算法设计与分析

ALGORITHMS

王子航 5200309 10155

INTRODUCTION

▷ Hilbert 10th - Problem

Is there an algorithm s.t. for any potential equation, within finite calculations, we can judge that the equation has an integer solution or not.

▷ Hilbert's Entscheidungsproblem 判定性问题

Is there an algorithm s.t. it can prove any thm. ?

Define "Algorithm".

▷ 1936. Church & Turing ⟶ Lambda Calculation & Turing Machine (respectively).

▷ Computation Problem. definition?

$$f: \{0,1\}^* \longrightarrow \{0,1\}^*.$$

有限长0,1到 到 有限长0,1到的映射.

$\{0,1\}^* := \bigcup_{n\geq 0} \{0,1\}^n$.

▷ Turing Machine

k-belt Turing Machine : a three-element tuple (Γ, Q, δ). (字符集, 状态)

- the 1st one : input ("test paper")
- others : 草稿纸 ("draft")

δ: "$Q\times\Gamma^k \to Q\times\Gamma^{k-1}\times\{L,S,R\}^k$" 状态转移

▷ Church-Turing Thesis : Any "effective computation" is equivalent to Turing Machine, i.e.

Computation Problem $f: \{0,1\}^* \to \{0,1\}^*$ is computable.

$\Leftrightarrow$ $\exists$ a Turing Machine T, s.t. $T(x) = f(x)$.

$\Leftrightarrow$ $\exists$ a C++ Program P, s.t. $P(x) = f(x)$.

Thm. Not all problems are computable.

Proof. $\{0,1\}^* \cong \mathbb{N}$. $\{f \mid f: \{0,1\}^* \to \{0,1\}^*\} \cong \mathbb{N}^{\mathbb{N}} \succ 2^{\mathbb{N}} \cong \mathbb{R} \succ \mathbb{N}$.

↑ A Turing Machine can be represented by 0,1-seq.

Qed. □

(cont'd)

An example for incomputable problems: HALTING PROBLEM

▷ Halting Problem

$$x \in \mathbb{N} \mapsto P_x \in \{\text{C++ Programmes}\}$$

Consider function: $Halt(x,y) = \begin{cases} 1 & y\text{作为}P_x\text{输入时会死循环} \\ 0 & y \cdots P_x \cdots \text{不会死循环} \end{cases}$

Proof. Assume computable. exists $A(x,y) = Halt(x,y)$.

Consider program B:

```
int main( int x)
{
  if A(x,x)==1 then return 0;      // 对角线方法. 对角线上取反
  else for(;;);                    // loop forever
}
```

	1	2	3	4	⋯	← P_x: 所有程序. 可列
1	B(1)					
2		B(2)				← $a_{ij} = Halt(i,j)$.
3			B(3)			
⋮				B(4)		
inputs					⋱	

$\forall k.\quad A(k,k)=1 \rightarrow B(k)=0 \neq P_k(k)$

$=0 \rightarrow$ forever loop $\neq P_k(k)$.

$\forall k.\ B \neq P_k$.

exist no such program B. Contradiction.

Thus, HALTING PROB is not computable. Qed. □

> Back to H. Ep. Does not exist.

> Back to H-to Prob. Can be proved not computable.

- O-representation.

$f, g: \mathbb{N} \to \mathbb{N}$. $f = O(g)$. $g = \Omega(f)$ iff. $\exists c \ \forall n \in \mathbb{N}. \ f(n) \leq cg(n)$

$f = o(g)$ $g = \omega(f)$ iff. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

f比g低阶

$f = O(g) \wedge g = O(f) \Rightarrow f = \Theta(g)$. 同阶.

常见: $\log n$, n, $n\log n$, $n\sqrt{n}$, n^2, $\cdots$, 2^n, 2^{2^n}, $\cdots$

$O(\sqrt{n})$ 一般针对 0,1-到的长度

e.g. Fibonacci

利用 $f(n) = f(n-1) + f(n-2)$

- input: n (a $\log n$-bit sequence)
- 考虑多位加法用时.
- 时间复杂度 $O(n^2)$. 实际对 input 是指数级别 $O((e^{bits})^2)$

$F(n)$的位数约为 $n\log(\frac{\sqrt{5}+1}{2}) \approx n\log 1.6 \approx n$.

利用通项?

$$\begin{pmatrix} f(n) \\ f(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f(n-1) \\ f(n-2) \end{pmatrix} = \cdots = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} f(1) \\ f(0) \end{pmatrix}.$$

快速幂. 矩阵乘幂 $\log n$ 次完成上述计算.

n位 × n位用时 $O(n^2)$. 最坏情况: $O(n^2 \log n)$

实际: 每次乘法 $T(i) = (2^i)^2$.

总复杂度 $O((2^1)^2) + O((2^2)^2) + \cdots + O((2^k)^2) = O(n^2)$.

DIVIDE AND CONQUER

- Intro: Merge Sort.

$$T(n) = \begin{cases} 1, & n=1 \\ 2T(\frac{n}{2}) + O(n). & n>1. \end{cases}$$

$\dots\rightarrow$ $T(n) \le 2T(\frac{n}{2}) + cn$

$\le 2\left(2T(\frac{n}{4}) + c\cdot\frac{n}{2}\right) + cn$

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

$= 8T(\frac{n}{8}) + 3cn = \dots$

$$T(n) = 2^k T(\frac{n}{2^k}) + kcn = n\,T(1) + cn\log n = O(n\log n)$$

Minimal Time Complexity of Sorts with comparison and Switch Operation only?

▷ $O(n\log n)$

Proof. $A(a[1,\dots n]) \longrightarrow b[1,\dots,n]$. Time complexity: $T(n)$

$n!$ inputs (for fixed sorted b)

For each ~~sorted~~ step, " $>$ " " $=$ " " $<$ ". 3 Cases.

i.e. for an input (fixed; unknown pattern) $\xrightarrow{1\text{ step}}$ 3 possible patterns.

$T(n)$ steps $\longrightarrow$ $3^{T(n)}$ $\ge n!$ $\longrightarrow$ $T(n) \ge \log_3 n!$

#(possible inputs) when algo. done #(Possible inputs) $= \Theta(n\log n)$.

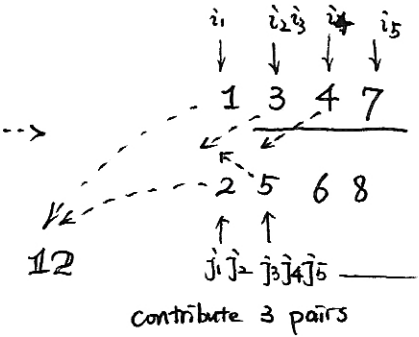
[$n! \le n^n$. ; $\log n! \le n\log n$; $n! \ge (\frac{n}{2})^{\frac{n}{2}}$; $\log n! \ge \frac{n}{2}\log\frac{n}{2} = \Theta(n\log n)$.]

$n(n-1)(n-2)\cdots\cdots 3\cdot 2\cdot 1$

后半个 $> \frac{n}{2}$

- Inverse Pairs (Inversions)

4 3 7 1 | 2 8 6 5 --->



j++ 时:

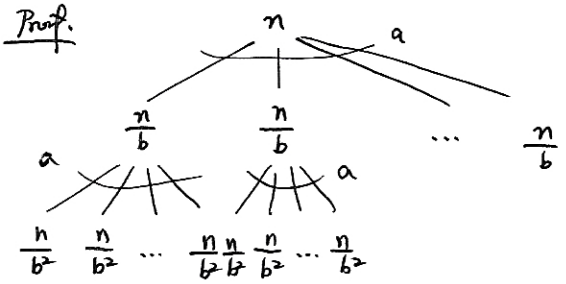
Sum += (mid - i + 1)

- Master Theorem

Thm. $T(n) = aT(n/b) + O(n^d)$,

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a. \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Prof.



Level 0 $\sim n^d \cdot c$

Level 1 $\sim a \cdot (\frac{n}{b})^d \cdot c$

Level 2 $\sim a^2 (\frac{n}{b^2})^d \cdot c$

⋮

const / const

Level $\log_b n =: k$ $\sim a^k (\frac{n}{b^k})^d \cdot c$

$$\therefore T(n) = \sum_{k=0}^{\log_b n} c \cdot a^k \cdot \left(\frac{n}{b^k}\right)^d = c \cdot n^d \cdot \sum_{n=0}^{\log_b n} \left(\frac{a}{b^d}\right)^k.$$

$$= \begin{cases} O(n^d) & \text{if } \frac{a}{b^d} < 1 \\ O(n^d \log_b n) & \text{if } \frac{a}{b^d} = 1 \\ O(n^{\log_b a}) & \text{if } \frac{a}{b^d} > 1 \end{cases}$$

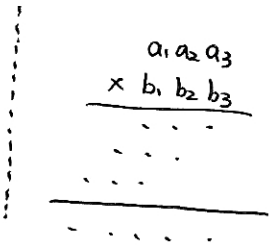
Qued. □

$$T(n) = c \cdot n^d \cdot \left(\frac{a}{b^d}\right)^{\log_b n}$$
$$= c \cdot n^d \left(\frac{a^{\log_b n}}{b^{d \log_b n}}\right)$$
$$= c \cdot a^{\log_b n} = \Theta(n^{\log_b a})$$

- Multiplication

$a = a[1 \cdots n]$ 存3一个整数 $a_i \in \{0,1\}$

$b = b[1 \cdots n]$ $b_i \in \{0,1\}$



$O(n^2)$

Divide and Conquer?

> $a = a_L \cdot 2^{n/2} + a_R$

> $b = b_L \cdot 2^{n/2} + b_R$

$$\Rightarrow a \times b = a_L b_L \cdot 2^n + (a_L b_R + a_R b_L) \cdot 2^{n/2} + a_R b_R$$

↑ 后补0即可.

加法. $O(n)$

$$T(n) = 4 \cdot T(n/2) + O(n)$$

By Master Thm., $T(n) = O(n^{\log_2 4}) = O(n^2)$.

□

$O(n)$ part is hard to optimize. "4" is too many. ⟶ What to do?

Gauss: We only need "$a_L b_L$", "$a_R b_R$" and "$a_L b_R + a_R b_L$"

$a_L b_R + a_R b_L = (a_L + a_R)(b_L + b_R) - a_L b_L - a_R b_R$

$(a_L + a_R)(b_L + b_R)$

$\Rightarrow T(n) = 3T(n/2) + O(n) = O(n^{\log_2 3}) \approx O(n^{1.59})$. □

Better!

- Matrix Multiplication

$A = (a_{ij})_{i,j \in [n]}$ $B = (b_{ij})_{i,j \in [n]}$ $C_{ij} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$ $O(n^3)$

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \quad X \cdot Y = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$T(n) = 8T(n/2) + O(n^2) \longrightarrow O(n^3)$

Strassen's Algorithm:

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

$P_1 := A(F-H)$ $P_2 := (A+B)H$

$P_3 := (C+D)E$ $P_4 := D(G-E)$

$P_5 := (A+D)(E+H)$. $P_6 := (B-D)(G+H)$

$P_7 := (A-C)(E+F)$

$T(n) = 7T(n/2) + O(n^2) \longrightarrow O(n^{2.81})$.

- k^{th}-Largest Problem

$a[1, \cdots n]$ $k \in [n]$ Output the k^{th} largest number in a.

▷ Sort. $O(n \log n)$

▷ Divide and Conquer.

Divide — Similar to Quick Sort. $a[$ (pivot) $\ldots]$

↓ $O(n)$

$a[\ \ ]$ pivot $[\ \ ]$

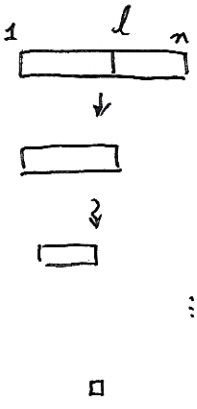
$<$ pivot $>$ pivot

→ We know pivot is ℓ-largest number.

$\begin{cases} \ell > k \rightarrow \text{LEFT}. & k\text{-largest} \\ \ell < k \rightarrow \text{RIGHT}. & (k-\ell)\text{-largest}. \\ \ell = k \rightarrow \text{BINGO}. \end{cases}$

$T(n) \leq T(\max\{\ell-1, n-\ell\}) + O(n)$ → Worst: $O(n^2)$

ℓ is a uniform random number in $\{1, 2, \cdots, n\}$. $n, n-1, \cdots, 1$



$X_1 = n$

下次 $X_2 =$ a random variable

X_3

$\vdots$

X_n

$$T(n) = T(X_1) = T(X_2) + cX_1$$
$$= T(X_3) + cX_2 + cX_1$$
$$= O\left(\sum_{i=1}^{n} X_i\right)$$

$$\mathbb{E}T(n) = c\mathbb{E}\left(\sum_{i=1}^{n} X_i\right) = c\sum_{i=1}^{n} \mathbb{E}X_i$$

CLAIM $\forall i.\ \mathbb{E}[X_{i+1} \mid X_i] \leq \frac{3}{4} X_i$

[Proof.]

$$\mathbb{E}[X_{i+1} \mid X_i = a] = \mathbb{E}[\max\{a-\ell, \ell-1\}]$$
$$= \mathbb{E}[\max\{a-\ell, \ell-1\} \mid a-\ell > \ell-1] \cdot \Pr[a-\ell > \ell-1]$$
$$+ \mathbb{E}[\max\{a-\ell, \ell-1\} \mid a-\ell \leq \ell-1] \Pr[a-\ell \leq \ell-1]$$
$$= \mathbb{E}\left[a-\ell \mid \ell < \tfrac{a+1}{2}\right] \Pr\left[\ell < \tfrac{a+1}{2}\right] + \mathbb{E}\left[\ell-1 \mid \ell \geq \tfrac{a+1}{2}\right] \Pr\left[\ell \geq \tfrac{a+1}{2}\right]$$
$$\leq \frac{3}{4}a \cdot \left(\Pr[a-\ell > \ell-1] + \Pr[a-\ell \leq \ell-1]\right)$$
$$\leq \frac{3}{4}a.$$

$\mathbb{E}[X_{i+1} \mid X_i] \leq \frac{3}{4} X_i.$

Qed. □

$\Rightarrow \mathbb{E}[\mathbb{E}[X_{i+1} \mid X_i]] \leq \frac{3}{4}\mathbb{E}X_i \iff \mathbb{E}X_{i+1} \leq \frac{3}{4}\mathbb{E}X_i \Rightarrow \mathbb{E}X_{i+1} \leq (\frac{3}{4})^i \cdot n.$

$\therefore \mathbb{E}T(n) \leq c \cdot \sum_{i=1}^{n} \mathbb{E}[X_i] \leq \underbrace{c \cdot \sum_{i=1}^{n} (\frac{3}{4})^{i-1}}_{\text{const}} \cdot n = O(n)$ □

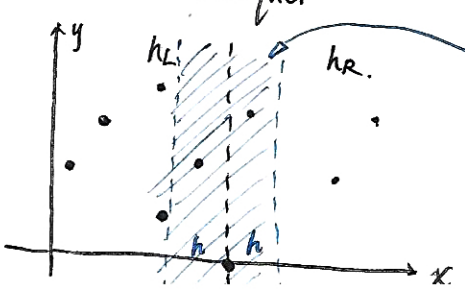
- Point Pairs with Minimal Euclidean Distance

$\mathbb{R}^2$. n points. $S = \{(x_i, y_i) \mid i = 1, 2, \cdots, n\}$ $d((x_1,y_1),(x_2,y_2)) = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}$

Minimal Distance = ?

