Lecture1

排序评价

最坏情况
平均案例
需要假设输入的统计分布
最佳情况
(O ∩ Ω) 等价于 Θ
"Θ"含义
Θ(g(n))={f(n):存在正的常数c1,c2和n0使得对于所有n>=n0有0<=c1g(n)<=f(n)<=c2g(n)}
理解：同阶无穷大
"O"含义
f(n)=O(g(n)):存在常数c,n0使得对于n>=n0有0<=f(n)<=cg(n)
理解：g(n)是f(n)的同阶或高阶无穷大
"Ω"含义
f(n)=Ω(g(n))={f(n):存在常数c,n0使得对于n>=n0有0<=cg(n)<=f(n)}
理解：g(n)是f(n)的同阶或低阶无穷大
例子：n^(1/2)=Ω(lgn) (c=1,n0=16)

插入排序

理解：从第二个位置开始，将其与前面的值比较，
直到找到第一个不比它大的数的位置插入
最坏情况Θ(n^2)
平均情况sigma(Θ(j/2))=Θ(n^2)
对于较小的n，比较快

归并排序

理解：代码层面进行的是逐个划分（多数时候是对称二分），
当达到出口后返回，并总是将前面更深层递归函数的返回结果进行排序合并，
然后返回给更浅层递归函数
合并n个元素花费时间Θ(n)
T(n)=2T(n/2)+Θ(n) (n>1)
这样划分的树有lgn层，每层合并消耗Θ(n)共计Θ(nlgn)
最坏情况归并排序优于插入排序，即
归并排序 渐进地 战胜插入排序
实践n>30就更优了

解递归

代入法
理解：猜解，代入验证
例子：$T(n) = 4T(\frac{n}{2}) + n$ 猜测$T(k) \leq ck^3$
得证$T(n) = O(n^3)$
事实上可以通过假设$T(k) \leq c_1k^2 - c_2k$验证$T(n) = O(n^2)$
迭代法

对于上述例子可以计算结果为$\Theta(n^2)$

Master

Lecture2

主定理

T(n) = aT(n/b) + f(n)

a代表子问题个数 n/b代表子问题规模 f(n)代表划分与合并开销

分三种情形

- 运行时间由叶子上的成本主导
  - $f(n) = O(n^{\log_b a - \epsilon})$ ε为大于0的常数
  - 则$T(n) = O(n^{\log_b a})$
  - 说明：O可以同时替换为Θ
- 运行时间均匀地分布在整个树上
  - $f(n) = \Theta(n^{log_b a}(\lg n)^k), k \geq 0$的常数
  - 则$T(n) = \Theta(n^{log_b a}(\lg n)^{k+1})$
- 运行时间以根部的成本为主
  - $f(n) = \Omega(n^{\log_b a + \epsilon})$ ε为大于0的常数
  - 若同时有af(n/b)<=cf(n) (c<1) 则T(n)=Θ(f(n))

实际使用可采取$\frac{f(n)}{n^{\log_b a}}$

$T(n) = 4T(\frac{n}{2}) + \frac{n^2}{\lg n}$

不符合主定理任何一个，采取代入法

$T(n) = \Theta(n^2 \lg \lg n)$

$\sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$

分治法

- 矩阵乘积
  - $n * n$矩阵=$2 * 2$个$(n/2) * (n/2)$矩阵
  - 相乘包含8个乘法运算（矩阵层面）和4个加法运算（矩阵层面）
  - 故有 T(n) = 8T(n/2) + Θ(n^2)
  - 8:8个乘法运算 Θ(n^2): 即$4 * (n/2)^2$代表矩阵相加
  - 推知T(n)=Θ(n^3)
- 多项式乘积
  - A(x) = a0 + a1x + a2x^2 +...+ anx^n
  - B(x) = b0 + b1x + b2x^2 +...+ bnx^n
  - C(x) = A(x)•B(x) = c0 + c1x +c2x^2+...+ c2nx^2n
  - 普通算法
    - Θ(n^2)
  - 分治算法
    - A(x)=P(x)+x^(n/2)Q(x)
    - B(x)=R(x)+x^(n/2)S(x)
    - C(x) = P(x)·R(x)
    - + x^(n/2)(P(x)·S(x) + R(x)·Q(x))
    - + x^nQ(x)·S(x)
    - 划分消耗n/2+n/2 计算4个子多项式乘积
    - T(n)=4T(n/2)+ Θ(n) 推知T(n)=Θ(n^2)
    - 优化：(a+by)(c+dy) = ac + (ad + bc)y + bdy^2
      - m1 = (a+b)·(c+d)

m2 = a·c
m3 = b·d
可直接计算出ad + bc = m1 - m2 - m3
T(n) = 3T(n/2) + Θ(n) 推知T(n)=Θ(n^lg3)
验证n*n矩阵AB=C
随机选择向量r = (r1,..., rn)
验证(AB)r ?= Cr

Lecture3

Randomized Quicksort

```
x=A[r]
i=p
while A[i] <= x and i <= r
      do i = i + 1
for j = i + 1 to r
       do if A[j] <= x
               then exchange A[i] and A[j]
                                    i = i + 1
return i–1
```

这里i指出了第一个大于x的位置
快速排序分析
partition对称例如1/2 1/2
T(n)=Θ(nlgn)
partition不均匀时例如1/10 9/10
cnlog_10^n <= T(n) <= cnlog_10/9^n + O(n)
最坏情况，每次有一边没元素，当元素倒序排列就可能发生这种情况
T(n)=Θ(n^2)
时而幸运时而不幸（对称和元素单边情况交替）
L(n) = 2(L(n/2–1) + Θ(n/2)) + Θ(n)
= 2L(n/2–1) + Θ(n)
= Θ(nlgn)
围绕随机元素分区，实践运作良好
Randomized-Partition(A, p, r)
1.i=Random(p, r)
2.exchange A[r] and A[i]
3.return Partition(A, p, r)
平均运行时间Θ(nlgn)
概率论知识（仅记录遗忘点）
随机变量X,Y独立,如果任取x,y,Pr{X=x and Y=y} = Pr{X=x}·Pr{Y=y}
并且会有E[XY] = E[X]·E[Y]
lg(n/2)=lgn-1

