



# グラフの探索

深さ優先探索と幅優先探索

# グラフ探索

## ○ 単純な探索

- ある頂点に到達できるか
- 到達できる頂点を列挙する
- 二つのアルゴリズム
  - 深さ優先
  - 幅優先

## ○ 単純でない探索:後述

- 最小な木
- 最短経路

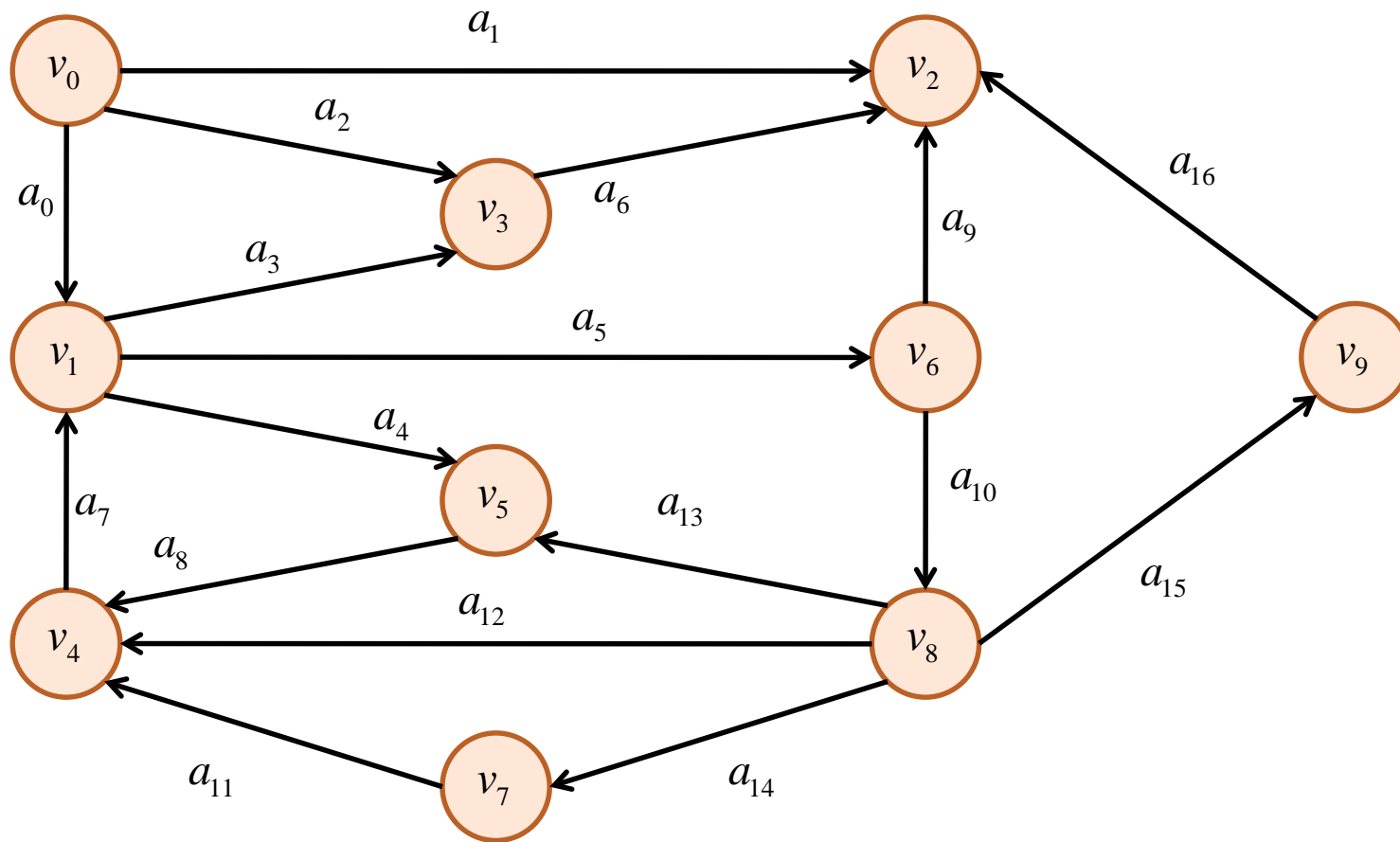


# 深さ優先探索DFS (DEPTH-FIRST SEARCH)

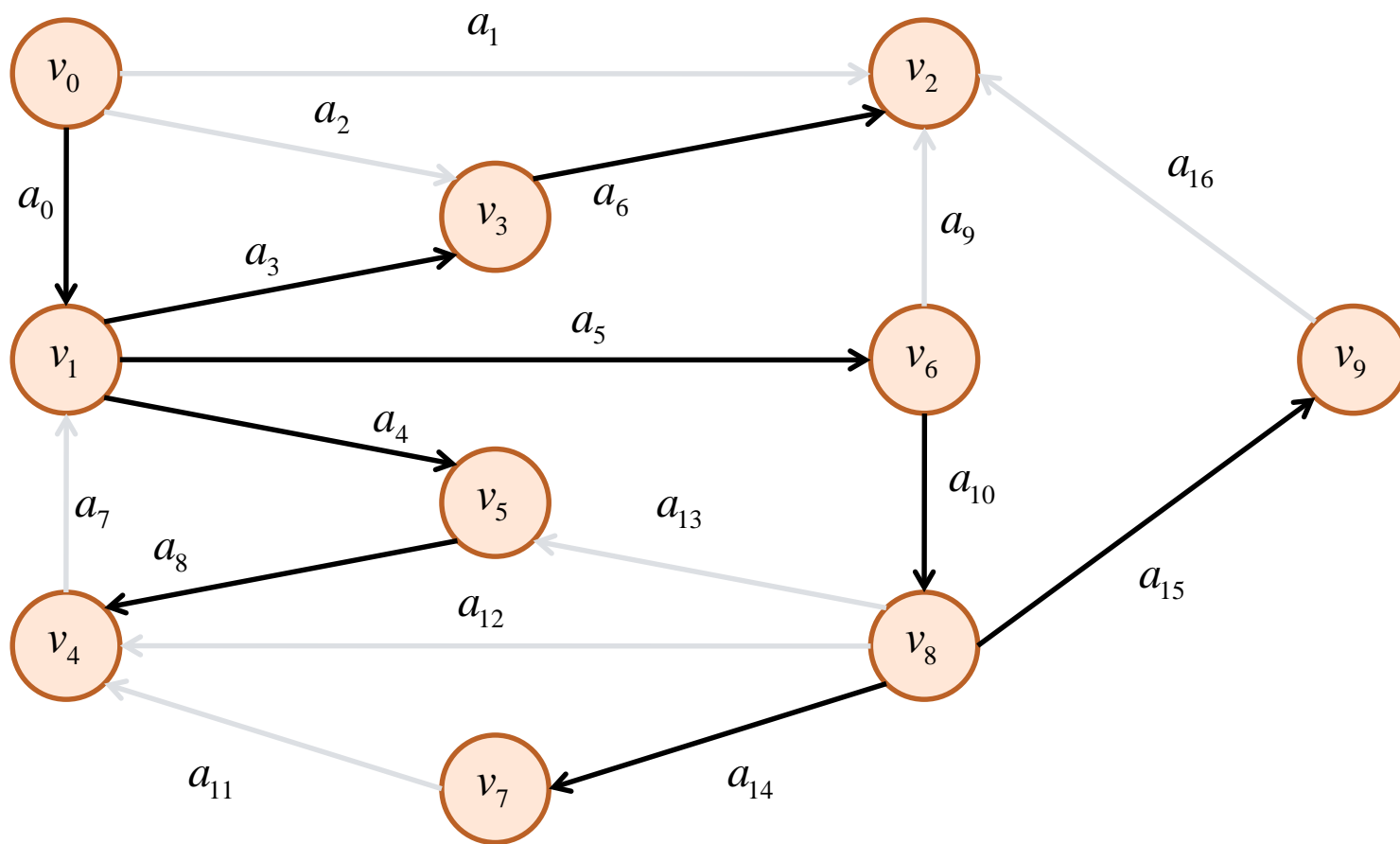
- 出発点を定める
- たどれる限り、弧をたどる
  - それ以上進めなくなるまで
  - 新たな点が無くなるまで
- 戻って、別の弧をたどる
  
- 結果としてできる木(spanning tree)は、深いものができる



# 深さ優先探索DFS (DEPTH-FIRST SEARCH)



# 結果



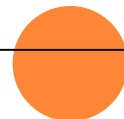
## 再帰的関数で表現

- $L$ : 既にチェックした点のリスト
- $v$ : 現在の頂点

再帰的な探索

グラフを深い方向に探索

```
search(v, L){
  // v から出る全ての弧
  forall(a ∈ δ+v){
    w = δ-a // 反対側の頂点
    if(w ∉ L){
      L = L ∪ {w}
      search(w, L)
    }
  }
  // これ以上進めない
  return
}
```

