第2章 连通和遍历

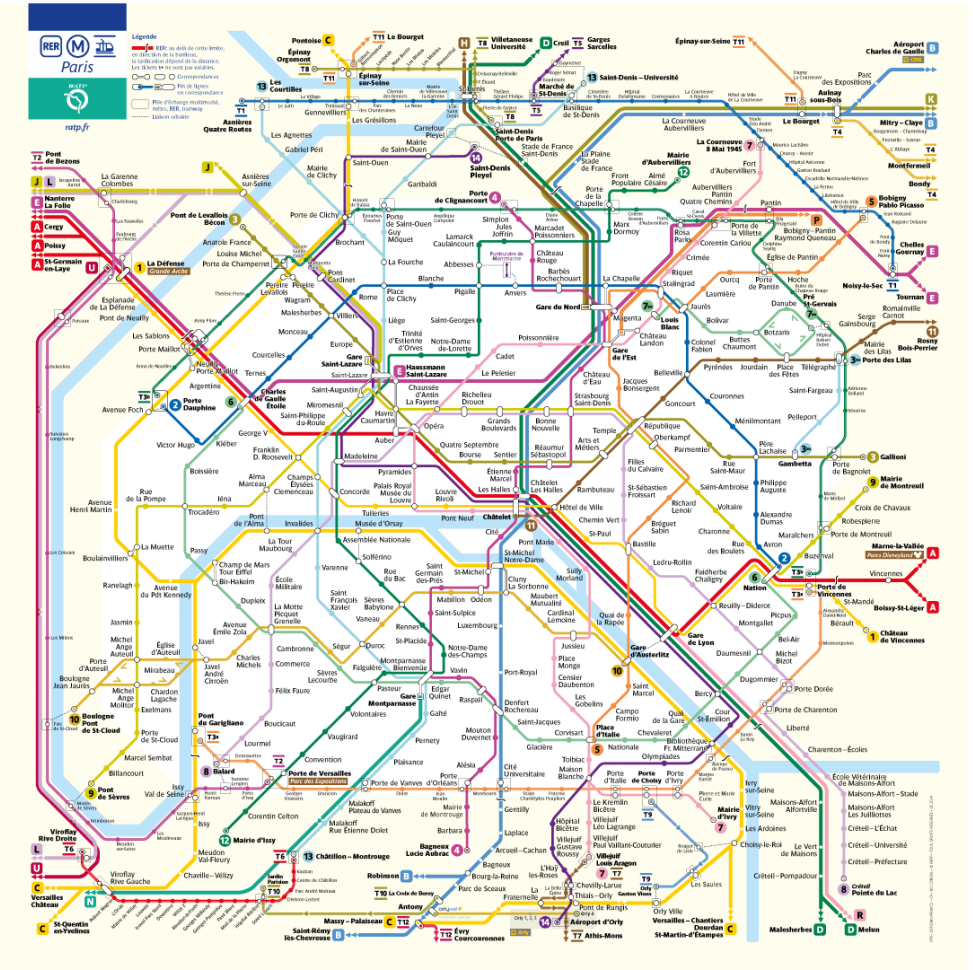
程龚
南京大学 计算机科学与技术系
gcheng@nju.edu.cn
http://ws.nju.edu.cn/~gcheng

本章内容

- 第2.1节 连通和DFS
- 第2.2节 割点和割边
- 第2.3节 距离和BFS
  - 第2.3.1节 理论
  - **第2.3.2节 算法**

如何计算两个顶点间的距离？如何计算一个顶点和图中所有顶点间的距离？

- 至少需要坐几站？



https://www.ratp.fr/plan-de-ligne/img/metro/Plan-Metro.1722015166.png

BFS算法（宽度优先搜索算法）

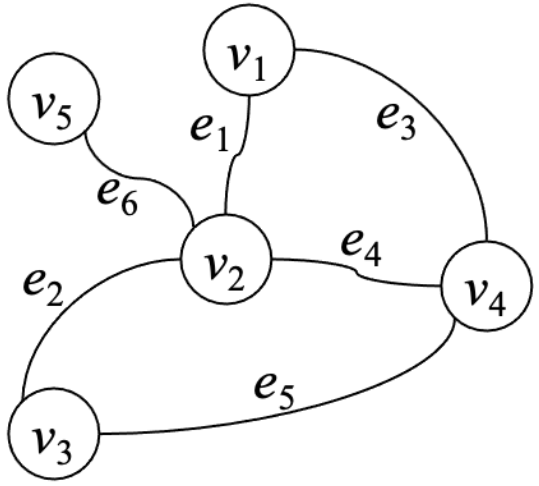
- 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 宽度优先：遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



BFS算法

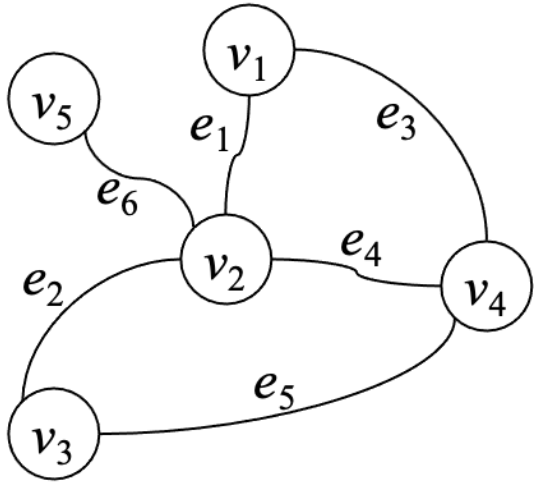
- 每个顶点的visited属性：布尔型变量，表示该顶点是否被访问过

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



BFS算法

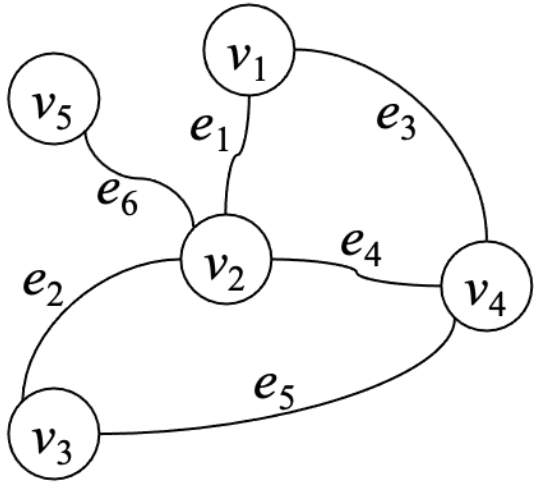
- 每个顶点的visited属性：布尔型变量，表示该顶点是否被访问过
- 每个顶点的d属性：整数型变量，初值为∞，表示该顶点和u间的距离

