# Data Structure and Algorithm, Spring 2018
# Solution to Homework 1

**Release: Tuesday, March 13, 2018**

TA E-mail: `dsa1@csie.ntu.edu.tw`

## Rules and Instructions

- In homework 1, the problem set contains 5 problems and is divided into two parts, non-programming part (problem 1, 2, 3) and programming part (problem 4, 5).
- Please go to *DSA Judge* (`https://dsa.csie.org`) and complete the first problem to get familiar with the usage of the judge system.
- For problems in non-programming part, you should combine your solutions in ONE pdf file, and then hand it in via the judge system. Your solution must be as simple as possible. At the TAs' discretion, too complicated solutions will be counted as wrong.
- For problems in programming part, you should write your code in C programming language, and then hand them in via the judge system. You have 5 chances per day for each problem. The compilation command of the judge system will be `gcc main.c -static -std=c11 -O2 -lm`.
- Discussions with others are encouraged. However, you should write down your solutions in 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.
- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$LateScore = (\frac{86400 - DelayTime(sec.)}{86400})OriginalScore$$

## Non-Programming Part

**Problem 1.** Stack/Queue (15 *points*)

You have learned in class that stack is a FILO/LIFO data structure and that queue is a FIFO/LILO data structure. Please describe a procedure under the following scenarios. They are all independent. You can assume that all data structure mentioned below have infinite capacity and all operations are valid. That is, no `Pop()` will be called on an empty stack and no `DeQueue()` will be called on an empty queue.

a. (4 *points*) Please use 2 queues and $O(1)$ extra space to simulate a stack. That is, please implement `Pop()`, `Push()` and `IsEmpty()` of a stack.

**Solution.**

Let two queues be `q1` and `q2`.

`Push()`: Enqueue the element into the non-empty queue. (If both are empty, enqueue arbitrarily one of them)

`Pop()`: Dequeue all the elements of non-empty queue and enqueue them into the other empty queue sequentially, until only one element left and return it.

`IsEmpty()`: Check whether both of queues are empty; return `true` if they are, otherwise return `false`.

```
void Push(data_type element){
  if(q1.IsEmpty() && q2.IsEmpty())
    q1.Enqueue(element);
  else if(!q1.IsEmpty())
    q1.Enqueue(element);
  else
    q2.Enqueue(element);
}

data_type Pop(void){
  if(q1.IsEmpty()){
    while(!q2.IsEmpty()){
      tmp = q2.Dequeue();
      if(!q2.IsEmpty())
```

```
        q1.Enqueue(tmp);
      else
        return tmp;
    }
  }
  else{
    while(!q1.IsEmpty()){
      tmp = q1.Dequeue();
      if(!q1.IsEmpty())
        q2.Enqueue(tmp);
      else
        return tmp;
    }
  }
}

bool IsEmpty(stack){
  if(q1.IsEmpty() && q2.IsEmpty())
    return true;
  else
    return false;
}
```

b. (4 *points*) Please use 2 stacks and $O(1)$ extra space to simulate a queue. That is, please implement `DeQueue()`, `EnQueue()` and `IsEmpty()` of a queue.

**Solution.**

Let two stacks be `s1` and `s2`.

`EnQueue()`: Push the element into `s1`.

`DeQueue()`: If `s2` is empty pop all the elements of `s1` and push them into `s2` sequentially until `s1` is empty; then, pop the top element of `s2` as the return value.

`IsEmpty()`: Check whether both of stacks are empty; return `true` if they are, otherwise return `false`.

```
void EnQueue(data_type element){
  s1.Push(element);
}

data_type DeQueue(void){
  if(s2.IsEmpty()){
    while(!s1.IsEmpty()){
      s2.Push(s1.Pop());
    }
  }
  return s2.Pop();
}

bool IsEmpty(void){
  if(s1.IsEmpty() && s2.IsEmpty())
    return true;
  else
    return false;
}
```

You are given a stack $S$ and a queue $Q$. They both have infinite capacity, and they both store positive integers. The numbers of elements of $S$ and $Q$ will both be not greater than $n$. Please describe a procedure to determine whether there exists common element between $S$ and $Q$ in the following scenarios. If there exists, please output arbitrarily one of the common elements. If not, please output 0. The scenarios are independent. For each problem, if it's not trivial, please briefly explain why your procedure is correct and why the time/space complexity meets the constraints.

c. (2 *points*) Guaranteed that the elements in $S$, from bottom to top, are strictly increasing, and the elements in $Q$, from rear to front, are also strictly **increasing**. In $O(n)$ time, $O(1)$ extra space.

