



算法导论问题解答(二)

NANJING UNIVERSITY

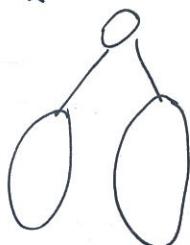
PRCNU CN

17-3. 解：

- a. ①用中序遍历的方法求出该子树对键元素的顺序，存储于数组中 $O(n)$
 ②选用中位数作为根，以之为划分数组为两部分，对每部分递归
 构建树 $f(n) = 2f(\frac{n}{2}) + 1 \quad O(n)$

$$\begin{aligned} c_i &= c_i + \text{至}(D_i) - \text{至}(D_{i-1}) \\ &= c_i - \text{至}(D_{i-1}) \quad (\text{讲义知重建 } m \text{ 经典子树需 } O(m)) \\ &= m - \text{至}(D_{i-1}) \end{aligned}$$

要使 $c_i \in O(1)$, 需令 $\text{至}(D_{i-1}) = O(m)$ 即可。即 m 个单位的努力够
 支付重建 m 个经典子树的代价。
 要求 c 的最小值，我们应该先重建 $\sum_{x \in T} O(n)$ 最小的情况。
 如图，假设右子树破坏了 2 平衡。?



$$\begin{aligned} \text{故有 } \sum \Delta(x) &= (x.\text{right.size} - x.\text{left.size}) \\ &> 2m - m + 1 + 2m \\ &= 2m - m + 1 \end{aligned}$$

$$\text{则 } \text{至}(l) = C \times (2m - m + 1) \in O(m)$$

设方程 $C \times (2m - m + 1) > m \Rightarrow C > \frac{m}{2m - m + 1} > \frac{1}{2}$

- e. ①插入或删除的过程的复杂度与查找的复杂度相同，为 $O(\lg n)$
 ②出现一棵子树的重建代价的摊还代价为 $O(1)$ ，而重建需要
 要重建的子树从根一直到插入或删除的结点，最多 $O(\lg n)$ ，
 故其复杂度（根结点）为 $O(\lg n)$

17-4.

- a. 若一棵红黑树红黑相间，即根为黑， \lceil 层为红， \lfloor 层为黑，则按照情况1，将一直出现红色节往上升的情况，几(g_m)
(2)若一棵红黑树为全黑，按照删除时的情况2，将一直有双层黑的
结点上升一层，也为几(g_m)

c. 由d知，插入和情况2、3引起的结构性修改的操作还代价为 $O(1)$ ，
而对于情况1，势减少了1而已。每个单位的势足以支付任意一种
情况引起的结构性修改，故操作还代价为0。

d. 对于图13-4情况1，5从红 \rightarrow 黑+红，势从0 \rightarrow 0不变。

8从红 \rightarrow 黑+红，势从0 \rightarrow 1 1)

7从黑+红 \rightarrow 红，势从2 \rightarrow 0 -2，

故总的势减少1，由于每个单位的势足以支付结构性修改的
代价，故情况1的操作还代价为0。

对于情况2和3，结构性修改的次数为 $O(1)$

故总为 $O(1)$

e. 对于图13-4情况2，D从黑+红 \rightarrow 红，势从1 \rightarrow 0 -1

~~若B为黑，且B从红 \rightarrow 黑+红，势从0 \rightarrow 0~~
若B为黑，且B从红 \rightarrow 黑+红，势从0 \rightarrow 0

若B为黑，且B从黑+红 \rightarrow 黑+红，势从0 \rightarrow 0 -1

故总的势将减1或减2，同上可知操作还代价为0。

总为 $O(1)$

f. 由上知每个操作的操作还代价为 $O(1)$ 。



18.1-3. 解：由 $h \leq \log_2 \frac{n+1}{2} = \log_2 \frac{6}{2}$, 得 $h \leq 1$ 故只有两层

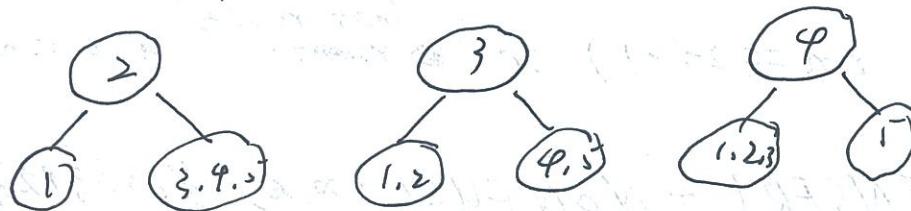
(1) 若根结点有 2 个孩子，则其有 3 个孩子。

② 因 $t=2$, 则每个结点至少有 $(t-1)=1$ 个关键结点，至多有 $(t-1)=3$ 个

(2) 若根结点有 3 个关键结点，则其有 3 个孩子，只有一种情况。



(2) 若根结点有 1 个关键结点，则其有 2 个孩子。



18.1-5. 解

① 每个黑结点吸收红孩子后，将只有 1, 2, 3 个孩子

② 吸收红孩子后，原红孩子的左子孩子将按 $x_i \leq x_j \cdot \text{key}_L$

③ 顺序插入划分

③ 每棵红黑树的根结点到叶结点 ~~路径~~ 的黑高度相等。故吸收后各路径高度相等

综上，为一棵 $t=2$ 的 B 树，亦即 2-3-4 树。

18.2-3. 解：

(1) 寻找关键字，一直向左找到最左的叶子结点，其第一个元素即为所求。

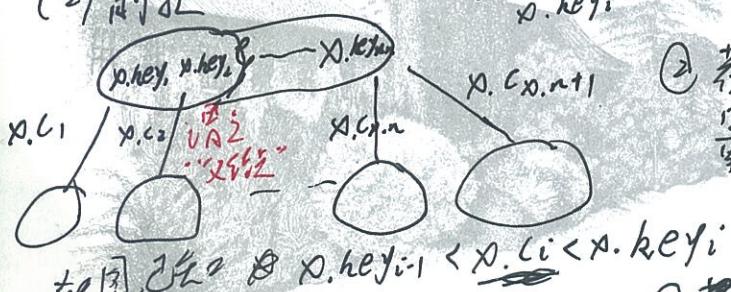
设 $x_h =$
 x_{key_i}

(2) 前驱

① 若 x_h 不在叶结点中，则找出 x_{i-1} ; 其最后一个结点即为所求

② 若 x_h 在叶结点且为最左的元素，若该叶结点为最左的叶结点，则 x_h 为前驱，否则前驱在

~~x_{key_i}~~ 是对应父结点的左结点



9. Hankou Road Nanjing China

③ otherwise, x_h 的前驱即为 Cable:0909
它的左结点。

18.2-5. 解：设叶结点的大小为 t' ，

~~由于每一个内结点都必须由叶结点分裂而来，(在最近顶端的根处)~~
~~而一个满叶结点要变成完全分离成两个非满叶结点，故代码修改~~
改如下：

① P282 B-TREE-SPLLI-CULLD(x, i) 第 1 行 ~~if~~ 前加
if $x.i.leaf$
 $t = t'$

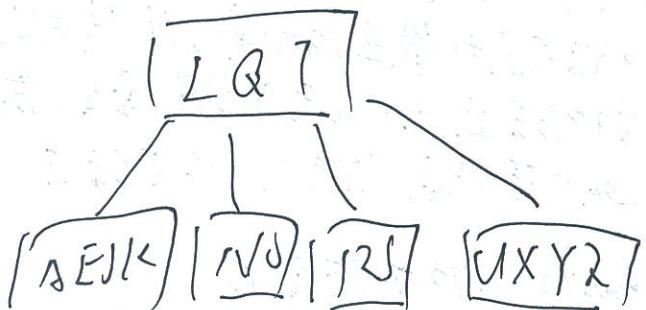
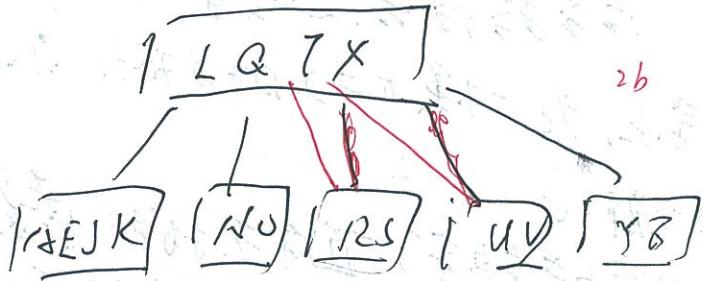
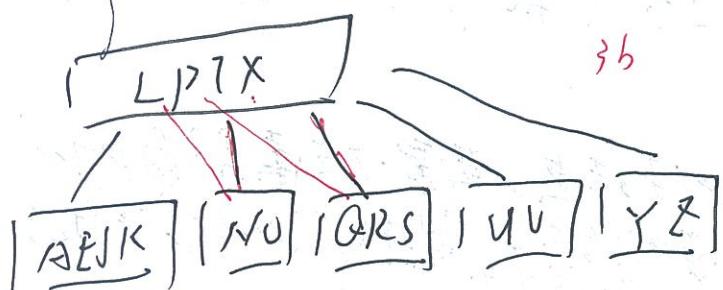
② P283 B-TREE-INSERT(l, h) 第 2 行 改为

~~if ($x.leaf$ and $r.n == 2t' - 1$) or ~~not~~ $x.leaf$ and $r.n == 2t - 1$~~

③ P284 B-TREE-INSERT - NONFULL(x, n) 第 13 行 ~~if~~ 改为

if $(x.ci.leaf \text{ and } n.ci.n == 2t' - 1) \text{ or } (\text{not } x.ci.leaf \text{ and } x.ci.n == 2t - 1)$

18.3-1 解





18-1. 解.

b. 磁盘存取: $\lceil \frac{n}{m} \rceil$ 次.CPU 时间 $\lceil \frac{n}{m} \rceil \times \Theta(m) = \Theta(\frac{n}{m})$ 次

c. 最坏情况就是, 对每一个元素的前面进行

然后: pop, pop, push, push

进 读 退 读

共 2 次读和 2 次写入,

磁盘存取:

故 CPU = $\Theta(m^2)$ d. 读入一页, 当发现操作位于 $\frac{2}{3}m$ 时, 读入下一页 (若已经到末尾)
 $\leq \frac{1}{3}m$ 时, 读入上一页平均而言: 每 m 次操作读 $\frac{1}{2}$ 次, 故平均磁盘存取时稍稍有
差异.时间为 $\Theta(1/m)$, CPU 时间为 $\Theta(1)$
从初始点: push → 读入下一页 push $\frac{1}{2}m$ → 读入下一页pop 是写 $\frac{1}{2}m$ → 读入上一页 pop $\frac{1}{2}m$ → 读入上一页

18-2. 解:

a. 插入时, 只有根的分裂才会增加高度, 故在插入时, 由根分裂出的那个结之后
为原根高度加 1, 其它时候不变. (即高度从下往上 (相对深度))

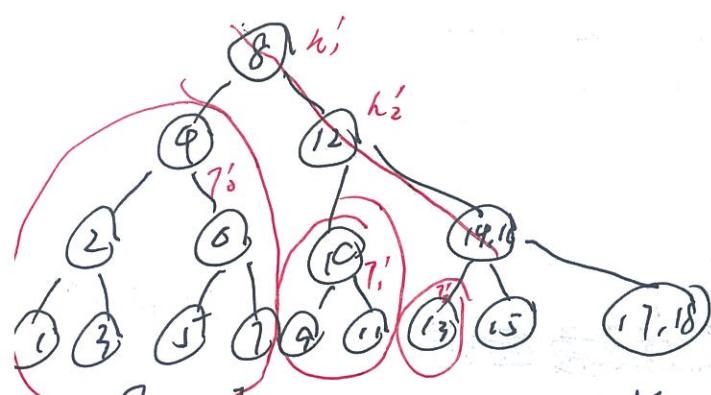
删除时, 只有根的根会并才会减少高度, 故当会并根的子结之时, 令根

消失, 其余结之高度不变.

b. 在插入操作, 从根向下寻找的路径中, 将满的结之分裂, 一直到找到
“高度为 h” (但是设 h' > h") 的最右的结, 往其父结之插入, 把
“以其以为新丁的根”

“ l' ” 分别作为 x 的左, 右孩子

c.

