HW8

Q1

(1)

本题不考虑图中存在负环的情况。

(1) 某同学在 Bellman-Ford 算法的基础上提出了一种基于队列的优化方案，并将其命名为“SPFA”(Shortest Path Faster Algorithm)，算法1给出了 SPFA 的伪代码：

Algorithm 1: Shortest Path Faster Algorithm

```
Input: G = ⟨V, E⟩, w : E → ℝ, s ∈ V
INITIALIZE-SINGLE-SOURCE(G, s);
Q ← {s};
while Q ≠ ∅ do
    u ← DEQUEUE(Q);
    foreach ⟨u, v⟩ ∈ G.E do
        if v can be relaxed by u then
            RELAX(u, v, w);
            if v ∉ Q then
                ENQUEUE(Q, v);
            end
        end
    end
end
```

(1.1)

(1.1) 我们将第一次 while 循环过程称为“第一轮”，第一轮中入队节点出队的全过程称为“第二轮”，以此类推。记 s 到 v 的所有最短路径中，所含节点数最少的路径的节点数为 $\varphi(s,v)$，试证明：s 到 v 的最短路径的长度能够在不超过第 $\varphi(s,v)-1$ 轮中被确定。

显然，此引理不仅直接导出了 SPFA 的正确性，同时也给出了 $O(|V||E|)$ 的最坏时间复杂度上界。

由数学归纳法，

当v = s时，s到v的最短路径长度能够在第1轮确定，此时$\varphi(s,s)=2$, 结论成立

设v的所有前驱结点集合为S，假设s到S的任意结点v'， 都有s到v'的最短路径的长度能够在不超过第$\varphi(s,v')-1$轮确定

对所有$v' \to v$的边做一遍松弛操作，取能够更新的边对应v'的集合设为S'，此时确定了s到v的最短路径长度，设s经过v'到达v所含的节点数最少，由归纳假设知，第$\varphi(s,v')-1$轮已经确定了s到v的最短路径长度，边$v' \to v$会在$\varphi(s,v')$轮，或$\varphi(s,v')-1$轮中进行relax，并确定s到v的最短路径长度

由$\varphi(s,v')=\varphi(s,v)-1$

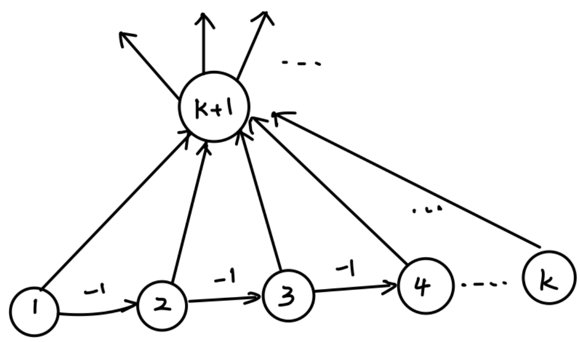
因此s到v的最短路径长度能够在不超过$\varphi(s,v)-1$轮中被确定

(1.2)

(*1.2) 尝试构造正权图列 $\{G_{n,m}\}, m \in \left[n-1, \frac{n(n-1)}{2}\right]$，满足：$|V_{n,m}| = \Theta(n), |E_{n,m}| = \Theta(m)$，使得 SPFA 在 $G_{n,m}$ 上的运行时间 $T(n,m) = \Omega(nm)$

提示：参考形如图1的图，构造恰当的边权使得节点 t 入队 $\Theta(n)$ 次。

这一构造说明：对于正权图而言，SPFA 在最坏情况下的性能严格劣于 Dijkstra 算法。



取$k=\frac{n}{2}$，对source = 1，$1 \to k+1$先入队的情况下，支配节点k+1会入队k次，也就是至少要做k轮循环遍历 即$\Theta(n)$轮，这时$T(n,m)=\Omega(nm)$

(2)

(2) 某同学在 Dijkstra 算法的基础上提出了一种能够处理有负权的图的变种算法，其伪代码见算法2。

Algorithm 2: Reentrant Dijkstra Algorithm

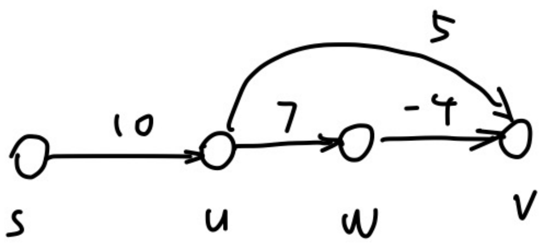
```
Input: G = ⟨V, E⟩, w : E → ℝ, s ∈ V
INITIALIZE-SINGLE-SOURCE(G, s);
S ← ∅;
Q ← G.V;
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q);
    S ← S ∪ {u};
    foreach ⟨u, v⟩ ∈ G.E do
        if v can be relaxed by u then
            RELAX(u, v, w);
            if v ∉ S then
                DECREASE-KEY(Q, v, v.d);
            else
                S ← S \ {v};
                INSERT(Q, v);
            end
        end
    end
end
```

