Data Structure and Algorithm
Homework #2
Due: 2:20pm, Tuesday, March 26, 2013
TA email: dsa1@csie.ntu.edu.tw

=== Homework submission instructions ===

- For Problem 1, submit your source code, a Makefile 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, Makefile, and the documentation file should be "`main.c`", "`Makefile`", and "`report.txt`", respectively; you will get some penalties in your grade if your submission does 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 through 5, submit the answers via the SVN server (electronic copy) or to the TA at the beginning of class on the due date (hard copy).
- Except the programming assignment, each student may only choose to submit the homework in only one way; either all in hard copies or all via SVN. 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 via SVN (the part that the grading TA chooses).
- If you choose to submit the answers of the writing problems through SVN, please combine the answers of all writing problems into only ONE file in the pdf format, with the file name in the format of "`hw2_[student ID].pdf`" (e.g. "`hw2_b01902888.pdf`"); otherwise, you might only get the score of one of the files (the one that the grading TA chooses).
- Discussions with others are encouraged. However, you should write down your solutions by your own words. In addition, for each problem you have to specify the references (the Internet URL you consulted with or the people you discussed with) on the first page of your solution to that problem.
- **NO LATE SUBMISSION IS ALLOWED** for the homework submission in hard copies - no score will be given for the part that is submitted after the deadline. For submissions via SVN (including the programming assignment and electronic copies of the writing problems), up to one day of delay is allowed; however, the score of the part that is submitted after the deadline will get some penalties according to the following rule (the time will be in seconds):

$$\text{LATE SCORE} = \text{ORIGINAL SCORE} \times (1 - DelayTime/86400)$$

Problem 1. Unrolled Linked List (30%)

One of the biggest advantages of linked lists over arrays is that inserting an node (element) at any location takes only $O(1)$ time. However, it takes $O(N)$ to find the k-th node in a linked list. There is a simple variation of the original singly linked list called *unrolled linked lists*. Unrolled linked lists have slightly better time complexity when looking for an element, but have slightly worse time complexity when inserting an element.

Unrolled linked lists consist of "blocks", which can hold more than one node. In each block, a circular linked list is used to connect all nodes. Figure 1 shows an example of the unrolled linked list.

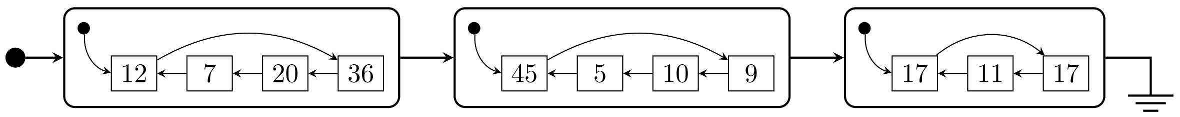


Figure 1: An unrolled linked list example. Blocks are shown as rounded rectangles. Heads of linked list are shown as solid circles.

In this problem, you are asked to implement unrolled linked lists. Assume that there will be no more than N elements in the unrolled linked list at any time. To simplify this problem, **all blocks, except the last one, should contain exactly $\lceil\sqrt{N}\,\rceil$ elements.** Thus, there will be no more than $\lfloor\sqrt{N}\,\rfloor$ blocks at any time. In such a data structure, we can find the k-th element in $O(\sqrt{N})$:

1. Traverse on the "list of blocks" to the one that contains the k-th node, i.e., the $(\lceil k/\lceil\sqrt{N}\rceil\rceil)$-th block. It takes $O(\sqrt{N})$ since we may find it by going through no more than $\lfloor\sqrt{N}\,\rfloor$ blocks.
2. Find the $(k \mod \lceil\sqrt{N}\rceil)$-th node in the circular linked list of this block. It also takes $O(\sqrt{N})$ since there are no more than $\lceil\sqrt{N}\,\rceil$ nodes in a single block.

   See Figure 2 for an example.

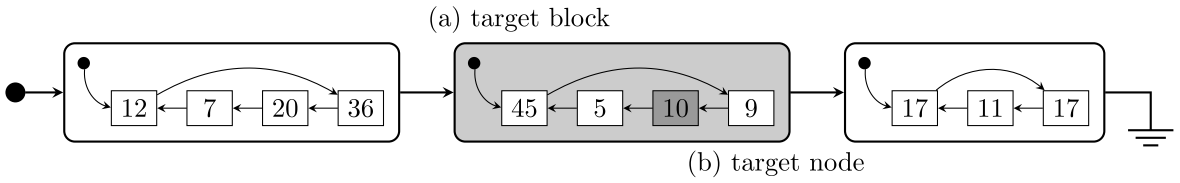


Figure 2: Assume that each block has 4 nodes ($\lceil\sqrt{N}\rceil = 4$). To find the 7th node ($k = 7$), (a) find the 2nd block first, since $\lceil 7/4\rceil = 2$. Then (b) find the 3rd node in the 2nd block, since 7 mod 4 = 3.

