Open topic
Bellman-Ford 算法与负环

张天昀

2018 年 11 月 5 日

南京大学计算机科学与技术系
171860508@smail.nju.edu.cn

Contents

1. 证明如果 G 从 s 出发能访问到负环，BF 算法一定能检测到负环。
2. 改造 BF 算法，输出图内所有能访问到的的负环。
3. 改造 BF 算法找出所有的负环。

BF 算法正确性证明

引理

如果 G 中含有负环，`Bellman-Ford` 算法的返回值是 `false`。

BF 算法正确性证明

引理

如果 G 中含有负环，`Bellman-Ford` 算法的返回值是 `false`。

假设 G 存在负环 $c = \langle v_0, v_1, \ldots, v_k \rangle$ $(v_0 = v_k)$。
根据负环的定义，有

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

用反证法：假设算法返回值是 `true`，那么 $\forall (u, v) \in E(G)$，

$$v.d \leq u.d + w(u, v).$$

所以 $\forall i$，

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$$

BF 算法正确性证明

将等式两边累加，得到

$$\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} v_{i-1} + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

由 $v_0 = v_k$，有 $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$。进而

$$0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

得到矛盾。 $\square$

输出找到的负环

基于刚才的证明可以推出：

输出找到的负环

基于刚才的证明可以推出：

On the $|V(G)|$-th pass

进行一轮额外的松弛操作，如果对于某条边 (u, v)，

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

那么 v 一定在负环上或者可以通过负环访问到。

输出找到的负环

基于刚才的证明可以推出：

On the $|V(G)|$-th pass

进行一轮额外的松弛操作，如果对于某条边 (u, v)，

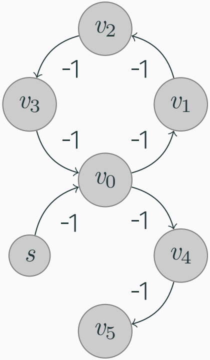
$$v.d > u.d + w(u, v)$$

那么 v 一定在负环上或者可以通过负环访问到。

找出这个负环

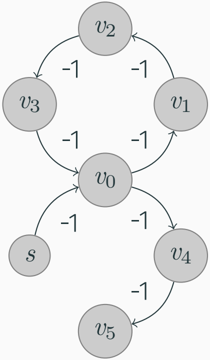
从 v 开始沿着 $v.d$ 的方向往回访问，直到某一个点（包括 v）出现了两次，就找到了这个负环。