**Solution.**

Let the stack and the queue be `s` and `q`.

step1. If at least one of `s` and `q` is empty, return `0`.

step2. Compare the top element of `s` with the front element of `q`; if they are identical, return it; if the former is bigger than the latter, pop the top element of `s`, otherwise, dequeue the front element of `q`.

step3. Repeat the step2 until there is a common element(return it) or one of `s` and `q` is empty(return `0`)

```
int FindCommonElement(q, s){
  /* step1 */
  if(q.IsEmpty() || s.IsEmpty())
    return 0;

  /* step2 */
  int p; // 1 for dequeue, 0 for pop
  tmpQ = q.DeQueue();
  tmpS = s.Pop();
  if(tmpQ > tmpS)
    p = 1;
  else if(tmpQ < tmpS)
    p = 0;
  else
    return tmpQ;

  /* step3 */
  while(!q.IsEmpty() && !s.IsEmpty()){
    if(p == 1)
      tmpQ = q.DeQueue();
    else if(p == 0)
      tmpS = s.Pop();

    if(tmpQ > tmpS)
      p = 1;
    else if(tmpS > tmpQ)
      p = 0;
    else
```

```
        return tmpQ;
    }
    return 0;
}
```

d. (5 *points*) Guaranteed that the elements in $S$, from bottom to top, are strictly increasing, while the elements in $Q$, from rear to front, are strictly **decreasing**. In $O(n)$ time, $O(1)$ extra space.

**Solution.**

Let the stack and the queue be `s` and `q`. Because the stack has infinite capacity, we can dequeue all the elements of `q` and push them into `s` sequentially, and then enqueue them back to `q` sequentially again. As a result, we have the same scenario as in the problem c, and follow the step of c.

```
int FindCommonElement(q, s){
    int i = 0;
    while(!q.IsEmpty()){
        s.Push(q.DeQueue());
        i++;
    }
    while(i--){
        q.EnQueue(s.Pop());
    }

    /* the following code is the same as (c) */
    /* step1 */
    if(q.IsEmpty() || s.IsEmpty())
        return 0;

    /* step2 */
    int p; // 1 for dequeue, 0 for pop
    tmpQ = q.DeQueue();
    tmpS = s.Pop();
```

```
  if(tmpQ > tmpS)
    p = 1
  else if(tmpQ < tmpS)
    p = 0;
  else
    return tmpQ;

  /* step3 */
  while(!q.IsEmpty() && !s.IsEmpty()){
    if(p == 1)
      tmpQ = q.DeQueue();
    else if(p == 0)
      tmpS = s.Pop();

    if(tmpQ > tmpS)
      p = 1;
    else if(tmpS > tmpQ)
      p = 0;
    else
      return tmpQ;
  }
  return 0;
}
```

**Problem 2.** Complexity (20 *points*)

In this part, if you would like to use any theorem which is not mentioned in class, please prove it in advance.

$$(\lg(n) = \log_2(n),\ \ln(n) = \log_e(n),\ \log(n) = \log_{10}(n))$$

1. (1 *points*) Please rank the following functions by the order of growth. No proof or extra statement is needed.

$$4^n, 8^n, n^2, n^n, n \lg n, e^{\ln(n)}, 2^{\sqrt{\lg n}}$$

**Solution.**

$n^n > 8^n > 4^n > n^2 > n \lg n > e^{\ln(n)} > 2^{\sqrt{\lg n}}$

2. (11 *points*) Prove or disprove the following statements. You should provide a formal proof or a counterexample for each statement. Please note that in the following statements, $f(n), g(n), i(n), j(n)$ are *non-negative*, *monotonically increasing* functions.
*non-negative*: The of ranges of these functions are all non-negative.
*monotonically increasing*:
If $n_1 > n_2$, then $f(n_1) \geq f(n_2), g(n_1) \geq g(n_2), i(n_1) \geq i(n_2), j(n_1) \geq j(n_2)$.

   a. (1 *points*) If $f(n) = O(i(n))$, $g(n) = O(j(n))$, then $f(n) - g(n) = O(i(n) - j(n))$.