When inserting a node, we have to re-arrange the nodes in the unrolled linked list to maintain the properties previously mentioned, that each block contains $\lceil \sqrt{N}\, \rceil$ nodes. Suppose that we insert a node x after the i-th node, and x should be placed in the j-th block. Nodes in the j-th block and in the blocks after the j-th block have to be shifted toward the tail of the list so that each of them still have $\lceil \sqrt{N}\, \rceil$ nodes. In addition, a new block needs to be added to the tail if the last block of the list is out of space, i.e., it has more than $\lceil \sqrt{N}\, \rceil$ nodes. An example is shown in Figure 3.

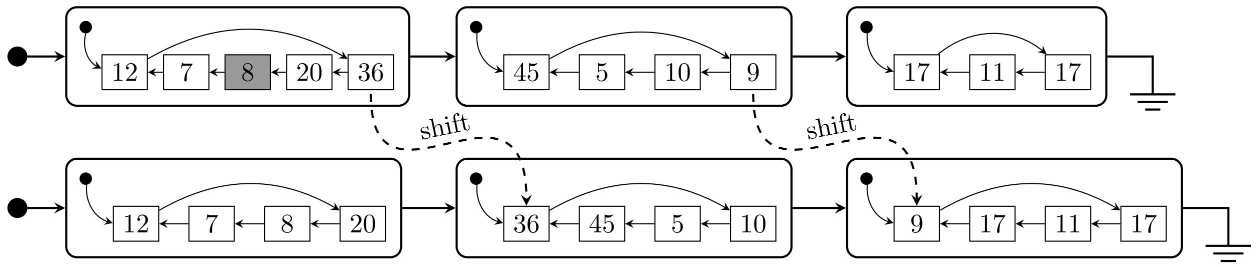
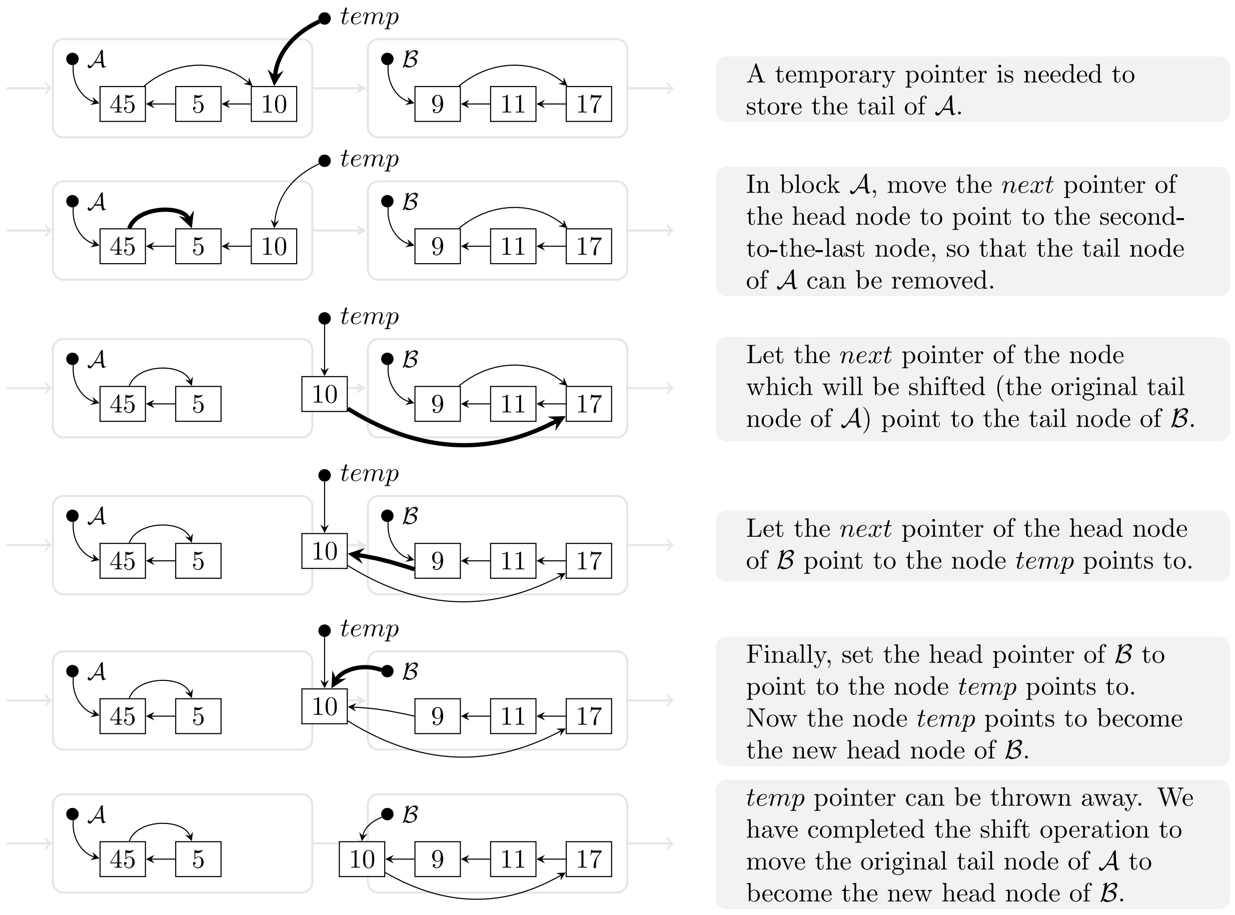