图示

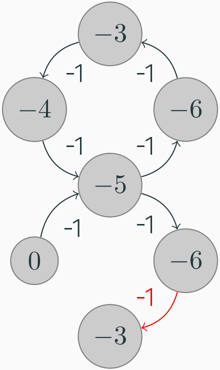


The original graph

图示

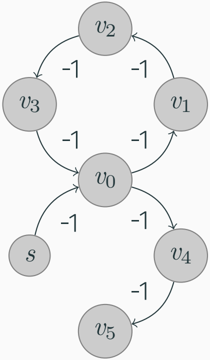


The original graph

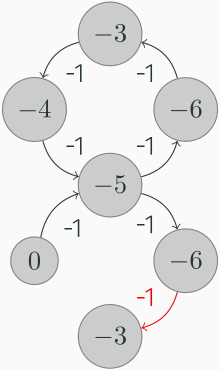


G after $|V(G)|-1$ passes

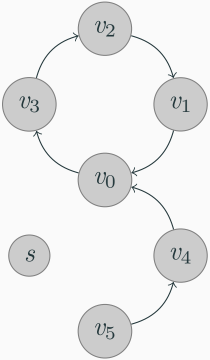
图示



The original graph

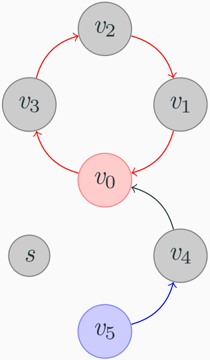


G after $|V(G)| - 1$ passes

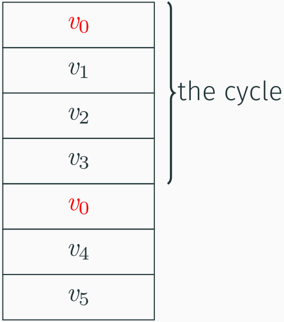


$G.\pi$ after $|V(G)| - 1$ passes

图示

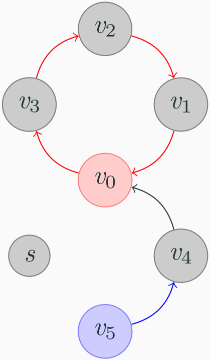


$G.\pi$ after $|V(G)|$ passes

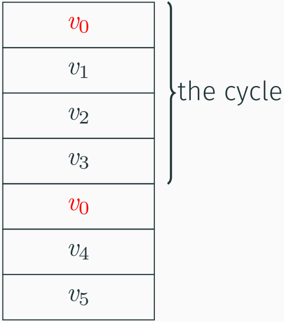


Visited nodes during the search

图示



$G.\pi$ after $|V(G)|$ passes



Visited nodes during the search

通过 `vis` 标记判断当前点是否已经访问过，从第二次访问到的点开始再进行一次循环就可以打印环上所有的点。

改造 BF 算法

Algorithm 1 Bellman-Ford with Negative Cycle Output

```
function BELLMAN-FORD-MOD(G, s, w)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for i = 1 to |V(G)| − 1 do
        for edge (u, v) ∈ E(G) do
            RELAX(u, v, w)
        end for
    end for
    for edge (u, v) ∈ E(G) do                ▷ One more pass
        if v.d > u.d + w(u, v) then
```

改造 BF 算法

Algorithm 2 Bellman-Ford with Negative Cycle Output

function BELLMAN-FORD-MOD(G, s, w)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for $i = 1$ to $|V(G)| - 1$ **do**
    for edge $(u, v) \in E(G)$ **do**
      RELAX(u, v, w)
    end for
  end for
  for edge $(u, v) \in E(G)$ **do** ▷ One more pass
    if $v.d > u.d + w(u, v)$ **then**
      $v.\pi = u$ ▷ What happens if a $|V|$-cycle?
      return NEGATIVE-CYCLE-AUX(v) ▷ Only one cycle
    end if
  end for
  return an empty vector
end function

改造 BF 算法

Algorithm 3 Output the negative cycle found with node v

```
function NEGATIVE-CYCLE-AUX(v)
    let c be a vector to hold the cycle
    let vis be an empty array of size |V(G)|
    create a pointer p initially pointed at v
```

改造 BF 算法

Algorithm 4 Output the negative cycle found with node v

```
function NEGATIVE-CYCLE-AUX(v)
    let c be a vector to hold the cycle
    let vis be an empty array of size |V(G)|
    create a pointer p initially pointed at v
    while vis[p] == false do            ▷ break at second visit
        vis[p] == true                  ▷ mark the pointed as visited
        p = p.π                         ▷ visit p's parent
    end while
```

改造 BF 算法

Algorithm 5 Output the negative cycle found with node v

function NEGATIVE-CYCLE-AUX(v)
  let c be a vector to hold the cycle
  let vis be an empty array of size $|V(G)|$
  create a pointer p initially pointed at v
  while $vis[p] ==$ `false` **do** ▷ break at second visit
    $vis[p] ==$ `true` ▷ mark the pointed as visited
    $p = p.\pi$ ▷ visit p's parent
  end while
  let $t = p$ and $p = p.\pi$
  while $p \neq t$ **do**
    push p into c
  end while
  push t into c and **return** c
end function

正确性证明

以上修改的 BF 算法只输出第一个被找到的负环。如果要找出所有的负环，可以增加一个标记数组 `inCycle`，如果当前的点属于一个负环且未被标记过，则调用函数输出这个负环并给环上的所有元素打上标记。

正确性证明

以上修改的 BF 算法只输出第一个被找到的负环。如果要找出所有的负环，可以增加一个标记数组 `inCycle`，如果当前的点属于一个负环且未被标记过，则调用函数输出这个负环并给环上的所有元素打上标记。

