

Data Structure

Xia Tian

Email: [xiat\(at\)ruc.edu.cn](mailto:xiat(at)ruc.edu.cn)

Renmin University of China

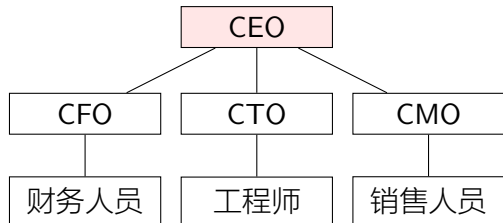


树和二叉树

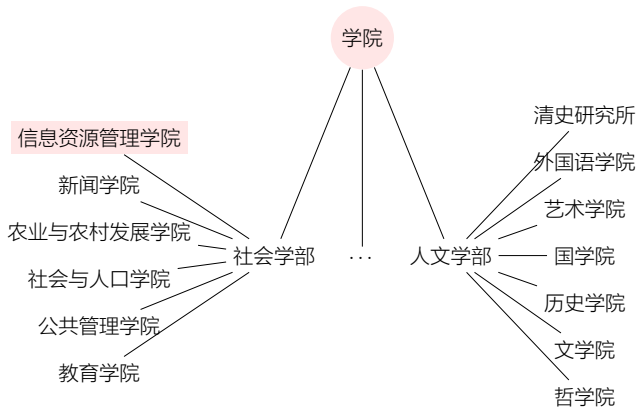
树型结构是结点之间有分支, 并且具有层次关系的结构, 类似于自然界中的树。树有很多应用, 比如 Unix 等操作系统中的目录结构。

```
→ github tree course_ds
course_ds
├── clean.py
├── dot
│   ├── tree-judge1.dot
│   ├── tree-judge1.pdf
│   ├── tree-judge2.dot
│   ├── tree-judge2.pdf
│   ├── tree-judge3.dot
│   ├── tree-judge3.pdf
│   ├── tree-judge4.dot
│   ├── tree-judge4.pdf
│   ├── tree-judge.pdf
│   ├── tree-represent1.dot
│   ├── tree-represent1.pdf
│   └── tree-term-demo.dot
├── ds.pdf
├── ds.tex
├── figs
│   └── search-block.tex
├── graph.tex
├── imgs
│   └── merge-sort.jpg
├── introduction.tex
├── LICENSE
├── _minted-ds
├── README.md
├── ruc_logo.png
└── search.tex
```

例子



Simple Company Hierarchy



人民大学学院设置

内容



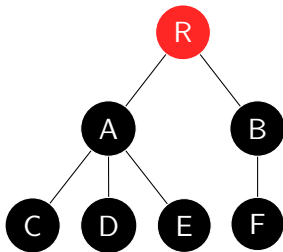
- 树的基本术语
- 二叉树
- 遍历二叉树与线索二叉树
- 树和森林
- 哈夫曼树



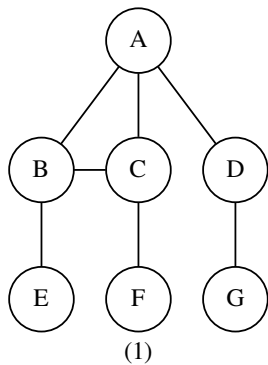
树 (TREE)

树 (Tree) 是 $n(n \geq 0)$ 个结点的有限集 T 。 T 为空时称为空树。当 $n > 0$ 时, 树有且仅有一个特定的称为根 (Root) 的结点, 其余结点可分为 $m(m \geq 0)$ 个互不相交的子集 T_1, T_2, \dots, T_m , 其中每个子集又是一棵树, 称为子树 (Subtree)。

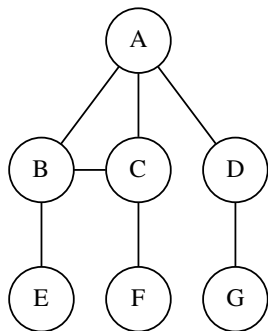
1. 各子树是互不相交的集合。
2. 除根结点, 其它结点有唯一前驱。
3. 一个结点可以有零个或多个后继。



