

算法期中复习

1. 记忆算法、公式正确性：数学归纳法、循环不变式

(1). 基础

归纳假设

演绎推理

(2). 初始化

保持

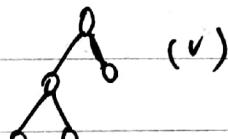
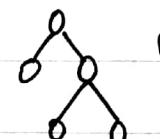
终止

\Rightarrow 引申：数学习式、算法：数学习归法（注： $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$ ）

算法：循环不变式

树：左右子树递归

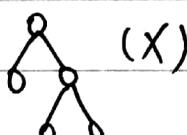
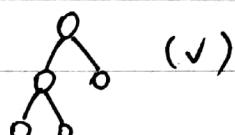
\Rightarrow PS: 2-tree: (v) 度=0 或 2 (v)



完美二叉树：



堆：



\Rightarrow PPS: 深度：最不根为0

高度：最叶为0

2. $O(g(n)) = \{f(n) : \exists C, n_0 > 0, 0 \leq f(n) \leq Cg(n) \text{ 对 } \forall n \geq n_0 \text{ 成立}\}$

$\Omega(g(n)) = \{f(n) : \forall C > 0, \exists n_0 > 0, 0 \leq f(n) < Cg(n) \text{ 对 } \forall n \geq n_0 \text{ 成立}\}$

$\Omega_2(g(n)) = \{f(n) : \exists C, n_0 > 0, 0 \leq cg(n) \leq f(n) \text{ 对 } \forall n \geq n_0 \text{ 成立}\}$

$\omega(g(n)) = \{f(n) : \forall C > 0, \exists n_0 > 0, 0 \leq g(n) < f(n) \text{ 对 } \forall n \geq n_0 \text{ 成立}\}$

$\Theta(g(n)) = \{f(n) : \exists C_1, C_2, n_0 > 0, 0 \leq C_1 g(n) \leq f(n) \leq C_2 g(n) \text{ 对 } \forall n \geq n_0 \text{ 成立}\}$

3. 复杂度排序：key：善于使用 \log 进行比较， $a^{\log_b c} = c^{\log_b a}$, $\log_b a = \frac{\log_c a}{\log_c b}$

$$(1). a^{\log_b c} = c^{\log_b a} : (\lg n)^{\lg n} = n^{\lg \lg n}, 4^{\lg n} = n^{\lg 4} = n^2 \text{ etc.}$$

$$(2) \text{ using logs: } (\sqrt{2})^{\log n} = \omega(2^{\sqrt{2 \log n}}), \log(\sqrt{2})^{\log n} = \log n \cdot \log \sqrt{2} = \frac{1}{2} \log n + \omega(\sqrt{2 \log n}) = \omega(\log^2 n)$$

$$2^{\sqrt{2 \log n}} = \omega(\log^2 n) : \log 2^{\sqrt{2 \log n}} = \sqrt{2 \log n} = \omega(2 \log \lg n) = \omega(\log \lg n)$$

$$(3). \text{ Stirling formula: } \log(n!) = \Theta(n \lg n) : \log n! \approx \log(\sqrt{2\pi n}) \left(\frac{n}{e}\right)^n = \Theta(n \lg n)$$

$$(4). \text{ Others: } \lg^*(g(n)) = (\lg^* n) - 1, \ln \ln n = \omega(2^{\lg^* n}) : \log_2 \lg^* n = \lg^* n, \log \ln \ln n = \omega(\lg^* n)$$

KOKUYO



由 扫描全能王 扫描创建

$$(5) e^n > 2^{n+1} = 2^n, 100^n > 10^n$$

$$(6). 2^{\frac{n+1}{n+1}} > 2^{\frac{2}{n}} > (n+1)! > n! > e^n > n \cdot 2^n > 2^n > (\frac{3}{2})^n$$

$$> (lg n)^{lg n} = n^{lg(n)} > (lg n)! > n^3 > n^2 = 4^{lg n} > n \lg n = lg(n!)$$

$$> n = 2^{lg n} > \sqrt{2}^{lg n} = \sqrt{n} > 2^{\sqrt{2 \lg n}} > \lg^2 n > \ln n > \sqrt{\lg n} > \ln \ln n$$

$$> 2^{\log^* n} > \log^* n = \lg^*(\lg n) > \lg(\lg^* n) > n^{lg^* n} = 2 = 1$$

4. 试解递归式

(1). 代数方法:

①. 猜测时间的形式成立, 然后数学归纳法证明, e.g. $\exists T(n) \leq c \cdot n$

② 替换变量法: e.g. $T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$

令 $m = \lg n$, 则 $T(2^m) = 2T(2^{\frac{m}{2}}) + m$ (考虑 \sqrt{n} 取整)

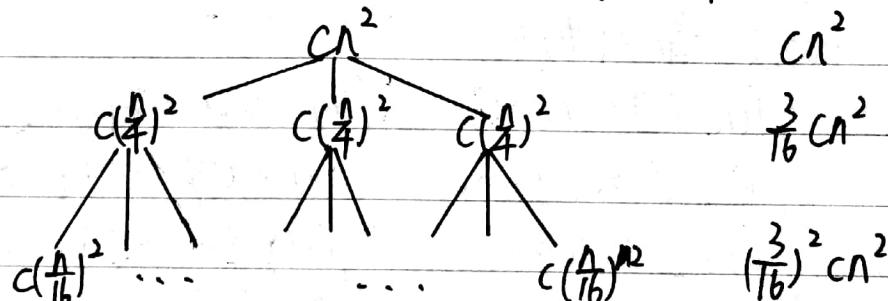
重命名 $S(m) = T(2^m)$ 且 $S(m) = 2S(\frac{m}{2}) + m$

$\therefore S(m) = O(m \lg m)$

$\therefore T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$

(2). 递归树方法:

①. 以 $T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + cn^2$ 为例, 假定 n 是 4 的幂.



$$\underbrace{T(1) \quad T(1) \quad \dots}_{3^{\log_4 n}} \quad \dots \quad T(1) \quad \Theta(n^{\log_4 3})$$

$$\therefore T(n) = cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \dots + (\frac{3}{16})^{\log_4 n} cn^2 = cn^2 + \Theta(n^{\log_4 3})$$

$$= \dots \quad (\text{去首项})$$

< ...

$$= \frac{16}{13}cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2)$$



不完美的树一定需要数归法证明.

$$\textcircled{1} \quad \text{以 } T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n) \text{ 为例}$$

\rightarrow 最长简单路径是 $n \rightarrow \frac{2}{3}n \rightarrow (\frac{2}{3})^2 n \rightarrow \dots \rightarrow 1$

$$\therefore h = \log_{\frac{2}{3}} n$$

\because 完全二叉树: 叶结点数 $\approx 2^{\log_2 n} = n^{\log_2 2}$

$$\therefore n^{1+\varepsilon} > n \log n$$

\therefore 叶结点代价总和 $\geq n \log n$

但叶结点缺失

\therefore 猜 $O(n \log n)$, 记 $T(n) \leq dn \log n$, 然后递归证明 (类比归纳法)

$$T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$$

$$\leq d\left(\frac{n}{3}\right) \log \frac{n}{3} + d\left(\frac{2n}{3}\right) \log \left(\frac{2n}{3}\right) + cn$$

$= \dots$

$$\leq dn \log n$$

12.

(3). 主定理: $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, $a \geq 1$, $b > 1$, $T(n)$ 定义于非负 N .

