数据结构第二次作业

沈俞霖

2016 年 10 月 29 日

课程 数据结构与算法
姓名 沈俞霖
专业班级 软件 51
学号 2151601013
邮箱 sylxjtu@outlook.com
提交日期 2016 年 10 月 29 日

目录

1 实现 List

1.1 作业题目

未排序的顺序数组、已排序的顺序数组、未排序的单循环链表和已排序的单循环链表为了简化问题，这些数据结构都仅仅存储 int 类型，并且实现了如下抽象对象类型

```
// ListInterface.java 线性表接口
package sylxjtu;

interface List{
  boolean search(int x);     // 查询值为 x 的元素是否存在
  boolean insert(int x);     // 插入值为 x 的元素
  int delete(int x);         // 删除下标为 x 的元素
  int successor(int x);      // 获得该线性表中值为 x 的元素直接后继元素
  int predecessor(int x);    // 获得该线性表中值为 x 的元素的直接前驱元素
  int minimum();             // 获得该线性表的最小元素
  int maximum();             // 获得该线性表的最大元素
  int kthElement(int k);     // 获得线性表中第 k 大元素，参数为指定的 k 值的大小
}
```

1.2 程序实现

包括数据结构实现和测试代码，完整代码详见 Github 代码仓库 [1]

无序数组

```
// UnsortedArray.java 未排序数组实现线性表
package sylxjtu;

public class UnsortedArray implements sylxjtu.List{
  int[] data = new int[1];
  int capacity = 1;
  int size = 0;
```

[1] https://github.com/sylxjtu/datastructure-homework/tree/master/homework2/src

```java
void extend(){
  int[] tmp = new int[capacity * 2];
  System.arraycopy(data, 0, tmp, 0, capacity);
  capacity *= 2;
  data = tmp;
}

int find(int x){
  for(int i = 0; i < size; i++){
    if(data[i] == x) return i;
  }
  return -1;
}

@Override
public boolean search(int x){
  if(find(x) != -1) return true;
  return false;
}

@Override
public boolean insert(int x){
  if(size == capacity) extend();
  data[size++] = x;
  return true;
}

@Override
public int delete(int x){
  assert x >= 0 && x < size : "Invalid Index";
  int ret = data[x];
  for (int i = x; i < size - 1; i++) {
```

```
      data[i] = data[i + 1];
    }
    size--;
    return ret;
  }

  @Override
  public int successor(int x){
    int ind = find(x);
    assert ind >= 0 && ind < size - 1 : "No successor";
    return data[ind + 1];
  }

  @Override
  public int predecessor(int x){
    int ind = find(x);
    assert ind > 0 && ind < size : "No predecessor";
    return data[ind - 1];
  }

  @Override
  public int minimum(){
    assert size > 0 : "List is empty";
    int ret = data[0];
    for (int i = 1; i < size; i++) {
      ret = data[i] < ret ? data[i] : ret;
    }
    return ret;
  }

  @Override
  public int maximum(){
    assert size > 0 : "List is empty";
```

```
    int ret = data[0];
    for (int i = 1; i < size; i++) {
      ret = data[i] > ret ? data[i] : ret;
    }
    return ret;
  }

  @Override
  public int kthElement(int k){
    assert k > 0 && k <= size : "Invalid k";
    int[] tmp = data.clone();
    return ArrayUtil.findKthElement(tmp, 0, size, k - 1);
  }

  public static void main(String[] args) {
    UnsortedArray a = new UnsortedArray();
    a.insert(1);
    a.insert(3);
    a.insert(2);
    assert a.search(1);
    assert !a.search(4);
    assert a.delete(0) == 1;
    assert !a.search(1);
    a.insert(1);
    assert a.size == 3;
    assert a.successor(2) == 1;
    assert a.predecessor(2) == 3;
    assert a.minimum() == 1;
    assert a.maximum() == 3;
    assert a.kthElement(2) == 2;
    try {
      assert false;
    } catch (AssertionError e) {
```

```
      System.out.println("Test passed");
      return;
    }
    throw new Error("Please enable assertions");
  }
}
```

