Binary Search Tree Lexicon

Implement a `lexicon` class to represent a text dictionary using a binary search tree (BST). The class must support the following operations:

```
class lexicon {
public:
    lexicon();
    ~lexicon();

    void insert(const string &s);
    int lookup(const string &s) const;
    int depth(const string &s);
    void replace(const string &s1, const string &s2);

    friend ostream &operator<<(ostream &out, const lexicon &l);
};
```

Requirements

1. **Word Constraints**:
   - Words are **non-empty** and contain **only lowercase Latin letters**.
   - Each word is stored in a unique BST node with its frequency (count of insertions).
2. **BST Structure**:
   - Left child: lexicographically smaller words.
   - Right child: lexicographically larger words.
   - The BST is **not required to be balanced**
3. **Methods**:
   - **`insert(s)`**: Adds `s` to the BST. If `s` exists, increments its frequency.
   - **`lookup(s)`**: Returns the frequency of `s`. Returns `0` if `s` is not found.
   - **`depth(s)`**: Returns the depth of `s` (root is at depth `0`). Returns `-1` if `s` is not found.
   - **`replace(s1, s2)`**: Replaces all occurrences of `s1` with `s2`:
     - If `s1` does not exist, do nothing.
     - If `s1` has frequency `k`, delete `s1` and update `s2` (insert `s2` with frequency `k` if it doesn’t exist; otherwise, add `k` to `s2`'s frequency).
     - **Deletion Rules**:
       - Node with two children: Replace with the **in-order predecessor**.
       - Node with one child: Replace with its child.
4. **Output Format**:
   - The `<<` operator prints words in **lexicographical order**, followed by their frequency.

Example Usage

```cpp
int main() {
    lexicon l;
    l.insert("the");
    l.insert("boy");
    l.insert("and");
    l.insert("the");
    l.insert("wolf");

    cout << "The word 'the' is found " << l.lookup("the") << " time(s)" << endl;
    cout << "The word 'and' is found at depth " << l.depth("and") << endl;
    cout << l;

    l.replace("boy", "wolf");
    cout << "After replacement:\n";
    cout << l;
    cout << "Now the word 'and' is found at depth " << l.depth("and") << endl;
}
```

Expected Output:

```
The word 'the' is found 2 time(s)
The word 'and' is found at depth 2
and 1
boy 1
the 2
wolf 1
After replacement:
and 1
the 2
wolf 2
Now the word 'and' is found at depth 1
```

Notes:

- Ensure your code is **memory-efficient** (no leaks).
- Test edge cases (e.g., deleting nodes with 0/1/2 children).