



# グラフの探索

オブジェクト指向プログラミング特論

2018年度

只木進一：工学系研究科

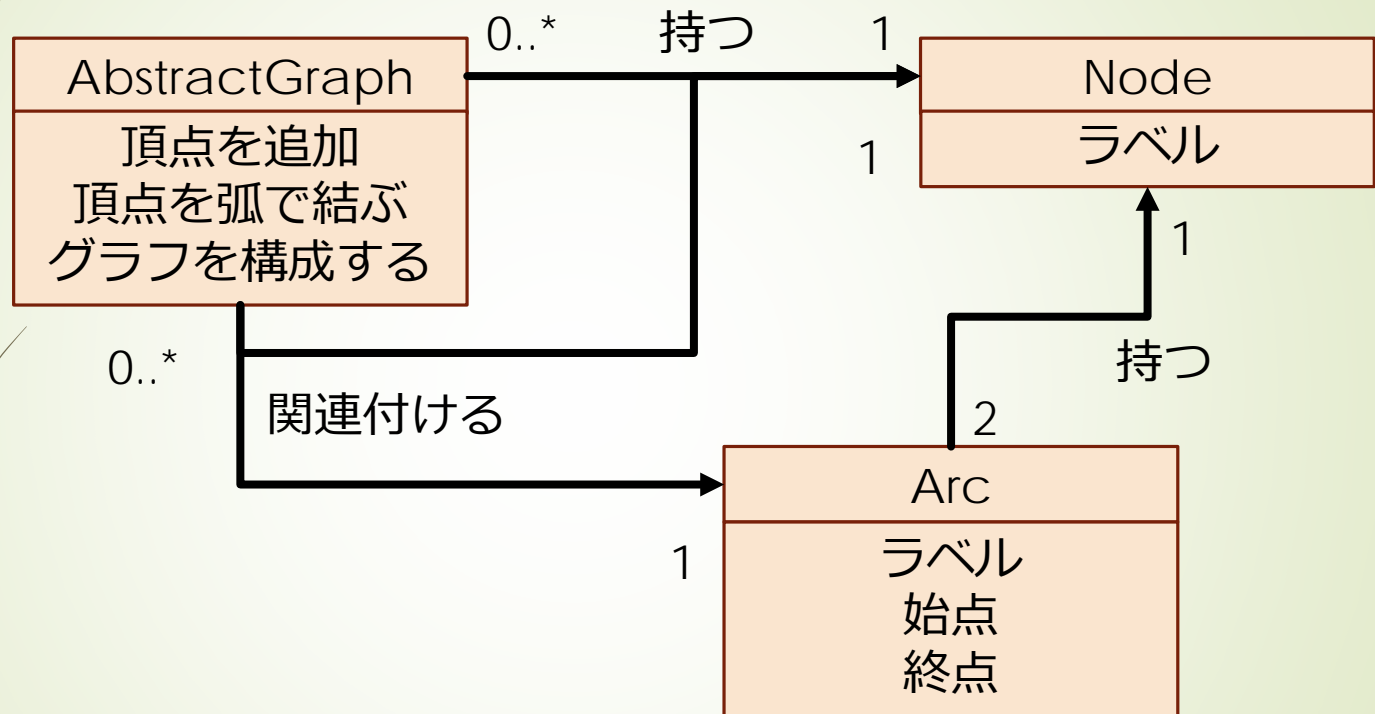
# グラフ(Graph)

- 頂点(Node)を弧(Arc)で連結したものをグラフという
  - 有向グラフと無向グラフ
- 要素の関係を表す
  - 人と人の関係、組織の関係、交通網、作業工程などなど

