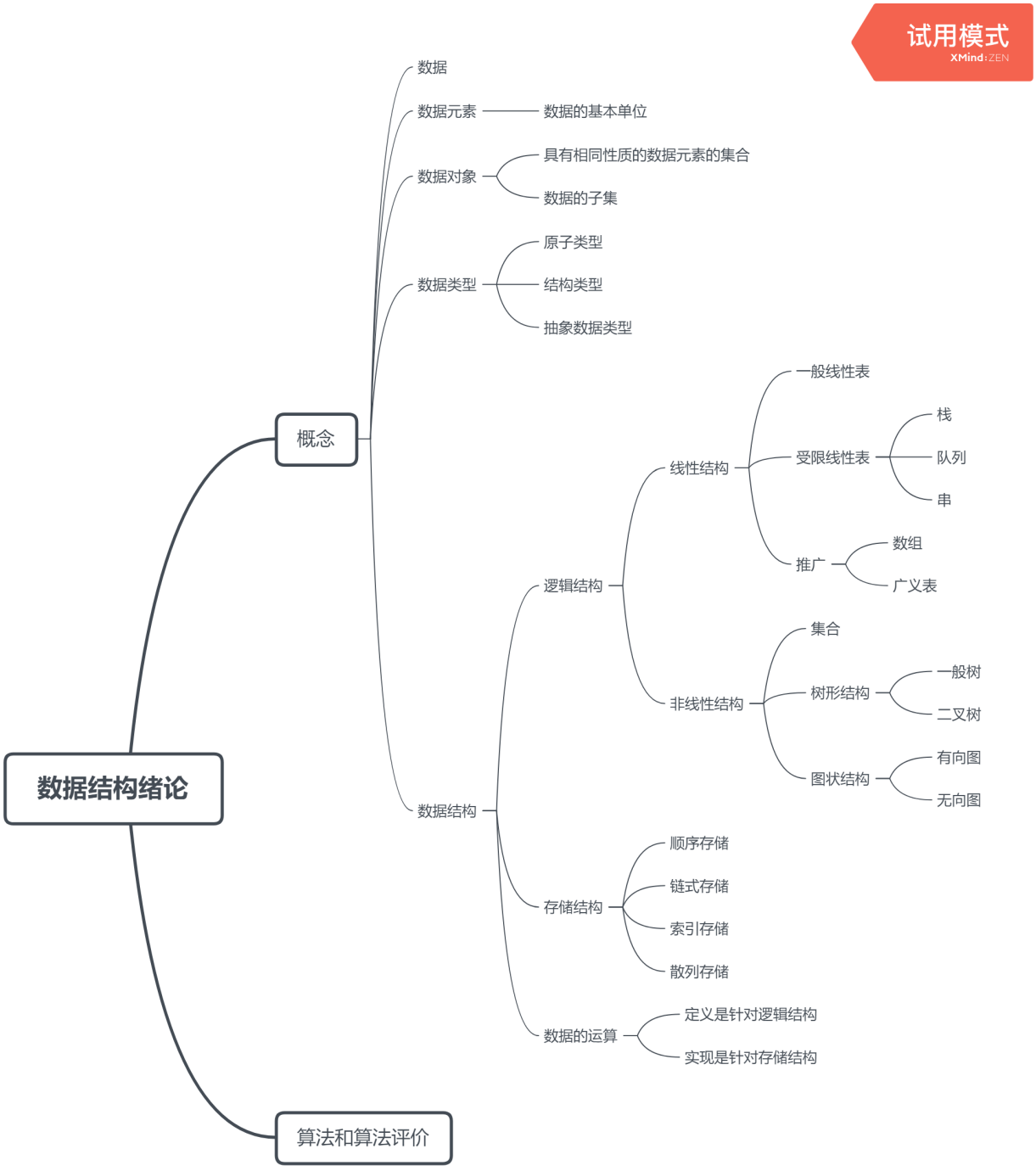
数据结构

一.绪论



1.1 基本概念

> 【2021 p3 1】
>
> 可以用（）定义一个完整的数据结构

> A.数据元素 B.数据对象 C.数据关系 D.抽象数据类型

抽象数据类型描述了数据的逻辑结构和抽象运算，通常使用（数据对象，数据关系，基本操作集）这样的三元组来表示

D

> 【2021 p3 2】
>
> 以下数据结构中，（）是非线性数据结构
>
> A.树 B.字符串 C.队列 D.栈

树是树形结构，选A

> 【2021 p3 3】
>
> 以下属于逻辑结构的是（）
>
> A.顺序表 B.哈希表 C.有序表 D.单链表

顺序表和哈希表以及单链表，既描述逻辑结构，又描述存储结构和数据运算

有序表是指关键字有序的线性表，仅描述元素之间的逻辑关系，既可以链式存储，也可以顺序存储

选C

> 【2021 p3 4】
>
> 以下与存储结构无关的术语是（）
>
> A.循环队列 B.链表 C.哈希表 D.栈

栈是一种逻辑结构，无法表示如何存储

> 【2021 p4 5】
>
> 以下关于数据结构的说法中，正确的是（）
>
> A. 数据的逻辑结构独立于其存储结构
>
> B. 数据的存储结构独立于其逻辑结构
>
> C. 数据的逻辑结构唯一决定了其存储结构
>
> D. 数据结构仅由其逻辑结构和存储结构决定

数据的逻辑结构是从面向实际问题的角度出发，只采用抽象表达方式，独立于存储结构；

数据的存储结构是逻辑结构在计算机上的映射，不能独立于逻辑结构而存在；

数据结构的三个要素：逻辑结构、存储结构、数据运算缺一不可

> 【2021 p4 6】
>
> 在存储数据时，通常不仅要存储各数据元素的值，还要存储（数据元素之间的关系）

> 【2021 p4 7】
>
> 链式存储设计时，结点内的存储单元地址（一定连续）

不同结点可以不连续，但是同一结点内一定连续的。

两种不同的数据结构，他们的逻辑结构和物理结构可能完全相同。比如二叉树和二叉排序树，

二叉树排序树可以采用二叉树的逻辑表示方式和存储方式，前者通常表示层次关系，后者通常用于排序和查找。

以查找为例，二叉树的时间复杂度为$O(n)$，二叉排序树的时间复杂度为$O(log_2 n)$

