Algorithms

算法概论

习题试解 β

吴彧文（atyuwen）

atyuwen@gmail.com

http://hi.baidu.com/atyuwen

（如有错误，敬请指正）

0 Prologue

Ex.0.1

a) $f = \Theta(g)$

b) $f = O(g)$

c) $f = \Theta(g)$

d) $f = \Theta(g)$

e) $f = \Theta(g)$

f) $f = O(g)$

g) $f = \Omega(g)$

h) $f = \Omega(g)$

i) $f = \Omega(g)$

j) $f = \Omega(g)$

k) $f = \Omega(g)$

l) $f = O(g)$

m) $f = O(g)$

n) $f = \Theta(g)$

o) $f = \Omega(g)$

p) $f = O(g)$

q) $f = O(g)$

Ex.0.2

根据等比数列求和公式：

$$S(n) = a_1 \times n, \quad if\ c = 1$$
$$S(n) = \frac{a_1(1-c^{n)}}{1-c}, \quad if\ c \neq 1$$

易得结论。

Ex.0.3

a) 数学归纳法，显然有：

$$F_6 = 8 \geq 2^{0.5\times 6} = 8$$
$$F_7 = 13 \geq 2^{0.5\times 7} \approx 11.3$$

若 $F_{n-1} \geq 2^{0.5\times(n-1)}$， $F_{n-2} \geq 2^{0.5\times(n-2)}$ 成立，则有：

$$F_n = F_{n-1} + F_{n-2} \geq 2^{0.5\times(n-1)} + 2^{0.5\times(n-2)} > 2\times 2^{0.5\times(n-2)} = 2^{0.5\times n}$$

得证。

b) 同样是数学归纳法，显然，对任意 $c > 0$，有：

$$F_1 = 1 \leq 2^c$$
$$F_2 = 1 \leq 2^{2c}$$

考虑归纳的递推过程，若 $F_{n-1} \leq 2^{c(n-1)}, F_{n-2} \leq 2^{c(n-2)}$ 成立，有：

$$F_n = F_{n-1} + F_{n-2} \leq 2^{c(n-1)} + 2^{c(n-2)} = 2^{cn} \times \left(\frac{1}{2^c} + \frac{1}{2^{2c}}\right)$$

于是只需要 $\frac{1}{2^c} + \frac{1}{2^{2c}} \leq 1$ 即可，令 $x = \frac{1}{2^c}$，即有 $x^2 + x - 1 \leq 0$，于是有：

$$\frac{-1-\sqrt{5}}{2} \leq x \leq \frac{-1+\sqrt{5}}{2}$$

将 $x=\frac{1}{2^c}$ 代入上式，可解得：$c \geq \log_2 \phi,$ 其中 $\phi=\frac{1+\sqrt{5}}{2}$，为黄金分割率，从这里也可以看到 Fibonacci 数列与黄金分割之间的奇妙联系。用计算器可以算得 $c_{\min} \approx 0.694$。

c) 即上面的 $c_{\min} \approx 0.694$

Ex.0.4

a) 两个 2×2 的矩阵相乘，仍然得一个 2×2 的矩阵，其中计算每个元素都分别需要两次乘法和一次加法，所以总共需要 8 次乘法和 4 次加法。

b) 分治，若 n 为偶数，有 $\mathrm{X}^{\mathrm{n}}=\left(\mathrm{X}^{n/2}\right)^2$，若 n 为奇数，有 $\mathrm{X}^{\mathrm{n}}=\mathrm{X}\bullet\mathrm{X}^{\mathrm{n\text{-}1}}$。显然计算 X^{n} 只需要 $O(\log n)$ 次矩阵乘法。

c) 若某次的中间结果为 $\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$，则有：

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\times\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}=\begin{bmatrix} a_{12} & a_{11}+a_{12} \\ a_{21} & a_{21}+a_{22} \end{bmatrix}$$

由上式可知，任意中间结果矩阵，与 X 相乘后，其最大元素不会大于原先的最大元素的 2 倍，所以可得，所有的中间结果都是 $O(2^n)$ 的，因此 bit 数是 $O(n)$ 的。

d) $O(\log n)$ 次矩阵乘法，每次相乘为 $M(n)$，当然总共是 $O(M(n)\log(n))$。

e) 设总的计算次数为 C_n，则有以下递归式：

$$C_n=C_{n/2}+M(n)$$

因 $M(n)=O(n^2)$，由**主定理**知 $C_n=O(M(n))$。

1 Algorithms with numbers

Ex.1.1

n 位数所能表示的最大值为 b^n-1，根据下列关系式：

$$b^2-1\geq 3(b-1)=3b-3\Leftrightarrow (b-1)(b-2)\geq 0$$

知，当 $b\geq 2$ 时，有 3 个 1 位数的和不超过 2 位。

Ex.1.2

因为 $2^4=16>10$，所以可知一个数的 2 进制位数不超过 10 进制位数的 4 倍。

设 b 为 2 进制表示的位数，d 为 10 进制表示的位数，则有：

$$\frac{b}{d}\approx\frac{\log_2 n}{\log_{10} n}=\log_2 10\approx 3.322$$

Ex.1.3

设只含根节点的树的高度为 1，则高度为 h 的满树的节点数为：

$$N(h)=1+d+d^2+\cdots d^{h-1}=\frac{1-d^h}{1-d}$$

于是由 $N(h)\geq n\Rightarrow h=\Omega(\log_d n)$。

准确表示的话有：$h=\lceil \log_d(dn-n+1)\rceil$。

Ex.1.4

Hint 很强大：

$$\log(n!)=O(\log(n^n))=O(n\log n)$$

$$\log(n!)=\Omega\left(\log\left(\left(\frac{n}{2}\right)^{n/2}\right)\right)=\Omega\left(\frac{n}{2}\bullet\log\frac{n}{2}\right)=\Omega(n\log n)$$

于是有：$\log(n!)=\Theta(n\log n)$。

Ex.1.5

这题的 Hint 更强大，考虑调和级数 $H=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}+\frac{1}{8}\cdots$

将每个分母约束到不大于它的 2 的幂形式，有：

$$H=O\left(H_u\right)=\frac{1}{1}+\frac{1}{2}+\frac{1}{2}+\frac{1}{4}+\frac{1}{4}+\frac{1}{4}+\frac{1}{4}+\frac{1}{8}\cdots$$

将这个新的级数想象成一颗二叉树的层序遍历，则每层和为 1，由 Ex.1.3 我们知道树的高度是 $\Theta\left(\log n\right)$ 的，即 H_u 也是 $\Theta\left(\log n\right)$，于是 $H=O\left(\log n\right)$。

然后，将每个分母约束到不小于它自身的 2 的幂，有：

$$H=\Omega\left(H_l\right)=\frac{1}{1}+\frac{1}{2}+\frac{1}{4}+\frac{1}{4}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}\cdots$$

忽略到第一项 1，后面又是一颗二叉树的层序遍历，且每层和为 1/2，以上同理可得，$H=\Omega(\log n)$，于是得到 $H=\Theta(\log n)$。

Ex.1.6

设有两二进制数 a，b，其中 $a=a_n a_{n-1}\cdots a_3 a_2 a_1 a_0$，则有：

$$a\times b=\sum_{i=0}^{n}\left(a_i\times b\right)\times 2^n=\sum_{i=0}^{n}\left(a_i\times b\right)<<n$$

这正是 Grade-School 方法的数学意义，于是得证。

Ex.1.7

设 $T\left(n,m\right)$ 为计算 n 位数乘 m 位数所需步数，且假设 $n\geq m$，（如果不满足的话交换一下两个乘数即可），则有：

$$T\left(n,m\right)=T\left(n,m-1\right)+O\left(n\right)$$

后面的 $O\left(n\right)$ 包含了一些不超过 $n+m$ 位数的加法和移位操作，（前面之所以需要 $n\geq m$ 就是为了保证在这里的所有操作都是 $O\left(n\right)$ 的，因为 $n+m\leq 2n$），于是可以得到：$T\left(n,m\right)=O\left(nm\right)$。

Ex.1.8

关于正确性：

a) 当 $x=0$ 时该算法显然是正确的。

b) 因 $r<y$，所以 $2r+1<2y$ 总是成立。

c) 当 x 为偶数时，有 $x=\left\lfloor \frac{x}{2} \right\rfloor \times 2=(yq+r)\times 2=2yp+2r$

d) 当 x 为奇数时，有 $x=\left\lfloor \frac{x}{2} \right\rfloor \times 2+1=(yq+r)\times 2+1=2yp+2r+1$

希望我已经“证明”了它的正确性了，下面来看时间复杂度：总共迭代 n 次，每次迭代需要 $O(n)$ 次操作，于是 $T(n)=O(n^2)$。

Ex.1.9

根据同余的定义，有：

$$x\equiv x' \bmod N \Rightarrow x=x'+rN$$
$$y\equiv y' \bmod N \Rightarrow y=y'+sN$$

于是：

$$x+y=x'+y'+(r+s)N \Rightarrow x+y\equiv x'+y' \bmod N$$
$$x\times y=x'y'+(sx'+ry'+rsN)N \Rightarrow xy\equiv x'y' \bmod N$$

Ex.1.10

$$a\equiv b \bmod N \Rightarrow a=b+rN=b+rsM \Rightarrow a\equiv b \bmod M$$

Ex.1.11

一般解这种题的思路就是要想方设法的凑 1，注意到：$6\times 6=35+1$，有：

$$4^{1536}\equiv 64^{512}\equiv(-6)^{512}\equiv 36^{256}\equiv 1^{256}\equiv 1 \pmod{35}$$

$$9^{4824}\equiv(9\times 9\times 9)^{1608}\equiv(-6)^{1608}\equiv 36^{804}\equiv 1^{804}\equiv 1 \pmod{35}$$

即：$4^{1536}-9^{4824}\equiv 1-1\equiv 0 \pmod{35}$。

Ex.1.12

这题很简单：

$$2^{2^{2006}} \equiv 4^{2^{2005}} \equiv 1^{2^{2005}} \equiv 1 \pmod 3$$

Ex.1.13

同样是凑 1，注意到：$5\times 6 = 31-1$，有：

$$\begin{aligned} 5^{30000} &\equiv 5^{20000}\times 5^{10000} \equiv 25^{10000}\times 5^{10000} \equiv (-6)^{10000}\times 5^{10000} \\ &\equiv (-30)^{10000} \equiv 1^{10000} \equiv 1 \pmod{31} \end{aligned}$$

$$\begin{aligned} 6^{123456} &\equiv 6^{82304}\times 6^{41152} \equiv 36^{41152}\times 6^{41152} \equiv 5^{41152}\times 6^{41152} \\ &\equiv (30)^{41152} \equiv (-1)^{41152} \equiv 1 \pmod{31} \end{aligned}$$

即：$5^{30000} - 6^{123456} \equiv 1-1 \equiv 0 \pmod{31}$。

Ex.1.14

采用 Ex.0.4 中的矩阵形式，但是对每一步的中间结果都进行模 p 操作，这样使得所有的中间结果都小于 p。每一步的乘法以及求模运算均是 $O\left((\log p)^2\right)$，于是得到总的时间复杂度为：

$$T(n,p) = O\left(\log n(\log p)^2\right)$$

如果 p 为一常数，则有：$T(n) = O(\log n)$。

Ex.1.15

欲使得 $ax \equiv bx \mod c \Rightarrow a \equiv b \mod c$ 成立，只需要保证 $x^{-1} \bmod c$ 存在：

$$ax \equiv bx \mod c \Rightarrow axx^{-1} \equiv bxx^{-1} \mod c \Rightarrow a \equiv b \mod c$$

由书可知，$x^{-1} \bmod c$ 存在的条件是 $\gcd(x,c) = 1$，即：x，c 互素。

另外可以看出来，这个条件其实是充分必要的，下面来换个方式证明，设 $ax \equiv bx \mod c$，则有：

$$ax = bx + ck \Rightarrow a = b + c\left(\frac{k}{x}\right)$$

在上式中，$c\left(\frac{k}{x}\right)$必为整数。若$x$，$c$互素,则有$\left(\frac{k}{x}\right)$必为整数，于是得出$a \equiv b \bmod c$。若$x$，$c$不为互素，此时$\left(\frac{k}{x}\right)$不一定为整数，比如可以取$k = \frac{x}{\gcd(c,x)}$，得$\frac{k}{x} = \frac{1}{\gcd(c,x)}$，当然这时$a$，$b$不一定模$c$同余。必要性得证。

Ex.1.16

需要注意的是，迭代乘方算法在遇到偶数次幂时，只需一次乘法就能将问题的规模减半，而遇到奇数次幂时则需要两次。考虑类似于$p = 2^n - 1$的幂次，它们的二进制表示全为 1，这样的话每一次迭代都将付出 2 次乘法的代价，比如取$p = 2^4 - 1 = 15$，则有迭代系列为：

$$a^{15} = \left(\left(a^2 \times a\right)^2 \times a\right)^2 \times a$$

总共需要 6 次乘法。如果采取下面的迭代系列：

$$a^{15} = \left(\left(a^2\right)^2 \times a\right)^3$$

则只需要 5 次。但是从算法设计和平均效率上来看，迭代乘方算法无疑是最优的。

Ex.1.17

算法 1，$x^y = x \times x^{y-1}$：

总共$O(y)$次迭代，设当前迭代步数为i，则在当前迭代中乘法的代价为$O(i \log x \times \log x)$，所以，总的时间复杂度为：$\sum_{i=1}^{y} i(\log x)^2 = O\left((y \log x)^2\right)$。

算法 2，$x^y = \left(x^{y/2}\right)^2 \times x^{y \bmod 2}$：

迭代次数为 $O(\log y)$，同样设当前迭代步数为 i，则在当前迭代中乘法的代价为 $O\left(2^{i-1}\log x \times 2^{i-1}\log x\right)$，总时间复杂度为：$\sum_{i=1}^{\log y} 2^i (\log x)^2 = O\left(y(\log x)^2\right)$。

Ex.1.18

算法 1，素因子分解：

$$210 = 2\times 3\times 5\times 7$$
$$588 = 2\times 2\times 3\times 7\times 7$$

于是得 $\gcd(210,588) = 2\times 3\times 7 = 42$。

算法 2，欧几里得辗转相除法：

$$\gcd(588,210) = \gcd(210,168) = \gcd(168,42) = \gcd(42,0) = 42$$

Ex.1.19

数学归纳法，显然有 $\gcd(F_2, F_1) = \gcd(1,1) = 1$。若有 $\gcd(F_n, F_{n-1}) = 1$，则：

$$\gcd(F_{n+1}, F_n) = \gcd(F_n + F_{n-1}, F_n) = \gcd(F_n, F_{n-1}) = 1$$

于是得证。

Ex.1.20

用扩展欧几里得算法可得：

$$20^{-1} = 4 \pmod{79}$$
$$3^{-1} = 21 \pmod{62}$$
$$21^{-1} = \ldots \pmod{91} \Leftarrow \gcd(91,21) = 7$$
$$5^{-1} = 14 \pmod{23}$$

Ex.1.21

在 $[0,11^3)$ 内，与 $11^3 = 1331$ 互素的整数个数为：$11^3 - 1 - (11^2 - 1) = 1210$。（注意到除所有 11 的倍数以外，其余的非 0 整数均与 1331 互素）。

Ex.1.22

因互素是相互的，所以当 a， N 均大于 1 时，有：

$$a^{-1} \bmod N \, exist \Rightarrow N^{-1} \bmod a exist$$

Ex.1.23

若 $ax \equiv ay \equiv 1 \bmod N$，则有：

$$axx \equiv ayx \bmod N \Rightarrow x \equiv y \bmod N$$

于是得证。

Ex.1.24

即是 Ex.1.21 的推广版，答案为： $p^n - 1 - \left(p^{n-1} - 1\right) = p^n - p^{n-1}$ 。

Ex.1.25

注意到 $2^7 \equiv 1 \bmod 127$,有 $2^{125} \equiv 2^{125 \bmod 7} \equiv 2^6 \equiv 64 \bmod 127$ 。

另外，根据 Hint：127 is prime，可以考虑使用素数的性质来求解，由费马小定理知：

$2^{126} \equiv 1 \equiv 2 \times 64 \bmod 127$，再根据除法定理，得： $2^{125} \equiv 64 \bmod 127$ 。

Ex.1.26

这道题的 Hint 似乎不太严密，由 Section1.4.2 我们知道： $x^{1+(p-1)(q-1)} \equiv x \bmod pq$，要得到 Hint 中的同余关系，还需要保证 x 与 pq 互素，即：

$$\gcd\left(x, pq\right) = 1 \Rightarrow x^{(p-1)(q-1)} \equiv 1 \bmod pq$$

令 $p = 2, q = 5$，则 $pq = 10, \left(p-1\right)\left(q-1\right) = 4$，有：

$$17^{17^{17}} \equiv 17^{\left(17^{17}\right) \bmod 4} \equiv 17 \equiv 7 \bmod 10$$

即个位数字为 7。

Ex.1.27

$$d = e^{-1} \bmod (p-1)(q-1) = 3^{-1} \bmod (352) = 235$$

$$M(e) = M^{e} \bmod N = 41^{3} \bmod 391 = 105$$

Ex.1.28

在这里$(p-1)(q-1)=60$，所以e就不能选 3 了，为了使加密过程尽可能快速，e当然是要越小越好，比如可以选$e=7 \Rightarrow d = e^{-1} \bmod 60 = 43$。

Ex.1.29

a) 是全域的，证明可参考书中过程。随机位需要$2\log m$。

b) 不是。假设有$h_{a_1,a_2}(x_1,x_2) = h_{a_1,a_2}(y_1,y_2)$，也即是有$a_1(x_1-y_1) \equiv a_2(y_2-x_2) \bmod m$，因$m=2^k$，这时$y_2-x_2$不一定存在唯一逆元。作为一个试探的特例，我们假设$x_1-y_1=0$，$y_2-x_2=2$，这时有$2a_2 \bmod m = 0$，当$a_2=0$时和$a_2=\frac{m}{2}$时该式均成立，于是可知这时$\Pr\{h_{a_1,a_2}(x_1,x_2) = h_{a_1,a_2}(y_1,y_2)\} > \frac{1}{m}$，所以这个散列函数族是非全域的。

c) 是全域的么？可以把整个散列看作是由两个部分组成，首先是将$[m-1] \to [m-1]$，然后再单独将$(m-1) \to [m-1]$。如果$i, j \in [m-1]$，由对称性知$\Pr\{h(i) = h(j)\} = \frac{1}{m-1}$，如果$i \in [m-1], j = m-1$，则也可以由对称性得到将$j$散列为$h(i)$的概率为$\frac{1}{m-1}$，所以该函数族好像是全域的。当然，这里的对称性只是我的一厢情愿而已，要想获得令人信服的证明似乎应该构造出函数集之间的某种一一映射才行。

Ex.1.30

a) 归并，将 n 个数分成 $\left\lceil \frac{n}{2} \right\rceil$ 组，分别将每组相加得到新的 $\left\lceil \frac{n}{2} \right\rceil$ 个数，然后重复这个过程，直到最后得到一个最终结果。组间是可以并行的，整个过程其实是一个树状结构，高度为 $O(\log n)$，每层自身的深度为 $O(\log m)$，于是总的深度为 $O(\log n \log m)$。

b) 如下式，其中 $\oplus$ 表示异或，xy 表示 x, y 按位与，$+$ 表示按位或，<<表示左移：

$$r = x \oplus y \oplus z$$
$$s' = xy + yz + xy$$
$$s = s' << 1$$

c) 上面这个 trick 的意义就在于，它能将 3 个数相加的操作归约为 2 个数相加，而且归约的过程可以很好的并行化，只需要一个 depth 就能完成。于是，类似于问题 a)，我们可以将 n 个数为成 $\left\lceil \frac{n}{3} \right\rceil$ 组，将每组分别约化成两个数相加，再将得到的加数继续约化，直到最后只剩两个数相加。整个归并过程的 depth 是 $O(\log_{3/2} n) = O(\log n)$ 的，再加上最后的那个加法为 $\log n$，所以总的深度也为 $O(\log n)$。

Ex.1.31

a) $Num_{bits} = \Theta(n \log n)$，详见 Ex.1.4。

b) 采用常规的迭代算法，有：

$$T(n) = O\left(\sum_{i=1}^{n} i \log i \times \log i \right) = O\left((\log n)^2 \sum_{i=1}^{n} i \right) = O\left((n \log n)^2 \right)$$

Ex.1.32

a) 可以用二分法来找整数根，时间复杂度为$O(\log N)$，或者用牛顿迭代法：

$$q^2 - N = 0 \Rightarrow q_{n+1} = q_n - \frac{{q_n}^2 - N}{2q_n} = \frac{{q_n}^2 + N}{2q_n}$$

一般可以选初值$q_0 = \left\lceil \frac{N}{2} \right\rceil$进行迭代，$q_{n+1} = \left\lfloor \frac{{q_n}^2 + N}{2q_n} \right\rfloor$，直到$q_{n+1} \geq q_n$时停止迭代，这时再验证$q_n^2 == N$即可。当$N$很大时，这个迭代的时间复杂度近似于$O(\log N)$，但是牛顿迭代法在根附近收敛很快，所以效率应该要好过二分法。

b) 若$N \neq 1$,则有$q \geq 2$，于是有：

$$N = q^k \geq 2^k \Rightarrow k \leq \log N$$

c) 依次用$2 \leq k \leq \log N$去试探，每次试探需要迭代$\log N$次，同时每次迭代时计算k次方需要$O(\log k)$，所以总共为：

$$\begin{aligned} T(N) &= O\left(\sum_{k=2}^{\log N} \log N \log k \right) \\ &= O\left(\log N \sum_{k=2}^{\log N} \log k \right) \\ &= O\left(\log N \log\left((\log N)! \right) \right) \\ &= O\left(\log N \log N \log(\log N) \right) = O\left((\log N)^2 \log(\log N) \right) \end{aligned}$$

Ex.1.33

注意到$lcm(x,y) = \frac{xy}{\gcd(x,y)}$，所以时间复杂度为：$O(n^2) + O(n^3) = O(n^3)$。

Ex.1.34

若前 $i-1$ 次均为字，第 i 次即掷得人头，这个事件发生的概率 $P(i)=(1-p)^{i-1}p$，

所以 $E=\sum_{i=1}^{\infty} i\times P(i)=\sum_{i=1}^{\infty} i(1-p)^{i-1}p$，利用等差比法求和，有：

$$(1-p)E=\sum_{i=1}^{\infty} i(1-p)^{i}p$$

$$E-(1-p)E=p+\sum_{i=1}^{\infty}(1-p)^{i}p=p+\frac{(1-p)p}{p}=1$$

$$\Rightarrow pE=1\Rightarrow E=\frac{1}{p}$$

Ex.1.35

a) 显然 1 是，另外，由于 $(p-1)^2=p(p-2)+1$，所以 $p-1$ 也是。

b) $(p-1)!=1\times 2\times 3\times\cdots\times(p-2)\times(p-1)$，共 $p-1$ 个因子相乘。显然地，对于所有素数，这 $p-1$ 个因子除于首尾的 1 和 $p-1$ 之后，必然还剩偶数个因子。设剩下的这偶数个因子的集合为 S，如果 S 不为空，由互逆的对称性可以知道，S 中的任何元素 e 的逆元（$e^{-1}\bmod p$）也在 S 中，这样构成了 $\frac{\|S\|}{2}$ 个互逆对。所以 S 中所有元素的乘积模 p 为 1，于是有：

$$\begin{aligned}(p-1)!&\equiv 1\times 2\times 3\times\cdots\times(p-2)\times(p-1)\\&\equiv 1\times(p-1)\\&\equiv -1 \bmod p\end{aligned}$$

c) 反证法，如果 $(N-1)!\equiv -1 \bmod N$，则有 $(N-1)!\times(-1)\equiv 1 \bmod N$，即 $(N-1)!$ 存在逆元。这需要 $\gcd((N-1)!,N)=1$，而如果 N 不为素数的话

$\gcd\left((N-1)!, N\right)$ 必大于 1，于是出现矛盾。从而得证。

d) 设 N 的二进制位数为 n，在用费马小定理测试素性时只需计算 a^{N-1}，因此只需要 $O(n)$ 次乘法。而用本定理则需要 $O\left(2^n\right)$ 次，显然是太慢了。