$\textcircled{1}$ 若 $\exists \varepsilon > 0$, $f(n) = O(n^{\log_b a - \varepsilon})$, 则 $T(n) = \Theta(n^{\log_b a}) \Rightarrow$ 多项式意义小

$\textcircled{2}$ 若 $f(n) = \Theta(n^{\log_b a})$, 则 $T(n) = \Theta(n^{\log_b a} \log n)$

$\textcircled{3}$ 若 $\exists \varepsilon > 0$, $f(n) = \Omega(n^{\log_b a + \varepsilon})$, 且 $\exists c < 1$, \forall 足够大 n 有 $af\left(\frac{n}{b}\right) \leq cf(n)$
则 $T(n) = \Theta(f(n)) \Rightarrow$ 多项式意义大且满足乙侧条件 $af\left(\frac{n}{b}\right) \leq cf(n)$

5. 数据库:

(1). $O(n \log n)$:

$\textcircled{1}$ BINARY-SEARCH(A[first..last], key)

二分查找(有序)

$\textcircled{2}$ 折半、二叉树、二进制的比特数 ... $\textcircled{3}$ FLEX-HEAP(A, p) 堆的修复将以

下标 p 为根的子树重新遵循大根堆的性质.

(2). $O(n)$:

$\textcircled{1}$ SEQUENTIAL-SEARCH(A[1..n], e)

遍历查找

$\textcircled{2}$ SELECT-MAX(A[1..n])

遍历找最大值.

$\textcircled{3}$ PARTITION(A, p, r)

快排序的划分

$\textcircled{4}$ MERGE(A, p, q, r)

归并中的合并

KOKUYO



由 扫描全能王 扫描创建

第KII

	<5> SELECT-ELINEAR (A, p, r, k)	期望从选择阶为k的元素 ($k-1$ 元素 $< p$, $n-k$ 元素 $> p$)
3.	<6> SELECT-WLINEAR (A, p, r, k)	最坏($O(n)$)选择阶为k的元素
	<7> TZX-HEAP (A, p)	(大根)堆的修复, 使以下标为根结点 而子树重新 遵循最大堆 而性质
	<8> CONSTRUCT-HEAP ($A[1..n]$)	(大根)堆的构建
(3).	$O(n \log n)$:	
	<1> MERGE-SORT (A, p, r)	归并排序}
	<2> QUICK-SORT (A, p, r)	快速排序} 从小到大
	<3> HEAP-SORT ($A[1..n]$)	堆排序}
	<4> Priority Queue Operations	优先队列操作
	INIT[ALIZE ($A[1..n]$)]	+ 初始化队列. $O(n)$
	GET-MAX (A)	+ 获得最大值. $O(1)$
	EXTRACT-MAX (A)	+ 提取出最大值. $O(n)$
	INCREASE-KEY (A, i, key)	+ 提高优先级. $O(\log n)$
	INSERT (A, key)	+ 插入元素. $O(\log n)$

(4). $O(n^2)$:

- <1> SELECTION-SORT ($A[1..n]$) 选择排序.
- <2> INSERTION-SORT ($A[1..n]$) 插入排序.
- <3> BUBBLE-SORT ($A[1..n]$) 冒泡排序

(5). 一些特殊的东西

<1> 并查集, n 为元素个数, l 为指令序列长度(并查指令)① 基于矩阵: $O(n^2)$ ② 基于数组: $O(nl)$

③ 基于根树:

+ UNION + FIND: $O(nl)$ + WEIGHTED-UNION + FIND: $O(n + l \log n)$ 

$\vdash \text{WEIGHTED-UNION + C-IND}: O((n+l) \log^*(n)) \approx O(n+l)$

<2> HASH 哈希表

① 封闭哈希(链接法): 不成功查找: $\Theta(1+\alpha)$

成功查找: $\Theta(1+\alpha)$

② 开放哈希(开放寻址法): 不成功查找: $\leq \frac{1}{1+\alpha}$

成功查找: $\leq \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

<3> 推中: $\text{PARENT}(i) = \lfloor \frac{i}{2} \rfloor$, $\text{LEFT}(i) = 2i$, $\text{RIGHT}(i) = 2i+1$

6. 二叉树

(1). 关键不等式 $n \leq L \leq m \rightarrow h \rightarrow$ 树高

(2). e.g. 比较排序算法最坏情况时间复杂度:

$$n! \leq L \leq 2^h$$

$n!$ 是排序结果全排列个数

2^h 是二叉树

$$\therefore h \geq \log(n!) = \Omega(n \log n)$$

7. 对称论证

基于比较的问题中, 构造工, 使 $T(A, L)$ 尽可能大, 情况尽可能坏, 然后对一切 A 取 $T(A, L)$ 尽可能小的下界

(1). Find 2nd Largest Number?

