122022010 Shrayank Jai Mistry
Description: Implementation of ARIES Algorithm.
Programming Language: Python

```python
from tabulate import tabulate
import copy

#* Keep track of the LSN no. for logs
LSN = 0

#* Storage for Log Records(On disk and on Memory)
LOG_DISK = {}
LOG_MEMORY = {}

#* Keep track of lastLSN flushed from memory to disk
flushedLSN = None

#* Keep track of the Last Completed checkpoint
masterRecord = [0, 0]

#* Class for log records
class LogRecord:
    def __init__(self, LSN, prevLSN, TxnId, type, dataItem, before, after, undoNext):
        self.LSN = LSN
        self.prevLSN = prevLSN
        self.TxnId = TxnId
        self.type = type
        self.dataItem = dataItem
        self.before = before
        self.after = after
        self.undoNext = undoNext

    def get_record(self):
        return [str(self.LSN), str(self.prevLSN), str(self.TxnId), str(self.type), str
(self.dataItem), str(self.before), str(self.after), str(self.undoNext)]
```

```python
#* Class for pages
class Page:
    def __init__(self, pageId, data):
        self.pageId = pageId
        self.pageLSN = None
        self.recLSN = None
        self.data = data

    def get_page(self):
        return [str(self.pageId), str(self.pageLSN), str(self.recLSN), str(self.data)]

#* Class for transactions in Active Transaction Table (ATT)
class ActiveTransaction:
    def __init__(self, id, status, lastLSN):
        self.transactionId = id
        self.status = status
        self.lastLSN = lastLSN

    def get_transaction(self):
        return [str(self.transactionId), str(self.status), str(self.lastLSN)]
```

```python
#* Represents the data on DISK(Non-volatile)
DISK_MEMORY = {
    '1': Page('1', {'A': 5, 'B': 15}),
    '2': Page('2', {'C': 10, 'D': 6}),
    '3': Page('3', {'E': 20, 'F': 6}),
}

#* Represents the data on MEMORY(Volatile)
MEMORY = {}

#* Mapping variables to the pages they belong
page_mapping = {
    'A': '1',
    'B': '1',
    'C': '2',
    'D': '2',
    'E': '3',
    'F': '3'
}

#* Tempory buffer for modified data_variables
temp_buffer = {}

#* Active Transaction Table(ATT)
ATT = {}

#* Dirty Page Table (DPT)
DPT = {}
```

```python
#* state = 1 (view before crash), state = 0 (view after recovery)
def view_system_state(state):
    if (state == 1):
        print('############################ SYSTEM STATE BEFORE CRASH ############################')

    if (state == 0):
        print('############################ SYSTEM STATE AFTER RECOVERY ############################')

    main_records = []
    print("  Log Records[On MEMORY]")
    for LSN, logR in LOG_MEMORY.items():
        main_records.append(logR.get_record())
    print(tabulate(main_records, headers = ["LSN", "prevLSN", "TxnId", "type", "data", "before", "after", "undoNext"]))

    print()
    disk_records = []
    print("  Log Records[On DISK]")
    for LSN, logR in LOG_DISK.items():
        disk_records.append(logR.get_record())
    print(tabulate(disk_records, headers = ["LSN", "prevLSN", "TxnId", "type", "data", "before", "after", "undoNext"]))

    print()
    print("  MEMORY [Volatile]")
    pages = []
    for pageId, page in MEMORY.items():
        pages.append(page.get_page())
