BLG 336E - Analysis of Algorithms II, Spring 2017

Project 2

Total Worth : 10% of your grade

Handed Out : 04.04.2017 Tuesday

Due : 18.04.2017 Tuesday - 23:59

Overview

Syntax analysis or syntactic analysis is the process of analyzing a string of symbols, either in natural language or in computer languages, conforming to the rules of a formal grammar which is also referred to as ***Parsing***. In natural language processing approaches, parsing is the analysis of a sentence or another string of words into its constituents, resulting in a parse tree showing their syntactic relations with each other.

You will implement a divide and conquer based parser to analyze the given strings to a parse tree and show the relations between the words. You will be given a set of language rules and you will design your divide and conquer based parser based on these rules. Your implementation must be in accordance with the divide and conquer methodology, first dividing the given sentence in two window size parts and then joining sub-trees and adding to parse tree and continue until the sentence is fully parsed. For an example, see the **RUN** section.

POS Tag Set:

DT (determiner), **NN** (nominal), **VR** (verb), **PR** (preposition), **AD** (Adjective),

Non-Terminals:

NP (noun phrase), **PP** (prepositional phrase), **VP** (verb phrase), **S** (sentence)

Grammar and Binary/Positional Transformation Rules:

$NP \rightarrow NN\ NN$	$PP \rightarrow PR\ NN$	$VP \rightarrow VR\ PP$	$S \rightarrow NP\ VP$
$NP \rightarrow AD\ NN$	$PP \rightarrow PR\ NP$	$VP \rightarrow NN\ VR$	$S \rightarrow NN\ VP$
$NP \rightarrow AD\ NP$		$VP \rightarrow NP\ VR$	
$NP \rightarrow DT\ NP$		$VP \rightarrow VR\ NP$	
$NP \rightarrow DT\ NN$			

POS Tags of Words:

DT → that | this | a | the
NN → book | flight | cat | mat | I | you | they
VR → booked | included | preferred | sat
PR → from | to | on | near | through
AD → big | heavy | beautiful | cheap

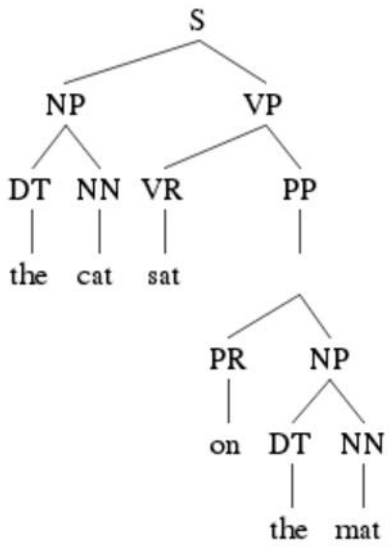
Input Sentence:

the cat sat on the mat

Output Sequence:

S[**NP**[**DT**(the) **NN**(cat)] **VP**[**VR**(sat) **PP**[**PR**(on) **NP**[**DT**(the) **NN**(mat)]]]]

Parse Tree: *The parse tree is not required to be printed in project—it is given as an example. You need to print the* ***output sequence*** *for your output.*



Requirements:

- Use and define the limited POS tag set, grammar rules and binary/positional transformation rules in your code.
- **[DIVIDE]** Two-stage division will be adopted. The window size of the division will be 2, meaning you will try to connect two adjacent POS tags for a given grammar rule. If there is no suitable grammar rule, you will switch the window size to the next two constituents.
  You must start testing windows starting from the rightmost two constituents, moving to the left. An example is given below:
  - **Input: [**W_1 W_2 W_3 **(W_4 W_5)**]
    - Check if there is a suitable transformation rule ($R_1 \rightarrow W_4\ W_5$) for the sequence [W_4 W_5]. If so, form a sub-tree with them and merge them into a

single constituent. Continue testing windows starting from the rightmost pair in the current input **[**W_1 W_2 **(W₃ R₁)]**. Otherwise, shift the window to the left as in **[**W_1 W_2 **(W₃ W₄)** W_5], and keep testing transformation rules.

- If no transformation rule can be applied, trace back from the recursion and keep testing windows. If you trace back to the input and cannot find any other sub-trees, this means that **there is no solution**. In this case, you will print a **syntax error** message**.**

- [**CONQUER**] Once you reach a single **S** (Sentence) by connecting the sub-trees, trace back while concatenating the sub-trees. This will give you the whole parse tree.

