

什么是数据结构

2023年6月25日 15:00

解决问题方法的效率，与数据的组织形式有关（图书贮藏方式），与空间的利用效率有关（递归输出N以内自然数）以及与算法的巧妙程度有关（多项式求值）

数据对象在计算机中的组织方式：

- 1.逻辑结构（线型结构、树、图等）
- 2.物理存储结构（例如选择是用数组还是用链表等）

抽象数据类型：

抽象数据类型（Abstract Data Type）

- 数据类型
 - 数据对象集
 - 数据集合相关联的操作集
- 抽象：描述数据类型的方法不依赖于具体实现
 - 与存放数据的机器无关
 - 与数据存储的物理结构无关
 - 与实现操作的算法和编程语言均无关

只描述数据对象集和相关操作集“是什么”，并不涉及“如何做到”的问题

屏幕剪辑的捕获时间: 2023/7/10 18:00

例4: “矩阵”的抽象数据类型定义

- **类型名称:** 矩阵 (**Matrix**)
- **数据对象集:** 一个 $M \times N$ 的矩阵 $A_{M \times N} = (a_{ij})$ ($i=1, \dots, M; j=1, \dots, N$) 由 $M \times N$ 个三元组 $\langle a, i, j \rangle$ 构成, 其中 a 是矩阵元素的值, i 是元素所在的行号, j 是元素所在的列号。
- **操作集:** 对于任意矩阵 $A, B, C \in \text{Matrix}$, 以及整数 i, j, M, N
 - **Matrix Create(int M, int N):** 返回一个 $M \times N$ 的空矩阵;
 - **int GetMaxRow(Matrix A):** 返回矩阵 A 的总行数;
 - **int GetMaxCol(Matrix A):** 返回矩阵 A 的总列数;
 - **ElementType GetEntry(Matrix A, int i, int j):** 返回矩阵 A 的第 i 行、第 j 列的元素;
 - **Matrix Add(Matrix A, Matrix B):** 如果 A 和 B 的行、列数一致, 则返回矩阵 $C=A+B$, 否则返回错误标志;
 - **Matrix Multiply(Matrix A, Matrix B):** 如果 A 的列数等于 B 的行数, 则返回矩阵 $C=AB$, 否则返回错误标志;
 -

屏幕剪辑的捕获时间: 2023/7/10 18:02

抽象: 在上例中, ElementType体现了不关心矩阵中元素的类型, MatrixAdd体现了不关心用什么语言实现, 而定义矩阵则表示不关心这个矩阵用什么形式存储。有什么好处?

1.信息隐藏 (Encapsulation): 在 C++ 中, 可以将数据和操作数据的方法绑定在一起, 并使用访问修饰符 (如 private 和 protected) 来隐藏内部实现。这意味着可以改变对象内部的实现方式, 而不会影响到使用该对象的代码。

2.代码重用和模块化: 抽象数据类型可以创建高度模块化的代码, 因为每个数据类型都可以作为一个单独的模块, 可以独立开发、测试和优化。一旦定义了一个抽象数据类型, 就可以在许多不同的地方多次使用, 而不需要重新编写代码。

3.提高可维护性: 通过使用抽象数据类型, 可以将复杂问题分解为更小的、更容易管理的部分。这使得代码更容易理解和维护。

4.易于实现抽象和设计: 抽象数据类型允许程序员将注意力从数据的具体实现转移到数据的行为和交互上, 有助于实现更高层次的抽象和设计。

5.提高代码的稳定性和安全性: 由于内部实现的隐藏, 外部代码不能直接访问和修改对象的内部状态, 从而降低了出错的可能性。

什么是算法

2023年7月10日 18:11

算法的定义：

定义

■ 算法 (Algorithm)

- 一个有限指令集
- 接受一些输入（有些情况下不需要输入）
- 产生输出
- 一定在有限步骤之后终止
- 每一条指令必须
 - 有充分明确的目标，不可以有歧义
 - 计算机能处理的范围之内
 - 描述应不依赖于任何一种计算机语言以及具体的实现手段

屏幕剪辑的捕获时间: 2023/7/10 18:12

什么是好的算法？

什么是好的算法？

- **空间复杂度 $S(n)$** —— 根据算法写成的程序在执行时占用存储单元的长度。这个长度往往与输入数据的规模有关。空间复杂度过高的算法可能导致使用的内存超限，造成程序非正常中断。
- **时间复杂度 $T(n)$** —— 根据算法写成的程序在执行时耗费时间的长度。这个长度往往也与输入数据的规模有关。时间复杂度过高的低效算法可能导致我们在有生之年都等不到运行结果。