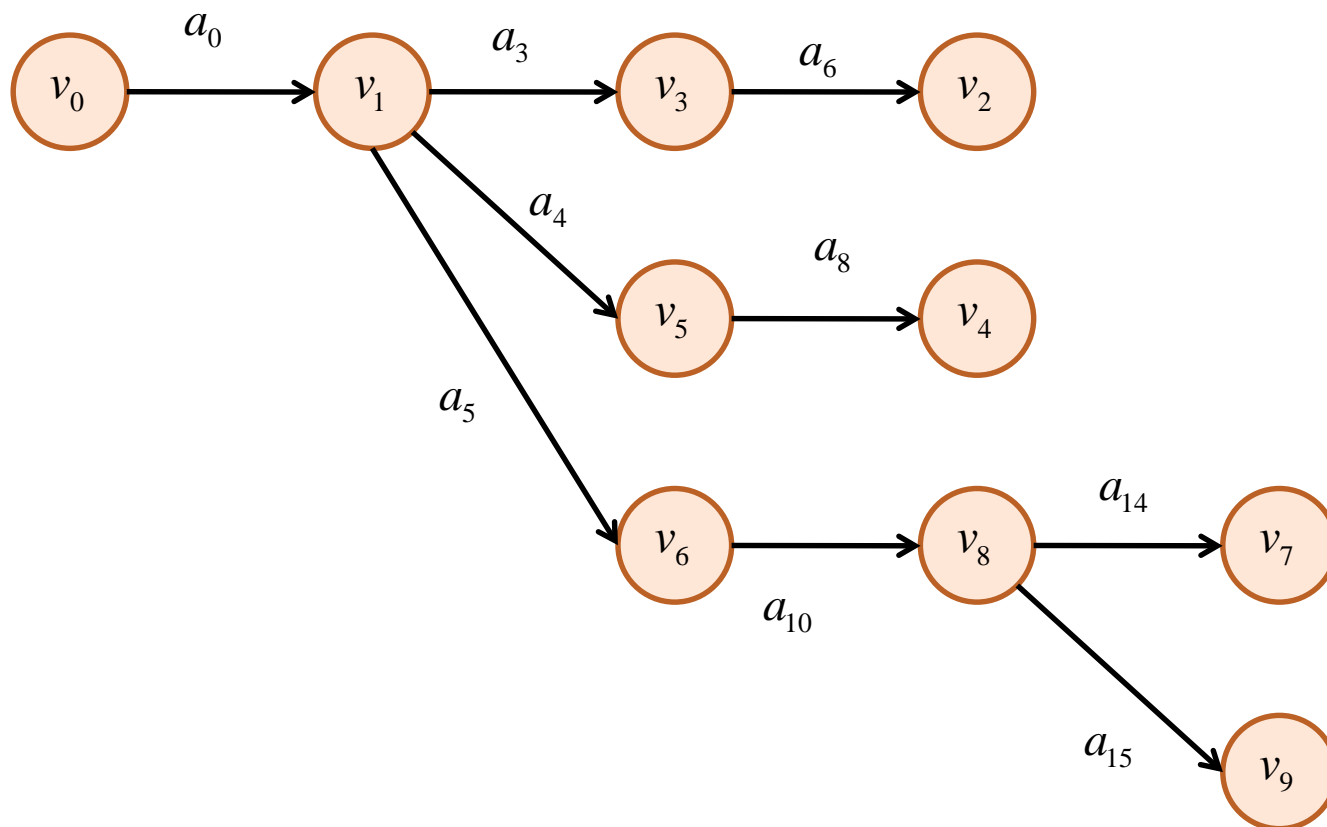


## 再帰呼び出しの様子

$$\begin{aligned} & (v_0, \{v_0\}) \rightarrow (v_1, \{v_0, v_1\}) \rightarrow (v_3, \{v_0, v_1, v_3\}) \rightarrow (v_2, \{v_0, v_1, v_2, v_3\}) \\ & \rightarrow (v_5, \{v_0, v_1, v_5\}) \rightarrow (v_4, \{v_0, v_1, v_4, v_5\}) \\ & \rightarrow (v_6, \{v_0, v_1, v_6\}) \rightarrow (v_8, \{v_0, v_1, v_6, v_8\}) \rightarrow (v_7, \{v_0, v_1, v_6, v_7, v_8\}) \\ & \rightarrow (v_9, \{v_0, v_1, v_6, v_8, v_9\}) \end{aligned}$$

