

# 算法作业 9

2207070213 赖立勋

November 18, 2024

## 1 PPT 补完计划

### 1.1 问题描述

请证明：

1. 无向图的广度优先遍历中，不存在 BE；
2. 无向图的广度优先遍历中，不存在 DE。

### 1.2 解答

#### 1.2.1 1. 证明无向图 BFS 中不存在后向边 (Back Edge)

证明：

设存在后向边  $(u, v)$ ，其中  $v$  是  $u$  的祖先节点。我们通过反证法证明这是不可能的。

1. 若  $v$  是  $u$  的祖先，则在 BFS 中：
  - $v.dis < u.dis$  (BFS 按层次遍历的性质)
  - 具体地， $u.dis = v.dis + k$ ，其中  $k \geq 2$  (因为  $v$  是祖先而非父节点)
2. 然而，由于  $(u, v)$  是无向图中的边：
  - 当  $v$  被发现时， $u$  作为  $v$  的邻居节点应立即被发现
  - 此时必有  $u.dis = v.dis + 1$
3. 这与步骤 1 中  $u.dis = v.dis + k$  ( $k \geq 2$ ) 矛盾

因此，无向图 BFS 中不存在后向边。

#### 1.2.2 2. 证明无向图 BFS 中不存在前向边 (Descendant Edge)

证明：

设存在前向边  $(u, v)$ ，其中  $v$  是  $u$  的后代但非子节点。同样使用反证法。

1. 由于  $v$  是  $u$  的后代但非直接子节点：
  - $v.dis \geq u.dis + 2$
  - (因  $v$  至少要经过  $u$  的一个子节点才能到达)
2. 但在无向图中，由于  $(u, v)$  是一条边：
  - 当  $u$  被 BFS 访问时 (变为灰色)

- $v$  作为  $u$  的邻居会被立即检查
- 若  $v$  为白色，会立即入队，且  $v.dis = u.dis + 1$
- 若  $v$  已被发现，则  $v.dis \leq u.dis + 1$

3. 这与步骤 1 中  $v.dis \geq u.dis + 2$  矛盾

因此，无向图的 BFS 中不存在前向边。

## 2 图解算法

### 2.1 问题描述

对图 9-1，设源点是 3，请给出 BFS 之后，每个节点的 parent 和 dis 值。(默认采用数字序)

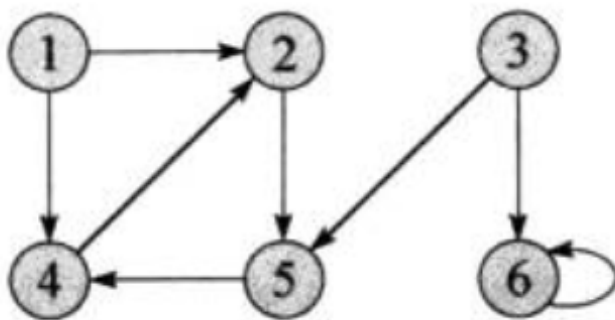


Figure 1: 图 9-1

对图 9-2，设源点是 u，请给出 BFS 之后，每个节点的 parent 和 dis 值。(默认采用字母序)

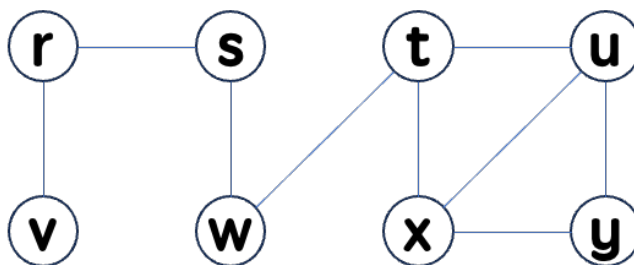


Figure 2: 图 9-2

## 2.2 解答

### 2.2.1 对图 9-1 执行 BFS 并确定每个节点的 parent 和 dis 值

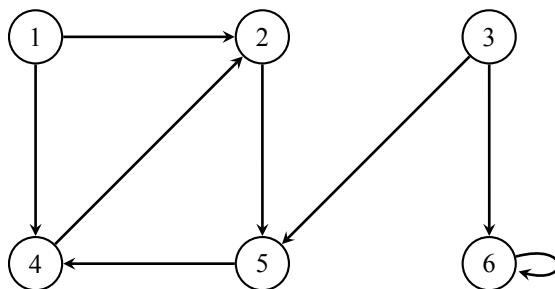


Figure 3: 图 9-1

#### 1. 初始化, 开始第一轮遍历:

- 所有节点染白, parent = -, dis =  $\infty$
- 3 入队, dis = 0

```
1  BFS-WALKER(G) // BFS遍历器
2  foreach v in [1,2,3,4,5,6] do
3      v.color:=WHITE;
4      v.parent=NULL
5      v.dis:=+∞;
6  foreach v in [3,1,2,4,5,6] do // 从3开始遍历
7      if v.color == WHITE then
8          BFS(v)
```

```
1  BFS(3)
2      Initialize an empty queue queNode;
3      3.color:=GRAY; // 3 染灰
4      3.dis:=0; // 3 dis=0
5      queNode.ENQUEUE(3); // 3 入队
```