Run:

1. Read the sentences from an input file and write the outputs to a file called "output.txt". Both files **must be encoded with UTF-8**.
2. For the example input "the cat sat on the mat":
3. Obtain **Input Sequence (with POS-Tag set):**
   DT(the) **NN**(cat) **VR**(sat) **PR**(on) **DT**(the) **NN**(mat)
4. Run the parsing algorithm. It should follow the steps below:
   a. Connect [**DT**(the) **NN**(mat)] with the rule *NP → DT NN:*
   NP[**DT**(the) **NN**(mat)]
   b. Now, connect PR(on) with NP with the rule *PP → PR NP:*
   PP[**PR**(on) **NP**[**DT**(the) **NN**(mat)]]
   c. Connect VR(sat) with PP with the rule *VP → VR PP:*
   VP[**VR**(sat) **PP**[**PR**(on) **NP**[**DT**(the) **NN**(mat)]]]
   d. Connect NN with VP with the rule *S → NP VP*. We reached S but we did not finish all the words DT(the) is not connected. Trace back to step c (cancel the *S → NP VP* connection), then shift the window to the left.
   e. Connect DT(the) NN(cat) with the rule *NP → DT NN*
   f. Connect NP and VP with the rule *S → NP VP.* We connected all words and reached S (sentence). Finish the program and print out the full parse tree:
   g.
   S[**NP**[**DT**(the) **NN**(cat)] **VP**[**VR**(sat) **PP**[**PR**(on) **NP**[**DT**(the) **NN**(mat)]]]]

!! Remember: You are obligated to use division from **rightmost pair**. For the given input sentence: "**They booked the flight through you**", there will be no parse-trees to be found for the lack of the grammar rule *VP → VP PP*. So the program should print out an error message: "SYNTAX ERROR".

Code (50 points)

Implement the divide and conquer parser in C++ exactly as defined in the project overview section. Write your program in a class called **Parser** encapsulating the necessary methods and attributes related to the algorithm.

Your class must have a **extractParseTree(…)** method that takes one input sequence (with POS-Tag set) and returns output sequence as given in the example. The problem also should give an error message (SYNTAX ERROR) if there are no parse-trees found.

You must also include a static **main(…)** method in your code expecting the path to an input file as a command line argument. Invoking the program from the command line should process these strings and print output sequence or the error message.

Your program should compile and run using the following commands:

```
g++ yourStudentID.cpp -o project2

./project2 input.txt
```

Your program will be tested with different input sentences of various lengths (within given language grammar). You should make sure the program runs properly with any sentence or prints out error message, otherwise you might not get full points for the code.

Report (50 points)

- [15 points] If the advantages of divide and conquer methodology were not to be used in your implementation, how would you formulate the problem?

- [15 points] What parameters does the complexity of your implementation depend on? How would you represent the worst case complexity in big O notation?

- [20 points] If the rule $VP \rightarrow NP\ VP$ were to be added to current set of grammar rules, this would cause ambiguity because it would overlap with the rule $S \rightarrow NP\ VP$. Briefly explain, how you would change your implementation to overcome this problem. Would it change the worst case complexity? If so, what would the complexity be in big O notation?

Submission

You should be aware that the Ninova system clock may not be synchronized with your computer, watch, or cell phone. Do not e-mail the teaching assistant or the instructors your submission after the Ninova site submission has closed. If you have submitted to Ninova once and want to make any changes to your report, you should do it before the Ninova submission system closes. Your changes will not be accepted by e-mail. Connectivity problems to the Internet or to Ninova in the last few minutes are not valid excuses for being unable to submit. You should not risk leaving your submission to the last few minutes. After uploading to Ninova,

check to make sure that your project appears there.

Policy: You may discuss the problem addressed by the project at an abstract level with your classmates, but you should not share or copy code from your classmates or from the Internet.

You should submit your own, individual project. Plagiarism and any other forms of cheating will have serious consequences, including failing the course.

Submission Instructions: Please submit your homework through Ninova. Please zip and upload all your files using filename HW2_studentID.rar. In the archived file, you must include your completed report StudentId file and all your program and header files.

All your code must be written in C++, and we must be able to compile and run on it on ITU's Linux Server (you can access it through SSH) using g++. You should supply one yourStudentID.cpp file that calls necessary routines for all questions (Multiple files are acceptable, as long as you state the compilation instructions in your report).

When you write your code, try to follow an object-oriented methodology with well-chosen variable, method, and class names and comments where necessary**. Your code must compile without any errors; otherwise, you may get a grade of zero on the coding part of the assignment.**

If a question is not clear, please let the teaching assistant know by email (torunoglud@itu.edu.tr).