## **多重背包问题优化**

有n种物品和一个容量为W的背包。第i种物品有mi个，每件重量为wi，价值为vi，求从这n种物品中挑选重量总和不超过W的物品的最大价值。

一般解法

for(int i = 1; i < n; i++){//枚举物品

for(int y = 1; y < C+1; y++){//枚举包的容量

if(y < W[i])//一件也装不了

{

f[i][y] = f[i-1][y];

} else {//可以取

int count = min(N[i], y/W[i]);//最多取多少件

f[i][y] = f[i-1][y];//不取

for(int k = 1; k <= count; k++) {//枚举取多少件

int temp = f[i-1][y-W[i]\*k] + k\*V[i];

if(temp >= f[i][y])

f[i][y] = temp;

}

}

}

}

### **转化成0-1背包问题**

多重背包问题，最简单的解法，就是转化成0-1背包问题。第i个物品有mi个，等价于有mi个相同的物品。但直接拆分成mi件物品并不是最好的方法。我们可以利用二进制来拆分。例如m1=13=20+21+22+6,我们将第一种物品共13件，拆分成20,21,22,6这四件，13以内的任何数字都可以通过这四种数字组合而成。



下面给出一个二进制拆分的多重背包模板：

const int N = 100, W = 100000;int cost[N], weight[N], number[N];int dp[W + 1];

int knapsack(int n, int w){

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

int num = min(number[i], w / weight[i]);

for (int k = 1; num > 0; k\*=2){

if (k > num) k = num;

num -= k;

for (int j = w; j >= weight[i] \* k; --j)

dp[j] = max(dp[j], dp[j - weight[i] \* k] + cost[i] \* k);

}

}

return dp[w];

}

时间复杂度为O(nlogM×W),实际应用已经足够好了。

# 单调队列优化多重背包模板(hdu 2191)

一般的多重背包时间复杂度

O(V∑N),二进制优化后O(V∑logN)，但有的题必须O(N\*V)才能过，这就必须用单调队列优化了

设物品i体积v[i]，价值w[i]，个数n[i]

多重背包dp方程：dp[i][j]=max{dp[i−1][j−k\*v[i]]+k\*w[i]} (0≤k≤min(n[i],j/v[i]))

设a=j/v[i]，b=j%v[i]，即j=a\*v[i]+b

在每一轮循环中i相同，即v、w、n都相同

dp方程变形为：dp[i][a\*v+b]=max{dp[i−1][(a−k)\*v+b]+k\*w} (0≤k≤min(n,a))

设s=a−k，那么a−min(n,a)≤s≤min(n,a)

代入dp方程得：

dp[i][a\*v+b]=max{dp[i−1][s\*v+b]−s\*w}+a\*w (a−min(n,a)≤s≤min(n,a))



令f[x]=dp[i][x\*v+b]，g[x]=dp[i−1][x\*v+b]−x\*w，h[x]=x\*w，那么上式即为：

f[a]=max{g[s]}+h[a] (a−min(n,a)≤s≤min(n,a))

g[s]即可实现O(1)转移，要注意满足s的范围

什么是单调队列：[http://blog.csdn.net/justmeh/article/details/5844650](http://blog.csdn.net/justmeh/article/details/5844650" \t "https://blog.csdn.net/yoer77/article/details/_blank)

用单调队列优化：[http://blog.csdn.net/flyinghearts/article/details/5898183](http://blog.csdn.net/flyinghearts/article/details/5898183" \t "https://blog.csdn.net/yoer77/article/details/_blank)

具体实现见代码

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

const int maxn=110;

int C,N,M,dp[maxn];

int Q[maxn][2],head,tail;*//Q[x][0]存g[s]，Q[x][1]存s以判断s是否符合限制条件*

int main(){

scanf("%d",&C);

while(C--){

memset(dp,0,sizeof(dp));

scanf("%d%d",&N,&M);

for(int i=1;i<=M;i++){

int v,w,n;

scanf("%d%d%d",&v,&w,&n);

for(int b=0;b<v;b++){*//枚举余数b*

head=tail=1;*//清空队列，加在这里是因为只能从余数相同的状态转移来，之前队列中的元素都不需要了*

for(int j=0;j<=(N-b)/v;j++){

int tmp=dp[j\*v+b]-j\*w;*//这里的dp是上一轮的，j代表s*

while(head<tail&&tmp>=Q[tail-1][0]) --tail;*//先插入以满足s≤min(n,a)的条件*

Q[tail][0]=tmp,Q[tail++][1]=j;*//这里j还是代表s*

while(Q[head][1]<j-min(n,j)) ++head;*//以下j代表a，删除队首以去除不满足a−min(n,a)≤s的状态*

dp[j\*v+b]=Q[head][0]+j\*w;*//队首即为最优*

}

}

}

printf("%d\n",dp[N]);

}

return 0;

}

优秀的背包问题模板：[http://blog.csdn.net/libin56842/article/details/9396649](http://blog.csdn.net/libin56842/article/details/9396649" \t "https://blog.csdn.net/yoer77/article/details/_blank)；

//单调队列优化，代码2

#include <iostream>#include <deque>#include <algorithm>

using namespace std;

struct Pack

{

int sum, cost;

Pack(int s, int c) : sum (s), cost(c) {}

};

const int Maxv = 1001;deque <Pack> Q;int N, V, F[Maxv];

int main()

{

cin >> N >> V;

for (int i = 1, p, w, c; i <= N; i ++)

{

cin >> p >> w >> c; p = min(p, V / w);

for (int j = 0; j < w; j ++)

{

Q.clear();

for (int k = 0; k <= (V - j) / w; k ++)

{

int y = F[k \* w + j] - k \* c;

while (Q.size() && Q.back().cost <= y) Q.pop\_back();

Q.push\_back(Pack(k, y));

if (Q.front().sum < k - c) Q.pop\_front();

F[k \* w + j] = Q.front().cost + k \* c;

}

}

}

cout << F[V] << endl;

return 0;