    print(tabulate(pages, ["pageId", "pageLSN", "recLSN", "data"]))

    print()
    print("  DISK MEMORY [NON-Volatile]")
    pages = []
    for pageId, page in DISK_MEMORY.items():
        pages.append(page.get_page())
    print(tabulate(pages, ["pageId", "pageLSN", "recLSN", "data"]))

    print()
    print("  Dirty Page Table(DPT)")
    dirty_pages = []
    for pageId, recLSN in DPT.items():
        dirty_pages.append([pageId, recLSN])
    print(tabulate(dirty_pages, headers = ["PageId", "recLSN"]))

    print()
    print("Active Transaction Table(ATT)")
    transactions = []
    for transactionId, transaction in ATT.items():
        transactions.append(transaction.get_transaction())
    print(tabulate(transactions, headers = ["TransactionId", "status", "lastLSN"]))

    print('#################################################################################')
```

```
import ds
import rec
import copy

def get_value(ins, a, b):
    if (ins.find('+') > -1): return a + b
    if (ins.find('-') > -1): return a - b
    if (ins.find('*') > -1): return a * b
    if (ins.find('/') > -1): return a / b
```

```
if __name__ == '__main__':
    schedule = open('schedule_input.txt', 'r')
    data = []

    for instruction in schedule:
        data.append(instruction.strip())

    #* Getting all instructions in the schedule
    instructions = data[0:]

    execute_instructions(instructions)
```

Start Instructions Execution:

```python
def execute_instructions(instructions):
    for ins in instructions:

        #* Normal Operations
        if (ins.find('(') != -1):
            operation = ins[0:ins.find('(')]
            if (operation == 'CRASH'):
                ds.view_system_state(1)
                ds.LOG_MEMORY.clear()
                ds.DPT.clear()
                ds.ATT.clear()
                ds.MEMORY.clear()
                rec.execute_recovery()

            if (operation == 'CHECKPOINT'):

                checkpointData = ds.CheckPoint(ds.ATT, ds.DPT)
                ds.masterRecord[0] = ds.LSN
                ds.masterRecord[1] = checkpointData

                log = ds.LogRecord(ds.LSN, None, None, 'CHECKPOINT', '-', '-', '-', '-')
                ds.LOG_MEMORY[ds.LSN] = log

                ds.LSN += 1

                #* Flushing pages from MEMORY TO DISK
                # pages_to_remove = []
                if ds.flushedLSN != None:
                    for pageId, recLSN in ds.DPT.items():
                        if recLSN <= ds.flushedLSN:
                            ds.DISK_MEMORY[pageId] = copy.deepcopy(ds.MEMORY[pageId])
                            # ds.MEMORY[pageId].pageLSN = None
                            # ds.MEMORY[pageId].recLSN = None
                            # pages_to_remove.append(pageId)

                    # #* Clearing out flushed pages from DPT
                    # for id in pages_to_remove:
                    #     ds.DPT.pop(id)
```

```python
if (operation == "BEGIN"):
                transactionId = ins[ins.find('(') + 1:ins.find(')')]
                log = ds.LogRecord(ds.LSN, None, transactionId, 'BEGIN', '-', '-', '-', '-')
                ds.LOG_MEMORY[ds.LSN] = log