(2). Find Max & Min?

(3). Weighted Key?

(4). Matrix Search?

(5). 25 Horses Select 1st, 2nd, 3rd?

(6)



8. 平摊双柱分析.

(1). 聚合分析: 写通项公式和底值

(2). 核算法 (记账). Amortized Cost Actual Cost Accounting Cost

(3). Array Doubling? ① 聚合分析: $C_i = \begin{cases} i & i=2^k(k \in \mathbb{Z}) \\ 1 & \text{其它} \end{cases}$, $\sum_{i=1}^n C_i \leq n + \sum_{i=0}^{\lfloor \log_2 n \rfloor} 2^i < n + 2n = 3n$

② 核算法: Insert(normal) 3 1 2
Insert(doubling) 3 $k+1$ $-k+2$

(4). 二进制计数器? ① 聚合分析: $\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < \sum_{i=0}^{\infty} \frac{1}{2^i} = n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$, n 底值 $O(n)$, 1 次操作 $O(1)$

② 核算法: Set 1 2 1 1
Set 0 0 1 -1

(5). Stack? ① 聚合分析: 需用中文叙述, 行操作的底值 $O(n)$, 1 次操作 $O(1)$

② 核算法: Push 2 1 1
Pop 0 1 -1
Multi-pop 0 min(k, s), s 为当前栈中元素数 $\frac{1}{2} - \min(k, s)$

(6). Two Stacks - One Queue?

核算法: Enqueue 3 1 2
Dequeue(normal) 1 1 0
Dequeue(complex) 1 $1+2t$ $-2t$

(7). Two Queues, One Stack?

核算法: push 3 1 2
pop (normal) 1 1 0
pop (complex) 1 $2t-1$ $-2t+2$

push 不操作中, 1 代表 Actual Cost 对于 push, Accounting Cost 为 pop-1, push-1.

(8). 三明治问题. Push ~~without~~ Pop 2 1 1

~~Push~~ (cheap) \emptyset \emptyset \emptyset
Push with Pop (Expensive) 1 $k+1$ $-k$



$$9) T(n) = T(n-1) + n^c \quad (\text{类比}) \quad \frac{n}{2} \cdot \left(\frac{n}{2}\right)^c \leq 1^c + 2^c + \dots + (n-1)^c + n^c \leq n \cdot n^c$$

由 $\frac{n}{2} \cdot \left(\frac{n}{2}\right)^c \leq T(n) \leq n \cdot n^c$

$$\therefore T(n) = \Theta(n^{c+1})$$

(2). $\log(n!) = \Theta(n \log n)$ (类比) \Rightarrow Stirling formula 亦可证。

$$\because \log(n!) = \log(1 \cdot 2 \cdot 3 \cdot \dots \cdot n) = \log 1 + \log 2 + \dots + \log n$$

$$\therefore \frac{n}{2} \log \frac{n}{2} \leq \log n! \leq n \log n$$

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

10. 算法编写

(1). PARTITION(A, p, r).

$\text{pivot} = A[r]; \quad \text{pivot} = \text{SELECT}(A, p, r)$.

$$i = p - 1;$$

for $j := p$ to $r-1$ do

if $A[ij] < \text{pivot}$

$$i = i + 1;$$

SWAP($A[i], A[j]$);

SWAP($A[i+1], A[r]$);

return $i+1$;

11. 微数

$$(1). 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \ln(n+1) + \gamma = \Theta(\log n)$$

(2). \sqrt{n} : 想 2^m 替换

$$(3). T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n} : T(n) = \Theta(n \log \log n)$$

$$(4). \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$(5). \sum_{i=1}^k i \cdot 2^i = (k-1)2^{k+1} + 2$$