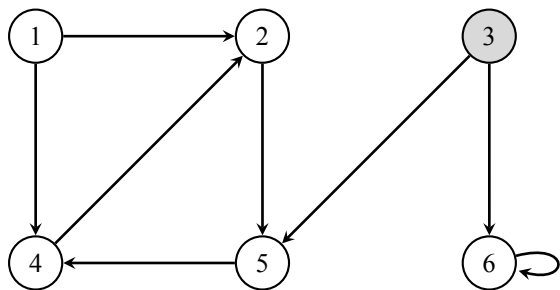


Figure 4: 图 9-1-0

节点编号	Parent	dis 值
1	-	$\infty$
2	-	$\infty$
3	-	0
4	-	$\infty$
5	-	$\infty$
6	-	$\infty$

Figure 5: 图 9-1-0 的 BFS 结果

2. 第一层 (dis = 0):

- 访问节点 3:
  - 邻居节点为 5 和 6。
  - 设置节点 5 和 6 的 dis = 1, parent = 3。
  - 将节点 5 和 6 入队。

```
1      while queNode = [3] != empty do
2          w:=queNode.DEQUEUE(); // 队头 3 出队, 赋给 w
3          foreach neighbor [5,6] of w = 3 do
4              if 5.color==WHITE then
5                  5.color:=GRAY; // 5 染灰
6                  5.parent:=3; // 5 parent=3
7                  5.dis:=3.dis+1; // 5 dis=1
8                  queNode.ENQUEUE(5); // 5 入队
9              if 6.color==WHITE then
10                 6.color:=GRAY; // 6 染灰
11                 6.parent:=3; // 6 parent=3
12                 6.dis:=3.dis+1; // 6 dis=1
13                 queNode.ENQUEUE(6); // 6 入队
14             <processing of node w>;
15             3.color:=BLACK; // 3 染黑
```

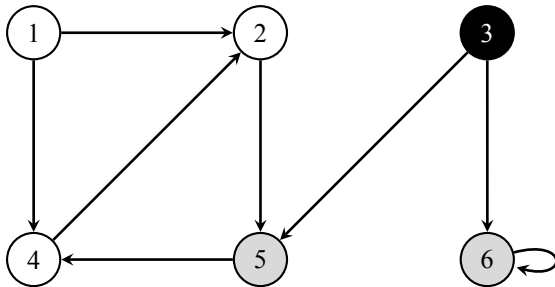


Figure 6: 图 9-1-1

节点编号	Parent	dis 值
1	-	$\infty$
2	-	$\infty$
3	-	0
4	-	$\infty$
5	3	1
6	3	1

Figure 7: 图 9-1-1 的 BFS 结果

3. 第二层 (dis = 1):

- 访问节点 5:
  - 邻居节点为 4。
  - 设置节点 4 的 dis = 2, parent = 5。
  - 将节点 4 入队。

```
1      while queNode = [5,6] != empty do // 数字序 5<6
2          w:=queNode.DEQUEUE(); // 队头 5 出队, 赋给 w
3          foreach neighbor [4] of w = 5 do
4              if 4.color==WHITE then
5                  4.color:=GRAY; // 4 染灰
6                  4.parent:=5; // 4 parent=5
7                  4.dis:=5.dis+1; // 4 dis=2
8                  queNode.ENQUEUE(4); // 4 入队
9          <processing of node 5>;
10         5.color:=BLACK; // 5 染黑
```

- 访问节点 6:
  - 邻居节点为自身, 无需操作。

```
1      while queNode = [6,4] != empty do
2          w:=queNode.DEQUEUE(); // 队头 6 出队, 赋给 w
3          foreach neighbor [6] of w = 6 do // 6 邻居是自己
4              if 6.color==WHITE then // False
5                  <processing of node 6>;
6          6.color:=BLACK; // 6 染黑
```

