8.高级搜索树

（a1）伸展树：逐层伸展

我要一步一步往上爬

在最高点乘着叶片往前飞

邓俊辉

deng@tsinghua.edu.cn

局部性

- Locality：刚被访问过的数据，极有可能很快地再次被访问

  这一现象在信息处理过程中屡见不鲜 //BST就是这样的一个例子

- BST：刚刚被访问过的节点，极有可能很快地再次被访问

  下一将要访问的节点，极有可能就在刚被访问过节点的附近

- 连续的m次查找（m >> n = |BST|），采用AVL共需$O(m \log n)$时间

- 利用局部性，能否更快？ //仿效自适应链表

- 策略：节点一旦被访问，随即调整至树根 //如此，下次访问即可...

- 问题：如何实现这种调整？调整过程自身的复杂度如何控制？

逐层伸展

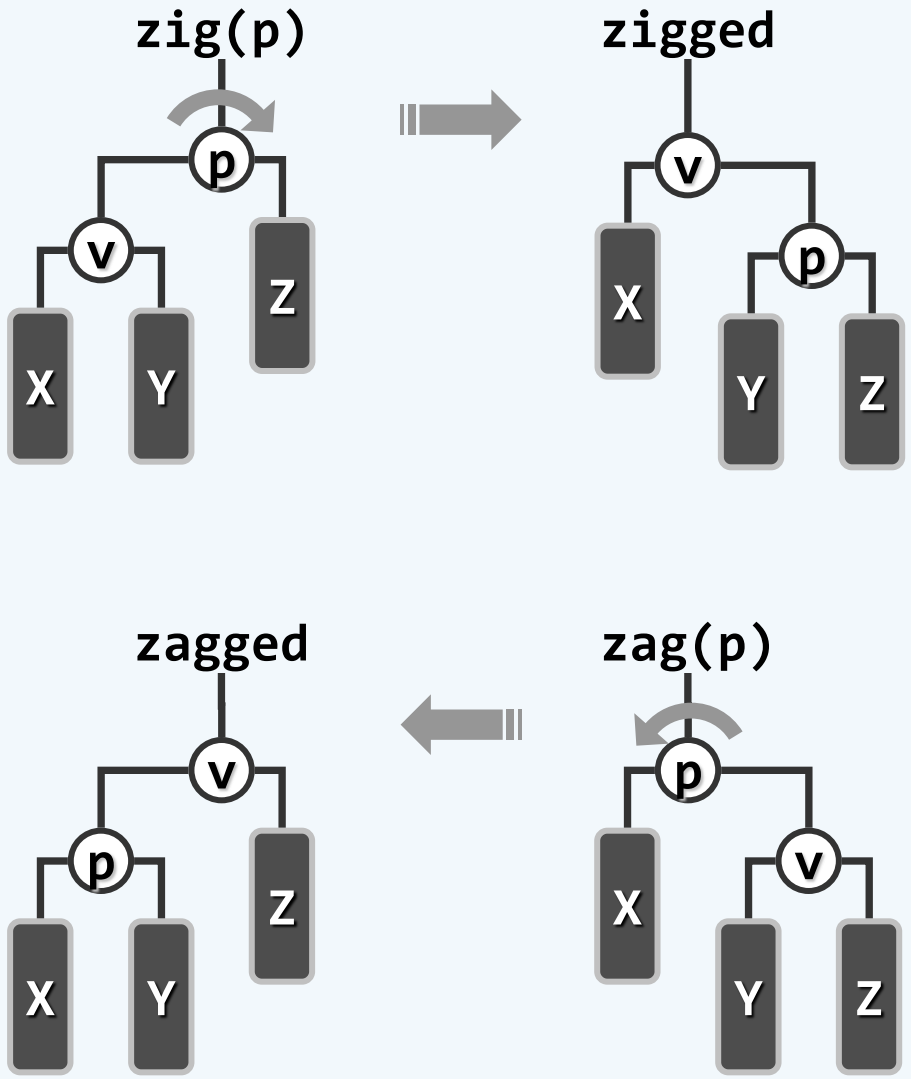
- 节点v一旦被访问，随即转移至树根

- 一步一步往上爬

  自下而上，逐层单旋

  ```
  zig( v->parent )
  zag( v->parent )
  ```

  直到v最终被推送至根



