计算思维通识教育

Computational Thinking

第3章 数据间的逻辑关系

主讲人：曹轶臻

联系方式：caoyizhen@cuc.edu.cn

计算思维是在时间和空间之间、在处理能力和存储容量之间进行折中的思维方法

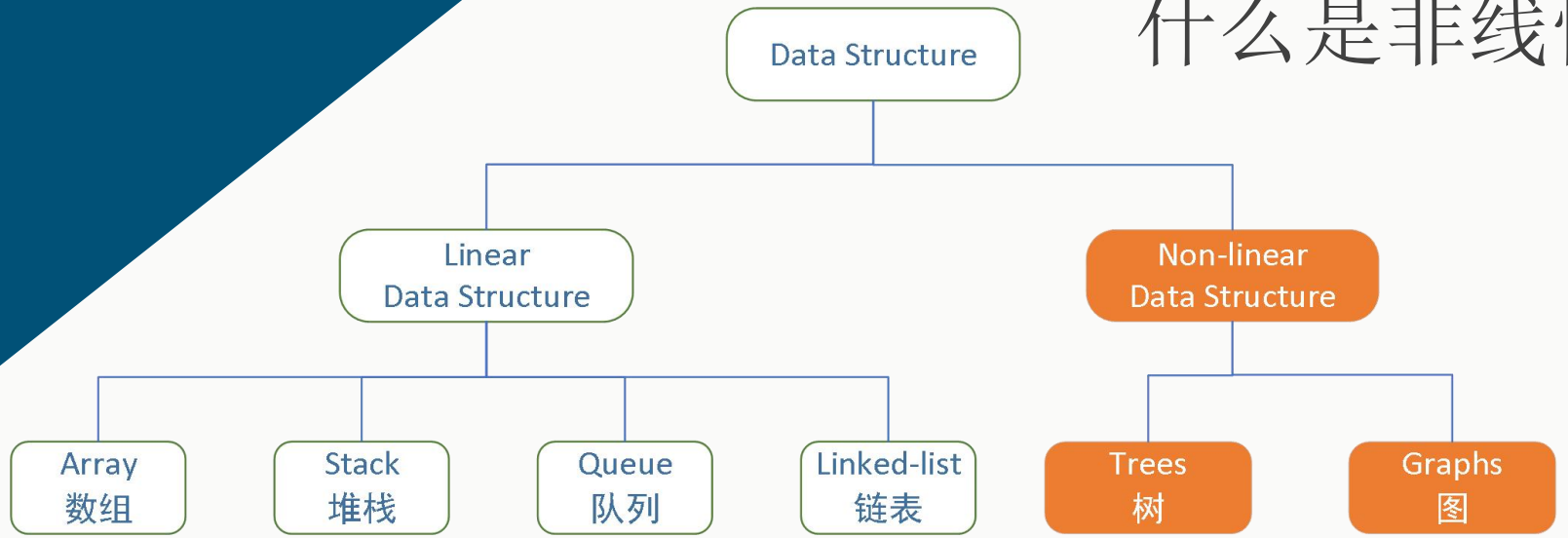
CONTENTS

计算机与网络空间安全学院
School of Computer and Cyber Sciences

计算思维通识教育
Computational Thinking

03

非线性数据结构



什么是非线性数据结构 3-1

树 3-2

图 3-3

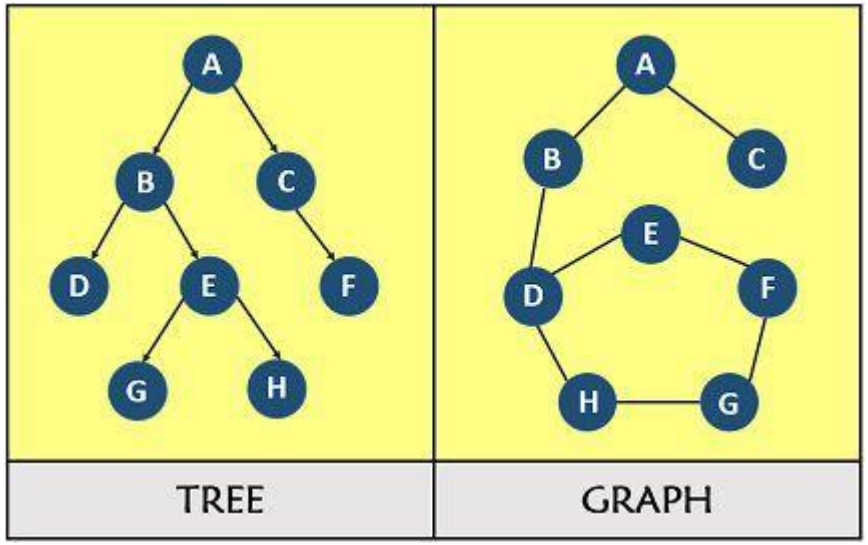
3-1 什么是非线性数据结构

按照数据间的逻辑关系，数据结构可分为

- 线性数据结构，是一种数据项按顺序或线性排序的结构，并且每个成员都附加到其前一个和下一个相邻元素。
- 非线性数据结构，是一种数据项不是按顺序排列的，每个成员可以通过多条路径连接到其他成员的数据结构。

