A1.

(1)

(1.1) 使用数学归纳法。

首先，如果 $\varphi(s,v)=2$，则 s 到 v 的最短路径就是 s 到 v 的边，因此在第一轮松弛时，v 的最短路径即被确定。

考虑 $\varphi(s,v)=k$，记这条路径上 v 的前驱节点为 u，u 的最短路在第 t 轮中被确定，则根据归纳假设，$t\le\varphi(s,u)-1=\varphi(s,v)-2$。分两种情况讨论：

- 如果 u 在第 t 轮中需要出队，且最短路被确定时尚未出队，则 u 的下一次出队在第 t 轮，出队时确定 v 的最短路。
- 如果 u 在第 t 轮中不需要出队，或者需要出队，但在最短路确定前已经出队，则 u 的下一次出队在第 $t+1$ 轮，出队时确定 v 的最短路。

无论哪种情况，v 的最短路被确定的轮数都不超过 $t+1=\varphi(s,v)-1$

(1.2) 如图1所示，构造有 $2n+1$ 个点和 $3n+m-1$ 条边的图 $G_{n,m}$（图中省略了上排 n 个节点之间的 m 条边），适当安排各边在邻接表中的顺序，使得在第一轮中，外圈节点从 t 开始逆时针入队。不难验证，在第 k 轮（$1\le k\le n$）中，点 t 的当前最短路径被松弛为 $3^{n-k}+k-1$，进而引发上排节点之间的 m 次松弛，因此用时下界为 $\Omega(nm)$

(2)

(2.1) 算法 2 与 Dijkstra 算法的区别是：它允许用负权边松弛已在 S 中的节点，并将其重新加入优先队列，这使得它能够像 Bellman-Ford 算法一样完成所有可能的松弛，因此正确地求出有非正权边的图上的最短路。

(2.2) 如图2所示，构造有 $2n-1$ 个点和 $3n-3$ 条边的图 G_n，不难验证，最右边的点的当前最短路会被依次松弛为 $0,-1,-2,\ldots,1-2^{n-1}$，共被松弛了 2^{n-1} 次，因此用时下界为 $\Omega(2^n)$

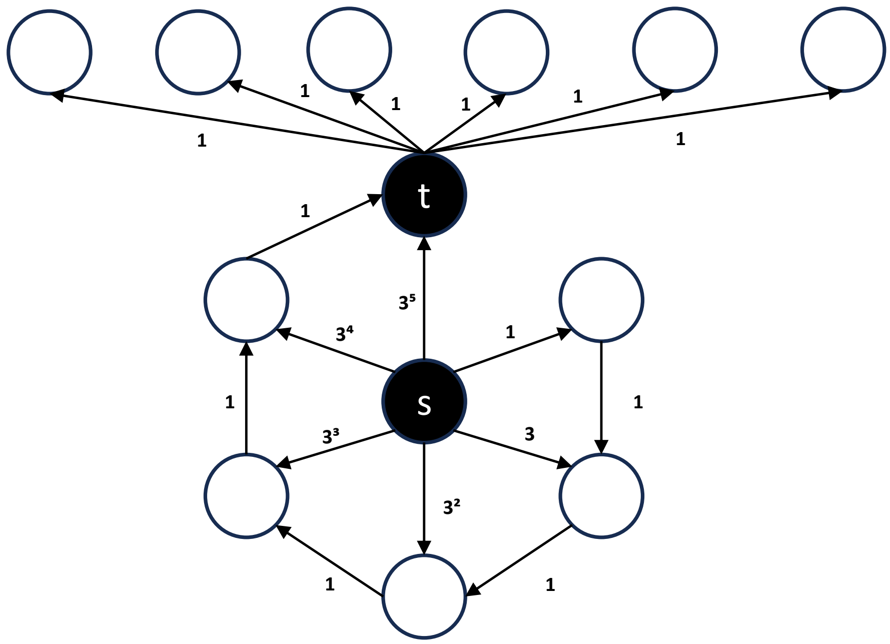

图 1: SPFA 在正权图上的较差情况示例

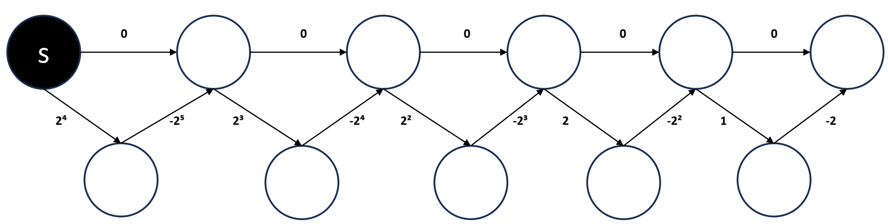

图 2: 算法 2 的较差情况示例

A2.