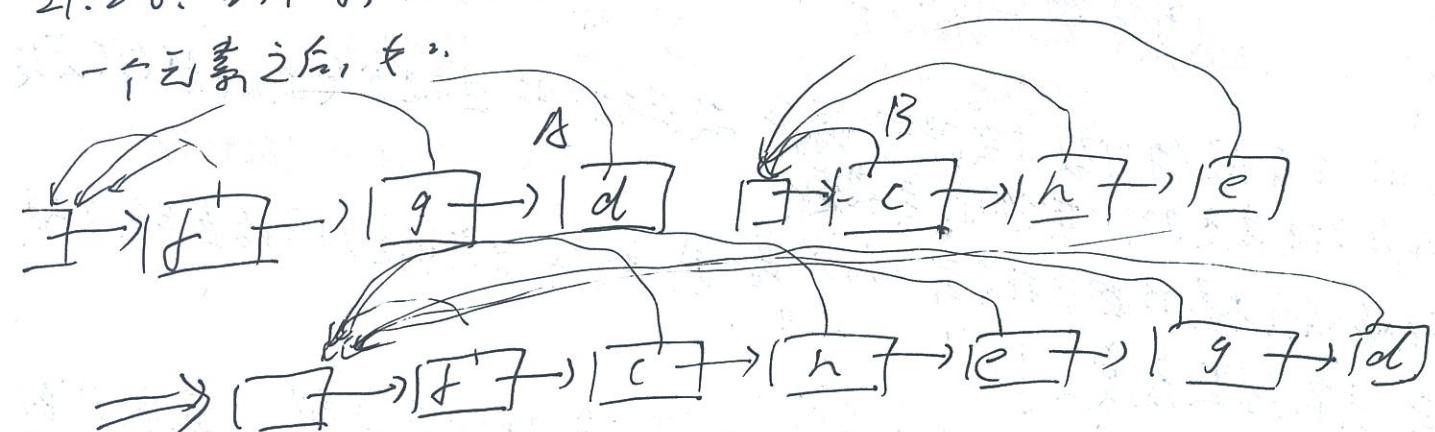


如果找寻 14 的过程中，若将 S' 划分为 T_0', T_1', T_2' 和 h_1', h_2'

易得 $h(T_0') = h(T_1') - 1$ 。(如果从 88 → 12 → 100%? $h(T_0') \leq h(T_{i-1}') - 1$ 吗)

在 T_2-T_4 中查找 x ，设其经过的路径为 h ，则由 c
可知其划分。然后我们以 h_1', h_2' 一等为关键字，依次
合并 T_0', T_1', T_2' 一。由于树的高度为 $\log_2 n = \lg n$ ，每次
连接由 b 知 \oplus 复杂度为 $O(1f(h'-h'')) = O(1+1) = O(1)$ ，故总
复杂度为 $O(\lg n)$

21.2-6. 合并时，假设 B 并入 A 。我们采取把 B 粘贴到 A 的第
一个元素之后， \Rightarrow



21.3-3. 解：① 执行 $n \times \frac{n}{2}$ make-set

② 按 $\text{union}(1, 2), \text{union}(3, 4), \dots, \text{union}(n-1, n)$
 $\text{union}(1, 3), \text{union}(5, 7), \dots, \text{union}(n-3, n-1)$

$\text{union}(\textcircled{1}, \frac{n}{2})$

构造一棵树 才执行 $\frac{n(n+1)}{2}$

③ 剩余 $(m - 2n + 1)$ 次操作 执行 find 操作



NANJING UNIVERSITY

PRCNU CN

则 find 操作作为 $\mathcal{O}((m-n)\lg n)$, 当 m=n 时, 使得 $\mathcal{O}(n \lg n)$ 时,
复杂度为 $\mathcal{O}(m \lg n)$

21.3-4. 每个结点新增一个 link, 指向其在集合的下一个顶点(每行) ~~整个~~ 集合
构成一个循环链表 (初始 $x.link = \emptyset$)

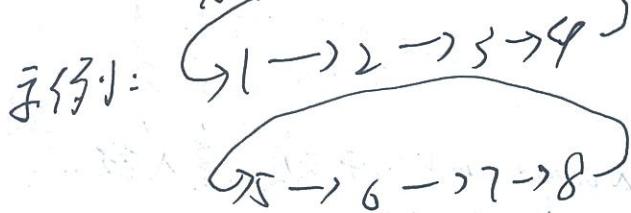
② 合并两个根时, 进行如下操作: (交换 x, y 的 link)

$$temp = y.link$$

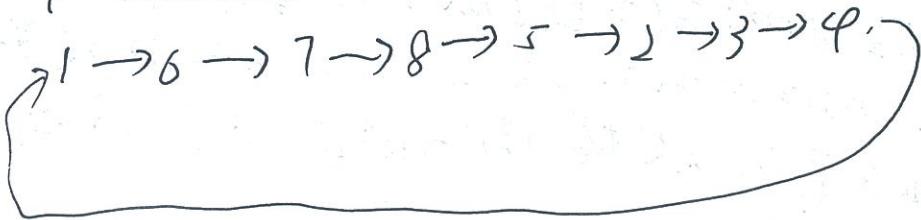
$$y.link = x.link$$

$$x.link = temp$$

即即即, 以 x 一直遍历即可



结果:



21.3-5. 解: 使用秩算法.

① make-set 花费 1 美元, 1 美元用于创建本身, 另一美元在 find-set 的
路径压缩过程中, 用于与根相连

② union-set 花费 1 美元, 用于两个结点相连

③ find-set 花费 1 美元, 用于访问结点和孩子, 主要于在查找路径的
结点, 前边 make-set 已经

故其 $O(m)$.

由于上述过程并没有用到按秩合并, 故都为 $O(m)$.

9. Hankou Road Nanjing China

Cable: 0909

21-1 解：程序对题给集合的运行过程：

	K_1	K_2	K_3	K_4	K_5	K_6	K_7	extracted
0	{9,8}	{3}	{9,2}	{1}	{}	{1,7}	{5}	19 2 3 4 5 6
1	{9,8}	{3}	{9,2,6}	{1}	{}	{1,5,7}		
2	{9,8}	{3}	{9,2,6}	{1}	{}	{1,5,7}	2	
3	{9,8}		{9,2,6,3}	{1}	{}	{1,5,7}	3	
4			{9,2,6,3,8}	{1}	{}	{1,5,7}	4	
5			{9,2,6,3,9,8}	{1}	{}	{1,5,7}	4	
6				{9,2,6,3,4}	{1,5,7}		4 3 2 6	
7						{1,5,7}		
8							{1,5,7}	
							4 3 2 6 8	
								组

掌握

1. ① 对于 $n=1$, 其处在它之后的 extract-min 中被填入数值,
此即为第 j 次 extract-min ($\Theta(j \leq k)$)
② 取出 $n=1$ 后, 此时总的数字减少 1, 合并 k_j 与 K_L , 相当于移除
第 j 次 extract-min, 故 extract-min 的次数也减少 1,
相当于一次规模减小的过程 (取出最小数填入
extract 数组)

由归纳性, 算法正确

2. ① 用构建不相交集合森林, 其中每棵树根的根指向所属集合
而 k_j 位于一个链表中, 其亦有一指针指向根

② 第二行程序 \leftrightarrow find

第 5 行 \leftrightarrow 求 $k_j \rightarrow next$

第 6 行 \leftrightarrow ~~link~~, 在链表中删除 k_j

复杂度: $n \times make-set$

至 $(n-1) \times link$ (union)

$make-set$

放高的操作次数 $m = O(n)$

复杂度为 $O(m \times (n)) = O(n^2)$



21-2解: b. MAKE-TREE(v)

~~$v.p = v$~~

$p.rank = 0$

$p.d = 0$

c. 如图,假设边的权值保持性质



$$\begin{cases} h_{v_1} = v_1.d \\ h_{v_2} = v_1.d + v_2.d \\ h_{v_3} = v_1.d + v_2.d + v_3.d \\ h_{v_4} = v_1.d + \dots + v_4.d \end{cases}$$

要使压缩后仍保持该性质,需 \leftarrow

$v_3.d = v_2.d + v_3.d, v_4.d = v_2.d + v_3.d + v_4.d.$

实现 $MAKE$:

FLND-RCGT(v)

~~if $htv \neq htv'$~~ ~~if $v.p \neq v'.p.p$~~

$y = v.p$

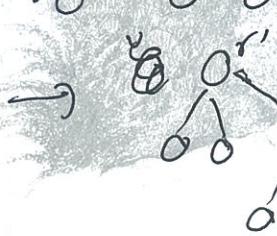
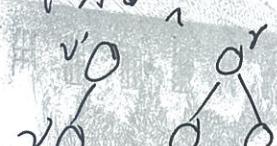
 $v.p = FLND-RCGT(y)$

$v.d = v.d + y.d$

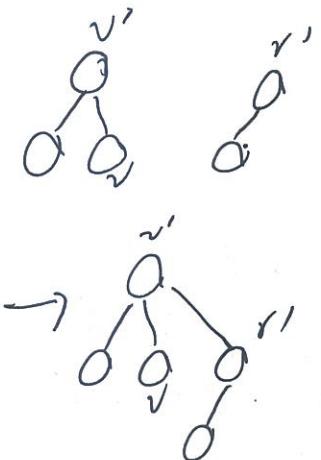
return $v.p$

FLND-DEPTH(v)

FLND-Root(v) //不需返回值

if $v == p.v$ return $v.d$ else return $v.d + v.p.d$ d. 我们要注意,在森林中, $GRAFT(r, v)$ 是使得 r 连接在 v 下,而我们为了实现按秩合并,有时需将 $v.root$ 连在 r 下。当 $r.rank > v.rank$ 时(r, v 分别为 r, v 的根),如图,要使连接后仍保持性质,我们知道原来以 r 为根的树的所有结点由于连在 v 下,故深度都得增加 $r.d$ 。那么,需令 $r.d = r.d + 1$,就可实现这点。然而,此时以 v 为根的树的所有结点深度都增加了 $r.d$,若需令 $v.d = v.d - r.d$,

② 若 $r'.rank < v'.rank$, 同上。



需有 $r'.d = r'.d + z + 1 - v'.d$, 其它不变, 代码如下:

$GRAFT(r, v)$

$r' = \text{FIND-ROOT}(r)$

$v' = \text{FIND-ROOT}(v)$

$z = \text{FIND-DEPTH}(v)$

if $r'.rank > v'.rank$

$\quad \quad \quad r'.p = r'$

$\quad \quad \quad r'.d = r'.d + z + 1$

$\quad \quad \quad v'.d = v'.d - r'.d$

$else$

$\quad \quad \quad r'.p = v'$

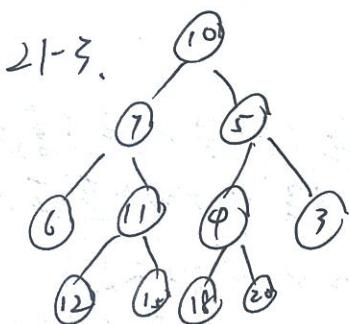
$\quad \quad \quad r'.d = r'.d + z + 1 - v'.d$

$\quad \quad \quad \text{if } r'.rank = v'.rank$

$\quad \quad \quad r'.rank = v'.rank + 1$

(c) 由于这 m 次操作可化为
make-tree, link, find-set, 以及

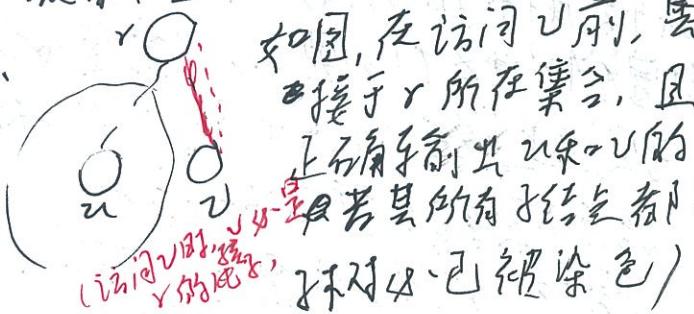
复杂度 $O(\alpha(mn))$



- a. 假设 v 会被访问两次, 第一个访问时, 其中一个祖先尚未被访问过, 为白色, 第二个不会执行。故只执行一次
b. 算法执行时, 当访问某结点某子树根结点, 其会创建一个集合, 然后递归访问其一棵子树根结点, 又创建一个集合, 访问完该子树返回后, 其会连接于上层根结点集合, 即每往下一层, 集合数加1, 返回上一层, 集合数减1, 故易用递归证明 访问 v 时, 集合数等于 v 的深度

- c. 如图, 在访问 v 前, 其祖先 ~~未染色~~ 已经连接于 v 所在集合, 且 v 已被染色, 故肯定上角输出 v 和 v 的 LCA (假设 ~~该树不存在~~ 若其所有子结点都已被访问, 则 v 的所在

感觉让
明的
事：





d. 阐释① MAKE-SET 运行 $\text{O}(n)$

② FIND-SET 运行 ~~$O(n+P)$~~ \downarrow $O(n+P)$

③ UNION 运行 $O(n)$

P 为在第 10 行里
运行次数

共 $O(n+P)$ MAKE-SET, FIND-SET, UNION,
复杂度为 $O((n+P) \times \alpha(n))$

22.1-3 解：

① 对于邻接链表来说，可以在遍历某一个节点的 Adj 时，将 i 增加到其余遇到的每个之后的 Adj 中，复杂度 $O(V+E)$ 不会混乱吗？觉得重新构建一个新的是邻接链表更好

② 对于邻接矩阵，需要设置矩阵 $O(V \times V)$

它的位置不是 $O(\text{常数})$

22.1-4 解：

设置与 V 的数目相等的数组 visited ，每次访问某一个顶点时的邻接顶点时，把该数组都置为 false，置该元素对应位置为 true。然后，依次访问该顶点的邻接顶点，每遇到一个顶点检查一下 visited ，若为 false，则置为 true，并将其加入 G' ，否则跳过。

复杂度： $O(V+E)$

22.1-5 解：

① 对于邻接链表来说，对于 u 的所有邻接顶点 v ，遍历 v 的邻接顶点，此即为 E^2 的边。由于可能重复，故需对新生成的链表作去重处理。（因为可能 $u \rightarrow v$ 有多条路径）

复杂度：①遍历生成新的链表：由于在遍历 $2n$ 个邻接顶点时，再遍历这些顶点的邻接顶点时，最多有 $|V|V|$ 个，故是 $O(VE)$ 。

② 约 $O(V+E)$

约是 $O(VE)$

②对于邻接矩阵，又需求矩阵的平方即有，复杂度为 $O(V^3)$ (用 Strassen 算法)
 可提高到 $O(V^{19})$

$$a_{ik} \neq 1 \Rightarrow a_{ij} \times a_{jk} = 1 \quad (k=1, 2 - |V|)$$

22.1-6. 解：列举出关于汇点的几个性质

- ① 若 $a_{ij} = 1$, 则 j 不可能为汇点 (出度不为0)
- ② 若第 j 行除了 a_{jj} 之外存在0, 则 j 不可能为汇点 (入度不为 $|V|-1$)
- ③ 若某 $a_{jj} = 1$, 则 j 不可能为汇点 (出度不为0)
- ④ 一个图至多只能存在一个汇点 (除该点外, 其它顶点的出度不为0).

(1) 判断某顶点 v 是否为汇点的算法：

- ① 检查第 v 行看是否存在1
- ② 检查第 v 列看是否存在除了 a_{vv} 不存在0

①②都为否定则为汇点

IS-SINK(A, v)

let A be $|V| \times |V|$

for $j = 1$ to $|V|$

if $a_{vj} \neq 1$

return false

易知复杂度 $O(V)$

for $i = 1$ to V

if $a_{ii} = 0$ and $i \neq v$

return false

return true.