Figure 3: After inserting the node with value 8 after the 2nd node in Figure 1, nodes have to be shifted toward the tail, so that all blocks, except the last one, have exactly $\lceil \sqrt{N} \rceil$ nodes.

Note that each "shift" operation, which includes removing a node from the tail of the circular linked list in a block and inserting a node to the head of the circular linked list in the block after, takes only $O(1)$. The total time complexity of an insertion operation for unrolled linked lists is therefore $O(\sqrt{N})$; there are at most $O(\sqrt{N})$ blocks and therefore at most $O(\sqrt{N})$ shift operations. Please take a look at Figure 4.



A temporary pointer is needed to store the tail of $\mathcal{A}$.

In block $\mathcal{A}$, move the *next* pointer of the head node to point to the second-to-the-last node, so that the tail node of $\mathcal{A}$ can be removed.

Let the *next* pointer of the node which will be shifted (the original tail node of $\mathcal{A}$) point to the tail node of $\mathcal{B}$.

Let the *next* pointer of the head node of $\mathcal{B}$ point to the node *temp* points to.

Finally, set the head pointer of $\mathcal{B}$ to point to the node *temp* points to. Now the node *temp* points to become the new head node of $\mathcal{B}$.

temp pointer can be thrown away. We have completed the shift operation to move the original tail node of $\mathcal{A}$ to become the new head node of $\mathcal{B}$.

Figure 4: Details of shift operations: shifting the node at the tail of block $\mathcal{A}$ to the head of block $\mathcal{B}$.

The following describes the input format. The first line of the input is a positive integer M, which is the number of operations which will be specified in the input. In other words, there will be no more than M insertions. Therefore, you may treat M as the maximum number of elements in the unrolled linked list, i.e., $N = M$.

The following M lines specify the M operations to be performed. Each operation is specified in the format of "`insert k x`" or "`query k`" in each line. Both k and x are non-negative integers.

"`insert k x`" represents an operation to insert a node with value x after the k-th node. In the special case of $k = 0$, you should insert x before the entire list.

"`query k`" represents an operation to print out the value of the k-th node of the list. You can assume that the operations specified in the input are always feasible to perform. There will be no query or insertion operations with a parameter k while there are less than k nodes in the current list. A set of sample input and output are provided below:

Sample Input

```
8
insert 0 5
insert 1 3
insert 1 4
query 1
query 2
query 3
insert 1 2
query 3
```

Sample Output

```
5
4
3
4
```

Your program should complete M operations in $O(M\sqrt{M})$ time; otherwise, you may exceed the time limit (5 seconds on the CSIE workstation). A set of sample input and output with large M is available on the course website for you to check if the running time of your program is acceptable.

Gift/Hint Here is an example of the C structure that can be used for unrolled linked lists:

Node

```
struct Node {
     struct Node *next;
     int value;
};
```

Block

```
struct Block {
     struct Block *next;
     struct Node *head;
     int nodeCount;
};
```

***Problem* 2.** Linked List (15%)

The following code is an implementation of singly linked lists.

```c
struct node {
    Data *data;
    struct node *next;
};
typedef struct node Node;
```

Let `x` be the pointer to the last node in the list, `x->next` must be `NULL`.

