Flip Bit

To learn how to use this template, check out the course "Data Structures and Algorithms in Python".

How to run the code and save your work

The recommended way to run this notebook is to click the "Run" button at the top of this page, and select "Run on Binder". This will run the notebook on mybinder.org, a free online service for running Jupyter notebooks.

This tutorial is an executable Jupyter notebook. You can *run* this tutorial and experiment with the code examples in a couple of ways: *using free online resources* (recommended) or *on your computer*.

Option 1: Running using free online resources (1-click, recommended)

The easiest way to start executing the code is to click the **Run** button at the top of this page and select **Run on Binder**. You can also select "Run on Colab" or "Run on Kaggle", but you'll need to create an account on Google Colab or Kaggle to use these platforms.

Option 2: Running on your computer locally

To run the code on your computer locally, you'll need to set up Python, download the notebook and install the required libraries. We recommend using the Conda distribution of Python. Click the **Run** button at the top of this page, select the **Run Locally** option, and follow the instructions.

Saving your work

Before staring the assignment, let's save a snapshot of the assignment to your Jovian profile, so that you can access it later, and continue your work.

```
project_name = "Flip-bits-Mansi"
```

```
!pip install jovian --upgrade --quiet
```

```
import jovian
```

```
jovian.commit(project=project_name)
```

```
[jovian] Creating a new project "mansishah9865/Flip-bits-Mansi"
[jovian] Committed successfully! https://jovian.ai/mansishah9865/flip-bits-mansi
```

```
'https://jovian.ai/mansishah9865/flip-bits-mansi'
```

Problem Statement

> You are given an array of integers ARR[] of size N consisting of zeros and ones. You have to select a subset and flip bits of that subset. You have to return the count of maximum one's that you can obtain by flipping chosen sub-array at most once. A flip operation is one in which you turn 1 into 0 and 0 into 1.

Source: coding ninjas

The Method

Here's the systematic strategy we'll apply for solving problems:

1. State the problem clearly. Identify the input & output formats.
2. Come up with some example inputs & outputs. Try to cover all edge cases.
3. Come up with a correct solution for the problem. State it in plain English.
4. Implement the solution and test it using example inputs. Fix bugs, if any.
5. Analyze the algorithm's complexity and identify inefficiencies, if any.
6. Apply the right technique to overcome the inefficiency. Repeat steps 3 to 6.

This approach is explained in detail in Lesson 1 of the course. Let's apply this approach step-by-step.

Solution

1. State the problem clearly. Identify the input & output formats.

While this problem is stated clearly enough, it's always useful to try and express in your own words, in a way that makes it most clear for you.

Problem

> we have given an array of elements 0 and 1 and we have to find maximum length of Sub array contains all elements as 1 after fliping the array

Input

1. **[1,0,0,0]**

Output

1. **4**

Based on the above, we can now create a signature of our function:

```
def flipbits(arr):
    pass
```

Save and upload your work before continuing.

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Updating notebook "mansishah9865/flip-bits-mansi" on https://jovian.ai
```

```
[jovian] Committed successfully! https://jovian.ai/mansishah9865/flip-bits-mansi
```

```
'https://jovian.ai/mansishah9865/flip-bits-mansi'
```

2. Come up with some example inputs & outputs. Try to cover all edge cases.

Our function should be able to handle any set of valid inputs we pass into it. Here's a list of some possible variations we might encounter:

1. **empty arr**
2. **all one**
3. **all zero**

(add more if required)

We'll express our test cases as dictionaries, to test them easily. Each dictionary will contain 2 keys: `input` (a dictionary itself containing one key for each argument to the function and `output` (the expected result from the function).

```
test = {
    'input': {
        "arr" : [1,0,0,1,0]

    },
    'output': 4
}
```

```

```

```

```

Create one test case for each of the scenarios listed above. We'll store our test cases in an array called `tests`.

```
tests = []
```

```
tests.append(test)
```

```
tests.append({
    'input': {
        "arr" : []
    },
    'output': 0
})
```

```
tests.append({
    'input': {
        "arr" : [1,1,1,1]
    },
```

```
    'output': 4
})
```

```
tests.append({
    'input': {
        "arr" : [0,0,0,0,0]
    },
    'output': 5
})
```

3. Come up with a correct solution for the problem. State it in plain English.

Our first goal should always be to come up with a *correct* solution to the problem, which may not necessarily be the most *efficient* solution. Come with a correct solution and explain it in simple words below:

1. We check for all sub arrays the total number of 1 we get in the whole array after flipping that particular sub array

Let's save and upload our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "mansishah9865/flip-bits-mansi" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/flip-bits-mansi
```

```
'https://jovian.ai/mansishah9865/flip-bits-mansi'
```

4. Implement the solution and test it using example inputs. Fix bugs, if any.

```
def flipbits(arr):

    l1 = arr[:]
    Max_count = 0
    count = 0
    for i in range(0,len(l1)):
        for j in range(i,len(l1)):
            count = 0
            l2 = l1[i:j+1]
            for k in l2:
                if k == 0:
                    k = 1
                    count += 1
                else:
                    k = 0
            for l in arr[j+1:]:
                if l == 1:
                    count += 1
            for kk in arr[0:i]:
                if kk == 1:
                    count += 1
            if Max_count < count:
```

```
                Max_count = count

    if sum(arr) > Max_count:
        return sum(arr)
    return Max_count
```