有序数组

```
// SortedArray.java 排序数组实现线性表
package sylxjtu;

public class SortedArray implements sylxjtu.List{
  int[] data = new int[1];
  int capacity = 1;
  int size = 0;

  void extend(){
    int[] tmp = new int[capacity * 2];
    System.arraycopy(data, 0, tmp, 0, capacity);
    capacity *= 2;
    data = tmp;
  }

  int find(int x){
    int l = 0, r = size, mid = (r + l) / 2;
    while(x != data[mid] && l < r - 1){
      if(data[mid] < x) l = mid;
      else r = mid;
      mid = (r + l) / 2;
    }
    return x == data[mid] ? mid : -1;
  }
```

```java
@Override
public boolean search(int x){
  if(find(x) != -1) return true;
  return false;
}

@Override
public boolean insert(int x){
  if(size == capacity) extend();
  if(size == 0 || x >= data[size - 1]) {
    data[size] = x;
  } else if(x <= data[0]) {
    for(int i = size; i > 0; i--){
      data[i] = data[i - 1];
    }
    data[0] = x;
  } else {
    for(int i = 0; i < size - 1; i++){
      if(data[i] < x && x <= data[i + 1]){
        for(int j = size; j > i + 1; j--){
          data[j] = data[j - 1];
        }
        data[i + 1] = x;
        break;
      }
    }
  }
  size++;
  return true;
}

@Override
public int delete(int x){
```

```
    assert x >= 0 && x < size : "Invalid Index";
    int ret = data[x];
    for (int i = x; i < size - 1; i++) {
      data[i] = data[i + 1];
    }
    size--;
    return ret;
  }

  @Override
  public int successor(int x){
    int ind = find(x);
    assert ind >= 0 && ind < size - 1 : "No successor";
    return data[ind + 1];
  }

  @Override
  public int predecessor(int x){
    int ind = find(x);
    assert ind > 0 && ind < size : "No predecessor";
    return data[ind - 1];
  }

  @Override
  public int minimum(){
    assert size > 0 : "List is empty";
    return data[0];
  }

  @Override
  public int maximum(){
    assert size > 0 : "List is empty";
    return data[size - 1];
```

```
  }

  @Override
  public int kthElement(int k){
    assert k > 0 && k <= size : "Invalid k";
    return data[k - 1];
  }

  public static void main(String[] args) {
    SortedArray a = new SortedArray();
    a.insert(1);
    a.insert(3);
    a.insert(2);
    assert a.search(1);
    assert !a.search(4);
    assert a.delete(0) == 1;
    assert !a.search(1);
    a.insert(1);
    assert a.size == 3;
    assert a.successor(2) == 3;
    assert a.predecessor(2) == 1;
    assert a.minimum() == 1;
    assert a.maximum() == 3;
    assert a.kthElement(2) == 2;
    try {
      assert false;
    } catch (AssertionError e) {
      System.out.println("Test passed");
      return;
    }
    throw new Error("Please enable assertions");
  }
}
```