算法期末复习

1. MIT教会我的：

- (1). 对称记忆
- (2). 平权定理
- (3). Cut & Paste: 比如 DP, MST, SSSP, Floyd-Warshall
- (4). 极地: SSSP, Floyd-Warshall.
- (5). DP: ①最优化结构 ②重叠子问题(与递归、分治的区别)

2. DFS

- (1). 代价 $O(|V| + |E|)$

(2). 应用:

①活动区间: $v.\text{discoverTime}$, $v.\text{finishTime}$

②拓扑排序: Postorder 处插入拓扑序从大到小

V 依赖 W

③关键路径: if $w.eft \geq v.est$ then $v.est = w.eft$; $v.CritDep = w$ (① \rightarrow ②)

④SCC: $\text{DFS}(G)$, $\text{DFS}(G^T)$ with nodeStack (在 $\text{DFS}(G)$ 中后序处入栈)

⑤割点桥: $w.back \geq v.\text{discoverTime}$ 是割点(非根), $w.back > v.\text{discoverTime}$ 是桥

⑥TE回退时 $v.back = \min\{w.back, v.back\}$

⑦发现BE时 $v.back = \min\{v.back, w.\text{discoverTime}\}$

3. BFS

- (1). 代价 $O(|V| + |E|)$

(2). 应用

①无权图的单源最短距离 (比 Dijkstra 算法快, 代价 $O(|V| + |E|)$)

②判断二分图: 每次取邻居且白色就翻转二部图染色, 若邻居灰色且二部图同色则不可

③寻找k度子图: 先对图中度 $< k$ 的点加入队列, 若队列非空, 取队首元素删点及相邻边, 新图中若仍有度 $< k$ 的点且不在队列继续加入队列, 直至队列为空

4. MST

- (1). PRIM 算法: 关键: if neighbor w is UNSEEN then 设候选边; 加入候选队列
 $\quad \quad \quad$ else if $vw.\text{weight} < w.\text{priority}$ then 改候选边; DECREASE-KEY
 $\quad \quad \quad$ 复杂度 $O(|V| + |E| \log |V|)$

KOKUYO



由 扫描全能王 扫描创建

(2). KRUSKAL 算法: 并查集判环 $O(|E| \log |E|)$

5. SSSP 单源最短距离:

(1). Dijkstra 算法: 和 PRIM 很像, 区别是 PRIM: newWeight = vw.weight

$$\text{Dijkstra: newPriority} = v.\text{priority} + v.w.\text{weight}$$

(2). DAG 图上 SSSP. 先广拓扑序, 然后根据拓扑序顺序解决问题

LSSP 最长路径: 拓扑序 } 再广拓扑序解决.

MSSP 边权乘积最小值: 拓扑序

6. 传递闭包与 Floyd-Warshall

(1). 传递闭包: 当 R 在上一行循环中被更新, $i=1 \sim n, j=1 \sim n, k=1 \sim n; R_{ij} = R_{ij} \vee (R_{ik} \wedge R_{kj})$;

(2). Floyd-Warshall: $k=1 \sim n, i=1 \sim n, j=1 \sim n: D_{ij}^{(k)} = \min\{D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\}$

(3). min-max; max-min: \rightarrow 前驱 $G_O[i][j] = G_O[k][j]$, 后继 $G_O[i][k] = G_O[i][j]$

① 最小吞吐率的最大值: $cap_{ij}^{(k)} = \max\{cap_{ij}^{(k-1)}, \min\{cap_{ik}^{(k-1)}, cap_{kj}^{(k-1)}\}\}$

7. DP

(1). 斜降链相乘: 关键: for low = n-1 to 1 do \Rightarrow low high 之间递归断开.