We can test the function by passing the input to it directly or by using the `evaluate_test_case` function from `jovian` .

```
from jovian.pythondsa import evaluate_test_case
```

```
evaluate_test_case(flipbits,test)
```

```
Input:
{'arr': [1, 0, 0, 1, 0]}

Expected Output:
4



Actual Output:
4

Execution Time:
0.034 ms

Test Result:
PASSED


(4, True, 0.034)
```

Evaluate your function against all the test cases together using the `evaluate_test_cases` (plural) function from `jovian` .

```
from jovian.pythondsa import evaluate_test_cases
```

```
evaluate_test_cases(flipbits,tests)
```

```
TEST CASE #0

Input:
{'arr': [1, 0, 0, 1, 0]}

Expected Output:
4


Actual Output:
4

Execution Time:
0.028 ms

Test Result:
PASSED


TEST CASE #1

Input:
{'arr': [1, 1, 0, 1, 0]}

Expected Output:
4


Actual Output:
4

Execution Time:
0.023 ms

Test Result:
PASSED


TEST CASE #2
```

Input:
{'arr': []}

Expected Output:
0

Actual Output:
0

Execution Time:
0.004 ms

Test Result:
PASSED

TEST CASE #3

Input:
{'arr': [1, 1, 1, 1]}

Expected Output:
4

Actual Output:
4

Execution Time:
0.017 ms

Test Result:
PASSED

TEST CASE #4

Input:
{'arr': [0, 0, 0, 0, 0]}

Expected Output:
5

```
Actual Output:
5

Execution Time:
0.022 ms

Test Result:
PASSED


SUMMARY

TOTAL: 5, PASSED: 5, FAILED: 0
```

```
[(4, True, 0.028),
 (4, True, 0.023),
 (0, True, 0.004),
 (4, True, 0.017),
 (5, True, 0.022)]
```

Verify that all the test cases were evaluated. We expect them all to fail, since we haven't implemented the function yet.

Let's save our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "mansishah9865/flip-bits-mansi" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/flip-bits-mansi
```

```
'https://jovian.ai/mansishah9865/flip-bits-mansi'
```

```

```

5. Analyze the algorithm's complexity and identify inefficiencies, if any.

```
time_complexity = "O(N3)"
```

```

```

```

```

```
jovian.commit()
```

```
[jovian] Updating notebook "mansishah9865/flip-bits-mansi" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/flip-bits-mansi
```

'https://jovian.ai/mansishah9865/flip-bits-mansi'

6. Apply the right technique to overcome the inefficiency. Repeat steps 3 to 6.

```
#Implementing kadane
```

```

```

```
jovian.commit()
```

```
[jovian] Updating notebook "mansishah9865/flip-bits-mansi" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/flip-bits-mansi
```

'https://jovian.ai/mansishah9865/flip-bits-mansi'

7. Come up with a correct solution for the problem. State it in plain English.

Come with the optimized correct solution and explain it in simple words below:

By using kadane's algorithm we can reduce time complexity upto O(N)

Let's save and upload our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "mansishah9865/flip-bits-mansi" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/mansishah9865/flip-bits-mansi
```

'https://jovian.ai/mansishah9865/flip-bits-mansi'

```
def flipbits_new(arr):
    Max_count = 0
    cur_count = 0

    for i in range(len(arr)):
        if arr[i] == 0:
            cur_count = cur_count + 1
        if arr[i] == 1:
            cur_count = cur_count -1
        if cur_count > Max_count:
            Max_count = cur_count
        if cur_count < 0:
            cur_count = 0
    return sum(arr) + Max_count
```

```

```

8. Implement the solution and test it using example inputs. Fix bugs, if any.

```
from jovian.pythondsa import evaluate_test_cases
```

```
evaluate_test_cases(flipbits_new,tests)
```

```
TEST CASE #0

Input:
{'arr': [1, 0, 0, 1, 0]}

Expected Output:
4


Actual Output:
4

Execution Time:
0.007 ms

Test Result:
PASSED


TEST CASE #1

Input:
{'arr': [1, 1, 0, 1, 0]}

Expected Output:
4


Actual Output:
4

Execution Time:
0.007 ms
```

Test Result:
PASSED

TEST CASE #2

Input:
{'arr': []}

Expected Output:
0

Actual Output:
0

Execution Time:
0.003 ms

Test Result:
PASSED

TEST CASE #3

Input:
{'arr': [1, 1, 1, 1]}

Expected Output:
4

Actual Output:
4

Execution Time:
0.005 ms

Test Result:
PASSED

TEST CASE #4

```
Input:
{'arr': [0, 0, 0, 0, 0]}

Expected Output:
5


Actual Output:
5

Execution Time:
0.005 ms

Test Result:
PASSED


SUMMARY

TOTAL: 5, PASSED: 5, FAILED: 0
[(4, True, 0.007),
 (4, True, 0.007),
 (0, True, 0.003),
 (4, True, 0.005),
 (5, True, 0.005)]
```

9. Analyze the algorithm's complexity and identify inefficiencies, if any.

```
time_complexity = "O(N)"
```

If you found the problem on an external platform, you can make a submission to test your solution.

Share your approach and start a discussion on the Jovian forum: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/78

```
jovian.commit()
```

```
```