# クラス設計

## graphパッケージ

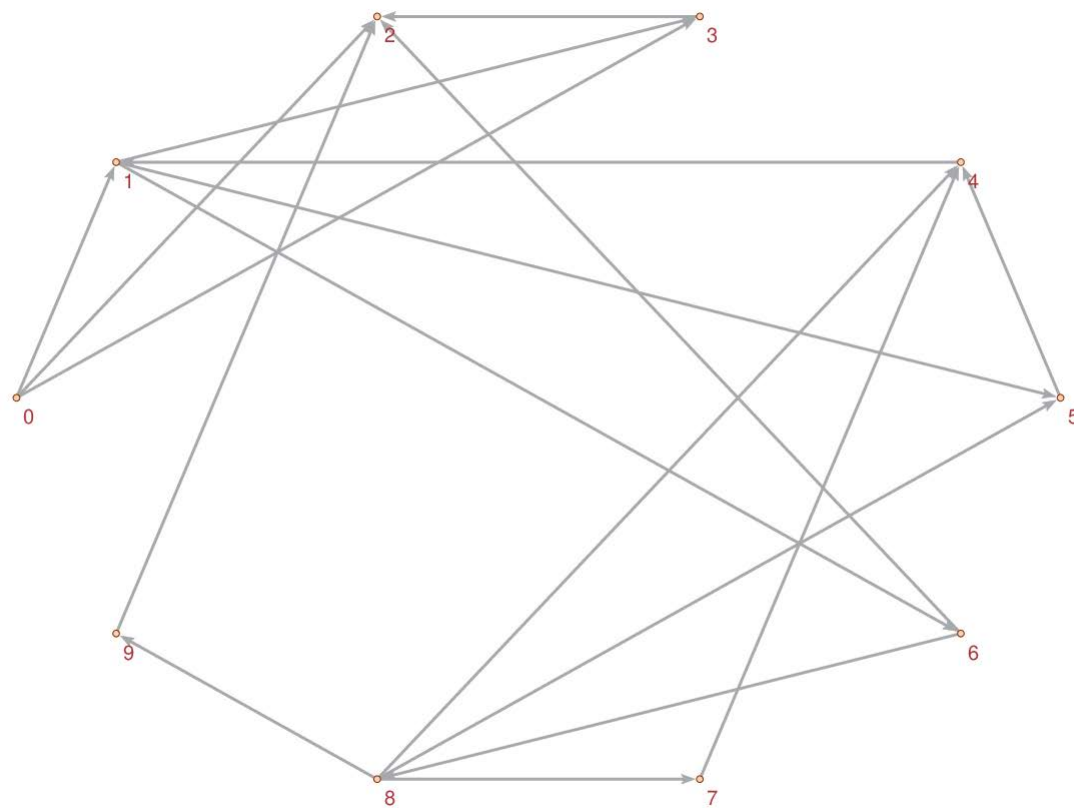
- AbstractGraphクラス
  - 頂点Nodeのリスト
  - 頂点から弧への写像
    - 頂点を始点とした弧
- Nodeクラス
- Arcクラス
  - 始点と終点を持つ



```
public abstract class AbstractGraph {  
  
    private final List<Node> nodes;  
    private final Map<Node, List<Arc>> node2arc;  
    public final String title;  
  
    public Graph(String title) {  
        this.title = title;  nodes = Utils.createList();  
        node2arc = Utils.createMap();  
    }  
  
    public void addNode(Node node) {  
        nodes.add(node);  
    }  
  
    public void addArc(Node from, Node to, String label) {  
        Arc arc = new Arc(from, to, label);  
        if (!node2arc.containsKey(from)) {  
            List<Arc> arcList = Utils.createList();  
            node2arc.put(from, arcList);  
        }  
        node2arc.get(from).add(arc);  
    }  
}
```

# グラフの定義

```
public class Graph1 extends AbstractGraph {  
  
    private final int n = 10;  
  
    public Graph1(String title) {  
        super(title);  
    }  
  
    @Override  
    public void construct() {  
        for (int i = 0; i < n; i++) {  
            Node nn = new Node(String.valueOf(i));  
            this.addNode(nn);  
        }  
        int k=0;  
        addArc(nodes.get(0), nodes.get(1), "a_"+String.valueOf(k));k++;  
        addArc(nodes.get(0), nodes.get(2), "a_"+String.valueOf(k));k++;  
        addArc(nodes.get(0), nodes.get(3), "a_"+String.valueOf(k));k++;  
        ...  
    }  
}
```



