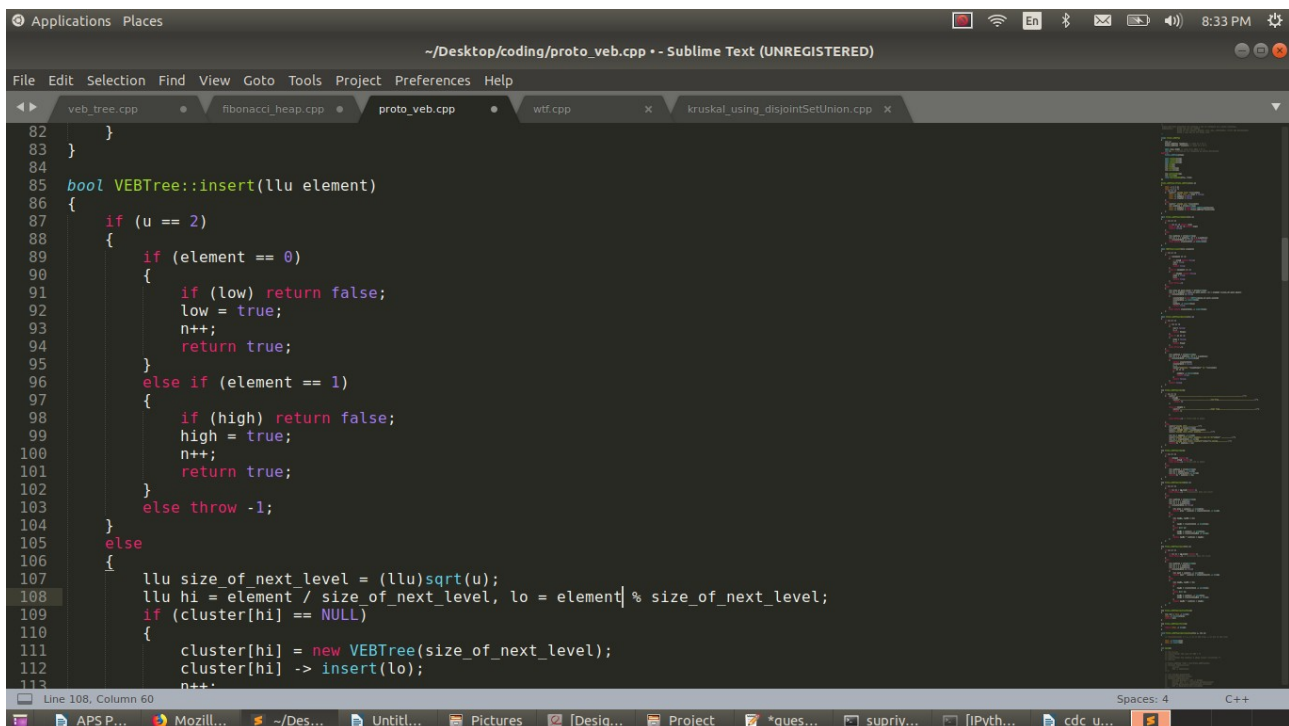


Comparison of implementations of Kruskal's Algorithm using van Emde Boas tree, Binomial Heap and Fibonacci heap

Rajat Kumar Verma & Supriya Priyadarshani
International Institute of Information Technology, Hyderabad

