Untitled

HTML Content

1. **Red–black BST with no extra memory.** Describe how to save the memory for storing the color information when implementing a red–black BST. 1 1

Note: these interview questions are ungraded and purely for your own enrichment. To get a hint, submit a solution.

Modify the structure of the BST to encode the color information.

Correct

Hint: modify the structure of the BST to encode the color information.

2. **Document search.** Design an algorithm that takes a sequence of n document words and a sequence of m query words and find the shortest interval in which the m query words appear in the document in the order given. The length of an interval is the number of words in that interval. 1 1

```
public static void main(String[] args) {
In in = new In("");
String[] words = in.readAllStrings();

ST<String, Queue<Integer>> windices = new ST<String, Queue<Integer>>();

for (int i = 0; i < words.length; i++) {
if (!windices.contains(words[i])) {
Queue<Integer> tmp = new Queue<Integer>();
tmp.enqueue(i);
windices.put(words[i], tmp);
}
else {
Queue<Integer> tmp = windices.get(words[i]);
tmp.enqueue(i);
windices.put(words[i], tmp);
}
}

int bestlo = -1;
int besthi = words.length;
String[] query = StdIn.readAllStrings();
Queue<Integer>[] queues = (Queue<Integer>[]) new Queue[query.length];
```

```
for (int i = 0; i < query.length; i++) {
queues[i] = windices.get(query[i]);
}

Queue<Integer> starts = windices.get(query[0]);

for (Integer start: starts) {
boolean end = true;
int lo = start;
int hi = lo;

for (int i = 1; i < queues.length; i++) {
while (!queues[i].isEmpty() && queues[i].peek() <= hi) queues[i].dequeue();
if (queues[i].isEmpty()) {
end = false;
break;
}
else {
hi = queues[i].peek();
}
}
if (end && hi - lo < besthi - bestlo) {
besthi = hi;
bestlo = lo;
}
}

if (bestlo >= 0) {
int interval = besthi - bestlo;
System.out.println("Shortest interval found: " + interval);
}
else {
System.out.println("Not found");
}
}
```

Correct

Hint: for each word, maintain a sorted list of the indices in the document in which that word appears. Scan through the sorted lists of the query words in a judicious manner.

3.

Generalized queue. Design a generalized queue data type that supports all of the following operations in logarithmic time (or better) in the worst case. 1 / 1pc

- Create an empty data structure.
- Append an item to the end of the queue.
- Remove an item from the front of the queue.
- Return the i^{th} item in the queue.
- Remove the i^{th} item from the queue.

```
class GeneralizedQueue<Item> {
private int index;
private RedBlackBST<Integer, Item> store;

GeneralizedQueue() {
index = 0;
store = new RedBlackBST<Integer, Item>();
}

public void append(Item item) {
store.put(index++, item);
}

public void removeFront() {
store.deleteMin();
}

public Item get(int i) {
int key = store.rank(i);
return store.get(key);
}

public void delete(int i) {
store.delete(store.rank(i));
}
}
```

Correct

Hint: create a red–black BST where the keys are integers and the values are the items such that the i^{th} largest integer key in the red–black BST corresponds to the i^{th} item in the queue.