# 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
  - それ以上すすめなくなるまで
  - 新たな頂点がなくなるまで
- 戻って、別の弧をたどる



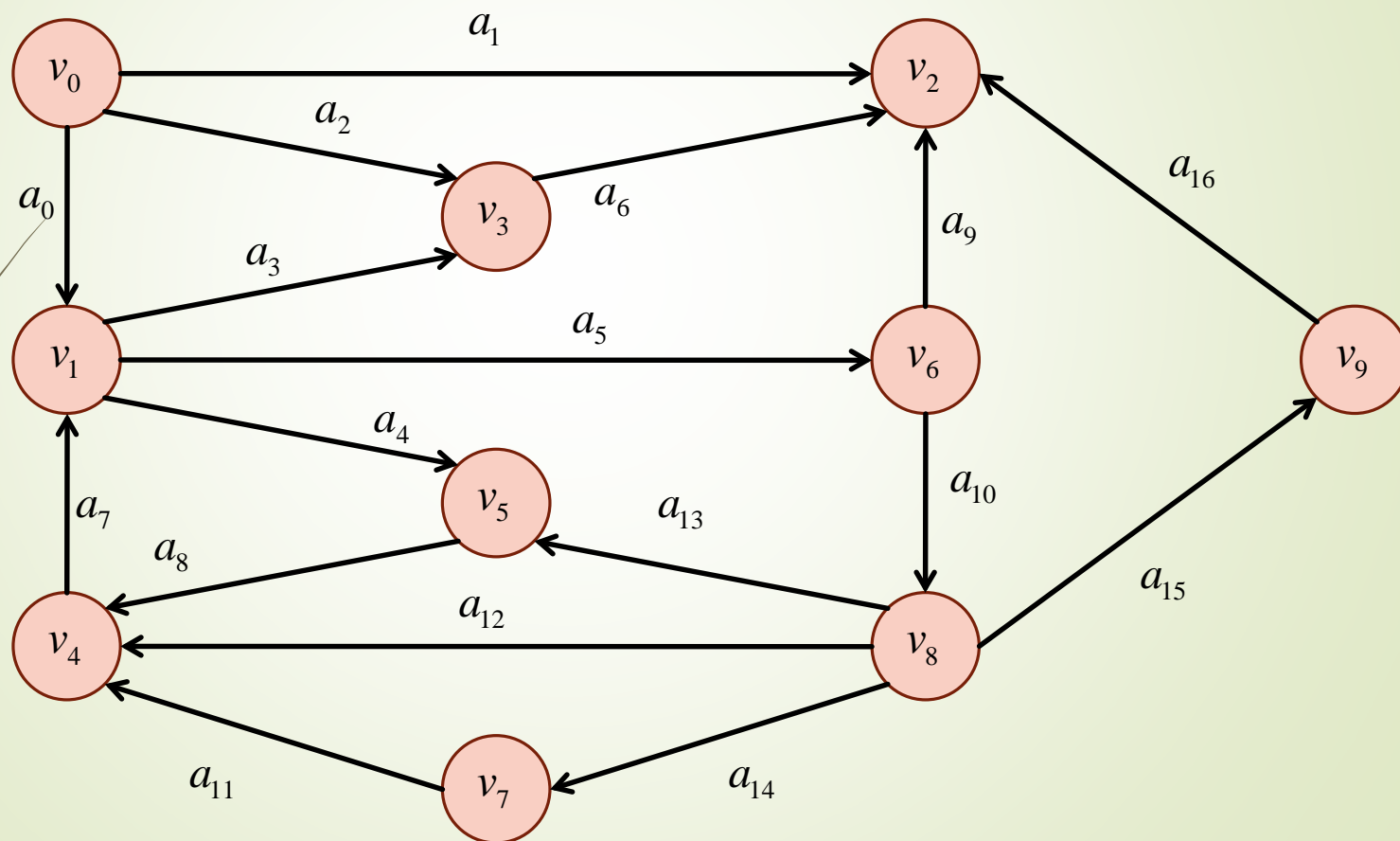
