|  |  |
| --- | --- |
| **0/1 KnapSack in C++** | |
| #include <iostream>  #include <vector>  using namespace std;  class ZeroOneKnapsack {  public:      int knapsack(int n, vector<int>& vals, vector<int>& wts, int cap) {          vector<vector<int>> dp(n + 1, vector<int>(cap + 1, 0));          for (int i = 1; i <= n; i++) {              for (int j = 1; j <= cap; j++) {                  if (j >= wts[i - 1]) {                      int remainingCap = j - wts[i - 1];                      if (dp[i - 1][remainingCap] + vals[i - 1] > dp[i - 1][j]) {                          dp[i][j] = dp[i - 1][remainingCap] + vals[i - 1];                      } else {                          dp[i][j] = dp[i - 1][j];                      }                  } else {                      dp[i][j] = dp[i - 1][j];                  }              }          }          return dp[n][cap];      }  };  int main() {      ZeroOneKnapsack solution;      // Input parameters      int n = 5;      vector<int> vals = {15, 14, 10, 45, 30};      vector<int> wts = {2, 5, 1, 3, 4};      int cap = 7;      // Compute maximum value using knapsack function      int maxVal = solution.knapsack(n, vals, wts, cap);      // Output the maximum value      cout << "Maximum value that can be obtained: " << maxVal << endl;      return 0;  } | **Dry Run of the ZeroOneKnapsack Problem:**  **Input:**  n = 5;  vals = {15, 14, 10, 45, 30};  wts = {2, 5, 1, 3, 4};  cap = 7;  **Step 1: Initialize the DP Table**  We initialize a 2D DP table of size (n + 1) x (cap + 1) to store the maximum values obtainable for each subproblem. Each cell dp[i][j] will represent the maximum value achievable with the first i items and a knapsack capacity of j.  Initially, the DP table is filled with zeros:   | **i\j** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   **Step 2: Fill the DP Table**  We iterate through each item (i = 1 to n) and each knapsack capacity (j = 1 to cap). The idea is to decide whether to include the current item or not.  **Item 1 (Value = 15, Weight = 2)**   * **Capacity 1**: dp[1][1] = 0 (Cannot include this item as the weight is greater than the capacity) * **Capacity 2**: dp[1][2] = max(dp[0][2], dp[0][0] + 15) = max(0, 15) = 15 * **Capacity 3**: dp[1][3] = max(dp[0][3], dp[0][1] + 15) = max(0, 15) = 15 * **Capacity 4**: dp[1][4] = max(dp[0][4], dp[0][2] + 15) = max(0, 15) = 15 * **Capacity 5**: dp[1][5] = max(dp[0][5], dp[0][3] + 15) = max(0, 15) = 15 * **Capacity 6**: dp[1][6] = max(dp[0][6], dp[0][4] + 15) = max(0, 15) = 15 * **Capacity 7**: dp[1][7] = max(dp[0][7], dp[0][5] + 15) = max(0, 15) = 15  | **i\j** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **1** | 0 | 0 | 15 | 15 | 15 | 15 | 15 | 15 | | **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   **Item 2 (Value = 14, Weight = 5)**   * **Capacity 1 to 4**: The weight is greater than the capacity, so we can't include this item. * **Capacity 5**: dp[2][5] = max(dp[1][5], dp[1][0] + 14) = max(15, 14) = 15 * **Capacity 6**: dp[2][6] = max(dp[1][6], dp[1][1] + 14) = max(15, 14) = 15 * **Capacity 7**: dp[2][7] = max(dp[1][7], dp[1][2] + 14) = max(15, 29) = 29  | **i\j** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **1** | 0 | 0 | 15 | 15 | 15 | 15 | 15 | 15 | | **2** | 0 | 0 | 15 | 15 | 15 | 15 | 15 | 29 | | **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   **Item 3 (Value = 10, Weight = 1)**   * **Capacity 1**: dp[3][1] = max(dp[2][1], dp[2][0] + 10) = max(0, 10) = 10 * **Capacity 2**: dp[3][2] = max(dp[2][2], dp[2][1] + 10) = max(15, 10) = 15 * **Capacity 3**: dp[3][3] = max(dp[2][3], dp[2][2] + 10) = max(15, 25) = 25 * **Capacity 4**: dp[3][4] = max(dp[2][4], dp[2][3] + 10) = max(15, 25) = 25 * **Capacity 5**: dp[3][5] = max(dp[2][5], dp[2][4] + 10) = max(15, 25) = 25 * **Capacity 6**: dp[3][6] = max(dp[2][6], dp[2][5] + 10) = max(15, 25) = 25 * **Capacity 7**: dp[3][7] = max(dp[2][7], dp[2][6] + 10) = max(29, 25) = 29  | **i\j** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **1** | 0 | 0 | 15 | 15 | 15 | 15 | 15 | 15 | | **2** | 0 | 0 | 15 | 15 | 15 | 15 | 15 | 29 | | **3** | 0 | 10 | 15 | 25 | 25 | 25 | 25 | 29 | | **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   **Item 4 (Value = 45, Weight = 3)**   * **Capacity 1 to 2**: Cannot include this item. * **Capacity 3**: dp[4][3] = max(dp[3][3], dp[3][0] + 45) = max(25, 45) = 45 * **Capacity 4**: dp[4][4] = max(dp[3][4], dp[3][1] + 45) = max(25, 55) = 55 * **Capacity 5**: dp[4][5] = max(dp[3][5], dp[3][2] + 45) = max(25, 55) = 55 * **Capacity 6**: dp[4][6] = max(dp[3][6], dp[3][3] + 45) = max(25, 70) = 70 * **Capacity 7**: dp[4][7] = max(dp[3][7], dp[3][4] + 45) = max(29, 70) = 70   **Item 5 (Value = 30, Weight = 4)**   * **Capacity 1 to 3**: Cannot include this item. * **Capacity 4**: dp[5][4] = max(dp[4][4], dp[4][0] + 30) = max(55, 30) = 55 * **Capacity 5**: dp[5][5] = max(dp[4][5], dp[4][1] + 30) = max(55, 30) = 55 * **Capacity 6**: dp[5][6] = max(dp[4][6], dp[4][2] + 30) = max(70, 30) = 70 * **Capacity 7**: dp[5][7] = max(dp[4][7], dp[4][3] + 30) = max(70, 75) = 75   **Step 3: Final DP Table**   | **i\j** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **1** | 0 | 0 | 15 | 15 | 15 | 15 | 15 | 15 | | **2** | 0 | 0 | 15 | 15 | 15 | 15 | 15 | 29 | | **3** | 0 | 10 | 15 | 25 | 25 | 25 | 25 | 29 | | **4** | 0 | 10 | 15 | 45 | 55 | 55 | 70 | 70 | | **5** | 0 | 10 | 15 | 45 | 55 | 55 | 70 | 75 |   **Result:**  The maximum value that can be obtained with a knapsack capacity of 7 is 75. |
| Output: Maximum value that can be obtained: 75 The maximum value that can be obtained is stored in dp[5][7] = 75. | |