数据结构与算法实战
第五讲 优先队列与集合

5.1 优先队列（Priority Queue）

5.1.1 什么是优先队列

优先队列（Priority Queue）：

- 出队：最高（最低）优先级的元素
- 入队：无特别要求

5.1.1 什么是优先队列

探讨实现方式:

- 数组?
- 链表?
- 排序数组?
- 排序链表?
- 二叉搜索树?
- AVL树?

5.1.1 什么是优先队列

	插入	查找	删除	评价
数组	$\Theta(1)$	$\Theta(N)$	$\Theta(1)$	查找慢
单链表	$\Theta(1)$	$\Theta(N)$	$\Theta(1)$	查找慢，空间费
排序数组	$O(logN)+O(N)$	$\Theta(1)$	$\Theta(1)$	需排序,插入慢,需移动
排序单链表	$O(N)+O(1)$	$\Theta(1)$	$\Theta(1)$	需排序,插入慢,空间费
二叉搜索树	$O(logN)$?	$O(logN)$?	$O(logN)$?	严重失衡
AVL树	$O(logN)$	$O(logN)$	$O(logN)$	操作复杂,功能太多

5.1.1 什么是优先队列

有没有更好的实现方案?

要求:

- 只关注“优先级最高（最低）元素出队”
- 空间省

思路:

空间省 => 顺序实现

减少比较或元素移动次数 => 逻辑结构是二维的

你想到了什么?

