算法导论笔记整理

berry

2022.3.5

目录

1 一些其他的东西

1.1 斯特林公式 Stirling formula

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

【为做题，可以简记为：$n! \sim \left(\frac{n}{e}\right)^n$】【斯特林公式!】【非常重要！赶紧补！背!】

欧拉常数 γ

$$\gamma := \lim_{n\to\infty}\left(\sum_{i=1}^{n}\left(\frac{1}{i}\right) - \ln(n)\right)$$

由此可知 $\sum_{i=1}^{k} \sim \Theta(\lg k)$。

1.2 还有其他的东西

【第一次作业（主定理）中用了不少暴力的求和，整理一下总结在这里】

2 主定理 Master Theorem

2.1 主定理

主定理 设 $a \geq 1$ 和 $b > 1$ 是常数，$f(n)$ 是一个函数，$T(n)$ 是定义在非负整数上的递归式：

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

则

1. 若对某个常数 $\epsilon > 0$ 有 $f(n) = O(n^{log_b a - \epsilon})$，则 $T(n) = \Theta(n^{log_b a})$
2. 若 $f(n) = \Theta(n^{log_b a})$，则 $T(n) = \Theta(n^{log_b a} lgn)$
3. 若对某个常数 $\epsilon > 0$ 有 $f(n) = \Omega(n^{log_b a + \epsilon})$，且对某个常数 $c < 1$ 有 $af(n/b) \leq cf(n), n \gg 1$，则 $T(n) = \Theta(f(n))$

对应到分治算法，$T(n)$ 是规模为 n 的问题的时间复杂度，a 是划分出的 n/b 规模的子问题的个数，$f(n)$ 是分解 (divide) 和合并 (combine) 的开销。对应到递归树上看，(3) 对应子节点开销占主要的问题，(1) 对应父节点（分解和合并）开销占主要的问题，(2) 对应子问题和分合开销相同的问题。

【主定理的证明跳过了】

3 分治法 Divide-and-Conquer

分解 devide，解决 conquer，合并 combine

3.1 矩阵乘法

给定两个同型方阵 $A_{n\times n}$ 和 $B_{n\times n}$，求它们的乘积 $C = AB$

3.1.1 一般方法

三层循环。时间复杂度为 $\Theta(n^3)$。

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

3.1.2 简单的分治算法

考虑将矩阵分块：

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},\ B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},\ C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

则原式转化为：

$$\begin{cases} C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ C_{12} = ... \\ C_{21} = ... \\ C_{22} = ... \end{cases}$$

原问题被拆分为 8 次 $n/2 \times n/2$ 的矩阵乘法和 4 次 $n/2 \times n/2$ 的矩阵加法。于是有：

$$T(n) = 8T(\frac{n}{2}) + \Theta(n^2) \tag{3.1}$$

由主定理，易得此分治法的时间复杂度仍为 $\Theta(n^3)$

3.1.3 Strassen 算法

考虑方程(1)，由主定理可知，限制其时间复杂度的主要因素在 $log_b a = log_2 8 = 3$。如果可以减少乘法运算的数量（对应 a），就可以降低算法的复杂度。

Strassen 提出一种只需要 7 次矩阵乘法的算法。他对 A 和 B 的分块矩阵做了一通操作，最终将计算改写为 7 次 $n/2 \times n/2$ 的矩阵乘法和 18 次 $n/2 \times n/2$ 的矩阵加法。详情可参考书 P45-47 或课件 lecture2(36/44)。

这样，递推公式变为

$$T(n) = 7T(\frac{n}{2}) + \Theta(n^2) \tag{3.2}$$

时间复杂度变为 $\Theta(n^{log_2 7=2.81})$

需要注意两点:

- 这个算法的矩阵加法次数明显增多了，因此复杂度上的常数比较大，在 n 较小时，效果差于 “简单的分治算法”
- 目前已经提出 $\Theta(n^{2.376})$ 的算法，但是它并不实用

3.2 多项式乘法

给定 $A(x)$ 和 $B(x)$ 两个 n 阶多项式，求其乘积。

$$A(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$$
$$B(x) = b_0 + b_1 x + b_2 x^2 + ... + b_n x^n$$

按照类似3.1的思路，可以做多项式的乘法。

3.2.1 简单做法

$$C(x) = A(x)B(x) = c_0 + c_1 x + c_2 x^2 + ... + c_{2n} x^{2n}$$

其中

$$c_i = \sum_{k=0}^{i} a_k b_{i-k},\ i = 1, 2, ..., 2n$$

上式称为**卷积**。

易见常规算法是 $\Theta(n^2)$ 的。

怎么实现这个卷积呢？可以固定 $b_0, b_1, ..., b_n$, 将 $a_n, ..., a_0$ 在 $b_0, ..., b_n$ 上滑动，每次滑动一格且计算乘积，求和。

3.2.2 分治做法

考虑

$$A(x) = (a_0 + ... + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}) + x^{\frac{n}{2}}(a_{\frac{n}{2}} + ... + a_n x^{\frac{n}{2}}) := P(x) + x^{\frac{n}{2}} Q(x)$$
$$B(x) = (b_0 + ... + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}) + x^{\frac{n}{2}}(b_{\frac{n}{2}} + ... + b_n x^{\frac{n}{2}}) := R(x) + x^{\frac{n}{2}} S(x)$$

将 $A(n)$ 和 $B(n)$ 分割为两部分，每部分的规模为 $n/2$。这样

$$C(x) = P(x)R(x) + (P(x)S(x) + R(x)Q(x))x^{\frac{n}{2}} + Q(X)S(x)x^n$$

可得递推关系式为 $T(n) = 4T(n/2) + \Theta(n)$。由主定理，算法是 $\Theta(n^2)$ 的。

3.2.3 改进分治做法

同3.1，上述分治算法没有降低复杂度的原因是：规模为 $n/2$ 的子问题较多。考虑减少 $n/2$ 的乘法操作，即可降低复杂度。

$$(P+Q)(R+S) = PR + (PS+QR) + QS$$

则只需计算 $(P+Q)(R+S), PR, QS$ 即可。$T(n) = 3T(n/2) + \Theta(n)$，复杂度降为 $\Theta(n^{log_2 3 \approx 1.59})$。当然常数（加减法操作）增大了。

3.2.4 更快的算法 (FFT)

用快速傅里叶变换 (FFT)，可以实现 $\Theta(nlgn)$ 的复杂度。详见这个链接

3.3 归并排序

【分成两半，各自排序，最后整合】

3.4 快速排序

【选定哨兵，小于哨兵放左边，大于哨兵放右边，然后左右两边分别递归快排】
【最坏情况下，每次哨兵都是最小 or 最大，则复杂度变成 $O(n^2)$ 了。随机算法可以规避最坏情况】

4 随机算法

4.1 矩阵乘法验证

给定三个同型方阵 $A_{n\times n}$，$B_{n\times n}$ 和 $C_{n\times n}$，验证是否有 $AB = C$
lecture2(44/44)

4.2 规避快速排序最坏情况

5 排序算法归纳

5.1 插入排序

这是第一次课讲的，就是引入算法复杂度用的。插入排序本身也非常简单，这里贴个伪代码就好了。

```
for j=2 to n
    key = A[j]
    i = j-1
    while (i>0 and A[i] > key):
        A[i+1] = A[i]
        i = i-1
    A[i+1] = key
```

5.2 归并排序

归并排序使用的是分治法的思想。

借助这种思想，还有二分查找【lec2 26/44】

5.3 堆排序

5.4 快速排序

5.4.1 规避快排的最坏情况

只要是比较排序，时间最优也是 $O(n \lg n)$【决策树。证明：lec4 11/60（$n!$ 个叶子节点的二叉树。深度 h 的二叉树有最多 2^h 个叶节点。所以 $2^h \geq n!$，得到 $h = O(n \lg n)$)】

5.5 线性时间排序

【这个标题不靠谱。线性时间排序就是顺序统计量啊】

5.6 顺序统计量

1) 找最大最小。需且仅需 $n-1$ 次比较。$O(n)$ 时间。

2) 同时得到最大最小。【考虑淘汰赛机制】每两个数一组，组内大者与 max 记录做比较，小者与 min 记录比较。这样有 n/2 组，每组有 3 次比较，共 3n/2 次比较【可以证明这是最少的比较方法?】

3) 找最大和次大。【考虑淘汰赛机制】找出最大者 n 次比较，有一棵二叉树。而次大者一定是和这个最大者比较过的元素，故只需检查最大者一路上来“打败”过的元素即可，有 lgn 次比较

