assignment_2

July 24, 2024

1 Practice Interview

1.1 Objective

The partner assignment aims to provide participants with the opportunity to practice coding in an interview context. You will analyze your partner's Assignment 1. Moreover, code reviews are common practice in a software development team. This assignment should give you a taste of the code review process.

1.2 Group Size

Each group should have 2 people. You will be assigned a partner

1.3 Part 1:

You and your partner must share each other's Assignment 1 submission.

1.4 Part 2:

Create a Jupyter Notebook, create 6 of the following headings, and complete the following for your partner's assignment 1:

- Paraphrase the problem in your own words.

Question 3 is about checking whether a given list of integers contains all of the possible integers within a given range. Specifically, the list contains a maximum of n number of integers, and the given range is 0 to n, and the list may contain duplicates.

Use cases may include checking whether a patient's hospital admission records contains reports on a daily basis, or whether a travel itinerary meets the required order of stops.

- Create 1 new example that demonstrates you understand the problem. Trace/walkthrough 1 example that your partner made and explain it.

[ ]:
```
lst_4 = [13,1,8,8,5,2,2,2]
max(lst_4)
```

[ ]:
```
13
```

Create a new example:

new_ex = [0,0,3,9,7]

The range to check is given by 0 and max(new_ex) which is 9

The function will check whether new_ex contains all of the integers given by list(range(0,9+1)), which are:

[0,1,2,3,4,5,6,7,8,9]

Note that alternatively, using list(range(0,9)) would work too, which gives only [0,1,2,3,4,5,6,7,8], because the integer 9 is already given by the list and needs not be checked.

The following elements are missing and should be returned by the function:

[1,2,4,5,6,8]

Partner's example:

lst_4 = [13,1,8,8,5,2,2,2]

The range to check is given by 0 and max(list_4) which is 13.

The function will check whether lst_4 contains all of the following integers:

[0,1,2,3,4,5,6,7,8,9,10,11,12,13]

By eyeballing, the following elements are missing, and should be returned by the function:

[0,3,4,6,7,9,10,11,12].

[ ]:
```
#Partner's solution:
def missing_num(nums: list):
    maximum = max(nums) # O(m)
    set_from_list = set(nums) #O(m)
    set_from_range = set(range(0,maximum)) #O(n)
    diff = set_from_range - set_from_list #O(n)
    if len(diff)>0: #O(n)
        return list(diff)
    else:
        return -1
```

[ ]:
```
missing_num([0,0,3,9,7])
```

[ ]:
```
[1, 2, 4, 5, 6, 8]
```

- Explain why their solution works in your own words.

The solution works using Python's built-in function that finds the difference of a set B from a given set A.

In this case, the given set A is the range of integers between 0 and n, which is called "set(range(0,max(nums)))" in the code, and the other set B is the set of integers provided, which is "set(nums)". When one subtracts set B from set A, Python returns those elements that are in set A but not in set B.

- Explain the problem's time and space complexity in your own words.

The time complexity of the solution is O(1), because the code contains a single step that operates on a list of n integers, but practically (or worse case) is O(n) because even with sets, Python is essentially performing operations on n integers on the backend.

The space complexity is O(2n), or more simply O(n), because the code needs to handle two sets of n integers at once.

- Critique your partner's solution, including explanation, and if there is anything that should be adjusted.

The solution is sensible and works well; a minor adjustment to the code is to sort the output list of missing numbers ascendingly, as illustrated in the original question, e.g. by using sorted(list(diff)).

Other than that, I have two suggestions. First, because numpy is implemented at a lower programming level compared with Python's standard set functions, using numpy, specifically np.setdiff1d(), might result in a faster execution. Second, the code may be made more flexible and with less lines by using a recursive function. This will include a base case for stopping where start > end of the given range, checking that "start" exists in the given list (if not, append "start" to the return list), and recurse at start + 1. This will however increase the time complexity to O(n), i.e. n number of steps during recursion, and worse case to O(n^2), with the backend list operations.

1.5 Part 3:

Please write a 200 word reflection documenting your process from assignment 1, and your presentation and review experience with your partner at the bottom of the Jupyter Notebook under a new heading "Reflection." Again, export this Notebook as pdf.

1.5.1 Reflection

Although I started looking at Q2 from a mathematical point of view, i.e. trying to exhaust the possible paths, I gradually realized that rather than using an algebraic formula that iterates over the possibilities, it would be easier to trace all the paths from nodes to nodes. At that point, I determined that the after defining a node object, the next step is to define a function to convert any list to a binary tree. Using a recursive defining structure, I followed the logic of preorder traversal (which can be used to copy trees). The difficult part, where I had to trial-and-error for quite many times, was to create one list for each path. The final solution I found was to create a shallow copy at each branching.

In terms of reviewing my partner's answer to A3, I very much appreciated the simplicity with which she approached the problem. The main algorithm is a subtraction of two sets and the code is very clean overall. Therefore, I thought in the direction of alternative approaches that could improve performance and flexibility. For performance, using numpy, which operates at a lower level than Python, might help to improve speed. For flexibility, the code could in fact be written recursively, by comparing each integer with a numpy array. While increasing time complexity, this approach may be useful in case further steps or requirements are to be included.

1.6 Evaluation Criteria

We are looking for the similar points as Assignment 1

- Problem is accurately stated

- New example is correct and easily understandable
- Correctness, time, and space complexity of the coding solution
- Clarity in explaining why the solution works, its time and space complexity
- Quality of critique of your partner's assignment, if necessary

1.7 Submission Information

Please review our Assignment Submission Guide for detailed instructions on how to format, branch, and submit your work. Following these guidelines is crucial for your submissions to be evaluated correctly.

1.7.1 Submission Parameters:

- Submission Due Date: `HH:MM AM/PM - DD/MM/YYYY`
- The branch name for your repo should be: `assignment-2`
- What to submit for this assignment:
  - This Jupyter Notebook (assignment_2.ipynb) should be populated and should be the only change in your pull request.
- What the pull request link should look like for this assignment: `https://github.com/<your_github_username>/algorithms_and_data_structures/pull/<pr_id>`
  - Open a private window in your browser. Copy and paste the link to your pull request into the address bar. Make sure you can see your pull request properly. This helps the technical facilitator and learning support staff review your submission easily.

Checklist: - [ ] Created a branch with the correct naming convention. - [ ] Ensured that the repository is public. - [ ] Reviewed the PR description guidelines and adhered to them. - [ ] Verify that the link is accessible in a private browser window.

If you encounter any difficulties or have questions, please don't hesitate to reach out to our team via our Slack at `#cohort-3-help`. Our Technical Facilitators and Learning Support staff are here to help you navigate any challenges.