证明这个算法的正确性的关键在于：已知 G 中存在负环，且

$$\exists(u, v) \in E(G), v.d > u.d + w(u, v)$$

证明以下两个命题成立：

正确性证明

以上修改的 BF 算法只输出第一个被找到的负环。如果要找出所有的负环，可以增加一个标记数组 `inCycle`，如果当前的点属于一个负环且未被标记过，则调用函数输出这个负环并给环上的所有元素打上标记。

证明这个算法的正确性的关键在于：已知 G 中存在负环，且

$$\exists(u, v) \in E(G), v.d > u.d + w(u, v)$$

证明以下两个命题成立：

(a). 前驱图 $G.\pi$ 中的环一定是负环。

正确性证明

以上修改的 BF 算法只输出第一个被找到的负环。如果要找出所有的负环，可以增加一个标记数组 `inCycle`，如果当前的点属于一个负环且未被标记过，则调用函数输出这个负环并给环上的所有元素打上标记。

证明这个算法的正确性的关键在于：已知 G 中存在负环，且

$$\exists(u, v) \in E(G), v.d > u.d + w(u, v)$$

证明以下两个命题成立：

(a). 前驱图 $G.\pi$ 中的环一定是负环。

(b). 在 $G.\pi$ 中从 v 开始，沿着 π 往前找一定能找到环；

证明基础

推论 1 （由 Relax 函数定义推出）

对于任意节点 v，

$$v.\pi \neq \textsc{Nil} \Leftrightarrow \begin{cases} v.d < 0, & v == s, \\ v.d < +\infty, & \text{o.w.} \end{cases}$$

证明基础

推论 1（由 Relax 函数定义推出）

对于任意节点 v，

$$v.\pi \neq \text{Nil} \Leftrightarrow \begin{cases} v.d < 0, & v == s, \\ v.d < +\infty, & \text{o.w.} \end{cases}$$

推论 2

如果 $G.\pi$ 不含环且 $s.d = 0$，那么 $G.\pi$ 是一棵以 s 为根的树。

证明基础

推论 1（由 Relax 函数定义推出）

对于任意节点 v，

$$v.\pi \neq \text{NIL} \Leftrightarrow \begin{cases} v.d < 0, & v == s, \\ v.d < +\infty, & \text{o.w.} \end{cases}$$

推论 2

如果 $G.\pi$ 不含环且 $s.d = 0$，那么 $G.\pi$ 是一棵以 s 为根的树。

- 所有的松弛操作都不会破坏 $G.\pi$ 的联通性。
- 每次松弛操作后，$v.d < +\infty$ 的节点被加入 $G.\pi$。
- 由于 $s.d = 0$，有且仅有 s 并没有被 Relax 函数更新过，有且仅有 s 没有父节点。s 是树的根节点。

证明 (a)：所有环都是负环

假设 $G.\pi$ 中存在非负环 $c = \langle v_0, v_1, \ldots, v_k \rangle$ $(v_0 = v_k)$，
且最后一个被松弛加入 $G.\pi$ 的边是 (v_{k-1}, v_k)。

证明 (a)：所有环都是负环

假设 $G.\pi$ 中存在非负环 $c = \langle v_0, v_1, \ldots, v_k \rangle$ $(v_0 = v_k)$，
且最后一个被松弛加入 $G.\pi$ 的边是 (v_{k-1}, v_k)。

这条边被松弛的条件是

$$
\begin{aligned}
v_0 = v_k &> v_{k-1} + w(v_{k-1}, v_k) \\
&= (v_{k-2} + w(v_{k-2}, v_{k-1})) + w(v_{k-1}, v_k) \\
&= \cdots\cdots \\
&= v_0 + \sum_{i=1}^{k} w(v_{i-1}, v_i) > v_0
\end{aligned}
$$