                ds.ATT[transactionId] = ds.ActiveTransaction(transactionId, 'UNDO', ds.LSN)
                ds.LSN += 1

            if (operation == 'READ'):
                data_variable = ins[ins.find('(') + 1:ins.find(',')]
                transactionId = ins[ins.find(' ') + 1:ins.find(')')]

                pageId = ds.page_mapping[data_variable]
                if ds.MEMORY.get(pageId):                               #* Page already present in MEMORY
                    continue
                else:
                    ds.MEMORY[pageId] = copy.deepcopy(ds.DISK_MEMORY[pageId])

            if (operation == 'WRITE'):
                data_variable = ins[ins.find('(') + 1:ins.find(',')]
                transactionId = ins[ins.find(' ') + 1:ins.find(')')]

                prevLSN = ds.ATT[transactionId].lastLSN
                ds.ATT[transactionId].lastLSN = ds.LSN

                pageId = ds.page_mapping[data_variable]
                before = ds.MEMORY[pageId].data.get(data_variable)
                after = ds.temp_buffer[data_variable]

                #* Writing Update Log record to MEMORY
                log = ds.LogRecord(ds.LSN, prevLSN, transactionId, 'UPDATE', data_variable, before, after, '-')
                ds.LOG_MEMORY[ds.LSN] = log

                #* Updating the pages in MEMORY
                ds.MEMORY[pageId].data[data_variable] = after
                ds.MEMORY[pageId].pageLSN = ds.LSN

                if ds.MEMORY[pageId].recLSN == None:
                    ds.MEMORY[pageId].recLSN = ds.LSN

                #* Adding pages to dirty page table (DPT)
                if ds.DPT.get(pageId) == None:
                    ds.DPT[pageId] = ds.LSN

                ds.LSN += 1
```

```
if (operation == 'COMMIT'):
                transactionId = ins[ins.find('(') + 1:ins.find(')')]

                # print('COMMITING TRANSACTION ' + str(transactionId))

                ds.ATT[transactionId].status = 'COMMIT'

                prevLSN = ds.ATT[transactionId].lastLSN
                ds.ATT[transactionId].lastLSN = ds.LSN

                #* Writing COMMIT Log record to MEMORY
                log = ds.LogRecord(ds.LSN, prevLSN, transactionId, 'COMMIT', '-', '-', '-', '-')
                ds.LOG_MEMORY[ds.LSN] = log

                #* Flushing all logs from MEMORY to DISK
                for LSN, log in ds.LOG_MEMORY.items():
                    ds.LOG_DISK[LSN] = log
                ds.LOG_MEMORY.clear()

                ds.flushedLSN = ds.LSN
                ds.LSN += 1


            if (operation == 'ABORT'):
                transactionId = ins[ins.find('(') + 1:ins.find(')')]

                ds.ATT[transactionId].status = 'ABORT'

                prevLSN = ds.ATT[transactionId].lastLSN
                ds.ATT[transactionId].lastLSN = ds.LSN

                #* Writing ABORT Log record to MEMORY
                log = ds.LogRecord(ds.LSN, prevLSN, transactionId, 'ABORT', '-', '-', '-', '-')
                ds.LOG_MEMORY[ds.LSN] = log

                ds.LSN += 1

                prev = prevLSN
                while prev != None:
                    log = ds.LOG_MEMORY[prev]
                    if log.type == 'UPDATE':
                        prevLSN = ds.ATT[transactionId].lastLSN
