算法导论

1、

一、图中点独立集是指图中顶点集的一个自己，其中任意两点之间没有边，找出图中最大点独立集是一个优化问题。（25 分）

1. 定义相应的判定问题。
2. 证明：该判定问题是 NP-问题。
3. 证明：如果判定问题能在多项式时间内解，则上述的优化问题也能在多项式时间内解。
4. 证明：该判定问题是 NP-完全问题。
5. 证明：如果独立集判定问题中独立集大小是常数，则该问题为 P 问题。

2、

问题 P：任给一无向图 G，判定它是否含有哈密顿（Hamilton）回路。我们知道问题 P 是 NP-完全问题。

问题 N：任给一带权完全图 H 与一正整数 k，判定 H 中是否含有总权数为 k 的哈密顿（Hamilton）回路，这就是著名的“旅行推销员问题”，我们知道问题 Q 也是 NP-完全问题。

假设对问题 Q 做如下简化：每条边的权值只能是 1 或 2

1. 简述如何利用问题 P 的 NP-完全性证明简化的问题 Q 仍是 NP-完全问题。
2. 给出具体的证明。

3、

1. 已知等腰三角形各边长，求高。
2. 已知直角三角形的任意两边长，求第三边的长度。

利用这两个问题解释**多项式时间规约**的概念，并说明多项式时间规约在计算机算法理论中的作用。

4、

举一个 NPC 问题的例子，说明采用随机算法能够在多项式时间内得到一个可行解，并解释如何判断解得“质量”？

5、

（15 分）如果你碰到一个问题，它已经被证明是 NP-完全问题，你可能选择用什么办法来处理该问题，如何来评价你的解决办法的优劣？

6、

简述什么是近似算法，什么是随机算法，如何评价这两类算法的优劣？

7、

三、S 是含 n 个自然数的集合，$T(n)$ 是单调递增函数，且存在常数 c（0<c<1），满足对任意自然数 n，$T(n/2) < cT(n)$。

证明：任给一求 S 的中位数（Medium）的算法，若其时间复杂度为 $T(n)$，则可利用该算法给出一个时间复杂度为 $O(T(n))$ 的算法求 S 中的第 k 个最小的数（1≦k≦n）（15 分）

8、

$W(n) = cn + w\lfloor n/2 \rfloor; \quad w(1) = 1$

证明：$W \in O(n)$

9、

一、（10 分）解释“一个算法的时间复杂度为 $O(n^2)$”的含义。如果在解决一个实际问题是有两个不同的算法可以选择，它们的时间复杂度分别是 $O(n^2)$和 $O(n \log n)$，你是否一定会选择后者，如果不是，还有其它哪些因素可以应该考虑？

10、

1. 以 Quicksort 算法为例，解释什么是最好情况时间复杂度、最坏时间复杂度、平均时间复杂度？

2. 在 Quicksort 算法中选择第一个元素为比较基准对象或者通过随机方法来选择一个元素为比较基准对象效果有差别吗？请给出解释。

11、

一、简单描述 Quicksort 算法的原理与过程。概述（即给出结果和理由，不需要严格的数学证明）最坏情况以及平均情况的时间复杂性分析。（15 分）

12、

最典型的最小生成树算法有 Prim 算法和 Kruskal 算法。

1. 用简洁的语言描述这两个算法的基本过程。
2. 各自采用什么样的数据结构可以高效的实现这两个算法。
3. 比较它们的代价，在实际应用中你会如何选择？

13、

任给一个连通带权图(假设权值为正实数)，求该图的一个生成树，并使得其所有边的权值和最小。这个问题成为“最小生成树问题”。解该问题最经典的两个算法均是基于“贪心策略”的。

（1）简述这两个算法的基本思想和过程概要；并解释它们如何体现“贪心策略”。

（2）比较这两个算法的效率。

（3）简述在实现时，这两个算法分别适合采用什么样的数据结构。

（4）以其中任意一个做例子，解释什么是“循环不变式”，并说明它在证明循环算法正确性时起什么作用。

14、

试述递归和数学归纳法有什么关联，你认为它们最重要的区别是什么？你能否举一个简单的例子，说明这个差别对计算机解题的影响。

解答部分

1-3 NPC 证明问题

P 问题：多项式时间内可以判定或者多项式时间内可以解决的问题。
NP 问题：多项式时间内可以验证的问题。
NPC 问题：NP 问题+NP 难度（即任意 NP 问题可以规约为 NPC 问题）。

