原书第2版

2010年7月第1版第15次印刷

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驱就不会改变：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，那么其前驱是，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，即的前驱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

习题

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

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

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.