算法竞赛中的JAVA使用笔记

2016-11-14

[Algorithm](https://www.myblog.link/category/#Algorithm)

[Algorithm](https://www.myblog.link/tag/#Algorithm)  [Java](https://www.myblog.link/tag/#Java)

* [输入与输出](https://www.myblog.link/2016/11/14/Note-of-java/#section)
  + [基本输入](https://www.myblog.link/2016/11/14/Note-of-java/#section-1)
  + [输入挂](https://www.myblog.link/2016/11/14/Note-of-java/#section-2)
  + [输出](https://www.myblog.link/2016/11/14/Note-of-java/#section-3)
  + [控制台输入输出重定向到文件](https://www.myblog.link/2016/11/14/Note-of-java/#section-4)
* [大整数与高精度](https://www.myblog.link/2016/11/14/Note-of-java/#section-5)
  + [大整数BigInteger](https://www.myblog.link/2016/11/14/Note-of-java/#biginteger)
  + [高精度BigDecimal](https://www.myblog.link/2016/11/14/Note-of-java/#bigdecimal)
  + [高精度开方](https://www.myblog.link/2016/11/14/Note-of-java/#section-6)
* [字符串与进制转换](https://www.myblog.link/2016/11/14/Note-of-java/#section-7)
  + [字符串基本操作](https://www.myblog.link/2016/11/14/Note-of-java/#section-8)
  + [进制转换](https://www.myblog.link/2016/11/14/Note-of-java/#section-9)
* [排序](https://www.myblog.link/2016/11/14/Note-of-java/#section-10)
  + [默认排序](https://www.myblog.link/2016/11/14/Note-of-java/#section-11)
  + [实现Comparator接口自定义比较器](https://www.myblog.link/2016/11/14/Note-of-java/#comparator)
  + [对自定义类的排序](https://www.myblog.link/2016/11/14/Note-of-java/#section-12)
  + [用lambda自定义比较器(仅 JAVA8 以上支持)](https://www.myblog.link/2016/11/14/Note-of-java/#lambda-java8-)
* [C++STL中部分数据结构在JAVA中对应的用法](https://www.myblog.link/2016/11/14/Note-of-java/#cstljava)
  + [set](https://www.myblog.link/2016/11/14/Note-of-java/#set)
  + [map](https://www.myblog.link/2016/11/14/Note-of-java/#map)
  + [vector](https://www.myblog.link/2016/11/14/Note-of-java/#vector)
  + [list](https://www.myblog.link/2016/11/14/Note-of-java/#list)
  + [priority\_queue](https://www.myblog.link/2016/11/14/Note-of-java/#priorityqueue)
  + [queue](https://www.myblog.link/2016/11/14/Note-of-java/#queue)
  + [stack](https://www.myblog.link/2016/11/14/Note-of-java/#stack)
  + [deque](https://www.myblog.link/2016/11/14/Note-of-java/#deque)

本文介绍了Java在程序算法竞赛解题时常用的一些知识，包括基本的输入输出、Java的优势大数高精类、字符串与进制转换、排序以及C++ STL中部分数据结构在JAVA中对应的用法，旨在作为C/C++选手使用Java解题时的参考，并不会介绍基础的Java入门语法。

本文在本人新博客的链接：<http://www.myblog.link/2016/11/14/Note-of-java/>

输入与输出

基本输入

Scanner in = new Scanner (System.in);//基本方法

Scanner in = new Scanner (new BufferedInputStream(System.in));//更快

XXX foo = in.nextXXX();//然后这样给一个XXX类型的变量从标准输入获取值

while(in.hasNext()) doSomeThing();//循环到EOF时这么写，后面也可以加上变量类型

输入挂

class InputReader {

BufferedReader buf;

StringTokenizer tok;

InputReader() {

buf = new BufferedReader(new InputStreamReader(System.in));

}

boolean hasNext() {

while (tok == null || !tok.hasMoreElements()) {

try {

tok = new StringTokenizer(buf.readLine());

} catch (Exception e) {

return false;

}

}

return true;

}

String next() {

if (hasNext())

return tok.nextToken();

return null;

}

int nextInt() {

return Integer.parseInt(next());

}

long nextLong() {

return Long.parseLong(next());

}

double nextDouble() {

return Double.parseDouble(next());

}

BigInteger nextBigInteger() {

return new BigInteger(next());

}

BigDecimal nextBigDecimal() {

return new BigDecimal(next());

