CSED 2014

External Sorting

Ahmed Ibraheem Ahmed	(2)
Ashraf Saleh Mohamed	(20)
Hazem Ibraheem Nada	(31)

12

Problem Statement

In this assignment, you will implement external sorting as covered in class. The main objective is to use a fixed amount of memory to sort a set of fixed size records stored in a heap file on the disk. You will work, among all other necessary packages and classes, with the interface to HeapFile class and the Scan class. The interface of the HeapFile supports the creation and deletion of files of unordered pages and allows you to scan the file page by page. You need to implement the ExternalSort class which extends Sort. you will write code to sort unordered records, return the next sorted element and clean up any temporary files used during the sorting process.

Data Structures

Priority Queue

We used a priority queue in merge method to sort the min (or max) record of every run that is being merged

Testing

We've performed several tests using several test methods

test1()

creates 1000 record (sorted descending)each of 3 field <int, int, int> and sorts them in an ascending order using the first field as a key.

test2()

creates 1000 record (sorted ascending)each of 3 field <int, int, int> and sorts them in a descending order using the first field as a key.

test3()

creates 1000 record (random)each of 3 field <int, int, int> and sorts them in a descending order using the first field as a key.

test4()

creates 1000 record (random) each of 4 field <int, string, int, string> and sorts them in an ascending order using the second field as a key.

Pseudo code

Method pass0()

```
{
    // the first step
    // tale B pages and store their records in an array of tuples and perform
    // quick sort on this array then create successive runs until there is no more tuples

    int size = number of records in one page * number of buff pages;
    tuple[] array;
    for (i=0 ; i< size; i++){
        array[i] = nextrecord;
        if array[i] = null then
            size = i;
            break
        endif
    }
    quickSort(array,0,size);
    HeapFile run ;
    for (i=0;i<size;i++)
        run.insert(array[i]);


}
```

Method merge(int leftRuns, int depth)

```
{
    while(leftRuns !=0){
        merged = min (n-1, leftRuns);
        leftRuns -= merged;
        //define an Iterator[merged] and a priority queue<Pair<Tuple,
runNo>> pr
```

```
                    //create new Run (heapFile)
                    While (pr not Empty){
                              pair temp = pr.poll();
                              currentRun.insert(temp.Tuple);
                              next = iterators[temp.runNo].getNext();
if (next!=null)
          pr.add(new pair(next,temp.runNo));
}
// there are more runs that are not merged yet.
if (# currentRuns > 1)
          Merge(# currentRun, depth+1)

}
}
```