4) 找第 k 大。【方法 1：快排（可以用随机版本。因为随机版本规避了最坏情况），同时每次只用递归一侧。$O(n)$ 的。方法 2：最坏情况下线性时间的顺序统计量算法（一组 5 个取中间，各组中间取中间的算法)】

【关于最坏情况线性时间的算法，只要分析最后两个子问题 T() 加起来系数不超过 1（要严格小于 1，如果不仔细考虑下取整和放缩的话），就是 O(n) 的。要是大于等于 1，则不是 O(n) 的。】

6 散列表 Hash Table

每个数据有唯一的 key 与之对应。所以我们存储数据时，只考虑存储 key 即可，通过 key 和数据的对应关系即可确定数据。

字典操作指插入、删除、查找这三个操作。

对直接寻址表而言，字典操作最坏情况下是 $O(1)$ 的。散列表能够在平均时间上保证字典操作是 $O(1)$ 的。散列表克服了直接寻址表的两个缺点：

1. 当全域 U（key 的可能取值）很大时，计算机可能存不下整张规模为 $|U|$ 的表

2. 实际存储用到的 key 集合 K 可能远小于 U，直接寻址表造成很大的空间浪费

因此，我们考虑散列表/哈希表。假设一共有 n 个可能的 key（键），哈希表中有 m 个槽。在本节中，我们总是假设 $n > m$，否则还不如用直接寻址表。

定义**装载因子** $\alpha = n/m$，表示平均哈希进每个槽的 key 的个数（如果采用链接法表示，α 就是一个链的平均存储单元数）。

使用链表法存储时，如果使用双向链表，则删除一个元素的代价是 $O(1)$ 的。（“删除”是指确定元素位置后，把该元素对应的数据单元删去的操作。确定元素位置属于“查找”操作）

6.1 简单均匀哈希假设

哈希表的平均性能取决于使用的哈希函数。哈希函数在下一节讨论。这里提出简单均匀哈希假设：假设每个 key $k \in K$ 都等可能地哈希进任意槽中。

在简单均匀哈希假设下，一次不成功查找的平均时间是 $\Theta(1+\alpha)$，一次成功的查找的平均时间也是 $\Theta(1+\alpha)$。【见书 P145-146】当 $n = \Theta(m)$ 时，这个代价是 $\Theta(1)$

所以，当散列表中槽数 m 与元素数 n 成正比，且采用双向链表解决冲突时，插入和删除在最坏情况下都只需要 $O(1)$ 时间。

6.2 哈希函数

6.2.1 除法哈希（取模）

$$h(k) = k \mod m \tag{6.1}$$

这里 m 应避免取 2 的幂次，因为这样相当于只使用到 key 值的低若干位。一般 m 的选取规则是：选一个不太接近 2 或 10 的幂次的素数。

有时哈希表规模为素数可能不方便。下面的乘法哈希通常效果更好，但这种取模哈希使用最为广泛。

6.2.2 乘法哈希

令 $m = 2^r$，即哈希表有 2^r 个槽。设计算机使用 w 位的字，取 A 为满足 $2^{w-1} < A < 2^w$ 的奇数。

$$h(k) = (A \times k \mod 2^w) >> (w - r) \tag{6.2}$$

注意：A 的选取不要太靠近 2^w。【这里乘法哈希和书上的公式形式不太一样，但是意思是一样的（书 148）】乘法哈希的速度：1) 模 2^w 计算很快；2) 右移 $w - r$ 位计算很快。

6.2.3 点积哈希

令 m 为素数。将键 k 分割为 $r+1$ 个数，每个数都落在 $\{0, 1, ..., m-1\}$ 范围内。即 $k =< k_0, k_1, ..., k_r >$，其中 $0 \le k_i < m$。令 $a =< a_0, a_1, ..., a_r >$，其中 a_i 是在 $\{0, 1, ..., m-1\}$ 中的随机数。定义哈希函数

$$h_a(k) = \sum_{i=1}^{r} a_i k_i \mod m \tag{6.3}$$

点积哈希有非常好的性质，但是计算复杂度高一些。在后面构造全域哈希函数簇的时候用到它了。

前面这些方法都有一个问题：可以构造一组数据 $\{x_n\}$，使得 $h(x_i)=t,\forall x_i$，这样散列表的性能就达到最差的 $O(n)$ 级别。
应对方法：引入随机。这样攻击者不知道选择的是哪个哈希函数，就无法设计上述攻击。

6.2.4 全域哈希 universal hashing

全域哈希内容比较多，用一个 subsection 来讲

6.3 全域哈希 universal hashing

Definition 令 U 为所有 key 的集合，H 是一组有限个数的哈希函数。每个 $h\in H$ 都是 U 映射到 $\{0,1,2,...,m-1\}$ 的函数。若对任意 $x,y\in U, x\neq y$，存在 $|\{h\in H: h(x)=h(y)\}|\leq |H|/m$，则称 H 是**全域的**。亦即，H 里任取一个哈希函数 h，任意两个不相等的 x 和 y，哈希时发生碰撞的概率不超过 $1/m$。

Theorem 令 h 是从 H 中均匀随机地选出的一个哈希函数，则对任意给定 x，与 x 出现碰撞的次数的期望 $<n/m$。即 $E(\#collisions\ with\ x)<n/m$，这里 # 表示次数。

证明：【来不及再打一遍了，见 ppt lec4-1 23/40 起】【证明结果发现 E(#..) = (n-1)/m】

6.3.1 一个全域哈希函数簇（基于点积哈希）

令 m 为素数。将键 k 分割为 $r+1$ 个数，每个数都落在 $\{0,1,...,m-1\}$ 范围内。即 $k=\langle k_0,k_1,...,k_r\rangle$，其中 $0\leq k_i<m$。令 $a=\langle a_0,a_1,...,a_r\rangle$，其中 a_i 是在 $\{0,1,...,m-1\}$ 中的随机数。定义哈希函数

$$h_a(k)=\sum_{i=1}^{r}a_ik_i \mod m$$

注意，这每一个哈希函数就是点积哈希。它们构成的哈希函数簇 H 共有 m^{r+1} 个函数。

Lemma 令 m 为素数。对任意 $z\in Z_m, z\neq 0$，存在一个唯一的 $z^{-1}\in Z_m$ 使得 $z\dot{z}^{-1}\equiv 1(\mod m)$。

证明：因为 Z_m^* 是一个乘法群，所以显然。详细证明需要群的知识，建议回顾代数结构课笔记（笑）。

Theorem 哈希函数簇 $H=\{h_a\}$ 是全域的。

证明：【26/40】要用到上面的引理。证明了对一组确定的 $a_1,a_2,...,a_r$，只有唯一存在的一个 a_0 会使得 x 和 y 冲突。这样 H 就满足全域的定义了。

6.3.2 另一个简单的全域哈希函数簇（同书 150）

令 p 是一个大于 n 的素数（即能包含所有元素的键），定义哈希函数

$$h_{a,b}(x)=((ax+b) \mod p) \mod m \tag{6.4}$$

函数簇 $H_{p,m}$ 定义为

$$H_{p,m}=\{h_{a,b}: 1\leq a<p, 0\leq b<p\}$$

即 $a \in Z_p^*, b \in Z_p$。可见这个函数簇中共有 $p(p-1)$ 个函数。$H_{p,m}$ 还有一个优点是哈希表的规模 m 可以是任意值，不一定要是素数。这一点将在下面的证明中看到。

Theorem $H_{p,m}$ 是全域的。

证明：【31/40 起】

6.4 完美哈希/完全哈希 perfect hashing

是一种静态的数据结构，即一旦键存入表中，键集合就不会再变化。完美哈希能提供最坏情况下 $O(1)$ 的搜索操作。【差完美哈希，开放寻址法，一致性哈希，即可解决本章。至于 cuckoo，bloom，counter-min，大概率不考，不看了】