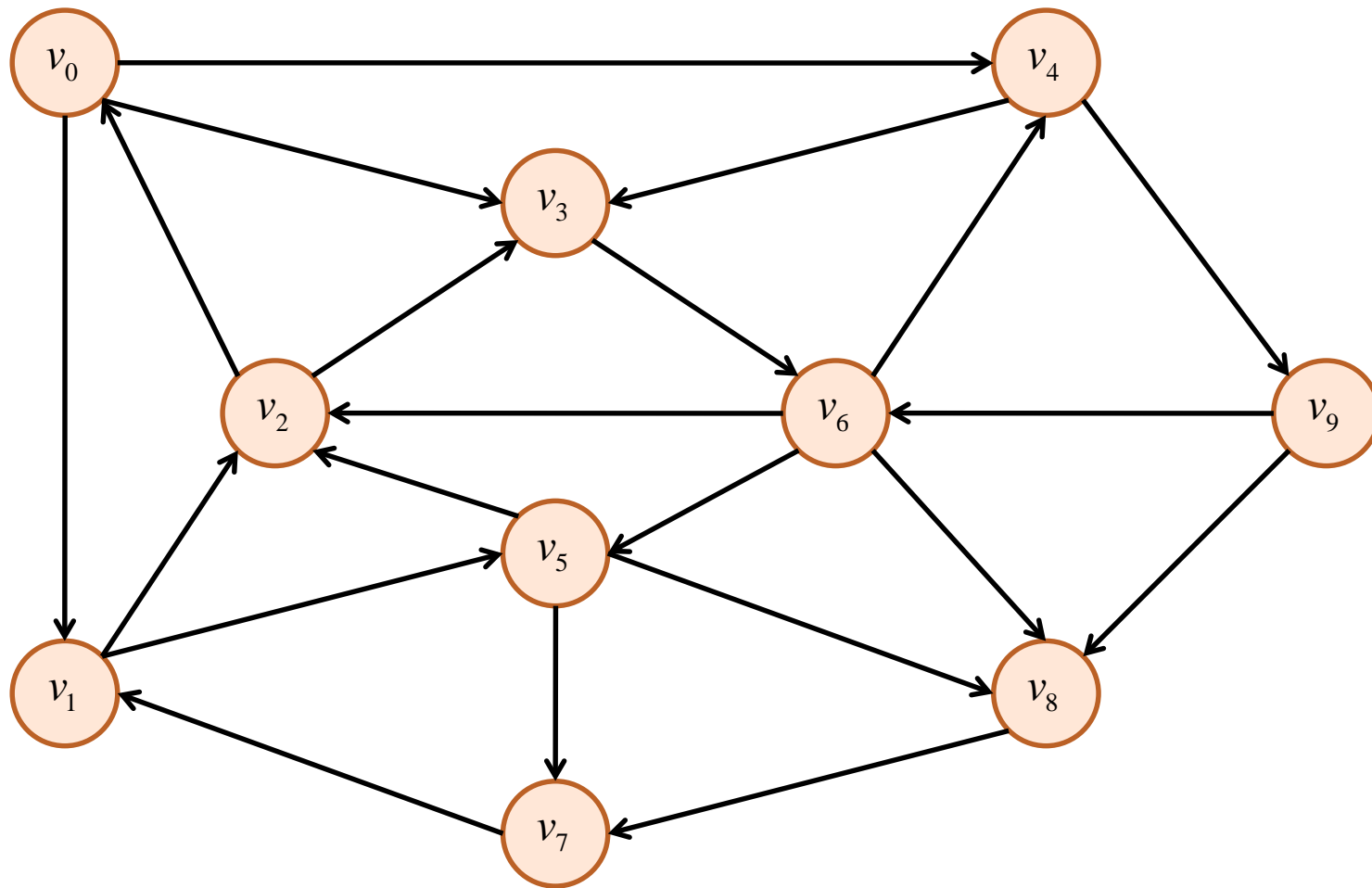


# 探索の様子(結果としてのSPANNING TREE)





## 例2

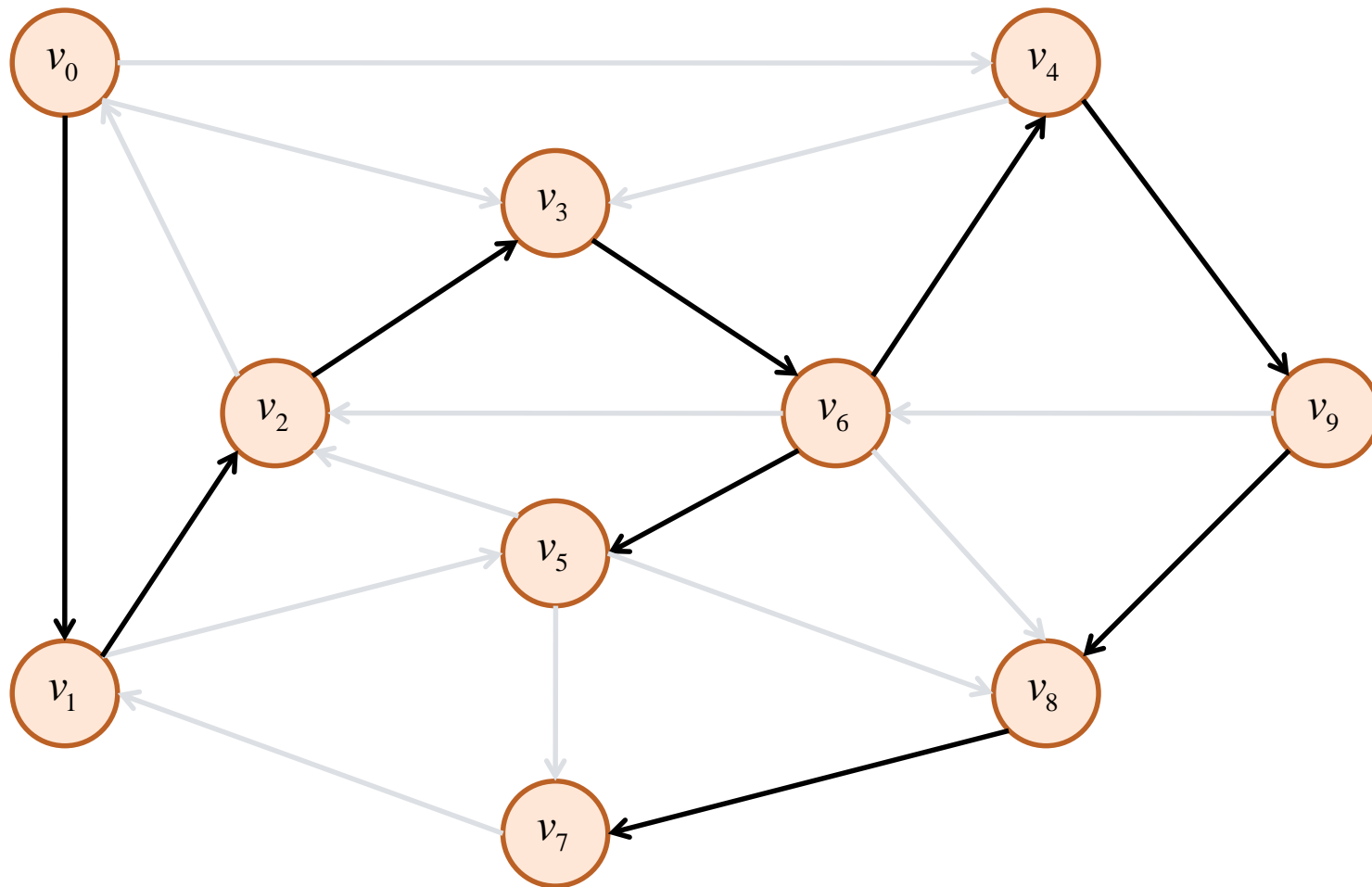


$$(v_0, \{v_0\}) \rightarrow (v_1, \{v_0, v_1\}) \rightarrow (v_2, \{v_0, v_1, v_2\}) \rightarrow (v_3, \{v_0, v_1, v_2, v_3\})$$

$$\rightarrow (v_6, \{v_0, v_1, v_2, v_3, v_6\}) \rightarrow (v_4, \{v_0, v_1, v_2, v_3, v_4, v_6\}) \rightarrow (v_9, \{v_0, v_1, v_2, v_3, v_4, v_6, v_9\})$$

**例2**  $\rightarrow (v_8, \{v_0, v_1, v_2, v_3, v_4, v_6, v_8, v_9\}) \rightarrow (v_7, \{v_0, v_1, v_2, v_3, v_4, v_6, v_7, v_8, v_9\})$

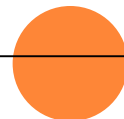
$$\rightarrow (v_5, \{v_0, v_1, v_2, v_3, v_5, v_6\})$$



## 再帰的関数で表現 (終点指定)

- $L$ : 既にチェックした点のリスト
- $v$ : 現在の頂点
- $d$ : 終点
- $f$ : 終点に到達したら真

```
search(v, L, d){
  //vから出る全ての弧
  forall(a ∈ δ+v){
    w = δ-a //反対側の頂点
    if(w ∉ L ∧ !f){
      L = L ∪ {w}
      if(w == d){
        f = true
        return
      }
      search(w, L, d)
    }
  }
  //これ以上進めない
  return
}
```

