第一章 绪论

求解问题的主要步骤

分析阶段：弄清所要解决的问题是什么，并用一种语言清楚地描述出来。

设计阶段：建立程序系统的结构，重点是算法的设计和数据结构的设计。

可行解：满足要求的普通解。

最优解：分组数最少的解。

次优解：分组数接近最优解的可行解。

编码阶段：采用适当的程序设计语言，编写出可执行的程序。

测试和维护：发现和排除在前几个阶段中产生的错误，经测试通过的程序便可投入运行，在运行过程中还可能发现隐含的错误和问题。

求解问题的两大类

数值计算、非数值计算

什么是数据结构?

按照逻辑关系组织起来的一批数据,按一定的存储方法把它存储在计算机中，并在这些数据上定义了相关运算的集合.

数据的逻辑结构、存储结构和数据的运算

四类逻辑结构

集合结构、线性结构、树形结构、图状或网状结构

四类存储结构

顺序存储结构、链接（链状）存储结构、索引存储结构、散列存储结构

算法的设计取决于逻辑结构，实现依赖于存储结构

算法的定义：算法是对特定问题求解方法和步骤的一种描述，它是指令的一组有限序列，其中每个指令表示一个或多个操作。

算法+数据结构=程序

算法的五个重要特性：输入、输出、有穷性、确定性、可行性

设计算法的四个基本要求：正确性、可读性、健壮性、高效性

空间效率、时间效率的相互制约

时间效率的度量方法（基本操作的执行次数）时间复杂度

空间效率的度量方法（额外辅助空间的数量）空间复杂度

第二章 线性表

线性表的逻辑结构

线性表的顺序存储结构，要求能够综合应用

顺序表上的插入、删除操作及其平均时间性能分析。利用顺序表设计算法解决简单的应用问题。

线性表的链式存储结构的单链表，要求能够综合应用。

单链表如何表示线性表中元素之间的逻辑关系。

单链表中头指针和头结点的使用。（头结点可以让第一个元素节点不用单独处理）

单链表上实现的建表、查找、插入和删除等基本算法，并分析其时间复杂度。

利用单链表设计算法解决简单的应用问题。

单链表的存储密度比顺序表低，但在许多情况下链式的分配比顺序分配有效；顺序表为静态结构，而链表为动态结构。
在单链表指定位置进行插入、删除运算比在顺序表里容易得多；顺序表通过数据元素移动完成，而链表通过修改指针完成。
对于顺序表，可通过简单的定位公式随机访问任一个元素，而在单链表中，需要顺着链逐个进行查找。因此，顺序表为随机存取结构，而链表不是。

单链表与顺序表的选择
（1）有利于基本运算的实现频繁访问：顺序表（随机存取）；频繁插入、删除：链表
（2）有利于数据的特性顺序表：难于事先估计数据元素个数，过大浪费空间，过小易产生溢出；
链表：动态申请，但存储密度比顺序表低；
（3）有利于软件环境（程序设计语言）
程序设计语言是否支持动态分配？Fortran、Basic 等不支持动态分配，此时只能选择顺序表。

线性表的链式存储结构的循环链表和双链表，要求能够简单应用

循环链表的定义，及其上算法与单链表上相应算法的异同点。

双链表的定义及其相关的算法。

第三章 字符串
字符串是一种线性结构，它是零或多个字符组成的有限序列。
字符串基本运算的实现。
基于字符串函数解决相关应用问题。

模式匹配是一个比较复杂的操作，目的是在 t 中找出和 p 相同的子串
朴素模式匹配算法简单，但由于回溯操作使得算法效率低[O(n*m)]。
改进方法是 KMP 无回溯模式匹配方法，该方法中首先根据模式本身计算得到 next 数组[O(m))，然后通过使用 next 数组避免匹配过程中的回溯操作，从而提高了匹配速度，减少了时间复杂度。KMP 算法的时间效率为 O(m+n)，但空间效率比朴素模式匹配方法降低。
Next 数组：

