HackerRank 题解

戴方勤 (soulmachine@gmail.com)

https://github.com/soulmachine/hackerrank

最后更新 2013-11-5

版权声明

内容简介

本书的目标读者是准备去北美找工作的码农，也适用于在国内找工作的码农，以及刚接触 ACM 算法竞赛的新手。

本书包含了 HackerRank(https://www.hackerrank.com/) 所有题目的答案，所有代码经过精心编写，编码规范良好，适合读者反复揣摩，模仿，甚至在纸上默写。

”ALGORITHMS” 和”MISCELLANEOUS CODE GOLF” 两部分的代码，使用 C++ 11；”ARTIFICIAL INTELLIGENCE” 和”FUNCTIONAL PROGRAMMING” 两部分的代码，使用 Scala。本书中的代码规范，跟在公司中的工程规范略有不同，为了使代码短（方便迅速实现）：

- Shorter is better。能递归则一定不用栈；能用 STL 则一定不自己实现。
- 不提倡防御式编程。不需要检查 malloc()/new 返回的指针是否为 nullptr；不需要检查内部函数入口参数的有效性。

本手册假定读者已经学过《数据结构》[1]，《算法》[2] 这两门课，熟练掌握 C++ 和 Scala。

GitHub 地址

本书是开源的，GitHub 地址：https://github.com/soulmachine/hackerrank

北美求职微博群

我和我的小伙伴们在这里：http://q.weibo.com/1312378

[1]《数据结构》，严蔚敏等著，清华大学出版社，http://book.douban.com/subject/2024655/

[2]《Algorithms》，Robert Sedgewick, Addison-Wesley Professional, http://book.douban.com/subject/4854123/

目录

第 1 章
Linked List

1.1 Print the elements of a linked list

If you’re new to working with linked lists, this is a great exercise to get familiar with them. You’re given the pointer to the head node of a linked list and you need to print all its elements in order, one element per line. The head pointer may be null meaning that the list is empty - in that case, don’t print anything!

Input Format

You have to complete the `void Print(Node* head)` method which takes one argument - the head of the linked list. You should NOT read any input from stdin/console.

Output Format

Print the elements of the linked list to stdout/console (using `printf` or `cout`) , one per line.

Sample Input

```
NULL
1->2->3->NULL
```

Sample Output

```
1
2
3
```

Explanation

There is no output for the first sample input, then print 1, 2 and 3 each in a new line.

分析

无

递归版

```
void Print(Node *head){
    if (head != nullptr) {
        cout << head->data << endl;
        Print(head->next);
    }
    return;
}
```

迭代版

```
void Print(Node *head){
    for (; head != nullptr; head = head->next) {
        cout << head->data << endl;
    }
    return;
}
```

相关题目

- Print the elements of a linked list in reverse, see §1.6

1.2 Insert a node at the tail of a linked list

You’re given the pointer to the head node of a linked list and an integer to add to the list. Create a new node with the given integer, insert this node at the tail of the linked list and return the head node. The head pointer given may be null meaning that the initial list is empty.

Input Format

You have to complete the `Node* Insert(Node* head, int data)` method which takes two arguments - the head of the linked list and the integer to insert. You should NOT read any input from stdin/console.

Output Format

Insert the new node at the tail and return the head of the updated linked list. Do NOT print anything to stdout/console.

Sample Input

```
NULL, data = 2
2 -> NULL, data = 3
```

Sample Output

```
2 -->NULL
2 --> 3 --> NULL
```

Explanation

1. We have an empty list and we insert 2.
2. We have 2 in the tail, when 3 is inserted 3 becomes the tail.

分析

无

代码

```
Node* Insert(Node *head, int data) {
    Node dummy{-1, head};
    Node *node;
    for (node = &dummy; node->next != nullptr; node = node->next);
    node->next = new Node();
    node = node->next;
    node->data = data;
    node->next = nullptr;
    return dummy.next;
}
```

相关题目

- 无

1.3 Insert a node at the head of a linked list

You’re given the pointer to the head node of a linked list and an integer to add to the list. Create a new node with the given integer, insert this node at the head of the linked list and return the new head node. The head pointer given may be null meaning that the initial list is empty.

