高级算法分析与设计

绪论

算法的时间复杂性分析

渐进的界

-	渐进上界	渐进下界	上界	下界	
符号	$f(n)=O(g(n))$	$f(n)=\Omega(g(n))$	$f(n)=o(g(n))$	$f(n)=w(g(n))$	
条件1	$0\leq f(n)\leq c\cdot g(n)$	$0\leq c\cdot g(n)\leq f(n)$	$0\leq f(n)<c\cdot g(n)$	$0\leq c\cdot g(n)<f(n)$	$f($
条件2	存在 c	存在	任意	任意	
阶	$f(n)\leq g(n)$	$f(n)\geq g(n)$	$f(n)<g(n)$	$f(n)>g(n)$	

渐进的界的定理

定理 1:比率代理

$lim_{n\to \inf}f(n)/g(n)$	value	result
-	$c>0$	$f(n)=\Theta(g(n))$
-	0	$f(n)=o(g(n))$
-	$+\inf$	$f(n)=w(g(n))$

推论

$n^d=o(r^n),r>1,d>0$
$ln(n)=o(n^d),d>0$

定理 2: 传递性定理

函数的阶之间的关系具有传递性。

定理 3: 加法定理

```
if f=O(h),g=O(h)
then f+g=O(h)
```

重要函数

对数函数

$log_2 n = \Theta(log_l n)$
$log_b n = o(n^\alpha)$
$a^{log_b n} = n^{log_b a}$

指数函数与阶乘

$n! = o(n^n)$
$n! = w(2^n)$
$log(n!) = \Theta(nlogn)$

函数按照阶排序

$2^{2^n}, n!, n2^n, (3/2)^n, (logn)^{logn} = n^{loglogn}, n^3$
$log(n!) = \Theta(nlogn), n = 2^{logn}, (logn)^2, logn, (logn)^{0.5}, loglogn, n^{1/logn} = 1$

算法的数学基础

序列求和

等差数列：$\sum_{k=1}^{n} a_k = \frac{n(a_1+a_n)}{2}$
等比数列：$\sum_{k=0}^{n} aq^k = \frac{a(1-q^{n+1})}{1-q}, \sum_{k=0}^{\inf} aq^k = \frac{a}{1-q}$
调和级数：$\sum_{k=1}^{n} \frac{1}{k} = ln(n) + O(1)$

递推方程求解

1. $f(n) = a \cdot f(n-1) + b$：不断带入函数
2. $f(n) = a \cdot f(n/c) + b$：换元，令 $n = c^t$，然后不断迭代
3. 高阶递推方程：差消法化简为一阶方程
4. 递归树
5. 主定理：$T(n) = aT(n/b) + f(n), a \geq 1, b>1$

1. $f(n)=O(n^{log_ba-\epsilon}),\epsilon>0,\rightarrow T(n)=\Theta(n^{log_ba})$
2. $f(n)=\Theta(n^{log_ba})\rightarrow T(n)=\Theta(n^{log_ba}logn)$
3. $f(n)=\Omega(n^{log_ba+\epsilon}),\epsilon>0,c<1,af(n/b)\leq cf(n),\rightarrow T(n)=\Theta(f(n))$

经典算法设计策略

分治法

复杂度分析

$T(n)=aT(n/b)+d(n)$
$d(n)=c,T(n)=O(n^{log_ba})(a\neq 1);O(logn)(a=1)$
$d(n)=cn,T(n)=O(n)(a<b);O(nlogn)(a=b);O(n^{log_ba})(a>b)$

DP

矩阵连乘

$f(i,j)=min(f(i,k)+f(k+1,j)+(i-1)\cdot k\cdot j)$

贪心

基于搜索的算法设计策略

回溯

子集树：当问题是要计算 n 个元素的子集，以便达到某种优化目标时，可以把这个解空间组织称一棵子集树。通常有 2^n 个叶子结点，$2^{n+1}-1$ 个结点，遍历的时间复杂度为 $\Omega(2^n)$。
排列树：当问题是要确定 n 个元素的满足某种性质的排列时，可以把这个解空间组织成一棵排列树。通常有 $(n-1)!$ 个叶子结点，遍历的时间复杂度为 $\Omega(n!)$。

启发式搜索

估价函数：$f(x)=g(x)+h(x)$,
$g(x)$为初始结点到x付出的实际代价，$h(x)$为节点x到目标节点的最优的估计代价。

NP 完全性

问题的复杂性分类

我们用多项式时间界限来定义一个问题的难易程度。一个问题可以在多项式时间内解决，称该问题是易解决的（易解问题）。不能的话就是不易解决的(难解问题)。

P 类问题与 NP 类问题

P 类问题：确定性计算模型下的易解的判定问题类，即在确定图灵机模型下多项式时间可解的判定问题类。
确定性算法：执行过程中每一步都有一个确定的选择，对于同样的输入，得到的结果严格一致。
NP （Nondeterministic Polynomial）类问题：非确定图灵机模型下多项式时间内可解的判断问题类，在确定图灵机下可验证的问题类。（可能多项式时间可解，也可能多项式时间不可解）
非确定性图灵机：非确定性图灵机允许状态转移函数具有不确定性。
非确定性算法：由猜想和验证组成的算法。

NP-hard 问题与 NPC 问题

归约：A 可以归约为 B。可以用问题 B 的解法来解决问题 A。
NP-hard：所有 NP 问题都可以多项式归约到问题 D，则问题 D 称为 NP-hard 问题。
NPC：所有 NP 问题都可以多项式归约到 NP 问题 D，则 NP 问题 D 称为 NPC 问题。