最优化问题→判定问题。最优化问题容易说明判定问题容易，逆否命题同样成立。
规约：$f(L_1) = L_2$等价于$L_1 \leq pL_2$。f是一个多项时间可以计算的函数，且保证L_1到L_2是相互封闭的映射子空间。
很多最优化问题是 NPC 问题，但是不追求最优化就是个 P 问题，是因为不追求最优化的判定问题等价于一个验证问题，即$\mathrm{P} \leq \mathrm{NP}$。

等价证明 NPC 问题：（即规约法）

1、 最大独立集问题

独立集定义：如果有一个顶点集合S，S中任意两个顶点之间都没边，则称S为一个独立集。
独立集问题描述：
给定一个图G=(V,E)和整数k，
问:是否存在至少大小为k的独立集S?

2、 最小顶点覆盖问题

顶点覆盖定义：如果一个顶点集合S，G中的任意一条边都与S中的某个顶点相关联，则称S为顶点覆盖。
顶点覆盖问题描述：
给定一个图G=(V,E)和整数k，
问：是否存在至多大小为k的顶点覆盖S?

3、 最大团问题

团：如果有一个顶点集合S，S中任意两个顶点之间都有边，则称S是一个团。

团问题描述：

给定一个图G=(V,E)和整数k，

问：是否存在至少大小为k的团S？

规约证明：（**补运算，所以多项式时间内可以规约 | 顶点补和边补**）

1.顶点覆盖规约到独立集

假设有一个顶点覆盖实例(G,k)，有一个大小为k的顶点覆盖S，则G中任意一条边e，他的两个端点中至少有一个点在S中，则他的两个端点中至多有一个点在V-S中，即V-S中任意两个点没有边，即V-S为独立集。

所以只要G中存在一个大小为k的顶点覆盖，则G中存在一个V-k的独立集。

2.独立集规约到团

假设有一个独立集实例(G,k)，有一个大小为k的独立集S，则S中任意两个点都没有边，

则对于$\overline{G}$来说，S中任意两个点都有边，即S在$\overline{G}$中是一个团。

只要在G中有一个大小为k的独立集，则$\overline{G}$中存在一个大小为k的团。

1、 哈密尔顿回路问题

哈密尔顿回路定义：对于无向图 G(V,E)，哈密尔顿回路即通过 V 的每个顶点的简单回路。

哈密尔顿回路问题描述：给定一个无向图 G 和整数 k，

问：是否存在至少大小为 k 的哈密尔顿回路 H？

2、 旅行商问题

给定一个完全图 G（V，E），为无向图 G 边赋权值的代价函数 c 和整数 k，求一条代价最小并经过所有顶点的路线，

问：是否存在代价最多为 k 的路线？

规约证明：

建立一个完全图 F（V，E），定义代价函数如下：

$$c(i,j)=\begin{cases}0,(i,j)\in H;\\1,(i,j)\ni H.\end{cases}$$

则 G 中存在一条哈密尔顿回路等价于 F 中存在一条代价至多为 0 的回路。

若将代价函数定义为：

$$c(i,j)=\begin{cases}1,(i,j)\in H;\\x,(i,j)\ni H.\end{cases}\quad(x>1)$$

则 G 中存在一条哈密尔顿回路等价于 F 中存在一条代价至多为|H|的回路。

4-6 近似算法与随机算法

解决 NPC 问题三种方法：$\begin{cases}n\text{ 较小时，指数级的运算可以接受；}\\\text{某些特殊情况，可以使用穷举或者列表的方式解决；}\\\text{在多项式时间内得到近似最优解}\begin{cases}\text{近似算法}\\\text{随机算法}\end{cases}。\end{cases}$