2.1. (5%) When `head` is of type `Node*` and is a pointer to the first node of a linked list, the function call `insert(&head, data, i)` would insert the given data into the list at the `i`-th position. If `i` is greater than the current number of nodes in the list, the new `Node` should be inserted at the end of the list.

If `i` is less than or equal to one, the new `Node` should be inserted at the front of the list.

You are asked to finish an recursive implementation of the C function

`insert(Node**, Data*, int)`.

To simplify the problem, you can assume that when the function is called the parameter `pNode` is never `NULL`.

```c
void insert(Node **pNode, Data* data, int i){
    if(i<=1 || *pNode==NULL){
        //TODO add some statements here
        return;
    }
    insert(______, data, ______);
}
```

2.2. (5%) You are asked to finish an iterative implementation of the C function

`insert(Node**, Data*, int)` using a `while`-loop.

You are not allowed to call any function in your implementation.

```c
void insert(Node** pNode, Data* data, int i){
    while(______){
    //TODO add some statements here
    }
    //TODO add some statements here
}
```

2.3. (3%) Describe the time spent in both implementations with the big-O notation in terms of n, the number of nodes in the list, using the method mentioned in the lecture.

2.4. (2%) Describe the additional space required in both implementations with the big-O notation in terms of n, the number of nodes in the list, using the method mentioned in the lecture. Note that the parameters of the function should also be considered.

Problem 3. Tree, Tree and Tree!! (20%)

"In computer science, a tree is a widely used data structure that simulates a hierarchical tree structure with a set of linked nodes." – From *Wikipedia*, the free encyclopedia.

We have learned about much knowledge of the tree structure, its variations, and its applications in the class. In the following problems, we will apply them to solve some interesting issues.

3.1. (10%) Tree traversal is the process of visiting all nodes in the tree structure. In the binary tree cases, there are three kinds of methods introduced in the class: preorder, inorder and postorder.

a. There is a binary tree shown in Figure 5. Each node is represented by a letter in upper case. Please write down the results of the three kinds of tree traversals (i.e, preorder, inorder, and postorder), and briefly describe how you derive them.

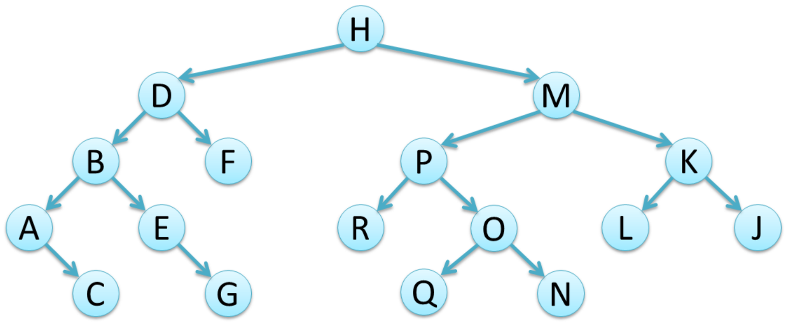


Figure 5: The tree in problem a.

b. In problem a, we have learned how to perform tree traversals from a binary tree. In this problem, we will try to derive the original binary tree from the results of tree traversals. Given **preorder** and **inorder** traversals of a binary tree, shown as follows, please derive and draw the original binary tree (in this case, there is only one UNIQUE binary tree that can be used to generate these traversals).

$$preorder : BVAQHERGFSKJODLMCWP$$
$$inorder : HQREFGSAJKOVDBMLWCP$$

There are cases, when a set of **preorder** and **postorder** traversals is given, more than one binary search tree can be used to generate them. Please give one such example.

3.2. (10%) The binary search tree (BST) is a data structure based on binary tree and can help us to insert, delete, and query values efficiently. Also, BST has lots of interesting properties in its structure and its time complexities for various operations.

a. Consider a BST T composed by n distinct values. Let node x be a leaf node and node y be its parent. Please prove or disprove that the value of y is either the smallest in the values larger than x or the largest in the values smaller than x.

b. In the cases of having n distinct values, on average BST can perform operations in $O(\log n)$ time (when random values are inserted into the BST). However, similar to the cases of several other data structure, the insertion operation could have worse running time when the values to be inserted are identical. Please derive that time complexities of the insert and query operations when n identical values are inserted into the BST using the standard BST algorithms mentioned in the lecture.

   In addition, please derive a different BST insertion algorithm, and show that it has better running time performance when the values to be inserted to the BST are identical.

Problem 4. Online Dictionary (15%)

You want to build an online English dictionary for your classmates. Before you start to write codes, however, you are facing the problem of which data structure you should use.