若 next[i]≥0，表示一旦匹配过程中 pi 与 tj 比较不等，可用 p 中以 next[i]为下标的字符与 tj 进行比较。

若 next[i]=-1，则表示 p 中任何字符都不必在与 tj 进行比较，下次比较从 tj+1 与 p0 开始。

next 计算方法：

(1) 求 p0…pi-1 中最大相同的前缀和后缀的长度 k;

(2) next[i]=k;

第四章 栈与队列
栈：后进先出
栈的顺序和链接表示
以及在这两种表示方式下基本运算的实现
栈满和栈空的条件及描述
利用栈解决简单应用问题
回溯法

队列：先进先出
环形队列和链接队列的表示及其基本运算
队列满和队列空的条件及描述
利用队列解决相关应用问题

基于栈的搜索被称为“深度优先搜索”(depth-first search)
基于队列的搜索被称为“宽度优先搜索”(width-first search)

线性结构：唯一前驱、后继
非线性结构：存在两个或两个以上的前驱或后继
　层次结构关系的问题可以用树形结构来表示

第五章 二叉树和树
二叉树：8 个重要性质
1. 在非空二叉树的第 i 层上至多有 2^i 个结点(i≥0)
2. 高度为 k 的二叉树中最多有 2^(k+1)-1 个结点(k≥0)
3. 对于任何一棵非空的二叉树，如果叶结点个数为 n0，度为 2 的结点个数为 n2，
   则有 n0= n2+ 1 （n=n0+n1+n2　n-1=n1+2*n2 ）
4. 满二叉树：如果一棵二叉树的任何结点或者是树叶，或有两棵非空子树，则此二叉树称作“满二叉树”
5. 完全二叉树：如果一棵二叉树至多只有最下面的两层结点度数可以小于 2，并且最下面一层的结点都集中在该层最左边的若干位置上。
   注:完全二叉树不一定是满二叉树
   (1). 具有 n 个结点的完全二叉树的高度 k 为 $[\log_2(n)]$
   (2). 根节点编号为 0，左右子女：2*i+1,2*i+2
   　　根节点编号为 1，左右子女 2*i，2*i+1
6. 扩充二叉树：把原二叉树的结点都变为度数为 2 的分支结点
   (1). 在扩充的二叉树里新增加的结点（树叶结点），称为外部结点。树中原有的结点称为内部结点。(外部节点=内部节点+1)
   (2). “外部路径长度”E: 在扩充的二叉树里从根到每个外部结点的路径长度之和
   “内部路径长度”I: 在扩充的二叉树里从根到每个内部结点的路径长度之和
   （E = I + 2n，其中 n 是内部结点个数）

存储方式：顺序存储（空间较浪费）、二叉链表（左右子女）、三叉链表（左右子女+父节点）

二叉树的三种深度优先遍历方法: DLR、LDR、LRD 的递归实现
唯一确定一颗二叉树:

1. 中根+先根
2. 中根+后根
3. 先根+后根不能唯一确定

二叉树的广度优先遍历算法（应用队列）

卡特兰数:
b(0)=1，b(1)=1
b(n)= b(0)*b(n-1)+b(1)*b(n-2) +...+ b(n-1)b(0)　(n>=2)
b(n)= (2n,n)/(n+1)

二叉树的应用

- 二叉树深度优先遍历算法的应用（递归）
  - 计算二叉树结点的个数
  - 计算二叉树叶结点的个数
  - 计算二叉树的高度
  - 计算二叉树的直径
- 二叉树层次遍历算法的应用
  - 求二叉树中第 k 层的结点个数
  - 二叉树最大宽度
    - 树的宽度是所有层中的最大宽度，每一层的宽度被定义为两个端点（该层最左和最右的非空结点，两端点间的空结点也计入长度）之间的长度。
- 堆与优先队列
  - 小根堆、大根堆
- 哈夫曼树及其应用
  - 设有一组（无序）实数｛w1, w2, w3,…,wm｝，现要构造一棵以 wi（i= 1,2,…，m）为权的 m 个外部结点的扩充的二叉树，使得带权的外部路径长度 WPL 最小。满足这一要求的扩充二叉树也称为“Huffman 哈夫曼树”或“最优二叉树”
  - 建造:
    - 利用优先队列，在 F 中选取两棵权值最小的树作为左右子树以构造一棵新的二叉树，且新二叉树的根结点的权值为其左右子树根结点权值之和。
    - 在 F 中删除这两棵树，同时将新得到的二叉树加入 F 中。
    - 重复上述步骤，直到 F 中只含一棵树为止。
  - 哈夫曼编码