                        ds.ATT[transactionId].lastLSN = ds.LSN
                        clr_log = ds.LogRecord(ds.LSN, prevLSN, transactionId, 'CLR', log.dataItem, log.after, log.before, l
og.prevLSN)
                        ds.LOG_MEMORY[ds.LSN] = clr_log

                        pageId = ds.page_mapping[log.dataItem]
                        #* Updating the pages in MEMORY
                        ds.MEMORY[pageId].data[log.dataItem] = log.before
                        ds.MEMORY[pageId].pageLSN = ds.LSN
                        ds.LSN += 1
                    prev = log.prevLSN

                #* Flushing all logs from MEMORY to DISK
                for LSN, log in ds.LOG_MEMORY.items():
                    ds.LOG_DISK[LSN] = log
                ds.LOG_MEMORY.clear()

                ds.flushedLSN = ds.LSN
                ds.LSN += 1
```

```
else:
            updating_data = ins[0]
            a = ins[ins.find('[') + 1]

            #* Code to Refactor
            b = ''
            index = ins.find(']') - 1
            while (ins[index] != ' '):
                b += ins[index]
                index -= 1
            b = b[::-1]
            b = int(b)

            pageId = ds.page_mapping[a]
            a = ds.MEMORY[pageId].data.get(a)
            updated_value = get_value(ins, a, b)
            ds.temp_buffer[updating_data] = updated_value
```

Recovery Code (After System Crash):
Analysis Phase:

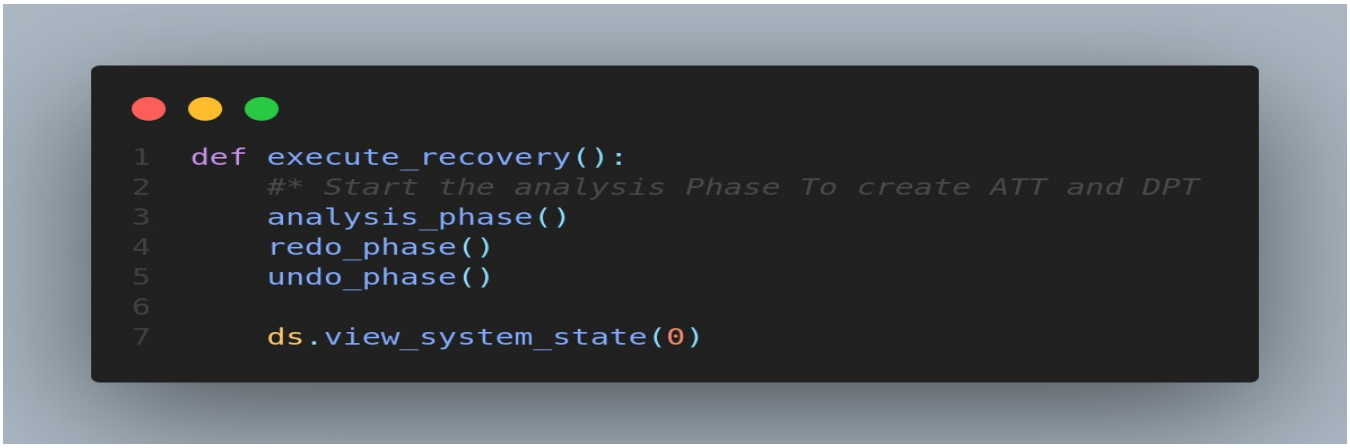
```python
import ds
import copy

def analysis_phase():
    lsn = ds.masterRecord[0]
    ds.ATT = copy.deepcopy(ds.masterRecord[1].ATT)
    ds.DPT = copy.deepcopy(ds.masterRecord[1].DPT)

    for pageId, recLSN in ds.DPT.items():
        ds.MEMORY[pageId] = copy.deepcopy(ds.DISK_MEMORY[pageId])

    while ds.LOG_DISK.get(lsn):
        log = ds.LOG_DISK[lsn]

            #* Add the transaction in ATT with status UNDO
        if log.type == 'BEGIN':
            ds.ATT[log.TxnId] = ds.ActiveTransaction(log.TxnId, 'UNDO', log.LSN)

        #* Add the page in DPT if not present and set recLSN with current LSN
        if log.type == 'UPDATE':
            pageId = ds.page_mapping[log.dataItem]

            if not ds.DPT.get(pageId):
                ds.DPT[pageId] = log.LSN

            ds.MEMORY[pageId] = copy.deepcopy(ds.DISK_MEMORY[pageId])
            ds.ATT[log.TxnId].lastLSN = log.LSN

        if log.type == 'COMMIT':
            ds.ATT[log.TxnId].status = 'COMMIT'
            ds.ATT[log.TxnId].lastLSN = log.LSN

        if log.type == 'ABORT':
            ds.ATT[log.TxnId].status = 'ABORT'
            ds.ATT[log.TxnId].lastLSN = log.LSN

        if log.type == 'END':
            ds.ATT.pop(log.TxnId)
        lsn += 1
```

Redo Phase:

```python
def redo_phase():
    LSN = 100

    #* Get Minimum LSN to start redo scanning
    for pageId, recLSN in ds.DPT.items():
        if LSN > recLSN: LSN = recLSN

    #* Redo all the update instructions
    while ds.LOG_DISK.get(LSN):
        log = ds.LOG_DISK[LSN]
        LSN += 1

        if log.type == 'UPDATE' or log.type == 'CLR':
            pageId = ds.page_mapping[log.dataItem]

            #* Redo the records for pages whose recLSN < LSN