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



BFS算法

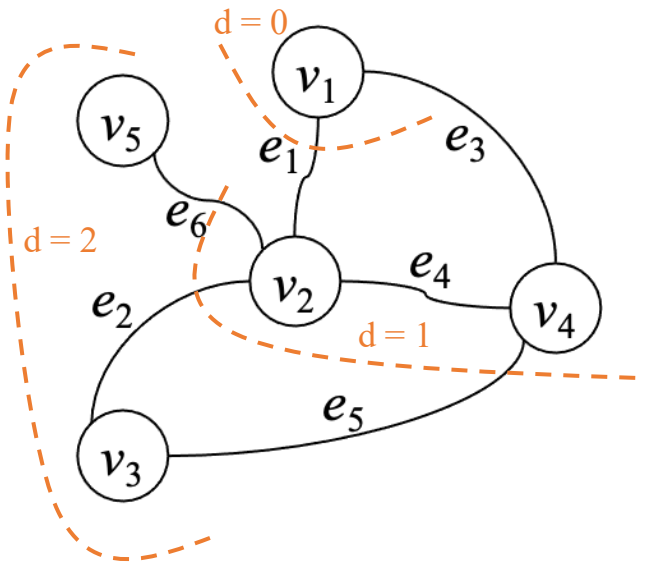
- 基本思路：由近及远，访问每个未被访问过的邻点

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



BFS算法

- 基本思路：由近及远，访问每个未被访问过的邻点
  - 每个顶点被访问后，被增加到一个初值为空的队列Q的队尾

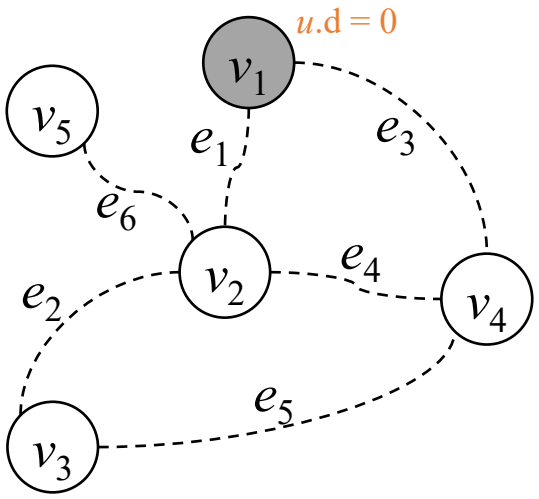
算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```

队列Q | v_1 | | | |



BFS算法

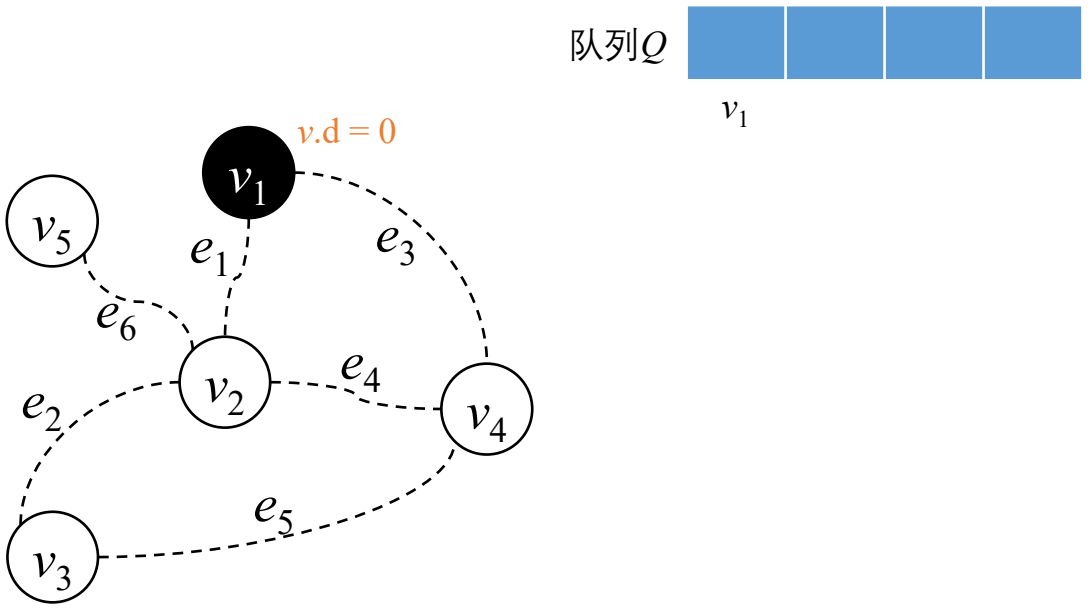
- 基本思路：由近及远，访问每个未被访问过的邻点
  - 每个顶点被访问后，被增加到一个初值为空的队列Q的队尾
  - 每轮while循环让排在Q的队首的顶点v出队列，直至Q为空

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



BFS算法

- 基本思路：由近及远，访问每个未被访问过的邻点
  - 每个顶点被访问后，被增加到一个初值为空的队列Q的队尾
  - 每轮while循环让排在Q的队首的顶点v出队列，直至Q为空
    - 对v的每个邻点w，若w未被访问过，则访问w，并将v的d属性值加1作为w的d属性值，将w增加到Q的队尾

