背包问题：枚举法和dp方法时间对比

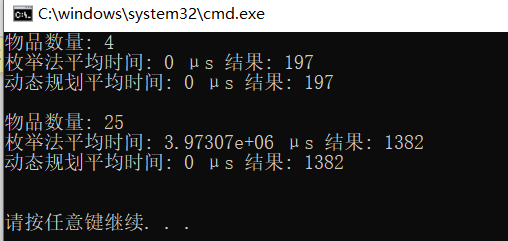
1. 代码：
2. #include <bits/stdc++.h>
3. #define MAXN 1050
4. #define MAXW 1050
5. #define ll long long
6. #define pii pair<int, int>
7. **using** **namespace** std;
9. ll dp[MAXN][MAXW];
10. pii item[MAXN];
11. **int** n\_values[100];
12. **int** w\_values[100];
14. ll knapsack\_bruteforce(**int** n, **int** w) {
15. ll ans = 0, one = 1;
16. **for** (ll i = 0; i < (one << n); i++) {
17. ll current\_weight = 0, current\_value = 0;
18. **for** (**int** j = 0; j < n; j++) {
19. **if** (i & (one << j)) {
20. current\_weight += item[j].first;
21. current\_value += item[j].second;
22. }
23. }
24. **if** (current\_weight <= w) {
25. ans = max(ans, current\_value);
26. }
27. }
28. **return** ans;
29. }
31. ll knapsack\_dp(**int** n, **int** w) {
32. **for** (**int** i = n - 1; i >= 0; i--) {
33. **for** (**int** j = 0; j <= w; j++) {
34. **int** wei = item[i].first, val = item[i].second;
35. **if** (j < wei) {
36. dp[i][j] = dp[i + 1][j];
37. } **else** {
38. dp[i][j] = max(dp[i + 1][j], dp[i + 1][j - wei] + val);
39. }
40. }
41. }
42. **return** dp[0][w];
43. }
45. **void** generate\_items(**int** n) {
46. srand(time(0));
47. **for** (**int** i = 0; i < n; ++i) {
48. item[i].first = rand() % 100 + 1;
49. item[i].second = rand() % 100 + 1;
50. }
51. }
53. **int** main() {
54. **int** lens = 2;
55. **int** test\_runs = 5;
56. n\_values[0] = 4; w\_values[0] = 200;
57. n\_values[1] = 25; w\_values[1] = 1000;
58. ll ans1 = 0, ans2 = 0;
59. **for** (**int** k = 0; k < lens; k++) {
60. **int** n = n\_values[k], w = w\_values[k];
61. generate\_items(n);
62. auto start\_bruteforce = chrono::high\_resolution\_clock::now();
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);
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);
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 ";
80. cout << "结果: " << ans2 << "\n" << "\n";
81. }
82. **return** 0;
83. }
84. 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：



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。