因此无法在单层迭代（例如 for item in queue1）中遍历或检索它们。用户需要多次运行才能完全遍历它们。

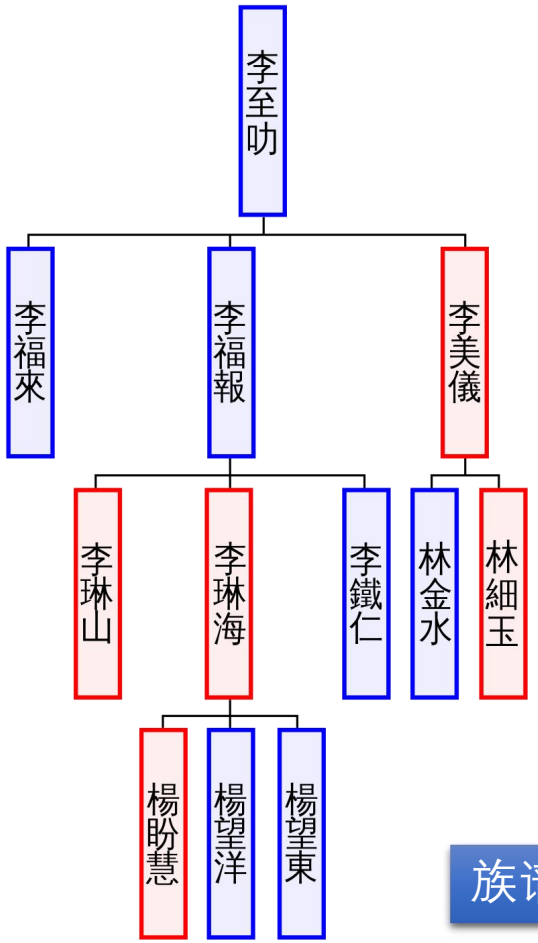
换句话说，非线性数据结构的一个数据元素可以连接到多个元素，以反映它们之间的特殊关系。