(2.1)

(2.1) 对比算法2与原始 Dijkstra 算法，简要说明为什么算法2能够处理有非正权边的图。

原始Dijkstra算法无法处理负权边的原因是，dijkstra是基于贪心的策略，如果已经找到$u \to v$的最短边，会将v结点入队，这只是一种局部最优，若存在 $u \to w, w \to v$，虽然$u \to w$的权值比$u \to v$更大，但由于$w \to v$可以是负权边，因此可能后者是一个全局最优的策略，这时候dijkstra算法跑不出正确的结果。

算法2，对所有可以被松弛边<u, v>，如果v已经入队时，重新入队，也就是v可以被再访问一遍，因此可以更新负权边



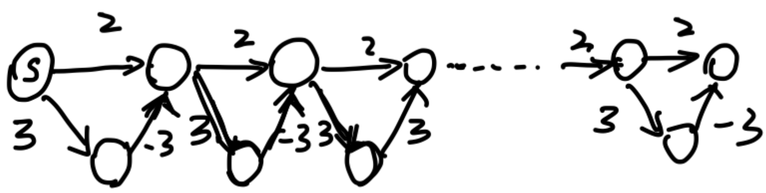
如上图，对于原始dijkstra，当$u \to v$之后，v被标记为访问，无法用$w \to v$去更新，如果采用算法2，这时候v会重新入队，因此可以被更新。

(2.2)

(*2.2) 尝试构造图列 $\{G_n\}$，满足：$|V_n| = \Theta(n), |E_n| = \Theta(n)$，使得算法2在 G_n 上的运行时间 $T(n) = \Omega(2^n)$

提示：考虑形如图2的图，构造恰当的边权使得算法运行过程中出现大量“回溯”。

这一构造说明：对于允许有非正权边的图，算法2在最坏情况下的性能严格劣于 Bellman-Ford 算法。



所有有边权为2的边的顶顶点分别入队2,4,8, ..., 2^{n-1}次，因此$T(n) = \Omega(2^n)$

Q2

(1)

给定图 $G=\langle V,E\rangle$ 以及深度优先森林 $G_\pi=\langle V,E_\pi\rangle$，记节点 v 在深度优先遍历中的发现时间戳为 $\mathrm{dfn}[v]$，G_π 中以 v 为根的子树的节点集为 S_v，S_v 中的节点经一条非树边可达的节点集为 T_v，T_v 中可达 v 的节点集为 P_v。定义：$\mathrm{low}[v]=\min\limits_{u\in\{v\}\cup P_v}\mathrm{dfn}[u]$

(1) 本问中的深度优先遍历以这样的顺序进行：起始点为节点 1，每当有多个节点可供选择时，总是按照编号从小到大的顺序进行选择。

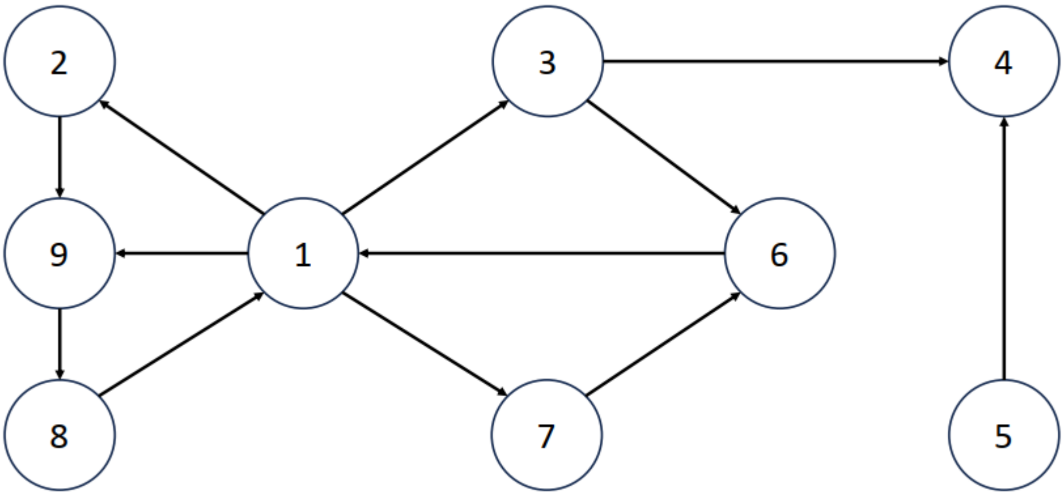


图 3: 示例图

(1.1)