Deliverables

1. Studied the recursive data structure of Van Emde Boas Tree.
2. Implemented the insert() function of VEB tree.



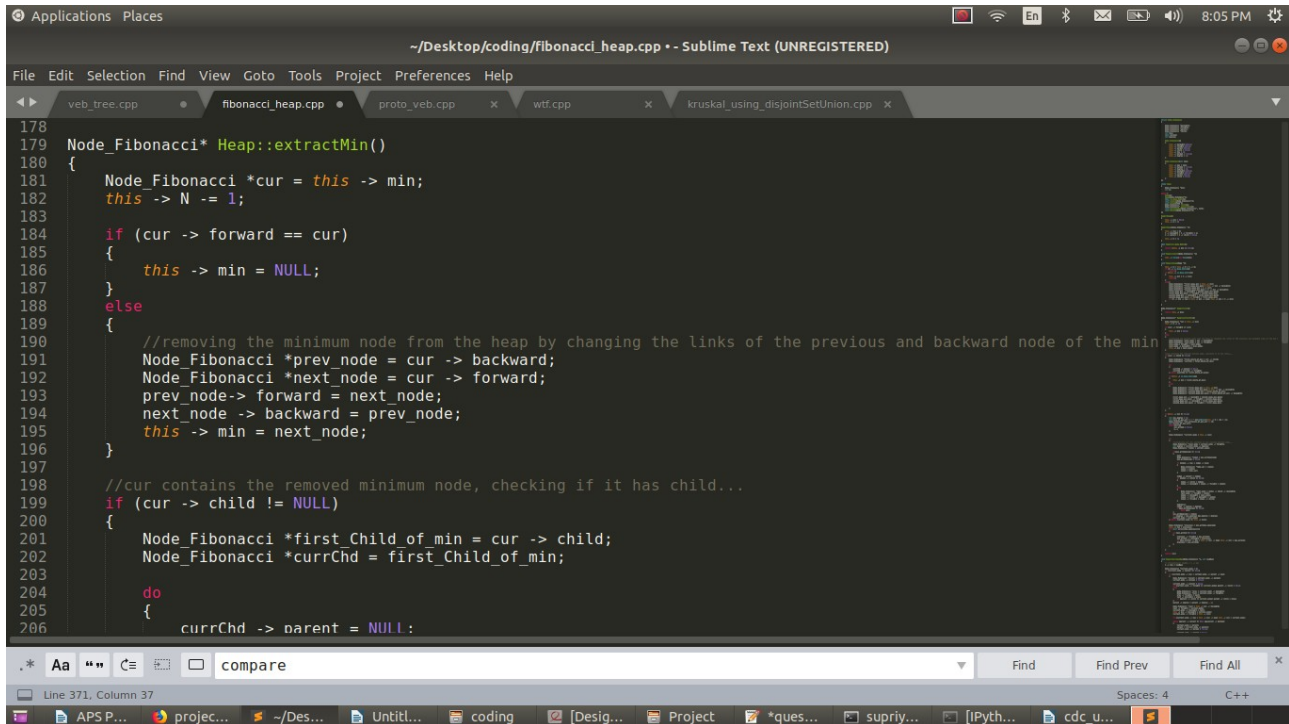
```
82     }
83 }
84
85 bool VEBTree::insert(llu element)
86 {
87     if (u == 2)
88     {
89         if (element == 0)
90         {
91             if (low) return false;
92             low = true;
93             n++;
94             return true;
95         }
96         else if (element == 1)
97         {
98             if (high) return false;
99             high = true;
100             n++;
101             return true;
102         }
103         else throw -1;
104     }
105     else
106     {
107         ll size_of_next_level = (llu)sqrt(u);
108         ll hi = element / size_of_next_level, lo = element % size_of_next_level;
109         if (cluster[hi] == NULL)
110         {
111             cluster[hi] = new VEBTree(size_of_next_level);
112             cluster[hi] -> insert(lo);
113             n++;
```

Time Complexity for insert is $O(\log \log u)$.

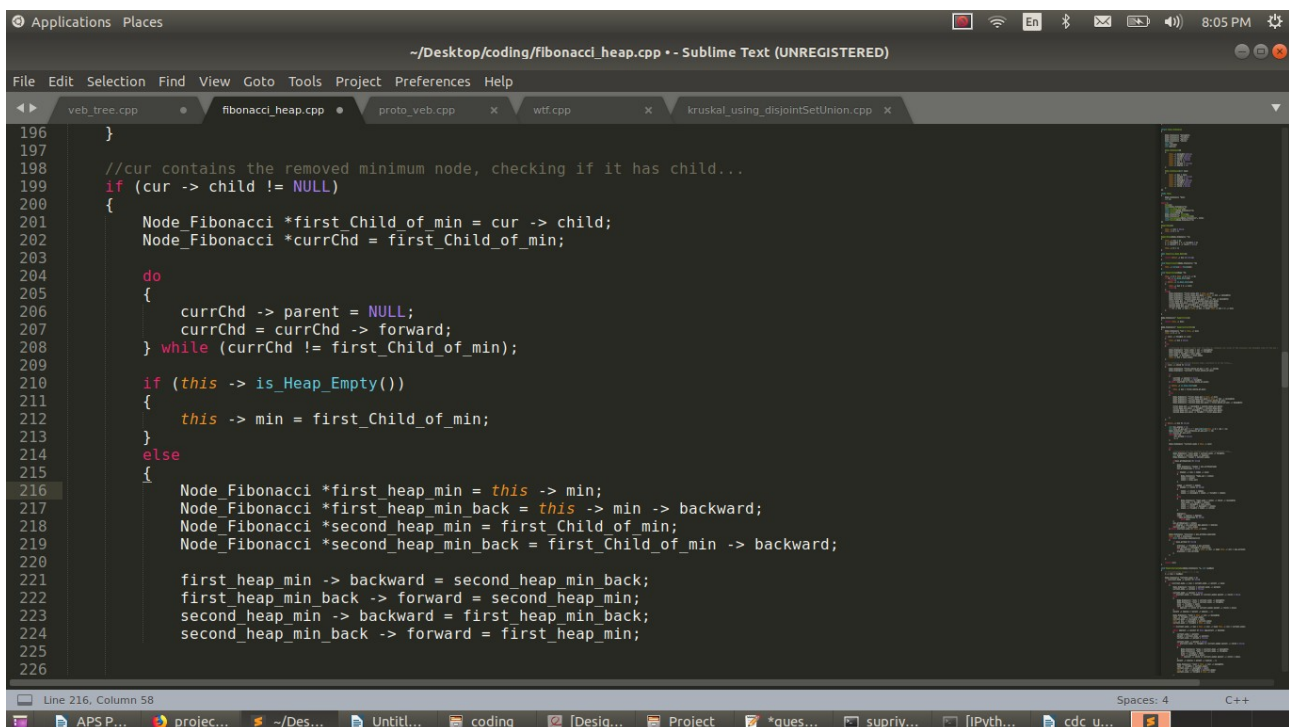
‘u’ is the universe, that is the total number of elements currently present in the array.