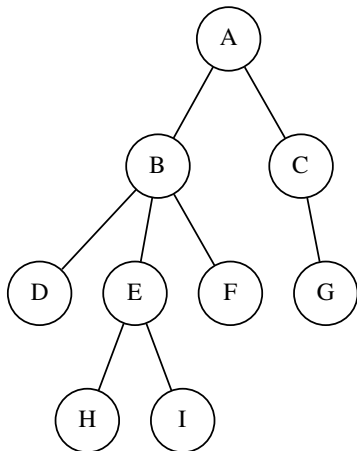
判断哪些是树结构



判断哪些是树结构

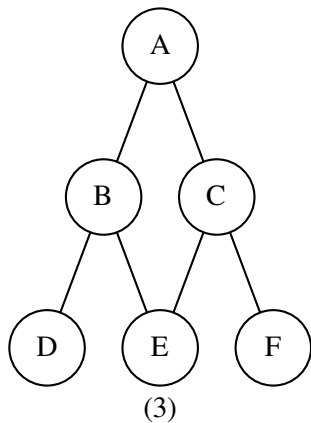


(1)

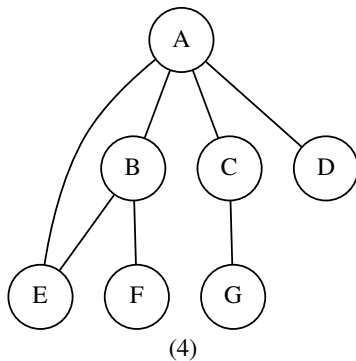
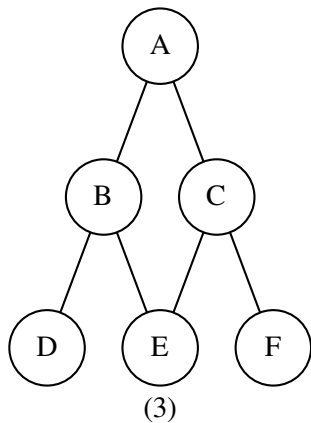


(2)

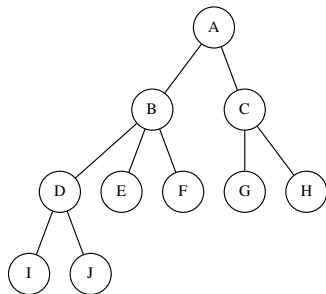
判断哪些是树结构



判断哪些是树结构

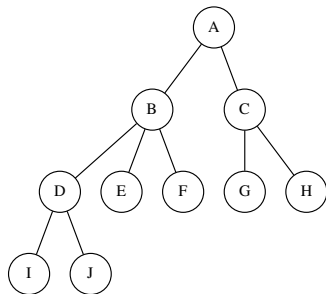


树的表示形式

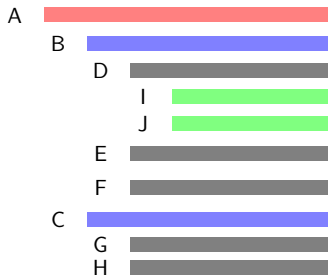


树形表示

树的表示形式



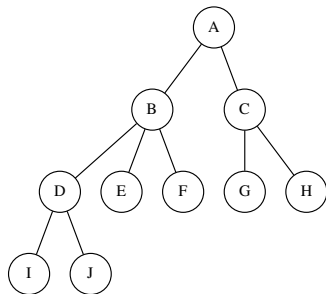
树形表示



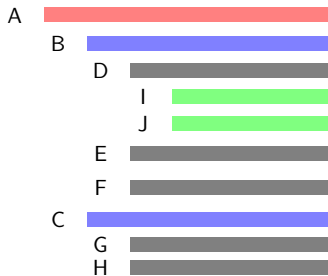
凹入表表示法



树的表示形式



树形表示

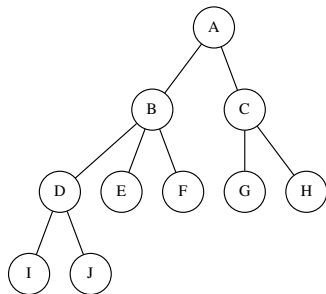


凹入表表示法

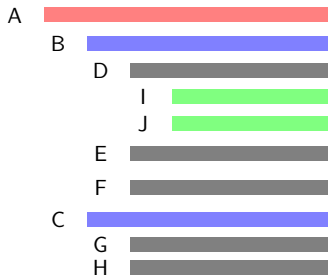
(A(B(D(I,J),E, F),C(G,H)))

