Report

Task:

- Create a function that takes in a pattern (I made it take in a list of patterns), an arbitrary trace, and then return a txt file with the calculated acceptance ratio for each

Notes:

- I had to update the code. I tried my best to optimize it and have it be efficient and accurate. I tested it and have shown examples in the following report.

Link: functions.py

https://github.com/NiharikaAdari/dataminereseearch/blob/main/gem5_traces/gem5-snoop/functions.py

• Compute Pattern Ratios (writes result to folder)

Pass in a trace, and folder, and a list of patterns. Uses the 2 functions listed below (find acceptance ratios, which inturn uses the removepatternfromtrace function)

```
"""
Computes acceptance ratios of each pattern from a list of patterns on a trace and
writes results to an output folder.
    Parameters:
    - trace: pass in a trace file
    - output_folder: Path to the output folder to write results.
    - patterns: pass in the list of patterns to get ratios from
"""


def compute_pattern_ratios(trace, output_folder, patterns):
```

-

Find acceptance ratios (does the work)

Pass in a list of patterns. It will find the acceptance ratio for each one on the trace passed in.

```
"""
    Find patterns in the trace and calculate acceptance ratios for each pattern.

    Args:
    - trace (list): List of integers representing the trace.
    - patterns (list): A list of patterns (each a list of numbers) to find in the
trace.

    Returns:
    - pair_acceptance_ratios: List of tuples representing pairs and their
acceptance ratios.

"""

def find_acceptance_ratios(trace, patterns):
```

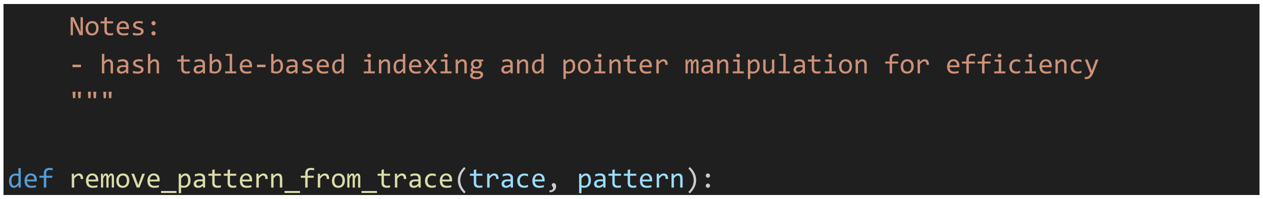
Remove pattern from trace

Has been updated to remove a pattern of any size from a trace.
Originally it was slow on a large trace but I made it more efficient by using a hash table indexing, and instead of using pop/modifying bucket size I used a pointer, so the performance is improved greatly.

```
    """
    Remove occurrences of a specified pattern from a trace list.

    Args:
    - trace (list): The trace list from which the pattern occurrences should be
removed.
    - pattern (list): The pattern (sequence of numbers) to be removed from the
trace.

    Returns:
    - list: A new trace list with the specified pattern occurrences removed.
```

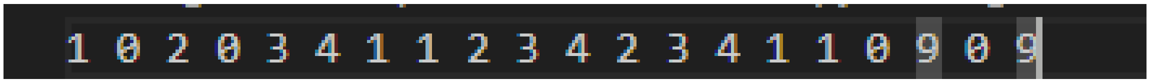


```
    Notes:
    - hash table-based indexing and pointer manipulation for efficiency
    """

def remove_pattern_from_trace(trace, pattern):
```

