线性　　　　非线性

表　　　树　　　图　　：逻辑结构

顺序　链式　索引　散列　：存储结构

查找；插入；删除；合并；拆分；遍历

字符串匹配

排序：交换排序：冒泡；快排

插入排序：直接；对分；二路；希尔

选择排序：

归并排序：

堆排序：树形选择；锦标赛；堆

第一章 绪论

1.1 数据抽象与二元关系

一、数据抽象

元素 + 关系 → 有向弱连通图

二、集合的笛卡尔积

$A \times B = \{(a,b), a \in A 且 b \in B\}$，扩展概念：n个集合的笛卡尔积.

三、二元关系

$A \times B$ 的一个子集，$A = B$ 时称为A上的一个二元关系R.

$A \times A \supseteq R = \{(a,b), a \in A 且 b \in A 且满足一定的条件\}$　a：前件；b：后件.

条件：
- 自反性：$\forall a \in A, (a,a) \in R$；反自反性：$\forall a \in A, (a,a) \notin R$.
- 对称性：$(a,b) \in R$ 则 $(b,a) \in R$；反对称性：$(a,b) \in R$ 且 $a \neq b$，则 $(b,a) \notin R$
- 传递性：$(a,b) \in R$ 且 $(b,c) \in R$，则 $(a,c) \in R$ ⇒ 可得关系的基 $T \subset R$.

note：传递性 ~~和反对称性~~ + 反自反性 → 反对称性：反证法.

① 等价关系：自反性、对称性、传递性 ⇒ 等价类 ⇒ 划分集合：并查集.

② 偏序关系：自反性、反对称性、传递性：不能判断是否任意序偶 $(a,b) \in R$（偏序）

③ 全序关系：$\forall a,b \in A$，有 $(a,b) \in R$ 或 $(b,a) \in R$，其中R为偏序关系 ⇒ 拓扑排序

④ 严格偏序关系（拟序关系）：反自反性、反对称性、传递性：eg: ⊂.

1.2 数据结构的基本概念

一、数据的逻辑结构

$B = (D, R)$　D：数据集合；R：D上的关系集合. 线性：表；非线性：树、图.

二、数据的存储结构

顺序、链式、索引、散列.

三、抽象数据类型ADT.

数据结构 + 操作.

NO.

Date . .

1.3 算法描述与分析.

一. 算法的基本特征

二. 算法描述.

三. 算法设计的常用方法

蛮力法： 查找A中最大元素、背包、字符串匹配…

分治法： 查找A中最大元素、归并排序、快速排序、希尔排序…

减治法： 找假球、二分查找、B-树查找…

回溯法： 迷宫、背包.

贪心法： KMP、Prim、Kruskal、Dijkstra …

动态规划法： Floyd、Warshall.

四. 算法的时间复杂度与空间复杂度.

时：渐近/最坏；加/乘；循环/递归.

第二章 线性表的顺序存储及其运算.

2.1 线性表的概念.

一. $B=(D,R)$ $D=\{a_i \mid i=1,2,\cdots,n\}$ $R=\{(a_i,a_{i+1}) \mid i=1,2,\cdots,n-1\}$.

2.2 顺序表及其运算.

考虑无序表的插入和有序表的插入、删除、查找.

限制：对存储空间有要求（连续）；插入删除效率.

2.3 栈

开0向上：top=-1；开0向下：top=ms.

双栈：充分利用静态存储空间.

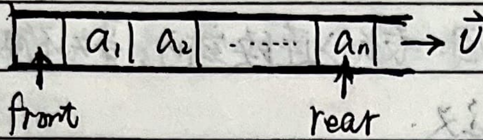
递归、分治法：

```
Divide(P)          //求解规模为n的问题P.
{ if (P的规模足够小) //直接求解P;
  else
     for(i=1;i<=k;i++) yi = Divide(Pi)    //求解子问题
  return Merge(y1,y2,…,yk);               //合并子问题的解.
}
```

递归工作栈：递推：push；回归：pop；保存局部变量、实际参数、返回地址。

2.4 队列

顺序队列：



循环队列：0单元接在ms-1单元后。清空：delete：front=rear=0

- 入队：(rear+1)%ms
- 退队：(front+1)%ms

- 队空：front==rear
- 队满：(rear+1)%ms==front

} 浪费一个存储单元。

优先队列：
- 新元素入队尾，出队找最优先元素
- 新元素根据优先权入队，出队是队头。

} 用堆实现。

2.5 数组与矩阵的表示

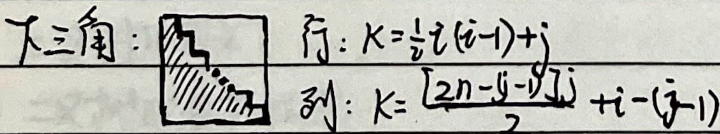
一、数组的顺序分配

以行为主：$AD(a_{ij})=AD(a_{11})+[(i-1)n+j-1]\cdot unit$；$AD(a_{ij})=AD(a_{00})+(in+j)unit$

