原书第2版

2010年7月第1版第15次印刷

第二部分 排序和顺序统计学

第6章 堆排序

6.5 优先级队列

When we use a heap to implement a priority queue, therefore, we often need to store a handle to the corresponding application object in each heap element. The exact makeup of the handle (such as a pointer or an integer) depends on the application. Similarly, we need to store a handle to the corresponding heap element in each application object. Here, the handle would typically be an array index. Because heap elements change locations within the array during heap operations, an actual implementation, upon relocating a heap element, would also have to update the array index in the corresponding application object.

第三部分 数据结构

第11章 散列表

11.3 散列函数

11.3.1 除法散列法

可以选作的值常常是与的整数次幂不太接近的质数.

11.3.2 乘法散列法

11.3.3 全域散列

设为有限的一组散列函数，它将给定的关键字域映射到中.这样的一个函数组称为是全域的，如果对每一对不同的关键字，满足的散列函数的个数至多为.换言之，如果从中随机地选择一个散列函数，当关键字时，两者发生碰撞的概率不大于，这也正好是从集合中随机地独立选择和时发生碰撞的概率.

11.4 开放寻址法

第12章 二叉查找树

12.1 二叉查找树

设是二叉查找树中的一个结点.如果是的左子树的一个结点，则. 如果是的右子树的一个结点，则.

定理12.1 如果是一棵包含个结点的子树的根，则调用INORDER-TREE-WALK()过程的时间为.

证明

用表示在一棵包含个结点的子树的根，则调用INORDER-TREE-WALK()过程的时间，则有

设，那么

练习

12.1-2

In a heap, a node’s key is both of its children’s keys. In a binary search tree, a node’s key is its left child’s key, but its right child’s key.

The heap property, unlike the binary-search-tree property, doesn’t help print the nodes in sorted order because it doesn’t tell which subtree of a node contains the element to print before that node. In a heap, the largest element smaller than the node could be in either subree.

Note that it the heap property could be used to print the keys in sorted order in time, we would have an -time algorithm for sorting, because building the heap takes only time.

12.2 查询二叉查找树

定理12.2 对一棵高度为的二叉查找树，动态集合操作SEARCH, MINIMUM, MAXIMUM, SUCCESSOR和PREDECESSOR等的运行时间均为.

12.3 插入和删除

定理12.3 对于高度为的二叉查找树，动态集合操作INSERT和DELETE的运行时间为.

Expected time to build a BST is asymptotically same as quicksort, that is .

练习

12.3-2

Worst case: , occurs when a linear chain of nodes results from the repeated TREE-INSERT operations.

Best case: , occurs when a binary tree of height results from the repeated TREE-INSERT operations.

12.4 随机构造的二叉查找树

二叉查找树的各基本操作的运行时间都是.

The depth of a node is the number of comparisons when it is inserted, so the average node depth is .

定理12.4 一棵在个关键字上随机构造的二叉查找树的期望高度为.

关于average node depth的结论并不能证明定理12.4，考虑一个满二叉树，带有一个长度为的单链，那么平均深度

练习

12.4-3

序列和产生相同的BST.

思考题

12-4 不同二叉树的数目

设表示包含个结点的不同的二叉树的数目，那么，当时

设为生成函数，那么

泰勒展开式

于是

注意斯特林公式

第13章 红黑树

13.1 红黑树的性质

We shall regard these NIL's as being pointers to external nodes (leaves) of the binary search tree and the normal, key-bearing nodes as being internal nodes of the tree(内结点).

一棵二叉查找树如果满足下面的红黑性质，则为一棵红黑树：

1. 每个结点或是红的，或是黑的.
2. 根结点是黑的.
3. 每个叶结点(NIL)是黑的.
4. 如果一个结点是红的，则它的两个儿子都是黑的.If a node is red, its parent is black.
5. 对每个结点，从该结点到其子孙结点的所有路径上包含相同数目的黑结点.

采用一个哨兵代表NIL. 哨兵的域为BLACK，其它域可以设置成任意允许的值.所有指向NIL的指针都被替换成指向哨兵的指针.

采用哨兵代替所有的NIL：所有的叶子以及根部的父结点.

从某个结点出发(不包括该结点)到达一个叶结点的任意一条路径上，黑色结点的个数为该结点的黑高度，记为.红黑树的黑高度定义为根结点的黑高度.

引理13.1 一棵有个内结点的红黑树高度至多为.

证明

先证明以某一结点为根的子树至少包含个内结点.对的高度使用归纳法.若的高度为，那么必为，以为根的子树包含个内结点.若为一高度为正的内结点，并且有两个子女(注意也算).每个儿子的黑高度为或，根据归纳假设，每个儿子为根的子树至少包含个内结点，于是以为根的子树至少包含个内结点.

设树的高度为，根据性质4)，从根结点到叶结点的任意一条简单路径上，至少有一半结点是黑色的，根的黑高度至少为，于是

练习

13.1-5

Every path contains black nodes. The longest path from to a descent leaf has length at least . The longest path contains nodes and at least half the nodes on the path are black, so

13.1-6

根据引理13.1的证明过程，至少有个内结点.

最多的时候，各层红黑间隔，有个内结点.

13.2 旋转

旋转前，旋转后.

练习

13.2-2

若，则无法右旋.若，则无法左旋.可能的旋转数等于边数.

13.2-4

With at most right rotations, we can convert any binary search tree into one that is just a right-going chain.

Let us define the right spine as the root and all descendants of the root that are reachable by following only right pointers from the root. A binary search tree that is just a right-going chain has all nodes in the right spine.

As long as the tree is not just a right spine, repeatedly find some node on the right spine that has a non-leaf left child and then perform a right rotation on .

This rotation increases the number of nodes in the right spine by . Any binary search tree starts out at least one node-the root-in the right spine. At most, rotations are needed to put all nodes in the right spine, so that the tree consists of a single right-going chain.

We could perform this sequence in reverse-turning each right rotation into its inverse left rotation.

至多需要次旋转.

13.2-5

Right-going chain不能通过right rotation转成其它二叉查找树.

设的左子树比的左子树多个结点，先通过次right rotation使得两者左子树结点数目相同，设左右子树分别有个结点，那么

13.3 插入

循环维持下列三个部分的不变式：

在循环的每一次迭代的开头，

1. 结点是红色.
2. 如果是根，则是黑色.
3. 如果红黑树性质被破坏，则至多只有一个被破坏，不是2)就是4).如果违反性质2)，因为是根且是红的.如果违反4)，原因是都是红的.

我们将要证明，循环的第一次迭代之前不变式为真，每次迭代都会使这个循环不变式保持成立，每次迭代会有两种可能的结果：指针沿着树上移，或者执行某些旋转后循环结束.在循环结束时，这个循环不变式会给我们一个有用的性质.

初始化：

从一棵正常的红黑树开始，新增一个红色结点.

a) 是新增的红色结点.

b) 如果是根，则是黑色.

c) 已经看到性质1),3),5)成立.如果性质2)被违反了，红色的根必然是新增的结点.如果性质4)被违反了，的子女是黑色的哨兵，并且新增之前没有违反，所以必然是因为都是红色.

保持：

根据循环不变式的b)部分，如果是根部，则是黑色.而是红色的时候，才进入一次的循环迭代，所以不是根，存在且是黑色的.

循环中需要考虑六种情况，其中三种与另外三种是完全对称的，视是的左孩子还是右孩子而定，只讨论是的左孩子情况，记的叔叔为.

情况1)：的叔叔是红色的.

是红色的，是黑色的.将改为黑色，改为红色以满足性质5).然后将作为新的，记为，开始下一次迭代.

来证明下一次的迭代的开头会保持这个循环不变式.

a) 是红色的.

b) 这个结点颜色不变.如果是根，一直都是黑的.

c) 情况1)不违反性质1),3)，且保持性质5).

情况2)：的叔叔是黑色的，是右孩子.

情况3)：的叔叔是黑色的，是左孩子.

情况2)中，改为指向左旋就转换为情况3)，且并不破坏性质5)，并且的地位保持不变.

情况3)中，改为黑色，改为红色并进行一次右旋，保持性质5).

来证明情况2)和情况3)保持了这个循环不变式.

a) 情况2)中指向红色的，情况2) 3)中的颜色不再改变.

b) 情况3)把改为黑色.如果在下一次迭代的开始是根，则是黑色的.

c) 性质1),3),5)得以保持.情况2) 3)修正了对性质4)的违反.

终止：

循环结束是因为是黑色的.如果是根，则是黑色的哨兵.循环结束时没有破坏性质4)，根据循环不变式，唯一可能会不成立的是性质2)，最后强制使得根结点为黑色保持住了性质2).故结束的时候红黑树的性质都成立.

练习

13.3-5

考虑最后一个插入的结点.

如果是黑色的，因为，不是根，那么的颜色不需要改变.

如果是红色的，跟踪fix的算法过程知道仍然会有红色结点.

13.3-6

用链表追踪记录插入时根到的的路径，则链表头，再次之.

13.4 删除

如果是红色的，被删除后，红黑树的性质仍得以保持，因为：

树中各结点的黑高没有变化.

不存在两个相邻的红结点.

不可能是根，根仍是黑的.

传给RB-DELETE-FIXUP的结点是两个结点中的一个：如果被删除前有一个不是的孩子，那么为唯一的孩子；如果没有孩子，就是哨兵.无论是哪种，的父结点都为先前的父结点.

如果被删除的结点是黑色的，会产生三个问题.首先，如果原来是根结点，而的一个红色的孩子成为根，则违反了性质2).其次，如果和(现在也是)都是红的，就违反了性质4).第三，删除将导致先前包含的任何路径上黑结点数少，性质5)被的一个祖先破坏了.

对于第一个问题，如果原来是根结点，而的一个红色的孩子成为根. RB-DELETE-FIXUP直接将置为黑色.性质2)仍然得以满足.

对于第二个问题，如果和(现在也是)都是红的，就违反了性质4). RB-DELETE-FIXUP直接将置为黑色.性质4)仍然得以满足.

对于第三个问题，一个办法就是把结点视为还有额外的一重黑色，当黑色结点删除时，将其黑色“下推”至子结点，结点是双重黑色或红黑，这样任意包含的路径上黑结点个数加，性质5)得以保持，但违反了性质1).属性仍然是RED(如果是红黑的)或BLACK(如果是双重黑的).一个结点的额外颜色反映指向它，而不是它的属性.

循环开始之初，如果是根结点，或者是红黑的，则不进入循环，直接把的颜色设为黑色，红黑树的所有性质都得到满足.

在循环中，总是指向具有双重黑色的非根结点，只讨论是的左孩子的情况，记的右孩子为.因为是双重黑色的，所以不可能是.

情况1)：的兄弟是红色的.

是红色的，所以孩子都是黑色的，而且不可能是.改为红色，改为黑色，再对进行一次左旋，并没有新破坏红黑树的性质.的新兄弟是旋转之前的一个黑色孩子，转化为情况2),3),4).

情况2)：的兄弟是黑色的，而且的两个孩子都是黑色的.

和去掉一重黑色，只有一重黑色而为红色，为了补偿，在之中增加一重新的黑色.令为新的重新开始循环，如果之前的是红的也就是新的是红黑的，例如从情况1)进入情况2)，则循环结束，直接把的颜色设为黑色，红黑树的所有性质都得到满足.

情况3)：的兄弟是黑色的，而且的左孩子是红色的，右孩子是黑色的.

设为红色，的左孩子为黑色，对进行一次右旋，并没有新破坏红黑树的性质.的新兄弟是旋转之前的红色左孩子，现在是一个有红色右孩子的黑结点，转化为情况4).

情况4)：的兄弟是黑色的，而且的右孩子是红色的.

将设为的颜色，和的右孩子设为黑色，并且对进行一次左旋.这样去掉了额外的黑色，红黑树的性质得以满足，可以将设为根退出循环.

练习

13.4-6

在case1中，是红色的，那么必然是黑色的.

思考题

13-3 AVL树

a)

在一棵高度为的AVL树中，

可知

若是奇数，那么

若是偶数，那么

可得

b)

注意旋转后树高要调整.

d)

Balance后height和insert前一样.

第14章 数据结构的扩张

第四部分 高级设计和分析技术

第15章 动态规划

Dynamic programming is applicable when the subproblems are not independent, that is, when subproblems share subsubproblems. In this context, a divide-and-conquer algorithm does more work than necessary, repeatedly solving the common subsubproblems. A dynamic-programming algorithm solves every subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time the subsubproblem is encountered.

The development of a dynamic-programming algorithm can be broken into a sequence of four steps.

1) characterize the structure of an optimal solution.