# 再帰的関数で表現

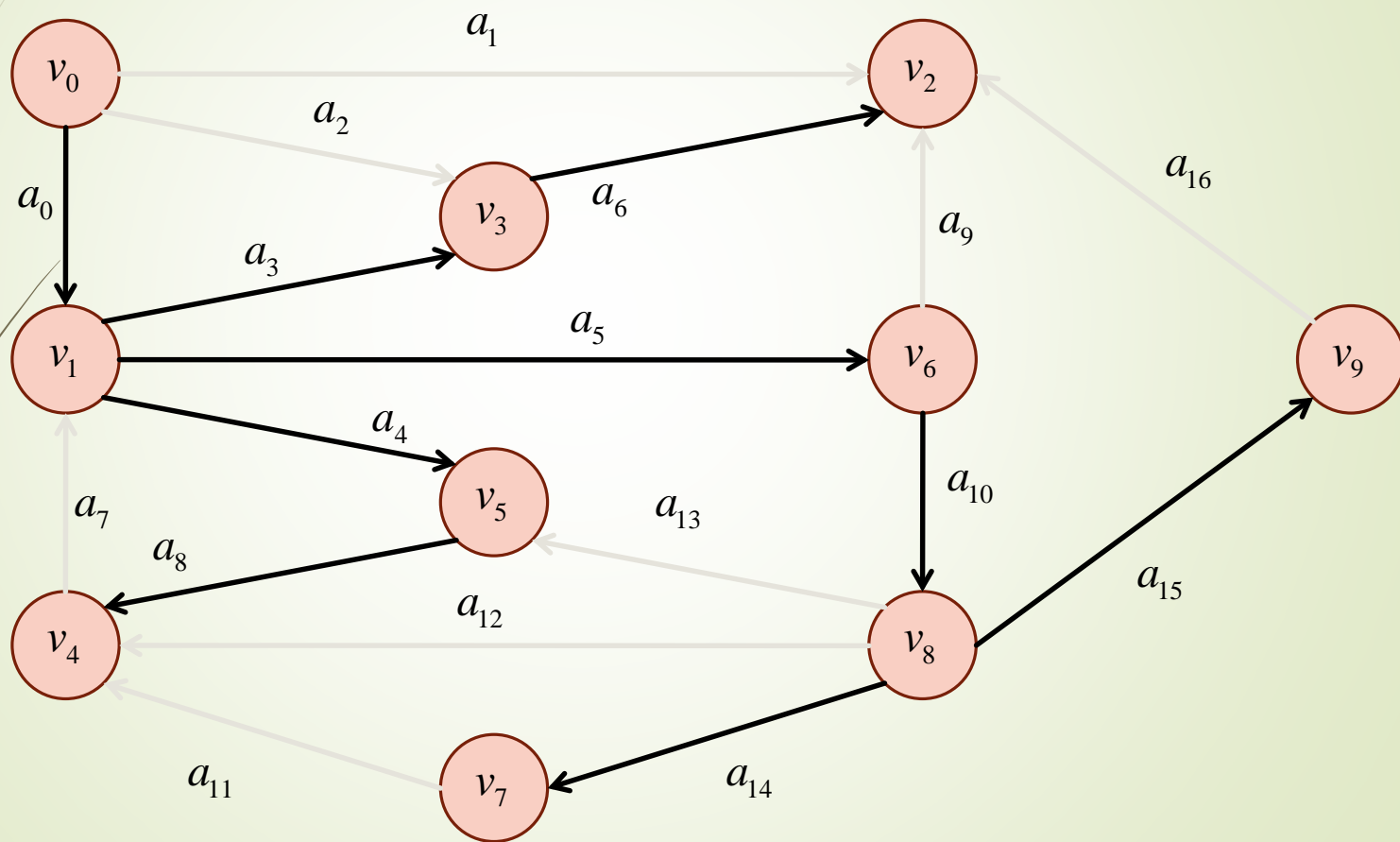
- $L$  : 既にチェックした点のリスト
  - 初期値は空集合
- $v$  : 現在の頂点
- $\delta^+v$  :  $v$ を始点とする弧の集合

