内容：

三层次五要素

|  |  |
| --- | --- |
| 数据表示 | 数据处理 |

|  |  |
| --- | --- |
| 抽象 逻辑结构 | 基本运算 |
| 实现 存储结构 | 算法 |
| 评价 | 不同数据结构的比较及算法分析 |

|  |  |  |
| --- | --- | --- |
| 数据 | 数据类型 | 数据元素 |

|  |  |  |
| --- | --- | --- |
| 数据类型 | 1.原子类型 | 值不能分解 |
|  | 2.结构类型 | 值由若干成分按某种结构构成 |

|  |  |
| --- | --- |
| 逻辑结构 | 数据间的相互关系 |
| 基本逻辑结构 | 1.集合结构 ：数据元素除同一类型外别无其他关系 |
|  | 2.线性结构：数据元素之间存在一对一的关系 |
|  | 3.树型结构：数据元素之间存在一对多的关系 |
|  | 4.图形结构：数据元素之间存在多对多的关系 |
| 树型和图形成为非线性结构 |  |

数据结构的形式定义为：数据结构是一个二元组：

Data-Structure：（D,R）

其中：D是数据元素的有限集，R是D上关系的有限集 （点和边）

物理结构

数据结构在计算机中的表示称为数据的物理结构，又称为存储结构

存储结构分为：顺序存储结构/链式存储结构

1.顺序：用数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系

2.链式：在每一个数据元素中增加一个存放地址的指针（），用此指针来表示数据元素之间的逻辑关系

ADT

ADT类型可以用一个三元组表示

ADT=(D,R,P)

D：数据对象，用结点的有限集合表示

R：D上关系的集合，用结点间的序偶（将两个元素 x,y 有顺序地放在一起构成一个组合(x,y)称为序偶）的集合来表示

P：对D的基本操作的集合

        Algorithm

[definition]有限指令集，遵循它可以完成特定任务

算法必须遵守的准则：

（1）definiteness 每条指令是明确的

（2）finiteness

（3）effectiveness be basic enough to carried out feasible

（4）input

（5）output

评价

Correctness

Readability

Robustness

Time & space complexities

描述

自然语言

图形语言 NS图/流程图/图的描述

算法语言 计算机语言/程设语言/伪代码

形式语言 数学方法

分析

包括事先分析和事后测试

事先分析 求出该算法的时间界限函数

事后测试

存在c,n0>0,对任意n>n0,有|f(n)|<=c|g(n)|

f(n)=O(g(n))

算法复杂度分析

|  |  |
| --- | --- |
| 比较同一问题的不同算法 | 从算法中选取一种对于所研究的问题来说是基本运算的原操作 |
| 算法执行时间 | 大致是    基本运算所需时间与运算次数乘积 |
| 算法基本运算 | 最深层循环内的语句 |
| 算法时间复杂度 | 算法的基本运算次数T（n) |
| T(n) = O(f(n)) |  |

算法时间复杂度分析

记号“O”读作“大O”，它表示随问题规模n的增大算法执行时间的增长率和f(n)的增长率相同。“O”的形式定义为：

|  |  |
| --- | --- |
| 堆（STL中为priority\_queue） | 定义：具有以下性质的完全二叉树：每个节点的值大于等于左右子节点的值（最大堆），反之定义最小堆 |
| BST 二叉搜索树 | 每个节点的值小于等于左子节点大于等于右子节点，便于搜索时选择路径 |
| 堆的操作 | 构建、插入、删除、堆排序 |

堆的实现

|  |  |
| --- | --- |
| 存储结构 | 数组 |
| 逻辑结构 | 完全二叉树 |
| 复杂度 | O（nlogn） |
| 插入 | 首先不考虑数值插入到完全二叉树的右下 == 数组末尾   进一步为满足堆的性质需上溯从而将元素放置正确位置 |

IMG_256 中缀转后缀

1.如果为字母直接输入

2.如果不是字母为符号

则弹栈至栈中符号优先级比当前符号低

3.输入完成后若栈不空则弹栈

树的非递归遍历

1.前序

While（！root）

有左子树时

访问并将节点入栈

否则

访问并弹栈

2.中序

Void inorder(root,visit){

Stack<node> s;

Node p = root;

While(p!=NULL||!s.empty){

While(p!=NULL){

S.push(p);

P=p->left;

}

If(!s.empty){

P=s.top();

Visit(p->elem);

S.pop();

P = p->right;

}

}

}

3.后序

后序的非递归比前序和中序更加复杂，要考虑由左子树转移而来还是右子树转移而来，可以在数据结构中加入tag

4.层序

Queue<node> q;

Node now = root;

Q.push(root);

Int \_size = 1;

While(!q.empty){

Now = q.front();

Visit(now->elem);

If(now->left){

Q.push(now->left);

\_size++;

}

If(now->right){

Q.push(now->right);

\_size++;

}

Q.pop();

}

IMG_257 堆的建立（最小堆）

筛选法

上滤

用于插入操作，将节点插入完全二叉堆的最后一个节点的后面，此后将其与其父节点比较直至其大于父节点或已为根

下滤

用于删除操作（删除的是堆顶节点），将完全二叉堆的最后一个节点放在第一个节点处，此后size--，然后将堆顶节点与其左右子节点比较，选择较小的与其交换直至无法进行操作

aoe网：

边表示活动，aoe网是一个带权有向无环图其中顶点表示事件/弧表示活动/权表示活动持续时间

路径长度：路径上各活动持续时间之和

关键路径：路径长度最长的路径

Ve(j):表示事件Vj的最早发生时间

Vl(j):表示事件Vj的最迟发生时间

E(i):表示活动ai的最早开始时间

L(i):表示活动ai的最晚开始时间

L(i)-e(i):表示完成活动的时间余量

IMG_258 关键活动：关键路径上的活动，即I(i)=e(i)的活动

求关键路径的算法（以邻接表为存储结构）

从源点V1出发，令VE[1]=0，按拓扑排序求各顶点的VE[I]

从汇点VN出发，令Vl[N]=VE[N]，按逆拓扑序列求其余各顶点的VL[I]

根据顶点的VE和VL值，计算每条弧的E[I]和L[I]

AVL树

avl树的旋转

LL/RR

把二层节点与根互换

LR

LL（下层节点）-》RR（根）

RL

RR（下层节点）-》LL（根）

散列函数的构造方法

1.除留余数法：

取不大于m的最接近m的质数p

2.数字分析法

λk = ∑i=1~k ((αk,I - n/r)^2)找出均匀度小的

3.平方取中法

内码的平方取中间n位且n = log2 r，r为地址总数

4.折叠法

对齐相加 1.移位法2.分界法