**DAA Assignment 4**

**0/1 Knapsack Using Dynamic Programming and Branch-and-Bound**

**Neeti Kurulkar**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**// Structure to represent an item**

**struct Item {**

**int value;**

**int weight;**

**};**

**// Function to solve 0-1 Knapsack problem using dynamic programming**

**int knapsackDP(int capacity, const vector<Item> &items) {**

**int n = items.size();**

**// DP table: dp[i][w] stores max value using first i items with capacity w**

**vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));**

**// Build the DP table**

**for (int i = 1; i <= n; i++) {**

**for (int w = 0; w <= capacity; w++) {**

**if (items[i - 1].weight <= w) {**

**// Option 1: Take the item**

**// Option 2: Don't take the item**

**dp[i][w] = max(dp[i - 1][w],**

**items[i - 1].value + dp[i - 1][w - items[i - 1].weight]);**

**} else {**

**// Item too heavy, cannot take it**

**dp[i][w] = dp[i - 1][w];**

**}**

**}**

**}**

**// Maximum value is in the last cell**

**return dp[n][capacity];**

**}**

**int main() {**

**int n, capacity;**

**// Input number of items**

**cout << "Enter the number of items: ";**

**cin >> n;**

**vector<Item> items(n);**

**// Input value and weight for each item**

**for (int i = 0; i < n; i++) {**

**cout << "Enter value and weight for item " << i + 1 << ": ";**

**cin >> items[i].value >> items[i].weight;**

**}**

**// Input knapsack capacity**

**cout << "Enter the capacity of the knapsack: ";**

**cin >> capacity;**

**// Solve 0-1 Knapsack problem**

**int maxValue = knapsackDP(capacity, items);**

**cout << "Maximum value that can be carried in the knapsack: " << maxValue << endl;**

**return 0;**

**}**

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**