**Solution.**

*Disprove.*
Let $f(n) = 2n, g(n) = i(n) = j(n) = n$
$\Rightarrow f(n) - g(n) = n, O(i(n) - j(n)) = O(0), n \neq O(0)$

   b. (2 *points*) If $f(n) = O(g(n))$, then $f^2 = O(g^2)$.

**Solution.**

*Prove.*
$f(n) = O(g(n))$

$\Rightarrow \exists c,\ n_0 \in \mathbb{N}$ s.t. $\forall n \geq n_0,\ f(n) \leq c \times g(n)$

$\Rightarrow \forall n \geq n_0,\ f^2(n) \leq (c \times g(n))^2) = c^2 \times g^2(n)$

Take $c' = c^2, n'_0 = n_0$

$\Rightarrow \forall n \geq n'_0,\ f^2(n) \leq c' g^2(n)$

$\Rightarrow f^2(n) = O(g^2(n))$

Another Solution due to Different Syntax Interpretation:
(interprete $f^2(n)$ as $f(f(n))$

*Disprove*

Take $f(n) = 2 \times 2^{2n},\ g(n) = 2^{2n} \Rightarrow f^2(n) = f(f(n)) = f(2 \times 2^{2n}) = 2 \times 2^{2 \times 2^{2n}} = 2 \times 4^{2^{2n}},\ g^2(n) = 2^{2^{2n}}$ Let $x = 2^{2n} \Rightarrow f^2(n) = 4^x,\ g^2(n) = 2^x \Rightarrow f^2(n) \neq O(g^2(n))$

c. (2 *points*) If $f(n) = \Omega(i(n)),\ g(n) = \Omega(j(n))$, then $f(n) + g(n) = \Omega(i(n) + j(n))$

**Solution.**

*Prove.*

$\exists c_1,\ n_1,\ c_2,\ n_2 \in \mathbb{N}$ s.t.

$\forall n \geq n_1,\ f(n) \leq c_1 \times i(n); \forall n \geq n_2,\ g(n) \leq c_2 \times j(n)$

$\Rightarrow \forall n \geq \max(n_1, n_2),\ f(n) + g(n) \geq c_1 \times i(n) + c_2 \times j(n) \geq \min(c_1, c_2) \times (i(n) + j(n))$

$\Rightarrow$ Take $c' = \min(c_1, c_2),\ n'_0 = \max(n_1, n_2)$

$\Rightarrow \forall n \geq n'_0,\ f(n) + g(n) \geq c' \times (i(n) + j(n))$

$\Rightarrow f(n) + g(n) = \Omega(i(n) + j(n))$

d. (1.5 *points*) If $f(n) = O(g(n)),$ then $2^{f(n)} = O(2^{g(n)})$

**Solution.**

*Disprove*

Let $f(n) = 2n,\ g(n) = n$

$\Rightarrow 2^{2n} = 4^n \neq O(2^n)$

e. (1.5 *points*) If $2^{f(n)} = O(2^{g(n)}),$ then $f(n) = O(g(n))$

**Solution.**

*Prove*

$2^{f(n)} = O(2^{g(n)})$

$\Rightarrow \exists c,\, n_0 \in \mathbb{N}$ s.t. $\forall n \geq n_0,\ 2^{f(n)} \leq c \times 2^{g(n)}$

$\Rightarrow \forall n \geq n_0,\ f(n) \leq \lg c + g(n)$

$\because g(n)$ is monotonically increasing

$\therefore \exists c' \in \mathbb{N}$ s.t. $\forall n \geq n_0,\ c' \times g(n) > \lg c$

$\Rightarrow \forall n \geq n_0,\ f(n) \leq \lg c + g(n) \leq (c'+1) * g(n)$

$\Rightarrow f(n) = O(g(n))$

f. (2 *points*) $f(n) + g(n) = \Theta(\max(f(n), g(n)))$

**Solution.**

*Prove*

$\max(f(n),\, g(n)) \leq f(n) + g(n) \leq 2 \times \max(f(n) + g(n))$

$\Rightarrow$ Take $c_1 = 1$, $c_2 = 2$, and then the statement is therefore proved.