(1)

(1.1) 如图3所示，黑色边为树边，红色边为后向边，蓝色边为前向边，绿色边为横向边。强连通分量组成的有向无环图见图4。

(1.2) 如图5所示，黑色边为树边，红色边为后向边，黑色节点为割点，虚线边为桥。边上的箭头表示探索的方向。

(2) 根据 low 数组的定义，易得以下递推式：

$$\mathrm{low}[u] = \min\left\{\mathrm{dfn}[u], \min_{v\text{在}G_\pi\text{中是}u\text{的子节点}} \mathrm{low}[v], \min_{\substack{\langle u,v\rangle\text{为非树边}\\ v\rightsquigarrow u}} \mathrm{dfn}[v]\right\}$$

因此问题归结于：当 $\langle u,v\rangle$ 是非树边时，如何判断 v 是否可达 u。我们可以用一个栈记录“所有已访问节点中，有哪些节点能够连通到当前节点”。每当访问新的节点时，就将其压入栈中。每当节点 v 访问结束、即将回溯到父节点之前，判断 $\mathrm{dfn}[v]$ 和 $\mathrm{low}[v]$ 的大小关系：如果 $\mathrm{low}[v] < \mathrm{dfn}[v]$，那么只要 $\mathrm{low}[v]$ 未出栈，v 就仍能通过 $\mathrm{low}[v]$ 连通到新节点，因此 v 现在不能出栈，而应该和 $\mathrm{low}[v]$ 一同出栈；如果 $\mathrm{low}[v] = \mathrm{dfn}[v]$，则从 v 出发的所有路径都终止于 S_v，之后 v 再无连通到新节点的可能性，因此可以将 v 出栈。与此同时，栈中所有晚于 v 入栈的节点也一同出栈。

在无向图中，上述栈不是必需的，因为 $u \rightsquigarrow v$ 本身就意味着 $v \rightsquigarrow u$

(3)

(3.1) 显然，一个强连通分量中有且仅有一个“代表节点”的 dfn 值与 low 值相等，代表节点是此强连通分量中第一个被访问的点。在 (2) 的栈维护过程中，强连通分量中的所有点都会和代表节点一同出栈，由此可以找出所有强连通分量。此外，代表节点在 G_π 中的入边构成了强连通分量有向无环图中的边。

(3.2) 显然，对于节点 u：

- 如果 u 是 DFS 起始点（也即 G_π 中的根），则 u 是割点当且仅当：u 在 G_π 中的出度大于 1。
- 如果 u 不是 DFS 起始点，那么 u 是割点当且仅当：存在 G_π 中的子节点 v，使得 $\mathrm{low}[v] \ge \mathrm{dfn}[u]$

(3.3) 显然，桥必须是树边。记树边 $e = \langle u,v\rangle, \mathrm{dfn}[u] < \mathrm{dfn}[v]$，不难发现，$e$ 是桥当且仅当：$\mathrm{low}[v] > \mathrm{dfn}[u]$

