Reg.No : 24BCE0554
Name : Partha Pratim Gogoi
Topic : 0-1 Knapsack

Algorithm

```
1  zeroOneKnapsack(                                          [T.C: O(nw)]
           maxCapacity,
           weights[],
           values[],
           totalItems
   ):
   1.1     dp[totalItems+1][totalItems+1] = 0
   1.2     for item 1→totalItems:                             [T.C: O(n)]
     1.2.1   for capacity 0→maxCapacity:                      [T.C: O(w)]
        1.2.1.1   excludeItem = dp[item-1][capacity]
        1.2.1.2   if weights[item-1] <= capacity
        1.2.1.3       includeItem = values[item-1]
                                  + dp[item-1][capacity-weights[item-1]]
        1.2.1.4   dp[item][capacity] =max(includeItem, excludeItem)
   1.3     return dp[totalItems][maxCapacity]

   where,
        n = total number of items
        w = total capacity of knapsack
```

Time Complexity

Total Time Complexity = O(n.w)

Source Code

```
#include <iostream>
#include <vector>
using namespace std;

int zeroOneKnapsack(
            int maxCapacity,
            vector<int> weights,
            vector<int> values,
            int totalItems
      ) {
      // ---------------------------------------------------------------- //
      // --- dp[i][j] = max value using first i items with capacity j --- //
      // ---------------------------------------------------------------- //
      vector<vector<int>> dp(totalItems+1, vector<int>(maxCapacity+1, 0));

      // ---------------------- //
      // --- Build DP table --- //
      // ---------------------- //
      for (int item=1; item <= totalItems; item++) {
            for (int capacity=0; capacity <= maxCapacity; capacity++) {

                  // --- Option 1: Current item not taken
                  int excludeItem = dp[item-1][capacity];

                  // --- Option 2: Current item taken
                  int includeItem = 0;
                  if (weights[item-1] <= capacity) {
                        includeItem =
                              values[item-1]
                              + dp[item-1][capacity - weights[item-1]];
                  }

                  // --- Compare both and take best choice
                  dp[item][capacity] = max(includeItem, excludeItem);
            }
      }

      return dp[totalItems][maxCapacity];
}

int main() {
      vector<int> weights = {1,4,6,7};
      vector<int> values = {1,6,2,4};
      int maxCapacity = 7;

      cout << "Max Profit: ";
      cout << zeroOneKnapsack(
                  maxCapacity,
                  weights,
                  values,
                  weights.size()
            ) << endl;

      return 0;
}
```

Sample Output

```
24BCE0554

◄ 0s ◎ ./'0-1 Knapsack'/a.out
Max Profit: 7
```