数据结构期末串讲

主讲人：吴奇宣

软件学院2021级本科生
2025 人工智能导论课程信类助教组长
2024 程序设计基础课程助教组长
2023 数据结构课程助教
2023 程序设计基础课程助教

串讲时间：2025年06月18日 15:00

北京航空航天大學
BEIHANG UNIVERSITY

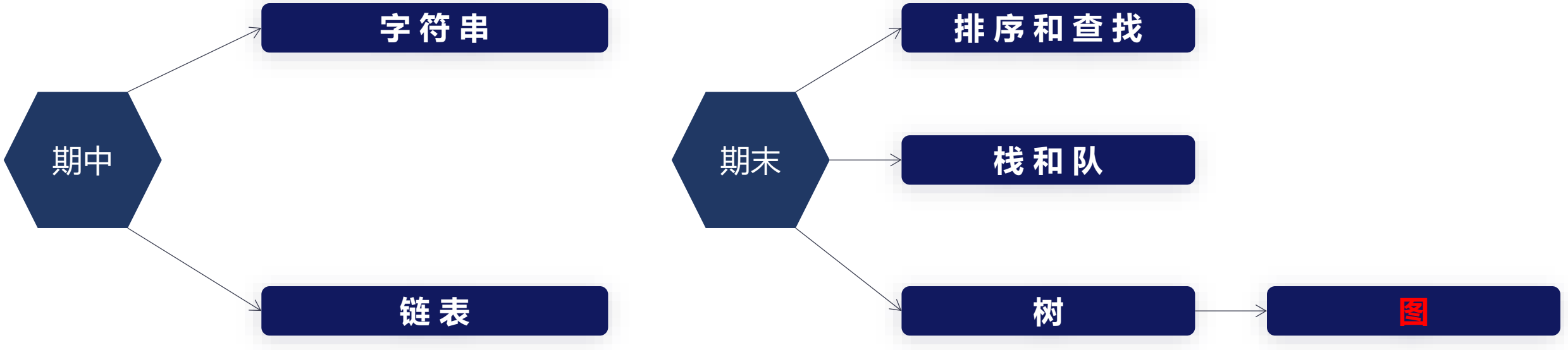
目录

CONTENTS

目录页 CONTENTS PAGE

Part 1: 数据结构回顾

期中期末侧重考点



- 结构体 -> **排序和查找**
- 链表 -> **栈和队**
- 树 -> **图**

1 – 结构体

需求：存储若干个学生的信息，包括姓名、年龄和成绩

常规写法：

```
char name[1005][10];
int age[1005];
double score[1005];
```

存在的不足

- 明明对应着同一个学生，但是各个变量之间看上去毫无关系
- 对三个数组的操作经常是绑定的，要对多个数组都进行类似的处理

1 – 结构体

结构体的实现：

```
struct Student {
    char name[20];
    int age;
    float score;
};

struct Student stu[1005];
```

优 点

- 对应着同一个学生的不同信息，封装在了同一个结构里
- 对同一个学生不同信息的操作，可以一起被完成（指的是交换、赋值、传参等）

结构成员、结构类型、结构变量（以及他们的赋值）、自引用结构

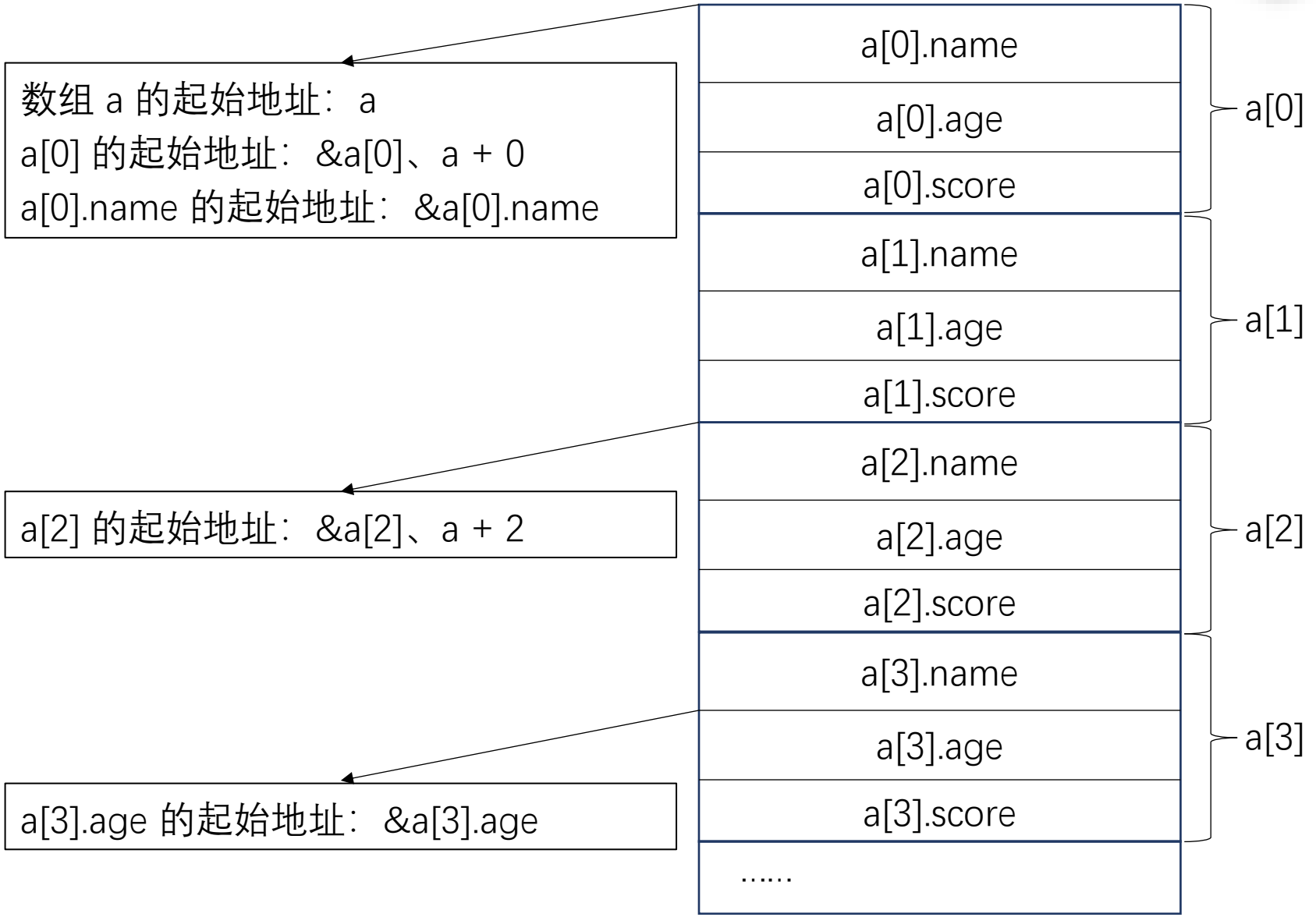
不同变量难管理

统一打包最省力

2 - 数组

struct Student 类型

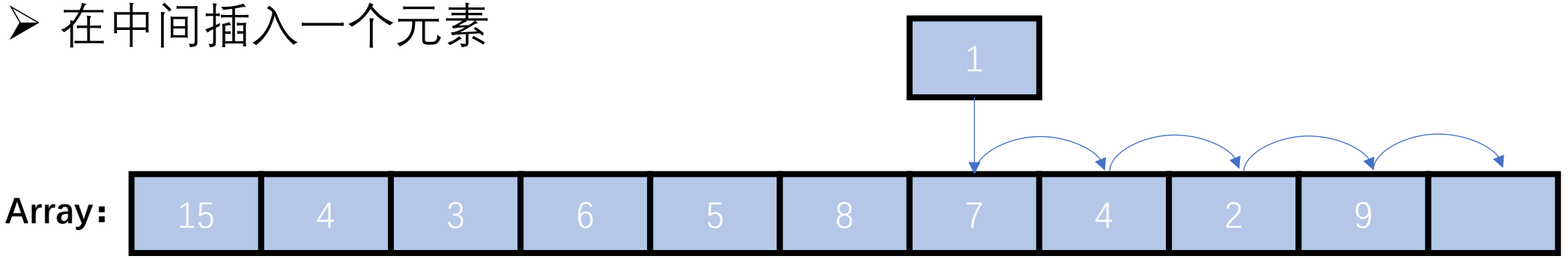
name : char [20]
age : int
score : double



