i )$ 
 *
 * 背包问题两种不同的问法:
 *      1. 恰好装满背包
 *          初始化时, 除了  $F(0) = 0$ , 其他  $F(j)$  均设置为  $-INF$ 
 *      2. 未强调装满背包
 *          均设置为0
 */
// 01背包
// 朴素版
void zero1_ez() {
    int n,m; cin>>n>>m;
```

```

vector<PII> vp(n+1);
// for(auto &t: vp) cin>>t.vf>>t.vs; // weight value
for(int i = 1; i <= n; ++i) cin>>vp[i].vf>>vp[i].vs;
vector<vector<int>> f(n+1, vector<int>(m+1));
for(int i = 1; i <= n; ++i) {
    for(int j = 0; j <= m; ++j) {
        if(j < vp[i].vf) { // 如果不满足, 需要将之前的最优价值赋值给当前
            f[i][j] = f[i-1][j];
        } else f[i][j] = max(f[i-1][j], f[i-1][j - vp[i].vf] + vp[i].vs);
    }
}
cout<<f[n][m];
}

// 优化版
// 后面只会用到weight后面的dp函数, 所以可以用一维来回滚
void zero01_hd() {
    int n,m; cin>>n>>m;
    vector<PII> vp(n+1);
    vector<int> f(m+1);
    for(int i = 1; i <= n; ++i) cin>>vp[i].vf>>vp[i].vs;
    for(int i = 1; i <= n; ++i) {
        for(int j = m; j >= vp[i].vf; --j) {
            f[j] = max(f[j], f[j - vp[i].vf] + vp[i].vs);
        }
    }
    cout<<f[m];
}

namespace other_01_bag{

void NC17871() {
    // 链接: https://ac.nowcoder.com/acm/problem/17871
    // CSL手上有n个苹果, 第i个苹果的质量是wi, 现在他想把这些苹果分给他的好朋友wavator和
    tokitsukaze。
    // 但是CSL为了不让他们打架, 根据质量决定尽量地均分成两堆分给他们。现在CSL想知道到底给每个人分多
    少质量的苹果。
    int n; cin>>n;
    vector<int> a(n + 1);
    int sum = 0;
    for(int i = 1; i <= n; ++i) cin>>a[i], sum += a[i];
    vector<int> f(sum + 1);
    for(int i = 1; i <= n; ++i) {
        for(int j = sum; j >= a[i]; --j) {
            f[j] = max(f[j], f[j - a[i]] + a[i]);
        }
    }
    int mi = min(f[(sum + 1)/2], sum - f[(sum + 1)/2]);
    cout<<mi<<" "<<sum - mi;
}

void P1877() { // 选与不选 --> 减 与 加
    // https://www.luogu.com.cn/problem/P1877
    int n,st,ma; cin>>n>>st>>ma;
    vector<int> a(n+1);
    for(int i = 1; i <= n; ++i) cin>>a[i];
    vector<vector<int>> f(n + 1, vector<int>(ma + 1));
    f[0][st] = 1;
    for(int i = 1; i <= n; ++i) {
        for(int j = 0; j <= ma; ++j) {

```

```

        if(j - a[i] >= 0) f[i][j] = f[i][j] || f[i-1][j - a[i]];
        if(j + a[i] <= ma) f[i][j] = f[i][j] || f[i-1][j + a[i]];
    }
}
for(int i = ma; i >= 0; --i) {
    if(f[n][i]) {
        cout<<i; return ;
    }
}
cout<<-1;
}
}
// 完全背包
// 朴素版
void entire_ez() {
    int n,m; cin>>n>>m;
    vector<PII> vp(n+1);
    vector<vector<int>> f(n+1, vector<int>(m+1));
    for(int i = 1; i <= n; ++i) cin>>vp[i].vf>>vp[i].vs;
    for(int i = 1; i <= n; ++i) {
        int k = m / vp[i].vf;
        for(int z = 0; z <= k; ++z) {

        }
    }
}
// 优化版
void entire_hd() {
    int n,m; cin>>n>>m;
    vector<PII> vp(n+1);
    vector<int> f(m+1);
    for(int i = 1; i <= n; ++i) cin>>vp[i].vf>>vp[i].vs;
    for(int i = 1; i <= n; ++i) { // 转为01背包问题
        for(int j = vp[i].vf; j <= m; ++j) {
            f[j] = max(f[j], f[j - vp[i].vf] + vp[i].vs);
        }
    }
    cout<<f[m];
}
// 多重背包
void multi_() {
    int n,m; cin>>n>>m;
    vector<PII> vp;
    for(int i = 0; i < n; ++i) {
        int v,w,s; cin>>v>>w>>s;
        int k = 1;
        while(k <= s) { // 二进制优化
            int tv = v * k;
            int tw = w * k;
            s -= k;
            k <<= 1;
            vp.push_back({tv,tw});
        }
        if(s > 0) {
            int tv = v * s;
            int tw = w * s;
            vp.push_back({tv,tw});
        }
    }
}

```

```

    }
    n = vp.size();
    vector<int> f(m+1);
    for(int i = 0; i < n; ++i) { // 转为01背包问题
        for(int j = m; j >= vp[i].vf; --j) {
            f[j] = max(f[j], f[j - vp[i].vf] + vp[i].vs);
        }
    }
    cout<<f[m];
}

// 二维费用的背包问题
//      对于每件物品，具有两种不同的费用；选择这件物品必须同时付出这两种代价；对于每种代价都有一个可付出的最大值（背包容量）。
// 状态转移方程：  $F(i, v, u) = \max(F(i-1, v, u), F(i-1, v-v[i], u-u[i]) + w[i])$ 

void costbag2d() {
    int n,m,v; cin>>n>>v>>m;
    vector<array<int,3>> va(n); // vol weight value
    for(auto &t: va) cin>>t[0]>>t[1]>>t[2];
    vector<vector<int>> f(v+1, vector<int>(m+1));
    for(auto t: va) { // 转为01背包问题
        for(int i = v; i >= t[0]; --i) {
            for(int j = m; j >= t[1]; --j) {
                f[i][j] = max(f[i][j], f[i - t[0]][j - t[1]] + t[2]);
            }
        }
    }
    cout<<f[v][m];
}

// 分组背包
//      n件物品，一个容量为m的背包。这些物品被分为若干个组，每组中的物品相互冲突，最多选一件。
//      01背包

void groupbag() {
    int n,m; cin>>n>>m;
    vector<vector<PII>> vvp; // 分分组
    for(int i = 0; i < n; ++i) {
        vector<PII> tmp;
        int cnt; cin>>cnt;
        for(int j = 0; j < cnt; ++j) {
            int los,val; cin>>los>>val;
            tmp.push_back({los,val});
        }
        vvp.push_back(tmp);
    }
    vector<int> f(m+1);
    for(auto tt: vvp) { // 组
        for(int i = m; i >= 0; --i) { // 背包容量在组物品遍历的上一个for循环中
            for(auto t: tt) {
                if(i >= t.vf) {
                    f[i] = max(f[i], f[i - t.vf] + t.vs);
                }
            }
        }
    }
    cout<<f[m];
}

```

}}