以列为主：$AD(a_{ij})=AD(a_{11})+[(j-1)n+i-1]\cdot unit$；……

二、规则矩阵的压缩存储

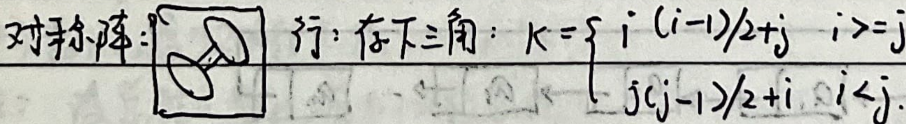
下三角：



行：$K=\frac{1}{2}i(i-1)+j$

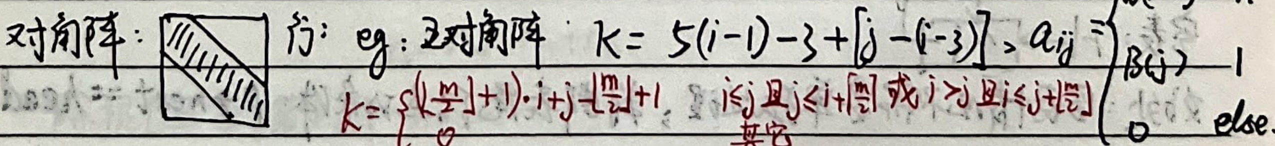
列：$K=\frac{[2n-(j-1)]j}{2}+i-(j-1)$

对称阵：



行：存下三角：$K=\begin{cases} i(i-1)/2+j & i>=j \\ j(j-1)/2+i & i<j \end{cases}$

对角阵：



行：eg：五对角阵 $K=5(i-1)-3+[j-(i-3)]$，$a_{ij}=\begin{cases} B(k) & [2\sim n-1] \\ B(k-1) & n \\ B(j) & 1 \\ 0 & else \end{cases}$

$K=\begin{cases} ([\frac{m}{2}]+1)\cdot i+j-[\frac{m}{2}]+1 & i<j 且 j<i+[\frac{m}{2}] 或 i>j 且 i<j+[\frac{m}{2}] \\ 0 & 其它 \end{cases}$

三、稀疏矩阵的三列二维数组表示——三元组顺序表B(行、列、值)

- POS[k]：稀疏矩阵中第k行第一个非0元在B中行号。POS[i]=POS[i-1]+NUM[i-1]
- NUM[k]：稀疏矩阵第k行非0元个数。NUM[B[i,1]]=NUM[B[i,1]]+1

亚龙纸制品

NO.

Date · ·

第三章 链表.

3.1 线性表的链式存储.

一. 线性链表的概念.

提出链表结构的原因：表长 > 连续空间；事先不能确定表长；表长经常变化

二. 线性链表及其结构定义.

插入的特殊情况：空表；插入第一结点；找不到插入位置.

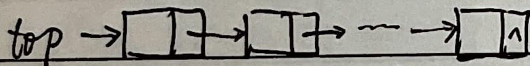
删除的特殊情况：空表；删除第一结点；找不到删除位置.

合并运算：选定基本链. $O(m+n)$

分解运算(按奇偶)：$O(n)$.

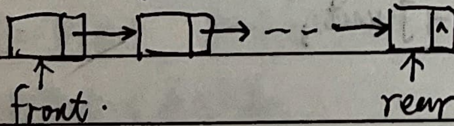
3.2 链式栈与链式队列.

一. 链式栈.



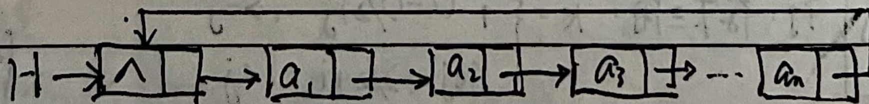
空闲结点用链式栈存储.(静态时)

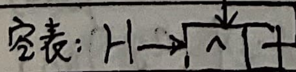
二. 链式队列.



空闲结点同样要收集起来.(静态时)

3.3 循环链表.



空表：

好处：空表情况不需要单独处理；判断表尾结点的条件：p→next == head.

合并运算：不考虑空表；当一链已完另一链还剩时需找到该链尾链回新链链头

分解运算：要新分配表头结点；按奇偶，要选择基本链.

3.4 多重链表

3.5 广义表

一. 广义表的概念:

线性结构;递归定义;表深=表的嵌套层数=子表结点的最大层数。表长=next数(第一层)。

head() tail()

二. 广义表的存储结构

附加表头结点:	flag=0	sublist	∧
原子:	flag=1	data	next
子表:	flag=2	sublist	next

第四章 树与二叉树

4.1 树的基本概念

根结点(root);结点(node);结点的度:子树数;树的度:max{结点的度}

从根结点起依次为1层、2层… 树的深度:最大层

性质 { n结点,度d_i($i=1,2,\dots,n$) $\Rightarrow$ $n=\sum_{i=1}^{n} d_i+1$.
n结点k叉树,最小深度:$\lceil \log_k(n(k-1)+1) \rceil$:除叶子外皆满度.

4.2 二叉树

一. 二叉树的定义.

二. 二叉树的基本性质.

① 叶子数n_0,度为2的结点n_2 $\Rightarrow$ $n_0=n_2+1$.