Lecture4

比较排序下界

- 目前为止所看到的的排序算法都是比较排序
  - 插入、合并、快速、堆排序
- 最好的 最坏情况运行时间为O(nlogn)
- Decision-tree
  - 节点表示：i:j
  - 左子树写明ai<=aj时应当进行的下一步比较
  - 右子树写明ai>=aj时应当进行的下一步比较
  - 叶子会指示排序结果
  - 决策树可以模拟任何比较排序，算法的运行时间=所走路径的长度。
  - 最坏情况下的运行时间=树的高度。
  - 排序n个元素的决策树高度至少有Ω(nlgn)
    - 证明过程用到Stirling公式 推出 n!>=(n/e)^n
    - $n! = \sqrt{2\pi n}(\frac{n}{e})^n(1 + \Theta(\frac{1}{n}))$
    - *利用放缩夹逼能证明log(n!) = Θ(nlgn)
      - log(n!) = log1 + ... + logn ≥ n/2 * log(n/2) = n/2 * logn - n/2 = c1 * nlogn
      - log(n!) = log1 + ... + logn ≤ nlogn = c2 * nlogn

order statistics以及线性时间Median算法（select）

- 顺序统计量
- 期望为线性时间的选择算法
  - 在n个元素中寻找第i小的元素
  - RAND-SELECT(A, p, q, i)   _ ith smallest of A[p...q]
    - if p = q then return A[p]
    - r=RAND-PARTITION(A, p, q)
    - k = r – p + 1   _ k = rank(A[r])
    - if i = k then return A[r]
    - if i < k
      - then return RAND-SELECT(A, p, r – 1, i)
      - else return RAND-SELECT(A, r + 1, q, i – k)
    - 以上的期望上界是Θ(n)
- 最坏情况为线性时间顺序统计量
  - 线性时间Median算法
    - 理解：选择算法：
    - n个元素分为5个元素一组（Θ(n)），插入排序找出每一组的中位数（Θ(1)），
    - 对于每一组取出的所有中位数，我们递归调用选择算法找出它们的中位数x（T(n/5)），
    - 按照x进行对组进行partition(Θ(n))，k表示低区元素数目+1，则x即为第k小的数，如果符合所求就返回，

    - 反之，则在高区或低区的元素中递归寻找
    - 需要额外考究的是x的位置，因此至少有[n/10]个组中位数<=x这些组有3个数<=x
    - 因此至少3[n/10]个元素小于x，同理至少3[n/10]个元素大于x
    - n>50时3[n/10]>n/4,因此知道在高区或者低区递归寻找的开销在T(3n/4)

    - 综上T(n)=T(1/5n)+T(3/4n)+Θ(n)
    - 代入法可解T(n)=Θ(n)

求frequent item的Misra-Gries算法

给定数据序列a1,a2,...,am,ai∈{1,2,...,n}
求数在序列中出现的频数相关问题
Misra-Gries算法
读入数为x
if已经有x的计数器，增加
else if没有x的计数器，但是计数器使用的数量还没到达c个，则创造一个x的计数器并初始化为1
else把所有计数器减1，删除值为0的技术器
空间O(c(logm+logn))
时间O(logc)每个数据
Misra-Gries算法输出的数据项并不一定是频繁项，但是频繁项一定在输出结果之中
原来的频数fj-输出的频数$\overline{fj} \leq \frac{m}{c}$,原因是每次减1都说明读入了除了j以外的c个数

Lecture4-hashing-1

哈希函数：全域哈希、Perfect hashing

一个好的hash函数应该将key均匀分配到表的槽中
1.h(k) = k mod m
极度的缺陷情况：
m=2^r导致k的分配只取决于k二进制表示的低r位
2.乘法方法
h(k) = (A·k mod 2^w) rsh (w – r)
计算机w位的字 rsh右移 A为奇数且$2^{w-1} < A < 2^w$且A不要离2^w太近
3.点乘法
m为素数，把k分解为r+1个{0,1,...,m-1}数字k_i
随机挑选a=<a0,a1,...,ar>,ai∈{0,1,...,m-1}
$h_a(k) = \sum(a_i k_i) mod m$
全域哈希：
随机选择hash函数
理解：H是hash函数集合，把U映射到{0,1,...,m-1}，如果对于U中的所有x≠y
h∈H满足h(x)=h(y)，这样的函数h个数<=|H|/m，那么H就是全域的
这意味着随机选择h的时候对于x，y的映射冲突最多只有1/m概率，推知假设n个key插入m个槽则E[与x冲突数量]<n/m
将集合H划分为m份，对于任何两个不相等的key，只有1份包含的hash函数是会使得两者的hash值相等
点乘法的哈希函数是全域的且hash函数集合|H|=m^(r+1)
对于x与y表示成的不相同的xi与yi（至少存在一对），它的系数ai会与其余选定的aj后得出的唯一值冲突，
并且其余aj可以任取，因此h_a函数会有m^r=|H|/m个导致x和y冲突
*对于素数m，任取z属于Z_m且z不为0，存在唯一z^(-1)使得
$z * z^{-1} \equiv 1 (mod m)$
一个简单全域哈希例子分析见纸质笔记
$h_{(a,b)}(x) = ((ax + b) mod\ p) mod\ m$
Perfect hashing
给定n个key，建立静态hash表，大小为m=O(n)，使得在最坏情况搜索花费时间为Θ(1)
二级框架，每级采用全域哈希
如果把n个键哈希到m=n^2个槽里面，且用到的hash函数从全域集合中随机取出，则期望碰撞次数

<1/2

分析：$C_n^2 * \frac{1}{n^2} < \frac{1}{2}$

ball-and-bin模型

m个球，n个桶，随即把球扔进桶里
Xi表示第i个桶里球的数量
k≙max(X1,X2,⋯,Xn)
求k的期望分布
1.m=o(sqrt(n))
Pr(k>1)=o(1)
k=1 w.h.p(with high probability)
2.m=Θ(sqrt(n))
compute Pr(k>1) again
k=1 or 2 w.h.p
3.m=n
找到合适的x使得Pr(k<=x)=1-o(1)
k=Θ(ln/lnln(n)) w.h.p
4.m>=nln(n)
k=Θ(m/n) w.h.p