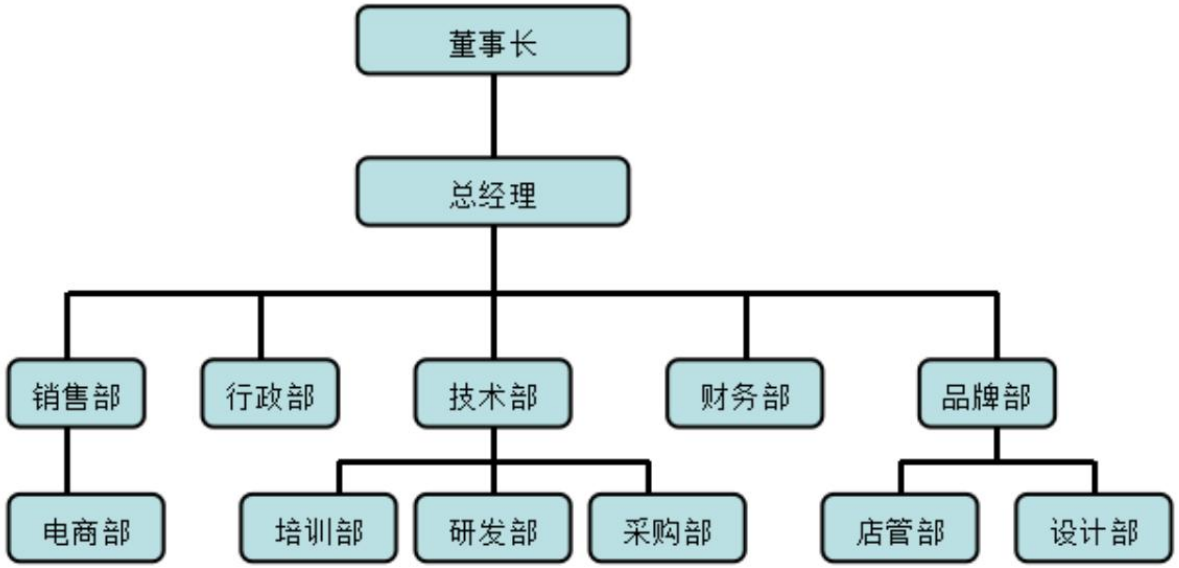
典型的非线性数据结构是树和图，它们的数据元素按层次相互连接。

3-2 树 Tree

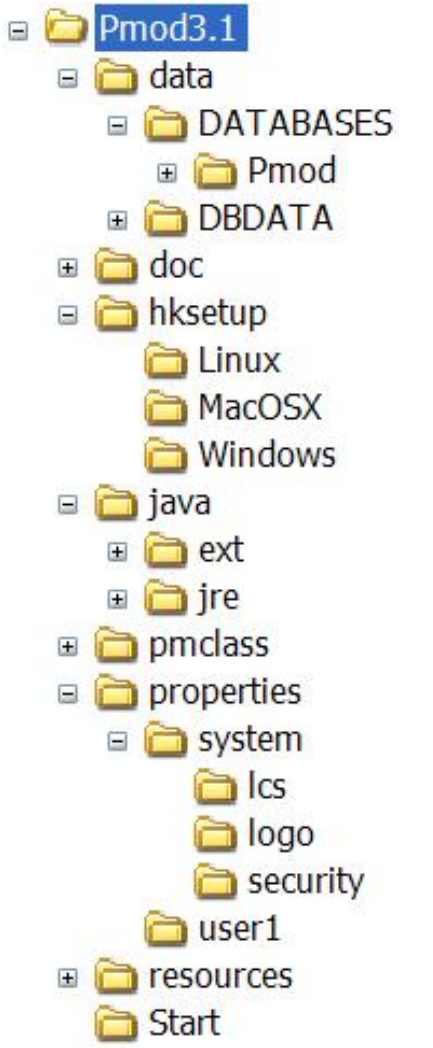
使用树的一个原因可能是因为要存储带有层次结构的信息。



族谱



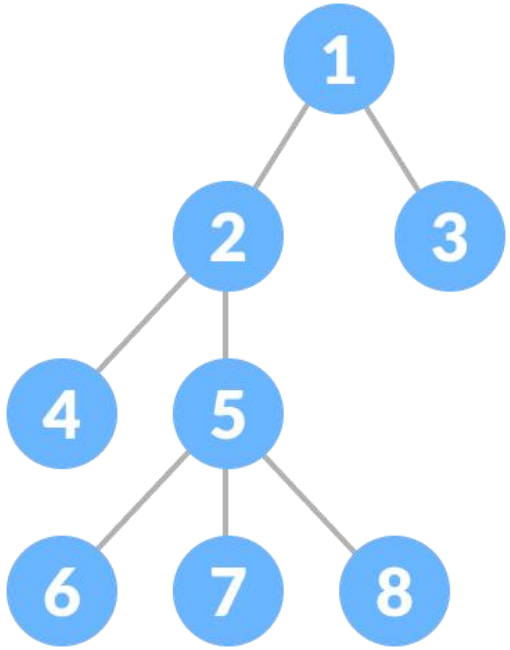
公司组织架构



文件目录结构

3-2 树 Tree

树是一种非线性分层数据结构，由通过边连接的节点组成。树中不含有环。



节点（node）是包含键或值以及指向其子节点的指针的实体。

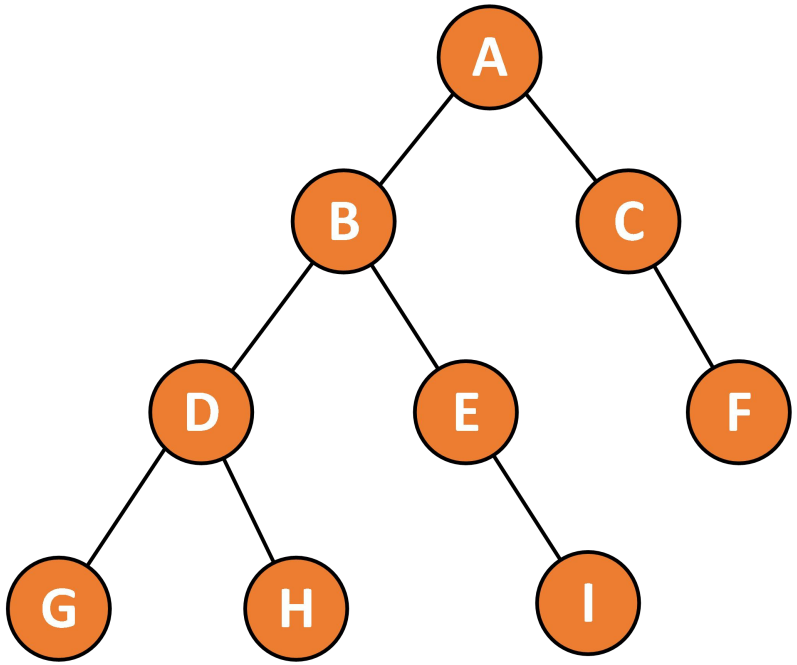
根（root）是树的最顶层节点。

边（edge）是任何两个节点之间的连接。

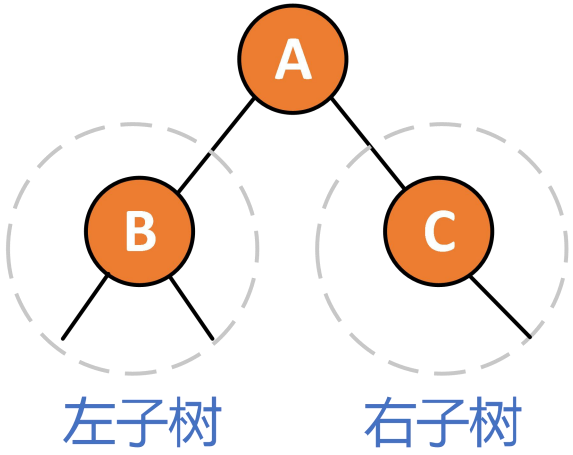
度（degree）是该节点的分支总数。

3-2 树 Tree

在一棵树中，根据节点之间层次关系的不同，对节点的称呼也有所不同。



- A是B和C的上级，则A是B和C的父节点（parent node），B和C是A的子节点（child node）
- G、H、I、F 都没有子节点，被称为叶子节点

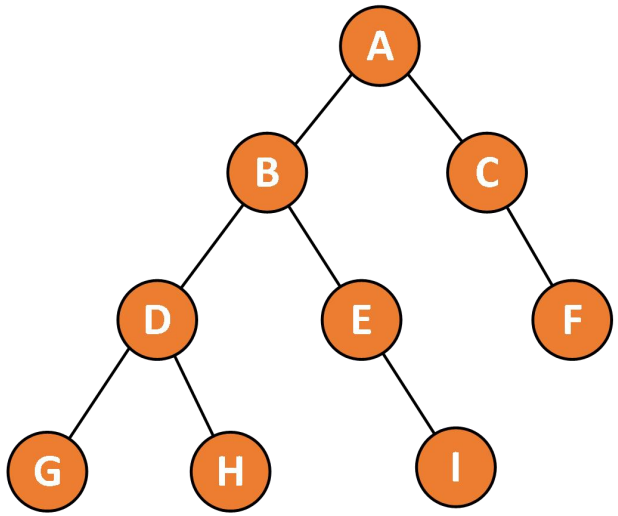


- 我们也可以将子节点视为子树（sub-tree）
- 节点的度也可定义为节点的子树个数

3-2 树 Tree

当有了一棵树之后，还需要用深度、层来描述这棵树中节点的位置：

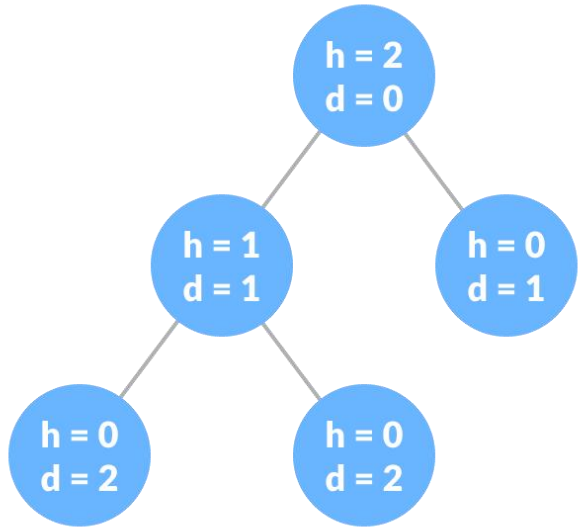
- 节点的层次从根节点算起，根为第一层，根的“孩子”为第二层，依此类推；
- 树的高度（也称为深度）是根节点的高度。