}

}

输出

System.out.println(n); //基本的输出方法

PrintWriter out = new PrintWriter(new BufferedOutputStream(System.out));//使用缓存加速，比直接使用System.out快

out.println(n);

out.printf("%.2f\n", ans); // 与c语言中printf用法相同

控制台输入输出重定向到文件

FileInputStream fis = new FileInputStream("b.in");

System.setIn(fis);

PrintStream ps = new PrintStream(new FileOutputStream("b.out"));

System.setOut(ps);

大整数与高精度

大整数BigInteger

import java.math.BigInteger;

//主要有以下方法可以使用：

BigInteger add(BigInteger other)

BigInteger subtract(BigInteger other)

BigInteger multiply(BigInteger other)

BigInteger divide(BigInteger other)

BigInteger [] dividedandRemainder(BigInteger other) //数组第一位是商，第二位是余数

BigInteger pow(int other)// other次方

BigInteger mod(BigInteger other)

BigInteger gcd(BigInteger other)

int compareTo(BigInteger other) //负数则小于,0则等于,正数则大于

static BigInteger valueOf(long x)

//输出数字时直接使用 System.out.println(a) 即可

高精度BigDecimal

BigDecimal add(BigDecimal other)

BigDecimal subtract(BigDecimal other)

BigDecimal multiply(BigDecimal other)

BigDecimal divide(BigDecimal other)

BigDecimal divide(BigDecimal divisor, int scale, BigDecimal.ROUND\_HALF\_UP)//除数，保留小数位数，保留方法四舍五入

BigDecimal.setScale()方法用于格式化小数点 //setScale(1)表示保留一位小数，默认用四舍五入方式

高精度开方

//求sqrt(x)，保留前n位数字（不是小数点后n位），n位后直接舍弃（非四舍五入）

private static BigDecimal sqrt(BigDecimal x, int n) {

BigDecimal ans = BigDecimal.ZERO;

BigDecimal eps = BigDecimal.ONE;

for (int i = 0; i < n; ++i) {

while (ans.pow(2).compareTo(x) < 0) {

ans = ans.add(eps);

}

ans = ans.subtract(eps);

eps = eps.divide(BigDecimal.TEN);

}

return ans;

}

字符串与进制转换

字符串基本操作

String st = "abcdefg";

char [] ch;

ch = st.toCharArray(); // 字符串转换为字符数组.

for (int i = 0; i < ch.length; i++){

ch[i] += 1; //字符数组可以像C++ 一样操作

}

System.out.println(ch); // 输入为“bcdefgh”.

进制转换

String s = Integer.toString(a, x); //把int型数据转换乘X进制数并转换成string型

// 0123456789abcdefghijklmnopqrstuvwxyz, 2<=x<=36

int b = Integer.parseInt(s, x);//把字符串当作X进制数转换成int型

排序

默认排序

原型：

Arrays.sort(int[] a, int fromIndex, int toIndex)

这种形式是对数组部分排序，也就是对数组a的下标从fromIndex到toIndex-1的元素排序， 注意：下标为toIndex的元素不参与排序哦！

实现Comparator接口自定义比较器

原型：

public static <T> void sort(T[] a,int fromIndex, int toIndex, Comparator<? super T> c)

将整形数组从大到小排序:

public class Main {

public static void main(String[] args) {

//注意，要想改变默认的排列顺序，不能使用基本类型（int,double, char）

//而要使用它们对应的类

Integer[] a = {9, 8, 7, 2, 3, 4, 1, 0, 6, 5};

//定义一个自定义类MyComparator的对象

Comparator cmp = new MyComparator();

Arrays.sort(a, cmp);

for(int i = 0; i < a.length; i ++) {

System.out.print(a[i] + " ");

}

}

}

//Comparator是一个接口，所以这里我们自己定义的类MyComparator要implents该接口

//而不是extends Comparator

class MyComparator implements Comparator<Integer>{

//返回值为负则o1排在o2前面，反正在后面，为0则表示相等

@Override

public int compare(Integer o1, Integer o2) {

//如果n1小于n2，我们就返回正值，如果n1大于n2我们就返回负值，

//这样颠倒一下，就可以实现反向排序了

if(o1 < o2) {

return 1;

}else if(o1 > o2) {

return -1;

}else {

return 0;

}

}

}

对自定义类的排序

对自己定义的类，除了上述在sort时制定比较器，还可以类似C++重载<，在定义类的时候实现Comparable接口，然后用方法1的语法进行排序，这样比较简洁，推荐！ 以下是一个完整例子：

import java.util.Arrays;

class Point implements Comparable<Point>{

int x,y;

//自定义的比较函数，跟2的语法类似，此例中先x后y从小到大排序

@Override

public int compareTo(Point o) {

return x!=o.x? x-o.x: y-o.y;