*two-choices扩展

Lecture4-hashing-2

开放寻址法（分析linear probe）

hash函数取决于key和探测数
h : U × {0, 1, ..., m−1} -> {0, 1, ..., m−1}
<h(k,0), h(k,1), ..., h(k,m−1)> 是{0, 1, ..., m−1}上的迭代
线性探测
给定一个普通的hash函数h'(k)
h(k,i) = (h'(k) + i) mod m
缺点是占用槽使平均搜索时间变长（开始的时候堆在一起，后面迭代路径很长）
Double hashing
h(k,i) = (h1(k) + i * h2(k)) mod m
h2(k)必须是相对于m的素数
比如m取2的幂次，h2(k)只产生奇数
开放寻址法分析
假设均匀散列：每个key有同样可能性将m!排列组合中的任何一个作为探测序列
给定开放寻址哈希表，负载系数α=n/m<1，
失败探测（插入新元素）期望$\leq \frac{1}{1-\alpha}$
理解证明：一开始总得先探测一次，然后有n/m的概率会是冲突，
如果冲突则又需要探测一次，然后会有(n-1)/(m-1)的概率会是冲突，一直下去
这个期望式子可以放缩成关于α的等比数列求和，得到$\frac{1}{1-\alpha}$
成功探测期望最多$\frac{1}{\alpha}\ln(\frac{1}{1-\alpha})$
理解证明：E[插入第i+1元素的探测次数]<=1/(1-i/m)=m/(m-i)
对i求和并除以n，放缩成积分式子求解

Cuckoo hashing及分析

两个表T1，T2，大小m=(1+ε)n，两个hash函数
查看T1和T2
首先在T1插入元素的时候，如果hash出来的位置是空位则直接插入，
否则把这个位置元素挤出来，在T2表里hash寻位
失败的可能
- 空间不够，出现环路
- 操作链太长
- 发现终止：时间限制
- 可能性Θ(1/n)
- 解决方案：rehashing
  - 导致插入最坏情况O(n)
  - save rehashing 优化插入最坏情况O(logn)

Cuckoo图
- 集合能成功存储 等价于 连边构成的图最多有一个环

最坏情况 查询和删除只需要2次访问 可并行

Bloom filter

给定域U上的集合S={x1,x2,...,xn},问y是否在S中
初始化一个均为0的m位数组，把每个xj hash k次（使用k个hash函数）
如果H_i(xj)=a,把B[a]=1设置为1
检验y也采用上面的方法映射，要求所有k次映射的位置上都填了1才能确实y在S中
当然也可能存在，检验通过，但的确不在S中的情况
权衡
- size m/n
- time k
- error f=Pr[false pos]=(1-p)^k其中 $p = Pr[cellisempty] = (1-\frac{1}{m})^{kn} \approx e^{-\frac{kn}{m}}$

Count-sketch
初始化
- $C[1\dots t][1\dots k]=0, k=\frac{2}{\epsilon}, t=\log(\frac{1}{\delta})$
- 选择t个独立的hash函数h1,...,ht:[n]->[k]，每个都是来自2全域集合

处理(j,c)
- for i = 1 to t
  - do C[i][h_i(j)] = C[i][h_i(j)] + c

输出
- f_a = min_(1<=i<=t)C[i][h_i(a)]

consistent hashing及分析

用途
- 由许多用户共享的网络缓冲区，通过汇总许多用户最近页面请求，减少延迟
- 上述需要大量快速存储，并进行有效检索
- 希望将大规模缓存分散到多台机器上

方法
- 将机器和对象hash在同一个范围内，为了分配对象x，计算hi(x),
- 然后向右遍历直到找到第一台机器的hash h_m(Y)，那么将x分配给Y
- 分析：当有服务器增减，只需要把数据移动到别的服务器即可
- 时间O(logn)
  - 使用二叉搜索树，把分配了的服务器的索引存在树里

用h(x)在logn时间找后继节点
创造多个机器和hash的复制

Lecture5-amortized analysis

平摊分析

Binary Counter
case1
k位数组，表示0-2^k-1
操作：+1 开销：被翻转的位数
aggregating
直接对n次操作开销求和，取平均值做平摊开销
换个视角，对所有加1操作的同一位翻转次数求和（例如k=5，最低位总共会翻转32 = 2^5 = 2^5 / 2^0次）
则总共开开销求和<=2n,即O(n),进而平摊开销=O(n)/n=O(1)
case2
k位数组，表示0-2^k-1
操作：+1 开销：第i位翻转开销为2^i
accounting
实际开销低于分配的金额则将多余的存储进银行，反之从存储中扣除
设置0->1=2$ 1->0=0$
当把0翻转成1，支付1$，然后存储1$；当把1翻转成0，从银行支付开销
分析：数组中每有1个1意味着银行都有其1$的存款，所以当存在把1翻转成0的操作时必定是有存款的
potential
把势能与数据结构联系起来 势能是"造成损害的潜力"
平摊开销=实际开销+新潜力-旧潜力
基本规则
总是非负数
从0开始
意味着一连串n个操作成本最多是平摊开销的n倍
平摊开销$\overline{c_i}$ 实际开销c_i $\overline{c_i} = c_i + \varphi(D_i) - \varphi(D_{i-1})$
$\sum(\overline{c_i}) = \sum(c_i) + \phi(D_n) \geq \sum(c_i)$
寻找势能方法：寻找让数据结构坏情况
在Binary Counter例子中，有很多1就是坏情况
φ(Counter)=1的数量
势能增加=(0->1翻转数量)-(1->0翻转数量)<=1-(1->0翻转数量)（分析：这里把0变为1意味着不可能再进位了）
平摊开销=实际开销+势能增加<=(1+(1->0翻转数量)) + (1-(1->0翻转数量))=2
所以总开销最多2n
Dynamic Table
case1
分配内存重新插入 n个插入操作 表满要double
最坏的一次开销可能Θ(n),但总开销<<n*Θ(n)=Θ(n^2)
用ci表示第i次插入开销 如果i-i=2^k则ci=i；否则ci=1
aggregating
开销求和<=3n
accounting
设置第i次插入开销3$；平摊开销ci，立即支付1$，存储2$进银行