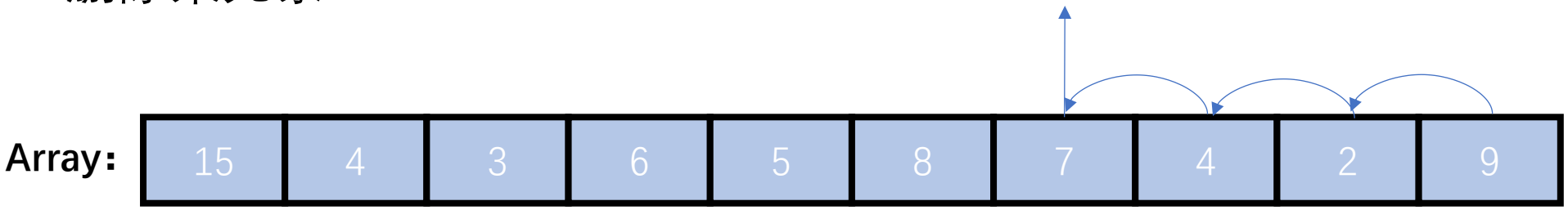
2 - 数组

基于地址空间的连续，自然地得到了一个逻辑上的线性关系

- 访问第 i 个元素：O(1)
- 在中间插入一个元素



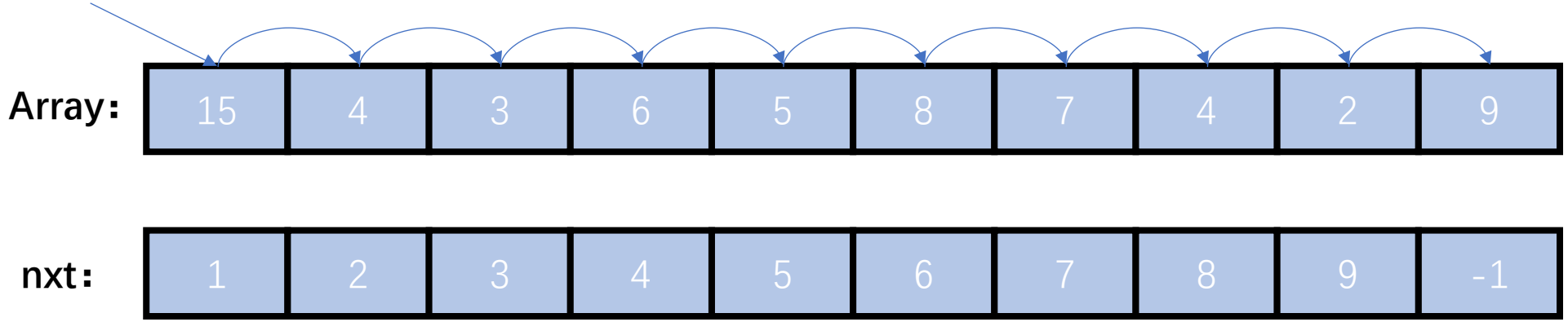
- 删除某元素



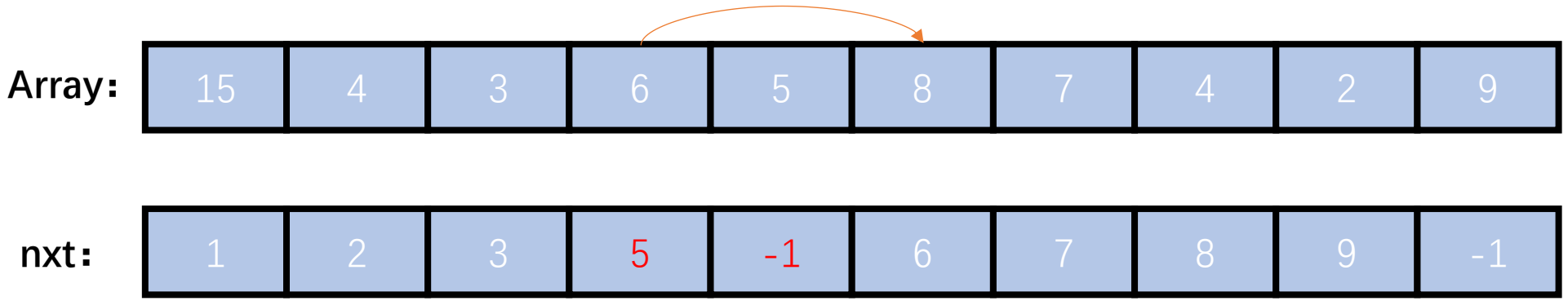
3 – 链表

为什么会出现这样的问题?

数组实现的链式存储:

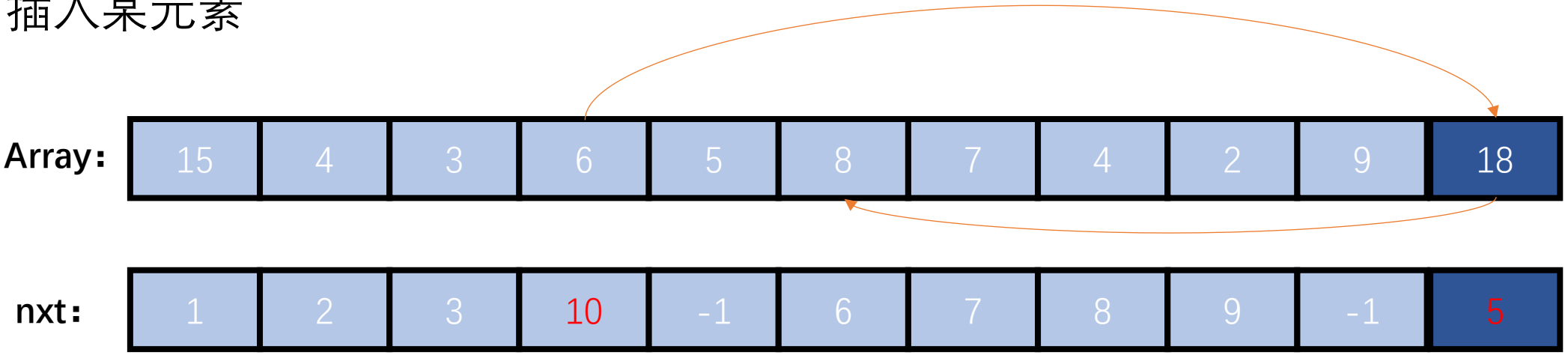


- 删除某元素



3 – 链表

- 插入某元素



基于数组的链式顺序表

```
struct node {
    int num;
    int pre, nxt;
};
struct node a[105];
```

动态申请释放空间

基于自引用的链式顺序表

```
typedef struct node  node;
typedef struct node* nptr;
struct node {
    int num;
    nptr pre, nxt;
};
nptr head;
```

