Complexity

Student Evaluation of Tutors

UQ and ITEE takes pride in the quality of its teaching staff and encourages you to provide constructive feedback of the course tutors through the SETutor survey process.

Take the first five minutes of this tutorial to complete the survey for the tutor(s) with whom you have had the most contact this semester. The survey links are available on BlackBoard, in the Administrative Items folder under Learning Resources.

Uniqify

For this task, we want to write a function which removes duplicates from a list, analyse the computational complexity of the function, and then improve it.

1. Write a function `uniqify(items)` which takes a list, and returns a list containing each element of `items` without any duplicates of the elements in `items`. For example:

   ```
   >>> uniqify([8, 3, 5, 8, 9, 4, 5, 2, 3, 5])
   [8, 3, 5, 9, 4, 2]
   ```

   Now, suppose that `items` has length `n`. Approximate the number of 'steps' of computation the function requires to finish (in terms of `n`). Express your final answer as a computational complexity class (constant / logarithmic / linear / quadratic / exponential).

2. Discuss how the complexity could be improved if the `items` list was sorted to begin with. Write a function `uniqify2` which removes duplicates from a list, by first sorting the list.

   Note: the order of elements in the output list is unimportant, as long as each appears only once.

   Hint: `sorted(items)` will give a copy of `items` in sorted order, without modifying the original list. It runs in $O(nlog(n))$ time.

   ```
   >>> uniqify2([8, 3, 5, 8, 9, 4, 5, 2, 3, 5])
   [2, 3, 4, 5, 8, 9]
   ```

   Then, repeat the same analysis as above to determine the computational complexity of `uniqify2`, in terms of the length of the input list `n`.

3. In this course, we have already encountered a data type which contains a notion of *uniqueness*. Discuss how to use this to write a function `uniqify3` which does the same task, with a better time complexity. Then, write such a function, and determine its complexity.

4. Is it possible to solve this task with a faster time complexity than in `uniqify3`?

Justify your answer.

The following code snippet will run the three `uniqify` functions multiple times on a large input, and measure the running time. Copy and paste the code into the source file containing the three function definitions, and run it. It may take a few minutes to finish, so move on to the next task while you are waiting.

```
import random
import timeit

for func, number in [('uniqify3', 1000), ('uniqify2', 1000), ('uniqify', 1)]:
    print('\n' + func + ':')
    old = None
    for N in [1000, 2000, 4000, 8000, 16000, 32000, 64000] :
        time = min(timeit.repeat(func + '(range(N))',
                    'from __main__ import N,' + func,
                    number=number)) * 1000 / number
        print ('n = {:>5}: {:.4f} ms'.format(N, time) +
               ('' if old is None else ' = {:.2f} times slower'.format(time/old)))
        old = time
```

Aside: Sets

If the data in our collection must be unique, then lists are not the most optimal type to use. Python offers a data type called a set, where every element is unique. Sets behave similarly to dictionaries, but without having a value associated with each unique element.

Additional Examples

The next section of the tutorial contains two functions. For each function, using similar methods to the uniqify section, analyse the time complexity of the function by estimating the number of 'steps' in the code. Note that this code is not necessarily the optimal way to provide the functionality but is instead provided as practice for analysing complexity.

1. The `is_n_digits` function takes two integers, `x` and `n`, and returns `True` if `x` is an `n`-digit number. Analyse the time complexity of the `is_n_digits` function in terms of the size of the `n` parameter:

```
def is_n_digits(x, n):
    for guess in range(10**(n-1), 10**n):
        if guess == x:
            return True
    return False
```

2. The `search` function takes a list of `numbers`, **sorted in ascending order**, and a number, `x`, and returns `True` if `x` is an element of `numbers` and `False` otherwise. Analyse the time complexity of the search function in terms of the length of the list, `numbers`:

```
def search(numbers, x):
    if numbers == []:
        return False

    if len(numbers) == 1:
        return numbers[0] == x

    mid = len(numbers) // 2
    if x < numbers[mid]:
        return search(numbers[:mid], x)
    else:
        return search(numbers[mid:], x)
```

Past Exam Practice

Attempt some of the past exam questions, accessible from the UQ Library website. If there is time, the tutors will suggest questions to attempt. The tutorial in week 13 will go over some questions and solutions for a past exam.