5.1.1 什么是优先队列

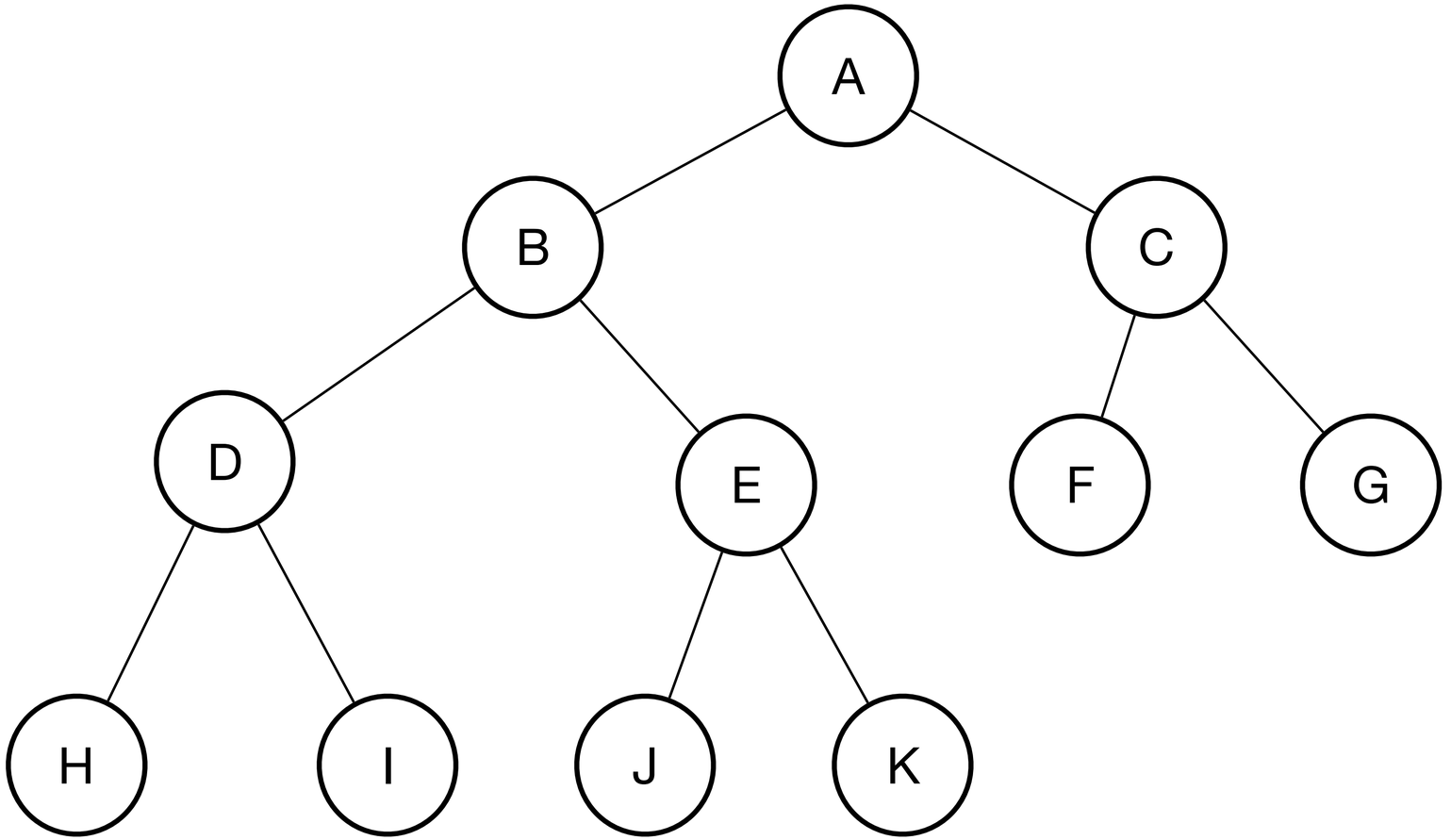
完全二叉树！

可以用数组存储！

两个问题：

1、要使最高（最低）优先级元素迅速出队，该元素放到哪儿呢？

2、这个树如何组织，使得插入、删除操作比较便捷呢？



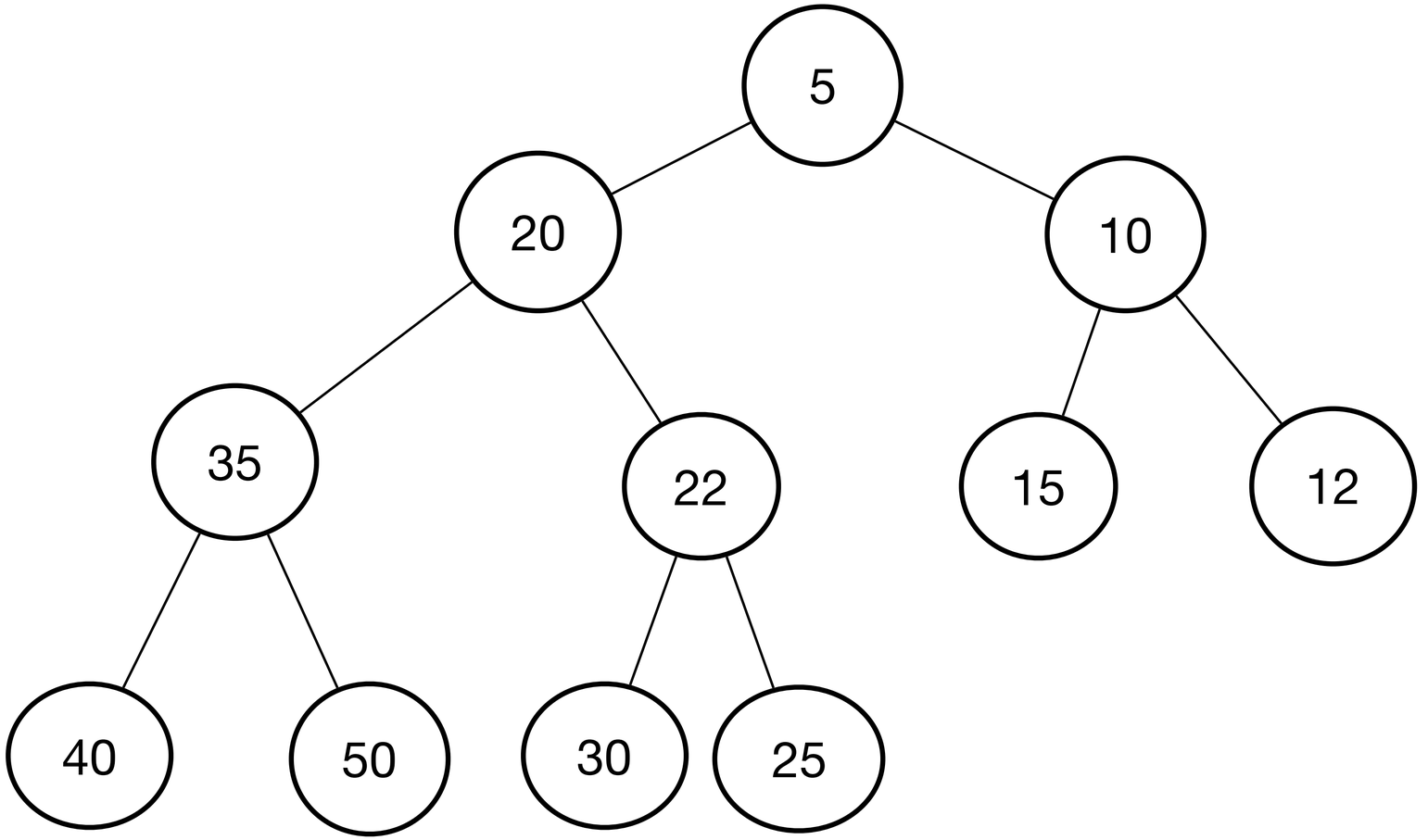
5.1.1 什么是优先队列

最小树：

每个节点的元素均不大于其子节点的元素。

最小堆：（二叉堆）

即是完全二叉树又是最小树。

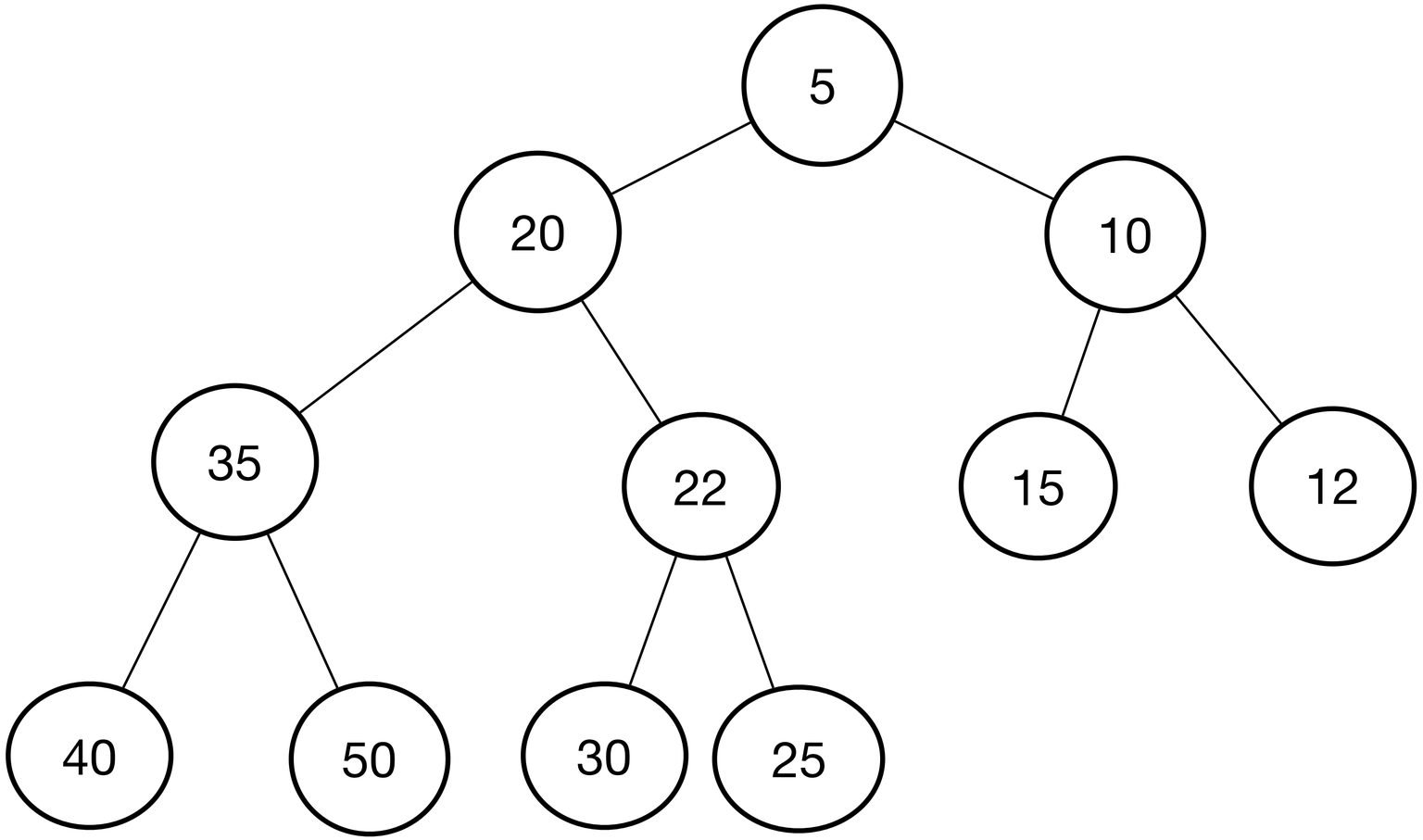


我们用“堆”（heap）来实现“优先队列”！

5.1.2 堆的创建与插入操作

创建：空堆

插入：插在哪儿呢?