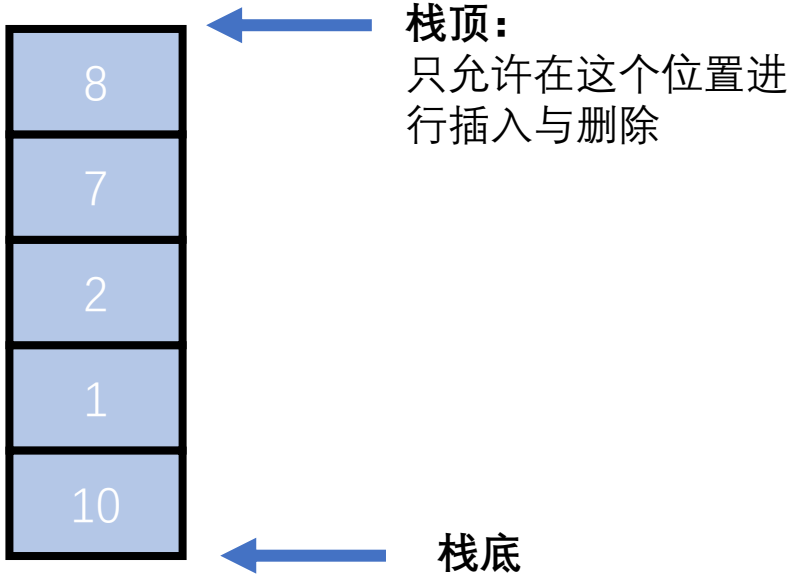
3 – 顺序存储 vs 链式存储

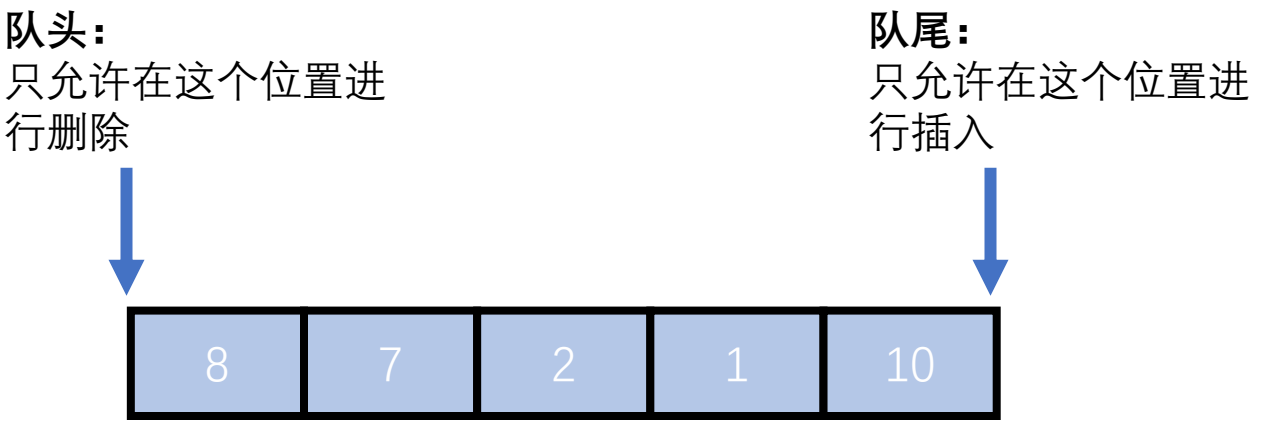
单链表、双链表、循环链表的实现；常见操作的比较

操作	顺序存储	链式存储
检索线性表中第 i 个元素	O(1)	O(n)
查找某元素在线性表中的位置	O(n)	O(n)
在指定位置存入元素值	O(1)	O(1)
在指定位置插入元素	O(n)	O(1)
删除指定位置的元素	O(n) （可以使用标记数组的方式删除，效率大大提升）	O(1)
查找某元素的前驱	O(1)	单链表：O(n) 双链表：O(1)
查找某元素的后继	O(1)	O(1)

4 – 栈和队

相当于退化版的链表





- 判断括号序列是否合法
- 中缀表达式转后缀表达式
- 函数的递归调用
- 文本编辑撤销操作
- 浏览器的前进后退

- 打印任务排队
- **广度优先搜索（层序遍历）**
- CPU 进程调度

4 – 栈和队

栈和队的操作并不会改变连续性，所以可以用数组实现

栈

```
// top 维护栈顶，栈底一直是 0
int stk[1005], top = -1;

void push(int a) {
    stk[++top] = a;
}

int pop() {
    return stk[top--];
}
```

队

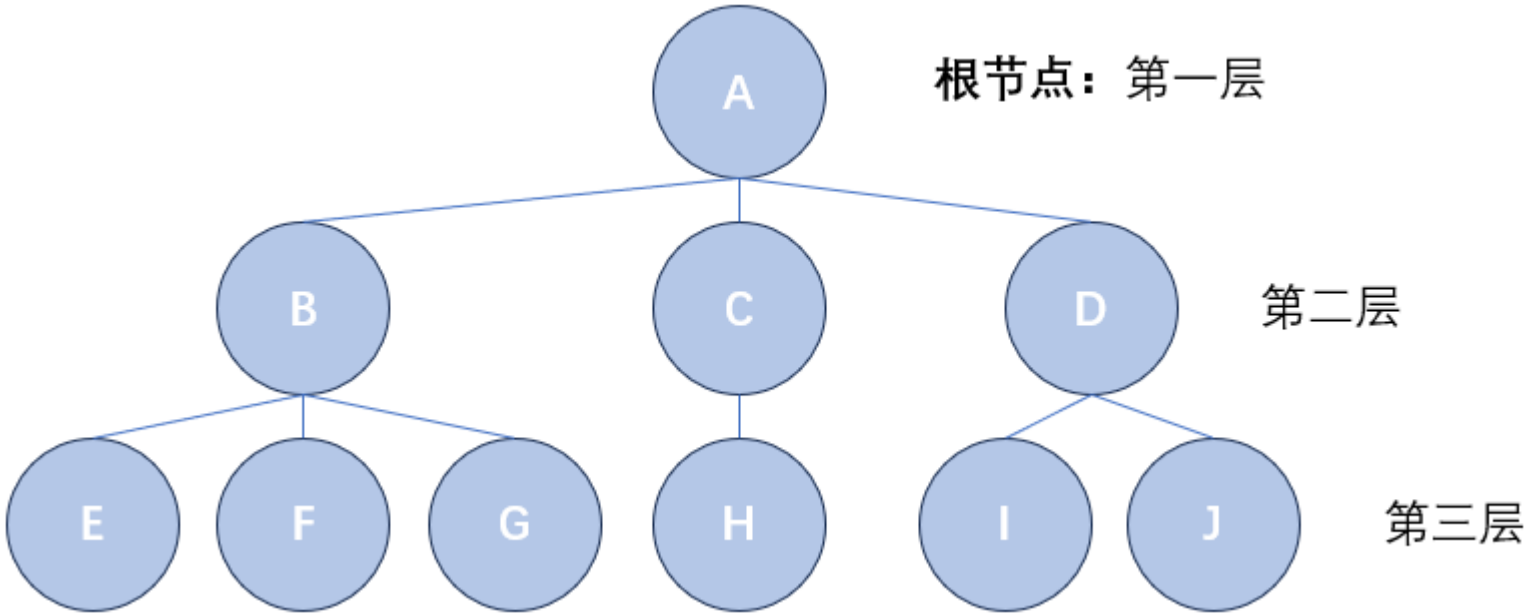
```
// que 代表队数组，front 代表队首，rear 代表队尾
int que[1005];
int front = 0, rear = -1;

void enqueue(int a) {
    que[++rear] = a;
}

int dequeue() {
    return que[front++];
}
```

栈队相关的小算法：

如计算栈的容量、判断是否是合法的入栈/出栈序列、中缀表达式转换等

5 – 树

前面三种结构都有一个共同的特点：单前驱，单后继，都是线性的数据结构



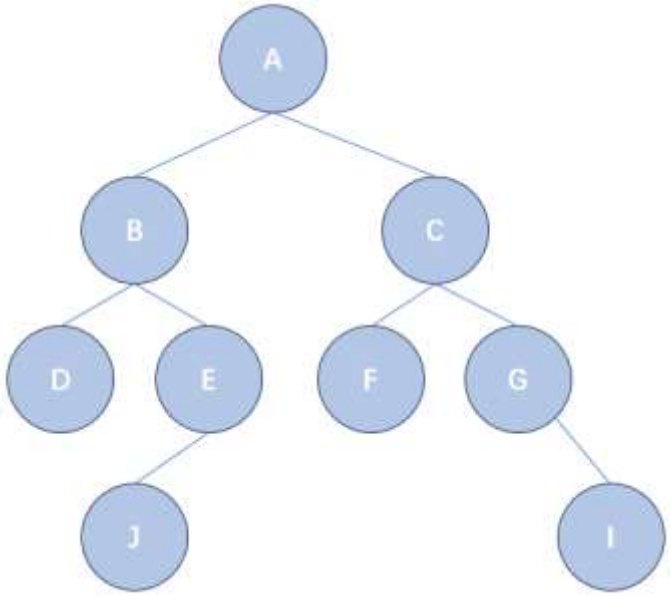
单前驱，多后继

树中最重要的两个关系： **每个节点有唯一的前驱元素**

边数（度数）= 节点数 - 1

5 – 树

一种比较特殊的树：二叉树




```
struct node {
    int val;
    nptr ls, rs;
    // 根据情况写前驱元素指针域
    // nptr fa;
};
```

