数据结构上机练习题

整理 徐振康

2015年8月11日14:37:4

# 说明

数据结构上机练习题

注：所有题目均来自于<http://zju.acmclub.com/>。

目录

[说明 I](#_Toc427067591)

[1323:算法2-1：集合union 1](#_Toc427067592)

[题目描述 1](#_Toc427067593)

[输入格式 1](#_Toc427067594)

[输出 1](#_Toc427067595)

[样例输入 1](#_Toc427067596)

[样例输出 1](#_Toc427067597)

[提示 2](#_Toc427067598)

[答案 2](#_Toc427067599)

[1325:算法2-3~2-6：Big Bang 8](#_Toc427067600)

[题目描述 8](#_Toc427067601)

[输入格式 10](#_Toc427067602)

[输出 10](#_Toc427067603)

[样例输入 10](#_Toc427067604)

[样例输出 11](#_Toc427067605)

[提示 11](#_Toc427067606)

[1326:算法2-8~2-11：链表的基本操作 12](#_Toc427067607)

[题目描述 12](#_Toc427067608)

[输入格式 13](#_Toc427067609)

[输出 13](#_Toc427067610)

[样例输入 13](#_Toc427067611)

[样例输出 14](#_Toc427067612)

[提示 14](#_Toc427067613)

[1327:算法2-13~2-16：静态链表 15](#_Toc427067614)

[题目描述 15](#_Toc427067615)

[输入格式 17](#_Toc427067616)

[输出 17](#_Toc427067617)

[样例输入 17](#_Toc427067618)

[样例输出 18](#_Toc427067619)

[提示 20](#_Toc427067620)

[1328:算法2-18~2-19：双向循环链表 20](#_Toc427067621)

[题目描述 20](#_Toc427067622)

[输入格式 22](#_Toc427067623)

[输出 22](#_Toc427067624)

[样例输入 23](#_Toc427067625)

[样例输出 23](#_Toc427067626)

[提示 23](#_Toc427067627)

[1329:算法2-23：一元多项式加法 23](#_Toc427067628)

[题目描述 23](#_Toc427067629)

[输入格式 24](#_Toc427067630)

[输出 24](#_Toc427067631)

[样例输入 25](#_Toc427067632)

[样例输出 25](#_Toc427067633)

[提示 25](#_Toc427067634)

[1787:求最大值 25](#_Toc427067635)

[题目描述 25](#_Toc427067636)

[输入格式 25](#_Toc427067637)

[输出 25](#_Toc427067638)

[样例输入 25](#_Toc427067639)

[样例输出 25](#_Toc427067640)

[提示 25](#_Toc427067641)

[1330:算法3-1：八进制数 26](#_Toc427067642)

[题目描述 26](#_Toc427067643)

[输入格式 26](#_Toc427067644)

[输出 26](#_Toc427067645)

[样例输入 26](#_Toc427067646)

[样例输出 26](#_Toc427067647)

[提示 26](#_Toc427067648)

[1741:算法3-2：行编辑程序 27](#_Toc427067649)

[题目描述 27](#_Toc427067650)

[输入格式 28](#_Toc427067651)

[输出 28](#_Toc427067652)

[样例输入 28](#_Toc427067653)

[样例输出 28](#_Toc427067654)

[提示 28](#_Toc427067655)

[1742:算法3-3：迷宫 29](#_Toc427067656)

[题目描述 29](#_Toc427067657)

[输入格式 29](#_Toc427067658)

[输出 29](#_Toc427067659)

[样例输入 29](#_Toc427067660)

[样例输出 30](#_Toc427067661)

[提示 30](#_Toc427067662)

[1102:迷宫问题 30](#_Toc427067663)

[题目描述 30](#_Toc427067664)

[输入格式 31](#_Toc427067665)

[输出 31](#_Toc427067666)

[样例输入 31](#_Toc427067667)

[样例输出 31](#_Toc427067668)

[提示 31](#_Toc427067669)

[1743:算法3-4：表达式求值 31](#_Toc427067670)

[题目描述 31](#_Toc427067671)

[输入格式 33](#_Toc427067672)

[输出 33](#_Toc427067673)

[样例输入 33](#_Toc427067674)

[样例输出 33](#_Toc427067675)

[提示 33](#_Toc427067676)

[1744:算法3-5：n阶Hanoi塔问题 33](#_Toc427067677)

[题目描述 33](#_Toc427067678)

[输入格式 34](#_Toc427067679)

[输出 34](#_Toc427067680)

[样例输入 34](#_Toc427067681)

[样例输出 34](#_Toc427067682)

[提示 35](#_Toc427067683)

[1745:算法3-7：银行排队 35](#_Toc427067684)

[题目描述 35](#_Toc427067685)

[输入格式 35](#_Toc427067686)

[输出 35](#_Toc427067687)

[样例输入 35](#_Toc427067688)

[样例输出 35](#_Toc427067689)

[提示 35](#_Toc427067690)

[1909:堆栈的使用 36](#_Toc427067691)

[题目描述 36](#_Toc427067692)

[输入格式 36](#_Toc427067693)

[输出 36](#_Toc427067694)

[样例输入 36](#_Toc427067695)

[样例输出 36](#_Toc427067696)

[提示 36](#_Toc427067697)

[2113:字符统计 36](#_Toc427067698)

[题目描述 36](#_Toc427067699)

[输入格式 37](#_Toc427067700)

[输出 37](#_Toc427067701)

[样例输入 37](#_Toc427067702)

[样例输出 37](#_Toc427067703)

[提示 37](#_Toc427067704)

[2114:字符序列模式识别 37](#_Toc427067705)

[题目描述 37](#_Toc427067706)

[输入格式 37](#_Toc427067707)

[输出 37](#_Toc427067708)

[样例输入 37](#_Toc427067709)

[样例输出 37](#_Toc427067710)

[提示 37](#_Toc427067711)

[1747:算法4-2：字符串连接 37](#_Toc427067712)

[题目描述 37](#_Toc427067713)

[输入格式 38](#_Toc427067714)

[输出 38](#_Toc427067715)

[样例输入 38](#_Toc427067716)

[样例输出 38](#_Toc427067717)

[提示 38](#_Toc427067718)

[1748:算法4-4：字符串插入 39](#_Toc427067719)

[题目描述 39](#_Toc427067720)

[输入格式 39](#_Toc427067721)

[输出 39](#_Toc427067722)

[样例输入 39](#_Toc427067723)

[样例输出 39](#_Toc427067724)

[提示 39](#_Toc427067725)

[1749:算法4-5：求子串位置的定位函数 39](#_Toc427067726)

[题目描述 39](#_Toc427067727)

[输入格式 40](#_Toc427067728)

[输出 40](#_Toc427067729)

[样例输入 40](#_Toc427067730)

[样例输出 40](#_Toc427067731)

[提示 41](#_Toc427067732)

[1750:算法4-7：KMP算法中的模式串移动数组 41](#_Toc427067733)

[题目描述 41](#_Toc427067734)

[输入格式 41](#_Toc427067735)

[输出 41](#_Toc427067736)

[样例输入 42](#_Toc427067737)

[样例输出 42](#_Toc427067738)

[提示 42](#_Toc427067739)

[1751:算法4-6：KMP字符串模式匹配算法实现 42](#_Toc427067740)

[题目描述 42](#_Toc427067741)

[输入格式 43](#_Toc427067742)

[输出 43](#_Toc427067743)

[样例输入 43](#_Toc427067744)

[样例输出 43](#_Toc427067745)

[提示 43](#_Toc427067746)

[1785:字符串连接 43](#_Toc427067747)

[题目描述 43](#_Toc427067748)

[输入格式 43](#_Toc427067749)

[输出 43](#_Toc427067750)

[样例输入 43](#_Toc427067751)

[样例输出 43](#_Toc427067752)

[提示 43](#_Toc427067753)

[1791:后缀子串排序 43](#_Toc427067754)

[题目描述 43](#_Toc427067755)

[输入格式 44](#_Toc427067756)

[输出 44](#_Toc427067757)

[样例输入 44](#_Toc427067758)

[样例输出 44](#_Toc427067759)

[提示 44](#_Toc427067760)

[1800:统计字符 44](#_Toc427067761)

[题目描述 44](#_Toc427067762)

[输入格式 44](#_Toc427067763)

[输出 44](#_Toc427067764)

[样例输入 45](#_Toc427067765)

[样例输出 45](#_Toc427067766)

[提示 45](#_Toc427067767)

[1804:字符串的反码 45](#_Toc427067768)

[题目描述 45](#_Toc427067769)

[输入格式 45](#_Toc427067770)

[输出 45](#_Toc427067771)

[样例输入 45](#_Toc427067772)

[样例输出 45](#_Toc427067773)

[提示 45](#_Toc427067774)

[1805:首字母大写 45](#_Toc427067775)

[题目描述 45](#_Toc427067776)

[输入格式 46](#_Toc427067777)

[输出 46](#_Toc427067778)

[样例输入 46](#_Toc427067779)

[样例输出 46](#_Toc427067780)

[提示 46](#_Toc427067781)

[1808:字符串的查找删除 46](#_Toc427067782)

[题目描述 46](#_Toc427067783)

[输入格式 46](#_Toc427067784)

[输出 46](#_Toc427067785)

[样例输入 46](#_Toc427067786)

[样例输出 46](#_Toc427067787)

[提示 46](#_Toc427067788)

[1971:字符串链接 47](#_Toc427067789)

[题目描述 47](#_Toc427067790)

[输入格式 47](#_Toc427067791)

[输出 47](#_Toc427067792)

[样例输入 47](#_Toc427067793)

[样例输出 47](#_Toc427067794)

[提示 47](#_Toc427067795)

[1983:字符串处理 47](#_Toc427067796)

[题目描述 47](#_Toc427067797)

[输入格式 47](#_Toc427067798)

[输出 47](#_Toc427067799)

[样例输入 47](#_Toc427067800)

[样例输出 47](#_Toc427067801)

[提示 47](#_Toc427067802)

[1995:反序字符串 47](#_Toc427067803)

[2025:比较字符串 48](#_Toc427067804)

[题目描述 48](#_Toc427067805)

[输入格式 48](#_Toc427067806)

[输出 48](#_Toc427067807)

[样例输入 48](#_Toc427067808)

[样例输出 48](#_Toc427067809)

[提示 48](#_Toc427067810)

[2057:字符串排序 48](#_Toc427067811)

[题目描述 48](#_Toc427067812)

[输入格式 48](#_Toc427067813)

[输出 48](#_Toc427067814)

[样例输入 48](#_Toc427067815)

[样例输出 49](#_Toc427067816)

[提示 49](#_Toc427067817)

[2110:矩阵对角求和 49](#_Toc427067818)

[题目描述 49](#_Toc427067819)

[输入格式 49](#_Toc427067820)

[输出 49](#_Toc427067821)

[样例输入 49](#_Toc427067822)

[样例输出 49](#_Toc427067823)

[提示 49](#_Toc427067824)

[1752:算法5-1：稀疏矩阵转置 49](#_Toc427067825)

[题目描述 49](#_Toc427067826)

[输入格式 50](#_Toc427067827)

[输出 50](#_Toc427067828)

[样例输入 50](#_Toc427067829)

[样例输出 50](#_Toc427067830)

[提示 50](#_Toc427067831)

[1970:计算两个矩阵的乘积 51](#_Toc427067832)

[题目描述 51](#_Toc427067833)

[输入格式 51](#_Toc427067834)

[输出 51](#_Toc427067835)

[样例输入 51](#_Toc427067836)

[样例输出 51](#_Toc427067837)

[提示 51](#_Toc427067838)

[2009:矩阵最大值 51](#_Toc427067839)

[题目描述 51](#_Toc427067840)

[输入格式 51](#_Toc427067841)

[输出 52](#_Toc427067842)

[样例输入 52](#_Toc427067843)

[样例输出 52](#_Toc427067844)

[提示 52](#_Toc427067845)

[2010:矩阵转置 52](#_Toc427067846)

[题目描述 52](#_Toc427067847)

[输入格式 52](#_Toc427067848)

[输出 53](#_Toc427067849)

[样例输入 53](#_Toc427067850)

[样例输出 53](#_Toc427067851)

[提示 53](#_Toc427067852)

[2029:对称矩阵 53](#_Toc427067853)

[题目描述 53](#_Toc427067854)

[输入格式 53](#_Toc427067855)

[输出 53](#_Toc427067856)

[样例输入 53](#_Toc427067857)

[样例输出 54](#_Toc427067858)

[提示 54](#_Toc427067859)

[1757:算法6-1~6-4：二叉链表存储的二叉树 54](#_Toc427067860)

[题目描述 54](#_Toc427067861)

[输入格式 56](#_Toc427067862)

[输出 56](#_Toc427067863)

[样例输入 56](#_Toc427067864)

[样例输出 56](#_Toc427067865)

[提示 56](#_Toc427067866)

[1758:算法6-5~6-7：线索二叉树 56](#_Toc427067867)

[题目描述 56](#_Toc427067868)

[输入格式 57](#_Toc427067869)

[输出 58](#_Toc427067870)

[样例输入 58](#_Toc427067871)

[样例输出 58](#_Toc427067872)

[提示 58](#_Toc427067873)

[1759:算法6-8~6-11：用树表示的等价问题 58](#_Toc427067874)

[题目描述 58](#_Toc427067875)

[输入格式 59](#_Toc427067876)

[输出 59](#_Toc427067877)

[样例输入 59](#_Toc427067878)

[样例输出 59](#_Toc427067879)

[提示 59](#_Toc427067880)

[1760:算法6-12：自底向上的赫夫曼编码 60](#_Toc427067881)

[题目描述 60](#_Toc427067882)

[输入格式 61](#_Toc427067883)

[样例输入 61](#_Toc427067884)

[样例输出 61](#_Toc427067885)

[1814:剩下的树 61](#_Toc427067886)

[题目描述 61](#_Toc427067887)

[输入格式 61](#_Toc427067888)

[输出 61](#_Toc427067889)

[样例输入 61](#_Toc427067890)

[样例输出 62](#_Toc427067891)

[1905:二叉树 62](#_Toc427067892)

[输入格式 62](#_Toc427067893)

[输出 62](#_Toc427067894)

[样例输入 62](#_Toc427067895)

[样例输出 63](#_Toc427067896)

[1910:二叉树遍历 63](#_Toc427067897)

[题目描述 63](#_Toc427067898)

[输入格式 63](#_Toc427067899)

[输出 63](#_Toc427067900)

[样例输入 63](#_Toc427067901)

[样例输出 63](#_Toc427067902)

[1914:树查找 63](#_Toc427067903)

[题目描述 63](#_Toc427067904)

[输入格式 63](#_Toc427067905)

[输出 64](#_Toc427067906)

[样例输入 64](#_Toc427067907)

[样例输出 64](#_Toc427067908)

[1772:算法9-5~9-8：二叉排序树的基本操作 64](#_Toc427067909)

[题目描述 64](#_Toc427067910)

[输入格式 65](#_Toc427067911)

[输出 66](#_Toc427067912)

[样例输入 66](#_Toc427067913)

[样例输出 66](#_Toc427067914)

[提示 66](#_Toc427067915)

[1919:二叉排序树 66](#_Toc427067916)

[题目描述 66](#_Toc427067917)

[输入格式 66](#_Toc427067918)

[输出 66](#_Toc427067919)

[样例输入 66](#_Toc427067920)

[样例输出 66](#_Toc427067921)

[提示 67](#_Toc427067922)

[1920:二叉搜索树 67](#_Toc427067923)

[题目描述 67](#_Toc427067924)

[输入格式 67](#_Toc427067925)

[输出 67](#_Toc427067926)

[样例输入 67](#_Toc427067927)

[样例输出 67](#_Toc427067928)

[提示 67](#_Toc427067929)

[2014:二叉树遍历 67](#_Toc427067930)

[题目描述 67](#_Toc427067931)

[输入格式 68](#_Toc427067932)

[输出 68](#_Toc427067933)

[样例输入 68](#_Toc427067934)

[样例输出 68](#_Toc427067935)

[提示 68](#_Toc427067936)

[1762:算法7-4,7-5：图的遍历——深度优先搜索 68](#_Toc427067937)

[题目描述 68](#_Toc427067938)

[输入格式 69](#_Toc427067939)

[输出 69](#_Toc427067940)

[样例输入 69](#_Toc427067941)

[样例输出 69](#_Toc427067942)

[提示 69](#_Toc427067943)

[1763:算法7-6：图的遍历——广度优先搜索 70](#_Toc427067944)

[题目描述 70](#_Toc427067945)

[输入格式 70](#_Toc427067946)

[输出 70](#_Toc427067947)

[样例输入 70](#_Toc427067948)

[样例输出 71](#_Toc427067949)

[1764:算法7-7,7-8：无向图的连通分量和生成树 71](#_Toc427067950)

[题目描述 71](#_Toc427067951)

[输入格式 72](#_Toc427067952)

[输出 72](#_Toc427067953)

[样例输入 72](#_Toc427067954)

[样例输出 72](#_Toc427067955)

[提示 72](#_Toc427067956)

[1765:算法7-9：最小生成树 73](#_Toc427067957)

[题目描述 73](#_Toc427067958)

[输入格式 73](#_Toc427067959)

[输出 74](#_Toc427067960)

[样例输入 74](#_Toc427067961)

[样例输出 74](#_Toc427067962)

[提示 74](#_Toc427067963)

[1766:算法7-10,7-11：关节点和重连通分量 74](#_Toc427067964)

[题目描述 74](#_Toc427067965)

[输入格式 75](#_Toc427067966)

[输出 75](#_Toc427067967)

[样例输入 75](#_Toc427067968)

[样例输出 75](#_Toc427067969)

[提示 75](#_Toc427067970)

[1767:算法7-12：有向无环图的拓扑排序 76](#_Toc427067971)

[题目描述 76](#_Toc427067972)

[输入格式 76](#_Toc427067973)

[输出 77](#_Toc427067974)

[样例输入 77](#_Toc427067975)

[样例输出 77](#_Toc427067976)

[1768:算法7-15：迪杰斯特拉最短路径算法 77](#_Toc427067977)

[题目描述 77](#_Toc427067978)

[输入格式 78](#_Toc427067979)

[输出 78](#_Toc427067980)

[样例输入 78](#_Toc427067981)

[样例输出 79](#_Toc427067982)

[提示 79](#_Toc427067983)

[1769:算法7-16：弗洛伊德最短路径算法 79](#_Toc427067984)

[题目描述 79](#_Toc427067985)

[输入格式 80](#_Toc427067986)

[输出 80](#_Toc427067987)

[样例输入 80](#_Toc427067988)

[样例输出 80](#_Toc427067989)

[提示 80](#_Toc427067990)

[1813:还是畅通工程 81](#_Toc427067991)

[题目描述 81](#_Toc427067992)

[输入格式 81](#_Toc427067993)

[输出 81](#_Toc427067994)

[样例输入 81](#_Toc427067995)

[样例输出 82](#_Toc427067996)

[1953:畅通工程 82](#_Toc427067997)

[题目描述 82](#_Toc427067998)

[输入格式 82](#_Toc427067999)

[输出 82](#_Toc427068000)

[样例输入 82](#_Toc427068001)

[样例输出 82](#_Toc427068002)

[提示 82](#_Toc427068003)

[1954:畅通工程 83](#_Toc427068004)

[题目描述 83](#_Toc427068005)

[输入格式 83](#_Toc427068006)

[输出 83](#_Toc427068007)

[样例输入 83](#_Toc427068008)

[样例输出 83](#_Toc427068009)

[1955:继续畅通工程 83](#_Toc427068010)

[题目描述 83](#_Toc427068011)

[输入格式 83](#_Toc427068012)

[输出 83](#_Toc427068013)

[样例输入 84](#_Toc427068014)

[样例输出 84](#_Toc427068015)

[提示 84](#_Toc427068016)

[1908:连通图 84](#_Toc427068017)

[题目描述 84](#_Toc427068018)

[输入格式 84](#_Toc427068019)

[输出 84](#_Toc427068020)

[样例输入 84](#_Toc427068021)

[样例输出 85](#_Toc427068022)

[提示 85](#_Toc427068023)

[1774:算法10-1：直接插入排序 85](#_Toc427068024)

[题目描述 85](#_Toc427068025)

[输入格式 85](#_Toc427068026)

[输出 85](#_Toc427068027)

[样例输入 85](#_Toc427068028)

[样例输出 85](#_Toc427068029)

[提示 85](#_Toc427068030)

[1775:算法10-2：折半插入排序 86](#_Toc427068031)

[题目描述 86](#_Toc427068032)

[输入格式 86](#_Toc427068033)

[输出 86](#_Toc427068034)

[样例输入 86](#_Toc427068035)

[样例输出 86](#_Toc427068036)

[提示 86](#_Toc427068037)

[1776:算法10-4,10-5：希尔排序 86](#_Toc427068038)

[题目描述 86](#_Toc427068039)

[输入格式 87](#_Toc427068040)

[输出 87](#_Toc427068041)

[样例输入 87](#_Toc427068042)

[提示 87](#_Toc427068043)

[1777:算法10-6~10-8：快速排序 88](#_Toc427068044)

[题目描述 88](#_Toc427068045)

[输入格式 88](#_Toc427068046)

[输出 88](#_Toc427068047)

[样例输入 88](#_Toc427068048)

[样例输出 88](#_Toc427068049)

[提示 89](#_Toc427068050)

[1778:算法10-9：简单选择排序 89](#_Toc427068051)

[题目描述 89](#_Toc427068052)

[输入格式 89](#_Toc427068053)

[输出 89](#_Toc427068054)

[样例输入 89](#_Toc427068055)

[样例输出 89](#_Toc427068056)

[1779:算法10-10,10-11：堆排序 90](#_Toc427068057)

[题目描述 90](#_Toc427068058)

[输入格式 90](#_Toc427068059)

[输出 90](#_Toc427068060)

[样例输出 91](#_Toc427068061)

[提示 91](#_Toc427068062)

[1780:算法10-12~10-14：归并排序 91](#_Toc427068063)

[题目描述 91](#_Toc427068064)

[输入格式 92](#_Toc427068065)

[输出 92](#_Toc427068066)

[样例输入 92](#_Toc427068067)

[样例输出 92](#_Toc427068068)

[提示 92](#_Toc427068069)

[1781:算法10-15~10-17：基数排序 92](#_Toc427068070)

[题目描述 92](#_Toc427068071)

[输入格式 93](#_Toc427068072)

[输出 93](#_Toc427068073)

[样例输入 93](#_Toc427068074)

[样例输出 93](#_Toc427068075)

[提示 93](#_Toc427068076)

[1782:谁是你的潜在朋友 94](#_Toc427068077)

[题目描述 94](#_Toc427068078)

[输入格式 94](#_Toc427068079)

[输出 94](#_Toc427068080)

[样例输入 94](#_Toc427068081)

[样例输出 94](#_Toc427068082)

[1791:后缀子串排序 94](#_Toc427068083)

[题目描述 94](#_Toc427068084)

[输入格式 95](#_Toc427068085)

[输出 95](#_Toc427068086)

[样例输入 95](#_Toc427068087)

[样例输出 95](#_Toc427068088)

[提示 95](#_Toc427068089)

[1923:排序 95](#_Toc427068090)

[题目描述 95](#_Toc427068091)

[输入格式 95](#_Toc427068092)

[输出 95](#_Toc427068093)

[样例输入 95](#_Toc427068094)

[样例输出 95](#_Toc427068095)

[1924:成绩排序 96](#_Toc427068096)