1.2 算法评价

> 【2021 王道 p7 3】
>
> 以下算法的时间复杂度为（）

```
void fun(int n) {
 int i = 1;
 while(i <= n)
     i = i * 2;
}
```

设`i=i*2`语句运行了t次，$2^t = n$，所以$t = log_2 n$

时间复杂度为$O(log_2 n)$

> 【2021 王道 p7 7】
>
> 【2014 408统考真题】
>
> 下列程序段的时间复杂度是（）

```
count = 0;
for(k = 1;  k <= n; k *= 2)
 for(j = 1; j <= n; j++)
     count++;
```

我们假设n为2的指数次幂2^k，则`for(j = 1; j <= n; j++)`语句执行的次数为t，则有$2^t = n$，则$t = log_2 n$

而内部`count++`语句在每个循环执行n次，共执行$nlog_2 n$次

所以时间复杂度为$O(nlog_2 n)$

> 【王道 2021 p8 12】
>
> 下面说法中，错误的是（）
>
> I.算法原地工作的含义是指不需要任何额外的辅助空间
>
> II.在相同规模n下，复杂度为O(n)的算法在时间上总是由于复杂度为$O(2^n)$

> III.所谓时间复杂度，是指在最坏的情况下估算算法执行时间的一个上界
>
> IV.同一个算法，实现的语言越高级，执行效率越低

算法原地工作指的是算法所需要的辅助空间为常量

所以错误的只有I

> 【王道2021 p8 2.1】
>
> 算法所需时间由下列递归方程表示，试求出该算法的时间复杂度的级别
>
> $$T=\begin{cases}1, & n=1\\ 2T(n/2)+n, & n>1\end{cases}$$
>
> 式中，n是问题规模，为简单起见，设n是2的整数次幂

▲记住，这个是所需时间的表达式，不是某个函数的递归表达式，所以直接算结果就行了

设n=2^k，T=2T(2^{k-1})+2^k

=2(2T(2^{k-2})+2^{k-1}) + 2^k

=2(2(2T(2^{k-3})+2^{k-2})+2^{k-1})+2^k

...