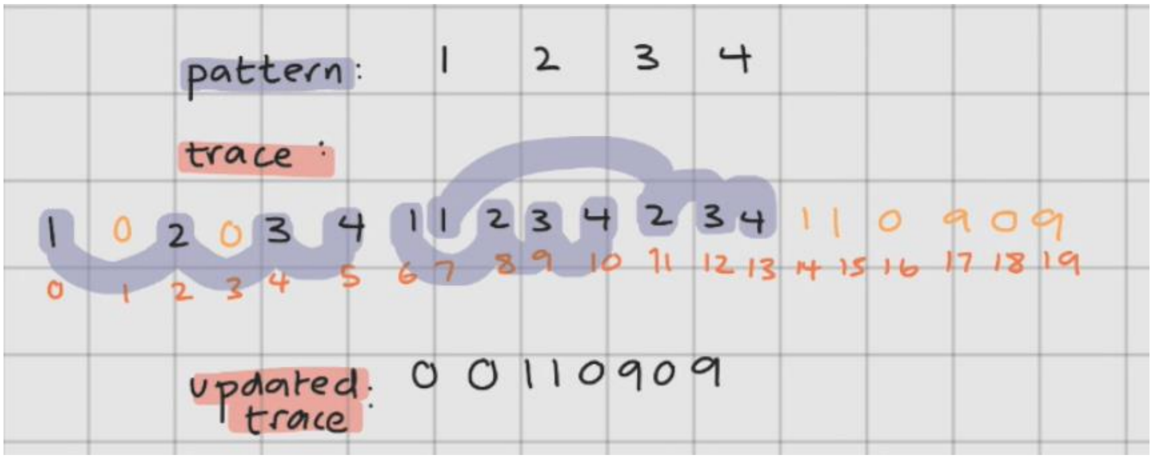
Algorithm Explanation with Examples:

Say I have this example trace:



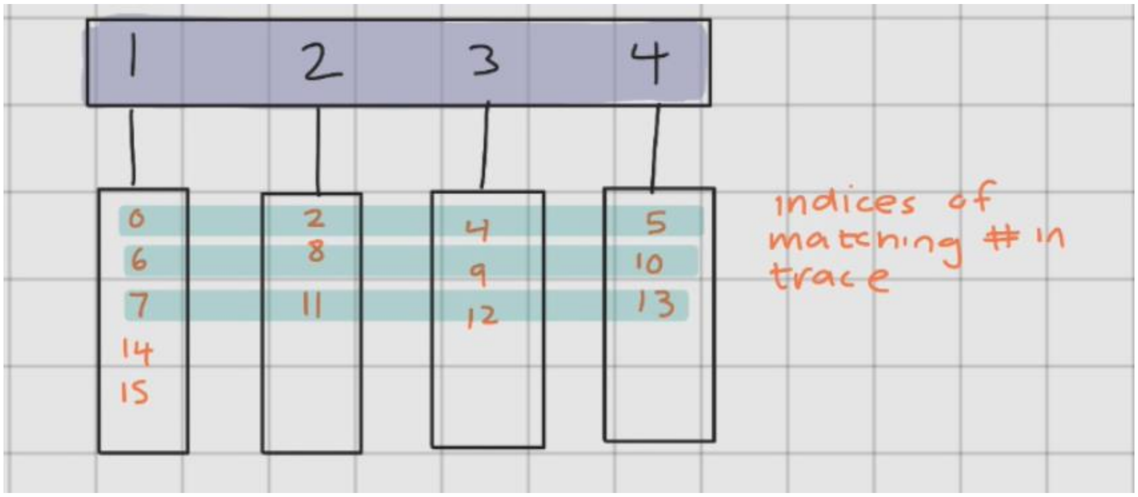
```
1 0 2 0 3 4 1 1 2 3 4 2 3 4 1 1 0 9 0 9
```

And I want to remove the pattern [1,2,3,4].

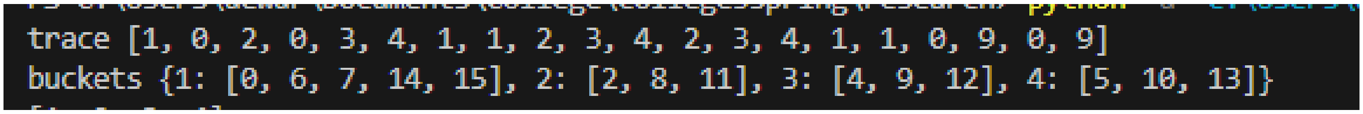


So first, it makes a hash table for the pattern itself. And each number is a key to a bucket, which stores the indices in the trace that matches.

