Let’s Try Solving a Problem!

- Flip A: https://codeforces.com/gym/444888/problem/A

Which Segment to Flip?

- Want to flip as many 1's as possible, as few 0's as possible
- Intuition says: flip longest segment containing only 1's
  - But it's wrong: 1 1 1 1 1 1 **0** 1 1 1 1 1 1 **0** 1 1 1 1 1

Trying to Fix It

- Find "lonely" 0's and flip it and the max chains of 1's to the left/right
  - Still wrong for 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1
- Flip chains of 1's with at most two 0's in them, at most three 0's in them, etc.
  - Can extend above example by adding more 0 1 1 1 1
  - Like: 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1
  - Sometimes not beneficial to include any 0's: 1 1 1 1 1 1 0 0 0 1
  - "flip chains with at most k 0's": how to decide magic number k?

How About a Different Strategy

- Start at some random 1, keep extending left/right if the "balance of 0's and 1s" is good
  - If the chain length of 1's to the left/right beyond the chain of 0's is $\geq$ the chain length of 0's
- This is getting harder to code
- It sounds right, but if you get Wrong Answer*, you start worrying:
  - Is my idea correct, but the code is just buggy?
  - Is there some test case where the strategy doesn't work, no matter if the code is perfect?

How About a Different Strategy

- In fact, the idea is wrong, can get unlucky at start point surrounded by lots of 0's:

  1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1

- How about… repeat multiple times and take the best result?
  - Nope: 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 1
  - Strategy fails no matter where you start
  - Need to allow losing more 0's than 1's gained in order to capture multiple separate long chains of 1's

How About a Different Strategy

- You can invent a strategy that deals with this case and works for all previous examples
- You may find another test case that breaks your solution, then come up with an even trickier strategy
- You may find another test case that breaks your solution, then come up with an even trickier strategy
- You may find another test case that breaks your solution, then come up with an even trickier strategy

But At Some Point, You Need to Wonder...

- How can I be sure there isn't another, trickier test case that breaks my trickier solution? When will this end?
- Trickier and trickier strategies are harder and harder to code
  - Is my idea correct, but the code is just buggy?
  - Is there some test case where the strategy doesn't work, no matter if the code is perfect?
- Can the problem even be solved by some kind of strategy like this? Or do I need some completely different approach?
  - In fact, you do!

An Easy Solution

- Just try all possible flips!
  - Try all possible starting points where to flip
  - For each possible start point, try all possible ending points
- For each possibility, count the number of 1's obtained by considering that possibility
- Keep the minimum over all possibilities

An Easy Solution

```
best = n
for i in range(n):
    for j in range(i, n):
        best = min(best, score_if_flip(i, j))
```

- Of course, need to write `score_if_flip`, but that's much easier

The Key Idea

- **Trying all possible answers is always going to yield a correct solution!**
- You have a computer that can do repetitive operations fast and doesn't get tired or bored
- Let it do all the hard work
- If it's hard to invent a smart way to decide what the answer is, then *just don't be smart*!
- Instead, use **brute force**
  - AKA **trial-and-error** AKA **guess-and-check**

The Key Idea

Google

brute

adjective

adjective: **brute**

1. characterized by an absence of reasoning or intelligence.
   "a brute struggle for social superiority"
   - merely physical.
     "we achieve little by brute force"
     synonyms: physical, crude, fleshly, bodily, violent
     "by sheer brute strength he almost reached the top of the incline"
   - fundamental, inescapable, and unpleasant.
     "the brute necessities of basic subsistence"

The Key Idea (When You Don't Have the *Key*)

- Try 0000
- Try 0001
- Try 0002
- Try 0003
- …

The Same Strategy Works For All Problems!

- How many multiples of 7 between 777,777 and 7,777,777?
  - Just go through all numbers! Increment a counter every time we see a number divisible by 7

The Same Strategy Works For All Problems!

```
for i in range(a, b + 1):
    if i % 7 == 0:
        count += 1
```

The Same Strategy Works For All Problems!

- Given a grid of $n \times m$ integers (may be negative), draw a rectangle so that the sum of the numbers inside the rectangle is maximized
  - Rectangle must have sides parallel/perpendicular to the lines of the grid

sum = 8

3	-1	4	-1
-5	9	-2	6
5	-3	5	-8
-9	7	-9	3
2	-3	8	-4

sum = 0

The Same Strategy Works For All Problems!

- Given a grid of $n \times m$ integers (may be negative), draw a rectangle so that the sum of the numbers inside the rectangle is maximized
    - Just try all possible rectangles! Keep track of the maximum sum seen so far

The Same Strategy Works For All Problems!

```
for start_r in range(num_rows):
  for start_c in range(num_cols):
    for end_r in range(start_r, num_rows):
      for end_c in range(start_c, num_cols):
        best = max(best, score(start_r, end_r, start_c, end_c))
```

For all upper-left corners

For all lower-right corners

For all rectangles

The General Strategy...