3. a. (4 *points*) Define a function $f(n)$ as below:

$$f(n) = \begin{cases} 1 & \text{, if } n = 1 \\ 2f(\lfloor \frac{n}{2} \rfloor) + n & \text{, otherwise} \end{cases}$$

Please find out the tightest bound of $f(n)$ with $\Theta$ notation. You should provide a proof to get credits. Answers without any proof will NOT get any credits. (Note: If you want to use the Master Theorem, then you should prove it first)

Solution:

**Recursion Tree Method**

Please refer to the picture attached.

b. (5 *points*) Define a function $T(n)$ as below:

$$T(n) = \begin{cases} 1 & \text{, when } n = 1 \\ 2T(\lfloor \sqrt{n} \rfloor) + \lg n & \text{, otherwise} \end{cases}$$

Prove that $T(n) = O(\lg n \times \lg\lg n)$. (Hint: Transformation of variables.)

**Solution.**

Let $k = \lg(n)$, $n = 2^k$

$\Rightarrow T(2^k) = 2T(2^{k/2}) + k$

Let $S(k) = T(2^k)$

$\Rightarrow S(k) = 2S(k/2) + k$

By a.

$S(k) = O(k \lg k)$

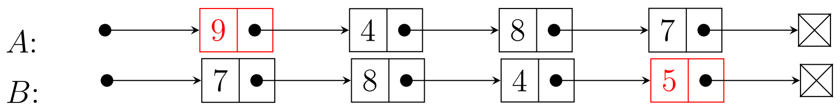
$\Rightarrow T(2^k) = O(k \lg k)$

$\Rightarrow T(n) = O(\lg n \times \lg\lg n)$

**Problem 3.** Linked List (25 *points*)

For each problem, if it's not trivial, please briefly explain why your procedure is correct and why the time/space complexity meets the constraints.

1. (5 *points*) You are given the heads of two read-only singly linked lists $A$ and $B$. Both $A$ and $B$ are loop-free and store positive integers. The integers in $A$, from head to tail, form a sequence $\langle a \rangle$, and the integers in $B$, from tail to head, form a sequence $\langle b \rangle$. Guaranteed that $\langle a \rangle$ and $\langle b \rangle$ are of same length $n$, and there exists exactly one $i$ s.t. $a_i \neq b_i$. Please describe a procedure to find $i$.

   $\langle a \rangle = (9, 4, 8, 7), \langle b \rangle = (5, 4, 8, 7), i = 1, a_i = 9, b_i = 5.$

   Your procedure should take the heads of $A$ and $B$ as its input, then output $i$, while $i$ is the index mentioned above. Both $A$ and $B$ are read-only, you cannot modify them.
   In $o(n^2)$ time, $O(1)$ extra space.

**Solution.**

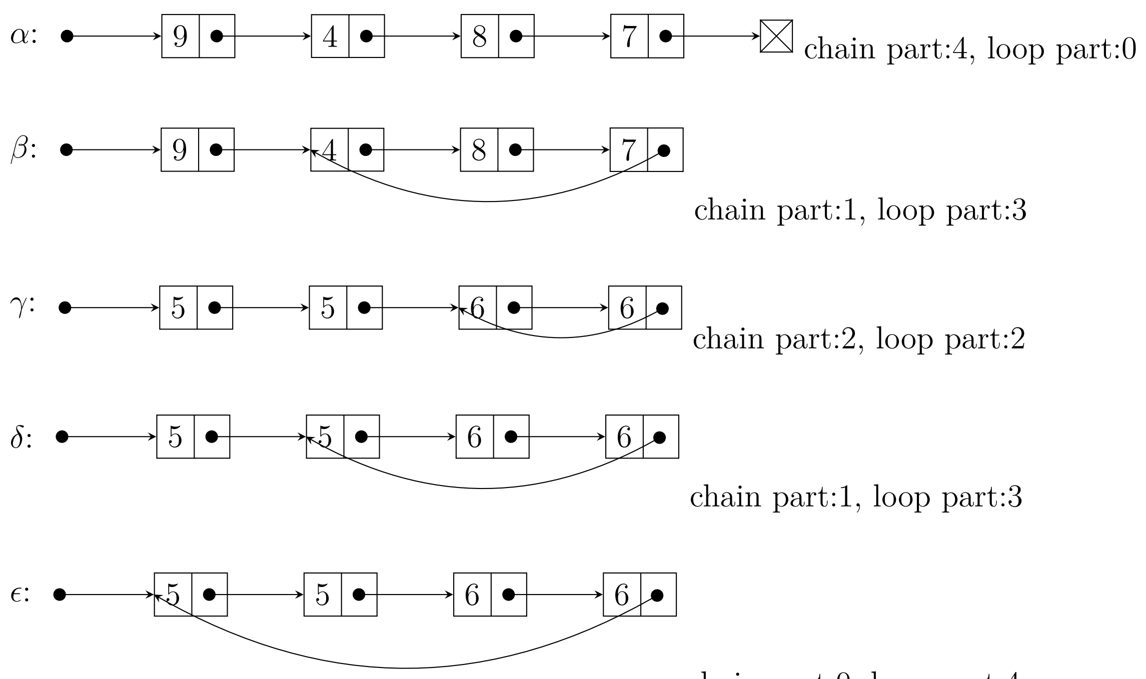
It's easy to verify that $\forall P \subseteq [1,n]$, $\sum\limits_{j\in P} a_j = \sum\limits_{j\in P} b_j$ iff $i \notin P$.