2) recursively define the value of an optimal solution.

3) compute the value of an optimal solution in a bottom-up fashion.

4) construct an optimal solution from computed information.

15.1 装配线调度

每一条装配线有个装配站.装配线的第个装配站表示为，和功能相同但是所需时间不同，上需要的装配时间记为.底盘进入装配线的时间为，装配完离开装配线的时间为.

把已经通过装配站的底盘从装配线移走所花的时间是. The problem is to determine which stations to choose from line 1 and which to choose from line 2 in order to minimize the total time through the factory for one auto.

步骤1：通过工厂最快路线的结构

The fastest way through station is either

the fastest way through station and then directly through station , or

the fastest way through station , a transfer from to , and then directly through station

步骤2：一个递归的解

令表示一个底盘从起点到装配站的最快可能时间，那么底盘通过工厂所有路线的最快时间满足

可得递推公式

用追踪最优解的构造过程.通过装配站的最快路线经过，则记为，否则记为.定义为这样的装配线，其内的装配站被通过整个工厂的最快路线所使用.的值可以帮助找到一个最快的路线.

步骤3：计算最快时间

令为递归算法中引用的次数，那么

根据递归式

练习

15.1-5

表明

表明

于是

15.2 矩阵链乘法

是矩阵，是矩阵，那么是矩阵，计算需要的乘法次数是.

矩阵链乘法问题可表述如下：给定个矩阵构成一个链，矩阵维数为，对乘积以一种最小化标量乘法次数的方式进行全加括号.

在矩阵链乘法问题中，并没有把矩阵相乘.目的仅是确定一个具有最小代价的矩阵相乘顺序.

计算全部括号的重数

设表示一串个矩阵可能的全部加括号方案数.当时，.当时，

步骤1：最优加全部括号的结构

对，的任何全部加括号形式都把乘积分为和，其中.加括号的代价就是计算和的代价之和，再加上两者相乘的代价.

假设的一个最优加全部括号把乘积在和之间分开，那么其中的全部加括号必须是的一个最优加全部括号，的全部加括号，必须是的一个最优加全部括号.

步骤2：一个递归解

设为计算所需的标量乘法运算次数的最小值，整个问题最小值就是.

显然有.假设最优加全部括号把乘积在和之间分开，其中.注意是矩阵，是矩阵.所以可以得到

但是实际上可能取，于是得到递归公式

定义为这样一个值，在该处分裂乘积后可得一个最优加全部括号，即等于使得的值.

步骤3：计算最优代价

使用自底向上的表格法来计算最优代价.输入是一个序列.用辅助表保存，辅助表保存计算时取得最优代价的值.

步骤4：构造一个最优解

记录了对乘积在和之间进行分裂以取得最优加全部括号时的的值.

练习

15.2-2

MATRIX-CHAIN-MULTIPLY(A, s, i, j)

if i=j

return A[i];

else if i+1=j

return MATRIX-MULTIPLY(A[i], A[j]);

else

B1 = MATRIX-CHAIN-MULTIPLY(A, s, i, s[i,j]);

B2 = MATRIX-CHAIN-MULTIPLY(A, s, s[i,j]+1, j);

return MATRIX-MULTIPLY(B1, B2);

15.2-4

表示被引用的次数.根据算法

15.2-5

归纳法.

15.3 动态规划基础

最优子结构

如果一个问题的最优解中包含了子问题最优解，则该问题具有最优子结构.

最优子结构在问题域中以两种方式变化：

1) 有多少个子问题被使用在原问题的一个最优解中，以及

2) 在决定一个最优解中使用哪些子问题时有多少个选择.

非正式地，一个动态规划算法的运行时间依赖于两个因素的成绩：子问题的总个数和每一个子问题有多少种选择.

动态规划以自底向上的方式来利用最优子结构.问题的代价通常是子问题的代价加上选择本身的开销.

一些细微之处

已知一个有向图和结点.

无权最短路径：找出一条从到的包含最少边数的路径.这样的一条路径必须是不包含回路的简单路径.

无权最长简单路径：找出一条到的包含最多边数的简单路径.

无权最短路径具有最优子结构.假设，任何从到的路径必然包含一个中间顶点.路径可以分解为子路径.如果是从到的最短路径，那么必定是从到的一条最短路径，必定是从到的一条最短路径.

无权最长简单路径问题没有最优子结构，两个子问题不是独立的.

重叠子问题

当一个递归算法不断地调用同一问题时，我们说该最优问题包含重叠子问题.动态规划算法总是充分利用重叠子问题，每个子问题只解一次，把解保存在一个需要时就可以查看的表中，而每次查表的时间为常数.

令表示调用RECURSIVE-MATRIX-CHAIN来计算个矩阵的链的一个最优加全部括号要花费的时间，那么，当时

做备忘录

加了备忘的递归算法为每一个子问题的解在表中记录一个表项.开始时，每个表项都包含一个特殊的值，以表示该表项有待填入.当在递归算法的执行中第一次遇到一个子问题时，就计算它的解并填入表中.以后每次遇到该子问题时，只要查看并返回表中先前填入的值即可.

练习

15.3-1

对每一个可能的把矩阵链分开的位置，递归算法会寻找左边的最优加全部括号和右边的最优加全部括号，并把这两者组合.

根据第12章思考题12-4，枚举所有可能的复杂度为.

假设递归算法其余步骤用时至多为，那么，当时

设时，有，那么

15.4 最长公共子序列

给定一个序列，另一个序列是的一个子序列，如果存在的一个严格递增下标序列，使得对于所有的有.

给定两个序列和，如果既是的一个子序列又是的一个子序列，那么称是和的公共子序列.

在最长子序列问题中，给定两个序列和，希望找出和的最大长度的公共子序列.

步骤1：描述一个最长公共子序列

给定一个序列，对于，定义的第个前缀，其中是个空序列.

定理15.1(LCS的最优子结构) 设和为两个序列，并设为和的任意一个LCS.

1) 如果，那么，而且是和的一个LCS.

2) 如果，那么蕴含是和的一个LCS.

3) 如果，那么蕴含是和的一个LCS.

定理15.1表明两个序列的LCS也包含了两个序列的前缀的一个LCS，这说明LCS问题具有最优子结构性质.

步骤2：一个递归解

定义为序列和的一个LCS的长度，可得递归式

注意LCS会因为问题的条件而排除子问题.

步骤3：计算LCS的长度

把的值填入表中，同时维护表以简化最优解的构造，其中的表项对应于计算时选择的最优子问题的解.

对，仅依赖于是否有，以及项的值，这几个都在之前计算.

步骤4：构造一个LCS

由表用来快速构造和的一个LCS.

If we need to reconstruct the elements of an LCS, the smaller table does not keep enough information to retrace our steps in time.

练习

15.4-3

需要LCS-LENGTH和LOOKUP-LENGTH两个函数，LCS-LENGTH调用LOOKUP-LENGTH

15.4-4

只保存正在计算的一行和前面一行即可，注意到都为，所以只需要的空间.

注意分隔，一行的空间就足够了，递推式中之前的存为一个临时变量.

15.4-5

排序，然后寻找LCS.

15.4-6

设输入序列为，长度为，第个前缀.设的最长递增子序列长度为，在中长度为的递增子序列可能会有多个，记其中最大元素也就是最后一个元素最小的子序列为，其最后一个元素为，那么有.假设有，那么在长度为的递增子序列中选择后个元素构成长度为的递增子序列，其最后一个元素，与是长度为的递增子序列中最后一个元素最小的矛盾.可知有.特别地，是中最小的元素.

对某个，假设其相应的已知，考察加入元素成为后的变化.

如果，那么就是中的最小元素，成为.因为是中的最小元素，所以并不会改变长度的递增子序列.于是此时.

如果.并入会得到一个长的递增子序列.如果中最长递增子序列长度大于，那么其属于的部分长度会大于，与是中最长递增子序列长度相矛盾.于是此时.若所有长度为的递增子序列中最后一个元素最小者不是，那么中必然有长的递增子序列，矛盾.于是此时.对于的情形，，并入会得到长度为的递增子序列，但是之前已经存在，而且，所以这个新序列并不是长度为的递增子序列中最后一个元素最小者.于是此时.

如果.并入会得到一个长、最后一个元素是的递增子序列，而之前中长度为、最后一个元素最小的递增子序列是，其最后一个元素是而，于是中长、最后一个元素最小的递增子序列，其最后一个元素. 对于的情形，，并入会得到长度为的递增子序列，但是之前已经存在，而且，所以这个新序列并不是长度为的递增子序列中最后一个元素最小者.于是此时.对于的情形，中长的最后一个元素最小的递增子序列还是中长的最后一个元素最小的递增子序列，即.

从增大到，就得到了最长递增子序列的长度，以及相应的长度的递增子序列中最后一个元素最小的.

在上面的循环过程中，只要归入某个，它所属递增子序列中的前趋就不会改变：要么没有，要么是一个固定的元素，直到它被新的覆盖为止.

被某个并入成为时其在原输入序列中的下标记为，那么其前趋是，其在中的下标是.之后和它之前的中的值保存在了中，而之后可能会因为新并入的而被覆盖，相应的的值也会被修改.所以用保存，即的前趋在原输入序列的下标，就可以不用跟踪保存所有的而得到最长递增子序列.

15.5 最优二叉查找树

形式地，给定一个由个互异的关键字组成的序列，且关键字有序，我们想从这些关键字中构造一棵二叉查找树.对每个关键字，一次搜索的概率是.某些搜索的值可能不在内，因此有个虚拟键代表不在内的值.具体地，代表所有小于的值，代表所有大于的值，对于，虚拟键代表所有位于和之间的值.每个关键字是一个内部结点，每个虚拟键是一个叶子.对每个虚拟键，一次搜索对应的概率是.每次搜索要么成功，要么失败，因此

假设在一棵给定的二叉查找树内一次搜索的代价为检查结点个数，亦即在内搜索发现的结点深度加上.所以在内搜索一次的期望代价为

对给定的一组概率，我们的目标是构造一个期望搜索代价最小的二叉查找树.把这种树称作最优二叉查找树.

步骤1：一棵最优二叉查找树的结构

一棵二叉查找树的任意一棵子树必定包含连续范围内的关键字，和相应的虚拟键作为叶子.

如果一棵最优二叉查找树有一棵包含关键字的子树，那么这棵子树对关键字和虚拟键的子问题也必定是最优的.

给定关键字，假设是包含这些键的一棵最优子树的根.根的左子树包含关键字和虚拟键，右子树包含关键字和虚拟键.只要检查所有候选根，而且确定所有包含关键字和的最优二叉查找树，就保证可以找到一棵最优的二叉查找树.

如果在一棵包含关键字的子树中，选取作为根，那么左子树包含关键字，实际上没有真实的关键字但是包含虚拟键.如果选取作为根，那么右子树包含关键字，实际上没有真实的关键字但是包含虚拟键.

步骤2：一个递归解

选取子问题域为找一个包含关键字的最优二叉查找树，其中.当时没有真实的关键字只有虚拟键.定义为搜索一棵包含关键字的最优二叉查找树的期望代价.最终目的是计算.

当时没有真实的关键字只有虚拟键，.

当时，需要从中选择一个根，然后用关键字构造一棵最优二叉查找树作为其左子树，用关键字构造一棵最优二叉查找树作为其右子树.定义

注意有

当一棵树成为一个结点的子树时，子树中每个结点的深度增加.那么左子树贡献的搜索代价为

其中是在左子树中的深度，于是

但是实际上可能取，于是得到递归公式

用记录包含关键字的最优二叉查找树的根的下标.

步骤3：计算一棵最优二叉查找树的期望搜索代价

注意，对应只包含虚拟键的子树，所以对应只包含虚拟键的子树，对应只包含虚拟键的子树..因此所有的保存在表中，保存在表中，保存在表中.并且有递推公式

练习

15.5-1

void ConstructOptimalBST(int \*\*root, int size, int i, int j)，分i + 1 < j,i+1=j,i=j三种情况.

15.5-3

仍为.

15.5-4

记包含关键字和相应的虚拟键的最优二叉查找树为，根在序列中的下标为，搜索的期望代价为.假设，那么属于中的左子树，这显然与二叉搜索树的性质相矛盾.于是有.

对某个以及相应的，最里层会循环次，对于一个会循环

故算法的复杂度为.

思考题

第16章 贪心法

贪心算法是使所做的选择看起来是当前最佳的，期望通过所做的局部最优选择来产生一个全局最优解.