Here are the details. Assume you have collected N English words in total. Each word is a sequence of at most L letters. To simplify the problem, let's just consider words consisted of 26 lowercase alphabets('`a`'-'`z`') for now. You hope that your online dictionary contains all N collected words and is able to handle the following operations:

a. **list-all**: List all of the N words in ordinary dictionary order. [1] [2]

b. **search**: When a user types a word, it should tell the user whether this word is in the dictionary.

c. **maintain**: You, as a administrator, can add a word or delete a word from the dictionary.

You quickly come up with a naive data structure:

Use any sort algorithm to sort N words in dictionary order to build such a dictionary.

Then save these words into a 2-D array: `char D[N][L]` by such order.

For the following problems, when specifying the complexities, please show them in terms of N and L.

4.1. (2%) What is the time complexity for building such a dictionary?

4.2. (3%) What is the time complexity for **list-all**? How about **search**?

However, you, as a nature lover, cannot tolerate any data structure without tree.

4.3. (4%) Please design a data structure for building the dictionary with a tree. (Hint: If each node represents a letter, and have at most 26 child nodes...?) What are the time complexity and the space complexity of your data structure?

4.4. (3%) How do you **search** a word in your tree-base data structure? What is the time complexity?

4.5. (3%) Please describe a method to do **list-all** in $O(NL)$.

4.6. (Bonus 5%) What are the time complexities of the **maintain** processes for these two data structures?

[1] Open any dictionary, you can feel the spirit of the dictionary order. The dictionary order of a set of words is obtained by sorting them by their first letter, then their second letter, and so on.

[2] For instance, the dictionary order of the words {"RainbowDash", "Twilight", "Rarity", "Rainbow"} are {"Rainbow", "RainbowDash", "Rarity", "Twilight"}.

	Naïve	Yours
Space Complexity	$O(NL)$	(3)
Bulid	(1)	(3)
Link-all	(2)	(4)
Search	(2)	(4)
Maintain (bonus)	(6)	(6)

Table 1: The complexity of operations.

Problem 5. Interesting Puzzle (20%)

Once upon a time, there was a traveller called *Earthworm*.

One day, he arrived at a big lake which contained a lot of islands. However, there was something strange. He found that even there were many bridges connecting them, only one bridge led to the lakeside. Then, he walked to the bridge, and found a signboard next to it. The sign read:

> This is a puzzle.
>
> There is an arrow which points to the next island from each island.
>
> If you start travelling from here and follow the arrows, you can reach all islands.
>
> If you're smart enough, you will discover that there exists a loop, and the shape of the path connecting all islands is like the symbol ρ.
>
> Now, the problem is ...
>
> can you find out the **length** of the loop in the path?

He was good at designing algorithms, so he solved this problem efficiently. How about you?

Assume that there were n islands in the lake but *Earthworm* did not know this information. In other words, you cannot treat this information as known condition in your algorithm.

For each sub-problem, derive the time and space complexities of your algorithm in terms of n.

(you do not need to consider the time and space costs of the input data)

You can use the following structure to describe your algorithm clearly.

```
struct Island
{
    struct Island *next;
};
struct Island *lakeside;
```

5.1. (3%) Design an algorithm that achieves the goal and runs in $O(n^2)$ time, but uses only $O(1)$ additional space.

5.2. (3%) If there is no limitation of the additional space used, design an algorithm that runs in $O(n)$ time.

5.3. (8%) Try to design a better algorithm step by step:

(a) Given k, design an algorithm to determine whether "*the length of the loop is smaller than k if the distance between the loop and lakeside is smaller than k*". If so, find out the length of the loop. In addition, your algorithm should run in $O(k)$ time and use only $O(1)$ additional space.

(b) Now, we set k to start from 1. Check if the conditions described above are satisfied. If so, then the length of the loop can be found. Otherwise, we set k to be 2 times of the previous value, and continue to check if the conditions are satisfied.

First, please show that this algorithm will eventually stop, i.e., the conditions will eventually be satisfied. Then, prove that it runs in $O(n)$ time and uses only $O(1)$ additional space.

5.4. (6%) If two people start from lakeside and walk like the following code:

```
ptr1 = ptr1->next;
ptr2 = ptr2->next->next;
```

(a) Prove `ptr1 == ptr2` will happen in some time and when this is true `ptr1` (and `ptr2`) will point to an island inside the loop.

(b) Design an algorithm that uses these results and show that it runs in $O(n)$ time and uses only $O(1)$ additional space.

5.5. (Bonus 5%) If we already know the length of the loop is m, design an algorithm to find out the beginning of the loop (the nearest island to lakeside) in $O(n)$ time and uses only $O(1)$ additional space.