Input Format

You have to complete the `Node* Insert(Node* head, int data)` method which takes two arguments - the head of the linked list and the integer to insert. You should NOT read any input from stdin/console.

Output Format

Insert the new node at the head and return the head of the updated linked list. Do NOT print anything to stdout/console.

Sample Input

```
NULL , data = 1
1 -> NULL , data = 2
```

Sample Output

```
1 --> NULL
2 --> 1 --> NULL
```

Explanation

1. We have an empty list, on inserting 1, 1 becomes new head.
2. We have a list with 1 as head, on inserting 2, 2 becomes the new head.

分析

无

代码

```
Node* Insert(Node *head, int data) {
    Node *node = new Node();
    node->data = data;
    node->next = head;
    return node;
}
```

相关题目

- 无

1.4 Insert a node at a specific position in a linked list

You’re given the pointer to the head node of a linked list, an integer to add to the list and the position at which the integer must be inserted. Create a new node with the given integer, insert this node at the desired position and return the head node. A position of 0 indicates head, a position of 1 indicates one node away from the head and so on. The head pointer given may be null meaning that the initial list is empty.

Input Format

You have to complete the `Node* Insert(Node* head, int data, int position)` method which takes three arguments - the head of the linked list, the integer to insert and the position at which the integer must be inserted. You should NOT read any input from stdin/console. position will always be between 0 and the number of the elements in the list (inclusive).

Output Format

Insert the new node at the desired position and return the head of the updated linked list. Do NOT print anything to stdout/console.

Sample Input

```
NULL, data = 3, position = 0
3 -> NULL, data = 4, position = 0
```

Sample Output

```
3 --> NULL
4 --> 3 --> NULL
```

Explanation

1. we have an empty list and position 0. 3 becomes head.
2. 4 is added to position 0, hence 4 becomes head.

分析

无

代码

```
Node* InsertNth(Node *head, int data, int position) {
    Node dummy{ -1, head };
    Node *curr = &dummy;
    for (int i = 0; i < position; ++i)
```

```
        curr = curr->next;
    Node *node = new Node();
    node->data = data;
    node->next = curr->next;
    curr->next = node;
    return dummy.next;
}
```

相关题目

- 无

1.5 Delete a node from a linked list

You’re given the pointer to the head node of a linked list and the position of a node to delete. Delete the node at the given position and return the head node. A position of 0 indicates head, a position of 1 indicates one node away from the head and so on. The list may become empty after you delete the node.

Input Format

You have to complete the `Node* Delete(Node* head, int position)` method which takes two arguments - the head of the linked list and the position of the node to delete. You should NOT read any input from stdin/console. position will always be at least 0 and less than the number of the elements in the list.

Output Format

Delete the node at the given position and return the head of the updated linked list. Do NOT print anything to stdout/console.

Sample Input

```
1 -> 2 -> 3 -> NULL, position = 0
1 -> NULL , position = 0
```

Sample Output

```
2 --> 3 --> NULL
NULL
```

Explanation

1. 0th position is removed, 1 is deleted from the list.
2. Again 0th position is deleted and we are left with empty list.

分析

无

代码

```
Node* Delete(Node *head, int position) {
    Node dummy{ -1, head };
    Node *curr = &dummy;
    for (int i = 0; i < position; ++i)
        curr = curr->next;
    Node *tmp = curr->next;
    curr->next = curr->next->next;
    delete tmp;
    return dummy.next;
}
```

相关题目

- 无

1.6 Print the elements of a linked list in reverse

You’re given the pointer to the head node of a linked list and you need to print all its elements in reverse order from tail to head, one element per line. The head pointer may be null meaning that the list is empty - in that case, don’t print anything!

Input Format

You have to complete the `void ReversePrint(Node* head)` method which takes one argument - the head of the linked list. You should NOT read any input from stdin/console.

Output Format

Print the elements of the linked list in reverse order to stdout/console (using printf or cout) , one per line.