广义表表示

树的表示形式



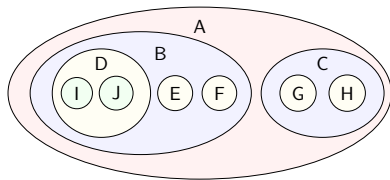
树形表示



凹入表表示法

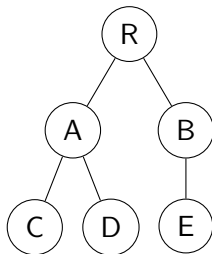
$(A(B(D(I,J),E,F),C(G,H)))$

广义表表示



嵌套集合表示

- 树 (tree)
- 子树 (sub-tree)
- 结点 (node)
- 结点的度 (degree)
- 叶子 (leaf)
- 孩子 (child)
- 父亲 (parents)
- 兄弟 (sibling)
- 祖先
- 子孙
- 树的度 (degree)
- 结点的层次 (level)
- 树的深度 (depth)
- 有序树
- 无序树
- 森林



二叉树 (Binary Tree)

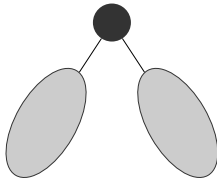
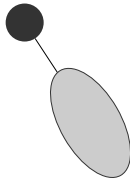
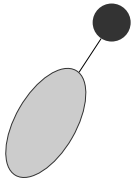


- 二叉树是一种树型结构, 它的每个结点至多只有两个子树, 分别称为左子树和右子树。二叉树是有序树。
- 二叉树是 $n(n \geq 0)$ 个结点构成的有限集合。二叉树或为空, 或是由一个根结点及两棵互不相交的左右子树组成, 并且左右子树都是二叉树。
- 在二叉树中要区分左子树和右子树, 即使只有一棵子树。这是二叉树与树的最主要的差别。

二叉树的一个重要应用是在查找中的应用。当然, 它还有许多与搜索无关的重要应用, 比如在编译器的设计领域。

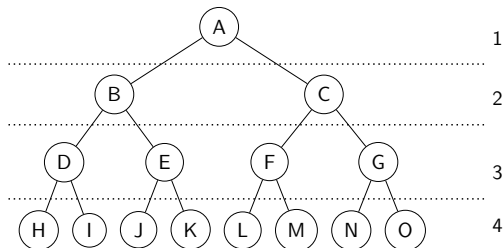
二叉树的五种形态

1. 空二叉树;
2. 只有根结点 (左右子树都为空);
3. 只有左子树 (右子树为空);
4. 只有右子树 (左子树为空);
5. 左右子树均不空。





请观察二叉树, 并回答下列问题



1. 二叉树的第 i 层最多有多少个结点?
2. 二叉树深度为 k , 则它最多有多少个结点?
3. 二叉树有 n 个节点, 请问它最小深度是几?
4. 二叉树叶子的数目和度为 2 的节点的数目是否相等? 如果不等, 又是什么关系?

二叉树的性质



- 性质 1: 二叉树的第 i 层至多有 2^{i-1} 个结点。
- 性质 2: 深度为 k 的二叉树至多有 $2^k - 1$ 个结点 ($k \geq 1$)。
- 性质 3: 二叉树中终端结点数为 n_0 , 度为 2 的结点数为 n_2 , 则有 $n_0 = n_2 + 1$ (试证明)

二叉树的性质



二叉树中终端结点数为 n_0 , 度为 2 的结点数为 n_2 , 则有 $n_0 = n_2 + 1$

- 设二叉树中度为 1 的结点数为 n_1 , 二叉树中总结点数为 N , 则有:

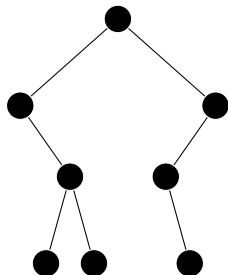
$$N = n_0 + n_1 + n_2$$

- 再考虑二叉树中的分支数 (每个节点有唯一一个入的分支, 根节点除外; 再考虑出的分支数量), 则有:

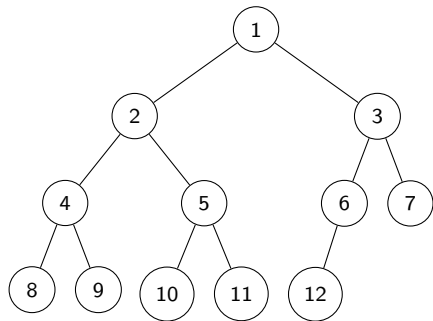
$$N - 1 = n_1 + 2 \times n_2$$

- 整理可得:

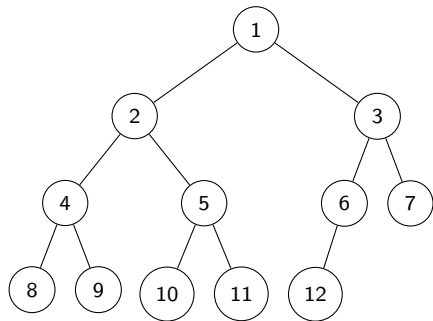
$$n_0 = n_2 + 1$$



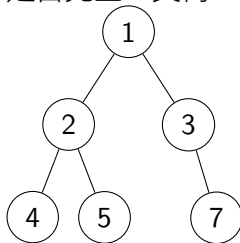
完全二叉树



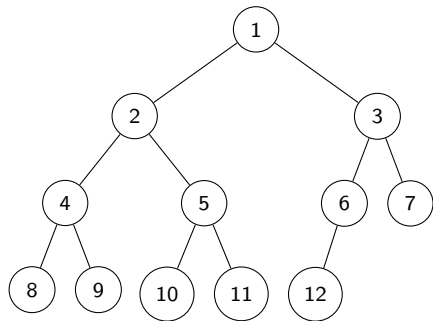
完全二叉树



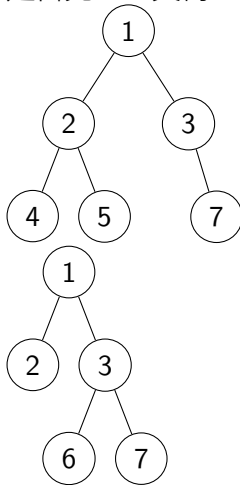
是否完全二叉树?



完全二叉树



是否完全二叉树?



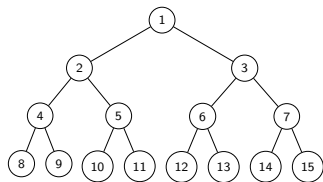
二叉树的性质



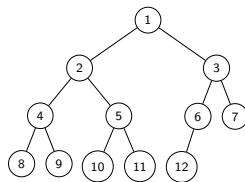
- 性质 4: 具有 n 个结点的完全二叉树的深度为:

$$\lfloor \log_2 n \rfloor + 1$$

对于完全二叉树, 设深度为 k , 由 $2^{k-1} - 1 < n \leq 2^k - 1$ 可知, $2^k - 1 \leq n < 2^k$, 则 $k - 1 \leq \log_2 n < k$ (参考性质 2: $2^{k-1} - 1 < n$: 比大于深度为 $k - 1$ 的满二叉树的数)



满二叉树



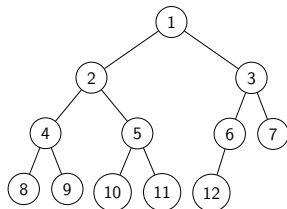
完全二叉树

测试



对于完全二叉树:

1. 若完全二叉树有叶子结点出现在第 k 层, 它可能还有 () 层的叶子结点;
2. 若某结点的右子树的最大层次为 L , 则其左子树的最大层次为 ();
3. 若按如图所示的编号方式, 试求出编号为 i 的节点的父节点和子节点的编号.

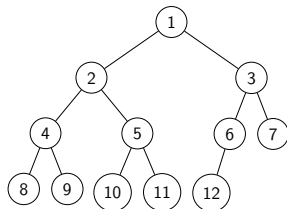


测试

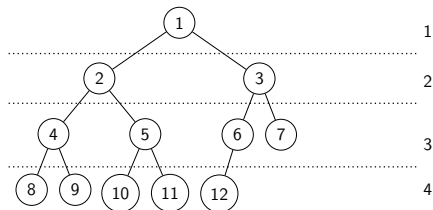


对于完全二叉树:

1. 若完全二叉树有叶子结点出现在第 k 层, 它可能还有 () 层的叶子结点;
2. 若某结点的右子树的最大层次为 L , 则其左子树的最大层次为 ();
3. 若按如图所示的编号方式, 试求出编号为 i 的节点的父节点和子节点的编号.