(2) 完整算法：

UNIVERSAL-SINK(A)

let A be $|V| \times |V|$

$i=j=1$
 $i \leq |V|$ and $j \leq |V|$

while $i \leq |V|$ and $j \leq |V|$

if $a_{ij} \neq 1$

~~$i = i + 1$~~

else $j = j + 1$

$s = 0$

if $i > |V|$

return "no sink"

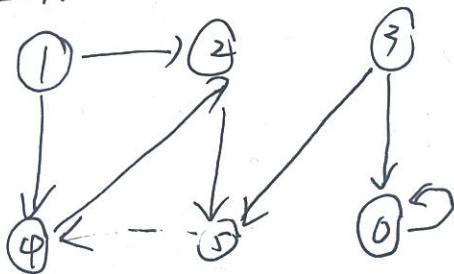
elseif IS-SINK(A, i) = false

return "no sink"

else return i "is a sink".



证明：



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	0	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

之所以再检查这部分
是由于

如图虚线为算法运行过程。

① 当遇到 $a_{ij} = 1$ 时，显然 j 为分支点。

② 当算法终止时 $i > |V|$ ，显然不存在分支。

③ 当算法终止时 $i \leq |V|$ ，易知对任何 $i+1 \sim i-1$ ，由①知 i 不为分支点。

故只需证 $(i+1) \sim |V|$ 不可能为分支点。
观察上图，当 $a_{ij} = 0$ 时，我们向左走。而显然 $j \geq i$ (这里甲我们从右往左走)。
可以预处理矩阵，扫描对角线，凡是出现 1 的必不为分支点，故当 $a_{ij} = 0$ 且 $j > i$ 时，由 $a_{ij} = 0$ ，知 $i \rightarrow j$ 无边，故 j 不可能为分支点，故当 $a_{ij} = 0$ 且 $j \leq i$ 时，向右的同时，^{且 $j < i$} 基本也隐含了 j 不为分支点。则当循环结束而 $i \leq |V|$ 时，对于 $(i+1) \sim |V|$ ，其必存在某 $i' h \leq i$ ，有 $a_{ki'} = 0$ ，
故 i' 不可能为分支点，分支点只能为 i 。
<sup>因为 i, h 对应 e_j 的所有顶点 (with $i < h$)
 i, h 相等，都对应于 e_j 的边 (i=h)
且该式 = $i \times 1 - 1$</sup>

22.1.

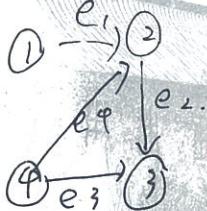
$$\text{令 } C = BB^T,$$

$$= b_{ij} \times b_{kj} \quad (j \text{ 为 } e_i \text{ 和 } e_k \text{ 的交点})$$

$$\text{则 } C_{ih} = \sum b_{ij} \times b_{kh} \quad (i \rightarrow e_j \times e_j \rightarrow h)$$

故当 $i=h$ 时， C_{ih} 即为顶点 i 的度；

当 $i \neq h$ 时， C_{ih} 表示 i 与 h 之间的边数。



$$\begin{matrix} & c_1 & c_2 & c_3 & c_4 \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & -1 & 0 & 1 \\ 3 & 0 & 1 & 1 & 0 \\ 4 & 0 & 0 & -1 & -1 \end{pmatrix} & \rightarrow & \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix} \end{matrix}$$

22.1-8. 解：

① 预期时间是 $O(1)$

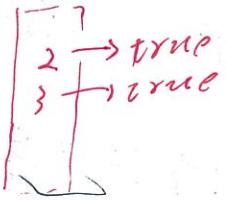
② 缺点是需要更多的空间

③ 可以用二叉搜索树，将搜索时间降为 $O(\deg(v))$ $\xrightarrow{\text{假设}} \xrightarrow{\text{2-3}}$
缺点：实现较复杂，搜索时间也不如 Hash 表

22.2-6 解：



如图，支线部分不能由 BFS
得出。



22.2-7. 解

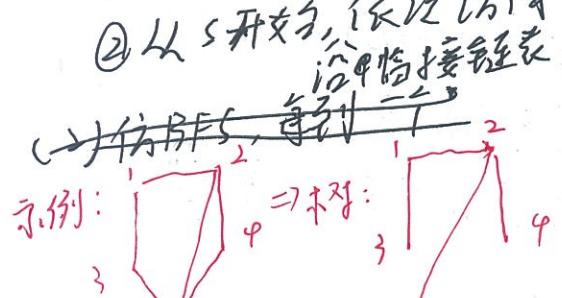
仿 BFS 算法，从任意一个结点开始，初始令其涂为红色。依次遍历其直接邻接顶点，其涂为黑色。以后每~~开始~~一个结点的邻接顶点 v 时，若 v 已访问，则检查 u, v 颜色是否相同，相同则终止算法；若 v 未访问，则将 v 涂为与 u 不同的颜色

22.2-8. 解：

进行两次 BFS。第一次 BFS 后，找出 d 最大的某结点，从该结点开始再进行 BFS，得出的新距离即为所求。
如何证明正确性：

22.2-9. 解

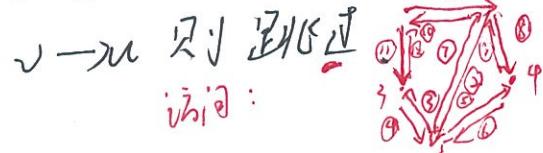
(1) ① 用 BFS 生成一棵广度优先树，记录下每个结点的前驱结点
② 从 s 开始，依次访问



- ① 若 (u, v) 在 ~~BFS~~ 原始图中，而不在 BFS 树中
 访问 v 并返回 u
 说明树是 $u \rightarrow v$
- ② 若 (u, v) 在 BFS 树中，且 v 不为 u 的前驱，
 则继续访问 v
- ③ 若 (u, v) 在 BFS 树中，且 v 为 u 的前驱，
 访问完①②后
 返回 u

示例：

注意，返回前驱前，记录该结点已访问，下次遇到中直接边
 ①访问④时，~~直接~~ $s \rightarrow 2 \rightarrow 4$ ，访问过，故不进行这一点





22.3-1. 角度：

有向图：

	白	灰	黑
白	空	BC	C
灰	TF	TFB	TFC
黑		B	空

无向图

	白	灰	黑
白	TB	TB	
灰	TB	TB	TB
黑		TB	TB

22.3-8 角度：



和图，假设以 s 开始，
访问 u 和 v 。

22.3-10 角度：

 DFS-VISIT(G, u)

time = time + 1

u.d = time

u.color = GRAY

 for each $v \in G$.Adj[u] do

 if v .color == white

print "(u, v) is a tree edge"

 $v.\pi = u$

 DFS-VISIT(G, v)

 else if v .color == gray

print "(u, v) is a Back edge"

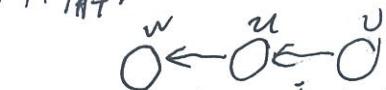
 } else if $v.d > u.d$

print "(u, v) is a forward edge"

else print "(u, v) is a cross edge"

当为无向图时，这一部分不会被执行，可以去掉也可以掉（只能出现 v .color == 黑的情况）

22.3-11 解：



如上，先访问 w ，再 v ，再 u 则 uv 将成为深度优先树的惟一路径。

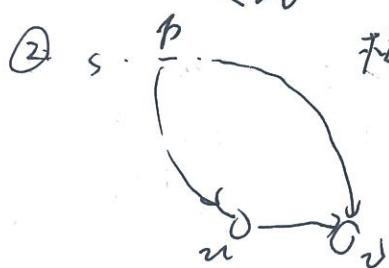
亦即若对于有出边和入边的结点 v ，我们从 v 出边的一端开始访问，再访问 v ，再访问 v 入边的一端，将出现这种情况。

22.3-13 角度：在某些 DFS 中，

(1) 若图中出现前向或横向的边，则该图不为单连通图。



前向边是在 uv 一路访问到 w 后，退回 u 后又发现了一条 $u \rightarrow v$ 的边，显然此图不为单连通。

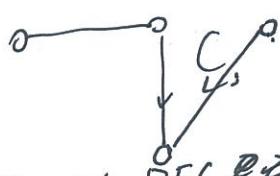


横向边是在 uv 一路访问到 v 后，退回 u 后，又访问到 v ，此时发现 $u \rightarrow v$ 这条边，即为横向边。故 uv 有两条路径。

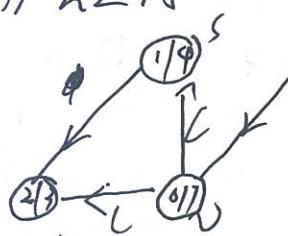
(2) 算法：对每个

(2) 在有的图从某顶点进行 DFS 可能出现多棵森林，

树与树之间的横向边不能说明图不为单连通。



(3) 只运行一次 DFS 也无法确定图是否为单连通。
如图，从 s 开始访问，无法判断。



(4) 算法：对每个顶点运行一次 DFS，时间复杂度 $O(V^2(V+E))$ 。



22.4-2. 解答：

$\text{SIMPLE-PATHS}(u, v)$

if $u = v$

 return 1

else if $u.\text{paths} \neq \text{NIL}$

 return $u.\text{paths}$

else

 for each $w \in \text{adj}[u]$

$u.\text{paths} += \text{SIMPLE-PATHS}(w, v)$

复杂度 $O(V+E)$

return $u.\text{paths}$.

22.4-3. 解答：

一个无向图无强连通仅当图中无后向边（即没有本环边）。

算法：运行DFS，若出现后向边，则含环路，否则无环。

由一个无环图必有 $|E| \leq |V| - 1$ ，故不论图无环，有环。

算法扫描遍历过的边数，至多 $|V|$

22.4-4. 解答：

错误。因为按照书中算法，从不同起始点开始可能生成不同数目的环。

如：



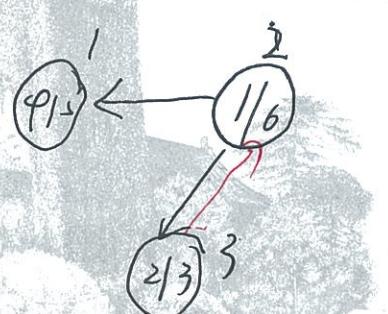
从 a 开始，序列为 abc 该边：c → a

从 b 开始，序列为 bca 该边：a → b, c → a.

则对 G 运行算法 将得到 {3, 2, 1}

为同一强连通分量

22.5-3. 解答：



22.5-4 解：

任取 $e(v_i, v_j) \in ((G^T)^{SCC})^T$, 则 $(v_j, v_i) \in (G^T)^{SCC}$, 即即存在
 $\alpha \in C_j, \beta \in C_i$, 使得 $(\alpha, \beta) \in G^T$, 则 $(\beta, \alpha) \in G$, 故
 $(v_i, v_j) \in G^{SCC}$ 。反之亦可由 $(v_i, v_j) \in G^{SCC} \rightarrow (v_j, v_i) \in ((G^T)^{SCC})^T$

22.5-5 解： 算法

- ① 运行强连通量，生成一组强连通分量 $O(V+E)$
- ② 对每行强连通分量标号为 $1 \sim h$, 令同一分量的所有顶点对该标号叫
- ③ 扫描所有边，删除所有连接不同分量的边，用这些边所在分量标号
(u, v) 标记将被边，加入某子集 S 中 $O(E)$
- ④ 对该集合进行星数排序， $O(E)$
- ⑤ 扫描集合，每次遇到某元素与前一元素不同时，取出单来，所给即为所求。

22.5-6. 解： 满足 强连通

所求 G' 应该

- ① 每个分量只包含一个环
- ② 两个分量之间只含一条边

算法：

- ① 仿 22.5-5 得到一个分量图，任意两结点间多一条边 $O(V+E)$
- ② 令 $E' = \emptyset$, 依次扫描每个分量 ③ 设某分量所有顶点为 v_1, v_2, \dots, v_h
则往 E' 加入 $v_1 \rightarrow v_2, v_2 \rightarrow v_3, \dots, v_h \rightarrow v_1$ $O(V)$
- ④ 对 G^{SCC} 的每条边，从这条边所属的两个分量任取两个顶点 (x, y) ,
并加入 E' $O(E)$

22.5-7 解：

(1) 证明 G 为单连通且仅当其分量图含一条且经过多个分量的链

(我们可以通过形而的分量图求拓扑排序) 属于

① 完备性：由于各个分量内的顶点两两可达，故对任意两个分量的
两个顶点 u, v , 由于其处于一条链上，故必单向 $u \rightarrow v$ 或
 $v \rightarrow u$ 成立。



必要性：假设无法构成一条链，则拓扑排序后的图

必有相邻两结点 v_i, v_{i+1} 使得 $v_i \rightarrow v_{i+1}$ 不存在。

由于拓扑性质，知 v_i 无法反向到某个 $v_h (h < i)$ ，

再由 $v_h \rightarrow v_{i+1}$ ，也不存在 $v_i \rightarrow v'_h \rightarrow v_{i+1} (h' > i+1)$ 。

故 v_i, v_{i+1} 不连通，与图为半连通矛盾。

算法：(1) 行 22.5 得出 G^{SCC} 的分量图 $G^{\text{SCC}}(V+E)$

(2) 对 G^{SCC} 进行拓扑排序 $\Theta(V+E)$

(3) 假设 G^{SCC} 有 k 个分量，拓扑序排序后形成

v_1, v_2, \dots, v_k

依次检验 $v_1 \rightarrow v_2, v_2 \rightarrow v_3, \dots, v_{k-1} \rightarrow v_k$ ，
若都存在边，则图为半连通的。

复杂度： $\Theta(V+E)$

22-1. 解：

a. 1. 若 (u, v) 为前向边或后向边，则其中一个顶点为另一个顶点的祖先，
假设 u 为 v 祖先，则访问 u 时， $u \rightarrow v$ 将被访问，为对边，矛盾。

3. 若 (u, v) 为横向边，则访问 u 时， v 未已在队列，否则 (u, v)

为不对边。由于 v 在队列， $v.d \leq u.d + 1$ 。

b. 1. 若 (u, v) 为前向边，则 v 为 u 的后代，显然访问 u 时 $u \rightarrow v$ 将
被访问。

3. 同 a. 3.