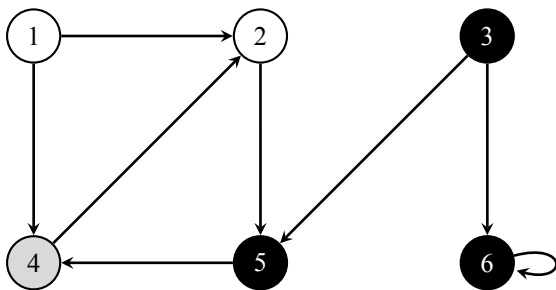


Figure 8: 图 9-1-2

节点编号	Parent	dis 值
1	-	$\infty$
2	-	$\infty$
3	-	0
4	5	2
5	3	1
6	3	1

Figure 9: 图 9-1-2 的 BFS 结果

4. 第三层 (dis = 2):

- 访问节点 4:
  - 邻居节点为 2。
  - 设置节点 2 的 dis = 3, parent = 4。
  - 将节点 2 入队。

```
1      while queNode = [4] != empty do
2          w:=queNode.DEQUEUE(); // 队头 4 出队, 赋给 w
3          foreach neighbor [2] of w = 4 do
4              if 2.color==WHITE then
5                  2.color:=GRAY; // 2 染灰
6                  2.parent:=4; // 2 parent=4
7                  2.dis:=4.dis+1; // 2 dis=3
8                  queNode.ENQUEUE(2); // 2 入队
9          <processing of node 4>;
10         4.color:=BLACK; // 4 染黑
```

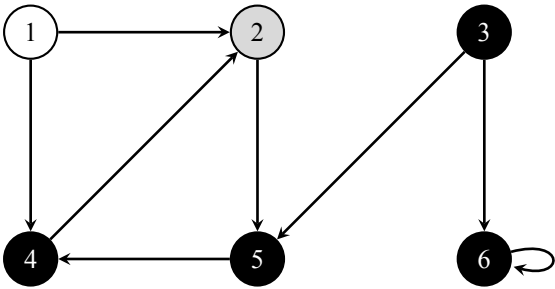


Figure 10: 图 9-1-3

节点编号	Parent	dis 值
1	-	$\infty$
2	4	3
3	-	0
4	5	2
5	3	1
6	3	1

Figure 11: 图 9-1-3 的 BFS 结果

5. 第四层 (dis = 3):

- 访问节点 2:
  - 邻居节点为 5, 5 已出队, 无需操作。

```
1 while queNode = [2] != empty do
2   w:=queNode.DEQUE(); // 队头 2 出队, 赋给 w
3   foreach neighbor [5] of w = 2 do
4     if 5.color==WHITE then // False
5       <processing of node 2>;
6     2.color:=BLACK; // 2 染黑
```

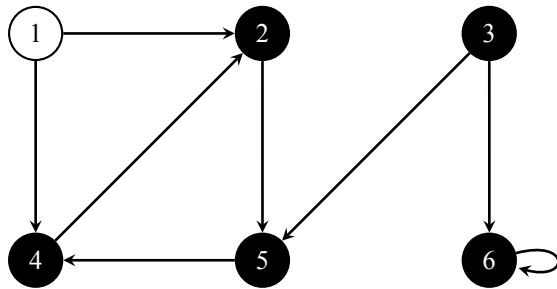


Figure 12: 图 9-1-4

节点编号	Parent	dis 值
1	-	$\infty$
2	4	3
3	-	0
4	5	2
5	3	1
6	3	1

Figure 13: 图 9-1-4 的 BFS 结果

6. 开始第二轮遍历:

- 1 入队, dis = 0

```
1 foreach v in [3,1,2,4,5,6] do // 1为白色
2   if v.color == WHITE then
3     BFS(v)
```

```
1 BFS(1)
2   Initialize an empty queue queNode;
3   1.color:=GRAY; // 1 染灰
4   1.dis:=0; // 1 dis=0
5   queNode.ENQUEUE(1); // 1 入队
```

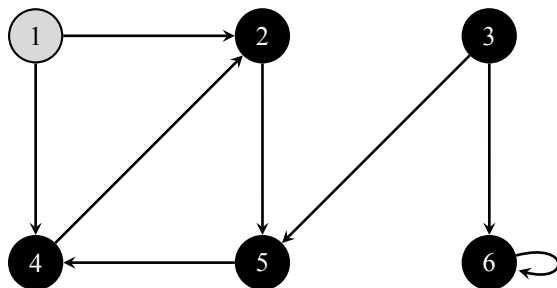


Figure 14: 图 9-1-5

节点编号	Parent	dis 值
1	-	0
2	4	3
3	-	0
4	5	2
5	3	1
6	3	1

Figure 15: 图 9-1-5 的 BFS 结果

7. 第二轮，第一层 (dis = 0):