- 在分析一般算法的效率时，我们经常关注下面两种复杂度
 - 最坏情况复杂度 $T_{worst}(n)$
 - 平均复杂度 $T_{avg}(n)$

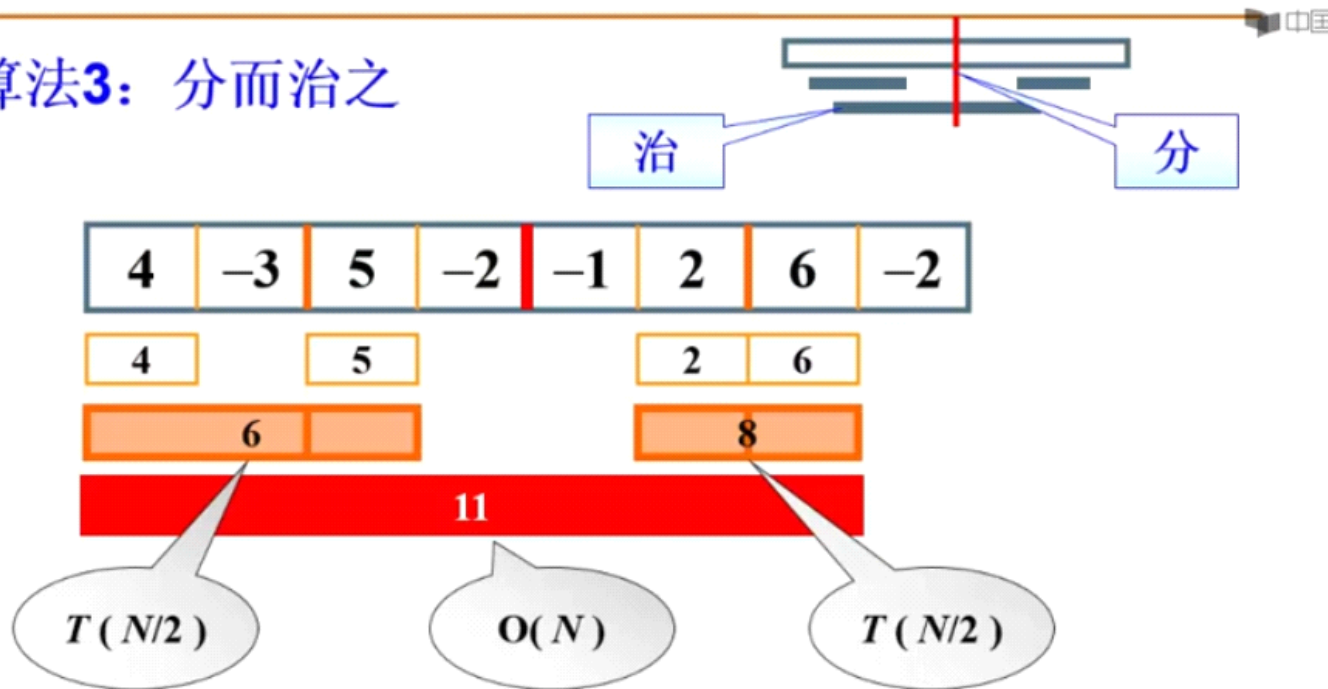
$$T_{avg}(n) \leq T_{worst}(n)$$

分而治之

2023年7月11日 9:26

最大子列和问题：

算法3：分而治之



$$\begin{aligned} T(N) &= 2T(N/2) + cN, \quad T(1) = O(1) \\ &= 2[2T(N/2^2) + cN/2] + cN \\ &= 2^k O(1) + c k N \quad \text{其中 } N/2^k = 1 \\ &= O(N \log N) \end{aligned}$$


屏幕剪辑的捕获时间: 2023/7/11 9:35

在线处理算法

2023年7月11日 9:39

算法4：在线处理

```
int MaxSubseqSum4( int A[], int N )
{ int ThisSum, MaxSum;
  int i;
  ThisSum = MaxSum = 0;
  for( i = 0; i < N; i++ ) {
    ThisSum += A[i]; /* 向右累加 */
    if( ThisSum > MaxSum )
      MaxSum = ThisSum; /* 发现更大和则更新当前结果 */
    else if( ThisSum < 0 ) /* 如果当前子列和为负 */
      ThisSum = 0; /* 则不可能使后面的部分和增大，抛弃之 */
  }
  return MaxSum;
}
```



$T(N) = O(N)$

屏幕剪辑的捕获时间: 2023/7/11 9:45