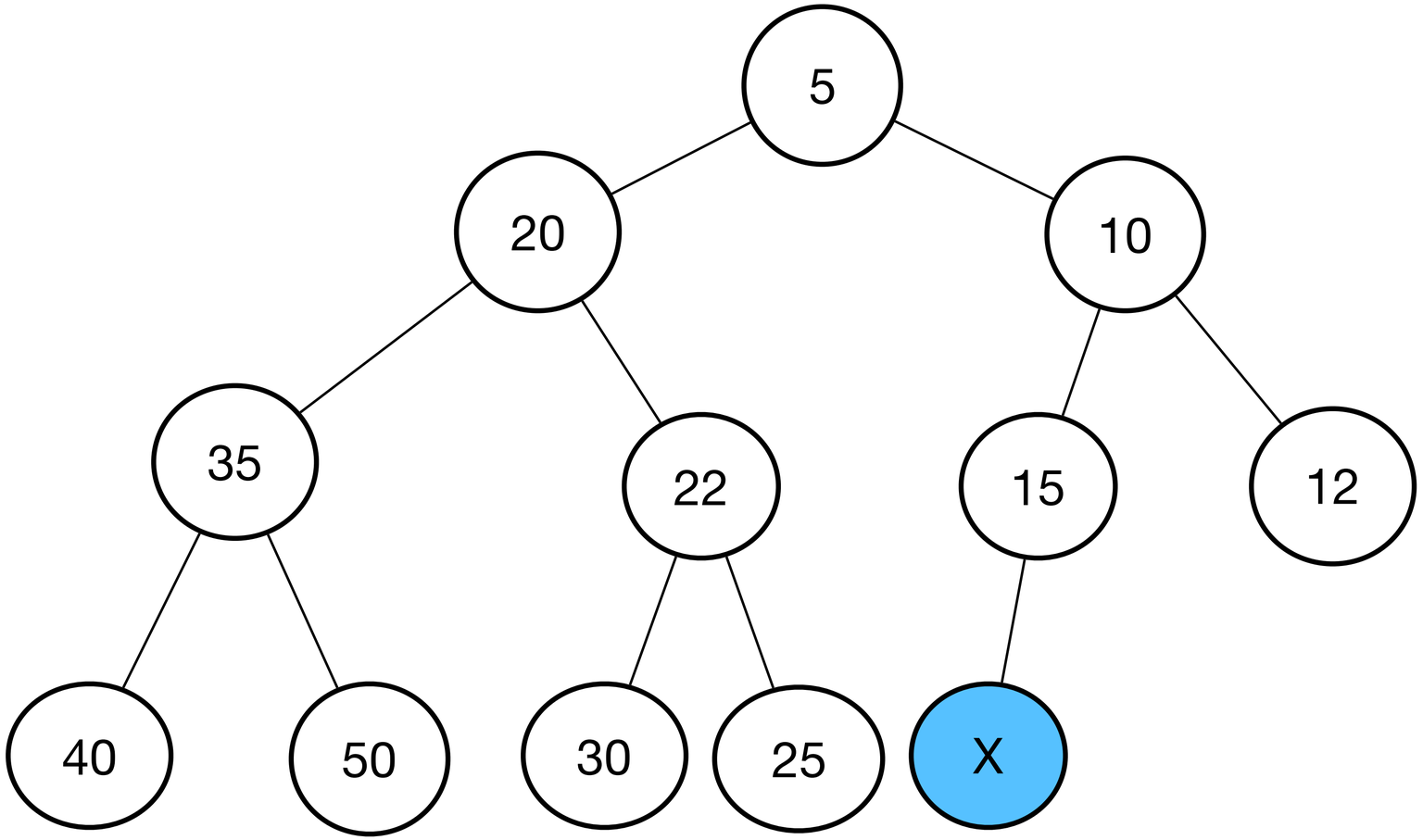


5.1.2 堆的创建与插入操作

创建：空堆

插入：插在最后！

然后呢?