16.1 活动选择问题

设有一个需要使用某一资源的个活动组成的集合，该资源一次只能被一个活动占用.每个活动有开始时间和结束时间，且.一旦被选择后，活动就占据时间区间.如果互不重叠，即或，则称和是兼容的.活动选择问题就是要选择出一个由互相兼容的问题组成的最大子集合.

活动选择问题的最优子结构

是中在结束之后开始，在开始之前结束的活动的子集.实际上包含了所有与和兼容的活动，并且与不晚于结束和不早于开始的活动兼容.加入虚拟活动，约定，于是.

假设活动已经按照结束时间的单调递增顺序排序：

那么当时，就有.若有，那么，得矛盾，可知.于是若活动按照结束时间单调递增顺序排序，子问题空间被用来从中选择最大兼容活动子集，其中.Our space of sub-problems is to select a maximum-size subset of mutually compatible activities from , for .

We will sometimes speak of the sets as sub-problems rather than just sets of activities. It will always be clear from the context whether we are referring to as a set of activities or the subproblem whose input is that set.

考虑某个非空子问题，并假设的解包含某活动，于是.用活动生成两个子问题：和.的解是连同活动在内的和的解的并集.因此的解的活动数是和的解活动数之和再加.

假设现在已知的最优解包含活动，则包含在最优解中针对的解和针对的解必定是最优的.一个非空子问题的任意解必然包含了某项活动，而中的任一最优解都包含了子问题实例和的最优解.

一个递归解

设为中最大兼容子集中的活动数..

如果在的最大兼容子集中被使用，则子问题和的最大兼容子集也被使用，.递归定义为

将动态规划解转化为贪心解

定理16.1 对于任意非空子问题，是中具有最早结束时间的活动：

那么

1) 活动在的某最大兼容活动子集中被使用.

2) 子问题为空，所以选择将使子问题为唯一可能非空的子问题.

证明

1)

设是的某最大兼容活动子集，将中的活动按照结束时间的单调递增顺序排序，并设是第一个活动.若则得证.若，那么构造.因为在中是第一个活动而，于是的活动也是互不重叠的.与包含的活动数目相同，因此是包含的的最大兼容活动子集.

2)

若存在使得，那么，有，与是中具有最早结束时间的活动矛盾.

每个子问题都包含了最近结束的活动，而且活动的数目将随子问题的不同而变化.贪心选择使得剩下的、未调度的时间最大化.

练习

16.1-1

运行时间为.

16.1-3

简单的想法是使用GREEDY-ACTIVITY-SELECTOR选出最大兼容子集，再在中再次使用贪心法选出最大兼容子集，不断重复这个过程直至所有的活动得到分配.最坏情况下耗时.

将所有活动的开始时间、结束时间按照单调递增顺序排序并逐个扫描.如果是某个活动的开始时间，则分配给当前可用的教室.如果是某个活动的结束时间，则回收教室置为可用状态.

这个算法保证使用的教室数是最少的.设活动是第一个安排在教室进行的活动.这说明其它的个教室在时刻都有活动正在进行. 所以任何安排方式要保证同时有个活动正在进行，至少需要个教室.

注意：排序时如果时刻相同，代表活动结束的时刻优先.

16.1-4

与已选出的活动兼容的活动中选择生存期最短的方法，考虑例子

只选择出，并不是最优解.

选择开始时间最早且与已选活动兼容的方法，本节的例子只增加，则只选择出，并不是最优解.

16.2 贪心策略的基本内容

More generally, we design greedy algorithms according to the following sequence of steps:

1. Cast the optimization problem as one in which we make a choice and are left with one sub-problem to solve.

2. Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.

3. Demonstrate that, having made the greedy choice, what remains is a sub-problem with the property that if we combine an optimal solution to the sub-problem with the greedy choice we have made, we arrive at an optimal solution to the original problem.

贪心选择性质

一个全局最优解可以通过局部最优(贪心)选择来得到，即当考虑做何选择时，只考虑当前问题的最佳选择而不考虑子问题的结果，然后再解决选择之后所出现的子问题.

最优子结构

要做的是证明将子问题的最优解与所做的贪心选择合并后，可以得到原问题的一个最优解.

贪心法与动态规划

背包问题 有件物品，第件物品价值元重磅，此处和都是整数.只能装下磅的背包，为一整数，每件物品或被带走或者留下.带哪几样总的价值最大.

部分背包问题中，可以带走物品的一部分.

练习

16.2-1,16.2-6

假设将所有物品按照单位重量的价值单调递减排序.子问题是从第件物品开始选择，带走总重量一定总价值最大的物品.那么的最优解必然尽可能多的带走当前单位重量价值最高的第件物品，如若不然，总可以把一部分重量替换为当前单位重量价值最高的第件物品，使得总重量不变的情况下总价值更大.

设所有物品单位重量价值的中位数为，并把物品分为三个集合：

并且记

如果，就递归地解决集合上背包容量为的背包问题.

如果，则分两种情况：

如果，那么就带走集合的所有物品，递归地解决集合上背包容量为的背包问题.

如果，那么就带走集合的所有物品，递归地解决集合上背包容量为的背包问题.

复杂度满足

16.2-2

设在前件物品中选择，总重量不超过的物品的最大价值为.

如果，那么必然无法选择第件物品，有.

如果，那么有两种可能：一种是选择了第件物品，总价值为；一种是没有选择，即.实际的为这两者的最大值.得到了递推公式

16.2-3

假设将所有物品按照价值单调递减排序，因为，所以实际上就是按照单位重量的价值单调递减排序.使用贪心法解决.

16.2-4

设出发地、目的地分别是为第个和第个加油站，途中共有个加油站.子问题为在第个加油站加满油后用最少的加油次数到达第个加油站.若最优解在第个加油站停车加满油，那么其中和一定是和的最优解.

实际上，如果从第个加油站出发行驶了尽可能远的距离到达第个加油站，那么这个方案一定是的最优解，并且这个和合并，一定是原问题的一个最优解.如果选择在第个加油站加油，那么和的停车加油次数都为次，但是的停车加油次数不少于.可知做出贪心选择——行驶尽可能远的距离到达第个加油站，与子问题的最优解合并后得到的最优解.

16.2-5

设点集按照单调递增的顺序排列. 子问题为覆盖的最小的单位闭区间集合，若其最优解包含不止一个单位闭区间，那么总有一部分单位闭区间覆盖，另一部分覆盖，这两部分闭区间一定是和的最优解.

将中的所有区间按照左端点单调递增排序.考虑区间，这个区间覆盖了，一定是的一个最优解，并且和合并，一定是原问题的一个最优解.

16.2-7

设，那么

16.3 赫夫曼编码

可变长编码对频度高的字符赋以短码，对频度低的字符则赋以较长一些的编码.

前缀编码.

没有一个编码是另一个编码的前缀，这样的编码称为前缀编码.可以证明，由字符编码技术所获得的最优数据压缩总可用某种前缀编码来获得，因此将注意力集中到前缀编码上并不失一般性.

解码过程有一种关于前缀编码的表示方法就是叶子为给定字符的二叉树，在这种树中，一个字符的编码解释为从根到该字符的路径，其中表示转向左子结点，表示转向右子结点.

A binary tree whose leaves are the given characters provides one such representation. We interpret the binary code-word for a character as the path from the root to that character.

文件的一种最优编码总是由一棵满二叉树表示的，树中每个非叶结点都有两个子结点.如果是包含待编码字符的字母表且所有字符频度为正，则表示最优前缀编码的树中恰有个叶子，和个内结点.

给定对应一种前缀编码的二叉树，对字母表中的一个字符，表示在文件中出现的频度，表示的叶子在树中的深度，也就是字符的编码的长度，编码一个文件需要的位数是

定义为树的代价.

构造赫夫曼编码

设是一个包含个字符的集合，且每个字符是出现频率为的对象.是一个以为关键字的最小优先级队列，用来识别要合并的两个频度最低的对象.两个对象合并的结果是一个新对象，其频度为被合并的两个对象的频度之和.

赫夫曼算法的正确性

引理16.2 设是一个字母表，其中每个字符具有频度.设和为中具有最低频度的两个字符，则存在的一种最优前缀编码，其中和的编码长度相同但最后一位不同.

证明

设表示任意一种最优前缀编码，和为中具有最大深度的兄弟叶结点且.不妨设.因为和具有最低频度，所以.

交换和在树中的位置得到树，交换和在树中的位置得到树那么

同理

于是.因为是最优的所以，进而，也表示一种最优前缀编码，且其中和为具有最大深度的兄弟叶结点，即编码长度相同但最后一位不同.

The total cost of the tree constructed is the sum of the costs of its mergers.

引理16.3 设是一个字母表，其中每个字符具有频度.设和为中具有最低频度的两个字符.设为字母表移去和，再合并新字符的字母表，即；如一样为定义，其中.设为表示字母表上最优前缀编码的任意一棵树.那么将中的叶子结点替换成具有和孩子的内部结点所得到的树，then the tree , obtained from by replacing the leaf node for with an internal node having and as children, 表示字母表上的一个最优前缀编码.

证明

对有，但有，于是

假设不表示的一个最优前缀编码，根据引理16.2存在的一个最优前缀编码，其中和为具有最大深度的兄弟叶结点，那么有.如果把中的和移去并将其父结点替换为叶子结点得到，那么也代表了的一种前缀编码于是有，和前面计算同理

于是

矛盾.于是表示的一个最优前缀编码.

根据引理16.2和引理16.3可以得到

定理16.4 过程HUFFMAN产生一种最优前缀编码.

练习

16.3-1

假设不是满二叉树，存在一个度为的结点，那么这个结点的子女完全可以被父结点吸收，从而减少编码长度.

16.3-3

The total cost of a tree for a code can also be computed as the sum, over all internal nodes, of the combined frequencies of the two children of the node.

若字符具有频度，那么在合并的过程中，从叶子结点到根一共累加了次.

16.3-4

有.若那么有.若在表示最优前缀编码的一棵树中有，那么

可见交换和会减少总的编码长度，在最优前缀编码中.

16.3-5

, show how to represent any optimal prefix code on using only bits.

对字母表中的字符使用定长编码，每个字母编码长度为.

遍历处理最优前缀编码树时，如果结点是叶结点，这一位bit值为，那么其后的位存储定长编码；如果结点非叶结点，这一位bit值为，然后递归处理其左右子女.最优前缀编码树共有个结点，所以一共需要 bits.

16.3-7

设字符频度，那么有.在构造Huffman树时，首先合并，合并后的频度，成为频度最大的结点.实际上下次合并成的.每合并频度最小的结点，新的结点一定是频度最大的结点.最终每个叶结点到根的距离都接近相等.

16.3-8

Notice that the number of possible source files using bits and compressed files using bits is . Since any compression algorithm must assign each element to a distinct element the algorithm cannot hope to actually compress the source file.

思考题

16-1

假设已经有分钱找硬币问题的一个最优解，其中使用了面值为的硬币，硬币数为，那么其中对应分钱的部分也是子问题的一个最优解.

a)

就是要证明，分钱找硬币问题的一个最优解包含面值不超过的硬币中面值最大的.

若，那么，只能使用分硬币.

若，那么.如果不使用分的话，可以用个分的替代.

若，那么.如果不使用分的话，可以用个分或者个分的替代.

若，那么.如果不使用分的话，只能用更多的小额硬币替代.

Thus, we have shown that there is always an optimal solution that includes the greedy choice, and that we can combine the greedy choice with an optimal solution to the remaining sub-problem to produce an optimal solution to our original problem. Therefore, the greedy algorithm produces an optimal solution.

b)

硬币的单位是.

设分钱找硬币问题的最优解中面额的硬币数量为，，那么当时，有.如若不然，假设有，那么把其中个面额的硬币替换为个面额的硬币，使用的硬币数减少了，与已经是最优解矛盾.

贪心选择会使用面额的硬币.若只使用面额的硬币，总共的金额为

无法满足要求.

c)

硬币集合为，，或硬币集合为，.贪心算法不能产生最优解.

d)

设硬币的面额数为，分钱找硬币问题需要的最少硬币数为，于是有递推公式

16-2

a)

设任务顺序为, 第个任务的结束时间, 结束时间总和为

下面证明当时，结束时间总和最小，即

最大.设有而且, 如果交换, 那么有

不会变得更大.

第五部分 高级数据结构

第18章 B树

许多数据库系统使用B树或B树的变形来储存信息.

第21章 用于不相交集合的数据结构

These applications often need to perform two operations in particular: finding the unique set that contains a given element and uniting two sets.

21.1 不相交集合上的操作

不相交集合数据结构保持一组不相交的动态集合.We identify each set by a representative, which is some member of the set. We care only that if we ask for the representative of a dynamic set twice without modifying the set between the requests, we get the same answer both times.