The size of array is always taken of the order $(2^2)^k$ because at every upper level during recursive call, the clusters are divided into $u^{1/2}$ clusters.

3. Implemented Kruskal using Fibonacci Heap.



```
178
179 Node_Fibonacci* Heap::extractMin()
180 {
181     Node_Fibonacci *cur = this -> min;
182     this -> min = NULL;
183
184     if (cur -> forward == cur)
185     {
186         this -> min = NULL;
187     }
188     else
189     {
190         //removing the minimum node from the heap by changing the links of the previous and backward node of the min
191         Node_Fibonacci *prev_node = cur -> backward;
192         Node_Fibonacci *next_node = cur -> forward;
193         prev_node -> forward = next_node;
194         next_node -> backward = prev_node;
195         this -> min = next_node;
196     }
197
198     //cur contains the removed minimum node, checking if it has child...
199     if (cur -> child != NULL)
200     {
201         Node_Fibonacci *first_Child_of_min = cur -> child;
202         Node_Fibonacci *currChd = first_Child_of_min;
203
204         do
205         {
206             currChd -> parent = NULL;
```



```
196     }
197
198     //cur contains the removed minimum node, checking if it has child...
199     if (cur -> child != NULL)
200     {
201         Node_Fibonacci *first_Child_of_min = cur -> child;
202         Node_Fibonacci *currChd = first_Child_of_min;
203
204         do
205         {
206             currChd -> parent = NULL;
207             currChd = currChd -> forward;
208         } while (currChd != first_Child_of_min);
209
210         if (this -> is_Heap_Empty())
211         {
212             this -> min = first_Child_of_min;
213         }
214         else
215         {
216             Node_Fibonacci *first_heap_min = this -> min;
217             Node_Fibonacci *first_heap_min_back = this -> min -> backward;
218             Node_Fibonacci *second_heap_min = first_Child_of_min;
219             Node_Fibonacci *second_heap_min_back = first_Child_of_min -> backward;
220
221             first_heap_min -> backward = second_heap_min_back;
222             first_heap_min_back -> forward = second_heap_min;
223             second_heap_min -> backward = first_heap_min_back;
224             second_heap_min_back -> forward = first_heap_min;
225
226         }
```

Above is the snapshot of a part of extractMin() function of Fibonacci Heap which takes $O(\log n)$ time to extract the current minimum from the heap. Repeating the process for 'n' number of times, returns us the sorted form of the input array which will take $O(n \log n)$ time. Other functions of

Fibonacci Heap, like, insert, first and merge take $O(1)$ time and, decreaseKey() function $O(1)$ time (amortized cost) .

Amortized cost = actual cost + change in potential cost.

Project Delivery Plan

1. Implementation of Fibonacci Heap and using the insert() and extract_min() operations implement Kruskal.
2. Implementation of Van Emde Boas Tree and using the insert(), delete() and extract_min() operations implement Kruskal.
3. Implementation of Binomial Heap and using the insert() and extract_min() operations implement Kruskal.
4. Study and compare the time complexities of all the implementations.

Technology Used

Plotly API

Online Resources

<https://www.youtube.com/watch?v=ZrV7GiuMNo4>
https://www.youtube.com/watch?v=_KgllZVMshg&t=909s
<https://www.youtube.com/watch?v=hmReJCupbNU&t=2196s>
<http://web.stanford.edu/class/archive/cs/cs166/cs166.1166/lectures/14/Slides14.pdf>

Git Repository

Repository Name : Comparison_Of_Algorithms
https://github.com/sup19-19/Comparison_Of_Algorithms.git

Testing Plan

Testing with the integer array of size greater than the size of RAM.

End User Documentation

There will be a menu provided to the user displaying options for comparison amongst the three algorithms on different parameters .

October 23, 2018