层	高度	深度
1	3	0
2	2	1
3	1	2
4	0	3

节点的高度是从节点到最深叶节点（即从该节点到叶节点的最长路径）的边数。

节点的深度是从根到节点的边数。

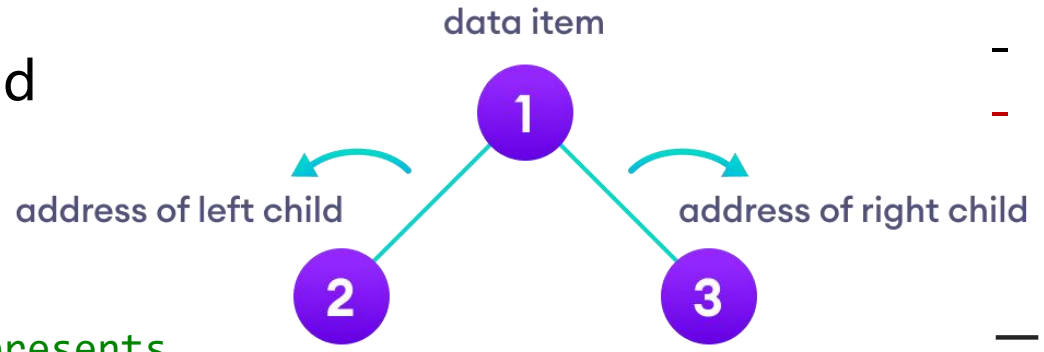


3-2 树 Tree

二叉树 Binary Tree

二叉树是一种树状数据结构，其中每个节点最多可以有两个子节点。

二叉树的每个节点由三项组成：
- data item
- address of left child
- address of right child



二叉树的基本操作：
- 插入一个元素
- 删除一个元素
- 搜索元素
- 遍历元素（有三种遍历方法）

```
# A Python class that represents
# an individual node in a Binary Tree
class Node:
    def __init__(self,key):
        self.left = None
        self.right = None
        self.val = key
```

二叉树的辅助操作：
- 寻找树的高度
- 查找树的级别
- 查找整棵树的大小

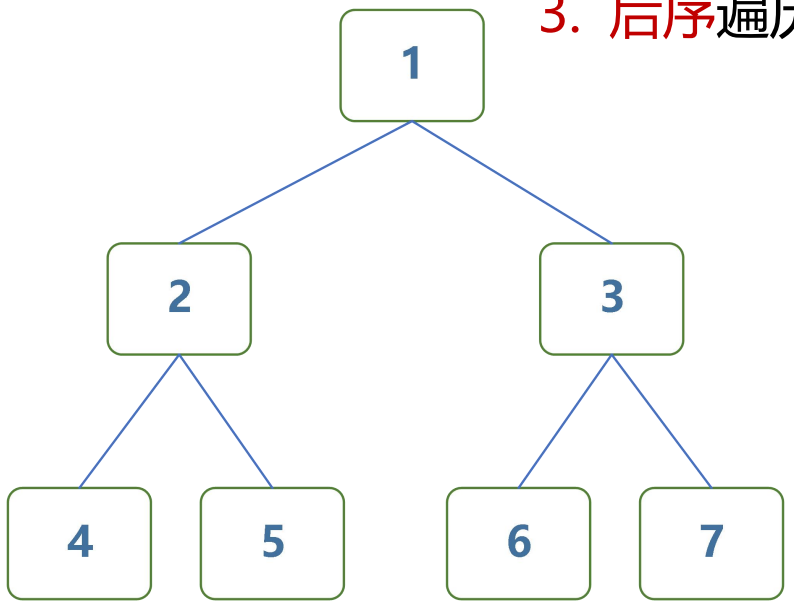
3-2 树 Tree

二叉树的遍历

二叉树是非线性的，因此节点的遍历可以有多种路径。

1. 前序遍历（PreOrder Traversal）：访问根节点-遍历左子树-遍历右子树
2. 中序遍历（InOrder Traversal）：遍历左子树-访问根节点-遍历右子树
3. 后序遍历（PostOrder Traversal）：遍历左子树-遍历右子树-访问根节点

三件事的顺序不同罢了。
从树到子树的遍历是一种**递归**结构。



1. PreOrder Traversal: 1-2-4-5-3-6-7
2. InOrder Traversal: 4-2-5-1-6-3-7
3. PostOrder Traversal: 4-5-2-6-7-3-1

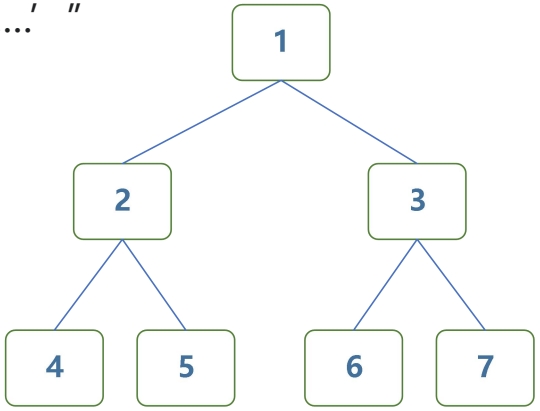
3-2 树 Tree

递归（Recursion）的思想

在计算机科学中，是指在函数的定义中使用函数自身的方法。

你小时候可能听过这个故事：

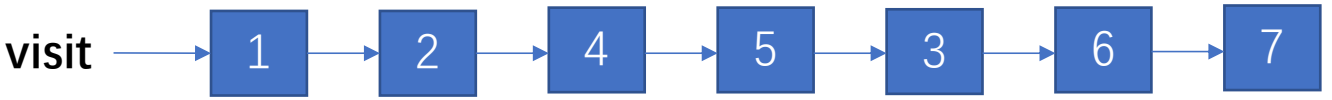
从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？“从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？‘从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？……’”



以二叉树的前序遍历（PreOrder Traversal）为例：

其规则为：访问根节点 - 遍历左子树 - 遍历右子树

```
# 前序遍历二叉树
def preorder(node):
    if node is not None:
        visit(node)  # 访问根节点
        preorder(node.left_child)  # 遍历左子树
        preorder(node.right_child)  # 遍历右子树
```