树：链接存储表示（父、孩子、孩子兄弟）

- 树的子表表示法[孩子表示法]
- 树的长子-兄弟表示法[孩子兄弟表示法]
- 树的父指针表示法[双亲表示法]
- 树林的存储表示

顺序存储

- 带右链的先根次序表示
- 带双标记的先根次序表示

带双标记的层次次序表示
带度数的后根次序表示

树的遍历（深度优先、宽度优先）
深度优先（先根、中根、后根）

树、树林与二叉树的转换
树或树林与二叉树之间有一个自然的一一对应关系。
树、树林转换为二叉树
方法：左手拉第一个孩子，右手拉下一个兄弟，只留父节点和长子之间的连线
树对应的二叉树其根节点的右子树总是空的
二叉树转换为树或树林
① 如某结点为其父母的左子女，则把该结点的右子女，右子女的右子女，……，都与该结点的父母用虚线连接起来；
② 去掉原二叉树中所有父母到右子女的连线；
③ 整理上面得到的树或树林，并将虚线改为实线。
树/树林的先序遍历序列一定和其对应的二叉树的先序遍历序列相同。
树/树林的后序遍历序列一定和其对应的二叉树的中序遍历序列相同。

并查集
一种特殊的集合，由一些不相交的子集构成（通常是在开始时让每个元素构成一个单元素的集合），其基本操作是:
①查 find：判断两个结点是否在同一个集合中。
②并 union：合并两个集合。
多用于求解等价类问题，常常在使用中以树林来表示。
路径压缩

第六章 字典与检索
衡量一个检索算法效率的主要标准
检索过程中和关键码的平均比较次数，即平均检索长度 ASL(Average SearchLength)
ASL=p1*c1+p2*c2+……+pn*cn
n 是字典中元素的个数
pi 是查找第 i 个元素的概率，若不特别声明，一般认为每个元素的检索概率相等，即 pi=1/n
ci 是找到第 i 个元素的比较次数

顺序表示
顺序检索、二分法检索（分治法）

索引表示
如果每个索引项对应一个字典中元素，这种索引称为密集索引。
如果每个索引项对应字典中的一组元素，这种索引称为稀疏索引。
解决不等长元素字典的表示问题。

不同的值需要的空间大小不同，这样的字典难以采用顺序存储和散列存储。

引入索引就可以将包含大量属性信息并且不等长元素的字典的处理转换成对索引结构的处理。

当字典中有大量元素时，对目录表（或者任何排序的顺序表）还可以建立目录，实现分层管理的目的。

分块检索：块间检索+块内检索（效率介于顺序、折半检索之间）

链接表示

二叉排序树（定义、性质、检索、插入、构造、删除相等检索概率下最佳二叉排序树（定义、构造）

二叉排序树（bst，binary search tree）具有以下性质：

如果任一结点的左子树非空，则左子树中的所有结点的关键码都小于根结点的关键码；

如果任一结点的右子树非空，则右子树中的所有结点的关键码都大于根结点的关键码。

左、右子树也分别为二叉排序树；

最好情况下，二叉排序树的平均检索长度为 $O(\log_2 n)$。

最坏情况下，二叉排序树的平均检索长度为 $O(n)$。

一般情况下，二叉排序树的平均检索长度为 $O(\log_2 n)$。

最佳二叉排序树、平衡二叉排序树

散列表示

散列表（哈希表）

散列函数：直接定址法（线性函数值：a*key +b）、数字分析法、中平方法、折叠法、基数转换法、除余法（常用，key％p，p 的选择非常重要，通常选择小于等于散列表长度 m 的某个最大素数）、乘余取整法

