Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITL502 **DATE:29/9/22**
COURSE NAME: Advanced Data Structures Laboratory **CLASS/Div: A**
SAP ID:60003200076 **Academic Year 2022-23**

EXPERIMENT NO. 1

CO/LO: LO1

AIM: To perform amortized analysis for stack operations

Theory:

Amortized analysis : The concept of amortized analysis is to compute the time complexity on a sequence of operations rather than finding a single case of worst case time ,which maybe very high but rare within the sequence.

In amortized analysis ,both the costly and less costly operations are considered together on a sequential run of the algorithm and compute their average ,which often provides a tighter bound of worst case complexity.

In amortized analysis ,time required to perform a sequence of data structure operations is averaged over all the successive operations performed .That is ,a large cost of one operation is spread out over many operations(amortized), where the others are less expensive.

There are three methods for amortized analysis viz

1. Aggregate method:
2. Accounting method:
3. Potential method:

Aggregate method:

In aggregate analysis, there are two steps. First, we must show that a sequence of n operations takes T(n) time in the worst case. Then, we show that each operation takes T(n)/n time, on average. Therefore, in aggregate analysis, each operation has the same cost

Accounting method:

Here, each operation is assigned a charge, called the amortized cost. Some operations can be charged more or less than they actually cost. If an operation's amortized cost exceeds its actual cost, we assign the difference, called a credit, to specific objects in the data structure. Credit can be used later to help pay for other operations whose amortized cost is less than their actual cost. Credit can never be negative in any sequence of operations.

Potential method:

The potential method is similar to the accounting method. However, instead of thinking about the analysis in terms of cost and credit, the potential method thinks of work already done as potential energy that can pay for later operations

Program:

Aggregate method:

```python
import numpy
import random as r

cost=0
s=[]
ele=1

def push(ele):
    global cost
    s.append(ele)
    cost=cost+1

def pop():
    global cost
    if len(s)!=0:
        print(s[len(s)-1])
        del s[len(s)-1]
        cost=cost+1
    else:
        print("underflow")


def multipop(no):
    global cost
    n=min(no,len(s))
    for i in range(0,n):
        pop()

#k=int(input("enter no of operations : "))
k=5
operations=numpy.random.uniform(0,3,k)
o=operations.astype(int)

for i in o:
    if i==0:
        push(ele)
    elif i==1:
        pop()
    elif i==2:
        x=r.randrange(1,10)
        print("muptipopin ",x," times")
        multipop(x)
print("0: push \t1: pop\t2: multipop")
print("operations : ", o)
print("aggregate amortized cost : ",cost/k)
```

Accounting method:

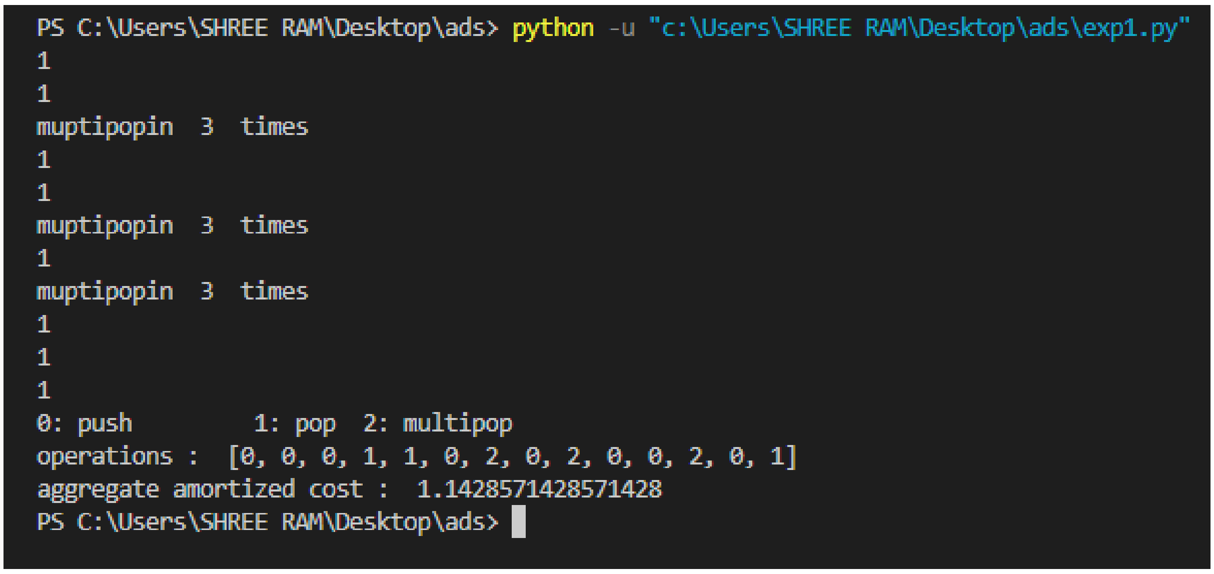
```
import numpy
import random as r

cost=0
c=[]
s=[]
ele=1

def push(ele):
    global cost
    s.append(ele)
    cost=cost+2
    if(len(c)!=0):
        c.append(1+c[len(c)-1])
    else:
        c.append(1)

def pop():
    global cost
    if len(s)!=0:
        print(s[len(s)-1])
        del s[len(s)-1]
        cost=cost+0
        c.append(c[len(c)-1]-1)
    else:
        print("underflow")



def multipop(no):
    global cost
    n=min(no,len(s))
    for i in range(0,n):
        pop()

#k=int(input("enter no of operations : "))
k=5
operations=numpy.random.uniform(0,3,k)
o=operations.astype(int)

for i in o:
    if i==0:
        push(ele)
    elif i==1:
        pop()
    elif i==2:
        x=r.randrange(1,10)
        print("muptipopin ",x," times")
        multipop(x)
print("0: push \t1: pop\t2: multipop")
print("operations : ", o)
print("accounting amortized cost : ",c)
```