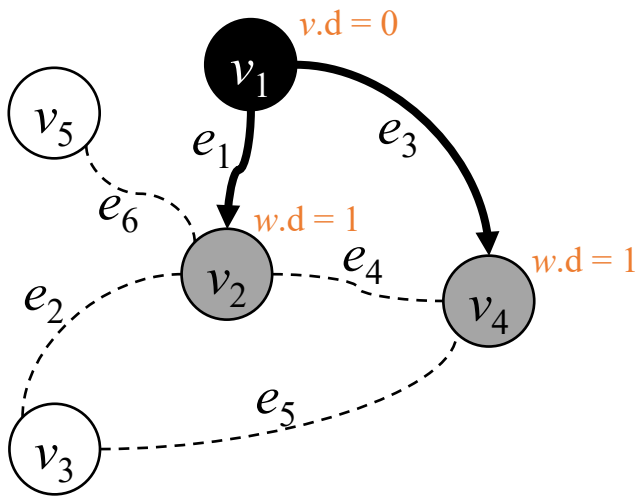
算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```

队列Q | v_2 | v_4 | | |



BFS算法

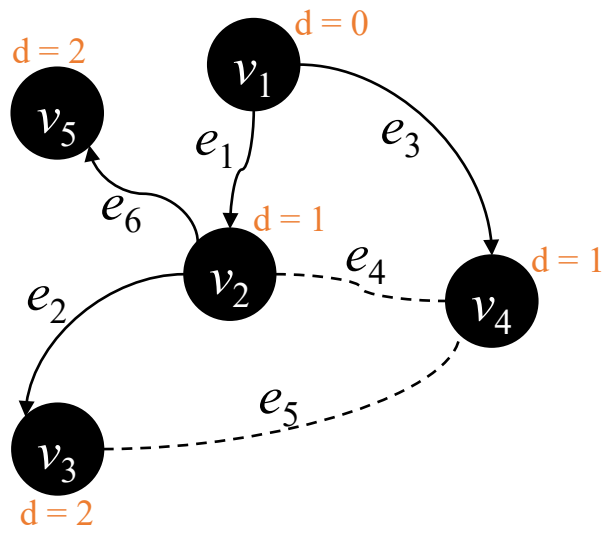
- 基本思路：由近及远，访问每个未被访问过的邻点
  - 每个顶点被访问后，被增加到一个初值为空的队列Q的队尾
  - 每轮while循环让排在Q的队首的顶点v出队列，直至Q为空
    - 对v的每个邻点w，若w未被访问过，则访问w，并将v的d属性值加1作为w的d属性值，将w增加到Q的队尾
    - 算法运行结束时，和u连通的所有顶点都被访问过，其visited属性值为true，其d属性值为其和u间的距离

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



BFS算法

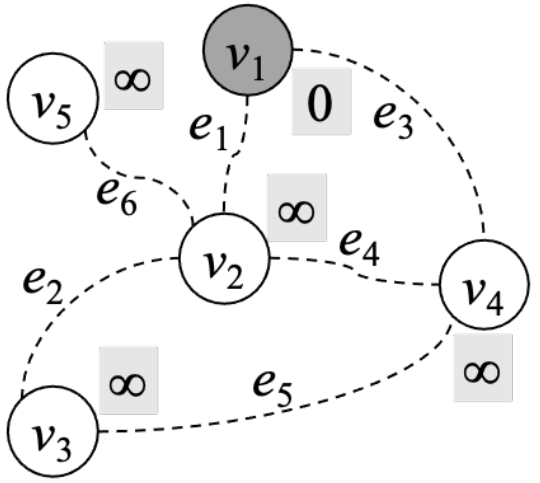
- 例如：从v_1出发，调用BFS(G, v_1)
  - v_1.visited ← true
  - v_1.d ← 0
  - v_1入队列

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



队列Q 

白色顶点：未入队列
灰色顶点：队列中
黑色顶点：已出队列
数字：d属性值

虚线：未发生入队列操作的相邻顶点
粗实线箭头：下一步入队列的邻点
细实线箭头：已入队列的邻点

BFS算法

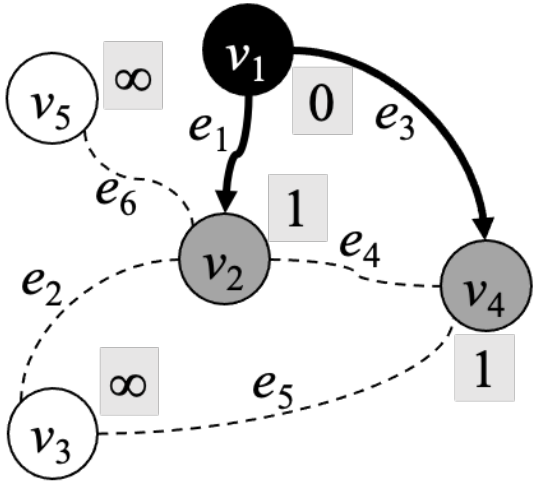
- 判断队列不为空，v_1出队列
- 判断v_1的邻点v_2.visited为false（v_4同理）
  - v_2.visited ← true
  - v_2.d ← v_1.d + 1
  - v_2入队列

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



队列Q | v_2 | v_4 | | |

白色顶点：未入队列
灰色顶点：队列中
黑色顶点：已出队列
数字：d属性值

虚线：未发生入队列操作的相邻顶点
粗实线箭头：下一步入队列的邻点
细实线箭头：已入队列的邻点

BFS算法

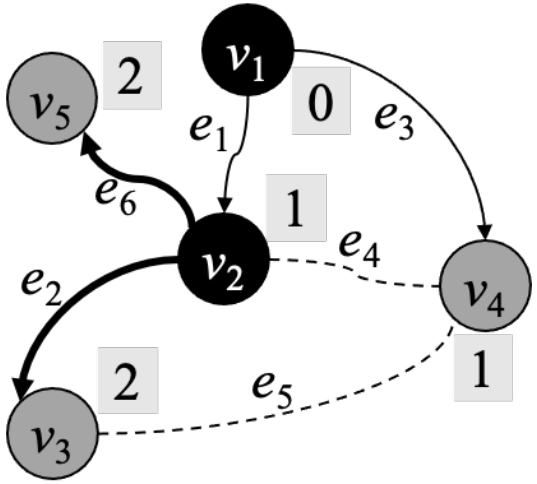
- 判断队列不为空，v_2出队列
- 判断v_2的邻点v_1.visited为true（v_4同理）
- 判断v_2的邻点v_3.visited为false（v_5同理）
  - v_3.visited ← true
  - v_3.d ← v_2.d + 1
  - v_3入队列