存储表示与碰撞的处理：

闭散列法

线性探查法：若在地址为 d0=H(key)的单元发生碰撞，则依次探查下述地址单元：d0+1,d0+2,…,m-1,0,1,…,d0-1(m 为基本存贮区的长度)直到找到一个空单元，或查找到关键码为 key 的元素为止

二次探查法：探查增量序列依次为：1^2，-1^2，2^2，-2^2，…，±(k)^2,(k<=m/2)

即地址公式是 d_2k-1= (d0+k^2)％m　　d_2k= (d0–k^2)％m

伪随机数序列探查：随机制造增量序列，然后得到了探查序列

双散列探查法：探查序列是原来关键码值的函数

删除节点时，不能置空，而要做标记（墓碑），在墓碑处不能直接插入新的元素，防止后面已经有相同的关键码

负载因子：α=n/m

开散列法（α较大时使用）

拉链法：碰撞时元素通过链表发到溢出区

检索方法的选择需要综合考虑

时间复杂度是判断检索方法的重要指标，但不是唯一指标

哈希检索，时间复杂度可以到 O(1)，查找速度最快，但需要构建哈希函数，设计解决冲突的方法；

二分检索，需要元素有序顺序存储，排序开销需要考虑；顺序检索，对检索表无要求，在数

据量小时使用方便。

第八章 排序

见 PPT 比较和总结图

排序的分类

按稳定性：稳定排序和不稳定排序（相对次序是否保持不变）

按数据存储介质：内部排序和外部排序（数据在内存或外存）

按辅助空间：原地排序和非原地排序（辅助空间用量是否为 O(1)）

按主要操作：比较排序和基数排序

比较排序方法：用比较元素大小的方法

插入类：每一趟将无序序列区一个或几个待排序的记录，按其排序码大小“插入”到已排序序列中

选择类：每趟从待排序记录序列中“选择”关键字最小或最大的记录加入到已排序序列中

交换类：两两比较待排序记录的排序码，“交换”不满足顺序要求的偶对，从而得到其中排序码最大或最小的记录，把它加入到已排序序列中

归并类：通过“归并”两个或两个以上记录的有序序列

分配排序（基数排序）：不比较元素的大小，根据数据各位的取值确定其有序位置

评价排序算法好坏的标准

时间复杂度:它主要是分析记录关键字的比较次数和记录的移动次数

空间复杂度:算法中使用的内存辅助空间的多少

稳定性

算法本身的复杂程度也是考虑的一个因素

（分最好、最坏、平均三种情况估算）

插入排序

每一趟将待排序序列区一个或几个待排序的记录，按其排序码大小“插入”到前面已经排序的文件中的适当位置，直到全部插入完为止

直接插入排序：将未排序码依次插入到已排序码中的适当位置（顺序比较，记录移动）

若文件初态为正序，则算法的时间复杂度为 O(n)

若初态为逆序，则时间复杂度为 O(n^2)

时间复杂度 O(n^2)

辅助空间 O(1)

稳定

二分法插入排序：直接选择排序插入时采用二分法来进行比较

最坏的情况为 n^2/2

最好的情况为 n

平均移动次数为 O(n^2)

辅助空间 O(1)

稳定

表插入排序：在直接插入排序的基础上减少移动的次数，采用链表存储

时间效率 O(n^2)
辅助空间：O(n)　指针
稳定

shell 排序：取一个整数 d1<n，把全部记录分成 d1 个组，所有距离为 d1 倍数的记录放在一组中，先在各组内排序
然后取 d2<d1 重复上述分组和排序工作
直到 di=1，即所有记录放在一组中为止
各组内的排序可以采用直接插入法，也可以采用后面讲到的其它排序方法