实例

- 伸展过程的**效率**

  是否足够地高？

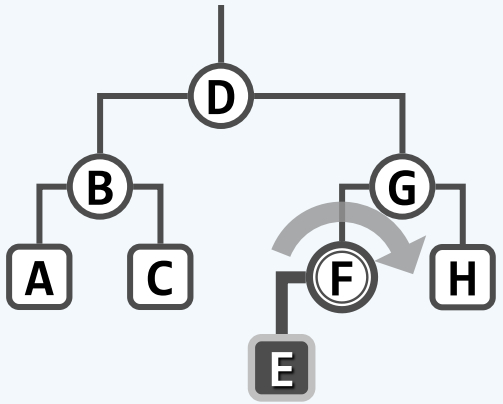
- 就逐层伸展的策略而言

  这取决于

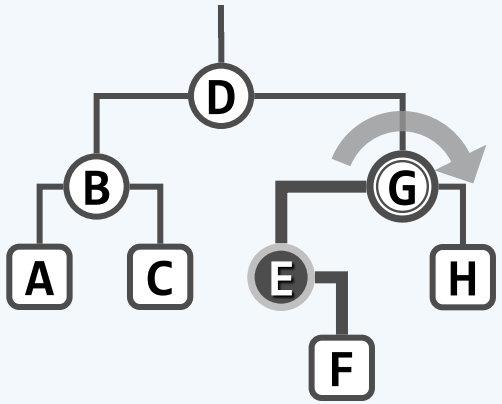
  树的**初始形态**和

  节点的**访问次序**

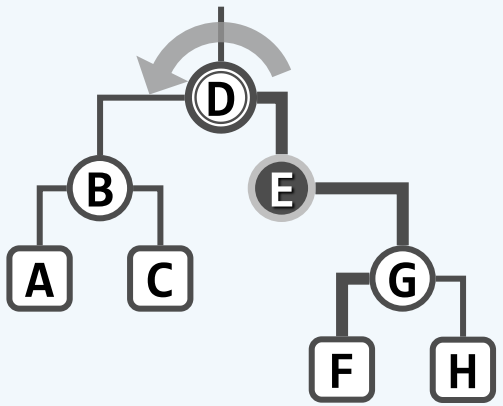
访问E之后，做zig(F)
(a)



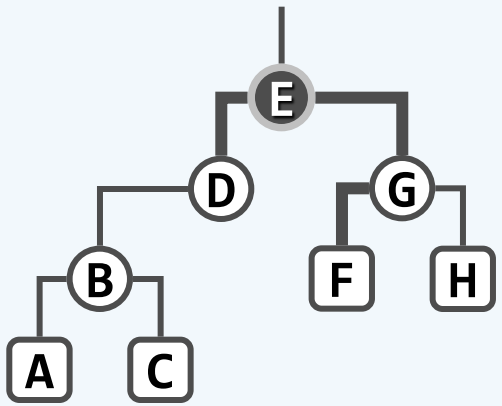
zig(G)
(b)



zag(D)
(c)

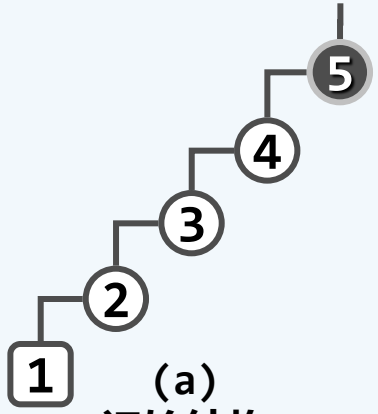


等价变换：经3次旋转，E调整至树根
(d)

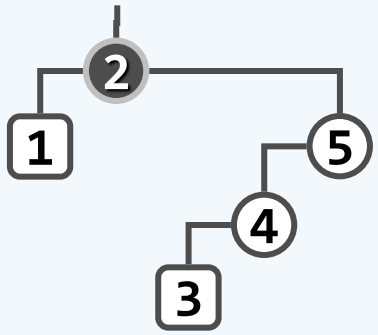


最坏情况

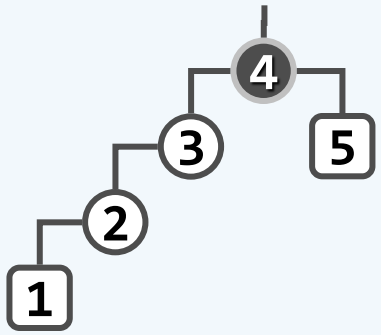
❖ 旋转次数呈周期性的算术级数演变：每一周期累计$\Omega(n^2)$，分摊$\Omega(n)$！