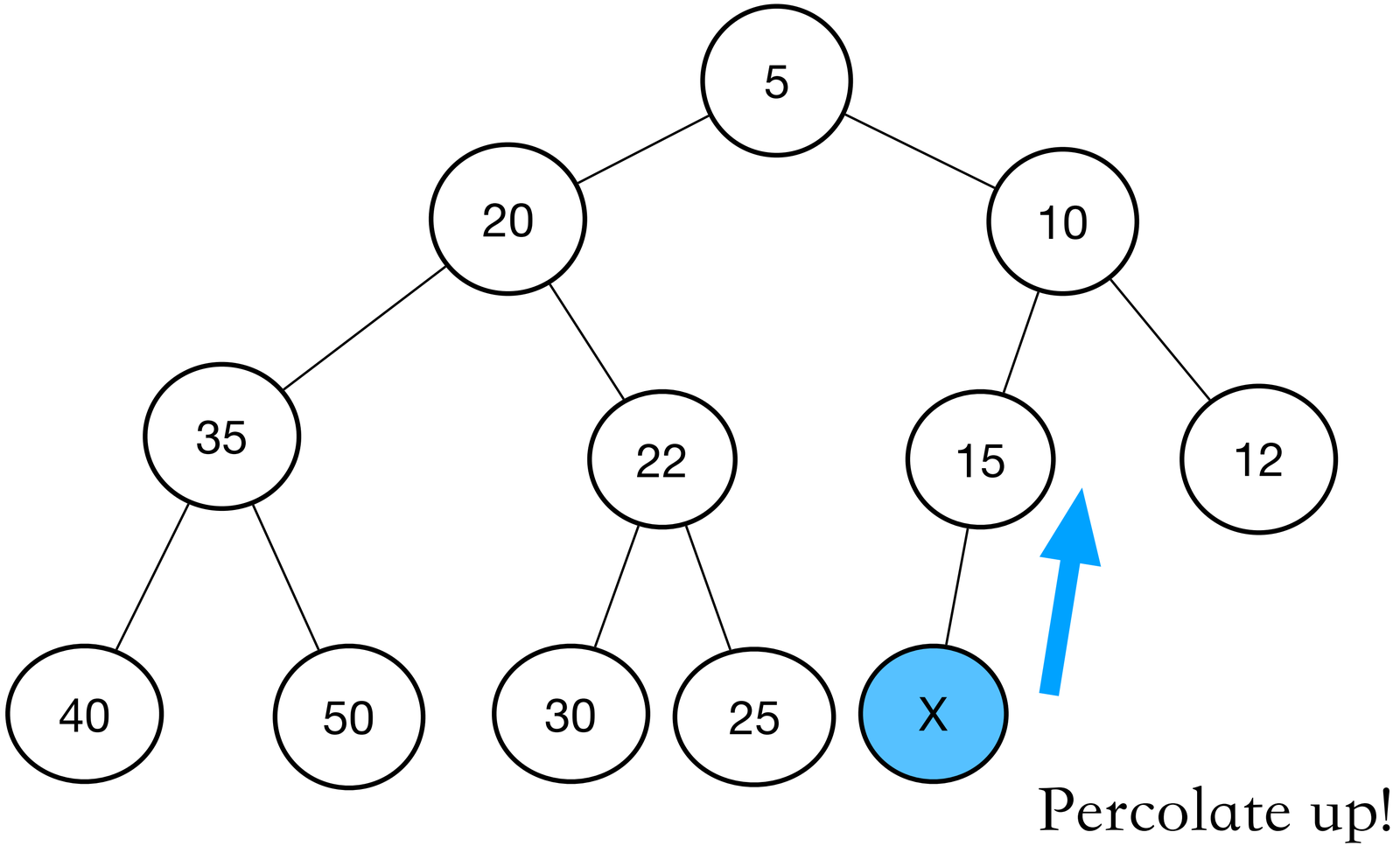


5.1.2 堆的创建与插入操作

创建：空堆

插入：插在最后！

然后呢?



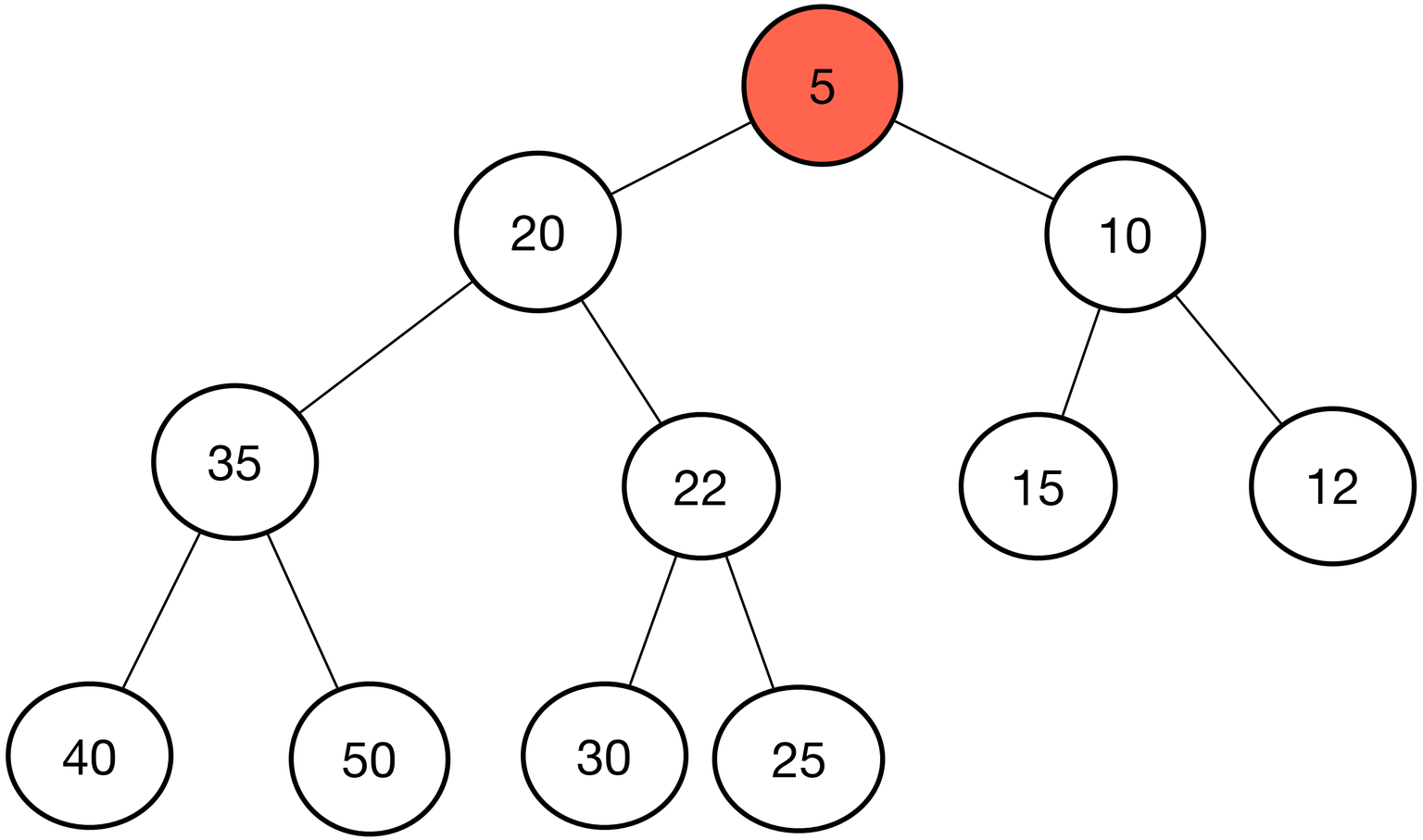
5.1.3 堆的删除操作

删除：删哪一个?