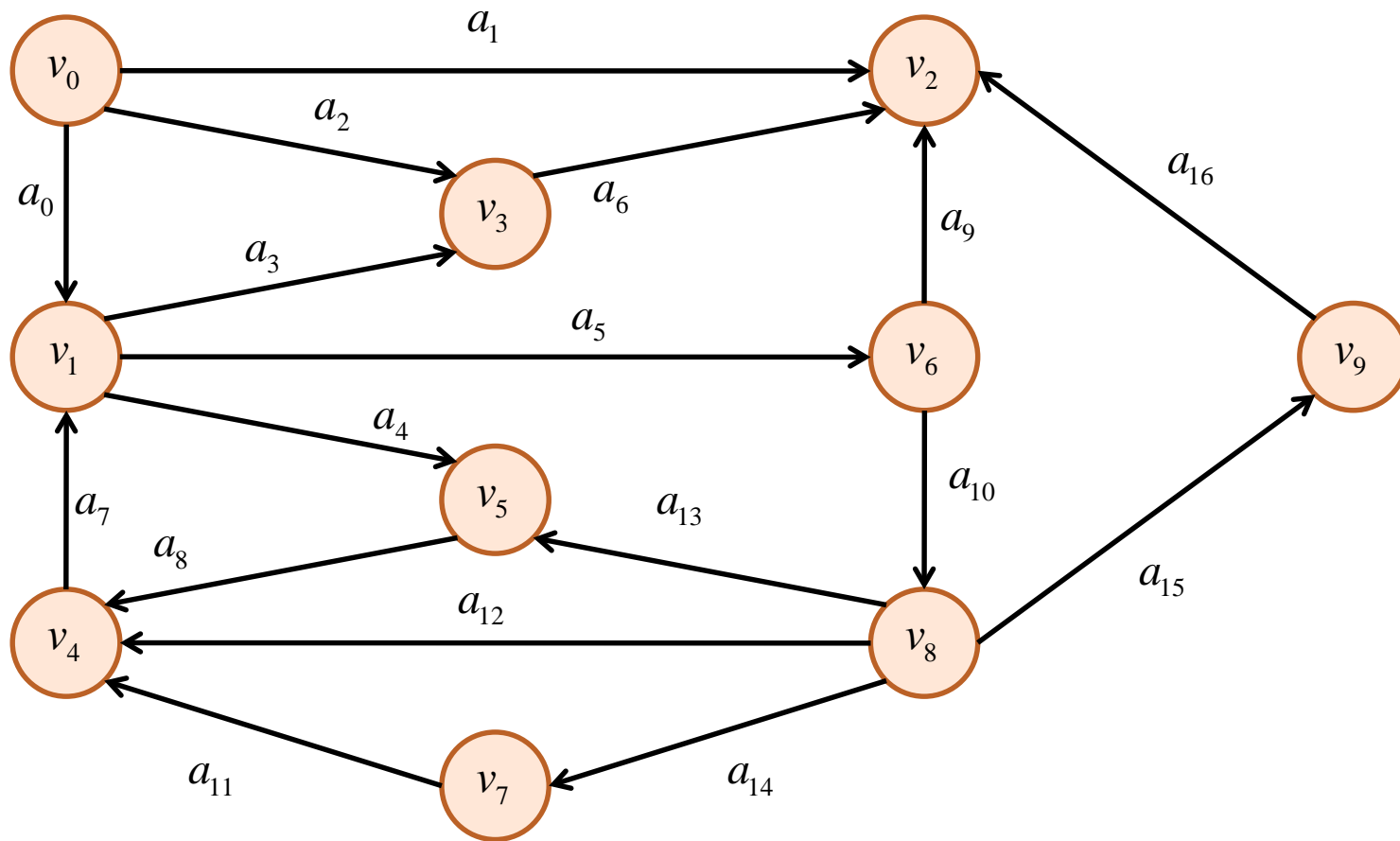


# 幅優先探索(BREADTH-FIRST SEARCH)

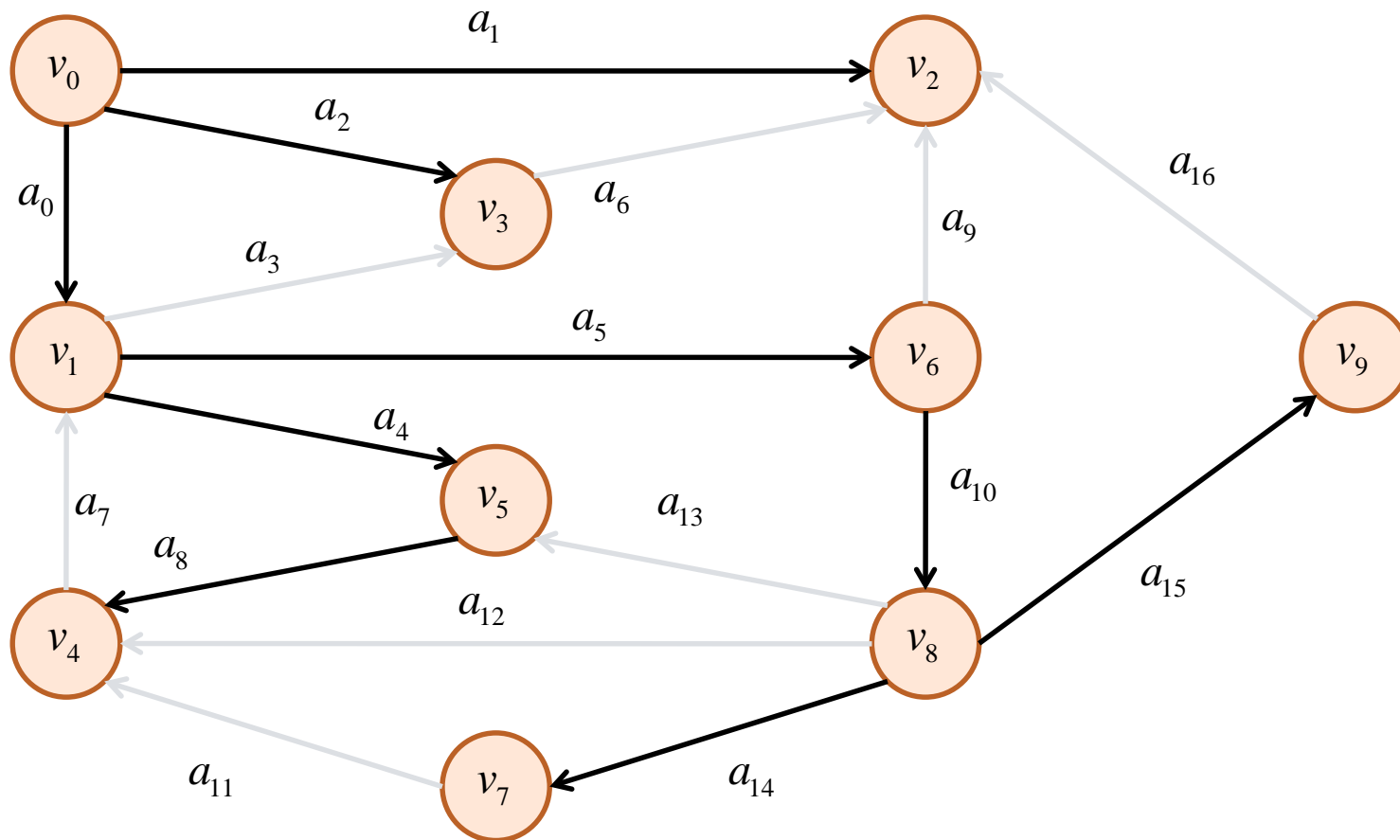
- 出発点を定める
- 出発点に直接繋がっている点に印を付ける
- 印を付けた点に直接繋がっている点に印を付ける
  
