Data Structure and Algorithm

Homework #2

Due: 2:00pm, Thursday, March 29, 2012

TA email: dsa1@csie.ntu.edu.tw

=== Homework submission instructions ===

- For Problem 1, submit your source code, a shell script to compile the source, and a brief documentation to the SVN server (katrina.csie.ntu.edu.tw). You should create a new folder "hw2" and put these three files in it.
- The filenames of the source code, the shell script, and the documentation file should be "transfer.c", "compile.sh", and "report.txt", respectively. **The shell script should compile your source codes to generate an executable binary with the filename 'transfer' (new rule!)**. You will get some penalties in your grade if your submission do not follow the naming rule.
- The documentation file should be in plain text format (.txt file). In the documentation file you should explain how your code works, and anything you would like to convey to the TAs.
- For Problem 2 to Problem 5, submit the answers through the CEIBA system (electronic copy) or to the TA in R508 (hard copy).
- Each student may only choose to submit the homework in one way; either all as hard copies or all through CEIBA except the programming assignment. If you submit your homework partially in one way and partially in the other way, you might only get the score of the part submitted as hard copies or the part submitted through CEIBA (the part that the TA chooses).
- If you choose to submit the answers of the writing problems through CEIBA, please combine the answers of all writing problems into only ONE file in the doc/docx or pdf format; otherwise, you might only get the score of one of the files (the one that the TA chooses).
- Discussions with others are encouraged. However, you should write down your solutions by your own words. In addition, you have to cite the sources you consulted from or people you discussed with on the first page of your solution to that problem. The TA can deduct up to 100% of the score for any problem without the citation.
- No late submission of the homework will be given any score (for that portion).

Problem 1. Transfer a letter (30%)

Imagine you are in one of the "branches" of the world. Yes, the world, just like the revisions on

the SVN server, has many branches, each having individual time and space. Ridiculous, is it? But please listen to the story for a while–I promise it will be within one A4 page! Let's come back to the branches. Every branch of the world has different speed of time.

Unfortunately, you are on a very, very terrible branch. The human beings are so arrogant that they seriously destroy the nature environment while developing their technologies. The sky is full of dust and the sunlight only shows up one or two times in a month and is extremely weak. Hurricanes, abnormally, appear more frequently than your homework at school! With more and more uncontrollable natural disasters, it seems that there is no effective way to prevent the coming of "the end of the world" (in the point of view of the people in this branch).

Fortunately, one of the craziest technologies developed by the human beings in this branch is to transfer a letter to another branch of the world. There might not be hopes in saving this poor branch, but it would be possible to remind people in other branches not to make the same mistake! Remember the time in some branches are slower, so maybe it is not too late for them!

According to the scientists who invents the technology, there are some limitations. They said that repeating words in a letter would interfere with each other while being transferred. This would eventually corrupt the content of the letter. In short, the letter to be transferred cannot contain repeating words! We all know that some of the English words must appear for many times in a letter; instead of trying to use distinct words throughout the letter, a more practical way is to transform the letter, without losing any information, into a form that every word occurs only one time.

The transformation is not complicated, though. Before describing the method, we have to give the definition of a "word". A word is a maximal-length sequence consisting of 26 English alphabets, including the upper-case and lower-case ones; 52 in total. For example, the sentence "This is Ted's favorite T-shirt" contains 7 words: "This", "is", "Ted", "s", "favorite", "T", and "shirt".

At first, the transformed letter is empty. You are required to chain the words in the original letter into a list. For every word you encounter, if it is the first time the word occurs, **copy the word to the transformed letter** and **put the word to the front of the list**. If the encountered word has occurred before, copy the "position number" of the word in the list to the transformed letter, and then move the word to the front of the list. Note that the word is case-sensitive. For example, "Bug" and "bug" are different words!

For any character that is not an alphabet, copy it directly to the transformed letter. Repeat the procedure until you reach the end of the letter, or the end of the world. (Not funny at all.)

What you have to do for this problem is to write a program to accomplish this transformation. You can forget the crazy story written by the boring TA once you finish solving this problem, so please start as soon as possible!

- Input

You are given a letter in the input. There can be multiple lines in the letter, so please read until EOF. The length of a word is always smaller than 50. There is no assumption for the number of distinct words in the letter. Any printable character may appear in the letter, except for numbers.

- Output

  Please output the transformed letter in the same format as in the input, but replace repeated words with their position number. **Note that the position number starts from 0.**

- Sample Input

  Data Structure and Algorithm

  This is a required course for the undergraduate students in the Department of Computer Science and Information Engineering. The course will focus on the implementation and the use of data structures and how to utilize them to solve problems.