这很显然

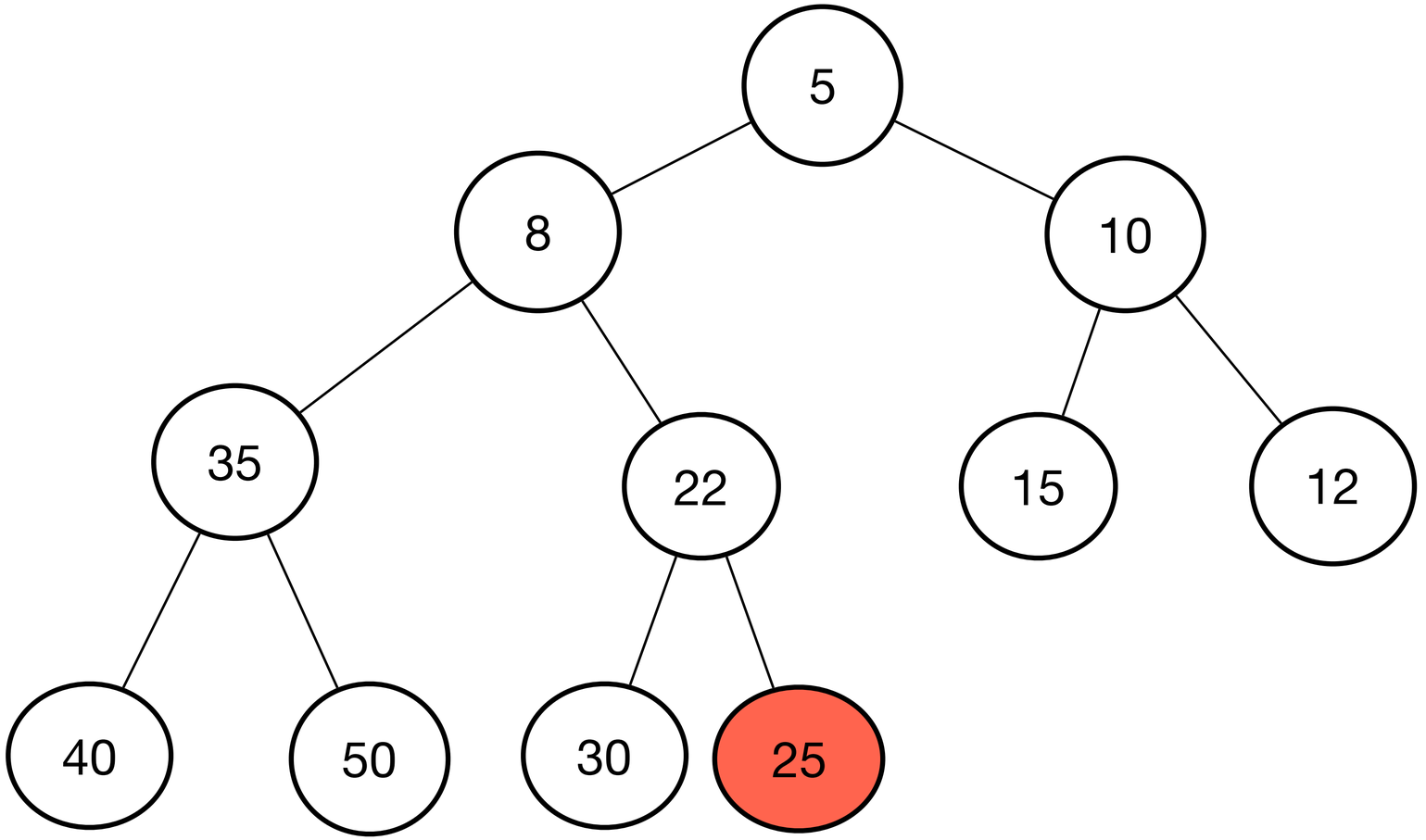
然后呢?

从它的两个儿子中选一个来补充这个位置吗?



5.1.3 堆的删除操作

假设我们“近视眼”，站得远一些看，会是另一幅情景

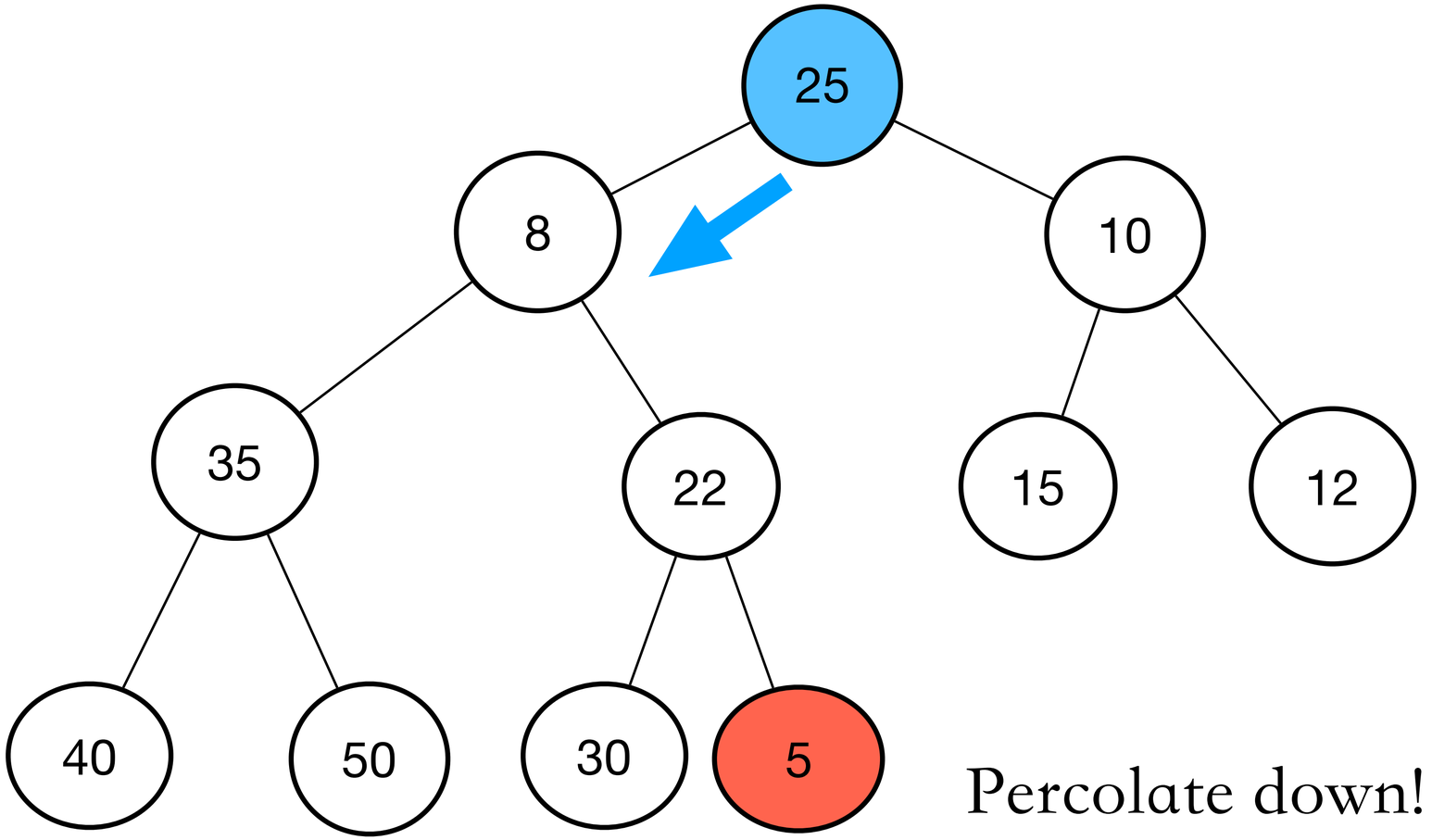


5.1.3 堆的删除操作

假设我们“近视眼”，站得远一些看，会是另一幅情景。

移动“12”的过程，没有“空腔”，所以向左、向右都没问题

(题外话：从图中，我们还注意到了什么？)



5.1.4 基于一个列表创建堆

给出列表：90 20 70 10 30 40 15 25 35 45 11 80 ，如何基于这些元素，构建一个最小堆?