对图3深度优先遍历

各结点的dfn值如下

$dfn[1]=1,dfn[2]=2,dfn[3]=5,dfn[4]=6,dfn[5]=9$

$dfn[6]=7,dfn[7]=8,dfn[8]=4.dfn[9]=3$

各结点的low值如下

$low[1]=1,low[2]=1,low[3]=1,low[4]=6,low[5]=6$

$low[6]=1,low[7]=7,low[8]=1,low[9]=1$

边的类型

$1\rightarrow 2:$ 树边$,1\rightarrow 9:$ 前向边$,1\rightarrow 3:$ 树边$,1\rightarrow 7:$ 树边

$2\rightarrow 9:$ 树边

$3\rightarrow 6:$ 树边$,3\rightarrow 4:$ 树边

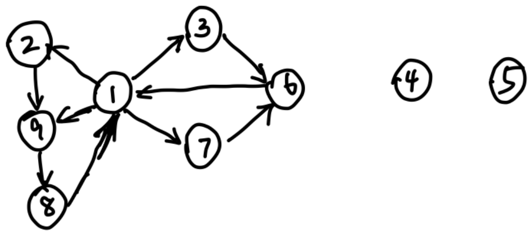
$5\rightarrow 4:$ 横向边

$6\rightarrow 1:$ 后向边

$7\rightarrow 6:$ 横向边

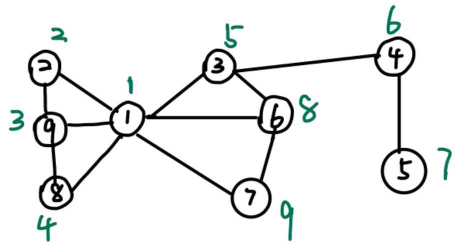
$8\rightarrow 1:$ 后向边

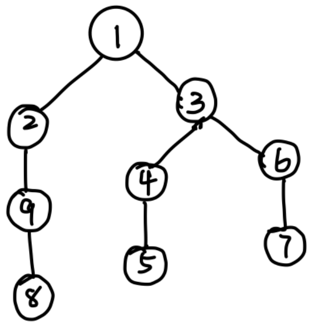
$9\rightarrow 8:$ 树边



(1.2)

(1.2) 将图3中的有向边改为无向边并进行深度优先遍历，标注出每条边属于哪种类型（树边、后向边、前向边、横向边），写出各节点的 dfn 值以及 low 值，找出图中的所有割点和桥。一个点（一条边）是割点（桥）是指：将它删除会导致图的连通分量数量增加。





dfn如下

$dfn[1]=1, dfn[2]=2, dfn[3]=5, dfn[4]=6, dfn[5]=7$

$dfn[6]=8, dfn[7]=9, dfn[8]=4, dfn[9]=3$

low如下

$low[1]=1, low[2]=1, low[3]=1, low[4]=6, low[5]=7$

$low[6]=1, low[7]=1, low[8]=1, low[9]=1$

树边：

$(1,2),(2,9),(9,8),(1,3),(3,4),(4,5),(3,6),(6,7)$

后向边：

$(1,9),(1,8),(1,6),(1,7)$

割点：1，3，4

桥：$(3,4),(4,5)$

(2)

(*2) 设计线性最坏时间复杂度的算法计算所有节点的 low 值。

提示：对于有向图，需要使用某种数据结构维护“已被发现的节点中，有哪些可达当前节点”。

```
void tarjan(int u)
{
    dfn[u] = low[u] = ++timestamp;
    stk[++top] = u, in_stk[u] = true;
    for(int i = h[u]; i != -1; i = ne[i])
    {
        int j = e[i];
        if(!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }
        else if(in_stk[j])
        {
            low[u] = min(low[u], dfn[j]);
        }
    }
}
```

邻接表存储一个图，用stk模拟栈实现DFS，同时in_stk表示是否访问过

对于low数组，其值等于所有它所有可达结点中low和本身dfn最小的，因此可以在dfs的过程中回溯的时候更新

(3)

(*3)low 数组可用于解决许多与连通性有关的问题。

(*3.1) 简要描述如何利用 low 数组找出有向图中的所有强连通分量。

(*3.2) 简要描述如何利用 low 数组找出无向图中的所有割点。

(*3.3) 简要描述如何利用 low 数组找出无向图中的所有桥。

(3.1)

结点u搜索完毕后，如果low[u] = dfn[u]，说明以u为根节点的所有字数上以及栈中在u内的元素构成了一个强联通分量

删除这些栈上元素，重复操作就可以得到所有强联通分量

```
if(dfn[u] == low[u])
{
    int y;
    ++scc_cnt;
    do{
        y = stk[top--];
        in_stk[y] = false;
        id[y] = scc_cnt;
    }while(y != u)
}
```