【完美哈希看完了，证明部分简单过的。大致记得怎么回事吧】

6.5 开放寻址法

【探查序列】【线性探查、二次探查、双重哈希/双重散列】【还没仔细看，6.14 来看】

6.6 一致性哈希 consistent hashing

7 摊还分析 Amortized Analysis

7.1 聚合分析

7.2 核算法

【要保证剩的钱数始终非负】

7.3 势函数法

【要保证和初始状态相比，势变化量始终非负】

8 动态规划 Dynamic Programming

【这一章的笔记都是瞎整理的！时间不够了】

8.1 例子

有向无环图的最短路 $dist(v) = \min_{(u,v)\in E}\{dist(u) + d(u,v)\}$【最长路径也是类似】

【动态规划问题都相当于是有向无环图问题。且一旦依赖关系成环，dp 就失效了】

最长递增子序列 $L(j) = \max_{(i,j)\in E}\{L(i)+1\}$。$L(j)$ 表示以 j 结尾的最长递增子序列长度。【我记得作业有一个类似的，但是要返回最长的这个序列，而不仅是序列的长度。要去看一下】

8.2 带权重的区间调度

每个任务有起始时间、结束时间、价值 v。要使做的任务的总价值最高（只有一个人在做任务）。【方法：贪心、枚举、动态规划】

所有任务按结束时间从早到晚排序，记为任务 1,2，...。对任务 j，记 p(j) 是最后一个和 j 任务不冲突的任务。则 $d[0]=0, d[j]=\max\{v_j+d[p(j)], d[j-1]\}$

8.3 解法

1) 暴力递归调用。指数级的复杂度，大量子问题被重复计算

2) 备忘录。即我们平时用的打表【表打好后，$O(1)$ 返回任何数据的最优解】

3) 自底向上。即根据要什么，去求对应需要的那些，递归直到最小问题，即可逐层返回。【只针对我们要的数据的最优解计算，沿途算出来的一些最优解都是顺带的。但是自底向上可以一路保留走过来的路径，e.g. 能够给出串具体是什么，而不是只能给出串的长度】

对带权重的区间调度问题，备忘录是 $O(n\lg n)$ 的

8.4 最长公共子串

$$d[i][j]=\begin{cases} d[i-1][j-1]+1, & if\ x[i]=y[j] \\ \max\{d[i][j-1], d[i-1][j]\}, & otherwise \end{cases}$$

8.5 背包问题

令 $d[k][b]$ 表示只使用物品 $1,...,k$，在背包容量不超过 b 的情况下的最优解。s 表示物品大小，v 表示物品价值。

$$d[k][b]=\begin{cases} 0, & if\ k=0 \\ d[k-1][b], & if\ s_k>b \\ \max\{v_k+d[k-1][b-s_k], d[k-1][b]\}, & otherwise \end{cases}$$

8.6 带权独立集（WIS）问题

最大的点集子集，使得这个子集中的任意两个顶点在原图中不相邻。

这个问题是 NP 难的。故我们这里只考虑图是一个树的情况。

$d[u]=\max\{\sum_{v\in C(u)} d[v], w_u+\sum_{C(C(u))} d[v]\}$

8.7 旅行商问题

dp 算法的复杂度是 $O(n^2 2^n)$。暴力算法是 $O(n!)$ 的。

令 $C(S,j), j\in S$ 表示从 1 出发经过 S 中所有点最后到 j 的最短路长度。$C(S,j)=\min\limits_{i\in S-\{j\}}(C(S-\{j\},i)+d_{ij})$

8.8 最优二分查找树

9 贪心法

9.1 证明贪心的正确性

1) stay ahead。证明每一步的结果总是不差于最优解在这一步的结果

2) 上下界。证明贪心算法达到了上/下界

3) 交换论证。证明最优解经过一些不使结果更坏的变换，可以到达贪心解

9.2 区间调度问题

【回顾：有 dp 算法】【这里假设权重都为 1. 则有更快的算法：贪心】

【qh 6.15 在群里发的图片：贪心是一种限制条件更加严格的情况下，时间复杂度低于 dp 的最优化问题解法。贪心和动规本质上是不排斥的，都是最优化问题的解法】

解法：最早结束时间优先。按结束时间从早到晚排序，按次序选择任务（当然每个任务加进来以后，和它冲突的任务就都不能选了）

证明：stay ahead

9.3 区间分解/染色

【把若干任务分配到最少数量的处理器上/把若干课程安排到最少数量的教室里】

算法：最早开始时间优先。

证明：贪心算法到达了下界。【定义了“深度”的概念。某时刻同时运行的任务数称为这个时刻的深度】

9.4 最小化延迟问题

【单处理器。每个任务有最晚完成时间 d（但是可以超时，超时部分算延迟）、运行耗时 t，设开始时间 s，结束时间 $f = s + t$（这是不准抢占的模型啊）】【定义任务 i 的延迟 $l_i = \max\{0, f_i - d_i\}$，要求最小化总延迟】

算法：最早 ddl 优先。

证明：交换论证。交换相邻的“逆序”任务【只要有逆序，一定有相邻逆序】

9.5 部分背包问题（之前的是 0-1 背包问题）

按照性价比从高到低选取即可

9.6 数据压缩

【定长编码，没有优化空间。我们考虑变长编码】

【霍夫曼编码】【前缀码。变长编码不能是前缀码。一棵不满的二叉树不可能对应一个最优编码】

10 最小生成树

10.1 图表示方法

【图表示方法：邻接矩阵啥的，lec8 5/63】

握手引理 $\sum_{v\in V} deg(v) = 2|E|$

10.2 生成树

一个定理 生成树的子图是生成树【类比：最短路的子路是最短路】

10.3 Prim 算法

【用贪心实现的算法】

【类似 Dijkstra 算法！时间复杂度也和 Dijkstra 一样。有三种数据结构的实现，对应复杂度不同。斐波那契堆的时间是 $O(V \lg V + E)$】

10.4 Kruskal 算法

【用并查集实现的算法】

【初始化，每个点是一个集合。每次找跨集合的边中最小权重的那个，合并两个集合】

11 并查集 disjoint set

并查集有三个操作：MAKE-SET(x)，FIND(x)，UNION(x,y)

11.1 链表实现

一条链表是一个并查集，链表的第一个节点作为集合的代表元。可见书 P326 图片。时间复杂度如下：

MAKE-SET	FIND	UNION
$O(1)$	$O(1)$	$O(n)$

11.2 森林实现

11.2.1 普通森林实现

11.2.2 按秩合并

11.2.3 路径压缩

11.3 按秩合并、路径压缩的时间复杂度（摊还分析）

MAKE-SET 和 UNION 操作都是 $O(1)$ 的。故我们只分析 FIND 的时间复杂度，考虑 m 次 FIND 的摊还代价。

11.3.1 准备工作

首先给出一些观察和引理。

Observation 1 对任意节点 x，以 x 为根节点的子树的深度/树高不超过 x 的秩 $rank(x)$

Observation 2 对所有非根节点（即 $parent(x) \neq x$），有 $rank(x) < rank(parent(x))$（即 $rank(x) \leq rank(parent(x)) + 1$）

Lemma 1 以 x 为根的树至少有 $2^{rank(x)}$ 个节点。

用数学归纳法证明。

1) $rank(x) = 0$ 时，即 x 是孤立点时，树有 $1 = 2^0$ 个节点。

2) 假设 $rank(x) = k$ 时成立，则 $rank(x) = k + 1$ 时：考虑到秩的增加只能发生在两棵同秩子树合并时，所以树 x 一定是从两棵 $rank = k$ 的子树合并而来，由假设知两棵子树都至少有 2^k 个节点，因此树 x 至少有 $2^{k+1} = 2^{rank(x)}$ 个节点。

综上 1)2)，得证。

Lemma 2 秩为 k 的节点至多有 $n/2^k$ 个，$\forall k \geq 0$。

证明：一个秩为 k 的节点代表一棵至少 2^k 个节点的子树，这样的子树至多 $n/2^k$ 个。

Lemma 3 所有节点的秩最大不超过 $\lg n$。

由 **Lemma 2** 易得，$n/2^k \geq 1$，故 $k \leq \lg n$。

Lemma 4 $rank \geq k$ 的节点至多 $n/2^{k-1}$ 个。

证明：$rank \geq k$ 的节点总数为 $\sum_{i=k}^{\lg n} n/2^i < \sum_{i=k}^{\infty} n/2^i \sim n/2^{k-1}$。

Observation 3 一个节点只要变为非根节点，它的秩就永远固定了（非根节点不可能再变成根节点）。

11.3.2 摊还分析（核算法）

对所有节点按照 $\log^* r$ 进行分组，$\log^* r$ 值相同的在一组。这里 r 是节点的秩，$\log^* r$ 是满足 $\underbrace{\lg \lg ... \lg}_{k\ times} r \leq 1$ 的最小的整数 k。亦即，第 i 组中的全部节点都满足 $2^{\overbrace{2^{\cdots^2}}^{i\ times}} < r \leq 2^{\overbrace{2^{\cdots^2}}^{(i+1)\ times}}$。将每一组记为区间 $(k, 2^k]$，其中 k 本身是 2 的迭代幂次。

