数据结构课程讲稿第4章

contents

目录

01 引言

主题概述

线性数据结构

线性数据结构是一种数据元素之间存在一对一关系的数据结构，包括数组、链表、栈、队列等。

非线性数据结构

非线性数据结构是一种数据元素之间存在一对多或多对多关系的数据结构，包括树、图、哈希表等。

数据结构在计算机科学中的重要性

数据结构是计算机科学中的基础概念，它决定了程序和算法的效率。掌握数据结构对于解决实际问题、优化程序性能以及设计高效算法至关重要。

课程目标和意义

课程目标

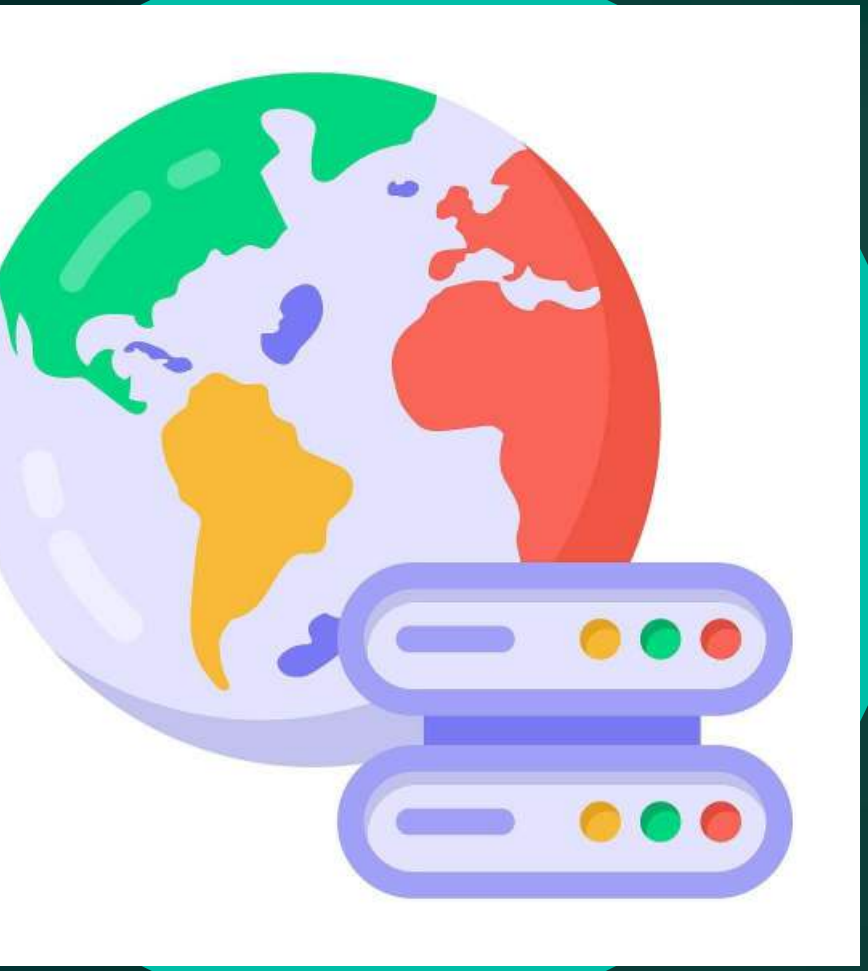
本课程的目标是帮助学生掌握常见的数据结构及其操作，理解不同数据结构的特性和适用场景，并能够在实际问题中灵活运用。

课程意义

学习数据结构有助于培养学生的逻辑思维、问题解决和算法设计能力。同时，数据结构在计算机科学、软件工程、数据库系统等领域有着广泛的应用，对于学生未来的职业发展具有重要意义。