Letting x denote an object, we wish to support the following operations:

MAKE-SET(x) creates a new set whose only member (and thus representative) is . Since the sets are disjoint, we require that not already be in some other set.

UNION(x, y) unites the dynamic sets that contain and , say and , into a new set that is the union of these two sets. We assume that the two sets are disjoint prior to the operation. The representative of the resulting set is any member of , although many implementations of UNION specifically choose the representative of either or as the new representative. Since we require the sets in the collection to be disjoint, conceptually we destroy sets and , removing them from the collection . In practice, we often absorb the elements of one of the sets into the other set.

FIND-SET(x) returns a pointer to the representative of the (unique) set containing .

We shall analyze the running times of disjoint-set data structures in terms of two parameters: , the number of MAKE-SET operations, and , the total number of MAKE-SET, UNION, and FIND-SET operations.

CONNECTED-COMPONENTS(*G*)

1 **for** each vertex *v* ∈ *V*[*G*]

2 **do** MAKE-SET(*v*)

3 **for** each edge (*u*, *v*) ∈ *E*[*G*]

4 **do if** FIND-SET(*u*) ≠ FIND-SET(*v*)

5 **then** UNION(*u*, *v*)

SAME-COMPONENT(*u*, *v*)

1 **if** FIND-SET(*u*) = FIND-SET(*v*)

2 **then return** TRUE

3 **else return** FALSE

In an actual implementation of this connected-components algorithm, the representations of the graph and the disjoint-set data structure would need to reference each other.

练习

21.1-2

如果在同一个连通分支中，最后必然在同一个集合中.

21.2 不相交集合的链表表示

Each set is represented by its own linked list. The object for each set has attributes head, pointing to the first object in the list, and tail, pointing to the last object. Each object in the list contains a set member, a pointer to the next object in the list, and a pointer back to the set object. Within each linked list, the objects may appear in any order. The representative is the set member in the first object in the list.

To carry out MAKE-SET(x), we create a new linked list whose only object is x. For FIND-SET(x), we just follow the pointer from x back to its set object and then return the member in the object that head points to.

A simple implementation of union

We perform UNION(x, y) by appending y’s list onto the end of x’s list. The representative of x’s list becomes the representative of the resulting set.

A weighted-union heuristic

In the worst case, the above implementation of the UNION procedure requires an average of time per call because we may be appending a longer list onto a shorter list; we must update the pointer to the set object for each member of the longer list. Suppose instead that each list also includes the length of the list (which we can easily maintain) and that we always append the shorter list onto the longer, breaking ties arbitrarily. This is simple weighted-union heuristic.

Theorem 21.1

Using the linked-list representation of disjoint sets and the weighted-union heuristic, a sequence of MAKE-SET, UNION, and FIND-SET operations, of which are MAKE-SET operations, takes time.

Proof

Because each UNION operation unites two disjoint sets, we perform at most UNION operations over all. We now bound the total time taken by these UNION operations.

Consider a particular object . We know that each time ’s pointer was updated, must have started in the smaller set. The first time ’s pointer was updated, therefore, the resulting set must have had at least members. Similarly, the next time ’s pointer was updated, the resulting set must have had at least members. Continuing on, we observe that for any , after ’s pointer has been updated times, the resulting set must have at least members. Since the largest set has at most members, each object’s pointer is updated at most times over all the UNION operations. Thus the total time spent updating object pointers over all UNION operations is .

We must also account for updating the tail pointers and the list lengths, which take only time per UNION operation. The total time spent in all UNION operations is thus . The time for the entire sequence of operations follows easily. Each MAKESET and FIND-SET operation takes time, and there are of them. The total time for the entire sequence is thus .

练习

21.3 不相交集合森林

We represent sets by rooted trees, with each node containing one member and each tree representing one set. In a disjoint set forest, each member points only to its parent. The root of each tree contains the representative and is its own parent.

We perform the three disjoint-set operations as follows. A MAKE-SET operation simply creates a tree with just one node. We perform a FIND-SET operation by following parent pointers until we find the root of the tree. The nodes visited on this simple path toward the root constitute the find path. A UNION operation, causes the root of one tree to point to the root of the other.

Heuristics to improve the running time

The first heuristic, union by rank, is similar to the weighted-union heuristic we used with the linked-list representation. The obvious approach would be to make the root of the tree with fewer nodes point to the root of the tree with more nodes. Rather than explicitly keeping track of the size of the subtree rooted at each node, we shall use an approach that eases the analysis. For each node, we maintain a rank, which is an upper bound on the height of the node. In union by rank, we make the root with smaller rank point to the root with larger rank during a UNION operation.

The second heuristic, path compression, is also quite simple and highly effective. We use it during FIND-SET operations to make each node on the find path point directly to the root. Path compression does not change any ranks.

Pseudocode for disjoint-set forests

The FIND-SET procedure is a two-pass method: as it recurses, it makes one pass up the find path to find the root, and as the recursion unwinds, it makes a second pass back down the find path to update each node to point directly to the root.

第六部分 图算法

引言

符号代表，意为.用表示图的顶点集，表示其边集.

第22章 图的基本算法

搜索一个图是有序地沿着图的边访问所有顶点.图的搜索技术是图算法领域的核心.

22.1 图的表示

图的邻接表表示由一个包含个列表的数组所组成，包含图中所有和顶点相邻的顶点.邻接表需要的存储空间为.

本书中大部分图算法都假定输入的图的存储结构是邻接表形式.

图的邻接矩阵表示法中，假设顶点按照某种方式编号为，那么的邻接矩阵为一个的矩阵，满足

一个图的邻接矩阵表示需要占用的存储空间为.

练习

22.1-5

有向图的平方是图， if and only if for some , both and . That is, contains an edge between and whenever contains a path with exactly two edges between and .

22.1-6

根据题意，如果汇存在，必然唯一.

如果，那么，顶点不是汇，特别是如果，那么顶点有自身环不是汇.所以如果第行有存在，顶点就不是汇.

如果，那么，顶点不是汇.所以如果第列除了以外有

存在，顶点就不是汇.

这个方法可以判定顶点是不是汇.

int i = 0, j = 0;

while (i < m\_matrix.*size*() && j < m\_matrix.*size*())

{

if (m\_matrix[i][j])

++i;

else

++j;

}

当循环退出时，若顶点满足，那么第行有存在，顶点就不是汇.若此时，那么所有的顶点都不是汇.若此时，那么有，所有的列都有存在.若顶点满足，即此时检测过的最大行号，所以第列已发现的不会在位置，所以仅可能是汇.

22.1-7

是一个矩阵，的位置元素为

如果，那么，含义为顶点的入度和出度之和.

如果，若同号代表边同时进入或离开顶点，这是不可能的.如果异号代表边进入顶点离开顶点，或者离开顶点进入顶点.所以含义是之间边数的相反数.

22.2 广度优先搜索

在给定图和一个特定的源顶点的情况下，广度优先搜索系统地探索中的边，以期发现可从到达的所有顶点，并计算到所有这些可达顶点之间的距离(即最少的边数).该搜索算法同时还能生成一棵根为、且包括的可达顶点的广度优先树.对从可达的任意顶点，广度优先树中从到的路径对应于图中从到的一条最短路径，即包含最少边数的路径.

算法首先会发现和距离的所有顶点，然后才会发现和距离的其他顶点.

灰色标记那些已经被发现，但是还没有完全搜索其邻接表的顶点.

BFS(*G*, *s*)

1 **for** each vertex *u* ∈ *V* [*G*] - {*s*}

2 **do** *color*[*u*] ← WHITE

3 *d*[*u*] ← ∞

4 *π*[*u*] ← NIL

5 *color*[*s*] ← GRAY

6 *d*[*s*] ← 0

7 *π*[*s*] ← NIL

8 *Q* ← Ø

9 ENQUEUE(*Q*, *s*)

10 **while** *Q* ≠ Ø

11 **do** *u* ← DEQUEUE(*Q*)

12 **for** each *v* ∈ *Adj*[*u*]

13 **do if** *color*[*v*] = WHITE

14 **then** *color*[*v*] ← GRAY

15 *d*[*v*] ← *d*[*u*] + 1

16 *π*[*v*] ← *u*

17 ENQUEUE(*Q*, *v*)

18 *color*[*u*] ← BLACK

This while loop maintains the following invariant:

At the test in line 10, the queue consists of the set of gray vertices.

分析

在初始化后，再没有任何顶点被置为白色，每个顶点只进出队列一次.只有每个顶点即将出队时，才会扫描其邻接表，因而每个顶点的邻接表至多被扫描一次.

BFS的总运行时间为.

最短路径

定义顶点到之间的最短路径距离为从到的任何路径中最少的边数；如果从到之间没有通路，则.

引理22.1 设是一个有向图或无向图，为的任意一个顶点，则对任意边，有：

证明

如果从顶点可达顶点，则从也可达顶点.从到的最短路径不可能比从到的最短路径加上边更长，因此不等式成立.如果从顶点不可达顶点，则，不等式仍然成立.

引理22.2 设是一个有向图或无向图，并假设算法BFS从中某一给定源顶点开始执行.在执行终止时，对每个顶点，BFS所计算出来的满足.

证明

We use induction on the number of ENQUEUE operations. Our inductive hypothesis is that for all .

归纳的基础是BFS过程中第行顶点被插入队列后的情形.此时，而对，有.

考虑从顶点的邻接表开始搜索，归纳假设蕴含.搜索过程中发现了白色顶点.根据14-17行，被置为灰色所以不会再次插入队列，而且的值不会再被修改，而且根据归纳假设和引理22.1可知

所以对于再一次的ENQUEUE operation命题成立，所以归纳假设成立.

引理22.3 假设过程BFS在图上的执行过程中，队列包含顶点，其中是队列的头，是队列的尾，则，且.

证明

对队列的操作次数使用归纳法.开始时，队列仅包含顶点，引理自然成立.

在归纳步骤中，必须证明队列中去掉和插入一个顶点后，引理仍然成立.

如果队列头被从队列中去掉了.若此时队列为空，引理自然成立.设成为新的队列头，根据归纳假设，于是，余下的不等式不受影响，引理依然成立.

对于顶点插入队列的情况.根据BFS的过程可以知道，之前的队列头从队列中删除，在扫描的邻接表时，插入了顶点，于是新的队头满足，而新的就是.根据归纳假设，根据算法，于是，余下的不等式不受影响，引理性质得以保持.

推论22.4 假设在BFS的执行过程中将顶点和顶点插入了队列，且先于入队.那么当入队时，有.

证明

注意在BFS的算法过程中，每个顶点至多只进入队列一次且被赋予一个有限的值，根据引理22.3立得.

定理22.5(广度优先搜索的正确性) 设是一个有向图或无向图，并假设算法BFS从中某一给定源顶点开始执行.那么，在执行过程中，BFS可以发现源顶点可达的每一个顶点.在运行终止时，对所有的，都有.此外，对任意从可达的顶点，从到的最短路径之一是从到的最短路径再加上边.

证明

假设存在一些顶点，其值不等于从到这个顶点的最短路径距离，是其中最短路径最小的顶点.根据引理22.2，有，其实应为.若顶点是从不可达的，那么矛盾，于是是从可达的.设从到的最短路径上的前趋为顶点，那么有.

因为是值不等于最短路径距离的顶点中最小的，而，所以，于是

考虑BFS算法中，顶点从队列出队的时间，在的邻接表中.若此时是白色的，那么根据算法过程，，矛盾.若此时是黑色的，那么根据算法过程，早于出队，根据推论22.4，，矛盾.若此时是灰色的，那么是在某个早于的出队时被置灰，且，矛盾.

于是对所有的，有.所有从可达的顶点都会被发现，如果没有发现就有无穷大的值.最后注意到如果，那么.于是可以通过取一条从到的最短路径，再加上边的方法，得到一条从到的最短路径.

广度优先树

对于图及给定的源顶点，可以更为形式化地定义其前趋子图，其中

引理22.6 当算法BFS应用于某一个有向图或无向图时，同时要构造出域，使得前趋子图是一棵广度优先树.

证明

在BFS算法中，当且仅当且，即是从可达的时才有，因而是由中从可达的顶点所组成的，所以是连通的.再根据定义和定理B.2，实际上就是一棵树，称为广度优先树.进一步根据定理B.2，对所有的，在中都有唯一的从到的简单路径.通过归纳地应用定理22.5，即中从到的最短路径之一是从到的最短路径再加上边，可知广度优先树中从到的简单路径就是中从到的一条最短路径.

练习

22.2-3

运行时间.

22.2-4

已经证明.

