Mann Bhanushali
60004210142
C 21
Comps

15/15 [illegible] 14/04/24

# AA Experiment 2c

Aim: To identify and implement the potential method of amortized analysis.

Theory:-
The worst case running time might give an overly pessimist analysis for algorithms performing a sequence of operations on data structures. In amortized analysis, we average time required to perform a sequence of operations over the number of operations performed.
The potential approach focuses on how the current potential, may be calculated directly from the data structure's current state. It involves usage of a potential function that keeps track of the state of the data structure.

Methodology:-
A potential function $\phi(h)$ is maintained where $h$ is the state of the data structure. The amortized cost of an operation is :- $cost = c + \phi(h') - \phi(h)$ where $c$ is the actual cost of the operation, $h'$ is state after operation and $h$ is state before the operation.

Implementation:-
For stack as arrays, after each operation, the length of the data structure is observed. $\phi(h)$ is the length of data structure in state $h$. For example, initially let a stack have 0 elements. After a push (actual cost = 1), 1 ~~elements~~ element exists.
$\therefore$ amortized cost = $1 + 1 - 0 = 2$. After 'n' pushes, amortized cost = $1 + n - (n-1) = 2$. $\therefore$ Amortized cost after

Sundaram FOR EDUCATIONAL USE

Mann
600042210140
C21

push operations is still O(1).

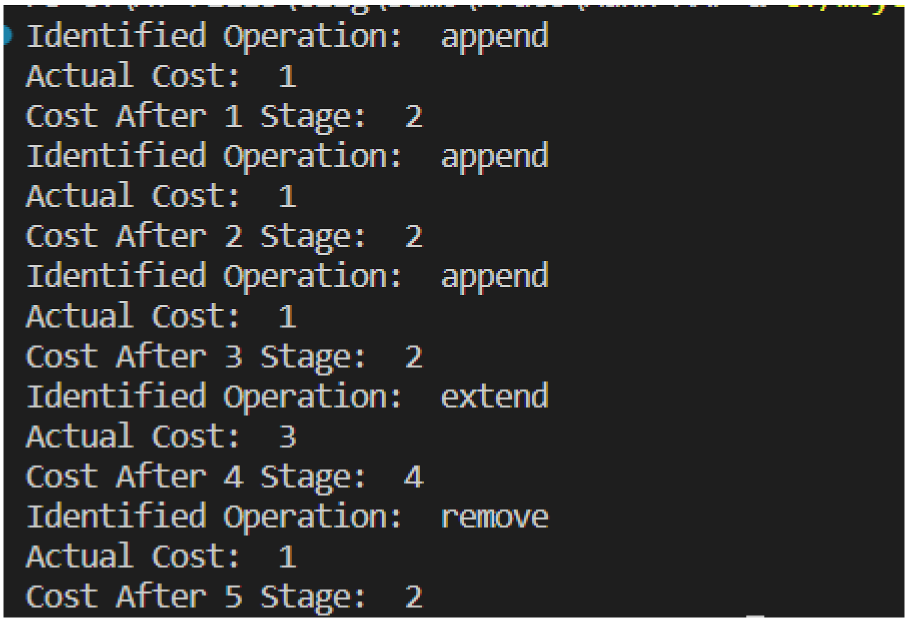
Conclusion:-
Thus, we implemented the potential method of amortized analysis ~~of~~ to stacks and arrays. ~~The potential function~~ Potential method ~~allows~~ focuses only on the current state allowing ignorance of historical states.

FOR EDUCATIONAL USE

Code:

```python
import re
import random

# Considering each operation costs 1 unit for amortized costs

code = """
my_list = [1, 2, 3]
my_list.append(4)
my_list.append(5)
my_list.append(10)
my_list.extend([5, 6])
my_list.remove(2)
"""
n=3
list_operations = ['append', 'extend', 'insert', 'remove', 'pop', 'index',
'count', 'sort', 'reverse']

pattern = re.compile(r'\b(?:' + '|'.join(list_operations) + r')\b\s*\(.+?\)')

amortized_costs = {operation: random.randint(1, 10) for operation in
list_operations}

credit = 10

matches = pattern.findall(code)

actual_costs = {
    'append': 1,
    'extend': n,
    'insert': 1,
    'remove': 1,
    'pop': 1,
    'index': 1,
    'count': n,
    'sort': n,
    'reverse': n
}
state=0
i=0
for match in matches:
    operation_name = re.match(r'\b(\w+)', match).group(1)
    actual_cost = actual_costs[operation_name]
    state=actual_cost+(state+1)-state
    print("Identified Operation: ", operation_name)
    print("Actual Cost: ", actual_cost)
    print(f"Cost After {i+1} Stage: ", state)
    i+=1
```

Output:



```
Identified Operation:  append
Actual Cost:  1
Cost After 1 Stage:  2
Identified Operation:  append
Actual Cost:  1
Cost After 2 Stage:  2
Identified Operation:  append
Actual Cost:  1
Cost After 3 Stage:  2
Identified Operation:  extend
Actual Cost:  3
Cost After 4 Stage:  4
Identified Operation:  remove
Actual Cost:  1
Cost After 5 Stage:  2
```