Ex.1.36

a) $p \equiv 3 \bmod 4 \Rightarrow p+1 \equiv 0 \bmod 4 \Rightarrow p+1 = 4k$。

b) 首先，化简一下有：$\left(a^{(p+1)/4}\right)^2 = a^{(p+1)/2} = a \times a^{(p-1)/2}$。由费马小定理知 $\left(a^{(p-1)/2}\right)^2 \equiv 1 \bmod p$，再由 Ex.1.35.a 的结论可得：$a^{(p-1)/2} \equiv 1 \bmod p$ 或 $a^{(p-1)/2} \equiv p-1 \bmod p$。对于这两种情况，如果 $a^{(p-1)/2} \equiv 1$ 成立的话，当然有 $\left(a^{(p+1)/4}\right)^2 \equiv a \times a^{(p-1)/2} \equiv a \bmod p$，但是如果 $a^{(p-1)/2} \equiv p-1 \bmod p$ 呢？

假设 $a^{(p-1)/2} \equiv p-1 \bmod p$ 时，存在 $x^2 \equiv a \bmod p$，将后式代入前式得：$\left(x^2\right)^{(p-1)/2} \equiv (x)^{(p-1)} \equiv p-1 \bmod p$，这与费马小定理冲突，所以可知，当 $a^{(p-1)/2} \equiv p-1 \bmod p$，$a$ 的模 p 平方根是不存在的。终上所述，如果 $a^{(p-1)/2} \equiv 1$，那么 a 的模 p 平方根为 $\left(a^{(p+1)/4}\right)^2$，否则 a 的模 p 平方根不存在。

Ex.1.37

a) 列表如下，似乎题目要求是要竖表，不过也没关系了☺。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
0	1	2	3	4	0	1	2	3	4	0	1	2	3	4

规律是：在上图中，以颜色标识的二元组各不相同，且遍历了 $\left[\{0,1,2\},\{0,1,2,3,4\}\right]$ 所有可能的排列。

b) 对于二元组 $\{j,k\}$，如果同时存在 i_1, i_2 使得 $i_n \equiv j \bmod p,\ i_n \equiv k \bmod q$，则有：

$$
\begin{aligned}
&i_1 \equiv j \bmod p,\ i_1 \equiv k \bmod q \\
&i_2 \equiv j \bmod p,\ i_2 \equiv k \bmod q \\
&i_1 - i_2 \equiv 0 \bmod p,\ i_1 - i_2 \equiv 0 \bmod q \\
&\Rightarrow i_1 - i_2 \equiv 0 \bmod pq
\end{aligned}
$$

因 i_1, i_2 均小于 pq，所以必有 $i_1 - i_2 = 0$，唯一性得证。如果把 Φ 看到是 $\{i\}$ 到 $\{(j,k)\}$ 的一个映射，因为元素个数相等，每个 i 均对应到一个 (j,k)，且由上面可知不存在多对一的情况，所以 Φ 必是一一映射。

c) 公式好长：$i = \left\{ jq\left(q^{-1} \bmod p\right) + kp\left(p^{-1} \bmod q\right) \right\} \bmod pq$，当然有 $0 \le i < pq$，另外注意到 $(n \bmod pq) \equiv n \bmod p,\ (n \bmod pq) \equiv n \bmod q$，得：

$$
i \equiv j \bmod p,\ i \equiv k \bmod q
$$

d) 对于素数 $p_1, p_2, p_3, \cdots p_n$，有 $0 \le i < p_1 p_2 p_3 \cdots p_n$，与 n 元组 $(j_1, j_2, j_3, \cdots, j_n)$ 形成一一对应，其中 $j_m = i \bmod p_m, m \in [1,n]$。另有：

$$
i = \left\{ \sum_{m=1}^{n} j_m \bullet w_m \bullet \left(w_m^{-1} \bmod p_m \right) \right\} \bmod \prod_{m=1}^{n} p_m,\ w_m = \prod_{l=1, l \ne m}^{n} p_l
$$

Ex.1.38

a) 直觉告诉我，这里的 r 只需满足两个条件：

1） $10^r - 1 > p$，即是保证存在 r 位的十制数比 p 大。

2） $10^r \equiv 1 \bmod p$，即是保证对任意 $i, (0 < i < 9)$，有 $i \equiv i \times 10^r \bmod p$。

以上只是直觉性的猜测，所以这里就不详细证明或证伪了。于是可以算得，当 $p = 13$ 时，$r = 6$，因为 $10^6 \equiv 1 \bmod 13$。当 $p = 17$ 时，$r = 16$，因为 $10^{16} \equiv 1 \bmod 17$。

b) 由费马小定理知 $10^{p-1} \equiv 1 \bmod p$，所以 r 最大只能是 $p-1$，此外 r 也可能在

$\frac{p-1}{n}$ 处，也只可能在 $\frac{p-1}{n}$ 处取得。为嘛？设 t 是最小的使得 $10^t \equiv 1 \bmod p$ 的整数，且 $t \neq \frac{p-1}{n}$，由费马小定理 $10^{p-1} \equiv 1 \bmod p$ 知，必然有 $10^{(p-1) \bmod t} \equiv 1 \bmod p$。于是 $(p-1) \bmod t$ 也满足条件，这与 t 是最小的矛盾。从而得知，r 只可能在 $\frac{p-1}{n}$ 处取得。

Ex.1.39

利用费马小定理 $a^{p-1} \equiv 1 \bmod p$，有：$a^{b^c} \equiv a^{(b^c) \bmod p-1} \bmod p$，两次 Modular Exponentiation 操作，如果 a, b, c, p 是 n 位数，则时间复杂度为 $O(n^3)$。

Ex.1.40

反证法：如果存在 $x^2 \equiv 1 \bmod p, x \neq \pm 1 \bmod p$，且 p 为素数，则有：

$$(x \bmod p)^2 - 1 \equiv 0 \bmod p \Leftrightarrow (x \bmod p+1)(x \bmod p-1) \equiv 0 \bmod p$$

因为 $x \neq \pm 1 \bmod p$，所以可知 $1 \leq x \bmod p-1 < x \bmod p+1 < p$，又 p 为素数，知 $x \bmod p \pm 1$ 与 p 互素，于是由 $(x \bmod p+1)(x \bmod p-1) \equiv 0 \bmod p$，根据除法定理得：$x \bmod p \pm 1 \equiv 0 \bmod p$，显然矛盾。于是 p 只能是合数。

Ex.1.41

a) 如果 a 是一个非零的 quadratic residue 模 N，即存在 $a \equiv x^2 \bmod N$，显然同时也应该有 $a \equiv (N-x)^2 \equiv (-x)^2 \bmod N$，$N$ 是奇数所以 $x \neq N-x$。此外，除了 $x, N-x$ 之外还有没有可能存在别的数 y，使得 $a \equiv y^2 \bmod N$ 呢，设存在：

$$x^2 \equiv y^2 \bmod N \Leftrightarrow [(x+y) \bmod N][(x-y) \bmod N] \equiv 0 \bmod N$$

其中 $x+y\neq N, x\neq y$。类似于 Ex.1.40，根据除法定理可推出矛盾。所以可得，满足 $a\equiv x^2 \bmod N$ 的 x 有且恰好只有两个。

b) 首先 0 是一个，除去 0 之外每一对 x 与 $N-x$ 可得一个 quadratic residue 模 N，所以总共有 $1+\frac{N-1}{2}=\frac{N+1}{2}$ 个。

c) 在这一问中 N 可以不必是素数了，取 $a=9, N=16$，注意 $16=(5+3)(5-3)$，有 $9\equiv 3^2\equiv 5^2\equiv 13^2\equiv 11^2 \bmod 16$。

Ex.1.42

对 e 的要求是与 $(p-1)$ 互素，先求出 $d=e^{-1} \bmod (p-1)$，则有 $(m^e)^d \equiv m^{ed \bmod (p-1)} \equiv m \bmod p$，时间复杂度为 $O(n^3)$。

Ex.1.43

设 $N=pq$，$C=(p-1)(q-1)$，则有 $ed\equiv 1 \bmod C$，代入 $e=3$，即 $3d=kC+1 \Rightarrow k=\frac{3d-1}{C}$。因为 $0<d<C$，于是 $k=1,2$。利用 k 值试探计算得 C，由 N, C 可容易解得 p, q。

Ex.1.44

设 $E_i=M^{e_i} \bmod N_i$，其中 $e_i=3$，$N_i=p_iq_i$。则 $M^3\equiv E_i \bmod p_i$，$M^3\equiv E_i \bmod q_i$。由中国剩余定理知，在 $\prod_{i=1}^{3} p_iq_i=\prod_{i=1}^{3} N_i$ 范围内，这样的 M^3 是存在且唯一的。若：

$$S = E_1 \times N_2 N_3 \left\{ \left(N_2 N_3 \right)^{-1} \bmod N_1 \right\}$$
$$+ E_2 \times N_1 N_3 \left\{ \left(N_1 N_3 \right)^{-1} \bmod N_2 \right\}$$
$$+ E_3 \times N_1 N_2 \left\{ \left(N_1 N_2 \right)^{-1} \bmod N_3 \right\}$$

则 $M^3 = \left(S \bmod \prod_{i=1}^{3} N_i \right)$ 满足条件 $M^3 \equiv E_i \bmod q_i$，且唯一。开立方即得到 M 。

Ex.1.45

a) 略，如果不清楚请自行 Google.

b) 利用 $\left(M^d \right)^e \equiv M \bmod N$ 来进行验证过程。正确性证明和安全性分析同 RSA。

c) 再略。

d) $N = 391 = 17 \times 23$，所以 $d = \left\{ 17^{-1} \bmod \left(16 \times 22 \right) \right\} = 145$ 。

Ex.1.46

a) 将密文 $M^e \bmod N$ 交给 *Bob* 进行数字签名后为 $\left(M^e \right)^d \bmod N$，即得原文 M 。

b) 为了打乱密文，可以选择一个与 N 互素的随机数 r，计算出 $r^e \bmod N$，再将 $M^e r^e$ 发给 *Bob* 进行数字签名，得到 $\left(M^e r^e \right)^d \equiv Mr \bmod N$，于是可解得原文 $M \equiv Mr \times r^{-1} \bmod N$ 。

2 Divide-and-conquer Algorithms

Ex.2.1

$$10011011\times10111010=2^8\times P_{11}+2^4\left(P_{13}-P_{11}-P_{12}\right)+P_{12}$$
$$P_{11}=1001\times1011=2^4\times P_{21}+2^2\left(P_{23}-P_{21}-P_{22}\right)+P_{22}$$
$$P_{12}=1011\times1010=2^4\times P_{31}+2^2\left(P_{33}-P_{31}-P_{32}\right)+P_{32}$$
$$P_{13}=10100\times10101=2^4\times P_{41}+2^2\left(P_{43}-P_{41}-P_{42}\right)+P_{42}$$
$$\vdots$$
$$P_{21}=4,P_{22}=3,P_{23}=15$$
$$P_{31}=4,P_{32}=6,P_{33}=20$$
$$P_{41}=25,P_{42}=0,P_{43}=30$$
$$\Rightarrow P_{11}=99,P_{12}=110,P_{13}=420$$
$$\Rightarrow 10011011\times10111010=28830$$

Ex.2.2

因为 $\log_b bn=\log_b b+\log_b n=1+\log_b n$，于是区间 $\left[\log_b n,\log_b bn\right]$ 内必存在一个整数 k，且当然有 $b^k\in\left[n,bn\right]$。

Ex.2.3

a) 展开递归式有：

$$\begin{aligned}T\left(n\right)&\le 3T\left(n/2\right)+cn\\&\le 3\left(3T\left(n/4\right)+cn/2\right)+cn=9T\left(n/4\right)+\left(1+3/2\right)cn\\&\le 9\left(3T\left(n/8\right)+cn/4\right)+\left(1+3/2\right)cn=27T\left(n/8\right)+\left(1+3/2+\left(3/2\right)^2\right)cn\\&\vdots\end{aligned}$$

即 $T\left(n\right)\le 3^kT\left(n/2^k\right)+\sum_{i=0}^{k-1}\left(3/2\right)^icn$，代入 $k=\log_2 n$，得：

$$T\left(n\right)\le 3^{\log_2 n}\times1+2\left(\left(3/2\right)^{\log_2 n}-1\right)=O\left(n^{\log_2 3}\right)$$

b) 直接展开可得关系式$T(n) \le T(n-k)+ck$，代入$k=n-1$，得：

$$T(n) \le c(n-1)+O(1)=O(n)$$

Ex.2.4

$$A: T(n)=5T(n/2)+O(n) \Rightarrow T(n)=O\left(n^{\log_2 5}\right) \approx O\left(n^{2.32}\right)$$

$$B: T(n)=2T(n-1)+O(1) \Rightarrow T(n)=O\left(2^n\right)$$

$$C: T(n)=9T(n/3)+O\left(n^2\right) \Rightarrow T(n)=O\left(n^2 \log n\right)$$

当然是算法C最优。

Ex.2.5

a) $T(n)=O\left(n^{\log_3 2}\right) \approx O\left(n^{0.63}\right)$

b) $T(n)=O\left(n^{\log_4 5}\right) \approx O\left(n^{1.16}\right)$

c) $T(n)=O(n \log n)$

d) $T(n)=O\left(n^2 \log n\right)$

e) $T(n)=O\left(n^3 \log n\right)$

f) $T(n)=O\left(n^{3/2} \log n\right)$

g) $T(n)=O(n)$

h) $T(n)=O\left(n^{c+1}\right)$

i) $T(n)=O\left(n^{\log_4 5}\right) \approx O\left(n^{1.16}\right)$

j) $T(n)=O\left(c^n\right)$

k) $T(n)=O(\log \log n)$

Ex.2.6

a) 略。

b) $c(t)=\sum_{i=0}^{t}a(i)b(t-i)$，观察书中图形有：

$$\begin{cases}a(i)=1/t_0 & (0\le i\le 11)\\ a(i)=0 & (i>11)\end{cases}$$

$$\begin{cases}b(0)=1 & \\ b(i)=0 & (i\ne 0)\end{cases}$$

于是得对应多项式 $P=1/t_0\sum_{i=0}^{11}x^i$ 。

Ex.2.7

1. Sum：当 $n=1$ 时， $sum=1$，否则 $sum=\sum_{k=0}^{n-1}\omega^k=\frac{\omega^n-1}{\omega-1}=0$ 。

2. Product：$product=\prod_{k=0}^{n-1}\omega^k=\omega^{n(n-1)/2}=e^{i(n-1)\pi}$，于是当 n 为奇时，$product=1$。

当 n 为偶时， $product=-1$ 。

Ex.2.8

a) 第一问， $(1,0,0,0)\xrightarrow{FFT}(1,1,1,1)$ ；第二问， $\omega=i$ ；第三问，

$$(1,0,0,0)\xleftarrow{FFT}\left(\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{4}\right)。$$

b) 如下式所示：

$$(1,0,1,-1)\xrightarrow{FFT}(1,i,3,-i)$$

$$(1,0,1,-1)\xleftarrow{FFT}\left(\frac{1}{4},-\frac{i}{4},\frac{3}{4},\frac{i}{4}\right)$$

Ex.2.9

a) 令 $A(x)=x+1=(1,1,0,0)$，$B(x)=(1,0,1,0)$，再令 $\omega=i$，进行快速傅利叶变换有：$A(x)\xrightarrow{FFT}(2,1+i,0,1-i)$，$B(x)\xrightarrow{FFT}(2,0,2,0)$，于是得 $C'(x)=A'(x)B'(x)=(4,0,0,0)$，所以 $C'(x)\xleftarrow{FFT}C(x)=(1,1,1,1)$。

b) 此时可令 $A(x)=(1,1,2,0)$，$B(x)=(2,3,0,0)$。由 $\omega=i$，进行快速傅利叶变换有：$A(x)\xrightarrow{FFT}(4,i-1,2,-i-1)$，$B(x)\xrightarrow{FFT}(5,2+3i,-1,2-3i)$，于是得 $C'(x)=A'(x)B'(x)=(20,-i-5,-2,i-5)$，再进行逆变换得 $C'(x)\xleftarrow{FFT}C(x)=(2,5,7,6)$。

Ex.2.10

构造 5 节点 Lagrange 插值多项式有：

$$\begin{aligned}L_4(x)&=y_0l_0(x)+y_1l_1(x)+y_2l_2(x)+y_3l_3(x)+y_4l_4(x)\\&=2\times\frac{(x-2)(x-3)(x-4)(x-5)}{(1-2)(1-3)(1-4)(1-5)}\\&\quad+1\times\frac{(x-1)(x-3)(x-4)(x-5)}{(2-1)(2-3)(2-4)(2-5)}\\&\quad+4\times\frac{(x-1)(x-2)(x-3)(x-5)}{(4-1)(4-2)(4-3)(4-5)}\\&=-20+\frac{137}{3}x-\frac{125}{4}x^2+\frac{25}{3}x^3-\frac{3}{4}x^4\end{aligned}$$

Ex.2.11

证明：

$$
\begin{aligned}
XY &= \begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} E & F \\ G & H \end{bmatrix} \\
&= \left(\begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}+\begin{bmatrix} 0 & B \\ 0 & 0 \end{bmatrix}+\begin{bmatrix} 0 & 0 \\ C & 0 \end{bmatrix}+\begin{bmatrix} 0 & 0 \\ 0 & D \end{bmatrix}\right)\begin{bmatrix} E & F \\ G & H \end{bmatrix} \\
&= \begin{bmatrix} AE & AF \\ 0 & 0 \end{bmatrix}+\begin{bmatrix} BG & BH \\ 0 & 0 \end{bmatrix}+\begin{bmatrix} 0 & 0 \\ CE & CF \end{bmatrix}+\begin{bmatrix} 0 & 0 \\ DG & DH \end{bmatrix} \\
&= \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}
\end{aligned}
$$

Ex.2.12

$T(n)=2T(n/2)+1 \Rightarrow T(n)=\Theta(n)$。

Ex.2.13

a) 图略，$B_3=1$，$B_5=2$，$B_7=5$。如果n是偶数，这颗树就不可能是 *full* 的了。对一颗 *full* 的二叉树来说，如果我们移除某个节点的两个叶子节点，剩下的也是一棵 *full* 的二叉树，继续这个移除过程到最后一定会只剩下根，所以可知该树的节点数必为奇数。

b) 动态规划。考虑一颗n节点而且 *full* 的二叉树T_n，其左子树的节点数分别可能为$1,3,5,...,n-2$，而相应的右子树的节点数分别为$n-2,n-4,n-6,...1$，由简单的排列组合关系可以得出下式：

$$
\begin{aligned}
B_n &= B_1B_{n-2}+B_3B_{n-4}+B_5B_{n-6}+...+B_{n-2}B_1 \\
&= \sum_{i=1,3,5...}^{n-2} B_iB_{(n-1-i)}
\end{aligned}
$$

如果存在系列C_n满足$C_0=1$，且有递归式$C_{n+1}=\sum_{k=0}^{n} C_k \bullet C_{n-k}$，与上面$B_n$的关系式相比较，可以看出，$B_{2n+1}=C_n$，这个$C_n$即是著名的 Catalan 数，另有

通项公式 $C_n = \frac{1}{n+1}\binom{2n}{n}$，及递推式 $C_n = \frac{4n-2}{n+1}C_{n-1}$。关于 Catalan 数的性质及推导可以在任何一本组合数学教材上找到，这里不再详细探讨。

c) 由 Catalan 数的一阶递推式 $C_n = \frac{4n-2}{n+1}C_{n-1}$，容易看出 $B_n = O\left(2^n\right)$。

Ex.2.14

可以改写归并排序的 *merge* 过程，在排序时同时也将重复元素抛弃掉，时间复杂度为 $O\left(n\log n\right)$，也可以直接先排序，再用 $O\left(n\right)$ 的时间剔除重复元素，时间复杂度还是为 $O\left(n\log n\right)+O\left(n\right)=O\left(n\log n\right)$。

Ex.2.15

直接帖出代码，下面这个函数返回两个值 l 和 r，其中 l 是分区后的子序列 S_L 的最后一个元素的索引值加 1，而 r 为 S_R 的第一个元素的索引值。

```
template <typename Type>
void Partion(Type *array, int begin, int end, Type v, int &l, int &r)
{
    l = begin;
    for (int i=begin; i!=end; ++i)
    {
        if (array[i] < v)
            swap(array[i], array[l++]);
    }
    r = l;
    for (int j=l; j!=end; ++j)
    {
        if (array[j] == v)
            swap(array[j], array[r++]);
    }
}
```

Ex.2.16

假设 $A[\bullet]$ 的序号从 1 开始。首先将 x 与 $A[1]$ 比较，如果相等，返回真。如果 $x < A[1]$ 返回假。如果 $x > A[1]$，再将 x 与 $A[2]$ 比较，如果相等，返回真，如果 $x < A[2]$ 返回假。如果 $x > A[2]$，接下来再将 x 与 $A[3]$ 比较？当然不是，因为这样就成 $O(n)$ 的了。我们应该采取某种指数增长的形式，比如接下来与 $A[4]$ 比较，再接下来同 $A[8], A[16], A[32]...$，于是在 $O(\log n)$ 的时间内，就可以找到一个范围 $A[2^i] < x < A[2^{i+1}]$，再在这个范围内进行二分查找，总时间复杂度为 $O(\log n)$。

Ex.2.17

对于数组 $A[1,...,n/2,...n]$，如果 $A[n/2] > n/2$，由 A 中元素有序且唯一可知，$A[i] > i\,(n/2 \le i \le n)$，所以接下来只需要继续在 $A[1,...,n/2]$ 中寻找，如果 $A[n/2] = n/2$，返回 $n/2$，否则在 $A[n/2,...n]$ 中继续寻找。$T(n) = T(n/2) + 1 \Rightarrow T(n) = O(\log n)$。

Ex.2.18

一次询问只能产生两种结果，根据询问结果的是与否可以将数组 A 分成两个部分。在最好的情况下，这两个部分是均等的，这样无论询问结果如何都可以将问题规模缩小一半。于是可知，最后要定位到一个唯一的元素，需要询问 $\Omega(\log n)$ 次。一些更详细的探讨可以 Google：信息论、信息熵、排序、称球问题。

Ex.2.19

a) 时间复杂度：

$$\begin{aligned} T(k,n) &= O(n+n) + O(2n+n) + O(3n+n) + \cdots + O((k-1)n+n) \\ &= O(2n+3n+\cdots kn) = O(k^2 n) \end{aligned}$$

b) 分治，先分别 *merge* 前 $\left\lfloor \frac{k}{2} \right\rfloor$ 个序列和后 $\left\lceil \frac{k}{2} \right\rceil$ 个序列，再将中间结果组合起来：

$$\begin{cases} T(k,n) = T\left(\lfloor k/2 \rfloor, n\right) + T\left(\lceil k/2 \rceil, n\right) + O(kn) \\ T(1,n) = O(1) \end{cases}$$
$$\Rightarrow T(k,n) = O(k \log kn)$$

Ex.2.20

采用计数排序，先花 $O(n)$ 的时间找到 $\max(x_i)$ 和 $\min(x_i)$，然后建立一个大小为 $S = \max(x_i) - \min(x_i) + 1 = M + 1$ 的数组 $C\left[\min(x_i), ..., \max(x_i)\right]$ 用于计数。遍历数组 $x[1,...,n]$，对于每个出现的 $x[i]$，对应的 $C[x[i]]$ 自增 1。最后再遍历数组 C 进行输出，如果 $C[i]$ 大于 0，则将元素 i 输出 $C[i]$ 次到结果数组里。总的时间复杂度为 $O(n+S) = O(n+M)$。由于计数排序不是基于比较的，所以不受下界 $O(n \log n)$ 的限制。

Ex.2.21

a) 又见可爱的 Hint，$\sum_i^n |x_i - \mu|$ 即是点 u 到点 x_i 的距离之和。如果 u 不是中位数，比如在中位数的右边，那么将它将左移动一个元素，设这段移动的距离为 d，这样点 u 到它左边的所有点的距离都将减少 d，而到它本身以及右边的所有点的距离都将增加 d，由于其左边的元素多于右边的元素，所以总的来说 $\sum_i^n |x_i - \mu|$ 是减小了，于是得证。

b) 证明：

$$\begin{aligned}\sum_i^n (x_i-\mu)^2 &= \sum_i^n (x_i-\mu_2+\mu-\mu_2)^2 \\ &= \sum_i^n \left((x_i-\mu_2)^2+2(x_i-\mu_2)(\mu-\mu_2)+(\mu-\mu_2)\right)^2 \\ &= \sum_i^n (x_i-\mu_2)^2+n(\mu-\mu_2)^2+2(\mu-\mu_2)\left(\sum_i^n (x_i)-n\mu_2\right) \\ &= \sum_i^n (x_i-\mu_2)^2+n(\mu-\mu_2)^2\end{aligned}$$

c) $\mu_\infty = \min(x_i)+(\max(x_i)-\min(x_i))/2$。

Ex.2.22

设两个序列分别为 $A[1,...,m]$， $B[1,...,n]$，首先比较 $A[m/2]$ 和 $B[n/2]$，再根据比较结果进行判断，不失一般性，我们假设 $A[m/2] \geq B[n/2]$，再由 A,B 均有序可知， $A[m/2+1,...,m]$ 的所有元素肯定排在 $A \cup B$ 的 $m/2+n/2$ 名之后，而 $B[1,...,n/2]$ 肯定排在 $A \cup B$ 的 $m/2+n/2$ 名以内。所以，如果 $k \leq m/2+n/2$，就可以将 $A[m/2+1,...,m]$ 剔除掉，继续在剩余元素中寻找第 k 小元素。如果 $k > m/2+n/2$，那么就可以将 $B[1,...,n/2]$ 剔除掉，继续在剩余元素中寻找第 $k-n/2$ 小元素。于是可得以下递归式：

$$\begin{cases} T(m,n) = T(m/2,n)+1 & \textit{if cond}... \\ T(m,n) = T(m,n/2)+1 & \textit{else} \\ T(0,n) = T(m,0) = O(1) \end{cases}$$