▷ 枚举. $O(n^2)$

▶ Divide and Conquer



$h := \min\{h_L, h_R\}$

What if all is within $[-h, h]$ range?

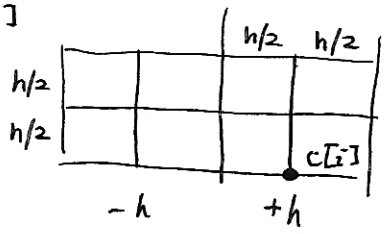
(See Next Page).

CLAIM For any $C_i = 1, \cdots, m$. within $[-h, +h]$ range.

We just need to check $j = i+1, \cdots, i+7$. If $d(c[i], c[j]) < h$. Update.

(Preprocess: Sort by y) Otherwise. —— h.

[Proof.]



← 如果真有距离小于 h 的. 一定落在这八个格子中.

且不可能与 $C[i]$ 同格. (否则 $h \to < h/2$)

FAST FOURIER TRANSFORM

- Polynomial Multiplication

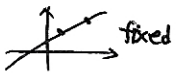
$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$$
$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_d x^d \quad \sim \text{Coefficient Representation}$$

$$C(x) = A(x)B(x) = c_0 + c_1 x + \cdots + c_{2d} x^{2d}$$

Can we calculate it even faster? ↓ e.g. $\tilde{O}(d \log d)$?

$$\Rightarrow c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0 = \sum_{i=0}^{k} a_i b_{k-i}. \quad \sim O(d^2).$$

Prop. $f(x)$ is a polynomial of degree $\le d$. Given $(x_0, f(x_0)), (x_1, f(x_1)), \cdots (x_d, f(x_d))$ then $f(x)$ is fixed.

In 1D case: 两点确定一条直线 $f(x) = ax + b$ 

A natural idea: if we can get $2d+1$ points $(x_i, C(x_i))$. then $C(x)$ is fixed.

i.e. INPUT: $(x_1, A(x_1)), (x_2, A(x_2)), \cdots (x_{2d+1}, A(x_{2d+1}))$. Value Representation
$(x_1, B(x_1)), (x_2, B(x_2)), \cdots (x_{2d+1}, B(x_{2d+1}))$.

$\Rightarrow (x_1, A(x_1)B(x_1)), (x_2, A(x_2)B(x_2)), \cdots (x_{2d+1}, A(x_{2d+1})B(x_{2d+1}))$. $\sim O(d)!$

Here multiplication of two constants are viewed as taking constant time. (in fact, $O(bit^{1.57})$) Ignored. Const.

* we can choose bit = 1. (Larger Number: $x = 2$)

We have the following relationship.

① Coefficient Representation ⇄ (Evaluation / Interpolation) ⇄ Value Representation

▶ Evaluation.

Rewrite the equations above as: $A(x) = a_0 + a_1 x_0 + \cdots + a_{n-1} x^{n-1}$.

If we pick $x_1, x_2, \cdots x_n \cdots$ randomly? $\rightarrow O(n^2)$. ×

How to select a better $(x_1, x_2, \cdots x_n)$?

$n = 2^m$.

We select: $1, 2, 3, \cdots, \frac{n}{2}$
$-1, -2, -3, \cdots, -\frac{n}{2}$ → $\frac{n}{2}$ 次求得 3 所有的 x^2. x^3, …

$x^2 = (-x)^2$ ⇓ $(-x)^3 = -x^3$ ↘

总共用 $\frac{n}{2}$ 次求得 3!

② Divide and Conquer!

$$A(x) = A_0 + A_1 x + A_2 x^2 + \cdots + A_{n-1} x^{n-1}$$
$$= \underbrace{(A_0 + A_2 x^2 + A_4 x^4 + \cdots + A_n x^{n-2})}_{A_{even}(x^2)} + x \underbrace{(A_1 + A_3 x^2 + A_5 x^4 + \cdots + A_{n-1} x^{n-2})}_{A_{odd}(x^2)}$$

$A_{even}(u) = A_0 + A_2 u + A_4 u^2 + \cdots + A_n u^{n/2-1}$; degree $\le n/2$

$A_{odd}(u) = A_1 + A_3 u + A_5 u^2 + \cdots + A_{n-1} u^{\frac{n-2}{2}}$ degree $\le n/2$

We have $A(a) = A_{even}(a^2) + a\,A_{odd}(a^2)$

$A(-a) = A_{even}(a^2) - a\,A_{odd}(a^2)$

Then we apply the same process to $A_{even}(\cdot)$ and $A_{odd}(\cdot)$. ← Divide and Conquer.

We pick $x_1 \quad -x_1 \quad x_3 \quad -x_3 \quad \cdots \quad x_{n-1} \quad -x_{n-1}$

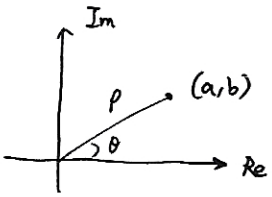
Only need to calculate $A_{even}/A_{odd}(x_1^2) \quad A_{even}/A_{odd}(x_3^2) \quad \cdots \quad A_{even}/A_{odd}(x_{n-1}^2)$ $\Rightarrow O(n)$ get $A(x_i)$.

$$\boxed{T(n) = 2T(n/2) + O(n)} \Rightarrow T(n) = O(n \log n)$$

■ Note that in the 2nd step, we require: $x_1^2,\ x_2^2 = -x_1^2,\ x_3^2,\ x_4^2 = -x_3^2 \ldots\ldots$

③ complex number!

▶ Review of Complex Numbers



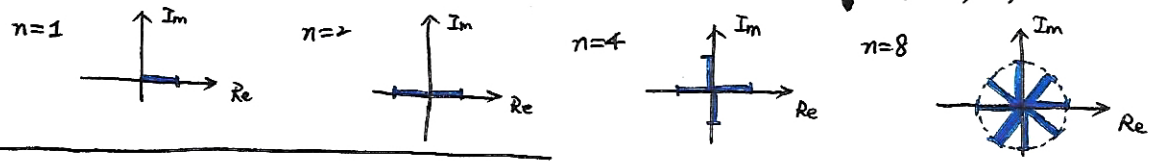
$z = a + bi \in \mathbb{C}$ ~ easy to add

$z = \rho e^{j\theta} \in \mathbb{C}$ (where $\tan\theta = \frac{b}{a}$. $\rho = \sqrt{a^2+b^2}$) ~ easy to multiply

$z^n = 1 \Rightarrow z = ?$ (n次单位根)

We write z as $\rho \cdot e^{j\theta}$. $|z| = \rho = 1$. $\Rightarrow z = e^{j\theta}$. $z^n = e^{nj\theta} = 1$

$\therefore nj\theta = 2k\pi \ (k \in \mathbb{Z}^+) \Rightarrow z = e^{\frac{2k\pi}{n}j}$ $(k = 1, 2, \cdots, n)$



(cont'd)

degree = 0. = 1−1

degree = 1 = 2−1

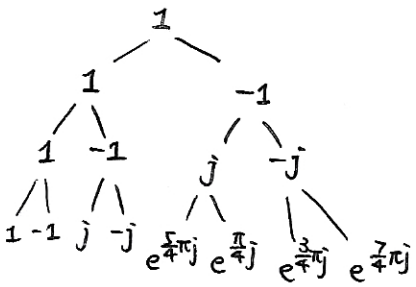
degree = 3 = 4−1

degree = 7 = 8−1

⋮

degree = $n = 2^w - 1$

divide and conquer



$z^1 = 1$ ~ $\omega = 1$

$z^2 = 1$ ~ $\omega = -1$. $\{\omega^0, \omega^1\}$

$z^4 = 1$ ~ $\omega = j$ $\{\omega^0, \omega^1, \omega^2, \omega^3\}$

$z^8 = 1$. ~ $\omega = e^{\frac{\pi}{4}j}$ $\{\omega^0, \omega^1, \cdots \omega^7\}$

$z^n = 1$ ~ $\omega = e^{\frac{2\pi}{n}j}$ $\{\omega^0, \cdots \omega^n\}$

Thus, we select $\left\{ e^{j\frac{2k\pi}{n}} \right\}$ $(k = 1, 2, \cdots, n)$ as evaluation number.

```
Function FFT(A, ω): // requirement: deg(A) ≤ n-1,   ω = e^{(2π/n)j}
                    // Expected Output: A(ω^0), A(ω^1), ... A(ω^{n-1})
  If ω = 1 then
    return A(1);
  Compute A_even, A_odd s.t. A(x) = A_even(x^2) + x A_odd(x^2).
  FFT(A_even, ω^2);
  FFT(A_odd, ω^2);
  For i = 0 → n-1 do
    A(ω^i) = A_even(ω^{2i}) + ω^i A_odd(ω^{2i});
  endfor
  return A(ω^0), A(ω^1), ... A(ω^{n-1}).
```

▶ Interpolation

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

we already have

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2n-2} & \cdots & \omega^{(n-1)^2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$m_{jk} = \omega^{jk}$ ~ $M_n(\omega)$ (Van der Monde Matrix) Vandermonde Matrix

$j = 0, 1, \cdots, n-1$; $k = 0, 1, \cdots, n-1$

$$|M_n(\omega)| = \prod_{i,j} (\omega_i - \omega_j)$$

compute the inverse of V.M.? ~ $O(n^2)$. Too slow.

But $M_n(\omega)$ is not just some "normal" V.M.!

CLAIM. $M_n(\omega)$ is orthogonal.

[Proof] $u, v \in \mathbb{C}^n$, $\langle u, v \rangle = u^* v$. 共轭转置

Let $M_n(\omega) = [v_0, v_1, \cdots v_{n-1}]$.

$$\langle v_j, v_k \rangle = \sum_{l=0}^{n-1} \omega^{jl} \omega^{-kl} = \begin{cases} n & j = k \\ \sum_{l=0}^{n-1} (\omega^{j-k})^l = \frac{1-\omega^{(j-k)n}}{1-\omega^{j-k}} = 0 & j \neq k \end{cases}$$

In fact:

FFT

INPUT: $\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$

OUTPUT: $\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix}$

$O(n \log n)$ complete the Matrix Multiplication

What FFT really does:

<coefficient> $\xrightarrow{FFT(a,\omega)}$ <value>

$a = (a_0, \cdots a_{n-1})^T$? $M_n(\omega) \cdot a$

$[M_n(\omega)]^{-1} \cdot [M_n(\omega) \cdot a]$

Thus, $M_n^*(\omega) \cdot M_n(\omega) = nI = n\,\mathrm{diag}(1, 1, \cdots 1)$.

$M_n(\omega)^* M_n(\omega)(j,k) = v_j^* v_k$

$\Rightarrow [M_n(\omega)]^{-1} = \frac{1}{n} M_n^*(\omega)$ □

Let $M_n'(\omega) = (m'_{jk})_{n\times n}$. $\quad m'_{jk} = \frac{1}{n}\overline{(\omega^{kj})}^{\bullet} = \frac{1}{n}\overline{(\omega^{jk})}^{\bullet} = \frac{1}{n}\omega^{-jk}$ (共軛)

$:= [M_n(\omega)]^{-1}$

$\therefore \boxed{[M_n(\omega)]^{-1} = \frac{1}{n} M_n(\omega^{-1})}$

→ We computed the inverse. ✓

Still, computing ω^{-jk} seems time-consuming.

↓

Use FFT!

$\sim O(n \log n)$

<coefficient> $\xrightarrow{FFT(a,\omega)}$ <value>

$\xleftarrow{FFT(v,\omega^{-1})}$

$a = (a_0, \cdots a_{n-1})^T$

$= [M_n(\omega)]^{-1} \cdot v$

$= \frac{1}{n} M_n(\omega^{-1}) \cdot v$

$v = M_n(\omega) \cdot a$

$= (v_0, \cdots v_{n-1})^T$

□

* Look back at FFT again in the Perspective of Matrix Multiplication

$$\begin{bmatrix} A(x_0) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{matrix} \\ j \end{matrix}\overset{k}{\begin{bmatrix} & \\ & \omega^{jk} \\ & \end{bmatrix}} \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix} \xrightarrow{G} \begin{matrix} \\ j \\ \\ j+\frac{n}{2} \end{matrix}\overset{2k \quad 2k+1}{\left[\begin{array}{cc} \omega^{2jk} & \omega^{j}\cdot\omega^{2jk} \\ \hline \omega^{2jk} & -\omega^{j}\cdot\omega^{2jk} \end{array}\right]} \begin{bmatrix} a_0 \\ a_2 \\ \vdots \\ a_{n-2} \\ \hline a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$$= \begin{bmatrix} [M_{n/2}]\,a_e + \mathrm{diag}\{\omega^j\}[M_{n/2}]\,a_o \\ \hline [M_{n/2}]\,a_e - \mathrm{diag}\{\omega^j\}[M_{n/2}]\,a_o \end{bmatrix}$$

Function FFT(a, ω): // input $a = (a_0, \cdots a_{n-1})^T$. $\omega = e^{\frac{2j\pi}{n}}$

if $\omega = 1$ then return a;

$(S_0, S_1, \cdots S_{n/2-1}) =$ FFT$((a_0, a_2, \cdots a_{n-2}), \omega^2)$

$(S_0', S_1', \cdots S_{n/2-1}') =$ FFT$((a_1, a_3, \cdots a_{n-1}), \omega^2)$

for $i = 0 \to \frac{n}{2}-1$ do

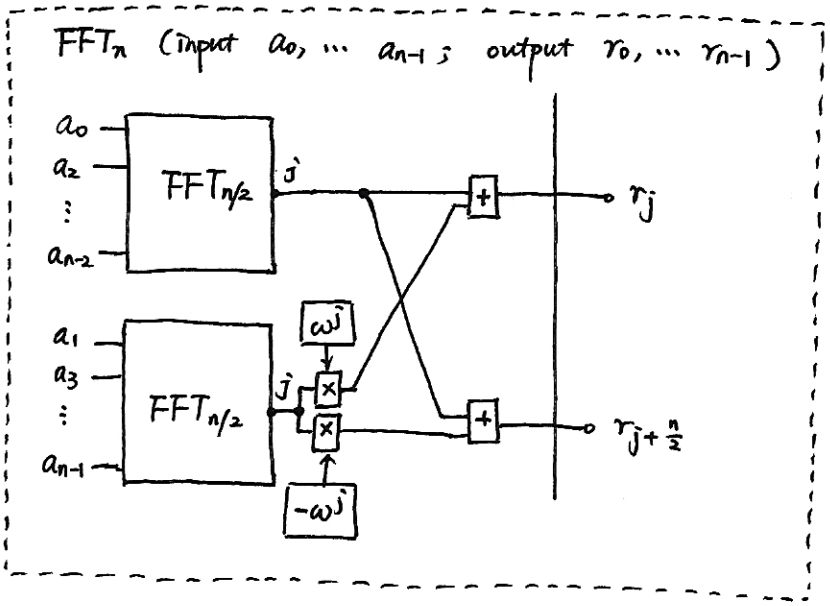
$r_i \leftarrow S_i + \omega^i S_i'$

$r_{i+\frac{n}{2}} \leftarrow S_i - \omega^i S_i'$

end for

return $\vec{r}$

In fact, we implement FFT in circuit-level. (hardware-level).