近似算法是以给定的近似比阈值(即$\rho(n)\le\varepsilon$-1)为上界,在多项式时间内求解最优化问题的方法。所谓“近似”，就是指结果不一定是最优的，但是也在可以承受的范围内，而且可以比精确求解消耗更少的资源。

若对问题的输入规模 n，有一函数 ε(n)使得

$$\left|\frac{c-c^*}{c^*}\right| \leqslant \varepsilon(n)$$

则称 ε(n)为该近似算法的相对误差界。近似算法的性能比 ρ(n)与**相对误差界** ε(n)之间显然有如下关系：

$$\varepsilon(n) \leqslant \rho(n)-1$$

随机算法是随机化输入数据，再运用概率期望公式求解期望运行时间和期望运行代价的方法；

随机算法分为两类：

（1）拉斯维加斯（Las Vegas）算法，要么给出正确解，要么无解。无解时，需要再次调用该算法。要么给出问题的正确答案，要么得不到答案。反复求解多次，可使失效的概率任意小。

（2）蒙特卡罗（Monte Carlo）算法，总能给出解，但可能是正确解也可能是错误解。多次调用该算法可降低错误率。总能得到问题的答案，偶然产生不正确的答案。重复运行，每一次都进行随机选择，可使不正确答案的概率变得任意小。

至于用哪种算法取决于具体的应用。对于一些不允许出错的应用则采用 Las Vegas 算法，但若是允许小概率错误的话，Monte Carlo 算法更加省时。

随机算法的优点：对于某一给定的问题，随机算法所需的时间与空间复杂性，往往比当前已知的、最好的确定性算法要好；到目前为止设计出来的各种随机算法，无论是从理解上还是实现上，都是极为简单的；随机算法避免了去构造最坏情况的例子。

与随机算法相对的还有**概率分析（针对确定性算法复杂度的分析）**，即先假设数据满足一个先验分布，然后利用概率分布公式求解平均运行时间和平均运行代价的方法。

1、近似算法

（1）评价该解决办法的优劣性，采用近似比ρ(n)：求解最优化问题时给每个可能解一个正的代价，定义近似解的代价为 C，最优解的代价为 W。对于规模为 n 的输入，若满足

$$\max\left\{\frac{C}{W},\frac{W}{C}\right\} \leq \rho(n)$$

则称该近似算法有近似比ρ(n)。

（2）近似算法可以解决的问题：$\begin{cases}2\text{ 近似算法(常数近似比)：顶点覆盖、旅行商问题}\\ \text{贪心近似算法(对数近似比)：集合覆盖问题}\end{cases}$

