1 绪论

数据: 客观描述事物属性的数, 字符及所有能被输入到计算机中并被程序识别和处理的符号集合

数据元素: 数据的基本单位, 作为一个整体进行考虑和处理. 根据业务确定具体构成.

数据对象: 具有相同属性的数据元素的集合. **不强调数据元素之间的关系**

数据类型: 1) 原子类型: 不可再分. 2) 结构类型: Struct. 3) 抽象数据类型 ADT: 数学化的语言, 定义数据的取值范围, 结构形式, 操作 (逻辑结构, 数据运算). 不关心存储结构, 与具体实现无关.

数据结构: 相互之间存在一种或多种特定**关系**的**数据元素**的集合. **三要素**: 逻辑结构, 存储结构, 数据的运算

1.1 数据元素三要素

1. 逻辑结构: 数据元素之间的逻辑关系, 与存储无关. 线性结构 (线性表); 非线性结构 (集合, 树, 图 …).

注 哈希表属于存储结构, 有序表 (有序线性表) 属于逻辑结构

1) 集合: 元素之间只有**同属于一个集合**的关系

2) 线性结构: **只存在一对一**关系, 除首末节点, 其余节点只有唯一前驱和唯一后继

3) 树形结构: **一对多**关系

4) 图状结构/网状结构: **多对多**关系

2. 存储结构: 也称为物理结构, 是数据结构在计算机中的表示 (映像). **影响**: (a) 存储空间分配的方便程度; (b) 对数据运算的速度. **主要有**: 顺序存储, 链式存储, 索引存储, 散列存储

1) 顺序存储: 逻辑上相邻, 物理上也相邻. 优点: 可以随机存取; 缺点: 只能使用一整块存储单元

2) 链式存储: 逻辑上相邻, 物理上不要求相邻. 使用指针表示元素之间的逻辑关系. 优点: 不会出现碎片现象; 缺点: 只能顺序存取; 指针占据额外存储空间

3) 索引存储: 建立附加的索引表. 索引表中的每项称为**索引项**. 一般形式: **(关键字, 地址)**. 优点: 检索速度快; 缺点: 索引表占据额外空间, 增删数据也要修改索引表, 耗费较多时间

4) 散列存储: 通过关键字直接计算出存储地址. 优点: 查增删效率高; 缺点: 较差的散列函数会造成冲突, 解决冲突会带来时间空间开销

3. 数据的运算: 运算的**定义**针对**逻辑结构**; **实现**针对**存储结构**.

注 不同的数据结构, 其逻辑结构, 存储结构不一定不同. EX. 链表实现队列和栈; 图的存储又邻接表, 邻接矩阵

注 存储数据时, 要存储数据的值和数据元素之间的关系. 数据的操作方法, 数据元素类型, 存取方法隐含在实现或逻辑结构中, 或可以通过上下文推断, 不需要显式存入.

1.2 算法

五个重要特征: 缺少任何一个都不是算法

1) 有穷性: 算法要在有穷步完成, 每一步要在有穷时间内完成.

2) 确定性: 每条指令要有确切含义, 无歧义. 相同输入得到相同输出.

3) 可行性: 能够通过代码实现.

4) 输入: 0 或多个输入

5) 输出: **1 或多个**输出

好的算法: 只要满足上述五个特征, 就算是算法

1) 正确性: 正确解决问题; 2) 可读性; 3) 健壮性: 对非法数据有反应或处理; 4) 高效率和低存储量需求: 时间空间复杂度低.

注 程序可以无穷, 只要不关掉.

确定性的无歧义 EX. 48, 76, 76 选择最大的元素 -> 同值时确定最大 -> 顺序前/后最大 (稳定排序)

1.2.1 时间复杂度

对于问题规模 n, 时间复杂度记为:

$$T(n) = O(f(n))$$

$$O(1) < O(\log_2^n) < O(n) < O(n\log_2^n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

1.2.2 空间复杂度

对于问题规模 n, **只考虑与问题规模 n 相关的额外空间/辅助空间**. 空间复杂度记为:

$$S(n) = O(f(n))$$

注 算法原地工作: 所需的辅助空间为 $O(1)$

1.2.3 空间复杂度计算

1. $S(n) = O$(递归深度)

```
Function fun(n):
   ...
   return fun(n-1)
```

2. $S(n) = n + (n-1) + \cdots + 1 = \frac{n(n+1)}{2} -> O(n^2)$

```
Function fun(n):
   ...
   int a[n]
   fun(n-1)
   return
```

1.2.4 时间复杂度计算

1. 循环: 作表

```
int sum = 0;
for (int i=1; i<n; i *= 2)
   for (int j=0; j<i; j++)
      sum++;
```

令基本运算 sum++的执行次数为 x, 有:

i	j	x
1	0	2^0
2^1	$0 \sim 1$	2^1
2^2	$0 \sim 2^2 - 1$	2^2
$\cdots$	$\cdots$	$\cdots$
2^c	$0 \sim 2^c - 1$	2^c

运行次数之和 $\sum x = \frac{1(1-2^c)}{1-2} = 2^c - 1$

$2^c < n < 2^{c+1}$

$c < \log_2^n < c + 1$

$\log_2^n - 1 < c < \log_2^n$

故: $\sum x < n - 1$

故: $T(n) = O(n)$

2. 循环主体中变量参与循环条件判断

```
// 1
int i {1};
while (i <= n)
   i *= 2;
// 2
int y {5}:
while ((y+1) * (y+1) < n)
   y++;
```

1. 令基本运算 i *= 1的执行次数为 x, 则 $2^x \leq n$, 解得 $x \leq \log_2^n$, 故 $T(n) = O(\log_2^n)$

2. 令基本运算 y++的执行次数为 x, 则 $y = x + 5$, $(x + 5 + 1)^2 < n$, 解得 $x < \sqrt{n} - 6$, 故 $T(n) = O(\sqrt{n})$

3. 递归: 递推求解, 可能需要使用数学归纳 EX.

$$T(n) = 1 + T(n-1) = 1 + 1 + T(n-2) = \cdots = n - 1 + T(1)$$

即 $T(n) = O(n)$