所以 $T(m,n) = O(\log m + \log n)$。

Ex.2.23

a) 若 $majority(A_1)$ 和 $majority(A_2)$ 均不存在，那么 $majority(A)$ 也不存在。如

果至少存在其中之一，遍历 A 数组，找到这个（或两个） $majority$ 元素出现的次数，如果大于总元素的一半，说明已经找到。否则 $majority(A)$ 不存在。当然如果运气好遇到 $majority(A_1)=majority(A_2)$ 那就不用遍历了，直接返回即可。于是 $T(n)=2T(n/2)+O(n)\Rightarrow T(n)=O(n\log n)$。

b) 老题了，由于这 $n/2$ 对中，每对都至少要抛弃掉一个元素，所以剩下的元素不会多于 $n/2$。如果 $majority(A)$ 存在，设它为 x，且出现次数为 m，在将 A 缩减到 A' 的过程中，如果有 r 个 x 与非 x 元素分到一组，$m-r$ 个 x 与自己分到一组，于是缩减后剩余的总元素个数至多为 $n/2-r$，剩余的 x 个数为 $(m-r)/2$，因为 $m>n/2$，所以 $(m-r)/2>(n/2-r)/2$，这就证明了在 A' 中，x 仍是占优的。另外，这个条件只是充分而不是必要的，题目中的陈述应该有误，所以找到 $majority(A')$ 后还要在检查它在 A 中出现的次数。综上可得递归式：$T(n)=T(m)+O(n),\ m\le n/2\Rightarrow T(n)=O(n)$。这题的一些变形可以参见算法导论思考题 4-6，和《编程之美》第二章第三节。

Ex.2.24

a) 利用 Ex.2.15 的 Partition 操作，可以写出以下代码：

```
template <typename Type>
void QuickSort(Type *array, int begin, int end)
{
    if (begin < end)
    {
        int l, r, v = array[begin];
        Partion(array, begin, end, v, l, r);
        QuickSort(array, begin, l);
        QuickSort(array, r, end);
    }
}
```

b) 最坏情况下，$T(n)=T(n-1)+O(n)\Rightarrow T(n)=O(n^2)$。

c) 这个递归式应该是很清楚明了的，无须多言。现在来证明它的上界：

$$\begin{cases} T(n)=O(n)+\dfrac{2}{n}\sum_{i=1}^{n-1}T(i) \\ T(n-1)=O(n-1)+\dfrac{2}{n-1}\sum_{i=1}^{n-2}T(i) \end{cases}$$

$$\Rightarrow \frac{n-1}{n}T(n-1)=\frac{n-1}{n}O(n-1)+\frac{2}{n}\sum_{i=1}^{n-2}T(i)$$

$$\Rightarrow T(n)-\frac{n-1}{n}T(n-1)=O(n)-O(n-1)+\frac{1}{n}O(n-1)+\frac{2}{n}T(n-1)$$

$$\Rightarrow T(n)=\frac{n+1}{n}T(n-1)+O(1)\Rightarrow T(n)=O(1)\left(1+\frac{n+1}{n}+\frac{n+1}{n-1}+\frac{n+1}{n-2}+\cdots\right)$$

$$\Rightarrow T(n)=O(n\log n) \qquad (\text{why? 注意Ex. 1. 5的结论})$$

Ex.2.25

a) $z=pwr2bin(n/2)$，$T(n)=T(n/2)+(cn/2)^{\log_2 3}\Rightarrow T(n)=O\left(n^{\log_2 3}\right)$。

b) $return\ \ fastmutiply\left(dec2bin(X_L),pwr2bin(n/2)\right)+dec2bin(X_R)$。时间复杂度$T(n)=2T(n/2)+O\left(n^{\log_2 3}\right)\Rightarrow T(n)=O\left(n^{\log_2 3}\right)$。

Ex.2.26

乘方和乘法的渐近时间复杂度应该是一样的，从分治法的递归表达式可以看出，即使两乘数相等也并不能减少子问题的个数和规模。

Ex.2.27

a) $\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} a & b \\ c & d \end{bmatrix}=\begin{bmatrix} a^2+bc & b(a+d) \\ c(a+d) & bc+d^2 \end{bmatrix}$，5 次乘法。

b) 注意分治之后，得到的子问题不再是都是平方操作了。

c) 第一问：$AB+BA=(A+B)(A+B)-AA-BB$。

第二问：$AB+BA=\begin{bmatrix} 0 & XY \\ 0 & 0 \end{bmatrix}$。

第三问：$\begin{bmatrix} 0 & XY \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} X & Y \\ 0 & 0 \end{bmatrix}\begin{bmatrix} X & Y \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} X & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} X & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & Y \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & Y \\ 0 & 0 \end{bmatrix}$，

实际上因为 $\begin{bmatrix} 0 & Y \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & Y \\ 0 & 0 \end{bmatrix} = 0$，仅需 $2S(2n) + O(n^2)$ 即可。

Ex.2.28

$$H_k v = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix}\begin{bmatrix} v_u \\ v_d \end{bmatrix} = \begin{bmatrix} H_{k-1}(v_u + v_d) \\ H_{k-1}(v_u - v_d) \end{bmatrix}$$
$$\Rightarrow T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$

Ex.2.29

a) 循环不变式：设当循环到 i 时有 $z_i = a_i + a_{i+1}x + a_{i+2}x^2 + \cdots a_n x^{n-i}$，那么 $z_{i-1} = z_i x + a_{i-1} = a_{i-1} + a_i x + a_{i+1}x^2 + \cdots + a_n x^{n-(i-1)}$，于是得证。

b) 每次循环一次乘法一次加法，总共 n 次乘法 n 次加法。事实上 *Horner* 法则是最优的。

Ex.2.30

a) $\omega = 3 \text{ or } 5$，显然有 $\sum_{i=1}^{6} \omega^i \equiv \sum_{i=1}^{6} i \equiv 0 \bmod 7$。

b) 若取 $\omega = 5$，有：

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 5 & 4 & 6 & 2 & 3 \\ 1 & 4 & 2 & 1 & 4 & 2 \\ 1 & 6 & 1 & 6 & 1 & 6 \\ 1 & 2 & 4 & 1 & 2 & 4 \\ 1 & 3 & 2 & 6 & 4 & 5 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 3 \\ 2 \\ 4 \\ 6 \end{bmatrix}$$

c) 在 $M_n(\omega)$ 中任意取两列：

$$Col_{i+1}=\left[1,\omega^{i},\omega^{2i},\omega^{3i},...,\omega^{(n-1)i}\right]^{T}$$
$$Col_{j+1}=\left[1,\omega^{j},\omega^{2j},\omega^{3j},...,\omega^{(n-1)j}\right]^{T}$$

于是 $Col_{i+1}\bullet Col_{j+1}\equiv\sum_{k=1}^{n}\omega^{ki}\left(\omega^{-1}\right)^{kj}\bmod n+1$。这里的 ω^{-1} 即是 ω 模 $n+1$ 的逆元，即 $\omega\bullet\omega^{-1}\equiv 1\bmod n+1$，不失一般地，可假设 $i>j$，于是上式可变成 $\sum_{k=1}^{n}\omega^{k(i-j)}\equiv\frac{1-\omega^{n(i-j)}}{1-\omega^{(i-j)}}\equiv 0\bmod n+1$，可知 $M_n(\omega)$ 任意两列两行均正交。于是 $M_n(\omega)M_n\left(\omega^{-1}\right)\equiv nI\bmod n+1\Rightarrow M_n(\omega)^{-1}\equiv M_n\left(\omega^{-1}\right)n^{-1}\bmod n+1$。

另外由于 $n^{-1}\equiv n\equiv -1\bmod n+1$，最后得：

$$M_n(\omega)^{-1}\equiv -M_n\left(\omega^{-1}\right)\equiv nM_n\left(\omega^{-1}\right)\bmod n+1。$$

d) 令 $A=(1,1,1,0,0,0)$，$B=(-1,2,0,1,0,0)$，则有：

$$A\xrightarrow{FT}A'=(3,3,0,1,0,6)$$
$$B\xrightarrow{FT}B'=(2,1,1,3,4,4)$$
$$\Rightarrow AB\equiv(6,3,0,3,0,3)\bmod 7$$
$$\Rightarrow AB\xleftarrow{FT}C=(6,1,1,3,1,1)$$

Ex.2.31

a) 若 a,b 均为奇数，则 $a-b$ 为偶数，因 $\gcd(a,b)=\gcd(a-b,b)$，此时可套用第二种情况，即 $\gcd(a-b,b)=\gcd((a-b)/2,b)$。其余两种情况略。

b) 如 a)中所示。

c) 因为乘于或除于 2 和减法操作均是 $O(n)$ 的，于是有：

$$T(n)=T(n-1)+O(n)\Rightarrow T(n)=O(n^2)$$

比欧几里得辗转相除法$O\left(n^3\right)$更优。

Ex.2.32

a) 如果存在四个点，那么这四个点只能位于区域$\Omega_{d\times d}$的四个角上。所以如果再加一个点，必有某两个点的距离小于d，因此最多只能有四个。

b) 在分界线左右各取一个$d\times d$的区域，取成一个长方形区域$\vartheta_{d\times 2d}$，由第一问知在区域$\vartheta_{d\times 2d}$内最多包含L中的四个点和R中的四个点，共 8 个点。设中间的排序后的点集为C，选定其中的某个点P_0，显然在位于P_0之后且与点P_0的距离有可能小于d的点必然处于以P_0的y值为上界的一个区域$\vartheta_{d\times 2d}$内，于是很容易知道检查位于P_0之后的七个点便以足够。

c) 伪代码略，由于排序点集C的过程为$O\left(n\log n\right)$，所以有：

$$\begin{aligned}T\left(n\right)&=2T\left(n/2\right)+n\log n\\&\le O\left(n\right)+\sum_{i=1}^{\log n}n\log n^2=O\left(n\log n\log n\right)\end{aligned}$$

d) 预先将所有点排序，这样在每步迭代中就不用再排序了：

$$T\left(n\right)=T\left(n/2\right)+O\left(n\right)\Rightarrow T\left(n\right)=O\left(n\log n\right)$$

加上预排序所用的$O\left(n\log n\right)$，总的时间复杂也为$O\left(n\log n\right)$。

3 Decompositions of graphs

Ex.3.1

图不好画，所以略。

Ex.3.2

再略。

Ex.3.3

a) 按访问的顺序依次为 $A(1,7)$， $C(2,6)$， $D(3,4)$， $F(4,3)$， $G(5,1)$，$H(6,2)$， $E(7,5)$， $B(8,8)$

b) 源（sources）：A,B，汇（sinks）：G,H

c) B,A,C,E,D,F,H,G

d) 总共有 $2\times2\times2=8$ 种。

Ex.3.4

图（i）略，只考虑图（ii）。

a) 依次为：$\{D,H,F,I,G\}$， C， $\{A,B,E\}$。

b) 源为 $\{A,B,E\}$，汇为 $\{D,H,F,I,G\}$。

c) 略。

d) 1 条，从 $\{D,H,F,I,G\}$ 中任意引一条到 $\{A,B,E\}$ 即可。

Ex.3.5

对每一个顶点 V_1，遍历它的边，如果存在 $V_1\to V_2$，则在其转置或者反图 G^R 中找到顶点 V_2 并添加一条边 $V_2\to V_1$。显然时间复杂度为 $O(V+E)$。

Ex.3.6

a) 对任意一条边(u,v)，都同时为顶点u和顶点v贡献了一个“度”，所于每一个边对应了两个“度”，于是知$\sum_{u\in V} d(u) = 2|E|$。

b) 因为各顶点的度的和为偶数，于是可知度为奇数的顶点数必为偶数。

c) 不能，虽然在有向图中，同样有$\sum_{u\in V} d_{in}(u) + d_{out}(u) = 2|E|$，但是各顶点的入度之和的奇偶性不能确定。

Ex.3.7

a) 在图上进行 DFS，并将在 DFS 树上位于不同层的顶点交替地着上红黑二色，使得每条 tree edge 的两个顶点颜色都不相同。如果遇到一条 non-tree edge (u,v)，且有顶点u,v的颜色相同，则返回假，否则返回真。时间复杂度$O(V+E)$。

b) 根据上面的算法可以知道，一个图为二分的一个充要条件是对于每条 non-tree edge (u,v)，有顶点u,v的颜色不相同，也即是在 DFS 树上顶点u,v要间隔奇数层。而环的长度为间隔层数加 1，为偶数。于是得证。

c) 三种颜色。

Ex.3.8

a) 设一个顶点为$(m,n,11-m-n)$，分别表示 4，7，10 品脱容量的容器中所含的水量。题目即是问顶点$(4,7,0)$是否可达顶点$(2,k,11-2-k)$或者是顶点$(l,2,11-l-2)$。

b) 因这里顶点最多不会超过$5\times 8 = 40$个，所以 DFS,BFS 均可。

c) 其一：$(4,7,0)$，$(0,7,4)$，$(0,1,10)$，$(4,1,6)$，$(0,5,6)$，$(4,5,2)$，$(2,7,2)$。

Ex.3.9

分两次遍历每个顶点的每条边，第一次将每条边对应的两个顶点的度数加 1。第二次将每个顶点的所有邻接点的度数加起来即可。显然时间复杂度是线性的。

Ex.3.10

伪 C++代码如下：

```
void explore(Vertex v)
{
    Stack vertices;
    previsit(v);
    v.setVisited(true);
    vertices.push(v);
    while(!vertices.empty())
    {
        Vertex cv = vertices.back();
        bool done = true;
        for (int i=0; i!=cv.numNeighbors(); ++i)
        {
            Vertex cu = cv.getNeighbor(i);
            if (cu.visited() == false)
            {
                done = false;
                previsit(cu);
                cu.setVisited(true);
                vertices.push(cu);
                break;
            }
        }
        if (done)
        {
            postvisit(cv)
            vertices.pop()
        }
    }
}
```

Ex.3.11

设 e 的两个顶点为 (u,v)。先将边 e 断开（不需要将 e 删除，做一个标记即可），然后对顶点 u 进行 DFS，如果能到达顶点 v，则返回真，否则返回假。

Ex.3.12

因 $post(u) < post(v)$，所以 v 比 u 后出栈，这有两种可能：一是顶点 v 在顶点 u 出栈后才入栈，但是因为存在边 (u,v)，所以这种情况可以排除。另一种可能是，在 u 入栈时，v 已在栈中，这即意味着 v 是 u 的祖先。

Ex.3.13

a) 显然，把 DFS 树的某个叶子节点去掉后，该图仍然是连通的。
b) 一个首尾相连的环，比如：$A \to B \to C \to A$。
c) 设该图的 $meta-graph$ 为 $U \to V$，则任意添加一条从 V 到 U 的边后，该图即成为强连通的。

Ex.3.14

首先用 $O(V+E)$ 的时间计算每个顶点的入度。遍历所有顶点，如果某顶点的入度为 0，则将该顶点加到一个队列 $d_{in}zero$ 中。然后进入循环，每次循环中从 $d_{in}zero$ 取出一个元素进行输出，同时将它的所有邻接点的入度减 1，并将入度减 1 后为 0 的邻接点也加入到 $d_{in}zero$ 中。直到队列 $d_{in}zero$ 为空时结束循环。容易看出，这个算法的时间复杂度为 $O(V+E)$。

Ex.3.15

a) 即验证该有向图是否是强连通的。
b) 即验证顶点 town hall 是否位于整个路线图的一个汇（sink）强连通分支中。
因为找寻强连通子图的时间复杂度是线性的，所以上面两个验证操作均可以在线性时间内完成。

Ex.3.16

拓扑排序，可以采用类似 Ex.3.14 的算法，不同的是这里需要采用多个队列：首先将入度为 0 的顶点加入到第一个队列，然后将这些顶点的所有邻接点的入度减 1，如果遇到入度减 1 后为 0 的邻接点，则加入到第二个队列中，然后再处理第二个队列中所有顶点的邻接点……。这样第 i 个队列即为第 i 个学期的课表，队列的总个

数即是至少所需的学期数。

Ex.3.17

a) 因为 $Inf(p)$ 中的所有节点都被访问了无限次，它们之间当然都是强连通的。

b) 即验证图 G 中是否存在有不少于两个节点的强连通分支。

c) 即验证图 G 中是否存在有不少于两个节点的强连通分支，且该分支中至少存在一个“好”节点。

d) 将图 G 去掉所有“坏节点”后得图 G'，再验证图 G' 中是否存在有不少于两个节点的强连通子图。

Ex.3.18

先一次 DFS，为所有节点做上标记。首先把根节点标记为 1，其它节点的标记按照以下规则：如果节点 x 的标记为 L，则将它的左子节点标记为 $L0$，右子节点标记为 $L1$。任意两个节点 u,v，如果其中一个节点（比如 u）的标记为另一个节点（比如 v）的前缀，则 u 是 v 祖先。比如标记为 110 的节点就是标记为 11001001 的节点的祖先。

Ex.3.19

首先将 $z[\bullet]$ 初始化为 0，然后进行 DFS。在遍历过程中，若有节点出栈（比如 v），将当前栈顶的节点 u（即为 v 的父亲）的 z 值更新：$z[u]=\max(z[u],x[v])$。

Ex.3.20

这里需要 DFS 的栈也可以进行随机访问，设这个 DFS 栈为 $stack$，在遍历过程中，若有节点进栈（比如 v），则将它的 l 值更新为：

$$l(v)=l\left(stack.at\left(\max\left(stack.size(\ )-l(v),1\right)\right)\right)$$

其中 $stack.at(i)$ 操作即是取出在栈中位于第 i 位的顶点，设栈底元素为第 1 位。

Ex.3.21

在每个强连通子图上采用类似 Ex.3.7 的染色算法，在 DFS 过程中，若存在一条 forward/back/cross edge $e(u,v)$，且有 u,v 的颜色相同，则一定存在奇数长度的环。

怎么证明这个算法的正确性呢？back edge 的情况很好理解，跟 Ex.3.7 情况类似。对于 forward /cross edge 的情况，需要注意一个事实：在强连通图上，如果某顶点 v_1 到顶点 v_2 有两条路径，且这两条路径的长度奇偶性不同，显然，无论 v_2 到 v_1 的路径长度如何，该图都一定存在奇数长度的环，当然同时也一定存在偶数长度的环。

Ex.3.22

即检查该图的强连通 $meta-graph$ 中，是否只存在一个唯一的源（source）。

Ex.3.23

动态规划。建立一个数组 $path[\bullet]$ 用于记录从 s 点到所有顶点的路径数，首先将 $path[s]$ 初始化为 1，其它的初始化为 0。然后从 s 点开始进行 DFS，每当遇到任意边 $e(v,u)$（无论是 tree edge、forward edge、还是 cross edge），则修改到 u 点的路径数为 $path(u)=path(u)+path(v)$。最后输出 $path(t)$ 即可。

Ex.3.24

定义过程 $Search(G)$：若 G 中只有一个顶点，返回真。否则首先在图 G 中找到入度为 0 的顶点，如果这样的顶点不只一个，则返回假，否则设这个唯一的顶点为 s，将 s 的所有邻接点的入度减 1，并在 G 中除去顶点 s 后得到图 G'，然后在 G' 上递归的进行 $Search$ 过程，即返回 $Search(G')$。

Ex.3.25

a) 按拓扑顺序的逆序依次计算每个顶点，即设定待计算顶点的 $cost$ 值为它自身的 p 值与它的所有邻接点的 $cost$ 值中的最小值。
b) 将每个强连通分支作为一个顶点看待，并将它的 p 值设定为在该分支子图中所有顶点的 p 值中的最小值。所有的强连通分支即构成了一个 $meta-graph$，再将这个 $meta-graph$ 按 a)中的过程处理。

Ex.3.26

a) 先看必要性，在一条欧拉回路中，由于该回路是闭合的，所以对每一个顶点，访入和访出的次数都必须相同，又因该回路不重复地访问了所有的边，于是可知每个顶点的度数等于访入次数加上访出次数，当然为偶数。充分性只在连通图中有效：考虑一颗连通图的 DFS 树，从根开始，对每个节点访入访出各一次后又回到根节点。在这个 DFS 过程中作一点修改，如果某个节点的度数大于 2，那么反复地访问这个节点，直到经过了该点所有的边。因为每个节点的度数均为偶数，所以这个访入访出的过程总是可以配对的。这即形成了一条欧拉回路。在七桥问题中，由于所有顶点的度数均为奇，所以当然不存在欧拉回路。

b) 有向连通图 G 有欧拉路径当且仅当在该图中，最多只有两个度数为奇的顶点，这两个顶点即分别是欧拉路径的起点和终点。

c) 对于有向强连通图 G，当且仅当图 G 中的每一个顶点的入度和出度都相等时，图 G 具有欧拉回路。

Ex.3.27

由 Ex.3.6 知，在无向图中，度数为奇的顶点的个数必为偶数，所以当然可以分成两两一组。因为在该图的所有连通分支中，奇度顶点的个数也都为偶数，所以在这 n 对奇数度顶点中，必然存在一对连通的顶点 (u,v)，将 (u,v) 的路径边从图中去掉，得到一个包含 $n-1$ 对奇数度顶点的图，再在这个图上重复上述过程，可找到每一对顶点之间的路径，显然，它们之间是不相交的。

Ex.3.28

a) $x_1 = true, x_2 = true, x_3 = false, x_4 = true$。

b) 比如：$(x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3})$。

c) 略。

d) 此即意味着无论 x 成立或者 $\overline{x}$ 成立，也必须蕴含 $\overline{x}$ 或 x 成立，当然不可能满足。

e) 显然，只要找到一种组合方式满足对于图中的每一条边 $e(u,v)$，如果 u 为真，那么 v 也为真，那么也就等价于满足了 2-SAT 条件。假设某图 G 不存在同时包含某个变量和它的“反”（即逻辑非）的连通分支，对于它的某个汇连通分支来说，如果将这个强连通分支的所有顶点都设为真，这并不破坏逻辑图的相容性，因为汇强连通分支为真并不影响其它顶点的值，同时由于这些顶点的“反”

为假，所以也不影响其子孙的值（一个直觉的担扰是，若这些顶点的“反”存在祖先呢？那么这些祖先的值肯定不能为真，这样就影响到其它顶点了。当然，这个担扰是多余的，因为在汇强连通分支的所有顶点的“反”中，肯定不存在除它们自身以外的祖先，这一点根据 $p \to q$ 与 $\overline{q} \to \overline{p}$ 的等价性很容易证明）。

于是，可以把这个强连通分支和其中所有顶点的“反”从图中去掉，而且剩余的图仍然是可以满足的，同时也减小了问题的规模。

f) 按照上一问的过程处理即可，注意存储的时候应该保证 x 与 $\overline{x}$ 的索引号有某种对应关系。另外，“删除”某些顶点的操作可以改为做上某种标记，在后续处理时忽略掉已被标记过的顶点即可，这样就可以直接按照拓扑顺序的逆序依次处理，避免了重建强连通分支。因为求强连通分支和拓扑排序都是线性的，所以整个算法的时间复杂度也是线性的。

Ex.3.29

二元等价关系即相当于在无向图中某两个顶点之间的连通关系，这个关系的等价类即是图中的各个连通分支，显然无向图的各个连通分支都是不相交的，且有在各连通分支内，所有顶点都是连通的，而各连通分支彼此之间都是不连通的。

Ex.3.30

显然强连通关系满足自反性、对称性和传递性，所以是一个等价关系，而强连通分支正是这个关系对应的等价类。

Ex.3.31

a) 略。

b) bridges：(B,D)，(D,C)，(D,M)，separating vertices：B，D，L。

c) 若在两个双连通分支 C_1、C_2 中，存在着两个以上的共同顶点，取其中某两个顶点 u、v，显然在 C_1 和 C_2 中，u、v 都是连通的，设在 C_1 中 u、v 之间的路径为 P_1，在 C_2 中 u、v 之间的路径为 P_2，于是 P_1，P_2 就形成了一个环，这推出 C_1 与 C_2 也是双连通的，矛盾。所以在两个双连通分支不可能存在一个以上的相同顶点。如果仅存在一个，设为 v，这时 C_1、C_2 之间肯定不存在其它

