COSC2320: Data Structures and Algorithms
HW4: Time Complexity of Union and Intersection of Sets

1 Introduction

The goal os this homework is that you learn big $O()$ definition and learn how to apply rule of sums and rule of products to estimate $T(n)$. In this homework you will estimate (predict) time complexity $T(n)$ (# of operations, not clock time) to compute intersection and union of sets, counting operations. In our case the sets contain words. Since words can be repeated in input files (i.e. bags as input) your program should eliminate duplicates and also count the number of occurrences (i.e. sets as output). You will write a C++ program to estimate time complexity of union and intersection of two sets. You will count the number of computations to estimate time $T(n)$, find the big $O()$ function (with its two constants) and display estimated/measured numbers in tabular form.

2 Input

The input is one script file, as in previous homeworks. The script file will ask you to read 2 or more input files into doubly linked lists and operate on them. In order to simplify the problem, the formatting of the original files will be discarded. Additional aspects about input: All uppercase characters must be changed to lower case. The input files will contain no punctuation symbols. Another important detail is that input files might be empty, since the empty set ($\emptyset$) is a valid set. union(L1,L2,L3) will find the *union* of sets L1 and L2 and store the result in L3, whereas intersection(L1,L2,L3) will find the *intersection* of sets L1 and L2 and store the result in L3. The script will ask you to perform one operation between sets (either union or intersection). The time required for this operation will be recorded.

3 Program and output specification

The main program should be called "time". You can use the Command Line Parser that is provided in the TA's homepage. Notice that we are explicitly specifying that the output of the program should be written to the file named by result. When computing the union or intersection of two linked lists you must produce one occurrence of each word in the result. That is, the output is a set.

Syntax:

```
time script=input.script;result=out.txt
```

You are required to come up with $T(n)$ and big $O()$ AFTER you read/load the lists, but BEFORE the lists are sorted and the overall process is done. You must show this *estimate* is bounded by $O()$ after execution. Since $T(n)$ should be estimated using "worst case scenario" conditions, $O()$ should always be larger than the actual time required. The $O(g(n))$ function must be one of the fundamental time functions (just one term on n). Depending on how lists are sorted $T(n)$ can be $O(n^2)$ (default) or $O(n\log(n))$ (optional). The

tighter the bound and the $T(n)$ estimation the better. You should consider worst case for $O()$. It is expected each student will have different T(n) functions and different c and n_0 values, but the $O()$ bounds must hold.

For the estimate of the *actual* $T(n)$ you should count CPU operations simply by incrementing a *global counter* variable each time an operation (assignment, comparison, subscript access, mathematical operator, pointer access) is performed. Remember that in general comparisons tend to play a bigger role on any comparison-based algorithm. You can be as accurate as you want to count CPU operations. Use the rule of sums and rule of products to come up with the *estimated* $T(n)$. To find the big $O(g(n))$ you need to find c, n_0. It is a requirement that your c, n_0 values be as small as possible, but it is not necessary that $n_0 = 1$. In other words, we do not want loose bounds where $n \geq 100$.

Sorting: You can optionally use one efficient sort algorithm of your choice (mergesort, quicksort, heapsort). You can sort them using linked lists or, alternatively, more efficiently, use an intermediate array. This solution is acceptable because you know the exact size of the list and this allows the allocation of space dynamically for the array.

4 Output and Examples

You are required to produce an output table formatted as follows (in CSV format) for each set operation.

input1.script

```
read(A,'A.txt')
read(B,'B.txt')
union(A,B,C)
```

result1.txt

```
L1     L2                                                ------O(g(n))------
size size   operation     T(n) estimate   T(n) actual   c    n0   g(n)=n^2
 20    30   union                220          200       3   1        900
```

input2.script

```
read(A,'A.txt')
read(B,'B.txt')
intersection(A,B,C)
```

result2.txt

```
L1     L2                                                ------O(g(n))------
size size   operation     T(n) estimate   T(n) actual   c    n0   g(n)=n^2
 10    20   intersection         100          98        3   1        120
```

5 Requirements

- You should explain in your README file how you found c, n_0 for the specific functions in $T(n)$ and $O(g(n))$. You can show some sample output of your program to prove your bounds become tighter as n grows. You can show all mathematical details of the derivation with inequalities.
- You should sort AFTER reading the lists. You should not insert in order as you are reading the lists.
- You should estimate $T(n)$ and $O(g(n))$ AFTER you read the input lists (with possibly repeated words), but BEFORE you sort them. It is acceptable you estimate average case for $T(n)$ analyzing several potential inputs as long as the approximation gets better as n grows. For big $O(g(n))$ you have to assume worst case.
- Your program should be able to handle lists of thousands of words. There is no upper limit to the size of the input files. Your program must be able to create and handle multiple doubly linked lists.
- You can compute union/intersection recursively or non-recursively. The result should be correct in either case. It is preferable you compute these operation on sorted versions of the lists. Alternatively, you can compute the set operations with nested loops, making sure duplicate values are handled.

  Arrays are allowed to perform sorting efficiently, after the number of words in each list is known. Arrays are not allowed to read/load the lists (initially). Arrays are not allowed to write result lists. In other words, arrays can only be used for sorting efficiently as an intermediate processing step.
- Optional: as an *option* you can sort later with arrays with a more efficient $O(n\log_2(n))$ algorithm to count words and eliminate duplicates. Such algorithm can allocate the exact n without wasting memory. In the end, results from intersection/union should be stored on a doubly linked list.
- Testing: regardless of the algorithm you use to compute $\cap, \cup$ of two sets of words your $T(n)$, $O(g(n))$ should become tighter as $n \to \infty$.
- You are allowed to use loops and iterators when reading and writing from a file, in order to recycle the code you used in the previous homework set. However, you must create a recursive search method and a recursive list traversing method. (Hint: Look at the example in the book where you are shown how to print a list backwards)
- You must create a log file that records the length of each linked list after each operation. This log file should also be used to output any warnings and errors you might encounter.
- The program should not halt when encountering errors in the script. It should just send a message to the log file and continue with the next line. The only error that is unrecoverable is a missing script file, or a missing argument in the command line.
- Do not use the STL library. Your program should write error messages to the screen. Your program should not crash, halt unexpectedly or produce unhandled exceptions. Consider empty input, zeroes and inconsistent information. Each exception will be -10.
- Test cases. Your program will be tested with 10 test scripts, going from easy to difficult. You can assume 80% of test cases will be clean, valid input files. If your program fails an easy test script 10-20 points will be deducted. A medium difficulty test case is 10 points off. Difficult cases with specific input issues or complex algorithmic aspects are worth 5 points.