}

}

public class Main {

public static void main(String[] args) {

//Java里的数组要先new数组，再new每个元素，不是数组有了每个元素也就有了

Point[] p = new Point[3];

for(int i = 0;i < p.length;++i){

p[i] = new Point();

}

//不用上面的for把每个元素new出来直接进行下面的赋值会空指针的

//其实应该在Point里重载有参的构造函数，直接在new的时候初始化，这样代码简洁些

p[0].x = 3;

p[0].y = 3;

p[1].x = 1;

p[1].y = 4;

p[2].x = 3;

p[2].y = 1;

//sort还可以在第2、3个参数上指定排序起止

Arrays.sort(p);//先x后y从小到大排序

for(Point t:p){

System.out.println(t.x + " " + t.y);

}

}

}

用lambda自定义比较器(仅 JAVA8 以上支持)

方法2中对整形数组从大到小排序使用lambda可以直接写成：

Arrays.sort(a, (x,y)->(y-x));

方法3中排序自定义的Point类，不需要实现Comparable接口，可以直接这样：

Arrays.sort(p, (a,b)->(a.x!=b.x?a.x-b.x:a.y-b.y));

C++STL中部分数据结构在JAVA中对应的用法

set

如果要像C++中使用set进行去重，或者查询是否存在这方面的应用，在Java中主要使用HashSet类。 HashSet定义、插入、查询是否存在、删除元素的例子如下：

Set<Integer> s = new HashSet<Integer>();//无序，对应标准C++的unordered\_set

s.add(1);

System.out.println(s.contains(1) ? "1 is in set s" : "1 isn't in set s");

//根据key删除元素

m.remove(1);

Set遍历放在下文的Map中演示，因为Java中Map是转化为Set遍历的。

HashSet中元素是无序的（可以理解为顺序不确定），LinkedHashSet是遍历时是按照插入顺序排序的，TreeSet是升序排列的，最接近C++中的set，但是在没有要求元素有序的情况下，Java中一般是使用HashSet的（因为复杂度的优势?我感觉是这样的），这也是我在例子中使用HashSet来对应set的原因（其实我感觉C++中这种情况使用unordered\_set会更好啊，可能是因为C++11才出现，比较晚，所有不普及）。下节的map的情况与之类似。如果使用有序的TreeSet，还可以进行如下的查找操作：

TreeSet<Integer> s = new TreeSet<Integer>();

//使用s.add(1);等把1-5都加进去，代码省略

System.out.println(s.ceiling(3)); //>=3的最小的数，输出3

System.out.println(s.floor(3)); //<=3的最大的数，输出3

System.out.println(s.higher(3)); //>3的最小的数，输出4

System.out.println(s.lower(3)); //<3的最大的数，输出2

System.out.println(s.headSet(3)); //<3的数组成的TreeSet，输出[1, 2]

System.out.println(s.tailSet(3)); //>=3的数组成的TreeSet，输出[3, 4, 5]

System.out.println(s.subSet(2,4)); //>=2且<4的数组成的TreeSet，输出[2, 3]

System.out.println(s.subSet(2,false,4,true)); //>2且<=4的数组成的TreeSet，输出[3, 4]

map

如果只需要C++中map的key对value的映射功能，而不关心顺序，Java中一般使用HashMap类，例子如下：

//这里使用的是HashMap，是无序的，对于标准C++的unordered\_map

//定义与存取

Map<Integer, Integer> m = new HashMap<Integer, Integer>();

m.put(1, 111);

System.out.println(m.get(1));//如果get一个不存在的key，则返回null，否则返回对应value

//用迭代器遍历

Iterator<Entry<Integer, Integer>> it = m.entrySet().iterator();

while(it.hasNext()){

Entry<Integer, Integer> e = it.next();

System.out.println(e.getKey() + " " + e.getValue());

}

//根据key删除元素

m.remove(1);

//用for-each循环遍历

for(Map.Entry<Integer, Integer> e:m.entrySet()){

System.out.println(e.getKey() + " " + e.getValue());

