1.1匹配问题

对于一个实例可能有多于一个的稳定匹配。

稳定匹配:目前匹配(集合间元素匹配)不会再变了

G-S算法解决匹配问题(重复迭代)

1.2 五个典型问题

算法设计过程步骤：

1.充分数学化的精确性简洁地表达问题

2.针对该问题设计一个算法

3.通过证明它是正确的并且给出一个运行时间的界以确定算法的效率来分析算法

区间调度(解决方法:贪心)

带权重的区间调度(解决方法:动态规划)

二分匹配问题：给定一个任意的二部图G，求一个最大的匹配。(解决方法:增广)

独立集问题:给定图G=(V,E),找一个最大的独立集S，S(独立集：集合中的结点没有相连的)。

独立集问题可以对下述任何情况进行描述:你尝试从一组个体进行选择，其中某些个体之间存在着两两冲突。

竞争的便利店选址问题

二 算法分析基础

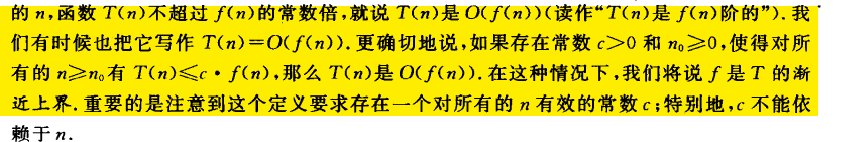
效率定义(1): 当实现一个算法时，如果它在真实的输入实例上运行得快，那么这个算法是有效的

效率定义(2): 在分析的层次上，如果一个算法与蛮力搜索(遍历)比较，最坏情况下达到质量上更好的性能，就说它是有效的。

效率定义(3):如果一个算法有多项式运行时间(O(n))，它就是有效的。

（上面三条定义不重要）

渐进的上界(大O)（函数的运行时间不会超出该数量级）



O(.)仅表示一个上界，不是函数准确的增长率。

渐进的下界（函数的运行时间不会低于该数量级）

存在常数c>0.n0>=0.使得对于所有n>n0,有T(n)>=cf(n).我们就说T(n)=(f(n))

渐进的紧的界

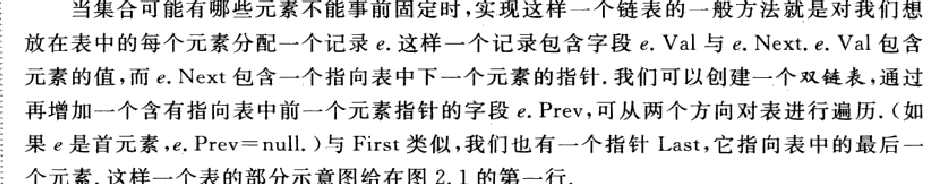
如果函数T(n)既是O(f(n))也是(f(n)),我们就说T(n)是(f(n))

对每个b>1和每个x>0,lo=0()(对数是增长的非常慢的函数)

(不管底是什么，只相差一个常数倍，因此底是a，还是b根本不重要)

2.3 用表和数组实现稳定匹配算法

2.3.1 数组和表



归并排序算法：将输入数的集合划分成相等的两块，递归地对每一半进行排序（O(nlogn)）

一个集合有n个元素，那么它的子集数为2^n

2.4 一般运行时间的概述

2.4.1 线性时间

计算最大数

归并两个排好序的数表

2.4.2 O(nlogn)时间

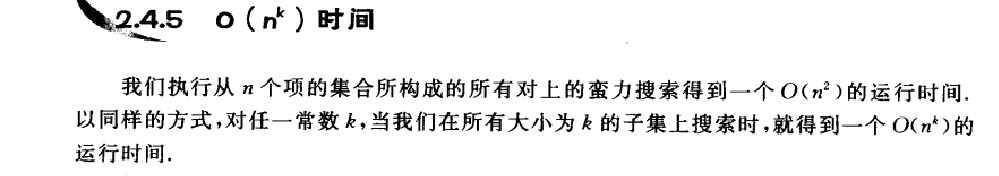
典型的例子：归并排序

2.4.3 平方时间:

假设平面上给定n个点，每个点由(x,y)坐标指定，找出最接近的点对(第五章有一算法，仅用O(nlogn)时间来找平面上的最邻近点对

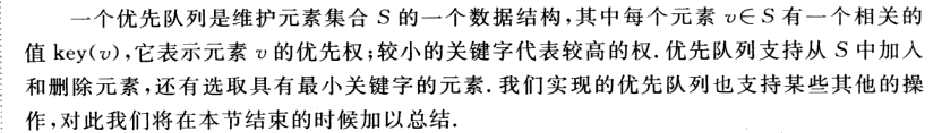
2.4.4立方时间

假设有一集合，S1,...Sn是其的n个子集，我们想知道这些子集中是否有某对子集是不相交的。



二分题用蛮力搜索的话，需要的运算量是n!

2.5 更复杂的数据结构:优先队列

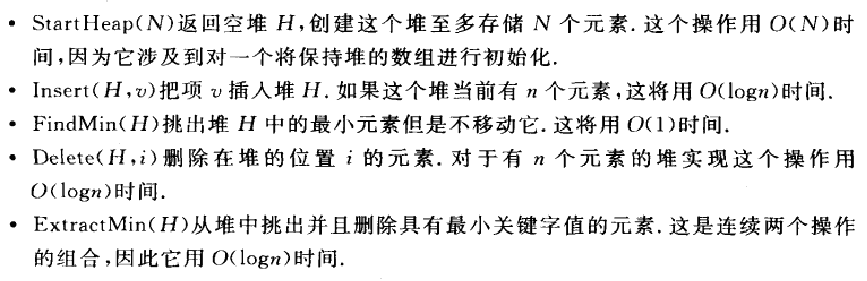


应用例子:管理实时事件(比如一台计算机上进程的调度)

2.5.2 实现优先队列的一种数据结构

从概念上说，我们认为堆是一颗平衡的二叉树

堆的顺序:任何元素的关键字至少在与它的父节点元素的关键字一样大



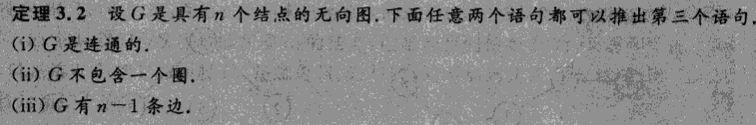
第三章 图

3.1 基本定义与应用

树：一个无向图如果是连通的，且不包含一个圈，我们就说它是一颗树，在某种更强的意义上，树就是最简单的连通图

每棵n个结点的树恰好有n-1条边。

森林:一个无回路(不包含圈)的图



3.2 图的连通性与图的遍历

3.2.1 宽度优先搜索(BFS)

s-t连通性(在G中是否存在一条从结点s到结点t的路径？)

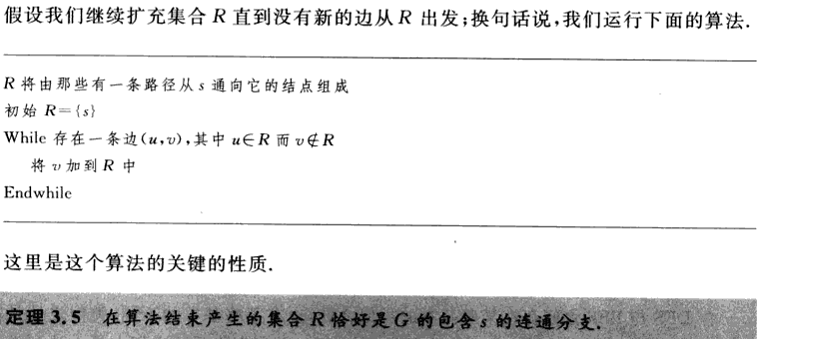
BFS：从s向外在所有可能的方向上探索，一次增加一层结点(一层一层的搜下去)