- 访问节点 1:
  - 邻居节点为 2 和 4。2 和 4 为黑色，无需操作。

```
1 while queNode = [1] != empty do
2   w:=queNode.DEQUE(); // 队头 1 出队，赋给 w
3   foreach neighbor [2,4] of w = 1 do
4     if 2.color==WHITE then // 2.color=BLACK False
5     if 4.color==WHITE then // 4.color=BLACK False
6     <processing of node w>;
7     1.color:=BLACK; // 1 染黑
```

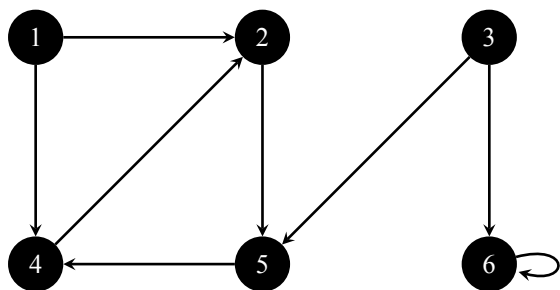


Figure 16: 图 9-1-6

节点编号	Parent	dis 值
1	-	0
2	4	3
3	-	0
4	5	2
5	3	1
6	3	1

Figure 17: 图 9-1-6 的 BFS 结果

8. 结束:

- 队列为空，while 循环结束。
- 外层节点全黑，BFS 结束。

```
1 while queNode = [] != empty do // queNode = [] False

1 foreach v in [3,1,2,4,5,6] do //全部为黑色
2   if v.color == WHITE then
3     BFS(v)
```

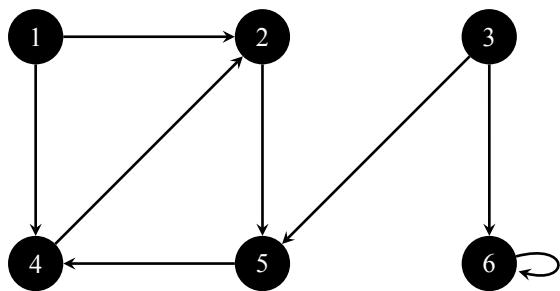


Figure 18: 图 9-1-7

节点编号	Parent	dis 值
1	-	0
2	4	3
3	-	0
4	5	2
5	3	1
6	3	1

Figure 19: 图 9-1-7 的 BFS 结果



2.2.2 对图 9-2 执行 BFS 并确定每个节点的 parent 和 dis 值

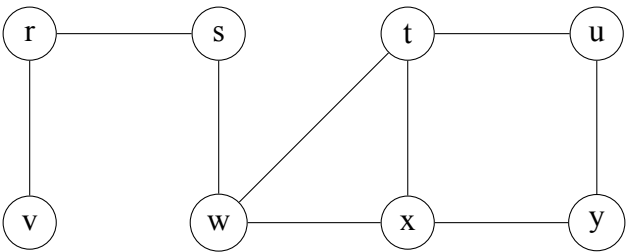


Figure 20: 图 9-2

1. 初始化:

- 所有节点染白, parent = -, dis =  $\infty$
- u 入队, dis = 0

```
1  BFS-WARRER(G)
2  foreach node v in [r,x,t,u,v,w,x,y] do
3      v.color:=WHITE;
4      v.parent=NULL
5      v.dis:=+∞;
6  foreach node v in [u,r,x,t,v,w,x,y] do
7      if v.color == WHITE then
8          BFS(v)
9          BFS(u)
10      Initialize an empty queue queNode;
11      u.color:=GRAY; // u 染灰
12      u.dis:=0; // u dis=0
13      queNode.ENQUEUE(u); // u 入队
```

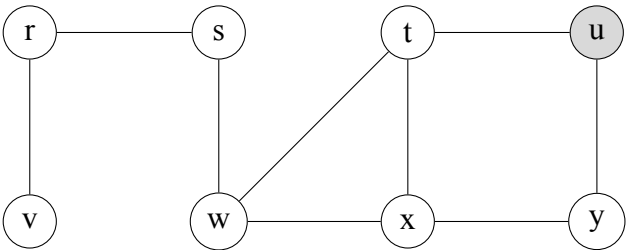


Figure 21: 图 9-2-0

节点	Parent	dis 值
r	-	$\infty$
s	-	$\infty$
t	-	$\infty$
u	-	0
v	-	$\infty$
w	-	$\infty$
x	-	$\infty$
y	-	$\infty$

