**CS323-22: Project 4 (CPP)**

**Yida Tao**

**Due date: Oct. 10, 2017**

Algorithm Steps:

Step 0:

- inFile <-- open the input file;

- read and count the number of date items in the inFile

- close inFile

- dynamically allocate heapAry[] of the size count+1

and initallize heapAry[0] to 0. Remember, heapAry[1] is the root of the tree

and heapAry[0] is the last index used in heapAry.

- outFile1 <-- open for writing

- outFile2 <-- open for writing

step 1: call buildHeap (see algorithm below)

step 2: call deleteHeap (see algorithm below)

step 3: close all files

**Source Code**

**#include <iostream>**

**#include <fstream>**

**#include <string>**

**using namespace std;**

**class HeapSort{**

**private:**

**int \*heapAry;**

**const int rootIndex = 1;**

**int fatherIndex;**

**int leftKidIndex;**

**int rightKidIndex;**

**int minKidIndex;**

**int maxSize;**

**void bubbleUp(int i){**

**if(isRoot(i)) return;**

**fatherIndex = i/2;**

**if(heapAry[i] < heapAry[fatherIndex]){**

**swap(i,fatherIndex);**

**bubbleUp(fatherIndex);**

**}**

**else return;**

**}**

**void bubbleDown(int i){**

**//if no child, return**

**if(isleaf(i)) return;**

**leftKidIndex= 2\*i;**

**//only left child**

**if(isOneChild(i)){**

**if(heapAry[i] > heapAry[leftKidIndex]){**

**swap(i,leftKidIndex);**

**}**

**return;**

**}**

**//2 children, swap with smaller one, call bubbledown again**

**minKidIndex = findMinKidIndex(i);**

**if(heapAry[i] > heapAry[minKidIndex]){**

**swap(i,minKidIndex);**

**bubbleDown(minKidIndex);**

**}**

**return;**

**}**

**//check isleaf**

**bool isleaf(int i){**

**bool b = false;**

**leftKidIndex = 2\*i;**

**rightKidIndex = 2\*i+1;**

**if(leftKidIndex > maxSize || rightKidIndex > maxSize){**

**b= true;**

**}**

**else if(heapAry[leftKidIndex] == -1 && heapAry[rightKidIndex] == -1){**

**b = true;**

**}**

**return b;**

**}**

**//check only has left child**

**bool isOneChild(int i){**

**bool b = false;**

**rightKidIndex = 2\*i+1;**

**if(heapAry[rightKidIndex] == -1){**

**b = true;**

**}**

**return b;**

**}**

**bool isRoot(int i){**

**return i == 1;**

**}**

**//find the index has smaller kid**

**int findMinKidIndex(int i){**

**rightKidIndex = 2\*i+1;**

**leftKidIndex = 2\*i;**

**int small;**

**if(heapAry[leftKidIndex] >= heapAry[rightKidIndex]){**

**small = rightKidIndex;**

**}**

**else small = leftKidIndex;**

**return small;**

**}**

**public:**

**HeapSort(int size){**

**heapAry = new int[size+1];**

**//initallize array by filled with -1**

**for(int i = 1; i<= size; i++){**

**heapAry[i] = {-1};**

**}**

**heapAry[0]=0;**

**maxSize = size;**

**}**

**~HeapSort(){**

**delete[] heapAry;**

**}**

**//read ints and build heap**

**void buildHeap(char\* in){**

**ifstream inFile;**

**inFile.open(in);**

**int i;**

**while(inFile>>i){**

**cout<<i<<endl;**

**insertOneDataItem(i);**

**}**

**inFile.close();**

**}**

**//heapsort by delete heap**

**void deleteHeap(){**

**cout<<"heapsort:";**

**while(!isHeapEmpty()){**

**//printHeap();**

**cout<<" "<<deleteRoot();**

**}**

**cout<<endl;**