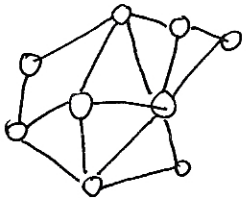


GRAPH

图 $G=(V,E)$ 有向图. 无向图

Adjacency Matrix 邻接矩阵 $\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & \cdots & & \end{pmatrix}$ $a_{ij} = \begin{cases} 1 & i \to j \text{ 有边} \\ 0 & i \to j \text{ 无边.} \end{cases}$

DFS: omitted

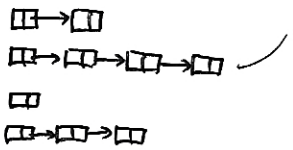


Coloring: NP-hard prob.

Representation of Graphs:

1) Adjacency Matrix

2) Adjacency Link List (邻接表)



- Depth First Search (DFS)

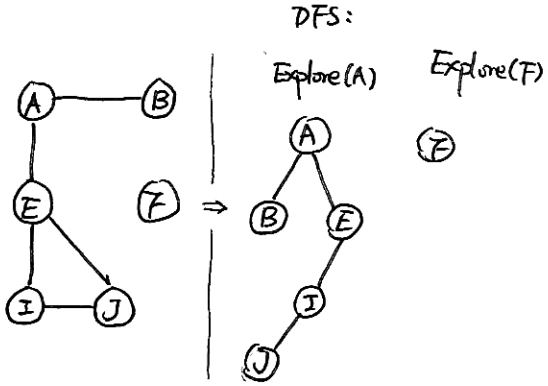
Input: $G=(V,E)$

Output: visited []

```
Explore (G, v)
    visited[v] ← True;
    previsit(v)        // for extension
    for each edge {v,u} ∈ E
        if visited[u] = false then
            Explore (G, u)
    postvisit (v)
```

```
DFS(G):
    for all v ∈ V:  visited[v] ← false;
    for all v ∈ V
        if visited[v] = false then  Explore (G,v)
```

DFS:



Adjacency Matrix: $O(n^2)$

Adjacency List: $O(n)$

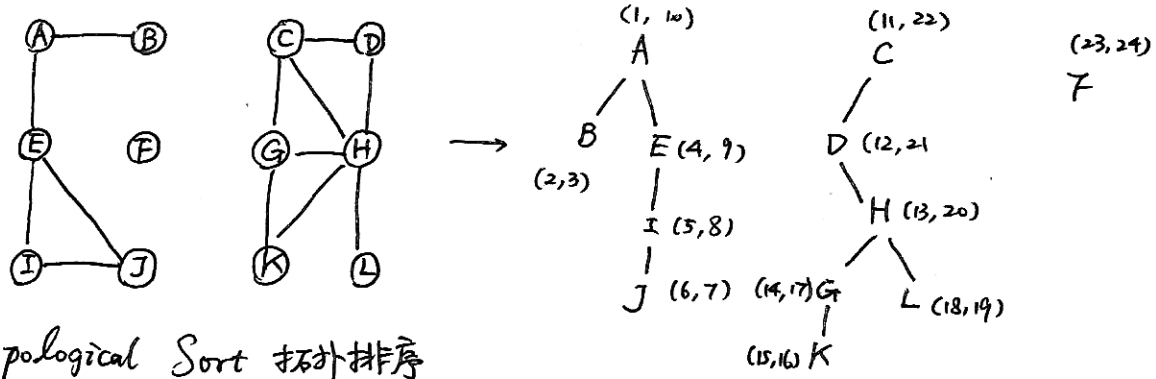
▷ Count Connected Components:

DFS(G)中计数即可 (从DFS → Explore 几次)

▷ Traverse 遍历

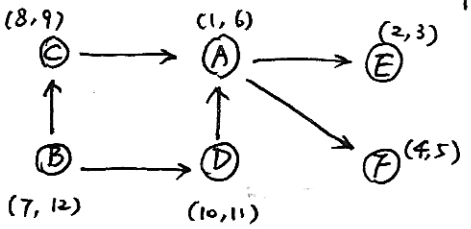
previsit(v): pre[v] = clock; clock++;

postvisit(v): post[v] = clock; clock++;

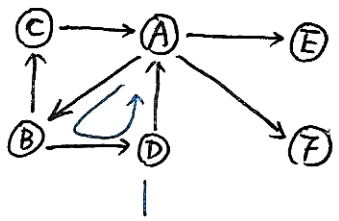


▷ Topological Sort 拓扑排序

DAG: Directed Acyclic Graph 有向无环图. —— 如何判断?



DAG



发现会再次访问到 visited = True 的节点 → 有环

Thm. In DAG. $u \to v$. Then $post(u) > post(v)$. [Proof.] Obvious.

后面全跑过了才会回来.

▷ Topological Sort: 按post() 大→小排序. ✓

e.g. 左上图. B, D, C, A, F, E.

	[入度为0]	[出度为0]
特殊节点:	B	E, F
	Source	sink
	源节点	汇节点
	max post	min post

注意到: 删去一个源节点后 → 剩下的图也是DAG.

▷ 每次找源, 删去它及它的边.; 对剩下的图进行该操作. ⇒ 删除序列即拓扑排序序列.

(入度为0)

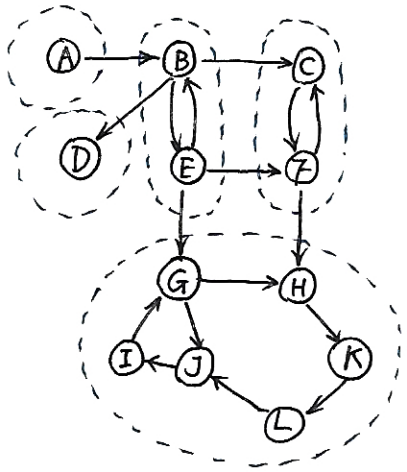
▷ Strongly-Connected Components 强连通分量

Def. "Connectivity" in Directed Graph : Strong Connectivity

$i \to j$ and $j \to i$

$\forall i, j \in C.\ i \to j,\ j \to i \Rightarrow$ Strongly-Connected Components

DAG: each vertex is a strongly-connected component.



Thm. For any graph G,

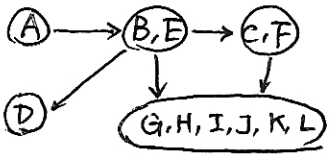
exists a partition

which consists of all "maximal"

strongly-connected components.

add another vertex then not strongly connected

Property 1. Meta Graph.



Exists a directed edge (from C_1 to C_2) between two components C_1, C_2.

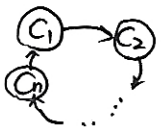
iff.

exist edges from $v \in C_1$ to $u \in C_2$.

(Prof: Obvious)

Property 2. Meta Graph is a DAG.

[Proof.] Directed. Trivial.

If exists a cycle. → 

→ $C_1, \cdots C_n$ should be a larger component.

Contradiction to "maximal" s-c component.

Find strongly-connected components using DFS.

- if start with A → explore all vertices
- if start with G → explore the maximal strongly-connected component where G belongs.

↑

This is because "GHIJKL" is the sink in the meta graph.

→ Discard GHIJKL.

C,F is the "new" sink.

→ Discard C,F.

B,E is the "new" sink.

→ Discard B,E

A is the "new" sink.

D is the "new" sink.

→ Discard D.

We find all strongly-connected components.

Property. In Meta Graph. Suppose $C \to C'$

Then we have [redacted]

$$\max_{u \in C} post(u) > \max_{v \in C'} post(v).$$

[Proof.] $C \xrightarrow{e} C'$.

If we explore $u \in C'$ first. Then we explore all vertices in C' before moving to C.
($C' \to C$ no edge)

If we explore $u \in C$ first. Then after visiting some vertices in C, we'll move to C' through edge e.

Explore all in C' then go back to C.

Either case, we get $\max_{u \in C} post(u) > \max_{v \in C'} post(v)$.

$\Rightarrow$ post(·)-maximal vertex : in the [redacted] "source" strongly-connected component.

(Meanwhile, the minimal is *not guaranteed* to be in the "sink" s-c component.)

→ But we want "sink" ..... We can **INVERT all edges in G!**

sink → source

source → sink.

(Meanwhile, s-c components stay the same!)

Def. $G^R := (V, E^R)$, where $G = (V, E)$. $E^R := \{(v,u) \mid (u,v) \in E\}$.

Algorithm. *Strongly-Connected Components Search*

Input: G

Output: SCC[]

DFS(G^R);

While $G^R \neq \emptyset$

  Find $v \in G^R$ with max $post(v)$

  Explore(G, v); // delete the vertex and all edges related to them in the current SSC.

  ↑ a SSC found.

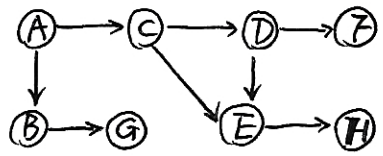
[redacted] (deleted in G^R)

$O(|V|+|E|)$

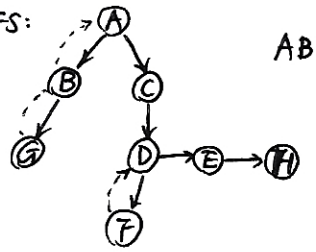
- Breadth - First Search (BFS)

DFS ~ Stack. BFS ~ Queue

e.g.



DFS:



ABGCDFEH

BFS:

ABCGDEFH


```
Procedure BFS (G, s)
    for all u ∈ V :  dist(u) = ∞
    dist(s) = 0
    Queue.push_back(s)
    While Queue is not empty
        u = Queue.pop();
        for edge (u,v) ∈ E
            if dist(v) = ∞ then
                Queue.push_back(v);
                dist(v) = dist(u) + 1;
        endfor
    endwhile
```

Claim. $dist(u)$ is the minimal distance between s and u in G.

[Proof.] We define the real minimal distance between s and u as $Dist(s,u)$.

By induction.

INDUCTION HYPOTHESIS.

$\forall d \in \mathbb{N}$. 1) $\forall v$. if $Dist(s,v) \le d$

At moment t_d. $dist(v) = Dist(s,v)$.

2) for other u. $dist(u) = \infty$.

3) the queue contains only vertices v with $dist(v) = d$.

BASE STEP. $d = 0$. ✓

INDUCTIVE STEP. $d = D$. ✓ ← I.H.

When we pop out all vertices with dist(·) = D.

we get moment [redacted]. t_{D+1}.

$t_D \sim t_{D+1}$. push-backed u. $dist(u) = D + 1$

[redacted].

u is 1 step away from v.

$Dist(s,u) = D+1$

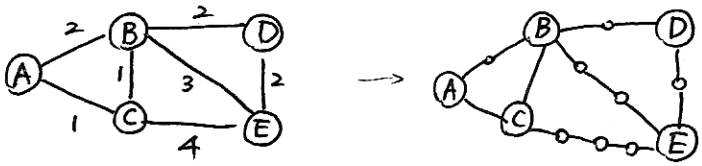
$dist(v)$

$v \in$ Queue.

⇒ 1) Holds.

2) ✓. 3) ✓. (Trivial.)

What about weighted graph?



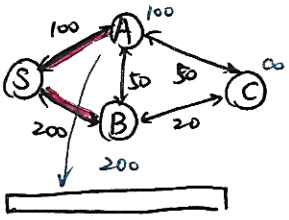
What if the weight is too large (100)?

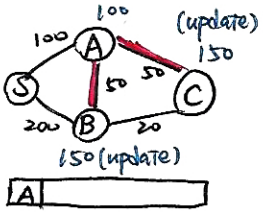
or is real (30.28)?

"alarm" — 模拟 "时刻" — 最短距离

alarm (多久会到我)

- Dijkstra's Algorithm





update all alarms

× Set alarms for s at time 0.

For those can't reach from s now, set ∞.

× Repeat until no more alarms.

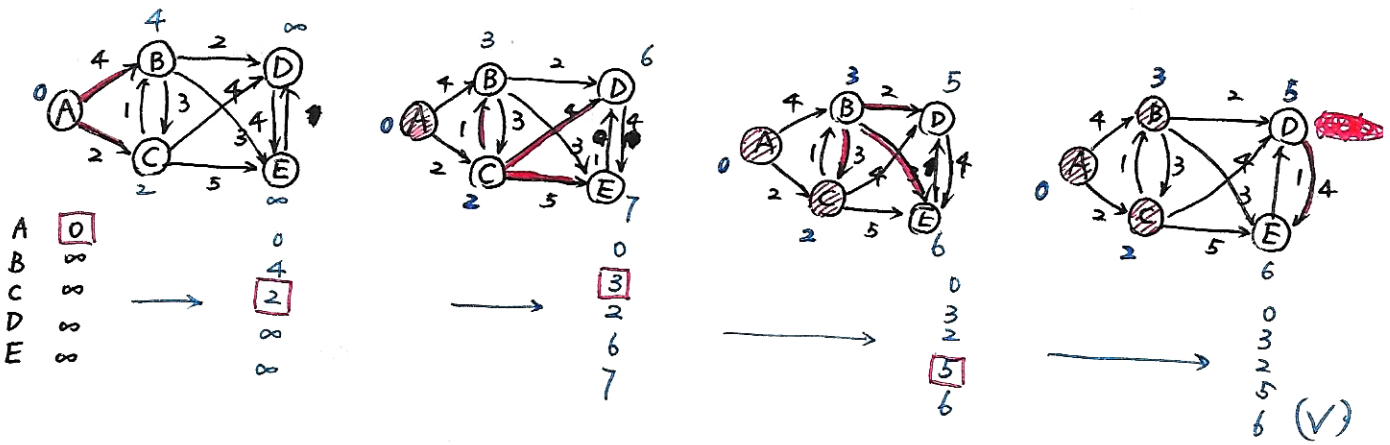
> pick the smallest alarm. ring at time T.

> For each neighbor v of u.

set its alarm to $\min\{alarm(v),\ T + l(u,v)\}$.

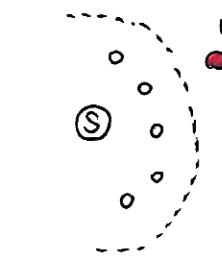
Procedure Dijkstra (G, s)

```
for all v ∈ V:  dist(v) = ∞.
dist(s) = 0;
H ← MakeQueue(V);          // unvisited vertices
while H is not empty
    u ← H.pop_min();       // pop out the u (unvisited) with min dist(·).
    for all (u,v) ∈ E:
        if dist(v) > dist(u) + l(u,v) then
            dist(v) ← dist(u) + l(u,v);   // 此处若增加记录u. → 最终可以返回 s→∀v. 的最短路径经过什么.
    end for
end while
```



Time complexity? → vulnerable to the realization of pop_min()

[Proof] the next vertex we find (visit)



dist(·) is already the minimal

By induction.

BASE STEP. $dist(v)=0$. minimal distance. ✓

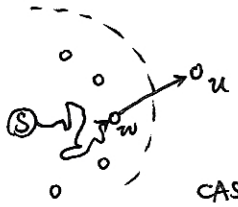
INDUCTION HYPOTHESIS. already minimal distance (guaranteed)

INDUCTIVE STEP. To prove for the red vertex u,

(the one with minimal dist(·) and unvisited)

dist(·) is the minimal distance from s to u

CASE 01. the minimal path is as follows. ($w \in$ visited, $w \to u$)

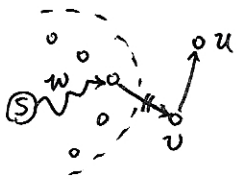


When we put w in visited list, we update all dist(·).