Figure 22: 图 9-2-0 的 BFS 结果

2. 第一层 (dis = 0):

- 访问节点  $u$ , 发现邻居  $t$  和  $y$
- 设置  $t$  和  $y$  的  $dis$  值为 1,  $parent$  为  $u$

```
1      while queNode = [u] != empty do
2          w1:=queNode.DEQUEUE(); // 队头  $u$  出队, 赋给  $w1$ 
3          foreach neighbor [t,y] of w1 = u do
4              if t.color==WHITE then
5                  t.color:=GRAY; //  $t$  染灰
6                  t.parent:=u; //  $t$  parent= $u$ 
7                  t.dis:=u.dis+1; //  $t$  dis=1
8                  queNode.ENQUEUE(t); //  $t$  入队
9              if y.color==WHITE then
10                 y.color:=GRAY; //  $y$  染灰
11                 y.parent:=u; //  $y$  parent= $u$ 
12                 y.dis:=u.dis+1; //  $y$  dis=1
13                 queNode.ENQUEUE(y); //  $y$  入队
14         <processing of node  $u$ >;
15         u.color:=BLACK; //  $u$  染黑
```

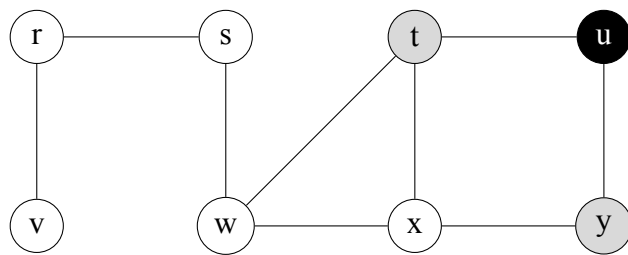


Figure 23: 图 9-2-1

节点	Parent	dis 值
r	-	$\infty$
s	-	$\infty$
t	u	1
u	-	0
v	-	$\infty$
w	-	$\infty$
x	-	$\infty$
y	u	1

Figure 24: 图 9-2-1 的 BFS 结果

3. 第二层 (dis = 1):

- 访问 t: 发现 w 和 x, 设置它们的 dis 值为 2, parent 为 t

```
1      while queNode = [t,y] != empty do
2          w1:=queNode.DEQUEUE(); // 队头 t 出队, 赋给 w1
3          foreach neighbor [u,w,x] of w1 = t do
4              if u.color==WHITE then // u.color=BLACK False
5                  if w.color==WHITE then
6                      w.color:=GRAY; // w 染灰
7                      w.parent:=t; // w parent=t
8                      w.dis:=t.dis+1; // w dis=2
9                      queNode.ENQUEUE(w); // w 入队
10                 if x.color==WHITE then
11                     x.color:=GRAY; // x 染灰
12                     x.parent:=t; // x parent=t
13                     x.dis:=t.dis+1; // x dis=2
14                     queNode.ENQUEUE(x); // x 入队
15                 <processing of node t>;
16                 t.color:=BLACK; // t 染黑
```

- 访问 y: 邻居均已访问或在队列中

```
1      while queNode = [y,w,x] != empty do
2          w1:=queNode.DEQUEUE(); // 队头 y 出队, 赋给 w1
3          foreach neighbor [u,x] of w1 = y do
4              if u.color==WHITE then // u.color=BLACK False
5                  if x.color==WHITE then // x.color=GRAY False
6                      <processing of node y>;
7                      y.color:=BLACK; // y 染黑
```

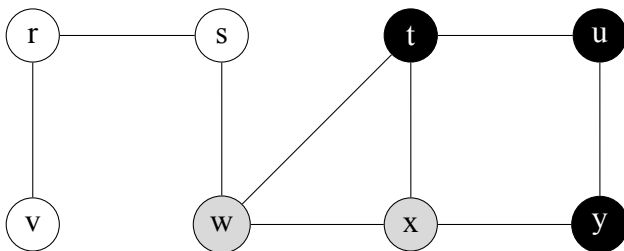


Figure 25: 图 9-2-1

节点	Parent	dis 值
r	-	$\infty$
s	-	$\infty$
t	u	1
u	-	0
v	-	$\infty$
w	t	2
x	t	2
y	u	1

Figure 26: 图 9-2-1 的 BFS 结果

4. 第三层 (dis = 2):

