Joule Thief

Programming Project 1

due on Tuesday, 10 October 2023

You are a nefarious, notorious, intensely narcissistic, nationally-renown jewel thief. You have traveled far and wide in search of the most prized possessions of the social elite. Except that you're not truly nefarious, due to an inexplicable weakness for kittens. Notoriety has been achieved through a particularly embarrassing episode involving pants, a pig, and a minor electrocution that may have caused the power grid in Manhattan to shut down (though it **has not** been proven). The power outage could have just been a coincidence, you see.

What was not a coincidence was the subsequent arrest of your microwaved butt and a slab of bacon. No pants were found at the scene of the crime. No questions were asked about pants. No answers were provided. The headlines following the event featured articles such as "Pearls Before Swine," "A Tale of Two Electrocutions," and "Joule Thief." You did not find the inherent humor in any of these articles. National frenzy took over, as everyone wanted a piece of the grossly incompetent thief. Hence, nationally-renowned.

As a parenthetical note, it bears pointing out that you weren't particularly good at thievery of any kind. So in point of fact, you are little more than an intensely narcissistic loser. These things sometimes happen to the best of people. They happen much more often to people like you.

Anyway, you're a greedy narcissistic loser, you needed work, and there was a local farmer who wanted to steal his neighbor's produce. It's a step down from priceless gems, but it's a living. You are given a *knapsack*, a pat on the back, and a directive to bring back the most valuable haul you can. Pick dem berries!

Job Details. Your task is to implement a *greedy algorithm* to solve the 0-1 Knapsack problem in `C++`. You may work in groups of two (2) or three (3) if you wish.

Your algorithm will greedily choose the item that has the best profit-to-weight ratio. As we discussed in class, this greedy algorithm will not necessarily find the optimal answer for any arbitrary input! The input is a text file with the following form:

```
7 25
orange 50 5
banana 60 10
kitchensink 140 20
strawberry 100 14
tangerine 10 5
puppy 25 12
grape 30 7
```

The first line of input gives the *number of items* in the file, followed by the *knapsack capacity.* Each subsequent line lists a type of item (such as an orange), with the value per item (in pesos) and a weight per item (in pounds). The goal is to choose the items that maximizes the value stored in the knapsack. The optimal solution to the above example is to use the $(50, 5)$ and $(140, 20)$ items for a total weight of 25 pounds and a value of 190 pesos.

Your program should accept as input the name of a text file of the form above, and it should output a list of items to select, together with the total weight and value.

Steps Before the Profit Phase. You should probably first write a priority queue in `C++` that supports the `enqueue` and `dequeue` operations. However, I have taken the liberty to write such a data structure for you. To use it, you will need to finish the `Item` header and use that as the object type that you use your priority queue with. You don't need to create a separate `.cpp` file for your `Item` implementation. You can just put the new code in the `.h` directly. For help on what *templates* are in C++, you can use Google to find good tutorial information. I recommend `http://users.cis.fiu.edu/~weiss/Deltoid/vcstl/templates#T2`. You can test that all of this code setup works on the simple test file that I provided.

Once you have done that, you should write a program that generates input files, so that you don't have to keep writing new ones to test your program, and you can see how well your eventual algorithm runs. This program should be able to take an input number n and W and generate a file following the above format. For item names, I would recommend `a`, `b`, etc. For larger input files, you may instead revert to using numbers as the item names, such as `01123` and `01124`, etc. For weights, choose a random integer that ranges from 1 to $3n$. Values can be chosen as you like, but should have a decent range so that they are "interesting".

Then, write the main program that will perform the following steps:

1. Create an item object that stores the name, weight, profit, and ratio.
2. Process the input file into your priority queue.
3. Consider each item in max ratio order, using the greedy algorithm.
4. Stop when you have considered all items.

Program Output Required. Run your program that builds text files for n (the number of items) being 10, 20, 100, 1000, and 10000 that follow the format above. Make sure your program works for these cases. I will create my own input files to test on your code to make sure it works. This is the output format that I need for your programs (without the `<` and `>` symbols):

```
<number-of-items-in-solution>
<total-weight-of-items-in-solution>
<total-profit-of-items-in-solution>

Items in the Solution:
<item-1-name> <item-1-pesos> <item-1-weight>
<item-2-name> <item-2-pesos> <item-2-weight>
....
```

What To Turn In. All things are printed. Digital must be emailed to me also.

- Your implementation of `item.h`.
- Program to generate text files.
- Greedy algorithm program for 0-1 knapsack (digital).
- Instructions for your project—format, how to run, etc.—if needed. (digital)
- Sample runs for 10 and 20.