数据结构导论

408不要求一定要用类实现，使用c会简洁一些

数据结构研究的是数值计算问题中的数据组织与操作的问题

逻辑结构

- 集合
- 线性结构，一对一
- 树形结构，一对多
- 图状结构，多对多

注意**数据包含数据元素，数据元素包含数据项**
e.g. 一个学生信息数据表，每个学生是数据元素，学生的具体信息（学号、成绩等）是数据项。

逻辑上把数据结构分为线性结构和非线性结构

线性结构包含：

```
线性表
栈
队列
串
```

非线性结构包含：

```
树
二叉树
图（有向图，无向图，稀疏图，稠密图，带权图）
广义表
多维数组
```

数据的四种基本**存储**结构分为

```
顺序（顺序表，循环队列，顺序栈，邻接矩阵）
索引
链式（单链表，双链表，循环链表，静态链表，链队列，链栈，二叉链树，三叉链树，线索二叉树，邻接表）
散列（哈希表）
```

数据结构中，逻辑结构与所使用的计算机无关

连续存储设计时，存储单元的地址一定连续
对于链式结构，则不一定

算法

特性：有穷性，确定性，可行性，输入，输出

好算法的标准：正确性，可读性，健壮性，通用性，效率与存储需求（与问题规模有关）

时间复杂度

在循环中
若变量在每一次循环都++/--，O(n)
若每次都×k，O(logkn)

- O(1)：常量阶，

```
++x;
s=0;
```

- O(logn)：对数阶，

```
//一种对数阶的例子
while(i<n){
    i *= 2
}
//另一种对数阶的例子
for(k=1;k<=n;k *= 2){
    ...
}
```

- O(n)：线性阶，

```
for(i=1;i<=n;i++){
    ...
}
```

- O(nlogn)：
- O(n^k)：（O(2^n) < O(n!) < O(n^n)），次方阶，

```
    for(i=1;i<=n;i++){
        for(j=1;j<=n;j++){
            ...
        }
```

```
    }
    //当套用多层循环时，若内外层循环变量无关，则其频度可以为O(n)*O(n)
```

关于递归的时间复杂度

1. n ---- n=n-1 ----> 1, O(n)
2. n ---- n=n/k ----> 1, O(logkn)
3. f(n) = f(n-1) + f(n-2), O(2^n)

关于多重循环

1. 若内外层无关：外×内
2. 若内外层有关：需要做递归，无法直接做乘法，

空间复杂度

算法运行时使用的存储空间大小
空间复杂度的度量：程序代码，执行数据，辅助空间/临时变量（此项即为空间复杂度）

e.g.

```
for(i=1;i<=n;i++){
    for(j...){
        x++;
        s+=x;
    }
}
/*
该代码中临时变量为i，j，x，s，空间复杂度为O(1)，常量阶
*/
```

==========**分割线**==========

线性表

链表 Operations见下图。其中T是基本数据类型，即ei的数据类型，我们无需定义； traverse(visit(T))中的visit(T)是回调函数， 必须由traverse函数的调用者提供，它访问（处理）每一个基本数据元素ei。

栈 push(), pop()

· 单向链表实现的队列，其入列操作发生在链表表尾，出列操作发生在链表表头， 需设置两个指针变量，一个指向链表表头，一个指向链表表尾。 · 单向循环链表实现的队列，其入列操作仍发生在链表表尾，出列操作仍发生在链表表头，

但是只需设置一个指向链表表尾的指针变量即可。

队列 put(), get()

顺序存储的完全二叉树，其空间利用率最高。

线性表的实现

顺序存储

```
//静态分配
int L[MAX_NUM];
//动态分配
int *L = malloc(MAX_NUM * sizeof(int));
```

链式存储

```
typedef struct node *link;
struct node{
    int item;
    link next;
};
link head;
```

栈、队列、数组

栈、队列，基本概念

栈 push(), pop()

队列 put(), get()

```
/*顺序存储*/
//栈
int S[MAX_NUM];
int top;    //栈顶下标
//队列
int Q[MAX_NUM];
int front;  //队头下标
int rear;   //队尾下标

/*链式存储*/
//栈
typedef struct node *link;
struct node{int item, link next;};
link top;   //顶部指针
//队列
typedef struct node *link;
struct node {int item, link next;};
link front; //队头指针
link read;  //队尾指针
```

应用

1. 栈的应用：中断机制，传参，临时变量，表达式求值转换，PostScript， etc；
2. 离散事件仿真，迷宫求解，网络服务， etc；

特殊矩阵压缩

上三角、下三角，etc

树、二叉树

二叉树典操作：前序、中序、后续便利

树典操作：前根、后根遍历

```
/*完全二叉树采用顺序结构，一般二叉树采用链式结构*/
//顺序存储
int T[MAX_NUM];
int root = 0;
//链式存储
typedef struct node *link;
struct node{
    int item;
    link left_child;
    link right_child;
};
link root;


/*遍历*/

//前序便利
void pre_order(link t, void visit(link)){    //visit传递了一个地址
    if (t == NULL) return;
    visit(t);    //访问根
    //递归访问自己的左右节点
    pre_order(t -> left_child, visit);
    pre_order(t -> right_child, visit);
}

//后序遍历
void post_order(link t, void visit(link)){
    if (t == NULL) return;
    //从下往上，先访问左右节点，最后访问根
    post_order(t -> left_child, visit);
    post_order(t -> right_child, visit);
    visit(t);
}

//复制二叉树
link copy(link t){
    if (t == NULL) return NULL;
```

```
    link s = malloc(sizeof *t); //构造一个新节点
    s -> item = t -> item;  //拷贝数据域
    //先复制左边，再复制右边，最后返回数据域
    s -> left_child = copy(t -> left_child);
    s -> right_child = copy(t -> right_child);
    return s;
}

//销毁二叉树
void destroy(link t){
    if (t == NULL) return;
    destroy(t -> left_child);
    destory(t -> right_child);
    free(t);    //释放根
}

//线索二叉树的基本构造
typedef struct node *link;
struct node{
    int item;
    bool left;
    bool right;
    link left_child;
    link right_child;
}
```

树的存储结构

1. 孩子表示法
2. 双亲表示法（*）
3. 长子-兄弟表示法（*）

树的应用

1. 二叉排序树 BST
2. 平衡二叉树 AVL，
3. （最优二叉树）Huffman树和Huffman编码

   根据字符再通讯信道中出现频率不同，给以不同的编码长度

   应用领域：文件压缩