- 访问 w: 发现 s, 设置其 dis 值为 3, parent 为 w

```
1      while queNode = [w,x] != empty do
2          w1:=queNode.DEQUEUE(); // 队头 w 出队, 赋给 w1
3          foreach neighbor [s,t,x] of w1 = w do
4              if s.color==WHITE then
5                  s.color:=GRAY; // s 染灰
6                  s.parent:=t; // s parent=w
7                  s.dis:=t.dis+1; // s dis=2
8                  queNode.ENQUEUE(s); // s 入队
9              if t.color==WHITE then // t.color=BLACK False
10             if x.color==WHITE then // x.color=GRAY False
11             <processing of node w>;
12             w.color:=BLACK; // w 染黑
```

- 访问 x: 邻居均已访问

```
1      while queNode = [x,s] != empty do
2          w1:=queNode.DEQUEUE(); // 队头 x 出队, 赋给 w1
3          foreach neighbor [t,w,y] of w1 = x do
4              if t.color==WHITE then // t.color=BLACK False
5              if w.color==WHITE then // w.color=BLACK False
6              if y.color==WHITE then // y.color=BLACK False
7              <processing of node x>;
8              x.color:=BLACK; // x 染黑
```

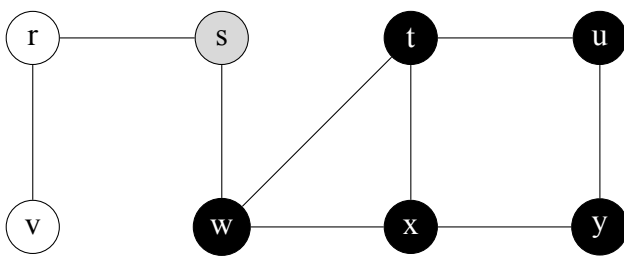


Figure 27: 图 9-2-3

节点	Parent	dis 值
r	-	$\infty$
s	w	3
t	u	1
u	-	0
v	-	$\infty$
w	t	2
x	t	2
y	u	1

Figure 28: 图 9-2-3 的 BFS 结果

5. 第四层 (dis = 3):

- 访问 s: 发现 r, 设置其 dis 值为 4, parent 为 s

```
1      while queNode = [s] != empty do
2          w1:=queNode.DEQUEUE(); // 队头 s 出队, 赋给 w1
3          foreach neighbor [r,w] of w1 = s do
4              if r.color==WHITE then
5                  r.color:=GRAY; // r 染灰
6                  r.parent:=s; // r parent=s
7                  r.dis:=s.dis+1; // r dis=4
8                  queNode.ENQUEUE(r); // r 入队
9              if w.color==WHITE then // w.color=BLACK False
10                 <processing of node s>;
11                 s.color:=BLACK; // s 染黑
```

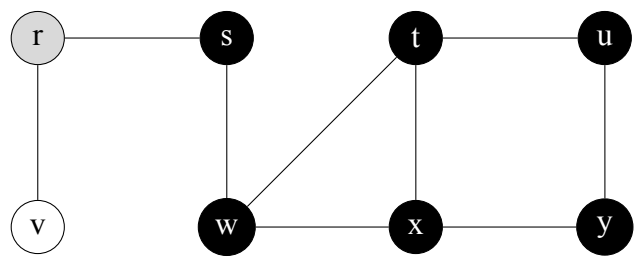


Figure 29: 图 9-2-4

节点	Parent	dis 值
r	s	4
s	w	3
t	u	1
u	-	0
v	r	5
w	t	2
x	t	2
y	u	1

Figure 30: 图 9-2-4 的 BFS 结果

6. 第五层 (dis = 4):

- 访问 r: 发现 v, 设置其 dis 值为 5, parent 为 r

```
1      while queNode = [r] != empty do
2          w1:=queNode.DEQUEUE(); // 队头 r 出队, 赋给 w1
3          foreach neighbor [s,v] of w1 = r do
4              if s.color==WHITE then // s.color=BLACK False
5                  if v.color==WHITE then
6                      v.color:=GRAY; // v 染灰
7                      v.parent:=r; // v parent=r
8                      v.dis:=r.dis+1; // v dis=5
9                      queNode.ENQUEUE(v); // v 入队
10         <processing of node r>;
11         r.color:=BLACK; // r 染黑
```

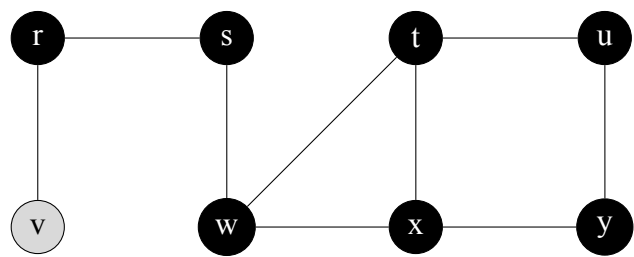


Figure 31: 图 9-2-5

节点	Parent	dis 值
r	s	4
s	w	3
t	u	1
u	-	0
v	r	5
w	t	2
x	t	2
y	u	1

Figure 32: 图 9-2-5 的 BFS 结果

7. 第六层 (dis = 5):