02 数据结构基础概念

数据结构定义



01

数据结构定义：数据结构是数据的组织、排列和表示的方式，它涉及到数据之间的逻辑关系和物理存储。

02

数据结构是计算机科学和软件工程领域中一个重要的概念，它影响着程序的性能、可维护性和可重用性。

03

数据结构是算法的基础，许多算法的实现都依赖于特定的数据结构。

数据结构的重要性

提高程序性能

合理的数据结构可以减少数据检索、插入和删除的时间复杂度，从而提高程序的执行效率。

促进代码可维护性

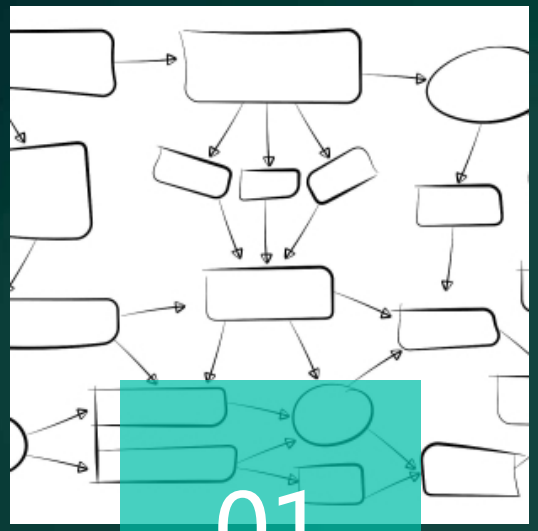
良好的数据结构有助于清晰地表达数据的逻辑关系，使代码更易于理解和维护。

提升软件质量

合理的数据结构有助于提高软件的可重用性和可扩展性，降低软件开发的成本。

```
        { path: '/inbox',              component: Inbox,              name: 'Inbox',
        { path: '/calendar',           component: Calendar,           name: 'Calendar',
        { path: '/locator',            component: Locator,            name: 'Locator',
        { path: '/tasks',              component: Tasks,              name: 'Tasks',
        { path: '/documents',          component: Documents,          name: 'Documents',
    ]
},
{ path: '/reports',                    component: Reports,            name: 'Reports',
{ path: '/',                           component: SubMenu,            name: 'Accounting',
    children: [
        { path: '/products',           component: Products,           name: 'Products',
        { path: '/orders',             component: Orders,             name: 'Orders',
        { path: '/timesheets',         component: Timesheets,         name: 'Timesheets',
        { path: '/invoices',           component: Invoices,           name: 'Invoices',
        { path: '/users',              component: Users,              name: 'Users',
    ]
},
{ path: '/',                           component: SubMenu,            name: 'Admin',
    children: [
        { path: '/roles',              component: Roles,              name: 'Roles',
        { path: '/users',              component: Users,              name: 'Users',
        { path: '/users/:id',          component: User,               name: 'User',
        { path: '/scripts',            component: Scripts,            name: 'Scripts',
        { path: '/surveys',            component: Surveys,            name: 'Surveys',
        { path: '/tags',               component: Tags,               name: 'Tags',
        { path: '/audits',             component: Audits,             name: 'Audits',
        { path: '/pipelines',          component: Pipelines,          name: 'Journeys',
        { path: '/groups',             component: Group,              name: 'Groups',
    ]
},
{ path: '/',                           component: SubMenu,            name: 'Settings',
    children: [
        { path: '/modules',            component: Modules,            name: 'Modules',
        { path: '/company-setup',      component: Company,            name: 'Company',
        { path: '/appointment-setup',  component: AppointmentSetup,   name: moduleTitle('Appointme
        { path: '/terminology',        component: Terminology,        name: 'Terminology',
        { path: '/workflows',          component: Workflows,          name: 'Workflows',
        { path: '/lead-setup',         component: LeadSetup,          name: 'Lead list',
    ]
```

