Ex2. (a) To find the minimum/maximum key, we need to find the most left child/ most right child which contain the key. The first(smallest) element in the most left child node is the minimum and the last(largest) element in the most right child node is the maximum.

To find the most left child/ most right child, we just need to search the left/ right child of the node (start from the root) until we reach a leaf node.

(The differences of code implementation should be considered when doing this operation. In our implementation, we store the maximum key value in the tree in the private property of the tree and update it whenever we insert or delete a key, so instead of the above operation, we can get the maximum key value by the operation **tree->maxKey**. Also, we have an pointer to the most left child node, i.e. the leaf node with the smallest key, so we can get the minimum key value by the operation **tree->leafHead->key**).

If the tree is empty, then there is no minimum/maximum key.

(c) First, we should find the node with key *k*. Since the leaf nodes are connected in B+ tree, we just start from the head of the leaf node, search the key in order.

To find the predecessor key:

If *k* has an index $i > 0$ (i starts from 0) in the node, we search the key with index *i-1* in the same node.

If *k* has index $i = 0$ (which means k is the first element in the node), we search the last key in the previous node. Since the leaf nodes are connected, it is easy to search the previous node. If there is no previous node, then there is no such predecessor key in the tree.

To find the successor key:

Assume the node has *n* keys.

If *k* has an index $i < n-1$, we search the key with index *i+1* in the same node.

If *k* has index $i = n-1$ (which means k is the last element in the node), we search the first key in the next node. If there is no next node, then there is no such successor key in the tree.