

05-B 树的表示

#数据结构邓神

表示法

接口

节点	功能
root()	根节点
parent()	父节点
firstChild()	长子
nextSibling()	兄弟
insert(i, e)	将e作为第i个孩子插入
remove(i)	删除第i个孩子（及其后代）
traverse()	遍历

```
template <typename T> class TreeNode{
};
template <typename T> class Tree{
    typedef TreeNode<T> node;
    typedef node* nodePtr;
public:
    nodePtr root();
    nodePtr parent();
    nodePtr nextSibling();
    nodePtr insert(int i,T const & e);
    bool remove(int i);
    void traverse();
};
```

以父节点构建树

任何一个节点（除了根节点）都有且仅有一个根节点

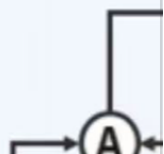
父节点

❖ 观察：除根外，任一节点有且仅有一个父节点

❖ 构思：将节点组织为序列，各节点分别记录

data 本身信息

parent 父节点的秩或位置



父节点

❖ 空间性能： $O(n)$ ✓

❖ 时间性能



parent()： $O(1)$ ✓



root()： $O(n)$ 或 $O(1)$



firstChild()： $O(n)$

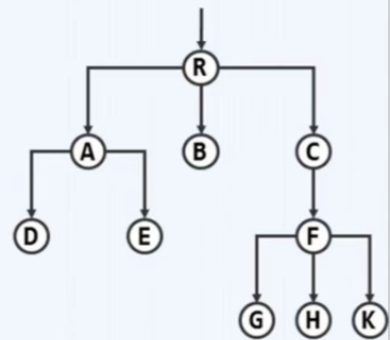


nextSibling()： $O(n)$

以孩子节点构建树

孩子节点

	data	children
0	A	→ 3 → 5 → ^
1	B	^
2	C	→ 6 → ^
3	D	^
4	R	→ 0 → 1 → 2 → ^
5	E	^
6	F	→ 7 → 8 → 9 → ^
7	G	^
8	H	^
9	K	^



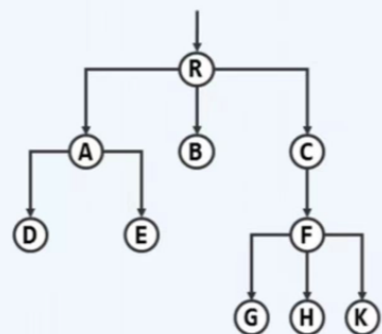
我们发现这个数据结构在查看孩子都十分方便

但是在找父亲的时候又变为了 $O(n)$

父亲节点 + 孩子节点

父节点 + 孩子节点

	data	parent	children
0	A	4	→ 3 → 5 → ^
1	B	4	^
2	C	4	→ 6 → ^
3	D	0	^
4	R	-1	→ 0 → 1 → 2 → ^
5	E	0	^
6	F	2	→ 7 → 8 → 9 → ^
7	G	6	^
8	H	6	^
9	K	6	^



无论是找父亲还是找孩子都很方便

每一个节点的child引用指向的数据集大小有很大差异

很多时候会讲问题变的十分复杂

长子 + 兄弟

长子 + 兄弟

✧ 每个节点均设两个引用

纵：firstChild()

横：nextSibling()

firstChild 指向 长子

nextSibling 指向下一个次子