证: 设总结点数n,度为1的结点n_1

$$n=n_0+n_1+n_2.$$

除根结点每个结点有一分支连入:

$$n-1=2n_2+n_1$$

解得

$$n_0=n_2+1.$$

亚龙纸制品

NO.

Date · · $\lceil \frac{n}{2} \rceil$ 叶子 $\lfloor \frac{n}{2} \rfloor$ 非叶.

② 完全二叉树：左→右满；理想平衡二叉树：任意满；满二叉树：全满

n结点，深度 $\lfloor \log_2 n \rfloor + 1$ 深h，结点数最多 2^h-1

③ i的左子：2i+1，右子：2i+2，父：$\lfloor (i-1)/2 \rfloor$ 数学归纳法证明(从0算起)

2i 2i+1 $\lfloor i/2 \rfloor$ (从1算起)

三、二叉树的存储表示

二叉链表；完全二叉树则可用顺序存储，如堆。

四、二叉树的运算.

前序遍历：访问→左→右.
中序遍历 左→访问→右.
后序遍历：左→右→访问.
} 时：O(n)；空：O($\log_2 n$)~O(n)；非递归算法：书P165；辅P46.

按层遍历：左入队→右入队→出队；循环. while(!EmptyQueue()).

五、二叉树计数. 中序+前序 or 中序+后序 or 中序+层序 ⇒ 元素上升法！

~~n个~~前序(n结点)对应的中序序列数：~~$\frac{1}{n+1}$~~ C_{2n}^{n} (中序对应的前序数/后序数).

由前序和中序确定二叉树的方法：前序确定根，中序确定左右子树.

六、线索二叉树

ltag : lkid : data : rkid : rtag. tag=0：原路；tag=1：加线索.

前序线索化：当前结点和前序前件加线索→左→右.
中序线索化：左→当前结点和中序前件加线索→右.
后序线索化：左→右→当前结点和后序前件加线索.
} 前件 (rtag==1) → 左指针 (lchild==NULL) → 右指针 (rchild==NULL)

~~45~~.

4.3 二叉树应用

一、把有序树转换为二叉树.

有序树→二叉链→二叉树：树的孩子-兄弟表示法；单支时有序树不分左右

二、最优二叉树 (huffman). 高度h：$2h-1 \le n \le 2^h-1$. $\begin{cases} n_0=n_2+1 \\ n_0+n_2=n \end{cases} \Rightarrow \begin{cases} n_0=\frac{n+1}{2} \\ n_2=\frac{n-1}{2} \end{cases}$

目标：$\sum_{i=1}^{n} weight(i) \cdot length(i) = min$

方法：每次从森林中选 Weight 最小的两棵树构成二叉树，root weight = lweight + rw

应用：huffman 编码(非前缀编码)：左0右1.

应用 三. 二叉搜索树（BST） 高度h：$h<n<2^n-1$

左<右：中序有序，插入时按中序顺序插入。时$O(\log n)$~$O(n)$，空：相同

删除：{ 叶子：直接删
单支结点（度为1）：删后接子
双支结点（度为2）：中序前件代替 → 删中序前件 { 叶子：…… 单支：…… } $O(\log n)$ ↓ $O(n)$.

四. 堆（heap）. 高度h：$2^{h-1}\le n\le 2^h-1$

大根：上>下；小根：上<下 ⇒ 顺序存储的完全二叉树，层序编号

回顾：左子 $2i+1$，右子 $2i+2$，父 $\lfloor(i-1)/2\rfloor$

插入：插入堆尾（右下）；对该元素 sift up · $O(\log n)$

删除：删堆尾，直删；删堆顶：将堆尾堆顶交换，对堆顶元素 sift down，$O(\log n)$

建立：{ 逐个元素执行插入（sift up）. $O(n\log n)$
建好后逐个（右下→左上）sift down：从 $\lfloor\frac{n}{2}\rfloor-1$ 开始逐层向上. $O(n)$.

4.4 树的运算.

一. 树的存储结构

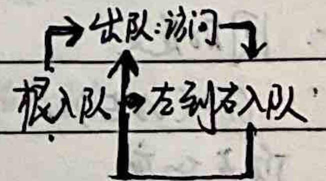
孩子：多重链表 or 左孩子右兄弟：二叉链表 or 父亲：多叉链表.

二. 树的遍历

先根遍历：DFT $O(n)$ ：查找、输出（广义表）、

后根遍历：$O(n)$ ：求深、清除、求结点数.

按层遍历：BFT $O(n)$（时），k^{h-1}（空）；k：度；h：深



4.5 树的应用

一. 解空间树

二. 树的遍历与回溯法. ：先根遍历解空间树.

NO.

Date . .

三. 划分等价类.

并查集的存储表示：树的多重链表表示.(父亲表示法).

并查集操作的实现：

- 建立初始并查集：数据：序号；父亲：-元素数)(初始为-1)
- 查找元素i所在的子集：向上找根结点值(总元素数).
- 合并两个子集合：以结点多子树的根结点为父.

应用：给定集合A和其上的等价关系R，通过合并操作得到等价类.