=2^iT(2^{k-i})+$i\times 2^k$

最后等于$2^kT(1)+k2^k=2^k+k2^k=2^k(k+1)$

$k=log_2n$

所以原式等于$n(log_2n+1)$

所以时间复杂度为$O(nlog_2n)$

> 该程序段的时间复杂度为？

```
for(int i = 1; i <=n; i++)
 for(int j = 1; j <= i; j++)
     for(int k = 1; k <= j; k++)
         x++;
```

将该程序段的时间复杂度计算公式写出来，即$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1$

$\sum_{j=1}^{i}\sum_{k=1}^{j}1=\frac{i(i+1)}{2}$，即原式为
$\sum_{i=1}^{n}\frac{i(i+1)}{2}=\frac{1}{2}+\frac{2}{2}+\cdots+\frac{n}{2}+\frac{1^2}{2}+\frac{2^2}{2}+\cdots+\frac{n^2}{2}=\frac{n(n+1)}{4}+\frac{n(n+1)(2n+1)}{12}$

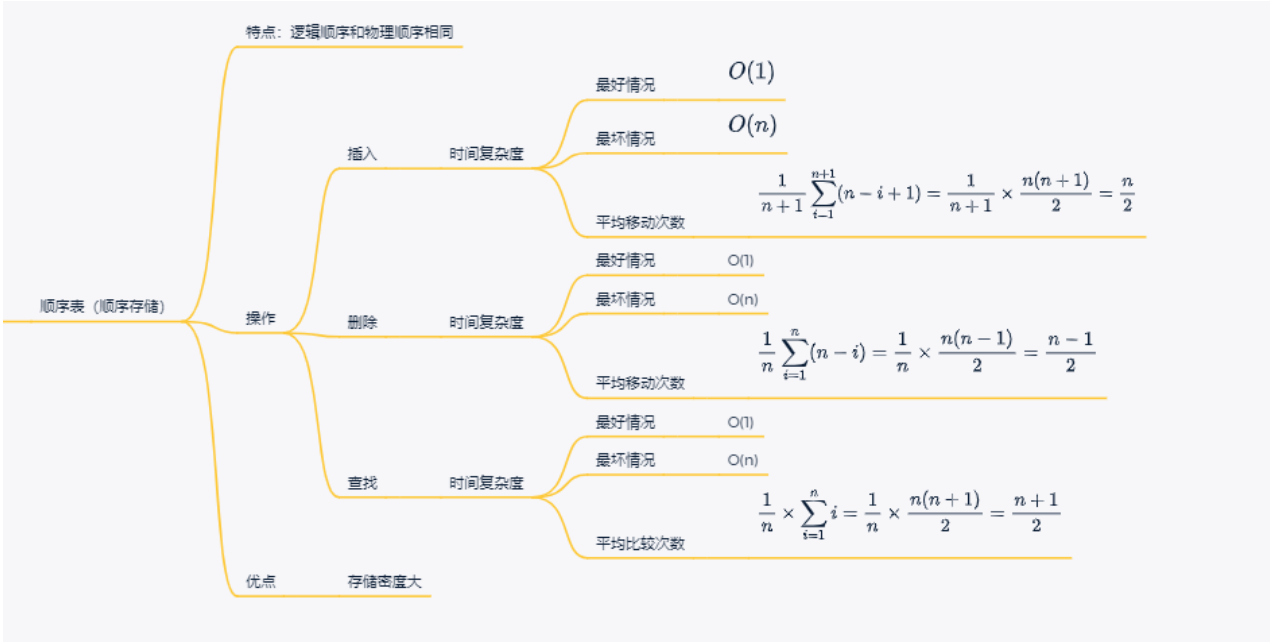
> 解：利用恒等式$(n+1)^3=n^3+3n^2+3n+1$,可以得到：
> $(n+1)^3-n^3=3n^2+3n+1$,
> $n^3-(n-1)^3=3(n-1)^2+3(n-1)+1$
> ……
> $3^3-2^3=3\times(2^2)+3\times2+1$
> $2^3-1^3=3\times(1^2)+3\times1+1$.
> 把这n个等式两端分别相加，得：
> $(n+1)^3-1=3(1^2+2^2+3^2+....+n^2)+3(1+2+3+...+n)+n$,
> 由于$1+2+3+...+n=(n+1)n/2$,
> 代入上式得：
> $n^3+3n^2+3n=3(1^2+2^2+3^2+....+n^2)+3(n+1)n/2+n$
> 整理后得：
> $1^2+2^2+3^2+....+n^2=n(n+1)(2n+1)/6$