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



队列Q | v_4 | v_3 | v_5 |

白色顶点：未入队列
灰色顶点：队列中
黑色顶点：已出队列
数字：d属性值

虚线：未发生入队列操作的相邻顶点
粗实线箭头：下一步入队列的邻点
细实线箭头：已入队列的邻点

BFS算法

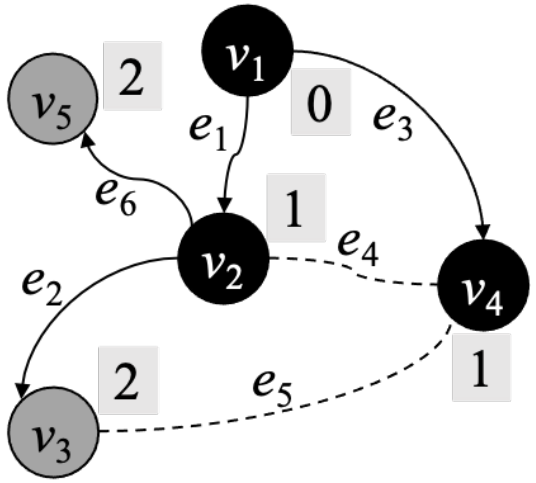
- 判断队列不为空，v_4出队列
- 判断v_4的邻点v_1.visited为true（v_2, v_3同理）

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



队列Q | v_3 | v_5 | | |

白色顶点：未入队列
灰色顶点：队列中
黑色顶点：已出队列
数字：d属性值

虚线：未发生入队列操作的相邻顶点
粗实线箭头：下一步入队列的邻点
细实线箭头：已入队列的邻点

BFS算法

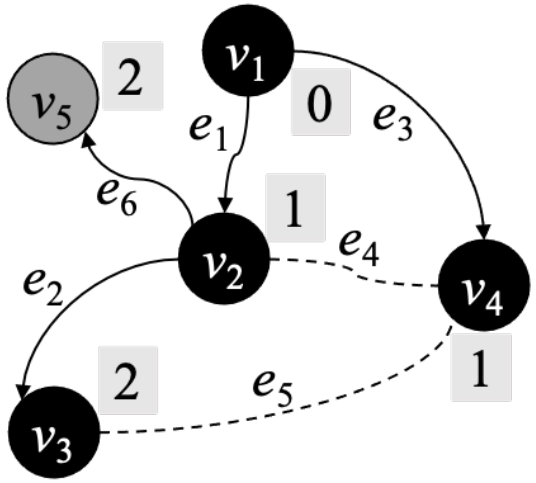
- 判断队列不为空，v_3出队列
- 判断v_3的邻点v_2.visited为true（v_4同理）

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



队列Q | v_5 | | | |

白色顶点：未入队列
灰色顶点：队列中
黑色顶点：已出队列
数字：d属性值

虚线：未发生入队列操作的相邻顶点
粗实线箭头：下一步入队列的邻点
细实线箭头：已入队列的邻点

BFS算法

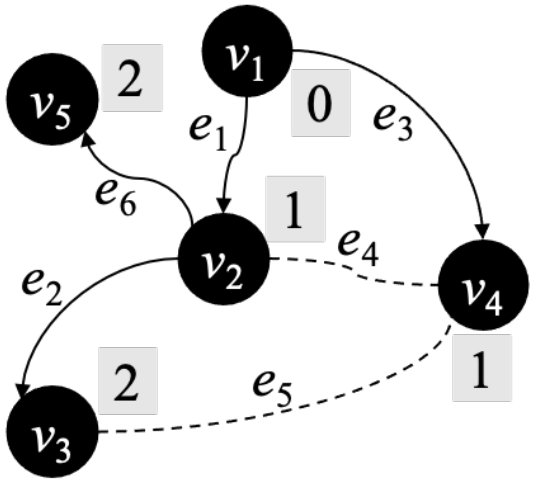
- 判断队列不为空，v_5出队列
- 判断v_5的邻点v_2.visited为true

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



队列Q

白色顶点：未入队列
灰色顶点：队列中
黑色顶点：已出队列
数字：d属性值

虚线：未发生入队列操作的相邻顶点
粗实线箭头：下一步入队列的邻点
细实线箭头：已入队列的邻点

BFS算法

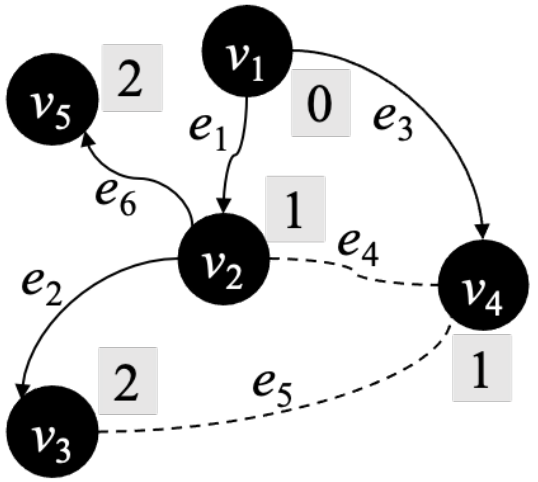
- 判断队列为空

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



队列Q

白色顶点：未入队列
灰色顶点：队列中
黑色顶点：已出队列
数字：d属性值

虚线：未发生入队列操作的相邻顶点
粗实线箭头：下一步入队列的邻点
细实线箭头：已入队列的邻点

BFS算法

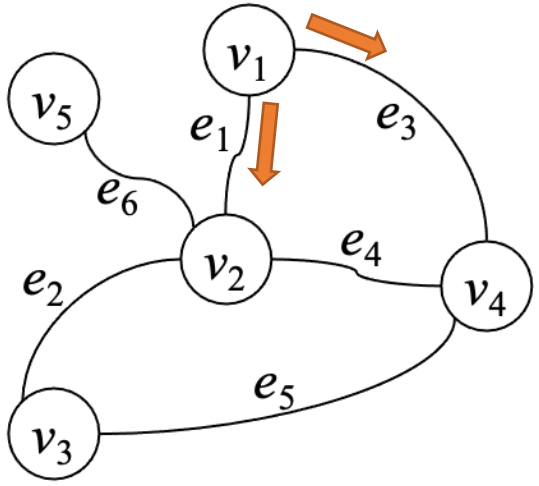
- 不同次运行BFS算法，各顶点的访问顺序可能不同
  - 取决于邻点的访问顺序