22.4. 角解:

算法: 求 G 的轻置 G' , 按 $L(u)$ 递增的顺序~~按 G~~ 调用DFS。

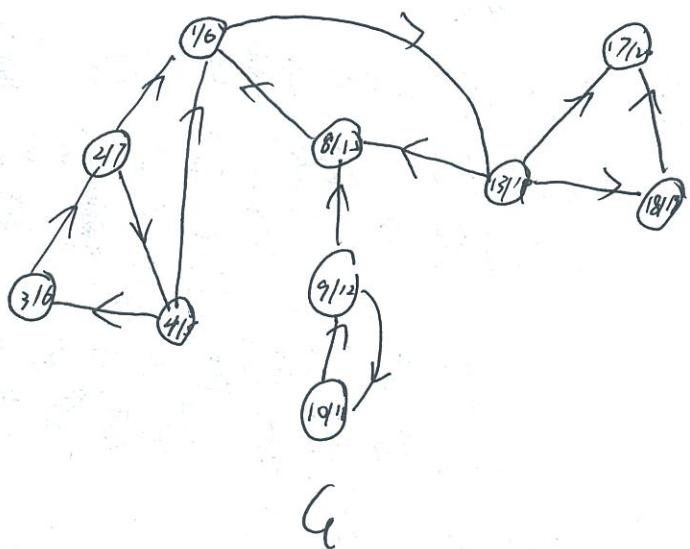
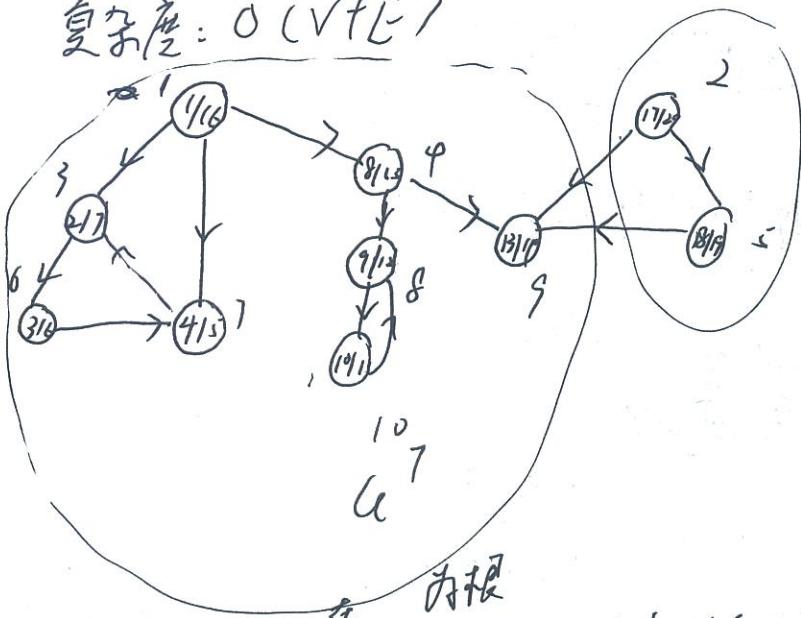
~~若 G 有环~~

则得到深度优先搜索森林。只任取一棵。

~~其根在以 v 为根~~

其根为 v , 则这棵树中的任何 u , 有 $\min(u) = v$.

复杂度: $O(VL)$



证明: 由于 u 在 v 的DFS树中, 故 $v \sim u$, 又 $v \in u$ 。

假设 $v \neq \min(u)$, 即 $v > \min(u)$, 设 $w = \min(u)$ 。

由于 $w < v$, 故 $w \neq v$, 而 v 为其所在DFS树的根, 故 $w \not\sim v$,

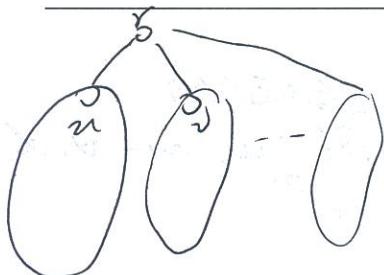
由于 $w \sim v$ 访问. 则 $w.f < v.d$. 又 u 在以 v 为根的DFS树中,
故 $v.d \leq u.d < u.f \leq v.f$, 而由于 $w \sim u$, 故而 u 在 w
的子树中, 故 u 后于 w 访问, 则 w 访问 w 时必将访问到 u ,
矛盾, 故 $v = \min(u)$.

22.2 解.

a. 必要性: 根结点是衔接点 \rightarrow 至少有两个子结点

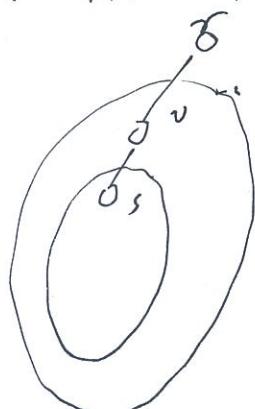
假设 x 有一个子结点, 则删除掉 x 后, 其由 x 结点
可到达图任意一个结点, 故图连通, 与 x 为衔接点矛盾。

必要性: 至少有两个子结点 $\rightarrow x$ 为衔接点。



和图，去除掉 s 后， v 和 u 相连通，只在访问 v 时，会沿着连通的边访问 v 所属分量，矛盾。

b. 充分性： v 子结点 s 且无后向边 $\Rightarrow v$ 为链行接点。



由于 s 和 s 后代无指向 v 的祖先的后向边，则去除掉 s 后 v 所属部分将与其它部分不连通，故 v 为链行接点。

必要性： v 为链行接点 \Rightarrow v 子结点 s 且无后向边。

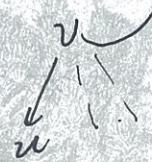
- ①若 v 无子结点，则去除掉 v 后图依然连通。
- ②假设没有后向边连向 v 的祖先，则去除掉 v 后图连通。

c. 递归地计算 $v.\text{low}$ ：(假设 (v, w) 为一条后向边)

$$\text{①若 } v \text{ 为叶结点, 则 } v.\text{low} = \min \left\{ \begin{array}{l} v.\text{d} \\ w.\text{d} \end{array} \right.$$



$$\text{②若 } v \text{ 不为叶结点, 则 } v.\text{low} = \min \left\{ \begin{array}{l} v.\text{d} \\ w.\text{d} \end{array} \right. \text{ } u.\text{low} \text{ (所有与 } v \text{ 直接或 } w \text{ 是 } v \text{ 的孩子)}$$



由上可知，算法运行次数与所有的度数成

正比，又度数之和为 9. HanKou Road Nanjing China

两倍边，故为 $O(E)$ 。

Cable:0909

d. ① 若 u 为根，则检查其是否有两个孩子
 ② 若 v 为根，检查是否有孩子 u ，且满足

$$u.\text{low} \geq v.\text{d} \quad (\text{当 } u \rightarrow v \text{ 有后向边时}, u.\text{low} = v.\text{d} \text{ 且后代无后向边})$$

$$\text{否则 } u.\text{low} = u.\text{d} > v.\text{d}$$

(这样需要检查所有 v 的孩子)

(或者：① 检查是否有孩子

② 检查 $v.\text{low} = v.\text{d}$)

f. 解法一：桥易知。桥 uv 为木对边。则某桥 (u, v) ，其中 v 为 u 的孩子。由桥 uv 不属于 G 的任何简单环路，知

$\begin{array}{c} u \\ \swarrow \quad \downarrow \quad \searrow \\ v \end{array}$ v 的后代 u 处于经 v 指向 u 的祖先的后向边，故 $v.\text{low} = v.\text{d} > u.\text{d}$ 。
故云需要检查每对木对边是否满足
 $v.\text{low} > u.\text{d}$ 即可。 (u, v)

解法二：对于桥 (u, v) ，首先， u, v 两端点都为一度顶点，
 否则图不连通。设 u 至少有 2 度以上，则若

$\begin{array}{c} u \\ \swarrow \quad \searrow \\ v \end{array}$ u 不为平行接点，去除掉 u 后， u 所属分量 u 与 v 之间边连通，构成环路，矛盾。
故 u 为平行接点，故 (u, v) 为桥 $\Leftrightarrow (u, v)$ 中一个为平行接点，一个为一度点，或者 u, v 都为平行接点。

故由 d 求出所有平行接点后，检查每个平行接点的边的另一端是否为平行接点或一度顶点即可。

g. 假设边 (u, v) 处于两个连通分量 C, C' 中，取 $(a, b) \in C, (c, d) \in C'$ ，
 则有 ~~$ab - hub - a, cd - hub' - c$~~ ，
 $ab - hub' - dc - v'v' - a$ ，故 ab, cd 处于同一环路中，

则 ab, cd 处于同一连通分量，矛盾。



NANJING UNIVERSITY

PRCNU CN

h. 先运行计算法, 计算出所有桥. 之后.

22-5.

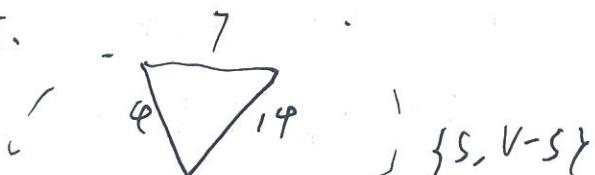


9. HanKou Road Nanjing China

Cable:0909

23.1-4. 解：取一个三条边权重相等的三角形图，~~使其任意~~
一条边是一条轻量级边，~~但由于有环，不是MST.~~

23.1-5.



(反证法我觉得更简单)

如图，任意取一个划分，使上述环的顶点一部分处于 S ，一部分处于 $V-S$ 。
易知不论怎样划分，最大的边都不会成为一条轻量级边。故在
构建最小生成树的时候一定可以避开这条边。

23.1-6. 任取两棵 MST T, T' ，设 $(u, v) \in T$ ，去掉 (u, v) 后， T 不连通，故 (u, v) 破可连成一个子树割 $(S, V-S)$ 。令 T' 中至连通 $(S, V-S)$ 的边为 (x, y) ，~~由其虽然也有贝沙姆~~

23.1-3，知 (x, y) 也为轻量级边。又 $(S, V-S)$ 的轻量级边唯一，故 $(u, v) = (x, y)$ ，依此类推， $T = T'$ 反例：+

补23.1-3. 令 $(u, v) \in T$ ， T 为某棵 MST。要在 T 中除掉 (u, v) 后， T 将分为两棵树 T_0, T_1 ，~~这两棵树的顶点构成一个子树割~~
 $(S, V-S)$ 。若存在 $(x, y) < (u, v)$ ，且 (x, y) ~~连通~~ 横跨 $(S, V-S)$ ，
 T_1 在 T_0 加入 (x, y) ，所得树将更短，矛盾，故 (u, v) 为
轻量级边。

23.1-7. ①显然该边集会使图连通的
若存在环，则可以去掉该环，使得权重减少，故无环。

②若有环，则可以去掉该环，使得权重减少，故无环。
故所得为树。（任意两个顶点之间存在唯一一条路径，则 G 为树）

反例1：





23.1-8. 假令 L, L' , L, L' 分别为 L, L' 的边权重的有序列表。

假设 L, L' 中第一对不同的边分别为 $(u, v), (u', v')$, 其为第 i 对边, 其中, $(u, v) \leq (u', v')$. 注意权重不同

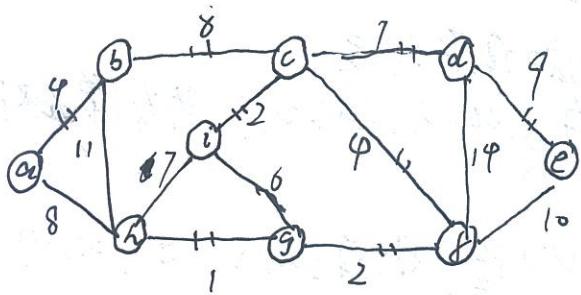
①若 L' 含 (u, v) , 则 (u, v) 位于 (u', v') 后, 因为 $(u', v') \in (u, v)$ 故 $(u, v) = (u', v')$, 我们把 L' 中 (u, v) 和 (u', v') 互换位置, 则此时 L', L 至少前 i 对边相同, 令算法再次运行找下一对不同边

②若 L' 不含 (u, v) , 则 L' 中必含 $u \rightarrow v$ 的另一条路径, 设在 L' 中加入 (u, v) 将串形成弦 C , 则易知弦 C 的任意一条边长度 $\leq (u, v)$. 设 L' 中去除掉 (u, v) 后形成树对 T_1, T_2 , 其亦为两个连通分量。

~~如果 L' 中没顶点集会形成一个切割 $(S, V-S)$~~ , 易知 $(x, y) = (u, v)$ 则在 L' 中 $u \rightarrow v$ 的路径里 ~~必含某条边连通~~ $(S, V-S)$, 易知 $(x, y) = (u, v)$ 由 $(x, y) \in L'$, 其要么 ~~即~~ 即为 (u', v') , 要么在 (u', v') 之后, 无论哪一种, ~~必有~~ $(x, y) = (u', v') \in (u, v)$. 若 (x, y) 在 (u', v') 之后, 我们令 (x, y) 与 (u', v') 交换位置, 并在 L 中用 (x, y) 替换 (u, v) , 此时得到的新的 L 依然为 MS1 且 L 与 L' 前 i 对边相等, 且路径 (x, y) 即为 (u, v) , 用 (u', v') 代替 L 中的 (u, v) . 然后, 我们继续执行算法找下一对不同边.

由①②, 算法运行到结束时, L 将与 L' 完全相同, 故 L 和 L' 之初始时必含有相同的边权重.

手写:



去掉 (b, c) 换成 (a, h) 依然为一棵 MST.

