Middle East Technical University | Department of Computer Engineering

CENG 242

Programing Language Concepts

Spring 2011-2012

Homework 4

Due date: 8 May 2012, Tuesday, 23:55

1 Objectives

In this homework you are going to implement the Dictionary Abstract Data Type you worked on in previous homework. This time, you will implement this as a class in C++, and include an additional functionality, namely, the list of synonyms.

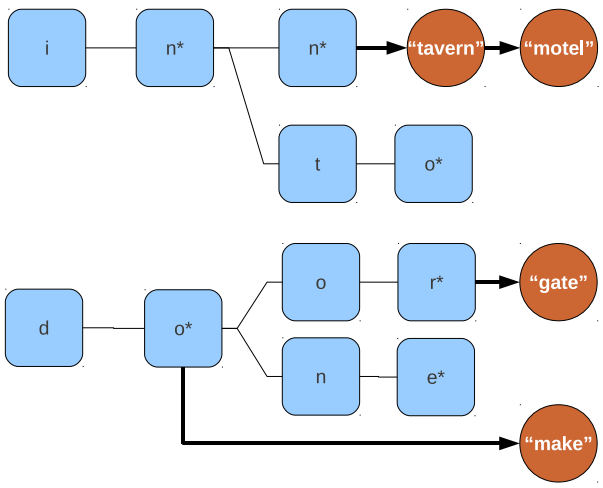

Figure 1: An example prefix tree with linked lists for possible synonyms for words

Just like in the previous homework, you will implement the dictionary as a prefix tree. During check of a list of symbols in prefix tree lookup, you start traversal from the root and based on the next symbol in the list you follow one of the branches. If no branches match the next symbol, search fails, thus the list is not included in the Dictionary. When there is no symbol left in the list and the node currently visited is final (shown as * in the Figure 1) the lookup is successful.

At the nodes of the tree, include a pointer to a linked list. This linked list should contain the synonyms (if exists) of the word that ends with the item of that node and starts with the item of the root of the tree this node belongs.

2 Tasks

You are going to write your module in a file called Dictionary.cpp. While you are free to define additional methods and/or variables, the following definitions must be made within your class:

```
/*
This constructor will create an empty dictionary.
*/
Dictionary();

/*
This constructor will create a dictionary containing all
the words in the given list.
*/
Dictionary(const std::vector<const char*> &list);

/*
  This is the copy constructor. Here, you should
  generate a Dictionary object such that it contains the
  same members as Dictionary d has.
*/
Dictionary(const Dictionary &d);

// The destructor
~Dictionary();

/*
Operator overloading:
The operators +,= and - should be overloaded such that they
can be used for merge, memberwise assignment of two Dictionary objects, and subs;
respectively
*/

/*
This function will merge
the dictionary in the first argument into the current dictionary.
This operator does not modify the original object.
*/
Dictionary operator+(const Dictionary& other) const;

/*
This function take the difference of the Dictionary object from the input
dictionary (b), and return it as a separate Dictionary object. This operator
does not modify the original object.
*/
```

```
Dictionary operator-(const Dictionary& other) const;

/*
  Memberwise assignment.
*/
Dictionary& operator=(const Dictionary& other);

/*
These are the versions of - and + where the result
is set to the original Dictionary object.
*/
Dictionary& operator-=(const Dictionary& other);
Dictionary& operator+=(const Dictionary& other);


/*
This function will check if a given item exists in the
dictionary. If it exists, check will return true.
*/
bool check(const char* item) const;

/*
This function will insert a given item to the dictionary
in the first argument. If the item already exists in the dictionary,
it is skipped, and the function should return false.
*/
bool insert(const char* item);

/*
This function will return an array of Item objects containing
all the words in the dictionary.
*/
std::vector<const char*> toList() const;

/*
This function will insert a list of synonyms to the entry itemOfInterest
in the dictionary, and return true for success.
If no such entry exists, it should return false.
*/
bool addSynonyms(const char* itemOfInterest, const std::vector<const char*> &synonyms);

/*
Implement the show function that complies with the explanation in the Sample I/O section.
*/
void show() const;

/*
In addition to the show function, you are also going to implement showSynonyms.
Given an item, this function should show all of its synonyms, ONE PER LINE. Return false
```

```
if the entry does not exist.
*/
bool showSynonyms(const char* itemOfInterest) const;
```

Since your codes are evaluated in a black-box fashion, it is critical that you implement the functions `show` and `showSynonyms` correctly.

Remark: Note that merge(operator +) and subs(operator -) operations are best carried out in prefix tree structures. Converting tree to list then inserting or deleting from first list is not an efficient solution. Your solutions will be tested on large dictionaries and if slow, you might loose points.

For the definition and usage of std::vector, check the following useful websites:

http://en.cppreference.com/w/cpp/container/vector

http://www.codeguru.com/cpp/cpp/cpp_mfc/stl/article.php/c4027/C-Tutorial-A-Beginners-Guide-to-stdvector-Part-1.htm

3 Sample I/O

Sample inputs and outputs will be posted in the upcoming days. On the other hand, note the following remarks:

- For `showSynonyms`, each synonym should be printed in *separate* lines.
- For `show`, you should print such that the letter to be printed will be indented with the number of '_' characters equal to its level. For example, the `show` output for a dictionary that contains only the words *tee* and *tea* should be:

```
_t-->
__e-->
___a-->*
___e-->*
```

4 Regulations

4.1 Honesty

We have zero tolerance policy for cheating and plagiarism. People involved in these acts will be punished according to the university regulations.

4.2 Newsgroup

You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates at least on a daily basis.

4.3 Evaluation

Your program will be evaluated automatically using black-box technique so make sure to obey the specifications. Your codes will be tested on inek machines, using valgrind with `--leak-check=full`. Make sure your code runs correctly and has no memory leakages - in other words, be sure to properly implement the destructor, and be wary of the memory management issues.

4.4 Restrictions

You must implement Dictionary using a prefix tree. Although evaluation will be blackbox, we will also examine your codes to make sure you implement Dictionary using a prefix tree. You are not allowed to import libraries other than stdio, stdlib and vector. You may NOT use string library, as the memory management is a concern that we would like to inspect.

4.5 Submission

You must submit your codes via COW. You must submit your source code compressed in hw4.zip. that contains the implementation for your Dictionary module.

4.6 Late Submission

See the Course Webpage for Late Submission policy.