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



思考题2.43

- 在BFS算法中，可否省略visited属性？

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```

思考题2.43

- 在BFS算法中，可否省略visited属性？
  - 可以

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

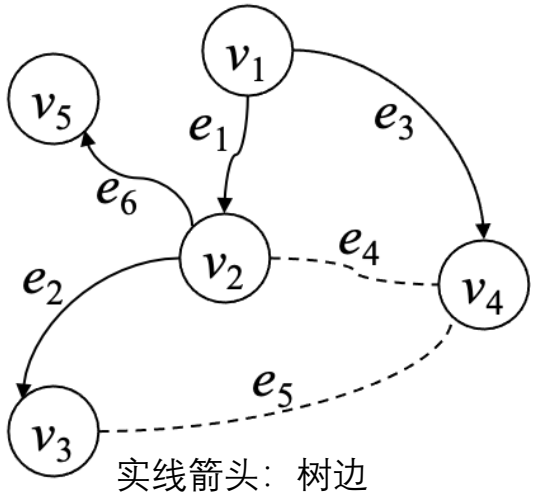
初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then   if w.d = ∞ then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```

BFS算法

- **BFS树**：所有树边的边导出子图
  - **树边**：从父顶点访问子顶点经过的边
  - **后向边**：其它边

	d	parent
v_1	0	null
v_2	1	v_1
v_3	2	v_2
v_4	1	v_1
v_5	2	v_2

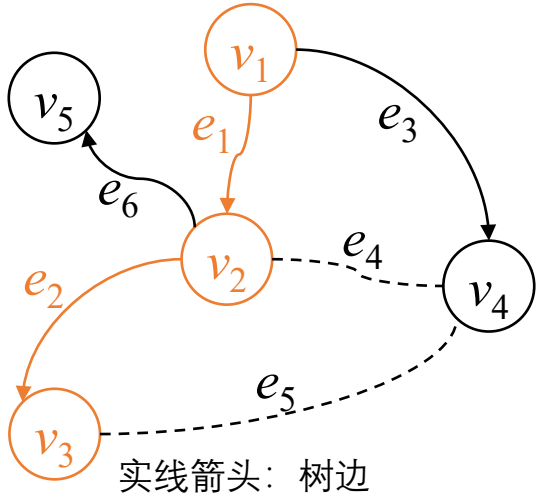


实线箭头：树边
虚线：后向边

思考题2.45

- 在BFS树中，以根顶点为起点的路有什么特征？

	d	parent
v_1	0	null
v_2	1	v_1
v_3	2	v_2
v_4	1	v_1
v_5	2	v_2



思考题2.45

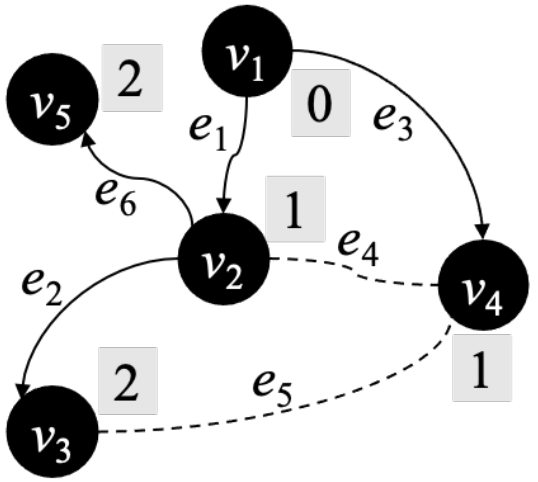
- 在BFS树中，以根顶点为起点的路有什么特征？
  - 是最短路

算法 2.3: BFS

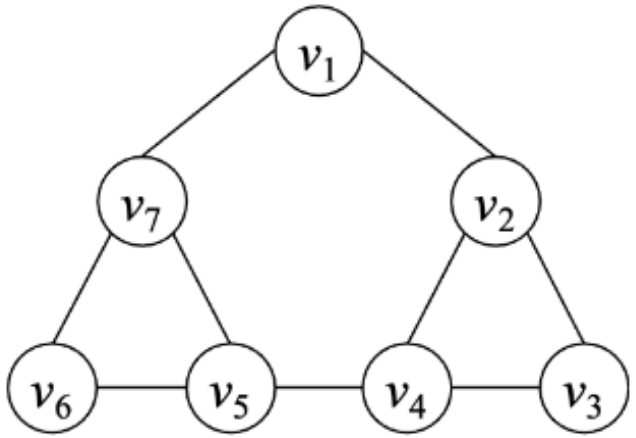
输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