再帰的な探索

グラフを深い方向に探索

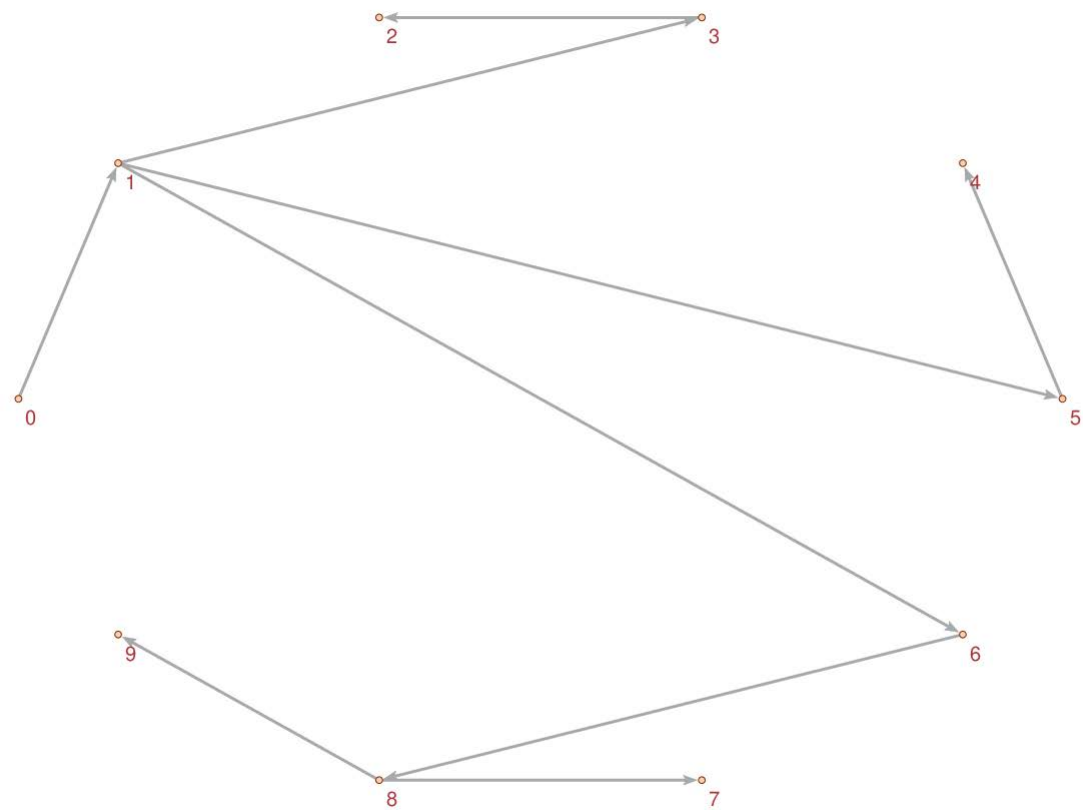
```
search(v) {  
    //vから出る全ての弧  
    forall(  $a \in \delta^+v$  ) {  
         $w = \partial^-a$  //wの終点  
        if(  $w \notin L$  ) {  
             $L \leftarrow L \cup \{w\}$   
            search(w)  
        }  
    }  
}
```





# 探索のクラス

- 探索の抽象クラス
  - グラフと始点を指定
  - 探索結果を返す
    - 使用した弧のリスト
    - グラフ
- DFSやBFSはサブクラスとして実装



# 深さ優先探索 非再帰的実装

- $L$  : すでにチェックした頂点のリスト
- $v_0$  : 探索の始点
- $Q$  : 後で探索すべき弧のスタック

```
 $L = \{v_0\}$   
 $Q \leftarrow \delta^+ v_0 // \text{push}$   
while (  $Q \neq \emptyset$  ) {  
     $a = Q.\text{pop}$   
     $w = \partial^- a // a$ の終点  
    if (  $w \notin L$  ) {  
         $L \leftarrow L \cup \{w\}$   
         $U$ は $w$ と $V \setminus L$ を結ぶ弧の集合  
         $Q \leftarrow U // \text{push}$   
    }  
}
```

$$L = \{v_0\}$$

$$Q = [a_0, a_1, a_2]$$

$$L = \{v_0, v_1\}$$

$$Q = [a_3, a_4, a_5, a_1, a_2]$$

$$L = \{v_0, v_1, v_3\}$$

$$Q = [a_6, a_4, a_5, a_1, a_2]$$

$$L = \{v_0, v_1, v_2, v_3\}$$

$$Q = [a_4, a_5, a_1, a_2]$$

$$L = \{v_0, v_1, v_2, v_3, v_5\}$$

$$Q = [a_8, a_5, a_1, a_2]$$

$$L = \{v_0, v_1, v_2, v_3, v_4, v_5\}$$

$$Q = [a_5, a_1, a_2]$$

$$L = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$Q = [a_{10}, a_1, a_2]$$