4.6 森林的遍历

一. 森林转换为二叉树：左孩子右兄弟

二叉树转换为森林：左孩子右兄弟

二. 森林的遍历

- 先根次序遍历
- 中根次序遍历(时称后根次序遍历)
- 后根次序遍历
- 按层遍历

} 左→右逐树执行

第五章 图

5.1 图的基本概念.

一. 图的定义

$G=(V,E)$ V:点集；E:边集. $G' \subseteq G$: G'子图.

二 顶点的度

无向图：相连边数；有向图：入度+出度=度. 总度=边数×2⇒偶数个奇点

三. 路径与连通.

无向图：连通图→连通分量；有向图：强连通图→强连通分量.

$n-1 \leq 边 \leq C_n^2$ (连通) (极大连通子图) $n \leq 边 \leq 2C_n^2$ (强连通). (极大强连通子图)

关节点：删之则图裂
重连通图：无关节点（任意一对点存在 ≥2 条路径）.

5.2 图的存储表示

一. 邻接矩阵表示

（$B=A^k$，$B[i,j]$ 是 $V_i - V_j$ 间长为 k 的路径的条数）

入度：$\sum_{j=0}^{n-1}$ 列 A[i][j]；出度：$\sum_{j=0}^{n-1}$ 行 A[i][j]　~~查找 ……~~；适于稠密图.

二. 邻接表表示.

（有向图：n+e，无向图：n+2e）

表头结点、边结点．出度 = ∑ 边结点　~~查找 ……~~；适于稀疏图.

入度 = ∑ 其它点连向该点的边结点.

5.3 图的遍历

一. 深度优先遍历（DFT）：邻接矩阵：$O(n^2)$；邻接表 $O(n+e)$. 空：$O(n)$.

类似于树的先根遍历；对每个连通分量分别进行，找最小的邻接点.

二. 广度优先遍历（BFT）：邻接矩阵：$O(n^2)$；邻接表 $O(n+e)$. 空：$O(n)$.

类似于树的层序遍历（需队列）；while(!EmptyQueue()).

5.4 图的最小生成树（MST）

生成树：不存在回路的连通图的子图（$V(G')=V(G)$）：n 顶点，n-1 条边

一. Prim 算法：$O(n^2)$　　连向生成树的最近边.

每一步总是把不在生成树中、离生成树最近的顶点（边）加入到生成树中（choose start）

二. Kruskal 算法：$O(e\log e)$.

每一步总是在边集中找不产生回路的最短边加入到生成树中.

5.5 最短路径问题

一. 单源最短路径：Dijkstra 算法：$O(n^2)$.（SPT）.

每一步总是把通过 SPT 与源点导通而离源点最近的顶点加入 SPT.

（每一步需更新各非 SPT 包含点到源点的可导通最短路径）.

NO.

Date · ·

二. 全源最短路径: Floyd 算法: $O(n^3)$ 空: $O(n^2)$

初始时 距离矩阵中的值是顶点两两间的直接距离(不经过其它点),以后依次加入每个顶点 V_k,若 $A^{(k-1)}[i][k]+A^{(k-1)}[k][j]<A^{(k)}[i][j]$,则更新 $V_i \to V_j$ 的最短路径,直到所有点都加入考虑. 此时的距离矩阵表示出全源最短路径.

5.6 关键路径

一. 拓扑序列与AOV网 最多 n! 个拓扑序列(全图);最少 1 个(线性序列).

AOV网: 有向无环图

拓扑序列: 由偏序关系得到的全序关系,不唯一: 不断取出源点直到图空

二. AOE网与关键路径

AOE网: 单源点、单汇点的有向无环图.

关键路径: 源点到汇点、带权路径长度最长的路径.

关键活动: 关键路径上所有边所代表的活动

事件:早 $V_e(j)=\max\{V_e(i)+weight(V_i \to V_j)\}$

事件:迟 $V_l(i)=\min\{V_l(j)-weight(V_i \to V_j)\}=V_e(n-1)-\max\{weight(V_i \to V_{n-1})\}$

活动:早 $e\langle V_i \to V_j\rangle=V_e(i)$

活动:迟 $l\langle V_i \to V_j\rangle=V_l(j)-weight(V_i \to V_j)$.

$e\langle V_i \to V_j\rangle=l\langle V_j \to V_i\rangle$ 的路径构成了关键路径,需加快其进度.

第六章 查找

6.1 顺序表查找

一. 顺序查找

适用于无序表, $O(n)$. 蛮力法

二. 对分查找

适用于有序表, $\lfloor \log_2 n\rfloor+1$ 减治法 对分查找树: 平衡二叉树 (虚拟).

递归实现(两次递归); 非递归实现(while 三个条件的判断)

NO.

Date . .

6.2 索引查找

一、索引存储结构：

线性表 —索引函数→ 索引表 + 子表；索引表存子表第一个结点的地址及信息（指针域+信息域）

适用顺序表表示索引表。

可用顺序表或链表存储子表。

两级查找：查找索引项（$i=g(k)$，k为关键字，i为子表号）→ 查找子表（顺序/二分）

多重索引（索引表太大）：对索引表再做索引。

二、分块查找

索引表为有序表，索引表中关键字项为每个子表元素的最大值，子表间有序，表内无序

m：索引表长、n：主表长（被均分为m子表），时：$\log_2(m+1)+\frac{n}{m}$ 或 $O(1)+\frac{n}{m}$.