- Just try all the possible candidate answers, do some appropriate action on each one (if valid)
  - Set a Boolean flag for **decision** (output is yes/no) problems
  - Keep the minimum/maximum for **optimization** problems
  - Increment a counter for **counting** problems
  - Print it or put it in a list when the problem is to simply literally enumerate the possibilities

The General Strategy...

```
for candidate in possibilities:
    if valid(candidate):
        # choose the appropriate action
        exists = True
        least = min(least, score(candidate))
        most = max(most, score(candidate))
        count += 1
        print(candidate)
```

Some Details to Consider

- Usually, candidate answers are not nicely given to us in a list, so we need to *invent* a way to go through all the possibilities
  - Count multiples of 7 in a range: "for all integers" or "for all multiples of 7"
  - Flip A: "for all positions to flip" → "for all start and end indices"
  - Grid rectangle sum:
    - "for all rectangles" → "for all upper-left and lower-right corners"
    - "for all corners" → "for all row, column pairs"

Some Details to Consider

- How to determine if a candidate is valid
  - Optional: only if the way to go through all possibilities includes non-possibilities
- How to score a candidate
- How to update the final answer after considering one candidate

But With Enough Practice, This Is Really Easy

- Little special thinking needed for the problem – same strategy works for all problems
- No need to worry about correctness – we're sure about it because we tried all possibilities

Allowed to Go Through More Possibilities Than Necessary, Even Clearly Invalid Ones

- In "count multiples of 7," we went through all integers in range, even though the real possibilities are only the multiples of 7
- Could have gone through only multiples, but requires more care
  - Skip by 7 at each iteration
  - Can't use the given number as start of range to consider, need to start at a multiple of 7
- Sometimes, it's easier and acceptable to consider a bigger set of candidates, as long as it contains all the possible answers, and then just *filter* `if valid(candidate)`

Another Way to Look At It: Complete Search

“Normal” Search

```
for x in input_list:
  if x == target:
    exists = True
```

What We’re Doing

```
for x in candidates:
  if valid(x):
    exists = True
    least = min(least, score(x))
    most = max(most, score(x))
    count += 1
```

Another Way to Look At It: Complete Search

- Call the set of possibilities considered by a solution the **search space**
- Usually there is one "obvious" search space which clearly contains all the possible answers and only the possible answers, no filtering required
  - Examples: flip A, grid rectangle sum

Another Way to Look At It: Complete Search

- Sometimes, multiple possible search spaces can work, and when it's tricky to include some problem rules directly into the search space, we consider a bigger search space and filter it
  - Example: count multiples
  - Usually, it's easier to test if a candidate follows the rules than to generate candidates that follow the rules, especially if there are many rules and they are complicated, hence "guess-and-check"

Another Way to Look At It: Complete Search

- The choice of search space affects the speed of the solution
  - Smaller search space → faster
    - Example: count multiples with skip by 7 is 7 times faster
  - Making the search space smaller requires additional information/structure/insight/intelligence
  - In the absence of such information, forced to consider bigger search spaces

What's Wrong with This Code? (Count Multiples of 7 in Range)

```python
def nearest_multiple_greater_than(x):
    for i in range(x + 1, x + 8):
        if i % 7 == 0:
            return i

# For C++ programmers, the 7 at the end is like i += 7
for i in range(nearest_multiple_greater_than(a), b + 1, 7):
    count += 1
```

What's Wrong with This Code? (Count Multiples of 7 in Range)

- `a` could be a multiple of 7, we should start at the nearest multiple greater than *or equal* to `a`

What's Wrong with This Code? (Flip A)

```
best = n
for j in range(n):
    best = min(best, score_if_flip(0, j))
```

What's Wrong with This Code? (Flip A)

```
best = n
for j in range(n):
    best = min(best, score_if_flip(0, j))
```

- Only considers flipping all *prefixes* of the input

Complete Search

```
# What's wrong with this?
for x in input_list[1:]:
    if x == target:
        exists = True
```

- We must make sure to check *everything*
- Otherwise, what we didn't check:
  - Might be the only valid answer (for decision problems)
  - Might be the best answer (for optimization problems)
  - Might need to be counted (for counting problems)

Practice Problems

- https://progvar.fun/problemsets/complete-search-iterative

The Same Strategy Works For All Problems!

- How many multiples of 7 between 777,777 and 7,777,777,777?
  - Plain brute force too slow
  - “Go only through multiples” solution kinda works
- How many multiples of 7 between 777,777 and 7,777,777,777,777?
  - Now, not even “go only through multiples” solution works

The Same Strategy Works For All Problems!

- Run this to generate input for Flip A
- Experiment: decrease `100000` and find out at what order of magnitude the solution switches from fast to slow

```
from random import choice
N = 100000
print(N)
print(''.join(choice('01') for i in range(N)))
```

The Same Strategy Works For All Problems!

- Run this to generate input for the grid rectangle sum problem
  - Experiment: decrease 1000 and find out at what order of magnitude it switches from fast to slow

```python
from random import randint

print(1000, 1000)
for i in range(1000):
    for j in range(1000):
        print(randint(-50, 50), end=' ')
    print()
```

Unfortunately, It's Too Slow!

- By not using any intelligence, we pay a price: inefficiency
- Brute force always gives the correct answer…if we are willing to wait

組み合わせの数え方

But I Thought Computers Were Fast...

- Yes, but they are not magic
- They are physical devices too and must obey the laws of physics
- There is a limit to how fast they can be
- However, we won't just give up or wait for better computers to arrive
- We can make the same computer solve the same problems faster, just by being smart again *but doing it properly this time*

Sometimes, We Do Need to Be Smart

- In this camp, we will learn some ways to be smart: how to get fast solutions *and* still be sure they are correct
  - First, we must learn what it even means for a solution to be fast
- However, with the help of a computer, sometimes, it's ok to be dumb
  - And we will learn exactly when it's acceptable to be dumb and when it's required to be smart

Challenge

- Solve Flip A when $n = 10^4$
- Solve Flip A when $n = 10^5$
- There are ways to make it faster!
- See if you can solve it after this camp!