的边，否定又将形成环。所以将v和它的边去掉之后，C_1、C_2将不再是连通的，于是知v是图中的一个割点。

d) 因为任意一个双连通分支与它的割点所能到达的其余任何顶点都不存在边，所以将所有双连通分量塌陷后，不存在回路，也即是一棵树（如果原图是连通的）。

e) 对一颗 DFS 树的根节点r来说，其所有子树之间的所有路径都将经过r，如果r存在某两个子树都不为空，则将根r去掉后，这两个子树就将断开，所以这时r是一个割点。另外，如果r只有不多于 1 个子树不为空，显然将r去掉不影响剩余部分的连通性。

f) 如果v的所有子树中的都存在着顶点有一条边指向v的某个祖先，显然将v去掉后它的子树中的所有顶点都是可达的。如果v存在有某个子树，在这个子树中的所有顶点都不存在指向v的祖先的边，那么当v被去掉后，这个子树将被断开，于是这时v是一个割点。

g) 在每个顶点v的 *previsit* 过程中，将$low(v)$初始化为$pre(v)$，若v存在回边，令t为这些回边$e(v,w)$中，$pre(w)$的最小值。在 *postvisit* 过程中，将v的父亲f的low值设置为：$low(f)=\min(low(f),low(v),t)$。

h) 对于 DFS 树中的某个顶点v，如果它有不止一个儿子或者有一个父亲，且存在某个儿子u使得$low(u)\geq pre(v)$，则v是一个割点。寻找双连通分支的 DFS 过程如下：在图中任选一条边入栈，然后进入循环，若栈非空，设栈顶元素为$e(a,b)$，若a不是割点，则将a的所有未被访问过的邻边入栈，同样若b不是割点，也将b的所有未被访问过的邻边入栈，若a，b为割点或者邻边都已被访问过，则$e(a,b)$出栈。当栈为空时，结束循环，此前所访问的所有边即构成一个双连通分支。将这个双连通分支从图中去掉，在剩余的边上继续这个 DFS 过程，可找到所有的双连通分支。综合上一问，可知整个算法的时间复杂度与 DFS 相同，即为$O(V+E)$。

4 Paths in graphs

Ex.4.1

请参考 Ex.3.1 的答案。

Ex.4.2

参考 Ex.3.2。

Ex.4.3

假设使用邻接矩阵表示法，有以下伪代码：

```
bool hasSimpleCyc4(Matrix edges[numVertices][numVertices])
{
  for (int i=0; i!=numVertices; ++i)
  {
    int flag[numVertices] = {0};
    for (int j=0; j!=numVertices; ++j)
    {
       if (j==i || edges[i][j] == 0) continue;
       for (int k=0; k!=numVertices; ++k)
       {
          if (k == j || k == i || edges[j][k] == 0) continue;
          if (flag[k] == 1)
            return true;
          else flag[k] = 1;
       }
    }
  }
  return false;
}
```

三重循环，于是时间复杂度为 $O\left(\left|V\right|^3\right)$。

Ex.4.4

注意回路中可以包含多条回边，例如若在 DFS 路径 $A \to B \to C \to D \to E$ 中存在两条回边 $D \to A$， $E \to A$，那么 $\left(A, D, E\right)$ 即构造了一个长度为 3 的回路。

Ex.4.5

把 BFS 过程改为使用两个队列，分别为队列 1 和队列 2。首先将 v 放入队列 1，然后进入循环：依次取出队列 1 中的顶点，并将这些顶点的邻接点放入队列 2，然后将队列 1 和队列 2 交换。直到遇到队列 1 中的某个顶点的邻接点为 u，循环结束。这时再考察在队列 1 剩余的顶点中，还有多少个与 u 相邻即可。

Ex.4.6

这些边即是从源点到其余所有点的最短路径所经过的边的集合，注意到在 Dijkstra 算法中，只为每个顶点保留了一条最短路径，所以在这个边集中不会有环的存在。

Ex.4.7

对树 $T=(V,E')$ 进行一次遍历，可以得到所有顶点的 *dist* 值，然后对图 G 的所有边进行一次 *update* 过程，如果所有顶点的 *dist* 均无变化，则说明 T 正是图 G 的最短路径树，否则说明某些顶点还存在更短的路径，或者在图 G 中存在 negative cycle。下面简单说明一下为什么如果 T 不是图 G 的最短路径树，则对图 G 的所有边进行一次 *update* 过程后，一定会使得某个顶点的 *dist* 值变小：可以将树 T 的顶点集 V 分为两个部分 (R,K)，构成从源 s 到 R 中的所有顶点的最短路径的所有边都包含于 E' 中。显然，有源点 $s\in R$。而 K 中所有顶点的最短路径均存在不属于 E' 的边。如果 K 是非空的，那么在 K 中，一定存在某个顶点 u，其最短路径的最后一条边为 (v,u)，$v\in R$。这样在对图 G 的所有边进行一次 *update* 过程后，就一定能将顶点 u 的 *dist* 值修正为真正的最短路径长度值。

Ex.4.8

寒，我当初就这么想过，这里需要注意到这个加常数的过程是不均等的。比如某两个点 (u,v)，它们之间的最短路径为 5，通过了 10 条边，则加了常数 c 之后它们之间的路径长度成了 $5+10c$。另外它们之间还有条路径，长度为 6，但是只通过了 3 条边，加了常数 c 之后这条路径的长度就成了 $6+3c$，反而有可能会比前面那条还短了。看来这个 F. Lake 教授还真是专门用来被 BS 的，另见 Ex.2.26。

Ex.4.9

这时 Dijkstra 算法是可以得出正确的最短路径的。

Ex.4.10

类似于 Bellman-Ford 算法，以 u 为源点，每次 $update$ 所有的边，循环 k 次即可。

Ex.4.11

依次以每个顶点 $s \in V$ 为源点，利用 Dijkstra 算法寻找到其余顶点的最短路径，这样在 $O\left(|V| \bullet |V|^2\right) = O\left(|V|^3\right)$ 时间内就找到了任意两个顶点之间的最短路径。这时，对于任意顶点对 (u,v)，有 (u,v) 之间的环的最短长度为 $dist(u,v) + dist(v,u)$。

于是枚举所有的边 $e(u,v)$，依次计算 $dist(u,v) + dist(v,u)$，取其中的最小值就得到了图 G 中最短环的长度，如果这个最小值是 ∞，则说明图 G 中不存在环。

Ex.4.12

设这条边为 $e(u,v)$，先在图 G 中除去边 e，再用 Dijkstra 算法寻找 (u,v) 之间的最短路径长度，最后再加上 $e(u,v)$ 的长度，就得到了包含边 e 的最短环长度。

Ex.4.13

a) 以 s 为起点 DFS 或 BFS 即可，遇到长度大于 L 的边则当作断路处理。
b) 即是求从 s 到 t 的所有路径的最长边的最小值，同样可以利用 Dijkstra 算法来完成。只是需要修改 $update$ 过程如下：

$$
\begin{aligned}
&\text{for all edges } (u,v) \in E: \\
&\quad \text{if } fuel(v) > \max\left(fuel(u), l(u,v)\right) \\
&\qquad fuel(v) = \max\left(fuel(u), l(u,v)\right) \\
&\qquad \cdots
\end{aligned}
$$

这个算法的正确性依赖于下面这个事实，如果路径 $s \to \cdots \to r \to t$ 所需的最小储油量为 l，则路径 $s \to \cdots \to r$ 所需的最小储油量定然小于等于 l，这使得

得 Dijkstra 算法总是能按照正确的顺序依次找到从源点 s 到其余所有顶点的路径的最长边的最小长度，也即是所需的最小储油量。

Ex.4.14

依次以每个顶点 $s \in V$ 为源点，使用 Dijkstra 算法寻找最短路径，可找到所有顶点到 v_0 的最短路径，以及 v_0 到其余所有顶点的最短路径。则对于顶点对 (u,v)，它们之间经过点 v_0 的最短路径长度为 $dist(u,v_0)+dist(v_0,v)$。

Ex.4.15

修改 Dijkstra 算法的 *update* 过程如下：

$$
\begin{aligned}
&\text{for all edges } (u,v) \in E: \\
&\quad \text{if } dist(v) > dist(u) + l(u,v) \\
&\qquad dist(v) = dist(u) + l(u,v) \\
&\qquad usp(v) = usp(u) \\
&\quad \text{else if } dist(v) = dist(u) + l(u,v) \\
&\qquad usp(v) = false
\end{aligned}
$$

Ex.4.16

a) 设节点 j 位于树的第 k 层第 m 位，则有 $j = 2^{k-1} - 1 + m$。其父节点位于第 $k-1$ 层的第 $\left\lceil \frac{m}{2} \right\rceil = \left\lfloor \frac{m+1}{2} \right\rfloor$ 位，其索引 p 即为 $2^{k-2} - 1 + \left\lfloor \frac{m+1}{2} \right\rfloor = 2^{k-2} + \left\lfloor \frac{m-1}{2} \right\rfloor$，

于是有：$p = \left\lfloor \frac{j}{2} \right\rfloor \Leftrightarrow j = 2p, 2p+1$。

b) 同样设 j 位于树的第 k 层第 m 位，得 $j = \frac{d^{k-1}-1}{d-1} + m$，其父节点索引 $p = \frac{d^{k-2}-1}{d-1} + \left\lfloor \frac{d-1+m}{d} \right\rfloor$，又 $j + d + 2 = \frac{d^{k-1}-d}{d-1} + d - 1 - m$，于是有：

$p=\left\lfloor \frac{j+d-2}{d} \right\rfloor \Leftrightarrow j=dp+2-d,dp+1-d,dp-d,\cdots,dp,dp+1$。

c) 对于索引为 i 的顶点，它在树中的高度为 $\log i$，于是它距树底的距离为 $h-\log i=\log n-\log i=\log(n/i)$。所以建堆的时间复杂度 $T(n)$ 为：$\sum_{i=1}^{n}\log(n/i)=\log\left(\prod_{i=1}^{n}\frac{n}{i}\right)$，令 $F(n)=\prod_{i=1}^{n}\frac{n}{i}$，有 $F(n+1)=F(n)\left(\frac{n+1}{n}\right)^n$。设 $F(n)\le c^n$，其中 c 为某个常数，代入递归式，有 $c^n\left(\frac{n+1}{n}\right)^n\le c^{n+1}$，因为 $\lim_{n\to\infty}\left(\frac{n+1}{n}\right)^n=e$，于是当 $c\ge e$ 时上式成立，于是可知 $F(n)=O(e^n)$，所以可得 $T(n)=\log(F(n))=O(\log(e^n))=O(n)$。这个建堆过程的最坏输入情况是初始数组按照某种逆序排列。

d) 利用 b)中的父子节点的索引关系，稍稍改动一下 bubbleup 和 minchild 过程即可。

Ex.4.17

a) 维护两个大小为 W 的数组 A、B，数组的每个元素都指向一个顶点链表。首先将源点 s 放到 $A[0]$ 位置上，然后进入循环：每次从 A 数组中下标最小的非空位置上取出一个顶点 u，*update* 它所有的邻边 $e(u,v)$，如果 $dist(v)<W$，则将 v 移动到 $A[dist(v)]$ 位置上，否则移动到 $B[dist(v)-u]$ 位置上。直到 A 中所有顶点都已被取出为止。接下来在数组 B 上进行同样过程，不过稍做变化，如果 *update* 到某一个顶点 u 的邻边 $e(u,v)$，如果 $dist(v)<2W$，则将 v 移动到 $B[dist(v)-W]$ 位置上，否则移动上 $A[dist(v)-2W]$ 位置上，直到 B 为空时又回到 A。重复上述过程，直到所有顶点都已被处理。由于每次 *deletemin* 是 $O(W)$ 的，所以总的时间复杂度为 $O(W|V|+|E|)$。

b) 大体算法同上，不过改成维护两个大小不超过W的二叉堆，其中堆中的每个元素都指向一个顶点链表。由于*insert* / *decreasekey*为$O(W)$，所以总的时间复杂度为$O\left(\left(|V|+|E|\right)\log W\right)$。

Ex.4.18

首先令$best[s]=0$，然后改写 Dijkstra 算法的*update*过程如下：

$$
\begin{aligned}
&\text{for all edges } (u,v)\in E:\\
&\quad \text{if } dist(v)>dist(u)+l(u,v)\\
&\qquad dist(v)=dist(u)+l(u,v)\\
&\qquad best(v)=best(u)+1\\
&\quad \text{else if } dist(v)=dist(u)+l(u,v)\\
&\qquad \text{if } best(v)>best(u)+1\\
&\qquad\quad best(v)=best(u)+1
\end{aligned}
$$

Ex.4.19

即相当于在图G中，每条边$e(u,v)$的长度增加了c_v。令$cost[s]=c_s$，改写 Dijkstra 算法的*update*过程如下：

$$
\begin{aligned}
&\text{for all edges } (u,v)\in E:\\
&\quad \text{if } cost(v)>cost(u)+l(u,v)+c_v\\
&\qquad cost(v)=cost(u)+l(u,v)+c_v\\
&\qquad \cdots\cdots
\end{aligned}
$$

Ex.4.20

为方便表示，设 Dijkstra 算法的时间复杂度为$Dijkstra(G)$。首先与s为源点，计算到其余所有点的最短路。遍历边集E'，对于任意边$e'(u,v)\in E'$，如果$dist(v,t)$未计算过，则以v为源点寻找到顶点t的最短路。设*dec*为图G加上边e'后

$dist(s,t)$ 的缩减值，则如果 $dist(s,u)+l(u,v)+dist(v,t)\geq dist(s,t)$，$dec=0$，否则 $dec=dist(s,t)-\left(dist(s,u)+l(u,v)+dist(v,t)\right)$，找出令 dec 最大的边 e' 即可，时间复杂度为 $O\left(Dijkstra(G)\bullet\max\left(|V|,|E'|\right)\right)$。

Ex.4.21

a) 建立有向图 $G(V,E)$，其中边 $e(i,j)$ 的长度 $l(i,j)=r(i,j)$。用 $money(x)$ 表示一个单位的 s 币最多能换到多少个单位的 x 币。首先令 $money(s)=1$，$money\left(x|_{x\neq v}\right)=0$，使用 Bellman-Ford 算法，并修改 *update* 过程如下：

$$\text{procedure update } \left((u,v)\in E\right):$$
$$\text{if } money(v)<money(u)\bullet l(u,v)$$
$$money(v)=money(u)\bullet l(u,v)$$

如果图 G 中不存在下一问中所述的 anomaly，那么任意两顶点之间的“最大兑换路径”至多有 $|V|-1$ 条边，所以 Bellman-Ford 算法一定能成功找到这条路径。

b) 即类似于“负权回路”的情况，在 Bellman-Ford 算法完成之后，再对所有边进行一次 *update* 过程，如果有某个顶点的 *money* 值增加了，说明存在 anomaly。

Ex.4.22

a) 如果存在负环，也即是有 $\sum w_{ij}=r\sum c_{ij}-\sum p_j<0\Rightarrow r<\dfrac{\sum p_j}{\sum c_{ij}}\leq r^*$。

b) 即是在所有环中，有 $r>\dfrac{\sum p_j}{\sum c_{ij}}$，于是有 $r>r^*$。

c) 按照题述的提示，在 $[0,R]$ 区间上进行二分查找，时间复杂度为 $O\left(|V|^3\log(R/e)\right)$。

5 Greedy algorithms

Ex.5.1

a) 19。
b) 两种。
c) 略。

Ex.5.2

略。

Ex.5.3

遍历图G中的边，并计数$|E|$，当$|E|>|V|-1$时立即返回真，时间复杂度为$O(|V|)$。

Ex.5.4

设每个连通分支中的顶点数依次为$n_1,n_2,...,n_k$，$n=\sum_{i=1}^{k}n_k$。由于在每个连通分支内有$e_i \geq n_i-1$，所以有$e \geq \sum_{i=1}^{k}e_i=\sum_{i=1}^{k}(n_i-1)=n-k$。

Ex.5.5

a) 假设最小生成树会发生改变，设在边权重增加前后的最小生成树分别为mst_1和mst_2，由于最小生成树有$|V|-1$条边，于是$cost(mst_1)+|V|-1>cost(mst_2)$，也即是$cost(mst_2)-(|V|-1)<cost(mst_1)$，这说明$mst_2$在边权重增加前比$mst_1$更优，矛盾，于是知最小生成树不会改变。

b) 最短路径有可能会改变，和 Ex.4.8 大同小异。

Ex.5.6

可以用归纳法，若顶点个数为n、且边权均不同的图的最小生成树唯一，则根据 cut property，很容易推出顶点个数为$n+1$、边权均不同的图的最小生成树也唯一。

Ex.5.7

把所有的边的权乘上-1，再求最小生成树即可。或者也可以直接在求最小生成树的算法上做些小修改，因为 cut property 对最大生成树也成立。

Ex.5.8

不可能。首先以 s 为源点，用 Prim 算法寻找最小生成树，设 v 为 s 的最近邻接点，则 $e(s,v)$ 一定是某个 MST 中的边。因为图 G 中所有边的权彼此不同且都为正，这可能得出两个结论：一是图 G 的 MST 唯一，二是以 s 为源点的最短路可以用 Dijkstra 算法进行搜索，这样可以得出 $e(s,v)$ 也是最短路径树的边。所以最短路径树与最小生成树至少有一条公共边 $e(s,v)$。

Ex.5.9

a) False，很容易找到反例。
b) True，根据 cut property 容易证明。
c) True，同样根据 cut property 可以证明。
d) True，还是 cut property。
e) True。
f) False，也很容易找反例。
g) False。
h) False。
i) True。
j) True。

Ex.5.10

由图 G 得到 H 的过程可以看成是由一系列对边的删除操作组成（如果被删除边与某个度为 1 的顶点相连结，则同时也将该顶点删除）。于是现在只需要证明，若在图 G 删除一条边 e 后得到图 G'，有 $mst_G - e \in mst_{G'}$ 即可。下面分三种情况讨论：若 e 不属于 mst_G，则最小生成树不变，显然满足 $mst_G - e \in mst_{G'}$。若 e 属于 mst_G，且 e 与 G 中的某个度为 1 的顶点相连，则 $mst_G - e$ 即为图 G' 的最小生成树，同样

有 $mst_G - e \in mst_{G'}$。若 e 属于 mst_G，且 e 所连结的两个顶点的度均大于 1，那么在 e 所在的环中找出不属于 mst_G 的最小权值边 c，则 $mst_G - e + c$ 即为图 G' 的最小生成树，这样依然有 $mst_G - e \in mst_{G'}$，于是得证。

Ex.5.11

略。

Ex.5.12

假设 n 是 2 的幂，首先依次将第 i 个元素（$i \equiv 1 \bmod 2$）与第 $i+1$ 个元素合并，然后再依次将第 i 个元素（$i \equiv 1 \bmod 4$）与第 $i+2$ 个元素合并，再然后依次将第 i 个元素与第 $i+4$ 个（$i \equiv 1 \bmod 8$）元素合并，……，直到最后一次将第一个元素与第 $n/2+1$ 个元素合并，这样，经过 $n-1$ 次 *union* 操作，就形成了一棵高度为 $\log n$ 的树。将剩余 $m-(n-1)$ 次操作用于 *find* 最底层的元素，取 m 为 $2(n-1)$，于是总的时间复杂度为 $O\left(\frac{m}{2}+\frac{m}{2}\log n\right)=O(m\log n)$。

Ex.5.13

$A(10)$，$C(110)$，$G(111)$，$T(0)$。

Ex.5.14

a) $a(0)$，$b(10)$，$c(110)$，$d(1110)$，$a(1111)$。

b) 1,875,000 bits。

Ex.5.15

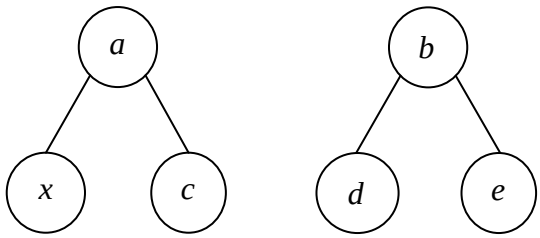
a) 比如 $(f_a, f_b, f_c)=(0.5, 0.2, 0.3)$。

b) 不可能，因为 (0) 是 (00) 的前缀。

c) 也不可能，编码冗余了，而 Huffman 是最优的。

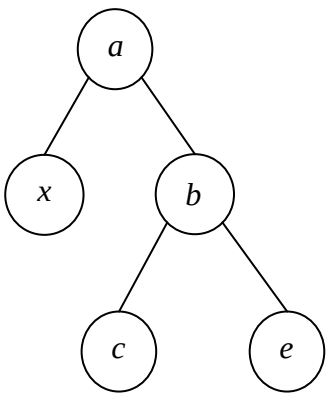
Ex.5.16

a) 假设字符 x 的频率大于 $\frac{2}{5}$，若 x 的编码长度大于 1，也即是在 Huffman 树中必会出现类似于以下图示的结构：



其中，节点 a，b 位于树中的同一层。若 b 子树比 a 子树先构造出来，即有 $f(b) \geq f(x), f(e) \geq f(x) \Rightarrow f(x)+f(d)+f(e) > \frac{6}{5}$。若 b 子树比 a 子树后构造出来，有 $f(c) \geq f(d)$，$f(c) \geq f(e)$，根据 Huffman 树的构造规则，可知 $f(b) = f(d)+f(e) \geq f(x) > \frac{2}{5}$，于是知 $f(c) > \frac{1}{5}$，所以有 $f(x)+f(c)+f(d)+f(e) > \frac{2}{5}+\frac{1}{5}+\frac{2}{5} = 1$，矛盾。这样就得出字符 x 的编码长度只能为 1。

b) 假设字符 x 的频率小于 $\frac{1}{3}$，若 x 的编码长度等于 1，也即是在 Huffman 树中必会出现类似于以下图示的结构：



显然，$f(b)=f(c)+f(e)>1-f(x)>\frac{2}{3}$，于是知 $f(c)$、$f(e)$ 中必有一个大于 $\frac{1}{3}$，这样 x 就不可能同时位于 $f(c)$ 和 $f(e)$ 的上层。所以任何频率小于 $\frac{1}{3}$ 的字符的编码长度必大于 1。若所有字符的频率都小于 $\frac{1}{3}$，也就不可能存在长度为 1 的编码。

Ex.5.17

最大编码长度为 $n-1$，比如取 $(f_1, f_2, f_3, ..., f_{n-1}, f_n)=\left(\frac{1}{2},\frac{1}{4},\frac{1}{8},...,\frac{1}{2^{n-1}},\frac{1}{2^{n-1}}\right)$。

Ex.5.18

a) 略，不过这里的数据好像有问题，因为所有字符的概率加起来大于 1。
b) 将 blank 出现的概率改为 17.3%后，算得 Huffman 编码的平均长度约为 4.171。
c) 上面的结果肯定比熵（约为 4.139）要大，因为在计算信息熵的时候允许有小数个比特，而实际上每个字符的编码长度都必需为整数。
d) 比如可以把特定的前缀、后缀或者是整个单词组合起来考虑。

Ex.5.19

a) 只需证明概率为 p_i 的元素的编码长度为 $\log\frac{1}{p_i}$ 即可。也即是证明概率为 $\frac{1}{2^k}$ 的元素 $c(k)$ 位于 Huffman 树的第 k 层（假设根在第 0 层）。首先，因为 $\frac{1}{2}>\frac{2}{5}$，且 $\frac{1}{2^n}<\frac{1}{3},(n>1)$，回忆一下 Ex.5.16 的结论可知，$c\left(\frac{1}{2}\right)$ 一定位于第一层，而且第一层所有元素的概率一定为 $\frac{1}{2}$。再考虑根为 $c\left(\frac{1}{2}\right)$ 的子树，同上可知，$c\left(\frac{1}{4}\right)$ 一定位于这棵子树的第一层（即原 Huffman 树的第二层），而且这一层所有元素的概率一定都为 $\frac{1}{4}$，……，这样一直推下去即可得出概率为 $\frac{1}{2^k}$ 的元