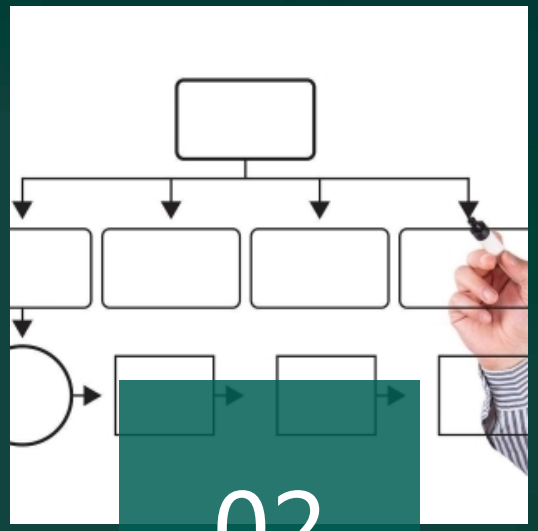
数据结构的分类



01

线性数据结构

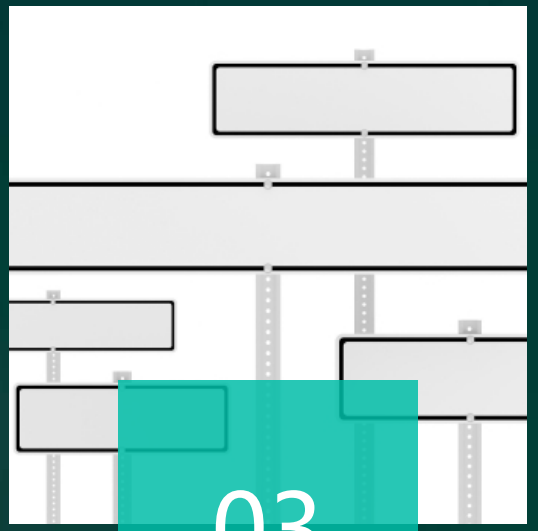
包括数组、链表、栈、队列等，它们按照一定的顺序存储数据，满足线性关系。



02

树形数据结构

以树形方式组织数据，包括二叉树、多叉树、B树等，满足层次关系。



03

图状数据结构

以图的方式表示数据之间的关系，包括邻接矩阵、邻接表等。

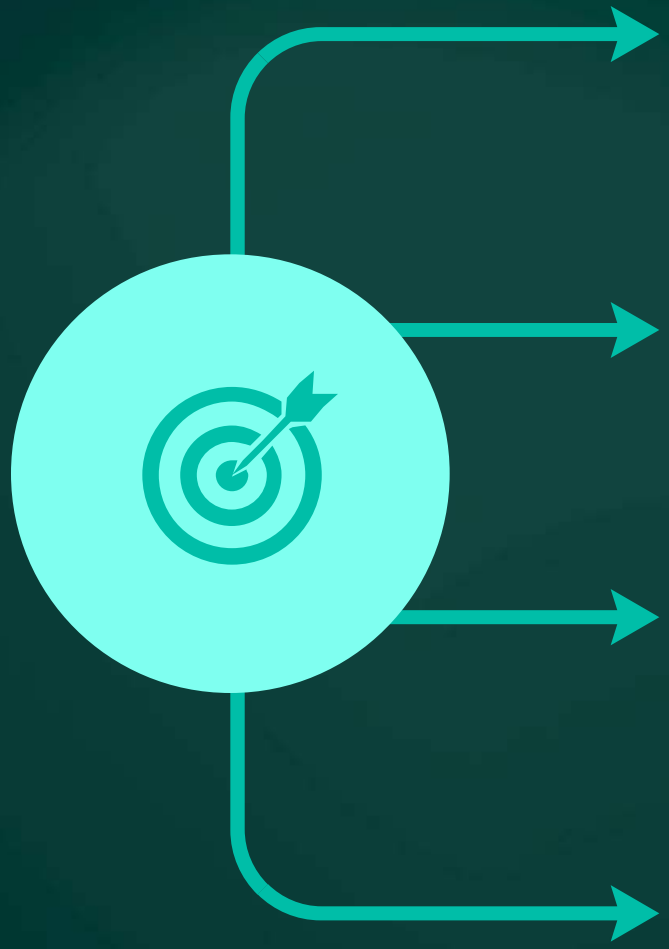
04

散列数据结构

利用哈希函数将键映射到位置，实现数据的快速检索和插入。

03 线性数据结构

线性表



01

线性表是数据结构中最基础的一种，它由n个元素组成，每个元素都有一个唯一的地址，并且满足元素的排列顺序。

02

线性表可以分为顺序存储和链式存储两种方式，顺序存储的线性表称为顺序表，链式存储的线性表称为链表。

03

