Java实现二叉堆、大顶堆和小顶堆

作者：有故事的人 来源：范文网 www.wtabcd.cn/fanwen/

本文原地址：https://www.wtabcd.cn/fanwen/zuowen/c554715f398305a344c06a88a9001761.html

目录

什么是二叉堆

二叉堆就是完全二叉树，或者是靠近完全二叉树结构的二叉树。在二叉树建树时采取前序建树就是建立的完全二叉树。也就是二叉堆。所以二叉堆的建堆过程理论上讲和前序建树一样。

什么是大顶堆、小顶堆

二叉堆本质上是一棵近完全的二叉树，那么大顶堆和小顶堆必然也是满足这个结构要求的。在此之上，大顶堆要求对于一个节点来说，它的左右节点都比它小；小顶堆要求对于一个节点来说，它的左右节点都比它大。

建堆

二叉堆建堆本质上和前序建堆差不多，只不过需要考虑的一点就是大小关系，这一点和二叉搜索树建树有点相似，所以可以得出结论，建树，本质上都是递归建树，只不过因为数据结构的大小要求不一样，需要的判断函数不一样，节点进入哪个位置也不一样。

大顶堆和小顶堆也分为稳定和不稳定的堆。稳定和不稳定指如果具备相同的值，那么他们的插入顺序应该和节点顺序一致。

程序实现

首先，定义出基本的堆结构

```
public class binaryheap {  private integer value;  private binaryheap leftchild;  private binaryheap righ
tchild;}
```

建堆过程与建二叉树过程一致

```
public static binaryheap buildheap(binaryheap binaryheap, integer value) {  if (objects.isnull(binaryheap)) b
inaryheap = new binaryheap();  if (objects.isnull(binaryheap.getvalue())) {    binaryheap.setvalue(value);
```

付出的爱return binaryheap; } if (objects.isnull(binaryheap.getleftchild())) { binaryheap binary heap1 = new binaryheap(); binaryheap1.setvalue(value); binaryheap.setleftchild(binaryheap1); } else if (objects.nonnull(binaryheap.getleftchild())) { if (objects.isnull(binaryheap.getrightchild())) { binaryheap binaryheap1 = new binaryheap(); binaryheap1.setvalue(value); 酱肉的制作方法 binaryheap.setrightchild(binaryheap1); } else { // todo: 2022/1/14 左右节点两种都不为null if (checknull(binaryheap.getleftchild())) buildheap(binaryheap.getleftchild(), value); else if (checknull(binaryheap.getrightchild())) buildheap(binaryheap.getrightchild(), value); else buildheap (binaryheap.getleftchild(), value); } } return binaryheap;}

主要原理就是如果当前节点的左节点为空，则把当前值放到左节点，如果左节点不为空，右节点为空，则把值放到右节点。如果左右节点都不为空，就将建树过程转移到下一层，如果左节点有为空的子节点，就转移给左节点，如果左节点没有为空的子节点，且右节点有为空的子节点，那么转移给右节点。如果左右节点都没有为空的子节点，那么也转移给左节点。

以序列2,3,4,5,9,6,8,7为例，按照该算法建立出来的二叉堆结构如下：

{ "value": 2, "left_child": { "value": 3, "left_child": { "value": 4, "left_child": { "value": 8, 6月的英文"left_child": null, "right_child": null }, "right_child": { "value": 7, "left_child": null, "right_child": null } }, "right_child": { "val ue": 5, "left_child": null, "right_child": null } }, "right_child": { "value": 1, "left_child": { "value": 9, "left_child": null, "right_child": null }, "right_child": { "value": 6, "l eft_child": null, "right_child": null } }}

建立大顶堆

大顶堆在建堆的基础上，有一个要求，根节点比左右子树的任何节点的值都大。那么建树的过程可以分为两步，对于每一个值，首先按照建树过程，会到二叉堆的最底部，然后通过不断的让自己与自己的根节点做比较，如果自己大于根节点，就交换自己与根节点的位置，递归回溯即可。

逻辑过程

假设现在红色节点组成的已经是一个大顶堆，现在新增了一乌克兰和俄罗斯的关系个节点到这个二叉堆中，而且是比任意节点都大，那么黑色箭头将是该节点的行动路线，它会反复与父级比较，如果大于父级，则交换和父级的关系。

程序实现

public static binaryheap up(binaryheap father) { if (objects.nonnull(father.getleftchild())) { if (father.getvalue() < father.getleftchild().getvalue()) { int c = father.getvalue(); father.setvalue(father.getleftchild().getvalue()); father.getleftchild().setvalue(c); } up(father.getleftchild()); } if (objects.nonnull(father.getrightchild())) { if (father.getvalue() < father.getrightchild().getvalue()) { int c = father.getvalue(); father.setvalue(father.getrightchild().getvalue()); father.getrightchild().setvalue(c); } up(father.getrightchild()); } return father;}