抓大头，找到最高次幂应该为$\frac{2n^3}{12}=\frac{n^3}{6}$

所以最后的时间复杂度应该为$O(n^3)$

二.线性表

2.1 顺序表




```c
#define MaxSize 50
typedef struct {
    int data[MaxSize];
    int length;
}SqList;

typedef struct {
    int* data;
    int MaxSize, length;
} SeqList;

int listInsert(SqList *L, int i ,int e) {
    if(i < 0 || i > L->length)
        return 0;
    if(L->length >= MaxSize) {
        printf("Maxsize, can't insert\n");
        return 0;
    }
    for (int j = L->length; j > i; j--)
        L->data[j] = L->data[j - 1];
    L->data[i] = e;
    L->length++;
    return 1;
}

int listDelete(SqList *L, int i, int *e) {
    if(i < 0 || i >= L->length)
        return 0;
    *e = L->data[i];
    for (int j = i; j < L->length - 1; ++j)
        L->data[j] = L->data[j+1];
    L->length--;
    return 1;
}
```

```
int locateElement(SqList *L, int e) {
    for (int i = 0; i < L->length; ++i)
        if(L->data[i] == e)
            return i;
    return -1;
}
```

> 【王道 2021 p14 2】
>
> 以下（）是一个线性表
>
> A.由n个实数组成的集合 B.由100个字符组成的序列 C.所有整数组成的序列 D.邻接表

A选项无先后顺序，C无穷，不符合有限性 D是存储结构

> 【王道2021 p18 4】
>
> 若线性表最常用的操作是存取第i个元素及其前驱和后继元素的值，为了提高效率，应该采用（）的存储方式
>
> A.单链表
>
> B.双向链表
>
> C.单循环链表
>
> D.顺序表

直接使用顺序表即可，因为其他链表都必须从首个位置开始依次查询，找到目的位置，所以效率低

> 【王道2021 p19 11】
>
> 顺序表的插入算法中，当n个空间已满时，可再申请增加分配m个空间，若申请失败，则说明系统中没有()可分配的空间

> A.m个 B.m个连续 C.n+m个 D.n+m个连续

选D，因为**malloc**是申请**n+m**个新的连续空间，然后将线性表原来的**n**个元素复制到新申请的**n+m**个连续的存储空间中

算法题

1

> 1. 从顺序表中删除具有最小值的元素（假设唯一）并由函数返回被删元素的值。空出的位置由最后一个元素填补，若顺序表为空，则显示出错信息并退出运行。

```
int del_min(SqList* L, int* res) {
    if(L->length == 0) return 0;
    int min = L->data[0];
    int min_index = 0;
    for (int i = 1; i < L->length; ++i) {
       if (L->data[i] < min) {
           min = L->data[i];
           min_index = i;
       }
    }
    *res = L->data[min_index];
    L->data[min_index] = L->data[L->length - 1];
    L->length--;
    return 1;
}
```

时间复杂度为O(n)

2

> 2. 设计一个高效算法，将顺序表 L 的所有元素逆置，要求算法的空间复杂度为 $O(1)$。

```
int reverse_list(SqList *L) {
    int temp;
    for (int i = 0; i < L->length / 2; ++i) {
       temp = L->data[i];
       L->data[i] = L->data[L->length - 1 - i];//对称的两个位置和应该为n-1，比如第一和最后一个位置分别为
       //0和n-1，那么他们的和为n-1，即L->length - 1
       L->data[L->length - 1 - i] = temp;
    }
    return 1;
}
```

时间复杂度为O(n)

3▲

3. 对长度为 n 的顺序表 L，编写一个时间复杂度为 $O(n)$、空间复杂度为 $O(1)$的算法，该算法删除线性表中所有值为 x 的数据元素。

方法一：

```
int del_all_x(SqList* L, int x) {
    if(L->length == 0) return 0;
    int k = 0;
    for (int i = 0; i < L->length; ++i) {
       if (L->data[i] != x)
           L->data[k++] = L->data[i];
    }
    L->length = k;
    return 1;
}
```

方法二：

```
int del_all_x2(SqList* L, int x) {
    if(L->length == 0) return 0;
    int i = 0;
    int k = 0;
    while(i < L->length) {
        if (L->data[i] == x)
            k++;
        else
            L->data[i - k] = L->data[i];
        i++;
    }
```

```
    L->length -= k;
    return 1;
}
```

方法三：

双指针做法

```
int del_all_x3(SqList *L, int x) {
    int h = 0;
    int t = L->length - 1;
    while(h < t) {
        while(h < t && L->data[h] != x)
            h++;
        while(t > h && L->data[t] == x)
            t--;
        int temp = L->data[h];
        L->data[h] = L->data[t];
        L->data[t] = temp;
    }
    L->length = h;
    return 1;
}
```