时间复杂度 O(n^2)　O(n^3/2)　O(n*log^2n)　不同增量序列
不稳定

选择排序

思想：每趟从待排序的记录序列中“选择关键字最小的记录”放置到已排序表的最前位置，直到全部排完。

关键问题：在剩余的待排序记录序列中找到最小关键码记录。

方法：直接选择排序、堆排序

直接选择排序：首先在所有记录中选出排序码最小的记录，与第一个记录交换
然后在其余的记录中再选出排序码最小的记录与第二个记录交换
以此类推，直到所有记录排好序

时间复杂度 O(n^2)　辅助空间 O(1)　不稳定

锦标赛排序（树形选择排序）：全部元素为叶子节点 $O(n\log_2 n)$

堆排序：（大根堆、小根堆）

建堆时间：4*n，每次选择：$\log_2 n$
平均时间复杂度 $O(n\log_2 n)$
辅助空间 O(1)
不稳定

交换排序

逆序：i<j 时，a[i]>a[j]

交换排序的基本思想：

若 i<j 时，而 a[i]>a[j]，则交换 a[i]与 a[j]。当对任何 i<j,a[i]<=a[j]时，排序完成。

冒泡排序：两两比较待排序相邻记录的排序码，交换不满足顺序要求的偶对，直到无逆序对为止。每一趟冒泡排序得到排序码最大或最小的记录，把它加入到已排序（有序）序列中。

最好时间复杂度是 O(n)
最坏时间复杂度为 O(n^2)
平均时间复杂度为 O(n^2)
算法中增加一个辅助空间 temp，辅助空间 O(1)
稳定

快速排序（基于分治思想）

① 轴值选择：从待排序列中选择轴值 k（参考点）为划分基准（轴值选择）
② 序列划分：划分为子序列 L 和 R，L 中记录都“<”k，R 中记录都“>=”k
③ 递归排序：对子序列进行递归划分，直到仅含 1 或 0 个元素

快速排序的记录移动次数不大于比较次数
最好时间复杂度为 $O(n\log_2 n)$
最坏时间复杂度应为 O(n^2)，快速排序退化为冒泡排序
平均时间复杂度是 $T(n)=O(n\log_2 n)$
平均空间代价为 $O(n\log_2 n)$
不稳定

归并排序（基于分治思想）
归并排序的主要思想是：
把待排序的文件分成若干个子文件，先将每个子文件内的记录排序
再将已排序的子文件合并，得到完全排序的文件
合并时只要比较各子文件第一个记录的排序码，排序码最小的记录为排序后的第一个记录，取出该记录
继续比较各子文件的第一个记录，找出排序后的第二个记录如此反复，经过一次扫描，得到排序结果
二路归并排序
①设文件中有 n 个记录，可以看成 n 个子文件，每个子文件中只包含一个记录
②先将每两个子文件归并，得到 n/2 个部分排序的较大的子文件，每个子文件包含 2 个记录
④ 再将子文件归并，如此反复，直到得到一个文件
时间复杂度：最好、最坏、平均都是 $O(n\log_2 n)$
辅助空间 O(n)
稳定
归并排序法可以用于内排序，也可以用于**外**排序

分配排序：一种借助多关键码排序思想对单关键码排序的方法
例子：扑克牌排序
低位优先，高位优先
基数排序：按低位优先来进行排序
例如：所有数先按个位分配，再按十位分配
时间复杂度：O(d*(r+n))
辅助空间 O(n+r)
稳定