三、索引文件

索引文件 { 主文件 / 索引表 { 索引项：关键字 { 与主文件各记录对应：稠密索引 / 对应主文件的一组记录：稀疏索引 ; 物理记录号

若索引表很大，对索引表再建立索引——查找表，查找项对应一组索引项。

插入操作：

对索引非顺序文件：

- 把记录写入主文件尾（若最后一块满，需新分配一个物理块）
- 把记录相应的索引项插入到索引表中：$i=g(k)$.

对索引顺序文件：

- 在主文件中查到插入位置，插入记录；
- 修改索引表（稀疏索引）；插入索引项到索引表中（稠密索引）.

删除操作：

- 在主文件中对删除记录加"删除"标记.
- 在索引表中删除相应记录的索引项（稀疏索引不做）.

亚龙纸制品

NO.

Date . .

△ 易于实现复合条件的检索

(顺序文件) △ 插入、删除不方便

(内链)

多重表文件：主文件——数据表；主索引表——主关键字；辅索引表——次关键字

倒排文件：辅索引表改为倒排表——相同次关键字的记录的物理记录号的顺序

(顺序表)

通过集合运算直接查取含多关键字的目标。

倒排表中各索引项长度不同，文件管理维护困难。

6.3 散列查找

线性hash表：计算 $i=H(k)$，若空则败，若是则成，若不是则顺序向后找 $i=i+1 \bmod M$

不能解决表溢出问题，冲突次数为移动距离。

随机hash表：计算 $i=H(k)$，…… 若不是则 $i=i_0+RN$，i_0：初位；RN：随机数

对经常有删除操作的情况不利。

外链hash表：计算 $i=H(k)$，取出第i项的头指针顺序查找（链表的查找）

需要较多的表外空间，时间复杂度变大。

溢出hash表：计算 $i=H(k)$，…… 若不是则顺序查找溢出表（统一放入溢出表）

6.4 树表查找

一. 二叉搜索树查找与对分查找的比较：动态查找/静态查找

二. AVL树 高h，$n \geq F(h+2)-1$ $(F(0)=F(1)=1)$，$\leq 2^h-1$

$\forall \alpha_i$，$|\alpha_i| \leq 1$，$\alpha_i = depth(左) - depth(右)$。

- P左子的左子树过深：LL：A为根、P为其右子，A右子为P左子。
- P右子A的右子树过深：RR：A为根，P为其左子，A左子为P右子。
- P左子A的右子树过深：LR：AC逆时转、PC顺时转
- P右子A的左子树C过深：RL：AC顺时转、PC逆时转

查找：$O(h) \sim O(\log_2 n)$。

插入、删除时只执行单地旋转

$\lceil \log_m[(m-1)N+1] \rceil$ $\lceil \log_{\lceil \frac{m}{2} \rceil}[\frac{\lceil \frac{m}{2} \rceil - 1}{2}(N-1)+1] \rceil + 1$.

NO.

Date

三. B-树.

① 每结点至多m棵子树, ② 根至少两棵子树, ③ 非根至少 $\lceil \frac{m}{2} \rceil$ 子树.

① 每结点至多m-1关键字, ② 根至少一个关键字 ③ 非根至少 $\lceil \frac{m}{2} \rceil - 1$ 关键字

设B-树深度h从一开始计数. (N个结点)

$$\lceil \log_m[(m-1)\cdot N+1] \rceil \le h \le \lceil \log_{\lceil \frac{m}{2} \rceil}[(N-1)\cdot \frac{\lceil \frac{m}{2} \rceil - 1}{2} + 1] \rceil + 1$$

设B-树深度h从0开始计数. (N个结点)

$$\lceil \log_m[(m-1)\cdot N+1] \rceil \le h \le \lceil \log_{\lceil \frac{m}{2} \rceil}[(N-1)\cdot \frac{\lceil \frac{m}{2} \rceil - 1}{2} + 1] \rceil.$$

B-树的查找: $O(\log_{\lceil \frac{m}{2} \rceil} N)$

分 $x = K_i$, $x < K_1$, $x > K_n$, $K_i < x < K_{i+1}$ 八种情况. $\log_{\lceil \frac{m}{2} \rceil}(\frac{N+1}{2})+1$

☆ B-树中空指针数 = 总关键字数 N+1. (空指针都在最底层).

B-树的插入: $O(\log_{\lceil \frac{m}{2} \rceil} N)$

查找 ⇒ { $n < m-1$: 直插
{ $n > m-1$: $K_{\lceil \frac{m}{2} \rceil}$ 上升插入父并分裂: 可能传递到根.