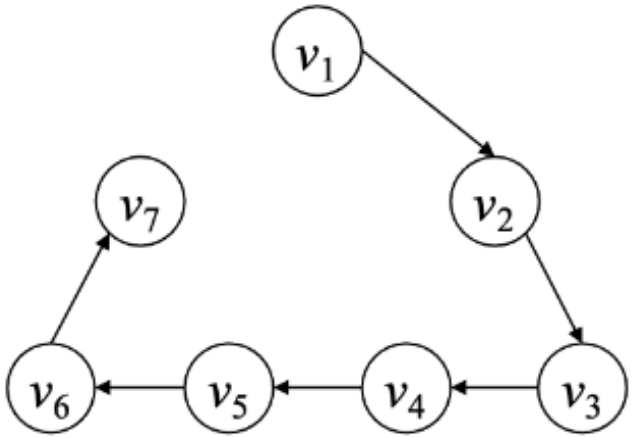
```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



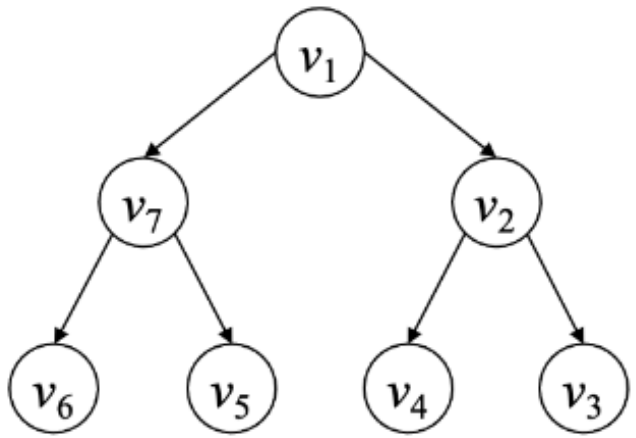
DFS树和BFS树的对比



图G



DFS树



BFS树

定理2.8

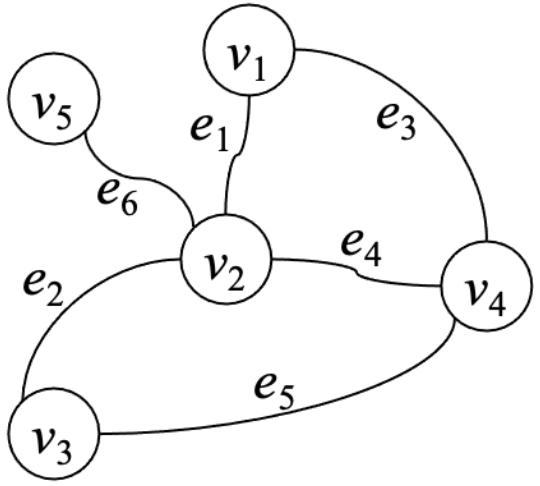
- 从顶点u出发运行BFS算法，恰能访问与u连通的所有顶点。
  - 与DFS算法正确性的证明类似。

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



引理2.2

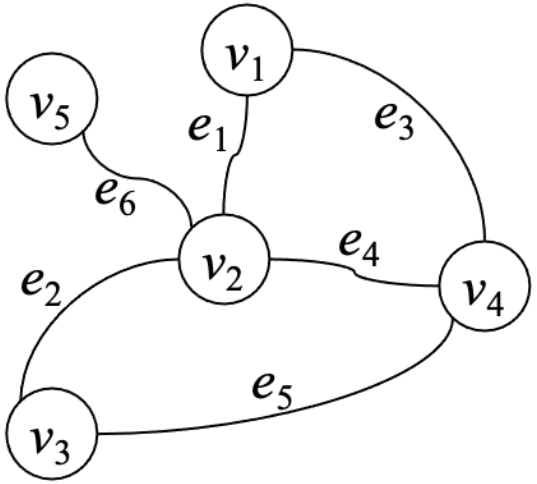
- BFS算法运行结束时，每个顶点的d属性值是其和出发点间的距离的上界。

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



引理2.2

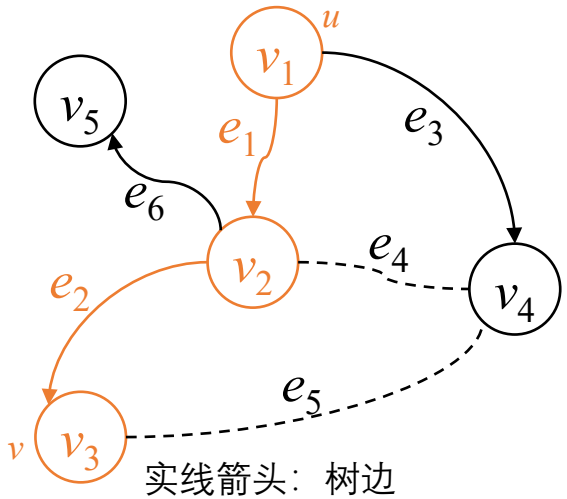
- BFS算法运行结束时，每个顶点的d属性值是其和出发点间的距离的上界。
  - 对于出发点u和每个顶点v，根据BFS算法，BFS树中u-v路经过的顶点的d属性值递增，
  - v的d属性值即u-v路的长度，是u和v间的距离的上界。

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



引理2.3

- BFS算法运行过程中，队列中所有顶点的d属性值组成非减序列，且队首和队尾顶点的d属性值相差至多1。

算法 2.3: BFS

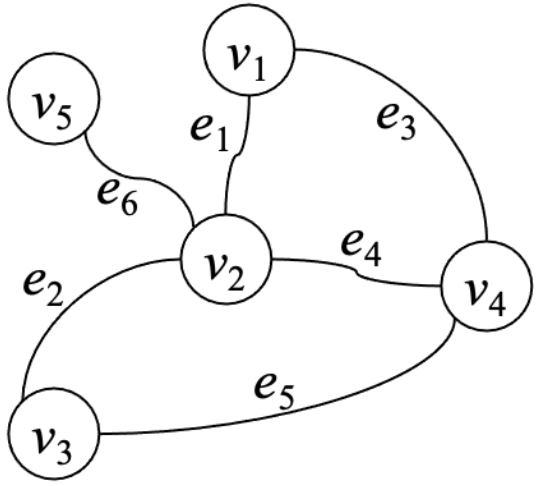
输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```

v.d	…	…	$\leq v$.d + 1

队首　　队尾



引理2.3

- BFS算法运行过程中，队列中所有顶点的d属性值组成非减序列，且队首和队尾顶点的d属性值相差至多1。
  - 只需证明每轮while循环条件判定前成立。
    - 第1轮while循环条件判定前成立。
    - 若本轮while循环条件判定前成立，则本轮while循环增加到队尾的所有顶点w都满足$w.\mathrm{d} = v.\mathrm{d} + 1$，因此，下轮while循环条件判定前仍成立。

算法 2.3: BFS

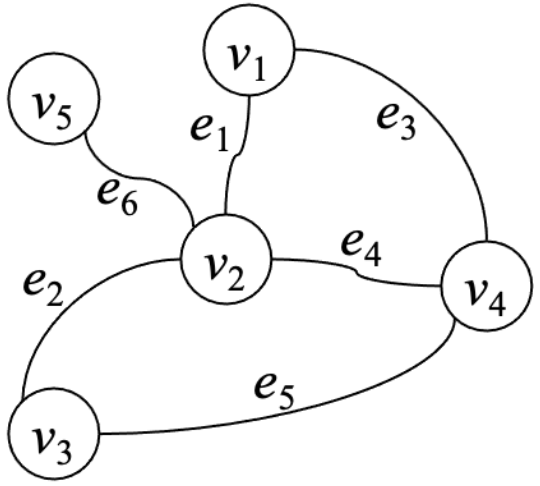
输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```

$v.\mathrm{d}$	…	…	$\leq v.\mathrm{d} + 1$

队首　　队尾



引理2.4

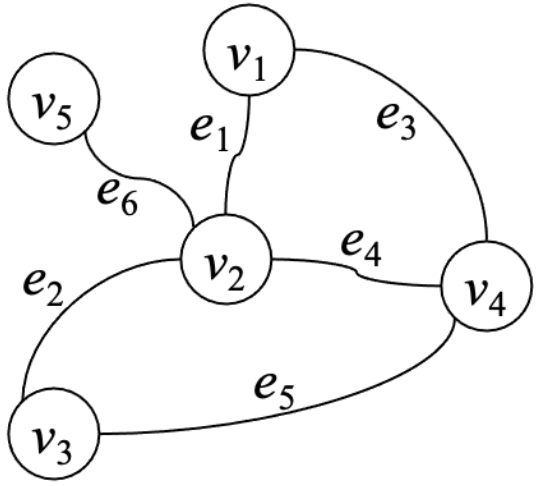
- BFS算法运行结束时，按访问顺序，所有顶点的d属性值组成非减序列。

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



引理2.4

- BFS算法运行结束时，按访问顺序，所有顶点的d属性值组成非减序列。
  - 顶点的访问顺序即入队列的顺序，由引理2.3，得证。

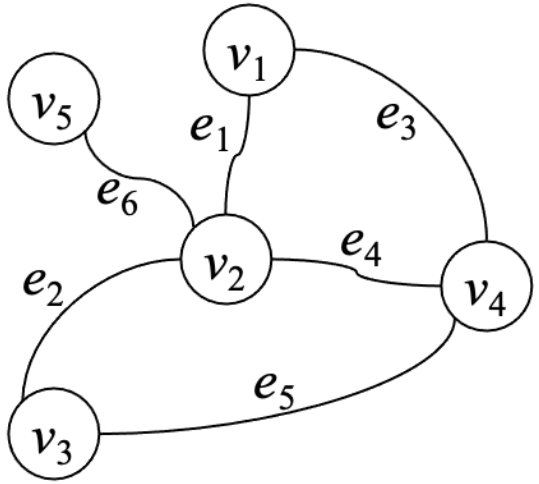
> 引理2.3 BFS算法运行过程中，队列中所有顶点的d属性值组成非减序列，……

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



定理2.9

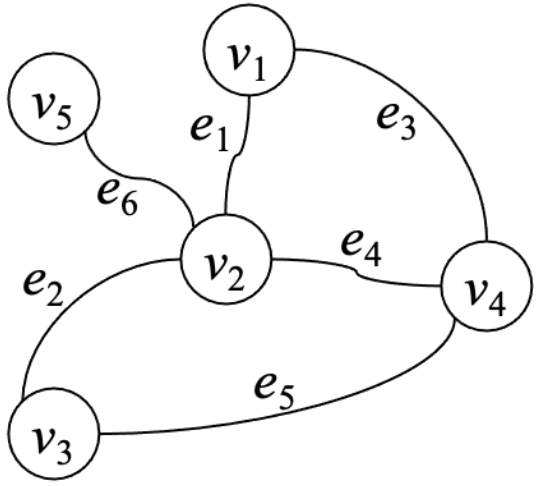
- BFS算法运行结束时，每个顶点的d属性值为其和出发点间的距离。

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



定理2.9

- BFS算法运行结束时，每个顶点的d属性值为其和出发点间的距离。
  - 采用反证法，假设存在顶点的d属性值不为其和出发点u间的距离，