回答： 简单！创建空堆，然后元素依次插入到堆中 ~

分析效率:

5.1.4 基于一个列表创建堆

给出列表：90 20 70 10 30 40 15 25 35 45 11 80 ，如何基于这些元素，构建一个最小堆?

回答： 简单！创建空堆，然后元素依次插入到堆中 ~

分析效率： $O(N \log N)$

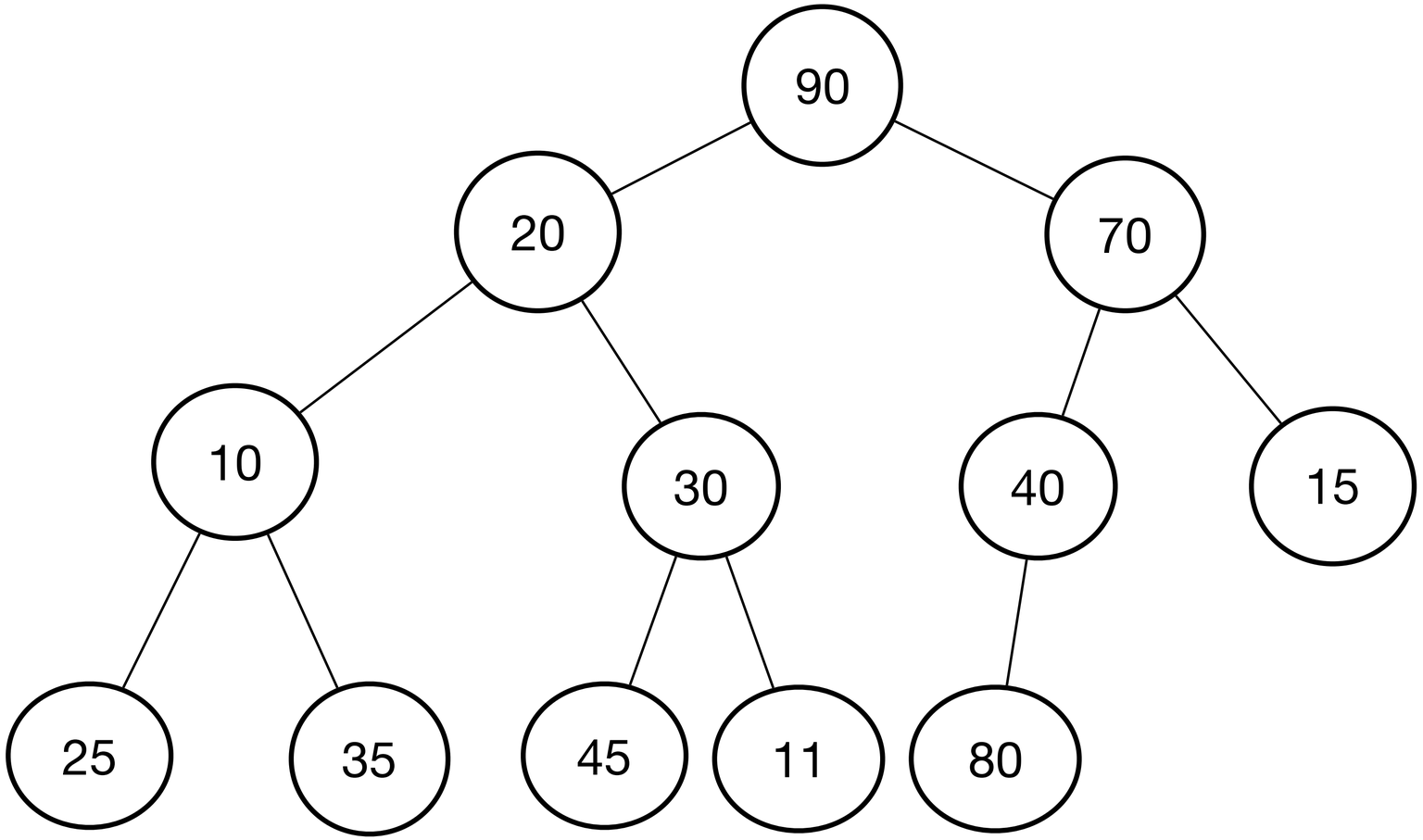
我们想怎么做? 比 $O(N \log N)$ 更快…… 那会是…… ?