- NPC 问题 D 的性质
  - D 是 NP-hard 问题
  - D 是 NP 问题

一些 NPC 问题

逻辑电路问题：给定一个逻辑电路，是否存在一种输入使输出为 True。

旅行商问题：

和图有关的 NPC 问题

团问题：给定无向图 $G=(V,E)$，团是 G 中的一个顶点子集 $\hat{V}$，其中任意两个点都由 E 中的一条边连接。一个团是 G 的一个完全子图。
顶点覆盖：图的顶点覆盖（有时是节点覆盖）是一组顶点的集合，使得图的每个边至少与集合中的一个顶点相连接。（用最少的顶点覆盖图中所有的边）
哈密顿回路：在一个图中，从某个顶点处罚，经过所有其他顶点一次且仅一次的回

路称为哈密顿回路。
最大匹配：给定一个图，要求选出一些边，使得这些边没有公共顶点，且边的数量最大。
最小边覆盖：用最少的边覆盖图中所有的顶点。
最小点覆盖：用最少的点覆盖图中所有的边。（实际上就是最小顶点覆盖）
最大独立集：一个图中点集合的子集，集合中所有的点都不相邻。
精确覆盖问题：给定一组集合，从里面找出来 M 个，使得这 M 个集合的元素恰好覆盖所有集合的元素。

和图有关的一些定义

欧拉回路：通过每条边恰好一次的回路
哈密顿回路：通过所有顶点一次且仅一次的回路

线性规划

线性规划的形式

线性规划的标准形式

$$\left.\begin{aligned} &\min \sum_{j=1}^{n} c_j x_j \\ &s.t. \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1,2,\cdots,m \\ &\qquad x_j \geq 0, \quad j = 1,2,\cdots,n. \end{aligned}\right\} \qquad (2.1)$$

c 为价格系数，b 为右端项。

典范形式

$$\left.\begin{aligned} &\min \vec{c}^{\,T} \vec{x}; \\ &s.t.\ A\vec{x} = \vec{b} \\ &\qquad \vec{x} \geq 0. \end{aligned}\right\} \qquad (2.2)$$

典范形式就是采用向量-矩阵表示法的标准形式。

一般形式

$$
\begin{cases}
\min \sum_{j=1}^{t} c_j x_j \\
s.t. \ \sum_{j=1}^{t} a_{pj} x_j \le b_p, \quad p = 1,2,\cdots,u \\
\sum_{j=1}^{t} a_{qj} x_j \ge b_q, \quad q = u+1,\cdots,u+v \\
\sum_{j=1}^{t} a_{rj} x_j = b_r, \quad r = u+v+1,\cdots,m \\
x_j \ge 0, \quad j = 1,2,\cdots,t.
\end{cases}
\qquad (2.5)
$$

一般形式与标准形式的关系

引入让 $\le$ 变成 $=$ 的变量称为松弛变量。引入让 $\ge$ 变成 $=$ 的变量称为剩余变量。

对偶问题（PPT 没有，可能不考）

对偶问题：每一个线性规划问题都伴随有另一个线性规划问题，称为对偶问题。原来的线性规划问题则称为原始线性规划问题，简称原始问题。

1. 目标函数对原始问题是极大化，对对偶问题则是极小化。
2. 原始问题目标函数中的收益系数是对偶问题约束不等式中的右端常数，而原始问题约束不等式中的右端常数则是对偶问题中目标函数的收益系数。
3. 原始问题和对偶问题的约束不等式的符号方向相反。
4. 原始问题约束不等式系数矩阵转置后即为对偶问题的约束不等式的系数矩阵。
5. 原始问题的约束方程数对应于对偶问题的变量数，而原始问题的变量数对应于对偶问题的约束方程数。
6. 对偶问题的对偶问题是原始问题，这一性质被称为原始和对偶问题的对称性。

线性规划的对偶

原问题为 max，则 min 的约束条件与 max 变量符号相同，min 的变量符号与 max 的约束条件的符号相反
原文题为min，即 max 的约束条件与 min 的变量符号相反，max 的变量符号与 main 的约束条件符号相同

近似算法和随机算法

近似算法

近似比

r (n)-近似算法：$max(C/C^*, C^*/C) \leq r(n)$，其中 $r(n)$ 称为近似比。

1. 近似比不会小于 1
2. 1-近似算法可以得到最优解
3. 一个 NPC 问题允许有多项式时间的近似算法
4. 近似算法的运行时间一般应该是多项式时间

可近似性分类

1. 完全可近似的：对任意小的 $\epsilon>0$，存在 $(1+\epsilon)-$ 近似算法
2. 可近似的：存在具有常数比的近似算法
3. 不可近似的：不存在具有常数比的近似算法

一些 NPC 问题的可近似性

问题	近似性
01 背包	完全
最小顶点覆盖	可近似
多机调度问题	
货郎问题	可近似
装箱问题	

随机算法

随机算法是一种使用概率和统计方法在其执行过程中对于下一计算步骤作出随机选择的算法。

随机算法的分类

1. 随机数值型算法
    1. 用于数值问题求解
    2. 输出往往是近似解

    3. 解的精确度与算法执行时间成正比
2. Sherwood 舍伍德型随机算法
    1. 一定能求得一个正确解
    2. 最坏与平均复杂性差别大时，加入随机性即可得到 Sherwood 算法
    3. 消除最坏行为与特定实例的联系
3. Las Vegas 拉斯维加斯型随机算法
    1. 一旦找到一个解，该解一定是正确的
    2. 找到解的概率与算法执行时间成正比
    3. 增加对问题反复求解次数，可以使找不到正确解的概率任意小
4. Monte Carlo 蒙特卡洛型随机算法
    1. 用于求解需要准确解的问题
    2. 可能给出错误解
    3. 正比

逻辑回归和神经网络

看 PPT 吧，懒得看了