This gives a motivation to start a binary search to find the smallest $P$ that contains $i$, that is, the set that contains $i$ itself.

```
PROCEDURE 4.1($A.head, B.head$)
$leftA \leftarrow A.head$
$leftB \leftarrow B.head$
$ans \leftarrow 1$
$length \leftarrow n$
while $length > 1$
    $halfLength \leftarrow \lfloor length/2 \rfloor$
    $middleA \leftarrow A.head$
    $middleB \leftarrow B.head$
    $sumA, sumB \leftarrow 0$
    for $i = 1$ to $halfLength$
        $sumA \leftarrow sumA + middleA.data$
        $sumB \leftarrow sumB + middleB.data$
        $middleA \leftarrow middleA.next$
        $middleB \leftarrow middleB.next$
    if $sumA = sumB$
        $leftA \leftarrow middleA$
        $leftB \leftarrow middleB$
        $ans \leftarrow ans + halfLength$
        $length \leftarrow length - halfLength$
    else $length \leftarrow halfLength$
output $ans$
```

2. (20 *points*) You are given the heads of two read-only singly linked list $A$ and $B$. The tail node might not point to $NULL$ but point to another node, so they might contain loop. Please follow the guide and design a procedure to determine whether the shapes of the linked lists are identical.

   For example, among the following linked lists, linked list $\beta$ and linked list $\delta$ are identical in shape. You may discover that a linked list contains at most one loop, and if there is one, then the loop must appears at the end. Thus, two linked lists are identical in shape if and only if their "chain parts" and the "loop parts" are of equal sizes respectively, while the data they store are irrelevant to their shapes.



α:
9
4
8
7
chain part:4, loop part:0
β:
9
4
8
7
chain part:1, loop part:3
γ:
5
5
6
6
chain part:2, loop part:2
δ:
5
5
6
6
chain part:1, loop part:3
ϵ:
5
5
6
6
chain part:0, loop part:4

   Both $A$ and $B$ are read-only, you cannot modify them.

   a. (5 *points*) Please complete the C function below to determine whether the given linked list has a loop. If so, return a pointer to an arbitrary node on the loop. Otherwise, return $NULL$.

      In $O(n)$ time, $O(1)$ extra space, while $n$ is the number of nodes of the given linked list. Please note that the only input is the head of the linked list, and you are ignorant of the value of $n$.

      (Hint: Consider two pointers pointing to the head initially and moving at different speed.)

**Solution.**

```c
typedef struct listnode{
    int data;
    struct listnode *next;
}ListNode;

ListNode* isLoop(ListNode* head){
    ListNode* ptr_slow, ptr_fast=head;
    while(ptr_slow!=ptr_fast&&ptr_fast){
        ptr_slow=ptr_slow->next;
        ptr_fast=ptr_fast->next;
        if(ptr_fast->next)ptr_fast=ptr_fast->next;
    }
    return ptr_fast;
}
```

b. (3 *points*) Please complete the C function below to determine the length of the loop of the given linked list. The length of the loop is defined as the number of nodes on the loop. You are free to use function `isLoop()` defined above.
In $O(n)$ time, $O(1)$ extra space.