5.1.4 基于一个列表创建堆

思路：

依次插入，慢在哪里？

怎么改进？



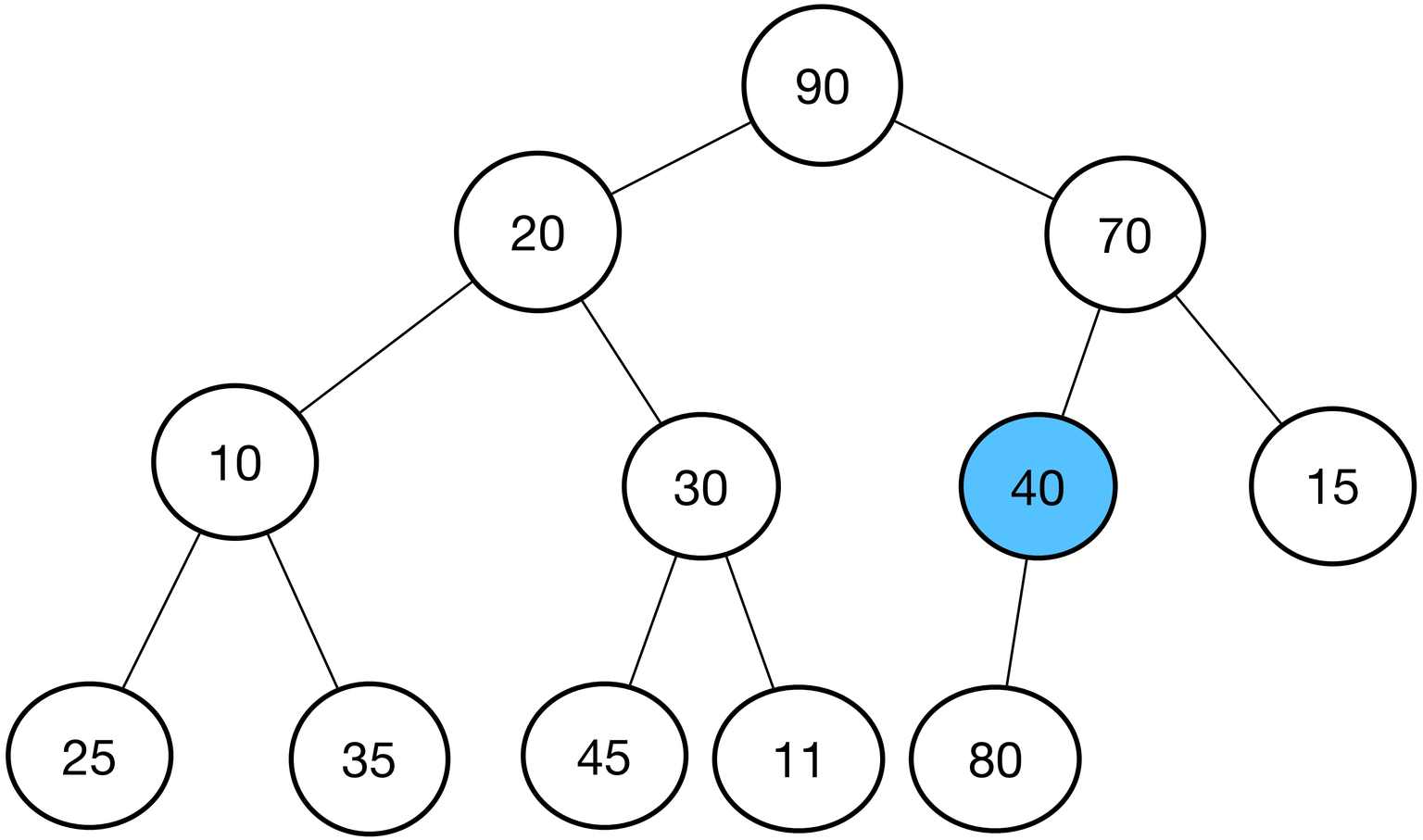
5.1.4 基于一个列表创建堆

percolateDown 6

percolateDown 5

percolateDown 4

……



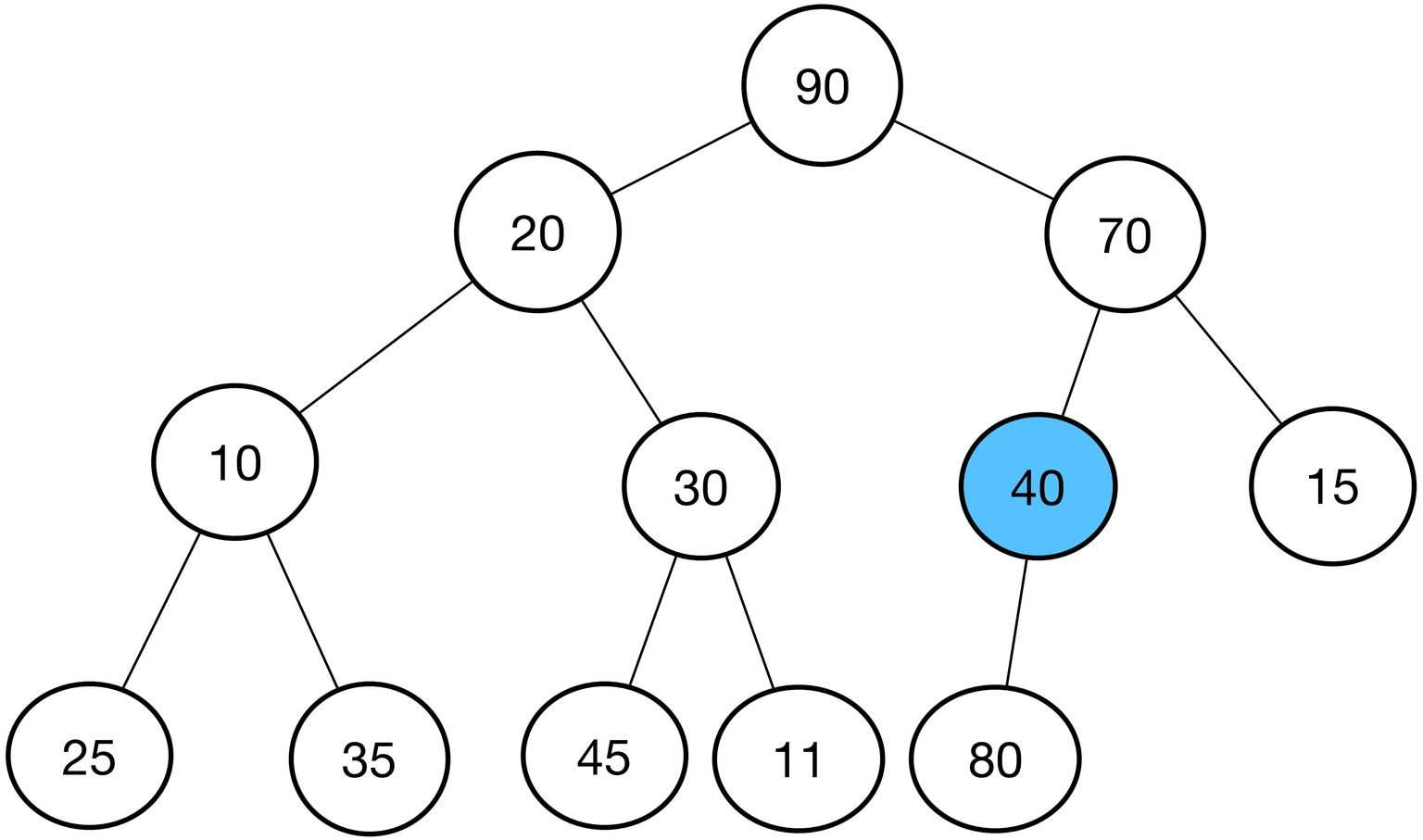
5.1.4 基于一个列表创建堆

percolateDown 6

percolateDown 5

percolateDown 4

……



$O(N)$

5.1.5 堆的其它操作

- 提高元素优先级
- 降低元素优先级
- 删除元素（不是堆顶）

（先思考：提高元素优先级，是增加还是减少元素的值？）

5.1.5 堆的其它操作

- 提高元素优先级　提高 + percolate up
- 降低元素优先级
- 删除元素（不是堆顶）

（先思考：提高元素优先级，是增加还是减少元素的值？）

5.1.5 堆的其它操作

- 提高元素优先级　提高 + percolate up
- 降低元素优先级　降低 + percolate down
- 删除元素（不是堆顶）

（先思考：提高元素优先级，是增加还是减少元素的值？）

5.1.5 堆的其它操作

- 提高元素优先级　提高 + percolate up
- 降低元素优先级　降低 + percolate down
- 删除元素（不是堆顶）　欲使其灭亡，先使其膨胀