Lemma 5 对上述分组而言，组数不超过 $\log^* n$。

因为 n 个节点的最大秩不超过 $\lg n$。

Lemma 6 在分组 $(k, 2^k]$ 中的节点数不超过 $n/2^{k-1}$。

因为秩超过 k 的节点数就不超过 $n/2^{k-1}$（**Lemma 4**）。

对分组 $(k, 2^k]$ 中的每一个节点，赋予它 2^k 块“钱”。对每个分组而言，赋予的总钱数不超过 $\frac{n}{2^{k-1}} \times 2^k = 2n$。发出去的总钱数不超过 $2n \log^* n$。

规定“付钱”规则如下：在运行 FIND 函数找过的路径上，每一条边 (u, v)（注：在路径压缩前，这个 v 就是 $parent(u)$），如果 u 和 v 在同一个分组中，则 u 来支付走过这条边的代价；如果 u 和 v 在不同的分组，则由 FIND 函数支付走过这条边的代价。

对所有非根节点 u，只要 FIND 函数经过 u 节点，就会把 u 的 $parent$ 改为当前的根节点。由 **Observation 2** 可知，只要原先的 $parent(u)$ 不是根节点，则路径压缩后 $parent(u)$ 的秩一定增加至少 1。又因为 u 所在的分组 $(k, 2^k]$ 区间长度为 $2^k - k \leq 2^k$，所以 u 这个节点至多支付 2^k 次（之后 $parent(u)$ 就和 u 不在一个分组了）。所以，我们分给每个节点的钱足够其支付 $parent(u)$ 不是根节点时的全部费用（$O(2n \log^* n)$）。而 $parent(u)$ 是根节点的情况在每次 FIND 操作中只会出现一次，对总共 m 次 FIND 操作，只要单独“补贴” m 块钱即可 $(O(n))$。由节点承担的总开销是 $O(2n \log^* n) + O(m)$。

由 **Lemma 5**，对每一次 FIND 操作，由 FIND 函数支付的代价不超过 $\log^* n$。所以 m 次 FIND 操作中由 FIND 函数支付的代价不超过 $m \log^* n$。

综上，m 次 FIND 操作总代价是 $O(2n \log^* n + m + m \log^* n) \sim O((m+n) \log^* n)$。一次 FIND 的均摊代价是 $O((1 + \frac{n}{m}) \log^* n)$，近似线性。

12 最短路算法

首先定义最短路权重

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \xrightarrow{p} v\}, & if\ have\ path\ from\ u\ to\ v \\ \infty, & else \end{cases}$$

> 注意如果有负回路，即某个回路的总权重为负，则一直在这个负回路上绕圈会使路径权重越来越小。因此讨论最短路问题时，需要一个基本假设：图中没有负回路。

12.1 基本定理

（最优子结构） 最短路径的子路径还是最短路径

反证法易证：否则原最短路径不是最短

（最短路的三角不等式） 对 u, v, x 三个节点，设 x 不在 u, v 的最短路上，则 $\delta(u, v) < \delta(u, x) + \delta(x, v)$

反证法易证：否则 $u \to x \to v$ 是 u, v 的更短路

12.2 图表示方法

1. 邻接链表。每个节点的全部邻边用链表串起来表示。lecture9[9/47]

2. 优先队列。优先队列可以用二叉堆、斐波那契堆等实现。lecture9[10/47]

12.3 单源最短路径问题

目标：给定一个节点 $s \in V$，要找出 $\delta(s,v), \forall v \in V$。

12.3.1 Dijkstra 算法

是一种贪心算法。仅适用没有负权重边的情况。

1. 维护集合 S，其中任意节点 $v \in S$，$\delta(s,v)$ 已确定

2. 每一步中，把 $u \in V-S$ 的 $dis(s,u)$ 最小的节点 u 放入 S 中

3. 更新所有和 u 有边相连的点 i 的 $dis(s,i)$

注意：$dis(s,u)$ 时刻表示的是“从 s 出发，经 S 中的若干个点，最后一步到达 u”的所有可能路中最短路的长度（权重）。如果从 S 到 u 没有直接相连的边，则 $dis(s,u)=\infty$。

【算法正确性的证明在 ppt 14/74】下面证明算法的正确性。用 $d(v)$ 简记 $dis(s,v)$。

Lemma $d(v) \geq \delta(s,v), \forall v \in V$ 始终成立。

证明：算法初始化时，$d(s)=0$，$d(v)=\infty, \forall v \neq s$；$\delta(s,s)=0$，$\delta(s,v) \leq \infty, \forall v$。引理成立。

假设 v 是第一个 $d(v) < \delta(s,v)$ 的节点，u 是导致 $d(v)$ 发生改变以至于不等式不成立的节点，即 $d(v)=d(u)+w(u,v)$。则

$$
\begin{aligned}
d(v) &< \delta(s,v) && supposition \\
&\leq \delta(s,u)+\delta(u,v) && triangle\ inequality \\
&\leq \delta(s,u)+w(u,v) && shortest\ path \leq specific\ path \\
&\leq d(u)+w(u,v) && v\ is\ the\ first\ violation
\end{aligned}
$$

即 $d(v) < d(u)+w(u,v)$，矛盾。所以假设不成立。所以引理成立。

Theorem 当 Dijkstra 算法停止时，$d(v)=\delta(s,v), \forall v \in V$。即算法的正确性。

【16/74】【这里简单写写。假设 u 是第一个加入 S 时 $d(u) \neq \delta(s,u)$ 的，则由 **Lemma** 可知 $d(u) > \delta(s,u)$。假设 s 到 u 的最短路 p，$(x,y) \in p$ 是 p 上第一条跨越 S 和 $V-S$ 的边，即 $x \in S, y \in V-S$。利用“最短路的子路是最短路”，推出 $d(y)=\delta(s,y) \leq \delta(s,u) \leq d(u)$，进而因为算法选择 u 而不是 y，说明 $d(y)=\delta(s,y)=\delta(s,u)=d(u)$，矛盾。】

【时间复杂度 20/74】

> 当图是无权重图时，可以使用 BFS 或 DFS 代替 Dijkstra，即用先进先出队列（普通队列）对 Dijkstra 的优先队列做改进。

12.3.2 广度优先搜索 BFS

12.3.3 深度优先搜索 DFS

如果不知道图中是否有负回路怎么办？Bellman-Ford 算法可以判定图中是否有负回路，如果没有负回路，可以返回最短路。

12.3.4 Bellman-Ford 算法

算法：【lec18 4/45】【初始化 $d(v)$ 和 Dijkstra 一样，然后迭代 $|V|-1$ 轮，每一轮中，以任意次序遍历所有边 (u,v)，如果 $d(v)>d(u)+w(u,v)$，则修改 $d(v)=d(u)+w(u,v)$。这每一轮称为一次“松弛”。经过 $|V|-1$ 轮松弛后，如果图中还有任何一条边 (u,v) 满足 $d(v)>d(u)+w(u,v)$，则图中有负回路，否则已经得到最短路（所有 $d(v)=\delta(s,v)$）。】

算法时间是 $O(VE)$ 的。

算法正确性【lec18 27/45】【暂时略过，来不及了。赌他考应用不考证明】【这个证明也很容易啊，我 5min 就看完了 hhh】

13 最大流算法

用 m 表示图 G 的边数，n 表示顶点数。

13.1 流

13.1.1 流的定义

满足三个性质：

1) 容量限制。$f(u,v)\le c(u,v),\forall u,v\in V$

2) 流量守恒。$\sum_{v\in V} f(u,v)=0,\forall u\in V-\{s,t\}$（中间点流量和为 0）。另一种形式为 $\sum_{v:(u,v)\in E} f(u,v)=\sum_{v:(v,u)\in E} f(v,u)$（流入 = 流出）

3) 反对称性。$f(u,v)=-f(v,u)$

流值 $|f|=\sum_{v\in V} f(s,v)=f(s,V)=f(V,t)$。若 u 和 v 之间没有路，则 $f(u,v)=c(u,v)=0$。

13.1.2 流抵消、流分解

流抵消 flow cancellation 如果有流 $u\xrightarrow{2:3}v$ 和流 $v\xrightarrow{1:2}u$，则可以抵消为流 $u\xrightarrow{1:3}v$ 和 $v\xrightarrow{0:2}u$。