- 访问 v: 邻居均已访问

```
1      while queNode = [v] != empty do
2          w1:=queNode.DEQUEUE(); // 队头 v 出队，赋给 w1
3          foreach neighbor [r] of w1 = v do
4              if r.color==WHITE then // r.color=BLACK False
5                  <processing of node v>;
6              v.color:=BLACK; // v 染黑
```

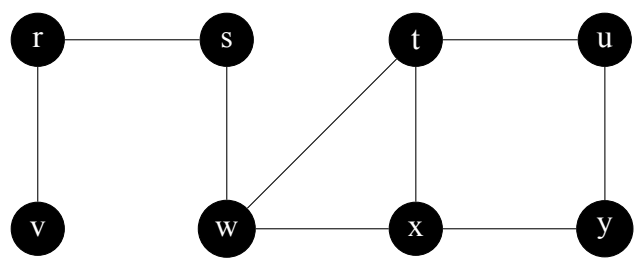


Figure 33: 图 9-2-6

节点	Parent	dis 值
r	s	4
s	w	3
t	u	1
u	-	0
v	r	5
w	t	2
x	t	2
y	u	1

Figure 34: 图 9-2-6 的 BFS 结果

8. 结束

- 队列为空，while 循环结束。
- 外层节点全黑，BFS 结束。

```
1      while queNode = [] != empty do // queNode = [] False

1      foreach v in [u,r,s,t,v,w,x,y] do //全部为黑色
2          if v.color == WHITE then
3              BFS(v)
```

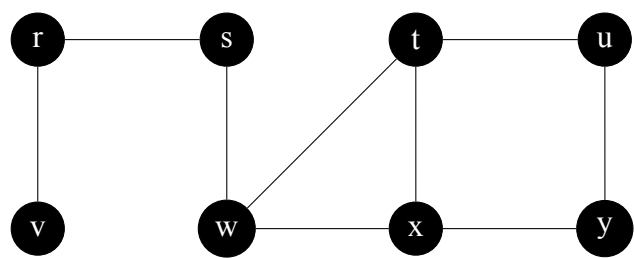


Figure 35: 图 9-2-7

节点	Parent	dis 值
r	s	4
s	w	3
t	u	1
u	-	0
v	r	5
w	t	2
x	t	2
y	u	1

Figure 36: 图 9-2-7 的 BFS 结果

### 3 有环图的检测

#### 3.1 问题描述

在 PPT 给出的 DFS 和 BFS 框架中，如何解决检测一个图是否存在环的问题？

#### 3.2 解答



### 3.2.1 PPT 给出的 DFS 和 BFS 框架

#### 1.DFS

```
1  DFS(v)
2      v.color:=GRAY
3      <Preorder processing of node v>
4      foreach neighbor w of v do
5          if w.color = WHITE then
6              <Exploratory processing of edge vw>
7              DFS(w)
8              <Backtrack processing of edge vw>
9          else
10             <Checking edge vw>
11             <Postorder processing of node v>
12             v.color:=BLACK
```

#### 2.BFS

```
1  BFS-WARRER(G) // 应为 BFS-WALKER 广度优先搜索遍历器
2  foreach node v in G do
3      v.color:=WHITE;
4      v.parent=NULL
5      v.dis:=+∞;
6  foreach node v in G do
7      if v.color == WHITE then
8          BFS(v)
```

```
1      BFS(G)
2          Initialize an empty queue queNode;
3          v.color:=GRAY;
4          v.dis:=0;
5          queNode.QUEUE(); // 应为 ENQUEUE 同13行
6          while queNode != empty do
7              w:=queNode.DEQUEUE();
8              foreach neighbor x of w do
9                  if x.color==WHITE then
10                     x.color:=GRAY;
11                     x.parent:=w;
12                     x.dis:=w.dis+1;
13                     queNode.ENQUEUE(x);
14             <processing of node w>; //我之前的缩进错了
15             w.color:=BLACK;
```