B-树的删除: $O(\log_{\lceil \frac{m}{2} \rceil} N)$

查找 ⇒ { 是叶子 { $n \ge \lceil \frac{m}{2} \rceil$: 直删
{ $n = \lceil \frac{m}{2} \rceil - 1$ 且相邻结点 $n \ge \lceil \frac{m}{2} \rceil$: 兄弟升至两者中, 换, 换后移至该点. (原两者中间其左子)
{ $n = \lceil \frac{m}{2} \rceil - 1$ 且相邻结点 $n = \lceil \frac{m}{2} \rceil - 1$: 父下降, 兄弟本身之合并, 删父...
{ 非叶子, 中序前件替之, 删中序前件. 删父时回到"是叶子"的三种情况.

6.5. 字符串匹配

一. 字符串及字符串匹配的基本概念.

正文n, 模式m.

二. 字符串匹配的简单算法

失败则后移一位. 成功: $m \sim m(n-m+1)$. 不成功: $n-m+1 \sim m(n-m+1)$.

三. KMP算法.

next[i]=j: 移动i-j位 while 第i位失配. $O(m+n)$.

NO.

Date . .

第七章 排序

7.1 交换排序

一. 冒泡排序

每次逆扫，将最值移入；记录最后一次发生移动的位置可提高效率.

比较：$\frac{1}{2}n(n-1)$. $O(n^2)$ 稳定. 考虑双向冒泡

二. 快速排序

分治法：每次选中间码将表大小二分，直至分完，选R[n] R[$\frac{n}{2}$] R[1]的中间值一

时 $O(n\log_2 n)\sim O(n^2)$ 空 $O(\log_2 n)\sim O(n)$ 不稳定. 适用大表.

7.2 插入排序

一. 直接插入排序

每次相邻元素入已排表，顺序查找已排表

比较：$n-1$ 移动：$2n-1$，$O(n^2)$ 稳定 适用小的基本有序表.

二. 对分插入排序

…… 对分查找已排表.

比较：$O(n\log_2 n)$ 移动：$O(n)$，$O(n^2)$，稳定.

三. 二路插入排序

用对分插入方法，把无序表中的元素逐个插入到有序表的前半部或后半部.

减治法：时 $\frac{n^2}{8}$，空 $O(n)$.

四. 希尔排序

分治法：将待排表按希尔序列划分，对各子表做直接插入排序

$O(n^{1.5})$ 不稳定.

7.3 选择排序

一. 简单选择排序

每次将无序表中最值接于有序表后（与无序表最左交换）.

比较：$\frac{1}{2}n(n-1)$ 移：$3(n-1)$ $O(n^2)$ 不稳定.

NO.

Date . .

二. 堆排序

树形选择排序：

锦标赛排序：选出最值后比较其途中所有兄弟结点选出次值 …… 稳定

时：$O(n\log_2 n)$ 空：$O(n)$.

堆排序：每次选出最值后将堆底代替堆顶，进行siftdown……

构造堆(siftdown)：$O(n)$，排序：$O(n\log_2 n)$，总 $O(n\log_2 n)$ 不稳定，适于大表.

7.4. 归并排序

一. 二路归并排序的概念.

归并(Merge)：将n个有序表合并成一个有序表的过程.

二路归并：将两个有序表合并成为一个有序表：分治：一直对半，小有序表做两两的归并.

二. 相邻有序子表的归并.

利用辅助空间(与序列同长)做归并，每趟归并完后依次将元素送回原表.

三. 二路归并排序的实现.

递推：归并趟数 $O(\log_2 n)$，每趟归并：$O(n)$ 总：$O(n\log_2 n)$，空 $O(n)$. 稳定.

7.5. 外排序简介

一. 外排序的基本步骤.

将含有n个记录的文件分成若干个长度为l的文件段(分它段)→内存→

文件段内对记录排序→有序文件段(归并段)→外存→

对所有归并段进行逐趟归并，每做一趟归并使归并段加大…

二. 外排序时间开销.

产生初始归并段的排序时间(内部排序)+内部归并时间(多趟)+外存读写时间

亚龙纸制品

NO.

Date

三. 败者树与最佳归并树.

败者树：败者为父，胜者为根/做父；比胜者树减少移动次数.

与胜者树的排序方法(锦标赛排序)类似.

$\frac{n}{L}$个归并段的k路归并趟数 $\log_k \frac{n}{L}$.

完成归并的时间复杂度 $O(n\log_k \frac{n}{L})$

最佳k路归并树：以初始归并段长度为权值的huffman树：

只有度为k的结点和度为0的叶子：$m_0=(k-1)m_k+1$

当m_0-1不是归并路数$k-1$的整倍数时，应以长度为0的$(k-1)-(m_0-1)\%(k-1)$个虚拟归并段补充叶子结点。 ⇒得到所有归并段进行归并的先后顺序.

