Competitive Programming

Lesson 5 – Greedy + Ad Hoc

Written by Jason Sun

and Puneet Bhat

1

Greedy

Choosing the best possible choice at every state to lead you to the answer.

Definition

“Greedy algorithms are a class of algorithms that make locally optimal choices at each step with the hope of finding a global optimum solution.”

https://www.geeksforgeeks.org/greedy-algorithms/

Speed?

Since you take choices at face value instead of considering the various possibilities and other states, this usually makes greedy algorithms really fast.

Tips

Since greedy is more about **a way of thinking** about a problem, and less about just knowing some knowledge to solve a problem, it makes it hard to learn. Aside from **practice**, there are some tips to keep in mind.

1

Goal

A problem is usually greedy if it wants the **best** answer (eg. min or max).

2

Problem

If it can be broken down into **subproblems** and making locally optimal choices leads to the answer, then it can be solved with greedy.

3

Solving

To solve, usually try to make as many **observations** as possible, and explore various types of locally optimal choices to check if any are **globally** optimal.

2

Ad Hoc

A problem that there is no specific technique for. Thus it requires a unique solution made and tailored for the exact problem.

Example

Problem

You were given the faces of an unsolved rubik's cube and needed to output the moves to solve it.

Type

This is ad hoc (assuming that solving a rubik’s cube isn’t common knowledge) because you will have to figure out the steps on solving a rubik's cube, and that isn’t at all related to any competitive programming topics.

Definitions

Think of these two terms as opposites, which makes both easier to understand, given the contrast.

Template

Problems that are significantly **easier** if you already have some similar **knowledge**.

For example, given a list of 1 million numbers, answer 1 million queries that ask for the sum of a range in the list of numbers. Well, obviously the solution is PSA, so all you do is implement a PSA and you've solved the problem. Simple and easy.

Ad Hoc

Problems that can **not** be made easier with any **previous** knowledge.

It can be trained for instead by **intuition** and **practicing** skills instead of learning knowledge.

Skills:

- Making better observations
- Learning how to make notes
- Efficient testing of various approaches

3

Examples

Some applications to exercise what you’ve learnt.

#1 – Coin Change

Given a **denomination** of coins with values {1, 5, 10} (in cents), determine the **minimum** number of coins needed to make the change for a **certain amount** of money. All input will be in cents.

What do you think this is?

Greedy **Ad Hoc** **Neither**

Given your intuition and what you've learnt, how would you approach solving this problem?

Sample Input
47
Sample Output
7
Explanation
10 + 10 + 10 + 10 + 5 + 1 + 1

#1 – Solution

Problem Type: Greedy

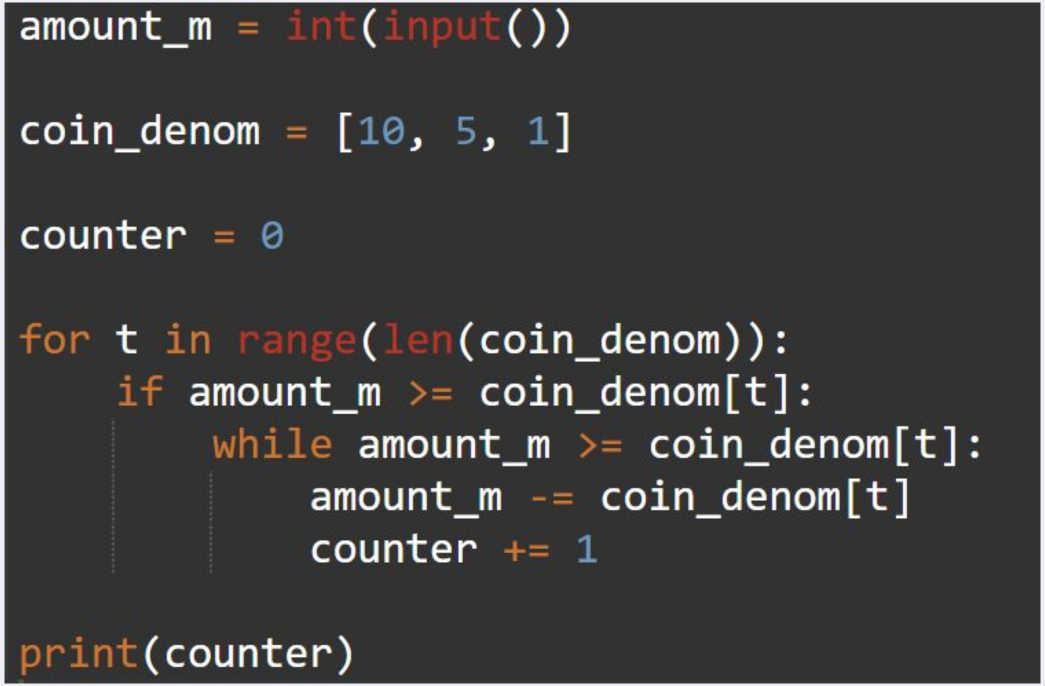
Explanation:

- We want to use the **minimum** number of coins possible to come up with a certain amount of money
- To do this, we must use a greedy algorithm to solve the problem piece-by-piece

Solution:

- We need to continuously subtract the **largest number** that isn't greater than our current **amount**, and keep track of the amount of times we do this
- Since 1 is a factor of 5 and 5 is a **factor** of 10, this means that it is always **worth it** to use **larger** numbers as they save you from using repeated smaller numbers to take away the **same** amount. For example, three 10s make 30, whereas you would need six 5s to make the same amount.