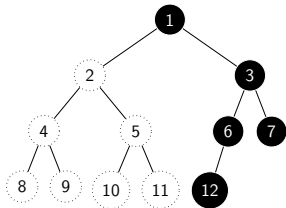
1. $k - 1, k + 1$
2. L or $L + 1$



- 性质 5: 对具有 n 个结点的完全二叉树的结点按层次顺序编号, 对任意结点 i 有:
 - ▶ (有关结点 i 的双亲) 若 $i = 1$, 则为二叉树的根结点, 没有双亲; 否则双亲结点的编号为: $\left\lfloor \frac{i}{2} \right\rfloor$
 - ▶ (有关结点 i 的孩子) 若 $n < 2 \cdot i$, 则结点 i 无左孩子; 否则左孩子编号是 $2 \cdot i$ 。
 - ▶ 若 $n < 2 \cdot i + 1$, 则结点 i 无右孩子; 否则右孩子编号为 $2 \cdot i + 1$



二叉树的存储结构：顺序存储



```
#define MAX_SIZE 100
```

```
typedef int SqBiTree[MAX_SIZE];
```

```
SqBiTree bt;
```

```
class SqBiTree {
```

```
    //static int MAX_SIZE = 100;
```

```
    //int[] data = new int[MAX_SIZE];
```

```
    List<Integer> data = new ArrayList<Integer>();
```

```
}
```

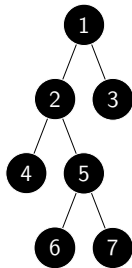
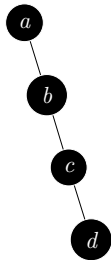


- 把结点安排成一个恰当的序列 (编号), 存储在数组中
- 便于“随机存取”



二叉树的存储结构：顺序存储

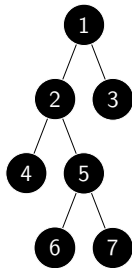
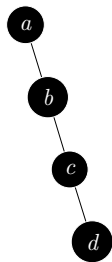
- 适用于完全二叉树





二叉树的存储结构：顺序存储

- 适用于完全二叉树

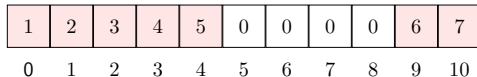
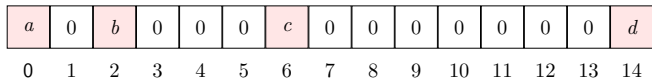
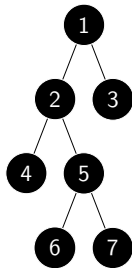
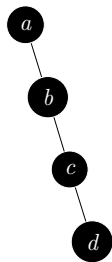


a	0	b	0	0	0	c	0	0	0	0	0	0	0	d
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14



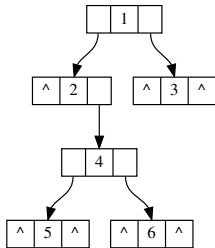
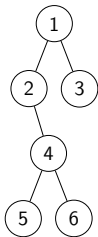
二叉树的存储结构：顺序存储

- 适用于完全二叉树





二叉树的链式存储: 二叉链表



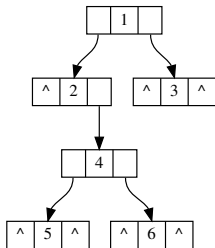
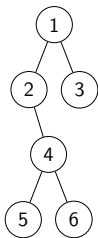
```
typedef struct BiTNode {// C Code  
    ElemType data;  
    struct BiTNode *lchild, *rchild;  
} BiTNode, *BiTree;
```

```
class BiTNode<T> {//Java Code  
    T data;  
    BiTNode lchild, rchild;  
}
```

思考: 含 n 个结点的二叉链表中有多少个空指针?