(3.2)

对于一个结点u，子结点v，如果$low(v) \geq dfn(u)$，说明u是一个割点

特殊的，对于根节点，如果存在两个以上的子结点，则u是割点

(3.3)

边(u,v)为桥当且仅当$low(v) > dfn(u)$，（u，v）是一个树边

Q3

本题中的“环”定义为“至少包含三条边的简单回路”，环的“大小”定义为边权和。本题仅考虑正权图的情况。

(2)

(2) 假设我们需要求出有向图中经过某个点的最小环。某同学注意到：任何一个经过 s 的环都是由一条入边 $\langle t, s\rangle$ 以及一条 s 到 t 的路径拼成的。

(2.1) 若图中不存在反向平行边，根据上述思路设计算法，并分析最坏时间复杂度。

(2.2) 若图中可能存在反向平行边，完善你在 (2.1) 中设计的算法，并分析最坏时间复杂度。

(2.1)

遍历所有s的入边<t,s>，对于每一条入边，由dijkstra算法求解s到t的最短路径，将入边权值与这条最短路径相加，求所有和中的最小值即可

dijkstra算法在用priority_queue优化的情况下时间复杂度是O（ElogV）的，一共要遍历O(E)次，因此时间复杂度为$O(E^2logV)$

(2.2)

在每次调用dijsktra算法的时候，删除<t,s>再调用，求出相应的入边权和最短路径和之后取最小值

Q4

(1)

布尔适定性问题与图论问题具有紧密的联系。

若一个合取范式的每个子句中恰有 k 个不同的“文字”，则称其为“k-CNF”。“文字”是指一个布尔变量或一个布尔变量的否定。k-SAT 问题是指：给定一个 k-CNF，判断它是否可满足（即是否存在成真赋值）。

例如：$(x \vee y) \wedge (\neg y \vee z) \wedge (\neg z \vee \neg x)$ 是一个含有 3 个子句的 2-CNF，x, y, z 是变量，$x, \neg x, y, \neg y, z, \neg z$ 是文字。当 x 为真，y, z 为假时，整个公式为真，因此它是可满足的。

(1) 任意一个 2-CNF 都可以被转化为有向图的形式，假设子句数量为 n，变量数量为 m。

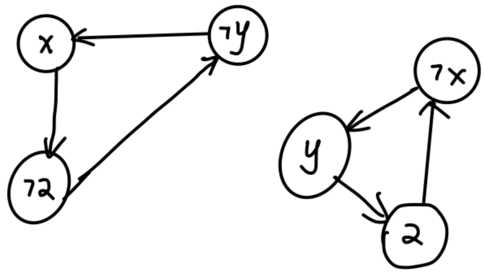
首先，对于每个子句 $p \vee q$，将其改写为 $(\neg p \to q) \wedge (\neg q \to p)$，这样就得到了一个由 $2n$ 个蕴含式相与所构成的布尔公式。

然后，建立一个含有 $2m$ 个节点的图，每个文字对应一个节点。对于前一步中每个子句 $p \to q$，在图中添加有向边 $\langle p, q \rangle$，当出现重边时，只保留其中一条。记所得到的有向图为 G。

(1.1)

其中一条。记所得到的有向图为 G。

(1.1) 将上面给出的 2-CNF 转化为有向图。



(2)

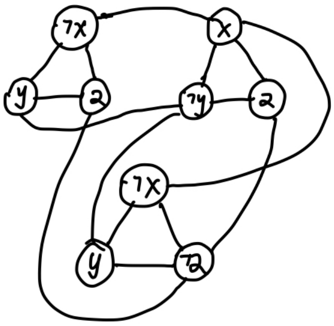
(2) 任意一个 3-CNF 都可以转化为无向图的形式，假设子句数量为 n。

首先，为每个子句中的每个文字建立一个节点，并且将同一个子句中的文字相连，此时图中一共有 n 个“三角形”。

然后，将不同子句中互为否定的文字相连，例如图中如果有 3 个 x 和 5 个 $\neg x$，且它们所在的子句各不相同，那么它们之间一共要连 15 条边。

(2.1)

(2.1) 将 $(\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$ 转化为无向图。



(2.2)

(2.2) 找出 (2.1) 所得到的图中的 3 个节点，使得它们两两不相邻，并通过这 3 个点给出原公式的一个成真赋值。

以上转化事实上构成了 3-SAT 问题到独立集问题的多项式时间归约。

$(\neg x \vee y \vee z)$中的$\neg x$

$(x \vee \neg y \vee z)$中的$\neg y$

$(\neg x \vee y \vee \neg z)$中的$\neg z$

令$\neg x = \neg y = \neg z = 1$，即$x = y = z = 0$

是原公式的一个成真赋值