- 結果としてできる木(spanning tree)は、幅の広いものができる



# 幅優先探索BFS (BREADTH-FIRST SEARCH)



# 結果



# 幅優先探索

- $L$ : すでにチェックした点のリスト: 初期  $L = \emptyset$
- $Q$ : 調査すべき点のキュー: 初期  $Q = \{r\}$

```
L =  $\emptyset$ 
Q = {r}
while(Q  $\neq \emptyset$ ){
    v = Qの先頭の取り出し
    forall(a  $\in \delta^+v$ ){
        w =  $\delta^-a$ 
        if( w  $\notin L$  && w  $\notin Q$  ){
            Qへwを追加
        }
    }
    L  $\leftarrow L \cup \{v\}$ 
}
```



## 註: キュー(Queue、待ち行列)

- 先入れ先出し (FIFO、First In First Out) のリスト構造
- `java.util.concurrent.ConcurrentLinkedQueue` クラス
- `add(E e)` 要素 `e` を追加
- `poll()` 先頭の要素を取り出す

要素取出



要素追加

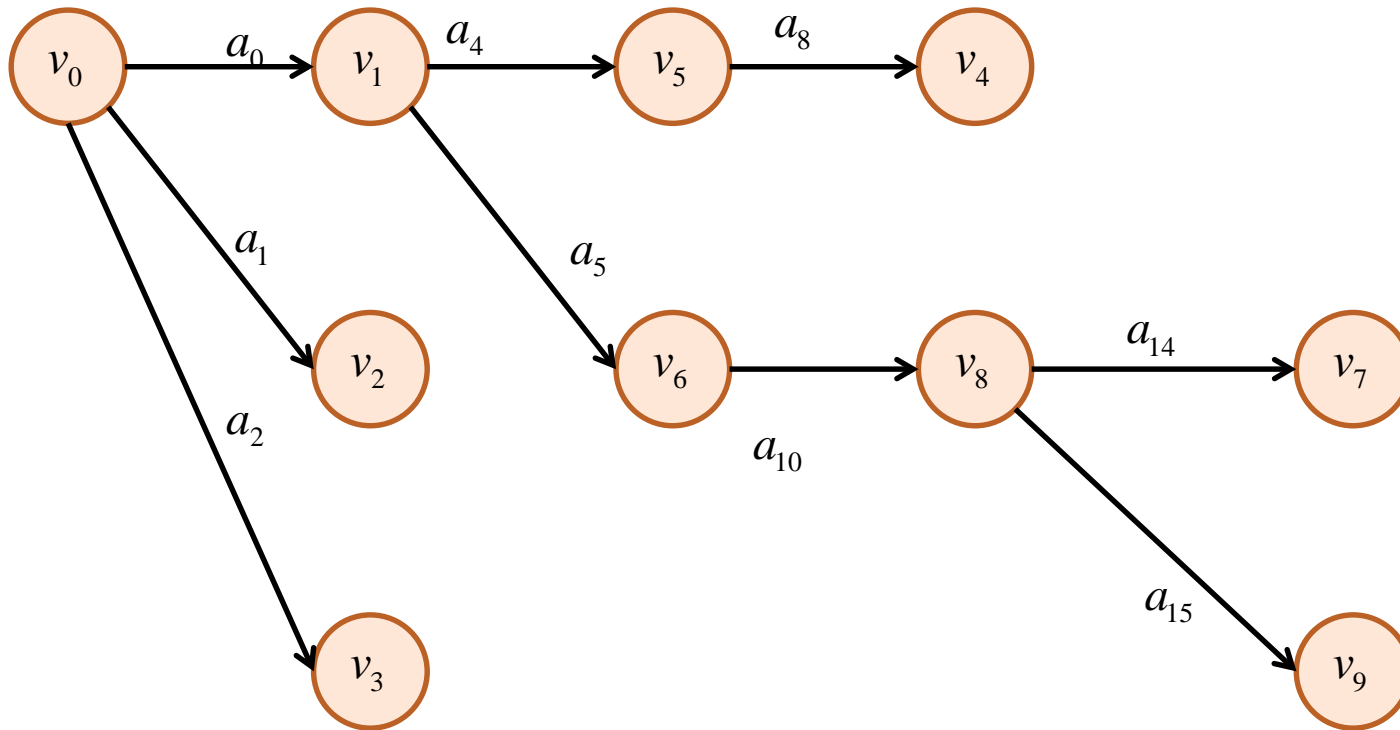




