**Assignment #5 (5%)**

Submission deadline: Sunday, **April 24, 2022** **(23:59)**

**Chapters:** This assignment covers Chapter 6 in the textbook

NAME: TALHA ABDULLAH QUID: 201903446

**Important Notes (must read):**

1. When submitting your work, you must use Blackboard, **NO other means like email submissions, are accepted.**
2. Assignments are to be solved **individually**.
3. A mark of zero (0) will be awarded for the whole assessment in which plagiarism was found to occur. Even if a single question is plagiarized**, the whole assignment will get zero** (0).
4. Submit your work as instructed below ***before*** the deadline**. No extension will be provided.**
5. Along with the MS Word submission file, you must submit separate Java files for the programs. Put all these files in a folder named **Assignment5\_QUID**. Compress this folder and submit it.
6. In the Word document, make sure that to add screenshots for input and output of your programs.

If you have any questions or doubts about any of the above-mentioned issues, please consult Eng. Alaa Hussein [alaa.hussein@qu.edu.qa](mailto:alaa.hussein@qu.edu.qa) . There is only one question in this assignment.

**Q1.** The Knapsack problem is a problem in combinatorial optimization. This problem also arises in resource allocation where the decision makers have to choose from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively. As such, this problem can be applied in many different contexts.

Given a set of items, each with a weight and a value, we want to determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. What makes Branch-and-Bound (B&B) algorithm more efficient than Backtracking in this context is that instead of traversing the tree in a predetermined order (depth-first, breadth-first, etc.), we traverse it based on the optimization criteria for the problem.

**Problem:** Let n items be given, where each item has a weight and a profit. The weights and profits are positive integers. Furthermore, let a positive integer W be given. Determine a set of items with maximum total profit, under the constraint that the sum of their weights cannot exceed W.

**Inputs:** Positive integers n and W, arrays of positive integers w and p, each indexed from 1 to n, and each of which is sorted in non-increasing order according to the values of p[i]/w[i].

**Outputs:** An integer maxprofit that is the sum of the profits of an optimal set.

Write a Java program using the **Best-First Search with Branch-and-Bound Pruning algorithm** for the **0-1 Knapsack problem** and demonstrate its correctness with some input values.

Answer: (please write your answer here, add required space if needed)

package Assignment5;

import java.util.Arrays;

import java.util.LinkedList;

import java.util.Queue;

public class BestSearchWithBranchAndBound {

public static void main(String [] args) {

int w=16;

Item arr[]= new Item[4];

arr[0]=new Item(2,40);

arr[1]=new Item(5,30);

arr[2]=new Item(7,35);

arr[3]=new Item(8,20);

int n=arr.length;

System.out.println("Maximum profit is"+ks(w, arr, n));

}

public static int findBound(Node n, int z,int w,Item arr[]) {

if(n.weight>=w) {

return 0;

}

else {

int profit=n.profit;

int i=n.level+1;

int totalweight=(int) n.weight;

while(i<z && totalweight+arr[i].weight<=w) {

totalweight+=arr[i].weight;

profit+=arr[i].value;

i++;

}

if(i<z) {

profit+=(w-totalweight)\*arr[i].value/arr[i].weight;

}

return profit;

}

}

public static int ks(int w,Item arr[],int n) {

for(int i=0;i<n-1;i++) {

Item first=new Item(arr[i].weight,arr[i].value);

Item second=new Item(arr[i+1].weight,arr[i+1].value);

boolean c=compare(first, second);

if(c==true) {

arr[i]=arr[i+1];

}

else {

arr[i]=arr[i];

}

}

Queue<Node>q = new LinkedList<Node>();

Node a = new Node(0, 0, 0, 0),b=new Node(0, 0, 0, 0);

a.level=-1;

a.profit=0;

a.weight=0;

q.add(a);

int maxP=0;

while(!q.isEmpty()) {

a=q.peek();

q.remove();

if(a.level== -1) {

b.level=0;

}

if(a.level==n-1) {

continue;

}

b.level=a.level+1;

b.weight=a.weight+arr[b.level].weight;

b.profit=a.profit+arr[b.level].value;

if(b.weight<=w && b.profit>maxP) {

maxP=b.profit;

b.bound=findBound(b,n,w,arr);

}

if(b.bound>maxP) {

q.add(b);

}

}

return maxP;

}

public static boolean compare(Item first,Item second) {

double r1=(double)first.value/first.weight;

double r2=(double)second.value/second.weight;

return r1>r2;

}

}

OUTPUT:

Graphical user interface, text, application, email

Description automatically generated