[题目描述 96](#_Toc427068097)

[输入格式 96](#_Toc427068098)

[输出 96](#_Toc427068099)

[样例输入 96](#_Toc427068100)

[样例输出 96](#_Toc427068101)

[提示 96](#_Toc427068102)

[1925:特殊排序 96](#_Toc427068103)

[题目描述 96](#_Toc427068104)

[输入格式 96](#_Toc427068105)

[输出 96](#_Toc427068106)

[样例输入 96](#_Toc427068107)

[样例输出 96](#_Toc427068108)

[提示 97](#_Toc427068109)

[1926:EXCEL排序 97](#_Toc427068110)

[题目描述 97](#_Toc427068111)

[输入格式 97](#_Toc427068112)

[输出 97](#_Toc427068113)

[样例输入 97](#_Toc427068114)

[样例输出 97](#_Toc427068115)

[提示 98](#_Toc427068116)

[1927:字符串内排序 98](#_Toc427068117)

[题目描述 98](#_Toc427068118)

[输入格式 98](#_Toc427068119)

[输出 98](#_Toc427068120)

[提示 98](#_Toc427068121)

[1951:大整数排序 98](#_Toc427068122)

[题目描述 98](#_Toc427068123)

[输入格式 98](#_Toc427068124)

[输出 98](#_Toc427068125)

[样例输入 98](#_Toc427068126)

[样例输出 99](#_Toc427068127)

[提示 99](#_Toc427068128)

[1998:成绩排序 99](#_Toc427068129)

[题目描述 99](#_Toc427068130)

[输入格式 99](#_Toc427068131)

[输出 99](#_Toc427068132)

[样例输入 99](#_Toc427068133)

[样例输出 99](#_Toc427068134)

[提示 99](#_Toc427068135)

[2005:字符排序 99](#_Toc427068136)

[题目描述 99](#_Toc427068137)

[输入格式 99](#_Toc427068138)

[输出 99](#_Toc427068139)

[样例输入 100](#_Toc427068140)

[样例输出 100](#_Toc427068141)

[2007:成绩排序 100](#_Toc427068142)

[题目描述 100](#_Toc427068143)

[输入格式 100](#_Toc427068144)

[输出 100](#_Toc427068145)

[样例输入 100](#_Toc427068146)

[样例输出 100](#_Toc427068147)

[提示 100](#_Toc427068148)

[2040:数组排序 100](#_Toc427068149)

[题目描述 100](#_Toc427068150)

[输入格式 101](#_Toc427068151)

[输出 101](#_Toc427068152)

[样例输入 101](#_Toc427068153)

[提示 101](#_Toc427068154)

[2057:字符串排序 101](#_Toc427068155)

[题目描述 101](#_Toc427068156)

[输入格式 101](#_Toc427068157)

[输出 101](#_Toc427068158)

[样例输入 101](#_Toc427068159)

[样例输出 102](#_Toc427068160)

[提示 102](#_Toc427068161)

[2062:姓名排序 102](#_Toc427068162)

[题目描述 102](#_Toc427068163)

[输入格式 102](#_Toc427068164)

[输出 102](#_Toc427068165)

[样例输入 102](#_Toc427068166)

[样例输出 102](#_Toc427068167)

[提示 102](#_Toc427068168)

[2080:整数奇偶排序 102](#_Toc427068169)

[题目描述 102](#_Toc427068170)

[输入格式 103](#_Toc427068171)

[输出 103](#_Toc427068172)

[样例输入 103](#_Toc427068173)

[样例输出 103](#_Toc427068174)

[提示 103](#_Toc427068175)

[1770:算法9-1：静态表的顺序查找 103](#_Toc427068176)

[题目描述 103](#_Toc427068177)

[输入格式 103](#_Toc427068178)

[输出 104](#_Toc427068179)

[样例输入 104](#_Toc427068180)

[样例输出 104](#_Toc427068181)

[提示 104](#_Toc427068182)

[1772:算法9-5~9-8：二叉排序树的基本操作 104](#_Toc427068183)

[题目描述 104](#_Toc427068184)

[输入格式 105](#_Toc427068185)

[输出 106](#_Toc427068186)

[样例输入 106](#_Toc427068187)

[样例输出 106](#_Toc427068188)

[提示 106](#_Toc427068189)

[1773:算法9-9~9-12：平衡二叉树的基本操作 106](#_Toc427068190)

[题目描述 106](#_Toc427068191)

[输入格式 107](#_Toc427068192)

[输出 107](#_Toc427068193)

[样例输入 107](#_Toc427068194)

[样例输出 107](#_Toc427068195)

[2112:查找最小的k个元素 107](#_Toc427068196)

[题目描述 107](#_Toc427068197)

[输入格式 107](#_Toc427068198)

[输出 107](#_Toc427068199)

[样例输入 107](#_Toc427068200)

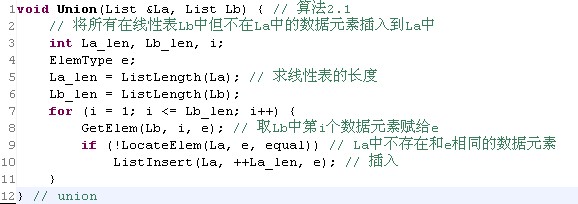
[样例输出 108](#_Toc427068201)

[提示 108](#_Toc427068202)

# 1323:算法2-1：集合union

## 题目描述

假设利用两个线性表LA和LB分别表示两个集合A和B（即：线性表中的数据元素即为集合中的成员），现要求一个新的集合A=A∪B。这就要求对线性表做如下操作：扩大线性表LA，将存在于线性表LB中而不存在于线性表LA中的数据元素插入到线性表LA中去。只要从线性表LB中依次取得每个元素，并依值在线性表LA中进行查访，若不存在，则插入之。上述操作过程可用下列算法描述之。



图：将两个列表合并的算法（C/C++描述）

上图算法中，在第8行取得集合B中的元素，然后再在第10行插入到集合A中。你的任务是先输出集合A和集合B中的元素，每个集合在一行中输出。然后每次在将集合B中的元素取出插入到集合A尾部后输出集合A中的元素。当然你的代码可以和上面的代码不一样，只要有相同的输出即可。

## 输入格式

有多组测试数据，每组测试数据占两行。第一行是集合A，第一个整数m（0<m<=100）代表集合A起始有m个元素，后面有m个整数，代表A中的元素。第二行是集合B，第一个整数n(0<n<=100)代表集合B起始有n个元素，后面有n个整数，代表B中的元素。每行中整数之间用一个空格隔开。

## 输出

每组测试数据输出n+2行：前两行分别输出集合A、集合B中的数据，后面n行是每次从B中取出元素插入到A尾部后的集合A。每行整数之间用一个空格隔开，每组测试数据之间用一行空行隔开。

## 样例输入

5 1 5 2 6 3  
3 1 7 9  
1 3  
2 2 7  
4 2 5 1 4  
4 1 2 4 5

## 样例输出

1 5 2 6 3  
1 7 9  
1 5 2 6 3  
1 5 2 6 3 7  
1 5 2 6 3 7 9  
  
3  
2 7  
3 2  
3 2 7  
  
2 5 1 4  
1 2 4 5  
2 5 1 4  
2 5 1 4  
2 5 1 4  
2 5 1 4

## 提示

提示：

1、使用数组时，给集合 A 分配的空间不小于200。因为将 B 中的元素添加到 A 中时，可能会超过 100 了。

2、利用 scanf("%d",&m) != EOF 来判断是否还有输入数据。

3、一个细节问题就是题目要求输出的格式是每行中元素之间用一个空格隔开，每组输出间用一个空行隔开。也就是说4个元素间只有3个空格，2组输出间只有1个空行。处理方法都一样。两种方法：一是除了第一个元素，后面的每个元素之前输出个空格；二是除了最后一个元素，前面的每个元素之后都输出一个空格。我往往采用第一种方式，因为许多编程语言中的数组都是从0开始的，而0正是判断语句中的“假”（当然Java中不是这样的）。

总结：

本题考查的是线性表的基本操作。实际上只考察了遍历和添加操作。虽然算法中使用的是“插入”，然而本题只要求插入到链表的尾部，因而只是添加而已。线性表按存储结构分为顺序表和链表。顺序表在插入时往往需要移动某些元素，而移动元素需要消耗大量时间。如果插入操作次数很多的话，采用链表会好些。但由于只是插入到线性表的尾部，因而也不必移动元素。所以采用顺序表解本题也不失为一个好方法。

如果采用顺序表，事先需要分配足够的内存。题目中 m 和 n 都是不大于100的，是不是给两个顺序表（数组实现）分配100的内存就够了呢？答案是否定的。因为将集合 B 添加到集合 A 中很可能就超过100个元素了。

还有，题目没有给定多少组测试数据，我们的方法就是判断是否读到了文件结尾。利用 scanf("%d",&m) != EOF 来作判断即可。

对于解本题的算法，题目描述中已经有了，我就不再赘述。除了基本操作以外，还要看怎么输出。实际上就是在每次插入后使用一个循环将集合 A 中的所有元素输出即可。

## 答案

#include<string.h>

#include<malloc.h> /\* malloc()等 \*/

#include<stdio.h> /\* EOF(=^Z或F6),NULL \*/

#include<stdlib.h> /\* atoi() \*/

#include<math.h> /\* floor(),ceil(),abs() \*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define LIST\_INIT\_SIZE 10 /\* 线性表存储空间的初始分配量 \*/

#define LISTINCREMENT 2 /\* 线性表存储空间的分配增量 \*/

typedef int ElemType;

typedef int Status;

typedef int Boolean;

typedef struct

{

ElemType \*elem; /\* 存储空间基址 \*/

int length; /\* 当前长度 \*/

int listsize; /\* 当前分配的存储容量(以sizeof(ElemType)为单位) \*/

} SqList;

Status InitList(SqList \*L) /\* 算法2.3 \*/

{

/\* 操作结果：构造一个空的顺序线性表 \*/

(\*L).elem=(ElemType\*)malloc(LIST\_INIT\_SIZE\*sizeof(ElemType));

if(!(\*L).elem)

exit(OVERFLOW); /\* 存储分配失败 \*/

(\*L).length=0; /\* 空表长度为0 \*/

(\*L).listsize=LIST\_INIT\_SIZE; /\* 初始存储容量 \*/

return OK;

}

Status ListInsert(SqList \*L,int i,ElemType e) /\* 算法2.4 \*/

{

/\* 初始条件：顺序线性表L已存在，1≤i≤ListLength(L)+1 \*/

/\* 操作结果：在L中第i个位置之前插入新的数据元素e，L的长度加1 \*/

ElemType \*newbase,\*q,\*p;

if(i<1||i>(\*L).length+1) /\* i值不合法 \*/

return ERROR;

if((\*L).length>=(\*L).listsize) /\* 当前存储空间已满,增加分配 \*/

{

newbase=(ElemType \*)realloc((\*L).elem,((\*L).listsize+LISTINCREMENT)\*sizeof(ElemType));

if(!newbase)

exit(OVERFLOW); /\* 存储分配失败 \*/

(\*L).elem=newbase; /\* 新基址 \*/

(\*L).listsize+=LISTINCREMENT; /\* 增加存储容量 \*/

}

q=(\*L).elem+i-1; /\* q为插入位置 \*/

for(p=(\*L).elem+(\*L).length-1; p>=q; --p) /\* 插入位置及之后的元素右移 \*/

\*(p+1)=\*p;

\*q=e; /\* 插入e \*/

++(\*L).length; /\* 表长增1 \*/

return OK;

}

Status ListTraverse(SqList L,void(\*vi)(ElemType\*))

{

/\* 初始条件：顺序线性表L已存在 \*/

/\* 操作结果：依次对L的每个数据元素调用函数vi()。一旦vi()失败，则操作失败 \*/

/\* vi()的形参加'&'，表明可通过调用vi()改变元素的值 \*/

ElemType \*p;

int i;

p=L.elem;

for(i=1; i<=L.length; i++)

{

if(i!=1)

printf(" ");

vi(p++);

}

printf("\n");

return OK;

}

int ListLength(SqList L)

{

/\* 初始条件：顺序线性表L已存在。操作结果：返回L中数据元素个数 \*/

return L.length;

}

Status GetElem(SqList L,int i,ElemType \*e)

{

/\* 初始条件：顺序线性表L已存在，1≤i≤ListLength(L) \*/

/\* 操作结果：用e返回L中第i个数据元素的值 \*/

if(i<1||i>L.length)

exit(ERROR);

\*e=\*(L.elem+i-1);

return OK;

}

int LocateElem(SqList L,ElemType e,Status(\*compare)(ElemType,ElemType))

{

/\* 初始条件：顺序线性表L已存在，compare()是数据元素判定函数(满足为1,否则为0) \*/

/\* 操作结果：返回L中第1个与e满足关系compare()的数据元素的位序。 \*/

/\* 若这样的数据元素不存在，则返回值为0。算法2.6 \*/

ElemType \*p;

int i=1; /\* i的初值为第1个元素的位序 \*/

p=L.elem; /\* p的初值为第1个元素的存储位置 \*/

while(i<=L.length&&!compare(\*p++,e))

++i;

if(i<=L.length)

return i;

else

return 0;

}

Status equal(ElemType c1,ElemType c2)

{

/\* 判断是否相等的函数，Union()用到 \*/

if(c1==c2)

return TRUE;

else

return FALSE;

}

void print(ElemType \*c)

{

printf("%d",\*c);

}

void Union(SqList \*La,SqList Lb) /\* 算法2.1 \*/

{

/\* 将所有在线性表Lb中但不在La中的数据元素插入到La中 \*/

ElemType e;

int La\_len,Lb\_len;

int i;

La\_len=ListLength(\*La); /\* 求线性表的长度 \*/

Lb\_len=ListLength(Lb);

for(i=1; i<=Lb\_len; i++)

{

GetElem(Lb,i,&e); /\* 取Lb中第i个数据元素赋给e \*/

if(!LocateElem(\*La,e,equal)) /\* La中不存在和e相同的元素,则插入之 \*/

ListInsert(La,++La\_len,e);

ListTraverse(\*La,print);

}

}

int main()

{

int n,m,a[105],b[105],cas = 0;

while(~scanf("%d",&n))

{

int j;

if(cas++)

printf("\n");

for(j = 0; j<n; j++)

scanf("%d",&a[j]);

scanf("%d",&m);

for(j = 0; j<m; j++)

scanf("%d",&b[j]);

SqList La,Lb;

InitList(&La);

for(j=0; j<n; j++)

ListInsert(&La,j+1,a[j]);

InitList(&Lb);

for(j=0; j<m; j++)

ListInsert(&Lb,j+1,b[j]);

ListTraverse(La,print);

ListTraverse(Lb,print);

Union(&La,Lb);

}

return 0;

}

1324:算法2-2：有序线性表的有序合并

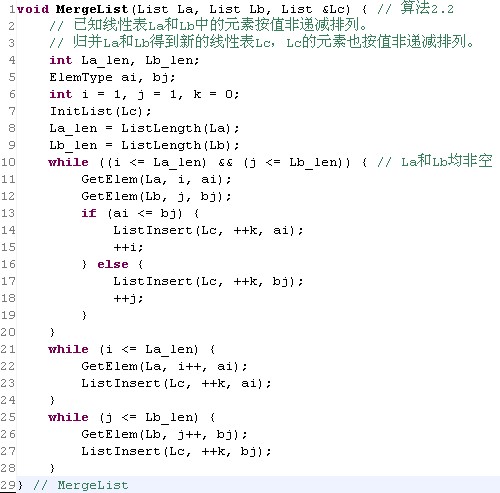
题目描述

已知线性表 LA 和 LB 中的数据元素按值非递减有序排列，现要求将 LA 和 LB 归并为一个新的线性表 LC， 且 LC 中的数据元素仍然按值非递减有序排列。例如，设LA=(3,5,8,11) ,LB=(2,6,8,9,11,15,20) 则

LC=(2,3,6,6,8,8,9,11,11,15,20)

算法描述如下：

从上述问题要求可知，LC中的数据元素或是LA中的数据元素，或是LB中的数据元素，则只要先设LC为空表，然后将LA或LB中的元素逐个插入到LC中即可。为使LC中元素按值非递减有序排列，可设两个指针 i 和 j 分别指向LA和LB中某个元素，若设 i 当前所指的元素为 a，j 所指的元素为 b，则当前应插入到 LC 中的元素 c 为 c = a < b ? a : b显然，指针 i 和 j 的初值均为1（实际写代码时往往是从 0 开始的），在所指元素插入 LC 之后，在 LA 或者 LB 中顺序后移。上述归并算法如下图：



图：有序列表有序插入算法

输入格式

有多组测试数据，每组测试数据占两行。第一行是集合A，第一个整数m（0<=m<=100）代表集合A起始有m个元素，后面有m个非递减排序的整数，代表A中的元素。第二行是集合B，第一个整数n(0<=n<=100)代表集合B起始有n个元素，后面有n个非递减排序的整数，代表B中的元素。每行中整数之间用一个空格隔开。

输出

每组测试数据只要求输出一行，这一行含有 m+n 个来自集合 A 和集合B 中的元素。结果依旧是非递减的。每个整数间用一个空格隔开。

样例输入

4 3 5 8 11

7 2 6 8 9 11 15 20

样例输出

2 3 5 6 8 8 9 11 11 15 20

提示

总结：

本题书中提供的算法是基于顺序表的。在使用顺序表时需要两倍于数据元素数目。如果使用链表则只需要存储一倍的元素。然而使用链表同样需要存储一倍的指针。所以对于这类问题数据结构的选取，如果数据域占用的空间很大则可以使用链表存储来节省空间，而对于数据域占用不大的情况，则使用顺序表也可以。

-->

# 1325:算法2-3~2-6：Big Bang

## 题目描述

复习考研累了的时候看看一集二十分钟左右的《生活大爆炸》也不失为一种乐趣。在剧中Sheldon可以说是一个极品，真不知Leonard是如何忍受这位极品室友成天的唠叨。

你知道么？Sheldon有一个神秘的小本本，记录了所有他从小开始讨厌的人名。Stuart这位漫画店老板就是小本本的一员哦，谁叫他常常毫不客气地挤兑Sheldon，曾多次赌赢过Sheldon呢。

Penny是一个漂亮的女孩，好奇心也很强。为了满足她的好奇心，我当回编剧让她意外知道了Sheldon的那个小本本放在了哪里。于是她几乎每天都去看，看看上面有哪些人。但是那个小本本上的人名实在太多。要知道她可是没上过大学在饭店里面当服务员啊。请聪明的你帮帮她处理处理那个小本本吧。

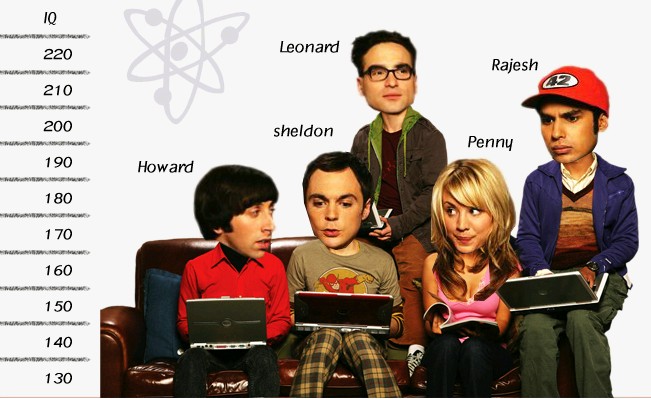


图1：《生活大爆炸》里的角色

Sheldon每天都会在小本本里记录些人名，当然有时也会与他们和好就会从小本本中将这个人名删除。我们假设Sheldon会在一个空的小本本上插入、删除、查询某个人。

要帮助Penny，你需要知道一个顺序表是怎么初始化、插入、删除以及查找的。下面我就将这些算法列举在下方。



图2：线性表的动态分配顺序存储结构以及初始化

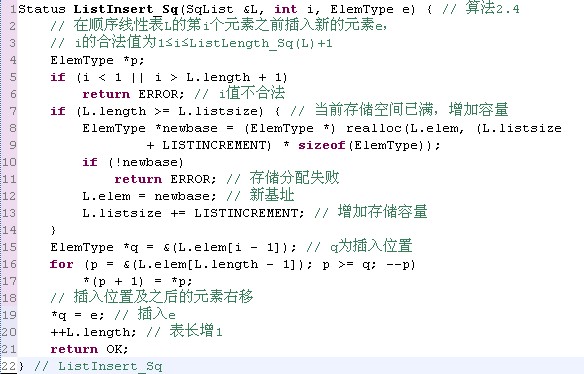


图3：线性表的插入算法

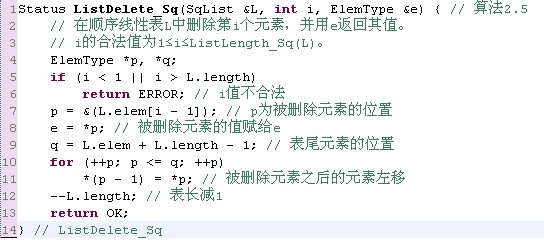


图4：线性表的删除算法



图5：线性表的查找算法

## 输入格式

输入数据只有一组，有很多行。每行的格式可能是下列一种：  
insert a name  
delete name  
show  
search name  
其中 a 是一个整数，代表在第a个名字前插入名字。name是一个姓名，只包含英文字母的大小写，每个名字不超过30个字符。  
输入保证不会插入列表中已经存在的姓名，不会删除列表中不存在的姓名，也不会搜索列表中不存在的姓名。

## 输出

起始时，列表是空的。只输出show和search name 的结果。show将列表中的姓名全部输出，search只输出找到该名字的序号（从1开始）。每次输出占一行，姓名间用空格隔开。如果列表中没有名字了，show时也要输出一个空行。

## 样例输入

insert 1 Stuart  
insert 2 Bernadette  
show  
search Stuart  
delete Stuart  
show  
insert 2 Stuart  
show  
insert 1 Amy  
insert 2 Leslie  
insert 3 Stephanie  
show  
delete Leslie  
show  
search Stuart

## 样例输出

Stuart Bernadette  
1  
Bernadette  
Bernadette Stuart  
Amy Leslie Stephanie Bernadette Stuart  
Amy Stephanie Bernadette Stuart  
4

## 提示

提示：

1、名字是不含空格的，指令也是一定的，所以可以用scanf("%s", str)来读取。

2、上述代码有些函数头中变量类型与变量之间有个&，这个表示该变量是引用类型的，是C++特性。在C语言中存在值传递与指针传递，值传递中形参不可以改变实参的值，需要通过指针来修改。而引用变量实际上就是实参的另一个名字，这种类型的形参改变会影响实参的值。

3、使用题目中的代码需要自己定义其中缺失的类型和变量，例如Status、OK以及Error等。

4、ElemType类型中可以只有一个字符数组用来存储姓名。

5、LocateElem\_Sq函数在调用时需要传递一个函数指针，可以这样定义：

Status cmp(ElemType e1, ElemType e2);

注意这个函数用来判断e1和e2是否相等，如果相等则返回非零值，否则返回0。因此可以在这个函数里直接返回 !strcmp(...)但最好需要改变返回类型。

6、内存分配以及字符串操作需要的头文件分别是stdlib.h和string.h需要被包含进来。

7、题目要求每个输出占一行，所以要注意换行。

总结：

1、实际上，题目中几乎将主要代码都写出来了。解决这道题使用上面的代码是可能复杂了点，但将各个功能独立出来是个不错的思路。以后修改就方便了，特别适用于代码量较大的程序。

