

本周作业

1、混合背包：P2623

2、分组背包：P1757

上周作业答案

练习 1:

有n个正整数，找出其中和为t(t也是正整数)的可能的组合方式。如：
n=5,5个数分别为1,2,3,4,5，t=5；
那么可能的组合有5=1+4和5=2+3和5=5三种组合方式。
输入
输入的第一行是两个正整数n和t，用空格隔开，其中1≤n≤20,表示正整数的个数，t为要求的和(1≤t≤1000)
接下来的一行是n个正整数，用空格隔开。
输出
和为t的不同的组合方式的数目。
样例输入
5 5
1 2 3 4 5
样例输出
3

代码:

```
5  #include <iostream>
6  using namespace std;
7  int n, t;
8  int num[30], dp[1010];
9  int main(){
10     cin >> n >> t;
11     for(int i = 1; i <= n; i++)
12         cin >> num[i];
13     dp[0] = 1;
14     for(int i = 1; i <= n; i++){
15         for(int j = t; j >= num[i]; j--){
16             dp[j] = dp[j] + dp[j - num[i]];
17         }
18     }
19     cout << dp[t];
20     return 0;
21 }
```

练习 2:

字符串编辑距离:

- Levenshtein距离是一种计算两个字符串间的差异程度的字符串度量 (string metric) 。
- Levenshtein距离就是从**一个字符串修改到另一个字符串时**，其中编辑单个字符（比如**修改、插入、删除**）所需要的最少次数。
- 俄罗斯科学家Vladimir Levenshtein于1965年提出了这一概念。
- 例如：由串“NOTV”修改为“LOVER”需要4次字符编辑操作
- 解释：NOTV -> LOTV -> LOV -> LOVE -> LOVER

代码:

```
16 白   for(int i = 1; i <= _s1.size(); i++) { //初始化
17     _dp[i][0] = i;
18 }
19 白   for(int j = 1; j <= _s2.size(); j++) { //初始化
20     _dp[0][j] = j;
21 }
22 白   for(int i = 1; i <= _s1.size(); i++) {
23     for(int j = 1; j <= _s2.size(); j++) {
24         if(_s1[i - 1] == _s2[j - 1]) { //字符串, 下标-1
25             _dp[i][j] = _dp[i - 1][j - 1];
26         }
27         else {
28             _dp[i][j] = min( min(_dp[i][j - 1], _dp[i - 1][j]) , _dp[i - 1][j - 1] ) + 1;
29         }
30     }
31 }
32 cout << _dp[_s1.size()][_s2.size()];
33 }
```

本周课堂内容

1、多重背包问题

有 N 种物品和一个容量为 V 的背包。第 i 种物品最多有 $p[i]$ 件可用，每件费用是 $w[i]$ ，价值是 $v[i]$ 。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

状态转移方程: $f[i][j] = \max(f[i-1][j-k*w[i]] + k*v[i], f[i-1][j]) \quad 0 \leq k \leq p[i]$

转一维: $f[j] = \max(f[j-k*w[i]] + k*v[i], f[j])$ 复杂度是 $O(V * \sum p[i])$

例题:

为了庆贺班级在校运动会上取得全校第一名成绩，班主任决定开一场庆功会，为此拨款购买奖品犒劳运动员。期望拨款金额能购买最大价值的奖品，可以补充他们的精力和体力。

输入格式

第一行二个数 $n(n \leq 500)$, $m(m \leq 6000)$ ，其中 n 代表希望购买的奖品的种数， m 表示拨款金额。接下来 n 行，每行 3 个数， v 、 w 、 s ，分别表示第 i 种奖品的价格、价值（价格与价值是不同的概念）和购买的数量（买 0 件到 s 件均可），其中 $v \leq 100$, $w \leq 1000$, $s \leq 10$ 。

输出格式

第一行：一个数，表示此次购买能获得的最大的价值（注意！不是价格）。

```
1  #include<iostream>
2  using namespace std;
3  int V,N;
4  int w[10010], c[10010], p[10010];
5  long long dp[10000010];
6  int main()
7  {
8      cin >> N >> V;
9      for(int i = 1; i <= N; i++){
10         cin >> w[i] >> c[i] >> p[i];
11     }
12     for (int i = 1; i <= N; i++) {
13         for (int j = V; j >= 1; j--) {
14             for (int k = 1; k <= p[i]; k++){
15                 if( j >= k * w[i] )
16                     dp[j] = max(dp[j], dp[j - k * w[i]] + k * c[i]);
17             }
18         }
19     }
20     cout << dp[V];
21     return 0;
22 }
```