for high = low+1 to n do

for k = low+1 to high-1 do

$$\text{bestCost} = \min\{\text{bestCost}, cost[low][k] + cost[k][high] +$$

(2). 最优汉密尔顿树: 关键: low high 之间选 root 作根, 同理, 很像 $P_{low} P_{high}$

(3). 编码算距离: $0 \ 1 \ 2 \ 3 \ \dots \ m \leftarrow$ 初始化

$$\text{递归: } DP[i][j] = \min \begin{cases} DP[i][j-1] + 1 \\ DP[i-j][j-1] + \begin{cases} 1 & \text{if } A[i][j] = B[i][j] \\ 0 & \text{if } A[i][j] \neq B[i][j] \end{cases} \end{cases}$$

(4). 破币兑换问题及各种变式

$$\text{coin}[i][j] = \text{coin}[i-1][j] \vee (\text{coin}[i][j-x_i] \wedge j \geq x_i)$$

① 面值 n 种, 硬币无限, 最少兑换: $\text{coin}[i][j] \leftarrow$ 前 i 种面值兑换 j 最少使用几枚硬币

$$\text{coin}[i][j] = \text{coin}[i-1][j] \vee (\text{coin}[i-1][j-x_i] \wedge j \geq x_i)$$

② 面值 n 种, 硬币每种 ≤ 1, 不能兑换: $\text{coin}[i][j] \leftarrow$ 前 i 种面值不能兑换 j

③ 面值 n 种, 硬币每种 ≤ k, 不能兑换: $\text{coin}[i][j] \leftarrow$ 前 i 种硬币不能兑换 j

$$\text{coin}[i][j] = \begin{cases} \text{true} & \text{false} \\ \text{dp}[i-1][j-x_k] = \text{true} \& i < k \& j - x_k \geq 0 \\ \text{otherwise} & \end{cases}$$

Campus

$$\text{或 } \text{coin}[i][j][1] = \text{coin}[i-1][j][1] \vee (\text{coin}[i][j-x_i][1] \wedge w \geq x_i)$$



由 扫描全能王 扫描创建

(5) $LCS[i]$ 题及各种变式

① 最大和连续子序列: $sum[i]$ 表示以结尾的连续子序列的最大和

$$sum[i] = \max \begin{cases} A[i] & \text{if } A[i] \text{ 在最大和连续子序列中} \\ sum[i-1] + A[i] & \text{if } A[i] \text{ 不在最大和连续子序列中} \end{cases}$$

② LCS: $L[i, j] = \begin{cases} L[i+1, j-1] + 1 & X[i] == Y[j] \\ \max\{L[i+1, j], L[i, j-1]\}, & X[i] \neq Y[j] \end{cases}$

③ X 可重叠 Y 不可重叠的 LCS: $L[i, j] = \begin{cases} L[i+1, j-1] + 1 & X[i] == Y[j] \\ \max\{L[i+1, j], L[i, j-1]\}, & X[i] \neq Y[j] \end{cases}$

④ X 为多重 k 次, Y 不可重叠的 LCS: $L[i, j, l] = \begin{cases} L[i+1, j-1, l-1] + 1 & X[i] == Y[j] \& l < k \\ \max\{-\infty\}, & \text{else} \end{cases}$

(6). 最长连续前后向子序列: $L[i, j]$ 表示 LCS 以 T_i 开头 T_j 结尾的长度最大值

$$L[i, j] = \begin{cases} 0 & \text{if } T_i \neq T_j \\ \max\{L[i+1, j-1] + 1, 1\} & \text{if } T_i = T_j \end{cases}$$

(7). 最长回文子序列 $L[i, j]$ 表示从 i 到 j 的最长回文子序列

$$L[i, j] = \begin{cases} L[i+1, j-1] + 2 & \text{if } S[i] = S[j] \\ \max\{L[i+1, j], L[i, j-1]\} & \text{if } S[i] \neq S[j] \end{cases}$$

$$\text{Init: } \begin{cases} L[i, i] = 1 \\ L[i, i+1] = \begin{cases} 2 & \text{if } S[i] = S[i+1] \\ 0 & \text{if } S[i] \neq S[i+1] \end{cases} \end{cases}$$