2、C语言中参数的传递分为值传递和指针传递，而C++中多了一个引用传递。值传递和指针传递都不可以改变传递进来的值，但指针可以改变其所指向的值。在C语言中，调用函数时传入的参数叫做“实参”，而在函数声明或定义中在函数头中的参数叫做“形参”。值传递与指针传递中，形参的改变是不影响实参的。C++中，引用传递，形参与实参实际上是同一个内容的不同名字，因而形参的变化会改变实参。引用传递是C++中一个很重要也很方便的特性，比如在可能会产生不必要的复制时，采用引用传递是一个很不错的解决方案。

-->

# 1326:算法2-8~2-11：链表的基本操作

## 题目描述

链表是数据结构中一种最基本的数据结构，它是用链式存储结构实现的线性表。它较顺序表而言在插入和删除时不必移动其后的元素。现在给你一些整数，然后会频繁地插入和删除其中的某些元素，会在其中某些时候让你查找某个元素或者输出当前链表中所有的元素。

下面给你基本的算法描述：

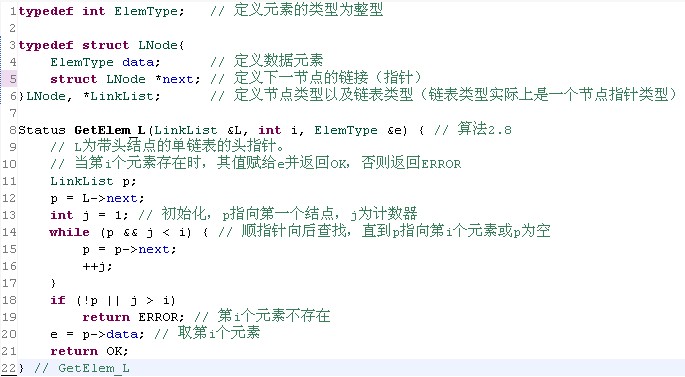


图1：链表类型的定义以及获得链表元素的算法描述

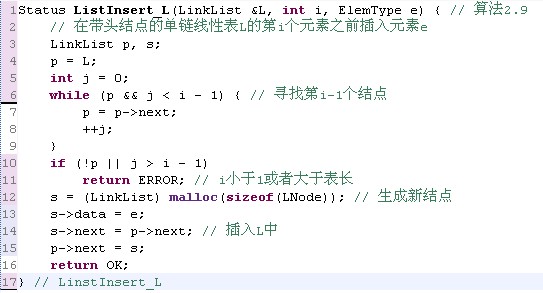


图2：链表的插入算法描述

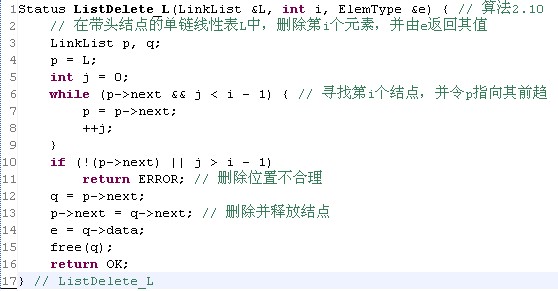


图3：链表的删除算法描述

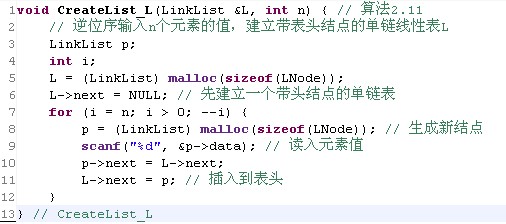


图4：链表的创建算法描述

## 输入格式

输入数据只有一组，第一行有n+1个整数，第一个整数是这行余下的整数数目n，后面是n个整数。这一行整数是用来初始化列表的，并且输入的顺序与列表中的顺序相反，也就是说如果列表中是1、2、3那么输入的顺序是3、2、1。

第二行有一个整数m，代表下面还有m行。每行有一个字符串，字符串是“get”，“insert”，“delete”，“show”中的一种。如果是“get”或者“delete”，则其后跟着一个整数a，代表获得或者删除第a个元素；如果是“insert”，则其后跟着两个整数a和e，代表在第a个位置前面插入e；“show”之后没有整数。

## 输出

如果获取成功，则输出该元素；如果删除成功则输出“delete OK”；如果获取失败或者删除失败，则输出“get fail”以及“delete fail”。如果插入成功则输出“insert OK”，否则输出“insert fail”。如果是“show”则输出列表中的所有元素，如果列表是空的，则输出“Link list is empty”。注：所有的双引号均不输出。

## 样例输入

3 3 2 1  
21  
show  
delete 1  
show  
delete 2  
show  
delete 1  
show  
delete 2  
insert 2 5  
show  
insert 1 5  
show  
insert 1 7  
show  
insert 2 5  
show  
insert 3 6  
show  
insert 1 8  
show  
get 2

## 样例输出

1 2 3  
delete OK  
2 3  
delete OK  
2  
delete OK  
Link list is empty  
delete fail  
insert fail  
Link list is empty  
insert OK  
5  
insert OK  
7 5  
insert OK  
7 5 5  
insert OK  
7 5 6 5  
insert OK  
8 7 5 6 5  
7

## 提示

提示：

1、因为输入数据中含有大量的插入和删除操作（不管你信不信，反正我信了），所以必须使用链表，否则很可能会超时。这也是考查链表的特性吧。

2、初始化链表的元素是倒序的，这个使用题目中创建列表的方法（从头部插入）就可以了。

总结：

这题考查的是链表的特性。顺序表中，怎样判断何时使用顺序表何时使用链表呢？就要看它们的特点了。顺序表的特点是随机存取、随机访问，也就是说如果存取和查询比较频繁的话使用顺序表比较合适；链表的特点是插入和删除时不必移动其后的节点，如果插入和删除操作比较频繁的话使用链表比较合适。

-->

# 1327:算法2-13~2-16：静态链表

## 题目描述

静态链表是使用顺序存储结构来实现的链表。严蔚敏《数据结构（C语言版）》在介绍静态链表时使用的是一个姓氏列表。

图1是书本上的静态链表示例，图（a）是初始化后插入了8个姓氏的链表，图（b）是在第5个元素前插入了“SHI”而删除了“WANG”的结果。

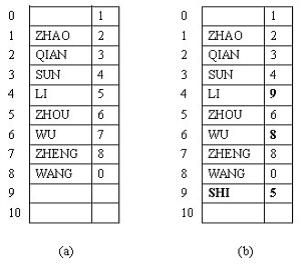


图1：静态链表示例

（a）修改前的状态；（b）修改后的状态

现在，我们就来实现一下这个静态链表。实际上静态链表与一般含有指针的链表没有太大的差别，只是静态链表的结点存放的空间不是在使用时临时分配的，而是在一开始就分配了固定的一些，一般是用数组。同时一般的链表使用指针来指向下一个结点而在静态链表中则使用数组下标了。大家如果看严蔚敏的书会发现书上的算法还是有些问题的。下面我就直接给大家展示一种静态链表的实现算法。

最重要的是模拟系统分配内存的过程。可以预先定义一个全局数组（space）作为后面分配的空间，然后再初始化这个数组，为以后分配做准备，如图2。初始化后，这个数组中的状态应该如图3(a)一样。这样，数组的第0个节点就是用来标识哪个结点可用来存储数据的。那么如果是含有头结点的静态链表，则一般开始时数组的第1个节点就是用来存放头结点的，此时数组第0个结点标识第2个结点可以用来存储数据，而第1个结点（静态链表的头结点）的下一个结点下标为0，标识着这个静态链表为空，如图3(b)。静态链表的初始化以及插入删除各种算法与一般的链表是相似的。具体描述如下：



图2：类型定义、用来模拟内存的数组定义以及初始化

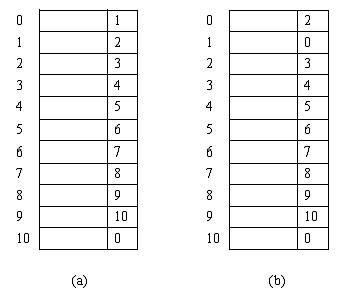


图3：模拟内存的数组状态。(a)初始化后的状态，(b)给静态链表分配头结点后的状态



图4：在静态链表中查找元素在space中的位置（相当于地址）

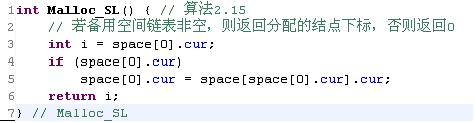


图5：给静态链表中的结点分配存储空间，实际上返回的是数组中可用的结点下标

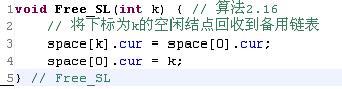


图6：释放静态链表中结点的存储空间

## 输入格式

静态链表的存储空间（图2中的space）始终只有11个节点，起始为空表。insert a e代表在第a个姓氏前插入姓氏e；delete a代表删除第a个姓氏；search e代表查找姓氏e的位置；show代表输出静态链表存储空间的状态。输入保证操作都合法。

## 输出

只遇到search和show时才输出。当遇到search时输出姓氏e在space中的位置；当遇到show时输出这11个结点的状态。姓氏占8个字符而数字占2个字符，姓氏左对齐。每个指令输出后面跟着含有20个星号的行。

## 样例输入

show  
insert 1 ZHAO  
show  
insert 2 QIAN  
show  
insert 3 SUN  
show  
insert 4 LI  
insert 5 ZHOU  
insert 6 WU  
insert 7 ZHENG  
insert 8 WANG  
show  
insert 1 ZHANG  
show  
search LI  
show

## 样例输出

         2  
         0  
         3  
         4  
         5  
         6  
         7  
         8  
         9  
        10  
         0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
         3  
         2  
ZHAO     0  
         4  
         5  
         6  
         7  
         8  
         9  
        10  
         0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
         4  
         2  
ZHAO     3  
QIAN     0  
         5  
         6  
         7  
         8  
         9  
        10  
         0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
         5  
         2  
ZHAO     3  
QIAN     4  
SUN      0  
         6  
         7  
         8  
         9  
        10  
         0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
        10  
         2  
ZHAO     3  
QIAN     4  
SUN      5  
LI       6  
ZHOU     7  
WU       8  
ZHENG    9  
WANG     0  
         0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
         0  
        10  
ZHAO     3  
QIAN     4  
SUN      5  
LI       6  
ZHOU     7  
WU       8  
ZHENG    9  
WANG     0  
ZHANG    2  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 5  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
         0  
        10  
ZHAO     3  
QIAN     4  
SUN      5  
LI       6  
ZHOU     7  
WU       8  
ZHENG    9  
WANG     0  
ZHANG    2  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 提示

提示：

1、怎样将字符串类型定义为ElemType呢？形如typedef int num一样，数组或者指针可以放在定义的类型名后面，例如将具有8个字符的姓氏定义为ElemType可以这样定义：typedef char ElemType[8]。

2、题目和书中给的算法描述还缺少静态链表的插入、删除以及显示，都需要自己写。

3、要求每个指令输出后跟一个空行，别忘了。

4、姓氏占8个字符，数字占2个字符，姓氏左对齐，可以这样输出printf("%-8s%2d");对于指令search也要输出占2个字符的数字。

5、静态链表初始化时将所有内存设为空，可以在InitSpace\_SL中使用下面的方法：

memset(space, 0 ,sizeof(space));

总结：

静态链表与一般链表极为相似：使用数组来模拟内存，使用数组下表来模拟内存中的地址。

-->

# 1328:算法2-18~2-19：双向循环链表

## 题目描述

双向链表是在结点中既保存了后一个结点指针又保存了前一个结点指针的链表。这种链表较单向链表而言能够快速查找某一结点的前后结点。下面给出双向链表的定义、插入以及删除算法描述。

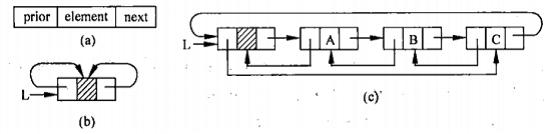


图1：双向链表示例

（a）结点结构；（b）空的双向循环链表；（c）含有三个结点的双向循环链表

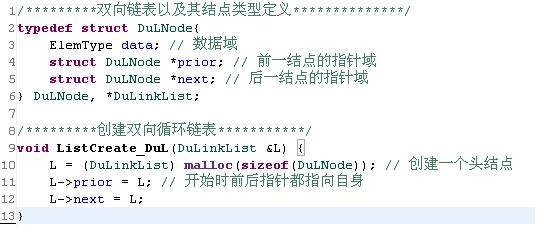


图2：双向链表的定义以及创建

双向链表在插入与删除时一定要注意其操作步骤的顺序。下面给出双向链表在插入与删除时的图示。

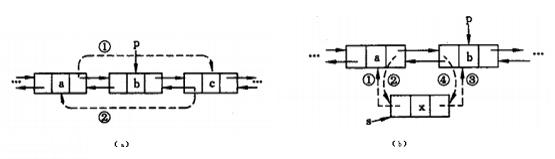


图3：双向链表插入与删除的图示

（a）双向链表的删除操作；（b）双向链表的插入操作

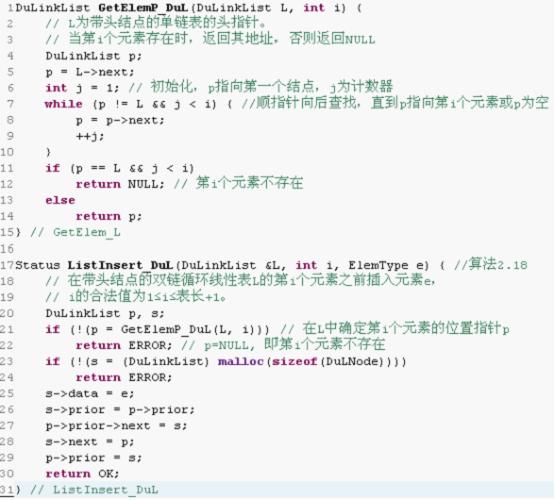


图4：双向链表的查找以及插入

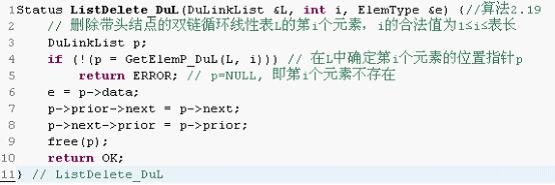


图5：双向链表的删除操作

## 输入格式

输入数据只有一组，包含很多行。每行有1~3个整数。第一个整数如果是0，则表示输出双向链表中的所有元素；第一个整数如果是1，表示插入1个整数，其后跟2个整数i、e代表在第i个位置插入e；第一个整数如果是2，表示删除1个整数，其后跟1个整数i，表示删除的位置为i。  
起始双向链表为空表。保证链表中每个元素不会重复，同时所有的操作都合法。

## 输出

当需要输出双向链表中的所有元素时输出，每次输出一行。整数间用一个空格隔开。

## 样例输入

1 1 2  
0  
1 2 7  
0  
2 1  
0  
1 2 4  
1 3 5  
1 2 6  
0  
2 3  
0

## 样例输出

2  
2 7  
7  
7 6 4 5  
7 6 5

## 提示

提示：

1、如果使用switch，注意每个case后面使用break。

2、结构体定义时，因为定义里面有用到这个类型的指针，所以需要在开始struct后面写上结构体名。

3、注意循环链表全部遍历结束的条件是遍历的指针是否又指向了头结点。

总结：

1、双向链表的重要之处在于插入和删除时的操作顺序，切勿弄乱顺序而丢失数据。

2、循环链表全部遍历的结束条件需要注意，不是非空，而是判断是否到头结点了。

本题考查的是双向循环链表，所以上述都需要注意。

-->

# 1329:算法2-23：一元多项式加法

## 题目描述

让我们重温小学初中时那些一元多项式的加法吧，不同的是现在使用计算机来帮我们计算了。多年前不想写作业的梦想终于快要实现了！下面就给出书上的算法。



图：链表实现一元多项式加法

## 输入格式

输入数据包含多组测试数据，每组数据包含两行一元多项式。每个多项式包含若干对整数，每对整数的第一个是系数，第二个是指数。每个多项式不超过100项，整数间用空格隔开，并且指数是递减的。

## 输出

每组测试数据输出一行结果，每个整数后面空一格。(包括行尾)

## 样例输入

3 2 4 1 7 0  
2 4 1 1  
2 3  
1 4  
3 2 4 1 7 0  
2 4 -4 1

## 样例输出

2 4 3 2 5 1 7 0   
1 4 2 3   
2 4 3 2 7 0

## 提示

提示：

1、由于多项式元素的重要信息在系数和指数，所以可以定义结点类型为含两个整数的结构体，一个代表系数而另一个代表指数。 2、定义完数据类型后，主要的就是怎么读取数据了。由于每个多项式占一行，所以可以用gets来读取一行，并判断是否为空行：while(gets(strA) && strlen(strA))...然后就将字符串中的数据转换为多项式类型。此时使用到一个字符串处理函数char \* strtok ( char \* str, const char \* delimiters )。这个函数的主要功能是将字符串str按delimiters中的字符分割。使用这个字符串处理函数时注意在处理某个字符串时首次使用时传递的参数是字符串指针而以后在使用时传递的参数是NULL。 3、下面的算法与有序序列有序合并算法类似。因为是多次循环，如果里面含有迭代变量（i，j之类的）注意下次循环时初值对不对。

总结：

多项式加法的算法与有序序列有序合并的算法类似，注意多项式元素类型的定义即可。

-->

# 1787:求最大值

## 题目描述

输入10个数，要求输出其中的最大值。

## 输入格式

测试数据有多组，每组10个数。

## 输出

对于每组输入,请输出其最大值（有回车）。

## 样例输入

10 22 23 152 65 79 85 96 32 1

## 样例输出

max=152

## 提示

-->

# 1330:算法3-1：八进制数

## 题目描述

将十进制数转换为八进制，并输出。



图：将十进制数转换为八进制并输出

## 输入格式

输入包含若干十进制正整数。

## 输出

输出相应的八进制数，每个占一行。

## 样例输入

1  
2  
3  
7  
8  
9  
19  
10020345

## 样例输出

1  
2  
3  
7  
10  
11  
23  
46162771

## 提示

提示：

书上有相应的算法，需要补充缺失的函数。

总结：

1、数值转换使用到堆栈，但是用函数调用（系统的堆栈）将会更为方便。

2、书中的算法实际上只能处理正整数，你有更好的方法还能够处理0和负整数么？

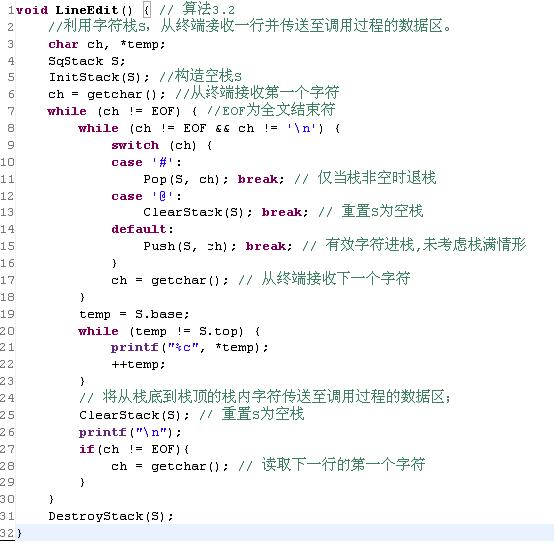
-->

# 1741:算法3-2：行编辑程序

## 题目描述

      一个简单的行编辑程序的功能是：接收用户从终端输入的程序或数据，并存入用户的数据区。由于用户在终端上进行输入时，不能保证不出差错，因此，若在编辑程序中，“每接收一个字符即存入用户数据区”的做法显然不是很恰当。较好的做法是，设立一个输入缓冲区，用以接收用户输入的一行字符，然后逐行存入用户数据区。允许用户输入出差错，并在发现有误时可以及时更正。例如，当用户发现刚刚键入的一个字符是错的时，可补进一个退格符“#”，以表示前一个字符无效；如果发现当前键入的行内错误较多或难以补救，则可以键入一个退行符“@”，以表示当前行中的字符均无效。例如假设从终端接收了这样的两行字符：  
whil##ilr#e(s#\*s)  
    outcha@    putchar(\*s=#++);  
则实际有效的是下列两行：  
while(\*s)  
    putchar(\*s++);

      为此，可设这个输入缓冲区为一个栈结构，每当从终端接收了一个字符之后先作如下判别：如果它不是退格符也不是退行符，则将该字符压入栈顶；如果是一个退格符，则从栈顶删去一个字符；如果它是一个退行符，则将字符栈清为空栈。上述处理过程可用下面算法描述之：



图：行编辑程序算法

## 输入格式

若干行程序或者数据，每行不超过200个字符。

## 输出

经过行编辑程序处理过后的输出。

## 样例输入

whil##ilr#e(s#\*s)  
outcha@ putchar(\*s=#++);

## 样例输出

while(\*s)  
putchar(\*s++);

## 提示

-->

# 1742:算法3-3：迷宫

## 题目描述

有一个 10 x 10 的迷宫，起点是‘S’，终点是‘E’，墙是‘#’，道路是空格。一个机器人从起点走到终点。当机器人走到一个通道块，前面已经没有路可走时，它会转向到当前面向的右手方向继续走。如果机器人能够过，则留下足迹‘\*’，如果走不通，则留下标记‘!’。

下面给出书中的算法，请你模拟机器人的走法输出最终的状态。



图：迷宫算法

## 输入格式

一个 10 x 10 的二维字符数组。

## 输出

机器人走过的路径状态。

## 样例输入

##########  
#S #   # #  
#  #   # #  
#    ##  #  
# ###    #  
#   #    #  
# #   #  #  
# ### ## #  
##      E#  
##########

## 样例输出

##########  
#\*\*#!!!# #  
# \*#!!!# #  
#\*\*!!##  #  
#\*###    #  
#\*\*\*#    #  
# #\*\*\*#  #  
# ###\*## #  
##   \*\*\*\*#  
##########

## 提示

提示：

位置，元素以及迷宫的类型定义：

typedef struct{

**int** r, c;      // 以行号和列号作为“坐标位置”类型

}PosType;

typedef struct{

**int** ord;       // 通道块在路径上的序号

    PosType seat; // 通道块在迷宫中的“坐标位置”

**int** di;            // 从此通道块走向下一通道块的“方向”

}SElemType;           // 定义堆栈元素的类型

typedef struct{

**char** arr[10][11];

}MazeType; // 定义迷宫类型（二维字符数组）

总结：

可以试着自己写个dfs（深度优先搜索）。

-->

# 1102:迷宫问题

## 题目描述

小明置身于一个迷宫，请你帮小明找出从起点到终点的最短路程。  
小明只能向上下左右四个方向移动。

## 输入格式

输入包含多组测试数据。输入的第一行是一个整数T，表示有T组测试数据。  
每组输入的第一行是两个整数N和M（1<=N,M<=100）。  
接下来N行，每行输入M个字符，每个字符表示迷宫中的一个小方格。  
字符的含义如下：  
‘S’：起点  
‘E’：终点  
‘-’：空地，可以通过  
‘#’：障碍，无法通过  
输入数据保证有且仅有一个起点和终点。