素 $c(k)$ 位于 Huffman 树的第 k 层，于是得证。

b) 当 $p_i = \frac{1}{n}$ 时信息熵最大，当 $p_k = 1, p_{i,(i \neq k)} = 0$ 时信息熵最小。

Ex.5.20

a) 对树 T 中的每个未被访问的叶子节点 v，设其对应的边为 $e(u,v)$，若节点 u 已被访问，说明不存在完美匹配，否则将节点 u 和 v 标记为已访问，并将边 e 从树中删除（如果将 e 删除后 u 成了叶子节点，则将 u 也删除）。反复进行上述过程，到最后如果所有的顶点都已被访问，则说明树 T 存在完美匹配。

b) 求图 G 的最大生成树，设这个最大生成树的边集为 E_m，则 E 除去 E_m 中的边后所得的差域 $E - E_m$ 即为所求的 E'。

Ex.5.21

a) 设这个环的顶点集为 C，最大权边为 $e(u,v)$。设在某一时刻，C 被 cut 成两个部分 s_1, s_2，其中 s_1 包含 u，而 s_2 包括 v。因 C 是一个环，所以连结 s_1, s_2 的边至少有两条，其中至少有一条不为 e 的边 e'，其权不大于 e，由 cut property 可知，存在不含有 e 的最小生成树。

b) 因为每次删除的都是图 G 中权最大的环边，由题述中的性质可证。

c) 参考 Ex.3.11。

d) $O\left(|E|^2\right)$。

Ex.5.22

a) 不变。

b) 将 e 加入 T 中，必然形成一个环，将该环中权值最大的边去掉即可。

c) 不变。

d) 在图 G 中找到 e 所在的环（如果不存在则最小生成树不变），设在环中且不在 T 中的权值最小的边为 e'，如果 $w(e') < \hat{w}(e)$，则在树 T 中将 e 替换为 e'。

Ex.5.23

在图G中，将U删除后，得到子图G'，求出G'的最小生成树T'（如果G'并非是连通的，则图G不存在U全部是叶子节点的最小生成树）。对于U中的每个顶点v，遍历该顶点的所有邻边，找到与T'相连结（利用并查集）且权值最小的一条边e，将e加入到T'中。在按上述过程处理完U中的所有顶点后，T'即为所求。

Ex.5.24

采用记账分析法，设每次改变一个二进制位需要付出一块钱的代价。在将某个二进制位b从0变成1的时候，我们给b两块钱，其中一块钱用于这次改变操作，另外一块钱余下来作为b的存款，这样以后将b从1变回到0时就不用再给钱了。由于在每次*increment*操作中只将某一位从0变为1（另外还可能需要将若干位从1变成0，但这些操作可以由这些位自己的存款来支付），所以只用付出两块钱，而*reset*操作也是不用再给钱的。于是n次*increment*操作和*reset*操作总共只需要付出最多$2n$块钱，即时间复杂度为$O(n)$。

Ex.5.25

并查集，先对每个变量进行*makeset*。然后处理相等约束，对相等的变量进行*union*操作。最后检查不等约束，若不等号两边的变量位于同一个集合，则说明该约束条件是不可满足的，若所有位于不等号两边的变量都位于不同集合中，即说明该约束条件是可满足的。

Ex.5.26

a) 比如$(3,3,1,1)$。

b) 1. 如果v_1不与$v_2, v_3, ..., v_{d_1+1}$之中的某个顶点相连（设为v_i），则v_1肯定与v_{d_1+1}之后的某个顶点相连（设为v_j），显然$i < j \le n$，且$\{v_1, v_i\} \notin E$，$\{v_1, v_j\} \in E$。因为所有顶点的度都为正，所以v_i的邻结点集$N(i)$肯定不为空。若$N(i)$中的所有顶点都与v_j相连，又因v_1与v_j相连，可得$d_i < d_j$，矛盾。于是可知，在$N(i)$中必存在一个顶点u，使得$\{u, v_i\} \in E$，$\{u, v_j\} \notin E$，得证。

2. 将v_1连向v_i，u连向v_j，并删除原来的$\{v_1,v_j\}$、$\{u,v_i\}$。

3. 如果v_1不与$v_2,v_3,...,v_{d_1+1}$之中的某个顶点相连，则执行第 2 步的变换，直到v_1与所有$v_2,v_3,...,v_{d_1+1}$都相连为止。

c) 首先找到$d_1,d_2,...,d_n$的最大元素d_i，然后再找到前d_i+1大的元素。将d_i从序列中去除掉，然后将随后的d_i个元素减 1。不断的重复上述过程，如果其中某一步出现负数或者是有$d_i+1>n$，则说明不存在图G满足该度序列。由于寻找前k大元素可以在$O(n)$内完成，所以该算法的时间复杂度为$O(n^2)$。或者可以采用二项堆，时间复杂度为$O((n+m)\log n)$。

Ex.5.27

利用给定的关系建立无向图$G(V,E)$，然后求出每个顶点的度。遍历所有顶点：若有顶点v的度满足关系$d(v)<5$，显然v就不能来参加 party 了，将v加入到一个待删除队列中。若$d(v)>|V|-1-5$，那么v也不可能来参加 party，因为即使将v的某个邻结点排除在邀请名单之外，这使得$d(v)$减小了 1，但是同时$|V|$也减小了 1，所以依然保持$d(v)>|V|-1-5$。所以，也将v加入到待删除队列中。遍历完成之后，将待删除队列中的所有顶点删除，同时更新这些顶点的邻结点的度。然后在剩余的子图上重复这个遍历过程，直到某一轮遍历完成后，待删除队列为空为止。因为$|E|=O(|V|^2)$，所以整个算法的时间复杂度为$O(|V|^2+|E|)=O(n^2)$。

Ex.5.28

略。

Ex.5.29

与二叉 Huffman 树的构造过程类似，不同的是每次需要选出三颗权值最大的子树。

Ex.5.30

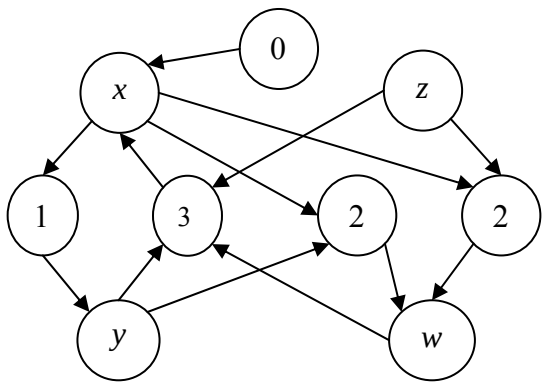
令 $w_i = f_i \bullet c_i$，则 $cost = \sum_{i=1}^{n} w_i \bullet l_i$，这就又回到了平凡的 Huffman 树形式。所以只需要以 w_i 为权构造 Huffman 树即可。

Ex.5.31

如果第 i 个被服务的客人需要的服务时间为 t_i，于是所有客人需要等待的总时间为 $w = (n-1)t_1 + (n-2)t_2 + \cdots + t_{n-1}$。不难看出，欲使 w 取到最小值，只需要按照 t_i 的升序安排服务顺序即可，时间复杂度为 $O(n \log n)$。

Ex.5.32

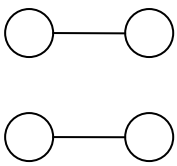
首先读取所有的 Implications 并按照以下方式生成一张有向图：对于每一个 Horn clause，将每一个左手边的 literal（后简称为 l-literal）连接到一个中间节点，并利用这个中间节点来记录这些 l-literals 的个数，然后将中间节点连接到 r-literial（即右手边的 literal）。例如 $(w \wedge y \wedge z) \Rightarrow x, (x \wedge z) \Rightarrow w, x \Rightarrow y, \Rightarrow x, (x \wedge y) \Rightarrow w$ 可以被表示为：



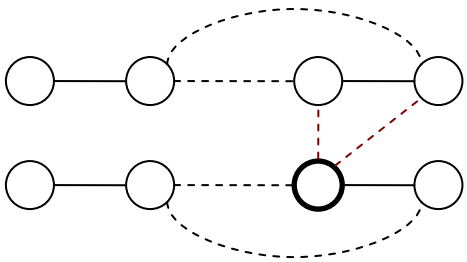
接下来先将所有 literals 的值设为 *false*，然后进行 DFS：从“0”节点出发，将它所连结的 literal 设为 *true*，同时将这个 literal 所连的中间结点的值减 1，如果有某个中间结点的值减 1 后为 0，则继续在这个结点上进行搜索，否则跳出。DFS 过程结束后再对 negative clauses 进行验证即可。

Ex.5.33

用归纳法来构造，设 o 为最优解，g 为贪心解。首先考虑 $n=4$ 的情况，我们可以很容易构造出下面这个图使得 $o=2$，$g=\log 4$：



现在考虑 $n'=8$ 的情况，现在需要考虑的是怎么将两个 $n=4$ 情况的图连结在一起，并使得 $o'=o=2$，$g'=\log 8=g+1$。下面是一种构造：



从上图中很容易看到整个构造过程是怎么进行的，首先，黑实线部分为两个 $n=4$ 的原始构造图，然后用黑色虚线将某一边的“中心点”（即最优解所选的点）与右边的相对应的那一半顶点连结起来，这保证了 $o'=o=2$，最后用红色虚线将右边边的某一个顶点（加粗）与其余所有节点连结起来，这将使得该点的覆盖的顶点数（5）大于任何一个“中心点”所覆盖的顶点数（4）。因此，贪心法第一个选择的节点将是该结点，此时，由于右边所有顶点都已被覆盖，而左边除去左下方的那个“中心点”外，其余均未被覆盖。可以证明在 $n\geq 4$ 时，左边这个中心点被覆盖并不会改变这半边的 g 值，于是有 $g'=g+1$。注意，在经过上述合并操作之后，“中心点”并没有改变，而且所覆盖的顶点数始终为 $\frac{n}{2}$，所以这使得这个构造可

以推广到一般情形，即是有：$o\left(2^{k+1}\right)=o\left(2^{k}\right)$，$g\left(2^{k+1}\right)=g\left(2^{k}\right)+1$，于是得：$o\left(n\right)=2$，$g\left(n\right)=\log n$，即为题中所求。

6 Dynamic programming

Ex.6.1

最大子段和问题。设 s_j 是以 j 为结束位置的最大子段和，则有 $s_j = \max\left(s_{j-1} + a_j, a_j\right)$，于是整个串的最大子段和即为 $\max\limits_{1\le j\le n}\left(s_j\right)$。

Ex.6.2

设 p_i 表示停留在第 i 个旅馆时的总最小罚值，其中 $p_0 = 0$。枚举上一个落脚点位置 j，则有 $p_i = \min\limits_{0\le j<i}\left(p_j + \left(200 - \left(a_i - a_j\right)\right)^2\right)$，于是 p_n 即为整个旅程的最小罚值。

Ex.6.3

设 r_i 表示在前 i 个位置开分店的最大总利润，$r_0 = 0$。若 j 为上一个满足 $m_i - m_j \ge k$ 的位置，则根据是否在当前位置开分店可以分成两种情况，取其中能获取较大利润的选择，也即是有 $r_i = \max\left(r_{i-1}, r_j + p_i\right)$，最后返回 r_n 即可。

Ex.6.4

设 v_i 表示前 i 个字符组成的子串是否能被有效的分隔，$v_0 = true$。若用 OR 表示连续求或，$\wedge$ 表示逻辑与，$substring\left(m,n\right)$ 表示序号位于 $\left(m,n\right]$ 区间内的字符组成的子串，则有 $v_i = \underset{0<=j<i}{OR}\left(v_j \wedge dict\left(substring\left(j,i\right)\right)\right)$。最后如要输出分割后的单词，记录一下状态转移路径即可。

Ex.6.5

a) 设 n_i 为在某一列的前 i 行的格子里放置石子的 pattern 数目，初始化 $n_0 = 1, n_1 = 2$。类似于 Ex.6.3，根据是否在第 i 个位置放置石子分成两个情况，

并将这两种情况的 pattern 数相加可得 $n_i = n_{i-1} + n_{i-2}$。这正好就是 Fibonacci 数列，于是可得 $n_4 = 8$。

b) 设这 8 种 type 依次编号为 0-7，并且已有函数 $c(i, j)$ 来判断 i，j 两种 type 是否兼容。再设 $s(i,t)$ 为在前 i 列放置石子且最后一列的 type 为 t 时的最优值，$v(i,t)$ 表示在第 i 列按照 t 样式放置石子时所得到的值。于是有 $s(i,t) = \max_{c(t,s)=true} \left(s(i-1,s) + v(i,t)\right)$。最后返回 $\max_{0 \le t < 8} \left(s(n,t)\right)$ 即可。注意在这里因为石子的个数为 $2n$，所以不需要考虑石子不够用的情况。

Ex.6.6

先建立好查询表 $m[3][3]$，$n[3][3]$，其中 $m[r][i] \bullet n[r][i] = r, (1 \le i \le 3)$。然后设 $b(i, j, t)$ 表示位于 $[i, j]$ 之间的子表达式相乘是否能得到 t。则有状态转移方程：

$$b(i, j, t) = \underset{i \le k < j}{OR} \left(\underset{1 \le l \le 3}{OR} \left(b(i, k, m[t][l]) \wedge b(k+1, j, n[t][l]) \right) \right)$$

式子好像挺复杂，其实思路还是很清楚的，跟矩阵链乘法类似。最后返回 $b(1, n, a)$ 即可。

Ex.6.7

定义 $p(i, j)$ 来表示在子串 $x[i, j]$ 是否为回文串。于是有以下关系式：

$$\begin{cases} if \ \ i \ge j : p(i, j) = ture \\ if \ \ x[i] = x[j] : p(i, j) = p(i+1, j-1) \\ otherwise : p(i, j) = \ false \end{cases}$$

枚举所有子串，按照上面的关系式依次检查其是否为回文串，然后找出最长的回文子串即可。由于子串的个数为 $O(n^2)$，所以整个算法的时间复杂度也为 $O(n^2)$。

Ex.6.8

注意这里要求子串连续，所以与最长公共子序列问题略有不同。定义 $L(i, j)$ 为子串

$x[0,i]$ 和 $y[0,j]$ 的最长右对齐公共子串长度（右对齐的意思是指这个公共子串是 $x[0,i]$ 和 $y[0,j]$ 的后缀）。于是有：

$$\begin{cases} if\ \ x[i] \neq y[j] : L(i,j) = 0 \\ otherwise : L(i,j) = L(i-1, j-1) + 1 \end{cases}$$

求得所有的 $L(i,j)$ 之后，找出最长的一条即可。时间复杂度为 $O(mn)$。

Ex.6.9

首先将 m 个分割点按位置排序后存入数组 $d[1,...,m]$，然后定义 $c(i,j,r,s)$ 为分割子串 $s[i,j]$ 所需要的代价，且其中所用的分割点为子数组 $d[r,s]$。于是有：

$$\begin{cases} c(i,j,r,s) = \min\limits_{r \le k \le s} \left(l(i,j) + c(i, d[k], r, k-1) + c(d[k]+1, j, k+1, s) \right) \\ l(i,j) = j - i + 1 \end{cases}$$

上式中的 $l(i,j)$ 为子串 $s[i,j]$ 的长度，也即是该子串第一次被划分时所需要的代价。

最后返回 $c(1,n,1,m)$ 即可，整个算法的时间复杂度为 $O(n^2 m^2)$。

Ex.6.10

设 $h(i,j)$ 为掷出前 i 个硬币，出现 j 个人头的概率。于是有：

$$\begin{cases} if\ \ j > i : h(i,j) = 0 \\ else : h(i,j) = p_i \bullet h(i-1, j-1) + (1 - p_i) \bullet h(i-1, j) \end{cases}$$

最后返回 $h(n,k)$ 即可，时间复杂度为 $O(nk)$。

Ex.6.11

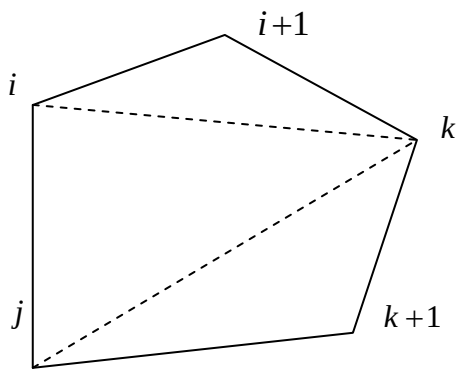
定义 $L(i,j)$ 为子串 $x[0,i]$ 和 $y[0,j]$ 的最长公共子序列长度，于是有：

$$\begin{cases} if\ \ i=0\ or\ \ j=0:L(i,j)=0 \\ else\ if\ \ x[i]=y[j]:L(i,j)=L(i-1,j-1)+1 \\ else:L(i,j)=\max\left(L(i-1,j),L(i,j-1)\right) \end{cases}$$

最后返回 $L(n,m)$ 即可，注意与 Ex.6.8 的区别。

Ex.6.12

首先按照提示进行编号，再定义函数 $dist(i,j)$ 表示顶点 i， j 之间的距离。对于子问题 $A(i,j)$，考虑边 $e(i,j)$，显然，这条边在最优分割后一定在某个三角形内，如下图：



为了找出这个三角形，枚举与边 $e(i,j)$ 相对的顶点 $k\in(i,j)$，于是有以下关系式：

$$\begin{cases} if\ \ k=i+1:A(i,j)=dist(k,j)+A(k,j) \\ if\ \ k=j-1:A(i,j)=dist(i,k)+A(i,k) \\ else:A(i,j)=dist(i,k)+dist(k,j)+A(i,k)+A(k,j) \end{cases}$$

然后取其中使得代价最优的 k 值即可。整个算法的时间复杂度为 $O(n^2)$。

Ex.6.13

a) 比如 $\{1,2,10,3\}$。

b) 定义 $r(i,j)$ 表示在子串 $s[i,j]$ 中进行游戏时所获得的最大分数值。假设对手也

总是采取最优的策略，于是有：

$$\begin{cases} r_i = s_i + \min\left(r\left(i+2, j\right), r\left(i+1, j-1\right)\right) \\ r_j = s_j + \min\left(r\left(i+1, j-1\right), r\left(i, j-2\right)\right) \\ r\left(i, j\right) = \max\left(r_i, r_j\right) \end{cases}$$

其中 r_i 是当玩家 1 第一步选择卡片 i 时所能得到的分值，r_j 则是选择卡片 j 时所能得到的分值。$r\left(i, j\right)$ 当然是取 r_i 与 r_j 的较大者。此外还需要记录下这一步的选择 $c\left(i, j\right)$：

$$\begin{cases} if\ \left(r_i > r_j\right) : c\left(i, j\right) = pickfront \\ else\ c\left(i, j\right) = pickback \end{cases}$$

作为边界情况，当 $j-i=1$ 时，显然有 $r\left(i, j\right) = \max\left(s_i, s_j\right)$。

Ex.6.14

每次沿着边沿剪下一块布，这样剩余的布料又可以形成最多三块矩阵面料。设 $P\left(w,h\right)$ 是在宽为 w，高为 h 布料上所能得到的收益（显然，$P\left(w,h\right) = P\left(h,w\right)$），考虑每一次切割式样的两种情况，于是有：

$$\begin{cases} P_h\left(w,h,i\right) = c_i + P\left(a_i, h-b_i\right) + P\left(w-a_i, b_i\right) + P\left(w-a_i, h-b_i\right) \\ P_v\left(w,h,i\right) = c_i + P\left(b_i, h-a_i\right) + P\left(w-b_i, a_i\right) + P\left(w-b_i, h-a_i\right) \\ P\left(w,h\right) = \max\limits_{1\le i\le n}\left(\max\left(P_h\left(w,h,i\right), P_w(w,h,i)\right)\right) \end{cases}$$

其中，$P_h\left(w,h,i\right)$ 是将第 i 个式样按照给定的形状（即宽为 a_i，高为 b_i）直接切下来时的收益，而 $P_v\left(w,h,i\right)$ 则是将第 i 个式样按照旋转 90 度后的形状（即宽为 b_i，高为 a_i）切下来时的收益。最后返回 $P\left(X,Y\right)$ 即可。

Ex.6.15

设 $P\left(i, j\right)$ 为此时 A 队赢的概率，则有：

$$\begin{cases} if\ (i=n): p(i,j)=1 \\ if\ (j=n): p(i,j)=0 \\ else: p(i,j)=p(i+1,j)/2+p(i,j+1)/2 \end{cases}$$

Ex.6.16

和 TSP 不同的地方在于本题目的起始顶点（home）可以访问多次，而且不必访问所有的顶点。以 $B(S,j)$ 来表示在 garage sale 集为 S，结束位置为 j 的子问题上的最大收益值。注意 j 的前导顶点既可能是某个 garage sale，也可能是 home，于是有：

$$\begin{cases} B(S,j)_g = \max\limits_{i\in S, i\neq j}\left(B(S-\{j\},i)+v_j-d_{ij}\right) \\ B(S,j)_h = \max\limits_{i\in S, i\neq j}\left(B(S-\{j\},i)+v_j-d_{i0}-d_{0j}\right) \\ B(S,j)=\max\left(B(S,j)_g, B(S,j)_h, v_j-d_{0j}\right) \end{cases}$$

最后找出最大的 $\max\limits_{j\in\{g\}} B(\{g\},j)-d_{j0}$ 即可。当然，如果这个值小于 0，则返回 0。

Ex.6.17

设 $b(i)$ 表示面值 i 是否能被找开。于是有：

$$\begin{cases} if\ (i<0): b(i)=false \\ if\ (i=0): b(i)=true \\ else: b(i)=\underset{1\le j\le n}{OR}\left(b(i-x_j)\right) \end{cases}$$

最多共有 $O(v)$ 个状态，每次状态转移的代价为 $O(n)$，于是整个算法的时间复杂度为 $O(nv)$。

Ex.6.18

设 $b(i,j)$ 表示只使用前 i 个币种时是否能找开面值 j。考虑是否使用第 i 个币种可以分成两种情况：如果使用，因为每个币种只能使用一次，有 $b(i,j)=b(i-1,j-x_i)$，

如果不使用，则$b(i,j)=b(i-1,j)$，于是整个状态转移方程可表示为：

$$\begin{cases} b(i,j)_u = b(i-1, j-x_i) \\ b(i,j)_n = b(i-1,j) \\ b(i,j) = b(i,j)_u \vee b(i,j)_n \end{cases}$$

其中$\vee$表示逻辑或。最多共有$O(nv)$个状态，状态转移代价为$O(1)$，于是整个算法的时间复杂度为$O(nv)$。

Ex.6.19

同 Ex.6.17，只不过需要在状态空间中增加一个维度来记录可以使用的硬币数k。即是有：$b(k,i)=\underset{1\le j\le n}{OR}\left(b\left(k-1,i-x_j\right)\right)$。

Ex.6.20

设$c(i,j)$为单词子集$w[i,j]$所构成的最优查找树的查找代价，注意最优查找树的子树也同样是最优查找树，于是有：$c(i,j)=\min_{i\le k\le j}\left(c(i,k-1)+c(k+1,j)\right)+\sum_{i\le k\le j} p_k$。

Ex.6.21

即是最大独立集的补集。

Ex.6.22

请参考 Ex.6.18。

Ex.6.23

设$p(i,j)$为只考虑前i台机器，且总预算为j时的最大正常运行概率。再令$t(j)=\sum_{i=1}^{j} c_i$，于是有：$p(i,j)=\max_{1\le k, kc_i\le j-t(i-1)}\left(p(i-1,j-kc_i)\bullet\left(1-(1-r_i)^k\right)\right)$。

Ex.6.24

a) 观察书中 Figure 6.4，由于计算某一个方块的只需要知道该方块的左上、上、左

三个方向上的相邻方块的值，于是在每一时刻只需要存储当前行和上一行的状态值就以足够，为方便可以用滚动数组来存储，空间复杂度为$O(n)$。

b) 为了与书中的代码保持一致，先将n,m互换一下，即还是用m表示行，而n表示列。用一个$n/2+1$列（因为只需要考虑$j>n/2$的情况）的滚动数组K来存储当前行和上一行的k值，然后在 edit distance 算法中加入以下伪代码：

```
K(i,0) = i
if j > n/2
  if E(i,j) == E(i-1,j) + 1
    K(i,j-n/2) = K(i-1,j-n/2)
  else if E(i,j) == E(i,j-1)+1
    K(i,j-n/2) = K(i,j-n/2-1)
  else if E(i,j) == E(i-1,j-1) + diff(i,j)
    K(i,j-n/2) = K(i-1,j-n/2-1)
```

c) 由递归关系容易看出，这个算法的时间复杂度为：

$$\begin{aligned} T(m,n) &= O(mn)+\frac{1}{2}O(mn)+\frac{1}{4}O(mn)+\frac{1}{8}O(mn)\cdots \\ &= O(mn) \end{aligned}$$