BFS 不仅确定了s可以达到的结点， 它也计算了到它们的最短路径(因为它知道了每个结点处在第几层，代表了s到该层所有结点的距离

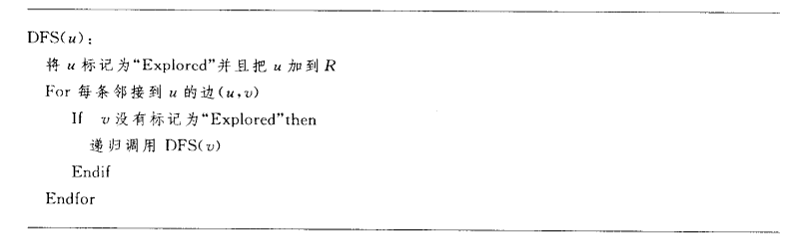
宽度优先搜素树(某种特定方式产生的树)：

从某个结点s开始，利用宽度优先搜索产生的树

3.2.2探查一个连通分支

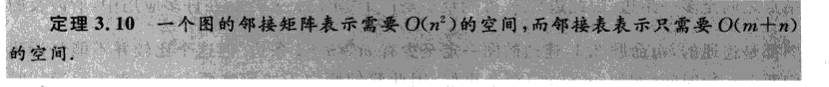


3.2.3 深度优先搜索(DFS)



3.3.1 图的表示

n个结点，m条边

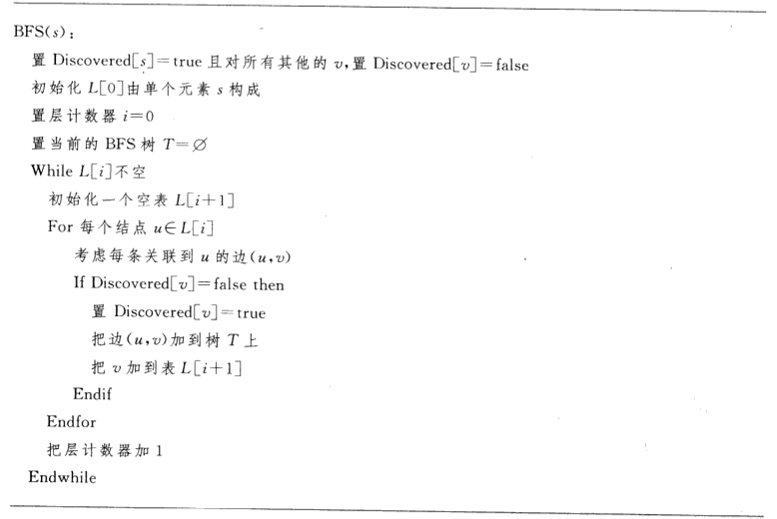


3.3.2 队列与栈

一个队列是一个集合，按照先进先出次序从中取出元素：按照与加入元素相同的次序来取出元素。一个栈是一个集合，我们按照后进先出次序从中取出元素。队列和栈很容易用双链表来实现。