## 输出

对于每组输入，输出从起点到终点的最短路程，如果不存在从起点到终点的路，则输出-1。

## 样例输入

1  
5 5  
S-###  
-----  
##---  
E#---  
---##

## 样例输出

9

## 提示

-->

# 1743:算法3-4：表达式求值

## 题目描述

算数四则运算的规则是1）先乘除，后加减；2）从左算到右；3）先括号内，后括号外。

由此，算式4+2\*3-10/5的计算顺序为4+2\*3-10/5=4+6-10/5=4+6-2=8。

给定一个以“#”作为结束符的算式，求出算式的结果。

给出严蔚敏《数据结构（C语言）》中的一段算法描述以作参考：

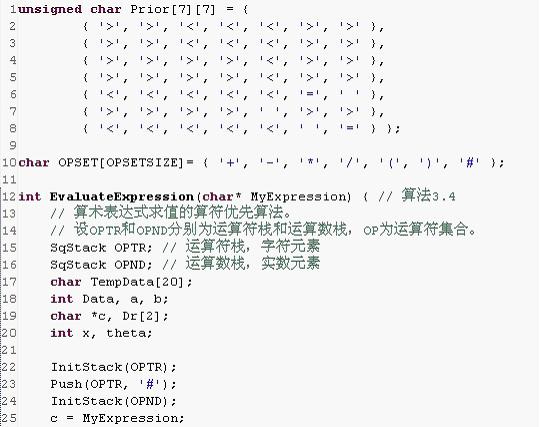


图1：表达式求值算法

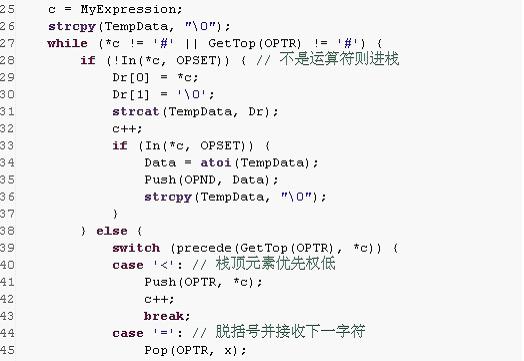


图2：表达式求值算法（续）



图3：表达式求值算法（续）

## 输入格式

以“#”结尾的表达式，运算数为正整数。每个表达式占一行。

## 输出

输出表达式运算的结果。

## 样例输入

4+2\*3-10/5#  
3\*(7-2)#  
2\*3/2#

## 样例输出

8  
15  
3

## 提示

提示：

使用栈来解决本题，很多人都会想到。但怎样建栈，却带来了问题。同样，严书上的代码实际上也给大家带来了问题。看过严书光盘中代码的人应该知道，代码中使用了两个栈，一个是存储运算符的，类型为char；另一个存储运算数，类型为float。而操作两个栈的函数都一样。要知道，除非像C++中使用泛型，C语言中却基本不能实现这样的操作。所以在C语言环境中需要将这两个栈结合在一起。由于char与int有种特别的联系，可以使用int来代替char存储运算符。

总结：

注意灵活运用栈，要是能够学习C++使用template就更好了。可以模拟STL了。

-->

# 1744:算法3-5：n阶Hanoi塔问题

## 题目描述

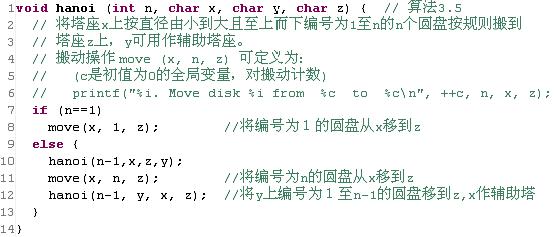
假设有三个分别命名为X、Y和Z的塔座，在塔座X上插有n个直径大小各不相同、依小到大编号为1,2,...,n的圆盘。现要求将X轴上的n个圆盘移至塔座Z上并仍按同样顺序叠排，圆盘移动时必须遵循下列规则：

1）每次只能移动一个圆盘；

2）圆盘可以插在X、Y和Z中的任一塔座上；

3）任何时刻都不能将一个较大的圆盘压在较小的圆盘之上。

如何实现移动圆盘的操作呢？当n=1时，问题比较简单，只要将编号为1的圆盘从塔座X直接移至塔座Z上即可；当n>1时，需利用塔座Y作辅助塔座，若能设法将压在编号为n的圆盘之上的n-1个圆盘从塔座X（依照上述法则）移至塔座Y上，则可先将编号为n的圆盘从塔座X移至塔座Z上，然后再将塔座Y上的n-1个圆盘（依照上述法则）移至塔座Z上。而如何将n-1个圆盘从一个塔座移至另一个塔座的问题是一个和原问题具有相同特征属性的问题，只是问题的规模小1，因此可以用同样的方法求解。由此可得如下图算法所示的求解n阶Hanoi塔问题的C函数。



图：求解n阶Hanoi塔问题的C函数

现在就请你将上述算法实现吧。

## 输入格式

输入数据有多组，每组1个整数n，表示Hanoi塔的阶数。

## 输出

将每次移动（move）按照以下格式输出：%2d. Move disk %d from %c to %c\n  
上述格式中第一个整数表示第几次移动，第二个整数表示移动第几个圆盘，后两个字符表示将圆盘从哪个塔座移至哪个塔座上。每组输出后面输出一个空行。

## 样例输入

1  
2  
3

## 样例输出

 1. Move disk 1 from X to Z  
  
 1. Move disk 1 from X to Y  
 2. Move disk 2 from X to Z  
 3. Move disk 1 from Y to Z  
  
 1. Move disk 1 from X to Z  
 2. Move disk 2 from X to Y  
 3. Move disk 1 from Z to Y  
 4. Move disk 3 from X to Z  
 5. Move disk 1 from Y to X  
 6. Move disk 2 from Y to Z  
 7. Move disk 1 from X to Z

## 提示

提示：

1、算法描述中输出使用的是%i，实际上就是使用%d。

2、输出时移动次数是两位的，我们保证输入量不会太大。

3、如果使用全局变量来记录移动的次数，注意每次在进行一个新的一组测试时，需要将这个全局计数变量设为0。

总结：

Hanoi塔（汉诺塔）是一个典型的利用递归解决的问题。如果你对迭代（循环）更有偏好，不妨用迭代来实现试试。

-->

# 1745:算法3-7：银行排队

## 题目描述

我们大多都有在银行排队的经历，唉，那坑爹的排队啊！现在就让我们来算算我们这些客户平均需要等多久吧。  
每天刚开始时银行会开m个窗口来为我们total个客户办理业务，当有客户需要办理业务时，先选择可以办理业务的窗口，如果有多个窗口可以办理业务就选择空闲时间最长的窗口，如果有多个窗口空闲的时间一样长，则选择序号小的窗口办理业务。假设我们每个人来到的时间和办理业务所需要的时间（为了简化问题，采用整数表示时间）都知道了。现在请你算算我们平均需要等待多久呢？

## 输入格式

有多组测试数据，每组数据开始有两个正整数m(<20)和total(<200)，后面有total对整数，对应客户先后到来的时间以及办理业务所需的时间。

## 输出

平均等待的时间，保留两位小数。

## 样例输入

2 6 1 3 4 1 5 3 9 2 13 4 13 3  
3 14 0 3 2 2 2 4 5 4 7 2 11 3 12 3 12 4 12 1 13 3 15 4 19 1 22 3 23 2  
2 5 0 6 0 5 0 6 7 1 7 2

## 样例输出

0.00  
0.29  
1.20

## 提示

提示：

题目中选择办理的窗口有三个状态，实际上从序号自小到大查找可以最早办理业务的窗口就已经满足上述三个状态了。可以使用数组来模拟列表。

总结：

实际上数组既可以模拟堆栈又可以模拟队列。

-->

# 1909:堆栈的使用

## 题目描述

堆栈是一种基本的数据结构。堆栈具有两种基本操作方式，push 和 pop。Push一个值会将其压入栈顶，而 pop 则会将栈顶的值弹出。现在我们就来验证一下堆栈的使用。

## 输入格式

对于每组测试数据，第一行是一个正整数 n，0<n<=10000(n=0 结束)。而后的 n 行，每行的第一个字符可能是'P’或者'O’或者'A’；如果是'P’，后面还会跟着一个整数，表示把这个数据压入堆栈；如果是'O’，表示将栈顶的值 pop 出来，如果堆栈中没有元素时，忽略本次操作；如果是'A’，表示询问当前栈顶的值，如果当时栈为空，则输出'E'。堆栈开始为空。

## 输出

 对于每组测试数据，根据其中的命令字符来处理堆栈；并对所有的'A’操作，输出当时栈顶的值，每个占据一行，如果当时栈为空，则输出'E’。当每组测试数据完成后，输出一个空行。

## 样例输入

5  
P 75  
O  
O  
P 60  
A  
7  
A  
O  
P 73  
P 49  
A  
O  
P 3  
0

## 样例输出

60  
  
E  
49

## 提示

-->

# 2113:字符统计

## 题目描述

输入一行字符串（字符串长度小于10000，由大小写字母、阿拉伯数字和空格组成），分别统计出其中大写英文字母，小写英文字母、阿拉伯数字和空格的个数。

## 输入格式

一行字符串（字符串长度小于10000，由大小写字母、阿拉伯数字和空格组成）

## 输出

分别输出大写英文字母，小写英文字母、阿拉伯数字和空格的个数，用空格隔开

## 样例输入

h0u84nfx 7

## 样例输出

0 5 4 1

## 提示

-->

# 2114:字符序列模式识别

## 题目描述

试写一个算法，识别字符序列是否为形如‘子序列1&子序列2’模式的字符序列，其中子序列2是子序列1的逆序列，0＜子序列字符串长度＜1000，且都为小写字母。输出YES或者NO。

## 输入格式

一行字符序列

## 输出

YES或NO

## 样例输入

hello&ollhe

## 样例输出

NO

## 提示

-->

# 1747:算法4-2：字符串连接

## 题目描述

将给定的字符串连接起来。书中的算法描述如下：



图：字符串连接算法

## 输入格式

三对字符串，每对字符串占一行，用空格隔开。每个字符串只包含数字和英文字母大小写且长度不超过100。

## 输出

将后一个字符串连接到前一个字符串后面，如果结果字符串长度超过100，输出一行“Result String is cutted.”否则将结果字符串输出来。

## 样例输入

hello acmclub  
123 456  
doyour best

## 样例输出

helloacmclub  
123456  
Doyourbest

## 提示

提示：

如果按照书上的算法定义字符串数据结构，其实也可以使用字符数组（长度限制在127），那么第一个字符就用来存储字符的长度。

总结：

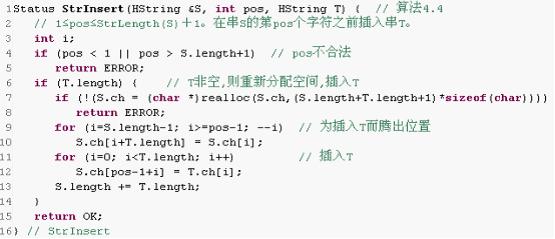
字符与整数在一定范围内可以相互转换。这个性质可以好好利用。

-->

# 1748:算法4-4：字符串插入

## 题目描述

将一个字符串插入到另一个字符串当中。算法描述如下：



图：字符串插入算法

## 输入格式

输入只有一行，两个字符串（stra,strb）和一个整数 i。字符串仅由英文大小写字母或数字组成。输入的字符串长度不超过127。

## 输出

将后一个字符串插入到前一个字符串的第i个字符前。输出插入后的结果。

## 样例输入

strng i 4

## 样例输出

string

## 提示

提示：

由于HString是由字符指针和整数构成，因而其结构和操作相对比较简单。但是，对于输入，可能需要多余的操作了。首先使用C字符串来获取字符串，然后获得输入字符串的长度，然后再分配内存。分配的长度要比输入的字符串的长度大1，你知道的。同时上述算法插入时也要注意字符串结尾（上述算法需要补充）。

注意输入的字符串长度不大于127，但结果字符的字符串长度可能大于127。

总结：

通过熟悉这个算法，能够理解C语言字符串和C++中的string类的实现了。

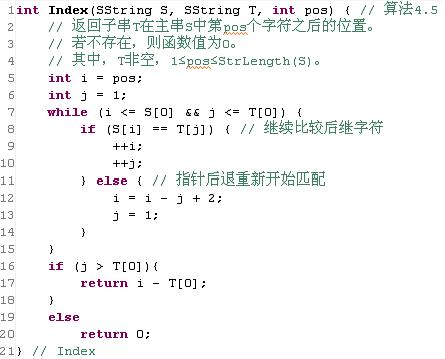
-->

# 1749:算法4-5：求子串位置的定位函数

## 题目描述

在算法4-1中已经描述过一种定位子串的算法，但其依赖字符串的其他操作（使用了算法4-3描述的子串提取以及字符串比较）。

下面给出书中另一种相对独立的定位子串算法：



图：求子串位置的定位函数

书中的算法思想是这样的：分别利用计数指针i和j指示主串S和模式串T中当前正待比较的字符位置。算法的基本思想是：从主串S的第pos个字符起和模式的第一个字符比较之，若相等，则继续逐个比较后续字符，否则从主串的下一个字符起再重新和模式的字符比较之。依次类推，直至模式T中的每个字符依次和主串S中的一个连续的字符序列相等，则称匹配成功，函数值为和模式T中第一个字符相等的字符在主串中的序号，否则称匹配不成功，函数值为零。  
       你的任务是将S中每次比较的字符输出来，并将匹配的序号输出。

## 输入格式

3组字符串，每组字符串占一行。每行包含由空格分隔的两个字符串，字符串仅由英文小写字母组成且长度不大于100。3组字符串，每组字符串占一行。每行包含由空格分隔的两个字符串，字符串仅由英文小写字母组成且长度不大于100。

## 输出

每组数据输出2行，第一行是S中每次比较的字符，第二行是后一个字符串在前一个字符串中的位置，如果不匹配，则输出0。

## 样例输入

string str  
thisisalongstring isa  
nosubstring subt

## 样例输出

str  
1  
thisisisa  
5  
nosubsubsttring  
0

## 提示

提示：

数据结构和算法4-1~4-3一致，均用字符数组来表示。具体解法只要在书中算法描述中将在每次比较时输出S中对应的字符即可，最后将函数结果存储下来并输出。

总结：

对字符串细致（内部）操作的的模拟，完全可以使用字符数组来实现。

-->

# 1750:算法4-7：KMP算法中的模式串移动数组

## 题目描述

字符串的子串定位称为模式匹配，模式匹配可以有多种方法。简单的算法可以使用两重嵌套循环，时间复杂度为母串与子串长度的乘积。而KMP算法相对来说在时间复杂度上要好得多，为母串与子串长度的和。但其算符比较难以理解。

在KMP算法中，使用到了一个next数组。这个数组就是在比较失配时母串指针不必回溯，而子串指针移动相应位置即可。我们给出书中next数组的算式表示以及算法，请你实现之。

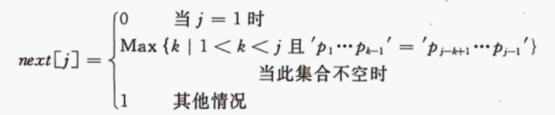


图1：next数组的算式表示

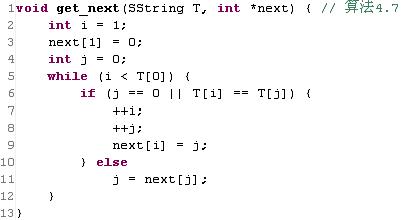


图2：next数组的算法表示

## 输入格式

一个模式串，仅由英文小写字母组成。长度不大于100。

## 输出

输出模式串对应的移动数组next。每个整数后跟一个空格。

## 样例输入

abaabcac

## 样例输出

0 1 1 2 2 3 1 2

## 提示

提示：

数据结构与以前相同，只用字符数组即可，只要调用上述函数即可。

总结：

KMP算法在字符串模式匹配算法中比较高效，具体KMP算法将在以后介绍。

**参考代码：**

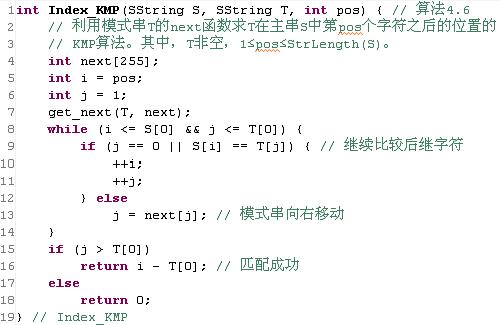
-->

# 1751:算法4-6：KMP字符串模式匹配算法实现

## 题目描述

KMP算法是字符串模式匹配算法中较为高效的算法之一，其在某次子串匹配母串失败时并未回溯母串的指针而是将子串的指针移动到相应的位置。严蔚敏老师的书中详细描述了KMP算法，同时前面的例子中也描述了子串移动位置的数组实现的算法。前面你已经实现了子串移动的数组，现在就来利用该数组来实现KMP模式匹配。

下面是相应的算法：



图：KMP算法

## 输入格式

3组字符串，每组字符串占一行。每行包含由空格分隔的两个字符串，字符串仅由英文小写字母组成且长度不大于100。

## 输出

每组数据输出1行，输出后一个字符串在前一个字符串中的位置，如果不匹配，则输出0。

## 样例输入

string str  
thisisalongstring isa  
nosubstring subt

## 样例输出

1  
5  
0

## 提示

**提示：**  
表示字符串的数据结构依然是字符数组。  
**总结：**  
KMP算法调用很简单，但难的是理解算法的思想。掌握算法的思想才能说是掌握算法。

-->

# 1785:字符串连接

## 题目描述

不借用任何字符串库函数实现无冗余地接受两个字符串，然后把它们无冗余的连接起来。

## 输入格式

每一行包括两个字符串，长度不超过100。

## 输出

可能有多组测试数据，对于每组数据，  
不借用任何字符串库函数实现无冗余地接受两个字符串，然后把它们无冗余的连接起来。  
输出连接后的字符串。

## 样例输入

abc def

## 样例输出

abcdef

## 提示

-->

# 1791:后缀子串排序

## 题目描述

对于一个字符串，将其后缀子串进行排序，例如grain  
其子串有：  
grain  
rain  
ain  
in  
n  
然后对各子串按字典顺序排序，即：  
ain,grain,in,n,rain

## 输入格式

每个案例为一行字符串。

## 输出

将子串排序输出

## 样例输入

grain  
banana

## 样例输出

ain  
grain  
in  
n  
rain  
a  
ana  
anana  
banana  
na  
nana

## 提示

-->

# 1800:统计字符

## 题目描述

统计一个给定字符串中指定的字符出现的次数。

## 输入格式

测试输入包含若干测试用例，每个测试用例包含2行，第1行为一个长度不超过5的字符串，第2行为一个长度不超过80的字符串。注意这里的字符串包含空格，即空格也可能是要求被统计的字符之一。当读到'#'时输入结束，相应的结果不要输出。

## 输出

对每个测试用例，统计第1行中字符串的每个字符在第2行字符串中出现的次数，按如下格式输出：  
c0 n0  
c1 n1  
c2 n2  
...   
其中ci是第1行中第i个字符，ni是ci出现的次数。

## 样例输入

I  
THIS IS A TEST  
i ng  
this is a long test string  
#

## 样例输出

I 2  
i 3  
  5  
n 2  
g 2

## 提示

-->

# 1804:字符串的反码

## 题目描述

一个二进制数，将其每一位取反，称之为这个数的反码。下面我们定义一个字符的反码。如果这是一个小写字符，则它和字符'a’的距离与它的反码和字符'z’ 的距离相同；如果是一个大写字符，则它和字符'A’的距离与它的反码和字符'Z’的距离相同；如果不是上面两种情况，它的反码就是它自身。

举几个例子，'a’的反码是'z’；'c’的反码是'x’；'W’的反码是'D’；'1’的反码还是'1’；'$'的反码还是'$'。  
一个字符串的反码定义为其所有字符的反码。我们的任务就是计算出给定字符串的反码。

## 输入格式

输入每行都是一个字符串，字符串长度不超过 80 个字符。如果输入只有!，表示输入结束，不需要处理。

## 输出

对于输入的每个字符串，输出其反码，每个数据占一行。

## 样例输入

Hello   
JLU-CCST-2011   
!

## 样例输出

Svool   
QOF-XXHG-2011

## 提示

-->

# 1805:首字母大写

## 题目描述

对一个字符串中的所有单词，如果单词的首字母不是大写字母，则把单词的首字母变成大写字母。  
在字符串中，单词之间通过空白符分隔，空白符包括：空格(' ')、制表符('\t')、回车符('\r')、换行符('\n')。

## 输入格式

输入一行：待处理的字符串（长度小于100）。

## 输出

可能有多组测试数据，对于每组数据，  
输出一行：转换后的字符串。

## 样例输入

if so, you already have a google account. you can sign in on the right.

## 样例输出

If So, You Already Have A Google Account. You Can Sign In On The Right.

## 提示

-->

# 1808:字符串的查找删除

## 题目描述

给定一个短字符串（不含空格），再给定若干字符串，在这些字符串中删除所含有的短字符串。

## 输入格式

输入只有1组数据。  
输入一个短字符串（不含空格），再输入若干字符串直到文件结束为止。

## 输出

删除输入的短字符串(不区分大小写)并去掉空格,输出。

## 样例输入

in  
#include   
int main()  
{  
  
printf(" Hi ");  
}

## 样例输出

#clude  
tma()  
{  
  
prtf("Hi");  
}

## 提示

注:将字符串中的In、IN、iN、in删除。

-->

# 1971:字符串链接

## 题目描述

不用strcat函数，自己编写一个字符串连接函数MyStrcat(char dstStr[],char srcStr[])，注意要单独编成函数，提交的时候提交全部的程序，包括main。

## 输入格式

输入包含多组数据，每组测试数据由两个待连接字符串组成，中间用一个空格分隔，字符串内部不含空格，且不为空串，每个字符串不多于20个字符。

## 输出

输出连接后的字符串。

## 样例输入

tian qin

## 样例输出

tianqin

## 提示

-->

# 1983:字符串处理

## 题目描述

读入两个字符串，字符串除了数字还可能包括 '—'、'E'、'e'、'．'，相加之后输出结果，如果是浮点型，要求用科学计数法表示（最多包含10个有效数字）。

## 输入格式

输入包含多组测试数据。

每组输入占两行，每行一个字符串，测试数据保证字符串的构成严格按照题目中的描述。

## 输出

输出两个数字相加的结果，每组输出占一行。

## 样例输入

34.56  
2.45e2

## 样例输出

2.7956e2

## 提示

-->

# 1995:反序字符串

题目描述

输入任意4 个字符，并按反序输出。

输入格式

输入第一行表示测试样例个数m，接下来m行每行一个字符串。

输出

输出m行，分别对应输入字符串的反序。

样例输入

2  
abcd  
eerd

样例输出

dcba  
dree

提示

-->

# 2025:比较字符串

## 题目描述

输入两个字符串，比较两字符串的长度大小关系。

## 输入格式

输入第一行表示测试用例的个数m，接下来m行每行两个字符串A和B，字符串长度不超过50。

## 输出

输出m行。若两字符串长度相等则输出A is equal long to B；若A比B长，则输出A is longer than B；否则输出A is shorter than B。

## 样例输入

2  
abc xy  
bbb ccc

## 样例输出

abc is longer than xy  
bbb is equal long to ccc

## 提示

-->

# 2057:字符串排序

## 题目描述

先输入你要输入的字符串的个数。然后换行输入该组字符串。每个字符串以回车结束，每个字符串少于一百个字符。  
如果在输入过程中输入的一个字符串为“stop”，也结束输入。  
然后将这输入的该组字符串按每个字符串的长度，由小到大排序，按排序结果输出字符串。

## 输入格式

字符串的个数，以及该组字符串。每个字符串以‘\n’结束。如果输入字符串为“stop”，也结束输入.