二叉树的遍历有多种方式：**先序、中序、后序、层序**

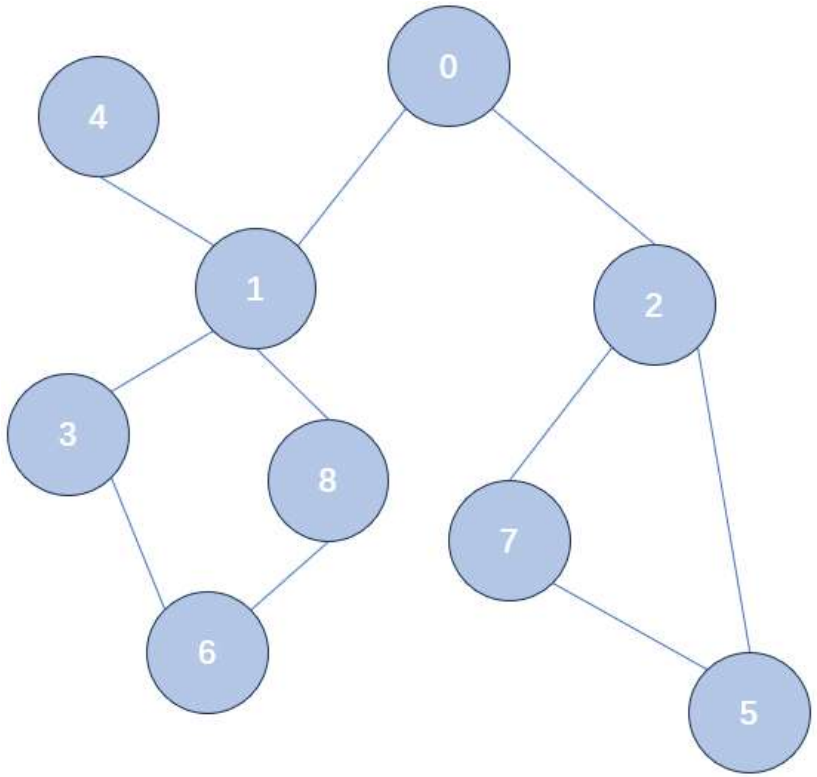
对于**普通的树**，也有先序、后序和层序，**没有中序**

树的相关概念，树的遍历，树相关的计算，Huffman树，推断二叉树，二叉搜索树，表达式树

```
void pre_order(nptr p) {
    if(p == NULL)
        return ;
    printf("%c ", p -> i);
    pre_order(p -> ls);
    pre_order(p -> rs);
}
```

6 – 图

在树的基础上，拥有多前驱、多后继（可以说树是一种特殊的图，可以用建图的方式来建一般的树）

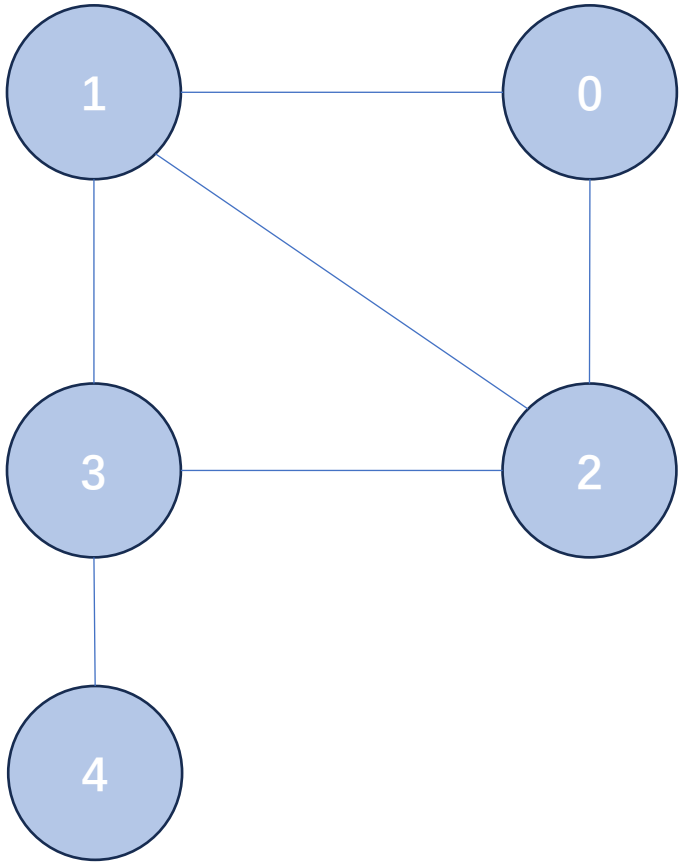


相关概念

- 有向图：入度、出度
- 无向图
- 带权图

6 – 图

图的存储:



临接矩阵

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

临接表（Table：链的数组）

下 标	对 应 的 链
0	1 -> 2
1	0 -> 2 -> 3
2	0 -> 1 -> 3
3	1 -> 2 -> 4
4	3

6 – 图

图的遍历:

临接矩阵

```
void dfs(int u) {
    // 标记为已访问
    printf("%d ", u);
    vis[u] = 1;
    // 遍历所有顶点
    for (int v = 0; v < n; v++) {
        // 如果有边且未访问
        if (e[u][v] && !vis[v])
            dfs(v);
    }
}
```

临接表

```
void dfs(int u) {
    printf("%d ", u);
    vis[u] = 1;
    for (nptr p = e[u]; p != NULL; p = p->nxt) {
        int v = p -> val;
        if (!vis[v]) dfs(v);
    }
}
```

6 – 图

图的算法：

- **Dijkstra 求单源最短路（千万千万要掌握）**
- 最小生成树 Kruskal 算法
- 最小生成树 Prim 算法
- 拓扑排序

图的存储、图的相关概念、AOE 网

7 – 排序和查找

七种排序

- 选择排序
- 冒泡排序
- 插入排序
- 希尔排序
- 堆排序
- 归并排序
- 快速排序

四种查找

- 顺序查找
- 二分查找
- 索引查找
- 散列查找

原理我们先不在此讲解，可以翻看我的仓库，上面有详细讲解

堆排序

先来介绍什么是堆。

有的时候，我们并不关心数组整体的有序性，仅仅关心一个数组里谁最大，那我们可以怎么办？有的同学可能就会说，我们可以先进行排序，然后直接取出最大的元素。可是如果我增加需求，我可能会增加若干次操作，每次操作都可能增加或者删除当前最大的元素，经过几次操作后再问你谁是最大的元素，你还是得进行一次排序之后，才能告诉我谁是最大的元素，即便实际上大部分元素的有序性是没有被改变的。

堆就是用来解决这个问题的，堆支持的操作有向堆中插入元素，删除最大的元素，以及询问当前堆中最大的元素，并且前两种操作的时间复杂度都是 O(logn)，最后一种操作的时间复杂度是 O(1) 的。

堆的实现就是一颗完全二叉树，不过在这棵二叉树上，满足所有节点的元素都比子结点的元素大，比如下面就是一个堆（严格来讲，叫做大顶堆；还有要求父节点都比子节点小的，用来维护最小值的，叫做小顶堆）：

