Please don't look around or talk with others. Cheating results in 0 credits!

2018-6-15

1. Examination time: 14:00-16:00.
2. Submission time: Please start to submit your source code by Tsinghua Web Learning (网络学堂) at least before 16:10. The submission site will be closed at 16:10. You may also submit your source code by USB drive to our TA if you cannot access internet. Please NOTE that late submissions are NOT accepted.
3. Submission requirement: Please follow the rules for our homework submission.

1. Segment tree (Credits 65%)

In computer science, a segment tree also known as a statistic tree is a tree data structure used for storing information about intervals, or segments [1]. It allows querying which of the stored segments contain a given point. For example, given the following segments: *[0, 3), [2, 5), [3, 7), [5, 8), [2, 8), [8, 9), [8, 18), [12, 18)*, a query on point *3* returns the following segments: *[2, 5), [3, 7), [2, 8)*.

A segment tree *T* is a binary tree. Its leaves correspond to the elementary intervals in an ordered way: the leftmost leaf corresponds to the leftmost interval, and so on. The internal nodes of *T* correspond to intervals that are the union of elementary intervals: the interval *Int(N)* corresponding to node *N* is the union of the intervals corresponding to the leaves of the tree rooted at *N*. Each node in *T* corresponds to the segments that span through its interval, but do not span through the interval of its parent. It is popular to implement a segment tree using an array due to its high efficiency. Attached is a sample implementation of the segment tree. Please improve the segment tree according to the following requirements:

(1) After calling `tree.search(3);` in main.cpp, there are 3 overlapping segments. Add a member variable named "`segIDs_`" in class SegmentTree to store the overlapping segments' ids. Then use stream operator overloading (`std::cout << tree << std::endl;`) to print the ids of the overlapping segments. Please note that each time

function tree.search is called, segIDs_ should be cleared first to store the new results. (10%)

(2) Using the added variable segIDs_, implement a member function to count the number of overlapping intervals for a given query point: std::cout << tree.count(3) << std::endl; (10%)

(3) Instead of adding member variables (segIDs_), implement a function object to obtain the ids of the overlapping segments. You may add new member functions if needed. A suggested format of using the function object is as follows: void SegmentTree::search(const int& point, BaseFuncObj& callBackFunObjIds) const; Then the ids could be printed by cout << callBackFunObj << endl;. (10%)

(4) Implement two more function objects. One is for directly counting the number of overlapping segments; the other is for immediately (as soon as the matched segment is found) printing the matched overlapping segments to standard output (prints the segment's start_, end_ and id_，每找到一个匹配就及时打印匹配信息). Using **polymorphism**, make all the three function objects work together with the following single search function:

void SegmentTree::search(const int& point, BaseFuncObj& callBackFunObj) const;

Moreover, the three function objects could also be printed by cout << callBackFunObj << endl;.

(20%)

(5) Make templates named TSegmentTree and TNode, which accept different types of "start_ and end_ (i.e. long), and different types of id_ (i.e. string). Test the templates using the above-mentioned long and string types. (15%)

2. Code reuse (Credits 35%)

From Wikipedia, the definition of a trie is as follows: "In computer science, a trie, also called digital tree and sometimes radix tree or prefix (前缀) tree (as they can be searched by prefixes), is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings (以字符串为查询关键字)."

Read the given source code of a basic trie implementation and try to understand the code. Then write a subclass named MyTrie of the base class BaseTrie by public inheritance. The goal is to reuse the searching functions in class BaseTrie to search for all the strings stored in the trie structure matching a given prefix (在 trie 结构中搜索匹配某个前缀的所有字符串).

Requirements:

(1) Write MyTrie.h and MyTrie.cpp for subclass (子类) MyTrie (check main.cpp for the required interfaces); (10%)

(2) Add new code into main.cpp to test class MyTrie:

(a) Randomly generate 15 character strings of 8 alphabets (a~z) in length (随机生成 15 个字符串，每个字符串长度为 8 个字符，且由字母 a 至 z 组成). Insert the 15 character strings into the object of MyTrie, with each string corresponding to a different integer key value (每个字符串对应不同整数键值); (5%)

(b) Search all the character strings starting with a given prefix string (e.g. "a", "b", "c"), and print the strings to visually check (通过打印信息检查结果的正确性) that results are correct. (5%)

(3) Write the destructors to avoid memory leaks (you may modify class BaseTrie if needed). (5%)

(4) Use Adapter design pattern to implement another class named SuffixTrie, which adapts MyTrie for searching suffix strings (搜索匹配某个后缀的所有字符串). Randomly generate the character strings, and use a given suffix string for testing the correctness of the design. (10%)