    则由引理2.2，这些顶点的d属性值大于其和u间的距离，将这些顶点中和u间距离最小的顶点记作v。

> 引理2.2 BFS算法运行结束时，每个顶点的d属性值是其和出发点间的距离的上界。

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```

定理2.9

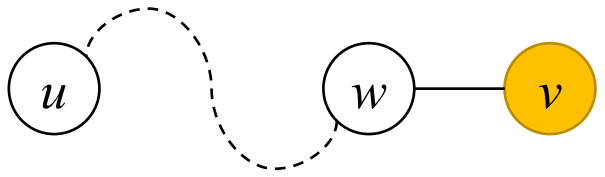
- BFS算法运行结束时，每个顶点的d属性值为其和出发点间的距离。
  - 采用反证法，假设存在顶点的d属性值不为其和出发点u间的距离，
    则由引理2.2，这些顶点的d属性值大于其和u间的距离，将这些顶点中和u间距离最小的顶点记作v。
  - 根据BFS算法，$u.\mathrm{d} = 0 = \mathrm{dist}(u, u)$，因此，$v \neq u$，
    对于任意一条最短u-v路经过的v的前一个邻点w（可能是u），$w.\mathrm{d} = \mathrm{dist}(u, w) < \mathrm{dist}(u, v) < v.\mathrm{d}$。

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



定理2.9

- BFS算法运行结束时，每个顶点的d属性值为其和出发点间的距离。
  - 当顶点w出队列时，分情况讨论顶点v：
    - 若v未入队列，则$v.\mathrm{d} = w.\mathrm{d} + 1$，与右式矛盾。
    - 若v已入队列，则由引理2.3，$v.\mathrm{d} \le w.\mathrm{d} + 1$，与右式矛盾。
    - 若v已出队列，则由引理2.4，$v.\mathrm{d} \le w.\mathrm{d}$，与右式矛盾。

$w.\mathrm{d} = \mathrm{dist}(u, w) < \mathrm{dist}(u, v) < v.\mathrm{d}$

引理2.3 BFS算法运行过程中，……，且队首和队尾顶点的d属性值相差至多1。

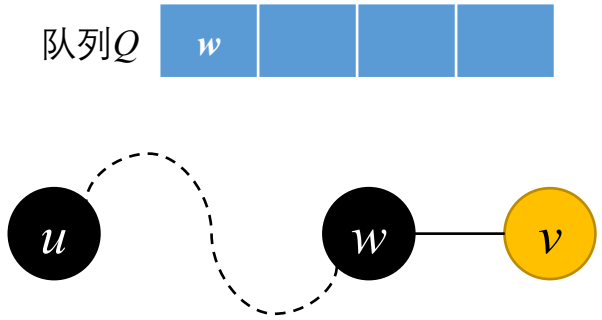
引理2.4 BFS算法运行结束时，按访问顺序，所有顶点的d属性值组成非减序列。

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



BFS算法

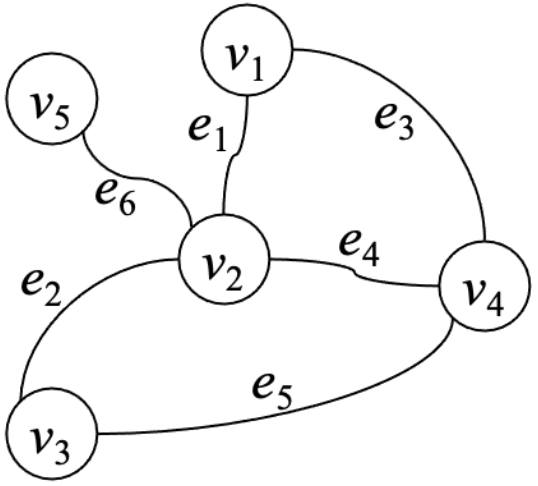
- 时间复杂度：$O(n+m)$

算法 2.3: BFS

输入: 图 $G=\langle V,E\rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；
队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



BFS算法

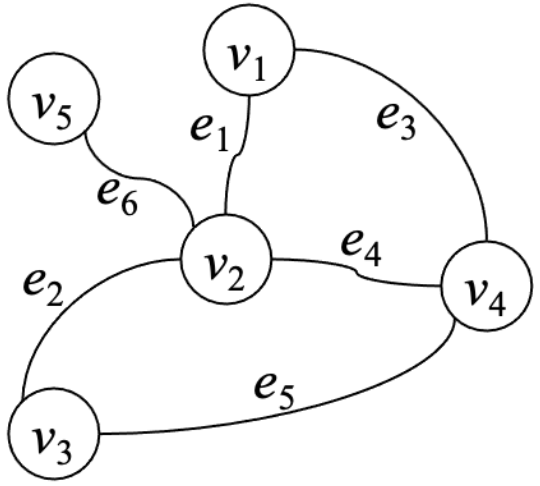
- 利用BFS算法判定顶点u和v是否连通：
- 利用BFS算法判定图G是否连通：
- 利用BFS算法计算顶点u和任意顶点间的距离：

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```



BFS算法

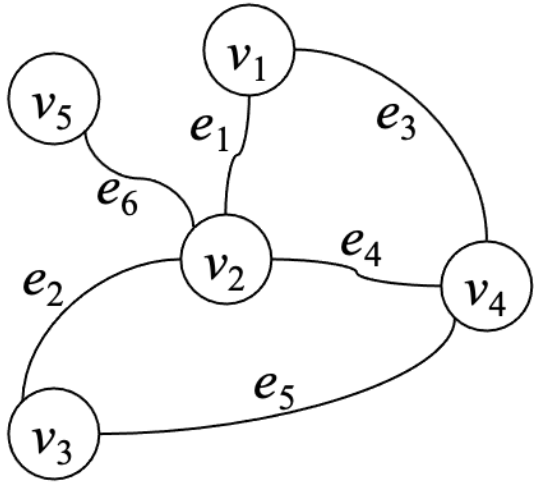
- 利用BFS算法判定顶点u和v是否连通：
  - 从u出发运行一次算法，若访问过v，则u和v连通，否则不连通
- 利用BFS算法判定图G是否连通：
  - 从任意一个顶点出发运行一次算法，若访问过图中所有顶点，则G连通，否则不连通
- 利用BFS算法计算顶点u和任意顶点间的距离：
  - 从u出发运行一次算法，顶点的d属性值即该顶点和u间的距离

算法 2.3: BFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

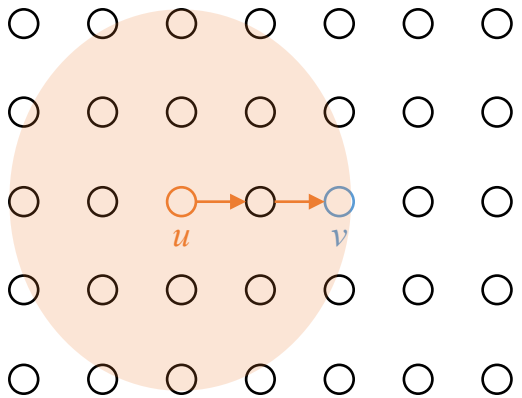
初值: 顶点集 V 中所有顶点的 visited 初值为 false，d 初值为 ∞；队列 Q 初值为空