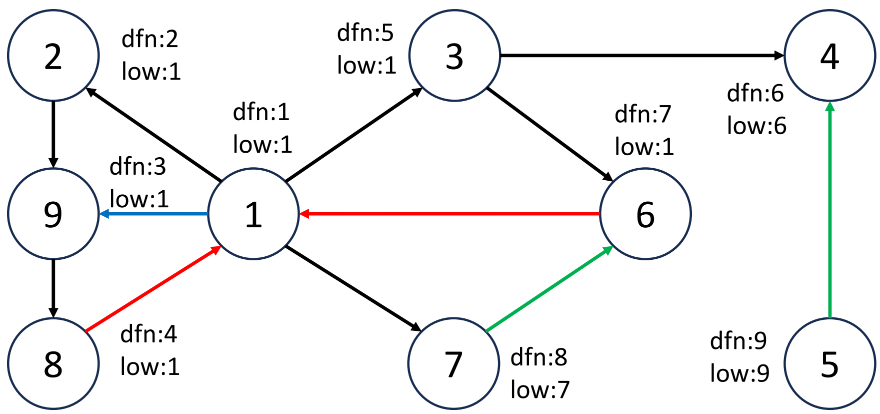

图 3: 有向图 DFS

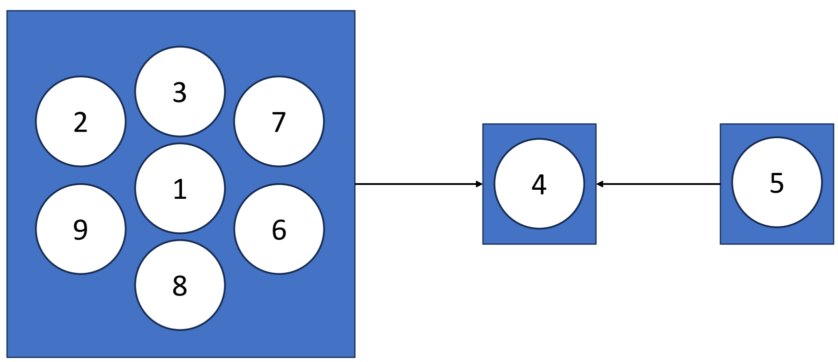

图 4: 连通分量图

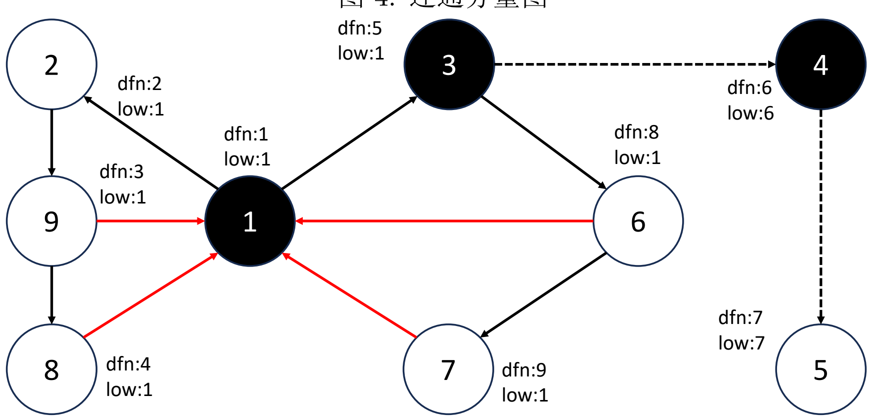

图 5: 无向图 DFS

A3.

(1) 对于给定的 w, u, v，假设它们的编号分别为 k, i, j，根据“环”的定义，需要找到一条 u 到 v 的最短简单路径，且路径上所有点的编号均小于 k。联想 Floyd 算法的过程，这一最短路径的长度即为 $d_{ij}^{(k-1)}$，对 k, i, j 进行枚举，可以在 $O(|V|^3)$ 时间内找到最小环：

$$\min_{k\geq 3}\min_{\substack{i,j<k\\ i\neq j}}\left(d_{ij}^{(k-1)}+d_{jk}^{(0)}+d_{ki}^{(0)}\right)$$

另外：如果仿照 (2.2) 中的做法，枚举所有边并运行 Dijkstra 算法，最坏时间复杂度为 $O\left(|V||E|\lg|V|+|E|^2\right)$，当 G 为稀疏图时较优。

(2)

(2.1) 对于入边 $\langle t, s\rangle$，根据“环”的定义，需要找到 s 到 t 的最短简单路径，且这条路径上至少包含三个点。因为图中没有反向平行边，因此可以直接运行以 s 为源点的 Dijkstra 算法，所得到的最短路一定满足上述条件。遍历所有入边，最终得到的最小环的大小为：

$$\min_{\langle t,s\rangle\in E}\left(w(t,s)+\delta(s,t)\right)$$

最坏时间复杂度为 $O(|V|\lg|V|+|E|)$

(2.2) 当存在反向平行边时，Dijkstra 算法有可能将 s 到 t 的直接连边作为最短路，这样构成的环不符合定义。因此，对于每条入边 $\langle t, s\rangle$，需要将 $\langle s, t\rangle$（如果存在）删除后运行 Dijkstra 算法。最坏情况需要运行 $|V|-1$ 次 Dijkstra 算法，因此最坏时间复杂度为 $O\left(|V|^2\lg|V|+|V||E|\right)$

(3) 首先，不断地删除度数为 1 的节点，直到不能再删为止，记此时得到的图为 G'。若 $k=0$，则 G' 是一个环，直接输出其大小即为结果。否则，继续删除所有度数为 2 的节点，但需要合并所在边的边权，记此时得到的图为 $G^\star$。在 $G^\star$ 上求出最小环即可（特别地，此时图上可能存在重边，且允许两个顶点通过两条不同的重边成环）。

显然，$|E'|-|V'|=k$。记 G' 中度数为 2 的节点个数为 t，则其余节点的度数均不小于 3，因此有：$2|E'|=\sum\limits_{v\in V'}\deg(v)\geq 2t+3(|V'|-t)$ ，即 $t\geq|V'|-2k$。每删除一个度数为 2 的节点，节点数和边数都减一，因此 $|V^\star|=|V'|-t\leq 2k, |E^\star|=|E'|-t\leq 3k$。$k$ 为常数，因此 $|V^\star|$、$|E^\star|$ 均为 $O(1)$，求出 $G^\star$ 的最小环只需 $O(1)$ 时间，算法总运行时间为 $O(|V|)$

A4.

