Department of education and science of Ukraine

National technical university of Ukraine

«Kyiv polytechnic institute the name of Igor Sikorsky»

Faculty of informatics and computing engineering

Department of the computing engineering

Laboratory work №8

Discipline: «The algorithms theory»

Topic: «Dynamic programming»

EXECUTED:

The first-year student

of FICT group IP-95

*Guskov Danil*

The Student book number- 9505

CHECKED:

Associate Professor

The Computer Engineering Department

PhD, SR

Antoniuk A.I.

Kyiv - 2020

**TASK**

**Goal:**

Implementation of a dynamic programming algorithm for the backpack problem.

Dynamic programming is a powerful enough method for developing algorithms for solving problems that can be decomposed into smaller problems, from which, in the end, the initial problem is solved.

**Task variant: 5**

Applying dynamic programming to the backpack problem is one of the prime examples of the power of this approach. This task is formulated as follows:

Given are n different items known about their size, or weight, wi, and cost vi. There is a backpack in which to put these items. The backpack is known for its capacity (total size, or weight, of items that can be placed in the backpack) - W. It is necessary to select a set of items S among all given items that (1) their total dimension does not exceed the capacity of the backpack W and (2) the total value of the objects in the set S is the highest possible among all other sets.

To solve the backpack problem by dynamic programming, it is necessary to first formulate the solution of the initial problem by solving smaller problems. To do this, denote by S - the maximum number of items that can be placed in a backpack. Suppose that the last object No. n belongs to S, then S - {n} is the optimal solution for the first n-1 objects and the backpack capacity (W - wn).

It is now possible to formulate a recursive rule to solve the problem. Denote by V (i, x) the highest value such that (1) only the first i objects are considered and (2) the total size of the objects does not exceed x.

Then:

V (i, x) = max {V (i - 1, x), V (i - 1, x - wi) + vi} (\*)

Note: when wi> x, V (i, x) = V (i - 1, x).

Therefore, the dynamic programming algorithm must sort through subtasks of all possible dimensions, which are defined by two variables:

• and - number of objects (1, n)

• x - backpack capacity (1, W)

The speed of such an algorithm is O (nW). Thus, it cannot be fully attributed to polynomial algorithms and therefore it has a pseudopolynomial characteristic.

**SOFTWARE CODE**

**'use strict';**

**// Class for items**

**class Item {**

**constructor(wi, vi){**

**this.wi = wi;**

**this.vi = vi;**

**}**

**}**

**// find max between two values**

**const max = (a, b) => (a > b) ? a : b;**

**// IMP //**

**const fillInBackpack = (items, w) => {**

**// count of items**

**const n = items.length-1;**

**const itemsInBackpack = [];**

**//**

**const putInBackpack = (i) => {**

**}**

**// A recursive rule to solve a task**

**// i - the number of items (1, n)**

**// x - backpack capacity (1, w)**

**const v = (i, x) => {**

**if(i == -1) {**

**return 0;**

**}**

**const a = v(i-1, x);**

**let b = 0;**

**let m = 0;**

**if(x >= items[i].wi){**

**b = v(i-1, x-items[i].wi) + items[i].vi;**

**m = max(a, b);**

**if(m === b){**

**if(!itemsInBackpack.includes(items[i])){**

**itemsInBackpack.push(items[i]);**

**}**

**} else {**

**if(itemsInBackpack[itemsInBackpack.length-1] === items[i]){**

**itemsInBackpack.pop();**

**}**

**}**

**} else {**

**m = a;**

**}**

**return m;**

**}**

**const cost = v(n, w);**

**return { cost, itemsInBackpack };**

**}**

**// USAGE //**

**// inputed items**

**const inputItems = [**

**[10, 12],**

**[1, 2],**

**[5, 6],**

**[7, 16],**

**[8, 4],**

**[22, 40],**

**[18, 15],**

**[15, 30],**

**[19, 25],**

**[1, 1],**

**[9, 9],**

**[12, 6],**

**[6, 12]**

**];**

**// inputed backpack "W" parameter**

**const w = 100;**

**// create items**

**const items = [];**

**inputItems.forEach( item => {**

**items.push(new Item(...item));**

**});**

**console.log('\n--- Input "W" parameter: ');**

**console.log(w);**

**console.log('--- Input items: ');**

**console.dir(items);**

**const {cost, itemsInBackpack} = fillInBackpack(items, w);**

**console.log('\n--- Cost items in backpack: ');**

**console.log(cost);**

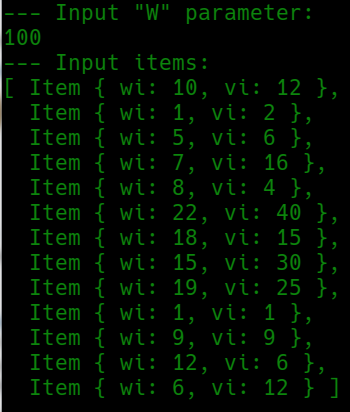
**console.log('--- Items in backpack: ');**

**console.dir(itemsInBackpack);**

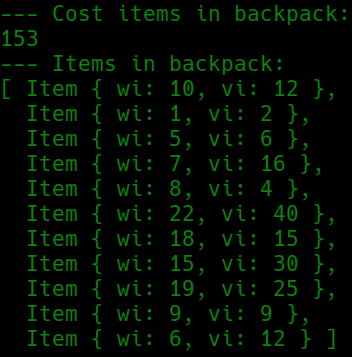
**console.log('\n')**

**RESULTS OF THE PROGRAM WORK**

The input:



Output:



**CONCLUSIONS**

Familiarized with the topic of laboratory work.

Have acquired relevant work skills.

An appropriate test program has been developed.

The results of the successful work of the test program above confirm the correctness of the chosen decisions, the ultimate goal of the work has been achieved.