$L: \{ (g, h), (g, f), (i, c), (a, b), (c, f), (c, d), (b, e), (d, e) \}$

$L': \{ (g, h), (i, c), (g, f), (c, f), (a, b), (c, d), (a, h), (d, e) \}$

23.1-10 设 T 不为 MST, 设新的 MST 为 T' , 设减少了权重的边为 (x, y) 则 $w(T') < w(T) - k$. ~~由于 T' 不可能含 (x, y)~~ 由于 T' 不可能含 (x, y) 也不可能不含, 故 $w(T') \leq w(T) + k < w(T)$, 与 T 为原图 MST 矛盾.

23.1-11 设减少了权重的边为 (u, v) . 则在原树 T 中存在 $u \rightarrow v$ 的路径, 若往 T 中加入 (u, v) , 则将形成环(考察 e), 若 (u, v) 是 $T - (u, v)$ 中的最长边且短, 取代使之, 否则不变.

23.2-2. 读数 α . 构造一个数组 A , 其中 $A[u] = (v, w)$, $v \in T$, $w = (u, v)$.

PRIM-ADJ(G, w, r)

```

 $T = \{r\}$ 
for  $i = 1$  to  $V$ 
  if  $Adj[r, i] \neq 0$ 
     $A[i] = (r, w(r, i))$ .

```

```

for  $i = 1$  to  $V-1$ 
   $h = \min_i A[i]$ 
   $T = T \cup \{h\}$ 
   $h, z = A[h].1$ 
  for  $i = 1$  to  $V$ 

```

if $Adj[h, i] \neq 0$ and $Adj[h, i] < A[i]$
 $A[i] = (h, Adj[h, i])$

即初始化化数组 A 后, 对每次扫描数组指出最小边, 将顶点加入 T 后, 扫描 V 对应矩阵那一行, 更新最小边值, 复杂度 $(V-1) \times V = O(V^2)$



23.2.3 最小堆: $O(EgV)$, 堆波那契堆: $O(E+VgV)$

当 $E = w(V)$ 时, $f_1 = w(V)gV$, $f_2 = w(V)+VgV$

若 $E < w(V) > VgV$, 则 $f_1 = w(V)gV$, $f_2 = w(V)$, f_1 更大.

若 $w(V) < VgV$, 则 $f_1 = w(V)gV$, $f_2 = VgV$, f_2 更大.

综上, 当 $E = w(V)$ 时, 堆波那契堆的实现更优.

23.2.4

① 基数排序 E 条边, 未耗时 $O(E+V) = O(E)$

② 其它操作 $O(E2(V))$

若 $O(E2(V))$

当 权重在 $1 \sim W$ 时,

① 基数排序 E 条边, $O(E+W) = O(E)$ (W 为常数)

② $O(E2(V))$

依然为 $O(E2(V))$
依然为 $O(E2(V))$ 且法复杂度 $\left\{ \begin{array}{l} \text{① 处理 } V \text{ 个顶点 } O(V) \\ \text{② EXTRACT-MIN } V \times O(V \times gV) \\ \text{③ DECREASE-KEY } O(V \times gV) \end{array} \right.$

23.2.5. 已知 prim 算法复杂度 $O(Wt + Vt)$, 其中 t 为时间常数
① 当范围在 $1 \sim W$ 时, 我们可以构建一个数组 $Q[0 \sim W+1]$, 其中 $Q[w+1]$ 表示权为无限的边, 其它用来存储对应权的循环链表. 则 EXTRACT-MIN 的复杂度为 $O(1)$, DECREASE-KEY 的复杂度为 $O(1)$ (只需将 $W+1$ 的复杂度为 $O(W) = O(1)$, DECREASE-KEY 的复杂度为 $O(V+E) = O(E)$)

对应边从一个槽移到另一个槽 (即复杂度 $O(V+E) = O(E)$)
从第 1 个开始, 指到第一个非空的链表

② 当权重在 $1 \sim V$ 时, 上述算法复杂度为 $O(V \times V) = O(V^2)$ (EXTRACT-MIN 复杂度为 $O(V)$), 此时, 若采用 Fibonacci 树, 复杂度为 $O(E+VgV)$, 算法要更优. 若采用 vEB 树, 每个操作复杂度为 $lg(lgV)$, 则总复杂度为 $O((E+V)lg(lgV)) = O(Elg(lgV))$

23.2-6.

我们来理清 Kruskal 和 Prim 算法 复杂度的决定因素

(1) Kruskal

① 快排 $O(E \lg E)$

② $\left\{ \begin{array}{l} \text{MAKE } \cancel{\text{SET}} \cancel{\text{V}} O(V) \\ \text{FIND } O(E) \\ \text{UNION } O(V) \end{array} \right. \rightarrow O(E^2 V)$

(2) Prim 算法 (使用 Fibonacci 树)

① 处理 V 顶点 $O(V)$

② $V \in EXTRACT-MIN$ $O(V \lg V)$

③ $O(E) \rightarrow DECREASE-KEY $O(E \cancel{\lg V})$$
 $\rightarrow O(E + V \lg V)$

当权重分布在 $[0, 1]$ 内时，我们可以对 E 边使用桶排序，复杂度
缩小到 $O(E)$ ，则 Kruskal 算法复杂度为 $O(E + E^2 V) = O(E^2 V)$
此明显比 Prim 算法更优。

23.2-7.

引理：设 T 为 $G = (V, E)$ 的一棵 MST，考虑图 $G' = (V', E')$ ，其中 G' 为 G 的
一个子图。令 $\bar{T} = E - T$ 是 E 中除去 T 的其它边。则 G' 存在一棵 MST
其不包含任何 \bar{T} 的边。

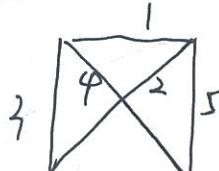
证明略。

由引理我们知道只需考虑最多 $(V-1) + V$ 条边。条数为 $(V+1) \bar{T}$

用 Fibonacci 树来实现 Prim 算法，总复杂度为 $O(E' + V' \lg V')$
 $= O(2V + V \lg V) = O(V \lg V)$

23.1.

a. 由于所有权重都互不相同，故横跨每个切割的轻量级边唯一，由 23.1-6
可知 MST 唯一。



如图，MST 为 1+2+4

代价为 2+5+1 或 1+3+4



b. 假设存在任何一棵生成最小生成树不小于 T 至少相差两条边，

设 (u, v) 为 $T-T'$ 中权值最小的边。把 (u, v)
 $\begin{matrix} T \\ T-T' \end{matrix}$ $\begin{matrix} T' \\ T'-T \end{matrix}$ 添加入 T' ，必形成一个环 C ，则 C 必含边 $(x, y) \in T-T'$
 (否则， C 中任何边都属于 T ， T 将有环)。把下证
 $(u, v) < (x, y)$ 。我们把 (x, y) 添加入 T ，其也必
 形成环 C' ，其中 C' 也必有边 $(u', v') \in T-T'$ 。
 若 $(u', v') > (x, y)$ ，我们可以用 (x, y) 替换 (u', v') ，
 得到新的生成树权值小于 T ，矛盾。故 $(u, v) < (x, y)$ ，
 则 $(u, v) \leq (u', v') < (x, y)$ 。我们在环 C 中用 (u, v) 替代
 (x, y) ，要得到树 T'' ，易知 $T \neq T' \neq T''$ ，且 $w(T'') < w(T')$ ，
 与 T'' 为次优 MST 矛盾。

23. 可以用 DFS 或 BFS，这里用 BFS。

可以用 DFS 或 BFS，这里用 BFS。
 用 BFS 可以由某顶点开始遍历所有顶点，设 u 为起始结点，遍
 历 v 时，设其父结点为 x ，若 $w(v, x) > w(u, v)$ ，则 $w[u, x] = w[v, x]$ ，
 否则 $w[u, x] = w[u, v]$ 。依此

BFS-MAX(T, u)

for each vertex $u \in V$
 for each vertex $v \in V$
 $\text{map}[u, v] = \text{NIL}$

$Q = \emptyset$

ENQUEUE(Q, u)

while $Q \neq \emptyset$

$x = \text{DEQUEUE}(Q)$

for each $v \in \text{adj}[x]$

if $\text{map}[u, v] = \text{NIL}$ and $w[u, v] < \text{map}[u, v]$

if $x = u$ or $w(x, v) > \text{map}[u, v]$

$\text{map}[u, v] = (x, v)$

else

$\text{map}[u, v] = \text{map}[u, x]$

ENQUEUE(Q, v)

由于对每个顶点都需要运行
 一次 BFS，且每次 BFS 需扫描
 MST 里边的边，为 $O(V^2)$ 条，
 故复杂度为 $O(V^2)$

d. 由于依次构造 MST 可由 MS1 去掉一条边，再并上另一条边。
 故我们可以在依次考察所有 $E - T$ 的边 (u, v) ，用
 它替换 T 中 $u \rightarrow v$ 路径上的最大权重的边，使得
 $w[u, v] - \max[u, v]$
 ~~$w[u, v] - \max[u, v]$~~ 最小。

算法：

- ① 用 Fibonacci 树来实现 Prim 算法，计算出 MST，复杂度 $O(E + V \log V)$
- ② 计算所有的 $\max[u, v]$ ，复杂度 $O(V^2)$
- ③ 依次考察所有边，计算 $\frac{w[u, v] - \max[u, v]}{\max[u, v]}$ 取最小值，复杂度 $O(E)$

但由于 $E < V^2$ ，故总复杂度为 $O(V^2)$

23-2. 和 Prim 边连接
 a. 7 形成类似 Kruskal 算法，由于每次添加一个新结点（~~连通~~ 添加到已连通的结点，则添加一个，否则添加两个）。添加一个时，~~每~~ 相当于只与这个新结点如图，已有 a, b, 我们要添加 c, 则选择的边为人连边中最短的 (a, c)，此与 Prim 算法相似。添加两个时，即与 Kruskal 算法相似，连通了两个孤立的点，设边 (u, v) ，设其为 u 的邻接边最短的。因为不管怎样划分，u 必属于某个闭合集，要与任意一个其它的连通切割连通，必经过 u 的某条邻接边，而 (u, v) 为其中最短的，故其必为安全的。（添加一个顶点的情况也是同理）。当 7 形成后，~~该~~ MST-Prim 算法生成的最小生成树，其即为连通几个切割的最小生成树。
 故 7 有一样的 MST。



- b. 由图中每两个顶点形成每条边两个独立时，其形成的 $|G'|V|$ 最小，
为 $\frac{|V|}{2}$
- c. 注意到，此处中算该中的 union 总是合并某一个连通分量，而不是某个集合，故我们可以不用并查集。
改用一个数组 α 表示每个顶点的所属集合，即当并集 (u, v) 时
只需要简单检查 $\alpha[v] = \alpha[u]$ 即可。

复杂度：
 ① $V \uparrow \text{MAKE-SET}$ $O(V) \cancel{\in O(E)}$
 ② UNION $O(V)$
 ③ FIND $O(E)$

由于每行操作复杂度为 $O(1)$ ，故共 $O(E)$

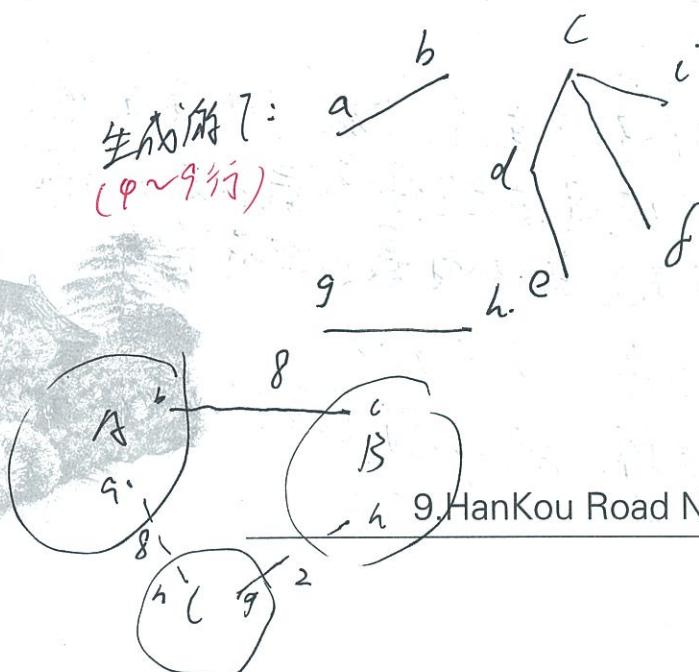
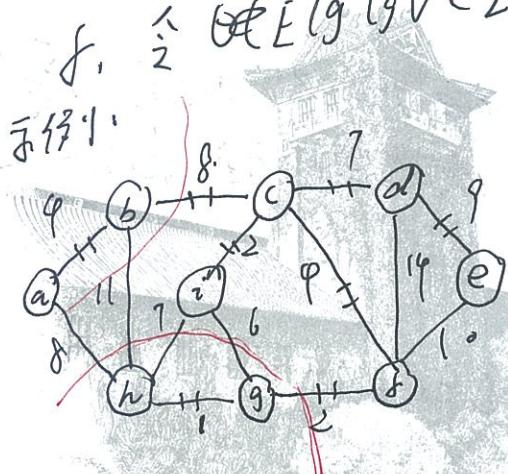
(若用并查集，则为 $O(E \alpha(V))$) \Rightarrow Fibonacci 例

d. Prim 算法的复杂度为 $O(E + V \lg V)$ 。由于每次运行 MST-REDUCE，
将 $|V|$ 至少缩小一半，假设每次运行 k 次，总复杂度为

$E + (V/2)^k \lg (V/2^k) + kE = (k+1)E + \frac{V \lg V}{2^k} - \frac{V k}{2^k}$, 令 $k = \lg \lg V$,

可得总复杂度 $O(E \lg (\lg V))$ 。证明留作。

可得总复杂度 $O(E \lg (\lg V)) < E + V \lg V \Rightarrow E < \frac{V \lg V}{\lg \lg V - 1} = O(\frac{V \lg V}{\lg \lg V})$



23-3a. 若 MST 不是并领生成树，则该 MST 最长的边为 (u, v) , 且只存在某棵并领生成树，其任意边权值小于 $w(u, v)$. 在 MST 中除掉 (u, v) , 将形成两棵树，也形成了两个分离。由 MST 的生成树 $w(u, v)$ 为连接这两个分离最小权值的边，而并领生成树存在连接这两个分离更小的边，矛盾。

b. ①扫描图的每条边，去掉权值大于 α 的边 $O(E)$
 ②用 DFS 判断图是否连通，若是，则存在 MST，由 α 和 β 存在并领生成树，~~至多一个~~ $O(V+E)$

c. ①对 E 条边找出其中位数 $median$ $O(E)$
 ②用 b 中算法判断是否并领生成树的值是否不超过 α $O(E)$
 ③若是，则去掉所有大于 median 的边，递归执行算法
 若不是，则按 23-2 中，~~把~~ 小于 median 的 $\frac{E}{2}$ 条边收缩为若干个顶点
 按 23-2 中 C 的实现复杂度为 $O(E)$ ，且收缩后，剩余 $\frac{E}{2}$ 条边，递归执行算法

$$\text{复杂度: } f(E) = f\left(\frac{E}{2}\right) + O(E) \in O(E).$$

23-4.

a. 正确。由于 $\emptyset \neq e$ 连通，故图中存在包含 e 的环，由 23.1 算法
 去除掉 e 后，该图依然存在最小生成树，依次递归往，最后必然
 去除 $|V|-1$ 条边，其必为最小生成树。

执行：①排序 $O(E \lg E)$
 ②去除掉一条边后，对图运行 DFS，共 $O(E(V+E))$
~~复杂度~~ $O(E^2)$



b. 显然错误

复杂度: MAKE-SET $O(V)$
 FINISH $O(E)$
 UNION $O(V)$
 合 $O(E \cdot V)$

c. 由 23.1-5 知, 每次删除除所选边都还是 MST 所需边, 依此进行
 最后必然形成一棵树.