Potential method:

```
import numpy
import random as r
pe=0
p=[]
cost=0
acost=0
s=[]
ele=1

def push(ele):
    global cost,pe
    s.append(ele)
    #cost=cost+1
    #acost=acost+2
    pe=pe+1
    p.append(pe)

def pop():
    global cost,pe
    if len(s)!=0:
        print(s[len(s)-1])
        del s[len(s)-1]
        #cost=cost+1
        #acost=acost+0
        pe=pe-1
        p.append(pe)
    else:
        print("underflow")


def multipop(no):
    global cost
    n=min(no,len(s))
    for i in range(0,n):
        pop()

#k=int(input("enter no of operations : "))
k=5
operations=numpy.random.uniform(0,3,k)
o=operations.astype(int)

for i in o:
    if i==0:
        push(ele)
    elif i==1:
        pop()
    elif i==2:
        x=r.randrange(1,10)
        print("muptipopin ",x," times")
        multipop(x)
print("0: push \t1: pop\t2: multipop")
print("operations : ", o)
print("potential energy : ",p)
```

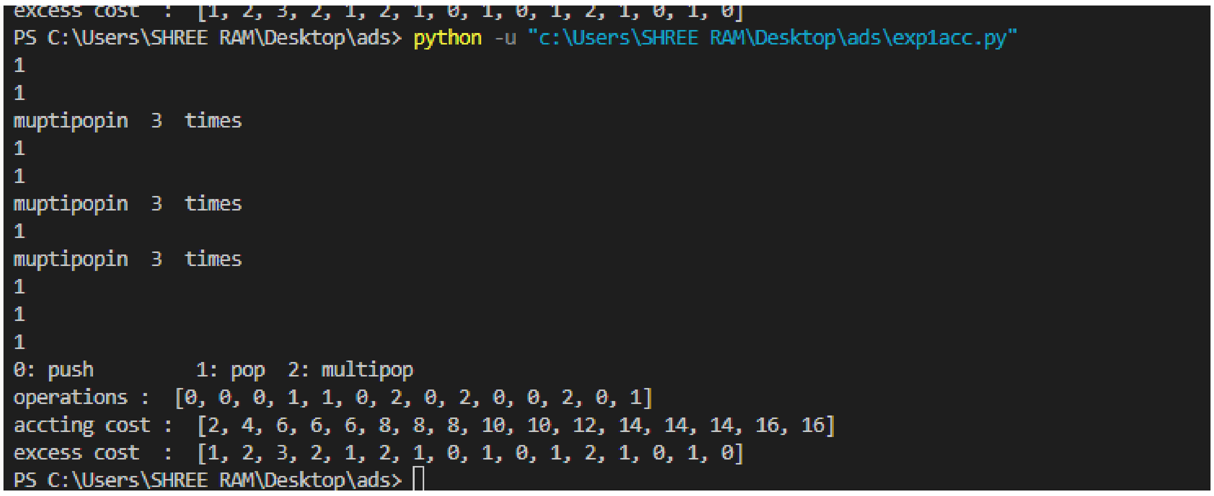
<u>Output</u>:

Aggregate method:



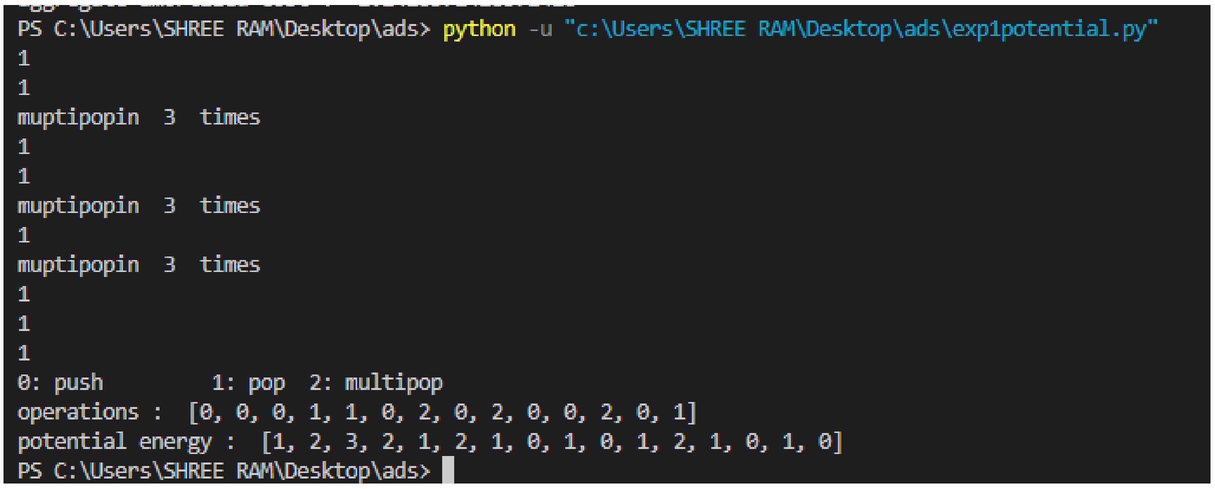
```
PS C:\Users\SHREE RAM\Desktop\ads> python -u "c:\Users\SHREE RAM\Desktop\ads\exp1.py"
1
1
muptipopin  3  times
1
1
muptipopin  3  times
1
muptipopin  3  times
1
1
1
0: push         1: pop  2: multipop
operations :  [0, 0, 0, 1, 1, 0, 2, 0, 2, 0, 0, 2, 0, 1]
aggregate amortized cost :  1.1428571428571428
PS C:\Users\SHREE RAM\Desktop\ads>
```

Accounting method:



```
excess cost  :  [1, 2, 3, 2, 1, 2, 1, 0, 1, 0, 1, 2, 1, 0, 1, 0]
PS C:\Users\SHREE RAM\Desktop\ads> python -u "c:\Users\SHREE RAM\Desktop\ads\exp1acc.py"
1
1
muptipopin  3  times
1
1
muptipopin  3  times
1
muptipopin  3  times
1
1
1
0: push         1: pop  2: multipop
operations :  [0, 0, 0, 1, 1, 0, 2, 0, 2, 0, 0, 2, 0, 1]
accting cost :  [2, 4, 6, 6, 6, 8, 8, 8, 10, 10, 12, 14, 14, 14, 16, 16]
excess cost  :  [1, 2, 3, 2, 1, 2, 1, 0, 1, 0, 1, 2, 1, 0, 1, 0]
PS C:\Users\SHREE RAM\Desktop\ads> 
```

Potential method:



```
PS C:\Users\SHREE RAM\Desktop\ads> python -u "c:\Users\SHREE RAM\Desktop\ads\exp1potential.py"
1
1
muptipopin  3  times
1
1
muptipopin  3  times
1
muptipopin  3  times
1
1
1
0: push         1: pop  2: multipop
operations :  [0, 0, 0, 1, 1, 0, 2, 0, 2, 0, 0, 2, 0, 1]
potential energy :  [1, 2, 3, 2, 1, 2, 1, 0, 1, 0, 1, 2, 1, 0, 1, 0]
PS C:\Users\SHREE RAM\Desktop\ads>
```

Analysis:

0 = PUSH   1 = POP   2 = MULTI POP

operations : [0, 0, 0, 1, 1, 0, 2, 0, 2, 0, 0, 2, 0, 1]

AGGREGATE METHOD This is a sequence of 14 operations with 8 push, 3 pop, 3 multipop(3)

the actual cost of push & pop is O(1)
the trivial actual cost of multi pop(n) is O(n)
ie O(3)

that is

$$\frac{8\times O(1) + 3\times O(1) + 3\times O(3)}{14} \le \frac{8\times O(n) + 3\times O(n) + 3\times O(n)}{14}$$

$$\le \frac{14\,O(n)}{14}$$

$$\le O(n)$$

However cost of multi pop operations in sequence cannot exceed $O(1)\times(8-3) = 5\times O(1)$

$\therefore$

$$\frac{8\times O(1) + 3\,O(1) + O(1)(8-3)}{14}$$

$$= \frac{(8+3+5)\,O(1)}{14} = \frac{16}{14}O(1)$$

$$= 1.142851\ldots\ O(1)$$

Thus all operations incur O(1) amortized costs

Accounting method :

similar to earlier discussion, we assign 2 credits for push & 0 credits for pop
even for multipop we assign 0 credits

Thus amortized cost of push is 2 = O(1)
for pop is 0 = O(1)
for multipop is 0 = O(1)

Potential Method :

similar to earlier discussion, each push generates 1 excess potential energy and each pop takes 1 potential energy

Thus amortized cost is O(1)

Conclusion:

Thus we have performed amortized analysis for stack operations of push, pop and multipop using aggregate method and observed similar behavior in accounting and potential method.

References:

https://brilliant.org/wiki/amortized-analysis/#:~:text=There%20are%20three%20main%20types,method%2C%20and%20the%20potential%20method.