Sample Input

```
1 -> 2 -> NULL
2 -> 1 -> 4 -> 5 -> NULL
```

Sample Output

```
2
1
5
4
1
2
```

Explanation

1. First list is printed from tail to head hence 2,1
2. Similarly second list is also printed from tail to head.

分析

无

迭代版

```
#include <stack>
void ReversePrint(Node *head) {
    stack<int> s;
    for (; head != nullptr; head = head->next) {
        s.push(head->data);
    }
    while (!s.empty()) {
        auto e = s.top();
        s.pop();
        cout << e << endl;
    }
}
```

递归版

```
void ReversePrint(Node *head) {
    if (head != nullptr) {
        ReversePrint(head->next);
        cout << head->data << endl;
    }
}
```

相关题目

- Print the elements of a linked list, see §1.1

1.7 Reverse a linked list

You’re given the pointer to the head node of a linked list. Change the next pointers of the nodes so that their order is reversed. The head pointer given may be null meaning that the initial list is empty.

Input Format

You have to complete the `Node* Reverse(Node* head)` method which takes one argument - the head of the linked list. You should NOT read any input from stdin/console.

Output Format

Change the next pointers of the nodes that their order is reversed and return the head of the reversed linked list. Do NOT print anything to stdout/console.

Sample Input

```
NULL
2 -> 3 -> NULL
```

Sample Output

```
NULL
3 --> 2 --> NULL
```

Explanation

1. Empty list remains empty
2. List is reversed from 3,2 to 2,3

分析

无

迭代版

```
Node* Reverse(Node *head) {
    if (head == nullptr || head->next == nullptr) return head;

    Node *prev = head;
    for (Node *curr = head->next, *next = curr->next; curr;
        prev = curr, curr = next, next = next ? next->next : nullptr) {
        curr->next = prev;
    }
    head->next = nullptr;
    return prev;
}
```

递归版

```
Node* Reverse(Node *head) {
    static Node* tail;
    if (head == nullptr || head->next == nullptr) {
        tail = head;
        return head;
    }

    Node *head_next = Reverse(head->next);
    if (tail != nullptr) {
        tail->next = head;
        head->next = nullptr;
    }
    tail = head;
    return head_next;
}
```

相关题目

- Reverse a doubly linked list, see §1.15

1.8 Compare two linked lists

You’re given the pointer to the head nodes of two linked lists. Compare the data in the nodes of the linked lists to check if they are equal. The lists are equal only if they have the same number of nodes and corresponding nodes contain the same data. Either head pointer given may be null meaning that the corresponding list is empty.

Input Format

You have to complete the `int Compare(Node* headA, Node* headB)` method which takes two arguments - the heads of the two linked lists to compare. You should NOT read any input from stdin/console.

Output Format

Compare the two linked lists and return 1 if the lists are equal. Otherwise, return 0. Do NOT print anything to stdout/console.

Sample Input

```
NULL, 1 -> NULL
1 -> 2 -> NULL, 1 -> 2 -> NULL
```

Sample Output

```
0
1
```

分析

递归版

```
int CompareLists(Node *headA, Node* headB) {
    if (headA == nullptr && headB == nullptr) return true;
    if (headA == nullptr || headB == nullptr) return false;
    return headA->data == headB->data  && CompareLists(headA->next, headB->next);
}
```

迭代版

```
int CompareLists(Node *headA, Node* headB) {
    for (; headA != nullptr || headB != nullptr;
        headA = headA ? headA->next : nullptr,
        headB = headB ? headB->next : nullptr) {
        if (headA == nullptr || headB == nullptr) return 0;
        if (headA->data != headB->data) return 0;
    }
    return 1;
}
```

Explanation

1. We compare an empty list with a list containing 1. They don’t match, hence return 0.
2. We have 2 similar lists. Hence return 1.

相关题目

- 无

1.9 Merge two sorted linked lists

You’re given the pointer to the head nodes of two sorted linked lists. The data in both lists will be sorted in ascending order. Change the next pointers to obtain a single, merged linked list which also has data in ascending order. Either head pointer given may be null meaning that the corresponding list is empty.

Input Format

You have to complete the `Node* MergeLists(Node* headA, Node* headB)` method which takes two arguments - the heads of the two sorted linked lists to merge. You should NOT read any input from stdin/console.

Output Format

Change the next pointer of individual nodes so that nodes from both lists are merged into a single list. Then return the head of this merged list. Do NOT print anything to stdout/console.