无序链表

```java
// UnsortedLinkedList.java 未排序链表实现线性表
package sylxjtu;

public class UnsortedLinkedList implements sylxjtu.List{
  Node head;
  Node tail;
  int size = 0;

  Node find(int x){
    if(size == 0) return null;
    if(head.value == x) return head;
    Node cur = head.next;
    while(cur.value != x && cur != head){
      cur = cur.next;
    }
    if(cur == head) return null;
    else return cur;
  }

  @Override
  public boolean search(int x){
    if(find(x) != null) return true;
    return false;
  }

  @Override
  public boolean insert(int x){
    if(size == 0){
      head = tail = new Node(x);
      head.next = head;
    } else {
      tail.next = new Node(head, x);
```

```
      tail = tail.next;
    }
    size++;
    return true;
  }

  @Override
  public int delete(int x){
    assert x >= 0 && x < size : "Invalid Index";

    Node cur = head;
    for(int i = 0; i < x; i++){
      cur = cur.next;
    }
    int ret = cur.value;

    if(x == 0) head = tail.next = head.next;
    else {
      cur = head;
      for(int i = 0; i < x - 1; i++){
        cur = cur.next;
      }
      cur.next = cur.next.next;
    }

    size--;
    return ret;
  }

  @Override
  public int successor(int x){
    Node ind = find(x);
    assert ind != null && ind != tail : "No successor";
```

```
    return ind.next.value;
  }

  @Override
  public int predecessor(int x){
    Node ind = find(x);
    assert ind != null && ind != head : "No predecessor";
    Node cur = ind.next;
    while(cur.next != ind){
      cur = cur.next;
    }
    return cur.value;
  }

  @Override
  public int minimum(){
    int ret = head.value;
    for(Node cur = head.next; cur != head; cur = cur.next){
      ret = cur.next.value < ret ? cur.next.value : ret;
    }
    return ret;
  }

  @Override
  public int maximum(){
    int ret = head.value;
    for(Node cur = head.next; cur != head; cur = cur.next){
      ret = cur.next.value > ret ? cur.next.value : ret;
    }
    return ret;
  }

  @Override
```

```java
public int kthElement(int k){
  assert k > 0 && k <= size : "Invalid k";
  int[] arr = new int[size];
  Node cur = head;
  for(int i = 0; i < size; i++, cur = cur.next){
    arr[i] = cur.value;
  }
  return ArrayUtil.findKthElement(arr, 0, size, k - 1);
}

public static void main(String[] args) {
  UnsortedLinkedList a = new UnsortedLinkedList();
  a.insert(1);
  a.insert(3);
  a.insert(2);
  assert a.search(1);
  assert !a.search(4);
  assert a.delete(0) == 1;
  assert !a.search(1);
  a.insert(1);
  assert a.size == 3;
  assert a.successor(2) == 1;
  assert a.predecessor(2) == 3;
  assert a.minimum() == 1;
  assert a.maximum() == 3;
  assert a.kthElement(2) == 2;
  try {
    assert false;
  } catch (AssertionError e) {
    System.out.println("Test passed");
    return;
  }
  throw new Error("Please enable assertions");
```

```
  }
}
```

有序链表

```
// SortedLinkedList.java 排序链表实现线性表
package sylxjtu;

public class SortedLinkedList implements sylxjtu.List{
  Node head;
  Node tail;
  int size = 0;

  void print(){
    Node cur = head;
    for (int i = 0; i < size; i++) {
      System.out.print(cur.value + " ");
      cur = cur.next;
    }
    System.out.println();
  }

  Node find(int x){
    if(size == 0) return null;
    if(head.value == x) return head;
    Node cur = head.next;
    while(cur.value != x && cur != head){
      cur = cur.next;
    }
    if(cur == head) return null;
    else return cur;
  }

  @Override
```

```java
public boolean search(int x){
  if(find(x) != null) return true;
  return false;
}

@Override
public boolean insert(int x){
  if(size == 0){
    head = tail = new Node(x);
    head.next = head;
  } else {
    if(head.value > x){
      tail.next = new Node(head, x);
      head = tail.next;
    } else {
      for(Node cur = head; ; cur = cur.next){
        if(cur.next.value > x || cur == tail){
          cur.next = new Node(cur.next, x);
          if(cur.next.next == head) tail = tail.next;
          size++;
          return true;
        }
      }
    }
  }
  size++;
  return true;
}

@Override
public int delete(int x){
  assert x >= 0 && x < size : "Invalid Index";
```

```
    Node cur = head;
    for(int i = 0; i < x; i++){
      cur = cur.next;
    }
    int ret = cur.value;

    if(x == 0) {
      head = tail.next = head.next;
    } else {
      cur = head;
      for(int i = 0; i < x - 1; i++){
        cur = cur.next;
      }
      cur.next = cur.next.next;
    }

    size--;
    return ret;
  }

  @Override
  public int successor(int x){
    Node ind = find(x);
    assert ind != null && ind != tail : "No successor";
    return ind.next.value;
  }

  @Override
  public int predecessor(int x){
    Node ind = find(x);
    assert ind != null && ind != head : "No predecessor";
    Node cur = ind.next;
    while(cur.next != ind){
```

```
      cur = cur.next;
    }
    return cur.value;
  }

  @Override
  public int minimum(){
    return head.value;
  }

  @Override
  public int maximum(){
    return tail.value;
  }

  @Override
  public int kthElement(int k){
    Node cur = head;
    for(int i = 0; i < k - 1; i++, cur = cur.next)
      ;
    return cur.value;
  }

  public static void main(String[] args) {
    SortedLinkedList a = new SortedLinkedList();
    a.insert(1);
    a.insert(3);
    a.insert(2);
    assert a.search(1);
    assert !a.search(4);
    assert a.delete(0) == 1;
    assert !a.search(1);
    a.insert(1);
```

```
    assert a.size == 3;
    assert a.successor(2) == 3;
    assert a.predecessor(2) == 1;
    assert a.minimum() == 1;
    assert a.maximum() == 3;
    assert a.kthElement(2) == 2;
    try {
      assert false;
    } catch (AssertionError e) {
      System.out.println("Test passed");
      return;
    }
    throw new Error("Please enable assertions");
  }
}
```

