Experiment 1 B

Shashwat Shah
60004220126
TY Btech Comps B

Aim: Perform Amortized Analysis using accounting method.

Theory: Amortized analysis is a method used to analyze the performance of algorithms that perform a sequence of operations, where each individual operation may be fast, but the sequence of operation may be slow as a whole. It is used to determine the average cost per operation, allowing for a more accurate comparison of algorithm that perform different number of operations.

Accounting Method

→ It can be useful in understanding the performance of algorithm that performs a sequence of operations with varying cost.

→ It can be applied to a wide range of data structures and algorithms.

→ Unlike the aggregated analysis the accounting method assigns a different cost to each type of operation

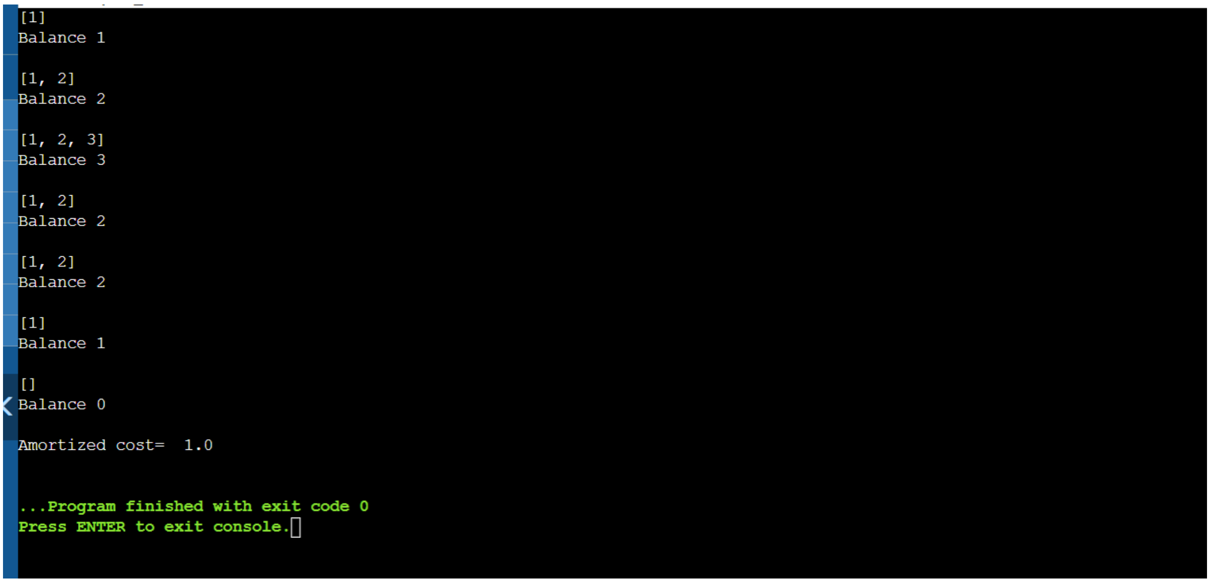
→ The accounting method is much like managing your personal finances. You can estimate the cost of your operations however you like, as long as, at the end of the day, the amount of money you have set aside is enough to pay bills.

→ The estimate cost of an operation may be greater on less than its actual cost.
Correspondingly the surplus of one operation can be used to pay the debt of other operations

Conclusion : Thus we studied about the accounting method in amortized analysis.

```
def accounting(n):
    size = 1
    total = 0
    dcost = 0
    icost = 0
    bank = 0

    print("Elements\tDoubling Copying Cost\tInsertion Cost\tTotal Cost\tBank\t\tSize")

    for i in range(1, n + 1):
        icost = 1
        if i > size:
            size *= 2
            dcost = i - 1
        total = icost + dcost
        bank += (3 - total)

        print(i, "\t\t\t", dcost, "\t\t\t", icost, "\t", total, "\t\t", bank, "\t\t", size)

        icost = 0
        dcost = 0

    n = int(input("Enter number of elements:"))
    print("Accounting method")
    accounting(n)



class AccountingStack:
    def __init__(self):
        self.stack = []
        self.cost = 0
        self.balance = 0

    def push(self, item):
        self.stack.append(item)
        self.cost += 1
        self.balance += 1
        self.printstack()

    def pop(self):
        self.stack.pop()
        self.cost += 1
        self.balance -= 1
        self.printstack()

    def multipop(self, k):
        for i in range(k):
            self.pop()

    def printstack(self):
        print(self.stack, "\nBalance", self.balance, "\n")


s = AccountingStack()
s.push(1)
s.push(2)
s.push(3)
s.pop()
s.printstack()
s.multipop(2)
print("Amortized cost= ", s.cost / 6)
```

Output :



```
[1]
Balance 1

[1, 2]
Balance 2

[1, 2, 3]
Balance 3

[1, 2]
Balance 2

[1, 2]
Balance 2

[1]
Balance 1

[]
Balance 0

Amortized cost=  1.0


...Program finished with exit code 0
Press ENTER to exit console.
```