要求算法

一、线性链表与循环链表的插入、删除、合并、分解运算的实现算法。

1. 线性链表的插入（循环链表不用考虑表头）

分配一个新结点 → 置a到新结点值域 → 处理空表或元素b处于第一个结点的情况 → 从第2个结点开始查找元素b → 若找到b，把a插在b前，否则插到链尾。

2. 线性链表的删除（同上）

在表中查找元素b → 没有找到b，删除失败 → 处理b在第一个结点中的情况 → 其他情况做正常删除 → 释放被删除的结点。

3. 线性链表的合并（循环链表多出将新链尾链回表头的操作）

处理空链情况 → 取X为基本链 → 当X、Y链都还有未合并的结点，使y_i所在结点链接到x_i所在结点之后 → 若y链还有未合并结点，把x_i接在y_i后。

4. 线性链表的分解（循环链表：设X为基本链，从中拆出Y）分奇偶

二、二叉树遍历算法

求深度（后序遍历）

```
if(BT==NULL) return 0;
else{ int dep1=本函数(BT->left);
      int dep2=本函数(BT->right);
      if(dep1>dep2) return dep1+1;
      else return dep2+1;
}
```

层序：BTreeNode* p;

```
if(BT!=NULL) EnQueue(BT);
while(!EmptyQueue())
{ p=OutQueue(); 访问p->data
  if(p->Lchild!=NULL) 入队之
  if(p->Rchild!=NULL) 入队之
}
```

三、建立线索二叉树的算法、线索二叉树的遍历算法

1. 建立线索二叉树的算法 eg 中序线索

亚龙纸制品

NO.

Date . .

```
void thread (BT)
{ static Node* pre = NULL;
  if (BT != NULL)
  { thread (BT->lchild);
    if (pre != NULL && pre->rtag == 1)   // 当前结点的中序前件需加线索
       pre->rchild = BT;
    if (BT->lchild == NULL)              // 当前结点左指针改为向前件的线索
       { BT->ltag = 1; BT->lchild = pre; }
    if (BT->rchild == NULL)              // 当前结点右指针修改标志
       BT->rtag = 1;
    pre = BT;                            // 当前结点置为前件
    thread (BT->rchild);
  }
}
```

2. 利用线索遍历二叉树 eg 中序线索

```
   Node *p = BT                 // 初始指向根
 if (p != NULL)
 {   while (p->ltag == 0)  p = p->lchild;  // 找中序序列首结点
 do
   { 访问 p->data;
     if (p->rtag == 1)  p = p->rchild;   // 是线索，走向线索指示的中序后件
     else                                // 是孩子，则找到中序后件
       { p = p->rchild;
         while (p->ltag == 0)  p = p->lchild;
       }
   } while (p != NULL)
 }
```

NO.

Date . .

四. 对分查找的递归、非递归算法.

1. 递归实现.

当左<=右, 取中间点

{如果中间点值<待查值, 中间点=递归右半边(中间+1作参数)

如果中间点值>待查值, 中间点=递归左半边(中间-1作参数).

否则中间点值=待查值.

2. 非递归实现.

while 左<=右, 取中间点

{如果中间点值<待查值, 左=中间+1.

如果中间点值>待查值, 右=中间-1

否则中间点值=待查值.

五. 字符串匹配的简单算法和kmp算法.

1. 蛮力法

两层循环, 里层检查是否匹配, 外层移动模式串.

2. kmp.

```
void ~~int MatchKMP(主文s, 模式p)~~ Next(模式p, int n[])
{  int m=strlen(p); n[0]=0;
   int i=1; j=0;    //初始化比较位置.
   while (i<m)
   { if (p[i]==p[j]) {n[i]=j+1; //部分匹配串长度加1
     i++; j++;}              //比较位置各进一.
     else if (j>0)  j=n[j-1]
     else {n[i++]=0;}        //j在串头时部分匹配串长度为0.
   }
}.
```

亚龙纸制品

NO.

Date . .

```
int KMP-match(正文T, 模式p)
{ int n=strlen(T), m=strlen(P);
  int i=0, j=0, nxt[Maxsize];
  Next(P, nxt);
  while(i<n)
  {  if(T[i]==p[j])                    //已经匹配j+1个字符.
        { if(j==m-1) return i-j      //匹配成功.
          else{i++; j++;}
        }
     else                               //失配
        { if(j>0)  j=nxt[j-1];       //移动到部分匹配串.
          else     i++;
        }
  }
  return -1;
}.
```

六. 排序算法.

1. 冒泡排序

```
void bubble (a left right).
{ for (int i=left; i<right; i++)
     for (int j=right; j>i; j--)
        发现逆序, 立即进行交换操作, compExch(a[j-1], a[j]);
}
```

改进: flag记录一趟比较中最后一次发生交换的位置

2. 快速排序

线性分割算法：

```
int Partition ( a, left right)
{ 选取表中元素 a[k], k∈[left, right];
  int i=left, j=right;
  while (i!=j)                                   // i,j相遇时本趟分割完成
  { while ( a[j] >= a[k] && i<j ) j--;           //j向左移直到a[j]<a[k].
    if (i<j)                                     // a[j]<a[k]
      { a[i]=a[j]; i++;                          // a[j]与a[i]换.
        while (a[i] <= a[k] && i<j) i++;         //i向右移直到a[i]>a[k].
        if (i<j)                                 // a[i]>a[k]
        { a[j]=a[i]; j--;}                       // a[i]与a[j]换
      }
  }
  a[i]=a[k];                                     // a[k]放入分割点 i 的位置
  return i;                                      //返回本趟的分割点 i.
}
```

快速排序：

```
void QuickSort ( a, left, right).
{ int i;
  if (right >left)                        //表长不小于2.
  i = partition ( a, left, right);        //序列的划分操作.
    QuickSort (a, left, i-1);             //左边快排
    QuickSort (a, i+1, right);            //右边快排.
  }
}.
```

亚龙纸制品

NO.