Thus, dist(·) is the path $s \to \cdots \to w \to u$. ✓.

CASE 02. the minimal path is: $s \to \cdots \to w \to \cdots \to v \to u$.

visited | unvisited



In this case. $dist(v) < dist(u)$. We should choose v instead.

This case is impossible.

Thus, Dijkstra's correctness is verified.

Qed. □

▷ Time complexity

$$|V| \cdot T(\text{pop\_min}()) + |E| \cdot T(\text{update dis}(\cdot))$$

What kind of queue have the fastest pop_min? — Priority Queue!

	pop_min()	update dis(·)	Total
Array	$O(\lvert V\rvert)$	$O(1)$	$O(\lvert V\rvert^2)$
Binary Heap	$O(\log\lvert V\rvert)$	$O(\log\lvert V\rvert)$	$O((\lvert V\rvert+\lvert E\rvert)\log\lvert V\rvert)$
* d-ary Heap	$O(d\log_d\lvert V\rvert)$	$O(\log_d\lvert V\rvert)$	$d\lvert V\rvert\log_d\lvert V\rvert + \lvert E\rvert\log_d\lvert V\rvert$

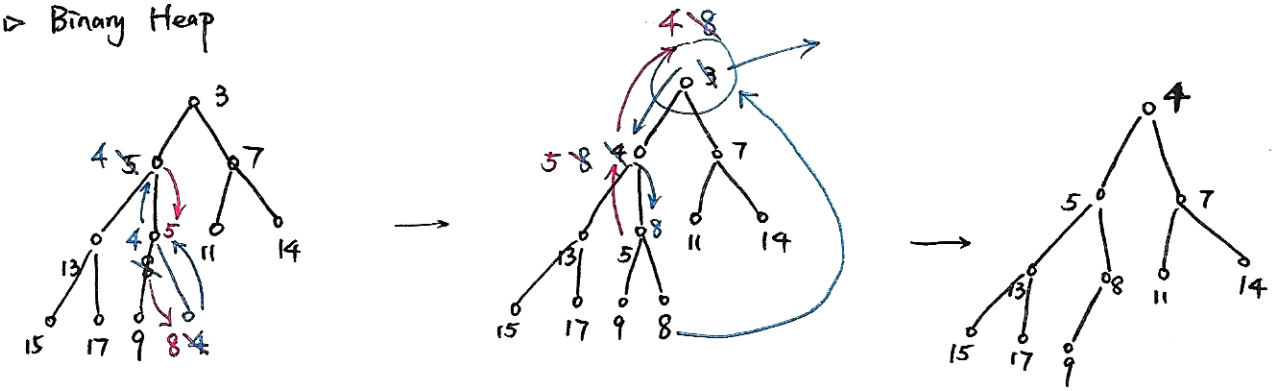
$|E| \sim |V|^2$

稠密圖 → Array 佳 (B.H: $O(|V|^2 \log|V|)$)

稀疏圖 → Binary Heap 佳

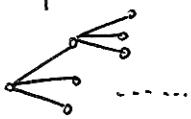
$|E| \sim |V|$.

▷ Binary Heap



add a new element (push_back)　　pop the min

▷ d-ary Heap

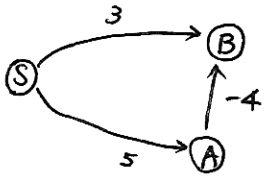


（反正我不想用它）

• Bellman-Ford Algorithm

Problems with Dijkstra: negative weight?

e.g.

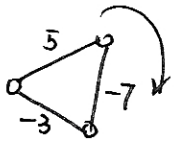


Dijkstra: visit B. dist(B) = 3! → WA

in fact: $\min dist(S \to B) = 5 + (-4) = 1$.

What leads to this faulty result? ~ all vertices are pushed into queue for only one time.

Moreover,



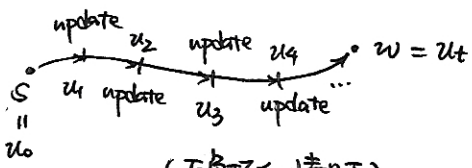
minimal distance DNE. ~ 负环

What still can work? ~ update.

$dist(v) = \min\{dist(v),\ dist(u) + l(u,v)\}$.

harmless!

Can we find a "correct" order? ←---------- Dijkstra in fact "update" all dist(·) in a specific order!



（无负环的情况下）

$u_0 \to u_1 \to \cdots \to u_t = v$ is a correct order.

↑ How to find this? ..... (this is exactly the shortest path.)

Method 1. Repeat for $|V|-1$ times:

[for all $(u,v) \in E$

update $\{u,v\}$. ← Harmless. only descend. never increase.

$O(|V||E|)$

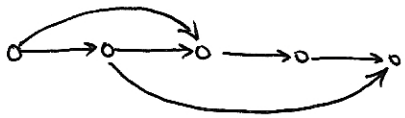
We can judge whether a negative cycle exists or not. by B-F Algorithm.

→ 一直在update. update $|V|-1$ 次. 则有负环.

▷ SPFA: One way to implement Bellman-Ford Algorithm.

```
Input: G, s
Queue.push_back(s);
for v∈V: dist[v] = ∞;
dist[s] = 0;
While Queue is not empty
    u ← Queue.dequeue();
    for v ∈ Neighbor(u)
        if dist[v] > dist[u] + l(u,v) then
            dist[v] ← dist[u] + l(u,v)
            if v ∉ Queue then Queue.push_back(v);
                             count[v]++;   // count[v] > |V|. 则有负环
            endif
        endif
    endfor
endwhile
```

- Shortest Path for DAG



We know any path in a DAG.

the path goes one-way in the ~~order~~ of Topological-sorted order!
sequence

(所有边都从"前"指向"后", 这里"前""后"是拓扑排序结果)

We just need to update the distance in the topological order.

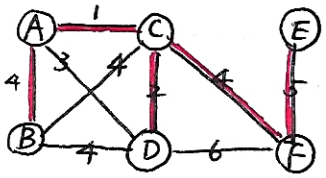
```
for v∈V (in the topological order)
    for every (u,v)∈E
        update {u,v}
```

← Faster Bellman-Ford!

GREEDY ALGORITHM 贪心算法

- Minimum Spanning Tree (MST)

Property. No cycle. (Trivial.)

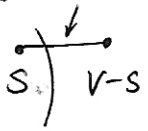


▷ Kruskal Algorithm

Every time we select the smallest edge which will not introduce a cycle. (connects a new node.)

[Proof of Correctness] (in fact proved in Discrete Mathematics)

Def. Cut.

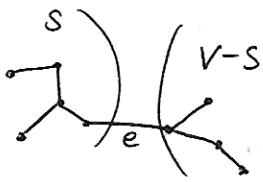


By induction.

1) $X \subseteq E$. (X is a part of some MST). $X=(S, E_X)$
2) $S \subseteq V$. $E(S, V-S) \cap X = \emptyset$
3) let e be the lightest edge in $E(S, V-S)$.

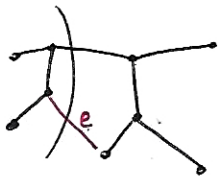
We prove $X \cup \{e\}$ is a part of some MST.

CASE 01. $e \in MST$. Correct.



The MST.

CASE 02.



The MST

$M \subseteq E$ is the MST containing X

$e \notin M$.

Then $M \cup \{e\}$ contains a cycle. (MST's property)

exists e' connecting S and $V-S$. $w(e')=w(e)$ (e is the lightest)

$M \setminus e' \cup \{e\}$ is also MST.

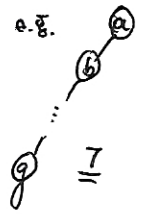
Thus, Kruskal Algorithm is correct.

```
Procedure Kruskal (G, w)
for all u∈V: makeset(u);          // each vertex is a set of its own
X ← ∅;
sort E by weight;
for all {u,v} ∈ E: (in increasing order of weight)
    if class(u) ≠ class(v) then    // class(·): find the set where · belongs
        add {u,v} to X;
        union(u,v);
```

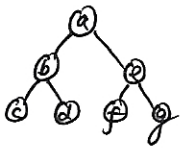
▷ Union-Find Set 并查集

```
makeset(v):  π(v) = v        // π(·): the parent of v

find(x):
    while x ≠ π(x):  x ← π(x)
    return x;

union(x, y):
    r_x ← find(x);
    r_y ← find(y);
    if r_x = r_y then return;
    if rank(r_x) > rank(r_y) then
        π(r_y) ← r_x
    else
        π(r_x) ← r_y
        if rank(r_x) = rank(r_y) then
            rank(r_y)++;
```

// How to union will affect time complexity!

e.g. 

$\equiv 7$

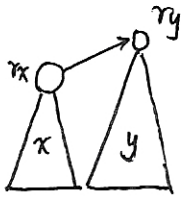
V.S. 

$\equiv 3$

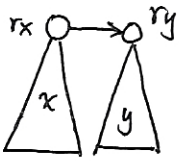
More balanced

We merge the shorter one into the higher one.

// rank(·): the height of the tree · belongs

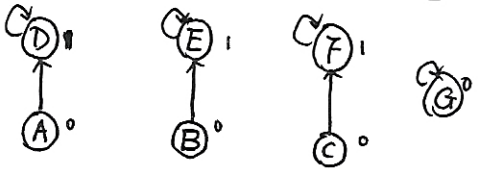


$\text{rank}(r_y)$



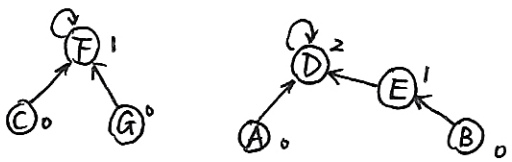
$\text{rank}(r_y)+1$

e.g. 

union(A, D) 

union(B, E)

union(C, F)

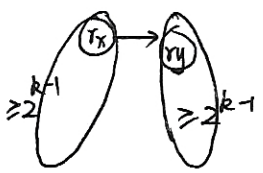
union(C, G) 

union(E, A)

Property. 1) $\forall x.\ \text{rank}(x) \le \text{rank}(\pi(x))$ ("=" iff. $x = \pi(x)$)

2) Any root node of rank k has $\ge 2^k$ nodes in its tree.

[Proof] Induction on k. $\text{rank}(r_x) = k-1$



$\text{rank}(r_y) = \text{rank}(r_x) = k-1 \xrightarrow{\text{INCR.}} k =: \text{rank}(r_y)'$

$2 \cdot 2^{k-1} = 2^k$. ✓

3) There are at most $n/2^k$ nodes of rank k,

i.e. $\max \text{rank}(\cdot) \leq \log_2 n$. ⟶ #查集 union: $O(\log |V|)$

[Proof]

$\#(\cdot)\ 2^k \leq n$.

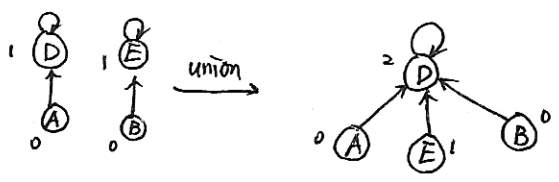
$\Rightarrow \#(\cdot) \leq \frac{n}{2^k}$ v.

▷ Prim Algorithm : direct implement of "cut property"

```
Procedure Prim
    for all u ∈ V:
        cost(u) = ∞
        prev(u) = null
    pick any initial value u0
    cost(u0) = 0
    H ← makequeue(V, cost)
    while H is not empty
        v ← H.pop_min()
        for each {v, z} ∈ E
            if cost(z) > w(v, z) then
                cost(z) ← w(v, z);
        end for
    end while
```

▷ Path Compression on Union-Find Set

```
find(x):
    if x ≠ π(x) then
        π(x) ← find(π(x))
    return π(x)
```



rank(·) is no longer the depth. "伪深度".

m operations (find/union) : $m \geq n$.

w.o. path compression $O(m \log n)$

w. path compression [Tarjan 1975] $O(m \cdot \alpha(n))$ → $\alpha(n)$: inverse of Ackerman function

N — #(atoms in the universe)

$\alpha(N) \leq 4$.

[Hopcroft & Ullman 1973] $O(m \log^* n)$

$\log^* n := k$, where $k = \#(\log_2$ operations on n until it $\le 1)$

i.e. $2^{2^{2^{\cdots^{2}}}}\Big\}\log^* n = n$.

$\log^*\left(2^{2^{2^{2}}}\right) = 5$. Regarded as "constant".

2^{65536}

$\{1\}, \{2\}, \{3,4\}, \{5,6,\cdots,16\}\{17,\cdots,2^{16}\}\{2^{16}+1,\cdots 2^{2^{16}}\}$ "amortized"

↖ "internal" 区间

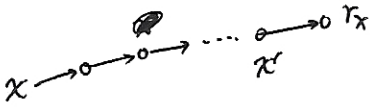
[Proof] 1) $rank(r_x) \in \{k+1, k+2, \cdots, 2^k\}$ 给 2^k 元.

Claim. 总共给所有点发的总钱数不会很多

We know rank(·) still holds. → Property 2&3 of Union-Find Set still holds.

$\#(\text{nodes w. rank} > k) \le \frac{n}{2^{k+1}} + \cdots = \frac{n}{2^k}$ ⇒ Total amount of money paid in $\{k+1, \cdots 2^k\} \le \frac{n}{2^k} \cdot 2^k = n$.

2) 收钱 Find(x)



CATAGORY(1): r_x. x' $(\pi(x') = r_x)$

CATAGORY(2): $\{y \mid rank(\pi(y))$ is in a higher internal than $rank(y)\}$.

CATAGORY(3): $\{y \mid rank(\pi(y))$ is in a same internal as $rank(y)\}$.

对每个 type(3) node 收1元.

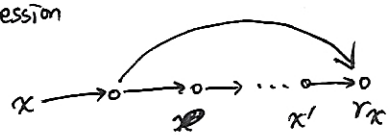
Cost of find(x) = 2 (type-(1)) + $\log^* n$ (type-(2)) + [money collected]

rank取值共有 $\log^* n$ 种

cost ≤ m operations ≤ $O(m \log^* n)$ + [Total money collected]

3) 无人破产.

In path compression



the new "parent" has higher rank than in the past.

find(x) 过程中 交钱 → 非根节点. rank(y) 自己不变

同时 path compression. rank(parent) ↑ (每次至少+1)

最多交 $(2^k - k)$ 次钱, parent 进入下一区间.

而之前被发了 2^k 元钱. 因此不会破产

∴ [Total money collected] ≤ [Total money paid] ≤ n.

⇒ Time complexity: $O(m \log^* n)$.

- **Task Assignment**

one task at a time.

n tasks $\quad \forall i \in [n]$. starting time: $s[i]$. finishing time: $f[i] \longrightarrow \max(\#\text{tasks})$?

Strategy 01. Sort by $s[\cdot]$.

Counter-Example. Task 1: $[1, 10]$ ← We choose

Task 2+3: $[3,5][6,8]$ ← Better ×

Strategy 02. Sort by $f[\cdot]-s[\cdot]$

Counter-Example. $[3, 5]$ ← We choose

$[1, 4][4, 8]$ ← Better ×

Strategy 03. Sort by $f[\cdot]$

[Proof of Correctness] We choose $\{s_1, s_2, \cdots s_\ell\} =: S$.