$$L = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_8\}$$

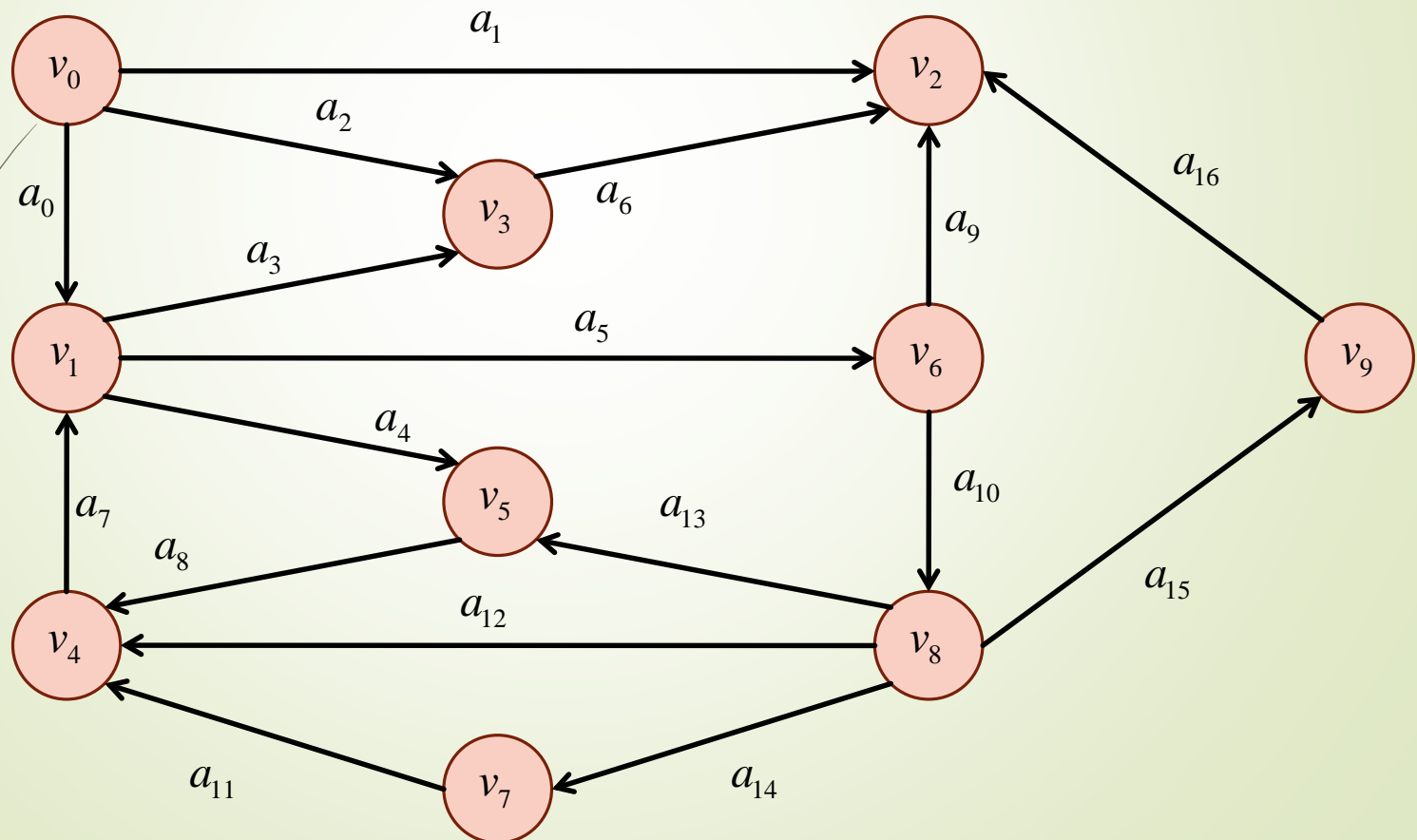
$$Q = [a_{14}, a_{15}, a_1, a_2]$$

$$L = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$$

$$Q = [a_{15}, a_1, a_2]$$

$$L = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$$

$$Q = [a_1, a_2]$$



## 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

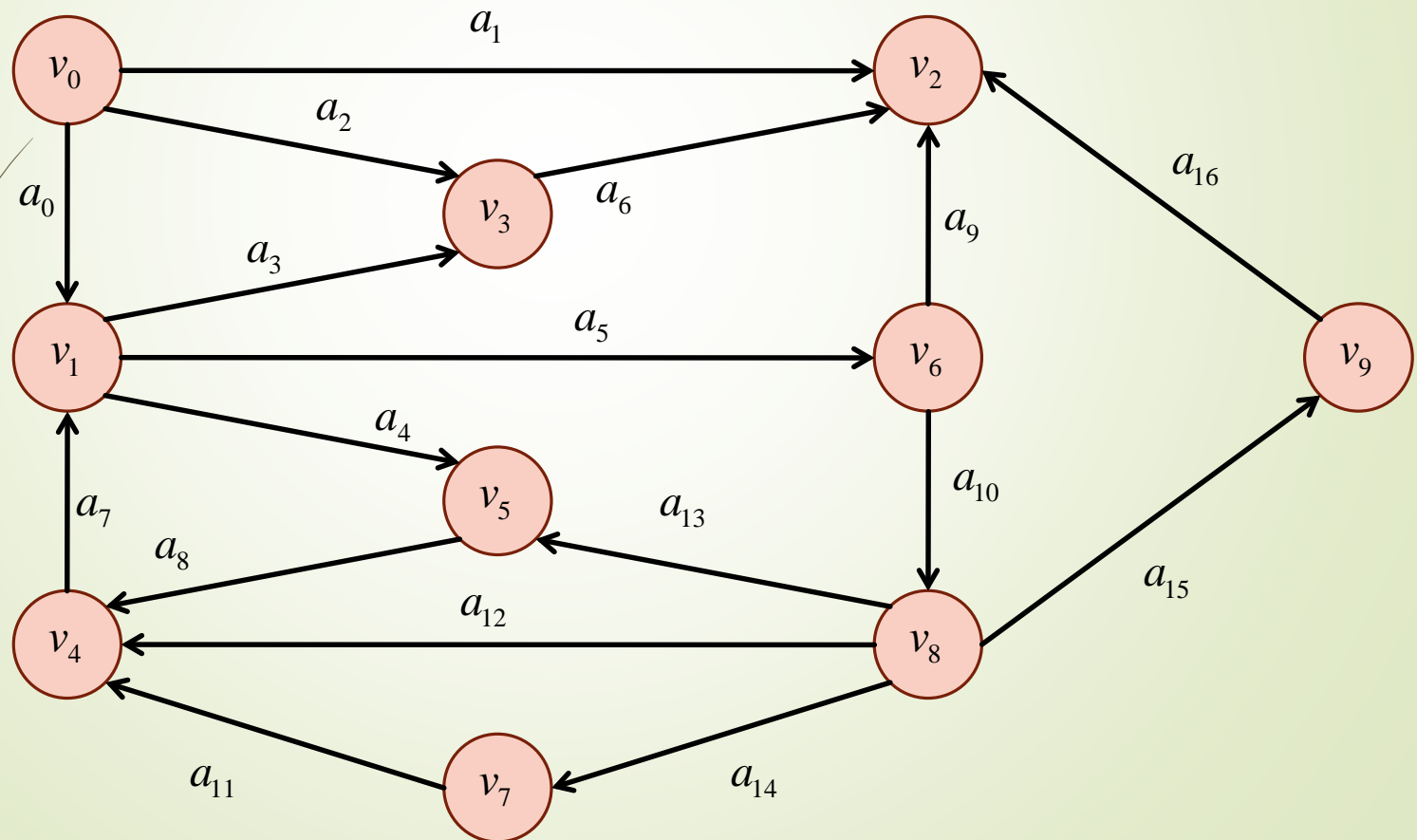
- $L$  : すでにチェックした点のリスト: 初期  $L = \phi$
- $Q$  : 調査すべき点のキュー: 初期  $Q = [r]$

```
L =  $\phi$ 
Q = [r]
while(Q  $\neq \phi$ ) {
    v = Q.poke//先頭
    forall( a  $\in \delta^+v$  ) {
        w =  $\partial^-a$ 
        if( w  $\notin L$  && w  $\notin Q$  ) {
            Q  $\leftarrow$  w
        }
    }
    L  $\leftarrow$  L  $\cup$  {v}
}
```