当向堆中插入元素时，我们先把它放在堆的最后（图标

散列查找（哈希查找）

先介绍一下散列的思想，比如我们想记住某人的电话号时，我们一般只用记住后四位，等我们换到一个电话时，我们一看后四位是 1234，我们就基本能知道这个是张三打来的电话，而不用把所有位都记住才能确定（这样的想法在日常生活中也是非常常用的，比如坐网约车时问你手机号后四位，取快递柜里的外卖时只需要输入手机后四位），假如我们有一个电话簿，我们可以用后四位记录对应的人，比如：

```
1234: 张三
2345: 李四
```

这样，我们把一个大整数映射为了一个只有四位的小整数，就能够当做数组下标了，就可以用 user[1] 表示手机尾号为 1 的人的名字。这就是哈希（散列）的思想，将某种数据类型转换为整数类型（比如大整数、字符串等）。

但是显然，我们在取外卖的时候可能会遇见这种情况：有其他人和我们手机尾号相同，外卖柜不知道我们想要取的到底是哪个外卖，也就是有多个元素的哈希值相同，这就叫做**哈希冲突**。结合日常生活中的经验，这种情况下，外卖柜会要求我们输入完整的手机号，从而确定我们到底是哪一个人，也就是通过存储完整的值来解决哈希冲突：

```
1234: 张三（18712341234） 王五（15812341234） 赵六（13012341234）
2345: 李四（18723452345）
```

也就是说，在空间方面，哈希的方法并没有什么优化，因为它还是要存储所有的值。但是在时间方面，哈希先通过计算哈希值的方式，缩小目标元素的范围，然后再在哈希值相等的元素里比对，看看谁的值相等，以此确定目标元素。简单来说，就是**先缩小目标范围，再逐个进行比对**。

对应的数据结构就叫做哈希表，一提到表（Table）同学们就不难想象到应该是一个链的数组，数组下标对应着哈希值，而一条链对应着哈希值相等（哈希冲突）的一堆元素。每条链一般用链式存储来实现（节省空间），所以一般使用散列查找时，先计算哈希值，找到对应的链，然后再在对应的链中使用顺序查找。下面给出一个参考代码：

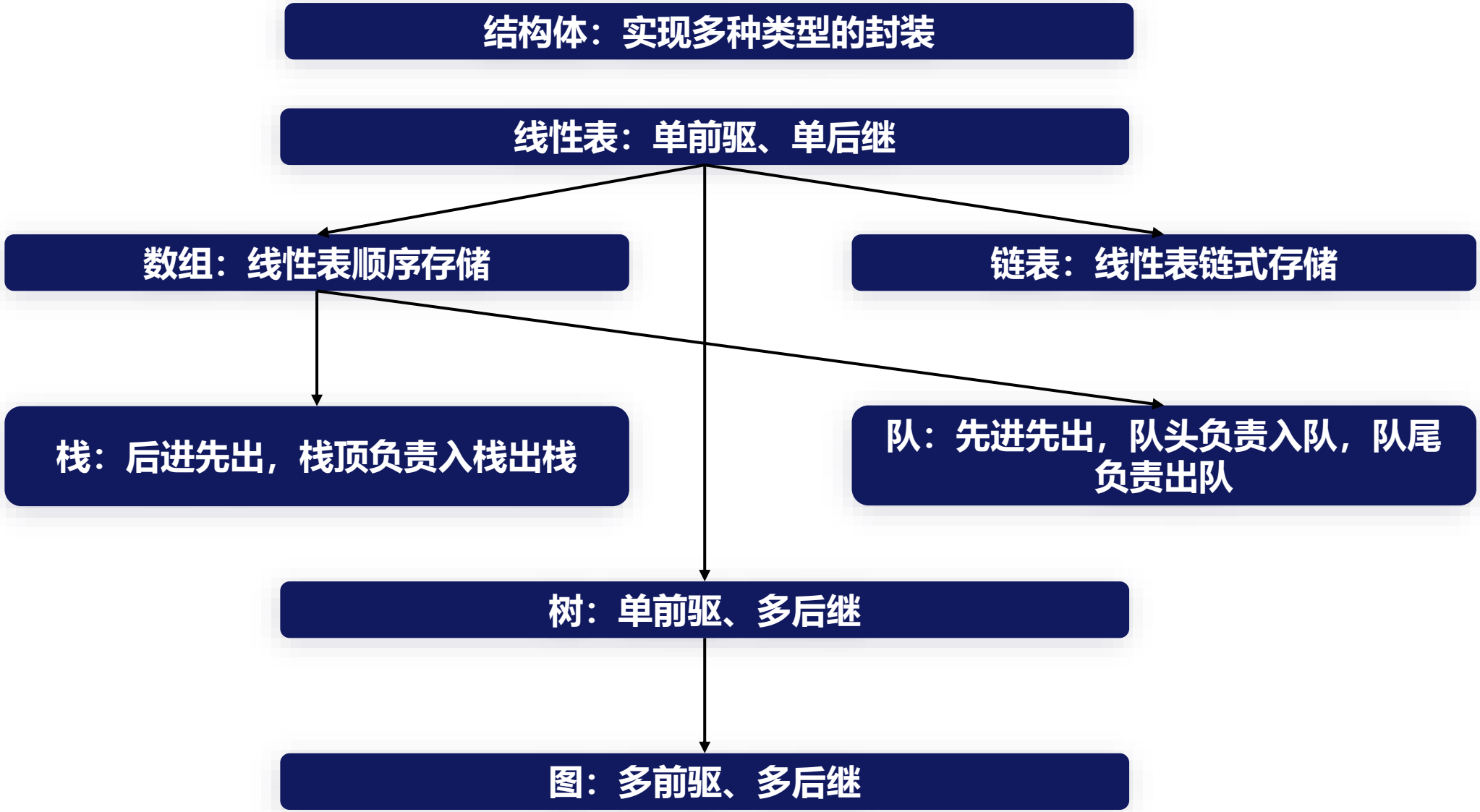
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 一般设定为一个比较大的素数
const int mod = 9007;
typedef struct node  node;
typedef struct node* nptr;

struct node {
    char *str;
    int freq;
    nptr nxt;
};
```

7 – 排序和查找

常考性质

排序算法	平均情况	最好情况	最坏情况	辅助空间	稳定性
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
希尔排序	$O(nlogn)$~$O(n^2)$	$O(n^{1.3})$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(1)$	不稳定
二路归并排序	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(n)$	稳定
快速排序	$O(nlogn)$	$O(nlogn)$	$O(n^2)$	$O(logn)$~$O(n)$	不稳定

8 – 总结


结构体：实现多种类型的封装
线性表：单前驱、单后继
数组：线性表顺序存储
链表：线性表链式存储
栈：后进先出，栈顶负责入栈出栈
队：先进先出，队头负责入队，队尾负责出队
树：单前驱、多后继
图：多前驱、多后继

Part 2: 题目讲解

选填与编程

- 选填主要有两种，一种侧重于考察某方法（如 Huffman 树权值，给定中序后序求前序），另一种侧重于考察对概念的理解（如哈希冲突的概念，排序的稳定性）

- 编程题第一题一般是结构体排序/字符串排序（排序仅为目的，不会限制手段）；第二题是栈和队的应用；第三题今年考图（感觉大概率会考最短路问题）

题目讲解

下列语句中，能正确进行字符串赋值的是 。

A. char* sp; *sp="BUAA" ; B. char s[10]; s="BUAA";

C. char s[10]; *s="BUAA"; D. char *sp="BUAA";

指针与数组名完全等价吗?

下面三种写法的空间分配情况：