执行: 用并查集, 当形成树时, 用 DFS 扫描找出环及权值最大的边.
 由于至多有一个环, 边数最多为 V , DFS 复杂度 $O(V)$, 最多需运行
 $O(E)$ 次, 合 $O(EL)$.

24.1-3 只要当集之循环后, 发现 $v.d$ 改变时, 即终止算法.

BELLMAN-FORD(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)

changes = ~~TRUE~~ TRUE

while changes = TRUE

changes = FALSE

for —

do —

RELAX- $m(u, v, w)$

if $d(v) > d(u) + w(u, v)$

—
changes = TRUE

24.1-4 若存在负权值边, 执行 VI-1 次后, 再执行原程序 5~7 行,
 若 $s \rightarrow v$ 存在负权值路径, 且 $v.z = u$, 则 $v.d > u.d + w_{uv}$
 故需修改第 7 行为 $v.d = -\infty$.
 9. Hankou Road Nanjing China
 Cable: 0909

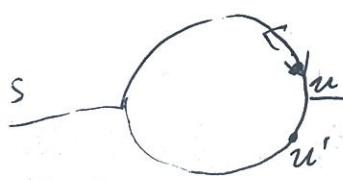
24.1.5. 我们只需在初态时, 令 $v \cdot d = 0$, 其它与 Bellman-Ford 算法一样。假设所有顶点至 v 的最短距离路径为 $\{v_0, v_1, \dots, v_n\}$, 且 $v = v_n$.

则在循环过程中, 处理出的松弛次序为 $(v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n)$ 故由路径松弛性质知, 此时 $v_n \cdot d = \delta(v_0, v_n)$. 由于 v_0 可取任何值, 故有

$$v \cdot d = \min(v_n \cdot d) \quad (v_0 = \dots)$$

(注: 这里允许 $v \rightarrow v = v$ 这种最小值, 若不允许, 则令 $v \cdot d$ 初始为 $\frac{1}{n}$ 表示入度边最小值)

24.1.6. 如图, 出现权值为负的环时, 沿 $(v, v, \dots), (v, \dots, v, \dots)$ 往前走, 就会进入死环, 故我们出现 $v \cdot d > u \cdot d$ 时,

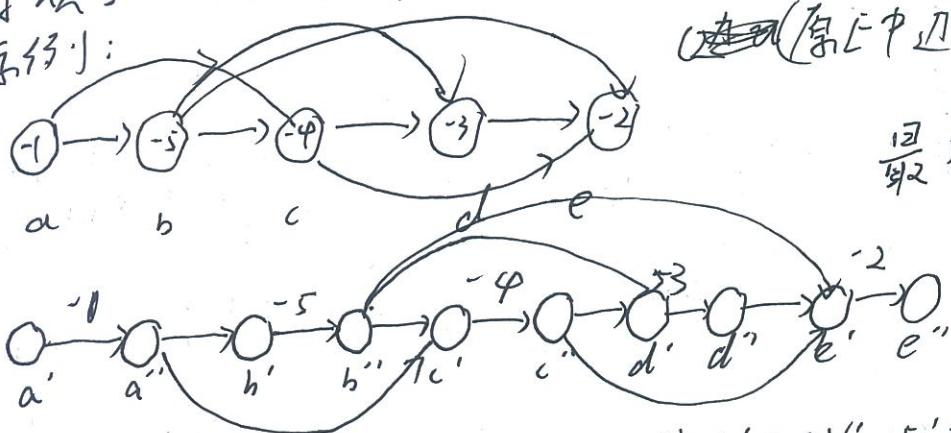


v 的情况时, 可以用 DFS 反向沿 v 往后走, 则必在 u' 处发现 u 为 ~~要~~ (后向边). 此时我们沿 (v, \dots, v, \dots) 一直输出出现 u' 即可。

24.2.3.

方法一: 令每个顶点分化成两个顶点, 即构造图 $G' = \langle V', E' \rangle$, 其中 V' 中每个顶点 v 对应 V' 中两个顶点 v', v'' , E' 仍包含 E 中每条边, 但权值为 0, “原 E ” 中出来的边为 $\langle v', v'' \rangle$, 其权重为 $w(v)$, 这样, 从 $s \rightarrow v$ 的最长路径转化为从 $s' \rightarrow v''$ 的最长路径。注: $|V'| = 2|V|$, $|E'| = |E| + |V|$

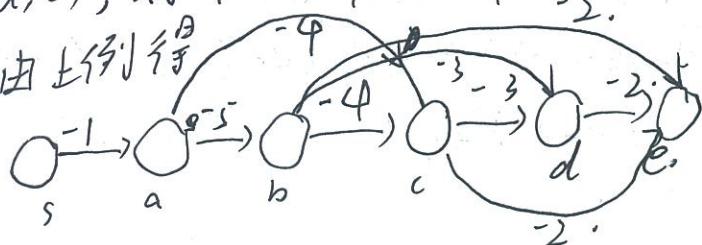
示例:



最大权值为: $|a \rightarrow c \rightarrow e| = 7$
(即取负后求最小, 得 7)

方法二: 为图 G 添加一个源点 s , 形成 $G' = \langle V', E' \rangle$, 其中 s 指向 G 中所有入度为 0 的顶点, 保留原 G 中所有边, 而对每个顶点 v 的权值, 变成给边 $\langle u, v \rangle$, 则 $|V'| = |V| + 1$, $|E'| \leq |E| + |V|$ (最多有 $|V|$ 个入度为 0 的点)

示例: 由上例得

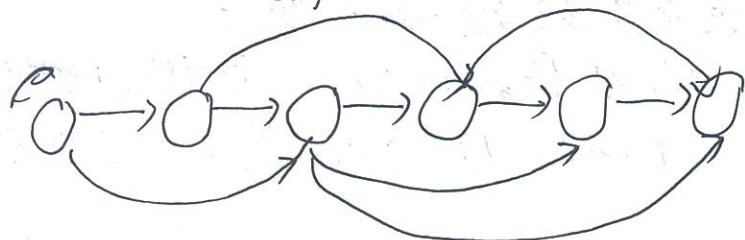




24.2-4. 我们可以计算从每个顶点出发的边数，再累加得到所有路径数。

18 PATHS(G)

q. topologically sort the vertices of G
for each vertex u , taken in reverse topologically sorted order
for each vertex $v \in G$. $\text{Adj}[u]$
 $u.\text{paths} = u.\text{paths} + v.\text{paths}$



21 12 7 3 1 0

$$21 + 12 + 7 + 3 + 1 + 0 = 50$$

复杂度：拓扑排序 $O(V+E)$

for 循环 $O(V+E)$

是 $O(V+E)$

(b) 可以计算以每个顶点终止的边的条数，从上向开始计算

24.3-2. 在证明过程中，我们假设 $u.d > \delta(s, u)$ ，则 $s \rightarrow u$ 的最长路径

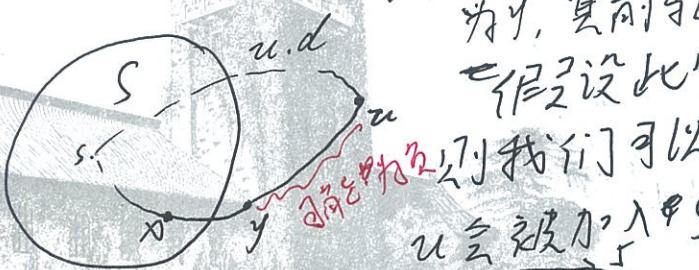
加上必经过 ~~最近~~ 的一些顶点，设最近 s 的

为 y ，其前驱为 x 。则有 $y.d = \delta(s, y)$

假设此时 $y \rightarrow u$ 为真，则 $y.d = \delta(s, y) > \delta(s,$

~~最近~~ y 时 $y.d > u.d > \delta(s, u)$ ，此时

u 会误加入 S 中，造成错误。



如图，在 s 加入 S 后，由于 $u.d = (s, u)$

$= 5 < y.d = (s, y) = 7$ ，故 u 会误

加入集合中，而实际 $s \rightarrow$



9. Hankou Road Nanjing China

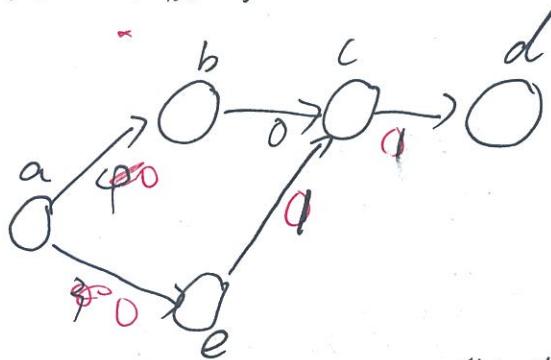
最短路径为 $s \rightarrow y \rightarrow u$: 0909

29.34. 对 S 之外的每个顶点 v , 检查其所有入边, 看是否满足
 ~~$v.d = \min\{u.d + w(u, v)\}$ 且 $u = v$.~~

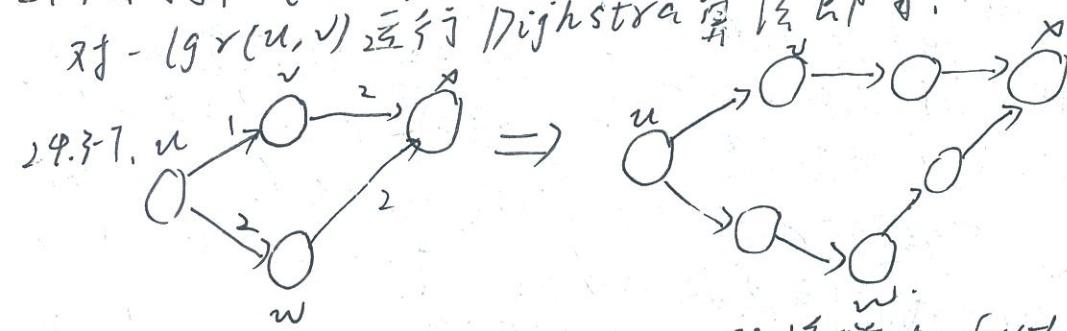
复杂度显然为 $O(V+E)$

证明: 假设所有顶点都检查正确, 但卻存在某些顶点
 不满足某棵最短路径树上. 设这些顶点中 ~~v~~ 中 $v.d$
 最小的为顶点 v , 又由所有边权重非负, 故 u_1, u_2, \dots, u_h
 为正确的顶点, 即 $u_i.d = \delta(S, u_i)$ ($i=1, 2, \dots, h$).
 $s \rightarrow v$ 的最短路径必通过 u_i , 而我们已知
 $u_i.d + w(u_i, v)$ 是最小的 u_i , 而我们已知
 $v.d = u_i.d + w(u_i, v)$, 故由收敛性知
 $v.d = \delta(S, v)$, 矛盾.

29.35. 如图, 加入 S 集合顶点的顺序为 e, a, c, b, d ,



29.36. 对每个 $r(u, v)$ 取对数为 $\lg r(u, v)$; 当 $r(u, v) = 0$ 时令 $\lg v = -\infty$,
 对 $-\lg r(u, v)$ 运行 Dijkstra 算法即可.



29.37. u 有 $1 \rightarrow 2 \rightarrow v$ 和 $u \rightarrow w \rightarrow z$.
 如图, 易知权重为 w 的边, 转化时将增加 $(w-1)$ 个顶点, 共有 $|E|$ 条
 边, 有 G' 共有 $|V| + \sum w(u, v) + |E|$
 由于 S 至每个顶点的最短路径长度不同, 故在广度优先搜索
 中, $v.d$ 小的顶点将被更大的顶点丢弃并被涂黑.