```
1  u.visited ← true;
2  u.d ← 0;
3  入队列 (Q, u);
4  while Q 非空 do
5      v ← 出队列 (Q);
6      foreach (v, w) ∈ E do
7          if w.visited = false then
8              w.visited ← true;
9              w.d ← v.d + 1;
10             入队列 (Q, w);
```

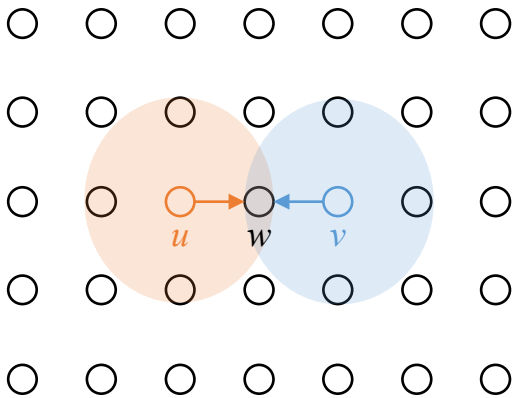


计算两个顶点间的距离

- **单向搜索**（如BFS算法）：从其中一个顶点出发，搜索到另一个顶点的一条最短路
- **双向搜索**：从两个顶点同时出发运行BFS算法，当两者首次“相遇”，即首次访问同一个顶点w时，将从u到w的最短路和从v到w的最短路拼接形成一条最短u-v路
  - 两棵BFS树更“浅”，找到最短路时访问的顶点总数更少，时间复杂度更低



单向搜索



双向搜索

请认真完成课后练习