"Heaven's Light is Our Guide"

Department of Computer Science & Engineering

RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY

Lab Report

Course No: CSE 2202

Course Name: Sessional Based on CSE 2201

Submitted to:

Biprodip Pal

Assistant Professor

Department of Computer Science & Engineering

Submitted by:

Md. Al Siam
Department of Computer Science & Engineering
Session: 2016-2017
Section: A
Roll No.: 1603008

Problem

0/1 Knapsack Problem.

Approach

1. Initialize N objects with their weights and profits.
2. For a Knapsack having 70% weight capacity of total weight of N objects:
   - Find the maximum profit for 0/1 knapsack using Brute force approach generating 2^n solutions and checking.
   - Find the maximum profit for 0/1 knapsack using greedy approach.

Greedy Approach

```
#include <bits/stdc++.h>
using namespace std;

typedef struct{int weight; int profit;} dataset;

bool compare(dataset a, dataset b){
   return a.profit > b.profit;
}

int main(){


   int n;
   cout << "How many data do you want to take? : ";
   cin >> n;

   dataset input[n+2];
```

```
    int total_wt = 0;

    cout << "Enter weights:\n";
    for(int i = 0; i < n; i++){
        scanf("%d", &input[i].weight);
        total_wt += input[i].weight;
    }

    int max_wt = (int) ((total_wt*70)/100);

    cout << "Enter profits:\n";
    for(int i = 0; i < n; i++){
        scanf("%d", &input[i].profit);
    }

    sort(input, input+n, compare);

    int total_profit = 0;
    dataset ans[n+2];
    int index = 0;

    clock_t start = clock();
    for(int i = 0; (i < n) && ((max_wt - input[i].weight) >= 0); i++){
        total_profit += input[i].profit;
        max_wt -= input[i].weight;
        ans[index].weight = input[i].weight;
        ans[index].profit = input[i].profit;
        index++;
    }
    clock_t stop = clock();
    double duration = (double)((stop - start) / 2.4e9);

    cout << "Time Required = " << setprecision(12) << ((stop - start) / 2.4e9) <<
endl;
```

```
    cout << "Maximum Profit: " << total_profit << endl;

    for(int i = 0; i < index; i++){
        cout << "Wt: " << ans[i].weight
        << " Profit: " << ans[i].profit << endl;
    }
}
```

Brute Force Approach

```
#include <bits/stdc++.h>
using namespace std;

int weight[1010];
int profit[1010];
int number_of_items;
int capacity;

int knapsac(int i, int w){
    if(i >= number_of_items) return 0;

    int profit1 = 0, profit2 = 0;

    if(w + weight[i] <= capacity){
        profit1 = profit[i] + knapsac(i+1, w+weight[i]);
    }

    profit2 = knapsac(i+1, w);

    return max(profit1, profit2);
}

int main(){

    int n;
```

```
    cout << "How many data do you want to take? : ";
    cin >> number_of_items;

    int total = 0;

    cout << "Enter weights:\n";
    for(int i = 0; i < number_of_items; i++){
        scanf("%d", &weight[i]);
        total += weight[i];
    }

    cout << "Enter profits:\n";
    for(int i = 0; i < number_of_items; i++){
        scanf("%d", &profit[i]);
    }

    capacity = (total*70)/100;
    cout << capacity << endl;

    clock_t start = clock();

    int ans = knapsac(0, 0);

    clock_t stop = clock();
    double duration = (double)((stop - start) / 2.4e9);

    cout << "Time Required = " << setprecision(12) << ((stop - start) / 2.4e9) <<
endl;

    cout << "Maximum profit = " << ans << endl;

}
```

Experimental Result

N weights[] Profits[]	Greedy Selected Weights	Greedy Profit	Greedy Time (sec)	Brute Force Profit	Brute Force Time (sec)
5 7 15 3 10 5 5 10 12 15 20	5 10 3	47	1.22232e-006	52	2.67541e-006
10 12 3 23 34 55 7 1 78 45 11 11 10 36 91 67 12 56 56 44 3	34 55 1 78	270	1.51000e-006	327	1.89225e-006
15 12 52 91 77 10 101 50 34 56 8 3 8 69 74 6 111 45 67 10 1 0 0 99 66 69 34 44 5 83 129	6 12 34 74 8 91 56 52 8 3 77	757	1.56875e-006	758	2.01951e-006