1.3 程序运行结果与时间统计

运行结果 所有程序均通过测试

1.4 算法分析

数据结构	无序数组	有序数组	无序链表	有序链表
search	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$
insert	$O(1)$	$O(n)$	$O(1)$	$O(n)$
delete	$O(n)$	$O(n)$	$O(n)$	$O(n)$
successor	$O(1)$	$O(1)$	$O(1)$	$O(1)$
predecessor	$O(1)$	$O(1)$	$O(1)$	$O(1)$
minimum	$O(n)$	$O(1)$	$O(n)$	$O(1)$
maximum	$O(n)$	$O(1)$	$O(n)$	$O(1)$
kthElement	$O(n)$	$O(1)$	$O(n)$	$O(k)$

2 创建布尔表达式计算器

2.1 题目描述

本题是要计算如下的布尔表达式：$(T|T)\&F\&(F|T)$ 其中 T 表示 True，F 表示 False。表达式可以包含如下运算符：! 表示 not，& 表示 and，| 表示 or，允许使用括号。

为了执行表达式的运算，要考虑运算符的优先级：not 的优先级最高，or 的优先级最低。计算器要产生 V 或 F，表达最终表达式计算的结果。

对输入的表达式的要求如下：

1. 一个表达式不超过 100 个符号，符号间可以用任意个空格分开，或者根本没有空格，所以表达式总的长度也就是字符的个数，它是未知的。
2. 要能处理表达式中出现括号不匹配、运算符缺少运算操作数等常见的输入错误。

2.2 问题分析

表达式文法为

$$\begin{aligned} expr &\to andexpr|expr \\ &\to andexpr \end{aligned} \tag{1}$$

$$\begin{aligned} andexpr &\to notexpr\&expr \\ &\to notexpr \end{aligned} \tag{2}$$

$$\begin{aligned} notexpr &\to !notexpr \\ &\to atom \end{aligned} \tag{3}$$

$$\begin{aligned} atom &\to T \\ &\to F \\ &\to (expr) \end{aligned} \tag{4}$$

规约为 LL(1) 文法为

$$expr \to andexpr\ rexpr \tag{5}$$

$$\begin{aligned} rexpr &\rightarrow |expr \\ &\rightarrow \epsilon \end{aligned} \tag{6}$$

$$andexpr \rightarrow notexpr\ randexpr \tag{7}$$

$$\begin{aligned} randexpr &\rightarrow \&andexpr \\ &\rightarrow \epsilon \end{aligned} \tag{8}$$

$$\begin{aligned} notexpr &\rightarrow !notexpr \\ &\rightarrow atom \end{aligned} \tag{9}$$

$$\begin{aligned} atom &\rightarrow T \\ &\rightarrow F \\ &\rightarrow (expr) \end{aligned} \tag{10}$$