  提高至堆顶 + 删除堆顶

（先思考：提高元素优先级，是增加还是减少元素的值？）

5.1.6 其他语言优先队列（堆）的使用

- C++
- Java
- Python

数据结构与算法实战
第五讲 优先队列与集合

5.2 集合（Set）

5.2.1 集合的概念

集合（Set）： $S = \{ x \mid P(x) \}$

- 元素无序、不重复
- 并、交、差等操作

5.2.2 集合的实现

- C： 自己手写一个
- C++： STL set （采用红黑树实现）
- Java： set
- Python： set

数据结构与算法实战
第五讲 优先队列与集合

5.3 不相交集（Disjoint Set）

5.3.1 不相交集的概念

场景：等价类

- 关系R： 对于集合S中每对元素(a,b)，a R b为true或false。

  若a R b为true，则说a与b有关系。

- 等价关系： 关系R是集合S上的等价关系，当且仅当：

  R是对称的（symmetric）、自反的（reflexive)、传递的（transitive）。

对称： a R b <==> b R a

自反：a R a

传递：a R b & b R c ==> a R c

5.3.1 不相交集的概念

场景：等价类

集合S = {a, b, c, d, e, f, g, h, i, j}，等价关系 ~ ，有：

a~b, c~d, b~f, a~j, e~h, b~i ， 则（1）集合S划为哪几个等价类?

=> {a, b, f, i, j}, {c, d}, {e, h}, {g}　是S的不相交子集！

（2）判断集合中的两个元素是否有等价关系?

=> 判断它们是否在同一个等价类

5.3.1 不相交集的概念

- 不相交集——“并查集”

与普通“集合”的不同

- 集合：本集合有哪些元素?
- 不相交集：本元素在哪个集合?

5.3.2 不相交集的操作

- 采用的数据结构

1	2	3	4	5	6	7	8	9	10
0	1	0	3	0	1	0	5	1	1
a	b	c	d	e	f	g	h	i	j

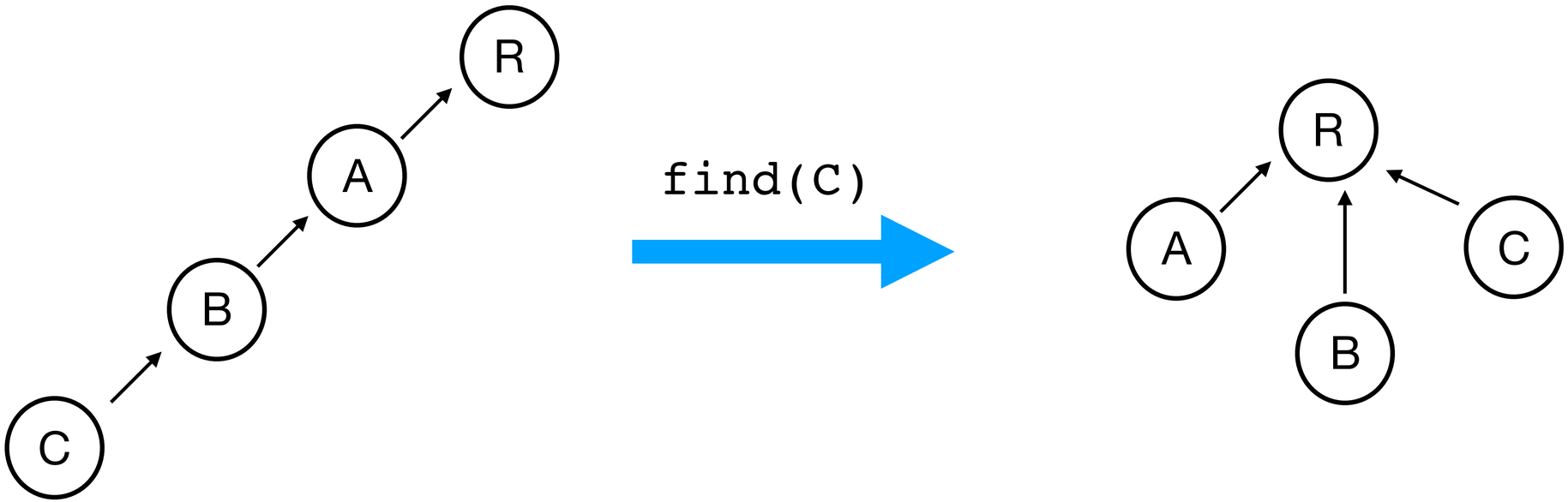
a~b, c~d, b~f, a~j, e~h, b~i

1~2, 3~4, 2~6, 1~10, 5~8, 2~9

- 查操作：Find
- 并操作：Union

5.3.2 不相交集的操作

- Union by size 按大小求并：把小树并到大树上
- Union by height(rank) 按高（秩）求并：把矮树并到高树上
- Path compression 路径压缩



5.3.3 实例：推断学生所属学校

- 既需要知道 集合里有哪些元素
- 也需要知道 某元素在哪个集合
- 如何实现?

举个 

5.3.3 实例：推断学生所属学校

某个比赛现场有来自不同学校的N名学生，给出M对“两人同属一所学校”的关系，请推断学校数量，并给出人数最多的学校的学生名单。

输入格式：

先输入一个在[2, 1000]范围的整数N，然后是N个用空格间隔姓名。接下来一行是正整数M，然后是M行，每行两个人名，表示同属一所学校。

输出格式：

先输出学校的数量，在下一行输出人数最多的学校学生名单。

5.3.3 实例：推断学生所属学校

输入样例：

8

Bill Ellen Ann Chris Daisy Flin Henry Grace

5

Ann Chris

Ellen Chris

Daisy Flin

Henry Ellen

Grace Flin

输出样例：

4

Ann Chris Ellen Henry