当表大小翻倍的时候，1$插入新项目，1$重新插入旧项目

理解：插入一个数的时候：支付自己插入开销 支付自己扩张时的移动开销 替前面某个数支付扩张时的移动开销

potential

$\varphi(D_i) = 2i - 2^{\lceil \lg i \rceil} = 2num_i - size_i$

case2

在前面基础上，改成表满了double，表不到一半要削减成1/2

accounting

ci=3 if插入，ci_=2 if删除

O(n)

self-adjust list

Move-to-front分析

自调整表就像是一种规则的表，

但是它的插入与删除操作都在表头进行，

同时当任一个元素被find访问时，

它就被移到表头而不改变其它元素的相对位置。

基于链表实现的自调整表比较简单，基于数组实现的则需要更多的小心

Lecture6-dp

memorize方法

表述最优解的结构

递归定义最优解的值

计算最优解的值通常采用自下而上的方式，从计算的信息中构建一个最优解

weighted interval schedule

工作j在sj开始，在fj结束，并有权重或值vj。

如果两个工作不重叠，则它们是兼容的。

目标：找到相互兼容的工作的最大权重子集

用p(j)定义最大索引i<j使得工作i是与j兼容的

OPT(j)由工作请求1、2、...、j组成的问题的最优解的值

OPT(j)=max{vj+OPT(p(j)), OPT(j-1)}

记忆化

将每个子问题结果存储在一个缓存M[]中根据需要查询

记忆化运行时间O(nlogn) 求p使用O(nlogn) 求递归计算O(n)

输出方案

```
Find-Solution(j) {
  if (j == 0) output nothing
  else if (vj + M[p(j)] > M[j-1]) {
    print j
    Find-Solution(p(j))
  }
  else {
    Find-Solution(j-1)
  }
}
```

自下而上动态规划解除递归

```
compute() {
    M[0] = 0
    for j = 1 to n
        M[j] = max(vj + M[p(j)], M[j-1])
}
```

矩阵连乘

A*B*C 考虑结合率操作数是不同的
对于A=A1A2...An
暴力算法
Ω(2^n)
优化算法
(A1A2...A_i)(A(i+1)A2...An)
N(i, j)=min(i<=k<j){N(i,k)+N(k+1,j)+d_id(k+1)d_(j+1)}但子问题有交叠
如果一个问题的最优解包含其子问题的最优解，那么这个问题就表现出最优子结构。
矩阵链乘法。一个AiAi+1...Aj的最佳括号内包含了AiAi+1...Ak和Ak+1Ak+2...Aj括号问题的最佳解

最长公共子序列(LCS)
给出两个序列x[1...m]和y[1...n]
找到两个序列共同的最长子序列
暴力算法
找x所有子串（2^m），在y中一一校对是不是y的子串（Θ(n)）
最坏情况时间Θ(n*2^m)
递归算法
定义c[i, j]=x[1...i]与y[1...j]的LCS的长度 转化为了求c[m, n]
c[i, j] = c[i - 1, j - 1] + 1 if x[i] = y[j]
max{c[i, j - 1], c[i - 1, j]} otherwise

带权重的最优二分查找树

定义二叉树的节点有权重 从根到所有节点，路径上经过节点数*权值求和WIPL
给定节点K2<K2<...<Kn 对应频数fi
求最小WIPL
算法
对于每个k，i<=k<=j
把Kk放置在T的根
查看BST的L $Ki, \ldots, K_{k-1}$
查看BST的R $K_{k+1}, \ldots, K_{i+j}$
$WIPL(T) = WIPL(L) + WIPL(R) + \sum_{i'=i}^{i+j} f_i$

多段线性回归

回归分析包含多个自变量

背包问题

给定有限集合S 正整数权值函数w:S->N 值函数:S->N 权重限制W∈N
找一个S的子集S'
$\sum_{a \in S'} v(a)$，使得 $\sum_{a \in S'} w(a) \leq W$
算法
用V(k, B)表示使用集合{1,2,...,k}中的项目且最多使用B空间的最高值方案的值

$$V(k,B)=\begin{cases}0 & \text{if k} = 0\\ V(k-1,B) & \text{if } w_k > B\\ max\{v_k+V(k-1,B-w_k),V(k-1,B)\} & otherwise\end{cases}$$

树上独立集

- 带权重独立集（WIS）
  - 点带权重 取出的点相互无连边 求点权重和最大的独立集选法
  - NP-hard
- 改成树上的带权独立集好解
  - 用C(v)表示节点v的子节点
  - $MIS(u)=max\{\sum_{v\in C(u)}MIS(v),w_u+\sum_{v\in C(u)}MIS(v)\}$
  - 记忆化
    - $U(u)=w_u+\sum_{v\in C(u)}N(v)$
    - $N(u)=\sum_{v\in C(u)}max\{U(v),N(v)\}$
    - 时间O(n)

旅行商问题

- 无向图 边上带长度
- 寻找最短路程使得游客能恰好经过每个顶点1次
- 暴力算法
  - O(n!)
- 目前已知最好算法
  - $O(n^2*2^n)$
- 算法
  - 对于包括1，j的城市子集$S\subseteq\{1,2,\ldots,n\}$，
  - 让C(S，j)为最短路径的长度，正好访问S中的每个节点一次，
  - 从1开始，在j结束。
  - $C(S,j)=min_{i\in S-\{j\}}\{C(S-j,i)+d_{ij}\}$

Lecture7-greedy

通过局部最优解获得全局最优解

单机任务调度

- 工作j在sj开始、在fj结束，两个工作不交叠称为兼容，
- 求最大互相兼容子集
  - 1.最早开始时间×
  - 2.最短时间间隔×
  - 3.最少冲突×
  - 4.最早完成时间
    - 将工作完成时间排序，然后从小到大加入集合，只要新工作和集合不冲突就加入
    - 时间O(nlogn)
    - 反证知该贪心算法得出的是最优解
- 课程j在sj开始、在fj结束，两个工作不交叠称为兼容，
- 求最少的教室数量以保证课程能在其时间开展，且互不冲突
  - 1.最早开始时间
    - 将工作开始时间排序，教室用优先队列排序，排序标准为最晚结束时间升序，
    - 在优先队列新建教室当且仅当，工作j与队列中所有教室不兼容，
    - 每次插入课程都要更新教室的最晚结束时间

