CodeCrunch

CS1010E 16/17S1 Knapsack (Question)

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Knapsack

Suppose you are a notorious art thief planning one last grand heist on an art museum. The museum carries a collection of art pieces each having a certain weight and value. You can only carry items up to a maximum weight of `w`. Your aim is to maximize the value of your heist.

As an example, consider a maximum weight of `25`, and the following items, each of the form , up for grabs:

```
<26ab0db9, 60, 10>
<6d7fce9f, 20, 5>
```

There are two ways to gather your loot, either

- **greedily** -- by grabbing items one at a time starting with the item with the most *value per weight*.

  Using the above example, you will first grab the items `<26ab0db9, 60, 10>`. The next item of a higher value per weight is `;` however picking that would exceed the maximum weight. So, we have to contend with picking `<6d7fce9f, 20, 5>`, by which time you would have accumulated a loot of value `80` with weight `15`.

- **optimally** -- realize that greed often leads to irrational decisions. You would be better off foregoing the item with the highest value per weight and instead grab the items and `<6d7fce9f, 20, 5>`. Your loot is now valued at `120`, while still keeping within the maximum weight of `25`.

  The optimum strategy requires us to find all possible ways of picking items, and the corresponding total value and weight of these items. Suppose there are a total of `3` items. Since each item can either be taken or not, there are altogether 2^3 `- 1 = 7` possible combinations (ignoring the combination of not taking anything). These `7` combinations can be represented with the values `{1,2,3,4,5,6,7}`. Below are several examples to illustrate how the different combinations of items are obtained.

  - When the number is `1`, divide this number by `2` until it becomes zero:
    - `1 ÷ 2 = 0 remainder 1`

    The remainder `{1}` denotes picking only the first item .

  - When the number is `4`, divide this number by `2` until it becomes zero:
    - `4 ÷ 2 = 2 remainder 0`
    - `2 ÷ 2 = 1 remainder 0`
    - `1 ÷ 2 = 0 remainder 1`

    The sequence of remainders `{0,0,1}` denotes picking only the third item `<6d7fce9f, 20, 5>`.

  - When the number `6` and divide this number by `2` until it becomes zero:
    - `6 ÷ 2 = 3 remainder 0`
    - `3 ÷ 2 = 1 remainder 1`
    - `1 ÷ 2 = 0 remainder 1`

  The sequence of remainders `{0,1,1}` denotes picking the second and third items `<26ab0db9, 60, 10>` and `<6d7fce9f, 20, 5>`. By doing the above with all seven values from `1` to `7`, we can obtain all possible ways of picking items.

Task

Write a program that first reads the maximum weight `w` followed by the number of items `n`. The program then reads each item in the following form:

```
i1  v1  w1
i2  v2  w2
.  .   .
.  .   .
in  vn  wn
```

where each line is a separate item description, consisting of a eight-character identifier, followed by the value and the weight of the item. The program outputs the details of the loot using both greedy and optimal strategies.

Take note of the following:

- Assume that each item's value and weight are positive integers and there will not be more than `10` items.
- There will be at least one item available to be picked.
- Only one sample test case is provided to test for format correctness. You should device your own test cases to test your program.

This task is divided into several levels. Read through all the levels (from first to last, then from last to first) to see how the different levels are related. **You may start from any level.**

Level 1

Name your program `knapsack1.c`

Write a program that first reads the maximum weight `w` followed by the number of items `n`, and outputs the values.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
./a.out
25 3
25 3
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < knapsack.in | diff - knapsack1.out
```

To proceed to the next level (say level 2), copy your program by typing the Unix command

```
cp knapsack1.c knapsack2.c
```

Level 2

Name your program `knapsack2.c`

Write a program that first reads the maximum weight `w` followed by the number of items `n`, and outputs the values. The program then repeatedly reads `n` items, each a triplet consisting of an eight-character identifier, the value, as well as the weight. Output all items on separate lines.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
./a.out
25 3
25 3
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < knapsack.in | diff - knapsack2.out
```

To proceed to the next level (say level 3), copy your program by typing the Unix command

```
cp knapsack2.c knapsack3.c
```

Level 3

Name your program `knapsack3.c`

Write a program that first reads the maximum weight `w` followed by the number of items `n`. The program then repeatedly reads `n` items, each a triplet consisting of an eight-character identifier, the value, as well as the weight. Output all items on separate lines together with the total value and weight.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
./a.out
25 3
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
total value=180; total weight=35
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < knapsack.in | diff - knapsack3.out
```

To proceed to the next level (say level 4), copy your program by typing the Unix command