流分解 flow decomposition 任意一个图 G 上的可行流 f 都可以被分解为至多 m 个环流或 $s-t$ 路径流。因为每分解出一个流，就至少消去这个流走过的路上的一条边（至少一条边上的流量降为 0）。

【这个看起来 trival，但是在后面证明 Edmonds-Karp 算法版本一、Dinic 算法的时候用处大大滴！】【lec10 9/51】

分解出的至多 m 个流中，只有 $s-t$ 路径流对流 f 的流值有贡献，所有环流都没有贡献 f 的流值。

13.1.3 流的性质（16/51）

Lemma 对 $G=(V,E)$ 上的流 f，令 $f(X,Y)=\sum_{x\in X,y\in Y} f(x,y)$，有：

1) $f(X,X)=0,\forall X\subset V$

2) $f(X,Y)=-f(Y,x),\forall X,Y\subset V$

3) $f(X\cup Y,Z)=f(X,Z)+f(Y,Z),\forall X,Y,Z\subset V$ 且 $X\cap Y=\emptyset$

【证明是容易的。没时间写了。】

13.2 残差网络/剩余网络 Residual Network

剩余容量定义为

$$c_f(u,v)=\begin{cases} c(u,v)-f(u,v), & if\ (u,v)\in E \\ f(u,v), & if\ (v,u)\in E \\ 0, & else \end{cases}$$

可见 $c_f(u,v)\geq 0$。

记图 $G=(V,E)$，则**残差网络/剩余网络** $G_f=(V,E_f)$，其中

$$E_f=\{(u,v)\in V\times V : c_f(u,v)>0\}$$

可见 $|E_f|\leq 2|E|$。

13.3 增广路径 Augmenting Path

若 G_f 中存在一条 $s-t$ 路径 p，则称这条路径 p 为 G 的**增广路径**。

若把增广路径添加到原图 G 上（即增广），可以将 G 的流值增加 $c_f(p)=\min_{(u,v}\in Pc_f(u,v)$。依据这个性质，可以设计 Ford-Fulkerson 算法找最大流。【注意，FF 算法只在有理数流量、容量的情况下能收敛，无理数情况下可能进入无限循环（lec10 15/51. 本质是有理数的确界可达，无理数确界不可达）】【这些在 15/51，没仔细看了】

13.4 割 cut

割 (S,T) 是图 G 的定点集合 V 的一个划分，$T=V-S$，$s\in S,t\in T$。**割的容量**定义为 $c(S,T)=\sum_{u\in S}\sum_{v\in T} c(u,v)$

直观上可以感受到，小容量的割会限制流的大小，使得流不能称为大容量流。实际上，这个感受可以由下面的引理得到。

Lemma 给定一个流 f，对任意割 (s,T)，有 $|f|=f(S,T)$。

证明：$f(S,T)=f(S,V)-f(S,S)=f(s,V)+f(S-s,V)=f(s,V)=|f|$

Corollary 对任意一个流，任意割的容量都是他的流值的上界。即对任意割 (S,T)，$f(S,T)\leq c(S,T)$。

证明：$|f|=f(S,T)=\sum_{u\in S}\sum_{v\in T} f(u,v)\leq \sum\sum c(u,v)=c(S,T)$

13.5 最大流最小割定理 Max-flow Min-cut Theorem

(最大流最小割定理) 下面三个命题等价:

1) f 是 G 的最大流。

2) 残差网络 G_f 不含任何增广路。

3) G 中存在某个割 (S,T), 满足 $|f|=c(S,T)$。

下面证明。

$1\Rightarrow 2$。假设存在一个增广路, 则沿这条增广路一定可以对 f 增广, 与 f 是最大流矛盾。

$3\Rightarrow 1$。任意割的容量都是 $|f|$ 的上界, 即 $|f|\le c(S,T), \forall S\cup T=V, S\cap T=\emptyset$。所以 $|f|=c(S,T)$ 的情况下, 不存在更大的流。即 f 是最大流。

$2\Rightarrow 3$。令 $S=\{v: s\to v \text{ 在 } G_f \text{ 中有路 }\}$, $T=V-S$。因为残差网络 G_f 不含任何增广路, 所以 $s\in S$, $t\in T$, 即这样的 (S,T) 是一个割。对任意 $u\in S, v\in T$, 有 $c_f(u,v)=0$ (否则 $v\in S$), 所以 $f(u,v)=c(u,v)$。所以 $|f|=f(S,T)=c(S,T)$。

13.6 Ford-Fulkerson 算法

```
Ford-Fulkerson()
01    let f(u,v) = 0 for each node u,v
02    while there exist augmentation path p in G_f
03          augment f by c_f(p)
```

注意: FF 算法是一类算法的统称。对于找增广路 f, 可以用 BFS, 也可以用 DFS。

13.6.1 算法正确性

FF 算法结束时, G_f 中已经没有增广路。由最大流最小割定理可知, 算法得到最大流。

13.6.2 时间复杂度

找每一条增广路, 不论 BFS 还是 DFS, 用时 $O(E)$。每条增广路为流值带来至少 1 的增加 (而且可能产生每个增广路只增加 1 的情况), 所以总用时 $O(E|f^*|)$。其中 f^* 是最大流。

这个复杂度不是 $|V|$ 或 $|E|$ 的多项式时间。如果最大流很大, 时间复杂度会很高。

一个每次增广流值只增加 1 的例子: 【27/51】

13.7 Edmonds-Karp 算法

用广度优先改进 Ford-Fulkerson 算法。版本一用到 Dijkstra 算法, 本质上也是一种广度优先。

13.7.1 版本一

> 思路: 每次都找容量最大的增广路。这样可以避免一次增广只增加 1 的情况。

修改 Dijkstra 算法, 使得每次都找最大容量的增广路 (即把 "最短路" 改成 "最长路", 算法本质不变)。

Claim 1 在最大流为 F 的图中，一定存在一条容量至少为 F/m 的 $s-t$ 路径。

证明：由流分解定理可知，流 F 至多分解成 m 条环流或路径流。则由鸽笼原理，得证。

Claim 2 Edmonds-Karp#1 算法至多进行 $O(m \lg F)$ 次迭代。

证明：设迭代次数为 k。由 **Claim 1** 可知，一次迭代后，G_f 剩余的流不超过 $F - F/m$；k 次迭代后，剩余不超过 $(1-1/m)^k \times F$。因为 $1 - x \le e^{-x}$，所以 $(1-1/m)^k \le e^{-kx}$。要经过 k 次使得 $(1-1/m)^k * F = 1$，则 $k \sim m \lg F$。

如果使用斐波那契堆，则每次 Dijkstra 算法时间是 $O(m \lg n)$ 的。结合 **Claim 2** 可知，Edmonds-Karp#1 总时间是 $O(m^2 \lg n \lg F)$。

13.7.2 版本二

> 思路：每次都找最短的增广路。这样可以降低找增广路的开销。【这也是书上的 EK 算法】

使用 BFS 算法找增广路，这样每次找到的都是最短的增广路。

Claim 增广的次数不超过 $\Theta(VE)$。

证明：这个 Claim 的证明比较复杂，需要引入层次图概念和单调性引理。这两个概念在下一小结介绍。这里直接使用单调性引理，来证明这个 Claim。

令 p 为一条增广路。定义**关键边**为增广路 p 中残差容量恰好等于路的残差容量的边，即关键边 $(u,v) \in p$ 满足 $c_f(p) = c_f(u,v)$。亦即，关键边是整条路 p 中残差容量最小的边。则如果这条边 (u,v) 在一轮增广时被增广，那么下一轮增广时，(u,v) 不会出现在 G_f 中，而是 (v,u) 会以之前 $c_f(u,v)$ 的残差容量出现在 G_f 中。当 (v,u) 在某轮增广时被增广之后，下一轮增广时 (u,v) 边才会出现在 G_f 中，也才有可能再次成为关键边。

假设这一轮增广使用 p 为增广路。由层次图的介绍可知，在 p 被增广前，对关键边 (u,v)，有 $\delta(v) = \delta(u) + 1(1)$。由上面一段分析知，对 p 增广后，只有 (v,u) 出现在 G_f 中，而 (u,v) 不会出现，直到 (v,u) 被增广后 (u,v) 才会再出现。考虑 (v,u) 被增广前，有 $\delta'(u) = \delta'(v) + 1$。由单调性引理可知，$\delta'(v) \ge \delta(v)$，再结合 (1) 式，有

$$\begin{aligned}\delta'(u) &= \delta'(v) + 1 \\ &\ge \delta(v) + 1 \\ &= \delta(u) + 2\end{aligned}$$