29.3-8 ① 从 s 到其它顶点的最短路径长度 $\leq (V-1)W$
 ② 构建数组 $A[0 \dots n]$, 其中 $n = (V-1)W$, 其中 $A[i]$ 存放到 s 距离为 i 的顶点
 ③ 构成的无向链表 对应
 ④ INSERT: 插入时, 直接插入链表 $O(1)$
 ⑤ EXTRACT-MIN: 移除时, 我们首先要找到最小的非空的链表
 注意到 EXTRACT-MIN 取出的值是单调递增的, 故我们每次查找时只需从上一次的查找点继续即可, 找到后, 只需移除链表 中任何一元素, 故总共未花费时间 $O(h)$ 即 $(V-1)W$

DECREASE-KEY: 只需把对应元素从链表中取出放到对应链表中 $O(1)$
 ⑥ 由于 INSERT, EXTRACT-MIN, DECREASE-KEY 操作共 $O(V+E)/2$,
 故这些操作总复杂度为 $O((V-1)W + V+E) = O(VW+E)$

29.3-9. 注意到每次 ~~加~~ ^{EXTRACT-MIN} 后, 优先队列里最多只有 $(V-1)^2$ 个不同值.
 假设某次我们往 S 加入 u , 然后对边 (u, v) 松弛, 则 $v.d \leq u.d + W$. 而对于其它 $V-S$ 中的顶点, 若其为有限, 则必为 $v.d \leq u.d + W$. 而对于其它 $V-S$ 中的顶点, 若其为无限, 则必为 $v.d \leq u.d + W$. 而对于其它 $V-S$ 中的顶点, 若其为无限, 则必为 $v.d \leq u.d + W$. 故 $v.d \leq u.d + W$, 则 $(V-S)$ 中其它顶点的 d 值 $\leq u.d + W$ 的顶点, 故 $v.d \leq u.d + W$, 故解得 $v.d$ 可能为 $u.d, u.d+1, \dots, u.d+W$, 共 $(W+1)$

我们可以用一棵红黑树来维持这个优先队列, 则 INSERT, EXTRACT-MIN, DECREASE-KEY 的复杂度都为 $O(\lg W)$, 故总复杂度由 $O((V-1)E)$

29.3-10. 如图, 用课本证明, 假设集 S 中加入的 u 不满足 $u.d = S(s, u)$, 则 $S \rightarrow u$ 的最短路径必经过 $(V-S)$ 的某些顶点, 设 y

则 $S \rightarrow u$ 的最短路径必经正 $y \rightarrow u$ 的长度必为正值 (因 $y \notin S$). 若不然, 由于只有从 S 发出的边为负, 故 $y \rightarrow u$ 的路径必经过 x , 则 $S \rightarrow x \rightarrow y \rightarrow u$ 构成环的路径必经过 x , 而若 $(s, x) \neq (x, y) \neq (y, u) \neq 0$, 则长度由 $u.d \leq (s, u)$ 而若 $(s, x) \neq (x, y) \neq (y, u) \neq 0$, 则 $u.d \leq (s, u)$, 故 $u.d \geq eu.d$ 将之 (s, u) , 由 $u.d \leq (s, u)$, 而 $u.d \leq (s, u)$, 故 $u.d \geq eu.d$ 矛盾, 若 $(s, x) \neq (x, y) \neq (y, u) \neq 0$, 则 $u.d \leq (s, u)$, 故 $u.d \geq eu.d$ 矛盾.

24.4-4 对于一个单源单目的地最短路径问题，我们已知。设 s 为起始点。

$$\delta(s) = 0$$

$$\delta(v) - \delta(u) \leq w(u, v) \quad \forall u, v \in V$$

要转化为 $\Delta x \leq b$ 的形式，我们可以令

$$\begin{cases} x_s = 0 \\ -x_s = 0 \\ x_v - x_u \leq w(u, v) \end{cases}$$

这样， $x_v = \delta(v)$ 是该方程组的一个解。

然而， $x_v = \delta(v)$ 却不是该方程组的唯一解。例如，当所有权值非负时，令所有 $x_i = 0$ ，可知满足方程组。为强调差别，让 $\delta(t) = x_t$ ，我们可以该方程组添加目标函数：求 x_t 的最大值。

下面证明 $\max_{t \in V} x_t = \delta(t)$ 。

首先，由于 $\delta(t)$ 为 x_t 的一个解，故有 $\delta(t) \leq \max_{t \in V} x_t$ 。

其次，令 $s \rightarrow t$ 的最短路径为 v_1, v_2, \dots, v_n ，则有

$$x_2 - x_1 \leq w(v_1, v_2), \quad \dots \Rightarrow x_n - x_1 \leq w(v_1, v_2) + \dots + w(v_{n-1}, v_n) = \delta(t),$$

$$x_n - x_{n-1} \leq w(v_n, v_{n-1}) \quad \text{即 } x_t - 0 \leq \delta(t)$$

故 $\delta(t) \geq \max_{t \in V} x_t$ ，
解得 $\delta(t) = \max_{t \in V} x_t$ 。

(假设该图存在负权值边，则上述方程组无解)。

24.4-5 我们可以不要 v_0 ，而令所有其它顶点的初始待求值为 0，

运行 Bellman-Ford 算法即可，其时间复杂度为 $O(m^2)$ 。

24.4-6 $x_i = x_j + b_{ij}$ 转化为 $x_i - x_j \leq b_{ij}$ ，即 $x_i - y_i \leq b_{ij}$ ，且 $y_i \leq x_i$ 。

24.4-8 我们可以证对于 $\Delta x \leq b$ 满足方程组的所有解，必有 $y_i \leq x_i$ 。

我们任取 x_i ，假设 $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_i$ 为 $v_0 \rightarrow v_i$ 的最短路径

$$\begin{cases} y_2 - y_1 \leq w(v_1, v_2) \\ y_3 - y_2 \leq w(v_2, v_3) \\ \dots \\ y_i - y_{i-1} \leq w(v_{i-1}, v_i) \end{cases} \Rightarrow y_i - y_1 \leq w(v_1, v_2) + \dots + w(v_{i-1}, v_i) = x_i - w(v_0, v_1) = x_i - y_1 \leq y_i - y_1 \leq x_i.$$



24.4-9 对于任意一组解 (x_1, x_2, \dots, x_n) , 我们设其中最大的一个为 x_h , 最小的一个为 x_1 , 对应图内的顶点分别记为 v_h 和 v_0 。由于对应图中, 任意一条最短路径 t_0, t_1, \dots, t_h , 其必有 $x(t_0, t_1) = 0$, 故原图的角平分至少有一个为 0, 即即为原图角平分的最大值。由 x_h 为某组角平分的最大值, 故其对应的 $\delta(v_h)$ 应 < 0 , 则必然存在 $v_0 \rightarrow v_h$ 的一条经过其它顶点的路径, 设最短的等为 γ , 其经过 $v_0, v_1, v_2, \dots, v_h$, 对应角平分为 $(x'_1, x'_2, \dots, x'_n)$, 则有

$$\begin{array}{c} v_h \\ | \\ v_0 \xrightarrow{\gamma} v_1 \xrightarrow{\gamma} v_2 \xrightarrow{\gamma} \dots \xrightarrow{\gamma} v_{h-1} \xrightarrow{\gamma} v_h \\ \text{D} \end{array} \quad \left\{ \begin{array}{l} x'_2 - x'_1 \leq w(v_1, v_2) \\ x'_3 - x'_2 \leq w(v_2, v_3) \\ \vdots \\ x'_h - x'_{h-1} \leq w(v_{h-1}, v_h) \end{array} \right. \Rightarrow \left\{ \begin{array}{l} x_h - x'_1 \leq w(v_1, v_2) + \\ w(v_2, v_3) + \dots + w(v_{h-1}, v_h) \\ = \delta(v_h, v_0) \end{array} \right.$$

而 $x_1 \geq x'_1$, 故有 $x_h - x'_1 \leq x_h - x_1 \leq \delta(v_h, v_0)$, 亦即 $\delta(v_h, v_0) \leq \delta(v_h, v_0) - \delta(v_1)$
 $(\min\{x_i\} - \max\{x_i\})$ 的最大值 $\leq \delta(v_h, v_0) - \delta(v_1)$

原题得证。

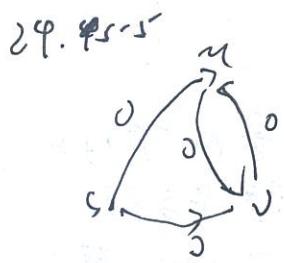
24.4-11. 遍对任意 $x_j - x_i \leq b$, 由于 x_j, x_i 为整数, 故必有 $x_j - x_i \leq \lfloor b \rfloor$, 故我们只需把所有 b 取 $\lfloor b \rfloor$ 即可。

24.4-12. 我们只需更改 RELAX 的判定条件。

若 $v_i.d > v_i.d + w(v_i, v_j)$, 令 $v_j.d = \lfloor v_i.d + w(v_i, v_j) \rfloor$

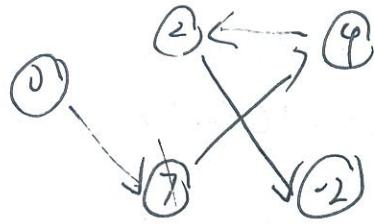
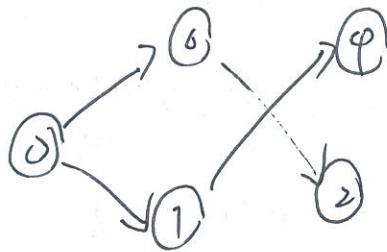
则修改后满足 $v_j.d - v_i.d \leq w(v_i, v_j)$

24.5-9. 注意到对任何顶点 $s.d = +\infty$, 而 $v.d$ 为 $s \rightarrow v$ 的某些路径值, 故 $s.d$ 为 $s \rightarrow s$ 为某条路径的值。若 $s.d \neq NIL$, 则 $s.d < 0$, 即 $s \rightarrow s$ 有一个负权值环。



则 $s \rightarrow u$ 的最短路径有 $s \rightarrow v \rightarrow u$,
 $s \rightarrow v$ 的最短路径有 $s \rightarrow u \rightarrow v$,
故 $u_1 = v$, $v_2 = u$. 形成环.

24.5-7. 对从生成的最短路径树, 用BFS 来从 s 开始遍历, 所得的每个顶点到 s 的值 $v_i.d$ 即为一个最短的松弛序列的长度, 其经过的距离即对应于松弛步聚. 由于树的长度为 $|V|-1$, 故得证.



24-1. 如图, 假设任意一条最短路径为: $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$,
任取连续的 $v_{i-1} \rightarrow v_i \rightarrow v_{i+1}$, 若 $v_{i-1} \rightarrow v_i$ 为反向, $v_i \rightarrow v_{i+1}$ 为正向, 则
我们在上一轮令 $v_i.d = \delta(s, v_i)$ 后, 1. 一轮就可以让 $v_{i+1}.d = \delta(s, v_{i+1})$
同理, 若为正反, 则在一轮里上半轮令 $v_i.d = \delta(s, v_i)$ 后, 下半轮就可以
让 $v_{i+1}.d = \delta(s, v_{i+1})$. 若都为正或反, 则在半轮里就可以修正.
故任意两对连续的 (v_{i-1}, v_i) , (v_i, v_{i+1}) , 至多在两轮里可以修正.
而我们知道 $v_1 = s$, 故第一轮里 $v_1 \rightarrow v_2$ 将得到最短值. 依次进行下去,
最多需要 $\lceil |V|/2 \rceil$ 轮.

示例: 反 正 反 正 反 正 反

1	✓					
2		✓	✓			
3			✓	✓		
4				✓	✓	
5					✓	✓

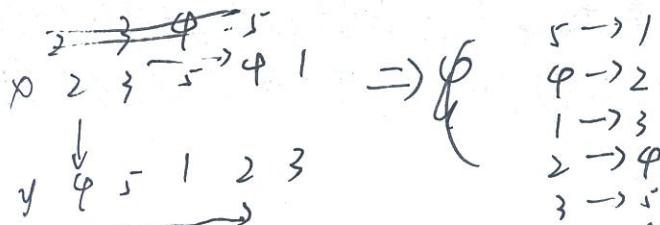
1	✓	✓				
2			✓	✓		
3				✓	✓	
4					✓	✓

9个数, 共4轮.

如: $3 \rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow 6 \rightarrow 5 \rightarrow 9 \rightarrow 7 \rightarrow 8$



24-2. b. 令盒子*x*和*y*分别按从小到大排序，设排序后为 $\{x'_1, x'_2, \dots, x'_d\}$, $\{y'_1, y'_2, \dots, y'_d\}$, 则依次检查是否满足 $x'_1 < y'_1, x'_2 < y'_2, \dots, x'_d < y'_d$ 即可。假定某次 $x'_i > y'_i$ ，这意味着*y*中比 $x'_i \sim x'_d$ 多 $(d-i+1)$ 个盒子的最多只有 $y'_i \sim y'_d$ 共 $(d-i)$ 个，故不嵌套。而若每项都满足，则显然按 $y_1, y_2 \dots y_d$ 的顺序挑选对应 x_i 即可，如：



复杂度：主要是排序，共 $O(d \log d)$

c. (1) 对*n*个盒子分别进行排序，共 $O(nd \log d)$

(2) 两两对这些盒子按中方法两两比较共 ~~$O(n^2 d^2)$~~ $O(n^2 d) = O(d n^2)$

(3) 构建DAG图，对盒 $B_i < B_j$, 则 $v_i \rightarrow v_j$ $O(n^2)$

(4) 找出该图的最长路径（会拓扑排序，再一），即所求
因为有 ~~n^2~~ 条边， $O(n^2)$ 条边，故复杂度为 $O(n^2)$ ~~每对结点比较~~

总复杂度 $O(d n^2 + d n \log d)$.