2、随机化近似算法，简称随机算法

（1）评价该解决方法的优劣性，采用随机化的近似比ρ(n)：规定与上述一样，定义该随机算法产生的期望代价为 C，最优解代价为 W。对于规模为 n 的输入，若满足

$$\max\left\{\frac{C}{W},\frac{W}{C}\right\} \leq \rho(n)$$

则称该近似算法有随机近似比ρ(n)。

（2）随机算法可以解决的问题：3-CNF 可满足性（3-合取范式：m 交含三并 | m 个子式）。

Remark：近似算法求解定点覆盖问题、随机算法证明 3-CNF 可满足行证明。

7-8、递归式求解问题

三、证明：则求第 k 个最小的数的最坏情况下的递归式可按如下构造

$$W(n)=\begin{cases}O(1), n=1\\ T(n)+W(n/2), n>1\end{cases}$$

展开可得

$$W(n)=T(n)+T(n/2)+T(n/2^i)+\cdots+W(1)$$

根据$T(n/2)<cT(n), 0<c<1$

$$W(n)<T(n)+cT(n)+c^2T(n)+\cdots+W(1)$$

$$<\frac{1}{1-c}T(n)+W(1)$$

从而可以得到$W(n)=O(T(n))$

证明：假定n/2为整数

$$2^n=C_n^0+C_n^1+\cdots+C_n^{n/2}+C_n^{1+n/2}+\cdots+C_n^n$$

$$\le nC_n^{n/2}$$

取c=1，$n_0=2$,可知当$n>n_0$时，有$c\frac{2^n}{n}\le\begin{bmatrix}n\\ n/2\end{bmatrix}$

证毕

Remark：主定理。

9、时间复杂度

时间复杂度是一种描述计算工作量与输入数据规模之间的变化趋势的度量，是指执行算法所需要的计算工作量，而不是指具体时间。

趋势的度量。当问题规模充分大时，某算法中基本语句的执行次数在渐进意义下的阶，就是这个算法的时间复杂度。通常用大 O 表示。这里 O（n^2）表示当 n 趋近无穷大时，至多需要 an^2+bn+c 的时间运行（abc 为常数）最高阶是 2 阶，忽略低阶和系数

实际问题中看问题规模，算法的复杂性，空间复杂度等情况，比如 n 的数字比较小时，可以采用 O(N^2)的算法，

10-11 快速排序算法与时间复杂性分析

1、 简单描述快速排序算法的原理和过程：(原址排序)

快速排序算法使用的是分而治之的思想，对于某个数组 A[p,q]，将其分为三部分：A[p,r-1]，A[r]，A[r+1,q]，其中，A[p,r-1]中的每个元素小于 A[r]，A[r+1,q]中的每个元素大于 A[r]。不断对 A[r]左右的两部分调用函数本身，形成递归。

```
Sort(A,p,q)
{
r=Part(A,p,q);
Sort(A[p,r-1],p,r-1);
Sort(A[r+1,q],r+1,q);
}
```

其中，

```
Part(A,p,q)
{
从 A[p,q]中选一个数 x 作为 A[r]的值；
确定 x 在 A 中的位置 r；
返回 r；
}
```

2、 计算复杂度问题

复杂度取决于 Part 中，对 x 和 r 的取值。

对于一般情况，是选取 A[p,q]中的第一个数作为 x。那么，最好的情况应该是每一次的 x 都取到当前数组的中间值，这样的话，计算复杂度属于二分递归：

T(n)=2T(n/2)+c·n，由主定理知，T(n)=O(nlgn)。最坏的情况应该是每一次的 x 都取到当前数组的最大值或者最小值，这样的话于插入排序一样，计算复杂度为 O(n^2)。平均的

情况属于概率分析，首先我们要假设数组 A 满足一个分布，再用概率分析的方法求解结算。这里我们自然认为 A 是随机排列的或者均匀分布，在计算平均情况的复杂度时，我们认定最好情况和最坏情况时交替出现的，即一次坏的情况：A[p]，A[p+1,q]，一次好的情况：A[p+1,r]，A[r]，A[r+1,q]，其中 r=(p+q+1)/2。这样的话，每一次交替的算法复杂度为：T(n)=2T(n/2)+c·n+c·(n-1)，由主定理知，T(n)=O(nlgn)，与最好的情况一样。

随机化快速排序方法，即 Part 中对 x 的选取不固定在第一个或者最后一个，而是在当前数组中随机取一个数。随机化快速排序算法的最坏情况与一般的排序算法一样，剩下我们就要评价期望计算复杂度而不是平均计算复杂度。经计算期望复杂度为 O(nlgn)。

3、 **确定性快速排序算法和随机化快速排序算法的比较**

虽然传统的快速排序算法的平均复杂度和随机化快速排序算法的期望复杂度一样，但是两个方法在本质上就不同。首先传统的快速排序算法是基于概率分析求得的，而随机化快速排序方法是将输入随机化的不确定性的方法，避免了最坏情况的构造。比如对于完全逆序或者最大值最小值交替排列的数组，传统的快速排序方法势必陷入最坏的情况，但是随机化排序算法将这种情况的发生降到最低。

Remark：其他的排序算法：插入排序、二分排序、堆排序的思路和复杂度。

12-13

1、 两个最小生成树算法(Kruskal 和 Prim)

最小生成树算法是寻找最小生成树的子集结构 A，并利用循环不等式结构不断添加权值最小的安全边，直至生成最小生成树的一种算法。根据添加安全边的方式不同，可将最小生成树算法分为 Kruskal 算法和 Prim 算法。