时间：优先队列操作O(n)*O(logn)+排序O(nlogn)=O(nlogn)

证明：找到一个时间点，至少有k节课同时在开展，则答案>=k，然后根据贪心构造出k个

且能够说明，当贪心构造需要增加第k个教室（意味着有至少k-1个工作（每个教室至少一个工作）比j开始早，但在j要加入时却仍未结束），则其他方案也至少要增加第k个教室

2.最早完成时间

3.最短时间间隔

任务调度最小延迟

处理器同时间调度一个工作j，j需要tj时间处理，预期完成时间dj，
j开始于sj则将结束于fj=sj+tj
记延迟为lj=max{0, fj-dj}
目标最小化最大的延迟
最早截止时间优先

哈夫曼编码

一个字符的编码不能是另一个字符的长编码的前缀
使用二叉树表示法，将所有字母作为叶子，应当是满二叉树（每个中间结点有两个子节点）
前缀匹配使得解码容易且精准
C片叶子有C-1个中间结点
$p_1 \geq p_2 \geq \cdots \geq p_m$ 求$\sum_i(p_i * l_i)$的最小值
如果pi>pj那么li<=lj
最长的两个字母有相同长度l
最长两个字母仅仅在二进制位最后一位不同
通过合并将问题化归为m-1规模的问题{p1,...,pm–1, pm} -> {p1,..., pm–2,pm–1+pm}
算法理解：对于当前集合中最小的两个pi,pj将它们求和后的值重新放入集合，
而在树上则将和作为两者父节点，并且该节点频数等于pi,pj的频数之和
时间O(nlogn)

Lecture8

图表示

邻接矩阵
Θ(V^2)存储
密集表示
邻接表
Θ(V+E)存储
稀疏表示
对于每个点v，Adj[v]列表表示与v连结的点
无向图列表大小为v的度数 有效图列表大小为v的出度

最小生成树算法

无向连接图，便带权重，生成树指其能连结所有结点的边组成的树
现希望一棵树边权重和最小即最小生成树（MST）
引理：一颗MST，删除一条边得到的两颗树，分别是其对应子图的MST
定理：给定A是V的子集，最小权重边(u, v)使得两端点一个在A，一个在V-A，则它一定在G的MST T上
cut规则
对于图形的任何一个cut，穿过切口的最小权重边必须在MST中

cycle规则

对于一个循环连边，最大权重边必定不能在MST中

Prim

Q为优先队列，表示那些还没有被连接入最小生成树的节点，排序依据是这些点与MST中节点的最小权重连边

Q=V

key[v]=∞,for any v∈V

key[s]=0,确定一个起始点

while Q ≠ ∅

do u = Extract-Min(Q)

for each v ∈ Adj[u]

do if v ∈ Q and w(u, v) <= key[v]

then key[v] = w(u, v)（Decrease-Key）

π[v] = u （实际操作为了不重复修改，只要把当前w(u, v)最小的那条做这样修改即可）

最后连结{(v, π(v))}得到的就是MST

理解：到当前已选所有节点最近的点加入集合

算法时间：|V| *T(Extract-Min)+Θ(E)*T(Decrease-Key)

(采用斐波那契堆可使得Θ(VlgV+E))

(采用二叉堆则为Θ(VlgV+ElgV)=Θ(ElgV))

Kruskal

T为空集合

for each v ∈ V

do MakeSet(v) (创建一个新的集合，v作为唯一元素放进去 O(1))

Sort E by edge weight

for each edge(u, v) ∈ E

do if FindSet(u) ≠ FindSet(v) (O(n))

then T = T ∪ {(u, v)}

Union(FindSet(u), FindSet(v)) O(1)

```
int fi(int x) {
    return (x == fi[x]) ? x : (fi[x] = fi(f[x]));
}
```

理解：以当前最小权重边为顺序，合并各点；使用并查集方法加速合并的过程

提高union-find性能

height rule

union时深度浅的做另一棵树 子树

路径压缩

find(x)

if x ≠ p(x)

then return find(p(x));

else return x;

m次操作总时间O(mlogn)平摊时间O(logn)

一旦一个节点成为非根节点，则其rank就永远确定了

Lecture9

带权图的最短路径

- 定义
  - G = (V, E) 权重函数 w: E -> R
  - w(p)=Σ*(i = 1 to k-1)w(v_i, v*(i+1))
  - δ(u, v)表示从u到v最短路径权重
  - 定理：最短路径的子路径也是最短路径
  - 定理：δ(u, v) <= δ(u, x) + δ(x, v)
- 单源最短路径
  - 从给定源节点s∈V出发，对于所有v∈V，寻找δ(s, v)
  - Mini优先队列
    - Insert(S, x)把x插入到集合S
    - Minimum(S)返回最小key的元素
    - Extract-Min(S)返回并移除最小key的元素
    - Decrease-Key(S, x, k)把x的key值减小到k
  - Dijkstra算法
    - 思想：贪心 仅对非负权重适用
    - 维护已知的到s的最短路径的节点集合S
    - 每个步骤，将与s的距离估计值最小的定点u∈V-S
    - 更新与u相邻的顶点的距离估计值
    - Dijkstra(g, w, s)
      - d[s]=0
      - d[v]=∞ for each v∈V-{s}
      - S=∅
      - Q=V(按key排序V-S的优先队列)
      - while Q ≠ ∅
        - do u = Extract-Min(Q)（这个操作包含查找和移除）
          - S = S ∪ {u}
          - for each v ∈ Adj[u]
            - do if d[v] > d[u] + w(u, v)
              - then d[v] = d[u] + w(u, v) （最后两行是松弛操作）
    - 时间=Θ(V)*T_Extract-Min + Θ(E)*T_Decrease-Key
      - (斐波那契堆Θ(VlgV+E))
      - (二叉堆Θ(VlgV+ElgV)=Θ(ElgV))

