

## 背包问题：枚举法和 dp 方法时间对比

### 一、 代码：

```
1. #include <bits/stdc++.h>
2. #define MAXN 1050
3. #define MAXW 1050
4. #define ll long long
5. #define pii pair<int, int>
6. using namespace std;
7.
8. ll dp[MAXN][MAXW];
9. pii item[MAXN];
10. int n_values[100];
11. int w_values[100];
12.
13. ll knapsack_bruteforce(int n, int w) {
14.     ll ans = 0, one = 1;
15.     for (ll i = 0; i < (one << n); i++) {
16.         ll current_weight = 0, current_value = 0;
17.         for (int j = 0; j < n; j++) {
18.             if (i & (one << j)) {
19.                 current_weight += item[j].first;
20.                 current_value += item[j].second;
21.             }
22.         }
23.         if (current_weight <= w) {
24.             ans = max(ans, current_value);
25.         }
26.     }
27.     return ans;
28. }
29.
30. ll knapsack_dp(int n, int w) {
31.     for (int i = n - 1; i >= 0; i--) {
32.         for (int j = 0; j <= w; j++) {
33.             int wei = item[i].first, val = item[i].second;
34.             if (j < wei) {
35.                 dp[i][j] = dp[i + 1][j];
36.             } else {
37.                 dp[i][j] = max(dp[i + 1][j], dp[i + 1][j - wei] + val);
38.             }
39.         }
```

```

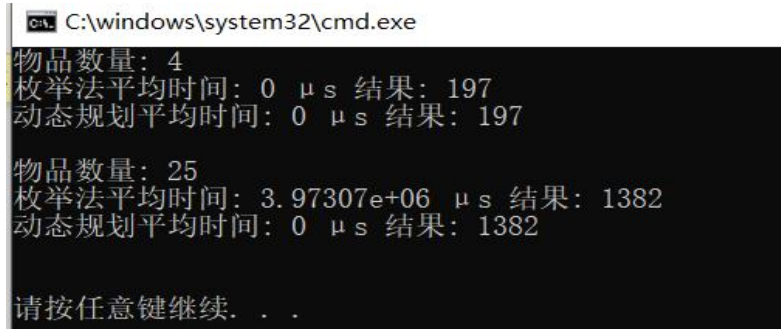
40.     }
41.     return dp[0][w];
42. }
43.
44. void generate_items(int n) {
45.     srand(time(0));
46.     for (int i = 0; i < n; ++i) {
47.         item[i].first = rand() % 100 + 1;
48.         item[i].second = rand() % 100 + 1;
49.     }
50. }
51.
52. int main() {
53.     int lens = 2;
54.     int test_runs = 5;
55.     n_values[0] = 4; w_values[0] = 200;
56.     n_values[1] = 25; w_values[1] = 1000;
57.     ll ans1 = 0, ans2 = 0;
58.     for (int k = 0; k < lens; k++) {
59.         int n = n_values[k], w = w_values[k];
60.         generate_items(n);
61.         auto start_bruteforce = chrono::high_resolution_clock::now();
62.
63.         for (int i = 0; i < test_runs; i++) {
64.             ans1 = knapsack_bruteforce(n, w);
65.         }
66.         auto end_bruteforce = chrono::high_resolution_clock::now();
67.         auto duration_bruteforce = chrono::duration_cast<chrono::microseconds>(end_bruteforce - start_bruteforce);
68.
69.         auto start_dp = chrono::high_resolution_clock::now();
70.         for (int i = 0; i < test_runs; i++) {
71.             ans2 = knapsack_dp(n, w);
72.         }
73.         auto end_dp = chrono::high_resolution_clock::now();
74.         auto duration_dp = chrono::duration_cast<chrono::microseconds>(end_dp - start_dp);
75.
76.         cout << "物品数量: " << n << "\n";
77.         cout << "枚举法平均时
间: " << duration_bruteforce.count() / (double)test_runs << " μs ";
78.         cout << "结果: " << ans1 << "\n";
79.         cout << "动态规划平均时
间: " << duration_dp.count() / (double)test_runs << " μs ";

```

```
79.         cout << "结果: " << ans2 << "\n" << "\n";  
80.     }  
81.     return 0;  
82. }
```

## 二、 分析

- 1、分别对物品数量为 4 和 25 的情况进行测试。
  - 2、对于每组物品数量，进行 5 次运行，累计运行时间，并计算平均值以减小时间波动对结果的影响。
  - 3、生成随机测试数据，包括物品的体积和价值。
- 运行代码，时间结果如下：



```
C:\windows\system32\cmd.exe  
物品数量: 4  
枚举法平均时间: 0 μs 结果: 197  
动态规划平均时间: 0 μs 结果: 197  
  
物品数量: 25  
枚举法平均时间: 3.97307e+06 μs 结果: 1382  
动态规划平均时间: 0 μs 结果: 1382  
  
请按任意键继续...
```

- 1、在物品数量较少的情况下（如物品数量为 4），枚举法和动态规划法在运行时间上相差不大。这是因为物品数量较少时，枚举法的计算量不大，所以运行时间较短。
- 2、当物品数量增加（如物品数量为 25）时，枚举法的运行时间显著增加，而动态规划法的运行时间仅略有增加。这表明动态规划法在处理大规模问题时具有更优越的性能。
- 3、在实验中，枚举法和动态规划法得到的背包最大价值结果相同，说明了动态规划方法的正确性。