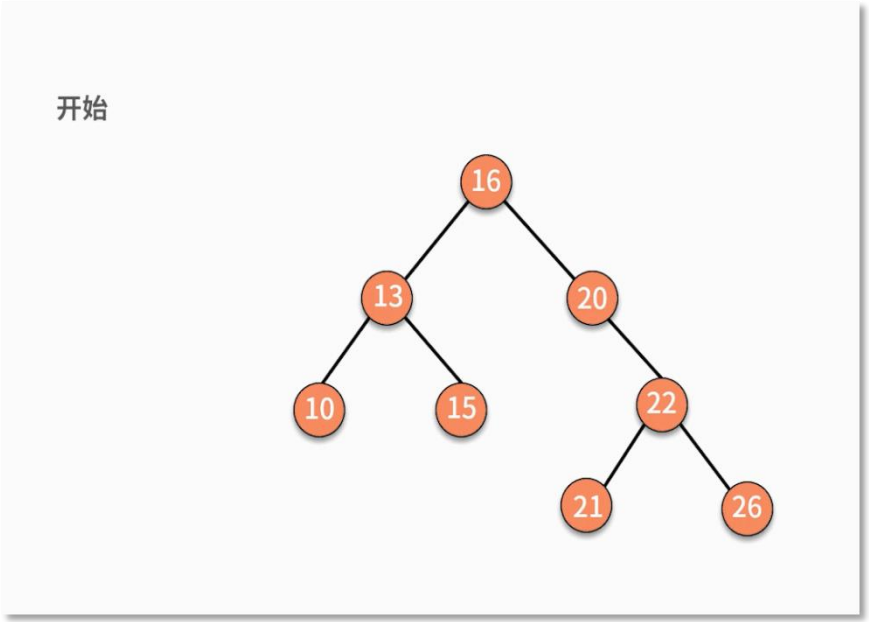


[完整的遍历代码详见畅课本单元参考资料]

3-2 树 Tree

二叉树的操作举例

二叉查找树（又称：二叉排序树）中的任意一个节点，其左子树中的每个节点的值，都要**小于**这个节点的值；其右子树中每个节点的值，都要**大于**这个节点的值。



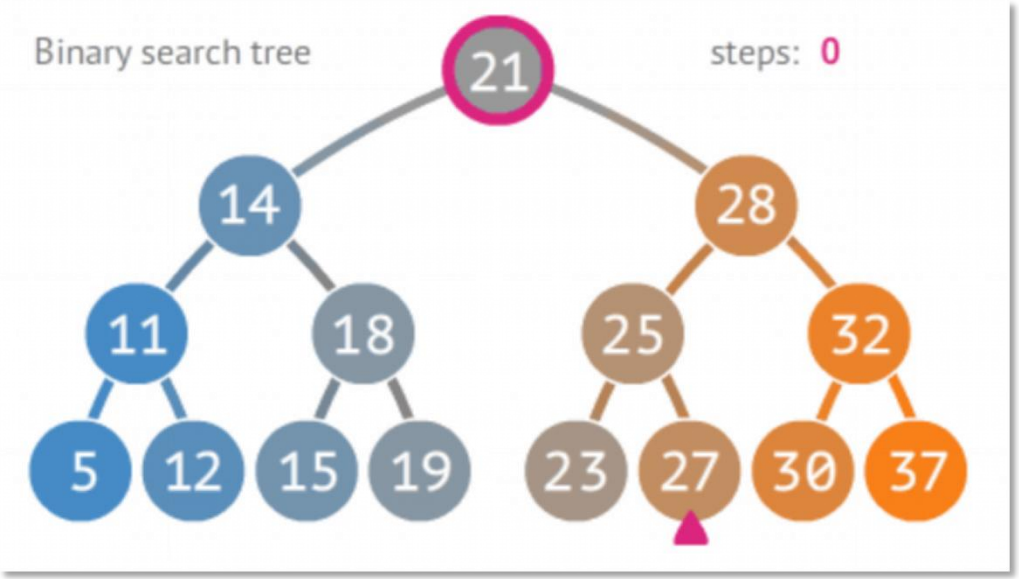
1. 插入一个数据元素：从root开始，如果要插入的数据比root的数据大，且root的右子节点不为空，则在root的右子树中继续尝试执行插入操作。直到找到为空的子节点执行插入动作。

