ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 335E

ANALYSIS OF ALGORITHM I
HOMEWORK 2 REPORT

RESEARCH ASSISTANT : Tacettin Ayar

DUE DATE : 12.12.2019

150160060 : NESLİHAN ÇEKİÇ

FALL 2019-2020

Contents

1 INTRODUCTION

In this assignment, we were asked to perform printing on the console screen according to the time of the events I created with the Min Heap method. For this purpose, I implemet a heap algorithm like max heap pseduo code on the course slide and change it with min heap features.

2 DEVELOPMENT ENVIRONMENT

The project was written in Visual Studio IDE and checked in SSH environment.Example run code is in the .txt file in the assignment. main.cpp is the main file of the project and the classes and their methods are defined in minheap.h.

3 DATA STRUCTURES AND VARIABLES

In minheap.h, event name, time, and start variables, created publicly in the class named Event, it holds the start or end time of an event. With this logic, it is necessary to create 2 objects belonging to an event. The methods I created for this class are as follows:

Constructor (with default values)
Copy constructor

```
//EVENT CLASS
class event {
public:
    string event_name;
    int time;
    bool start; //Holds if this object is start time or and time

    //Constructor:
    event(string e = "0", int t = 0, bool s = true) {
        event_name = e;
        time = t;
        start = s;
    }
    //Copy Constructor:
    event(const event &ev){
        event_name = ev.event_name;
        time = ev.time;
        start = ev.start;
    }
```

$=, <, >$ operator overloads

```
//Operator overloads:
void operator=(const event& ev) {
    event_name = ev.event_name;
    time = ev.time;
    start = ev.start;
}

bool operator > (event const& obj) {
    if (obj.time >= time) {
        return false;
    }
    else {
        return true;
    }
}
bool operator < (event const& obj) {
    if (obj.time < time) {
        return false;
    }
    else {
        return true;
    }
}
```

In the class called MinHeap, a vector is defined as an event type that holds all event objects and a variable that holds the heap size is defined as private. The methods I have defined for this class are as follows:

```
// A class for Min Heap
class MinHeap
{
    vector<event> harr; // vector of event(all events) in heap
    int heap_size; // Current number of elements in min heap

public:
    // Constructor :
    MinHeap(vector<event>);

    //Getter method of Heap size:
    int getHeap_size() { return heap_size; }

    // to heapify a subtree with the root at given index
    void MinHeapify(int);

    // to get index of parent and left, right child index
    int parent(int i) { return (i - 1) / 2; }
    int left(int i) { return (2 * i + 1); }
    int right(int i) { return (2 * i + 2); }

    // to remove the root which is the minimum element
    event RemoveRoot();

};
```

Constructor: adds the elements in the sent vector to heap.

```
MinHeap::MinHeap(vector<event> a)
{
    heap_size = 0;
    harr = a;
    vector <event>::iterator it;
        for (it = harr.begin(); it != harr.end(); ++it) {
            // First insert the new key at the end
            heap_size++;
            int i = heap_size - 1;
            harr[i] = *it;

            // Fix the min heap property if it is violated
            while (i != 0 && harr[parent(i)] > harr[i])
            {
                swap(&harr[i], &harr[parent(i)]);
                i = parent(i);
            }
    }

}
```

RemoveRoot (): Reorganizes the heap order by removing the root node.

```
// Method to remove minimum element (or root) from min heap
event MinHeap::RemoveRoot()
{
        if (heap_size == 1)
        {
            heap_size--;
            return harr[0];
        }

        // Store the minimum value, and remove it from heap
        event root = harr[0];
        harr[0] = harr[heap_size - 1];
        heap_size--;
        MinHeapify(0);

        return root;

}
```

MinHeapify (): heapify a subtree with the root at given index.

```
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l] < harr[i])
        smallest = l;
    if (r < heap_size && harr[r] < harr[smallest])
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}
```

I also created a simple swap function to relocate objects in the vector.

```
//Swap function for heap operations
void swap(event* x, event* y)
{
    event temp = *x;
    *x = *y;
    *y = temp;
}
```

4 PROGRAM FLOW

In main.cpp, the program flow is as follows. I have written read code to read 1 line in .txt file with the help of ifstream. I created two separate events for the start time and end time of one event then added these events to the event type vector I created. This process continued until the data in the .txt file was finished.

```
int main(int argc, char** argv)
{
    string e_name;
    int s_time, e_time;
    vector <event> event_list;

    //Reading file from text file, create objects of events and add to event type vector.
    ifstream file;
    file.open(argv[1]);
    while (!file.eof()) {
        file >> e_name >> s_time >> e_time;
        event e1(e_name, s_time, true);
        event e2(e_name, e_time, false);
        event_list.push_back(e1);
        event_list.push_back(e2);
    }
```

After reading ffile operations, I created an object in MinHeap and sent the filled vector with constructor function. With this function, all events are added to heap. I've created a time counter to keep time. I also created a temp event to see the events in the loop. Before the loop, I removed the root using the RemoveRoot method and started the process. If the time and the time of the event do not match, it prints NO EVENT and increase the time. Vice versa, 2 print code lines were written depending on whether they were start or end. Then the root was removed again with RemoveRoot. 2 different events can take place at the same time so i created another loop for this situtation. Lastly, I check heap size for check if the heap is empty or not. If heap is empty, it is provided to execute the program completion operations.

```
MinHeap heap(event_list); //Create Min Heap for event_list elements
int time=1; //Counts time
event temp; //Temp event object for loop

temp = heap.RemoveRoot(); //Remove root from min heap

while(heap.getHeap_size()>-1){ //Check if the heap empty

    if (temp.time != time) { //if removed node's time is not equal the time there is no event on that time interval
        cout << "TIME " << time << ": "<< "NO EVENT" << endl;
    }
    else {
        while (temp.time == time) { //if removed node's time is not equal the time there is event on that time interval
            if (temp.start == true) { //if event object is start time
                cout << "TIME " << time << ": " << temp.event_name << " STARTED" << endl;
            }
            if (temp.start == false) { //if event object is end time
                cout << "TIME " << time << ": " << temp.event_name << " ENDED" << endl;
            }
            if (heap.getHeap_size()==0) { //checck if heap is empty or not. If empty finish the program
                cout << "TIME " << time << ": " << "NO MORE EVENTS, SCHEDULER EXITS" << endl;
                file.close();
                return 0;
            }
            temp = heap.RemoveRoot(); //Remove root node again.
        }
    }

    time++;//increase time

}
```

5 CONCLUSION

As a result, with MinHeap I was able to perform operations faster than a normal sort function.