```
char s1[] = "BUAA";
char s2[10] = "BUAA";
char *s3 = "BUAA";
```

题目讲解

已知二叉树的中序序列为：BADCE，后序序列为：BDECA，则其前序序列为：_____

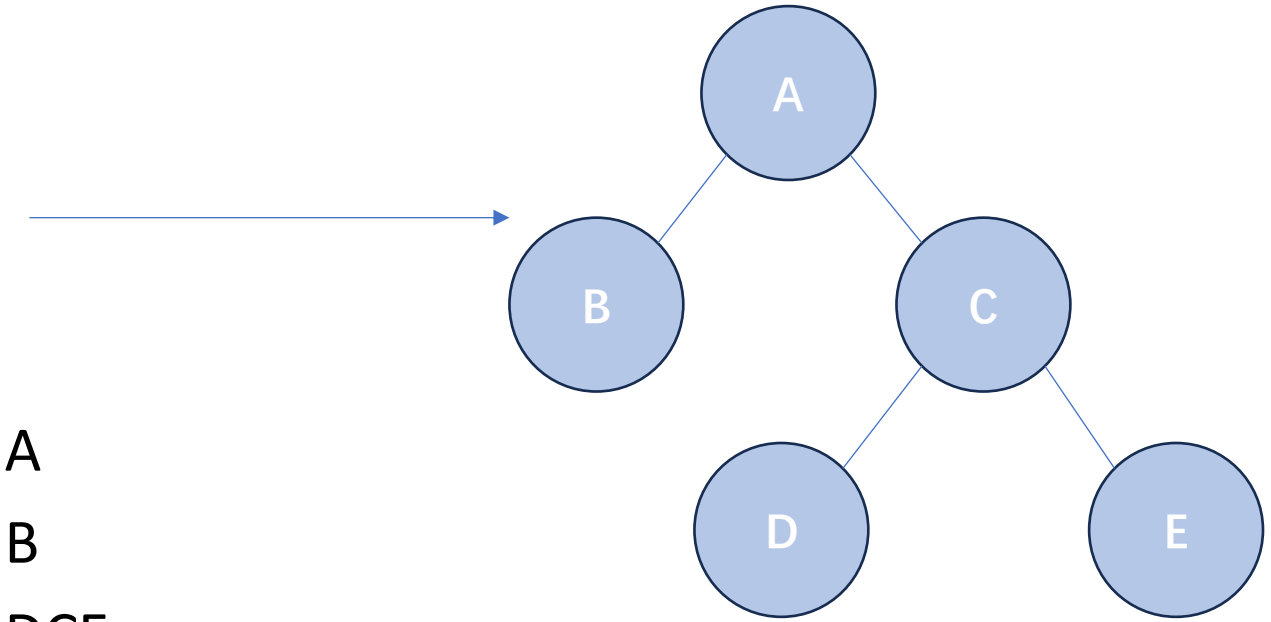
A. ADBEC　　B. DECAB　　C. DEBAC　　D. ABCDE

推断二叉树

根据后序遍历：左子树 -> 右子树 -> 根的访问方式，能知道什么？

后序：BDECA

中序：BADCE



- 根据后序性质发现根是 A
- 根据中序性质左子树是 B
- 根据中序性质右子树是 DCE
- 递归处理

题目讲解

编程题1. 汉明距离

【问题描述】

在信息论中，两个等长字符串之间的汉明距离（Harmming Distance）是指两个字符串中对应位置的不同字符的个数（区分大小写）。换句话说，它就是将一个字符串变换成另外一个字符串所需要替换的字符个数。例如：10110101与11011101的汉明距离为3，roses与cotes的汉明距离为2。从标准输入中输入一组等长字符串，以第一个字符串为基准，统计其与其它字符串的汉明距离，并按汉明距离由小至大输出字符串对及其汉明距离。

```
5
roses
cotes
Roses
coset
rotes
```

【样例输出】

```
roses Roses 1
roses rotes 1
roses coset 2
roses cotes 2
```

编程题一般以排序作为目的，并不要求排序方法

qsort 函数需要接受四个参数，依次是：

- 要排序数组的起始地址
- 要排序元素的个数
- 每个元素的大小（每个元素占多少字节）
- 排序规则对应的函数

```
qsort(a, n, sizeof(a[0]), cmp);
```

题目讲解


```
int cmp(const void *a, const void *b) {
    ...
}
```

- cmp 返回负数时：表示 a 指向的元素比 b 指向的元素小（逻辑上在前）
- cmp 返回 0 时，表示两个指针指向的元素一样大，谁在前谁在后无所谓（由于快排是不稳定排序，所以它们可能交换位置）
- cmp 返回正数时，表示 a 指向的元素比 b 指向的元素大

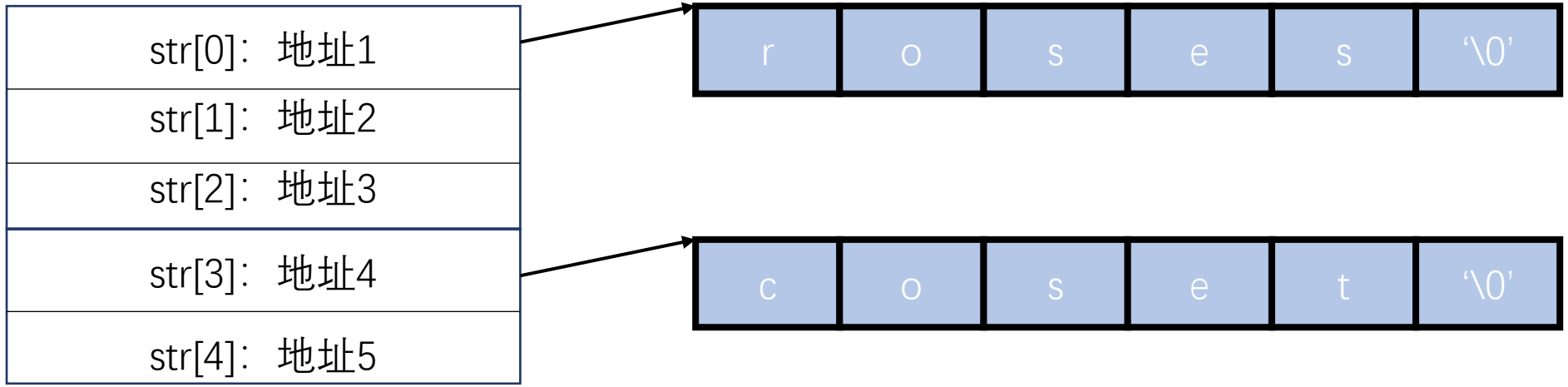
本题的实现

```
int cmp(const void *a, const void *b) {
    // 在存储字符串时，我们使用二维数组
    char *x = (char *)a;
    char *y = (char *)b;
    int dis_a = dis(x), dis_b = dis(y);
    if(dis_a != dis_b)
        return dis_a - dis_b;
    return strcmp(x, y);
}
```

题目讲解

字符串存储有两种方式

指针数组存字符串：



字符数组存字符串：

r	o	s	e	s	‘\0’
c	o	t	e	s	‘\0’
R	o	s	e	s	‘\0’

题目讲解

对字符串排序：两种方式

```
int cmp(const void *a, const void *b) {
    char *x = (char *)a;
    char *y = (char *)b;
    int dis_a = dis(x), dis_b = dis(y);
    if(dis_a != dis_b)
        return dis_a - dis_b;
    return strcmp(x, y);
}

int main() {
    char str[][6] = {"roses", "cotes", "Roses", "coset", "rotes"};
    t = str;
    qsort(str + 1, 4, sizeof(str[0]), cmp);

    for(int i = 0; i < 5; i++)
        puts(str[i]);
    return 0;
}
```

字符数组的方式

后面四个地址处存的字符串内容发生交换

c	o	t	e	s	'\0'
R	o	s	e	s	'\0'
c	o	s	e	t	'\0'
r	o	t	e	s	'\0'

R	o	s	e	s	'\0'
r	o	t	e	s	'\0'
c	o	s	e	t	'\0'
c	o	t	e	s	'\0'

题目讲解