2. 使用中序遍历，就可以输出一个从小到大的有序数据序列：
   10 13 14 15 16 20 21 22 26

二叉树的操作举例

二叉查找树中的任意一个节点，其左子树中的每个节点的值，都要**小于**这个节点的值；其右子树中每个节点的值，都要**大于**这个节点的值。

3. 搜索节点：从根节点出发，判断要搜索的节点和根节点的大小关系，然后在子树中递归执行搜索动作，如果相等就返回。

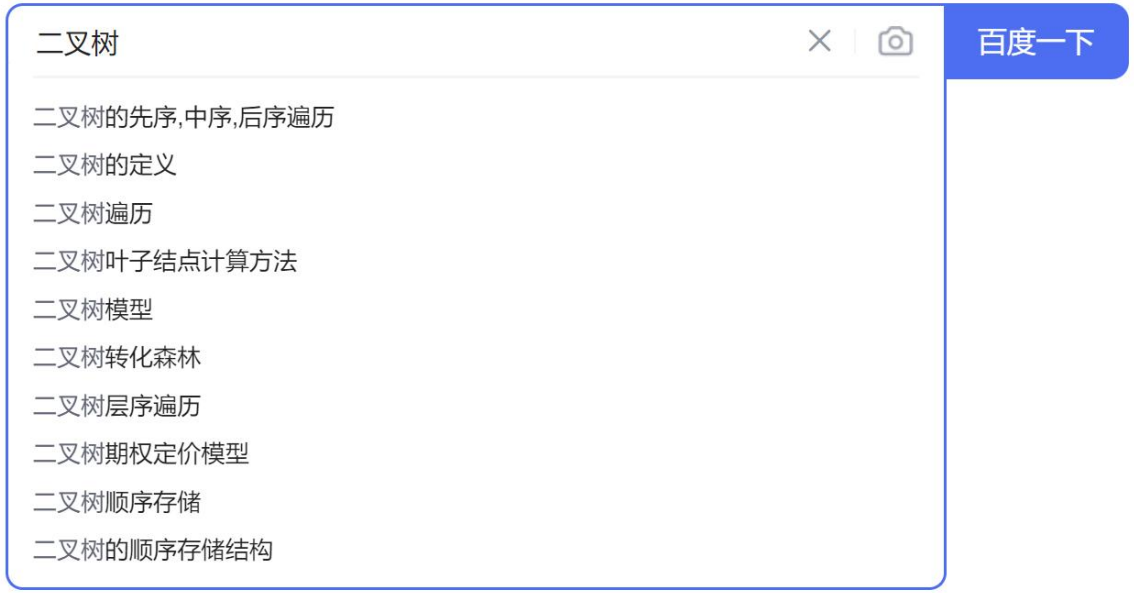


- 二叉搜索树不论哪一种操作，所花的时间都只和树的高度成正比。
- 既有链表的快速插入与删除操作的特点，又有数组快速查找的优势，所以应用十分广泛。

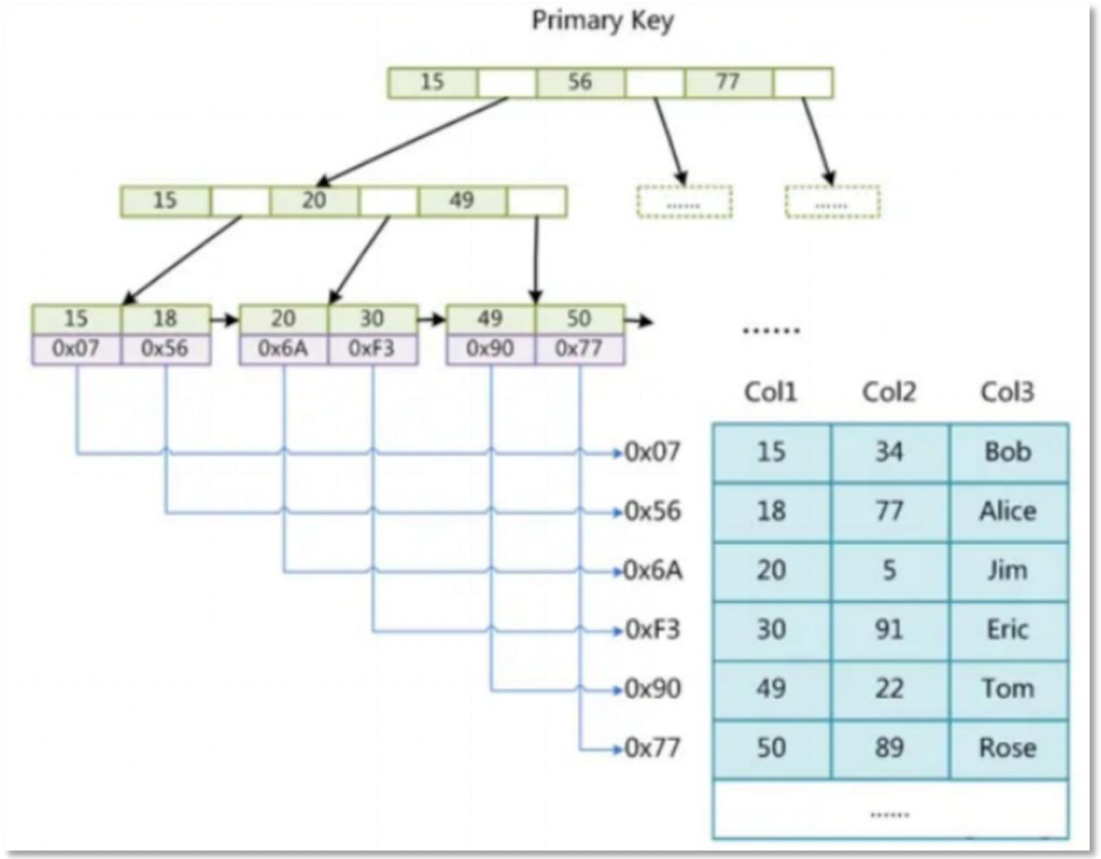
3-2 树 Tree

树的应用

1. 搜索引擎的关键词提示功能



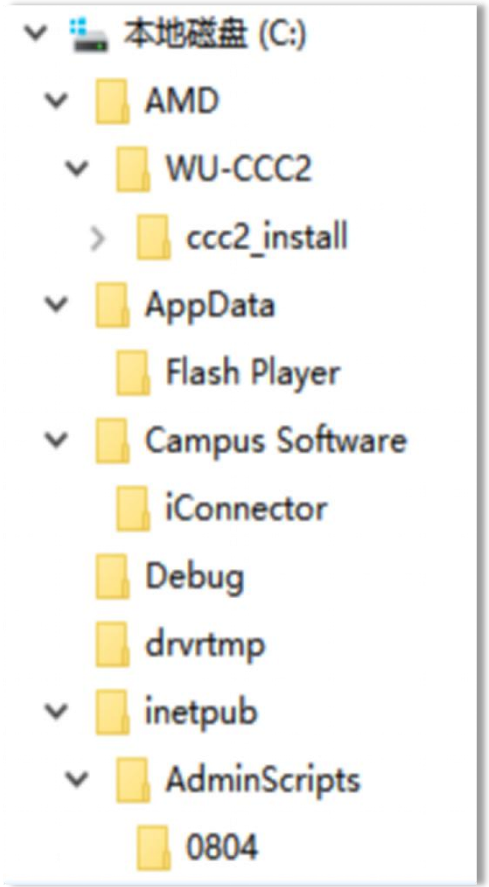
2. MySQL数据库的索引是树结构，查找效率极高



3-2 树 Tree

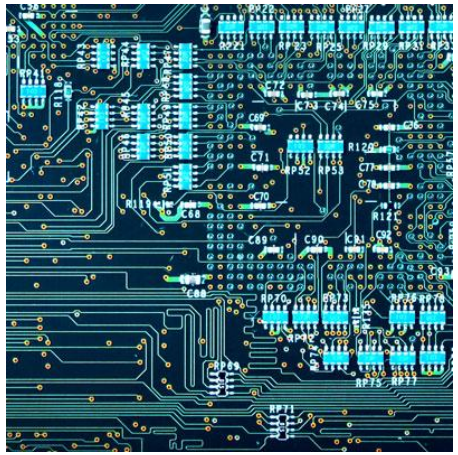
树的应用

3. Windows 和Linux的文件系统结构都是采用树进行存储的



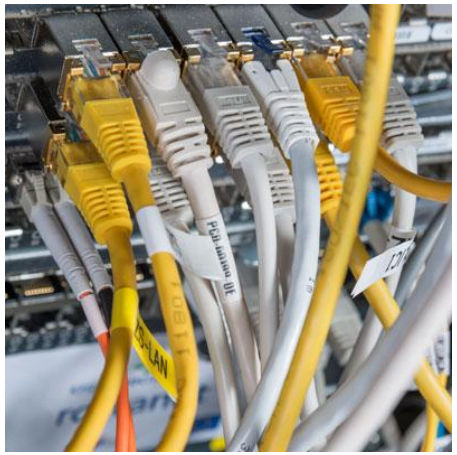

```
[root@VM_0_10_centos /]# tree -L 2
.
├── bin -> usr/bin
├── boot
│   ├── config-3.10.0-957.21.3.el7.x86_64
│   ├── efi
│   ├── grub