22.2-6

There are two types of professional wrestlers: "good guys" and "bad guys." Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have professional wrestlers and we have a list of pairs of wrestlers for which there are rivalries. Give an -time algorithm that determines whether it is possible to designate some of the wrestlers as good guys and the remainder as bad guys such that each rivalry is between a good guy and a bad guy. If is it possible to perform such a designation, your algorithm should produce it.

用一个个顶点条边的无向图表示职业摔跤手和他们之间的比赛关系，其中每个顶点代表一个职业摔跤手，一条边代表一场比赛.设某个选手为好选手，并且从其对应的顶点开始进行BFS，那么为偶数的顶点代表好选手，为奇数的代表坏选手.如果能访问所有的顶点，那么这样的指定是可行的.

22.2-7

The diameter of a tree is given by

that is, the diameter is the largest of all shortest-path distances in the tree.

将树视为有根树.记结点为根的子树高度为，直径为.那么当是叶结点时.当不是叶结点时

树中任意两个顶点由唯一一条简单路径相连，所以树的直径其实就是任意两个顶点之间简单路径长度的最大值.设为一条直径，那么的度都为.

若顶点在直径上，那么以为起点进行BFS，距最远的顶点必然是之一，如若不然，则可以替换掉中相应的片段，形成一条更长的路径，与是直径矛盾.

若顶点不在直径上，设以为起点进行BFS达到的最远的顶点是，且不是之一.若与有公共点，那么距最远的顶点必然是之一而不是，进而距离最远的顶点也是之一，矛盾，可知与没有公共点.

树是连通的，所以从到之间存在路径，又因为的度为，所以和存在不是的公共点.于是有

而，矛盾，可知必然是之一.

22.2-8

用DFS可解.

22.3 深度优先搜索

In depth-first search, edges are explored out of the most recently discovered vertex that still has unexplored edges leaving it. When all of edges have been explored, the search "backtracks" to explore edges leaving the vertex from which was discovered. This process continues until we have discovered all the vertices that are reachable from the original source vertex. If any undiscovered vertices remain, then one of them is selected as a new source and the search is repeated from that source. This entire process is repeated until all vertices are discovered.

Breadth-first search is usually employed to find shortest-path distances (and the associated predecessor subgraph) from a given source. Depth-first search is often a subroutine in another algorithm, so breadth-first search is limited to only one source whereas depth-first search may search from multiple sources.因而深度优先搜索产生的先辈子图可以由几棵树所组成.先辈子图定义为，其中

深度优先搜索的先辈子图形成了一个由数棵深度优先树所组成的深度优先森林.

开始时，每个顶点均为白色，搜索中被发现时即置为灰色，当其邻接表被完全检索之后被置为黑色.这可以保证每一个顶点在搜索结束时，只存在于一棵深度优先树中.

每个顶点有两个时间戳：当顶点第一次被发现并置为灰色时，记录第一个时间戳，当结束检查的邻接表并置为黑色时，记录下第二个时间戳.

对个顶点中的每一个，都对应一个发现事件和一个完成事件，所以时间戳为到之间的整数.对每一个顶点，有：

顶点在之前为白色，之间为灰色，之后为黑色.

DFS(*G*)

1 **for** each vertex *u* ∈ *V* [*G*]

2 **do** *color*[*u*] ← WHITE

3 *π*[*u*] ← NIL

4 *time* ← 0

5 **for** each vertex *u* ∈ *V* [*G*]

6 **do if** *color*[*u*] = WHITE

7 **then** DFS-VISIT(*u*)

DFS-VISIT(*u*)

1 *color*[*u*] ← GRAY ▹White vertex *u* has just been discovered.

2 *time* ← *time* +1

3 *d*[*u*] *time*

4 **for** each *v* ∈ *Adj*[*u*] ▹Explore edge(*u*, *v*).

5 **do if** *color*[*v*] = WHITE

6 **then** *π*[*v*] ← *u*

7 DFS-VISIT(*v*)

8 *color*[*u*] BLACK ▹ Blacken *u*; it is finished.

9 *f* [*u*] ▹ *time* ← *time* +1

DFS中第行和第行中的循环，不包括调用DFS-VISIT，占用时间为.只有对白色顶点才会调用DFS-VISIT，而且第一步就是把顶点置为灰色，所以对每个顶点，DFS-VISIT仅被调用一次.在对，调用DFS-VISIT时，行中的循环执行了次.因为，故DFS-VISIT中行总代价为.因此DFS运行时间为.

深度优先搜索的性质

定理22.7(括号定理) 在对一个(有向或无向)图的任何深度优先搜索中，对于图中任意两个顶点，下述三个条件中仅有一个成立：

区间和区间是完全不相交的，且在深度优先森林中，都不是对方的后裔.

区间完全包含于区间中，且在深度优先森林中，是的后裔.

区间完全包含于区间中，且在深度优先森林中，是的后裔.

证明

如果，分两种情况.

第一种.发现顶点时顶点还是灰色的，于是是的后裔.搜索返回顶点并完成之前，所有从出发的边都已经被探寻并已完成.于是区间完全包含于区间中.

第二种，有，区间和区间是完全不相交的.当一个顶点是灰色时，另一个顶点都不是被发现状态，都不是对方的后裔.

同理.

推论22.8(后裔区间的嵌套) 在对一个(有向或无向)图中，顶点是的后裔，当且仅当.

定理22.9(白色路径定理) 在一个(有向或无向)图的深度优先森林中，顶点是的后裔，当且仅当在搜索过程中于时刻发现时，可以从顶点出发，经过一条完全由白色顶点组成的路径到达.

证明

假设是的后裔.设是深度优先树中，从到的通路上的任意顶点，是的后裔于是，于是在时刻顶点是白色的.

假设于时刻发现时，从顶点出发的一条完全由白色顶点组成的路径上是第一个不是在深度优先树中后裔的顶点，而是这条通路上的前趋，那么要么是本身，要么是的后裔，有.时刻顶点还是白色的，于是.而顶点的搜索完成时，不可能有白色的邻接顶点，所以，因而.根据定理22.7以及证明过程，区间完全包含于区间中，于是顶点是的后裔.

边的分类

根据图上进行深度优先搜索所产生的深度优先森林，可以把图的边分为四种类型：

1) 树边(tree edge)，深度优先森林中的边.如果顶点是在探寻边时首次被发现的，就是一条树边.

2) 反向边(back edge)是深度优先树中，连接顶点到它的某一祖先顶点的那些边.有向图中可能出现的自环也被认为是反向边.

3) 正向边(forward edge)是指深度优先树中，连接顶点到它的某个后裔的非树边.

4) 交叉边(cross edge)是其他类型的边，存在于同一棵深度优先树中的两个顶点之间，条件是其中一个顶点不是另一个顶点的祖先.交叉边也可以在不同的深度优先树的顶点之间.

对于每条边，当该边被第一次探寻到时，即根据所到达的顶点的颜色，来对该边进行分类：

1) 白色表明它是一条树边.

2) 灰色表明它是一条反向边.

3) 黑色表明它是一条正向边或交叉边.

根据算法过程可以证明情况1). DFS-VISIT的调用栈形成了一条线性的后裔链，探寻总是从深度最深的灰色顶点开始，因而到达的另一个灰色顶点的边所达到的必是它的祖先.，可以证明情况2).情况3)详见练习22.3-4.

在无向图中，和是同一条边，the edge is classified as the first type in the classification list that applies.根据练习22.3-5，等价地，the edge is classified according to whichever of or is encountered first during the execution of the algorithm.

定理22.10 在对一个无向图进行深度优先搜索的过程中，的每一条边要么是树边，要么是反向边.

证明

设为中一条边，不失一般性，假设.那么时刻发现顶点时顶点还是白色的，注意在的邻接表中，故必定在完成之前(此时为灰色)，已经被发现并且完成了.如果边是沿着从到的方向被探寻到的，那么在该时刻之前顶点都是白色的.于是成为了一条树边.如果边是沿着从到的方向被探寻到的，那么在该边第一次被探寻时，顶点是灰色的，于是是一条反向边.

练习

22.3-4

a)

根据推论22.8，当且仅当时顶点是的后裔，即边是一条树边或正向边.

b)

假设有.若是同一顶点，那么有，是自环，因而是一条反向边. 若不是同一顶点，那么有，根据定理22.7，顶点是的后裔，是一条反向边.

假设是一条反向边.若是自环，有.若是的后裔，根据推论22.8有.

c)

假设有，根据定理22.7，都不是对方的后裔，是一条交叉边.

假设是一条交叉边，根据定理22.7，区间和区间是完全不相交的，进而有或.若，那么，当时刻发现顶点时，顶点是白色的，根据定理22.9，从顶点出发，经过一条完全由白色顶点组成的路径到达，实际上就是边，可知是的后裔，与是交叉边矛盾.只有成立.

22.3-11

All we need to show is that the vertices visited by each call to DFS-VISIT from DFS are exactly the vertices in one connected component of .

All vertices in a connected component are visited by one call to DFS-VISIT from DFS.

设为连通分支中第一个被DFS-VISIT访问的顶点，那么当时刻发现顶点时，中其它顶点都是白色的，根据定理22.9，中其它顶点都是深度优先森林中的后裔，这意味着其它顶点都在对的DFS-VISIT完成返回DFS之前被访问.

All vertices visited by one call to DFS-VISIT from DFS are in the same connected component.

因为是通过中的路径访问的，所以这是显然的.

22.3-12

A directed graph is singly connected if implies that contains at most one simple path from to for all vertices .

从每个顶点开始运行DFS.如果有正向边或交叉边，就意味着不是单连通的.

22.4 拓扑排序

A topological sort of a dag is a linear ordering of all its vertices such that if contains an edge , then appears before in the ordering. A topological sort of a graph can be viewed as an ordering of its vertices along a horizontal line so that all directed edges go from left to right.

TOPOLOGICAL-SORT(*G*)

1 call DFS(*G*) to compute finishing times *f*[*v*] for each vertex *v*

2 as each vertex is finished, insert it onto the front of a linked list

3 **return** the linked list of vertices

经拓扑排序的顶点以与完成时刻相反的顺序出现.

引理22.11 一个有向图是无回路的，当且仅当对进行深度优先搜索时没有得到反向边.

证明

假设存在反向边，那么在深度优先森林中，顶点是顶点的祖先，存在从到的通路，加上边构成了回路，矛盾.

假设有向图是有回路.设顶点是中第一个被发现的顶点，且是中通向的边.那么在发现顶点时，中其它顶点都是白色的，存在从到的白色路径，于是在深度优先森林中，是的后裔，是反向边.

定理22.12 TOPOLOGICAL-SORT(*G*)算法产生一个有向无回路图的拓扑排序.

证明

假设对某一已知的有向无回路图运行DFS，以便确定顶点的完成时刻.只要证明对任一对不同的顶点，若中存在从到的边，则.考虑DFS过程探索到边，若此时顶点为灰色，那么是反向边，与引理22.11矛盾.若顶点为白色，那么是的后裔，有.若顶点为黑色，那么此时已经完成了对顶点的探索，有.所以对有向无回路图中的任意边有.

练习

22.4-1

p n o s m r y v x w z u q t

22.4-2

用数组标记从顶点出发到顶点的通路的个数，那么有.

22.4-3

An undirected graph is acyclic (i.e., a forest) if and only if a DFS yields no back edges.

If there is a back edge, there is a cycle.

如果没有反向边，根据定理22.10，每一条边都是树边，因而没有回路.

对无回路图(a forest)有，所以有条不同的边时就可以确信存在回路.

22.4-4

"bad" edges are inconsistent with the ordering produced.

22.4-5

As we process each vertex from the queue, we effectively remove its outgoing edges from the graph by decrementing the in-degree of each vertex one of those edges enters, and we enqueue any vertex whose in-degree goes to . There is no need to actually remove the edges from the adjacency list, because that adjacency list will never be processed again by the algorithm: Each vertex is enqueued/dequeued at most once because it is enqueued only if it starts out with in-degree or if its in-degree becomes after being decremented (and never incremented) some number of times.

The algorithm outputs vertices in the right order ( before for every edge ) because will not be output until its in-degree becomes , which happens only when every edge leading into has been removed due to the processing (including output) of .

If there are no cycles, all vertices are output.

If there are cycles, not all vertices will be output, because some in-degrees never become .

22.5 强连通分支

有向图的一个强连通分支就是一个最大的顶点集合，对于中的每一对顶点，有；亦即是互相可达的

，其中.和有着完全相同的强连通分支，亦即，在中互为可达，当且仅当它们在中互为可达.

下列运行时间为的算法可以得出有向图的强连通分支.