**Solution.**

```c
unsigned int loopLength(ListNode* head){
    ListNode* loopNode=isLoop(head);
    if(!loopNode)return 0;
    int l=1;
    ListNode* tmpNode=loopNode->next;
    while(tmpNode!=loopNode){
        l++;
        tmpNode=tmpNode->next;
    }
    return l;
}
```

Please describe a procedure that determine whether $A$ and $B$ are of same shape. Your procedure should takes the heads of $A$ and $B$ as its input. If they are of same shape, output 1, otherwise, output 0. In:

c. (6 *points*) $O(n \lg n)$ time, $O(1)$ extra space.

d. (6 *points*) $O(n)$ time, $O(n)$ extra space.
$n$ is the sum of the numbers of nodes of $A$ and $B$, and you are ignorant of the value of $n$. You are free to use functions `loopLength()` and `isLoop()` defined above.

**Solution.**

To make the notation consistent, we will use IS-LOOP and LOOP-LENGTH to denote the two function introduced above in the following procedure.

For simplicity, let's define two procedure FIND-NODE and FIND-INDEX first.

```
FIND-NODE($head, index$)
1  $tmpNode \leftarrow head$
2  while $index > 0$
3      $index \leftarrow index - 1$
4      $tmpNode \leftarrow tmpNode.next$
5  return $tmpNode$
```

```
FIND-INDEX($head, node$)
1  $tmpNode \leftarrow head$
2  $index \leftarrow 1$
3  while $tmpNode \neq node$
4      $index \leftarrow index + 1$
5      $tmpNode \leftarrow tmpNode.next$
6  return $index$
```

FIND-NODE is a procedure that returns the node of the given index. FIND-INDEX is a procedure that returns the index of the given node. It's easy to verify that both procedure comsumes linear time and constant extra space.

Note that two linked lists are identical in shape if and only if their chain parts and loop parts are of same length respectively. In problem (b), we have introcuded a $O(n)$-time and $O(1)$-space procedure LOOP-LENGTH to calculate the length of the loop part of a given linked list. Thus, it remains to calculate *chainLength*, the length of the chain part. The followings are two procedures for calculating *chainLength*, and meets the constraints of (c) and (d) respectively.

The procedure Chain-Length-C exploits the fact that if $chainLength > 0$, then *chainLength* is exactly the index of the last node that is not on the loop. Thus, we can use binary search to find such index. The upper bound is set to be the index of a node on the loop, while the lower bound is set to be 1. We can check if a node is on loop or not by checking if the next *loopLength* node is equal to its self.

```
Chain-Length-C($head$)
$left \leftarrow 1$
$right \leftarrow 1$
$loopNode \leftarrow$ Is-Loop($head$)
$right \leftarrow$ Find-Index($loopNode$)
if $loopNode = NULL$
    return $right - 1$
while $right - left > 1$
    $middle \leftarrow \lfloor (left + right)/2 \rfloor$
    $middleNode \leftarrow$ Find-Node($head, middle$)
    $middleNode' \leftarrow$ Find-Node($head, middle +$ LoopLength($head$))
    if $middleNode = middleNode'$
        $right \leftarrow middle$
    else
        $left \leftarrow middle$
return $left$
```

In the procedure CHAIN-LENGTH-D, we first call ISLOOP to find a node, say, $L$, on the loop. Then, we record two arrays, *chainArray* and *loopArray*. The former stores the node sequence from *head* to $L$, while the latter stores the node sequence from $L.next$ to $L$, that is, the whole loop. Now we construct two pointer initially pointed to $L$, and they both move backward one node per iteration. One moves along *chainArray* and the others move along *loopArray*. Once the two pointer point to different node, the former will be pointing at the last node that is not on loop.

```
CHAIN-LENGTH-D($head$)
 1  $loopNode \leftarrow isLoop(head)$
 2  $index \leftarrow$ FIND-INDEX($head, loopNode$)
 3  if $loopNode = NULL$
 4      return $index - 1$
 5  let $chainArray[1 \,..\, index]$ and $loopArray[1 \,..\, loopLength(head)]$ be new arrays
 6  $tmpNode \leftarrow head$
 7  for $i = 1$ to $index$
 8      $chainArray[i] \leftarrow tmpNode$
 9      $tmpNode \leftarrow tmpNode.next$
10  for $i = 1$ to $loopLength(head)$
11      $loopArray[i] \leftarrow tmpNode$
12      $tmpNode \leftarrow tmpNode.next$
13  $i_{chain} \leftarrow index$
14  $i_{loop} \leftarrow loopLength(head)$
15  while $chainArray[i_{chain}] \neq loopArray[i_{loop}]$
16      $i_{chain} \leftarrow i_{chain} - 1$
17      $i_{loop} \leftarrow i_{loop} - 1$
18  return $i_{chain}$
```