### 3.2.2 框架中检测环的方法

#### 1.DFS

适用有向图，无向图：

```
1  DFS_WALKER(G) // DFS遍历器
2  foreach v in G do
3      v.color := WHITE
4  foreach v in G do
5      if v.color == WHITE then
6          DFS(v)
```

```
1  DFS(v)
2      v.color:=GRAY
3      <Preorder processing of node v>
4      foreach neighbor w of v do
5          if w.color = WHITE then
6              <Exploratory processing of edge vw>
7              DFS(w)
8              <Backtrack processing of edge vw>
9          else
10             <Checking edge vw>
```

```
1          if w.color == GRAY then
2              // 发现后向边，存在环
3              return true
```

```
1      <Postorder processing of node v>
2      v.color:=BLACK
```

## 2.BFS

### 1. 有向图:

```
1  BFS-WALKER(G) // BFS遍历器
2  foreach node v in G do
3      v.color:=WHITE;
4      v.parent=NULL
5      v.dis:=+∞;
6  foreach node v in G do
7      if v.color == WHITE then
8          BFS(v)
```

```
1      BFS(G)
2          Initialize an empty queue queNode;
3          v.color:=GRAY;
4          v.dis:=0;
5          queNode.ENQUEUE(v);
6          while queNode != empty do
7              w:=queNode.DEQUEUE();
8              foreach neighbor x of w do
9                  if x.color==WHITE then
10                     x.color:=GRAY;
11                     x.parent:=w;
12                     x.dis:=w.dis+1;
13                     queNode.ENQUEUE(x);
```

```
1          else if x.color == BLACK then
2              // 可能是另一棵树的黑色节点
3              // 获取并比较两个节点的所有祖先
4              if hasCommonAncestor(x, w) then
5                  // 发现环
```

```
1          <processing of node w>;
2          w.color:=BLACK;
```

```
1  hasCommonAncestor(node1, node2)
2      ancestors := new Set()
3      cur := node1
4      while cur.parent != NULL do
5          ancestors.add(cur.parent)
6          cur := cur.parent
7
8      cur := node2
9      while cur.parent != NULL do
10         if cur.parent in ancestors then
```

```

11         return true
12     cur := cur.parent
13
14     return false

```

## 2. 无向图:

```

1  BFS-WALKER(G) // BFS遍历器
2      foreach node v in G do
3          v.color:=WHITE;
4          v.parent=NULL
5          v.dis:=+∞;
6  foreach node v in G do
7      if v.color == WHITE then
8          BFS(v)

```

```

1  BFS(G)
2      Initialize an empty queue queNode;
3      v.color:=GRAY;
4      v.dis:=0;
5      queNode.ENQUEUE(v);
6      while queNode != empty do
7          w:=queNode.DEQUEUE();
8          foreach neighbor x of w do
9              if x.color==WHITE then
10                 x.color:=GRAY;
11                 x.parent:=w;
12                 x.dis:=w.dis+1;
13                 queNode.ENQUEUE(x);

```

```

1          else if x.color == GRAY then
2              // 发现交叉边，存在环
3              return true

```

```

1      <processing of node w>;
2      w.color:=BLACK;

```

## 4 DFS 与 BFS 生成树

### 4.1 问题描述

给定一个连通图  $G = \langle V, E \rangle$ ，和一个顶点  $u \in V$ 。假设已经找到了一棵以  $u$  为根节点的 DFS 树  $T$ ，又找到了一棵以  $u$  为根节点的 BFS 树  $T'$ ，满足  $T = T'$ 。

1. 如果  $G$  是一个无向图，请证明  $G = T$ （也就是说，图的所有边都是 TE）；
2. 如果  $G$  是一个有向图，整个结论是否还成立？

### 4.2 解答

解析：

通过分析有向图和无向图在 DFS 和 BFS 中可能出现的边的类型，我们可以得出结论。

#### 1. 无向图的情况（证明 $G = T$ ）

- 对于无向图  $G$ ：
  - DFS 中只可能存在 TE 和 BE
  - BFS 中只可能存在 TE 和 CE
- 若  $G$  在 DFS 中存在 BE：
  - 该 BE 边在同根节点的 BFS 中一定会被作为 TE 边
  - 这将导致 DFS 树不等于 BFS 树
- 因此：
  - $G$  在 DFS 中不能存在 BE
  - $G$  只能包含 TE
  - 所以  $G = T$

#### 2. 有向图的情况（结论不成立）

- 对于有向图  $G$ ：
  - DFS 中四种类型边（TE、BE、FE、CE）均可能存在
  - BFS 中除了 DE 外的边（TE、BE、CE）都可能存在
- 若  $G$  在 DFS 中存在 BE 或 CE：
  - 在 BFS 中同样会存在 BE 和 CE
  - 这种情况下  $G \neq T$
- 因此结论在有向图中不成立