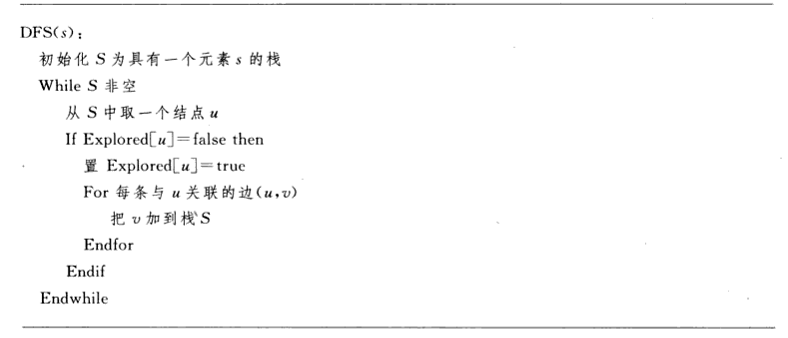
3.3.3 宽度优先搜索(BFS)的实现

BFS可以被想成用一个队列来选择哪个结点是下一步考虑的结点，而DFS用栈是有效的。



如果图是由邻接表表示给出的，BFS算法的上述实现将以O(m+n)时间(即以输入规模的线性时间)运行

3.3.4 深度优先搜索的实现





3.4 二分性测试: 宽度优先搜索的一个应用

二部图：二部图是一个图，其中结点集V可以用下述方式划分成集合X与Y，每条边的一端在X中，而另一端在Y中

3.4.1 问题

一个图如何判断是否是二部图——即是否能够将它按要求分成红结点和蓝结点。

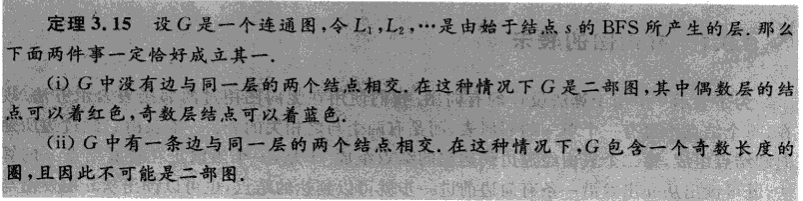
奇圈：奇数长度的圈

如果一个图是二部图，那么它不可能包含一个奇圈

3.4.2 设计算法

在BFS的基础上实现这一点，从某结点s出发，生成l0,l1,l2,...层，交替染色。如奇数层结点着蓝色，偶数层结点着红色，该过程结束时，我们只需要扫描所有的边，以确定是否存在一条边的两个端点是同色的。

3.4.3 分析算法



3.5 有向图中的连通性

3.5.1 有向图的表示

有向图的邻接表表示方法：每个结点不再只有一个单一的邻接表，而是有两个与之相关的表：一个表由它经过边所通向的结点组成，第二个表由经过边到达它的结点组成。

3.5.2 图搜索算法

宽度搜索优先与广度搜索优先

3.5.3 强连通性

一个有向图是强连通的，如果对任意两个结点u和v，存在一条从u到v的路径和一条从v到u的路径。

有向图中两个结点u，v是相互可达的，如果存在一条从u到v的路径并且也存在一条从v到u的路径

3.6 有向无圈图(DAG)与拓扑排序

3.6.1 问题

拓扑排序：我们说G的一个拓扑排序是它的结点作为v1,...,vn的一个排序，使得对每条边(vi,vj)我们有i<j(在一个拓扑排序中，所有的边从左指向右)

定理3.18 如果G有一个拓扑排序，那么G是一个DAG

逆命题：每一个DAG都有一个拓扑排序(成立)

3.6.2 设计与分析算法

计算G的拓扑排序：

找到一个没有进入边的结点v并且将它排在第一

从G中删除v

递归计算G-{v}的拓扑排序并把这个序接在v的后面

四 贪心算法

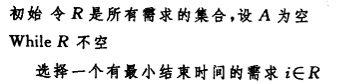
贪心算法：一个算法是贪心的，如果它是通过一些小的步骤来建立一个解，在每一步根据局部情况选择一个决定使得某些主要的指标能得到优化。

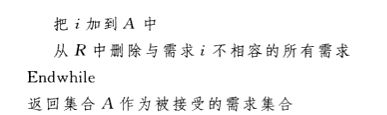
即使一个贪心算法不能得到精确的最优解，它也可以产生一个保证接近于最优解的解。

4.1 区间调度: 贪心算法领先

在一个区间段，尽可能满足多的需求(有单一资源和一组使用资源的n个需求，每个需求需要一个时间区间)