}

如需有序，与Set类似，有LinkedHashMap、TreeMap等类可以使用。

vector

在Java中，C++的vector对应的是ArrayList类。虽然Java中也有Vector这个类，但它是历史遗留下来的，不建议使用。

ArrayList<Integer> a = new ArrayList<Integer>();//创建一个储存整形的ArrayList

a.add(1); //向其最后添加“1”这个元素

a.add(2); //向其最后添加“2”这个元素

a.add(1, 3); //向其index为1的位置添加“3”这个元素，原来index为1及后续元素向后顺延一位；index以0起始

System.out.println(a); //输出a，结果为[1, 3, 2]

a.remove(1); //删除index为1的元素，注意不是删除值为1的元素

System.out.println(a); //输出a，结果为[1, 2]

a.remove(Integer.valueOf(1)); //删除值为1的元素

System.out.println(a); //输出a，结果为[2]

a.set(0, 1); //将index为0的元素的值改成1

System.out.println(a.get(0)); //取出index为0的元素并输出，结果为1

list

在Java 中，C++的list对于LinkedList类，其基本用法跟ArrayList类似，只是实现上使用链表而不是数组，从而在一些操作的复杂度上有变化，将上文代码的ArrayList改为LinkedList可直接使用，故在此省略。（其实它还实现了C++中queue、deque、stack等的功能，有使用链表实现的这些数据结构的需求的话可以用它。）

priority\_queue

在Java中，C++的priority\_queue对应的是PriorityQueue类（终于碰到名字像的了？用起来都是坑啊）。示例如下：

PriorityQueue<Integer> pq = new PriorityQueue<Integer>();//定义一个储存整形的优先队列，值【小】的在前

pq.offer(1);//将1添加进去，不能用add(),虽然能过编译！！！

pq.offer(3);

pq.offer(2);

//跟C++的不同，你可以遍历它，但是你会发现遍历的结果并不是排序了的……我这里输出1 3 2

for(int t :pq){

System.out.print(t + " ");

}

System.out.println();

System.out.println(pq.peek());//取出第一个值（默认是最【小】的那个），并不删除它，这句代码输出1！！！

System.out.println(pq.poll());//取出第一个值（默认是最【小】的那个），并且删除它，这句代码输出1！！！

System.out.println(pq);//输出剩下的元素，结果是[2, 3]，但是并不是排序之后的！！！这只是巧合，不信试试其他值

用起来发现方法名都变得不认识了，可以遍历但是又无序，取数据的时候默认还是取最小的，跟C++相反。当然，可以自定义比较器：

class MyComp implements Comparator<Integer> {

@Override

public int compare(Integer o1, Integer o2) {

return o2 - o1;

}

}

public class Main {

public static void main(String[] args) {

PriorityQueue<Integer> pq = new PriorityQueue<Integer>(new MyComp());

// ……

}

}

觉得麻烦？那就直接lambda搞起

PriorityQueue<Integer> pq2 = new PriorityQueue<Integer>((a, b) -> (b - a));

queue

C++中的queue在Java中可以使用ArrayDeque类，实例如下：

ArrayDeque<Integer> queue = new ArrayDeque<Integer>();

queue.offer(1);//成功返回true，失败返回false,别写成push了，否则……看看下个例子就知道了

queue.offer(2);

queue.offer(3);

System.out.println(queue.peek());//类似于C++中queue的front()，返回第一个元素

while (!queue.isEmpty()) {

System.out.println(queue.pop());//跟C++中的queue()一样可以删除第一个元素，但是会返回它，不像C++中是void的

}

上述代码的输出为1、1、2、3（我就不换行了）。

stack

C++中的stack在Java中使用ArrayDeque类（你没看错，还是它，我知道Java也有Stack类，那也是历史遗留问题），语法基本相同，下面是例子：

ArrayDeque<Integer> stack = new ArrayDeque<Integer>();

stack.push(1);//跟上面那个代码的不同之处就在这了

stack.push(2);

stack.push(3);

System.out.println(stack.peek());//类似于C++中stack的top()，返回栈顶元素

while(!stack.isEmpty()){

System.out.println(stack.pop());//跟C++中的pop()一样可以弹出栈顶元素，但是会返回它，不像C++中是void的

}

上述代码的输出为3、3、2、1（我就不换行了）。

deque

deque对应的……不用说了，就是ArrayDeque了。如果你已经被上面的方法名搞晕了的话，试试用下面几个：

ArrayDeque<Integer> deque = new ArrayDeque<Integer>();

deque.addFirst(2);

deque.addFirst(1);//往头加

deque.addLast(3);//往尾加

System.out.println(deque.getFirst());//从头取

System.out.println(deque.getLast());//从尾取

System.out.println(deque.removeFirst());//从头删

System.out.println(deque.removeLast());//从尾删

以上代码输出1、3、1、3。这些方法加入的时候已经满了则抛出IllegalStateException异常，读取或删除的时候为空则抛出NoSuchElementException异常。

不要忘了ArrayDeque是可以遍历的哟，包括把它当stack或者queue用的时候……