DEsign and analysis of algorithms

By Utsav Balar

201903103510391

BTech CE-C Batch 1

LAB MANUAL

**Practical 5**

**Aim: A thief robbing a store finds n items, ith item is worth vi dollers and weights wi pounds where vi and wi are integers. He wants to take as valuable a load as possible but he can carry atmost W pounds in his knapsack where W is**

**an integer. Which items should he take, where condition is that he is**

**allowed to take or select fractional part of an item?**

**w = 50, n = 3**

Table

Description automatically generated

**Code:**

// Knapsack problem algorithm

#include <stdio.h>

/\*\*

\* swap - Swapping algorithm

\* @x: Number 1

\* @y: Number 2

\*/

void swap(float \*x, float \*y)

{

float temp = \*x;

\*x = \*y;

\*y = temp;

}

/\*\*

\* fractional\_knapsack - Fractional knapsack algorithm using Greedy approach

\* @w: List of all the objects weight

\* @V: List of all the objects value

\* @W: Capacity of the knapsack

\* @n: Total numbers of objects

\*/

void fractional\_knapsack(float w[], float V[], float W, int n)

{

float x[3], weight = 0.0, profit = 0.0;

// Initialization

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

x[i] = 0.0;

for (int i = 0; weight < W; i++)

{

if (weight + w[i] < W)

{

x[i] = 1.0;

weight = weight + w[i];

profit = profit + V[i];

// printf("Object: %d: \tx[%d] = %0.1f\tweight = %0.1f\tw[%d] = %0.1f\tV[%d] = %0.1f\tW = %0.1f\n",

// i + 1, i + 1, x[i], weight, i + 1, w[i], i + 1, V[i], W);

}

else

{

x[i] = (W - weight) / w[i];

weight = W;

profit = profit + (x[i] \* V[i]);

// printf("Object: %d: \tx[%d] = %0.1f\tweight = %0.1f\tw[%d] = %0.1f\tV[%d] = %0.1f\tW = %0.1f\n",

// i + 1, i + 1, x[i], weight, i + 1, w[i], i + 1, V[i], W);

}

}

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

// {

// if (x[i] == 1.0)

// printf("\nObject added = %d\t\tProfit = $%0.1f\tWeight = %0.1f",

// i + 1, V[i], w[i]);

// else if (x[i] > 0.0)

// printf("\nAdded %0.1f part of Object %d\tProfit = $%0.1f\tWeight %0.1f",

// x[i], i + 1, (x[i] \* V[i]), w[i]);

// }

printf("\nThe Maximum Value for %d objects with load %0.1f = $%0.1f\n\n",

n, W, profit);

}

/\*\*

\* knapsack - Knapsack algorithm without considering fractional weight

\* @w: List of all the objects weight

\* @V: List of all the objects value

\* @W: Capacity of the knapsack

\* @n: Total numbers of objects

\*/

void knapsack(float w[], float V[], float W, int n)

{

float x[3], weight = 0.0, profit = 0.0;

// Initialization

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

x[i] = 0.0;

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

{

if (weight + w[i] < W)

{

x[i] = 1.0;

weight = weight + w[i];

profit = profit + V[i];

// printf("Object: %d: \tx[%d] = %0.1f\tweight = %0.1f\tw[%d] = %0.1f\tV[%d] = %0.1f\tW = %0.1f\n",

// i + 1, i + 1, x[i], weight, i + 1, w[i], i + 1, V[i], W);

}

}

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

// {

// if (x[i] == 1.0)

// printf("\nObject added = %d\t\tProfit = $%0.1f\tWeight = %0.1f",

// i + 1, V[i], w[i]);

// }

printf("\nThe Maximum Value for %d objects with load %0.1f = $%0.1f\n\n",

n, W, profit);

}

int main(void)

{

// Start: provided from the question

int object[3] = {1, 2, 3};

float weight[3] = {10, 20, 30};

float value[3] = {60, 100, 150};

float capacity = 50;

int n = 3;

// End:

/\* NOTE: Create an array for storing value/weight

\* for all the objects

\*/

float ratio[n];

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

{

ratio[i] = value[i] / weight[i];

// printf("Object: %d\t ratio=%0.1f\n", i, ratio[i]);

}

/\* NOTE: Re-Arrange the array in such a way

\* that value/weight is in descending order

\*/

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

{

for (int j = 0; j < n; j++)

{

if (ratio[i] > ratio[j])

{

swap(&ratio[i], &ratio[j]);

swap(&weight[i], &weight[j]);

swap(&value[i], &value[j]);

}

}

}

int choice = 0; // Default is Fractional(Greedy)

printf("As a thief yourself Enter the choice:\n0 = fractional\t1 = not fractional: \n");

scanf("%d", &choice);

// Validation for user choice

if (choice > 1 || choice < 0)

{

printf("Please input with either 0 or 1: ");

scanf("%d", &choice);

}

switch (choice)

{

case 0:

printf("Using Fractional values\n");

fractional\_knapsack(weight, value, capacity, n);

break;

case 1:

printf("Not using fractional values\n");

knapsack(weight, value, capacity, n);

break;

default:

break;

}

return 0;

}

**Output: (When fractional part was allowed)**

A picture containing text

Description automatically generated

**Output: (When fractional part was not allowed)**

Text

Description automatically generated

**Practical 6**

**Aim:**