**CS 218 DATA STRUCTURES**

# ASSIGNMENT 3

***SECTION SE and DS***

**Fall 2021**

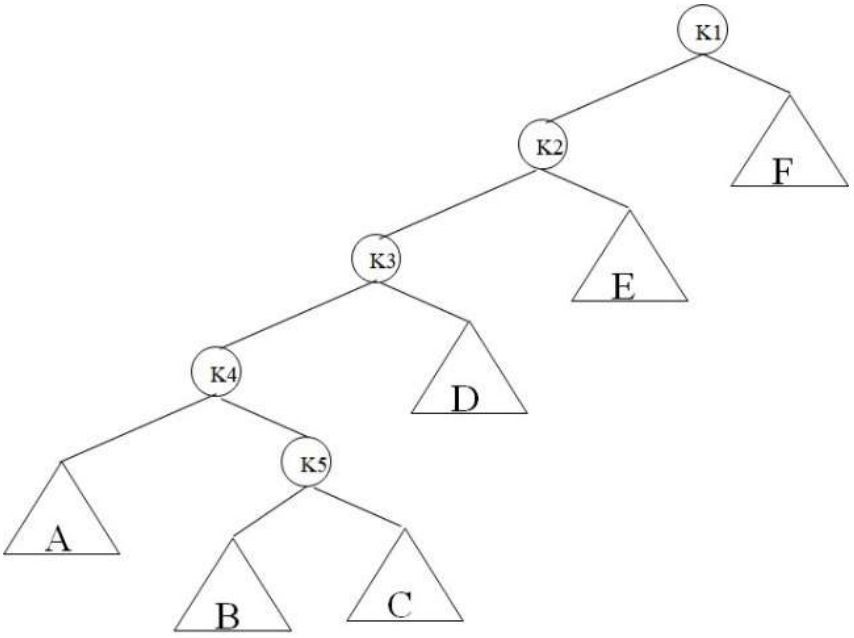
**DUE**: Dec 9, 2021

**SUBMISSION:** Upload the structured and well-written code in C++ on the google classroom. Undocumented (uncommented) code will be assigned a zero.

**PROBLEM BACKGROUND**

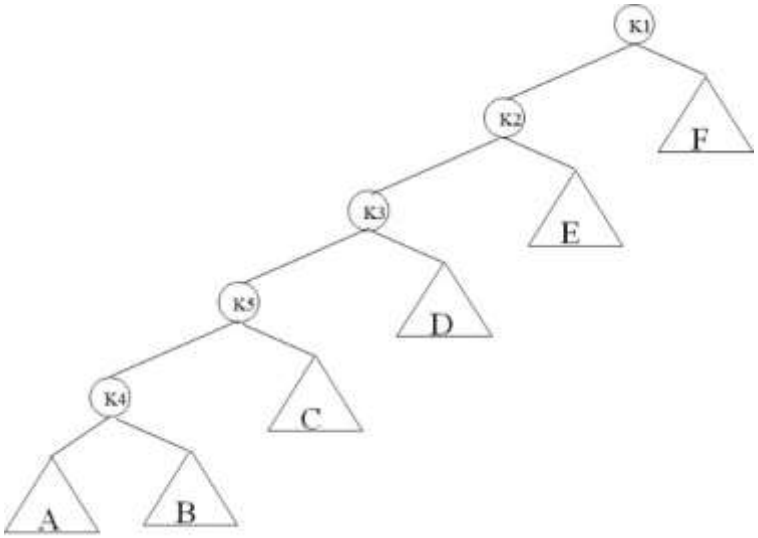
In the real world applications, most of the time only a chunk of the stored data is frequently used. For example, a hospital maintains the data of thousands of patients, but only the records of the currently admitted patients are accessed more frequently. In such a scenario, we can make our search operation efficient by placing frequently accessed data items near the root node. We modify the implementation of binary search tree such that recently accessed items are moved near the root node (these items are highly likely to be accessed again). This way, we can increase the efficiency of the search operation.

A simple way to move a particular data item to the root node is by rotating nodes on the search path with its parent node. Consider the following example where we want to move node with data item **K5** near the root node.

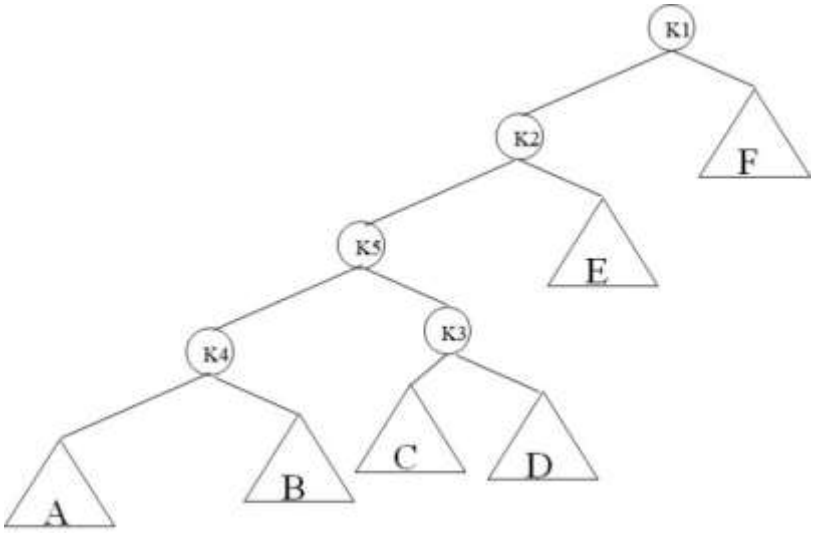



Here level of **K5** is **4** (root is at level 0), and we want to move **K5** to level 1. The following operations are performed.

