PRACTICAL -10

AIM: code and analyze to solve 0/1 knapsackusing dynamic programming.

CODE:

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

// Function to solve the 0/1 Knapsack problem using dynamic programming and return the DP table
vector<vector<int>> knapsack(int W, const vector<int>& weights, const vector<int>& values) {
    int n = weights.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    // Build the dp array
    for (int i = 1; i <= n; ++i) {
        for (int w = 0; w <= W; ++w) {
            if (weights[i - 1] <= w) {
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }
    return dp;
}

// Function to reconstruct the selected items
vector<int> getSelectedItems(int W, const vector<int>& weights, const vector<int>& values, const vector<vector<int>>& dp) {
    int n = weights.size();
    vector<int> selectedItems;
    int w = W;

    for (int i = n; i > 0 && w > 0; --i) {
```

```
        if (dp[i][w] != dp[i - 1][w]) {  // Item i-1 was included
            selectedItems.push_back(i - 1);
            w -= weights[i - 1];  // Reduce the weight
        }
    }
    return selectedItems;
}

// Function to print the DP table
void printDPTable(const vector<vector<int>>& dp) {
    int n = dp.size() - 1;
    int W = dp[0].size() - 1;
    cout << "DP Table:\n";
    for (int i = 0; i <= n; ++i) {
        for (int w = 0; w <= W; ++w) {
            cout << dp[i][w] << "\t";
        }
        cout << "\n";
    }
}

int main() {
    cout << "uday narayan\nURN= 2203572\n";
    int n, W;

    // Input number of items and maximum weight capacity
    cout << "Enter the number of items: ";
    cin >> n;
    cout << "Enter the maximum weight capacity of the knapsack: ";
    cin >> W;

    vector<int> weights(n), values(n);

    // Input weights and values for each item
```

```
    for (int i = 0; i < n; ++i) {
        cout << "Enter weight of item " << i + 1 << ": ";
        cin >> weights[i];
        cout << "Enter value of item " << i + 1 << ": ";
        cin >> values[i];
    }

    // Solve the knapsack problem and get the DP table
    vector<vector<int>> dp = knapsack(W, weights, values);

    // Get the selected items
    vector<int> selectedItems = getSelectedItems(W, weights, values, dp);

    // Output the maximum value that can be obtained
    cout << "Maximum value that can be obtained: " << dp[n][W] << endl;

    // Output the items included in the knapsack
    cout << "Items included in the knapsack: ";
    for (int item : selectedItems) {
        cout << "Item " << item + 1 << " ";  // Convert back to 1-based index
    }
    cout << endl;

    // Print the DP table
    printDPTable(dp);

    return 0;
}
```

Output:

```
uday narayan
URN= 2203572
Enter the number of items: 3
Enter the maximum weight capacity of the knapsack: 6
Enter weight of item 1: 2
Enter value of item 1: 1
Enter weight of item 2: 3
Enter value of item 2: 2
Enter weight of item 3: 4
Enter value of item 3: 5
Maximum value that can be obtained: 6
Items included in the knapsack: Item 3 Item 1
DP Table:
0       0       0       0       0       0       0
0       0       1       1       1       1       1
0       0       1       2       2       3       3
0       0       1       2       5       5       6
```