应使用动态规划

$$v_i \rightarrow v_{i+1} \rightarrow v_{i+2} \rightarrow \dots \rightarrow v_n$$

24-3. a. 由 $R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{n-1}, i_n] \cdot R[i_n, i_1] < 1$

$$\Rightarrow \frac{1}{R[i_1, i_2]} \cdot \frac{1}{R[i_2, i_3]} \cdots \frac{1}{R[i_{n-1}, i_n]} \cdot \frac{1}{R[i_n, i_1]} < 1$$

$$\Rightarrow \lg \frac{1}{R[i_1, i_2]} + \lg \frac{1}{R[i_2, i_3]} + \cdots + \lg \frac{1}{R[i_{n-1}, i_n]} + \lg \frac{1}{R[i_n, i_1]} < 0$$

因此，我们可以令 (v_i, v_j) 的权值 $w(v_i, v_j) = \lg \frac{1}{R[i, j]} = -\lg R[i, j]$
故问题可以转化为找一个图中是否存在负权值环。

另外构建一个源点 v_0 ，连接所有其它顶点，权值为0，用Bellman Ford算法来找判别图中是否存在负权值环。由于 $O(n^2 + n) = O(n^2)$

Ford算法来找判别图中是否存在负权值环。由于 $O(n^2 + n) = O(n^2)$
边，故复杂度为 $O(n^3)$

b. 运行 Bellman-Ford 算法时，扫描完 $|V| - 1$ 条边后，我们再多扫描一轮。若遇到某条边 $u \rightarrow v$ 的值，则沿 $u \rightarrow u \rightarrow u \dots \rightarrow v$ 继续往前直到重新回到 v 为止，即为所求。

29-4.

a. 由于所有边权重非负，故我们可以用 Dijkstra 算法，构建一个优先队列，用数组 $Q[0 \dots |E| + 1]$ 表示，其中 $Q[i]$ 用来存储所有权值为 i 的边。初始化所有 $d[v](v \neq s) = |E| + 1$ （因为随着算法运行， $d[v]$ 将逐渐减小）。
复杂度：（算法和 29.3-8-致）

EXTRACT-MIN: 由于每次查找操作需要找到最小的非空的 $Q[i]$ 且 ~~算~~ 算法中移除的边的权值逐渐递增，故最多只需
要数组 $|V|$ 次，复杂度为 $O(|E|)$

DECREASE-KEY: $O(1)$

总复杂度 $O(|E|)$

$$\begin{aligned} d_{\hat{w}_i}(u, v) &= w_i(u, v) + 2\delta_{i-1}(s, u) - 2\delta_{i-1}(s, v) \\ &\geq 2w_{i-1}(u, v) + 2\delta_{i-1}(s, u) - 2\delta_{i-1}(s, v) \\ &\geq 0 \end{aligned}$$

$$\begin{aligned} e. \delta_i(s, v) &= w_i(s, v_1) + w_i(v_1, v_2) + \dots + w_i(v_{k-1}, v_k) \quad (\text{假设 } s \rightarrow v \text{ 途径}) \\ &\text{经过 } s \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k, \\ &= \hat{w}_i(s, v) - 2\delta_{i-1}(s, s) + 2\delta_{i-1}(s, v_1) \\ &\quad + \hat{w}_i(v_1, v_2) - 2\delta_{i-1}(s, v_1) + 2\delta_{i-1}(s, v_2) \end{aligned}$$

$$\begin{aligned} &\quad + \hat{w}_i(v_{k-1}, v_k) - 2\delta_{i-1}(s, v_{k-1}) + 2\delta_{i-1}(s, v_k) \\ &= \hat{\delta}_i(s, v) - 2\delta_{i-1}(s, s) + 2\delta_{i-1}(s, v) \\ &= \hat{\delta}_i(s, v) + 2\delta_{i-1}(s, v), \end{aligned}$$

$$\begin{aligned} \text{故有 } \hat{\delta}_i(s, v) &= \delta_i(s, v) - 2\delta_{i-1}(s, v) \\ &\leq 2\delta_{i-1}(s, v) + |V| - 1 - 2\delta_{i-1}(s, v) \\ &= |V| - 1 \\ &\leq |E| \end{aligned}$$



f. ① 由 d 计算出所有 $w_i(u, v) \ O(E)$

② 由 $\delta_i^*(s, v) \leq E$, 仿 a 计算出所有 $\delta_i^*(s, v) \ O(E)$

③ 由 $\delta_i(s, v) = \delta_i^*(s, v) + 2\delta_{i-1}(s, v)$ 计算出所有的 $\delta_i(s, v) \ O(V)$

~~24-5~~
a. 由于所有环非负, 故最短的路径肯定可以去掉环, 最长为 $n-1$ 条边

b. $\delta_n(s, v)$ 也可以去掉环, 故 $\delta_n(s, v) = \delta_i(s, v) + \alpha$, 其中 $i = 0 \sim n-1$ 为某一个值, 故令 $h = i$, 必有式子成立

c. 易知 $\delta(s, v) \leq \delta(s, u) + \alpha$, 又我们可以由 u 路径 $\rightarrow v$,

则有 $\delta(s, v) - \alpha \geq \delta(s, u) \Rightarrow \delta(s, u) + \alpha \leq \delta(s, v)$

故 $\delta(s, v) = \delta(s, u) + \alpha$

d. 以 s 开始, 寻找到环上任意一点 u 的一条最短路径, 然后, 我们沿 u 开始绕环走, 直到路径长度为 n 为止, 此即求得待找结点 v .
由(6), 由 $s \rightarrow u$ 为最短路径, 故由 $s \rightarrow u$ 亦为最短路径,
则我们可以对 $s \rightarrow u \rightarrow v$ 削掉若干圈环, 必能令其长度 $< n$,
而各权值不变, 令该长度为 h , 则满足 $\delta_n(s, v) - \delta_h(s, v) = 0$.

e. 由(6), 对所有结点 $v \in V$, 有

$$\min_{0 \leq h \leq n-1} \frac{\delta_n(s, v) - \delta_h(s, v)}{n-h} \geq 0, \quad ①$$

而由(6)在任意最小平均权重上都存在 v , 有

$$\max_{0 \leq h \leq n-1} \frac{\delta_n(s, v) - \delta_h(s, v)}{n-h} = c$$

$$\text{故有 } \min_{0 \leq h \leq n-1} \frac{\delta_n(s, v) - \delta_h(s, v)}{n-h} \leq 0 \quad ②$$

两由 ①② 即可得结论



25.1.5
 ~~$L(n-1) = L(0) \times W$~~

$$L(0),$$

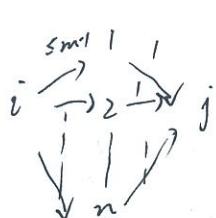
$$L(1) = L(0) \times W$$

$$L(2) = L(1) \times W$$

$$\vdots$$

$$L(n-1) = L^{(n-2)} \times W$$

即要求 $i \xrightarrow{sm} j, 2.1$



如：计算 1 出发的单源最短路径，有

$$L(0) = (0, \infty, \infty, \infty, \infty), W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$L(0) \times W =$$

$$L(1) = (0, 3, 8, \infty, -4), L(2) = L(1) \times W = (0, 3, 8, 2, -4)$$

$$L(3) = (0, 3, -3, 2, -4), L(4) = (0, 1, -3, 2, -4)$$

(分别对第 2 章课后习题 25-1 每个矩阵第一行)

该计算过程与 Bellman-Ford 相同。

该计算过程与 Dijkstra 与 Bellman-Ford 相同。

25.1.6、
 $L_{ij} \xrightarrow{w_{ij}} L_{ik} \xrightarrow{w_{kj}} L_{ij}$ 要求出 $i \rightarrow j$ 的最短距离。
 需求出满足 $L_{ij} + w_{kj} = L_{ik} + w_{ik}$ (其中 $k \neq j$)
 $L_{ik} + w_{kj}$ 的范围

$$IL_{ik} = \begin{pmatrix} 0 & 3 & 3 & 4 & 5 \\ 1 & 0 & 2 & 4 & 6 \\ 2 & 2 & 0 & 3 & 4 \\ 3 & 3 & 3 & 0 & 5 \\ 4 & 4 & 4 & 5 & 0 \end{pmatrix}$$

$1 \rightarrow 4$
 有 $1 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 4 : 2$.
 和 $1 \rightarrow 5 \rightarrow 4 : 2$
 如何解决？

25.1-9. 当有 $m \geq n-1$ 时，再多计算一次，若两个无环且同，则存在权值。

25.2-2. 若 $i \rightarrow j$ 有边，且 $w_{ij} = 1$ ，否则为 0。然后把 EXTEND-SLOW-TEST-PAIRS 的 m 改为 V , $T \rightarrow A$, 运行 SLOW-SLL-PAIRS. —————— (W) ^(L, u)

25.2-4. 由 $dij^{(h)} = \min(dij^{(h-1)}, di_k^{(h-1)} + dk_j^{(h-1)})$
 若去掉上标，有三种可能：
 $\begin{cases} di_j^{(h-1)}, di_k^{(h-1)} + dk_j^{(h-1)} \\ di_j^{(h-1)}, di_k^{(h)} + dk_j^{(h-1)} \\ di_j^{(h-1)}, di_k^{(h)} + dk_j^{(h)} \end{cases}$
 新的 $di_j^{(h)}$ 表示 $i \rightarrow h$ 经过 $(1, 2, \dots, h)$ ，若经过 h , $2/$
 而 $di_k^{(h)}$ 表示 $i \rightarrow h$ 经过 $(1, 2, \dots, h-1)$ ，不可简写。
 故 $di_k^{(h)} = di_k^{(h-1)}$, 同理 $dk_j^{(h)} = dk_j^{(h-1)}$,

故得证。

25.2-6. 若 $di_i^{(n)} < 0$, 则存在负权值环。

下证：存在负权值环 $\rightarrow di_i^{(n)} < 0$
 假设最短的一个环，其经过的最大标记点为 h ，且有



由于 $i \sim k$ 不在环上，且不存在 $(i \rightarrow i)$ 为最短环。故而
 $di_k^{(h-1)}, dk_i^{(h-1)}$ 为正确的最短路径值，而 $i \rightarrow h \rightarrow i$ 为负权值环。
 故 $di_i^{(h)}$ 将为负值。此后，由于任意一个值不会增加，故 $di_i^{(n)} < 0$ 。



$$25.2-7. \phi_{ij}^{(h)} = \begin{cases} \phi_{ij}^{(h-1)} & d_{ij}^{(h-1)} \leq d_{ih}^{(h-1)} + d_{hi}^{(h-1)} \\ \phi_{ij}^{(h)} & \dots \end{cases}$$

~~和~~ 最短 路径，我们以 ($\phi_{ij}^{(h)} = t$)

$$\phi_{it}^{(h-1)} \rightarrow t \rightarrow \phi_{itj}^{(h)}$$

$$g(u, v) \leq |VE|$$

25.2-8 我们给每条边赋值，由 24-4(a), 知
对任意一个 u , 我们可以在 $O(|E|)$ 内算出 $\delta(u, v)$ ($v=1, \dots, n$)
且 $O(|VE|)$ 时间可算出所有 $\delta(u, v)$ 。且若 $\delta(u, v) < \infty$,
则取 1, 否则为 0.

$$\hat{w}(u, v) = w(u, v) - w^*$$

$$25.3-4. \hat{w}(v_0, v_1, \dots, v_k) = \hat{w}(v_0, v_1) + \hat{w}(v_1, v_2) + \dots + \hat{w}(v_k, v_h)$$

$$= w(v_0, v_1) + \dots + w(v_h, v_h) - w^* - w^* - \dots - w^*$$

令 $w^* < 0$, 则 $v_0 \rightarrow v_h$ 经过的边越多, w^* 值越少, 容易
被经过少且的另一条 ~~最短~~ 路径取代

示例:



原来最短路径为
 $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$,
现为 $a \rightarrow e$

25.3-5. 考虑同引理 25.1 后半部分，易知运行 Bellman 后，
环路上的边之和依然为 0。而重新赋值后，
每条 $w(u, v) \geq 0$ ，而环路上的每条边之和为 0，故
必有 $w(u, v) = 0$

25.3-6. $s \leftarrow u$, 从 s 运行 Bellman, 则 $h(s, s) = 0$, $(s, u) = \infty$
故 $s @ \leftarrow @ u$ 则 $\hat{w}(u, s) = w(u, s) + h(u) - h(s) = 0 + \infty - 0 = \infty$
 $s @ (0) \leftarrow @ u$, 故 $\hat{s}(u, s) = \hat{w}(u, s) = \infty$, 且
 $\hat{s}(u, s) = \hat{s}(u, s) + h(s) - h(u) = \infty + 0 - \infty \neq 0$,
若图强连通，则任意 u , 对任意 v , 有 $h(v) < \infty$, 则有
 $h(v) \leq h(u) + \hat{s}(u, v)$, 故重新赋值后
 $\hat{s}(u, v) \geq 0$, 原算法性质依然保持

25-1.
a. for ($i \leftarrow 1$ to $|V|$)

for $j = 1$ to $|V|$
if $T[i, u] = 1$ and $T[v, j] = 1$ $O(|V|^2)$
 $T[i, j] = 1$

b. 假设 $1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow n$, 则闭包为 $\begin{pmatrix} 1 & 1 & \dots & 1 \\ 0 & 1 & 1 & \dots \\ 0 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots \\ 0 & \dots & 0 & 1 \end{pmatrix}$

添加 $(n \rightarrow 1)$, 则为全 1, 需更新 $n^2 - \frac{n(n-1)}{2} = \frac{n^2-n}{2} \in O(V^2)$, 故至少为 $O(V^2)$

c. 若 a 中, 原来已有 $T[i, v] = 1$, 则 $T[i, j] = 1$ 不需经过 $v \rightarrow u$ 算法忽略
已知任意 $n \in O(V^2)$ (至多 V^2 条边)

for $i = 1$ to $|V|$
if $T[i, u] = 1$ and $T[i, v] = 0$ 而这三行中, 由于 $T[v, j]$ 从未为 0,
则当 $j = v$ 时, 有 $T[i, j] = T[i, v] = 1$
for $j = 1$ to $|V|$ 放这三行每运行一次, 必至少有一个矩阵从 0 变为 1,
if $T[v, j] = 1$ 原来全为 0, 故至多只能运行 V^2 次,
 $T[i, j] = 1$ 加上第一行, 共 $O(V^3)$