基本设计思路：

A=空集；

While A 不是图 G 的最小生成树

找到权值最小且对 A 安全的边(u,v)；

（安全即该边加入不会破坏 A 式最小生成树子集的结构）

A=A∪{(u,v)}；

Return A

（1）Kruskal 算法

Kruskal 算法输入图 G(V,E)和权值函数 w。先将 V 中的每个顶点独立为一个集合，然后将 E 中的所有边按照权值大小排序，接着按照权值从小到大的顺序遍历所有边，并做以下操作：判断新加入边的两个顶点是否在同一棵树下，若否，则将该边并到目标最小生成树子集 A 中。

MST-Kruskal(G,w)

A=空集；

将 V 中的每个顶点独立为一个集合(或一个无边的树)；

将 E 中的所有边按照权值从小到大的顺序进行排序；

For 按照从小到大的顺序遍历 E 中的每一条边(u,v)

If 顶点 u 和 v 分属不同的树结构

Then A=A∪{(u,v)}；

然后合并顶点集{u}，{v}；

End

End

（2）Prim 算法

Prim 算法输入图 G(V,E)，权值函数 w 和最小生成树的根结点(即初始点)r。先构造树集合 S={r}和其补集 T=V-S，再从选取链接 S 和 T 的所有边中权值最小的边(u,v)，其中$u \in S$，$v \in T$，接着将 v 并入集合 S 中，相应的，从 T 中去掉顶点 v，并将新边(u,v)并入目标最小生成树子集 A 中。按照该循环不变式以此类推，直到 T 为空集。

MST-Prim(G,w,r)
A=空集；
S={r};
T=V-S;
While T 不等于空集
　　(u,v)为链接 S 和 T 的所有边中权值最小的边，且 u∈S，v∈T；
　　S=S∪{v};
　　T=T-{v};
　　A=A∪{(u,v)};

2、如何体现贪心策略？

证明贪心结构应说明以下几点：(1) 最优化问题具有最优子结构；(2) 当前做出局部最优选择后，当前子集的最优子结构循环不变。在最小生成树问题中，最优子结构为最小生成树的子集在该子图中也是最小权值树结构，每次选取当前权值最小且安全的边(即局部最优选择)，仍然满足该循环结构。Kruskal 算法是通过遍历所有边的方式，选取当前权值最小且安全的边；Prim 算法是在当前 S 和 T 的分割边中选取当前权值最小且安全的边。殊途同归，都解决了最小生成树问题。

3、这两个算法的复杂度和依赖的数据结构？

使用普通的二叉堆的话：
Kruskal 算法计算复杂度为：O(1)+O(V)+O(E·lgE)+O(E·lgV)=O(E·lgE),
　　由于$E < V^2$，则计算复杂度为：O(E·lgV)；
Prim 算法计算复杂度为：O(1)+O(V·lgV)+O(E·lgV)=O(E·lgV)；
如果使用 Fibonacci 堆，Prim 算法的计算复杂度可以优化为：O(1)+O(V·lgV)+O(E·1)=O(E+V·lgV)。
显然在顶点个数|V|明显小于边个数|E|时，适合采用 Prim 算法。

Remark：单源最小路径搜索法 Dijkstra 算法，过程、复杂度和适用数据结构(3)。

14、　数学归纳法和递归

http://blog.csdn.net/rosekisser/article/details/4094626
数学归纳法和递归的定义。
递归，直接或者间接调用本函数语句的函数称为递归函数，其必须满足两个条件：(1) 每一次调用自己，更接近于解；(2) 必须有边界条件或者终止准则。
联系：都包含递推的思想，在计算机算法设计中相互交织；
区别：(1) 数学归纳法强调方法，递归强调应用；
　　(2) 数学归纳法是从初始值出发，类似于

F(n)
{
i=1;
s=1;

```
while i==n+1
s=s*i;
i=i+1;
return s；
}
```

递归有边界条件，设计时是反向调用自己的，如：

```
F(n)
{
If n==1 return 1;
Return n*F(n-1)；
}
```

例子是计算 n 的阶乘。