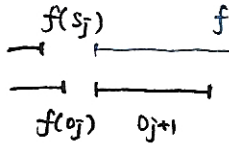
Let the optimal choice be $O = \{o_1, o_2, \cdots o_m\}$.

Obvious $m \geq \ell$. Now we prove $m = \ell$. (by contradiction).

Assume $\ell < m$. We prove that $\boxed{\forall i = 1, 2, \cdots \ell. \quad f(s_i) \leq f(o_i).}$

Induction on i.

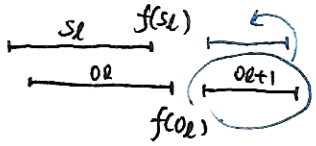
I.H. $f(s_i) \leq f(o_i)$ holds for $1, 2, \cdots j$



Then ← We would choose o_{j+1} instead (since $f(o_{j+1}) < f(s_{j+1})$.)

Thus, $f(s_{j+1}) \leq f(o_{j+1})$.

Therefore,



We would choose $o_{\ell+1}, \cdots o_m$ instead! Contradiction.

$\therefore \ell = m$.

Qed. □

- **Huffman Code**

Symbol set $\chi = \{A, B, C, D\}$.

		decode
A:	70	"0"
B:	3	"100"
C:	20	"101"
D:	37	"11"

prefix-free.

cost = $70 \times 1 + 3 \times 3 + 20 \times 3 + 37 \times 2 = 213$ (Better)

没有一个short码是另一个码的前缀.

		decode
A:	70	00
B:	3	01
C:	20	10
D:	37	11

cost = $(70+3+20+37) \times 2 = 260$

A general problem is:

t characters $\begin{cases} A \sim \text{frequency of appearance: } f(A) \\ B \sim \text{frequency} \quad \cdots \quad f(B) \\ C \sim \quad\quad\quad\quad f(C) \\ \vdots \end{cases}$ → How to encode these characters s.t. the cost is minimal?

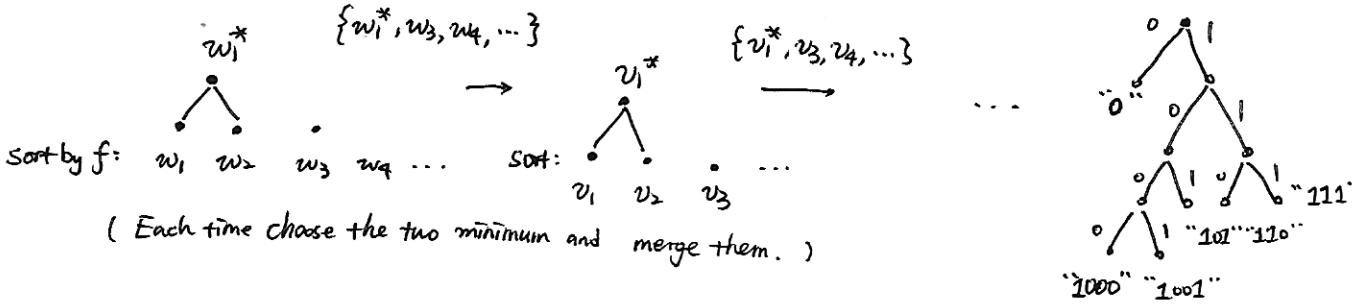
~ Huffman Code

* Strategy 01.

$0\cdots00 \sim A$
$0\cdots01 \sim B$ → $m = \sum_{i=1}^{t} f(i)$
$0\cdots10 \sim C$
$\vdots$

cost: $m\lceil \log_2 t \rceil$.

* Strategy 02.



(Each time choose the two minimum and merge them.)

[Proof of Huffman Code's Optimality]

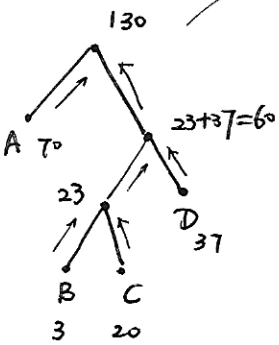
$\chi = [t]$ m — length of the string to encode

$\forall i \in [t]$. f_i — frequency of i in the string

Encode: $\chi \to \{0,1\}^*$.

$\forall i \in [t]$. $l_i := |\text{Encode}(i)|$. $\text{cost} = \sum_{i=1}^{t} l_i f_i$.

* weight(·) = weight(left child) + weight(right child)

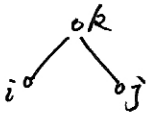


Def. $\forall v \in T$. $v \neq$ root.

$$\text{cost}(v) := \begin{cases} f_i & \text{if } v \text{ is a leaf with character } i \\ \text{cost}(u) + \text{cost}(w) & \end{cases}$$ where u and w are children of v.

Then: $\text{cost} = \sum_{\substack{v \in T, \\ v \neq \text{root}}} \text{cost}(v)$.

```
Procedure Huffman(f)
    Hl ← make_queue();      // the heap
    for i=1,2,··· t:    insert(Hl, (i, f_i))
    for k = n+1 → (2n-1)
        i ← Hl.pop_min();
        j ← Hl.pop_min();
```

create  ; [parent(i) ← k ; parent(j) ← k]

f(k) ← f(i) + f(j) ;

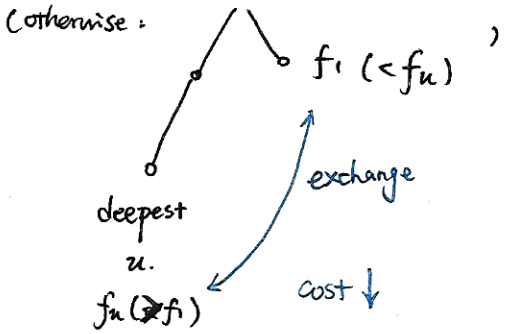
insert(Hl, (k, f(k)));

[Proof of Optimality] Assume $f_1 \le f_2 \le \cdots \le f_t$.

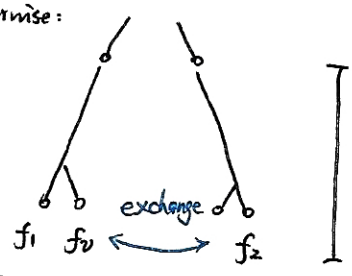
① ∃ an optimal encoding s.t.

f_1 & f_2 are the deepest and they share the same parent

(Otherwise:

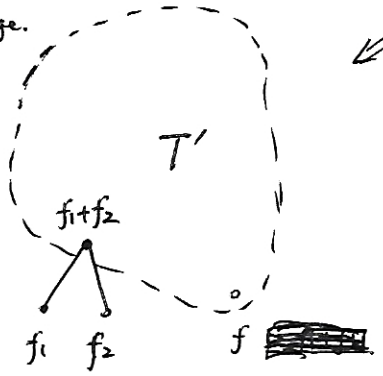


(Otherwise:



cost remain the same.

② Merge.



$$\text{minimize} \sum_{\substack{v \in T \\ v \neq \text{root}}} \text{cost}(v)$$

$$= \text{minimize}\ f_1 + f_2 + \sum_{\substack{v \in T' \\ v \neq \text{root}}} \text{cost}(v)$$

~ Induction.

Qed. □

▷ Digress to Information Theory

$p_i := f_i / m$ probability for the appearance of symbol i

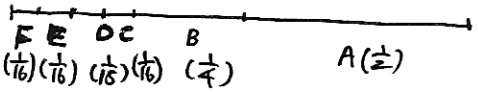
In some situation, Huffman Code address the entropy lower bound. $m \cdot H(x)$.

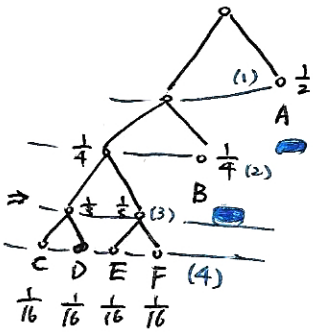
X: r.v. on χ w. $\Pr[X = i] = p_i$

$$H(X) = \sum_{i=1}^{t} p_i \log \frac{1}{p_i}$$

Shannon Entropy 熵

e.g. $\forall i \in \chi.\ p_i = 2^{-k_i}$



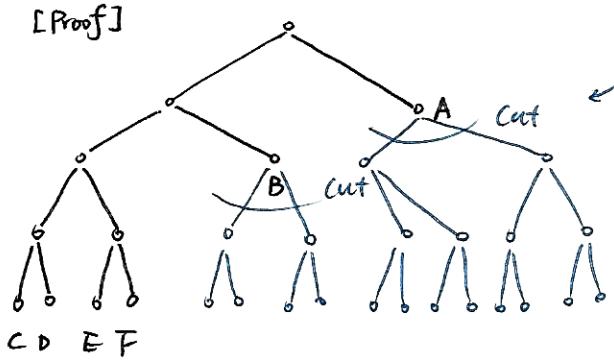


$$\text{cost} = \frac{1}{16}\cdot 4 + \frac{1}{16}\cdot 4 + \frac{1}{16}\cdot 4 + \frac{1}{4}\cdot 2 + \frac{1}{2}\cdot 1 = \sum_{i=1}^{t} p_i \log \frac{1}{p_i} = H(X).$$

Thm. Any prefix-free encoding of a m-symbol string requires $mH(x)+$ bits.

i.e. "$mH(x)$ is the amount of information contained in S".

[Proof]



prefix-free: cut-off some part of the tree.

$l = \max_i l_i$

$$\sum_{i=1}^{t} 2^{l-l_i} = 2^l$$

$$\Leftrightarrow \sum_{i=1}^{t} 2^{-l_i} = 1 \leq 1$$

↑ Kraft Inequality

Find an optimal encoding $\Leftrightarrow$ $\min \sum_{i=1}^{t} p_i l_i$

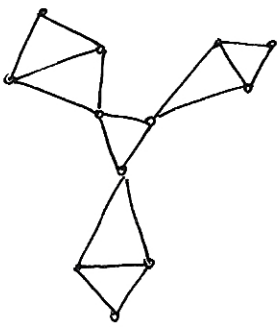
s.t. $\sum_{i=1}^{t} 2^{-l_i} = 1$

$(\forall i,\ l_i \geq 0)$ (in fact: l_i is integer)

Lagrangian Multiplier $\Rightarrow$ $l_i = \log \frac{1}{p_i}$ $\quad$ mincost $= \sum_{i=1}^{t} p_i \log \frac{1}{p_i} = H(X)$

(this can only be larger).

- Set Cover



$G = (V, E)$

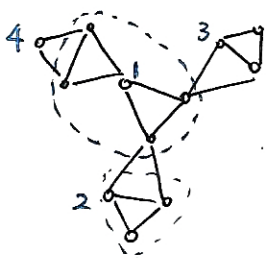
Choose $S \subseteq V$. (We construct hospitals on $s \in S$.)

$\forall i \in V$. $i \in S$ or $j \in S$ for some $\{i,j\} \in E$

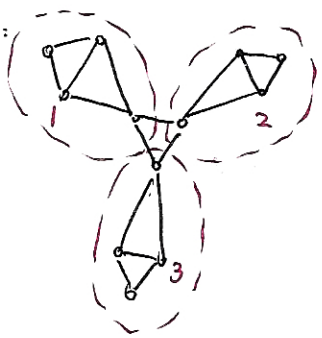
$\min |S|$.

U. $S_1, \dots S_m \subseteq U$.
Choose $\min [I]$. $I \in [m]$
s.t. $\bigcup_{i \in I} S_i = U$

Greedy Strategy: Each time we choose a S with largest cover. → 4.



in fact:



Greedy Algorithm does not return the best solution.

Assume the optimal solution chooses k sets.

The Greedy Algorithm chooses a sets.

Thm. $a \le \log n \cdot k$ (Greedy Algorithm is a log-approximation algorithm)

[Proof] Optimal Solution: $O = \{S_1, \cdots S_k\}$ $\quad n = |V|$.

n_t – #(uncovered vertices after round t)

$a = \min\{t \mid n_t = 0\}$

Our greedy algorithm: each step choose a set that cover maximal number of uncovered vertices.

① $n_0 = n$.

② $n_{t+1} \le n_t - n_t/k$ ← $\exists S_i \in O$, S_i covers at least n_t/k vertices.

$\le n_t(1 - \frac{1}{k})$ (O can cover V. → can also cover V_t. 抽屉原理. 至少有1个 S_i 满足).

$< n_t e^{-1/k}$

$\Rightarrow n_a < n \cdot e^{-a/k} \le 1 \quad \Rightarrow a \le \log n \cdot k$

$n_a = 0$

□

Set cover: NP-hard problem.

Thm. (Dinur & Steurer, 2013)

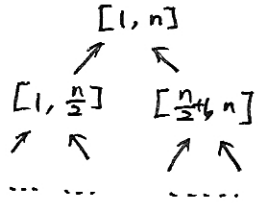
$\forall \varepsilon > 0$. exists no $(1-\varepsilon)\log n \cdot k$-algorithm for set cover, unless $NP = P$.

Dynamic Programming DYNAMIC PROGRAMMING

▷ Recall: Greedy & Divide and Conquer

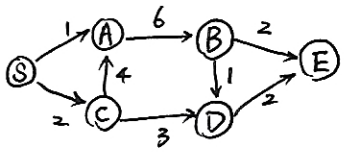
$\emptyset \to \{e_1\} \to \{e_1, e_2\} \to \cdots \to T$ ~ MST Prob. Prim/Kruskal: greedy

Divide & Conquer:



Problems are solved by solving subproblems. 子问题间关联: DAG (因此有确定顺序 (拓扑sort) 来解决这些子问题)

▷ Shortest Path (Reprise) on DAG.



Bellman-ford $O(|V|+|E|)$

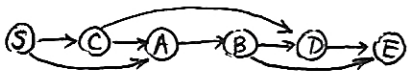
dist[i]: shortest path from s to i.

to reach D ⟶ must have reached C/B. 问题转化为 到达B 和 到达C

⇒ 递推式 (状态转移方程) $dist[u] = \min_{v \in Ngh(u)} \left(dist[v] + weight(v, u) \right)$

state: $\underbrace{v_1, v_2, \cdots v_k}_{Ngh(u)} \longrightarrow u$

Order: topological sort order (in DAG, 拓扑排序在前 to 点 → 在后 to 点) 任意边都满足



Implementation of a DP Algorithm (Direct Implementation)

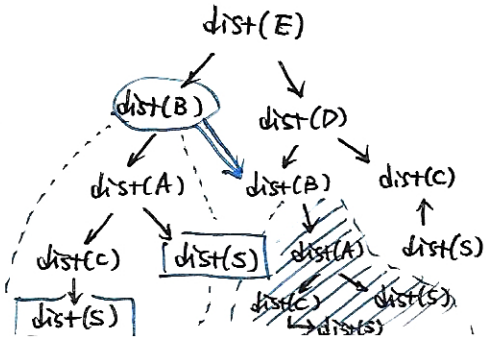
```
Topological sort on G
dist[s] = 0.   ∀ i ≠ s, dist[i] ← ∞.
for i ∈ V(G)   [in topological order]          O(|V|+|E|)
    dist[i] ← min_{j: (j,i)∈E} { dist[j] + l(j, i) }
return dist[t]
```

▷ Memorization 记忆化搜索

Normal Search

```
Function Dist(i)
    if i = s : return 0;
    return min_{j: (j,i)∈E} { Dist[j] + l(j, i) }
```

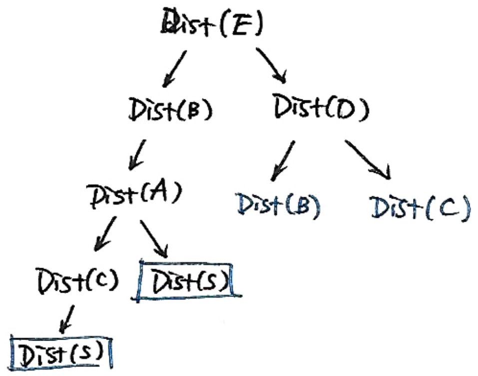
Might compute the same subquestion more than once!