4.1.1 设计贪心算法





A是一个相容的需求集

4.1.4 一个有关的问题：调度所有的区间

所有需求(区间)都需要处理，问题是如何用尽可能少的资源

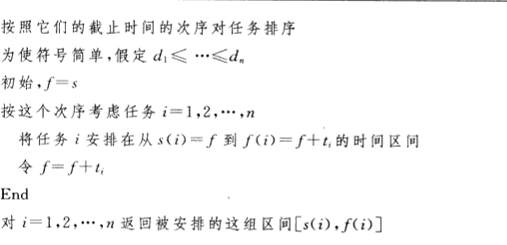
4.2 最小延迟调度： 一个交换论证

问题：每个任务有两个属性，任务消耗时间以及截至时间

目标: 最大延迟L是最小的

解法：最早截至解法

4.2.2 设计算法



4.3 最优超高速缓存: 一个更复杂的交换论证

4.3.1 问题

当处理存储分层的时候，有少量数据可以被快速存取，而大量数据需要更多的时间存取

超高速缓存是一个一般性的专有名词，它是指在一个快速存储器中存储少量数据以便减少与一个慢速存储器的交互而花费的时间。

给定一个存储访问的完全序列，什么是使得超高速缓存缺失尽可能少的收回调度？

4.3.2 设计与分析算法



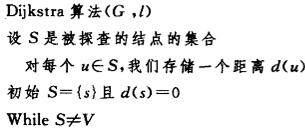
我们把它称作最远将来算法

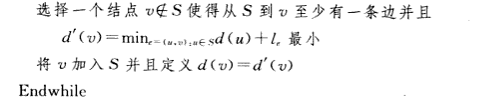
4.3.3 推广：在实际操作条件下的超高速缓存

实际情况下，一般不知道将来的情况

最近最少使用原则，这个原则建议从超高速缓存中收回最久以前被访问的项。

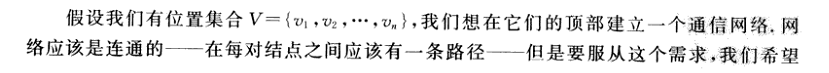
4.4 一个图的最短路径



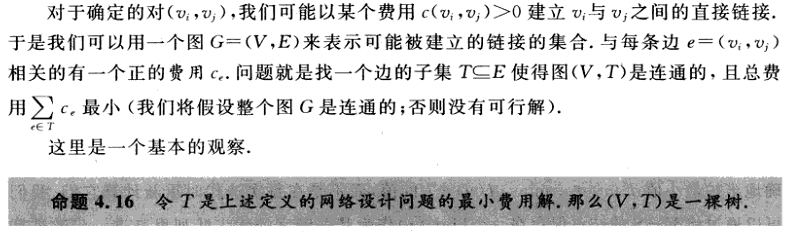


4.5 问题

4.5.1问题



尽可能便宜地建立它。



三个贪心算法，其中每一个都能正确地找到一颗最小生成树

1.Kruskal算法 初始没有任何边，按费用递增次序通过继续不断插入来自E中的边来建立一颗生成树 O(mlogn)

2.Prim :模拟Dijkstra O(mlogn)

3.逆向删除算法

4.6 实现Kruskal 算法: Union-Find 数据结构（并查集）

Union-Find数据结构将支持三种操作：

1.对集合S的MakeUnionFind(S)操作将返回集合S上的一个Union-Find数据结构，其中所有的元素都在分离的集合里

2.对于一个元素uS，操作Find(u)将返回包含u的集合的名字。

3.对两个集合A与B，操作Union(A,B)通过将集合A与B合并成一个集合来改变这个数据结构

4.6.2 一个用于Union-Find的简单数据结构

并查集由一个整数型的数组和两个函数构成。数组pre[]记录了每个点的前导点是什么，函数find是查找，join是合并。

***int pre[1000 ];***

***int find(int x)*                                                                                                         //查找根节点**

***{***

***int r=x;***

***while ( pre[r ] != r )                                                                                              //*返回根节点 r**

**r=pre[r ];**

***int i=x , j ;***

***while( i != r )                                                                                                        //*路径压缩**

***{***

***j = pre[ i ];*** // 在改变上级之前用临时变量  j 记录下他的值

**pre[ i ]= r ;** //把上级改为根节点

***i=j;***

***}***

***return r ;***

***}***

**void join(int x,int y)                                                                                                    //判断x y是否连通，**