STRONGLY-CONNECTED-COMPONENTS (*G*)

1 call DFS (*G*) to compute finishing times *f*[*u*] for each vertex *u*

2 compute *G*T

3 call DFS (*G*T), but in the main loop of DFS, consider the vertices

in order of decreasing *f*[*u*] (as computed in line 1)

4 output the vertices of each tree in the depth-first forest formed in line 3 as a

separate strongly connected component

假设的强连通分支为，顶点集为.对于中每个强连通分支，它都包含一个顶点.如果对某个以及某个，中包含了一条有向边的话，则就有一条边.从另一个方面看，收缩那些其关联顶点都处于的同一强连通分支内的边，即可得到图.

引理22.13 设和是有向图中两个不同的强连通分支，设，并假设中存在一条通路.那么中不可能还同时存在通路.

证明

设中存在通路，那么存在通路，于是和是互相可达的，与和是两个不同的强连通分支矛盾.

STRONGLY-CONNECTED-COMPONENTS中的值始终是指第一次DFS计算出来的发现和完成时间.

如果，定义，亦即分别是中任何顶点最早的发现时间和最晚的完成时间.

引理22.14 设和是有向图中两个不同的强连通分支.假设有一条边，其中，则.

证明

在DFS中，和中顶点最早的发现时间分别为.

如果.设为中第一个被发现的顶点，那么时刻中其它顶点和中全部顶点都是白色.在中有从出发的到中每一个顶点的、仅由白色顶点组成的通路，并且对任意顶点，有从出发的到的仅由白色顶点组成的通路，根据白色路径定理可知，在深度优先树中，中其它顶点和中全部顶点都是的后裔.根据推论22.8，有且.

如果.设为中第一个被发现的顶点，那么时刻中其它顶点都是白色.在中有从出发的到中每一个顶点的、仅由白色顶点组成的通路，根据白色路径定理，在深度优先树中，中其它顶点都是的后裔，于是.根据引理22.13，不存在从出发达到中顶点的路径，中没有从可达的顶点，于是时刻中的顶点仍是白色的.对任意有进而.

推论22.15 设和是有向图中两个不同的强连通分支.假设有一条边，其中，则.

证明

因为，所以，根据引理22.14有.

和有着完全相同的强连通分支，所以和也是的强连通分支.推论22.15表明，的每一条连接了两个不同的强连通分支的边都是从在第一个DFS中有更早完成时间的分支指向有着较晚完成时间的分支.

第二次在上执行的DFS是按照的降序进行的，也就是说如果从顶点开始，那么是最大的.如果属于强连通分支那么，而且是所有强连通分支中最大的.根据推论22.15，在中没有从到其它强连通分支的边，从开始的搜索不会访问其他分支中的顶点，于是根为的树仅包含了中的顶点.访问完中的顶点后，搜索从另外一个强连通分支中的顶点开始，的完成时间在所有除了以外的强连通分支中是最大的，根据推论22.15，在中从仅有的到其他分支的边必定是指向的，而中的顶点都已经被访问过.一般来说，在对进行DFS时，如果访问某个强连通分支中的顶点，那么从该强连通分支出来的到任何其它强连通分支的边必然是指向那些已经被访问过的.因此每棵深度优先树都恰好是一个强连通分支.

定理22.16 过程STRONGLY-CONNECTED-COMPONENTS (*G*)可以正确地计算出有向图的强连通分支.

证明

通过对第行对进行DFS发现的深度优先树的数目进行归纳，可以证明每棵树的顶点形成了一个强连通分支.归纳假设是第行中产生的前棵树都是强连通分支.的情形显然成立.

考虑产生的第棵深度优先树.假设这棵树的根顶点是而且属于强连通分支，根据搜索选择根的方式有，对于除了以外需要进一步访问的强连通分支，有.根据归纳假设，访问到时，中其它顶点都是白色的因而是的深度优先树中的后裔.根据推论22.15，中任何离开的边都是指向已经访问过的强连通分支.于是在对进行DFS的过程中，除了以外的任何强连通分支中的顶点都不会是的后裔，于是中根为的深度优先树中的顶点恰好形成了一个强连通分支.

注意有

练习

22.5-1

不变或减少

22.5-3

如果第二次在上执行的DFS是按照的升序进行的，也就是说如果从顶点开始，那么是最小的.如果属于强连通分支那么，不一定是所有强连通分支中最小的.

22.5-5

Give an -time algorithm to compute the component graph of a directed graph . Make sure that there is at most one edge between two vertices in the component graph your algorithm produces.

表示顶点所在的强连通分支的编号，取值范围为.先遍历所有的边构造集合

22.5-6

Given a directed graph , explain how to create another graph such that (a) has the same strongly connected components as , (b) has the same component graph as , and (c) is as small as possible. Describe a fast algorithm to compute .

The basic idea is to replace the edges within each SCC by one simple, directed cycle and then remove redundant edges between SCC’s. Since there must be at least edges within an SCC that has vertices, a single directed cycle of edges gives the -vertex SCC with the fewest possible edges.

先计算出的强连通分支和分支图.初始为空.如果某一个强连通分支有顶点，给中添加边.对中的边，在对应的强连通分支中任取顶点，在对应的强连通分支中任取顶点，给中添加边.

22.5-7

A directed graph is said to be semiconnected if, for all pairs of vertices , we have or . Give an efficient algorithm to determine whether or not is semiconnected. Prove that your algorithm is correct, and analyze its running time.

先计算出的强连通分支和分支图，设有个顶点，对进行拓扑排序得到顶点序列，然后验证是否构成中的一个线性链，即分支图中是否存在边.

根据强连通分支的定义，图是半连通的，当且仅当其分支图是半连通的.只需证明，分支图是半连通的，当且仅当存在其顶点的一个线性链.

显然，如果中存在顶点的线性链，那么必然是拓扑排序得到的结果.

如果分支图中存在顶点的线性链，那么是半连通的.

假设中不存在顶点的线性链，那么在拓扑排序返回的顶点序列中，至少存在两个相邻的顶点，没有相应的边.所有从出发的边的目标顶点都有，因此不存在从到的路径，根据拓扑排序的定义，也不存在从到的路径，因而不是半连通的.

思考题

22-3

An Euler tour of a connected, directed graph is a cycle that traverses each edge of exactly once, although it may visit a vertex more than once.

a)

An Euler tour is a single cycle that traverses each edge of exactly once, but it might not be a simple cycle. An Euler tour can be decomposed into a set of edge-disjoint simple cycles, however.

如果图存在欧拉回路，那么在组成欧拉回路的每个简单回路中，每个顶点的入度和出度都相等，进而在图中每个顶点的入度和出度都相等.

设对每个顶点都有，证明存在欧拉回路.

首先证明若顶点满足，那么可以构造包含顶点的回路.

可以沿从出发的边到达顶点，因为，所以可以沿从顶点出发的边到达下一个顶点.实际上这个步骤可以不断进行下去，对于以外的顶点，可以沿一条边到达并且沿另一条之前未访问过的边离开.因为图中的顶点和边数都是有限的，所以必然会回到停下来，得到了包含顶点的回路.

非构造证明方法：

设是边数大于等于、每个顶点的入度和出度相等，但是不存在欧拉回路的图中边数最少的.如前所述中必然包含回路，设是其中边数最多的回路，是中顶点的集合.若为空，那么，中的边构成回路，即就是欧拉回路，矛盾，所以非空.图中每个顶点的入度和出度仍然相等.对中的顶点入度等于出度的任意分支，根据假设有欧拉回路.如果和交集为空，那么根据图的连通性，存在一条从上的顶点到上顶点，且中间的边不属于的路径，注意每个顶点的入度等于出度，所以反过来也存在一条从到的路径，可以形成边数更多的回路，矛盾.若，那么并上构成一条边数更多的回路，矛盾.于是存在欧拉回路，即为其边数最多的回路.

构造证明方法：

如前所述，可以沿从出发的任意边到达顶点，因为，所以可以沿从顶点出发的边到达下一个顶点.实际上这个步骤可以不断进行下去，对于以外的顶点，可以沿一条边到达并且沿另一条之前未访问过的边离开.因为图中的顶点和边数都是有限的，所以必然会回到停下来，得到了包含顶点的回路.如果还有未访问过的离开的边，那么继续这一过程，直到所有从出发的边都被访问过为止，得到的.若中还有未访问的边，根据图的连通性，必然有未访问过的边离开之前的回路上的顶点，将分为和，对顶点重复前面的步骤得到回路，将这段插入得到.类似的步骤可以不断进行下去，直到中所有的边都被访问过.

b)

The algorithm is based on the idea in the constructive proof above.

We assume that is represented by adjacency lists, and we work with a copy of the adjacency lists, so that as we visit each edge, we can remove it from its adjacency list. The output of this algorithm is a doubly linked list  of vertices which, read in list order, will give an Euler tour. The algorithm constructs by finding cycles (also represented by doubly linked lists) and splicing them into . By using doubly linked lists for cycles and the Euler tour, splicing a cycle into the Euler tour takes constant time.

We also maintain a singly linked list in which each list element consists of two parts:

1. a vertex , and

2. a pointer to some appearance of in .

第23章 最小生成树

一个无向连通图，希望找出一个无回路的子集，它连接了所有的顶点，且其权值之和

为最小.无回路且连接所有顶点，所以必然是一棵树，称为生成树.把确定树的问题称为最小生成树问题.

23.1 最小生成树的形成

The generic method manages a set of edges , maintaining the following loop invariant:

Prior to each iteration, is a subset of some minimum spanning tree.

At each step, we determine an edge that we can add to without violating this invariant, in the sense that is also a subset of a minimum spanning tree. We call such an edge a safe edge for , since we can add it safely to while maintaining the invariant.

GENERIC-MST(*G*, *w*)

1 *A* ← Ø

2 **while** *A* does not form a spanning tree

3 **do** find an edge (*u*, *v*) that is safe for *A*

4 *A* ← *A* ∪ {(*u*, *v*)}

5 **return** *A*

A cut of an undirected graph is a partition of . We say that an edge crosses the cut if one of its endpoints is in and the other is in . We say that a cut respects a set of edges if no edge in crosses the cut. An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut. More generally, we say that an edge is a light edge satisfying a given property if its weight is the minimum of any edge satisfying the property.

定理23.1 Let be a connected, undirected graph with a real-valued weight function defined on . Let be a subset of that is included in some minimum spanning tree for , let be any cut of that respects , and let be a light edge crossing . Then, edge is safe for .

证明

设是包含的一棵最小生成树.如果包含边，那么证明已完成.假设不包含边.

因为是最小生成树，所以也是树中的顶点，而且中有从到的唯一的通路.因为通过割，即，那么在通路上至少有一条边满足，即通过割.因为割不妨害，故不属于.

根据自由树的性质，中去掉边就不再是连通的，但是再加入边就成为连通的成为一棵新的生成树.

因为是轻边，所以，进而

因为是最小生成树，所以，于是，也是一棵最小生成树.因为所以，所以对是安全的.

推论23.2 Let be a connected, undirected graph with a real-valued weight function defined on . Let be a subset of that is included in some minimum spanning tree for , and let be a connected component (tree) in the forest . If is a light edge connecting to some other component in , then is safe for .

证明

因为割不妨害，是该割的一条轻边，因此对来说是安全的.

练习

23.1-1

设为包含顶点但不包含顶点的集合，于是边是通过割的一条轻边.根据定理23.1，是空集，对来说是安全的，因而属于的某一最小生成树.

23.2 Kruskal算法和Prim算法

In Kruskal’s algorithm, the set is a forest whose vertices are all those of the given graph. The safe edge added to is always a least-weight edge in the graph that connects two distinct components. In Prim’s algorithm, the set forms a single tree. The safe edge added to is always a least-weight edge connecting the tree to a vertex not in the tree.

Kruskal算法

Kruskal’s algorithm finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge of least weight. Let and denote the two trees that are connected by . Since must be a light edge connecting to some other tree, Corollary 23.2 implies that is a safe edge for . Kruskal’s algorithm qualifies as a greedy algorithm because at each step it adds to the forest an edge of least possible weight.

循环的每一步，属于中每一个连通分支的顶点属于同一个并查集，反之亦然.

The running time of Kruskal’s algorithm as .

Prim算法

Prim’s algorithm has the property that the edges in the set always form a single tree. Each step adds to the tree a light edge that connects to an isolated vertex—one on which no edge of is incident. By Corollary 23.2, this rule adds only edges that are safe for ; therefore, when the algorithm terminates, the edges in form a minimum spanning tree. This strategy qualifies as greedy since at each step it adds to the tree an edge that contributes the minimum amount possible to the tree’s weight.