多重背包二进制优化

多重背包中，假设某件物品最多选择 12 件，则拆分数为：

1 2 4 5

转为 4 件 01 背包的物品为：

(w_i, v_i) , $(2w_i, 2v_i)$, $(4w_i, 4v_i)$, $(5w_i, 5v_i)$ 。

```
5  int _n, _m; // 物品数量, 背包总承重
6  int _w[110]; // 重量
7  int _v[110]; // 价值
8  int _dp[110];
9  int main () {
10     cin >> _n >> _m;
11
12     int w = 0, v = 0, s = 0;
13     for(int i = 1; i <= _n; i++) {
14         cin >> w >> v >> s;
15         for(int j = 1; j <= s; j <= 1) {
16             _num++;
17             _w[_num] = j * w;
18             _v[_num] = j * v;
19             s -= j;
20         }
21         if(s > 0) {
22             _num++;
23             _w[_num] = s * w;
24             _v[_num] = s * v;
25         }
26     }
27
28     // for(int i = 1; i <= _num; i++) {
29     //     cout << _w[i] << " " << _v[i] << endl;
30     // }
31     // 01 背包解
32     for(int i = 1; i <= _num; i++) {
33         for(int j = _m; j >= _w[i]; j--) {
34             _dp[j] = max(_dp[j - _w[i]] + _v[i], _dp[j]);
35         }
36     }
37
38     cout << _dp[_m];
39
40     return 0;
41 }
```

2、混合背包：

混合了 01、完全、多重背包的问题

有 N 种物品和一个容量是 V 的背包。物品一共有三类：

- (1) 第一类物品只能用 1 次；
- (2) 第二类物品可以用无限次；
- (3) 第三类物品最多只能用 s_i 次；

每种体积是 w_i ，价值是 c_i 。

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。

```

1  #include <iostream>
2  using namespace std;
3  int N,V;
4  int w[100], c[100], s[100], dp[100];
5  int main(){
6      cin >> N >> V;
7      for(int i = 1; i <= N; i++){
8          cin >> w[i] >> c[i] >> s[i];
9      }
10     for(int i = 1; i <= N; i++){
11         if(s[i] == -1){ //01 背包
12             for(int j = V; j >= w[i]; j--){
13                 dp[j] = max(dp[j], dp[j - w[i]] + c[i]);
14             }
15         }
16         else if(s[i] == 0){ //完全背包
17             for(int j = w[i]; j <= V; j++){
18                 dp[j] = max(dp[j], dp[j - w[i]] + c[i]);
19             }
20         }
21         else { //多重背包
22             for(int j = V; j >= 1; j--){
23                 for(int k = 1; k <= s[i]; k++){
24                     if(j >= k * w[i])
25                         dp[j] = max(dp[j], dp[j - k * w[i]] + k * c[i]);
26                 }
27             }
28         }
29     }
30     cout << dp[V];
31     return 0;
32 }

```

优化：01 可以看成是多重的简化版

```

7  for(int i = 1; i <= _n; i++) {
8      if(s[i] == 0) { //完全
9          for(int j = _w[i]; j <= _m; j++) {
10              _dp[j] = max(_dp[j], _dp[j - _w[i]] + _c[i]);
11          }
12      }
13      else { //多重 + 01
14          for(int j = 1; j <= _s[i]; j++) {
15              for(int k = _m; k >= _w[i]; k--) {
16                  _dp[k] = max(_dp[k], _dp[k - _w[i]] + _c[i]);
17              }
18          }
19      }
20  }

```

3、分组背包

给定 N 组物品，其中第 i 组有 C_i 个物品。第 i 组第 j 个物品的体积为 W_{ij} ，价值为 V_{ij} 。将若干个物品放入容量为 M 的背包，每组至多选择一个物品，在使物体总体积不超过 M 的前提下，物品的总价值和最大，求出此时的最大价值。

①有 N 组，第 i 组有 C_i 个物品：→需要用二维数组来存储物品的价值 $v[i][j]$ 和体积 $w[i][j]$ ， i 代表组号， j 代表第 j 个物品
因此输入过程如下：

```

cin >> N >> M;
for(int i = 1; i <= N; i++){
    cin >> c[i];
    for(int j = 1; j <= c[i]; j++){
        cin >> w[i][j] >> v[i][j];
    }
}

```

```

1  #include <iostream>
2  using namespace std;
3  int N, M, num;
4  int w[100][100], v[100][100], c[100], dp[100];
5  int main(){
6      cin >> N >> M;
7      for(int i = 1; i <= N; i++){
8          cin >> c[i];
9          for(int j = 1; j <= c[i]; j++){
10             cin >> w[i][j] >> v[i][j];
11         }
12     }
13     for(int i = 1; i <= N; i++){
14         for(int j = M; j >= 1; j--){
15             for(int k = 1; k <= c[i]; k++){
16                 if( j >= w[i][k] )
17                     dp[j] = max(dp[j], dp[j - w[i][k]] + v[i][k]);
18             }
19         }
20     }
21     cout << dp[M];
22     return 0;

```

转一维：进行空间优化，时间复杂度不变

```

for(int i = 1; i <= _n; i++) {
    cin >> _s;
    memset(_w, 0, sizeof(_w));
    memset(_v, 0, sizeof(_v));
    for(int j = 1; j <= _s; j++) {
        cin >> _w[j] >> _v[j];
    }
    for(int j = _m; j >= 1; j--) { // 体积
        for(int k = 0; k <= _s; k++) { // 组内选择
            if(j >= _w[k]) {
                _dp[j] = max(_dp[j], _dp[j - _w[k]] + _v[k]);
            }
        }
    }
}

```