Memorized Search $dist[s] \leftarrow 0$

Function $Dist(i)$

- if $dist[i] \neq \infty$: return $dist[i]$
- $dist[i] \leftarrow \min_{j:(j,i)\in E} \{ Dist(j) + l(j,i) \}$
- return $dist[i]$



只考察与我有关的状态 ⟶ 平均而言比 Bellman-Ford 要省时（后者把所有节点都考察了一遍）
记忆化搜索只考察了相关结点.

▷ Longest Increasing Subsequences

5 2 8 6 3 6 9 7
   △   △ △   △

$\{2,3,6,9\}$. $\{2,3,6,7\}$ 最长递增子序列（上升）

$\forall k \in [n]$ $F(k)$ = the length of the LIS in $a[1,2,\cdots k]$

$F(k+1) \longleftrightarrow F(k)$?

① 能接上. $F(k+1) = F(k)+1$

② 接不上. $F(k+1) = F(k)$ ⟶ can't determine.

$\forall k \in [n]$. $F(k)$ = the length of the LIS in $a[1,2,\cdots k]$ ending at $a[k]$.

$$\text{LIS in } a[1,\cdots,n] = \max_{k\in[n]} F(k)$$

$$F(k) = \max_{\substack{j<k \\ a[j]<a[k]}} \{ F(j) \} + 1$$

$O(n^2)$

* 还有优化. 以后再提

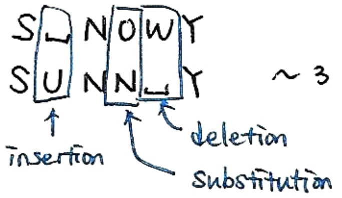
~~*其它优化：找连续上升子列结束后一段~~

▷ Edit Distance

SNOWY
↓
SUNNY

3 ways to edit
1) insertion
2) deletion
3) substitution

⇕ add "space". min (#(inconsistent)) 不一致个数



ABCDE
↓ substitute = 5
BCDEF

insert + deletion = 2

SNOWY → SUNNY

a[1, ⋯, n]

b[1, ⋯, m]

F[3,4]: minimal #(operations) needed for "SNO" → "SUNN"

F[k, l]. $k \in [n]$. $l \in [m]$

↓

Edit distance of a[1, ⋯ k] & b[1, ⋯ l]

Final answer: F[n, m]

如何计算子问题？

① ␣ / b[l] (insertion) $F[k,l] = F[k,l-1] + 1$

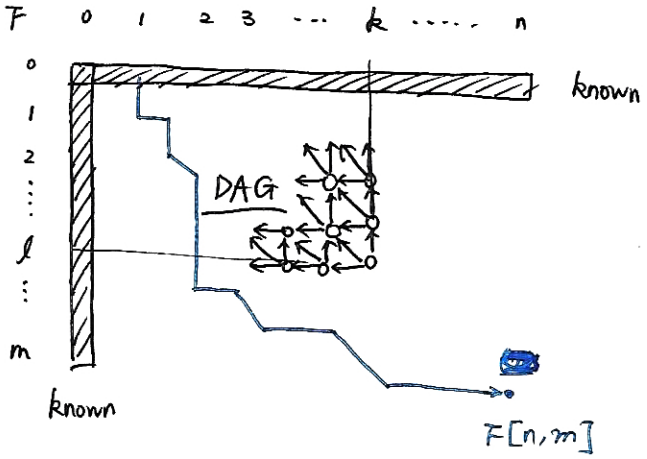
② a[k] / ␣ (deletion) $F[k,l] = F[k-1,l] + 1$

③ a[k] / b[l] (substitution) $F[k,l] = F[k-1,l-1] + \mathbb{1}[a[k] \neq b[l]]$

状态转移方程：$F[k,l] = \min\{F[k,l-1]+1,\ F[k-1,l]+1,\ F[k-1,l-1] + \mathbb{1}[a[k]\neq b[l]]\}$

Boundaries: $F[0,\cdot] = \cdot$ $(\forall j.\ F[0,j] = j)$

$F[\cdot,0] = \cdot$ $(\forall i\ F[i,0] = i)$



weight of edge $\in \{0, 1\}$.

"shortest path" on DAG.

▷ Knapsack Problem 背包问题

item	weight	value
1	6	$30
2	3	$14
3	4	$16
4	2	$9

Knapsack can hold 10kg at most

① w. repetition (可多次拿) ~ 48 (6+2+2)

② w.o. repetition (0/1) ~ 46 (6+4)

F[k]: max value one can carry with load $\leq k$. * 也可以 with load exactly k.

w. repetition. $F[k] = \max_{i:\, w_i \leq k} \{v_i + F[k-w_i]\}$

↑ 拿第i个item

↑ 拿之前 [redacted] 最大价值

$F[0] = 0$

Final ans: $F[W]$ / $\max_{0 \leq i \leq W} F[i]$

$O(nW)$

w.o. repetition → we need more information (whether i-th is ~~used~~ fetched or not)

$F[i,k]$: maximal value one can carry with load exactly k choosing items from item $1, \cdots, i$

$$F[i,k] = \max \{ F[i-1,k] \text{ (不取第}i\text{个)}, \quad F[i-1, k-w_i] + v_i \text{ (取第}i\text{个)} \}$$

$O(nw)$

"0/1 背包"

▷ Chain Matrix Multiplication

$A \times B \times C \times D$ → the minimal computations are?

[50×20] [20×1] [1×10] [10×100]

$C_{ij} = \sum_{k=1}^{y} A_{ik} B_{kj}$

$A \times B = C$ (x×y, y×z, x×z)

computation: $x \cdot y \cdot z$

1. $A \times ((B \times C) \times D)$ (20×10, 20×100) $20\times1\times10 + 20\times10\times100 + 50\times20\times100 = 120200$

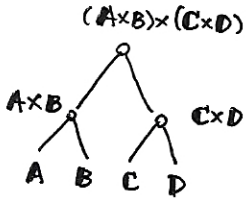
2. $(A \times B) \times (C \times D)$ (50×1, 1×100) $50\times20\times1 + 1\times10\times100 + 50\times1\times100 = 7000$. (Better!)

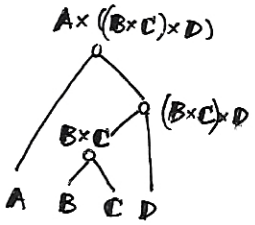
Given n matrices $M_1, M_2, \cdots M_n$, where M_i is a $m_{i-1} \times m_i$ matrix.

$m_0, m_1, \cdots m_n$

Decide our order to compute s.t. # (multiplication) is the minimum.

We use a tree to represent the expression. → leaf: matrices. non-leaf node: multiplication.

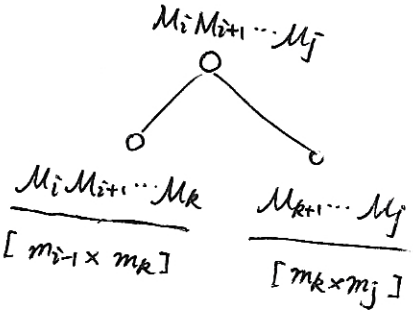




⇒ Decide how to construct a "tree" with minimal cost.

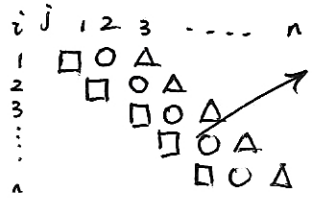
$F[i,j]$ = min cost to multiply $(M_i, M_{i+1}, \cdots M_j)$

$$= \min_{k \in [i,j-1]} \{ F[i,k] + F[k+1,j] + m_{i-1} m_k m_j \}$$



Boundary: $F[i,i] = 0$.

↑ We calculate these results in the order of the size of the "tree", i.e. $j-i$



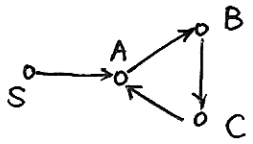
$O(n^2)$.

▷ Shortest Path on Directed Graph (Reprise #2)

D.P. ⟶ on DAG：因为有一个确定的顺序（拓扑排序）来决定 dist(·).

不会出现需要其它的 dist[·] 时该值仍未确定的情况.

如何在一般图上对最短路径进行DP？ ~ 构造 DAG？



$dist[i,v]$ = length of the shortest path between S and v with "at most" i intermediate vertices.

$$dist[i,v] = \min_{u:(u,v)\in E} \{ dist[i-1,u] + l_{u,v} ; dist[i-1,v] \} \quad (i \geq 1)$$

* 其实可以 $l_{v,v}=0$ 来做这点，但没必要

$$dist[0,v] = l_{s,v}. \quad (\text{有边时是边权，否则赋值为 } \infty)$$

$$dist[v] = dist[n-2,v]$$

最多把所有点都经历一遍 (除非有负环)

```
for i = 0 ⟶ n-2
    for ∀(u,v) ∈ E
        dist[i,v] = min{ dist[i-1,v], dist[i,v], dist[i-1,u] + l_{u,v} }
```

⟶ $dist[v] = \min\{ dist[v], dist[u] + l_{u,v} \}$

O(n,m)

Bellman-Ford.

（可能已经走过比 i 更多的中间点3，但只要无负环就只是提前到达而已）

（即 $dist[u] \leq dist[i-1,u]$）

第 i 轮过程中

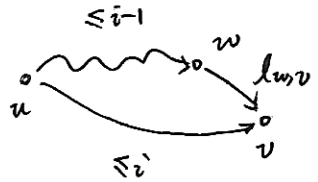
▷ All Pair Shortest Paths

$G=(V,E)$. $\forall u,v \longrightarrow dist[u,v]$

① $|V|=n$. $|E|=m$. ⟶ 对所有 $v \in V$ 跑一遍 Bellman-Ford $O(n^2m)$

② DP.

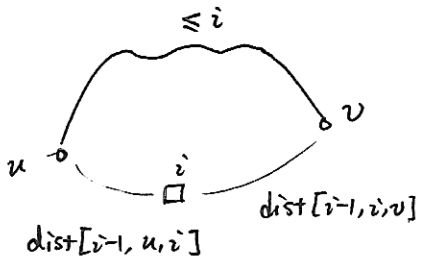
$dist[i,u,v]$: "length of the shortest path between u and v with at most i intermediate vertices."



$$dist[i,u,v] = \min_{w:(w,v)\in E} dist[i-1,u,w] + l_{w,v}$$

$dist[u,v] = dist[n-2,u,v]$ $O(n^2m)$

③ Floyd-Warshall

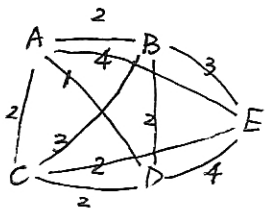


$dist[i,u,v] = \min\{ dist[i-1,u,v],\ dist[i-1,u,i] + dist[i-1,i,v] \}$

$dist[u,v] = l_{u,v}$

```
for i = 1 → n
    for j = 1 → n
        for k = 1 → n
            dist[j,k] = min{ dist[j,k], dist[j,i] + dist[i,k] }
```

▷ Traveling Salesman Problem (TSP). NP-Hard.



访问所有点并返回出发点.
最短路径是?

- 枚举: $(n-1)(n-2)\cdots = (n-1)!$ <枚举每一个点> $\sim O(n!)$
- DP.

dist[v]: the length of the shortest path starting from 1 and ending at v.

shortest path: $dist[v] = \min_u \{ dist[u] + l_{u,v} \}$.

↑ Problem: 没有记录经过了哪些点.

增加维度 ←

[Math]

1) $(\frac{n}{2})^{\frac{n}{2}} \le n! \le n^n$

$\log(n!) = \Theta(n \log n)$

2) Stirling.

$\log(\sqrt{2\pi n}(\frac{n}{e})^n) = \Theta(n \log n)$.

$\forall S \subseteq V$.

$1 \in S$.

$F[S,j]$:= the length of the shortest path starting from 1 and ending at j, and all vertices in S are visited exactly only once in the path

$O(2^n \cdot n^2)$

↑ 枚举j, 每个要对i取min

$$F(S,j) = \min_{\substack{i \in S \\ i \neq j}} \{ F(S-\{j\}, i) + l_{i,j} \}$$

Boundary: $F(\{1\}, 1) = 0$.

DAG order: sorted by $|S|$

$\log(2^n \cdot n^2) = n \log 2 + 2 \log n$

$< \log(n!) = \Theta(n \log n)$

▷ Color Coding NP-Hard

Given graph G. $|V| = n$. Given k. Prob: Does G contain a simple path of length k? (点数)

(边权都为1)

边数为 k-1

① $O(n^k)$ 暴力枚举

② a linear-time algorithm. $O(n+m)$. (given constant k)

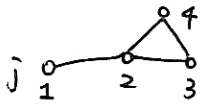
→ Colorful Path

$G=(V,E)$. $c: V \to [k]$.

A path is colorful if $\bigcup_{v \in P} \{c(v)\} = [k]$. Does G have a colorful simple k-path?

$F[S, j]$:= Whether exists a colorful simple path starting at j, using colors in S $\quad S \subseteq [k],\ j \in V$

e.g.



$F(\{1,2,3,4\}, j) = \text{true}$

$$F[S, j] = \bigvee_{i:(i,j)\in E} F[S - \{c(j)\},\ i]$$

Boundaries: $\forall i,\ F[\{c(i)\},\ i] = \text{true}$

Result: "$\exists j.\ F[[k], j] = \text{true}$" ⟶ exists a colorful simple k-path.

DAG order: sorted by $|S|$

$$O(2^k \cdot (n+m)) = O(n+m)$$

(2^k: constant)

□ Look back at non-color problem.

Fix a simple k-path P. — $\Pr[P \text{ is colorful}] \geq \frac{k!}{k^k}$ ← colorful 的方案 / ← 总的选色方案

Repeat T times
  randomly color G
  If $\exists$ colorful k-path in G, return True
End Repeat
return False

$\mathbb{E} = \frac{p}{1-p}$ ↑ Geometric Dis.

↳ $\mathbb{E}[$其Crepetitions until P is colorful$] = k^k/k!$

Select $T = 100 \frac{k^k}{k!}$ ⟶ 99% accuracy <Markov Inequality>

去随机化的版本: Universal Hash Function (Color-Coding, Alon-Zwick, Yuster.)

$$O\left(\frac{k^k}{k!} \cdot 2^k \cdot (n+m)\right)$$

↓

$f(k) \cdot \text{poly}(n)$ "Fixed-Parameter Tractable Algorithm" 固定参数 可解

▷ Maximal Independent Set on Tree

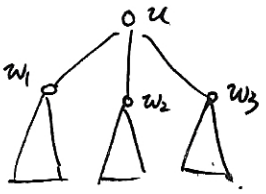
$G = (V, E)$. $S \subseteq V$ is an independent set iff. $\forall u, v \in S.\ \{u, v\} \notin E$.