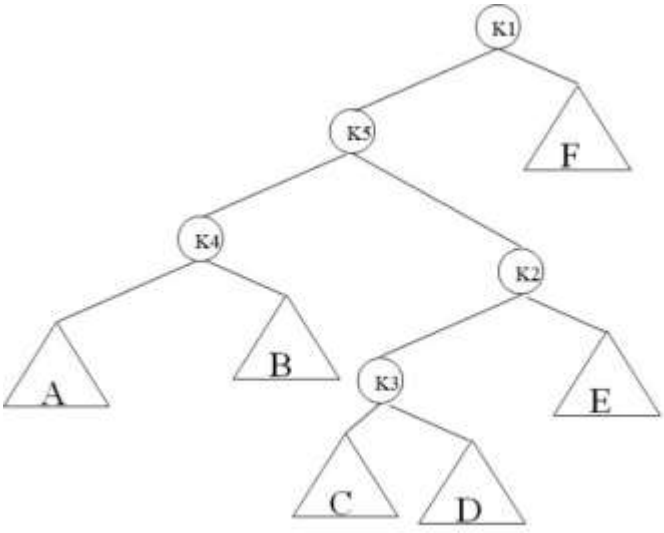
1




2




3



## REQUIREMENTS and IMPLEMENTATION

We want to design software for hospital management that can efficiently search the patients' data. The records of the patients are stored in a ***special binary search tree (BST), where*** the search operation is modified as discussed above. Each patient's record includes a patient id, patient name, admission date, disease diagnosis, and patient status (admitted/discharged) in the hospital management system. The BST tree will be constructed on patient Ids.

**IMPORTANT CLASSES**
You have to implement the following classes

**Class PatientRecord**
This class must have the following data members

- patient id
- patient name
- admission date
- disease diagnosis
- status (admitted/discharged)

**Class TNode**
This class must have the following members:

- Patient record
- left child
- right child

**Class HospitalData**
This class must implement the following data members and member functions:
**Data Members:**

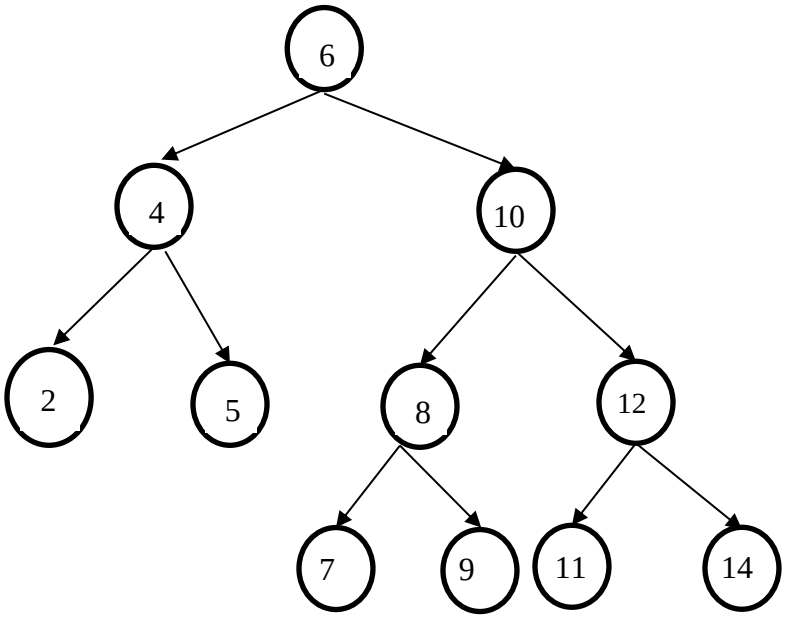
- Pointer to the root of the tree
- Size (number of records in the tree)

**Member function:**

1. **Constructor**
2. **Destructor**
3. **Insert:** This function takes a patient record as an input parameter, inserts it in the BST based on the patient id, and moves the newly inserted node to the root. It is assumed that the newly inserted node will be accessed shortly.
4. **Remove:** This function takes a patient id as an input parameter and removes the patient record from the tree.
5. **Print:** This function takes patient id (pid) as input parameter and outputs the record.
6. **PrintAdmitted:** This function outputs all the records of patients whose status is admitted in sorted order according to the pid.
7. **Search:** This function takes a patient ***pid*** and level number ***k*** as input parameters. It searches the patient with the given pid and returns the patient record if the patient exists. Otherwise, it must indicate that patient does not exist. Moreover, if a patient record is found, then move it to level k. *The example above shows the result of a call that searched node k5 is searched and moved to level 1.* If k is larger than the level of the desired patient record, then it will not be moved.
8. **Split:** This function splits the given BST tree into two BST trees in linear time (O(n)) and returns the second tree.

**Split Function details**

*The split function divides the patient record into two BSTs of the same size.* This operation finds the median of all the patient ids and construct two BSTs. One BST must contain records less than the median value, and the other BST must include records greater than the median. This operation must be performed in O(n), where n is the total number of patients. This can be done by moving the median element to level 0, and its left and right subtrees will be the desired trees. Consider the following sample tree. Its median value is 8.




The resultant tree after moving node 8 to level 0 will be