线性表的基本操作包括插入、删除、查找和修改等。

04

线性表的应用非常广泛，例如数组、列表、队列和栈等都是线性表的常见应用。

栈和队列

01

栈是一种特殊的线性表，它只允许在表的一端进行插入和删除操作，这种表的一端被称为栈顶，另一端被称为栈底。

02

栈的基本操作包括压栈、弹栈、查看栈顶元素和判断栈是否为空等。

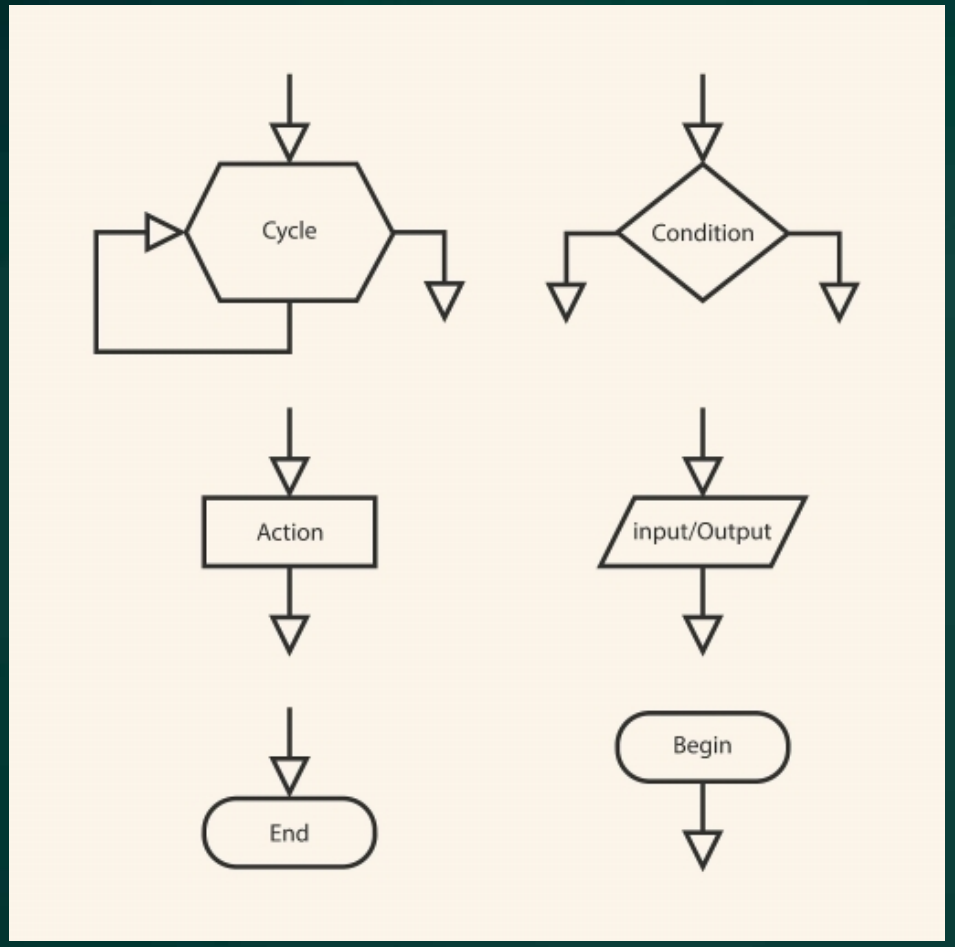
03

队列也是一种特殊的线性表，它只允许在表的另一端进行插入操作，而在表的另一端进行删除操作。

04

队列的基本操作包括入队、出队、查看队首元素和判断队列是否为空等。

特殊线性数据结构



特殊线性数据结构包括循环队列、循环链表、双向链表和双向循环链表等。

循环队列和循环链表在存储空间上进行了优化，它们将最后一个元素的下一个元素指向第一个元素，实现了空间的有效利用。

双向链表和双向循环链表在操作上进行了优化，它们可以在任何位置进行插入和删除操作，提高了操作的效率。