**//如果已经连通，就不用管了 //如果不连通，就把它们所在的连通分支合并起,**

**{**

**int fx=find(x),fy=find(y);**

**if(fx!=fy)**

**pre[fx ]=fy;**

**}**

4.7 聚类

4.7.1 问题

一个集合有指数多个不同的k聚类，怎么能有效地找出最大间隔的聚类？

4.7.2 设计算法

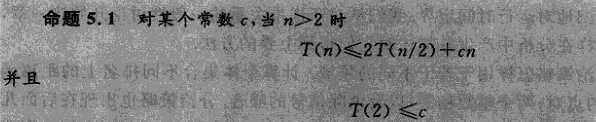
首先生成一颗最小生成树，然后减掉k-1条最贵的枝(边)，就得到了k个类

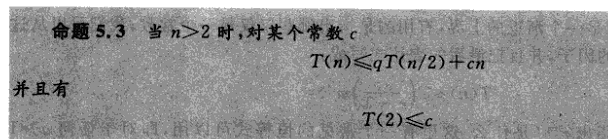
4.7.3 分析算法

由删除最小生成树T的k-1条最贵的边所构成的连通分支C1，C2，...,CK组成一个最大间隔的k聚类

4.8 Huffman码与数据压缩

五 分治策略





q=2，则5.3就是5.1,表示将某个问题每次分解成2个子问题：1-2-4-8-16....

当q>2，随着递归继续进行，每层的总工作量正在增加。当然，也是有界，界由定理5.4给出。



5.1归并排序

5.3计数逆序

5.4给定平面上的n个点，找最邻近的一对点

5.5 整数乘法

5.6 卷积与快速傅里叶变换

卷积，两个向量的卷积得到的结果是一个长为m+n-1的向量，元素是相同阶数的和；如

（a0b0(0阶),a1b0+a0b1(1阶),a2b0+a1b1+a0b2(2阶),....）

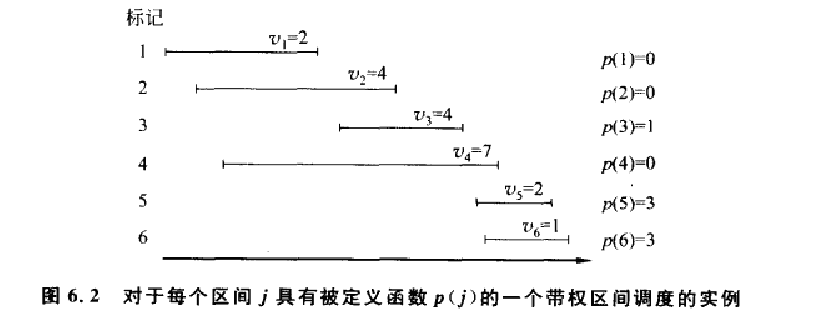
在实践中卷积最重要的应用就是信号处理

利用快速傅里叶变换，只用O(nlogn)次算术运算来计算两个向量的卷积

复数根

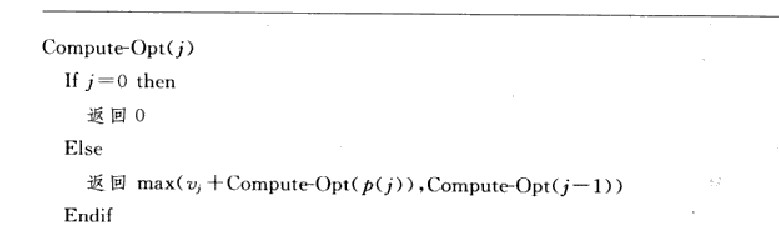
每个复数w= j=0,1,......k-1满足该方程

六 动态规划





这一max选择取决于j是否属于最优解Oj



在第一次计算时就把Compute-Opt值存在一个大家都可以访问的地方，然后在所有后来的递归调用中只是使用这些预先算好的值。这个存储已算好的值的技术叫做备忘录