二叉树的链式存储: 二叉链表



```
typedef struct BiTNode { // C Code  
    ElemType data;  
    struct BiTNode *lchild, *rchild;  
} BiTNode, *BiTree;
```

```
class BiTNode<T> { //Java Code  
    T data;  
    BiTNode lchild, rchild;  
}
```

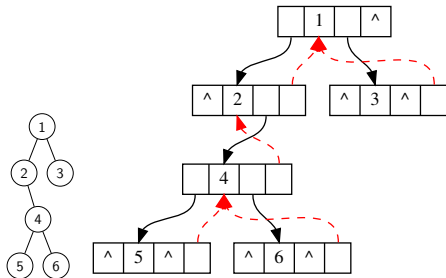
思考: 含 n 个结点的二叉链表中有多少个空指针?

指针域一共有 $2 * n$ 个, 分支共有 $N - 1$, 每个分支占用一个指针域, 所以空指针数量为: $2 * n - (n - 1) = n + 1$



二叉树的链式存储: 三叉链表

- 二叉链表不便查找父节点, 可加一个指向双亲的指针



```
typedef struct BiTNode {// C Code  
    ElemType data;  
    struct BiTNode *lchild, *rchild, *parent;  
} BiTNode, *BiTree;
```

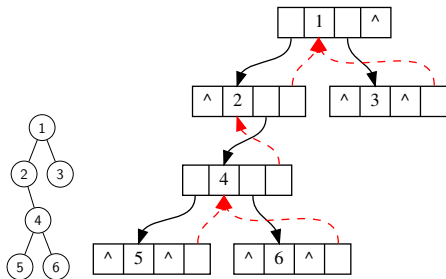
```
class BiTNode<T> {//Java Code  
    T data;  
    BiTNode lchild, rchild, parent;  
}
```

思考: 有 n 个结点的三叉链表有多少个空指针?



二叉树的链式存储: 三叉链表

- 二叉链表不便查找父节点, 可加一个指向双亲的指针



```
typedef struct BiTNode {// C Code  
    ElemType data;  
    struct BiTNode *lchild, *rchild, *parent;  
}BiTNode, *BiTree;
```

```
class BiTNode<T> {//Java Code  
    T data;  
    BiTNode lchild, rchild, parent;  
}
```

思考: 有 n 个结点的三叉链表有多少个空指针?

对于二叉链表, 指针域一共有 $2 * n$ 个, 分支共有 $N - 1$, 每个分支占用一个指针域, 所以空指针数量为 $2 * n - (n - 1) = n + 1$; 对于 **parent**, 根节点无父节点, 所以共有 $n + 1 + 1 = n + 2$

遍历二叉树和线索二叉树



在二叉树的一些应用中, 常常要求在树中查找具有某种特征的结点, 或者对树中全部结点逐一进行某种处理。这就引入了遍历二叉树的问题, 即如何按某条搜索路径巡访树中的每一个结点, 使得每一个结点均被访问一次且仅访问一次。



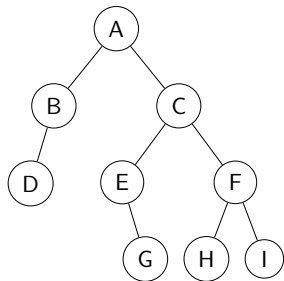
- 遍历是指按某种方式访问所有结点, 使每个结点被访问一次且只被访问一次。
- 二叉树的遍历是按一定规则将二叉树的结点排成一个线性序列, 即非线性序列线性化。
- 遍历的方式: 深度优先和广度优先, 深度优先又分为三种:
 - ▶ 先序次序
 - ▶ 中序次序 (对称序次序)
 - ▶ 后序次序

二叉树的遍历



- 1、先序遍历二叉树的操作定义为: 若二叉树为空, 则空操作; 否则
 - * (1) 访问根结点;
 - * (2) 先序遍历左子树;
 - * (3) 先序遍历右子树。
- 2、中序遍历二叉树的操作定义为: 若二叉树为空, 则空操作; 否则
 - * (1) 中序遍历左子树;
 - * (2) 访问根结点;
 - * (3) 中序遍历右子树。
- 3、后序遍历二叉树的操作定义为: 若二叉树为空, 则空操作; 否则
 - * (1) 后序遍历左子树;
 - * (2) 后序遍历右子树;
 - * (3) 访问根结点。

示例



- 先序: ABDCEGFHI
- 中序: DBAEGCHFI
- 后序: DBGEHIFCA
- 广度优先: ABCDEFGHI