第九章 图
图：无向图，有向图，边（弧）带权=>网络

简单图：不考虑顶点到自身的边，且图中不允许一条边重复出现

有向完全图：有 n(n-1)条边的有向图
无向完全图：有 n(n-1)/2 条边的无向图

度：与顶点 v 相关联的边数称为顶点的度，记为 D(v)
有向图中分为入度、出度，度=入度+出度

边数=所有顶点度数之和/2

通路、回路
有边重复出现的通路/回路称为复杂通路/回路
无边重复则称为简单通路/回路
若无边重复且无顶点重复称为初级通路/回路

路径：路径长度、简单路径、回路或环

欧拉通路：通过图中所有边一次且仅一次，行遍所有顶点的通路。
欧拉回路：通过图中所有边一次且仅一次，行遍所有顶点的回路。
无向图
有欧拉回路的充要条件是：图是连通的，且每一个顶点的度数都是偶数。
有欧拉通路的充要条件是：图是连通的，且仅有两个奇数度顶点（它们分别是欧拉通路的两个端点），或者没有顶点的度是奇数
有向图
有欧拉通路的充要条件：D 连通，除两个顶点外，其余顶点的入度均等于出度，这两个特殊的顶点中，一个顶点的入度比出度大 1，另一个顶点的入度比出度小 1。
有欧拉通路的充要条件：D 连通，D 中所有顶点的入度等于出度。

根：有向图中，若存在一顶点 v，从该顶点有路径可以到图中其它所有顶点，则称此有向图为有根图，v 称为图的根。显然，有根图中的根可能不唯一。

无向图
连通图: 若 V(G)中任意两个不同的顶点 vi 和 vj 都是连通的(即有路径), 则称 G 为连通图。
连通分量：无向图 G 中的极大连通子图称为 G 的连通分量。
有向图
强连通图：有向图 G=(V，E)中，若 V(G)中任意两个不同的顶点 vi 和 vj 都存在从 vi 到 vj 以及从 vj 和 vi 的路径，则称图 G 是强连通图。
强连通分量：有向图的极大强连通子图称为图的强连通分量。
强连通图只有一个强连通分量，就是其自身。非强连通的有向图有多个强连通分量。

网络：带权的连通图

生成树：针对对象——连通的无向图或强连通的有向图、有根的有向图（极小连通子图）
对于连通的无向图或强连通的有向图，从任一顶点出发，或是对于有根的有向图，从图的根顶点出发周游，可以访问到所有的顶点。周游时经过的边加上所有顶点构成了图的一个连通子图，称为图的一个生成树。
它含有图中的全部 n 个顶点，但只有足以构成一棵树的 n-1 条边，是连通图的一个极小连通子图。

顶点数为 n，边数大于 n-1 的无向图必定存在环；

图的存储：
邻接矩阵
无向图的邻接矩阵一定是一个对称方阵。
邻接矩阵表示图，很容易确定图中任意两个顶点之间是否有边相连。
无向图的邻接矩阵的第 i 行(或第 i 列)非零元素的个数为第 i 个顶点的度 D(vi)。
有向图的邻接矩阵的第 i 行非零元素个数为第 i 个顶点的出度 OD(vi)，第 i 列非零元素个数就是第 i 个顶点的入度 ID(vi)。
邻接表（逆邻接表）
由顺序存储的顶点表+ n 个链式存储的边表
无向图
有向图：出边表、入边表
当边数较少，邻接表节省空间；当边数较多，邻接矩阵节省空间
通过存储表示，顶点入度、出度计算

图的遍历：
DFS(深度优先) DFS 序列 DFS 生成树（递归）
BFS(广度优先) BFS 序列 BFS 生成树（队列）
不同存储表示（邻接矩阵、邻接表）下的实现算法

判断是否连通？
1. 可用 dfs 和 bfs 遍历图算法，判断是否所有点已被遍历过，未访问到的节点，那么该图一定是不连通的。
2. 并查集的方法：初始化时将每个顶点看作一个集合，则每给出一条边即把两个集合合并。最后有几个集合便有几个连通分量，若只有一个集合说明图连通。

回路的判断
1. 用 BFS 或 DFS 遍历，最后判断对于每一个连通分量当中，如果边数 m>=节点个数 n，那么该图一定存在回路。因此在 DFS 或 BFS 中，我们可以统计每一个连通分量的顶点数目 n 和边数 m，如果 m>=n 则 return false；直到访问完所有的节点，return true。
2. 设有向图具有 n 个顶点，若该图的每个顶点的出度至少为 1，入度也至少为 1，则图中一定有回路。
3. 对有向图，利用拓扑排序算法，删除入度为 0 的顶点，直到剩下的顶点的入度都不为 0 为止，此时说明存在回路。