所以在边 (u,v) 两次成为关键边的过程中，$\delta(u)$ 至少增加了 2。则如果一条边 (u,v) 成为关键边 k 次，$\delta(u)$ 至少增加了 $2(k-1)$。

因为对每个点 u，$\delta(u)$ 从 ≥ 0 开始，保持单调不减，且保持 $\le |V|-1$（共 $|V|$ 的点，层次图最深也只能有 $|V|-1$ 层）。所以对任意边 (u,v)，它至多成为关键边 $\dfrac{|V|-1}{2} + 1 \sim \Theta(V)$ 次。

又因为每轮迭代有一条关键边，一共有 $|E|$ 条边，所以共迭代至多 $\Theta(VE)$ 次。

至此得证。

因为 BFS 每次找增广路是 $\Theta(V+E) = \Theta(E)$ 的，结合 **Claim** 可知，Edmonds-Karp#2 算法时间是 $\Theta(VE^2)$ 的。

13.7.3 层次图和单调性引理

从 s 出发，到 v 的路径的边数为 i，则点 v 归入层次图的第 i 层。如果从 s 到 v 有多条路径，则 i 取边数最少的那条路径的边数。这样构建了从 s 到 t 的层次图，每个点都在一个唯一的层次中。【30/51】同一层内的所有节点之间的边全部去除，深层到浅层的边也去除，只剩下钱层到深层的单向边。

实际上 BFS 产生的就是这样的层次图。每个点被归入哪个层次，是由 BFS 最先遇到这个点的时间决定的，即由边数最短的路径决定的；同一层内的点互相之间的边，以及深层到浅层的边，在 BFS 时会被忽略。所以 Edmonds-Karp#2 算法用 BFS 找增广路，其实本质上是在这样的层次图上取层次深度最小的 $s-t$ 路径。

单调性引理 令 $\delta(v)=\delta_f(s,v)=G_f$ 中 s 到 v 的广度优先距离（即 s 到 v 的最短路径长度，亦即层次图中所在的层次数/深度）。则在 EK 算法运行过程中，$\delta(v)$ 单调不减。即，设流 f 增广为 f'，则 $\delta'(v)\geq\delta(v)$。

用数学归纳法证明。

1) $\delta'(s)=\delta(s)=0$

2) 考虑 $G_{f'}$ 中一条 s 到 v 的广度优先路 $s\to ...\to u\to v\to ...\to t$，假设 v 是这条路上第一个 $\delta'(v)<\delta(v)$ 的节点，下面来推出矛盾。

因为 1) 中证明了 s 不会是这样的 v，所以在 $s\rightsquigarrow v$ 的路上，v 前一定有一个节点 u。由假设，节点 u 已经满足 $\delta'(u)\geq\delta(u)$，且在这条路中，$\delta'(v)=\delta'(u)+1$，$(u,v)\in E_{f'}$。下面分类讨论增广前 (u,v) 是否在 E_f 中：

2.1) $(u,v)\in E_f$。则由最短路的三角不等式（最短路算法那一章“基本定理”部分）和归纳假设，有 $\delta(v)\leq\delta(u)+1\leq\delta'(u)+1=\delta'(v)$

2.2) $(u,v)\notin E_f$。则增广路中一定含有边 (v,u)。则 $\delta(v)=\delta(u)-1\leq\delta'(u)-1=\delta'(v)-2<\delta'(v)$

综上 2.1)2.2)，$\delta'(v)\geq\delta(v)$，这与假设矛盾。所以假设不成立，所以 $\delta'(v)\geq\delta(v)$。

综上 1)2)，数学归纳法证明结束。

13.8 Dinic 算法

Dinic 算法对 Edmonds-Karp#2 算法进行了改进。EK 算法每轮迭代，总是要从 s 出发进行 BFS，即每次都要回到层次图的起始点（0 层）。Dinic 算法引入了阻塞流的概念，使得不必每次寻找增广路都从 s 开始。

【没时间一点一点打了，见 ppt 吧。本质上，EK 每次增广都回到 s 重新找路，而 Dinic 只是在路上回退，这样在增广不导致层次图层数加深的情况下，Dinic 比 EK 节省了代价。当层次为 k 的层次图上没有增广路的时候，就要重新构造 G_f，这和 EK 是一样的，此时 Dinic 从 s 开始找路了。但是要知道，在层次为 k 的图上还有增广路的时候，EK 每次都重构 G_f 和整个层次图，但是仍然会找出一条深度为 k 的路，所以实际上它的这些重构层次图的操作是没有意义的，这也是 Dinic 比 EK 省时间的地方所在】

【Dinic 的一轮（构成一次阻塞流）代价是 $O(mn)$ 的，每次形成阻塞流都会使层次图深度至少 $+1$，所以总代价还是 $O(mn^2)$ 的。可能只是常数小一点？但是如果用比较好的数据结构，可以使构成一次阻塞流的代价是 $O(m\lg n)$】

13.8.1 阻塞流

13.8.2 Dinic 算法

13.9 单位容量图

单位容量图是每条边的容量都是 1 的图。即 $c_e = 1, \forall e \in E$。

Lemma 1 Dinic 算法在单位容量图上迭代次数为 $O(\min(m^{1/2}, 2n^{2/3}))$。

证明：分别从边和点的角度，得到了 min 中的两组数据。

1) 从边的角度。由容斥原理，当 s 到 t 的增广路长度已经 $\geq m^{1/2}$ 时，G_f 对应的层次图中一定存在相邻的两层，它们之间的边数（正向边数 + 反向边数）小于 $m^{1/2}$（否则总边数超过 m 了）。此时，因为每条边的容量是 1，所以 G_f 的最小割的容量 = 正向边数 − 反向边数 $\leq m^{1/2}$，所以 G_f 的最大流 $\leq m^{1/2}$。所以，当深度已经超过 $m^{1/2}$ 时，还需要不超过 $m^{1/2}$ 次迭代（即使每次增广流只增加 1，每次迭代只增广一次，迭代次数也 $\leq m^{1/2}$）。而深度到达 $m^{1/2}$，需要至多 $m^{1/2}$ 次迭代（由单调性引理，每次迭代深度都至少加 1），所以总共不超过 $2m^{1/2}$ 次迭代。

2) 从点的角度。当深度 $\geq 2n^{2/3}$ 时，一定存在相邻两层，它们的节点数都 $\leq n^{1/3}$（否则至少每隔一层就有一层节点数 $> n^{1/3}$，则至少 $1/2 \times 2n^{2/3} = n^{2/3}$ 个节点数 $> n^{1/3}$ 的层，总节点数超过 n 了）。则它们之间的边数不超过 $2n^{2/3}$（包括正向的至多 $n^{2/3}$ 条边和反向的至多 $n^{2/3}$ 条边）。类似 1) 分析可知，深度 $\geq 2n^{2/3}$ 时，还需要迭代次数不超过 $2n^{2/3}$。同时深度达到 $2n^{2/3}$ 需要迭代至多 $2n^{2/3}$ 次。所以总迭代次数不超过 $2n^{2/3} + 2n^{2/3} = 4n^{2/3}$) 次。

综上 1)2)，总迭代次数为 $O(\min(2m^{1/2}, 4n^{2/3})) = O(\min(m^{1/2}, 2n^{2/3}))$。

Lemma 2 在单位容量图中，可以在 $O(m)$ 时间内找到一个阻塞流。

证明：因为所有边的容量都是 1，所以在层次图中每找到一条增广路，整条路上的所有边都是关键边，增广后都会消失（反向不会被加入层次图，所以每次增广在层次图中的效果仅仅是关键边消失）。则每条边只能被增广一次，共 m 条边，且增广一条边的代价是常数的，所以至多 $O(m)$ 的代价。

Theorem 在单位容量图上，可以在 $O(m\min(m^{1/2}, 2n^{2/3}))$ 时间找到最大流。

证明：由 **Lemma 1** 和 **Lemma 2**，用 Dinic 算法即可。

13.10 网络流的应用

13.10.1 非交路径 edge disjoint path

要找最大非交路径数【lec11 7/41】可使用单位容量图的最大流建模。

Theorem 单位容量图的最大流流值和最大非交路径数相等。

13.10.2 二分图匹配 bipartite matching

要找最大匹配【11/41】我们讨论二分图的匹配问题。

建模：起始点 s 到所有左节点，左节点到右节点边变成单项边，所有右节点到 t。边权重：L 到 R 的边都是 ∞，$s \to L$ 和 $R \to t$ 都是 1。运行最大流算法。