```
int cmp(const void *a, const void *b) {
    char *x = *(char **)a;
    char *y = *(char **)b;
    int dis_a = dis(x), dis_b = dis(y);
    if(dis_a != dis_b)
        return dis_a - dis_b;
    return strcmp(x, y);
}

int main() {
    char *str[] = {"roses", "cotes", "Roses", "coset", "rotes"};
    qsort(str + 1, 4, sizeof(str[0]), cmp);

    for(int i = 0; i < 5; i++)
        puts(str[i]);
    return 0;
}
```

指针数组的方式

实际上，五个地址处存储的内容没有任何变化

注意实际写代码时，要自己读入，自己为指针分配合法空间

str[1]：地址1
str[2]：地址2
str[3]：地址3
str[4]：地址4

str[1]：地址2
str[2]：地址4
str[3]：地址3
str[4]：地址1

Part 3: 考 试 技 巧

利用编程巧解填空题

以下与 int *q[5]; 等价的定义语句是

(A) int q[5] ;
(B) int *q ;
(C) int *(q[5]) ;
(D) int (*q)[5] ;

指针数组与数组指针：

法一：用 () 来定性：(*p) 是指针；(p[]) 是数组

法二：数组指针是**指针**，同一环境下指针都一样大

```
11  int main() {
12      printf("%llu\n", sizeof(char *));
13      int *q1[5];
14      int (*q2)[5];
15      int *(q3[5]);
16      printf("%llu %llu %llu\n", sizeof(q1), sizeof(q2), sizeof(q3));
17      return 0;
```

问题　输出　调试控制台　终端　端口

```
PS D:\code> cd "d:\code\" ; if ($?) { gcc temp.c -o temp } ; if ($?) { .\temp }
8
40 8 40
```

谁是数组**指针**一目了然!

利用编程巧解填空题

对于判断一些合法写法的题，可以直接在本地验证

若有以下说明和语句,则下面表达式中值为1002的是 D

```
struct student
{
    int age;
    int num;
};
struct student stu[3] = {{1001, 20}, {1002, 19}, {1003, 21}};
struct student *p;
p = stu;
```

(A) (p++)->num
(B) (p++)->age
(C) (*p).num
(D) (*++p).age

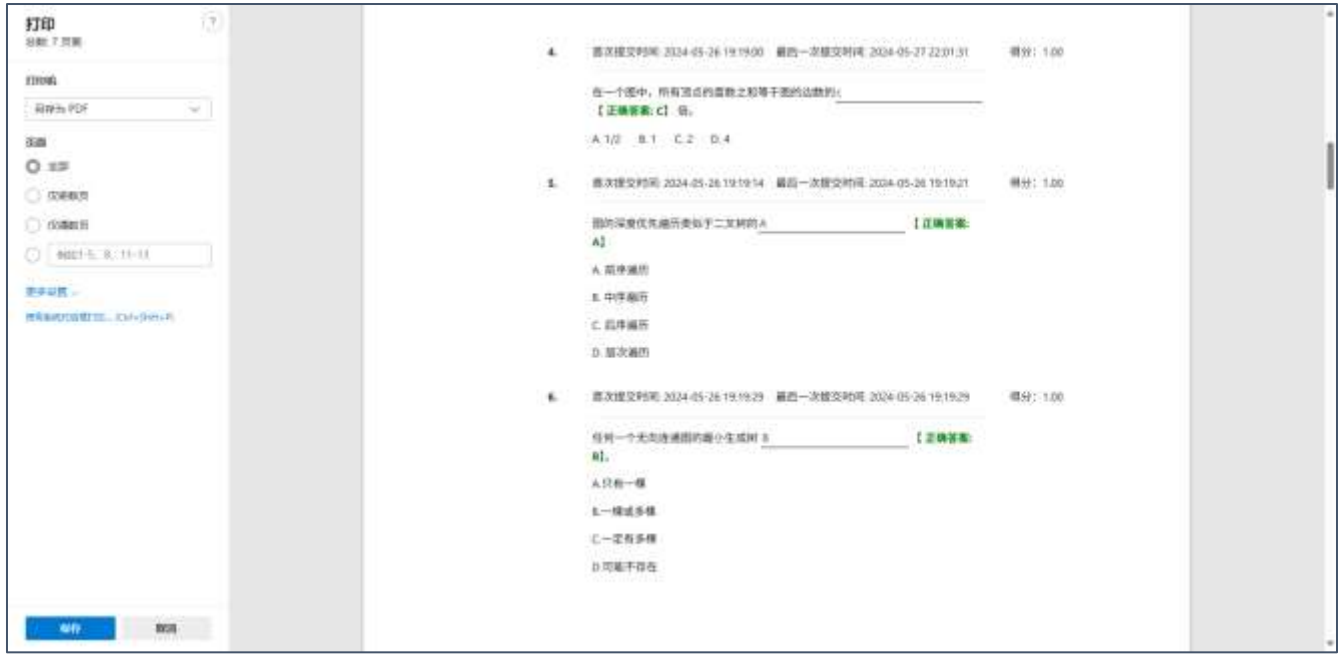
3. 若有以下说明和语句:

```
struct student{
    int age;
    int no;
};

struct student std, *p;
p = &std;
```

则以下对结构变量 std 中成员 no 的引用方式正确的是

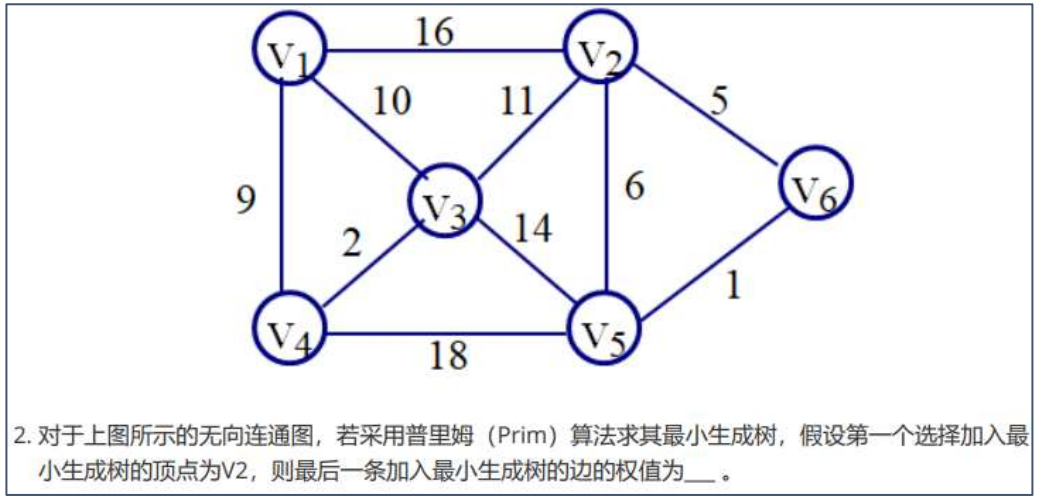
A. std->no B. p->no C. (*std).no D. *p.no

用 Ctrl + P 打印，然后就能在打印预览界面复制了



利用编程巧解填空题

对于一些稍涉及算法的题，实在不会可以提前准备好模板

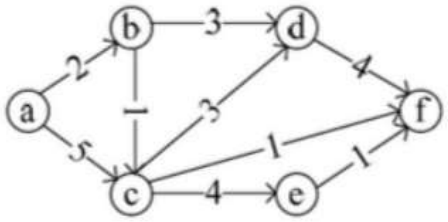


2. 对于上图所示的无向连通图，若采用普里姆（Prim）算法求其最小生成树，假设第一个选择加入最小生成树的顶点为V2，则最后一条加入最小生成树的边的权值为___ 。

1. 已知二叉树的前序序列为：ABDFGCEHJIK，中序序列为：BFDGAJHEIKC，则其后序序列为：

A. KIJHECGFDBA B. GDFBCKIEHJA C. FGDBJHKIECA D. JHIKEFFGDBA

2. 一有向带权图如下图所示，若采用迪杰斯特拉（Dijkstra）算法求源点a到其他各顶点的最短路径，得到的第一条最短路径的目标顶点是b，后续得到的其余各最短路径的目标顶点依次是。



A. c,d,e,f B. c,e,d,f C. c,f,e,d D. c,f,d,e

目录

- 栈队
  - 计算栈的最小容量
  - 判断是否是栈的合法输出序列
  - 中缀表达式转后缀表达式
  - 给定入栈出栈顺序求操作串
- 二叉树
  - 二叉查找树查找某元素的比较次数
  - 中序后序求先序（中缀后缀求前缀）
  - 中序前序求后序（中缀前缀求后缀）
  - 前序中序后序遍历完全二叉树的结果
  - 哈夫曼树的带权路径长度
- 排序与查找
  - 字符串排序-二维数组实现
  - 字符串排序-指针实现
  - 在大顶堆尾部插入元素调整过程的比较次数
  - 折半查找某元素的比较下标
- 图
  - Prim求最小生成树
  - Kruskal求最小生成树
  - 拓扑排序
  - 迪杰斯特拉求单源最短路

以简代繁

数据结构题目的数据量一般不会涉及由于算法效率不高的 TLE，甚至近几年期末第一道编程题排序的数据规模都是 100 个元素。

如果你真的不能理解 qsort，实在不会怎么写；或者考场上突然发现自己写的好像运行不了（比如字符串排序里，用指针数组存储字符串和用二维数组存储字符串比较函数写法是不同的，以前有同学一直都没有意识到，考试出了问题）。那你就没必要纠结于用 qsort。

还比如如果最后一道编程题真考了最短路，你不会写 Dijkstra ，你可以找一下 Floyd 算法的模板记一下，只有简单的三行：