            for pId, recLSN in ds.DPT.items():
                #* Checking for ignore conditions
                if pageId != pId or ds.DISK_MEMORY[pId].recLSN >= LSN:
                    continue
                else:
                    ds.MEMORY[pId].data[log.dataItem] = log.after
                    ds.MEMORY[pId].pageLSN = log.LSN

            #* Redo all records both UPDATE AND CLR
            # ds.MEMORY[pageId].data[log.dataItem] = log.after
            # ds.MEMORY[pageId].pageLSN = log.LSN

    transactions_to_remove = []
    for transactionId, transactionInfo in ds.ATT.items():
        if transactionInfo.status == 'COMMIT' or transactionInfo.status == 'ABORT':
            transactions_to_remove.append(transactionId)
            prevLSN = ds.ATT[transactionId].lastLSN
            log = ds.LogRecord(ds.LSN, prevLSN, transactionId, 'END', '-', '-', '-', '-')
            ds.LOG_MEMORY[ds.LSN] = log
            ds.LSN += 1

    for tid in transactions_to_remove:
        ds.ATT.pop(tid)
```

Undo Phase:

```python
def undo_phase():

    transactions_to_remove = []
    for transactionId, transaction in ds.ATT.items():
        LSN = transaction.lastLSN

        while LSN != None:
            log = ds.LOG_DISK[LSN]
            if log.type == 'UPDATE':
                prevLSN = ds.ATT[transactionId].lastLSN
                ds.ATT[transactionId].lastLSN = ds.LSN

                #* Adding CLR's for undo Updates
                clr_log = ds.LogRecord(ds.LSN, prevLSN, transactionId, 'CLR', log.dataItem, log.after, log.before, log.prevLSN)
                ds.LOG_MEMORY[ds.LSN] = clr_log

                #* Undoing the operations on pages
                pageId = ds.page_mapping[log.dataItem]
                ds.MEMORY[pageId].data[log.dataItem] = log.before
                ds.MEMORY[pageId].pageLSN = ds.LSN

                ds.LSN += 1
            LSN = log.prevLSN

        prevLSN = ds.ATT[transactionId].lastLSN
        ds.ATT[transactionId].lastLSN = ds.LSN
        log = ds.LogRecord(ds.LSN, prevLSN, transactionId, 'ABORT', '-', '-', '-', '-')

        ds.LOG_MEMORY[ds.LSN] = log

        #* Flushing all logs from MEMORY to DISK
        for LSN, log in ds.LOG_MEMORY.items():
            ds.LOG_DISK[LSN] = log
        ds.LOG_MEMORY.clear()

        ds.flushedLSN = ds.LSN
        ds.LSN += 1

        #* Flushing pages from MEMORY TO DISK
        pages_to_remove = []
        for pageId, recLSN in ds.DPT.items():
            if recLSN <= ds.flushedLSN:
                ds.DISK_MEMORY[pageId] = copy.deepcopy(ds.MEMORY[pageId])
                pages_to_remove.append(pageId)

        #* Clearing out flushed pages from DPT
        for id in pages_to_remove:
            ds.DPT.pop(id)

        #* Writing the TXN-END log record
        prevLSN = ds.ATT[transactionId].lastLSN
        log = ds.LogRecord(ds.LSN, prevLSN, transactionId, 'END', '-', '-', '-', '-')
        ds.LOG_MEMORY[ds.LSN] = log

        transactions_to_remove.append(transactionId)
        ds.LSN += 1

    for tid in transactions_to_remove:
        ds.ATT.pop(tid)
```


def execute_recovery():
#* Start the analysis Phase To create ATT and DPT
analysis_phase()
redo_phase()
undo_phase()
ds.view_system_state(0)