University of Waterloo
CS240, Winter 2023
Programming Question 2

Due Date: Wednesday, March 29, at 5pm

If you are having difficulties with the assignment, course staff are there to help (provided it isn't last minute).

Please read `https://student.cs.uwaterloo.ca/~cs240/w23/assignments.phtml#guidelines` for guidelines on submission. Submit the file `CS240KatieMap.cpp` to Marmoset.

It is a good idea to submit questions as you go so you aren't trying to create several PDF files at the last minute.

Late Policy: Assignments are due at 5:00pm, with the grace period until 11:59pm. Assignments submitted after 11:59 on the due date will not be accepted but may be reviewed (by request) for feedback purposes only.

Problem 1 Question [20 marks]

Implement `CS240KatieMap`, a class to construct and manipulate a kd-tree that also gets used as a dictionary. Points in a `CS240KatieMap` are key-value pairs of the form (k, v) where k is a `std::string` and v is an `int`. The first "coordinate" is the key, and the second "coordinate" is the value. Keys and values should be compared using standard comparison operators like `<`, `>=`, and `==`. The KVPs are represented using `std::pair`.

You must implement the `public` functions described on the next page. You may freely add additional `private` members and auxiliary data types.

You may use any of the libraries included at the top of the starter file. You may not use features from `<algorithm>`, any other container, or any other header that would trivialize the problem (even if they are included indirectly through one of the other headers). Do not `using namespace std;`, but you may use individual items from `std`. You must implement your own selection algorithm.

Place your program in file `CS240KatieMap.cpp`. We provide you with starter code that has a main function, accepting commands from standard input. You may assume all inputs we provide in testing are valid: we will never query a rectangle with width or height 0, and you will not be asked to construct a tree with two points on the same vertical or horizontal line. Furthermore, you may assume that all string keys will have size ≤ 16, so that all string operations are constant time. **Do not modify the `main` function in your submission.**

Remember the convention that points on a line belong to the region to the right/top of the line, i.e., if we split the interval $[a, c)$ at a coordinate b, then the split yields intervals $[a, b)$ and $[b, c)$. The topmost split must be on the key, i.e. on the string component of the pairs.

```cpp
using std::vector;
using std::string;
using kvp = std::pair<string, int>;

class CS240KatieMap {
  public:
    // Constructor: Builds a new tree on the KVPs provided.
    //              Runs in O(n log n) expected time.
    CS240KatieMap(vector<kvp> kvps);

    // all_points: Returns all KVPs in the tree, in the order they are
    //             visited in preorder traversal.
    vector<kvp> all_points();

    // get: Returns the value, if any, stored at the given key.
    //      Returns std::nullopt if there is no such key.
    //      Runs in O(sqrt(n)) time.
    // Note: Do not implement a separate dictionary data structure,
    //       or modify your kd-tree structure, in order
    //       to support get().
    std::optional<int> get(string key);

    // range: Query points in the "rectangle" defined by start and end.
    //        Specifically, if start = (key_start, value_start) and
    //                             end = (key_end, value_end)
    //        returns all points in the range
    //              [key_start, key_end) x [value_start, value_end)
    //        Runs in O(sqrt(n) + s) time
    //              where s is the size of the returned vector.
    vector<kvp> range(kvp start, kvp end);

    // key_splits: Return all keys (strings) that are used for splits,
    //             in the order they are visited in preorder traversal.
    //             Runs in O(n) time.
    vector<string> key_splits();

    // value_splits: Return all values (ints) that are used for splits,
    //               in the order they are visited in preorder traversal.
    //               Runs in O(n) time.
    vector<int> value_splits();

    // Destructor: Destroys the kd-tree, cleaning up all resources.
    //             Must not leak memory.
    ~CS240KatieMap();
};
```

Commands

The following commands are supported by the starter file main program. (You may modify the starter file to support additional commands as part of your own debugging, or to run tests, etc., but you must ensure that your submission has the unmodified 'main' function.)

All input is done using `std::cin >> variable`, so each value is a whitespace-separated word. Thus, keys cannot contain whitespace.

n Initializes a new `CS240KatieMap`.

Reads in one integer n indicating the number of points (key-value pairs) in the map. Then reads in n pairs, each one string followed by one integer. Calls the `CS240KatieMap` constructor on the points read in.

r Performs range search.

Given input `r k1 v1 k2 v2`, calls `CS240KatieMap::range(make_pair(k1, v1), make_pair(k2, v2))`. The resulting pairs are printed one on each line.

g Calls `CS240KatieMap::get()` on a string key read from the input.

Prints the value found, if any, or "Not found" if none is found.

p Calls `CS240KatieMap::all_points()`.

The resulting pairs are printed one on each line.

k Calls `CS240KatieMap::key_splits()`.

The resulting pairs are printed one on each line.

v Calls `CS240KatieMap::value_splits()`.

The resulting pairs are printed one on each line.

q Destroys the allocated map and quits.

There must not be any memory leaks.