这是个典型的时间换空间的算法，虽然时间复杂度还是为$O(mn)$，但是常数因子却增大了一倍，算法实现的难度也大大提高了。

Ex.6.25

设$b(i,c_1,c_2,c_3)$表示前i个元素是否能被分成三堆，且这三堆的和分别为c_1，c_2，c_3（还可以限定$c_1\le c_2\le c_3$以缩减状态空间大小）。再令s_i表示$\sum_{j=1}^{i}a_j$。于是有以下状态转移关系：

$$\begin{aligned} b(i,c_1,c_2,c_3) = & b(i-1,c_1-a_i,c_2,c_3) \\ & \vee b(i-1,c_1,c_2-a_i,c_3) \\ & \vee b(i-1,c_1,c_2,c_3-a_i) \end{aligned}$$

其中$\vee$表示逻辑或。最后返回$b\left(n,\frac{s_n}{3},\frac{s_n}{3},\frac{s_n}{3}\right)$即可，时间复杂度为$O(ns_n{}^3)$。

Ex.6.26

类似于最短编辑距离，有：

$$S(i,j)=\max\begin{pmatrix}S(i-1,j)+\delta(x[i],-)\\S(i,j-1)+\delta(-,y[j])\\S(i-1,j-1)+\delta(x[i],y[j])\end{pmatrix}$$

Ex.6.27

重定义子问题为 $S(i,j,k)$，其中 $k\in\{0,1,2\}$，分别对应在最末位置上的这三种匹配：

$x[i]\leftrightarrow y[j]$、$-\leftrightarrow y[j]$、$x[i]\leftrightarrow -$，于是有：

$$s(i,j,0)=\max_{0\le k<3}\left(s(i-1,j-1,k)\right)+\delta(x[i],y[j])$$

$$s(i,j,1)=\max\begin{pmatrix}s(i,j-1,0)-(c_0+c_1)\\s(i,j-1,1)-c_1\\s(i,j-1,2)-(c_0+c_1)\end{pmatrix}+\delta(-,y[j])$$

$$s(i,j,2)=\max\begin{pmatrix}s(i-1,j,0)-(c_0+c_1)\\s(i-1,j,1)-(c_0+c_1)\\s(i-1,j,2)-c_1\end{pmatrix}+\delta(x[i],-)$$

Ex.6.28

结合 Ex.6.1 和 Ex.6.26，有：

$$S(i,j)=\max\begin{pmatrix}\max\left(S(i-1,j)+\delta(x[i],-),\delta(x[i],-)\right)\\\max\left(S(i,j-1)+\delta(-,y[j]),\delta(-,y[j])\right)\\\max\left(S(i-1,j-1)+\delta(x[i],y[j]),\delta(x[i],y[j])\right)\end{pmatrix}$$

最后找到 $\max\limits_{1\le i\le n,1\le j\le m}\left(S(i,j)\right)$ 即可。

Ex.6.29

设 $V[j]$ 表示在只考虑 $x[1,j]$ 的子问题上的最大权值总和。容易看出，一定存在

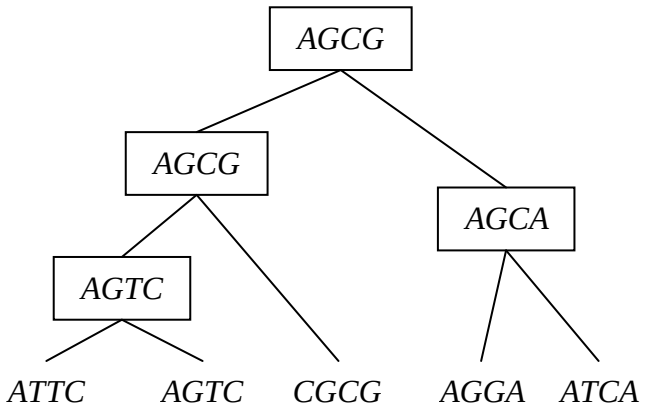
$x[1,j]$ 的某个后缀 $x[i,j]$，使得 $V[j]$ 为以下三个值之一：$V[i-1]$、$V[i-1]+w(i,j)$、$w(i,j)$。于是可得以下关系：

$$V[j]=\max_{1\le i<j}\left(\max\left(V[i-1],V[i-1]+w(i,j),w(i,j)\right)\right)$$

其中 $w(i,j)$ 即是三元组 (i,j,w) 对应的 w 值。算法的时间复杂度为 $O(n^2)$。

Ex.6.30

a) 如下图：



b) 根据提示，每次只考虑一位。定义状态 $S(T,c)$，其中 $c\in\{A,C,G,T\}$，来表示问题域为子树 T，且 T 的根节点为 c 时的最小分值，于是有：

$$\delta(i,j)=\begin{cases}1 \ if \ \ i=j\\ 0 \ otherwise\end{cases}$$

$$L=left-subtree(T)$$

$$R=right-subtree(T)$$

$$S(T,c)=\min_{a\in\Sigma,b\in\Sigma}\left(S(L,a)+S(R,b)+\delta(c,a)+\delta(c,b)\right)$$

由于每个内结点都有两个儿子，于是可得内结点的数目为叶子节点的个数减一，也即是有状态数 $s=O(n)$。又因为状态转移的代价为 $O(1)$，所以整个算法的时间复杂度为 $O(nk)$。

7 Linear programming and reductions

Ex.7.1

在 $x=5$， $y=2$ 时取得最大值 31，图略。

Ex.7.2

设从 Kansas 运往 New York 和 California 的荞麦分别为 x 和 $15-x$，从 Mexico 运往 New York 和 California 的荞麦分别为 y 和 $8-y$，于是可以得到以下的线性规划问题：

$$\begin{aligned}&\text{minimize } 2x+3(15-x)+4y+8-y\\&x+y=10\\&x,y\geq 0\end{aligned}$$

容易解得最小运费为 43，此时 $x=10$， $y=0$。

Ex.7.3

设携带的材料 1、材料 2、材料 3 分别为 x， y， z 立方米，于是可得 LP：

$$\begin{aligned}&\text{maximize } 1000x+1200y+12000z\\&x+y+z\leq 60\\&2x+y+3z\leq 100\\&0\leq x\leq 40\\&0\leq y\leq 30\\&0\leq z\leq 20\end{aligned}$$

当 $x=0$， $y=30$， $z=20$ 取得最大值 296000。

Ex.7.4

Regular Duff： x，Duff strong： y，于是有 LP：

$$\begin{aligned}&\text{maximize } x+1.5y\\&x-2y\geq 0\\&x+y\leq 3000\\&x,y\geq 0\end{aligned}$$

在 $x=2000$， $y=1000$ 时取得最大值 3500。P.S.感觉这题似乎应该再多一个约束。

Ex.7.5

a) Frisky Pup： x，Husky Hound： y，于是得 LP：

$$\begin{aligned}
&\text{maximize } \left(7-1\times1-2\times1.5-1.4\right)x+\left(6-1\times2-2\times1-0.6\right)y\\
&x+2y\le240000\\
&1.5x+y\le180000\\
&x\le110000\\
&x,y\ge0
\end{aligned}$$

b) 图略，解得最大利润为 222000，此时 $x=60000$， $y=90000$。

Ex.7.6

比如下面这个 LP：

$$\begin{aligned}
&\text{maximize } 2y-x\\
&x\ge y\\
&y\le100\\
&x,y\ge0
\end{aligned}$$

当 $x=100$， $y=100$ 时取得最大值 100。

Ex.7.7

a) 对于任意 a,b，该 LP 总是可行的。

b) $ab=0$。

c) $ab\neq0$， $a\neq b$。

Ex.7.8

除 a,b,c 外，再增加 8 个变量，$e_i,1\le i\le7$ 和 me，分别表示在每个采样点的绝对误差和总的最大绝对误差。于是可得到 LP 如下：

$$
\begin{aligned}
&\text{minimize } me \\
&for\ i = 1,7 \\
&\quad e_i \geq ax_i + by_i - c \\
&\quad e_i \geq -\left(ax_i + by_i - c\right) \\
&\quad me \geq e_i
\end{aligned}
$$

Ex.7.9

比如下面这个简单问题：

$$
\begin{aligned}
&\text{maximize } xy \\
&x + y \leq 10 \\
&x, y \geq 0
\end{aligned}
$$

根据算法几何平均值不等式知当 $x = y = 5$ 时， xy 取得最大值 25。

Ex.7.10

最大流为 13，相应的最小割为 $\left(\{S, C, F\}, \{A, B, D, E, G, T\}\right)$。

Ex.7.11

与之对偶的 LP 如下：

$$
\begin{aligned}
&\text{minimize } 3s + 5t \\
&2s + t \geq 1 \\
&s + 3t \geq 1 \\
&s, t \geq 0
\end{aligned}
$$

最优值为 2.2，此时 $x = 0.8, y = 1.4$（原问题）或者 $s = 0.4, t = 0.2$（对偶问题）。

Ex.7.12

$\boxed{1} + \boxed{2} \times 1/2 \Rightarrow x_1 - x_3/2 \leq 1.5$，又因 $x_3 \geq 0$，于是 $x_1 - 2x_3 \leq x - x_3/2 \leq 1.5$。

Ex.7.13

a) 略。

b) 设 R 和 C 的策略分别为 $\{x_1, x_2\}$ 和 $\{y_1, y_2\}$，其中 x_1, y_1 为出人头的概率，x_2, y_2

为出字的概率。于是可得以下 LP 及其对偶形式：

$$\begin{aligned} &\max\ z && &&\min\ w \\ &z \le x_1 - x_2 && &&w \ge y_1 - y_2 \\ &z \le -x_1 + x_2 && &&w \ge -y_1 + y_2 \\ &x_1 + x_2 = 1 && &&y_1 + y_2 = 1 \\ &x_1, x_2 \ge 0 && &&y_1, y_2 \ge 0 \end{aligned}$$

不难看出，该博弈的 value 为 0，最佳策略 $\{x_1, x_2\} = \{y_1, y_2\} = \left\{\frac{1}{2}, \frac{1}{2}\right\}$。

Ex.7.14

设 Joey 与 Tony 的策略分别为 $\{x_1, x_2\}$，$\{y_1, y_2, y_3\}$，有 LPs：

$$\begin{aligned} &\max\ z && &&\min\ w \\ &z \le 2x_1 - x_2 && &&w \ge 2y_1 - 3y_3 \\ &z \le -2x_2 && &&w \ge -y_1 - 2y_2 + y_3 \\ &z \le -3x_1 + x_2 && &&y_1 + y_2 + y_3 = 1 \\ &x_1 + x_2 = 1 && &&y_1, y_2, y_3 \ge 0 \\ &x_1, x_2 \ge 0 \end{aligned}$$

解得该博弈的 value 为-1，最佳策略 $\{x_1, x_2\} = \left\{\frac{1}{2}, \frac{1}{2}\right\}$，$\{y_1, y_2, y_3\} = \left\{0, \frac{2}{3}, \frac{1}{3}\right\}$。

Ex.7.15

直接用 lingo 暴之，得 $value = -\frac{1}{3}$。

Ex.7.16

在 lingo 中建立模型如下：

```
min = cal;
cal = t*21 + l*16 + s*371 + c*346 + o*884;
t*0.85 + l*1.62 + s*12.78 + c*8.39 >= 15;
t*0.33 + l*0.2 + s*1.58 + c*1.39 + o*100 >= 2;
t*0.33 + l*0.2 + s*1.58 + c*1.39 + o*100 <= 6;
t*4.64 + l*2.37 + s*74.69 + c*80.7 >= 4;
t*9 + l*8 + s*7 + c*508.2 <= 100;
```

```
l + s <= t + c + o;
```

注意在 lingo 中所有变量都是默认为非负的，所以不需要再加非负的约束。算得最优解为 $\{t,l,s,c,o\}=\{5.88,5.84,0.04\}$，单位为百克，此时的热量值为 232.5 千卡。

Ex.7.17

a) 最大流为 11，相应的最小割为 $(\{S,A,B\},\{C,D,T\})$。

b) 图略，S 可达的点集为 $\{A,B\}$，T 可达的点集为 $\{S,A,B,C,D\}$。

c) 瓶颈边有：$e(A,C)$，$e(B,C)$。

d) 略。

e) 设原图为 $G(V,E)$，余图为 $R(V,F)$，那么一条瓶颈边 $e(i,j)$ 具有以下性质：

$e(i,j)\in E$，同时在余图 R 中，$S\to i$ 且 $j\to T$（$S\to i$ 表示存在一条从顶点 S 到 i 的路径，$j\to T$ 同）。于是，在 R 中添加 $e(i,j)$ 后就形成了一条增量路径，从而增加了最大流。根据这些性质，可以在线性时间内找到所有的瓶颈边，步骤依次如下：先在余图 R 中，以 S 为源点 DFS 找到 S 可达的顶点集 I；再在 R 的反图 R^T 中以 T 为源点进行 DFS，得到可达 T 的顶点集 J；最后遍历边集 E，对于某条边 $e(i,j)$，如果 $i\in I$，$j\in J$，则说明 $e(i,j)$ 是瓶颈边。

Ex.7.18

a) 添加一个新顶点 S 连接到原来所有的源，再添加一个新顶点 T，并将原来的汇都连向 T，然后求 S 到 T 的最大流即可。当然，这些新添加的边的容量必须足够大，可以设成 INFINITE。

b) 把每个顶点 v 拆成两个顶点 v_1、v_2，添加一条边 $e(v_1,v_2)$，并令 $e(v_1,v_2)$ 的容量为原来该顶点的容量。然后将原图中以 v 为终点的边改成以 v_1 为终点，以 v 为起点的边改成以 v_2 为起点。这样处理后，就成了一个平凡的最大流问题了。

c) 在 LP 模型中，对每条边多加一个约束。

d) 修改一下流量守恒的约束即可。

Ex.7.19

首先由给定的最大流解可以很容易的得到余图 $R(V,F)$，然后在 R 中以 S 为源点进行 DFS：如果存在着有向路径 $S \to T$，根据增量路径定理可知，一定还可以构造出更大的流；否则说明该解的确是最优的。

Ex.7.20

为一条边关联 k 个变量 $f_e^{(1)}, f_e^{(2)}, \cdots f_e^{(k)}$，其中 $f_e^{(i)}$ 表示从源 s_i 出发，在边 e 内的流量。那么流 $f^{(i)}$ 的大小即是从 s_i 出发，且在 t_i 汇集的各条支流的总和，也即是有 $f^{(i)} = \sum_{(u,t_i)\in E} f_{(u,t_i)}^{(i)}$。于是可以建立 LP：

$$
\begin{aligned}
&\max \sum f^{(i)} \\
&\forall e \in E : f_e^{(1)} + f_e^{(2)} + \cdots f_e^{(i)} \le c_e \\
&\forall v \in V : \sum_{uv\in E} f_{uv}^{(i)} = \sum_{uw\in E} f_{uw}^{(i)} \\
&f^{(i)} = \sum_{(u,t_i)\in E} f_{(u,t_i)}^{(i)} \ge d_i \\
&f_e^{(i)} \ge 0
\end{aligned}
$$

Ex.7.21

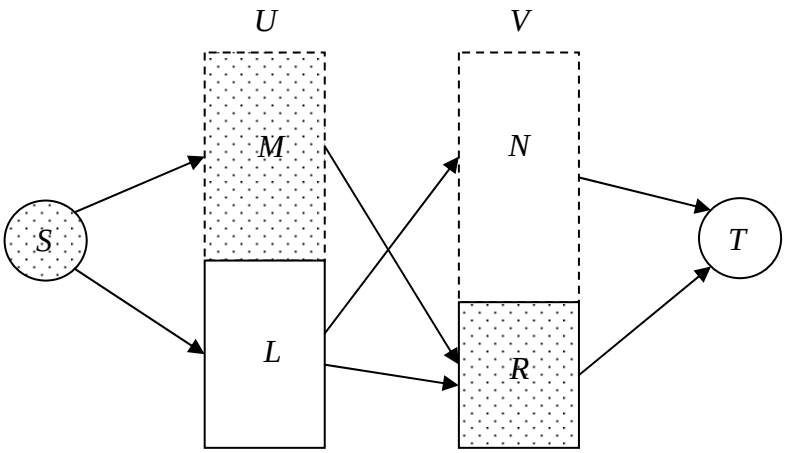
首先先求得最大流，再构造出余图 $R(V,F)$。显然，所谓的“关键边” $e(i,j)$ 的容量肯定已被占满了，这意味着在余图 R 中，顶点对 (i,j) 只存在反向边，即 $e(j,i) \in F$，$e(i,j) \notin F$。当然，在余图中 R 只存在反向边的顶点对不一定都是关键边的顶点，所以需要将这些不是关键边的顶点对剔除掉，方法如下：在 R 中以 S 为源点进行 DFS 搜索，设所经过的反向边集为 E'，那么对于所有 $e'(j,i) \in E'$，(i,j) 肯定不是关键边（Why？自己画画图就知道了）。于是，如果在余图 R 中，所有只存在反向边的顶点对集为 L，在 DFS 后得到的边集 E' 对应的顶点对集为 U，那么 $C = L - U$ 即是所求的关键边集。

Ex.7.22

参照 Ex.7.21 可知，在最大流已知的情况下，可以在线性时间内判断某条边是否是关键边。所以，如果 $e(u,v)$ 不是关键边，则修正后的最大流 $f'=f$，否则 $f'=f-1$。

Ex.7.23

考虑一个二分图 $G(U,E,V)$，其中顶点集 U、V 互不相交。设图 G 的最小顶点覆盖为 C，令 $L=U\cap C$，$R=V\cap C$，$M=U-L$，$N=V-R$。添加一个源 S 和汇 T 后，可得流网络如下：



在上图中，因为 $L\bigcup R$ 是最小顶点覆盖，所以从 M 到 N 不存在边。现在观察在上图中用阴影标记出来部分，它与剩余的部分形成了一个割 $(S\bigcup M\bigcup R, L\bigcup N\bigcup T)$。容易看出这个割的割边数目正好就是最小顶点覆盖的顶点数。于是求最小顶点覆盖 $L\bigcup R$ 也就转化成了求最小割 $(S\bigcup M\bigcup R, L\bigcup N\bigcup T)$。

另外，从这里也可以看出，在二分图中，最小顶点覆盖的顶点数等于最大匹配的边数，因为这二者的流模型是完全一样的。PS. 也可以从另外一个方面来证明：设最小顶点覆盖 C 的顶点数为 m，最大匹配的边数为 n，首先由匹配的性质易知 $m\geq n$。再回到上面的图，可以发现：对于 L 中任意 k 个顶点，都至少与 N 中的 k 个顶点相连结。否则就可以在 N 中找出小于 k 个顶点，替换掉 L 中的 k 个顶点，从而得到一个更小的顶点覆盖。同理，在 R 中的任意 k 个顶点也至少与 M 中的 k 个顶点相连结。由 Hall 定理，我们知道这意味着一个大小为 m 的顶点覆盖必然可以对应一个大小为 m 的匹配，所以又有 $n\geq m$。于是可知 $m=n$，得证。

Ex.7.24

a) 比如路径 $D \to F \to B \to E \to A \to H \to C \to I$ 。

b) 如果存在交互路径 P ，设在 P 中，属于 M 的边集为 P_M ，不属于 M 的边集为 P_{NM} ，由交互路径的定义知， P_{NM} 中的边刚好比在 P_M 中的边多一条，也即是 $\left|P_{NM}\right|-\left|P_M\right|=1$ 。于是在 M 中，将 P_M 替换为 P_{NM} 就可以构造一个比原来多一个边的匹配。这就说明了在最大匹配中不可能存在交互路径，但这只是证明了命题的一个方向，另外还需要证明的是，如果匹配 M 不存在交互路径，它一定就是最大匹配？注意到交互路径其实是与最大流模型中的增量路径等价的：设匹配 M 在流模型中的余图为 R ，若 M 不存在交互路径，那么在 R 中也就不存在增量路径，从而得知 M 一定是最大匹配。

c) 首先，为保证交互路径的交互性质，需要将所有的边加上方向。对于任意边 $e(i,j)\in E$ ，如果 $e(i,j)\notin M$ ，则规定方向为 $i \to j$ ，否则规定方向为 $j \to i$ （这样处理后得到的图其实就是在流模型中匹配 M 对应的余图）。然后从 V_1 中未被 M 覆盖的顶点集出发进行 BFS，若访问到 V_2 中的某个未被覆盖的顶点，即找到了一条交互路径。

d) 迭代的进行交互路径的搜索，每次找到一条交互路径都将使 M 的匹配数加 1。由于 $\left|M\right| \le \min\left(\frac{\left|V\right|}{2},\left|E\right|\right)$ ，所以最多只需迭代 $\min\left(\frac{\left|V\right|}{2},\left|E\right|\right)$ 次。由于每次迭代需时 $O\left(\left|V\right|+\left|E\right|\right)$ ，所以总时间复杂度为 $O\left(\left|V\right|\bullet\left|E\right|\right)$ 。

Ex.7.25

a) 由给定流网络可得 LP 及与其对偶的 LP 如下：

$\max\ f_{SA}+f_{SB}$	$\min\ y_1+3y_2+y_3+2y_4+y_5$
$f_{SA}, f_{AB}, f_{BT} \le 1$	$y_1+y_6 \ge 1$
$f_{AT} \le 2$	$y_2+y_7 \ge 1$
$f_{SB} \le 3$	$y_3-y_6+y_7 \ge 0$
$f_{SA}-f_{AB}-f_{AT}=0$	$y_4-y_6 \ge 0$
$f_{SB}+f_{AB}-f_{BT}=0$	$y_5-y_7 \ge 0$
$f_{SA}, f_{SB}, f_{AB}, f_{AT}, f_{BT} \ge 0$	$y_1, y_2, y_3, y_4, y_5, y_6, y_7 \ge 0$

b) 如上，用 lingo 可解得两者的最优值都为 2。

c) 先把答案写出来，至于为什么是这样的形式，看了后两问就知道了：

$$\min \sum_{e\in E} y_e c_e$$
$$\forall e(s,u) \in E : y_e + y_u \ge 1$$
$$\forall e(u,t) \in E : y_e - y_u \ge 0$$
$$\forall e(u,v) \in E, u \ne s, v \ne t : y_e - y_u + y_v \ge 0$$

d) 若一条路径为 $s \to u_1 \to u_2 \to u_3 \to \cdots \to u_n \to t$，于有由上面的 LP 可得：

$$y_{(s,u_1)} + y_{u1} \ge 1$$
$$y_{(u_1,u_2)} - y_{u_1} + y_{u_2} \ge 0$$
$$y_{(u_2,u_3)} - y_{u_2} + y_{u_3} \ge 0$$
$$\vdots$$
$$y_{(u_n,t)} - y_{u_n} \ge 0$$

将上面所有不等式相加即得 $y_{(s,u_1)} + y_{(u_1,u_2)} + y_{(u_2,u_3)} + \cdots + y_{(u_n,t)} \ge 1$。

e) 设流网格的一个割为 (L,R)，其中 $s \in L$，$t \in R$，那么这些变量的意义可以是这样：对于边 e，若 $y_e = 1$，那么表示 e 从 L 到 R 的一条边；对于顶点 u，$y_u = 1$ 即表示 $u \in L$，而 $y_u = 0$ 表示 $u \in R$。容易观察到，当这些变量被赋予这样的意义后，的确是与对偶 LP 中的约束关系相对应的，而目标函数恰好就是求最小割。

Ex.7.26

a) 首先需要明确的是，由于各个约束条件都为线性，所以实际上每个约束条件都

对应着一个半空间，这些半空间并起来形成了最后的可行域，于是可知这个可行域必然是非凹的。这也就表明了，如果$\{x\}$、$\{y\}$都是可行解，那么在它们之间的连线上的任意点$\{z\}=\omega\{x\}+(1-\omega)\{y\}$也同样是可行解。现在设在可满足系统$S$中的 not forced-euqal 不等式为$ie_1, ie_2, ie_3, ..., ie_4$，由 not forced-euqal 不等式的定义知必然存在一个解$\{x_1\}$使得ie_1不取等号，同样也存在$\{x_2\},\{x_3\},...\{x_n\}$分别使得$ie_2, ie_3, ..., ie_4$不取等号。于是我们可以取$\{x_a\}$为$\{x_1\},\{x_2\},...,\{x_n\}$的算术平均，由前面结论知$\{x_a\}$也肯定为$S$的一个解，也容易看出$\{x_a\}$可以使得$ie_1, ie_2, ..., ie_n$都不取等号，于是得证。

b) 分别在第i个不等式左边加上一个变量b_i（且有$b_i \geq 0$），然后以目标函数$\max \sum b_i$建立 LP。由于这个 LP 可能是无边界的，可以再加上约束条件$b_i \leq c_i$，其中c_i是某个正常数。如果在最优解中，$b_i = 0$，那么说明第i个不等式是 forced-euqal 的，否则为 not forced-euqal（待证明）。

Ex.7.27

设面值为x_i的硬币的使用数量为c_i，于是可得整数线性规划问题如下：

$$\begin{aligned} &\min\ v - \sum x_i c_i \\ &\sum x_i c_i \leq v \\ &c_i \geq 0 \end{aligned}$$