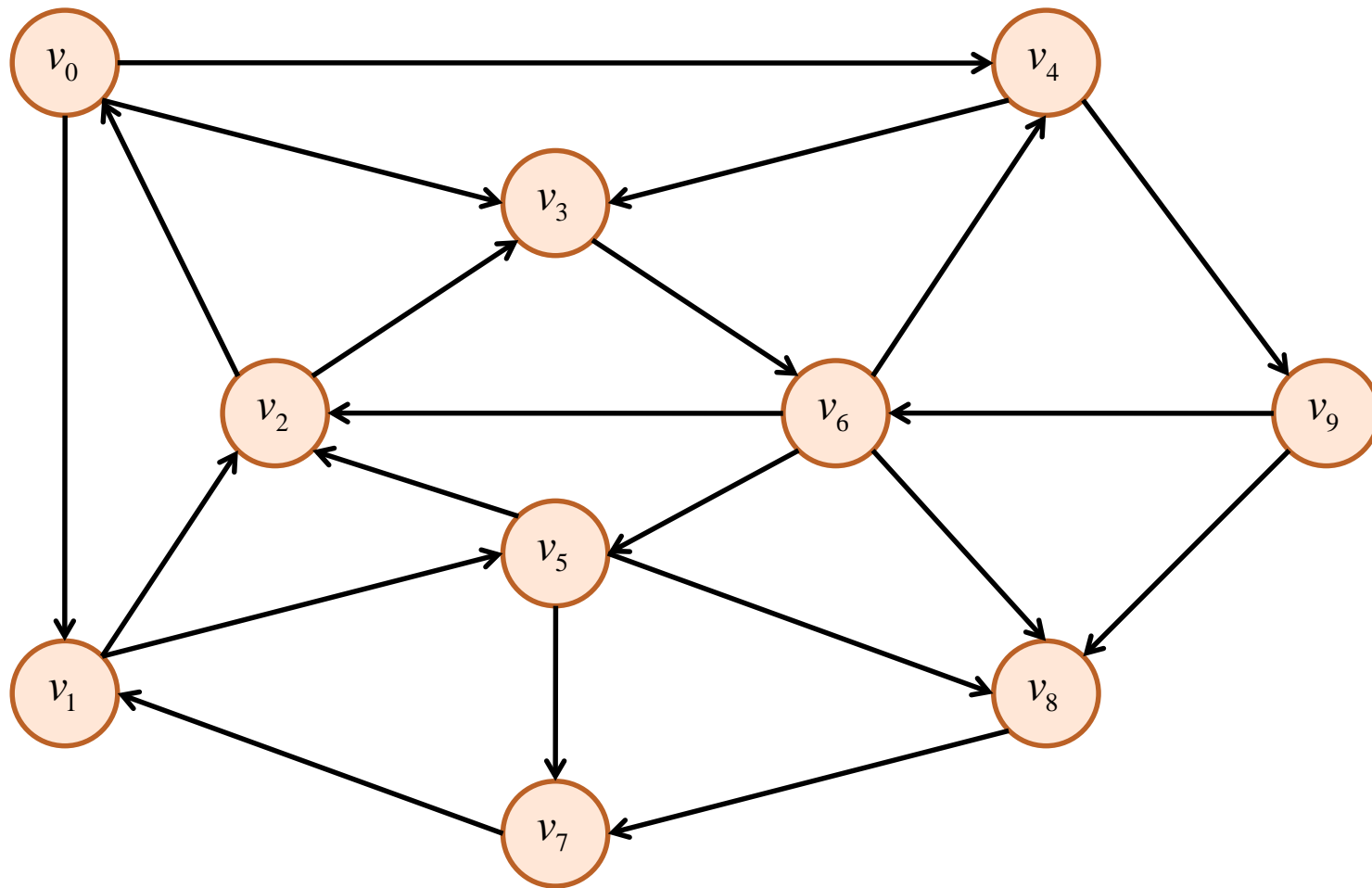
現在の頂点	$L$	$Q$
	$\emptyset$	$\{v_0\}$
$v_0$	$\{v_0\}$	$\{v_1, v_2, v_3\}$
$v_1$	$\{v_0, v_1\}$	$\{v_2, v_3, v_5, v_6\}$
$v_2$	$\{v_0, v_1, v_2\}$	$\{v_3, v_5, v_6\}$
$v_3$	$\{v_0, v_1, v_2, v_3\}$	$\{v_5, v_6\}$
$v_5$	$\{v_0, v_1, v_2, v_3, v_5\}$	$\{v_6, v_4\}$
$v_6$	$\{v_0, v_1, v_2, v_3, v_5, v_6\}$	$\{v_4, v_8\}$
$v_4$	$\{v_0, v_1, v_2, v_3, v_5, v_6, v_4\}$	$\{v_8\}$
$v_8$	$\{v_0, v_1, v_2, v_3, v_5, v_6, v_4, v_8\}$	$\{v_7, v_9\}$
$v_7$	$\{v_0, v_1, v_2, v_3, v_5, v_6, v_4, v_8, v_7\}$	$\{v_9\}$
$v_9$	$\{v_0, v_1, v_2, v_3, v_5, v_6, v_4, v_8, v_7, v_9\}$	$\emptyset$



# 探索の様子(結果としてのSPANNING TREE)



## 例2



## 例2