NP-hard Problem.

subproblem: $\forall u \in V.$ $T_u :=$ the subtree rooted at u

$F[u] :=$ size of the max independent set in T_u. Final Ans: $F[r]$

$$F[u] = \max \left\{ \underbrace{\sum_{w:\ \text{child of } u \text{ in } T_u} F(w)}_{u \notin S},\ \underbrace{\sum_{v:\ \text{grandchild of } u \text{ in } T_u} F(v) + 1}_{u \in S} \right\}$$

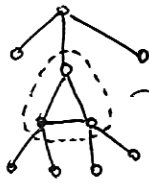


$F(u) = 1$ iff u is a leaf. (i.e. $T_u = \{u\}$).

$O(|V|)$. (Tree → $|E| = |V| - 1$)

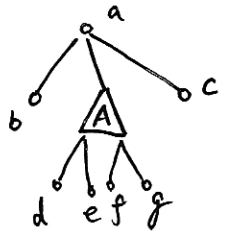
1) What if G is not a tree? ⇒ $O(2^{|V|} \cdot |E|)$.

2) What if G = Tree $\cup \{(u, v)\}$?



导致 T_u 不再 well-defined.

缩点 ⟹

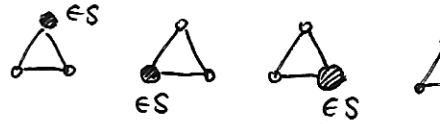


T_a. T_b. T_c

T_A. T_d. T_e. T_f. T_g

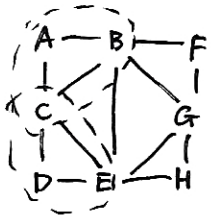
"Supernode"

对 T_A. 拆成以下情况:

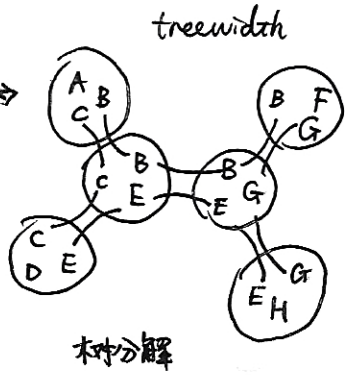
max {  4个状态

$O(|V|)$.

3)*



⇒



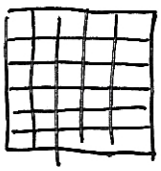
树分解

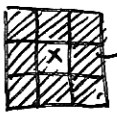
(Tree Decomposition)

Algorithmic Meta Thm.

对可用逻辑公式表示的图 $tw(G)$ 均可用 $O(|E|)$ 算法解决.

▷ State Compression

$m \times n$  求最多能放多少个互不攻击的国王.



1. $2^{m \times n}$. 每个格子的情况

2.  4×4 滑动.

$O(2^{m \times n})$

3. $F(i)$ —— 前 i 行最多放 # 个国王

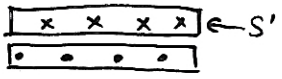
$F(i, S)$: 前 i 行. 第 i 行按 S 的方式放置

$i \in [m]$. $S \subseteq [n]$

$$F(i, S) = \sum_{S' \subseteq [n]} F(i-1, S')$$

S' 枚举第 $i-1$ 行放置项

S' 与 S 不冲突



$O(m \cdot 2^n \cdot 2^n) = O(m \cdot 4^n)$

▷ Largest Number in every k consecutive numbers

e.g. $k=4$

a:	3	5	7	12	3	5	4	5	6	7	5	10	8
result: (b)	3	5	7	12	12	12	12	5	6	7	7	10	10

$b[i] := \max_{\max\{1, i-k+1\} \le j \le i} a[j]$

1. $O(nk)$.

2. Priority Queue 优先队列

维护 $C = \{a[i-k+1], a[i-k+2], \cdots a[i]\}$

- $b[i] \leftarrow \max\{x \mid x \in C\}$ $O(1)$
- C. remove $\{a[i-k+1]\}$ $O(\log k)$
- C. add $\{a[i]\}$ $O(\log k)$

Binary Heap.

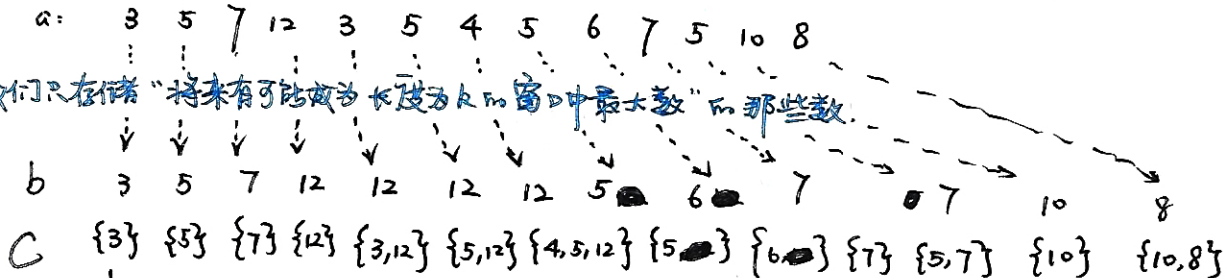
$O(n \log k)$

→ 是否可优化?

是不是所有的信息都会被用到?

(并非所有的都必须被存)

More optimization:



→ 我们只在乎"将来有可能成为长度为 k 的窗口中最大数"的那些数.

有 5 在, 5 离之后的 i 肯定比 3 更近, 5 又比 3 大.

(5 比 3 离之后的数更近, 3 比 5 先离开窗口, 而且还比 5 小.)

"3 不可能再有用!" → 直接替换掉

C 中始终只有: { 比其中的年轻但比我弱; 比我强但没我年轻.

C 

"年老" ⟶ 年轻

大 ⟶ 小

(来一个年轻且更大的数，比其小的数可以直接出队.)

(递减的优先队列)

C: contains number which have the potential to be the largest number in some k consecutive numbers in the future.

~~for $i = 1 \to n$~~

~~C.enqueue(a[i]);~~ ~~// Meanwhile, pop out C[tail] ≤ a[i].~~

~~b[i] ← C.pop-max();~~ ~~// In fact, C[head]~~

$C \leftarrow \emptyset$; head ← 0; tail ← -1;

for $i = 1 \to n$

if (head > 0 and C[i].time is out of date) head ← head + 1;

// 把滑出窗口的 pop 出去

While (tail ≥ head and C[i].num ≤ a[i]) tail ← tail - 1;

// 把比我年老又比我弱的 pop 出去

tail ← tail + 1;

C[tail] ← { i (time), a[i] (num) };

b[i] ← C[head];

▷ Longest Increasing Sequence (LCS)

[2] 8 [3] 1 [5 6] 4 [7] 2

1. F(i): the length of the LCS in a[1, 2, ... i] ending at i

$$F(i) = 1 + \max_{\substack{j<i \\ a[j]<a[i]}} F(j)$$

$O(n^2)$

2. G(i,j): $\forall i, j \in [n]$. consider all LCS in a[1, ..., i] of length j.

G(i,j) is the smallest ending number of these sequences.

("多小的代价" 可以在 1~i 中找到长为 j 的子串)

if can't find such sequences, $G(i,j) = \infty$.

$$LCS = \max \{ j \mid G(n,j) \neq \infty \}$$

Obvious: G(i,j) is increasing in j, decreasing in i. (Can be proved via contradiction.)

e.g. G(6,1) = 1. G(6,2) = 3 G(6,3) = 5. G(6,4) = 6. G(6,5) = ∞ ...

G(7,1) = 1

4 · (2,3,4)

~~$G(i,j) = G(i-1,j)$ if $a[j] > G(i-1,j)$ ← 看错~~

When to update $G(i,j)$?

$G(i,j) = G(i-1,j)$

Find j^* s.t. $G(i-1,j^*-1) < a[i] < G(i-1,j^*)$. $\Rightarrow G(i,j^*) = a[i]$

可以组成"代价"更小而长度 j 的 LIS.

$$G(i,j) = \begin{cases} a[i] & \text{if } j = j^* \\ G(i-1,j) & \text{o.w.} \end{cases}$$

How to find j^*? $\Rightarrow$ $G(i,j)$ is increasing! (by j) $\rightarrow$ 二分法都可 $O(\log n)$

$O(n \log n)$

▷ Egg Toughness Test

m identical eggs. a building with n level. ~~Determine it can at most remain~~

确定从哪一层坠落恰好不会碎. ~~complete ... falling from ... floor.~~

1. $m=1$. 只能 1, 2, 3, 4, ... (否则第一次就碎3. 无法确定) 如果

2. $m=2$? $O(\sqrt{n})$

让各种情况下扔的次数尽可能接近.

e.g. 10楼 —— 碎3. 另个 1~9 → 1~9次

└ 没碎 20楼 —— 碎3, 另个 11~19 → 2~10次

└ 没碎

.....

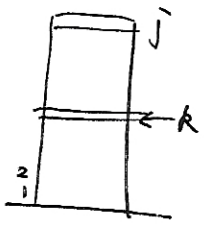
不均匀地分块. 可能会更优.

$F(i,j)$ = #(of tests if we have i eggs and j levels) → Ans: $F(m,n)$

$$F(i,j) = \min_{k=1,2,\cdots j} \left\{ \max \{ F(i-1,k-1), F(i,j-k) \} + 1 \right\}$$

$F(i-1,k-1)$: 前一层碎了(第k层) 换新的蛋 第1次试 → 可能的范围 0~k-1 楼

$F(i,j-k)$: 没有碎 则相当于站在第k层为地面的楼上扔鸡蛋验证 → 可能的范围 1~j楼 (k+1)



枚举k

总状态数 $mn \cdot n$

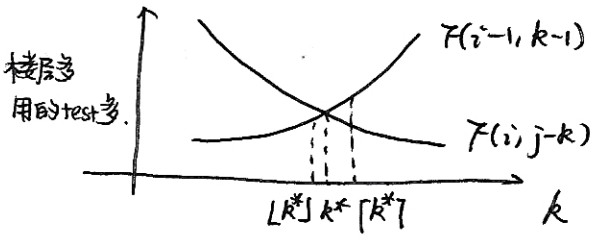
$O(m \cdot n^2)$

Boundaries: $F(1,\cdot) = \cdots$ $F(\cdot,0) = 0$

进一步优化. Observation.

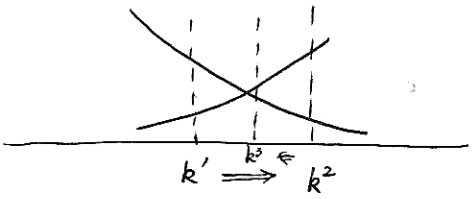
$f(i,k)$ is non-decreasing in k

$\forall\, i,k \quad f(i,k) \le f(i,k+1)$



k^* 使得: $f(i-1,k^*-1) \approx f(i,j-k^*)$

↑ 可以用二分法来查找 k^*. (min max、不断接近时最优)



寻找 k^*: $O(\log n)$

$\Rightarrow$ 时间复杂度: $O(n \log^2 n)$

Another Method: 换个角度再看同一个问题

$G(i,j)$ = max level one can determine with i tests and j eggs

Ans: $\min\{i^* \mid G(i^*, m) \ge n\}$

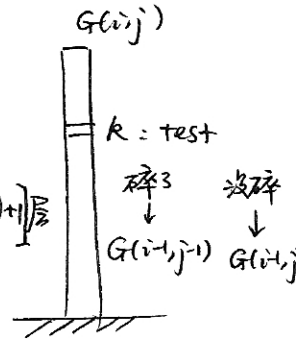
$G(i,j) = G(i-1,j) + G(i-1,j-1) + 1$

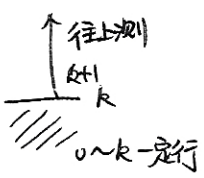
$G(i-1,j)$: (k层) 某一层 没碎

最多 那么可以测到这一层往上 这么楼和地方 (k+1层往上测)

$G(i-1,j-1)$: 某一层碎了. 测 0 ~ k层 要完全能测完 则 $k \le G(i-1,j-1)-1$ 所以最多测到这么多层

+1: 当前层 (k) 即第 $\lceil G(i-1,j-1)+1 \rceil$ 层





j: #(eggs) $O(\log n)$

i: 我们从 1 开始增加. (因为 $i-1 \to i$, 只要从下往上算即可)

最多算到 i^* 可行就不用往后算了.

$O(i^* \log n) \le O(\sqrt{n} \log n)$

$m=1$. $i^* = n$ (可特殊处理)

$m \ge 2$. $i^* \le \sqrt{n}$ ($m=2$ 时 $i \sim \sqrt{n}$)

实际上是: $O(n^{\frac{1}{m}} \log n)$.

一个显然的优化方式

当 $m \ge \lceil \log n + 1 \rceil$ 二分法一定最优

直接用二分法就行

只关心 $m \le \lceil \log n + 1 \rceil$ 时用 DP.

$\to O(n^2 \log n)$

LINEAR PROGRAMMING

- Review of Linear Optimization

$$\max\ c^Tx \qquad c\in\mathbb{R}^n$$
$$\text{s.t.}\quad Ax\le b \qquad x\in\mathbb{R}^n$$
$$x\ge 0 \qquad A\in\mathbb{R}^{m\times n}$$

Simplex 单纯形法

pivoting rule

Move to a neighbour with larger c^Tx. (贪心) (largest)

Worst case: exp-time 指数时间算法

* 但这些反例非常孤立

所以对每个要处理的 x 做一个小扰动

$A+\tilde{A}$ ($\tilde{A}$: Gaussian Noise)

$\longrightarrow$ O(Polynomial) * Smoothed Analysis

LP has a polynomial-time algorithm or not?

▷ Ellipsoid Algorithm 椭球法

▷ Interior Algorithm 内点法

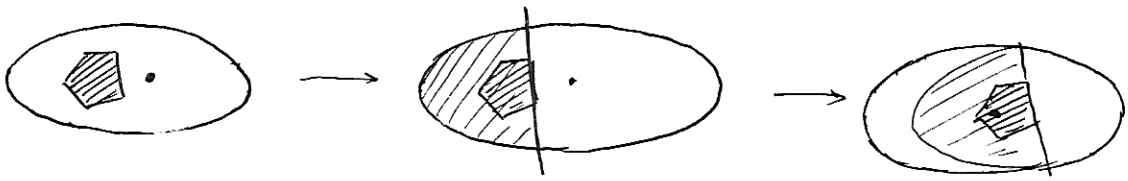
* Ellipsoid: ① 对任意点，输出 { YES 在可行域 / NO, c 不在，输出不满足的 restriction c. (返回1个即可)

② 始终维护一个椭球。椭球中心是否在可行域？(进行①过程) YES则完毕

(即把 $\max c^Tx$ s.t. … 转化为：$c^Tx\ge B$, s.t. … 看有无解 有解)

用①过程返回的限制条件切割椭球，并生成1个包含该椭球的新椭球，

重复②过程.



