**Experiment-9**

**Aim-** Implement Knapsack Problem based on Backtracking algorithm.

**Theory-**

In the 0/1 Knapsack Problem, we can also use backtracking to explore all possible combinations of including or excluding items. This method recursively explores both choices for each item and calculates the maximum value, pruning branches when it becomes clear that they cannot lead to an optimal solution. Backtracking is useful but inefficient for larger problem sizes due to its exponential time complexity.

**Software Used –** Visual Studio Code

**Code-**

#include <iostream>

#include <vector>

using namespace std;

// Backtracking function to solve the 0/1 Knapsack problem

int knapsackRec(vector<int>& val, vector<int>& wt, int W, int n) {

// Base case: if no items are left or capacity is 0

if (n == 0 || W == 0) {

return 0;

}

// Get the weight and value of the current item

int itemWt = wt[n - 1];

int itemVal = val[n - 1];

// Check if the current item can be included in the knapsack

if (itemWt <= W) {

// Explore both possibilities:

// 1. Include the current item

int include = knapsackRec(val, wt, W - itemWt, n - 1) + itemVal;

// 2. Exclude the current item

int exclude = knapsackRec(val, wt, W, n - 1);

// Return the maximum of including or excluding the item

return max(include, exclude);

} else {

// If the item can't be included, exclude it

return knapsackRec(val, wt, W, n - 1);

}

}

int main() {

// Input the number of items

int n;

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

cin >> n;

// Input the values and weights of the items

vector<int> val(n), wt(n);

cout << "Enter the values of the items: ";

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

cin >> val[i];

}

cout << "Enter the weights of the items: ";

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

cin >> wt[i];

}

// Input the capacity of the knapsack

int W;

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

cin >> W;

// Call the backtracking function

int maxProfit = knapsackRec(val, wt, W, n);

// Output the result

cout << "The maximum value that can be obtained is: " << maxProfit << endl;

return 0;

}

**Output-**

**A screen shot of a computer code

Description automatically generated**