```
for (k = 1; k <= n; k++)
    for (x = 1; x <= n; x++)
        for (y = 1; y <= n; y++)
            f[x][y] = min(f[x][y], f[x][k] + f[k][y]);
```

虽然这个算法求的是任意两点之间的最短路，相较于 Dijkstra 堆实现的 O(mlogm) 慢了许多，但好写的不是一点半点。

利用输出或者提前返回的方式发现 TLE 或者 REG 的位置

如果你的代码在本地运行某组数据时发生了 TLE（一直不运行结束） 或者 REG（运行结束但是没有正常退出，没有输出结果）

```
}
else if (op == 1)
{
    int flag = 0;
    int seek = rear;
    exit(0);
    while (seek <= front)
    {
        s = time_queue[seek].time;
        p = time_queue[seek].price;
```

```
else if (op == 1)
{
    int flag = 0;
    int seek = rear;
    puts("here");
    while (seek <= front)
    {
```

尝试使用 exit(0) 或 puts(“here”)，如果没能正常退出程序（或打印输出），说明出错代码在我们插入的代码之前，否则说明在我们写的代码之后。以此缩小出错代码的范围，最后找到出错的到底是哪一句代码。

如果想检验是否是循环内某次循环出问题，可以在每次循环开始和结束时分别输出不同信息（可以添加一些有意义的信息，比如输出相关的变量值），如果没能配对说明在该次循环的处理过程中出错了；如果一直输出说明死循环了。

利用提前返回的方式发现 TLE 或者 REG 的位置


```
}
else if (op == 1)
{
    int flag = 0;
    int seek = rear;
    exit(0);
    while (seek <= front)
    {
        s = time_queue[seek].time;
        p = time_queue[seek].price;
```

如果你的代码在本地运行了几个数组发现正常，结果交上去 REG 或者 TLE 了，那么你仍然可以用提前返回的方式判断出错代码的位置。

利用评测机的返回结果，什么时候 TLE 或者 REG 变成 WA 了，说明出错代码在插入代码之后，以此缩小出错代码的位置，锁定出错代码的范围。

可以考虑一些特殊情况，比如边界情况（如空闲空间申请一题中把所有块都给申请没了）

如果 WA 了

自己多造几组数据，不能光用眼睛看，自己瞪眼看代码是很难发现问题的。

发现了输出不符合预期的数字后，通过编译器自带的调试功能或者打印输出的方式（探针法），观察各个变量的值，找到不符合预期的变量值，并思考这个不符合预期的值是怎么来的。

```
else if (op == 1)
{
    int flag = 0;
    int seek = rear;
    printf("%d %d %d \n", seek, front, rear);
    while (seek <= front)
    {
        s = time_queue[seek].time;
        p = time_queue[seek].price;
        if (t <= s + 45 && price <= p && time_queue[seek].
```

自己多造几组数据，不能光用眼睛看，自己瞪眼看代码是很难发现问题的。

如果你用的编译器是 devcpp 或者小熊猫的话，有时候这俩软件的调试功能会有 bug（而且经常发生），建议使用探针法。

Part 4: 复 习 建 议

知识点分类

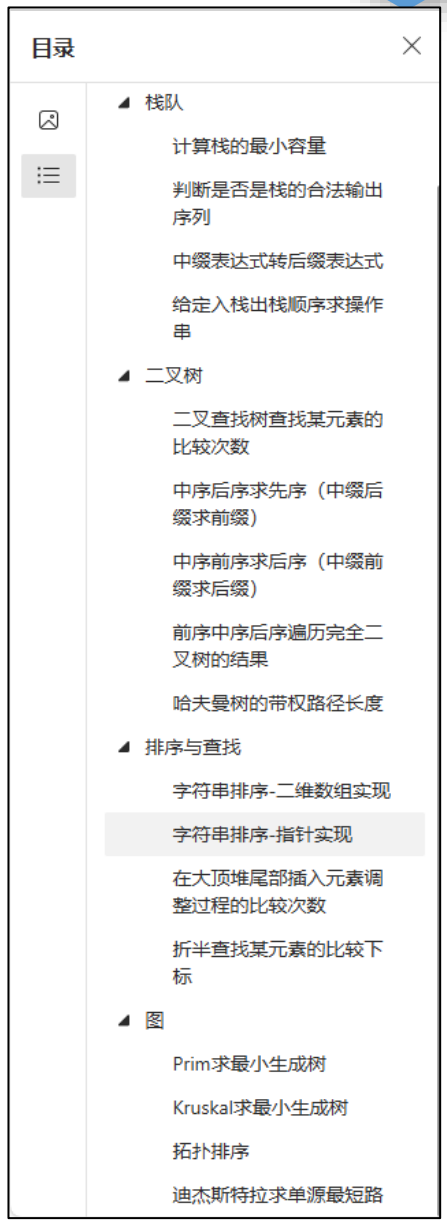
在复习的过程中，一定要明确，这道题知识点的定位一般是什么，没必要所有知识点都要掌握

- 编程题大概率会考的知识点（一定要掌握）：如结构体排序、字符串排序、最短路算法等
- 选填一般会考的小算法（基本上要学会手推，再不济存个模板）：比如给入栈出栈序列求栈的最小容量，中缀表达式转后缀，推断二叉树，Huffman 算法，最小生成树等
- 选填可能会考的概念性知识点（要保证自己能做上）：比如结构类型、结构成员、结构变量的区分；哈希冲突的判断；循环链表的特性等。
- 一些要记的知识点，除了记就不可能考别的了（要保证自己记住了，或者归纳整理了，到时候能查到）：如 B- 树的性质，稳定排序不稳定排序，常用数据结构常用操作的时间复杂度等。
- 纯不会考的知识点（99% 的情况下不可能考，为了备考的话建议直接跳过）：比如希尔排序，平衡树等。

BUAA DS 仓库

Github BUAA DS，目前已经 120 个 star

如果你作业题还没做完，或者有的题还没吃透，可以看一下仓库上的题解。同时仓库上还整理了一份选填算法的板子（如果你觉得不把握的话可以存一份，建议要是存的话考前一天晚上再去存一份看看，我这几天会更新）以及往年题的参考代码



往年题

往年的编程题尽可能都自己敲一敲，考前一定要多写写代码，不能只看、只想，一定要动手实践

一定要好好做一下结构体排序、字符串排序（前面这俩排序部分的千万不能错）、单源最短路这种题。当然图相关的其它算法最好也备个板子，但是考编程题的几率不高

致谢

感谢聆听，有问题随时联系！

预祝大家期末考试顺利