Date · ·

3 插入排序.

顺序

```
void Insert (...)                          //直接插入.
{  for (int j=1; j<len; j++)               //需要做len-1次插入.
   {  int temp = a[j];  int i=j-1;         //暂存无序子表的第一个元素(待插入元素)
      while ( i>=0 && a[i]>temp)           //i指向有序子表的最后位置
          { a[i+1] = a[i]; i--; }          //为temp腾出插入位置.
      a[i+1] = temp;                       //把待插入元素插入到有序表中.
   }
}.
```

对分

```
void BInsert (...)                         //对分插入.
{ for (int j=1; j<len; j++).
  {  int temp = a[j];    int low=0, high = j-1;
     while ( low<= high)                   //对分查找待插入元素在有序的位
         {  int mid =(low+high)/2;
            if (a[mid] >temp)   high = mid-1;
            else                low = mid+1;
         }
     for (int i=j-1; i<=high+1; i--)       //high+1 ~ j-1 元素向后平移一位.
         a[i+1]= a[i]
     a[high+1]= temp;
  }
}.
```

NO.

Date . .

4. 希尔排序

```
void Shell ( a , times , d[] ). // times :排序趟数, d[]:每趟间隔
{ for (int k = times ; k>=1 ; k--).          //做 times 趟排序,最后一趟 d[times]=1
  { int h = d[k];                             // 取本趟排序间隔
    for(int j = h ; j<len ; j++)              // 对各子表做插入排序.
        Insert (....);
  }
}
```

5. 归并排序

```
void MergeSort (a).          //需调用归并函数 merge (...).
{ int * work = new int[len];
  int length = 1;  // 初始有序子表长度.
  while (length < len).
  { int cur = 2*length;  // cur: 本趟归并得到的有序子表长度.
    for (int t = 0; t<len; t = t+cur).  // 一趟归并调用merge len/cur 次.
    { int low = t, high = t+cur-1, mid = t+length-1;
      if(high >len-1)  high = len-1;  // 最后一组的特殊处理.
      if (high > mid)  merge(a, low, mid, high, work);  // 两表归并.
    }
    length = cur;
  }
  delete []work;
}.
```

自底向上:

```
for (int m=1; m<=right-left; m=2*m)
    for (int i=left; i<=right-m; i+=m*2)
        merge(a, i, i+m, min(i+2*m-1, right);
```

自顶向下:

```
merge_sort(a, left, (left+right)/2);
merge_sort(a, (left+right)/2+1, right);
merge(a, left, (left+rig
```

亚光纸制品

NO.

Date . .

6. 选择排序.

```
void selectSort (a.).
{ for(int i=0; i<len-1; i++)          //需选择len-1次.
    { int pos=i;                       //pos记录本次选择的最小关键字项的位置.
      for(int j=i+1; j<len; j++)       //从无序子表中选最小关键字.
          if (a[j]<a[pos]) pos=j;
      if(pos!=i)                       //即 a[i]<a[pos].
          交换 a[i] a[pos].            //最小关键字项换到无序子表首位置.
    }
}.
```

7. 堆排序.

```
void heapSort (a--)
{ int len=right-left;  int *p=a+left;
  for(int k= (len-1)/2; k>=0; k--)   //自底向上构造堆.
      siftdown (p, k, len);          //对以k为堆顶的元素做筛选
  while(len>0)
    { exch(p[0], p[len]);            //堆顶和堆尾元素交换.
      len--;                         //堆规模减1.
      siftdown (p, 0, len);          //对当前处于堆中的元素做筛选
    }
};
```

NO.
Date . .

5. 归并排序(续).

```
void merge (int *c, int *a, int n, int *b, int m).   时O(n). 空:O(n)
{long i, j, k;
 for (i=0, j=0, k=0; k<n+m; k++)
 {  if (i==n)  {c[k]=a[j]; j++; continue;}
    if (j==m)  {c[k]=b[i]; i++; continue;}.
    c[k]=(a[i]<b[j])? a[i++]: b[j++];
 }
}
```

自底向上归并排序:

```
void merge-sort(int*a, int left, int right).   //merge(int*a, int p1, int p2, int p3)
{ for(int i=1; i<=right-left; i=2*i)            将a[p1]~a[p2-1]和
      for(int j=left; j<=right-i; j+=2*i)       a[p2]~a[p3]进行二路归并
         merge(a, j, j+i, min(j+2*i-1, right));  存于a[p1]~a[p3].
}.
```

自顶向下的归并排序:

```
void merge_sort (int *a, int left, int right).
{ if (right<=left) return;
  int mid = (left+right)/2;
  merge_sort (a, left, mid);
  merge_sort (a, mid+1, right);
  merge (a, left, mid+1, right);
}
```

一次归并: O(n)
归并趟数: $O(\log_2 n)$.
时: $O(n\log_2 n)$
空: O(n). 见merge(·)

亚龙纸制品