最小生成树：对于网络，其生成树中的边也带权，将生成树各边的权值总和称为生成树的权，并把权值最小的生成树称为最小生成树(Minimum Spanning Tree)
最小生成树的构造：（贪心法）
Prim 算法
基本思想：
将顶点分为两类：
生长点：已在生成树中的顶点
非生长点：未长到生成树中的顶点

① 首先从集合 V 中任取一顶点(例如取顶点 v0)放入集合 U 中这时 U={v0}，TE=NULL
② 然后在所有一个顶点在集合 U 里，另一个顶点在集合 V-U 里的边中，找出权值最小的边(u,v)(u∈U，v∈V-U)，将边加入 TE，并将顶点 v 加入集合 U
③ 重复上述操作直到 U=V 为止。这时 TE 中有 n-1 条边，T=(U，TE)就是 G 的一棵最小生成树

整个算法的时间复杂度为 O(n2)
算法适合于稠密图

Kruskal 算法

基本思想是：

1. 设 G=(V，E)是网络，最小生成树的初始状态为只有 n 个顶点而无边的非连通图 T=(V，φ)，T 中每个顶点自成为一个连通分量
2. 将集合 E 中的边按权递增顺序排列，从小到大依次选择顶点分别在两个连通分量中的边加入图 T，则原来的两个连通分量由于该边的连接而成为一个连通分量
3. 依次类推，直到 T 中所有顶点都在同一个连通分量上为止，该连通分量就是 G 的一棵最小生成树

Kruskal 算法用于稀疏图（顶点数多，而边数少）
Kruskal 算法需要判断回路，Prim 算法不需要判断回路

拓扑排序

AOV 网：如果在有向图中，用顶点表示活动，边表示活动间的某种约束关系，这样的有向图称为顶点活动网(ActivityOn Vertex network，简称 AOV 网)

拓扑序列：对于有向无环图 G = <V，E>，如果图中所有顶点可以排成一个线性序列，并且该线性序列具有以下性质：如果有向无环图 G 中存在顶点 vi 到 vj 的一条路径，那么在序列中顶点 vi 必在顶点 vj 之前。

拓扑排序：根据有向图建立拓扑序列

一个 AOV 网的拓扑序列不一定存在。

一个 AOV 网的拓扑序列不是唯一的。

拓扑排序的方法是：

（1）从有向图中选择一个入度为 0 的顶点将其输出。

（2）在有向图删除此顶点及其所有的出边。

反复执行以上两步，直到所有顶点都已经输出为止，此时整个拓扑排序完成；或者直到剩下的顶点的入度都不为 0 为止，此时说明有向图（AOV 网）中存在回路，拓扑排序无法再进行。

最短路径

路径长度：如果图是一个带权图，则路径长度为路径上各边的权值的总和。

最短路径长度：两个顶点间路径长度最短的那条路径称为两个顶点间的最短路径，其路径长度称为最短路径长度。

单源最短路径：给定带权有向图 G=(V,E)，其中每条边的权是非负实数。给定 V 中的一个顶点，称为源。要求计算从源到其它各个顶点的最短路径。

Dijkstra 算法基本思想（看 PPT）求单源最短路径

如果 v0 至 u 的最短路径经过 v1，那么 v0 到 v1 的路径也是 v0 到 v1 的最短路径。
按路径长度递增的次序产生最短路径。本质是贪心。时间复杂度 O(n^2)
Dijkstra 方法：求多源最短路径，O(n^3)
Floyd 算法（动态规划方法）
时间复杂度 O(n^3) 空间代价 O(n^2)
依次对某个顶点运用 Dijkstra 算法，与 Floyd 算法复杂度相同 O(n3)。
Dijkstra 算法要求没有负权边；Floyd 算法则仅仅要求没有负权环路就可以了。