Diagram:



Code (print statements to check)



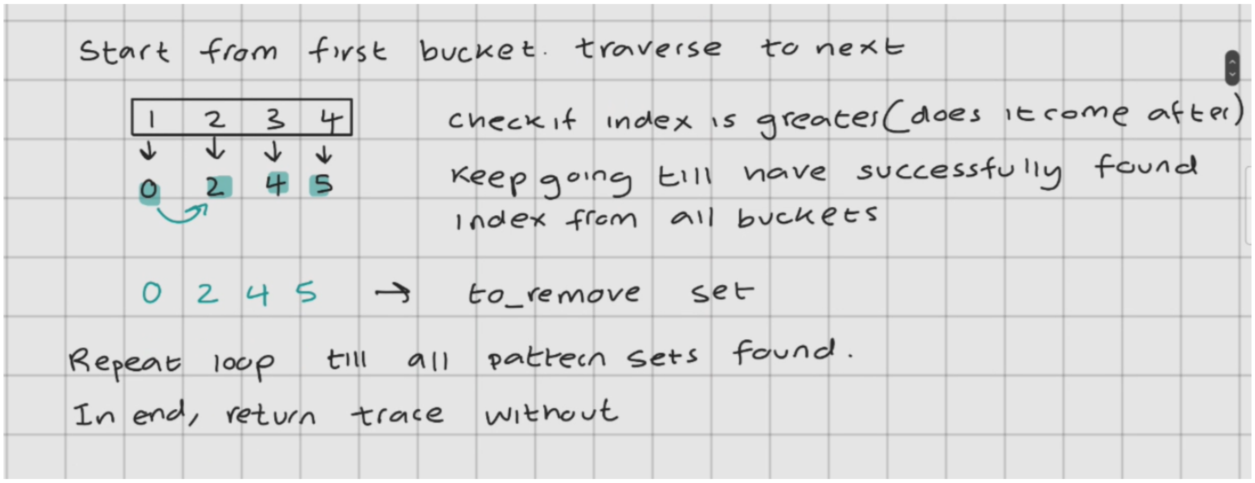
```
trace [1, 0, 2, 0, 3, 4, 1, 1, 2, 3, 4, 2, 3, 4, 1, 1, 0, 9, 0, 9]
buckets {1: [0, 6, 7, 14, 15], 2: [2, 8, 11], 3: [4, 9, 12], 4: [5, 10, 13]}
```

Then, starting from the first bucket, it picks an index. Then traverses through the buckets. It should find the next index that is greater than the one so far- meaning, in the trace it comes afterwards, sequentially. Keep going until an index is gotten successfully from all buckets. Repeat in a loop till all pattern sets are found.

If a set is found, its saved to a set to_remove, which updates each time.

In the end, return a trace without all the marked indices.

Diagram:



Start from first bucket. traverse to next

check if index is greater (does it come after)

Keep going till have successfully found index from all buckets

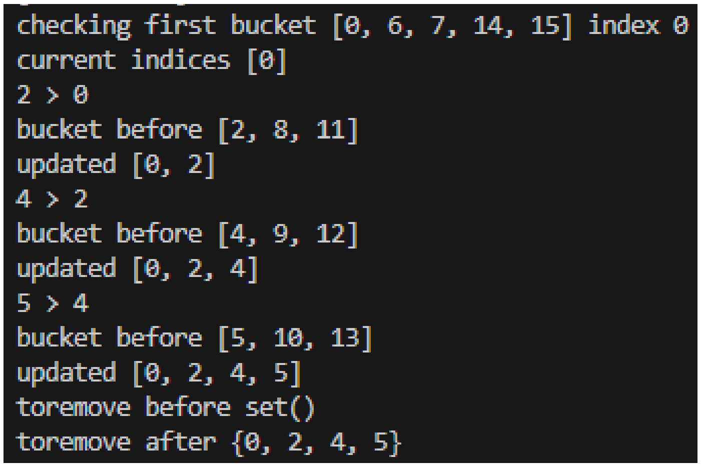
0 2 4 5 → to_remove set

Repeat loop till all pattern sets found.