## 输出

可能有多组测试数据，对于每组数据，  
将输入的所有字符串按长度由小到大排序输出(如果有“stop”，不输出“stop”)。

## 样例输入

4  
faeruhyytrjh tjytj  
hsrthts   hjnshtgfhs  
stop  
3  
htrskbns  
bsartanjsf tyjndyt  
nsr jj jtey

## 样例输出

faeruhyytrjh tjytj  
hsrthts   hjnshtgfhs  
htrskbns  
nsr jj jtey  
bsartanjsf tyjndyt

## 提示

根据输入的字符串个数来动态分配存储空间（采用new()函数）。每个字符串会少于100个字符。  
测试数据有多组，注意使用while()循环输入。

-->

# 2110:矩阵对角求和

## 题目描述

求一个n×n矩阵对角线元素之和，其中1≤n<100，矩阵元素都小于10000。

## 输入格式

第一行是一个正整数n。

然后是n行，每行对应矩阵的每行，元素间用空格隔开。

## 输出

第一行为主对角线元素和，第二行为副对角线的元素和。

## 样例输入

3  
5724 1478 9358  
6962 4464 5705  
8145 3281 6827

## 样例输出

17015  
21967

## 提示

-->

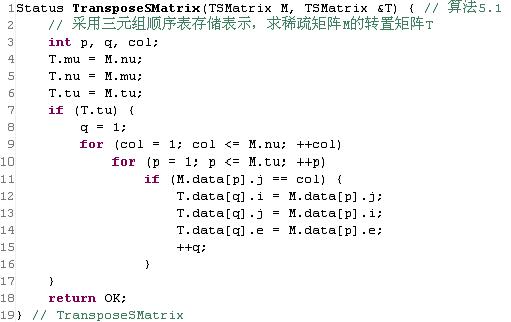
# 1752:算法5-1：稀疏矩阵转置

## 题目描述

稀疏矩阵的存储不宜用二维数组存储每个元素，那样的话会浪费很多的存储空间。所以可以使用一个一维数组存储其中的非零元素。这个一维数组的元素类型是一个三元组，由非零元素在该稀疏矩阵中的位置（行号和列号对）以及该元组的值构成。

矩阵转置就是将矩阵行和列上的元素对换。

现在就请你对一个稀疏矩阵进行转置。以下是稀疏矩阵转置的算法描述：



图：稀疏矩阵转置的算法描述

## 输入格式

输入的第一行是两个整数r和c（r\*c <= 12500），分别表示一个包含很多0的稀疏矩阵的行数和列数。接下来有r行，每行有c个整数，表示这个稀疏矩阵的各个元素。

## 输出

输出c行，每行有r个整数，每个整数后跟一个空格。该结果为输入稀疏矩阵的转置矩阵。

## 样例输入

6 7  
0 12 9 0 0 0 0  
0 0 0 0 0 0 0  
-3 0 0 0 0 14 0  
0 0 24 0 0 0 0  
0 18 0 0 0 0 0  
15 0 0 -7 0 0 0

## 样例输出

0 0 -3 0 0 15   
12 0 0 0 18 0   
9 0 0 24 0 0   
0 0 0 0 0 -7   
0 0 0 0 0 0   
0 0 14 0 0 0   
0 0 0 0 0 0

## 提示

提示：

我使用的严老师纸质书中用union类型来表示稀疏矩阵类型，这是有问题的，应该使用struct来表示该类型。

注意理解为什么转置算法中，以列从小到大来进行转置。实际上只需一个循环就能够完成转置而不需将列从小到大来处理，转置后的矩阵虽然是正确的但却乱掉了，以至于在各种处理中会增加复杂。（其实就本题而言，如果不以列从小到大处理将导致输出困难，输出的复杂度增加）

总结：

矩阵是一个应用很广泛的工具和课题。看看《黑客帝国》就知道了。现在初步给大家介绍矩阵的操作，以后有机会还会详细讨论矩阵的。

-->

# 1970:计算两个矩阵的乘积

## 题目描述

计算两个矩阵的乘积，第一个是2\*3矩阵，第二个是3\*2矩阵，结果为一个2\*2矩阵。

## 输入格式

输入包含多组数据，先输入一个2\*3矩阵，再输入一个3\*2矩阵。

## 输出

输出两个矩阵的乘积。

## 样例输入

1 1 1  
1 1 1  
1 1  
1 1  
1 1

## 样例输出

3 3  
3 3

## 提示

-->

# 2009:矩阵最大值

## 题目描述

编写一个程序输入一个m\*n的矩阵存储并输出，并且求出每行的最大值和每行的总和。  
要求把每行总和放入每行最大值的位置，如果有多个最大值，取下标值最小的那一个作为最大值。  
最后将结果矩阵输出。

## 输入格式

输入的第一行包括两个整数m和n(1<=m,n<=100)，分别代表矩阵的行和列的维数。  
接下来的m行每行有n个数，代表矩阵的元素。

## 输出

可能有多组测试数据，对于每组数据，输出按题目要求执行后的矩阵。

## 样例输入

2 4  
35 1 70 25   
79 59 63 65   
9 3  
82 28 62   
92 96 43   
28 37 92   
5 3 54   
93 83 22   
17 19 96   
48 27 72   
39 70 13   
68 100 36

## 样例输出

35 1 131 25  
266 59 63 65  
172 28 62  
92 231 43  
28 37 157  
5 3 62  
198 83 22  
17 19 132  
48 27 147  
39 122 13  
68 204 36

## 提示

-->

# 2010:矩阵转置

## 题目描述

输入一个N\*N的矩阵，将其转置后输出。要求：不得使用任何数组(就地逆置)。

## 输入格式

输入的第一行包括一个整数N,(1<=N<=100)，代表矩阵的维数。  
接下来的N行每行有N个整数，分别代表矩阵的元素。

## 输出

可能有多组测试数据，对于每组数据，将输入的矩阵转置后输出。

## 样例输入

1  
68   
3  
1 70 25   
79 59 63   
65 6 46

## 样例输出

68  
1 79 65  
70 59 6  
25 63 46

## 提示

-->

# 2029:对称矩阵

## 题目描述

输入一个N维矩阵，判断是否对称。

## 输入格式

输入第一行包括一个数：N(1<=N<=100)，表示矩阵的维数。  
接下来的N行，每行包括N个数，表示N\*N矩阵的元素。

## 输出

可能有多组测试数据，对于每组数据，  
输出"Yes!”表示矩阵为对称矩阵。  
输出"No!”表示矩阵不是对称矩阵。

## 样例输入

1  
68   
3  
1 70 25   
70 79 59   
25 59 63   
3  
6 46 82   
28 62 92   
96 43 28

## 样例输出

Yes!  
Yes!  
No!

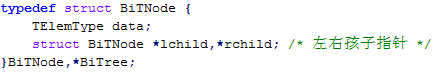
## 提示

-->

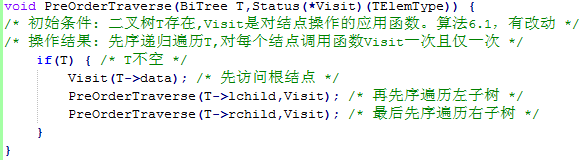
# 1757:算法6-1~6-4：二叉链表存储的二叉树

## 题目描述

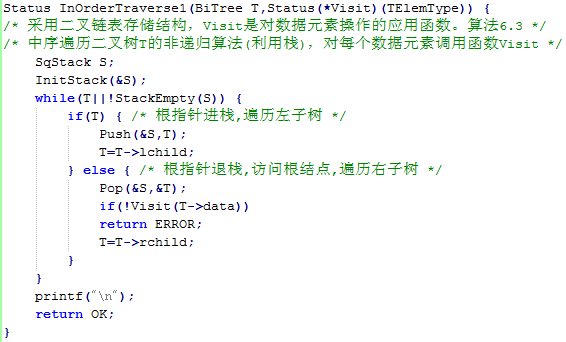
树形结构是一类重要的非线性数据结构，其中以树和二叉树最为常用。对于每一个结点至多只有两课子树的一类树，称其为二叉树。二叉树的链式存储结构是一类重要的数据结构，其形式定义如下：



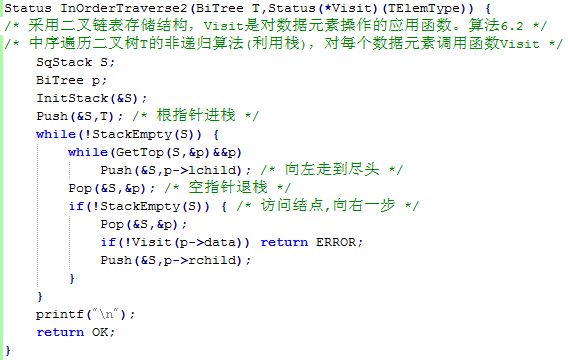
而二叉树的前序、中序遍历是非常重要的能够访问二叉树所有结点的算法，下面分别列出一种先序遍历和两种中序遍历的算法。



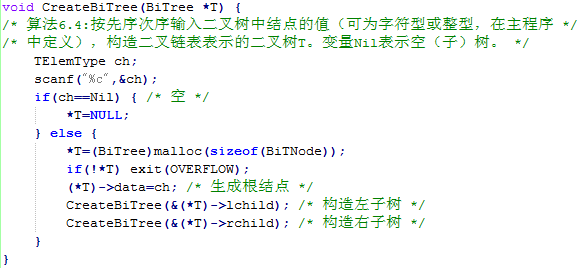
第一种中序遍历的方法（算法6.3）：



第二种中序遍历的方法（算法6.2）：



通过读入一个字符串，建立二叉树的算法如下：



在本题中，将会给出一个按照先序遍历得出的字符串，空格代表空的子节点，大写字母代表节点内容。请通过这个字符串建立二叉树，并按照题目描述中的一种先序遍历和两种中序遍历的算法分别输出每一个非空节点。

## 输入格式

输入只有一行，包含一个字符串S，用来建立二叉树。保证S为合法的二叉树先序遍历字符串，节点内容只有大写字母，且S的长度不超过100。

## 输出

共有三行，每一行包含一串字符，表示分别按先序、中序、中序得出的节点内容，每个字母后输出一个空格。请注意行尾输出换行。

## 样例输入

ABC  DE G  F

## 样例输出

A B C D E G F   
C B E G D F A   
C B E G D F A

## 提示

遍历是二叉树各种操作的基础，可以在遍历的过程中对节点进行各种操作。通过二叉树的遍历，可以建立二叉树。而先序、中序和后序遍历分别具有各自的特点，是探索二叉树性质的绝佳“武器”。

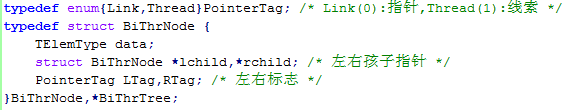
-->

# 1758:算法6-5~6-7：线索二叉树

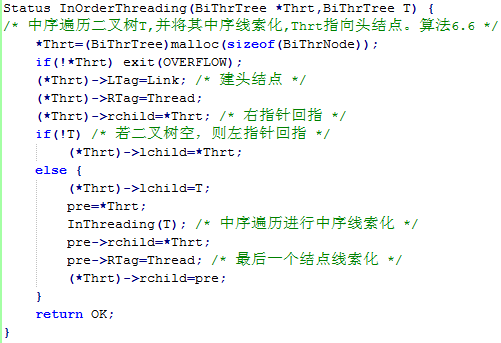
## 题目描述

在遍历二叉树的过程中，是按照一定的规则将二叉树中的结点排列成一个线性序列，从而得到二叉树中结点的先序序列或中序序列或后序序列。但是，当以二叉链表作为存储结构时，只能找到结点的左右孩子信息，而不能直接得到结点在任意一个序列中的前驱和后继的信息，而这种信息只有在遍历的动态过程中才能够得到。

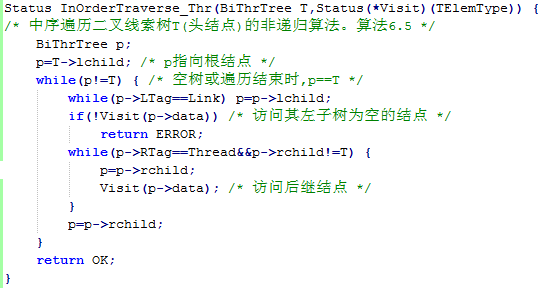
为了保存这种信息，就需要使用线索链表。其中指向结点的前驱和后继的指针，叫做线索。添加上线索的二叉树称之为线索二叉树。其结点定义如下：



下面给出按照中序遍历将二叉树中序线索化的算法：



在已经线索化的二叉线索树中，进行中序遍历的算法如下所示：



本题中，将会给出一个按照先序遍历得出的字符串，空格代表空的子节点，大写字母代表节点内容。请通过这个字符串建立二叉树，并按照题目描述中算法，中序遍历二叉树并中序线索化二叉树，之后中序遍历输出二叉线索树。

## 输入格式

输入只有一行，包含一个字符串S，用来建立二叉树。保证S为合法的二叉树先序遍历字符串，节点内容只有大写字母，且S的长度不超过100。

## 输出

共一行，包含一串字符，表示按中序遍历二叉线索树得出的节点内容，每个字母后输出一个空格。请注意行尾输出换行。

## 样例输入

ABC  DE G  F

## 样例输出

C B E G D F A

## 提示

通过线索化二叉树建立二叉线索树，将给普通二叉树添加更快捷的遍历方式。在线索树上进行遍历，只需要先找到序列中的第一个结点，然后依次找结点后继直至其后继为空时而止。而对于中序线索二叉树，由于其固有的性质，在遍历的过程中虽然时间复杂度依旧为O(n)，但常数因子将会比普通的算法减小且不需要栈结构辅助，是一种非常优秀的遍历算法。

-->

# 1759:算法6-8~6-11：用树表示的等价问题

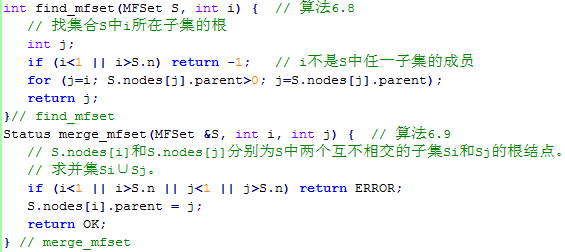
## 题目描述

在离散数学中，对等价关系和等价类的定义是：

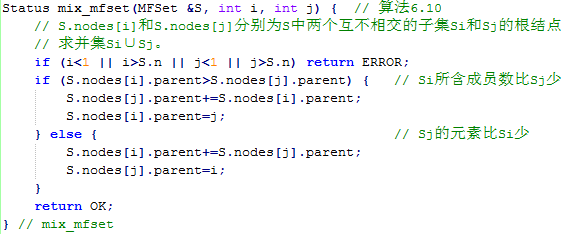
如果集合S中的关系R是自反的、对称的和传递的，则称它为一个等价关系。

等价关系是现实世界中广泛存在的一种关系，许多应用问题可以归结至等价类问题，这类问题通常被称为等价问题。

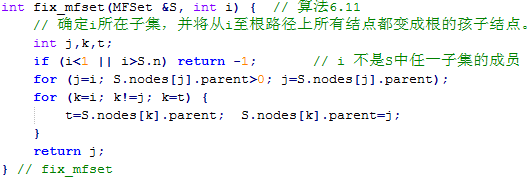
通过使用集合，能够解决等价问题。而集合可以通过双亲表示法的树结构进行保存。通过对树结构的操作，可以实现查找、归并等操作。查找操作和归并操作的算法如下：



在以上的归并操作中，由于表示集合的树的深度与树形成的过程有关，因此在最坏情况下全部归并操作将会有O(n2)的复杂度。而通过在归并时比较子集所含成员的数目，令成员少的归并至成员多的集合，将能够提高算法的效率。下面给出优化的归并操作算法：



另外，通过增加“压缩路径”的功能，即将所有从根到相应元素路径上的元素都变成树根的孩子。算法如下所示：



本题中，将会给出n个原本互不相交的集合及k次集合合并的操作。通过这k次合并，判断最终的某两个原始的集合是否被合并成了同一个集合。

## 输入格式

输入的第一行包含两个用空格隔开的正整数n和k，其中n不超过100，k不超过n-1。

之后的k行中，每行包含两个用空格隔开的正整数x和y，表示将x元素所在的集合和y元素所在的集合合并至同一个集合。保证x和y均在1至n之间。

最后一行中，包含两个正整数，表示需要判断是否在同一个集合的元素编号。

## 输出

共一行，包含字符串“YES”或“NO”，“YES”表示需判断的元素在同一个集合中，“NO”表示不在同一个集合中。请注意不需要输出引号，且行尾输出换行。

## 样例输入

5 2  
1 3  
2 3  
1 2

## 样例输出

YES

## 提示

以集合为基础结构的抽象数据类型可以有多种实现方法，比如用位向量表示集合或者用有序表表示集合等等。而如何高效的实现以集合为基础的抽象数据类型，取决于该集合的大小以及对此集合所进行的操作。

在本题中实现的MFSet抽象数据结构，又被称为并查集，是一种能够非常高效的实现集合的合并、查询等操作的数据结构。

-->

# 1760:算法6-12：自底向上的赫夫曼编码

## 题目描述

在通讯领域，经常需要将需要传送的文字转换成由二进制字符组成的字符串。在实际应用中，由于总是希望被传送的内容总长尽可能的短，如果对每个字符设计长度不等的编码，且让内容中出现次数较多的字符采用尽可能短的编码，则整个内容的总长便可以减少。另外，需要保证任何一个字符的编码都不是另一个字符的编码前缀，这种编码成为前缀编码。

而赫夫曼编码就是一种二进制前缀编码，其从叶子到根（自底向上）逆向求出每个字符的算法可以表示如下：



在本题中，读入n个字符所对应的权值，生成赫夫曼编码，并依次输出计算出的每一个赫夫曼编码。

## 输入格式

输入的第一行包含一个正整数n，表示共有n个字符需要编码。其中n不超过100。

第二行中有n个用空格隔开的正整数，分别表示n个字符的权值。

输出

共n行，每行一个字符串，表示对应字符的赫夫曼编码。

## 样例输入

8  
5 29 7 8 14 23 3 11

## 样例输出

0110  
10  
1110  
1111  
110  
00  
0111  
010

提示

赫夫曼树又名最优二叉树，它是一类带权路径长度最小的二叉树。通过构造赫夫曼树，我们可以得到赫夫曼编码，从而使得通信能够得到更高的效率。在本题中，构造赫夫曼树的过程使用了从叶子到根的逆向顺序，另外，还有一种从根出发直到叶子的赫夫曼编码构造算法，这将在下一题中进行讨论。

-->

# 1814:剩下的树

## 题目描述

有一个长度为整数L(1<=L<=10000)的马路，可以想象成数轴上长度为L的一个线段，起点是坐标原点，在每个整数坐标点有一棵树，即在0,1,2，...，L共L+1个位置上有L+1棵树。  
    现在要移走一些树，移走的树的区间用一对数字表示，如 100 200表示移走从100到200之间（包括端点）所有的树。  
    可能有M(1<=M<=100)个区间，区间之间可能有重叠。现在要求移走所有区间的树之后剩下的树的个数。

## 输入格式

两个整数L(1<=L<=10000)和M(1<=M<=100)。  
    接下来有M组整数，每组有一对数字。

## 输出

 可能有多组输入数据，对于每组输入数据，输出一个数，表示移走所有区间的树之后剩下的树的个数。

## 样例输入

4 2  
1 2  
0 2  
11 2  
1 5  
4 7  
0 0

## 样例输出

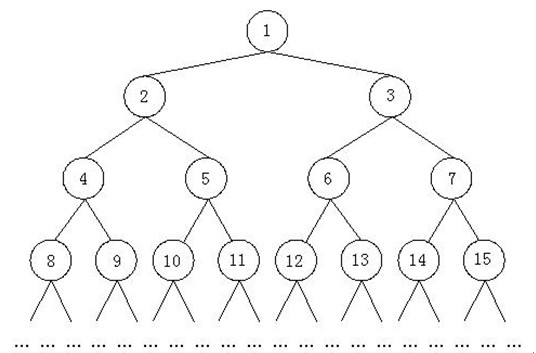
2  
5

提示

-->

# 1905:二叉树

题目描述



    如上所示，由正整数1，2，3……组成了一颗特殊二叉树。我们已知这个二叉树的最后一个结点是n。现在的问题是，结点m所在的子树中一共包括多少个结点。  
  
    比如，n = 12，m = 3那么上图中的结点13，14，15以及后面的结点都是不存在的，结点m所在子树中包括的结点有3，6，7，12，因此结点m的所在子树中共有4个结点。

## 输入格式

输入数据包括多行，每行给出一组测试数据，包括两个整数m，n (1 <= m <= n <= 1000000000)。最后一组测试数据中包括两个0，表示输入的结束，这组数据不用处理。

## 输出

 对于每一组测试数据，输出一行，该行包含一个整数，给出结点m所在子树中包括的结点的数目。

## 样例输入

3 7  
142 6574  
2 754  
0 0

## 样例输出

3  
63  
498

提示

-->

# 1910:二叉树遍历

## 题目描述

二叉树的前序、中序、后序遍历的定义：  
前序遍历：对任一子树，先访问跟，然后遍历其左子树，最后遍历其右子树；  
中序遍历：对任一子树，先遍历其左子树，然后访问根，最后遍历其右子树；  
后序遍历：对任一子树，先遍历其左子树，然后遍历其右子树，最后访问根。  
给定一棵二叉树的前序遍历和中序遍历，求其后序遍历（提示：给定前序遍历与中序遍历能够唯一确定后序遍历）。

## 输入格式

两个字符串，其长度n均小于等于26。  
第一行为前序遍历，第二行为中序遍历。  
二叉树中的结点名称以大写字母表示：A，B，C....最多26个结点。

## 输出

输入样例可能有多组，对于每组测试样例，  
输出一行，为后序遍历的字符串。

## 样例输入

ABC  
CBA  
ABCDEFG  
DCBAEFG

## 样例输出

CBA  
DCBGFEA

提示

-->

# 1914:树查找

## 题目描述

有一棵树，输出某一深度的所有节点，有则输出这些节点，无则输出EMPTY。该树是完全二叉树。

## 输入格式

输入有多组数据。  
每组输入一个n(1<=n<=1000)，然后将树中的这n个节点依次输入，再输入一个d代表深度。

## 输出

输出该树中第d层得所有节点，节点间用空格隔开，最后一个节点后没有空格。

## 样例输入

5  
1 2 3 4 5   
7  
7  
1 2 3 4 5 6 7   
2  
0

## 样例输出

EMPTY  
2 3

提示

-->

# 1772:算法9-5~9-8：二叉排序树的基本操作

## 题目描述

二叉排序树或者是一棵空树，或者是具有以下几条性质的二叉树：

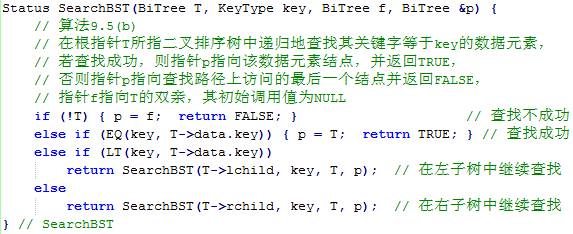
1.       若它的左子树不空，则左子树上所有结点的值均小于它的根节点的值；