算法的执行过程中，不在树中的所有顶点都放在一个基于域的最小优先级队列中.对每个顶点来说，是所有将与树中某一顶点相连的边中的最小权值；按照约定，如果不存在这样的边则.在算法的执行过程中，集合隐含地满足

当算法终止时，最小优先级队列是空的，而的最小生成树则满足：

If we implement as a binary min-heap, the total time for Prim’s algorithm is .

练习

第24章 单源最短路径

在最短路径问题中，给出的是一个带权的有向图，加权函数为从边到实型权值的映射.路径的权是指其组成边的所有权值之和：

定义从到的最短路径的权为：如果存在一条从到的路径，那么

否则.

从顶点到顶点的最短路径定义为的任何路径.

单源最短路径的变体

在本章中重点讨论单源最短路径问题：已知图，希望找出从某给定源顶点到每个顶点的最短路径.

最短路径的最优子结构

Shortest-paths algorithms typically rely on the property that a shortest path between two vertices contains other shortest paths within it.

引理24.1(最短路径的子路径是最短路径) 对于一给定的带权有向图，所定义的权函数为.设是从到的最短路径.对于任意，其中，设为中从顶点到顶点的子路径.那么是从到的最短路径.

负权值边

如果图存在一条从可达的负权回路，那么最短路径的权的定义就不能成立了.如果从到的某路径中存在一条负权回路，定义.

回路

一条最短路径不会包含正权回路.只要最短路径包含权回路，我们可以不断地将这些回路从路径中移除，直到产生出一个无环的最短路径为止.所以，不失一般性，可以假设找到的最短路径为无环的.

最短路径的表示

最短路径算法设置的属性，以便使源于顶点的前辈链表沿着从到的最短路径的相反方向排列.

为中所有具有非空前趋的顶点集合加上源点

有向边集是由中的顶点的值导出的边集：

设图是带权有向图，其加权函数为，并假定中不包含从源顶点可达的权值为负的回路，那么最短路径是良定义的.以为根的最短路径树是有向子图，其中，那么：

1) 是中从可达的顶点的集合

2) 形成了一棵以为根的有根树

3) 对所有，中从到的唯一简单路径是中从到的最短路径.

最短路径、最短路径树并不是唯一的.

松弛技术

对每个顶点，设置属性，描述从源顶点到的最短路径权值的上界，称为最短路径估计.

INITIALIZE-SINGLE-SOURCE(*G*, *s*)

1 **for** each vertex *v* ∈ *V*[*G*]

2 **do** *d*[*v*] ← ∞

3 π[*v*] ← NIL

4 *d*[*s*] 0

经过初始化后，所有的有，，当时.

在松弛一条边的过程中，要测试是否可以通过，对迄今找到的到的最短路径进行改进；如果可以改进的话，更新和.

RELAX(*u*, *v*, *w*)

1 **if** *d*[*v*] > *d*[*u*] + *w*(*u*, *v*)

2 **then** *d*[*v*] ← *d*[*u*] + *w*(*u*, *v*)

3 π[*v*] ← *u*

对任何实数，有.

对任何实数，有.

24.1 Bellman-Ford算法

The Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source. If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortest paths and their weights.

The algorithm relaxes edges, progressively decreasing an estimated on the weight of a shortest path from the source to each vertex until it achieves the actual shortest-path weight . The algorithm returns TRUE if and only if the graph contains no negative-weight cycles that are reachable from the source.

BELLMAN-FORD(*G*, *w*, *s*)

1 INITIALIZE-SINGLE-SOURCE(*G*, *s*)

2 **for** *i* ← **1 to** |*V*[*G*]| - 1

3 **do for** each edge (*u*, *v*) ∈ *E*[*G*]

4 **do** RELAX(*u*, *v*, *w*)

5 **for** each edge (*u*, *v*) ∈ *E*[*G*]

6 **do if** *d*[*v*] > *d*[*u*] + *w*(*u*, *v*)

7 **then return** FALSE

8 **return** TRUE

The Bellman-Ford algorithm makes passes over the edges of the graph, each pass is one iteration of the for loop of lines and consists of relaxing each edge of the graph once, so runs in time .

引理24.2 设是带权有向图，其权函数为，为源点，并假定中不含可达的负权回路.那么在BELLMAN-FORD第行for循环的次迭代后，对任何可达的顶点，有.

证明

考虑从可达的任意顶点，是任意一条从到的无环最短路径.路径至多有条边，所以. 第行for循环的次迭代，每一次都松弛了所有的条边.边在第次迭代被松弛.根据路径松弛性质，有.

推论24.3 设是带权有向图，其权函数为，为源点.对每一顶点，从到存在一条通路，当且仅当对运行Bellman-Ford算法，算法终止时，有.

定理24.4(Bellman-Ford算法的正确性) 设是带权有向图，其权函数为，为源点，对图运行Bellman-Ford算法.若不包含可达的负权回路，则算法返回TRUE，对所有顶点，有成立.前趋子图是以为根的最短路径树.如果包含从可达的负权回路，则算法返回FALSE.

证明

假设图不包含从可达的负权回路.对于，如果是从可达的，根据引理24.2，算法终止时有.如果是不可达的，根据无路径性质，仍有.前趋子图的性质以及这个结论隐含着是一棵最短路径树.此时对所有的边有

这样算法必然返回TRUE.

假设图包含一个从可达的负权回路，那么

假设算法返回值为TRUE，那么对有，于是

因为所以

矛盾，算法应该返回FALSE.

24.2 有向无回路图中的单源最短路径

The algorithm starts by topologically sorting the dag to impose a linear ordering on the vertices. If the dag contains a path from vertex to vertex , then precedes in the topological sort. We make just one pass over the vertices in the topologically sorted order. As we process each vertex, we relax each edge that leaves the vertex.

DAG-SHORTEST-PATHS(*G*, *w*, *s*)

1 topologically sort the vertices of *G*

2 INITIALIZE-SINGLE-SOURCE(*G*, *s*)

3 **for** each vertex *u*, taken in topologically sorted order

4 **do for** each vertex *v* ∈ *Adj*[*u*]

5 **do** RELAX(*u*, *v*, *w*)

The total running time is , which is linear in the size of an adjacency-list representation of the graph.

定理24.5 设是带权有向图，其权函数为，为源点，并假定中不含回路，则在DAG-SHORTEST-PATHS终止时，对任意顶点，有，且前趋子图是最短路径树.

证明

如果是从不可达的，根据无路径性质有.设是从可达的，存在最短路径.因为我们是依据拓扑排序对顶点进行处理的，所以上被松弛的边顺序为.路径松弛性质表明算法终止时有.根据前趋子图性质，是最短路径树.

24.3 Dijkstra算法

Dijkstra’s algorithm solves the single-source shortest-paths problem on a weighted, directed graph for the case in which all edge weights are nonnegative. In this section, therefore, we assume that for each edge .

Dijkstra’s algorithm maintains a set of vertices whose final shortest-path weights from the source have already been determined. The algorithm repeatedly selects the vertex with the minimum shortest-path estimate, adds to , and relaxes all edges leaving . In the following implementation, we use a min-priority queue of vertices, keyed by their values.

DIJKSTRA(*G*, *w*, *s*)

1 INITIALIZE-SINGLE-SOURCE(*G*, *s*)

2 *S* ← Ø

3 *Q* ← *V*[*G*]

4 **while** *Q* ≠ Ø

5 **do** *u* ← EXTRACT-MIN(*Q*)

6 *S* ← *S* ∪{*u*}

7 **for** each vertex *v* ∈ *Adj*[*u*]

8 **do** RELAX(*u*, *v*, *w*)

定理24.6(Dijkstra算法的正确性) 已知一带权有向图，其权函数的值为非负，源点为.对该图运行Dijkstra算法，则在算法终止时，对所有有.

证明

我们使用如下循环不变式：每次迭代开始时，对每个顶点，有.

初始化：

初始时，，循环不变式显然成立.

保持：

我们希望说明在每一轮迭代中，对加入集合中的顶点，有.

设是加入集合的第一个满足的顶点，考察还未加入的时刻.因为是第一个加入的顶点且，所以，而且此时非空.

若从到不存在路径，那么有，矛盾.所以从到存在路径，进而存在一条最短路径. 此时路径连接着中的一个顶点和中的一个顶点.设为上第一个属于的顶点，为的前趋.

因为是加入集合的第一个满足的顶点，所以有，当加入后边被松弛过，所以.

因为在从到的最短路径上而且在之前，而且边权值非负，所以有，进而

但是根据算法过程，此时应该有，所以只可能是

矛盾.所以当加入集合时有.

终止：

终止时，.对每个顶点，有.

推论24.7 已知一加权函数非负且源点为的带权有向图，若在该图上运行Dijkstra算法，则在算法终止时，前趋子图是以为根的最短路径树.

证明

根据定理24.6和前趋子图性质立得.

24.5 最短路径性质的证明

三角不等式

引理24.10(三角不等式) 设是带权有向图，其权函数为，源点为.那么对于边，有

对最短路径估计的松弛的效果

引理24.11(上界性质) 设是带权有向图，其权函数为.设为源点，INITIALIZE-SINGLE-SOURCE(*G*, *s*)对图进行了初始化.那么，对所有的有，而且这个不变式对图中边的任意系列松弛操作保持不变.更进一步说，一旦到达下界将不再改变.

证明

通过对松弛步骤数目的归纳，证明对所有的，有不变式.

在初始化后，有，如果属于一个负权回路，有，否则，无论哪种都有.而当时有.

考虑松弛边，根据假设在松弛前对所有的，有不变式.松弛边只可能改变，如果变化了，有

不变式仍然保持.

因为有，而且松弛过程不会增加的值，所以一旦，的值将不再改变.

推论24.12(无路径性质) 假设在带权有向图中，其权函数为，从源到给定顶点不存在路径.那么在INITIALIZE-SINGLE-SOURCE(*G*, *s*)对图初始化以后，有，在对于的边进行任意序列的松弛操作后，这个等式作为循环不变式仍然保持.

证明

根据上界性质，有，于是.

引理24.13 设为一个带权有向图，其权函数为，且.那么通过执行RELAX(*u*, *v*, *w*)松弛边后，有.

证明

松弛边只可能改变.如果松弛之前就有，那么不变.如果松弛之前，之后就有.

引理24.14(收敛性质) 设是带权有向图，其权函数为，为源点.对某些顶点，设为图中的最短路径.假定通过调用INITIALIZE-SINGLE-SOURCE(*G*, *s*)进行了初始化，然后在图的边上执行了包括调用RELAX(*u*, *v*, *w*)在内一系列松弛步骤.如果在调用之前，那么在调用之后的任意时间.

证明

如果在松弛边之前就有，那么根据上界性质，这个等式将会继续保持下去.

为图中的最短路径，根据引理24.1，即最短路径的子路径是最短路径，有.

根据引理24.13，松弛边后

而根据上界性质有，于是，而且这个等式之后一直保持.

引理24.15(路径松弛性质) 设是带权有向图，其权函数为，为源点.考虑任意从到的最短路径.如果通过调用INITIALIZE-SINGLE-SOURCE(*G*, *s*)进行了初始化，然后按照顺序进行了一系列的松弛步骤，包括松弛边，那么经过这些松弛后以及在以后的任意时刻，都有.无论其它边是否发生松弛(包括与的边交错地进行的松弛)，这一性质都始终保持.

证明

通过归纳法证明，路径上第条边被松弛以后有.

当，即上任意边被松弛以前，根据初始化，根据上界性质，的值将不再改变.

根据归纳假设有，为图中的最短路径，根据引理24.14，对边进行松弛可以得到，此后这个等式一直成立.

松弛和最短路径树

引理24.16 设是带权有向图，其权函数为，为源点，并假定中不含可达的负权回路.那么在图被INITIALIZE-SINGLE-SOURCE(*G*, *s*)进行了初始化后，前趋子图就构成以为根的有根树，在对边任意序列的松弛操作下仍然像不变式一样保持这个性质.

证明

初始时，中唯一顶点就是，引理显然成立.考虑经过一系列松弛步骤后的前趋子图.

首先注意中所有顶点都是从可达的.中除了以外的顶点都有一个非的前趋，当它被赋予这样一个非值的时候，就被赋予了一个有限的最短路径估计，这意味着它是从可达的.

假设某个松弛操作在中构造出了环，对有.不失一般性，假设对边的松弛形成了这个环.

对，有这意味着对的最后一次更新是，而可能会在之后的松弛过程中变得更小，所以在对边进行松弛之前有：

因为对边的松弛形成了这个环所以此时有：

可得

注意所以

意味着是从可达的负权回路，矛盾，于是是无环的.

