●●●● Not Frequent 0/7

Introduction to Greedy Algorithms

Authors: Darren Yao, Benjamin Qi
Contributor: Ryan Chou

Problems that can be solved by selecting the choice that seems to be the best at the moment at every step.

Language: Python Edit This Page

Prerequisites

- Bronze - Ad Hoc Problems
- Bronze - Introduction to Data Structures

TABLE OF CONTENTS

Greedy Algorithms

Some USACO Bronze problems that appear to be ad hoc can actually be solved using **greedy** algorithms. This idea will be covered in a future **module**, but we'll introduce the general mindset in this section.

RESOURCES

CPH	★ **6.1 - Coin Problem**	other examples are outside scope of bronze

Warning!
True **"greedy"** problems start to show up in silver, though the greedy mindset can be very helpful for bronze problems.

From the above:

> A **greedy** algorithm constructs a solution to the problem by always making a choice that looks the best at the moment. A greedy algorithm never takes back its choices, but directly constructs the final solution. For this reason, greedy algorithms are usually very efficient.

Greedy does not refer to a single algorithm, but rather a way of thinking that is applied to problems; there's no one way to do greedy algorithms. Hence, we use a selection of well-known examples to help you understand the greedy paradigm.

Mad Scientist

Mad Scientist

Bronze - Easy

Focus Problem – try your best to solve this problem before continuing!

Explanation

In this problem, the correct greedy solution is to continually flip the longest possible ranges of mismatching cows.

Mad Scientist has an excellent **editorial** with a video solution and intuitive proof.

It is highly recommended you read it to gain a better understanding of the greedy algorithm.

Implementation

Time Complexity: $\mathcal{O}(N)$

Copy PYTHON

```python
with open("breedflip.in") as f:
    cow_num = int(f.readline())
    a = f.readline().strip()
    b = f.readline().strip()

# diff[i] is true if the cows differ at the ith position
# notice that there's an extra false at the beginning
diff = [False for _ in range(cow_num + 1)]
for i in range(cow_num):
    diff[i + 1] = a[i] != b[i]

"""
count the number of times [false, true] occurs in diff
this is equivalent to the number of continous segments of trues
that extra false at the beginning comes in handy here,
since if we didn't have it we've have to have an edge case for
the first segment that doesn't have a preceding false
"""
min_flips = 0
for i in range(cow_num):
    if not diff[i] and diff[i + 1]:
        min_flips += 1

print(min_flips, file=open("breedflip.out", "w"))
```

Note that not all greedy problems necessarily require mathematical proofs of correctness. It is often sufficient to intuitively convince yourself your algorithm is correct.

Sometimes, if the algorithm is easy enough to implement, you don't even need to convince yourself it's correct; just code it and see if it passes. Competitive programmers refer to this as "Proof by AC," or "Proof by Accepted."

Problems