当然不能用一般的 LP 模型来求解，那样很有可能只能得到没有意义的分数解。

Ex.7.28

a) 设最短路径长度为s，于是只需证明$s \leq \sum_e l_e f_e$，且在某种情况下可以取得等号即可。我们知道，f可以分解成一条或者多条路径$p_1, p_2, p_3, ..., p_n$（见

Ex.7.31.c）。由于 $size(f)=1$，这意味着 $\sum_e l_e f_e$ 不小于这些路径的长度的某种加权平均，再由于这些路径的长度不小于 s，从而推出 $s \le \sum_e l_e f_e$。另外，若取最短路径为流路径，此时 $s = \sum_e l_e f_e$，这说明了上面的不等式可以取得等号。

b) 比划着写就行，如下：

$$
\begin{aligned}
&\min \sum_e l_e f_e \\
&\sum_{(s,u)\in E} f_{(s,u)} = 1 \\
&\sum_{(v,t)\in E} f_{(v,t)} = 1 \\
&\forall u \in V, u \ne s, u \ne t: \\
&\quad \sum_{(w,u)\in E} f_{(w,u)} - \sum_{(u,v)\in E} f_{(u,v)} = 0 \\
&\forall e \in E: f_e \ge 0
\end{aligned}
$$

c) 注意在上面的 LP 中，除最后的 $f_e \ge 0$ 之外，每个顶点各对应一个约束条件，共 $|V|$ 个约束条件。对于每个顶点 $v \in E$，将该顶点对应的约束条件乘上一个系数 x_v，最后再把所有的约束方程加起来可得 $\sum_{(u,v)\in E} (x_u - x_v) f_{(u,v)} = x_s - s_t$，联系到上面 LP 中的目标函数 $\min \sum_e l_e f_e$，容易看出其对偶 LP 即为题中所示。

d) 本题是求有向图的最短路，而前面是求无向图的最短路。

Ex.7.29

a) 用 x_i 来表示是否雇佣演员 i， y_j 表示是否得到老板 j 的投资，于是有 ILP：

$$
\begin{aligned}
&\max \sum y_j p_j - \sum x_i s_i \\
&for\ j \in [1,n] \\
&\quad for\ i \in L_j \\
&\qquad y_j \le x_i \\
&x_i, y_j \in \{0,1\}
\end{aligned}
$$

b) 在这里只需要说明，对于任意一个非整数解，都存在比它更优的整数解。由于 y_i 是由 x_i 决定的（最优情况下），因此可以只考虑 x_i 即可。首先，若非整数解 x_i 皆小于 1，若此时的收益值为负，那么直接令 $x_i = 0$ 就能得到一组更优的整数解。否则若此时的收益为正，设 $r = \max x_i$，显然 $0 < r < 1$。现在我们把每个 x_i 乘上一个放大系数 $\frac{1}{r}$，可以得到另一组解 x'_i，注意在这个新解中，有部分元素变成了 1。容易看出 x'_i 对应的收益值也是 x_i 对应的收益值的 $\frac{1}{r}$ 倍，因为此时的 y'_i 也可以提高为原来的 $\frac{1}{r}$ 倍。所以 x'_i 比 x_i 更优。现在来考虑 x_i 有部分元素为 1 的情况。设 $f = \max_{x_i<1} x_i$，那么将 x_i 中小于 1 的所有元素都乘于放大系数 $\frac{1}{f}$，或者将 x_i 中小于 1 的所有元素都设为 0，二者取其一，也必能得到一个更优的解 x'_i（这里就不再详细推导了）。通过前面的叙述可知，对于任意一个非整数解，经过上述若干次的变换之后，最终必然能得到一个更优的整数解。

Ex.7.30

该条件的必要性很显然：如果存在着完美匹配，那么任意一个男生子集 S 至少与 $|S|$ 个女生有一腿，否则必然出现一女同侍几夫。现在来证明充分性，即如果任意一个男生子集 S 都至少与 $|S|$ 个不同的女生相连结，那么一定存在完美匹配。考虑该二分匹配对应的流模型的任意一个割 (L, R)，其中 $s \in L$，$t \in R$。假设在 L 中，有 b 个男生，g 个女生，于是在 R 中即有 $n-b$ 个男生，$n-g$ 个女生。于是，s 与 R 中的 $n-b$ 个男生都相连结，L 中的 g 个女生都与 t 连结。此外，L 中的 b 个男生总共至少与 b 个女生相连，扣除在 L 中的 g 个女生，于是这 b 个男生至少与 R 中的 $b-g$ 个女生相连。将上面的连结边总和起来，可得这个割 (L, R) 之间的前向边数目至少为

$(n-b)+g+(b-g)=n$。既然这个割是任意的，这说明了任意割都至少为 n。由最大流最小割定理可知该模型的最大流即为 n，也就是说一定存在完美匹配。

Ex.7.31

a) Ford-Fulkerson 算法即是通过在余图中搜寻增量路径来求最大流（好像书中并没提到这个算法的名字）。如果第一次找到的增量路径为 $S \to A \to B \to T$，接下来再是 $S \to B \to A \to T$，这样每次都只找到一条流为 1 的增量路径，总共需要迭代 2000 次。

b) 类似于 Ex.4.13，在这里可以使用 Dijkstra 算法依赖于以下事实，如果一条路径 $s \to \cdots \to r \to t$ 的容量为 c，那么路径 $s \to \cdots \to r$ 的容量必定大于等于 c。现在来修改 Dijkstra 算法，设 $cp(v)$ 表从 s 到 v 的所有路径的最大容量值。首先将所有的 $cp(v)$ 初始化为 0，然后修改 $update$ 过程如下：

$$\begin{aligned}&\text{for all edges } (u,v)\in E:\\ &\quad \text{if } cp(v)<\min\left(cp(u),c(u,v)\right)\\ &\qquad cp(v)=\min\left(cp(u),c(u,v)\right)\\ &\qquad \cdots\end{aligned}$$

c) 考虑一个最小割，容易看出每条割边 e 都恰好对应着一条路径 p_e，使得 p_e 经过边 e，且有 p_e 上的流等于 $c(e)$。这些 p_e 汇总起来就成了最大流。因为割边的数目不会超过 $|E|$，于是可知，最大流可以由不超过 $|E|$ 条路径汇总而成。

d) 既然最大流 F 可以由不超过 $|E|$ 条路径汇总而成，那么“最肥”的那条路径的流一定不小于 $\frac{F}{|E|}$。令 c_t 为第 t 次迭代后的余图的最大流，于是有关系式：

$$c_{t+1} \le c_t - \frac{c_t}{|E|}$$

参考书中 5.4 节，可知迭代次数为 $O(|E| \bullet \log F)$。

8 NP-complete problems

Ex.8.1

若 S 为图 G 中所有边的长度之和，显然所求最短回路的长度不会超过 S 。倘若 TSP 能在多项式时间内解决，那么通过在区间 $[0, S]$ 内二分的进行 TSP 询问，从而也就能在多项式时间内解决 TSP-OPT。

Ex.8.2

如果图 G 中存在哈密顿回路（亦即是 Rudrata path），那么对于图 G 中的每一条边 e，询问 G 除去边 e 后是否还存在哈密顿回路。如果答案是“否”，说明 e 正好是回路中的一条边。如果答案是“是”，则将 e 中从图 G 中去掉。在上述过程完成之后，图 G 中剩余的边即是所求的回路。显然，若每次询问都是多项式时间的，那么整个过程可以在多项式时间内完成。

Ex.8.3

首先，易知 STINGY SAT 的解是可在多项式时间内验证的，因此属于 NP。另外，很容易可以将 SAT 归约到 STINGY SAT（将 k 设为所有变量的总个数即可），于是可知 STINGY SAT 为 NP 完全问题。

Ex.8.4

a) 显然 CLIQUE-3 的解是可以多项式时间内验证的，于是属于 NP。
b) 归约的方向反了，应该是 CLIQUE $\rightarrow$ CLIQUE-3，而不是 CLIQUE-3 $\rightarrow$ CLIQUE。
c) 注意求出 G 的团之后只能得到 $\overline{G}$ 的顶点覆盖，而非 G 的顶点覆盖。
d) 若图 G 中所有的顶点的度不大于 3，那么可知在图 G 的最大团中不会超过 4 个顶点。于是，直接枚举所有的顶点四元组即可求解，时间复杂度为 $O\left(|V|^4\right)$。

Ex.8.5

3D MATCHING $\rightarrow$ SAT：首先，对于任意两个互相冲突的三元组 t_i， t_j，最多只能选择其中一个，即 $\left(\overline{t_i} \vee \overline{t_j}\right)$。然后对于任意一个顶点 v，假设与该顶点相连的三元组为 t_v^1， t_v^2，…， t_v^n，那么至少要选择其中一个，即 $\left(t_v^1 \vee t_v^2 \vee \cdots \vee t_v^n\right)$。最后将这些

clauses 组合起来即构成一个 SAT 问题。这个转换过程显然是多项式的。

RUDRATA CYCLE $\rightarrow$ SAT：假设顶点的总数为 n，对于任意顶点 i，设它的邻接点为 p_i^1，p_i^2，…，p_i^m。首先.，顶点 i 必在哈密顿回路的某个位置上，即 $\left(x_{i1} \vee x_{i2} \vee \cdots x_{in}\right)$。然后，顶点 i 在哈密顿回路中不能出现多次，于是对于任意 $r \neq s$，有 $\left(\overline{x_{ir}} \vee \overline{x_{is}}\right)$。再次，在顶点 i 的邻接点中必然有某个顶点是它的后继。于是对于任意位置 j，设该位置的后继位置为 $k = j+1 \bmod n$，如果顶点 i 位于位置 j，那么它的邻接点中必有顶点位于位置 k，即 $\left(\overline{x_{ij}} \vee x_{p_i^1 k} \vee x_{p_i^2 k} \vee \cdots \vee x_{p_i^m k}\right)$。最后将这些 clauses 组合起来即构成一个 SAT 问题，这个转换过程当然也是多项式的。

Ex.8.6

a) 若每个 literal 最多出现一次，那么任意变量 v 出现的方式大致可以分为以下五种：
   1. 只在子句 c 中出现 v，此时可以令 v 为 $true$，同时将 c 去掉。
   2. 只在子句 c 中出现 $\overline{v}$，此时可以令 v 为 $false$，同时将 c 去掉。
   3. 只在子句 c 中出现 v 和 $\overline{v}$，此时 v 可以为任意值，同时仍将 c 去掉。
   4. 在两个子句中分别出现 v 和 $\overline{v}$，比如 $\left(v \vee a \vee b\right)\left(\overline{v} \vee c \vee d\right)$，此时可将 v 待定，并将这两个子句合并为 $\left(a \vee b \vee c \vee d\right)$。
   5. 在两个子句中出现且只出现 v 和 $\overline{v}$，即 $\left(v\right)\left(\overline{v}\right)$，此时无解。

   对每个变量都这样分情况处理，显然整个过程可在多项式时间内可以完成。

b) 回顾 3SAT 到 INDEPENDENT SET 的归约过程，如果每个 literal 最多出现两次，那么在构造的图 G 中，任何顶点的度都不会超过为 4。

Ex.8.7

按提示构造二分图 G。因为每个子句（clause）恰好包含三个文字（literal），所以任意 n 个子句总共包含了 $3n$ 个文字（可能有重复）。又因每个变量最多对应三个文字，所以这 $3n$ 个文字至少对应了 n 个变量。于是在图 G 中，任意 n 个子句都与不少于 n 个变量相连，由 Hall 定理知图 G 存在完美匹配。找出一个完美匹配，假设子句 c 与变量 v 配对，若 c 中包含了 v 的肯定，则将 v 设为 true，若 c 包含了 v 的否定，则将 v 设为 false，若 c 同时包含了 v 的肯定及否定，那么 v 可以任意取值。分别为每个配对进行上述赋值过程，即可找到一组可满足解。

Ex.8.8

首先很显然，EXACT 4SAT 属于 NP。现在通过将 3SAT 归约到 EXACT 4SAT 来证明后者的 NP 完全性。对于任意一个 3SAT 实例，如果其中某个子句中包含了同一个文字多次，那么可以缩减为一次，如果同时包含了某个变量的肯定及否定，那么可以将这个变量去掉。然后，可以再在每个子句中可以添加一些哑变量（即没用的辅助变量），这样就可以将每个子句所包含的文字数目扩充到四个。至此，即已将该 3SAT 实例转化成了一个 EXACT 4SAT 问题。

Ex.8.9

很容易将最小顶点覆盖归约到 HITTING SET。假设要求图 G 的最小顶点覆盖，可以建立一个 HITTING SET 实例，其中 S_1， S_2，…， S_n 即是图 G 的各条边，比如 $\{v_1, v_2\}$，$\{v_3, v_4\}$，…。通过二分式的询问（见 Ex.8.1），可以找到一个与 S_i 都相交的最小集合 H，这正好就是图 G 的最小顶点覆盖。

Ex.8.10

a) 令图 G 为一个环，环上的顶点数等于图 H 的顶点数。那么若 G 是 H 的同构子图，则说明 H 存在 Rudrata 回路。于是知 Rudrata 回路事实上是子图同构问题的一个特例。

b) 如果令 $g = |V| - 1$，即得到一条 Rudrata 路径。

c) 令 g 为子句的总数，即成 SAT。

d) 令 $b = \dfrac{a(a-1)}{2}$，此时这 a 个顶点两两相连，于是即成最大团问题。

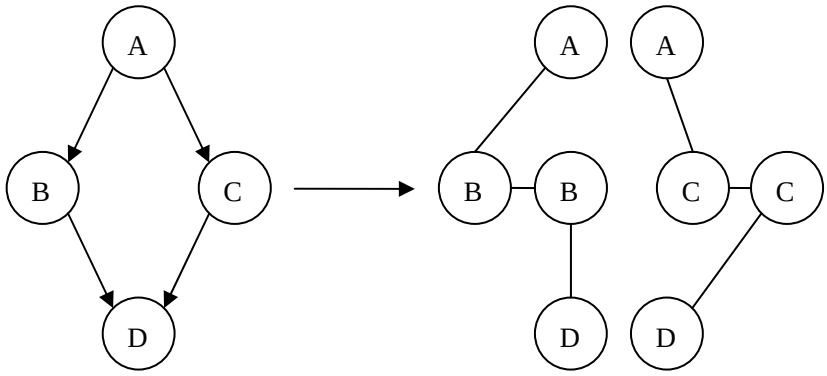
e) 令 $b = 0$，即成最大独立集问题。

f) 显然是最小顶点覆盖的一个推广。

g) Hint 中所描述的特例即是一个 TSP。

Ex.8.11

a) 假设有有向图 G，对于 G 中的每个顶点 v，将其分裂成 $i + o$ 新顶点，其中 i 是 v 的入度，o 是 v 的出度。然后将每个与 v 连结的有向边都转换成连向这 $i + o$ 个新顶点的无向边。为方便表示，在这些新顶点中，将与入边相连的顶点称为入顶点，与出边相连的顶点称为出顶点，这时还需要做的是，在每个入顶点与每个出顶点之间添加一条无向边。这样就将图 G 转化成了一个无向图 G'。现在只需在这个新的无向图 G' 上寻找 Rudrata 路径即可。一个示例如下图：



b) 在上一问的基础上，枚举所有顶点对(s,t)即可。

Ex.8.12

a) 显然k-SPANNING TREE 问题是可在多项式时间内验证的，因此是搜索问题。
b) 若$k=2$，此时的2-SPANNING TREE 实际上就是一条 Rudrata 路径。另外，这里好像有点问题，不知道我是否理解错了意思，因为当$k \geq |V|$时，显然只要一次 DFS 就能找出解。

Ex.8.13

a) 这个问题是属于 P 的，解法如下：选取任意一个$V-L$中的顶点s，以s为源点进行 DFS，每当访问到L中的顶点时，停止向下扩展，这样L中的顶点就成了叶节点。若 DFS 完成后，所有顶点都已被访问到，那么此时就找到了一棵满足要求的生成树。否则说明这样的生成树不存在。
b) 是 NP 完全的，因为恰好有一个叶节点的生成树即是一条 Rudrata 路径。
c) 是 NP 完全的，原因同 b)。
d) 是 NP 完全的，原因同 b)。
e) 是 NP 完全的，最多叶节点生成树的非叶子节点即构成最小连通支配集。最小支配集的 NP 完全性证明参考 Ex.8.20，而最小连通支配集的 NP 完全性证明与之类似，只需要在构造图G'时同时再将所有顶点（辅助顶点除外）两两相连即可。
f) 是 NP 完全的，原因同 b)。

Ex.8.14

可以将最大团问题归约到此问题。假设要求任意图$G(V,E)$中大小为k的团，可以在图G中添加k个相互独立的顶点，得到新图G'。这新加的k个顶点保证了图G'

存在大小为 k 的独立集，同时又不影响到原图的团。

Ex.8.15

可以将最大独立集问题归约到此问题。比如若要求任意图 $G(V,E)$ 中大小为 d 的独立集，可以令 $G_1 = G(V,E)$，再令 $G_2(V,\varnothing)$ 的顶点集与 G 相同，但是边集为空，也即是各个顶点相互独立。于是 G_1 与 G_2 存在着大小为 d 的公共子图，当且仅当图 G 存在着大小为 d 的独立集。

Ex.8.16

考虑怎么把一个 3SAT 实例转化成一个 EXPERIMENTAL CUISINE 问题：对于任意一个 3SAT 问题，若其某一子句 c 为 $(U \vee V \vee W)$，我们可以相应地建立 7 种 ingredients，分别为 $\{U\bar{V}\bar{W}_c, \bar{U}V\bar{W}_c, \bar{U}\bar{V}W_c, UV\bar{W}_c, \bar{U}VW_c, U\bar{V}W_c, UVW_c\}$。这 7 种 ingredients 代表了使得子句 c 成立的七种不同情况，比如 $UW\bar{V}_c$ 代表 U 和 W 同时为真，且 V 为假。它们之间当然是完全不兼容的，因此将其中两两之间的 discord 值设为 1。现在再考虑子句与子句之间的情况，对于任意两子句 i 和 j，将这两个子句中相矛盾的成分之间的 discord 值设为 1，比如 UAB_i 与 $\bar{U}CD_j$ 等。设子句总数为 n，再令 $p = 0$，若此时能选择的成份数为 n，那么即说明此 3SAT 能满足。另外，判断一个 3SAT 是否能满足与找出这个 3SAT 的解实际上是等价的（可以参考 Ex.8.2），因此若 EXPERIMENTAL CUISINE 在多项式时间内可解的话，3SAT 亦然。

Ex.8.17

由 NP 问题的等价性可知，任何一个输入规模为 n 的 NP 问题，可以在多项式时间内转化成为一个规模为 $p(n)$ 的 SAT 问题，而后者显然可以用穷举法在 $2^{p(n)}$ 时间内解决，于是得证。

Ex.8.18

显然素因子分解属于 NP，若 P = NP 便意味着素因子分解可以在多项式时间内完成。在 RSA 加密算法的应用中，若某个用户的公钥为 (N,e)，如果我们可以在多项式时

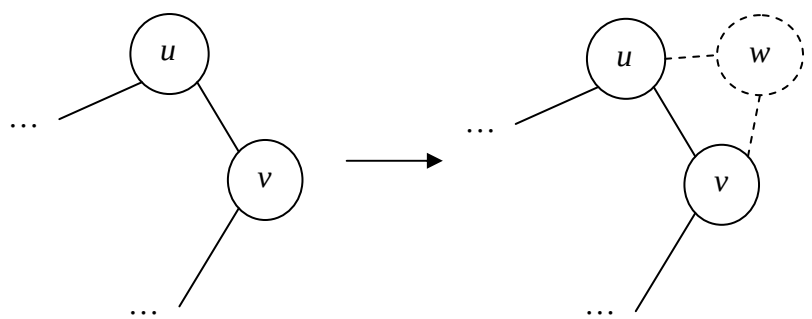
间内将 N 分解为 pq，那么很容易就能获得其密钥 $d = e^{-1} \bmod (p-1)(q-1)$。

Ex.8.19

可以将团问题归约到 KITE 问题。若要求图 $G(V,E)$ 的最大团，类似于 Ex.8.14，可以在图 G 中添加 $|V|$ 个新顶点，并将每个新顶点都连向原图中不同的某个顶点，共形成了 $|V|$ 条新边，这样就得了一个新图 G'。容易看出，在 G' 中存在大小为 $2g$ 的 kite 当且仅当 G 中存在大小为 g 的团。

Ex.8.20

可以将顶点覆盖问题归约到支配集问题。若要在图 $G(V,E)$ 中求得不大于 b 的一个顶点覆盖，可以先对图 G 做一个预处理：对每条边 $(u,v) \in E$，添加一个辅助顶点 w，及两条边 (u,w) 和 (v,w)，如下图所示：



对每条边都这样处理后得到一个新图 G'。容易看出，若原图 G 中存在不大于 b 的顶点覆盖，这个顶点覆盖也是新图 G' 的一个支配集。反过来，若新图 G' 中存在一个不大于 b 的支配集，那么对这个支配集进行一些处理后也能得到一个图 G 的不大于 b 的顶点覆盖。处理过程如下：设该支配集为 D，对于每条边 (u,v) 及相应的辅助顶点 w，若 $w \notin D$，则不用做任何处理，若 $w \in D$ 且 $u,v \notin D$，那么可以在 D 中将 w 替换成 u 或 v，若 $w \in D$ 同时 $u \in D \vee v \in D$，则直接将 w 从 D 中删掉即可。

Ex.8.21

a) 提示已经足够明显了，这里只需要注意到有向图上的 Rudrata 路径问题可以归约到无向图上的 Rudrata 路径问题（见 Ex.8.11）。

b) 对于每一个 k -string s，把它表示成一条有向边 (u,v)，其中 $u=s[1,k-1]$，$v=s[2,k]$。对所有的 k -string 依次这样处理，并合并相同的顶点，得到一个有向图 G。此时，还原原串 x 即相当于找出图 G 的欧拉路径。关于有向图的欧拉路径问题可以参考 Ex3.26。

Ex.8.22

a) 显然 FAS 是可在多项式时间内验证的，因此属于 NP。

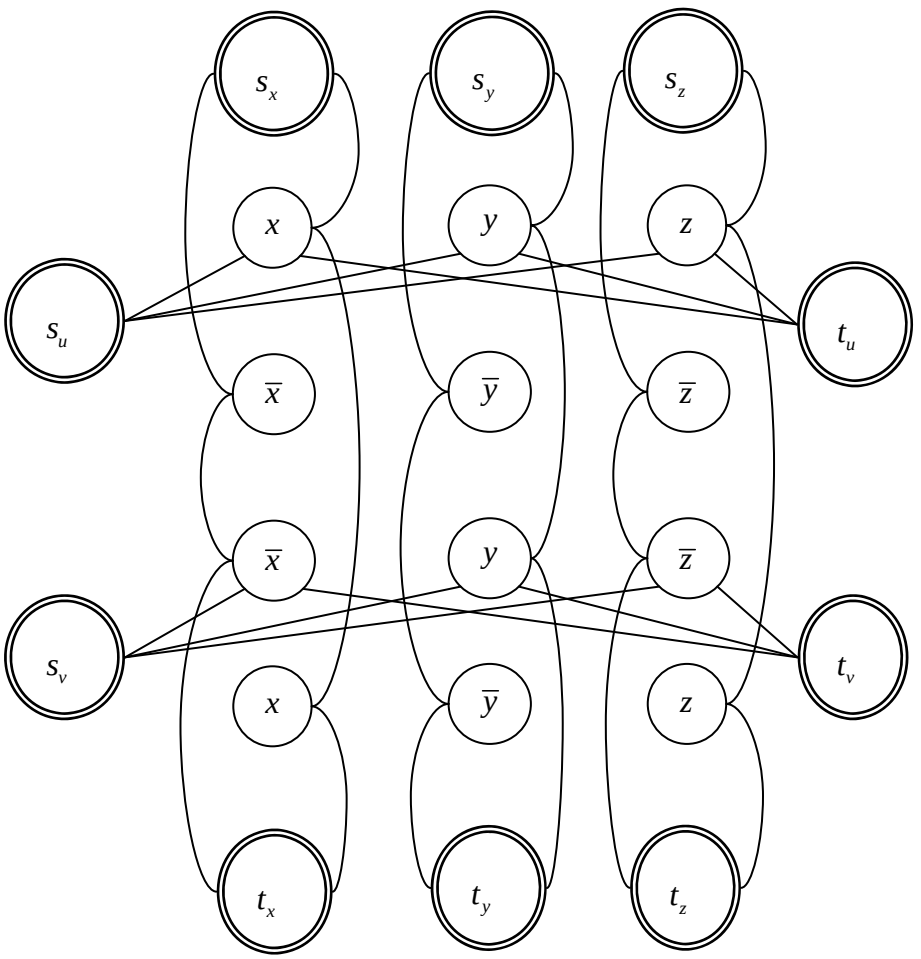
b) 设 G 的一个大小为 b 的顶点覆盖为 C，对于任意顶点 $v_i \in C$，设其在 G' 中相对应的顶点为 w_i 和 w_i'，则将边 (w_i, w_i') 添加到 E'。对 C 中的每个顶点都这样处理后，所得到的边集 E' 即是 G' 的一个大小为 b 的 feedback arc set。因为对于顶点 w_i 和 w_i'，当去掉边 (w_i, w_i') 后，所有与 w_i 相连的边都不可能位于任何一个环中，因为 w_i 不存在出边，同样，所有与 w_i' 相连的边也不可能位于任何一个环中，因为 w_i' 不存在入边。

c) 对于 G 中的任意一条边 (v_i, v_j)，设其在 G' 中相对应的顶点为 w_i、w_i'、w_j、w_j'，相对应的边为 (w_i, w_i')、(w_j, w_j')、(w_i', w_j)、(w_j', w_i)。若 E' 是 G' 的一个大小为 b 的 feedback arc set，显然，在这四条边中至少有一条边 e 属于 E'，否则就会形成环，而边 e 必然有个端点属于 $\{w_i, w_j\}$。若 w_i 是 e 的端点，则将 v_i 加入到 C，否则将 v_j 加入到 C。容易看出，在经过上述处理后，C 即是 G 的一个大小不超过 b 的顶点覆盖。