2.       若它的右子树不空，则右子树上所有结点的值均大于它的根节点的值；

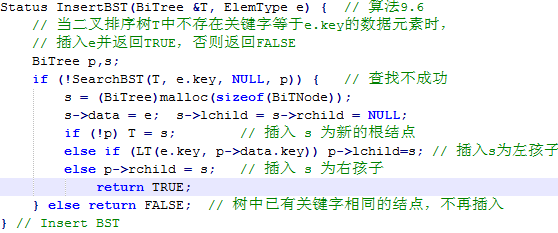
3.       它的左右子树也分别为二叉排序树。

二叉排序树又可以被称为二叉查找树，根据上述定义的结构不难知道，它的查找过程十分简单，只需要通过不断的将当前结点的值与需要查找的值进行比较，如果相等则直接输出，如果要查找的值更小则深入至左子树进行比较，否则就深入右子树进行比较，直到找到相应的值或者进入了一棵不存在的子树为止。

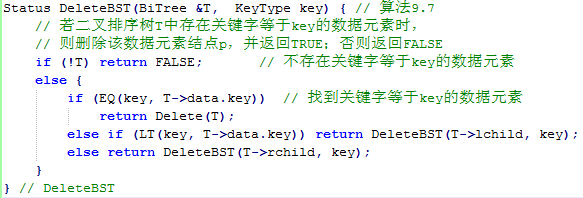
其查找过程可以描述如下：

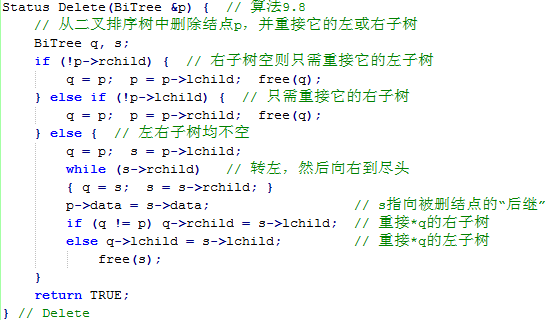


而其插入过程同样也十分简洁，可以描述如下：



而删除操作可以描述为如下的两个算法：





在本题中，读入一串整数，首先利用这些整数构造一棵二叉排序树。另外给定多次查询，利用构造出的二叉排序树，判断每一次查询是否成功。

## 输入格式

输入的第一行包含2个正整数n和k，分别表示共有n个整数和k次查询。其中n不超过500，k同样不超过500。

第二行包含n个用空格隔开的正整数，表示n个整数。

第三行包含k个用空格隔开的正整数，表示k次查询的目标。

## 输出

只有1行，包含k个整数，分别表示每一次的查询结果。如果在查询中找到了对应的整数，则输出1，否则输出0。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

8 3  
1 3 5 7 8 9 10 15  
9 2 5

## 样例输出

1 0 1

## 提示

在本题中，首先需要按照题目描述中的算法完成二叉排序树的构造过程，之后需要通过在二叉排序树中的不断向下查找，将需要查询的值与当前节点的值进行比较，直到确定被查询的值是否存在。

通过课本中的性能分析部分，不难发现二叉排序树的平均查找长度是和logn同数量级的，但是，在某些特殊情况下二叉排序树将会退化，使查找的效率大大降低，这时就需要引入二叉排序树的平衡操作，利用平衡二叉树来保证查找的效率始终维持在logn的数量级上。

-->

# 1919:二叉排序树

## 题目描述

输入一系列整数，建立二叉排序数，并进行前序，中序，后序遍历。

## 输入格式

输入第一行包括一个整数n(1<=n<=100)。接下来的一行包括n个整数。

## 输出

可能有多组测试数据，对于每组数据，将题目所给数据建立一个二叉排序树，并对二叉排序树进行前序、中序和后序遍历。每种遍历结果输出一行。每行最后一个数据之后有一个空格。

## 样例输入

1  
2   
2  
8 15   
4  
21 10 5 39

## 样例输出

2   
2   
2   
8 15   
8 15   
15 8   
21 10 5 39   
5 10 21 39   
5 10 39 21

## 提示

-->

# 1920:二叉搜索树

## 题目描述

判断两序列是否为同一二叉搜索树序列

## 输入格式

开始一个数n，(1<=n<=20) 表示有n个需要判断，n= 0 的时候输入结束。  
接下去一行是一个序列，序列长度小于10，包含(0~9)的数字，没有重复数字，根据这个序列可以构造出一颗二叉搜索树。  
接下去的n行有n个序列，每个序列格式跟第一个序列一样，请判断这两个序列是否能组成同一颗二叉搜索树。

## 输出

如果序列相同则输出YES，否则输出NO

## 样例输入

6  
45021  
12045  
54120  
45021  
45012  
21054  
50412  
0

## 样例输出

NO  
NO  
YES  
NO  
NO  
NO

## 提示

-->

# 2014:二叉树遍历

## 题目描述

编一个程序，读入用户输入的一串先序遍历字符串，根据此字符串建立一个二叉树（以指针方式存储）。  
例如如下的先序遍历字符串：  
ABC##DE#G##F###  
其中“#”表示的是空格，空格字符代表空树。建立起此二叉树以后，再对二叉树进行中序遍历，输出遍历结果。

## 输入格式

输入包括1行字符串，长度不超过100。

## 输出

可能有多组测试数据，对于每组数据，  
输出将输入字符串建立二叉树后中序遍历的序列，每个字符后面都有一个空格。  
每个输出结果占一行。

## 样例输入

a#b#cdef#####  
a##

## 样例输出

a b f e d c   
a

## 提示

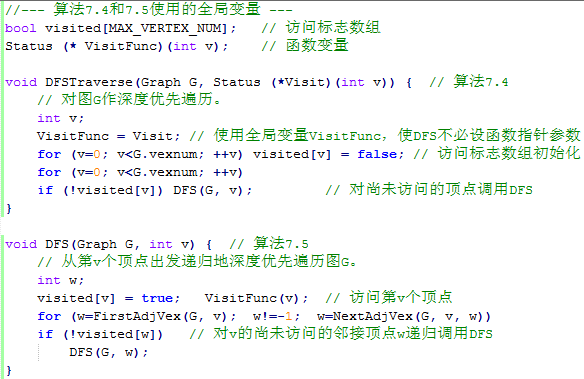
-->

# 1762:算法7-4,7-5：图的遍历——深度优先搜索

## 题目描述

深度优先搜索遍历类似于树的先根遍历，是树的先根遍历的推广。其过程为：假设初始状态是图中所有顶点未曾被访问，则深度优先搜索可以从图中的某个顶点v出发，访问此顶点，然后依次从v的未被访问的邻接点出发深度优先遍历图，直至图中所有和v有路径相通的顶点都被访问到；若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作为起始点，重复上述过程，直至图中所有顶点都被访问到为止。

其算法可以描述如下：



在本题中，读入一个无向图的邻接矩阵（即数组表示），建立无向图并按照以上描述中的算法遍历所有顶点，输出遍历顶点的顺序。

## 输入格式

输入的第一行包含一个正整数n，表示图中共有n个顶点。其中n不超过50。

以后的n行中每行有n个用空格隔开的整数0或1，对于第i行的第j个0或1，1表示第i个顶点和第j个顶点有直接连接，0表示没有直接连接。当i和j相等的时候，保证对应的整数为0。

输入保证邻接矩阵为对称矩阵，即输入的图一定是无向图。

## 输出

只有一行，包含n个整数，表示按照题目描述中的深度优先遍历算法遍历整个图的访问顶点顺序。每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

4  
0 1 0 1  
1 0 0 0  
0 0 0 1  
1 0 1 0

## 样例输出

0 1 3 2

## 提示

在本题中，需要熟练掌握图的邻接矩阵存储方式。在建立完成无向图之后，需要严格按照题目描述的遍历顺序对图进行遍历。另外，算法中描述的FirstAdjVex函数和NextAdjVex函数，需要认真的自行探索并完成。

通过这道题目，应该能够对图的深度优先搜索建立更加直观和清晰的概念。

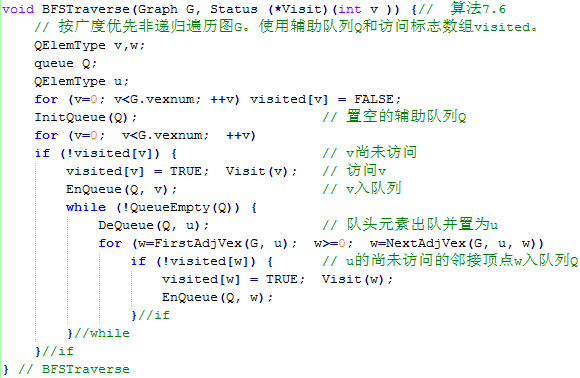
-->

# 1763:算法7-6：图的遍历——广度优先搜索

## 题目描述

广度优先搜索遍历类似于树的按层次遍历的过程。其过程为：假设从图中的某顶点v出发，在访问了v之后依次访问v的各个未曾被访问过的邻接点，然后分别从这些邻接点出发依次访问它们的邻接点，并使“先被访问的顶点的邻接点”先于“后被访问的顶点的邻接点”被访问，直至图中所有已被访问的顶点的邻接点都被访问到。若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作为起始点。重复上述过程，直至图中所有顶点都被访问到为止。

其算法可以描述如下：



在本题中，读入一个无向图的邻接矩阵（即数组表示），建立无向图并按照以上描述中的算法遍历所有顶点，输出遍历顶点的顺序。

## 输入格式

输入的第一行包含一个正整数n，表示图中共有n个顶点。其中n不超过50。

以后的n行中每行有n个用空格隔开的整数0或1，对于第i行的第j个0或1，1表示第i个顶点和第j个顶点有直接连接，0表示没有直接连接。当i和j相等的时候，保证对应的整数为0。

输入保证邻接矩阵为对称矩阵，即输入的图一定是无向图。

## 输出

只有一行，包含n个整数，表示按照题目描述中的广度优先遍历算法遍历整个图的访问顶点顺序。每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

4  
0 0 0 1  
0 0 1 1  
0 1 0 1  
1 1 1 0

## 样例输出

0 3 1 2

提示

在本题中，需要熟练掌握图的邻接矩阵存储方式。在建立完成无向图之后，需要严格按照题目描述的遍历顺序对图进行遍历。另外，算法中描述的FirstAdjVex函数和NextAdjVex函数，需要认真的自行探索并完成。在本题中需要使用队列结构，需要对队列的概念进行复习。

通过这道题目，应该能够对图的广度优先搜索建立更加直观和清晰的概念。

-->

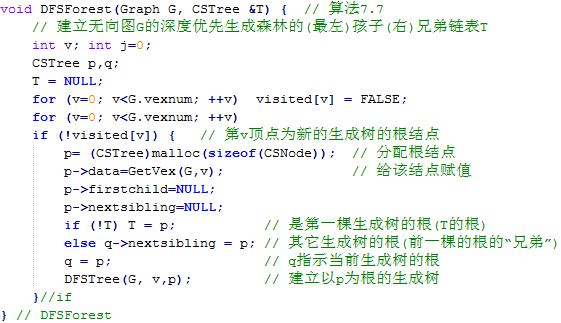
# 1764:算法7-7,7-8：无向图的连通分量和生成树

## 题目描述

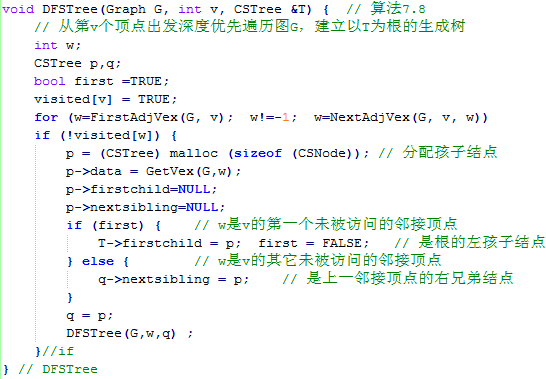
在对无向图进行遍历时，对于连通图，仅需从图中任一顶点出发，进行深度优先搜索或广度优先搜索，便可访问到图中所有顶点。对于非连通图，则需从多个顶点出发进行搜索，而每一次从一个新的起始点出发进行搜索的过程中得到的顶点访问序列恰为其各个连通分量中的顶点集。

对于非连通图，每个连通分量中的顶点集，和遍历时走过的边一起构成若干棵生成树，这些连通分量的生成树组成非连通图的生成森林。

假设以孩子兄弟链表作为生成森林的存储结构，则生成非连通图的深度优先生成森林的算法可以描述如下：



而建立以p为根的深度优先生成树的算法可以描述如下：



在本题中，读入一个无向图的邻接矩阵（即数组表示），建立无向图并按照以上描述中的算法建立无向图的生成森林。对于森林中的每一棵生成树，遍历所有顶点，并输出遍历顶点的顺序。

## 输入格式

输入的第一行包含一个正整数n，表示图中共有n个顶点。其中n不超过50。

以后的n行中每行有n个用空格隔开的整数0或1，对于第i行的第j个0或1，1表示第i个顶点和第j个顶点有直接连接，0表示没有直接连接。当i和j相等的时候，保证对应的整数为0。

输入保证邻接矩阵为对称矩阵，即输入的图一定是无向图。

## 输出

每一行输出无向图中的一棵生成树，表示按照题目描述中的深度优先遍历算法遍历相应的连通分量的访问顶点顺序。每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

6  
0 0 0 1 0 0  
0 0 1 1 0 0  
0 1 0 1 0 0  
1 1 1 0 0 0  
0 0 0 0 0 1  
0 0 0 0 1 0

## 样例输出

0 3 1 2   
4 5

## 提示

在本题中，需要掌握图的深度优先遍历的方法，并需要掌握无向图的连通性问题的本质。通过求出无向图的连通分量和对应的生成树，应该能够对图的连通性建立更加直观和清晰的概念。

-->

# 1765:算法7-9：最小生成树

## 题目描述

最小生成树问题是实际生产生活中十分重要的一类问题。假设需要在n个城市之间建立通信联络网，则连通n个城市只需要n-1条线路。这时，自然需要考虑这样一个问题，即如何在最节省经费的前提下建立这个通信网。

可以用连通网来表示n个城市以及n个城市之间可能设置的通信线路，其中网的顶点表示城市，边表示两个城市之间的线路，赋于边的权值表示相应的代价。对于n个顶点的连通网可以建立许多不同的生成树，每一棵生成树都可以是一个通信网。现在，需要选择一棵生成树，使总的耗费最小。这个问题就是构造连通网的最小代价生成树，简称最小生成树。一棵生成树的代价就是树上各边的代价之和。

而在常用的最小生成树构造算法中，普里姆（Prim）算法是一种非常常用的算法。以下是其算法的大致结构：



在本题中，读入一个无向图的邻接矩阵（即数组表示），建立无向图并按照以上描述中的算法建立最小生成树，并输出最小生成树的代价。

## 输入格式

输入的第一行包含一个正整数n，表示图中共有n个顶点。其中n不超过50。

以后的n行中每行有n个用空格隔开的整数，对于第i行的第j个整数，如果不为0，则表示第i个顶点和第j个顶点有直接连接且代价为相应的值，0表示没有直接连接。当i和j相等的时候，保证对应的整数为0。

输入保证邻接矩阵为对称矩阵，即输入的图一定是无向图，且保证图中只有一个连通分量。

## 输出

只有一个整数，即最小生成树的总代价。请注意行尾输出换行。

## 样例输入

4  
0 2 4 0  
2 0 3 5  
4 3 0 1  
0 5 1 0

## 样例输出

6

## 提示

在本题中，需要掌握图的深度优先遍历的方法，并需要掌握无向图的连通性问题的本质。通过求出无向图的连通分量和对应的生成树，应该能够对图的连通性建立更加直观和清晰的概念。

-->

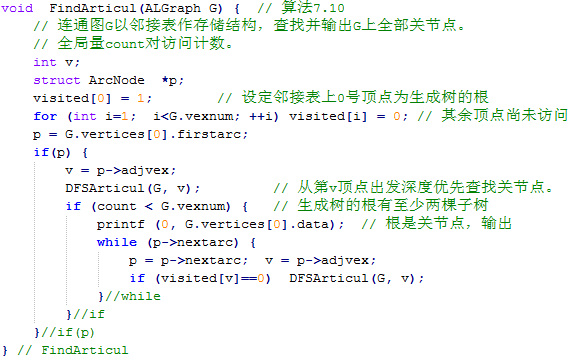
# 1766:算法7-10,7-11：关节点和重连通分量

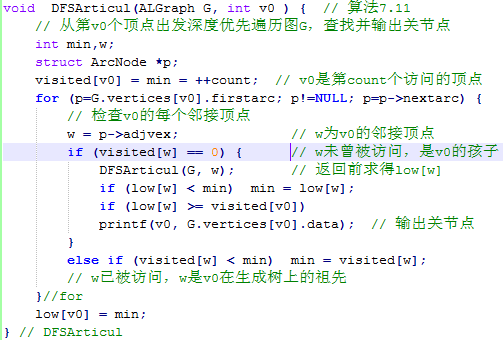
## 题目描述

假若在删去顶点v以及和v相关联的各边之后，将图的一个连通分量分割成两个或两个以上的连接分量，则称顶点v为该图的一个关节点。一个没有关节点的连通图称为重连通图。在重连通图上，任意一对顶点之间至少存在两条路径，则在删去某个顶点以及依附于该顶点的各边时也不会破坏图的连通性。

利用深度优先搜索可以求出图的关节点，并由此可以判断图是否是重连通的。

通过修改深度优先搜索遍历的算法便可以得到求关节点的算法，其算法描述如下：





在本题中，读入一个无向图的邻接矩阵（即数组表示），建立无向图并按照以上描述中的算法求出所有的关节点，并输出这些关节点。

## 输入格式

输入的第一行包含一个正整数n，表示图中共有n个顶点。其中n不超过50。

以后的n行中每行有n个用空格隔开的整数0或1，对于第i行的第j个整数，如果为1，则表示第i个顶点和第j个顶点有直接连接，0表示没有直接连接。当i和j相等的时候，保证对应的整数为0。

输入保证邻接矩阵为对称矩阵，即输入的图一定是无向图，且保证图中只有一个连通分量。

## 输出

第一行有一个整数x，即图中关节点的个数。

第二行输出x个整数，表示所有关节点的顶点编号，请按照编号从小到大的顺序输出。每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

4  
0 1 1 1  
1 0 0 0  
1 0 0 0  
1 0 0 0

## 样例输出

1  
0

## 提示

在本题中，需要掌握图的深度优先遍历的方法，并需要掌握通过深度优先搜索求得图中关节点的算法。通过生成深度优先生成树可以得出两类关节点的特性：

1.       若生成树的根有两棵或两棵以上的子树，则此根顶点必为关节点。

2.       若生成树中某个非叶子顶点v，其某棵子树的根和子树中的其他结点均没有指向v的祖先的回边，则说明v是关节点。

注意以上两点特性，就可以成功的通过深度优先搜索遍历的算法得出图中的关节点了。

-->

# 1767:算法7-12：有向无环图的拓扑排序

## 题目描述

由某个集合上的一个偏序得到该集合上的一个全序，这个操作被称为拓扑排序。偏序和全序的定义分别如下：

若集合X上的关系R是自反的、反对称的和传递的，则称R是集合X上的偏序关系。

设R是集合X上的偏序，如果对每个x,y∈X必有xRy或yRx，则称R是集合X上的全序关系。

由偏序定义得到拓扑有序的操作便是拓扑排序。

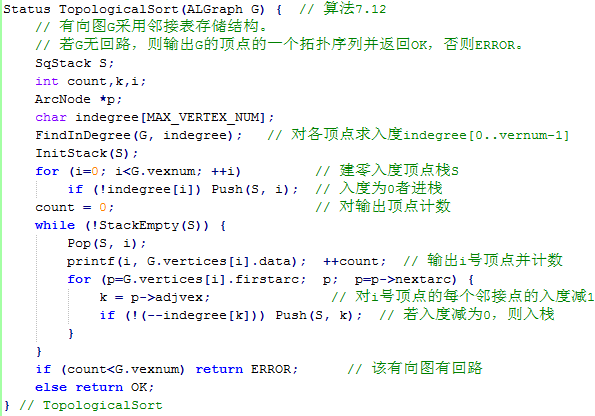
拓扑排序的流程如下：

1.       在有向图中选一个没有前驱的顶点并且输出之；

2.       从图中删除该顶点和所有以它为尾的弧。

重复上述两步，直至全部顶点均已输出，或者当前图中不存在无前驱的顶点为止。后一种情况则说明有向图中存在环。

采用邻接表存储有向图，并通过栈来暂存所有入度为零的顶点，可以描述拓扑排序的算法如下：



在本题中，读入一个有向图的邻接矩阵（即数组表示），建立有向图并按照以上描述中的算法判断此图是否有回路，如果没有回路则输出拓扑有序的顶点序列。

## 输入格式

输入的第一行包含一个正整数n，表示图中共有n个顶点。其中n不超过50。

以后的n行中每行有n个用空格隔开的整数0或1，对于第i行的第j个整数，如果为1，则表示第i个顶点有指向第j个顶点的有向边，0表示没有i指向j的有向边。当i和j相等的时候，保证对应的整数为0。

## 输出

如果读入的有向图含有回路，请输出“ERROR”，不包括引号。

如果读入的有向图不含有回路，请按照题目描述中的算法依次输出图的拓扑有序序列，每个整数后输出一个空格。

请注意行尾输出换行。

## 样例输入

4  
0 1 0 0  
0 0 1 0  
0 0 0 0  
0 0 1 0

## 样例输出

3 0 1 2

提示

在本题中，需要严格的按照题目描述中的算法进行拓扑排序，并在排序的过程中将顶点依次储存下来，直到最终能够判定有向图中不包含回路之后，才能够进行输出。

另外，为了避免重复检测入度为零的顶点，可以通过一个栈结构维护当前处理过程中入度为零的顶点。

-->

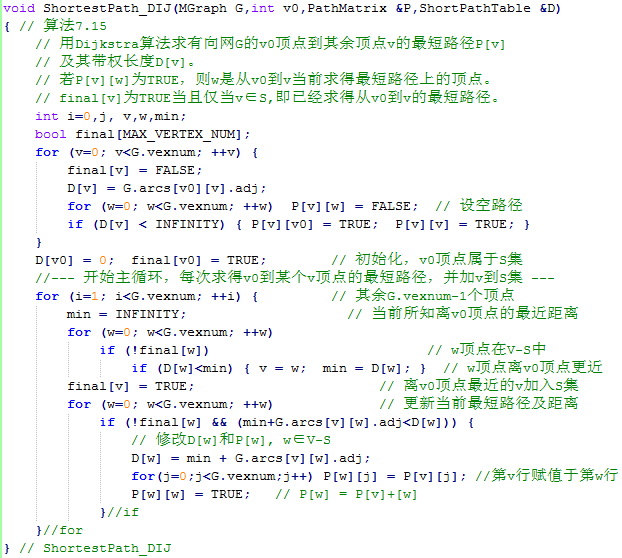
# 1768:算法7-15：迪杰斯特拉最短路径算法

## 题目描述

在带权有向图G中，给定一个源点v，求从v到G中的其余各顶点的最短路径问题，叫做单源点的最短路径问题。

在常用的单源点最短路径算法中，迪杰斯特拉算法是最为常用的一种，是一种按照路径长度递增的次序产生最短路径的算法。

可将迪杰斯特拉算法描述如下：



在本题中，读入一个有向图的带权邻接矩阵（即数组表示），建立有向图并按照以上描述中的算法求出源点至每一个其它顶点的最短路径长度。

## 输入格式

输入的第一行包含2个正整数n和s，表示图中共有n个顶点，且源点为s。其中n不超过50，s小于n。

以后的n行中每行有n个用空格隔开的整数。对于第i行的第j个整数，如果大于0，则表示第i个顶点有指向第j个顶点的有向边，且权值为对应的整数值；如果这个整数为0，则表示没有i指向j的有向边。当i和j相等的时候，保证对应的整数为0。