Sample Input

```
1 -> 3 -> 5 -> 6 -> NULL
2 -> 4 -> 7 -> NULL

15 -> NULL
12 -> NULL

NULL
1 -> 2 -> NULL
```

Sample Output

```
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7
12 -> 15 -> NULL
1 -> 2 -> NULL
```

Explanation

1. We merge elements in both list in sorted order and output.

分析

无

迭代版

```
#include <climits>
Node* MergeLists(Node *headA, Node* headB) {
    Node dummy;
    for (Node* p = &dummy; headA != nullptr || headB != nullptr; p = p->next) {
        int valA = headA == nullptr ? INT_MAX : headA->data;
        int valB = headB == nullptr ? INT_MAX : headB->data;
        if (valA <= valB) {
            p->next = headA;
            headA = headA->next;
        } else {
            p->next = headB;
            headB = headB->next;
        }
    }
    return dummy.next;
}
```

递归版

```
#include <climits>
Node* MergeLists(Node *headA, Node* headB) {
    if (headA == nullptr && headB == nullptr) return nullptr;
    Node *head = nullptr;
    int valA = headA == nullptr ? INT_MAX : headA->data;
    int valB = headB == nullptr ? INT_MAX : headB->data;
    if (valA <= valB) {
        head = headA;
        headA = headA->next;
    } else {
        head = headB;
        headB = headB->next;
    }

    head->next = MergeLists(headA, headB);
    return head;
}
```

相关题目

- 无

1.10 Get the value of the node at a specific position from the tail

You’re given the pointer to the head node of a linked list and a specific position. Counting backwards from the tail node of the linked list, get the value of the node at the given position. A position of 0 corresponds to the tail, 1 corresponds to the node before the tail and so on.

Input Format

You have to complete the `int GetNode(Node* head, int positionFromTail)` method which takes two arguments - the head of the linked list and the position of the node from the tail. positionFromTail will be at least 0 and less than the number of nodes in the list. You should NOT read any input from stdin/console.

Output Format

Find the node at the given position counting backwards from the tail. Then return the data contained in this node. Do NOT print anything to stdout/console.

Sample Input

```
1 -> 3 -> 5 -> 6 -> NULL, positionFromTail = 0
1 -> 3 -> 5 -> 6 -> NULL, positionFromTail = 2
```

Sample Output

```
6
3
```

分析

设两个指针 p, q，让 q 先走 $positionFromTail$ 步，然后 p 和 q 一起走，直到 q 走到尾节点，删除 `p->next` 即可。

代码

```
int GetNode(Node *head, int positionFromTail) {
    Node *p = head, *q = head;

    for (int i = 0; i < positionFromTail; i++)  // q 先走 n 步
        q = q->next;

    while (q->next) { // 一起走
        p = p->next;
        q = q->next;
    }

    return p->data;
}
```

相关题目

- 无

1.11 Delete duplicate-value nodes from a sorted linked list

You’re given the pointer to the head node of a sorted linked list, where the data in the nodes is in ascending order. Delete as few nodes as possible so that the list does not contain any value more than once. The given head pointer may be null indicating that the list is empty.

Input Format

You have to complete the `Node* RemoveDuplicates(Node* head)` method which takes one argument - the head of the sorted linked list. You should NOT read any input from stdin/console.

Output Format

Delete as few nodes as possible to ensure that no two nodes have the same data. Adjust the next pointers to ensure that the remaining nodes form a single sorted linked list. Then return the head of the sorted updated linked list. Do NOT print anything to stdout/console.

Sample Input

```
1 -> 1 -> 3 -> 3 -> 5 -> 6 -> NULL
NULL
```

Sample Output

```
1 -> 3 -> 5 -> 6 -> NULL
NULL
```

分析

无

递归版

```
void recur(Node *prev, Node *cur) {
    if (cur == nullptr) return;

    if (prev->data == cur->data) { // 删除 head
        prev->next = cur->next;
        delete cur;
        recur(prev, prev->next);
    } else {
        recur(prev->next, cur->next);
    }
}

Node* RemoveDuplicates(Node *head) {
    if (!head) return head;
    Node dummy{ head->data + 1, head }; // 值只要跟 head 不同即可

    recur(&dummy, head);
    return dummy.next;
}
```

递归版

```
Node* RemoveDuplicates(Node *head) {
    if (head == nullptr) return nullptr;

    for (Node *prev = head, *cur = head->next; cur; cur = cur->next) {
        if (prev->data == cur->data) {
            prev->next = cur->next;
            delete cur;
        } else {
            prev = cur;
        }
    }
    return head;
}
```

相关题目

- 无

1.12 Detect whether a linked list contains a cycle

You’re given the pointer to the head node of a linked list. Check if the next pointer of any of the nodes points to a previous node, thus forming a cycle.The head pointer given may be null meaning that the list is empty.