该方法放在普通建树方法之后，就是大顶堆的建树方法了，总的方法如下：

public static binaryheap bigpush(binaryheap binaryheap, integer value) { binaryheap = buildheap(bin

aryheap, value); up(binaryheap); return binaryheap;}

还是以序列2,3,4,5,9,6,8,7为例，按照该算法建立出来的大顶堆结构如下：

{ "value": 9, "left_child": { "value": 8, "left_child": { "value": 7, "left_child": { "value": 2, "left_child": null, "right_child": null }, "right_child": { "value": 4, "left_child": null, "right_child": null } }, "right_child": { "value": 3, "left_child": null, "right_child": null } }, "right_child": { "value": 6, "left_child": { "value": 1, "left_child": null, "right_child": null }, "right_child": { "value": 5, "left_child": null, "right_child": null } }}

建立小顶堆

小顶堆与大顶堆类似

逻辑过程

过程与大顶堆一致，不过此时是比父级小就和父级交换。

程序实现

```
public static binaryheap down(binaryheap father) { if (objects.nonnull(father.getleftchild())) {
if (father.getvalue() > father.getleftchild().getvalue()) { int c = father.getval
ue(); father.setvalue(father.getleftchild().getvalue()); father.getleftchil
d().setvalue(c); } down(father.getleftchild()); } if (objects.nonnull(fa
ther.getrightchild())) { if (father.getvalue() > father.getrightchild().getvalue()) {
int c = father.getvalue(); father.setvalue(father.getrightchild().getvalue());
father.getrightchild().setvalue(c); } down(father.getrightchild());
} return father;}
```

这个是向下走的过程，最终代码为：

```
public static binaryheap smallpush(binaryheap binaryheap, integer value) { binaryheap = buildheap(bi
naryheap, value); down(binaryheap); return binaryheap;}
```

以序列2,3,4,5,9,6,8,7为例，按照该算法建立出来的小顶堆结构如下：

{ "value": 1, "left_child": { "value": 3, "left_child": { "value": 4, "left_child": { "value": 8, "left_child": null, "right_child": null }, "right_child": { "value": 7, "left_child": null, "right_child": null } }, "right_child": { "value": 5, "left_child": null, "right_child": null } }, "right_child": { "value": 2, "left_child": { "value": 9, "left_child": null, "right_child": null }, "right_child": { "value": 6, "left_child": null, "right_child": null } }}

从堆顶取数据并重构大小顶堆

```
public static integer bigpop(binaryheap binaryheap) {  integer val = binaryheap.getvalue();  if
(binaryheap.getleftchild().getvalue() >= binaryheap.getrightchild().getvalue()) {  binaryheap.setvalue(bi
naryheap.getleftchild().getvalue());  binaryheap binaryheap1 = mergetree(binaryheap.getleftchild().getle
ftchild(), binaryheap.getleftchild().getrightchild());  up(binaryheap1);  binaryheap.setleft
child(binaryheap1);  } else {  binaryheap.setvalue(binaryheap.getrightchild().getvalue());
  binaryheap binaryheap1 = mergetree(binaryheap.getrightchild().getleftchild(), binaryheap.getrightchild().get
rightchild());  up(binaryheap1);  binaryheap.setrightchild(binaryheap1);  }
 return val;}public st浙江农林大学是几本atic integer smallpop(binaryheap binaryheap) {  integer val
= binaryheap.getvalue();  if (binaryheap.getleftchild().getvalue() <= binaryheap.getrightchild().getvalue()) {
  binaryheap.setvalue(binaryheap.getleftchild().getvalue());  binaryheap binaryheap1 =
mergetree(binaryheap.getleftchild().getleftchild(), binaryheap.getleftchild().getrightchild());  down(binar
yheap1);  binaryheap.setleftchild(binaryheap1);  } else {  binaryheap.setvalue(
binaryheap.getrightchild().getvalue());  binaryheap binaryheap1 = mergetree(binaryheap.getrightchild()
.getleftchild(), binaryheap.getrightchild().getrightchild());  down(binaryheap1);  binaryh
eap.setrightchild(binaryheap1);  }  return val;}
```

取出来之后，需要重新调用down或者up函数。以构建小顶堆，取出五次后的结果

```
public static void main(string[] args) {  int[] a = new int[]{2, 3, 1, 4, 5, 9, 6, 8,
7};  binaryheap binaryheap = new binaryheap();  for (int i = 0; i <
a.length; i++) {  binaryheap = smallpush(binaryheap, a[i]);  }
  system.out.println(json.tojson(smallpop(binaryheap)));  system.out.println(json.tojson(smallpop
(binaryheap)));  system.out.println(json.tojson(smallpop(binaryheap)));  system.out.printl
n(json.tojson(smallpop(binaryheap)));  system.out.println(json.tojson(smallpop(binaryheap)));
 system.out.println(json.tojson(binaryheap));  }
```

取完后的小顶堆为：

```
{  "value": 6,  "left_child": {  "value": 7,  "left_child": {
  "value": 8,  "left_child": null,  "right_child":
null  },  "right_child": null  },  "right_child": {  "val
ue": 9,  "left_child": null,  "right_child": null  }}
```

到此这篇关于java实现二叉堆、大顶堆和小顶堆的文章就介绍到这了,更多相关java内容请搜索www.887551.com以前的文章或继续浏览下面的相关文章希望大家以后多多支持www.887551.com！