## 输出

只有一行，共有n-1个整数，表示源点至其它每一个顶点的最短路径长度。如果不存在从源点至相应顶点的路径，输出-1。

请注意行尾输出换行。

## 样例输入

4 1  
0 3 0 1  
0 0 4 0  
2 0 0 0  
0 0 1 0

## 样例输出

6 4 7

## 提示

在本题中，需要按照题目描述中的算法完成迪杰斯特拉算法，并在计算最短路径的过程中将每个顶点是否可达记录下来，直到求出每个可达顶点的最短路径之后，算法才能够结束。

迪杰斯特拉算法的特点是按照路径长度递增的顺序，依次添加下一条长度最短的边，从而不断构造出相应顶点的最短路径。

另外需要注意的是，在本题中为了更方便的表示顶点间的不可达状态，可以使用一个十分大的值作为标记。

-->

# 1769:算法7-16：弗洛伊德最短路径算法

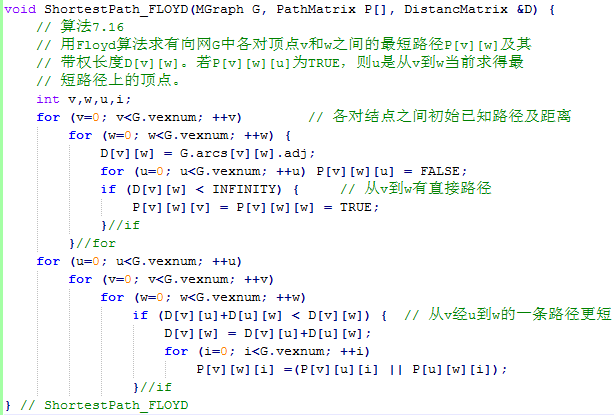
## 题目描述

在带权有向图G中，求G中的任意一对顶点间的最短路径问题，也是十分常见的一种问题。

解决这个问题的一个方法是执行n次迪杰斯特拉算法，这样就可以求出每一对顶点间的最短路径，执行的时间复杂度为O(n3)。

而另一种算法是由弗洛伊德提出的，时间复杂度同样是O(n3)，但算法的形式简单很多。

可以将弗洛伊德算法描述如下：



在本题中，读入一个有向图的带权邻接矩阵（即数组表示），建立有向图并按照以上描述中的算法求出每一对顶点间的最短路径长度。

## 输入格式

输入的第一行包含1个正整数n，表示图中共有n个顶点。其中n不超过50。

以后的n行中每行有n个用空格隔开的整数。对于第i行的第j个整数，如果大于0，则表示第i个顶点有指向第j个顶点的有向边，且权值为对应的整数值；如果这个整数为0，则表示没有i指向j的有向边。当i和j相等的时候，保证对应的整数为0。

## 输出

共有n行，每行有n个整数，表示源点至每一个顶点的最短路径长度。如果不存在从源点至相应顶点的路径，输出-1。对于某个顶点到其本身的最短路径长度，输出0。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

4  
0 3 0 1  
0 0 4 0  
2 0 0 0  
0 0 1 0

## 样例输出

0 3 2 1   
6 0 4 7   
2 5 0 3   
3 6 1 0

## 提示

在本题中，需要按照题目描述中的算法完成弗洛伊德算法，并在计算最短路径的过程中将每个顶点是否可达记录下来，直到求出每一对顶点的最短路径之后，算法才能够结束。

相对于迪杰斯特拉算法，弗洛伊德算法的形式更为简单。通过一个三重循环，弗洛伊德算法可以方便的求出每一对顶点间的最短距离。

另外需要注意的是，为了更方便的表示顶点间的不可达状态，可以使用一个十分大的值作为标记。而在题目描述中的算法示例使用了另外一个三维数组对其进行表示，这使原本的O(n3)时间复杂度增长到了O(n4)，这也是需要自行修改的部分。

-->

# 1813:还是畅通工程

## 题目描述

        某省调查乡村交通状况，得到的统计表中列出了任意两村庄间的距离。省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可），并要求铺设的公路总长度为最小。请计算最小的公路总长度。

## 输入格式

        测试输入包含若干测试用例。每个测试用例的第1行给出村庄数目N ( < 100 )；随后的N(N-1)/2行对应村庄间的距离，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间的距离。为简单起见，村庄从1到N编号。  
        当N为0时，输入结束，该用例不被处理。

## 输出

        对每个测试用例，在1行里输出最小的公路总长度。

## 样例输入

8  
1 2 42  
1 3 68  
1 4 35  
1 5 1  
1 6 70  
1 7 25  
1 8 79  
2 3 59  
2 4 63  
2 5 65  
2 6 6  
2 7 46  
2 8 82  
3 4 28  
3 5 62  
3 6 92  
3 7 96  
3 8 43  
4 5 28  
4 6 37  
4 7 92  
4 8 5  
5 6 3  
5 7 54  
5 8 93  
6 7 83  
6 8 22  
7 8 17  
0

## 样例输出

82

提示

-->

# 1953:畅通工程

## 题目描述

某省调查城镇交通状况，得到现有城镇道路统计表，表中列出了每条道路直接连通的城镇。省政府“畅通工程”的目标是使全省任何两个城镇间都可以实现交通（但不一定有直接的道路相连，只要互相间接通过道路可达即可）。问最少还需要建设多少条道路？

## 输入格式

测试输入包含若干测试用例。每个测试用例的第1行给出两个正整数，分别是城镇数目N ( < 1000 )和道路数目M；随后的M行对应M条道路，每行给出一对正整数，分别是该条道路直接连通的两个城镇的编号。为简单起见，城镇从1到N编号。   
    注意:两个城市之间可以有多条道路相通,也就是说  
    3 3  
    1 2  
    1 2  
    2 1  
    这种输入也是合法的  
    当N为0时，输入结束，该用例不被处理。

## 输出

对每个测试用例，在1行里输出最少还需要建设的道路数目。

## 样例输入

5 3  
1 2  
3 2  
4 5  
0

## 样例输出

1

## 提示

-->

# 1954:畅通工程

## 题目描述

省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可）。经过调查评估，得到的统计表中列出了有可能建设公路的若干条道路的成本。现请你编写程序，计算出全省畅通需要的最低成本。

## 输入格式

测试输入包含若干测试用例。每个测试用例的第1行给出评估的道路条数 N、村庄数目M (N, M < =100 )；随后的 N 行对应村庄间道路的成本，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间道路的成本（也是正整数）。为简单起见，村庄从1到M编号。当N为0时，全部输入结束，相应的结果不要输出。

## 输出

对每个测试用例，在1行里输出全省畅通需要的最低成本。若统计数据不足以保证畅通，则输出“?”。

## 样例输入

3 4  
1 2 1  
2 3 2  
3 4 3  
2 4  
1 2 1  
3 4 2  
0 5

## 样例输出

6  
?

提示

-->

# 1955:继续畅通工程

## 题目描述

省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可）。现得到城镇道路统计表，表中列出了任意两城镇间修建道路的费用，以及该道路是否已经修通的状态。现请你编写程序，计算出全省畅通需要的最低成本。

## 输入格式

测试输入包含若干测试用例。每个测试用例的第1行给出村庄数目N ( 1< N < 100 )；随后的 N(N-1)/2 行对应村庄间道路的成本及修建状态，每行给4个正整数，分别是两个村庄的编号（从1编号到N），此两村庄间道路的成本，以及修建状态：1表示已建，0表示未建。  
  
当N为0时输入结束。

## 输出

每个测试用例的输出占一行，输出全省畅通需要的最低成本。

## 样例输入

4  
1 2 1 1  
1 3 6 0  
1 4 2 1  
2 3 3 0  
2 4 5 0  
3 4 4 0  
3  
1 2 1 1  
2 3 2 1  
1 3 1 0  
0

## 样例输出

3  
0

## 提示

-->

# 1908:连通图

## 题目描述

给定一个无向图和其中的所有边，判断这个图是否所有顶点都是连通的。

## 输入格式

每组数据的第一行是两个整数 n 和 m（0<=n<=1000）。n 表示图的顶点数目，m 表示图中边的数目。如果 n 为 0 表示输入结束。随后有 m 行数据，每行有两个值 x 和 y（0<x, y <=n），表示顶点 x 和 y 相连，顶点的编号从 1 开始计算。输入不保证这些边是否重复。

## 输出

对于每组输入数据，如果所有顶点都是连通的，输出"YES"，否则输出"NO"。

## 样例输入

4 3  
4 3  
1 2  
1 3  
5 7  
3 5  
2 3  
1 3  
3 2  
2 5  
3 4  
4 1  
7 3  
6 2  
3 1  
5 6  
0 0

## 样例输出

YES  
YES  
NO

## 提示

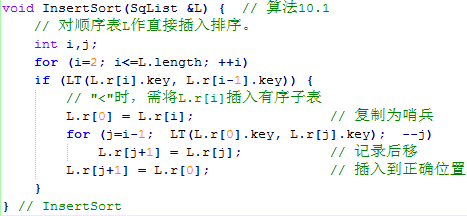
-->

# 1774:算法10-1：直接插入排序

## 题目描述

直接插入排序是一种最简单的排序方法，它的基本操作是将一个记录插入到已经排好序的有序表中，从而得到一个新的且记录数增加了1的有序表。

其算法可以描述如下：



在本题中，读入一串整数，将其使用以上描述的直接插入排序的方法从小到大排序，并输出。

## 输入格式

输入的第一行包含1个正整数n，表示共有n个整数需要参与排序。其中n不超过1000。

第二行包含n个用空格隔开的正整数，表示n个需要排序的整数。

## 输出

只有1行，包含n个整数，表示从小到大排序完毕的所有整数。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

10  
2 8 4 6 1 10 7 3 5 9

## 样例输出

1 2 3 4 5 6 7 8 9 10

## 提示

在本题中，需要按照题目描述中的算法完成直接插入排序的算法。直接插入排序算法的思路非常直接，就是通过将记录插入到已经有序的有序表中，并重排记录。

通过分析，不难发现直接插入排序的时间复杂度为O(n2)，并不是一种非常高效的排序方法。

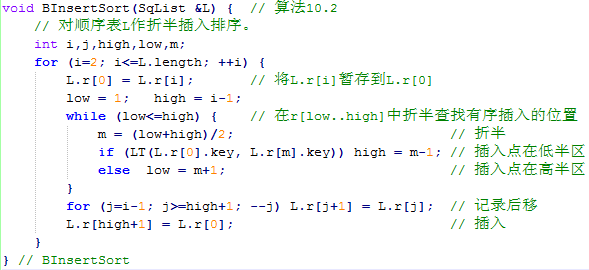
-->

# 1775:算法10-2：折半插入排序

## 题目描述

折半插入排序同样是一种非常简单的排序方法，它的基本操作是在一个已经排好序的有序表中进行查找和插入。不难发现这个查找的过程可以十分自然的修改成折半查找的方式进行实现。

折半插入排序的算法可以描述如下：



在本题中，读入一串整数，将其使用以上描述的折半插入排序的方法从小到大排序，并输出。

## 输入格式

输入的第一行包含1个正整数n，表示共有n个整数需要参与排序。其中n不超过1000。

第二行包含n个用空格隔开的正整数，表示n个需要排序的整数。

## 输出

只有1行，包含n个整数，表示从小到大排序完毕的所有整数。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

10  
2 8 4 6 1 10 7 3 5 9

## 样例输出

1 2 3 4 5 6 7 8 9 10

## 提示

在本题中，需要按照题目描述中的算法完成折半插入排序的算法。与直接插入排序算法不同，折半插入排序算法在查找插入位置时采用了折半查找的方案，减少了关键字之间的比较次数，但是记录的移动次数并没有发生改变，因此折半插入排序的时间复杂度依旧为O(n2)，同样不是一种非常高效的排序方法。

-->

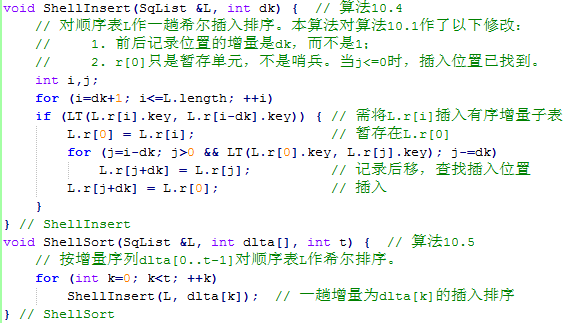
# 1776:算法10-4,10-5：希尔排序

## 题目描述

希尔排序又称“缩小增量排序”，它是一种属于插入排序类的排序方法，但是在时间效率方面较普通的插入排序方法有较大的改进。

希尔排序的基本思想是：先将整个待排序的序列分割成为若干子序列，并分别进行直接插入排序，当整个序列中的记录基本有序时，再对全体记录进行一次直接插入排序。

希尔排序的算法可以描述如下：



在本题中，读入一串整数，将其使用以上描述的希尔排序的方法从小到大排序，并输出。

## 输入格式

输入的第一行包含1个正整数n，表示共有n个整数需要参与排序。其中n不超过1000。

第二行包含n个用空格隔开的正整数，表示n个需要排序的整数。

## 输出

只有1行，包含n个整数，表示从小到大排序完毕的所有整数。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

10  
2 8 4 6 1 10 7 3 5 9

样例输出

1 2 3 4 5 6 7 8 9 10

## 提示

在本题中，需要按照题目描述中的算法完成希尔排序的算法。

希尔排序的时间复杂度分析是一个复杂的问题，因为其时间与所取的“增量序列”相关。目前为止尚未有人求得一种最好的增量序列。增量序列可以有多种不同的取法，但是需要注意的是，在选取增量序列时务必使序列中的值没有除1以外的公因子，且最后一个增量值必须等于1。

与直接插入排序相比，通过选择合适的增量序列，希尔排序算法能够使排序的效率得到部分的提高，但是由于其形式的复杂性和增量序列选择并没有定论，所以在实际应用中希尔排序是非常少见的。

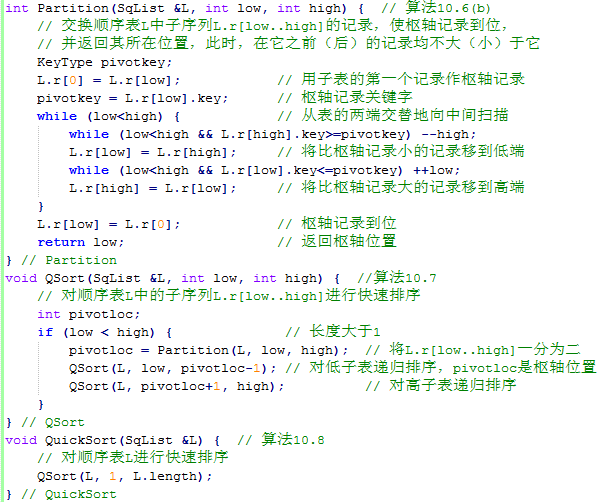
-->

# 1777:算法10-6~10-8：快速排序

## 题目描述

快速排序是对起泡排序的一种改进。它的基本思想是，通过一趟排序将待排序的记录分割成两个独立的部分，其中一部分记录的关键字均比另一部分的关键字小，在分成两个部分之后则可以分别对这两个部分继续进行排序，从而使整个序列有序。

快速排序的算法可以描述如下：



在本题中，读入一串整数，将其使用以上描述的快速排序的方法从小到大排序，并输出。

## 输入格式

输入的第一行包含1个正整数n，表示共有n个整数需要参与排序。其中n不超过100000。

第二行包含n个用空格隔开的正整数，表示n个需要排序的整数。

## 输出

只有1行，包含n个整数，表示从小到大排序完毕的所有整数。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

10  
2 8 4 6 1 10 7 3 5 9

## 样例输出

1 2 3 4 5 6 7 8 9 10

## 提示

在本题中，需要按照题目描述中的算法完成快速排序的算法。

快速排序是一种十分常用的排序算法，其平均时间复杂度为O(kn*ln*n)，其中n为待排序序列中记录的个数，k为常数。大量的实际应用证明，在所有同数量级的此类排序算法中，快速排序的常数因子k是最小的，因此，就平均时间而言，快速排序是目前被认为最好的一种内部排序方法。

而在C语言的常用编译器中，qsort函数是一个非常常用的快速排序函数。

-->

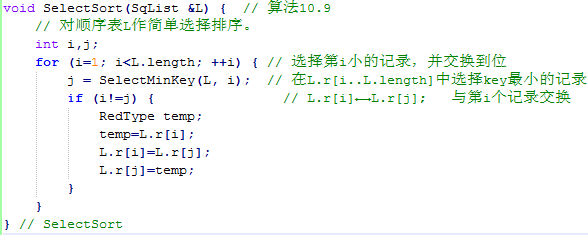
# 1778:算法10-9：简单选择排序

## 题目描述

选择排序的基本思想是：每一趟比较过程中，在n-i+1(i=1,2,...,n-1)个记录中选取关键字最小的记录作为有序序列中的第i个记录。

在多种选择排序中，最常用且形式最为简单的是简单选择排序。

简单选择排序的算法可以描述如下：



在本题中，读入一串整数，将其使用以上描述的简单选择排序的方法从小到大排序，并输出。

## 输入格式

输入的第一行包含1个正整数n，表示共有n个整数需要参与排序。其中n不超过1000。

第二行包含n个用空格隔开的正整数，表示n个需要排序的整数。

## 输出

只有1行，包含n个整数，表示从小到大排序完毕的所有整数。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

10  
2 8 4 6 1 10 7 3 5 9

## 样例输出

1 2 3 4 5 6 7 8 9 10

提示

在本题中，需要按照题目描述中的算法完成简单选择排序的算法。

简单选择排序是一种思路非常简单，结构简洁的排序算法。它的特点是，无论记录的初始排列形式如何，所需进行的关键字间的比较次数始终相同，均为n(n-1)/2。因此，其时间复杂度也固定在O(n2)。

-->

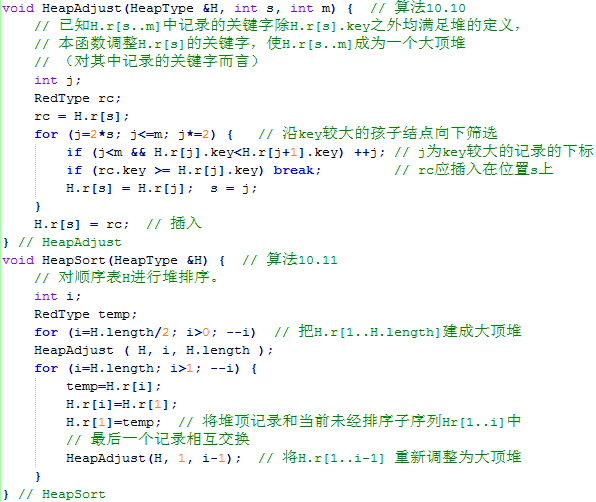
# 1779:算法10-10,10-11：堆排序

## 题目描述

堆排序是一种利用堆结构进行排序的方法，它只需要一个记录大小的辅助空间，每个待排序的记录仅需要占用一个存储空间。

首先建立小根堆或大根堆，然后通过利用堆的性质即堆顶的元素是最小或最大值，从而依次得出每一个元素的位置。

堆排序的算法可以描述如下：



在本题中，读入一串整数，将其使用以上描述的堆排序的方法从小到大排序，并输出。

## 输入格式

输入的第一行包含1个正整数n，表示共有n个整数需要参与排序。其中n不超过100000。

第二行包含n个用空格隔开的正整数，表示n个需要排序的整数。

## 输出

只有1行，包含n个整数，表示从小到大排序完毕的所有整数。

请在每个整数后输出一个空格，并请注意行尾输出换行。

样例输入

10  
2 8 4 6 1 10 7 3 5 9

## 样例输出

1 2 3 4 5 6 7 8 9 10

## 提示

在本题中，需要按照题目描述中的算法完成堆排序的算法。

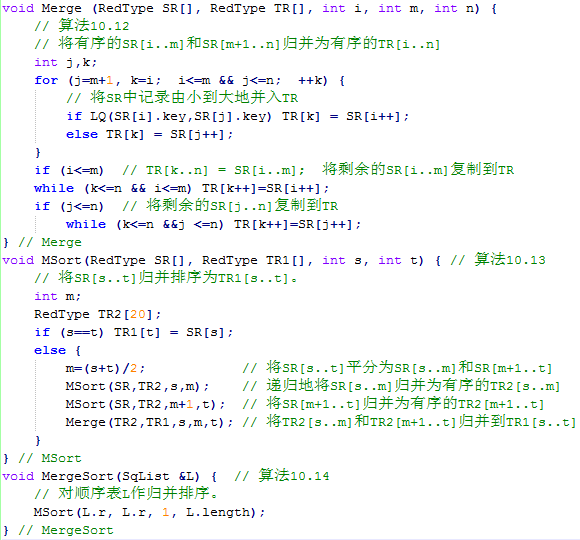
堆排序对于元素数较多的情况是非常有效的。通过对算法的分析，不难发现在建立含有n个元素的堆时，总共进行的关键字比较次数不会超过4n，且n个节点的堆深度是log2n数量级的。因此，堆排序在最坏情况下的时间复杂度是O(nlog2n)，相对于快速排序，堆排序具有同样的时间复杂度级别，但是其不会退化。堆排序较快速排序的劣势是其常数相对较大。

-->

# 1780:算法10-12~10-14：归并排序

## 题目描述

归并排序是基于归并操作完成的，而一次归并操作是通过两个或两个以上的有序表合并成一个新的有序表完成的。常见的归并排序是2-路归并排序，其核心操作是将一维数组中前后相邻的两个有序序列归并成一个有序序列。其算法可以描述如下：



在本题中，读入一串整数，将其使用以上描述的2-路归并排序的方法从小到大排序，并输出。

## 输入格式

输入的第一行包含1个正整数n，表示共有n个整数需要参与排序。其中n不超过100000。

第二行包含n个用空格隔开的正整数，表示n个需要排序的整数。

## 输出

只有1行，包含n个整数，表示从小到大排序完毕的所有整数。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

10  
2 8 4 6 1 10 7 3 5 9

## 样例输出

1 2 3 4 5 6 7 8 9 10

## 提示

在本题中，需要按照题目描述中的算法完成2-路归并排序的算法。

不难发现，2-路归并算法的时间复杂度为O(nlog2n)，且需要和原始数据等数量的辅助空间。递归形式的2-路归并排序算法在形式上比较简洁，但是实用性较差。通常在需要使用归并排序的场合，往往使用非递归形式的归并排序。