例如1 6 8 2 7 4 2 2

```
*****************************第一轮
h            t
↓            ↓
1 6 8 2 7 4 2 2
------------------------------
  h          t
  ↓          ↓
1 6 8 2 7 4 2 2
------------------------------
    h        t
    ↓        ↓
1 6 8 2 7 4 2 2
------------------------------
      h      t
      ↓      ↓
1 6 8 2 7 4 2 2
------------------------------
      h    t
      ↓    ↓
1 6 8 2 7 4 2 2
------------------------------
      h  t
      ↓  ↓
```

```
1 6 8 2 7 4 2 2
-----------------------------
      h   t                 交换
      ↓   ↓
1 6 8 4 7 2 2 2
**************************第二轮
      h   t
      ↓   ↓
1 6 8 4 7 2 2 2
-----------------------------
        h t
        ↓ ↓
1 6 8 4 7 2 2 2
-----------------------------
          h                 两个指针重合，运行结束
          t
          ↓
1 6 8 4 7 2 2 2
```

这种做法会改变相对位置，比如7和4

4

4. 从有序顺序表中删除其值在给定值 s 与 t 之间（要求 s < t）的所有元素，若 s 或 t 不合理或顺序表为空，则显示出错信息并退出运行。

方法一：

```
/**
 * 删除数组中s与t之间的元素
 * @param L 有序顺序表
 * @param s
 * @param t
 * @return
 */
int del_s_t(SqList* L, int s, int t) {
    int k = 0;//k记录在s,t之间的数的数量
    int j = 0;//结果指针
    if (s >= t || L->length == 0) {
        printf("参数不合理\n");
        return 0;
    }
    for (int i = 0; i < L->length; ++i) {
        if(L->data[i] >= s && L->data[i] <= t)
```

```
            k++;
        else
            L->data[j++] = L->data[i];
    }
    L->length -= k;
    return 1;
}
```

> 仿照第三题方法一的思想，就可以了

方法二

双指针思想，自认为不太简便

```
int del_s_t2(SqList* L, int s, int t) {
    int i = 0, j = L->length - 1;
    for (; i < L->length && L->data[i] < s; i++);
    if(i == L->length)
        return 0;//不存在s与t之间的数据
    for (; j >= 0 && L->data[j] > t; j--);
    j++;//因为j是从后往前找，找到的是最后一个符合s~t之间的元素，
    //加一后，是第一个大于t的元素
    while(j < L->length)
        L->data[i++] = L->data[j++];
    L->length = i;
    return 1;
}
```

5

> 5. 从顺序表中删除其值在给定值 s 与 t 之间（包含 s 和 t，要求 $s<t$）的所有元素，若 s 或 t 不合理或顺序表为空，则显示出错信息并退出运行。

与第四题方法一类似，不赘述

```
/**
 * 删除数组中s与t之间的元素
 * @param L 顺序表
 * @param s
 * @param t
 * @return
 */
int del_s_t(SqList* L, int s, int t) {
```

```
    int k = 0;//k记录在s,t之间的数的数量
    int j = 0;//结果指针
    if (s >= t || L->length == 0) {
        printf("参数不合理\n");
        return 0;
    }
    for (int i = 0; i < L->length; ++i) {
        if(L->data[i] >= s && L->data[i] <= t)
            k++;
        else
            L->data[j++] = L->data[i];
    }
    L->length -= k;
    return 1;
}
```

6

6. 从有序顺序表中删除所有其值重复的元素，使表中所有元素的值均不同。

```
/**
 * 删除有序数组中的重复元素
 * @param L 有序顺序表
 * @return
 */
int del_dup(SqList* L) {
    int i = 1;
    int k = 0;
    for (; i < L->length; i++)
        if(L->data[i] != L->data[k])
            L->data[++k] = L->data[i];
    L->length = k + 1;
    return 1;
}
```

7

7. 将两个有序顺序表合并为一个新的有序顺序表，并由函数返回结果顺序表。

```
int merge_list(SqList A, SqList B, SqList* C) {
    int i = 0, j = 0;
    int k = 0;
    C->length = 0;
```

```
    while(i < A.length && j < B.length) {
        if(A.data[i] < B.data[j]) {
            C->data[k++] = A.data[i++];
        } else {
            C->data[k++] = B.data[j++];
        }
        C->length++;
    }
    while(i < A.length) {
        C->data[k++] = A.data[i++];
        C->length++;
    }
    while(j < B.length) {
        C->data[k++] = B.data[j++];
        C->length++;
    }
    return 1;
}
```

8▲