```
cp knapsack3.c knapsack4.c
```

Level 4

Name your program `knapsack4.c`

Write a program that first reads the maximum weight `w` followed by the number of items `n`. The program then repeatedly reads `n` items, each a triplet consisting of an eight-character identifier, the value, as well as the weight. Output all items on separate lines in order of decreasing value per weight.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
./a.out
25 3
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
26ab0db9 60 10
b026324c 100 20
6d7fce9f 20 5
total value=180; total weight=35
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < knapsack.in | diff - knapsack4.out
```

To proceed to the next level (say level 5), copy your program by typing the Unix command

```
cp knapsack4.c knapsack5.c
```

Level 5

Name your program `knapsack5.c`

Write a program that first reads the maximum weight `w` followed by the number of items `n`. The program then repeatedly reads `n` items, each a triplet consisting of an eight-character identifier, the value, as well as the weight. By grabbing items greedily, output all items grabbed on

separate lines in order of decreasing value per weight. Output the final value and weight of the loot.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
./a.out
25 3
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
GREEDY:
26ab0db9 60 10
6d7fce9f 20 5
total value=80; total weight=15
```

```
./a.out
2 3
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
GREEDY:
total value=0; total weight=0
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < knapsack.in | diff - knapsack5.out
```

To proceed to the next level (say level 6), copy your program by typing the Unix command

```
cp knapsack5.c knapsack6.c
```

Level 6

Name your program `knapsack6.c`

Write a program that first reads the maximum weight `w` followed by the number of items `n`. The program then repeatedly reads `n` items, each a triplet consisting of an eight-character identifier, the value, as well as the weight. By grabbing items greedily, output all items grabbed on separate lines in order of decreasing value per weight. Output the final value and weight of the loot.

For the optimal strategy, output all possible ways of picking items.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
./a.out
25 3
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
GREEDY:
26ab0db9 60 10
6d7fce9f 20 5
total value=80; total weight=15
1:[b026324c]
2:[26ab0db9]
3:[b026324c][26ab0db9]
4:[6d7fce9f]
5:[b026324c][6d7fce9f]
6:[26ab0db9][6d7fce9f]
7:[b026324c][26ab0db9][6d7fce9f]
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < knapsack.in | diff - knapsack6.out
```

To proceed to the next level (say level 7), copy your program by typing the Unix command

```
cp knapsack6.c knapsack7.c
```

Level 7

Name your program `knapsack7.c`

Write a program that first reads the maximum weight `w` followed by the number of items `n`. The program then repeatedly reads `n` items, each a triplet consisting of an eight-character identifier, the value, as well as the weight. By grabbing items greedily, output all items grabbed on separate lines in order of decreasing value per weight. Output the final value and weight of the loot.

For the optimal strategy, output all possible ways of picking items and the corresponding value and weight of the combination.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
./a.out
25 3
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
GREEDY:
26ab0db9 60 10
6d7fce9f 20 5
total value=80; total weight=15
1:[b026324c] total value=100; total weight=20
2:[26ab0db9] total value=60; total weight=10
3:[b026324c][26ab0db9] total value=160; total weight=30
4:[6d7fce9f] total value=20; total weight=5
5:[b026324c][6d7fce9f] total value=120; total weight=25
6:[26ab0db9][6d7fce9f] total value=80; total weight=15
7:[b026324c][26ab0db9][6d7fce9f] total value=180; total weight=35
```

Click here to submit to CodeCrunch.

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < knapsack.in | diff - knapsack7.out
```

To proceed to the next level (say level 8), copy your program by typing the Unix command

```
cp knapsack7.c knapsack8.c
```

Level 8

Name your program `knapsack8.c`

Write a program that first reads the maximum weight `w` followed by the number of items `n`. The program then repeatedly reads `n` items, each a triplet consisting of an eight-character identifier, the value, as well as the weight. By grabbing items greedily, output all items grabbed on separate lines in order of decreasing value per weight. Output the value and weight of this greedy loot.

By grabbing items optimally, output all items grabbed on separate lines following the order as given in the problem description. If there are multiple such combinations, pick the earliest one. Output the value and weight of this optimum loot.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
./a.out
25 3
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
GREEDY:
26ab0db9 60 10
6d7fce9f 20 5
total value=80; total weight=15
OPTIMUM:
b026324c 100 20
6d7fce9f 20 5
total value=120; total weight=25
```

```
./a.out
2 3
b026324c 100 20
26ab0db9 60 10
6d7fce9f 20 5
GREEDY:
total value=0; total weight=0
OPTIMUM:
total value=0; total weight=0
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < knapsack.in | diff - knapsack8.out
```