与快速排序和堆排序相比，归并排序的特点在于其是一种稳定的排序方法。

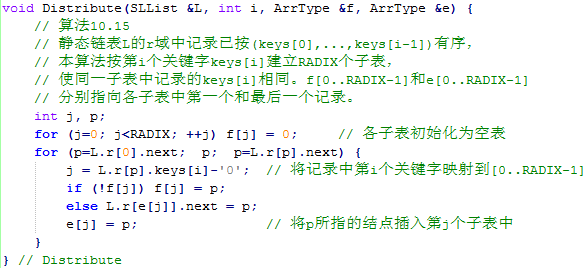
-->

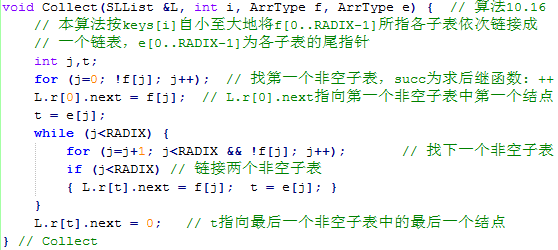
# 1781:算法10-15~10-17：基数排序

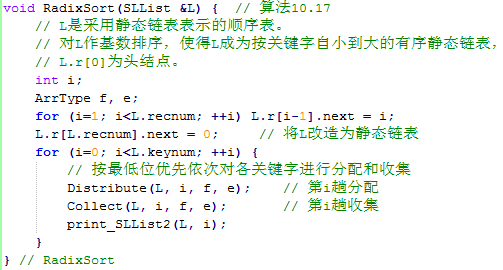
## 题目描述

基数排序是一种并不基于关键字间比较和移动操作的排序算法。基数排序是一种借助多关键字排序的思想对单逻辑关键字进行排序的方法。

通过对每一个关键字分别依次进行排序，可以令整个关键字序列得到完整的排序。而采用静态链表存储记录，并使用基数排序对记录进行排序操作的排序算法被称为链式基数排序。其算法可以描述如下：







在本题中，读入一串16位（16bit）正整数，将其使用以上描述的2-路归并排序的方法从小到大排序，并输出。

## 输入格式

输入的第一行包含1个正整数n，表示共有n个正整数需要参与排序。其中n不超过100000，保证所有正整数不大于32767。

第二行包含n个用空格隔开的正整数，表示n个需要排序的整数。

## 输出

只有1行，包含n个整数，表示从小到大排序完毕的所有整数。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

10  
2 8 4 6 1 10 7 3 5 9

## 样例输出

1 2 3 4 5 6 7 8 9 10

## 提示

在本题中，需要按照题目描述中的算法完成链式基数排序的算法。

不难发现，链式基数排序算法对于n个记录，假设每个记录包含d个关键字，每个关键字的取值范围是r\*d个值，则其时间复杂度为O(d(n+rd))，其中每一趟分配的时间复杂度为O(n)，每一趟收集的时间复杂度为O(rd)，整个排序过程需要进行d趟分配和收集。

通过分析链式基数排序，可以发现不基于关键字比较和移动的排序算法拥有与其他内部排序算法的时间复杂度。

-->

# 1782:谁是你的潜在朋友

## 题目描述

    “臭味相投”——这是我们描述朋友时喜欢用的词汇。两个人是朋友通常意味着他们存在着许多共同的兴趣。然而作为一个宅男，你发现自己与他人相互了解的机会 并不太多。幸运的是，你意外得到了一份北大图书馆的图书借阅记录，于是你挑灯熬夜地编程，想从中发现潜在的朋友。  
    首先你对借阅记录进行了一番整理，把N个读者依次编号为1,2,…,N，把M本书依次编号为1,2,…,M。同时，按照“臭味相投”的原则，和你喜欢读同一本书的人，就是你的潜在朋友。你现在的任务是从这份借阅记录中计算出每个人有几个潜在朋友。

## 输入格式

    每个案例第一行两个整数N,M，2 <= N ，M<= 200。接下来有N行，第i(i = 1,2,…,N)行每一行有一个数，表示读者i-1最喜欢的图书的编号P(1<=P<=M)

## 输出

    每个案例包括N行，每行一个数，第i行的数表示读者i有几个潜在朋友。如果i和任何人都没有共同喜欢的书，则输出“BeiJu”（即悲剧，^ ^）

## 样例输入

4 5  
2  
3  
2  
1

## 样例输出

1  
BeiJu  
1  
BeiJu

提示

-->

# 1791:后缀子串排序

## 题目描述

对于一个字符串，将其后缀子串进行排序，例如grain  
其子串有：  
grain  
rain  
ain  
in  
n  
然后对各子串按字典顺序排序，即：  
ain,grain,in,n,rain

## 输入格式

每个案例为一行字符串。

## 输出

将子串排序输出

## 样例输入

grain  
banana

## 样例输出

ain  
grain  
in  
n  
rain  
a  
ana  
anana  
banana  
na  
nana

## 提示

-->

# 1923:排序

## 题目描述

对输入的n个数进行排序并输出。

## 输入格式

输入的第一行包括一个整数n(1<=n<=100)。 接下来的一行包括n个整数。

## 输出

可能有多组测试数据，对于每组数据，将排序后的n个整数输出，每个数后面都有一个空格。  
每组测试数据的结果占一行。

## 样例输入

5  
5 4 3 1 2

## 样例输出

1 2 3 4 5

提示

-->

# 1924:成绩排序

## 题目描述

有N个学生的数据，将学生数据按成绩高低排序，如果成绩相同则按姓名字符的字母序排序，如果姓名的字母序也相同则按照学生的年龄排序，并输出N个学生排序后的信息。

## 输入格式

测试数据有多组，每组输入第一行有一个整数N（N<=1000），接下来的N行包括N个学生的数据。  
每个学生的数据包括姓名（长度不超过100的字符串）、年龄（整形数）、成绩（小于等于100的正数）。

## 输出

将学生信息按成绩进行排序，成绩相同的则按姓名的字母序进行排序。  
然后输出学生信息，按照如下格式：  
姓名 年龄 成绩

## 样例输入

3  
zhao 19 90  
qian 20 90  
sun 19 100

## 样例输出

qian 20 90  
zhao 19 90  
sun 19 100

## 提示

简单的排序题，用C++的sort十分方便。

-->

# 1925:特殊排序

## 题目描述

输入一系列整数，将其中最大的数挑出，并将剩下的数进行排序。

## 输入格式

输入第一行包括1个整数N，1<=N<=1000，代表输入数据的个数。

接下来的一行有N个整数。

## 输出

可能有多组测试数据，对于每组数据，

第一行输出一个整数，代表N个整数中的最大值，并将此值从数组中去除，将剩下的数进行排序。

第二行将排序的结果输出。

## 样例输入

5  
5 3 2 4 1

## 样例输出

5  
1 2 3 4

## 提示

如果数组中只有一个数，当第一行将其输出后，第二行请输出"-1"。

-->

# 1926:EXCEL排序

## 题目描述

Excel可以对一组纪录按任意指定列排序。现请你编写程序实现类似功能。

对每个测试用例，首先输出1行“Case i:”，其中 i 是测试用例的编号（从1开始）。随后在 N 行中输出按要求排序后的结果，即：当 C=1 时，按学号递增排序；当 C=2时，按姓名的非递减字典序排序；当 C=3 时，按成绩的非递减排序。当若干学生具有相同姓名或者相同成绩时，则按他们的学号递增排序。

## 输入格式

测试输入包含若干测试用例。每个测试用例的第1行包含两个整数 N (N<=100000) 和 C，其中 N 是纪录的条数，C 是指定排序的列号。以下有N行，每行包含一条学生纪录。每条学生纪录由学号（6位数字，同组测试中没有重复的学号）、姓名（不超过8位且不包含空格的字符串）、成绩（闭区间[0, 100]内的整数）组成，每个项目间用1个空格隔开。当读到 N=0 时，全部输入结束，相应的结果不要输出。

## 输出

对每个测试用例，首先输出1行“Case i:”，其中 i 是测试用例的编号（从1开始）。随后在 N 行中输出按要求排序后的结果，即：当 C=1 时，按学号递增排序；当 C=2时，按姓名的非递减字典序排序；当 C=3 时，按成绩的非递减排序。当若干学生具有相同姓名或者相同成绩时，则按他们的学号递增排序。

## 样例输入

4 1  
000001 Zhao 75  
000004 Qian 88  
000003 Li 64  
000002 Sun 90  
4 2  
000005 Zhao 95  
000011 Zhao 75  
000007 Qian 68  
000006 Sun 85  
4 3  
000002 Qian 88  
000015 Li 95  
000012 Zhao 70  
000009 Sun 95  
0 3

## 样例输出

Case 1:  
000001 Zhao 75  
000002 Sun 90  
000003 Li 64  
000004 Qian 88  
Case 2:  
000007 Qian 68  
000006 Sun 85  
000005 Zhao 95  
000011 Zhao 75  
Case 3:  
000012 Zhao 70  
000002 Qian 88  
000009 Sun 95  
000015 Li 95

## 提示

-->

# 1927:字符串内排序

## 题目描述

输入一个字符串，长度小于等于200，然后将输出按字符顺序升序排序后的字符串。

## 输入格式

测试数据有多组，输入字符串。

## 输出

对于每组输入,输出处理后的结果。

样例输入

tianqin

样例输出

aiinnqt

## 提示

注意输入的字符串中可能有空格。

-->

# 1951:大整数排序

## 题目描述

对N个长度最长可达到1000的数进行排序。

## 输入格式

输入第一行为一个整数N，(1<=N<=100)。  
接下来的N行每行有一个数，数的长度范围为1<=len<=1000。  
每个数都是一个正数，并且保证不包含前缀零。

## 输出

可能有多组测试数据，对于每组数据，将给出的N个数从小到大进行排序，输出排序后的结果，每个数占一行。

## 样例输入

4  
123  
1234  
12345  
2345

## 样例输出

123  
1234  
2345  
12345

## 提示

-->

# 1998:成绩排序

## 题目描述

从键盘输入4 个学生的数据(包括姓名、年龄和成绩)，按成绩从高到底排序，并输出其中成绩次高者的所有数据。

## 输入格式

 输入4行，每一行表示一个学生的数据，分别表示 姓名、年龄和成绩。

## 输出

 输出4行，表示排序后的结果，格式参见样例。

## 样例输入

aa 10 90  
bb 20 100  
cc 18 98  
dd 19 93

## 样例输出

bb 20 100  
cc 18 98  
dd 19 93  
aa 10 90

## 提示

-->

# 2005:字符排序

## 题目描述

﻿任意输入一个长度不超过20的字符串，对所输入的字符串，按照ASCII码的大小从小到大进行排序，请输出排序后的结果。

## 输入格式

输入第一行为样例数m，接下来m行每行一个字符串，字符串长度不超过20。

## 输出

输出m行表示排序完的字符串。

## 样例输入

1  
dcab

## 样例输出

abcd

提示

-->

# 2007:成绩排序

## 题目描述

用一维数组存储学号和成绩，然后，按成绩排序输出。

## 输入格式

输入第一行包括一个整数N(1<=N<=100)，代表学生的个数。  
接下来的N行每行包括两个整数p和q，分别代表每个学生的学号和成绩。

## 输出

按照学生的成绩从小到大进行排序，并将排序后的学生信息打印出来。  
如果学生的成绩相同，则按照学号的大小进行从小到大排序。

## 样例输入

1  
1 68  
3  
1 1  
2 70  
3 25

## 样例输出

1 68  
1 1  
3 25  
2 70

## 提示

-->

# 2040:数组排序

## 题目描述

输入一个数组的值,求出各个值从小到大排序后的次序。

## 输入格式

输入有多组数据。  
每组输入的第一个数为数组的长度n(1<=n<=10000),后面的数为数组中的值,以空格分割。

## 输出

各输入的值按从小到大排列的次序(最后一个数字后面没有空格)。

## 样例输入

1  
68   
15  
1 70 25 79 59 63 65 6 46 82 28 62 92 96 43

样例输出

1  
1 11 3 12 7 9 10 2 6 13 4 8 14 15 5

## 提示

-->

# 2057:字符串排序

## 题目描述

先输入你要输入的字符串的个数。然后换行输入该组字符串。每个字符串以回车结束，每个字符串少于一百个字符。  
如果在输入过程中输入的一个字符串为“stop”，也结束输入。  
然后将这输入的该组字符串按每个字符串的长度，由小到大排序，按排序结果输出字符串。

## 输入格式

字符串的个数，以及该组字符串。每个字符串以‘\n’结束。如果输入字符串为“stop”，也结束输入.

## 输出

可能有多组测试数据，对于每组数据，  
将输入的所有字符串按长度由小到大排序输出(如果有“stop”，不输出“stop”)。

## 样例输入

4  
faeruhyytrjh tjytj  
hsrthts   hjnshtgfhs  
stop  
3  
htrskbns  
bsartanjsf tyjndyt  
nsr jj jtey

## 样例输出

faeruhyytrjh tjytj  
hsrthts   hjnshtgfhs  
htrskbns  
nsr jj jtey  
bsartanjsf tyjndyt

## 提示

根据输入的字符串个数来动态分配存储空间（采用new()函数）。每个字符串会少于100个字符。  
测试数据有多组，注意使用while()循环输入。

-->

# 2062:姓名排序

## 题目描述

存储一组姓名，如Apple,Tom,Green,Jack 要求按照字典序排序并显示。

## 输入格式

输入第一行为样例数m，对于每个样例，第一行为人数n，接下来有n个姓名,n不超过10，每个名字长度不超过20。

## 输出

对于每个样例输出排序后的结果，每行一个姓名。

## 样例输入

1  
4  
Apple  
Tom  
Green  
Jack

## 样例输出

Apple  
Green  
Jack  
Tom

## 提示

-->

# 2080:整数奇偶排序

## 题目描述

输入10个整数，彼此以空格分隔。重新排序以后输出(也按空格分隔)，要求:  
1.先输出其中的奇数,并按从大到小排列；  
2.然后输出其中的偶数,并按从小到大排列。

## 输入格式

任意排序的10个整数（0～100），彼此以空格分隔。

## 输出

可能有多组测试数据，对于每组数据，按照要求排序后输出，由空格分隔。

## 样例输入

0 56 19 81 59 48 35 90 83 75   
17 86 71 51 30 1 9 36 14 16

## 样例输出

83 81 75 59 35 19 0 48 56 90  
71 51 17 9 1 14 16 30 36 86

## 提示

多组数据，注意输出格式

1. 测试数据可能有很多组，请使用while(cin>>a[0]>>a[1]>>...>>a[9])类似的做法来实现;  
2. 输入数据随机，有可能相等。

-->

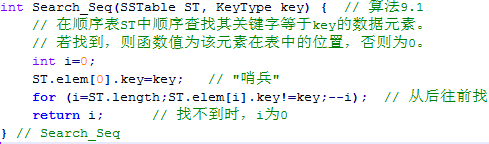
# 1770:算法9-1：静态表的顺序查找

## 题目描述

用顺序表或者线性链表表示静态查找表时，搜索函数可以采用顺序查找来实现。

通常顺序查找的查找过程是从表中的自后一个记录开始，逐个将记录的关键字和给定的查找值进行比较，如果某个记录的关键字与给定的值比较相等，则说明查找成功；否则如果直到第一个记录，所有的关键字都与给定的值不相等，说明表中没有响应的记录，查找失败。

其查找过程可以描述如下：



在本题中，读入一串整数，另外给定多次查询，判断每一次查询是否找到了相应的整数，如果找到则输出整数相应的位置。

## 输入格式

输入的第一行包含2个正整数n和k，分别表示共有n个整数和k次查询。其中n不超过500，k同样不超过500。

第二行包含n个用空格隔开的正整数，表示n个原始记录。

第三行包含k个用空格隔开的正整数，表示k次查询的目标。

## 输出

只有1行，包含k个整数，分别表示每一次的查询结果。如果在查询中找到了对应的整数，则输出其相应的位置，否则输出-1。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

8 3  
1 3 5 7 8 9 10 15  
9 2 5

## 样例输出

5 -1 2

## 提示

在本题中，需要按照题目描述中的算法完成静态查找表的查找过程。通过将需要查询的值与当前的元素值进行比较，不断向表头移动，直到确定被查询的值是否存在。

通过分析，不难发现顺序查找的时间复杂度为O(n)，并不是一种非常高效的查找方法。

-->

# 1772:算法9-5~9-8：二叉排序树的基本操作

## 题目描述

二叉排序树或者是一棵空树，或者是具有以下几条性质的二叉树：

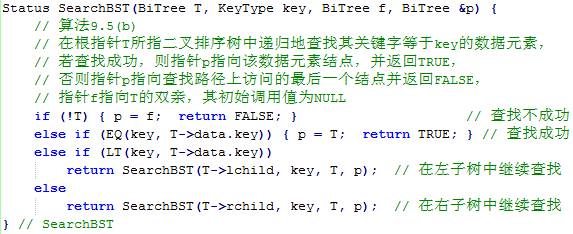
1.       若它的左子树不空，则左子树上所有结点的值均小于它的根节点的值；

2.       若它的右子树不空，则右子树上所有结点的值均大于它的根节点的值；

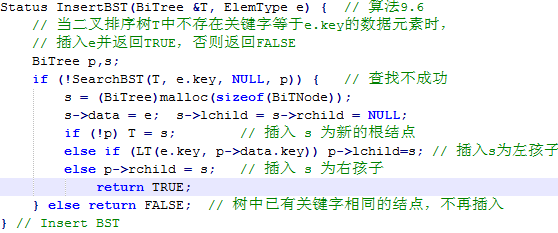
3.       它的左右子树也分别为二叉排序树。

二叉排序树又可以被称为二叉查找树，根据上述定义的结构不难知道，它的查找过程十分简单，只需要通过不断的将当前结点的值与需要查找的值进行比较，如果相等则直接输出，如果要查找的值更小则深入至左子树进行比较，否则就深入右子树进行比较，直到找到相应的值或者进入了一棵不存在的子树为止。

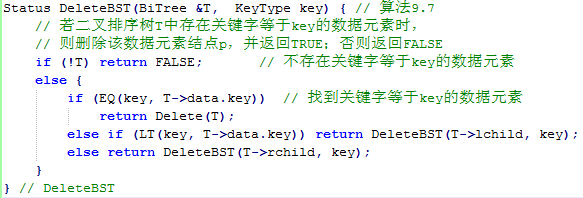
其查找过程可以描述如下：

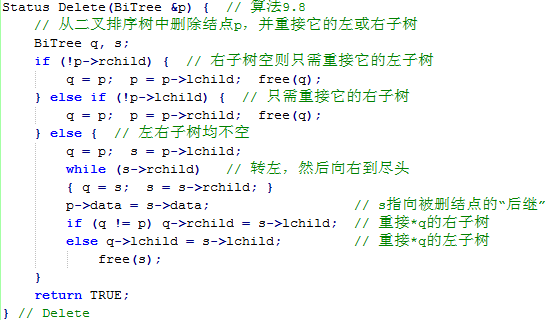


而其插入过程同样也十分简洁，可以描述如下：



而删除操作可以描述为如下的两个算法：





在本题中，读入一串整数，首先利用这些整数构造一棵二叉排序树。另外给定多次查询，利用构造出的二叉排序树，判断每一次查询是否成功。

## 输入格式

输入的第一行包含2个正整数n和k，分别表示共有n个整数和k次查询。其中n不超过500，k同样不超过500。

第二行包含n个用空格隔开的正整数，表示n个整数。

第三行包含k个用空格隔开的正整数，表示k次查询的目标。

## 输出

只有1行，包含k个整数，分别表示每一次的查询结果。如果在查询中找到了对应的整数，则输出1，否则输出0。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

8 3  
1 3 5 7 8 9 10 15  
9 2 5

## 样例输出

1 0 1

## 提示

在本题中，首先需要按照题目描述中的算法完成二叉排序树的构造过程，之后需要通过在二叉排序树中的不断向下查找，将需要查询的值与当前节点的值进行比较，直到确定被查询的值是否存在。

通过课本中的性能分析部分，不难发现二叉排序树的平均查找长度是和logn同数量级的，但是，在某些特殊情况下二叉排序树将会退化，使查找的效率大大降低，这时就需要引入二叉排序树的平衡操作，利用平衡二叉树来保证查找的效率始终维持在logn的数量级上。

-->

# 1773:算法9-9~9-12：平衡二叉树的基本操作

## 题目描述

平衡二叉树又称AVL树，它是一种具有平衡因子的特殊二叉排序树。平衡二叉树或者是一棵空树，或者是具有以下几条性质的二叉树：

1.       若它的左子树不空，则左子树上所有结点的值均小于它的根节点的值；

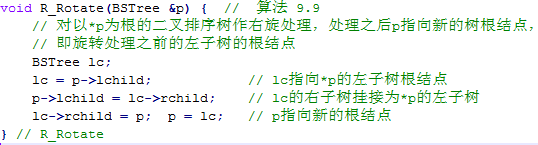
2.       若它的右子树不空，则右子树上所有结点的值均大于它的根节点的值；

3.       它的左右子树也分别为平衡二叉树，且左子树和右子树的深度之差的绝对值不超过1。

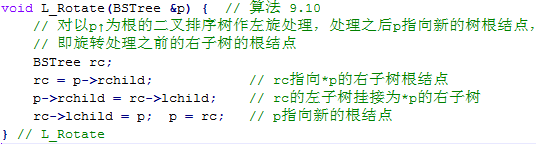
若将二叉树上结点的平衡因子定义为该结点的左子树深度减去它的右子树的深度，则平衡二叉树上的所有结点的平衡因子只可能为-1、0和1。只要二叉树上有一个结点的平衡因子的绝对值大于1，则这棵二叉树就是不平衡的。

通过平衡二叉树的性质不难得知，其插入、删除、查询都操作的时间复杂度均为O(log2n)。

维持平衡二叉树性质的操作可以被称为旋转。其中平衡二叉树的右旋处理可以描述如下：



而其左旋处理与右旋正好相反，可以描述如下：



在本题中，读入一串整数，首先利用这些整数构造一棵平衡二叉树。另外给定多次查询，利用构造出的平衡二叉树，判断每一次查询是否成功。

## 输入格式

输入的第一行包含2个正整数n和k，分别表示共有n个整数和k次查询。其中n不超过500，k同样不超过500。

第二行包含n个用空格隔开的正整数，表示n个整数。

第三行包含k个用空格隔开的正整数，表示k次查询的目标。

## 输出

只有1行，包含k个整数，分别表示每一次的查询结果。如果在查询中找到了对应的整数，则输出1，否则输出0。

请在每个整数后输出一个空格，并请注意行尾输出换行。

## 样例输入

8 3  
1 3 5 7 8 9 10 15  
9 2 5

## 样例输出

1 0 1

提示

在本题中，首先需要按照题目描述中的算法完成平衡二叉树的构造过程，之后需要通过在平衡二叉树中的不断向下查找，将需要查询的值与当前节点的值进行比较，直到确定被查询的值是否存在。

通过课本中的性能分析部分，不难发现平衡二叉树的查找、插入、删除等操作的时间复杂度均为O(log2n)，这是通过利用旋转操作使二叉树保持平衡状态而保证的。

-->

# 2112:查找最小的k个元素

## 题目描述

输入n个正整数（可以有重复的情况出现），输出其中最小的k个（如有重复的情况出现，重复输出），其中1≤k≤n≤10000，每个正整数小于10000。例如输入1、4、8、2、5、7、6、3这8个数字，则最小的4个数字为1、2、3、4。

## 输入格式

第一行，n和k，用空格隔开。

n个整数，用空格隔开。

## 输出

最小的k个正整数，且从小到大排序，用空格隔开。

## 样例输入

8 4  
1 2 3 4 5 6 7 8

## 样例输出

1 2 3 4

## 提示

-->