之后可以通过递归下降法求解

2.3 程序实现

```java
// BoolEquationEvaluate.java 布尔表达式求值
package sylxjtu;
import java.util.Scanner;
public class BoolEquationEvaluate {
  String s;
  int cur;
  BoolEquationEvaluate(String str){
    s = str + "$";
  }
  public boolean eval(){
    boolean ret = expr();
    if(s.charAt(cur) != '$'){
      System.err.println("Error: Unexpected " + s.charAt(cur) + " at position " + cur);
      System.err.println(s);
      for (int i = 0; i < cur; i++) {
```

```
        System.err.print(" ");
      }
      System.err.println("^");
      throw new Error("Invalid expression");
    }
    return ret;
  }
  void gonext(){
    do {
      cur++;
    } while(s.charAt(cur) == ' ');
  }
  boolean expr(){
    return andexpr() | rexpr();
  }
  boolean rexpr(){
    if(s.charAt(cur) == '|') {
      gonext();
      return expr();
    } else {
      return false;
    }
  }
  boolean andexpr(){
    return notexpr() & randexpr();
  }
  boolean randexpr(){
    if(s.charAt(cur) == '&') {
      gonext();
      return andexpr();
    } else {
      return true;
    }
```

```
  }
  boolean notexpr(){
    if(s.charAt(cur) == '!') {
      gonext();
      return !notexpr();
    } else {
      return atom();
    }
  }
  boolean atom(){
    if(s.charAt(cur) == 'T') {
      gonext();
      return true;
    } else if(s.charAt(cur) == 'F') {
      gonext();
      return false;
    } else if(s.charAt(cur) == '(') {
      gonext();
      boolean ret = expr();
      if(s.charAt(cur) != ')'){
        System.err.println("Error: Missing right branket at position " + cur);
        System.err.println(s);
        for (int i = 0; i < cur; i++) {
          System.err.print(" ");
        }
        System.err.println("^");
        throw new Error("Invalid expression");
      }
      gonext();
      return ret;
    } else {
      System.err.println("Error: Unexpected " + s.charAt(cur) + " at position " + cur);
      System.err.println(s);
```

```
      for (int i = 0; i < cur; i++) {
        System.err.print(" ");
      }
      System.err.println("^");
      throw new Error("Invalid expression");
    }
  }
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    while (scanner.hasNextLine()) {
      String line = scanner.nextLine();
      try {
        BoolEquationEvaluate x = new BoolEquationEvaluate(line);
        System.out.println(x.eval() ? 'V' : 'F');
      } catch (Error e) {}
      System.out.println();
    }
    scanner.close();
  }
}
```

2.4 程序运行结果与时间统计

运行结果 通过 POJ2109 测试

时间统计 POJ2109 测试，10 组表达式，用时 279ms

2.5 算法分析

根据递归下降法，每次扫描一个符号，故复杂度为 $O(N)$（N 为符号数）

3 队列实现基数排序

3.1 题目描述

利用队列实现对某一个数据序列的排序（采用基数排序），其中对数据序列的数据（第 1 和第 2 条进行说明）和队列的存储方式（第 3 条进行说明）有如下的要求：

1. 当数据序列是整数类型的数据的时候，数据序列中每个数据的位数不要求等宽，比如：
   1、21、12、322、44、123、2312、765、56
2. 当数据序列是字符串类型的数据的时候，数据序列中每个字符串都是等宽的，比如：
   ”abc”,”bde”,”fad”,”abd”,”bef”,”fdd”,”abe”
3. 要求重新构建队列的存储表示方法：使其能够将 n 个队列顺序映射到一个数组 listArray 中，每个队列都表示成内存中的一个循环队列

3.2 问题分析

首先确定排序的顺序，对于整数排序的顺序是按数的大小升序，对于字符串排序的顺序是字典序升序按照要求 3，通过一个循环数组实现队列的操作

3.3 程序实现

整数队列接口

```
// QueueInt.java 整形队列接口
package sylxjtu;
interface QueueInt {
  boolean empty();
  Pair front();
  void pop();
  void push(Pair elem);
}
```

整数队列实现

```java
// LoopArrayQueueString.java 循环数组实现字符串队列
package sylxjtu;
class LoopArrayQueueInt implements QueueInt{
  int capacity = 1;
  int size = 0;
  Pair[] arr;
  int fp, lp;
  LoopArrayQueueInt(){
    arr = new Pair[capacity];
  }
  void extend(){
    Pair[] tmp = new Pair[capacity * 2];
    System.arraycopy(arr, 0, tmp, 0, lp);
    System.arraycopy(arr, fp, tmp, fp + capacity, capacity - fp);
    fp += capacity;
    arr = tmp;
    capacity *= 2;
  }

  @Override
  public boolean empty(){
    return size == 0;
  }

  @Override
  public Pair front(){
    return arr[fp];
  }

  @Override
  public void pop(){
    assert size > 0 : "Queue is empty";
```

```
    fp++;
    fp %= capacity;
    size--;
  }

  @Override
  public void push(Pair elem){
    if(size == capacity) extend();
    arr[lp++] = elem;
    lp %= capacity;
    size++;
  }
}
```

字符串队列接口

```
// QueueString.java 字符串队列接口
package sylxjtu;
interface QueueString {
  boolean empty();
  String front();
  void pop();
  void push(String elem);
}
```