Theorem G 中的最大匹配和 G' 中的最大流流值相等。

【另外 ppt 17/41 讲了三种最大流算法下的时间复杂度。有两个算法是作业题，而且比较复杂。赌他不考吧】

13.10.3 图像分割 image segmentation

【18/41】这里只考虑前景和背景的分割。

建模：对每个像素点 i，与其上下左右相邻点之间有边。像素点构成点集，相邻点之间的边构成边集。每个像素点 i 有三个参数（提前给定好）：$a_i \geq 0$ 表示 i 在前景的概率，$b_i \geq 0$ 表示 i 在背景的概率，$p_{ij} \geq 0$ 表示 i 和 j 一个分在前景一个分在背景时的惩罚（因为相邻点更可能标签相同，所以对相邻点不同标签的分割方法给出一定的惩罚）

像素点分割成前景 A 和背景 B 两个集合，要使下式值最大：$\sum_{i\in A} a_i + \sum_{j\in B} b_j - \sum_{(i,j)\in E, |A\cap\{i,j\}|=1} p_{ij}$。即要前景和背景点的概率尽可能大，分界线上的惩罚尽可能小。

【后面还有一堆构造方法，不打字了，见 ppt 吧】转化为最小割问题了

13.11 引入随机的最小割算法

【考试不要求】

14 线性规划 linear programming

【主要了解建模。对算法设计不要求。即把问题建模成 lp 模型，把 lp 问题转化成对偶问题等】

【除了本章介绍的线性规划外，它的另一种用途是为 NP 难问题设计近似算法。这在近似算法那一章将多次看到】

线性规划的三要素 如下：

1) 决策变量（decision variables）$x_1, x_2, ..., x_n$

2) 目标函数（objective）$\sum_{j=1}^{n} c_j x_j$

3) 线性约束（linear constraints）$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, 1 \leq i \leq m$

给定系数 a_{ij}, c_j 和 b_i，其中 $1 \leq i \leq m, 1 \leq j \leq n$。要找到一组满足线性约束的决策变量，使得线性目标函数的值最大。

【有一些超平面、凸集之类的东西在 ppt 上，不抄了（来不及了）】

14.1 标准形式（原问题）

目标：

$$\max \boldsymbol{c}^T \boldsymbol{x}$$

约束：

$$A\boldsymbol{x} \leq \boldsymbol{b}$$
$$\boldsymbol{x} \geq \boldsymbol{0}$$

14.2 对偶问题 duality

思路：找目标函数的上界，如果上界越来越紧，就会越来越接近最大值。

目标：

$$\min \boldsymbol{b}^T\boldsymbol{y}$$

约束：

$$A^T\boldsymbol{y} \geq \boldsymbol{c}$$
$$\boldsymbol{y} \geq \boldsymbol{0}$$

可见把最大化问题对偶成了最小化问题。对偶问题再对偶，就回到最初的最大化问题。

14.3 弱对偶性

弱对偶性定理 设 x 是原问题的可行解，y 是对偶问题的可行解，则 $\boldsymbol{c}^T\boldsymbol{x} \leq \boldsymbol{b}^T\boldsymbol{y}$。即，原问题的最大值不超过对偶问题的最小值。

证明：【18/39】

【先到这吧。ppt 后面给了一些线性规划建模的例子，再往后是不要求的东西了】

15 NPC 问题

不是所有问题都有多项式时间算法。比如旅行商问题（Traveling Salesperson Problem, TSP）是一个经典的 NP 问题。我们动态规划那一章设计过一个动态规划算法，时间复杂度是 $O(n^2 2^n)$ 的。这也是目前已知的最优算法。团（Clique）问题是另一个难的问题，目前一直的最优算法是 $O(n2^n)$ 的。

我们可以通过证明下界来说明某问题就是很难，没有多项式时间算法（回顾我们证明过比较排序的下界是 $O(n \lg n)$）。但是证明下界也是很困难的。我们也可以证明一个问题和某个困难问题等价，来说明这个问题是困难的。

从一个角度说，我们当然希望能够解更多的问题（即更多的问题都是 P 的）。但是从另一个角度说，一些领域是需要困难问题的，例如单项函数（one way function）就应用于密码学中。密码学希望找到更好的困难问题，它要求问题的平均时间也是困难的，而不仅仅是最坏情况下困难。

【P,NP 等问题的定义，略】【NP 问题一定是判定问题而不是求解问题（?）】

【强调一点：NPC=NP+NP 难，即问题本身是 NP 的，且所有 NP 问题可以规约到它（有一个 NP 难问题可以规约到它即可）】

【lec13 17/58】规约与被规约的关系问题，要清晰。判断 NP 难的规约和这一页说的是相反的，Π' 是 NP 难，且 $\Pi' \leq \Pi$，有 Π 是 NP 难。

15.1 可满足性问题（SAT）satisfiability problem

【lec13 19/58】任何 NP 问题都可以规约到 SAT 问题（作为一个 Theorem，这里我们只接受不证明）。SAT 是 NPC 的。

SAT 的每个语句都是或符号连接的（析取）。实际上，与符号可以用或和非来实现，所以我们不妨规定 SAT 的语句都是或符号连接。

15.2 团问题 Clique

【lec13 22/58】团是一个图中的完全子图。团问题是找最大团的问题。

规约：SAT $\leq$ Clique。

对任意的 SAT 公式 $\phi = C_1, C_2, ..., C_m$，设这些公式包含变量 $x_1, x_2, ..., x_n$。现在构造一个图 $G = (V, E)$ 和一个数 k，使得 ϕ 可满足当且仅当 G 有一个规模 $\geq k$ 的团。

对 ϕ 中涉及的每一个文字 t，创建一个节点 v_t（即使是相同的文字，只要在不同的语句中出现，都要再加一个点）。创建边 $(v_t, v_{t'}$，当且仅当满足下列两个条件：1)t 和 t' 不在同一个语句中；2)t 不是 t' 的反。

Claim ϕ 可满足当且仅当 G 有 $\geq m$ 的团【证明略。一个语句为真，等价于语句中有至少一个文字为真】

15.3 独立集问题（IS）independent set

【lec13 29/58】独立集是不相邻的点组成的集合。独立集问题（IS）是找最大独立集的问题

规约：团问题 $G = (V, E), k$，构造对应的独立集问题 $G' = (V', E'), k'$。$k' = k, V' = V, E' = E^c$。这里 E^c 表示 E 的补。这个构造实际就是构造一个补图。

Claim C 是 G 中的一个团当且仅当 C 是 G' 中的一个独立集

15.4 顶点覆盖问题（VC）vertex cover

是用顶点覆盖所有边的问题。希望用尽可能少的顶点覆盖所有边。

规约：IS 问题 $G = (V, E), k$，构造对应的 VC 问题 $G' = (V', E'), k'$。$V' = V, E' = E, k' = |V| - k$

Claim S 是 G 中的独立集当且仅当 S 是 G' 中的点覆盖。

至此，SAT $\leq$ Clique $\leq$ IS $\leq$ VC。且有不证明的那个定理可知，它们都可规约到 SAT。它们都是 NPC 问题。

15.5 3SAT 问题

【2SAT 是 P 的，3SAT 仍是 NPC 的】

SAT 到 3SAT 规约【lec13 34/58】

15.6 k 着色问题 k-Colorability

对 $k=2$ 是容易的。对 $k \geq 3$ 是 NPC 的。

SAT 规约到 3 着色，3 着色规约到 k 着色。【SAT 规约到 3 着色，要看计算理论的作业】【可以让韬洋帮忙拿计算理论作业看看】

15.7 精确集合覆盖 Exact (Set) Cover

【42/58】给定一个有限集 X 和它的一组子集构成的集合 S，问是否存在 $S' \subset S$，使得 X 中每一个元素，都恰好在 S' 中的一个元素（S' 的元素是 X 的子集）里。

规约：3 着色规约到精确集合覆盖

15.8 整数规划（IP）Integer Programming

【lec13 48/58】线性规划增加变量是整数的条件。

规约：点覆盖 VC 规约到 IP【就是把 VC 问题建模成 IP 问题。每个点 v 用一个变量 x_v 表示，x_v 为 0 表示不选这个点，为 1 表示选这个点。则约束条件是 $x_u + x_v \leq 1, \forall (u,v) \in E$ 且 $x_v \in \{0,1\}, \forall v \in V$。目标是 $\min \sum x_v$】