(1)

(1.1) 如图6所示。

(1.2)

(1.2.1) 拓扑序相等意味着 p, q 在同一个强连通分量中，即 $p \rightsquigarrow \neg p$ 且 $\neg p \rightsquigarrow p$。记 p 到 $\neg p$ 的一条路径为 $(p = u_0, u_1, u_2, \ldots, u_{k-1}, u_k = \neg p)$。假设 2-CNF 可满足，任取一个成真赋值，有：$\forall 0 \leq i \leq k-1, u_i \rightarrow u_{i+1}$为真，使用归纳法易证明此时 $u_0 \rightarrow u_k$ 也为真，即 $p \rightarrow \neg p$ 为真，即 p 为假。同理，由 $\neg p \rightsquigarrow p$ 可知 p 为真，矛盾。因此 2-CNF 不可满足。

(1.2.2) 将变量按以下方式进行赋值，这显然具有线性最坏时间复杂度：

$$p\text{为}\begin{cases} \text{假} & ,\text{若 top}[p] < \text{top}[\neg p] \\ \text{真} & ,\text{若 top}[p] > \text{top}[\neg p] \end{cases}$$

用反证法证明此赋值一定是 2-CNF 的一个成真赋值：

假设这一赋值使得 2-CNF 中某个子句 $p \vee q$ 为假，那么 p, q 均为假，因此 $\text{top}[p] < \text{top}[\neg p]$ 且 $\text{top}[q] < \text{top}[\neg q]$。另一方面，$p \vee q$ 对应了 G 中 $\langle \neg p, q \rangle$ 和 $\langle p, \neg q \rangle$ 两条边，由 top 数组的定义知 $\text{top}[\neg p] \leq \text{top}[q]$ 且 $\text{top}[\neg q] \leq \text{top}[p]$。将以上四式连接即可推出矛盾。

(2)

(2.1) 如图7所示。

(2.2) 图7中涂黑的三个节点即为两两不相邻的节点，令这些节点所对应的文字为真即得到 3-CNF 的一个成真赋值：x, y, z 均为假。

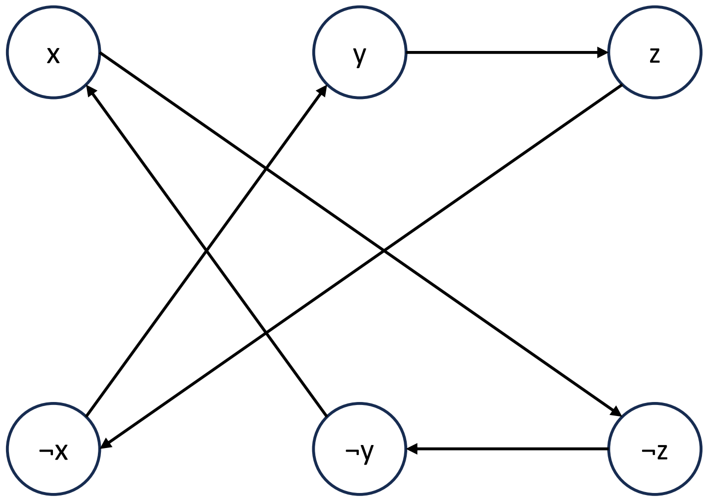

图 6: 2-CNF 有向图

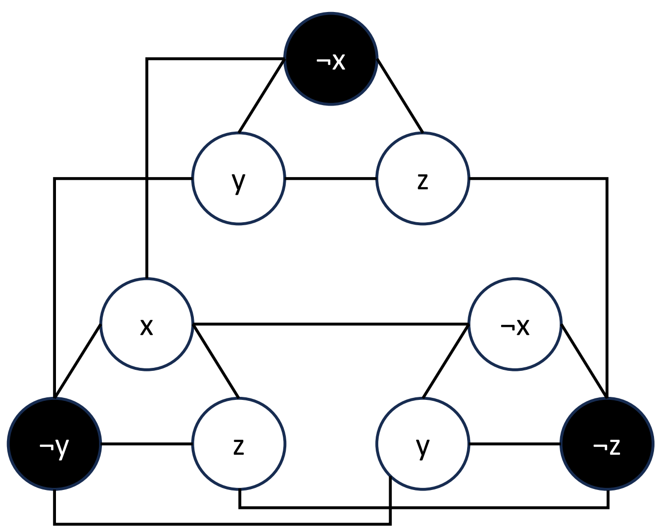

图 7: 3-CNF 无向图