* Interior_point Algo.: 找到可行域内的点，然后向 c^Tx 更优的方向移动

且保证移动后依然在可行域内

▷ Duality Problem

$$\max\ c^Tx =: f \qquad \min\ b^Ty =: \phi$$
$$\text{s.t.}\quad Ax\le b \longleftrightarrow \text{s.t.}\quad y^TA\ge c^T$$
$$x\ge 0 \qquad y\ge 0$$

(P) (D)

f^*. ϕ^*

Duality Thm.

$\hat{x}$ is a feasible set of (P)

$\hat{y}$ is a feasible set of (D)

Weak Duality (天然成立) $f^* \geq \phi^*$.

Strong Duality $f^* = \phi^*$ 一个必要条件: Slater's condition

- Approximation Algorithm

SET COVER

select $I \subseteq [m]$. { m sets $S_1, \cdots S_m$. $S_i \subseteq [n]$; n elements }

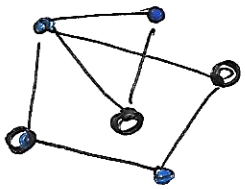
s.t. $\bigcup_{j \in I} S_j = [n]$. $\min |I|$. $\log n$

▷ Vertex Cover

$G = (V, E)$ $S \subseteq V$. S is a vertex cover iff. $\forall \{i,j\} \in E$, $i \in S$ or $j \in S$.

min vertex cover.

n elements ▷ E. $S_i = \{e \mid i \in e\}$.



min vertex cover $\Longleftrightarrow$ max independent set

(S is a vertex cover $\Longleftrightarrow$ $V \setminus S$ is an independent set)

$\forall i \in V$. introduce $x_i \in \{0,1\}$ $x_i = \begin{cases} 1 & \text{选 } i \\ 0 & \text{不选 } i \end{cases}$

min vertex cover $\Longleftrightarrow$

AIM: $\min \sum_{i=1}^{n} x_i$ subject to $\begin{cases} x_i + x_j \geq 1. & \forall \{i,j\} \in E \\ x_i \in \{0,1\} & \forall i \in V \end{cases}$ IP.

↑ 非线性的约束

⇓ Linear Programming Relaxation (松弛)

Integer Programming

$\min \sum_{i=1}^{n} x_i$ S.t. $\begin{cases} x_i + x_j \geq 1. & \forall \{i,j\} \in E \\ x_i \in [0,1] & \forall i \in V \end{cases}$ LP.

Obvious $opt(LP) \leq opt(IP)$. (∵ LP is a relaxation.)

We can get $\{x_i^*\}_{i \in V}$ of LP. in $O(\text{Polynomial})$ Time. ← optimal solution of (LP)

How to get the optimal solution of (IP)?

$\{x_i^*\} \xrightarrow{\text{rounding}} S$ ($S := \{i \mid x_i^* \geq \frac{1}{2}\}$) why 1/2 ?

Thm. S is a 2-approximation of minimum vertex cover.

i.e. ① S is a vertex cover.

② $|S| \leq 2\,OPT$.

[Proof.] ① $\forall \{i,j\} \in E$, $x_i^* + x_j^* \geq 1 \Rightarrow x_i^* \geq \frac{1}{2}$ or $x_j^* \geq \frac{1}{2} \Rightarrow i \in S$ or $j \in S$

$\therefore S$ is a vertex cover

② $OPT = OPT(IP) \geq OPT(LP)$

$$OPT(LP) = \sum_{i=1}^{n} x_i^* = \sum_{j:\, x_j^* < \frac{1}{2}} x_j^* + \sum_{k:\, x_k^* \geq \frac{1}{2}} x_k^* \geq \frac{1}{2}\left|\left\{k \mid x_k^* \geq \frac{1}{2}\right\}\right| = \frac{1}{2}|S|$$

$\therefore |S| \leq 2\,OPT.$

Qed. □

▷ Max-SAT

CNF: $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ Variables $= \{x_1, \cdots x_n\}$ $|C_j| = \ell_j$

Find an assignment s.t. "most" clauses are **True**.

$C_j = x_{j1} \vee x_{j2} \vee \cdots \vee x_{j\ell_j}$ $x_{j_k} = a$ or "$\bar{a}$" ($a \in$ Variables)

assignment $\sigma \in \{0,1\}^{|V|}$ $v(\sigma) := \#$(clauses satisfied by σ)

Method 1. Uniformly assign each $X_i = 0$ or 1.

$$\mathbb{E}[v(\sigma)] = \mathbb{E}\left[\sum_{j=1}^{m} \mathbb{1}[C_j \text{ is satisfied by } \sigma]\right] = \sum_{j=1}^{m} \Pr[C_j \text{ is satisfied}]$$

$$= \sum_{j=1}^{m}\left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) = \sum_{j=1}^{m}\left(1 - \frac{1}{2^{\ell_j}}\right) = \sum_{j=1}^{m}\left(1 - 2^{-\ell_j}\right)$$

↑ 总有1种是UNSAT.

$$\geq \frac{1}{2}m \geq \frac{1}{2}OPT \qquad (OPT \leq m)$$

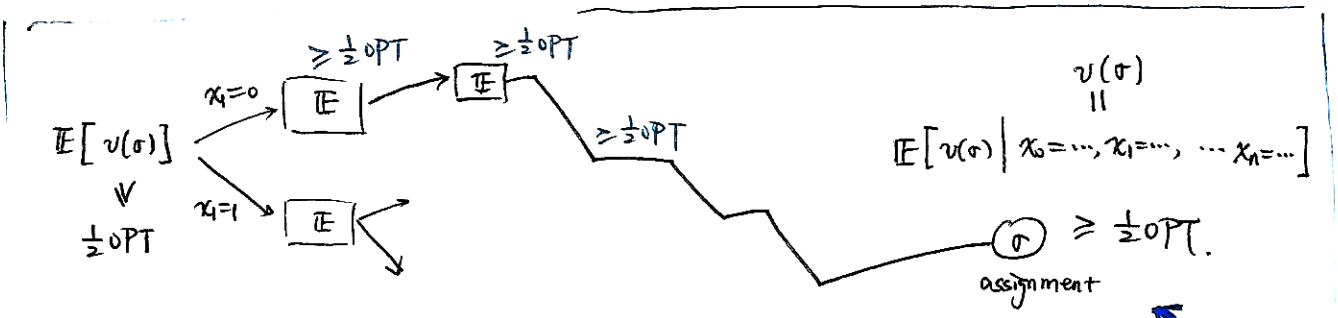
Method 2. Derandomization via Conditional Expectation

$$\frac{1}{2}OPT \leq \mathbb{E}_{\sigma}[v(\sigma)] = \mathbb{E}_{x_1}\left[\mathbb{E}_{\sigma}[v(\sigma) \mid x_1]\right] = \mathbb{E}_{\sigma}[v(\sigma) \mid x_1 = 0]\Pr[x_1 = 0] + \mathbb{E}_{\sigma}[v(\sigma) \mid x_1 = 1]\Pr[x_1 = 1]$$

$$= \frac{1}{2}\left[\mathbb{E}_{\sigma}[v(\sigma) \mid x_1 = 0] + \mathbb{E}_{\sigma}[v(\sigma) \mid x_1 = 1]\right] \longrightarrow \text{两者至少有一个比 } \tfrac{1}{2}OPT \text{ 大于等于.}$$

每次看一看 $\mathbb{E}_{\sigma}[v(\sigma) \mid x_1 = 0]$ 和 $\mathbb{E}_{\sigma}[v(\sigma) \mid x_1 = 1]$ 谁比较大.

e.g. $\mathbb{E}_{\sigma}[v(\sigma) \mid x_1 = 0]$ 比较大. 则 $x_1 \leftarrow 0$ ⇒ 一个确定 x_i ⇒ σ.



$\mathbb{E}_{\sigma}[v(\sigma) \mid x_1 = 0] \geq \frac{1}{2}OPT \Rightarrow x_1 = 0$. $\phi^{(1)} = \phi_{x \leftarrow 0}$

$\mathbb{E}_{\sigma \in \{0,1\}^{x_2, \cdots x_n}}[v(\sigma)] \geq \frac{1}{2}OPT$ (iterable)

转化为 ϕ' 上的问题

Method 3. $\frac{1}{2}$ is achieved when $l_j = 1$ in $1 - 2^{-l_j}$.

对单元素从句: $c_j = x_i$ $\Downarrow$ x_i 出现 (正出现) → 扔硬币 让 True 出现概率更高

$c_{j'} = \bar{x}_i$ $\Downarrow$ x_i 负出现 → $z_i := \bar{x}_i$. 要或 z_i 的正出现 → 扔硬币让 z_i = True 出现概率更高

若 x_i, $\bar{x}_i$ 都出现? → $\phi = x_i \wedge \bar{x}_i \wedge C'$ 最优也做不到. $OPT \le m-1$ 所以不管

Preprocess. $S = \{ i \mid$ both $x_i, \bar{x}_i$ appear as a singleton clause in $\phi \}$

$|S| = t \Rightarrow OPT \le m - t$

$\phi = \left[\bigwedge_{i \in S} (\bar{x}_i \wedge x_i) \right] \wedge C'$.

consider singleton in C'. $\bar{x}_k \Rightarrow z_k$ (其它地方也要替换) $\Rightarrow$ 得到 ϕ'.

~~$\forall$ singleton in C'.~~

For each variable z. toss a p-biased coin. $(p \ge \frac{1}{2})$ $z = \begin{cases} 1 & \text{w.p. } p \\ 0 & \text{w.p. } 1-p \end{cases}$

① singleton is satisfied w.p. $\ge p$.

② nonsingleton clauses in C', $c_j = \left(\bigvee_{i \in P_j} y_i \right) \vee \left(\bigvee_{k \in N_j} \bar{y}_k \right)$ (↑ 正出现, ↑ 负出现)

c_j is satisfied w.p. $\ge 1 - (1-p)^{|P_j|} p^{|N_j|} \ge 1 - p^{l_j} \ge 1 - p^2$

Worst Case: $|N_j| = l_j$ $(\because p < 1)$

使所有情况相接近? $p = 1 - p^2 \Rightarrow p = \frac{\sqrt{5}-1}{2} \approx 0.618$!

$$\mathbb{E}[v(\sigma)] \ge \underbrace{t}_{(\bar{x}_i \wedge x_i) \text{ 正好满足一半}} + \underbrace{\frac{(m-2t) \cdot p}{\text{其余 } (m-2t) \text{ 个 clause 被 SAT 的期望}}} \ge t + 0.618(OPT - t) \ge 0.618\, OPT.$$

□

Method 4. Method 3 中的 "Worst Case": $|N_j| = l_j$ ← "$\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{w}$" 就很烦

原因: 我们并不考虑 ϕ 中每个变量出现的 "结构". (每个 clause 中的具体结构)

每个变量之间并不等价. Method 3 用的依然是 uniform coin.

⇒ Nonuniform Coin!

$\forall i$. p_i: $\begin{cases} i \text{ 出现以 "正出现" 为主} & p_i \uparrow\uparrow \\ i \text{ 出现以 "负出现" 为主} & p_i \downarrow\downarrow \end{cases}$

How to determine such "~~factional costume~~" "costume" p_i? → LP!

Max SAT ⟷ Integer Programming

$\phi = c_1 \wedge c_2 \cdots \wedge c_m$

$\forall j \in [m]$ introduce z_j = "whether c_j is satisfied". ↘ indicator

$\forall i \in [n]$ introduce y_i = "$x_i = 1$". ↗ indicator

(IP)

$$\max \sum_{j=1}^{m} z_j \quad \text{Subject to} \quad \sum_{i \in P_j} y_i + \sum_{k \in N_j} (1 - y_k) \geq z_j \qquad \forall j \in [m]$$

$$y_i \in \{0,1\} \qquad \forall i \in [n]$$

$$z_j \in \{0,1\} \qquad \forall j \in [m]$$

⇓ Linear Programming Relaxation

$$\max \sum_{j=1}^{m} z_j \quad \text{s.t.} \quad \sum_{i \in P_j} y_i + \sum_{k \in N_j} (1 - y_k) \geq z_j \qquad \forall j \in [m]$$

$$y_i \in [0,1] \qquad \forall i \in [n]$$

$$z_j \in [0,1] \qquad \forall j \in [m]$$

$\{y_i^*\}, \{z_j^*\}$: optimal solution of (LP)

Rounding: for each variable x_i, toss a y_i^*-biased coin.

$$\Pr[c_j \text{ is UNSAT}] = \prod_{i \in P_j} (1 - y_i^*) \prod_{k \in N_j} y_k^* \leq \left[\frac{1}{l_j} \left(\sum_{i \in P_j} (1 - y_i^*) + \sum_{k \in N_j} y_k^* \right) \right]^{l_j}$$

$\frac{\sum x_i}{n} \geq \left(\prod x_i\right)^{\frac{1}{n}}$

$$= \left(\frac{1}{l_j} \left(l_j - \left(\sum_{i \in P_j} y_i^* + \sum_{k \in N_j} (1 - y_k^*) \right) \right) \right)^{l_j}$$

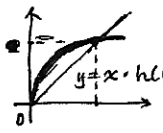
$$\leq \left(1 - \frac{z_j^*}{l_j}\right)^{l_j}$$

IV: $\sum_{i \in P_j} y_i^* + \sum_{k \in N_j} (1 - y_k^*) \geq z_j^*$

$h(z) = 1 - \left(1 - \frac{z}{l}\right)^l$

$h(0) = 0$. $h''(z) < 0$

$h(1) = 1 - \left(1 - \frac{1}{l}\right)^l$



凹函数.

$$OPT(IP) \leq OPT(LP) = \sum_{j=1}^{m} z_j^*$$

$$\mathbb{E}[v(\sigma)] \geq \sum_{j=1}^{m} 1 - \left(1 - \frac{z_j^*}{l_j}\right)^{l_j} \geq \sum_{j=1}^{m} z_j^* \left(1 - \left(1 - 1/l_j\right)^{l_j}\right)$$

$$\geq \min_{l_j} \left[1 - \left(1 - 1/l_j\right)^{l_j}\right] \cdot \sum_{j=1}^{m} z_j^*$$

$$\geq \left(1 - \frac{1}{e}\right) \cdot OPT$$

$(l_j \to \infty)$

Method 5. 发现: Method 3 worst case 在 $\ell_j = 1$. Method 4 worst case 在 $\ell_j = \infty$.

$\Rightarrow$ We choose the best of σ_1 (Given by M3) and σ_2 (Given by M4)

$$\mathbb{E}\left[\max\left\{v(\sigma_1), v(\sigma_2)\right\}\right] \geq \mathbb{E}\left[\frac{1}{2}\left(v(\sigma_1) + v(\sigma_2)\right)\right]$$

$$= \frac{1}{2}\mathbb{E}[v(\sigma_1)] + \frac{1}{2}\mathbb{E}[v(\sigma_2)].$$

$$\geq \frac{1}{2}\left[\sum_{j=1}^{m}\left(1-2^{-\ell_j}\right)z_j^* + \sum_{j=1}^{m} z_j^*\left(1-\left(1-1/\ell_j\right)^{\ell_j}\right)\right]$$

$$\geq \sum_{j=1}^{m} z_j^* \cdot \min_{\ell}\left[\frac{1}{2}\left(1-2^{-\ell}\right) + \left(1-\left(1-\frac{1}{\ell}\right)^{\ell}\right]\right.$$

$\leftarrow \ell = 2$

$$\geq 0.75\,\mathrm{OPT}$$

□