无权重图

- w(u, v) = 1 任取(u, v)∈E
- Dijkstra算法能优化使用FIFO队列
- BFS
  - 时间Θ(V+E)
  - BFS(G, W, S)
    - d[s]=0
    - d[v]=∞ for each v∈V-{s}
    - Q={s}
    - while Q ≠ ∅
      - do u = Dequeue(Q)
        - for each v = Adj[u]
          - do if d[v] = ∞
            - then d[v] = d[u] + 1

Enqueue(Q, v)

对于以下问题存在基于BFS的O(n+m)时间算法

测试图G是否是连接的

计算G的生成树

计算G的连接部分

计算对于G的每个节点x，s和v之间任何路径的最小边数

理解：尽可能解决邻近铺开的节点

DFS

输入图G（有向图或者无向图）

```
DFS(G)
    for each vertex u ∈ V
        do color[u] = white
    time = 0
    for each vertex u ∈ V
        do if color[u] = white
            then DFS-Visit(u)
DFS-Visit(u)
    color[u] = gray
    d[u] = time
    time = time + 1
    for each v ∈ Adj[u]
        do if color[v] = white
            then DFS-Visit(v)
    color[u] = black
    f[u] = time
    time = time + 1
```

理解：尽可能深入，无法深入时以退为进，尝试其他路径深入

u是v的祖先当且仅当区间[d[u], f[u]] 包含 [d[v], f[v]]

u与v不相关当且仅当两者区间不相交

图中含有负值权重循环路径

那么一些最短路径可能不存在

Bellman-Ford算法

找到一个从源s∈V到所有v∈V的所有最短路径长度或者确定负值权重循环的存在

```
BF
    d[s] = 0
    for each v ∈ V-{s}
        do d[v] = ∞
    for each i = 1 to |V|-1
        do for each edge(u, v)∈E
            do if d[v] > d[u] + w(u, v)
                then d[v] = d[u] + w(u, v)
    for each edge(u, v) ∈ E
        do if d[v] > d[u] + w(u, v)
            then report that a negative-weight cycle exists
```

时间O(VE)

强连通图

所有顶点对都可以互相到达
G^T是图G的转置，使得所有边的方向反向
Θ(V+E)从邻接表可计算强连通部分
Strongly-Connected-Components(G)
call DFS(G) => f[u]
compute G^T
call DFS(G^T) 在循环中要按照f[u]递减顺序
dfs顺序输出s.c.c的节点

Lecture10-network flow

网络流算法

图G=(V, E) 源节点s 汇节点t
非负容量c(u, v) 如果(u, v) ∉ E，那么c(u, v) = 0
流在G上f: V × V -> R
满足
$f(u,v) \leq c(u,v)$ 任取$u,v \in V$
$\sum_{v\in V} f(u,v) = 0$ 任取$u \in V - \{s,t\}$
$f(u,v) = -f(v,u)$
流的值$|f| = \sum_{v\in V} f(s,v) = f(s,V)$
目标找到最大流值（由于存在各边流量限制）
要将流看成速率而非量
残余流量 $c_f(u,v) = c(u,v) - f(u,v) \geq 0$
残余网络 图G_f表示那些残余流量严格大于0的边组成的图
理解：还有多少能过去或者还有多少能回来，如果能过去或者回来的量是0就不画这条边了
G_f上从s到t的一条增广路径p $c_f(p) = min_{(u,v)\in p} c_f(u,v)$
有$|f| \equiv f(s,V) = f(V,t)$

割

点集分成S和T=V-S

净流量$f(S,T) = \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{u\in S}\sum_{v\in T} f(v,u)$

割的容量$c(S,T) = \Sigma_{u\in S}\Sigma_{v\in T} c(u,v)$

最小割就是指网络中所有割里流量最小的

$f(S,T) \leq c(S,T)$（展开容易证明）

最大流最小割定理

如下是等价的：

f是G的最大流

残余网络G_f没有增广路径

$|f| = c(S,T)$，对于G的一些割(S, T)

Ford-Fulkerson

$f(u,v) \leftarrow 0, \forall u, v$

while ∃augmenting path p in G_f

do augment f by $c_f(p)$

每条路径O(E)(通过DFS或者BFS)

运行时间$O(E|f^*|)$，f^*是最大流

Edmonds-Karp

修改Dijkstra算法（shortest path）

从s到t的最大流F，必定存在一条从s到t的路径其容量至少为$\frac{F}{m}$

最多$O(m\log F)$次迭代

运行时间$O(m^2\log n\log F)$

通过BFS寻找增广路径（fattest path）

寻找增广路径时间$\Theta(V+E)\Rightarrow\Theta(E)$ 如果源能到达所有的节点

增广次数最多$\Theta(VE)\Rightarrow$ 运行时间$\Theta(VE^2)$

层级图

L_G是有向宽度优先搜索，从s往t的边留下，往回的边删除，组成的图

节点u的层级是其到s的最短路径长度

用$\delta(v)=\delta_f(s,v)=G_f$中的宽度优先搜索距离，$\delta(v)$是单调递增的

定理：增广步骤在E-K算法里是$\Theta(VE)$

Dinic（阻塞流）

如果G中的每个s-t路径，即原始图，都有一些边被饱和，那么G中的一个流f就是阻塞的。每个最大流显然也是一个阻塞流。

- 以深度优先的方式从源点到汇点遍历层级图尽可能前进，并跟踪从s到当前顶点的路径。如果一直走到t则找到一条增强路径，增强它。如果到达一个没有出边的顶点我们就删除该顶点并撤退

  Initialize(O(m))

  1.construct a new level graph L_G

  2.set u = s and p = [s]

  3.go to Advance

  Advance(O(mn))

  1.If there is no edge out of *u*, go to Retreat.

  2.Otherwise, let (*u*,*v*) be such an edge.

  3.Set *p* = *p*||[*v*] and *u* = *v*.

  4.If *v* ¹ *t* then go to Advance.

  5.If *v* = *t* then go to Augment.

  Retreat(O(m+n))

  1.If *u* = *s* then halt.

  2.Otherwise, delete *u* and all adjacent edges from *L**G* and remove *u* from the end of *p*.

3.Set *u* = the last vertex on *p*.

4.Go to Advance.

Augment(O(mn))

1.Let D be the bottleneck capacity along *p*.