- Sample Output

  Data Structure and Algorithm

  This is a required course for the undergraduate students in 3 Department of Computer Science 15 Information Engineering. The 13 will focus on 12 implementation 9 2 use 13 data structures 5 how to utilize them 2 solve problems.

- **Remark: You should implement a Linked-List to solve this problem.**

Problem 2. Back to stacks & queues (12%)

Suppose we use the following declarations for a program using singly linked list.

```
struct node {
  SomeType data;
  struct node *next;
};
typedef struct node Node;
```

1. (6%) Utilize the above declarations to implement a stack. You need to implement two methods: void push(Node **head, SomeType data) and SomeType pop(Node **head). All methods should run in $O(1)$. (You do not need to explain why they run in $O(1)$.)

2. (6%) Use this structure to implement a queue. You need to implement two methods: void enqueue(Node **head, SomeType data,Node **tail) and SomeType dequeue(Node **head, Node **tail). All methods should run in $O(1)$. (You do not need to explain why they run in $O(1)$.)

Problem 3. Double-ended queues (15%)

The double-ended queue is a data structure that is similar to a queue. However, we can push to or

pop from its both ends. Suppose we use the following declarations for a program using circularly doubly linked list.

```
struct node {
  SomeType data;
  struct node *prev, *next;
};
typedef struct node Node;
```

1. (12%) Use this structure to implement a double-ended queue. You need to implement four methods: void pushBack(Node **head, SomeType data), void pushFront(Node **head, SomeType data), SomeType popBack(Node **head) and SomeType popFront(Node **head). All methods should run in $O(1)$. (You do not need to explain why they run in $O(1)$.)

2. (3%) Suppose we use either "pushFront" or "pushBack" to enter elements $1, 2, 3, 4, 5$ into a double-ended queue in the given order, but not necessarily in consecutive operations. That is, we enter 2 into the double-ended queue after 1, 3 after 2, ..., and so on, but there could be "popFront" or "popBack" in between entering 1 and 2, 2 and 3, etc. Give a sequence of operations that will result in the following output. The sequence should consist of 10 pushBack, pushFront, popBack and popFront operations. If there is more than one solution, you only need to specify one.

   Example: 4,3,1,2,5
   Solution:

   - pushFront(1)
   - pushFront(2)
   - pushFront(3)
   - pushFront(4)
   - popFront() → got 4
   - popFront() → got 3
   - popBack() → got 1
   - popBack() → got 2
   - pushFront(5)
   - popFront() → got 5

   (a) 1, 5, 4, 3, 2

   (b) 5, 3, 2, 4, 1

(c) 1, 4, 2, 3, 5

Problem 4. More operations on lists (19%)

1. (7%) Given a singly linked list, write a function to find the $\lfloor n/k \rfloor_{th}$ element, where n is the total number of elements in the list. You can assume the pointer to the first element in the list and the value of k are given, but the value of n is not known in advanced. Also, the first element in the list is defined as the 0_{th} element.

2. (6%) Let $x = \{x_1, x_2, \ldots, x_n\}$ and $y = \{y_1, y_2, \ldots, y_m\}$ be two singly linked lists. Write a function to merge the two lists together to obtain a new singly linked list $z = \{x_1, y_1, x_2, y_2, \ldots, x_m, y_m, x_{m+1}, \ldots, x_n\}$ if $m \leq n$ and $z = \{x_1, y_1, x_2, y_2, \ldots, x_n, y_n, y_{n+1}, \ldots, y_m\}$ if $m > n$. Your function can modify the pointers (to other nodes) in nodes in x and y, but should use only $O(1)$ additional space. What is the time complexity of your function? Please explain.

3. (6%) Let $x = \{x_1, x_2, \ldots, x_n\}$ and $y = \{y_1, y_2, \ldots, y_m\}$ be two singly linked lists which are both sorted in ascending order. Write a function to merge the two lists to obtain a new singly linked list z which is also sorted in ascending order. Your function can modify the pointers (to other nodes) in nodes in x and y, but should use only $O(1)$ additional space. What is the time complexity of your function? Please explain.

Problem 5. Skip lists (24%)

A skip list is a data structure that stores a sorted list of elements. It is formed by a hierarchy of linked lists and supports efficient search and insert operations. In fact, the expected time (in the sense of probability) for inserting an element or searching for an element in a skip list is $O(logn)$, where n is the number of elements in the list. Learn this data structure and associated operations from any materials and present your understandings in the following questions:

1. (8%) How to represent the structure of a skip list? What is the space complexity of this representation?

2. (8%) How is insert operation performed? Please also analyze the time complexity of this operation.

3. (8%) How is search operation performed? Please also analyze the time complexity of this operation.