一、线性结构

1 线性表

数组

链表

2 堆栈

表达式求值

3 队列

二、树

基本概念

结点的度：结点儿子的个数

树的度：所以结点最大的度

深度：最大的层次

树的表示

儿子兄弟表示法

二叉树

二叉树

性质

度为2的树

每层最多2^{i-1}个结点

k层最多$2^k - 1$个结点

n_0（叶子结点个数） $= n_2$（度为2的结点个数） $+ 1$

(边的个数)$n_0 + n_1 + n_2 - 1 = 0 * n_0 + 1 * n_1 + 2 * n_2$

存储结构

1 顺序存储（完全二叉树），数组存储，子节点2*i，2*i+1，父节点i/2

2 链表存储（一般二叉树）

遍历

先序、中序、后序、层次

递归、非递归

中序非递归

二叉搜索树BST

左子树结点比根节点小，右子树结点比根节点大

删除：假如是叶子结点，直接删除；假如有一个孩子，把孩子连到父节点上；假如有两个孩子，用左子树中最大的值或者右子树的最小值替代

平衡二叉树AVL

左右子树高度差最大为1

插入：需要左旋和右旋来动态调整树的结构

堆

完全二叉树，且根节点比子节点都大（最大堆）

完全二叉树，且根节点比子节点都大（最小堆）

插入：不断和父节点比较，知道父节点比当前插入结点大，停止（O(logn)）

删除最大根：将最后一个元素放到根处，让它和儿子比较大小，直到他比所有儿子都大（和大的那个比），停止交换O(logn)

哈夫曼编码

每次把权值最小的两个二叉树合并，因此没有度为1的结点

所有数据位于叶子结点，保证编码没有二义性

并查集

三、图

表示

邻接表、邻接矩阵

遍历

DFS，BFS

最短路问题

单源最短路

无权图：简单BFS即可

有权图：Dijkstra算法

直接扫描所有未收录点，时间复杂度$O(|V|^2+|E|)$

将dist存在最小堆中，时间复杂度$O(|E|log|V|)$

多源最短路

V次单元最短路

Floyd算法$O(|V|^3)$：

```
void Floyd()
{   for ( i = 0; i < N; i++ )
      for( j = 0; j < N; j++ ) {
        D[i][j] = G[i][j];
        path[i][j] = -1;
      }
    for( k = 0; k < N; k++ )
      for( i = 0; i < N; i++ )
        for( j = 0; j < N; j++ )
          if( D[i][k] + D[k][j] < D[i][j] ) {
            D[i][j] = D[i][k] + D[k][j];
            path[i][j] = k;
          }
}
```

$$T = O(|V|^3)$$

最小生成树

prim

每次要选择距离最小的一个结点，以及用新的边更新其他结点的距离

kruskal

每次取最小的边，收录进来，假如收录后成回路（并查集判断两个点是否在两个不同的集合），就不收录，直到收录v-1条边，假如提前出来，说明原图不连通

拓扑排序

每次拿走一个入度为0的点

也可用来检测是有是个有向无环图DAG

四、排序

冒泡排序

每轮扫描前n-i个数字，比较相邻的数字，假如顺序不对，则交换

最好（已经排好）O(n)

最差（逆序）O(n^2)

插入排序

从左往右每次插入一个数字，把它放在合适的位置

最好（已经排好）O(n)

最差（逆序）O(n^2)

希尔排序

定义一个增量序列$D_M > D_{M-1} > \ldots > D_1 = 1$

对每个D_k进行D_k间隔排序

```
for (int D = n / 2; D > 0; D /= 2)
{//间隔从n/2 n/4 一直到1
    for (int i = D; i < n; i++)//插入排序
    {
        int pos = i, val = nums[i];
        while (pos >= D && nums[pos - D] > val)
        {
            nums[pos] = nums[pos - D];
            pos -= D;
        }
        nums[pos] = val;
    }
}
```

选择排序

每次从i~n-1选择最小的数字和i交换

堆排序

利用堆进行排序，把原数组变成最大堆，每次把根节点和最后一个结点交换，维护0~n-i-1这个最大堆

归并排序

每次给合并一个区间

稳定，时间复杂度O(nlogn)，但需要额外的空间

快速排序

找一个主元（可以是任意一个数，最左边的、三个数的中位数都可以）

把数组分为比主元大的，和比主元小的部分（使用i，j两个索引完成）

在对两个部分分别快速排序

```
void quicksort(int l,int r)
{
    if(l>=r)
        return;
    int p = nums[l];
    int i = l;
    int j = r;
    while(i<j)
    {
        while(i<j&&nums[j] > p)
            j--;
        if(i<j)
            nums[i++] = nums[j];
        while(i<j&&nums[i] < p)
            i++;
        if(i<j)
            nums[j--] = nums[i];
    }
    nums[i] = p;
    quicksort(l, i - 1);
    quicksort(i + 1, r);
}
```

表排序

间接排序，修改指针

基数排序

桶排序的加强版：假如数字的个数n比较小，但范围m比较大，比如n=10，m=1000000，一般的桶排序就效果一般

次位优先：举个例子：n个整数，范围0-100，可以先按个位数桶排，在根据十位数桶排，再根据百位数桶排，最终得到排序结果

主位优先：比如扑克牌先按花色分，再在各个花色里排序

对比

五、散列hash查找

算出查找对象的位置

1、计算位置

比如数字：求余

字符串：使用ASCII码，把ASCII码值当做每个位上的值，把看成一个32进制的数字

2、解决冲突

往后放之类的

开放地址法：线性探测（一个一个往后找）、平方探测（往$+-i^2$，$i \leq tablesize/2$找）、分离链接法（把冲突的数据放在一个链表里）

六、KMP

match/next数组

$$match(j)=\begin{cases}满足\ p_0\cdots p_i=p_{j-i}\cdots p_j\ 的最大\ i(<j) \\ -1 \quad 如果这样的\ i\ 不存在\end{cases}$$

pattern	a	b	c	a	b	c	a	c	a	b
j	0	1	2	3	4	5	6	7	8	9
match	-1	-1	-1	0	1	2	3	-1	0	1

```
void buildmatch(string patten)
{
    int i, j, m = patten.size();
    match[0] = -1;
    for (j = 1; j < m;j++)
```

```
    {
        i = match[j - 1];
        while(i>=0&&patten[i+1]!=patten[j])
            i = match[i];
        if(patten[i+1]==patten[j])
            match[j] = i + 1;
        else
            match[j] = -1;
    }
}
int kmp(string str,string patten)
{
    int n = str.size();
    int m = patten.size();
    int s = 0, p = 0;
    while(s<n&&p<m)
    {
        if(str[s]==patten[p])
        {
            s++;
            p++;
        }
        else if(p>0)
            p = match[p - 1] + 1;
        else
            s++;
    }
    return p == m ? s - m : -1;
}
```