2.Augment by the path flow along *p* of value D, adjusting residual capacities along *p*.

3.Delete newly saturated edges.

4.Set *u* = the last vertex on the path *p* reachable from *s* along unsaturated edges of *p*.

5.Set *p* = the portion of *p* up to and including *u*.

6.Go to Advance.

⇒每个phase需要O(mn)⇒总时间$O(mn^2)$

理解：正常增广，删除G_L中已经满的边，然后从G_f中取出那些流向无法继续深入的端点的边，舍弃流向回到s方向的边，增广直至G_L没有路径到达t

Lecture11-network flow-2

单位流图

所有边的容量为1

Dinic算法会花费$O(min(m^{\frac{1}{2}},2n^{\frac{2}{3}}))$次迭代

找到一个阻塞流的时间需要$O(m)$

寻找最大流$O(m*min(m^{\frac{1}{2}},2n^{\frac{2}{3}}))$

不相交路径问题

给定一个数字图G = (V, E)和两个节点s和t，找出最大数量的边相接的s-t路径。

如果两条路径没有公共边则路径不相交

定理：边相接的s-t路径的最大数量等于最大流量值。

匹配

无向图G=（V，E），如果每个节点最多出现在M的一条边上，那么$M\subseteq E$就是一个匹配

最大匹配 最大基数匹配

二元匹配

输入：无向图$G=(L\cup R，E)$。

如果每个节点最多出现在M的一条边上，那么$M\subseteq E$就是一个匹配。
最大匹配 最大基数匹配。

考虑最大流形式，即L外侧加上源点s，并给每条边流入1，R外侧加上汇点t，并从每条边汇入；最大匹配数量就是新图的最大流

算法

通用增广路径$O(m|f^*|)=O(mn)$

capacity scaling$O(m^2\log C)=O(m^2)$

最短增广路径$O(mn^{\frac{1}{2}})$

最小割做法

收缩 找不穿过最小割的边，合并它的端点成为1个节点，直到只剩2个节点

$O(m)$时间来挑选边，n次合并$\Rightarrow O(mn)$时间

采用随机手段选取边

分析：假设最小割为c，那么最小度数至少为c，则至少$\frac{nc}{2}$条边

$Pr[min-cut\ edge]\le\frac{c}{\frac{nc}{2}}=\frac{2}{n}$

$Pr[success]\approx\frac{2}{n^2}$

Lecture11-LP

线性规划

- 给定m条线性等式，n个变量$\{x_i\}$，系数和等式右边值给定，寻找符合条件的$\{x_i\}$，注意不允许类似$x_1\cdot x_2\ge 100$、$\log x_3$出现
- LP标准形式，给定a_{ij},c_j和$b_i,1\le i\le m,1\le j\le n$,寻找$x_1,x_2,\ldots,x_n$(均≥0),使得

$$\sum_{j=1}^{n}c_jx_j$$

  值最大，并且满足如下限制

$$\sum_{j=1}^{n}a_{ij}x_j\le b_i,1\le i\le m$$

- 凸集的交点是凸的
- 网络流模型 变量f_{uv}表示边(u,v)的正值流，计算$\Sigma_v f_{sv}$

  限制：对于所有的边$(u,v),0\le f_{uv}\le c(u,v)$

  对于所有$v\notin s,t,\Sigma_u f_{uv}=\Sigma_u f_{vu}$
- LP有对偶式,限制改成≥,求值改成min

应用

- 计算 起始点s到目的点t的最短路径（边长度$l(u,v)\ge 0$）

  变量$d_v,v\in V$

  目标$max\ d_t$

  限制

$$s.t.\ d_s = 0$$
$$d_v - d_u \le l(u,v), \forall (u,v) \in E$$

- 最大流

  最大化$\sum_{v\in V} f_{sv} - \sum_{v\in V} f_{vs}$

  满足约束

$$f_{uv} \le c(u,v), for\ every\ u,v \in V$$
$$\sum_{v\in V} f_{vu} = \sum_{v\in V} f_{uv}, for\ every\ u \in V - \{s,t\}$$
$$f_{uv} \ge 0, for\ every\ u,v \in V$$

- 最小费用流

  最小化$\sum_{(u,v)\in E} a(u,v) f_{uv}$

  满足约束

$$f_{uv} \le c(u,v), for\ every\ u,v \in V$$
$$\sum_{v\in V} f_{vu} - \sum_{v\in V} f_{uv} = 0, for\ every\ u \in V - \{s,t\}$$
$$\sum_{v\in V} f_{sv} - \sum_{v\in V} f_{vs} = d$$
$$f_{uv} \ge 0, for\ every\ u,v \in V$$

- 多商品流

  最小化0 我们不去最小化任何目标函数，只需确定是否存在这样一个流

  满足约束

$$\sum_{i=1}^{k} f_{iuv} \le c(u,v), for\ every\ u,v \in V$$
$$\sum_{v\in V} f_{iuv} - \sum_{v\in V} f_{ivu} = 0, for\ every\ i = 1,2,\ldots,k\ and\ u \in V - \{s,t\}$$
$$\sum_{v\in V} f_{i,s_i,v} - \sum_{v\in V} f_{i,v,s_i} = d_i, for\ every\ i = 1,2,\ldots,k$$
$$f_{iuv} \ge 0, for\ every\ u,v \in V\ and\ for\ every\ i = 1,2,\ldots,k$$

Lecture12-FFT

- 使用FFT可在两个多项式相乘时，将复杂度降低到$O(n\log n)$
- DFT（离散傅里叶变换）

  ω_n表示单位根，$\omega_n = e^{\frac{2\pi i}{n}} = \cos(\frac{2\pi}{n}) + i\sin(\frac{2\pi}{n})$

  当n是偶数时

$$\{(\omega_n^0)^2, \ldots, (\omega_n^{n-1})^2\} = \{\omega_{\frac{n}{2}}^0, \ldots, \omega_{\frac{n}{2}}^{\frac{n}{2}-1}\}$$

  证明：$(\omega_n^j)^2 = e^{2(\frac{2\pi i}{n})j} = e^{\frac{2\pi i}{\frac{n}{2}}j} = \omega_{\frac{n}{2}}^j$