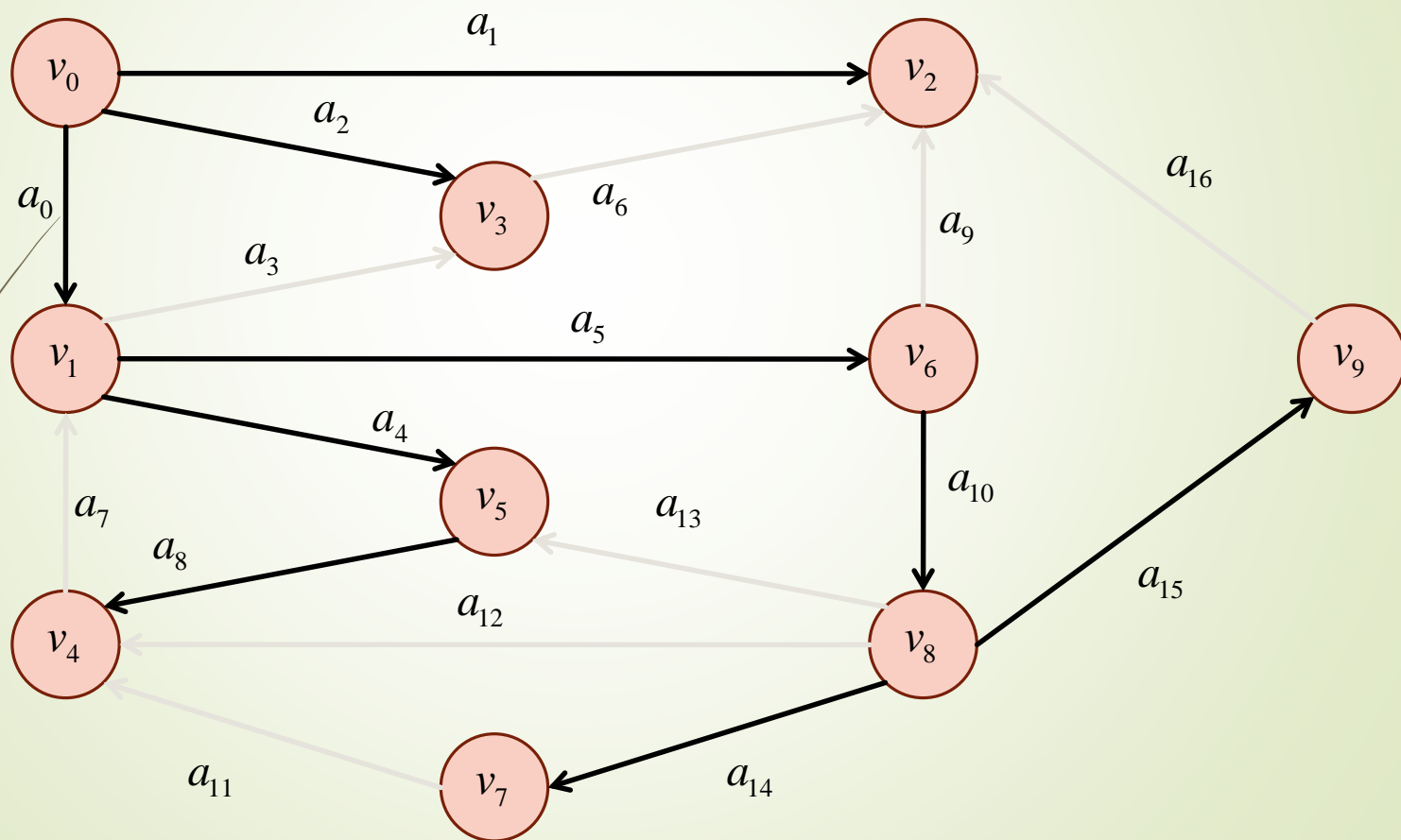
$$\begin{array}{llllll}
 L = \emptyset & L = \{v_0\} & L = \{v_0, v_1\} & L = \{v_0, v_1, v_2, v_3\} & L = \{v_0, v_1, v_2, v_3, v_5\} & L = \{v_0, v_1, v_2, v_3, v_5, v_6\} \\
 Q = [v_0] & Q = [v_1, v_2, v_3] & Q = [v_2, v_3, v_5, v_6] & Q = [v_5, v_6] & Q = [v_6, v_4] & Q = [v_4, v_8]
 \end{array}$$

$$\begin{array}{lll}
 L = \{v_0, v_1, v_2, v_4, v_3, v_5, v_6\} & L = \{v_0, v_1, v_2, v_4, v_3, v_5, v_6, v_8\} & L = \{v_0, v_1, v_2, v_4, v_3, v_5, v_6, v_7, v_8\} \\
 Q = [v_8, v_7, v_9] & Q = [v_7, v_9] & Q = [v_9]
 \end{array}$$

$$\begin{array}{l}
 L = \{v_0, v_1, v_2, v_4, v_3, v_5, v_6, v_7, v_8, v_9\} \\
 Q = [ ]
 \end{array}$$



## 結果



# クラス設計

## 三つの探索の共通部分

```
abstract public class GraphSearch {  
  
    protected List<Arc> arcList; //探索結果となる弧のリスト  
    protected List<Node> nodeList; //すでに探索した頂点のリスト  
    protected final Graph graph; //対象となるグラフ  
    //コンストラクタ  
    public GraphSearch(Graph graph) {this.graph = graph;}  
    //探索の実行  
    abstract public List<Arc> doSearch(Node start);  
  
    protected void initialize(){  
        arcList = Utils.createList(); nodeList = Utils.createList();  
    }  
    //探索の実装部分  
    abstract protected void searchSub(Node v);  
}
```