Input Format

You have to complete the `int HasCycle(Node* head)` method which takes one argument - the head of the linked list. You should NOT read any input from stdin/console.

Output Format

Check whether the linked list has a cycle and return 1 if there is a cycle. Otherwise, return 0. Do NOT print anything to stdout/console.

Sample Input

```
1 --> NULL
1 --> 2 --> 3
      ^     |
      |     |
       -----
```

Sample Output

```
0
1
```

分析

最容易想到的方法是，用一个哈希表 `unordered_map<int, bool> visited`，记录每个元素是否被访问过，一旦出现某个元素被重复访问，说明存在环。空间复杂度 $O(n)$，时间复杂度 $O(N)$。

最好的方法是时间复杂度 $O(n)$，空间复杂度 $O(1)$ 的。设置两个指针，一个快一个慢，快的指针每次走两步，慢的指针每次走一步，如果快指针和慢指针相遇，则说明有环。参考 http://leetcode.com/2010/09/detecting-loop-in-singly-linked-list.html

代码

```
int HasCycle(Node* head) {
    // 设置两个指针，一个快一个慢
    Node *slow = head, *fast = head;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) return true;
    }
    return false;
}
```

Explanation

1. First list has no cycle, hence return 0
2. Second list is shown to have a cycle, hence return 1.

相关题目

- 无

1.13 Find the merge point of two joined linked lists

You’re given the pointer to the head nodes of two linked lists that merge together at some node. Find the node at which this merger happens. The two head nodes will be different and neither will be NULL.

Input Format

You have to complete the `int FindMergeNode(Node* headA, Node* headB)` method which takes two arguments - the heads of the linked lists. You should NOT read any input from stdin/console.

Output Format

Find the node at which both lists merge and return the data of that node. Do NOT print anything to stdout/console.

Sample Input

```
1 --> 2 --> 3 --> NULL
      ^
      |
1-----

1 --> 2 --> 3 --> NULL
            ^
            |
      1-----
```

Sample Output

```
2
3
```

Explanation

1. As shown in the Input, 2 is the merge point.
2. Similarly 3 is merging point

分析

思路一：分别遍历两个单链表，计算出它们的长度 M 和 N，假设 M 比 N 大，则长度为 M 的单链表先前进 $M-N$，然后两个单链表同时前进，前进的同时比较当前的两个元素，如果相等，则必是交点。

思路二：将单链表的尾部链接到其中一个单链表的首节点，就构成了一个环，环的入口点就是交点。

代码

```
int ListLength(const Node *head) {
    int len = 0;
    for (; head; head = head->next)
        ++len;
    return len;
}

int FindMergeNode(Node *headA, Node *headB) {
    const int M = ListLength(headA);
    const int N = ListLength(headB);
    if (M < N) return FindMergeNode(headB, headA);

    for (int i = M - N; i > 0; --i)
        headA = headA->next;

    for (; headA; headA = headA->next, headB = headB->next) {
        if (headA == headB)
            return headA->data;
    }
}
```

相关题目

- 无

1.14 Insert a node into a sorted doubly linked list

You’re given the pointer to the head node of a sorted doubly linked list and an integer to insert into the list. The data in the nodes of the list are in ascending order. Create a node and insert it into the appropriate position in the list. The head node might be NULL to indicate that the list is empty.

Input Format

You have to complete the `Node* SortedInsert(Node* head, int data)` method which takes two arguments - the head of the sorted, doubly linked list and the value to insert. You should NOT read any input from stdin/console.

Output Format

Create a node with the given data and insert it into the given list, making sure that the new list is also sorted. Then return the head node of the updated list. Do NOT print anything to stdout/console.