│   ├── grub2
│   ├── initramfs-0-rescue-0ea734564f9a4e2881b866b82d679dfc.img
│   ├── initramfs-3.10.0-957.21.3.el7.x86_64.img
│   ├── symvers-3.10.0-957.21.3.el7.x86_64.gz
│   ├── System.map-3.10.0-957.21.3.el7.x86_64
│   ├── vmlinuz-0-rescue-0ea734564f9a4e2881b866b82d679dfc
│   └── vmlinuz-3.10.0-957.21.3.el7.x86_64
├── data
├── dev
│   ├── autofs
│   ├── block
│   ├── bsg
│   ├── btrfs-control
│   ├── bus
│   ├── cdrom -> sr0
│   ├── char
│   ├── console
```

3-3 图 Graph



计算机芯片

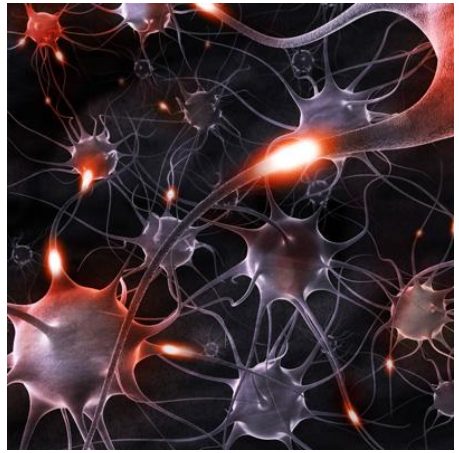
交通路网



计算机网络

每天我们都被无数的连接和网络所包围：公路和铁路、电话线、互联网、电子电路，朋友和家人之间甚至还有社交网络。

物流网络



神经网络

人际网络

所有这些例子都可以表示为图（Graph）

你还能想到其他例子吗？

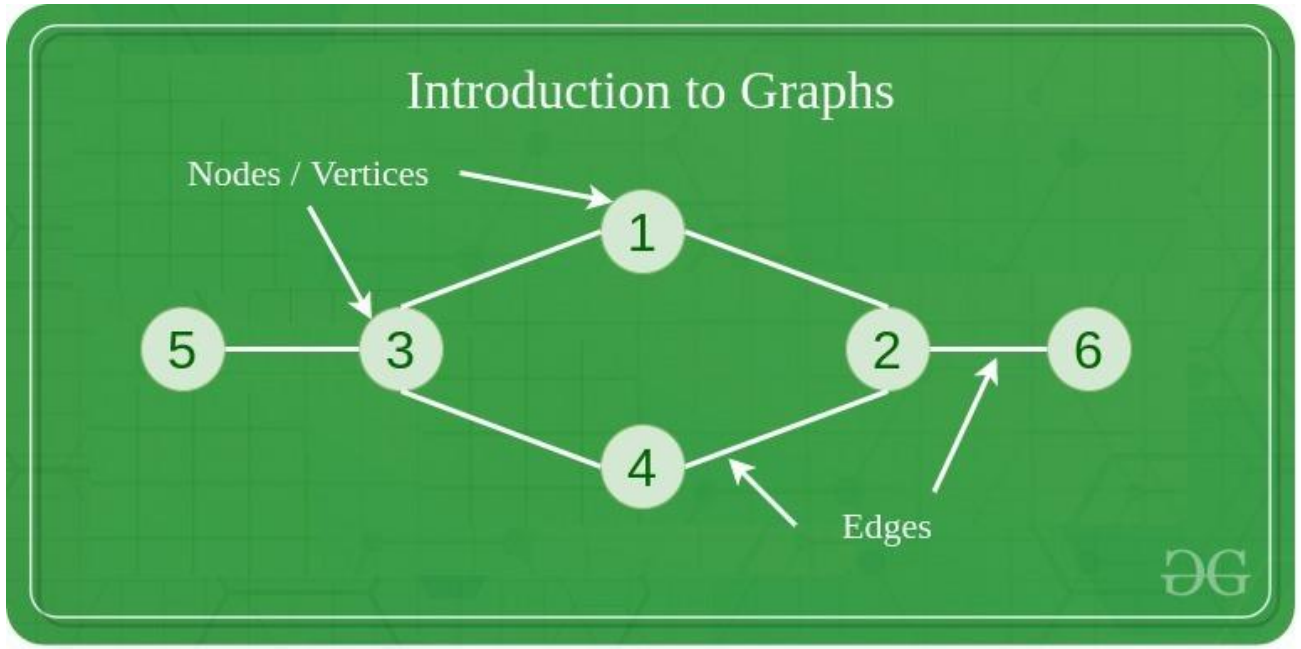
图源：https://mathigon.org/course/graph-theory/introduction