字符串队列实现

```
// LoopArrayQueueString.java 循环数组实现字符串队列
package sylxjtu;
class LoopArrayQueueString implements QueueString{
  int capacity = 1;
  int size = 0;
  String[] arr;
  int fp, lp;
  LoopArrayQueueString(){
```

```
    arr = new String[capacity];
  }
  void extend(){
    String[] tmp = new String[capacity * 2];
    System.arraycopy(arr, 0, tmp, 0, lp);
    System.arraycopy(arr, fp, tmp, fp + capacity, capacity - fp);
    fp += capacity;
    arr = tmp;
    capacity *= 2;
  }

  @Override
  public boolean empty(){
    return size == 0;
  }

  @Override
  public String front(){
    return arr[fp];
  }

  @Override
  public void pop(){
    assert size > 0 : "Queue is empty";
    fp++;
    fp %= capacity;
    size--;
  }

  @Override
  public void push(String elem){
    if(size == capacity) extend();
    arr[lp++] = elem;
```

```
    lp %= capacity;
    size++;
  }
}
```

整数基数排序

```
// IntegerRadixSort.java 正整数基数排序
// 考虑到模运算、除运算的速度，设置基数为 16，以便将其转化为位运算
// x % 16 -> x & 0xF
// x / 16 -> x >> 4
// StringRadixSort.java 字符串基数排序
package sylxjtu;

public class IntegerRadixSort {
  static void print(int[] arr){
    for(int i = 0; i < arr.length; i++){
      System.out.println(arr[i]);
    }
  }
  static void sort(int[] arr){
    LoopArrayQueueInt[] qarr = new LoopArrayQueueInt[16];
    Pair[] tarr = new Pair[arr.length];
    for (int i = 0; i < arr.length; i++) {
      tarr[i] = new Pair(arr[i], arr[i]);
    }
    for (int i = 0; i < qarr.length; i++) {
      qarr[i] = new LoopArrayQueueInt();
    }
    if(arr.length == 0) return;
    for(int l = 1; ; l++){
      boolean flag = false;
      for(int i = 0; i < arr.length; i++){
        if((tarr[i].curWeight & 0xF) != 0) flag = true;
```

```
        qarr[(tarr[i].curWeight & 0xF)].push(tarr[i]);
        tarr[i].curWeight >>= 4;
      }
      if(!flag) break;
      int j = 0;
      for(int i = 0; i < 16; i++){
        while(!qarr[i].empty()){
          tarr[j] = qarr[i].front();
          j++;
          qarr[i].pop();
        }
      }
    }
    for (int i = 0; i < arr.length; i++) {
      arr[i] = tarr[i].value;
    }
  }
  public static void main(String[] args) {
    int[] arr = {1, 21, 12, 322, 4, 123, 2312, 765, 56};
    print(arr);
    System.out.println("");
    sort(arr);
    print(arr);
  }
}
```

字符串基数排序

```
// StringRadixSort.java 字符串基数排序
package sylxjtu;
public class StringRadixSort {
  static void print(String[] arr){
    for(int i = 0; i < arr.length; i++){
      System.out.println(arr[i]);
```

```
    }
  }
  static void sort(String[] arr){
    LoopArrayQueueString[] qarr = new LoopArrayQueueString[26];
    for (int i = 0; i < qarr.length; i++) {
      qarr[i] = new LoopArrayQueueString();
    }
    if(arr.length == 0) return;
    int len = arr[0].length();
    for(int l = len - 1; l >= 0; l--){
      for(int i = 0; i < arr.length; i++){
        qarr[arr[i].charAt(l) - 'a'].push(arr[i]);
      }
      int j = 0;
      for(int i = 0; i < 26; i++){
        while(!qarr[i].empty()){
          arr[j] = qarr[i].front();
          j++;
          qarr[i].pop();
        }
      }
    }
  }
  public static void main(String[] args) {
    String[] arr = {"abc","bde","fad","abd","bef","fdd","abe"};
    print(arr);
    System.out.println("");
    sort(arr);
    print(arr);
  }
}
```

3.4 程序运行结果

运行结果 通过两组样例测试

3.5 算法分析

复杂度分析 基数排序复杂度为 $O(kN)$ ，其中 k 为最大数据长度，k 为数组大小