Sample Input

```
NULL , data = 2
NULL <- 2 <-> 4 <-> 6 -> NULL , data = 5
```

Sample Output

```
NULL <-- 2 --> NULL
NULL <-- 2 <--> 4 <--> 5 <--> 6 --> NULL
```

Explanation

1. We han an empty list, 2 is inserted.
2. Data 5 is inserted such as list remains sorted.

分析

无

代码

```
#include <climits>
Node* SortedInsert(Node *head, int data) {
    Node dummy{ INT_MIN, head, nullptr };
    Node *prev = &dummy;
    while (prev->next && prev->next->data < data) {
        prev = prev->next;
    }
    Node *new_node = new Node();
    new_node->data = data;
    new_node->next = prev->next;
    new_node->prev = prev;
    prev->next = new_node;
    if (new_node->next) new_node->next->prev = new_node;

    dummy.next->prev = nullptr;
    return dummy.next;
}
```

相关题目

- 无

1.15 Reverse a doubly linked list

You’re given the pointer to the head node of a doubly linked list. Reverse the order of the nodes in the list. The head node might be NULL to indicate that the list is empty.

Input Format

You have to complete the `Node* Reverse(Node* head)` method which takes one argument - the head of the doubly linked list. You should NOT read any input from stdin/console.

Output Format

Change the next and prev pointers of all the nodes so that the direction of the list is reversed. Then return the head node of the reversed list. Do NOT print anything to stdout/console.

Sample Input

```
NULL
NULL <- 2 <-> 4 <-> 6 -> NULL
```

Sample Output

```
NULL
NULL <-- 6 <--> 4 <--> 2 --> NULL
```

Explanation

1. Empty list, so nothing to do.
2. 2,4,6 become 6,4,2 o reversing in the given doubly linked list.

分析

与 §1.7节"Reverse a linked list" 很类似。

迭代版

```
Node* Reverse(Node* head) {
    if (head == nullptr || head->next == nullptr) return head;

    Node *prev = head;
    for (Node *curr = head->next, *next = curr->next; curr;
        prev = curr, curr = next, next = next ? next->next : nullptr) {
            curr->next = prev;
            prev->prev = curr;
    }
    head->next = nullptr;
    prev->prev = nullptr;
    return prev;
}
```

递归版

```
Node* Reverse(Node* head) {
    static Node* tail;
    if (head == nullptr || head->next == nullptr) {
        tail = head;
        return head;
    }
    Node *head_next = Reverse(head->next);
    head_next->prev = nullptr;
    if (tail != nullptr) {
        tail->next = head;
        head->prev = tail;
        head->next = nullptr;
    }
    tail = head;
    return head_next;
}
```

相关题目

- Reverse a linked list, see §1.7

第 2 章 Sorting

2.1 Intro to Tutorial Challenges

About Tutorial Challenges

Many of the challenges on HackerRank are difficult and assume you already know the relevant algorithms very well. These tutorial challenges are different. They break down algorithmic concepts into smaller challenges, so you can learn the algorithm by doing the challenges.

These challenges are intended for people who know some programming and now want to learn some algorithms. You could be a student majoring in Computers, a self-taught programmer, or an experienced developer who wants an active algorithms review!

The first series of challenges covers sorting. These are the different challenges:

Tutorial Challenges - Sorting

Insertion Sort challenges

- Insertion Sort 1 - Inserting
- Insertion Sort 2 - Sorting
- Correctness of Algorithms
- Running Time of Algorithms

Quicksort challenges

- Quicksort 1 - Partition
- Quicksort 2 - Sorting
- Quicksort In-place (advanced)
- Running time of Quicksort

Counting sort challenges

- Counting Sort 1 - Counting
- Counting Sort 2 - Simple sort
- Counting Sort 3 - Preparing
- Full Counting Sort (advanced)

There will also be some challenges where you apply what you learned.

About the Challenges

The challenges will describe some topic and then ask you to code a solution. As you progress through the challenges, you will learn concepts in algorithms. On each challenge, you will receive input on STDIN and you will need to print the correct output to STDOUT.

For many challenges, helper methods will be provided for you to process the input into a useful format, like an array. You can use these methods to get started with your program, or you can write your own input methods if you want. Your code just needs to print the right output to each test case.

Sample Challenge

This is a simple challenge to get things started. Given a sorted array (ar) and a number (V), can you print the index location of V in the array?