3-3 图 Graph

图是由顶点（Verticals）和边（Edges）组成的非线性数据结构。

顶点有时也称为节点（Node），边是连接图中任意两个节点的线或弧。

用数学的语言说，图由一组顶点（V）和一组边（E）组成，用 G(E, V) 表示。



- 图将信息中的实体，以及实体之间的关系，分别抽象表达成为顶点以及顶点间的边这样的数据结构。
- 可用于解决许多现实生活中的问题。

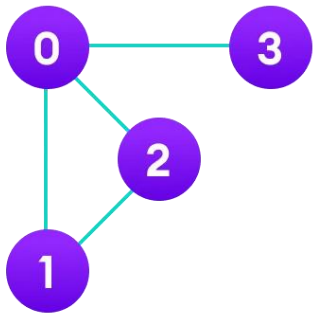
3-3 图 Graph

图的表示　图通常以两种方式表示：

1. 邻接矩阵

用二维数组表示顶点之间的关联关系

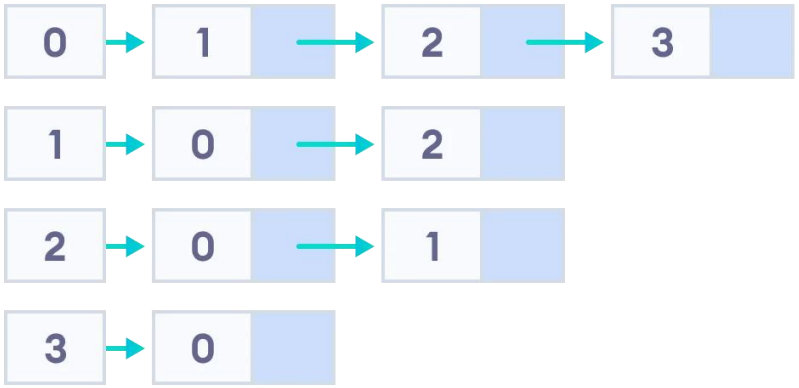
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0



- 简单直观，可快速查询顶点间的关联关系。
- 但是，占用太多空间。试想，如果一个图有100万个顶点，其中只有100个顶点之间有关联 （这种情况叫做稀疏图），却不得不建立一个10000 X 10000的二维数组，实在太浪费了。

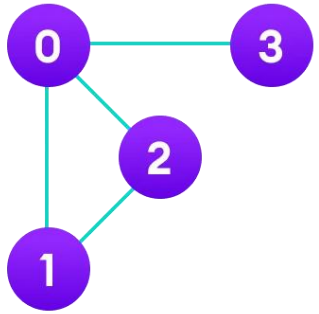
2. 邻接表

邻接表将图表示为链表数组



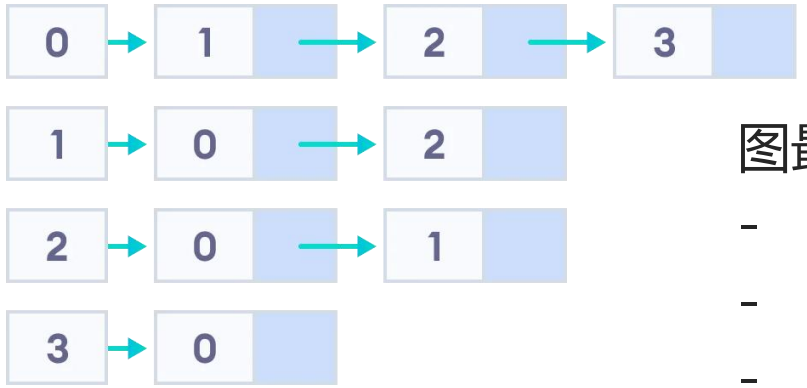
- 邻接表在存储方面是高效的，因为我们只需要存储边的值。对于具有数百万个顶点的图，这可能意味着节省了大量空间。
- 查找邻接表没有邻接矩阵快，因为必须首先探索所有连接的节点才能找到它们。

3-3 图 Graph



图的实现

	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0



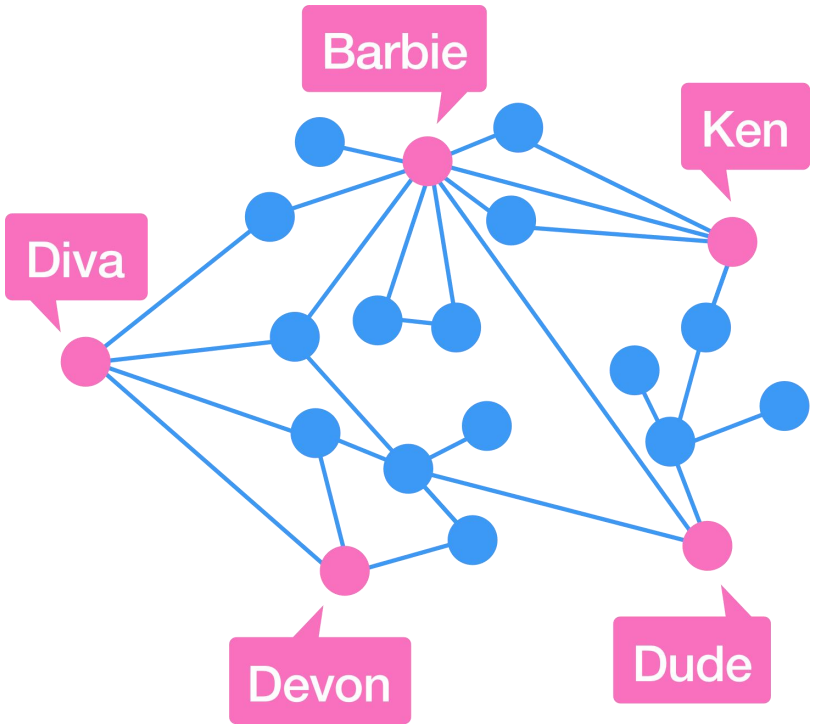

```
graph = [[0, 1, 1, 1],
         [1, 0, 1, 0],
         [1, 1, 0, 0],
         [1, 0, 0, 0]]
```

```
graph = {0: {1, 2, 3},
         1: {0, 2},
         2: {0, 1},
         3: {0}}
```

图最常见的图操作：

- 检查元素是否存在于图中
- 图遍历
- 向图中添加元素（顶点、边）
- 寻找从一个顶点到另一个顶点的路径

图的应用举例



人际关系，每个节点表示一个人，
连边表示两个人是朋友关系。

- 谁的朋友最多?
- 发邀请，谁发出的邀请最多?
  - 如果只能发给自己的朋友?
  - 如果还可以发给朋友的朋友?

Name	Number of friends	Number of friends of friends not already friended	Total guests invited
Barbie	9	4	13
Ken	4	7	11
Diva	4	3	7
Devon	3	3	6
Dude	3	14	17

本次课程结束，请继续加油！

Computational Thinking

计算机与网络空间安全学院
School of Computer and Cyber Sciences