15.9 背包问题 Knapsack

原始的背包问题是给定背包容量 B，要使装下的物品总价值尽可能高。这里改成判定问题：给定期望的总价值 B，判定能否找到一个总重不超过 W（即找到容量不超过 W 的背包），装下价值不少于 B 的物品。【因为 NP 问题必须是判定问题不能是求解问题】

【规约？ppt 上好像没给?】

15.10 子集和问题 Subset Sum

给定数值集合 S 和目标值 B，判断是否存在一个子集 $S' \subset S$，使子集中元素和恰好为 B。

规约：精确覆盖规约到 Subset Sum【lec13 53/58】【构造 p 进制 0-1 串】

15.11 集合分割问题 Partition

给定数值集合 S 和目标值 B，判断是否存在一个子集 $S' \subset S$，使子集中元素和恰好等于子集外元素和（即把 S 等分成两半）。

上面三个问题的规约：
Partition 规约到 Subset Sum【显然啊】
Subset Sum 规约到 Partition：引入两个新元素 $N-B$ 和 $N-(\Sigma-B)$。这里 Σ 是原集合 S 的元素和，$N > \Sigma$。这样转化为总和为 $2N$ 的集合分割问题【因为 $N > \Sigma$，所以新加入的两个元素不可能分在同一组】
Partition 规约到 Knapsack：集合中每个元素 i 视为权重为 i 的一个物品，背包问题的参数 $W = B = \Sigma/2$

15.12 装箱问题 Bin Packing

给定有限数集 S，箱子大小 B。用最少的箱子把 S 中元素全部装起来。

规约：Partition 规约到 Bin Packing【容易的。构造箱子大小是集合元素和的一半，则 Partition 有解当且仅当两个箱子可装】

15.13 哈密顿回路 Hamiltonian Circuit

图上所有节点都恰好访问一次的回路。可以是有向图也可以是无向图

规约：点覆盖 $\leq$ 哈密顿回路【lec13 55/58】【这个规约我没看懂】

15.14 旅行商问题（TSP）Traveling Salesman

有向图，边上带权，给定 k，找权重和 $\leq k$ 的哈密顿回路。

规约：哈密顿回路 $\leq$TSP

16 近似算法

难问题的近似解，可以用多项式时间得到。

16.1 负载均衡 Load Balancing

记机器数为 m，任务数为 n。近似算法的解为 L，最优解为 L^*。

【这个不是有贪心算法吗？注意看一下这个和贪心那里讲的任务分配的那个问题的区别】【区别在于，贪心那里的任务，开始时间和结束时间是固定的，而这里只有任务的持续时间，何时开始也交由我们的算法安排】

Claim 即使只有两台机器，负载均衡问题也是 NP 难的【Partition 规约到负载均衡】

16.1.1 简单贪心解法

总是把任务分配给当前负载最轻的机器

Theorem 简单贪心解法是 2-近似的。【lec14 10/72】【这个 2 是紧界，可以举出例子：m 机器，m(m-1) 任务权重 1，最后一个任务权重 m】

16.1.2 最长运行时间（LPT）优先的贪心算法

由上面 2-紧界的例子受到启发：先考虑运行时间长的任务。

Lemma 3 如果有超过 m 个任务，则 $L^* \geq 2t_{m+1}$

证明：【因为前 $m+1$ 个任务分配到 m 台机器，至少两个任务在同一台机器上。且任务按运行时间降序排列，所以这台机器上的负载至少 $2t_{m+1}$】

Theorem LPT 贪心算法是 3/2-近似的。【lec14 16/72】【这个 3/2 界是不紧的。另有论文证明了 LPT 是 4/3-近似的。太复杂了，不要求】

证明：$L_i = (L_i - t_j) + t_j \le L^* + L^*/2 = 3/2L^*$。这里 t_j 是分配在机器 i 上的最后一个任务 j 的运行时间。

16.2 点覆盖

16.2.1 简单贪心解法

每次随机选一条没被覆盖的边，任取一个顶点加入覆盖集中。

【复杂度：通过构造一个巧妙的 L,R 二分图，证明了近似度是 $\Omega(\lg r)$ 的】【这里 r 是最优解的值。也就是说，当最优解很大的时候，贪心解的答案可能大到没谱了】

16.2.2 聪明的贪心算法

每次选取能够再覆盖最多的边的点。即为每个点维护它的未覆盖邻边的数量。

【复杂度：如果把这个算法应用到上面构造的 L,R 二分图，还是会选出 R 做结果，所以，最坏情况下近似度还是 $\Omega(\lg r)$ 的】

16.2.3 2-近似的算法

每次随机选一条边，把这条边的两个顶点都放在覆盖集里，然后删去这两个点的所有邻边。

Theorem 这个算法是 2-近似的多项式时间算法。

证明：易证算法是多项式时间的，下面证明算法是 2-近似的。记算法解为 C，最优解为 C^*。【算法中的 A 的定义 lec14 26/72】则 $|C^*| \ge |A|$，$|C| = 2|A|$。所以 $|C| \le 2|C^*|$。【A 是最大匹配（为什么?）应该说 A 是匹配。这里也用不到最大匹配的性质啊。】

16.3 旅行商问题 TSP

Theorem 1 对任意 α，找到 TSP 问题的 α-近似算法是 NP 难的。

证明：可以从哈密顿回路问题规约过来。对哈密顿回路问题的图 G，构造 α-近似旅行商问题的图 G' 如下

$$w(u,v) = \begin{cases} 1, & if\ (u,v) \in E \\ \alpha|V| + 1, & otherwise \end{cases}$$

则在 G 里找到哈密顿回路当且仅当 G' 中找到 α 精度的旅行商问题近似解。【这个当且仅当没证】

Theorem 2 若图满足三角不等式，即 $w(u,v) \ge w(u,x) + w(x,v)$，则有多项式时间的 3/2-近似算法。

16.3.1 一个 2-近似算法

【lec14 29/72】【找最小生成树，倍增最小生成树的边数，在倍增后的 “树” 上找欧拉回路（一定有），按照欧拉回路上每个点第一次出现的顺序，确定 TSP 的回路】

【证明略。见 ppt lec4 30/72】

这个 2 是紧界。31/72 有例子。

16.3.2 一个 3/2-近似算法

前面算法的问题在于，如果一个节点的度数已经是偶数，就没必要再复制它的边了。【这里把最小生成树中节点度数为奇数的那些点（一定是偶数个点）单独拿出来，在原图 G 上找这些点的代价最小的完美匹配 M，把 M 中的边加入树中。然后再找欧拉回路等等，与上面算法一样】M 加入 T 之后，得到的就不一定还是树了。匹配 M 中的边是从 G 里来的，不是从 T 里来的。但是不影响仍然有欧拉回路。

Lemma 设 $V' \subset V$ 且 $|V'|$ 是偶数。令 M 为 V' 的代价最低的完美匹配，则 $cost(M) \leq OPT/2$.

证明：【cost(V' 上的一个环路)<=OPT cost(M)<=(1/2)cost(V' 上的这个回路)】

【证明 1.5-近似度：$cost(C) \leq cost(\mathcal{T}) \leq cost(T) + cost(M) \leq 3/2OPT$】

【还有更多结论，不做要求 34/72】

16.4 带权集合覆盖 Weighted Set Cover

【顶点覆盖是 NP 难问题，顶点覆盖是集合覆盖的子例，故集合覆盖是 NP 难问题（顶点覆盖能规约到集合覆盖）】

【一种是要用最小代价覆盖所有元素，一种是要在固定代价内覆盖尽可能多的元素。这里讨论的是第一种】【不要求取出的覆盖集合之间互不相交（集合覆盖要求每个元素恰好出现在一个覆盖集合中，即覆盖集合两两不相交）】

16.4.1 用整数规划求近似解

问题可以建模成整数规划问题，但是证书规划问题没办法多项式时间求解，所以松弛为线性规划来解。当然这样的解只是近似解。最后要把线性规划得到的结果舍入到整数，返回原问题的近似整数解。

【lec14 37/72 起。整数规划建模和松弛到线性规划都是容易的，关键是 40/72 的舍入操作比较难想。希望考试不会要求自己给这种算法吧。。。】

Theorem 这个算法是 f-近似的。

【搞不动了。。。先看点别的吧】