The next section describes the input format. You can often skip it if you are using included methods.

Input Format

There will be three lines of input:

- V - the value you are looking for.
- n - the size of the array.
- ar - n numbers that makes up the array.

Output Format

Output the index of V in the array.

The next section describes the constraints and ranges of the input. Occasionally you want to check here to see how big the input could be.

Constraints

$1 \leq n \leq 1000$

$-1000 \leq x \leq 1000, x \in ar$

This “sample” shows the first input case. It is often useful to skip to the sample to understand a challenge.

Sample Input

```
4
6
1 4 5 7 9 12
```

Sample Output

```
1
```

Explanation

$V = 4$. The 4 is located in the 1th spot on the array, so the answer is 1.

分析

二分查找

代码

```
#include <iostream>
using namespace std;

/** 数组元素的类型 */
typedef int elem_t;
/**
* @brief 有序顺序表的折半查找算法.
*
* @param[in] a 存放数据元素的数组，已排好序
* @param[in] n 数组的元素个数
* @param[in] x 要查找的元素
* @return 如果找到 x，则返回其下标。 如果找
* 不到 x 且 x 小于 array 中的一个或多个元素，则为一个负数，该负数是大
* 于 x 的第一个元素的索引的按位求补。 如果找不到 x 且 x 大于 array 中的
* 任何元素，则为一个负数，该负数是（最后一个元素的索引加 1）的按位求补。
*/
int binary_search(const elem_t a[], const int n, const elem_t x) {
    int left = 0, right = n - 1, mid;
    while (left <= right) {
        mid = left + (right - left) / 2;
        if (x > a[mid]) {
            left = mid + 1;
        }
        else if (x < a[mid]) {
            right = mid - 1;
        }
        else {
            return mid;
        }
    }
    return -(left + 1);
}


int main() {
    int V, n;
    cin >> V >> n;
    int *ar = new int[n];
    for (int i = 0; i < n; ++i)
        cin >> ar[i];
    const int pos = binary_search(ar, n, V);
    cout << pos << endl;
    return 0;
}
```

相关题目

- 无

2.2 Insertion Sort - Part 1

Sorting

One common task for computers is to sort data. For example, people might want to see all their files on a computer sorted by size. Since sorting is a simple problem with many different possible solutions, it is often used to introduce the study of algorithms.

Insertion Sort

These challenges will cover Insertion Sort, a simple and intuitive sorting algorithm. We will first start with an already sorted list.

Insert element into sorted list

Given a sorted list with an unsorted number V in the right-most cell, can you write some simple code to insert V into the array so it remains sorted?

Print the array every time a value is shifted in the array until the array is fully sorted. The goal of this challenge is to follow the correct order of insertion sort.

Guideline: You can copy the value of V to a variable, and consider its cell “empty”. Since this leaves an extra cell empty on the right, you can shift everything over until V can be inserted. This will create a duplicate of each value, but when you reach the right spot, you can replace a value with V.

Input Format

There will be two lines of input:

- s - the size of the array
- ar - the sorted array of integers

Output Format

On each line, output the entire array every time an item is shifted in it.

Constraints

$1 \leq s \leq 1000$

$-10000 \leq x \leq 10000, x \in ar$

Sample Input

```
5
2 4 6 8 3
```

Sample Output

```
2 4 6 8 8
2 4 6 6 8
2 4 4 6 8
2 3 4 6 8
```

Explanation

3 is removed from the end of the array.

In the 1st line 8 > 3, 8 is shifted one cell right.

In the 2nd line 6 > 3, 6 is shifted one cell right.

In the 3rd line 4 > 3, 4 is shifted one cell right.

In the 4th line 2 < 3, 3 is placed at position 2.

分析

无

代码

```
void print(const vector<int> &ar) {
    for (auto e : ar)
        cout << e << " ";
    cout << endl;
}

void insertionSort(vector<int>  &ar) {
    const auto tmp = ar[ar.size() - 1];
    int i;
    for (i = ar.size() - 2; i >= 0 && ar[i] > tmp; --i) {
        ar[i + 1] = ar[i];
        print(ar);
    }
    ar[i+1] = tmp;
    print(ar);
}
```

相关题目

- 无