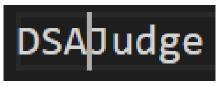
## Programming Part

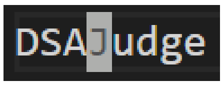
**Problem 4.** Notepad -- (20 *points*)

You are required to simulate a very lightweight text editor: *Notepad--*. The following table shows all the effective keys in this editor and their effects. All the other keys are ineffective and **should be ignored** by the editor.

| | |
|---|---|
| `a-z` | Overwrite the selected segment with the character; if nothing is selected, simply write the character. After that, move the cursor to the end of the inserted character, and exit *selection mode*. |
| `H` | Move the cursor leftward by one. If the cursor is at the beginning of the file, do nothing. |
| `L` | Move the cursor rightward by one. If the cursor is at the end of the file, do nothing. |
| `I` | Move the cursor to the beginning of the file. |
| `A` | Move the cursor to the end of the file. |
| `V` | Enter *selection mode*. If currently in *selection mode*, exit *selection mode*. |
| `D` | If nothing is selected, do nothing. Otherwise, cut the selected segment and overwrite the clipboard with it, and then exit *selection mode*. |
| `P` | Overwrite the selected segment with the content in the clipboard; if nothing is selected, simply paste the clipboard content. After that, clear the clipboard, move the cursor to the end of the pasted content, and exit *selection mode*. |

Note that in *Notepad--*, the cursor is NOT supposed to lie just upon a character; instead, it's always between two characters or at the beginning/end of file. Briefly speaking, it looks like the left figure instead of the right one.





Also note that the selection mode works just like keeping <shift> key pressed in Notepad or Word. More precisely, in *selection mode*, the ”selected segment” is the segment between the current cursor and the cursor position when you enter *selection mode*. While not in selection mode, nothing is selected.
Suppose *Arvin* just open an empty file with *Notepad--*, and the clipboard is initially empty. Given the series of keys pressed by *Arvin*, what would the file eventually be?

### Input Format

The first line is an integer $T$, indicating that there are $T$ test cases. For every test case, there

is a string composed of lowercase and uppercase characters, which is the series of keys *Arvin* keys in.
$T \leq 5$.
The length of each string $\leq 10^6$

### Output Format

For every test case, output the final content of the file.

### Input Constraint

For 30% of the testcases, there won't be any `I`, `A`, `V`, `D` or `P` in the input.
For 70% of the testcases, there won't be any `I` or `A` in the input.
For 100% of the testcases, there are no other constraints.

### Sample Input

```
5
tleVHMEOWHHwaVHBBHac
dsahahaVHHHHDggHHHHHPLLLLLgggqaq
notepadVHHHDHHVLHDHHDP
arvinLVHHDHHHHPVLPLDLP
dicxintheIVLLLLmeowAbox
```

### Sample Output

```
ac
hahadsagggggqaq
note
inrv
meowinthebox
```

**Solution.**
`https://hackmd.io/s/r1xgbdo_5G`

**Problem 5.** The Crafty TA (20 *points*)

As a crafty TA, *Maoq* would like to give students as few points as possible. The rules of giving points are as below. The student got $N$ points originally, and *Maoq* can adjust his points by removing $k$ arbitrary digits, but this removal shouldn't yield any leading zeros. Given $N$ and $k$, what's the minimum points the student would get?
Note: A single 0 is counted as leading zero.

## Input Format

The input contains 2 integers, $N$ and $k$. Both $N$ and $k$ will not contain any leading zeros.
$1 \leq N \leq 10^{100000000}$
$1 \leq k < len(N)$

## Output Format

An integer, which is the minimum points the student would get.

## Input Constraint

For 5% of the testcases, $len(N) \leq 10$, and each digit of $N$ is not zero.
For 30% of the testcases, $len(N) \leq 10^4$, and each digit of $N$ is not zero.
For 50% of the testcases, $len(N) \leq 10^4$.
For 100% of the testcases, there are no other constraints.

## Sample Input

```
30400 2
```

## Sample Output

```
300
```

**Solution.**
`https://hackmd.io/s/rkPN5tyjG`