因为无环，中的顶点沿着属性回溯必然会回到，即对任意，在中存在从到的路径.

假定从到存在两条简单路径，这意味着，矛盾.中从到仅存唯一的简单路径，是一棵以为根的有根树.

引理24.17(前趋子图性质) 设是带权有向图，其权函数为，为源点，而且假定中不含可达的负权回路.通过调用INITIALIZE-SINGLE-SOURCE(*G*, *s*)进行了初始化，然后在的边上执行一系列松弛操作，得到对所有的，有.因此前趋子图是一个以为根的最短路径树.

证明

只需要证明对，中的唯一简单路径就是中从到的最短路径.设.

对有，于是

因为是从到的任意路径权值的下界，于是只有.

第七部分 算法研究问题选编

第30章 多项式与快速傅里叶变换

A signal is given in the time domain: as a function mapping time to amplitude. Fourier analysis allows us to express the signal as a weighted sum of phase-shifted sinusoids of varying frequencies. The weights and phases associated with the frequencies characterize the signal in the frequency domain.

多项式

任何严格大于一个多项式次数的整数都是这个多项式的次数界.

如果

那么

其中

30.1 多项式的表示

点值表示法

一个次数界为的多项式的点值表示就是个点值对所形成的集合：

其中所有各不相同，且当时.

定理30.2 当输入和输出都采用系数形式来表示多项式时，就能在的时间内，计算出两个次数界为的多项式的积.

第32章 字符串匹配

假设文本是一个长度为的数组，模式是一个长度为的数组.进一步假设和的元素都是属于某个有限字母表表中的字符.字符数组和常称为字符串.

如果，并且，即对有，则说模式在文本出现且位移为.如果在出现且位移为，则称是一个有效位移，否则称为无效位移.

32.3 利用有限自动机进行字符串匹配

有限自动机

一个有限自动机是一个元组，其中：

是状态的有限集合

是初始状态

是一个接收状态集合

是有限的输入字母表

是一个从到的函数，称为的转移函数.

如果有限自动机在状态时读入了输入字符，则它从状态变为状态. Whenever its current state is a member of , the machine is said to have accepted the string read so far. An input that is not accepted is said to be rejected.

第33章 计算几何学

33.3 寻找凸包

The convex hull of a set of points, denoted by , is the smallest convex polygon for which each point in is either on the boundary of or in its interior.

Graham’s scan

GRAHAM-SCAN(*Q*)

1 let *p*0 be the point in *Q* with the minimum *y*-coordinate,

or the leftmost such point in case of a tie

2 let 〈*p*1, *p*2, ..., *pm*〉 be the remaining points in *Q*,

sorted by polar angle in counterclockwise order around *p*0

(if more than one point has the same angle, remove all but

the one that is farthest from *p*0)

3 PUSH(*p*0, *S*)

4 PUSH(*p*1, *S*)

5 PUSH(*p*2, *S*)

6 **for** *i* ← 3 **to** *m*

7 **do while** the angle formed by points NEXT-TO-TOP(*S*), TOP(*S*),

and *pi* makes a nonleft turn

8 **do** POP(*S*)

9 PUSH(*pi, S*)

10 **return** *S*

Jarvis’s march

33.4 寻找最近点对

We now consider the problem of finding the closest pair of points in a set of points. “Closest” refers to the usual euclidean distance: the distance between points and is . Two points in set may be coincident, in which case the distance between them is zero.

分治算法

Each recursive invocation of the algorithm takes as input a subset and arrays and , each of which contains all the points of the input subset . The points in array are sorted so that their -coordinates are monotonically increasing. Similarly, array is sorted by monotonically increasing -coordinate. Note that in order to attain the time bound, we cannot afford to sort in each recursive call. We shall use “presorting” to maintain this sorted property without actually sorting in each recursive call.

A given recursive invocation with inputs and first checks whether . If so, the invocation simply performs the brute-force method described above: try all pairs of points and return the closest pair. If , the recursive invocation carries out the divide-and-conquer paradigm as follows.

Divide: Find a vertical line that bisects the point set into two sets and such that , all points in are on or to the left of line , and all points in are on or to the right of . Divide the array into arrays and , which contain the points of and respectively, sorted by monotonically increasing -coordinate. Similarly, divide the array into arrays and , which contain the points of and respectively, sorted by monotonically increasing -coordinate.

Conquer: Having divided into and , make two recursive calls, one to find the closest pair of points in and the other to find the closest pair of points in . The inputs to the first call are the subset and arrays and ; the second call receives the inputs and . Let the closest-pair distances returned for and be and , respectively, and let .

Combine: The closest pair is either the pair with distance found by one of the recursive calls, or it is a pair of points with one point in and the other in . The algorithm determines whether there is a pair with one point in and the other point in and whose distance is less than . Observe that if a pair of points has distance less than , both points of the pair must be within units of line . Thus they both must reside in the -wide vertical strip centered at line . To find such a pair, if one exists, we do the following:

1. Create an array , which is the array with all points not in the -wide vertical strip removed. The array is sorted by -coordinate, just as is.

2. For each point in the array , try to find points in that are within units of . As we shall see shortly, only the points in that follow need be considered. Compute the distance from to each of these points, and keep track of the closest-pair distance found over all pairs of points in .

3. If , then the vertical strip does indeed contain a closer pair than the recursive calls found. Return this pair and its distance . Otherwise, return the closest pair and its distance found by the recursive calls.

Correctness

We shall now prove that we need only check the points following each point in array .

Suppose that at some level of the recursion, the closest pair of points is and . Thus, the distance between and is strictly less than . Point must be on or to the left of line and less than units away. Similarly, is on or to the right of and less than units away. Moreover, and are within units of each other vertically. Thus, and are within a rectangle centered at line . (There may be other points within this rectangle as well.)

Consider the square forming the left half of this rectangle. Since all points within are at least units apart, at most points can reside within this square. Similarly, at most points in can reside within the square forming the right half of the rectangle.

Let us assume without loss of generality that precedes in array . Then, even if occurs as early as possible in and occurs as late as possible, is in one of the positions following .

第八部分 附录：数学基础知识

B. 集合等离散数学结构

B.4 图

有向图是一对，其中是有穷集，是上的二元关系.集称为的边集合，它的元素称为边.自身环，即从顶点到自身的边，是可能的.

在一个无向图中，边集由无序顶点对而不是有序顶点对组成.在无向图中不允许自身带环，因此每个边都由两个不同的顶点组成.

如果是有向图的一条边，则称离开顶点进入顶点. 如果是无向图的一条边，则称与顶点关联.

如果是图的一条边，则称顶点与顶点相邻.在有向图中，顶点与顶点相邻有时记为.

在无向图中，一个顶点的度是指与之关联的边的条数.如果一个顶点的度为，则称其是孤立的.在有向图中，顶点的出度是以它为起点的边的条数，入度是以它为终点的边的条数，度等于出度与入度之和.

在图中，从顶点到顶点且长度为的路径是顶点序列，且满足且对有.路径的长度就是路径中边的条数.路径包含顶点和边.如果存在一条从到的路径，则称是从经由可达的.如果是有向的，可以记为.如果路径上各顶点均不重复，则称这样的路径为简单路径.

路径的子路径是它的顶点的一个连续子序列.

在有向图中，如果且路径至少包含一条边，则称路径形成回路.如果各不相同，则称回路为简单回路.自身环是长度为的回路.如果存在整数使得对有，则路径与路径形成相同的回路.一个不存在自身环的有向图称为简单图.在无向图中，如果且各不相同，则路径形成简单回路.不存在回路的图是无回路图.

如果无向图的每对顶点都有路径相连，则称其为连通图.在“可达”关系下，顶点的等价类称为图的连通分支.如果无向图仅有一个连通分支，即每个顶点都是其它顶点可达的，则称其为连通的.

如果有向图中每对顶点都相互可达，则称其为强连通图.在相互可达关系下顶点的等价类称为有向图的强连通分支.如果有向图仅有一个强连通分支，则称其是强连通的.

如果存在双射，使得当且仅当时，则称图与图同构.即可以在保持和中相应边的情况下，将中的顶点重新标注成中的顶点.

如果，则称图是图的子图.给定集合，中关于的子图是图其中.

给定一个无向图，它的有向版本是有向图，其中当且仅当.即图中每条无向边在有向图中被两条有向边和代替.

给定一个有向图，它的无向版本是无向图，其中当且仅当且.即无向图包含中去掉方向的边，并且去除了自身环.

在有向图中，顶点的邻居是的无向版本中与邻接的顶点.在无向图中如果和邻接，则它们是邻居.

每对顶点都邻接的无向图称为完全图.如果无向图的顶点可以划分为两个集合，使得对有，则称为二分图.

无回路的无向图称为森林，连通的、无回路的无向图是自由树，称为dag.

沿边收缩无向图得到图，其中，是新顶点.边集通过从中删除边，对每个射到的顶点删除，添加新边得到.

练习

B.4-1

握手引理：如果是无向图，那么.

B.4-3

对顶点数使用归纳法.

B.5 树

B.5.1 自由树

自由树是一个连通的、无回路的无向图.如果一个无向图是无回路但可能是非连通的，称为森林.

定理B.2(自由树的性质)

令为一个无向图.下面的表述是等价的.

1) 是自由树.

2) 中任意两个顶点由唯一一条简单路径相连.

3) 是连通的，但是从中去掉任何边后得到的图都是非连通的.

4) 是连通的，且.

5) 是无回路的，且.

6) 是无回路的，但是添加任何边到后得到的图包含回路.

证明

1)证2)

是自由树是连通的，所以任意两个顶点之间由至少一条简单路径相连.令为两个顶点，它们由不同的简单路径相连.令为两条路径首次分叉的顶点，为两条路径首次交汇的顶点.在上的后继顶点为，在上的后继顶点为且.是从经过到的子路径，是从经过到的子路径.连接和反向得到回路，和是树无回路矛盾.因此如果是树，那么中任意两个顶点由唯一一条简单路径相连.

2)证3)

令为中的边，是从到的路径且是唯一路径.将从中去掉就不存在从到的路径，成为非连通的.

3)证4)

根据练习B.4-3，有.需要通过对顶点数使用归纳法证明.若顶点数，命题显然成立.假设并且当顶点数小于时命题成立.因为是连通的，但是从中去掉任何边后得到的图都是非连通的，所以从去掉任意一条边得到个连通子图，每个子图都满足条件3)，所以每个子图都有，累加起来得到，而，于是.

4)证5)

假设是连通的，且，证是无回路的.假设有一个包含个顶点的简单回路.令为这个回路构成的的子图.如果，因为是连通的，所以在中必然存在某个顶点与中某个邻接.定义子图，其中，并且.如果，则继续进行这一步骤，直到得到，其中，然而是的子集应该有，矛盾.

5)证6)

假设是无回路的，且.若有个连通分支，每个连通分支都是一个自由树，根据1)可得5)，因而满足，累加起来得到，于是只有.所以只有个连通分支，就是一个树.根据1)可得2)，任意两个顶点由唯一一条简单路径相连，那么添加任何边到都会形成回路.

6)证1)

假设是无回路的，但是添加任何边到后得到的图包含回路，证是连通的.假设为中不邻接的任意两个顶点，添加边得到回路，回路中除了以外的边都在中，表明在中之间存在路径，是连通的.

B.5.2 有根树和有序树

有根数有一个特殊的顶点称为根.有根数的顶点通常称为结点.

没有子女的结点称为外部结点或叶结点，非叶结点称为内部结点.

有根数中结点的子女数目称为的度.从根到结点的路径长度称为在中的深度.结点在树中的高度是从结点向下到某个叶结点最长简单路径中边的条数.树的高度等于根的高度，也等于树中结点的最大深度.

有序树是子女结点有序的有根数.

B.5.3 二叉树与位置树

满二叉树：每个结点或者是叶结点，或者度数为，不存在度为的结点.

在位置树中，结点的子女用不同的正整数标识.如果没有结点被标识成整数，则结点的第个子女缺失.叉树是每个结点的标识超过的子女均缺失的位置树.

完全叉树所有叶结点都有相同深度，并且所有内部结点度都为.高度为的完全叉树的内部结点个数为

练习

B.5-2

无向版本中，任意两个顶点由唯一一条简单路径相连.

B.5-3

若非根结点度为，作为图的顶点度为；非根结点度为，作为图的顶点度为；根结点度为，作为图的顶点度为；根结点度为，作为图的顶点度为；叶结点度为，作为图的顶点度为.

若度为结点数为，度为结点数为，叶结点数为.那么不管根结点度为多少中都多算了一次，根据握手引理可得

即

同时有

可得