(8). 序列最坏拆分回文次数: 从 $i \sim j$ 中间选 k 分, $C[i, j] = \begin{cases} 0 & S[i, \dots, j] \text{ 本身就是回文} \\ \min\{C[i, k-1] + 1 + C[k, j]\} & \text{else} \end{cases}$

(9). 树的最小顶点覆盖: $\begin{cases} dp[i][0] = \min\{dp[k][0], dp[k][1]\}; \\ dp[i][1] += dp[k][0]; \end{cases}$
最后取 $\min\{dp[\text{root}][0], dp[\text{root}][1]\}$ 即可。



8. NP

(1). 优化问题、判定问题

(2). P是NPC, 证明Q也是NPC? 将P归约到Q

(3). 一般与特例: 若已知特例P为NPC, 将一般P'进行某些具体设定即可将P'归约到P从而一般问题P'为NPC

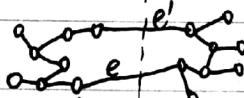
(4). 等价问题: 若已知P为NPC, P和PP等价也可记得P'为NPC

(5). 実在不清楚关系时一定要想着可以画图帮助理解简单.

9. 食心.

(1). MST \rightarrow MCE框架:

Cut



(2). SSSP \rightarrow BestFS框架:



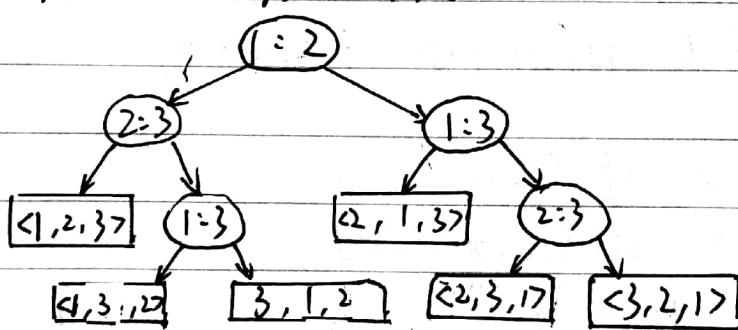
Fresh \Rightarrow Fringe (调度器) \Rightarrow Finished

(3). 其他应用.

① 相容任务调度: 选择任务结束时间最早的贪心

② Huffman编码: 将频率最低的两个字符换成它们编码的父节点, 递归编码
新字符串集

10. 3个元素比较排序的决策树



11. Cut Property \Rightarrow 跨越Cut最近的 \in MST \rightarrow 沿此Cut的边最轻的, 加上它可构造MST

Cycle Property \Rightarrow 属于Cycle中最重的 \notin MST \rightarrow 沿此Cycle的边最重的, 删去它可构造MST

12. DFS图结构.

①. 白、灰、黑

②. TE, CE, BE, DE

③. v. discoverTime, v. finishTime, TOPO, SCC

Campus



由 扫描全能王 扫描创建

(4).

	TE	BE	CE	DE
DFS 有向图	✓	✓	✓	✓
无向图	✓	✓	✗	✗ ✗
BFS 有向图	✓	✓	✗ ✓	✗
无向图	✓	✗	✓	✗

(5).

	有向图	无向图
DFS	$BE \Leftrightarrow Cycle$	$BE \Leftrightarrow Cycle$
BFS	$BE \Rightarrow Cycle$	$CE \Leftrightarrow Cycle$

但 $Cycle \Rightarrow BE$ 或 CE

13. DP 问题的结构

(1). 1维: Array, Sequence, String . 子问题: 前缀, 后缀

(2). 2维: ① $X[1, \dots, m], Y[1, \dots, n]$, 子问题: $X[i, \dots, j], Y[i, \dots, j]$ 由 ② $X[1, \dots, n]$, 子问题: $X[i, \dots, j]$

(3). 3维: Floyd-Warshall

(4). 图: ① 根树 T 子问题: 根的子树

② 有向图 G 子问题: 拓扑序当中前面、后面的点

(5). 背包: 子问题: 子集, etc.



由 扫描全能王 扫描创建