矛盾。

证明 (b)：一定能找到环

由推论 1 可以推出以下结论：
假设沿着 $v.\pi$ 寻找没有找到环，说明寻找在且仅在 s 处停止且 $s.d = 0$。

证明 (b)：一定能找到环

由推论 1 可以推出以下结论：
假设沿着 $v.\pi$ 寻找没有找到环，说明寻找在且仅在 s 处停止且 $s.d = 0$。此时 $G.\pi$ 中寻找的路径就是 s 到 v 的最短路径，也就是 G 中 s 到 v 的简单最短路径。由于这个最短路径的每一段都已经被松弛过了，根据 Path-relaxation property，$v.d = \delta(s, v)$ 不可能再被更新了。与

$$\exists(u, v) \in E(G), v.d > u.d + w(u, v)$$

矛盾。

证明 (b)：一定能找到环

由推论 1 可以推出以下结论：
假设沿着 $v.\pi$ 寻找没有找到环，说明寻找在且仅在 s 处停止且 $s.d = 0$。此时 $G.\pi$ 中寻找的路径就是 s 到 v 的最短路径，也就是 G 中 s 到 v 的简单最短路径。由于这个最短路径的每一段都已经被松弛过了，根据 Path-relaxation property，$v.d = \delta(s, v)$ 不可能再被更新了。与

$$\exists (u, v) \in E(G), v.d > u.d + w(u, v)$$

矛盾。

同时，由于最长的环的长度是 $|V(G)|$，经过 $|V(G)|$ 轮松弛后所有的环的每一段都已经被松弛过加入 $G.\pi$ 了。所以如果搜索在 s 处都没有停止，就没有可以停下来的地方了，一定会找到环。

时间复杂度分析

- Bellman-Ford 的时间复杂度是 $O(|V||E|)$

时间复杂度分析

- Bellman-Ford 的时间复杂度是 $O(|V||E|)$
- 找环的时间复杂度是 $O(|V|) + O(|V|) = O(|V|)$

时间复杂度分析

- Bellman-Ford 的时间复杂度是 $O(|V||E|)$
- 找环的时间复杂度是 $O(|V|) + O(|V|) = O(|V|)$

总时间复杂度仍然是 $O(|V||E|)$

找出 G 中所有的负环

对于非强连通的图，要找出所有的负环，只需要：

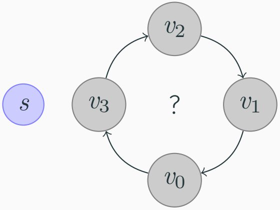
找出 G 中所有的负环

对于非强连通的图，要找出所有的负环，只需要：

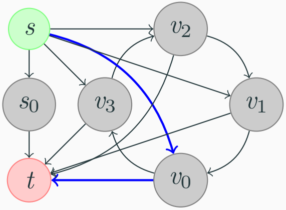
增加两个虚点 s 和 t，对于所有的点 v，连两条边 (s,v) 和 (v,t)，然后以 s 点为源点跑一遍修改过的 Bellman-Ford 即可。

找出 G 中所有的负环

对于非强连通的图，要找出所有的负环，只需要：



Unreachable

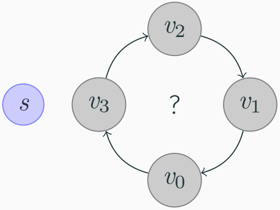


Reachable

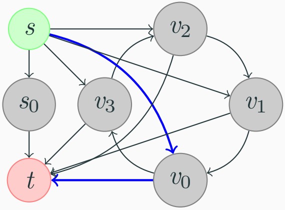
增加两个虚点 s 和 t，对于所有的点 v，连两条边 (s, v) 和 (v, t)，然后以 s 点为源点跑一遍修改过的 Bellman-Ford 即可。

找出 G 中所有的负环

对于非强连通的图，要找出所有的负环，只需要：



Unreachable



Reachable

增加两个虚点 s 和 t，对于所有的点 v，连两条边 (s, v) 和 (v, t)，然后以 s 点为源点跑一遍修改过的 Bellman-Ford 即可。

时间复杂度：$O(2|V|) + O(|V| \times (|E| + |V|)) = O(|V||E| + |V|^2)$。

Thanks

Q & A