第十章 算法设计与技术

穷举法：利用了计算机计算速度快且准确的特点，是最为朴素和有效的一种算法。
贪心法(Greedy)：分阶段挑选最优解，可以较快得到近似解。
分治法：把原问题分解为独立的子问题。
动态规划：把原问题分解为重叠的子问题。记住之前的答案。
回溯法、深度优先搜索(DFS)：通常用递归或者栈来实现。
广度优先搜索(BFS)：通常用队列实现。

穷举法

从问题可能的解空间中列举所有可能的解，并使用给定的检验条件进行判定
最直接最基本的方法

贪心算法

贪心法的基本思路是在对问题求解时，会把问题切分为不同的阶段，总是做出当前看起来最佳的选择，最终得到整个问题的最优解。
贪心策略通常只考虑局部最优的策略，最终得到全局的最优解。
贪心法的优势：算法简单，时间和空间复杂性低
贪心算法解题步骤
1. 分解：将原问题分解为若干独立的阶段
2. 解决：对于每个阶段依据贪心策略进行贪心选择，求出局部的最优解。
3. 合并：将各个阶段的解合并为原问题的一个可行解

经典的贪心算法
地图着色、信号灯颜色
哈夫曼算法
最小生成树算法：Prim 算法、Kruskal 算法
单源最短路径：Dijkstra 算法
拓扑排序

分治法

分治思想
对于简单问题，我们可以用一段较短的代码来获得问题的解。

但实际中很多问题都是比较复杂的，简短的代码是无法解决问题的。这时，就需要我们将一个复杂问题分解成若干个简单问题，并逐一求解；然后再把几个简单问题的解综合起来，得到整个复杂问题的解。————就是分而治之的思想。

分治法的基本思想是“分而治之”，把一个大问题分成两个或多个规模较小的与原问题相似的子问题

分治法的基本步骤

1. 分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题
2. 解决：若子问题规模较小，而容易被解决则直接解决，否则递归地解各个子问题
3. 合并：将各个子问题的解合并为原问题的解

经典的分治算法

二分检索

二路归并排序

汉诺塔

快速排序

幂乘算法

动态规划

没有准确的数学表达式和定义精确的算法,依赖分析者的经验和技巧。动态规划强调具体问题具体分析。

动态规划的基本原理

动态规划适用的问题特征

最优子结构、重叠子问题、无后效性

① 最优化原理：一个最优化策略具有这样的性质，不论过去的状态和决策如何，余下的诸决策必须构成最优策略。

② 无后效性：如果给定某一阶段的状态，则在这一阶段以后过程的发展不受该阶段以前各段状态的影响。

思路

① 将原问题分解为子问题

② 确定状态

③ 确定一些初始状态（边界状态）的值

④ 确定状态转移方程

动态规划是一种方法论，是一种解决问题的通用思路。

很多方法都含有动态规划的思想

KMP 算法：

利用 NEXT 数组记录最佳匹配位置•KMP 中求 NEXT 数组的算法：

已知 NEXT[0,…,i-1],求 NEXT[i]

DP 与分治法的异同

■同：原问题→子问题

•动态规划算法把求解过程变成一个多步判断过程，每一步对应于一个子问题（与分治法类似，将待求解问题分解成若干个子问题），先求解子问题，然后从这些子问题的解得到原问题的解。

■不同：子问题边界
分治法分解的子问题之间相互独立，且与原问题相同。
DP 分解得到的子问题往往不是互相独立的。
■DP 的思路：如果我们能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，这样就可以避免大量的重复计算，节省时间。

回溯法
•基本思想：在问题的解空间树中，按深度优先策略，从根结点出发，搜索解空间树。算法搜索至解空间树的任意一点时，先判断该结点是否包含问题的解，如果肯定不包含，则跳过以该结点为根的子树，逐层向其祖先结点回溯，否则，进入该子树，继续深搜。
•回溯法解题步骤：
针对所给问题，定义解空间。确定易于搜索的解空间结构。以深搜方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。