COL106 Assignment4
Name: Dipen Kumar
Entry No: 2018CS50098
Group Number: 3

Trie- This has four .java file-

1. Person.java- class which is stored in trie with key as its name. It has a field to store phone number.
2. TrieInterface.java- given to us as question to implement Trie.
3. TrieNode.java- class which has two field object of type generic T and a arrayList of TrieNode<T>.
4. TrieDriverCode.java- given by instructor. (Not to Change)
5. Trie.java-
    a. public boolean delete(String word)- search the word to delete and store the last encountered junction. If this word is prefix of some otherword than just make the obj at that TrieNode=null else delete make TrieNode=null at junction. Return true if deleted else false. Complexity is O(95*n) where n is the length of the word.
    b. public TrieNode<T> search(String word)- search the word in O(n).
    c. public TrieNode<T> startsWith(String prefix)- all similar to search() only difference here is we also return node if it has obj==null. Complexity O(n).
    d. public void printTrie(TrieNode trieNode)- done recursively. Depth first search. Complexity O(size of the graph) where size is number of nodes where here will also count null TrieNode.
    e. public boolean insert(String word, Object value)- search with the word and give value to that TriNode. Return true if value was earlier null else false because duplicate key trying to insert. Complexity O(n) where n is size of the word.
    f. public void printLevel(int level)- done recursively. I am going to all children of the root which is one level higher than the root and now I am calling this function with its children as root and level=level-1. When I get level=0, I print its character. I also terminate the recursion when no children left but here don't print the character print empty line. Complexity is similar to depth first search O(n). here n will not be entire graph but till required level. Complexity for sorting lexographically is O(n^2).
    g. public void print()- here I used printLevel method to print level from 1 to the level I get empty line. Complexity is O(n^2) where n is tree size.

RedBlack Tree- Two method implemented are-

1. public void insert(T key, E value)- I searched where to insert the node. Complexity of search binary tree with height here of order log(n) will be O(log(n)). I then made three case. Parent is black easiest case insert and return. When parent is red and uncle is not red, restructuring and return. Last case both parent and uncle are red. Here first recoloring is required and then again we check all these three cases with current=current.parent.parent. Complexity will be as worst as going to top that is again travelling back that is approx. we have travelled down and up. Hence effective height 2log(n). hence O(log(n))
2. public RedBlackNode<T, E> search(T key)- Search is same as any other binary tree. Go left child if value is smaller than key else if greater than key than go right else this is the required key you are searching. Complexity is O(log(n)).

Priority Queue- I have make one extra class Node<T>. It has a field to store an object of T and also a integer value. Basically this integer value it the time at which T object is inserted in the FIFO_maxheap/priority_Queue. Two method implemented are-

1. public void insert(T element)- I used a arrayList of Node<T> to implement maxheap. When insert is called I add Node at the bottom of array. I then bubble up if my insert value is strictly greater than root else I don't bubble up. If I reach root i.e. index=0, I stop. Complexity is O(log(n)).
2. public T extractMax()- Here I return the value at root i.e. at index=0. Now I have to restructure this array to make it again a maxheap. I do, I take a element at last index and put it at index=0. I then bubble down. In case of conflict when two have same priority, preference is given to those who have time = lesser i.e. who have come earlier i.e. first in first out. Complexity is O(log(n))

Project Management- There are three class Project contains name, priority and budget. User contain name. Job contain name, project, user, runtime, jobstatus and end time. I have used seven data structures-

1. Trie<Project> trie to store project when created.
2. Trie<User> trie2 to store user when created.
3. Trie<Job>trie3 to store job when created.
4. MaxHeap<Job> maxheap to store job when created. Also when added from rbtree when budget is increased. Deleted when handle_emptyline

and run_to_completion and these values were stored either in List<Job> printc or rbtree.

5. RBTree<Project,Job> rbtree
6. List<Job> printc
7. MaxHeap<Project> printr

Methods-

1. Handle_project- create a project and store in trie and printr
2. Handle_user- create a user and store in trie2
3. Handle_job- create a job and store in trie3 and maxheap
4. Handle_query- query for a job search a job from trie3.
5. Handle_empty_line- takes a job from maxheap and execute it and put it in printc i.e. list of Job for first in first out output of completed job. If budget is less put it in rbtree i.e. not ready queue. Update the status of Job, endtime, global time, decrease its corresponding project budget.
6. Handle_add- search that project from trie and increase its budget. Also search that project from rbtree and insert all jobs in list in maxheap.
7. Handle_schedule- takes a job from maxheap and execute it and put it in printc i.e. list of Job for first in first out output of completed job. If budget is less put it in rbtree i.e. not ready queue. Update the status of Job, endtime, global time, decrease its corresponding project budget.
8. run_to_completion- call schedule if job is remaining.
9. Print_stats- use printc which has stored completed jobs and print it from index i=0 to last to get FIFO order. For unfinished job to print. I first take a project from printr which is a heap of project to ensure to get highest priority project then I search that job in rbtree and print entire list of pending jobs in FIFO manner.

Note: Various operations of used data structures are already mentioned above. Hence, I skipped that in order to make it a sort and sweet report.