- FFT（快速傅里叶变换）

  将$a(x)$分割成$a^{[0]}(x)$和$a^{[1]}(x)$

$$a^{[0]}(x) = a_0 + a_2 x + \cdots + a_{n-2} x^{\frac{n}{2}-1}$$
$$a^{[1]}(x) = a_1 + a_3 x + \cdots + a_{n-1} x^{\frac{n}{2}-1}$$

因此$a^{[0]}(x^2) + x a^{[1]}(x^2) = a(x)$

$P = \{(\omega_n^0)^2, \ldots, (\omega_n^{n-1})^2\} \Rightarrow P = \{\omega_{\frac{n}{2}}^0, \ldots, \omega_{\frac{n}{2}}^{\frac{n}{2}-1}\} \Rightarrow |P| = \frac{n}{2}$

计算$a(\omega_n^j) = a^{[0]}((\omega_n^j)^2) + \omega_n^j a^{[1]}((\omega_n^j)^2)$

因此只需要在$\frac{n}{2}$个点上递归地计算两个度数为$\frac{n}{2}-1$的多项式就可以了!

时间$T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n \log n)$

- Inverse DFT

  给定值$a(\omega_n^0), a(\omega_n^1), \ldots, a(\omega_n^{n-1})$，由$y_0, y_1, \ldots, y_{n-1}$

  目标计算系数$a_0, a_1, \ldots, a_{n-1}$

  算法

  $a_j = y((\omega_n^{-1})^j)$

$$\sum_{l=0}^{n-1} \omega_n^{jl} \omega_n^{-lk} = \begin{cases} n & if\ j = k \\ 0 & otherwise \end{cases}$$

- 两个多项式相乘 在$O(n \log n)$时间解决

  输入

$$a(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$
$$b(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$$

  输出

$$c(x) = a(x) * b(x) = c_0 + c_1 x + \cdots + c_{2n-2} x^{2n-2}$$
$$c_i = a_0 b_i + a_1 b_{i-1} + \cdots + a_{i-1} b_1 + a_i b_0$$

  基于FFT的算法

  把a,b扩张成2n-2度数

  计算$a(\omega_{2n}^0) \ldots a(\omega_{2n}^{2n-2})$和$b(\omega_{2n}^0) \ldots b(\omega_{2n}^{2n-2})$（FFT）

  计算$c(\omega_{2n}^j) = a(\omega_{2n}^j) \cdot b(\omega_{2n}^j), j = 0 \ldots 2n-2$

  计算$c_0, c_1, \ldots, c_{2n-2}$（inverse FFT）

Lecture13-NPC

NP-hard

对于一个问题Π使得所有的$\Pi^{'} \in NP$，有$\Pi' \leq \Pi$，那么它是NP-hard

NP-complete

一个NP-hard问题同时属于NP问题，那么它是NP-complete

SAT

- 给定公式ϕ，有m条句子$C_1, C_2, \ldots, C_m$,有n个变量

  检查是否存在对变量赋值，使得公式可满足
- $SAT \in NPC$

Clique

- 输入无向图$G=(V,E)$,值K
- 输出是否存在一个子集$C \subseteq V, |C| \geq K$,使得$C$中每一对顶点都有一条边在它们之间
- 理解：找完全子图

独立集

- 输入无向图$G=(V,E)$
- 输出是否存在一个子集$S \subseteq V, |S| \geq K$,使得没有一对S中的节点有边相连

节点覆盖

- 输入无向图$G=(V,E)$
- 输出是否存在一个子集$C \subseteq V, |C| \leq K$，使得对于任意给定的边，都会和至少一个$C$节点相连

Exact-3SAT

- 输入给定公式ϕ，有m条句子$C_1, C_2, \ldots, C_m$,有n个变量，且$|C_i|=3$，$for\ 1 \leq i \leq m$
- 检查是否存在对变量的赋值使得公式可满足

k-可着色

- 给图中顶点染色，使得有连边的两个顶点着色不同
- 检查是否存在染色方式

Exact 节点覆盖

- 给定有限集合X

$$X=\{x_1, x_2, \ldots, x_n\}$$
$$S=\{X_1, X_2, \ldots, X_m\}, X_j \subseteq X$$
$$S' \subseteq S: \forall x_i \in X, \exists X_j \in S', x_i \in X_j\ and\ x_i \notin X_{j}, (j' \neq j)$$

Knapsack

TSP

哈密尔顿回路

- 有向或者无向图$G=(V,E)$,一个环路经过且仅经过图中的每个节点1次

Partition

- 给定有限集S和权重函数$w:S\to N$，确定是否存在子集$S'\subseteq S$使得

$$\sum_{a\in S'}w(a)=\sum_{a\in S-S'}W(a)$$

Lecture14-15-approximation

- 一个算法是一个优化问题的$\alpha-$近似算法，如果:
  该算法在多项式时间内运行
  该算法产生的解决方案总是在最优解决方案的一个系数α之内。
- 对于最小化问题,$\alpha>1$

$$A(I)\leq\alpha\cdot OPT(I)$$

- 对于最大化问题,$\alpha<1$

$$A(I)\geq\alpha\cdot OPT(I)$$

- 理解：理论$OPT(I)$最小，你的近似算法$A(I)$不一定能算出最小，所以至少比最小的大，这个系数α则给出了我们的上界；反之亦然

Load Balancing

- 输入：m台相同的机器；n个工作，工作j有处理时间t_j。
  工作j必须在一台机器上连续运行。
  一台机器一次最多可以处理一个作业。

  定义。假设J(i)是分配给机器i的工作子集，机器i的负载为$L_i=\sum_{j\in J(i)}t_j$
  定义。makespan是任何机器上的最大负载$L=max_iL_i$
  负载平衡：将每个作业分配到一台机器上，使makespan最小。
- 近似算法：贪心，找当前最小负载的机器，给它分配工作

  结合优先队列，需要时间$O(n\log n)$
- 贪心算法是2-近似算法

节点覆盖

- 给定无向图，从中取出一些节点，使得对于任意给定的边，都会和至少一个节点相连
- 近似算法：贪心，寻找当前度最大节点，加入集合
- 贪心算法是2-近似算法