**}**

**//insert one data to first empty spot of array**

**void insertOneDataItem(int i){**

**if(isHeapFull()){**

**cout<<"this array is full, can not insert number!";**

**return;**

**}**

**heapAry[0]++;**

**heapAry[heapAry[0]] = i;**

**bubbleUp(heapAry[0]);**

**printHeap();**

**}**

**//delete root and print it**

**int deleteRoot(){**

**int oldroot = heapAry[rootIndex];**

**heapAry[rootIndex] = -1;**

**swap(rootIndex,heapAry[0]);**

**heapAry[0]--;**

**bubbleDown(rootIndex);**

**return oldroot;**

**}**

**//print heap**

**void printHeap(){**

**for(int i =0; i < heapAry[0]; i++ ){**

**cout<< heapAry[i]<<",";**

**}**

**cout<<heapAry[heapAry[0]]<<endl;**

**}**

**//swap two index**

**void swap(int i1, int i2){**

**int x = heapAry[i1];**

**heapAry[i1] = heapAry[i2];**

**heapAry[i2] = x;**

**}**

**bool isHeapEmpty(){**

**bool b = false;**

**if(heapAry[0]==0){**

**b = true;**

**}**

**return b;**

**}**

**bool isHeapFull(){**

**bool b = false;**

**if(heapAry[0]>= maxSize){**

**b = true;**

**}**

**return b;**

**}**

**};**

**int main(int argc, char \*argv[]){**

**HeapSort \*myHeap;**

**int count = 0;**

**if(argv[1]==NULL) {**

**cout<<"no parameter"<<endl;**

**return 0;**

**}**

**ifstream inFile;**

**inFile.open(argv[1]);**

**if(!inFile.is\_open()){**

**cout<<"cant find file"<<endl;**

**return 0;**

**}else {**

**int i;**

**while(inFile>>i){**

**count++;**

**}**

**inFile.close();**

**}**

**//finish counting**

**myHeap = new HeapSort(count);**

**streambuf \*console = cout.rdbuf();**

**ofstream out1;**

**out1.open(argv[2]);**

**ofstream out2;**

**out2.open(argv[3]);**

**//set output to outfile 1**

**cout.rdbuf(out1.rdbuf());**

**myHeap->buildHeap(argv[1]);**

**//set output to outfile 2**

**cout.rdbuf(out2.rdbuf());**

**myHeap->deleteHeap();**

**cout.rdbuf(console);**

**cout<<"done"<<endl;**

**out1.close();**

**out2.close();**

**delete myHeap;**

**}**

**Input**

23

82 7 28

9 19 28 9

7 13

21

4

81 19 36

18 320

61 7

3

4

5

16

17

**Output 1**

23

1,23

82

2,23,82

7

3,7,82,23

28

4,7,28,23,82

9

5,7,9,23,82,28

19

6,7,9,19,82,28,23

28

7,7,9,19,82,28,23,28

9

8,7,9,19,9,28,23,28,82

7

9,7,7,19,9,28,23,28,82,9

13

10,7,7,19,9,13,23,28,82,9,28

21

11,7,7,19,9,13,23,28,82,9,28,21

4

12,4,7,7,9,13,19,28,82,9,28,21,23

81

13,4,7,7,9,13,19,28,82,9,28,21,23,81

19

14,4,7,7,9,13,19,19,82,9,28,21,23,81,28

36

15,4,7,7,9,13,19,19,82,9,28,21,23,81,28,36

18

16,4,7,7,9,13,19,19,18,9,28,21,23,81,28,36,82

320

17,4,7,7,9,13,19,19,18,9,28,21,23,81,28,36,82,320

61

18,4,7,7,9,13,19,19,18,9,28,21,23,81,28,36,82,320,61

7

19,4,7,7,7,13,19,19,18,9,28,21,23,81,28,36,82,320,61,9

3

20,3,4,7,7,7,19,19,18,9,13,21,23,81,28,36,82,320,61,9,28

4

21,3,4,7,7,4,19,19,18,9,7,21,23,81,28,36,82,320,61,9,28,13

5

22,3,4,7,7,4,19,19,18,9,7,5,23,81,28,36,82,320,61,9,28,13,21

16

23,3,4,7,7,4,19,19,18,9,7,5,23,81,28,36,82,320,61,9,28,13,21,16

17

24,3,4,7,7,4,17,19,18,9,7,5,19,81,28,36,82,320,61,9,28,13,21,16,23

**Output 2**

heapsort: 3 4 4 5 7 7 7 9 9 13 16 17 18 19 19 21 23 28 28 36 61 81 82 320