#1 – Sample Code



```python
amount_m = int(input())

coin_denom = [10, 5, 1]

counter = 0

for t in range(len(coin_denom)):
    if amount_m >= coin_denom[t]:
        while amount_m >= coin_denom[t]:
            amount_m -= coin_denom[t]
            counter += 1

print(counter)
```

**Python Solution by Puneet Bhat*

#2 – Odd Appearances

Given a natural number **N** (< 10^6) and **N** integers, determine the amount of integers that appear an odd number of times. After, print each number that appears an odd number of times, as well as how many times it has appeared from **least** to **most** appearances.

It must be noted that all numbers that appear an odd number of times will have unique numbers of appearances.

What do you think this is?
Greedy **Ad Hoc** **Neither**

Given your intuition and what you've learnt, how would you approach solving this problem?

Sample Input:

```
7
1 3 3 7 7 7 2 2
```

Sample Output:

```
2
1 1
7 3
```

Explanation

The two numbers 1 and 7 both appear an odd number of times, whereas 3 and 2 appear an even number of times. As such, 1 (appeared 1 time) and 7 (appeared 3 times) are outputted.

#2 – Solution

Problem Type: Ad Hoc

Explanation:

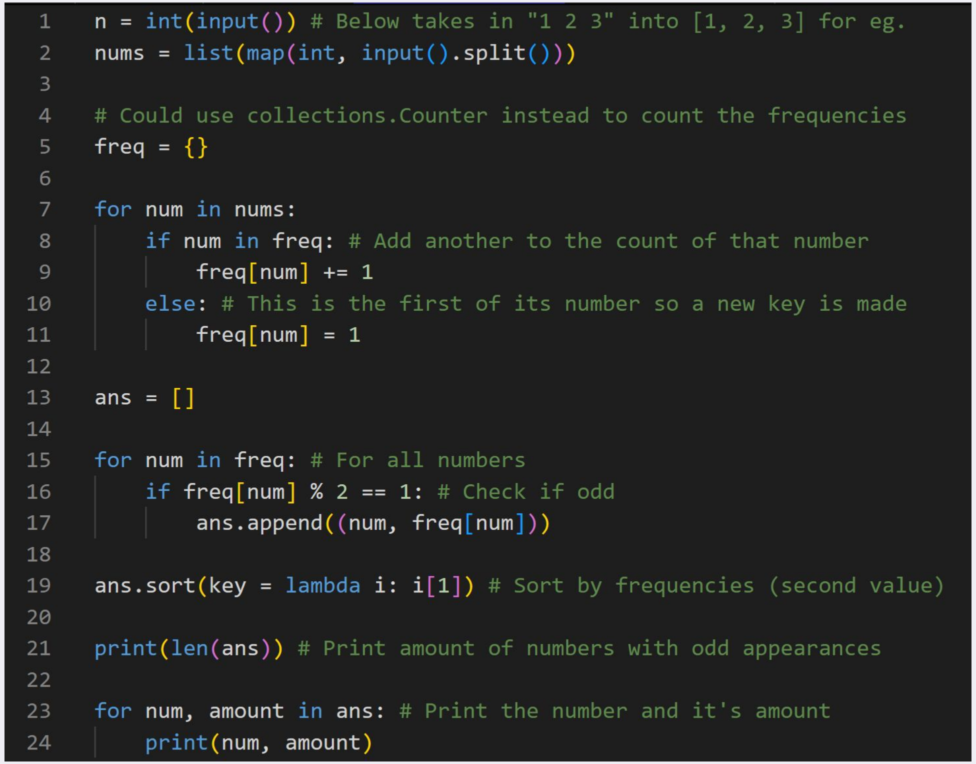
- We are trying to find all numbers that have odd frequencies in our given list and output them as mentioned in the problem

Solution:

- In this problem, it is ideal to use a **map** to keep track of the **frequencies** of each number in the list
- We then **loop** through each number and it's frequency in the map, storing the numbers that have an odd frequency in a **list**
- Then we sort the list in **ascending order** based on frequencies and print out each number, in order, in the created list

Note: *Sorting the list can be done by using another dictionary, or by simply using a key.*

#2 – Sample Code



```python
n = int(input()) # Below takes in "1 2 3" into [1, 2, 3] for eg.
nums = list(map(int, input().split()))

# Could use collections.Counter instead to count the frequencies
freq = {}

for num in nums:
    if num in freq: # Add another to the count of that number
        freq[num] += 1
    else: # This is the first of its number so a new key is made
        freq[num] = 1

ans = []

for num in freq: # For all numbers
    if freq[num] % 2 == 1: # Check if odd
        ans.append((num, freq[num]))

ans.sort(key = lambda i: i[1]) # Sort by frequencies (second value)

print(len(ans)) # Print amount of numbers with odd appearances

for num, amount in ans: # Print the number and it's amount
    print(num, amount)
```

**Python Solution by Jason Sun*

^.^

Practice

If you want to truly improve, practice! Do the homework as well as some DMOJ questions outside of class. It can be a fun hobby!

Homework

Junior

1) Distance – https://dmoj.ca/problem/tle17c6p1
2) Home Row – https://dmoj.ca/problem/dmopc20c5p1
3) CCC '13 J4 – Time on task – https://dmoj.ca/problem/ccc13j4
4) CCC ‘16 S2 – Tandem Bicycle – https://dmoj.ca/problem/ccc16s2

Senior

1) Carol's Carnivorous Plants – https://dmoj.ca/problem/thicc17p1
2) Safe from Rooks – https://dmoj.ca/problem/dwite10c5p2
3) A Typical Codeforces Problem – https://dmoj.ca/problem/tcc1p1

Thanks!

Do you have any questions?

https://github.com/CryoJS/DSA-Handbook