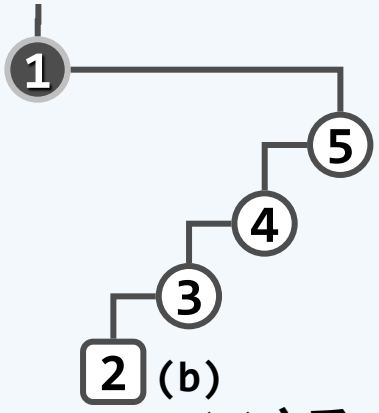
(a)
初始结构



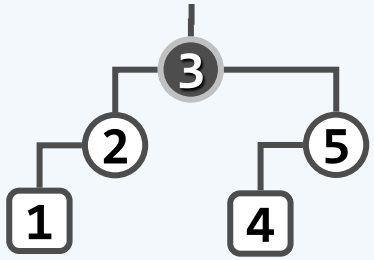
(c)
search(2)之后



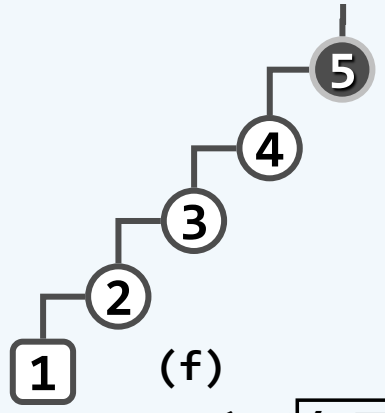
(e)
search(4)之后



(b)
search(1)之后



(d)
search(3)之后



(f)
search(5)之后复原

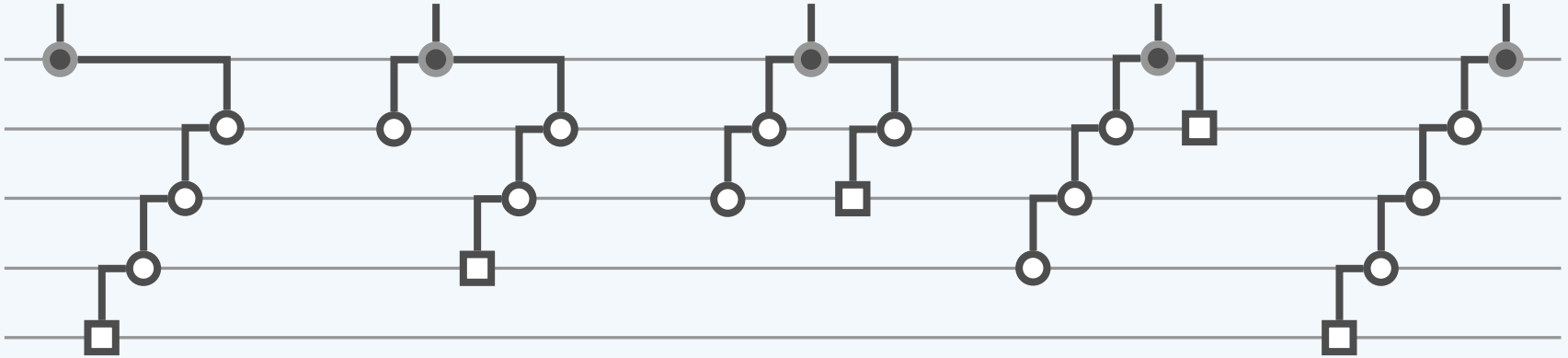
低效率的根源

❖ 最坏情况，问题出在哪里？

1)全树拓扑始终呈单链条结构，等价于一维列表

2)被访问节点的深度，呈周期性的算术级数演变，平均为Ω(n)：

```
n - 1, n - 2, n - 3, ..., 3, 2, 1; n - 1, ...
```



❖ 问题的症结既已确定，便可针对性地改进...