In end, return trace without

Code (print statements)

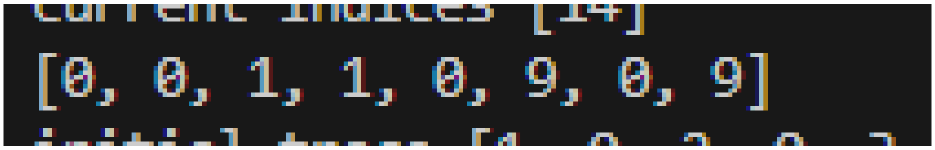
This shows the first iteration, it found a pattern and then stores it in the toremove set.



```
checking first bucket [0, 6, 7, 14, 15] index 0
current indices [0]
2 > 0
bucket before [2, 8, 11]
updated [0, 2]
4 > 2
bucket before [4, 9, 12]
updated [0, 2, 4]
5 > 4
bucket before [5, 10, 13]
updated [0, 2, 4, 5]
toremove before set()
toremove after {0, 2, 4, 5}
```

End result:

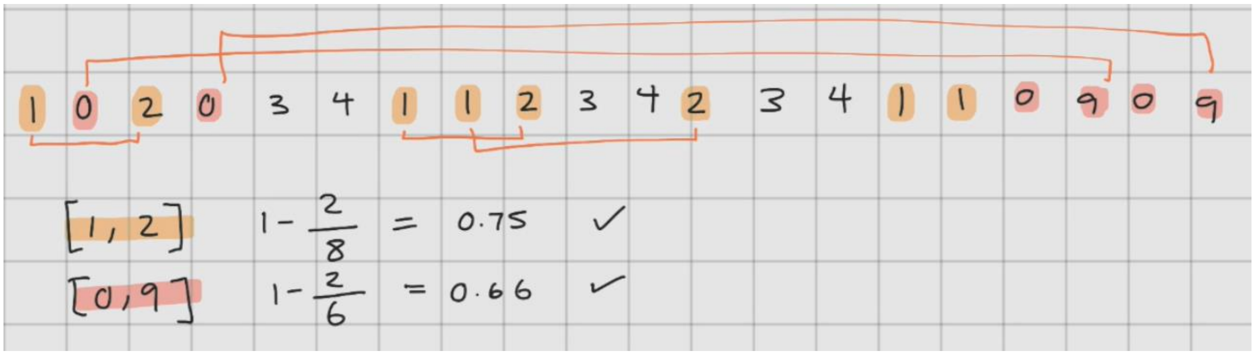
The returned trace removed all the patterns successfully and correctly. This matches the first diagram showed where I saw it manually.



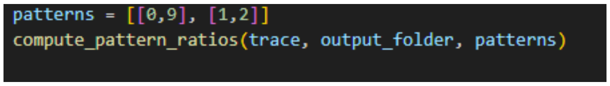
```
current indices [14]
[0, 0, 1, 1, 0, 9, 0, 9]
```

Example: Showing the use of the computer_pattern_ratios function

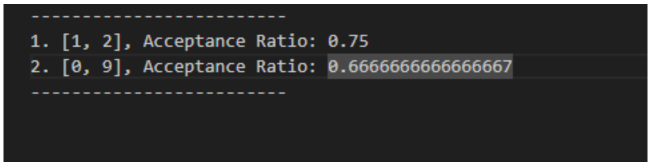
I pass in the list of patterns I want to use on the trace. It calculates the ratios for each one.



Pattern_AcceptanceRatios.txt in the unslicedtrace-1-patterns folder gives:



```
patterns = [[0,9], [1,2]]
compute_pattern_ratios(trace, output_folder, patterns)
```



```
-------------------------
1. [1, 2], Acceptance Ratio: 0.75
2. [0, 9], Acceptance Ratio: 0.6666666666666667
-------------------------
```

This is correct.