19 Knapsack

#include <stdio.h>

// Structure to represent an item in the knapsack problem

typedef struct Item {

int value;

int weight;

double ratio; // Value-to-weight ratio

} Item;

// Function to sort items in descending order of their value-to-weight ratio

int compare\_items(const void\* a, const void\* b) {

const Item\* item1 = (Item\*)a;

const Item\* item2 = (Item\*)b;

return item2->ratio - item1->ratio;

}

// Function to solve the fractional knapsack problem using the greedy approach

double fractional\_knapsack(Item items[], int n, int capacity) {

// Sort items by value-to-weight ratio in descending order

qsort(items, n, sizeof(Item), compare\_items);

// Initialize total profit and remaining capacity

double total\_profit = 0.0;

int remaining\_capacity = capacity;

// Iterate through items in the sorted order

for (int i = 0; i < n && remaining\_capacity > 0; i++) {

// If the item's weight is less than or equal to the remaining capacity, add the entire item

if (items[i].weight <= remaining\_capacity) {

total\_profit += items[i].value;

remaining\_capacity -= items[i].weight;

} else {

// Otherwise, add a fraction of the item to maximize profit within the remaining capacity

double fraction = (double)remaining\_capacity / items[i].weight;

total\_profit += fraction \* items[i].value;

remaining\_capacity = 0;

}

}

return total\_profit;

}

int main() {

int n, capacity;

printf("Enter the number of items: ");

scanf("%d", &n);

printf("Enter the capacity of the knapsack: ");

scanf("%d", &capacity);

Item items[n];

printf("Enter the value and weight of each item:\n");

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

scanf("%d %d", &items[i].value, &items[i].weight);

items[i].ratio = (double)items[i].value / items[i].weight; // Calculate value-to-weight ratio

}

double max\_profit = fractional\_knapsack(items, n, capacity);

printf("Maximum profit using fractional knapsack approach: %.2lf\n", max\_profit);

return 0;

}