Ex.8.23

在提示的基础上再做以下处理：对任意子句 $c=(l_1 \vee l_2 \vee l_3)$，分别将 s_c 与 l_i、l_i 与 t_c

连结起来形成三条路径，因为任意 l_i 为真都使得 c 为真。另外，对于每个变量 v，将 s_v 与所有 v 串联起来，再连向 t_v，从而形成一条路径，再将 s_v 与所有 $\overline{v}$ 串联起来，连向 t_v，又形成一条路径。在这两条路径中，必然要选择其中一条，这保证了所有变量的一致性，即如果有任意子句选择了 v，则其余子句就不能再选择 $\overline{v}$ 。下面举一个简单的例子，假设要验证 CNF：$(x \vee y \vee z)(\overline{x} \vee y \vee \overline{z})$ 是否可以被满足，令 $u=(x \vee y \vee z)$，$v=(\overline{x} \vee y \vee \overline{z})$，可得 NODE-DISJOINT PATH 问题如下图：



9 Coping with NP-completeness

Ex.9.1

设子句个数不大于n，且可满足的 2SAT 问题可在$T(n)$时间内回溯得解。现在考虑某个可满足的包含$n+1$个子句的 2SAT 实例，设其形式为$(a \vee b)(\cdots)(\cdots)\cdots$，不失一般地，将其从变量$a$处展开，首先令$a = false$，那么下一步$b$一定为真，这时可以将子句$(a \vee b)$去除，余下的子句最多只有$n$个。若余下的这些子句可满足，于是这些剩余子句组成的子 2SAT 可以在$T(n)$时间内找到解，再与$a = false$、$b = true$合并也就得到了原 2SAT 的解。若剩下的这些子句不可满足，那么在这个分支的每一层展开中，必然会出现 singleton（即只含一个 literal 的子句）（Why？，因为如果剩余的子句都有两个 literal，那么这些子句一定是原 2SAT 的一个独立的部分。于是若这些剩余子句不可满足，则原 2SAT 也不可满足，矛盾），从而使得这个分支可以在线性时间内被终止。于是回溯后再进入$a = true$的分支，将子句$(a \vee b)$去除后也得到一个子句数不超过n的子 2SAT 实例，这个子实例一定是可满足的，因此也可在$T(n)$时间内找到解，与$a = true$合并后即得到原 2SAT 的解，此时的时间代价为在$a = false$分支上展开所用的线性时间再加上$a = true$分支上的$T(n)$时间，即相当于递归式$T(n+1) = O(n) + T(n)$，从该式可知$T(n)$为多项式时间，于是得证。

Ex.9.2

略。

Ex.9.3

再略。

Ex.9.4

任意选取顶点$v \in |V|$，将v加入集合S，并在$|V|$中将v的所有邻接点删掉（最多有

d 个邻接点）。然后重复上述过程，直到$|V|$为空为止。此时就得到了一个独立集S。观察上面的过程可知，在选择一个顶点的同时，最多删去了d个顶点，于是有：$|S| \geq \frac{|V|}{1+d} \geq \frac{1}{1+d}|MIS|$。另外，凭直觉，如果每次都选取度最小的顶点可能会有机会得到相对更优的解。

Ex.9.5

a) 对于任意边$e(u,v) \in T'$，若同时有$e \in T$，那么在这个边上就不需要交换了。

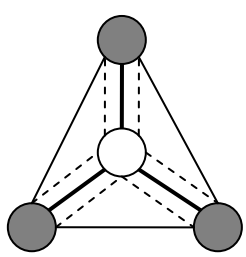
   若$e \notin T$，那么在T中一定存在其它路径$p(u,v)$，而且在路径$p(u,v)$中必然有一条边e'不属于T'（否则就形成环了）。于是在T上进行边交换(e,e')，这样就使得T增加了一条T'中的边，同时去除了一条不属于T'的边。显然在经过最多$|V|-1$次边交换后，T就能变换成T'。

b) 只需要注意到，树T是最小生成树，当且仅当在树T上不存在边交换(e,e')，使得$w(e) < w(e')$。这个性质根据 cut property 很容易证明。

c) 最多需要$|V|-1$次迭代。

Ex.9.6

假设T_s是图G的最小 Steiner 树。根据题示，将T_s按照 TSP 近似算法中的方式处理，即将每一条边使用两次，从而形成一个回路。现在我们从这个回路上的某一点出发，跳过不属于V'的顶点，最终得到的路线即是一个只包含V'中顶点的回路，如下图所示：



根据三角不等式可知，后面这个回路的代价不超过最小 Steiner 树代价的两倍。因此，

只包含V'的最小生成树的代价也不会超过最小 Steiner 树代价的两倍，于是得证。

Ex.9.7

a) 当$k=2$时，即是求s_1、s_2之间的最小割。这个问题显然可以用流模型在多项时间内解决。

b) 先找出s_1、s_2之间的最小割。将这个最小割中的边删掉后，s_1、s_2不可能同时再与s_3连通。若是s_1还与s_3连通，则继续找出s_1与s_3之间的最小割，若是s_2还与s_3连通，再找出s_2与s_3之间的最小割即可。因为多路分割肯定不小于任意两个顶点之间的最小割，所以上述算法的 ratio 不会超过 2。

c) 对于每个端点s_i，求得它与其余端点的最小割C_i。在所有的割C_i中，抛弃其中最大的那个割，将剩下的合并，即得一个初始的解A。枚举S中的任意两条边e_1、e_2，若在A中去除e_1、e_2后，此时所有端点s_i之间的最小割不大于 1，且包含了同一条边e_3，则在A中用e_3替换e_1、e_2，从而得到一个更优的解A'。然后继续下去还可以进行 3 换 2、4 换 3 等搜寻过程（不过这个算法能有效吗？怀疑）。

Ex.9.8

a) 若一个 SAT 实例能被满足，则通过最大 SAT 即能找到一个满足所有子句的解。

b) 对于任意子句j，其被满足的概率为 1 减去该子句中所有 literal 都为假的概率，亦即$\Pr(j)=1-\frac{1}{2^{k_j}}$。于是有：

$$E=\sum_{j=1}^{m}1\bullet\Pr(j)=\sum_{j=1}^{m}\left(1-\frac{1}{2^{k_j}}\right)\geq\frac{m}{2}$$

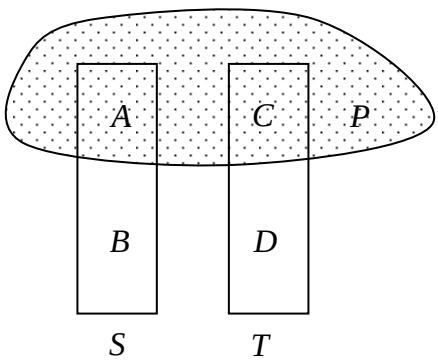
其中E为可满足子句个数的期望值。

c) 根据提示，在依次为每个变量的赋值时，总是选择能使得最多的当前尚未满足的子句得以满足的值。这样，在每一步选择时，总会使得一部分子句马上被满足，也会抛弃掉一部分不可能被满足的子句。当为所有变量都选择了值之后，所有被满足的子句加上所有被抛弃的子句等于总的子句数。由所采用的策略赋值知，每一步被满足的子句数都不小于被抛弃的子句数，于是可得最终被满足

的子句不小于$\frac{1}{2}$。

Ex.9.9

a) 若一个局部的最优分割为(S,T)，其中$T=V-S$。现在假设最大分割为(P,Q)，$Q=V-P$。令$A=S\cap P$，$B=S-A$，$C=T\cap P$，$D=T-C$，如下图：



以$w(A,B)$来表示集合A、B之间的边权和，因为(S,T)是局部最优的，所以对于S中的任意顶点v，有$w(\{v\},T)\geq w(\{v\},S-\{v\})$，从而可以推出$w(A,B)\leq w(A,T)$，$w(C,D)\leq w(S,C)$。于是最大分割满足：

$$\begin{aligned} w(P) &= w(A,B)+w(C,B)+w(A,D)+w(C,D) \\ &\leq w(A,T)+w(B,T)+w(S,D)+w(S,C) \\ &= 2\bullet w(S,T) \end{aligned}$$

b) 不是。

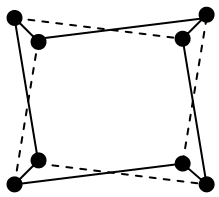
Ex.9.10

a) 找出一个反例即可，书上第 283 页已经给出了一个反例。

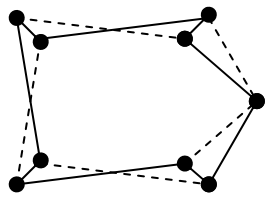
b) 这道题要找出一个构造性证明并不是很容易，我尝试了许多次后仍然没有找到一个通用的构造方法，使得一个 TSP 回路至少需要交换$\left\lceil \frac{n}{2} \right\rceil+1$条边才能得到更优解（如果有人找出来了千万要 email 我）。但是如果将条件放宽一点，改成至

少需要交换$\left\lceil \frac{n}{2} \right\rceil$条边，似乎就容易多了，下图示意了两种可能的构造方法：

a)

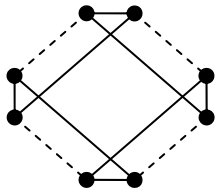


even

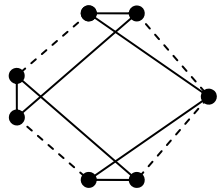


odd

b)



even



odd

在上图中，实线部分是某个初始解，虚线部分是得到一个更优解时需要增加的边。除了给出的实线边与虚线边之外，图中任意两顶点间不再存在其它的边（或者相距足够远）。可以想象，当为每条边设置好合理的长度值之后，要得到一个比实线回路更优的解，则必须用到所有的虚线边，也即至少需要交换$\left\lceil \frac{n}{2} \right\rceil$条边。

c) 若存在最优解 p 与当前解 c 有共同边，那么显然 $(n-1)$-change 的局部搜索可以找出最优解，于是只需考虑 p 与 c 不存在共同边的情况。下面需要引入一个定理：若一个图存在两条不相交 Hamilton 回路，则必然也存在第三条。这个定理并不是很容易证明，想了解证明过程的可以搜 Sloane 的一篇论文：“Hamiltonian cycles in a graph of degree 4”。设 p 的边集为 E_p、c 的边集为 E_c，由上面的定理可知，必然存在一条不同于 p 和 c 的回路 i，其中 i 各包含了 p 和 c 的一部分边，即有 $E_i \in E_p + E_c$。另外也可以想象，回路 i 的补，即 $E_p + E_c - E_i$ 也构成了一条 Hamilton 回路 j。显然，回路 i 和回路 j 都是 c 在 $(n-1)$-change 搜索过程中可达的。用 $\omega(\ )$ 来表示一条回路的代价，如果 $(n-1)$-change 局部搜索不能找出最优解，那么说明 c 比 i、j 更优，即 $\omega(E_i) \le \omega(E_c)$，$\omega(E_j) \le \omega(E_c)$，

将这两个不等式相加，并注意到 $E_i + E_j = E_c + E_p$，有 $\omega(E_p) \le \omega(E_c)$，这与 p 是最优解矛盾。于是知 $(n-1)$-change 的局部搜索确实是精确的。

d) 集合覆盖的 IMPROVEMENT 的 NP 完全性很容易看出来，因为最多不会超过 $O(n)$ 次 IMPROVEMENT 就一定能得到最优解。至于 TSP 的 IMPROVEMENT 的 NP 完全性，则需要注意到从 RUDRATA CYCLE 到 TSP 的归约过程，该过程说明了所有边长都为 1 或 2 的一类特殊 TSP 问题仍然是 NP 完全的，而这一类 TSP 问题需要的 IMPROVEMENT 次数显然也不会超过 $O(n)$。

e) 略。

10 Quantum algorithms

Ex.10.1

a) 这两个状态的 joint state 与 Bell state 比较，可得：

$$\alpha_0\beta_0 = \frac{1}{\sqrt{2}}$$
$$\alpha_0\beta_1 = 0$$
$$\alpha_1\beta_0 = 0$$
$$\alpha_1\beta_1 = \frac{1}{\sqrt{2}}$$

将上面第 1、4 式相乘得 $\alpha_0\alpha_1\beta_0\beta_1 = \frac{1}{2}$，而将第 2、3 式相乘得 $\alpha_0\alpha_1\beta_0\beta_1 = 0$，矛盾。于是知贝尔态不可能被分解。

b) 若结果为 0，此时状态变为 $|00\rangle$，若结果为 1，则状态变为 $|11\rangle$。

c) 若第一位测量的结果为 0，则为 0，若第一位测量的结果为 1，则为 1。

d) 传说中的超远距离瞬时传输。

Ex.10.2

a) $|00\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle \xrightarrow{CNOT} \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$

b) $|01\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle \xrightarrow{CNOT} \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$

c) $|10\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|10\rangle \xrightarrow{CNOT} \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle$

d) $|11\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|11\rangle \xrightarrow{CNOT} \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle$

Ex.10.3

输入向量的周期$k=1$，无偏移，代入 P303 页公式即可得$\left|\beta\right\rangle=\frac{1}{\sqrt{k}}\sum_{j=0}^{k-1}\left|\frac{jM}{k}\right\rangle=\left|0\right\rangle$。

Ex.10.4

此时可认为输入向量的周期$k=M$，偏移$l=j$。于是，同样代入 P303 页的公式，

可得$\left|\beta\right\rangle=\frac{1}{\sqrt{k}}\sum_{i=0}^{k-1}\omega^{liM/k}\left|\frac{iM}{k}\right\rangle=\frac{1}{\sqrt{M}}\sum_{i=0}^{M-1}\omega^{ji}\left|i\right\rangle$。

Ex.10.5

对照$\left|\beta\right\rangle$和$\left|\beta'\right\rangle$的第$i$个系数有：

$$\beta_i=\frac{1}{\sqrt{M}}\sum\nolimits_j\omega^{ij}a_j$$

$$\beta'_i=\frac{1}{\sqrt{M}}\sum\nolimits_j\omega^{i(j+l)}a_j$$

于是可得$\beta'_i=\beta_i\omega^{il}$。现在令$\left|a\right\rangle=\sum_{j=0}^{M/k-1}\sqrt{\frac{k}{M}}\left|jk\right\rangle$，根据 P303 页中第一个 claim 可得$\left|\beta\right\rangle=\frac{1}{\sqrt{k}}\sum_{j=0}^{k-1}\left|\frac{jM}{k}\right\rangle$。将$\left|a\right\rangle$偏移$l$之后，得到$\left|a'\right\rangle=\sum_{j=0}^{M/k-1}\sqrt{\frac{k}{M}}\left|jk+l\right\rangle$，进行 QFT 之后得到$\left|\beta'\right\rangle$。于是根据上面的关系$\beta'_i=\beta_i\omega^{il}$，并且注意到仅当$i=j\left(\frac{M}{k}\right)$时$\beta_i$不为 0，最后可得$\left|\beta'\right\rangle=\frac{1}{\sqrt{k}}\sum_{j=0}^{k-1}\omega^{l(jM/k)}\left|\frac{jM}{k}\right\rangle$。

Ex.10.6

在下面的推导中，以$\otimes$表示张量积，以$\left|+\right\rangle$表示$\frac{1}{\sqrt{2}}\left|0\right\rangle+\frac{1}{\sqrt{2}}\left|1\right\rangle$，以$\left|-\right\rangle$表示

$\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$。由 Hadamard gate 的定义及其自反性易知 $|0\rangle\xleftrightarrow{H}|+\rangle$，$|1\rangle\xleftrightarrow{H}|-\rangle$，于是本题就只需要证明下面的四个关系式成立即可：

$$|+\rangle\otimes|+\rangle\xrightarrow{CNOT}|+\rangle\otimes|+\rangle$$
$$|-\rangle\otimes|+\rangle\xrightarrow{CNOT}|-\rangle\otimes|+\rangle$$
$$|+\rangle\otimes|-\rangle\xrightarrow{CNOT}|-\rangle\otimes|-\rangle$$
$$|-\rangle\otimes|-\rangle\xrightarrow{CNOT}|+\rangle\otimes|-\rangle$$

证明上面这四个关系式可以直接进行计算验证，也可以根据量子状态的线性特征得到一个更简单的证明，首先来证明前两个关系式，由简单的门运算我们可以写出：

$$|0\rangle\otimes|+\rangle\xrightarrow{CNOT}\frac{1}{\sqrt{2}}|00\rangle+\frac{1}{\sqrt{2}}|01\rangle=|0\rangle\otimes|+\rangle$$
$$|1\rangle\otimes|+\rangle\xrightarrow{CNOT}\frac{1}{\sqrt{2}}|11\rangle+\frac{1}{\sqrt{2}}|10\rangle=\frac{1}{\sqrt{2}}|10\rangle+\frac{1}{\sqrt{2}}|11\rangle=|1\rangle\otimes|+\rangle$$

由上面两关系式可知，$|0\rangle$ 和 $|1\rangle$ 与 $|+\rangle$ 进行张量积，再通过 $CNOT$ 后保持不变，由于 $|+\rangle$ 和 $|-\rangle$ 都为 $|0\rangle$ 和 $|1\rangle$ 的线性组合，于是可知：

$$|+\rangle\otimes|+\rangle\xrightarrow{CNOT}|+\rangle\otimes|+\rangle$$
$$|-\rangle\otimes|+\rangle\xrightarrow{CNOT}|-\rangle\otimes|+\rangle$$

前两式得证，现在来证明后两式，同样由简单的门运算可以写出：

$$|0\rangle\otimes|-\rangle\xrightarrow{CNOT}\frac{1}{\sqrt{2}}|00\rangle-\frac{1}{\sqrt{2}}|01\rangle=|0\rangle\otimes|-\rangle$$
$$|1\rangle\otimes|-\rangle\xrightarrow{CNOT}\frac{1}{\sqrt{2}}|11\rangle-\frac{1}{\sqrt{2}}|10\rangle=-\left(\frac{1}{\sqrt{2}}|10\rangle-\frac{1}{\sqrt{2}}|11\rangle\right)=-|1\rangle\otimes|-\rangle$$

注意上面的第一式在经过 CNOT 门后保持不变，而第二式的右边反了号，于是同样根据线性组合关系可知：

$$|+\rangle\otimes|-\rangle\xrightarrow{CNOT}|-\rangle\otimes|-\rangle$$
$$|-\rangle\otimes|-\rangle\xrightarrow{CNOT}|+\rangle\otimes|-\rangle$$

Ex.10.7

a) 略。

b) 将 a 连接到 C-SWAP 门的第一位，b 连接到第二位，c 连接到第三位。其中 a、b 作为输入，c 为输出，初始时将 c 令为 0。容易看出，经过 C-SWAP 门后，当且仅当 a、b 都为 1 时，c 为 1，否则 c 为 0。

c) 这不就是一个 CNOT 门么？

d) 我们知道，仅用 NOT 门以及 AND 门就可以构造出任何逻辑电路（所谓的 Functional completeness），于是将传统电路的 NOT 门替换成量子 NOT 门，AND 门替换成量子 C-SWAP 门即得到量子电路 Q。

e) 因为 NOT 门和 C-SWAP 门是自反的，于是将 Q 的输入输出互换即得到 Q^{-1}。

f) 对于输入 $|x,0,0\rangle$，经过 Q 后得到 $|x,y,z\rangle$，再用 CNOT 门对 $|y\rangle$ 进行 fanout 操作，得到 $|x,y,z,y\rangle$，然后以上面结果中的 $|x,z,y\rangle$ 连接到 Q^{-1} 的输入（需要交换一下 $|z\rangle$ 和 $|y\rangle$ 的连接位置），经过 Q^{-1} 操作后，得 $|x,y,0,0\rangle$，即为所求。

Ex.10.8

a) 这里只需要证明 $x \bmod p$ 与 $(p-x) \bmod p$ 的 order 不可能同时奇。若 r 为 $x \bmod p$ 的 order，那么 r 一定能整除 $p-1$，也就是说 r 一定是 $\frac{p-1}{k}$ 的形式（Why？可以参考 Ex.1.38.b）。现在再来考虑 $(p-x) \bmod p$ 的 order，设为 s，同上可知 s 也一定为 $\frac{p-1}{k}$ 的形式。于是可以将满足 $x^t \equiv 1 \bmod p$，且 $t < p$ 的所有 t 表示为集合 $R = \{r, 2r, 3r, ... p-1\}$，同理可将满足 $(p-x)^t \equiv 1 \bmod p$，且 $t < p$ 的所有 t 表示为集合 $S = \{s, 2s, 3s, ... p-1\}$。由 $(p-x)^{2r} \equiv (-x)^{2r} \equiv 1 \bmod p$，可知 $2r \in S$。由 $x^{2s} \equiv (-x)^{2s} \equiv 1 \bmod p$ 可知 $2s \in r$。于是有关系 $2r = ms$、$2s = nr$，其中 m、n 为正整数。由此关系式得 $mn = 4$，从而 m、n 的取值只能是三种情况：$m=1$，$n=4$、$m=2$，$n=2$、$m=4$，$n=1$，分别对应了 $2r = s$、$r = s$、$r = 2s$ 这三种关系。若 r 为奇，

则 $r=s$ 意味着 $1\equiv -1 \bmod p$，这只有当 $p=2$ 时才成立。$r=2s$ 显然更不可取，于是只能有 $s=2r$，所以 s 必定为偶，于是得证。

b) 设 $x \bmod N$ 的 order 为 k，于是 $x^k \equiv 1 \bmod N$，推出 $\left(x \bmod p\right)^k \equiv 1 \bmod p$，$\left(x \bmod q\right)^k \equiv 1 \bmod q$。若 $x \bmod p$ 的 order 为 r，$x \bmod q$ 的 order 为 s，于是有 $k=lcm\left(r,s\right)$。仅当 r、s 都为奇时，k 才为奇。再根据中国剩余定理可以知道 $\left(x \bmod p\right)$ 与 $\left(x \bmod q\right)$ 是均匀分布的，于是由 a)中的结论，可知 k 为偶的概率不小于 $1-\frac{1}{2}\bullet\frac{1}{2}=\frac{3}{4}$。

c) 还是用 k 来表示 $x \bmod N$ 的 order（题目中是 r），假设有 $x^{k/2} \equiv -1 \bmod N$，可推出 $\left(x \bmod p\right)^{k/2} \equiv -1 \bmod p$，$\left(x \bmod q\right)^{k/2} \equiv -1 \bmod q$。若 r 为奇数，那么因为 k 为偶，因此一定有 $k=2mr$，其中 $m \in \mathbb{N}$。因为 $\left(x \bmod p\right)^r \equiv 1 \bmod p$，有 $\left(x \bmod p\right)^{k/2} \equiv \left(x \bmod p\right)^{mr} \equiv 1 \bmod p$，与前面的 $\left(x \bmod p\right)^{k/2} \equiv -1 \bmod p$ 矛盾，从而知道 r 不能为奇数。同理可知 s 也不能为奇数。又因 k 为偶，所以 r、s 中必然有一个是偶数。于是可知，$x^{k/2} \equiv -1 \bmod N$ 不大于 r、s 同时为偶的概率除以 r、s 中有一个为偶的概率，即为 $\frac{1/4}{3/4}=\frac{1}{3}$。