グラフの探索

オブジェクト指向プログラミング特論

2016年度

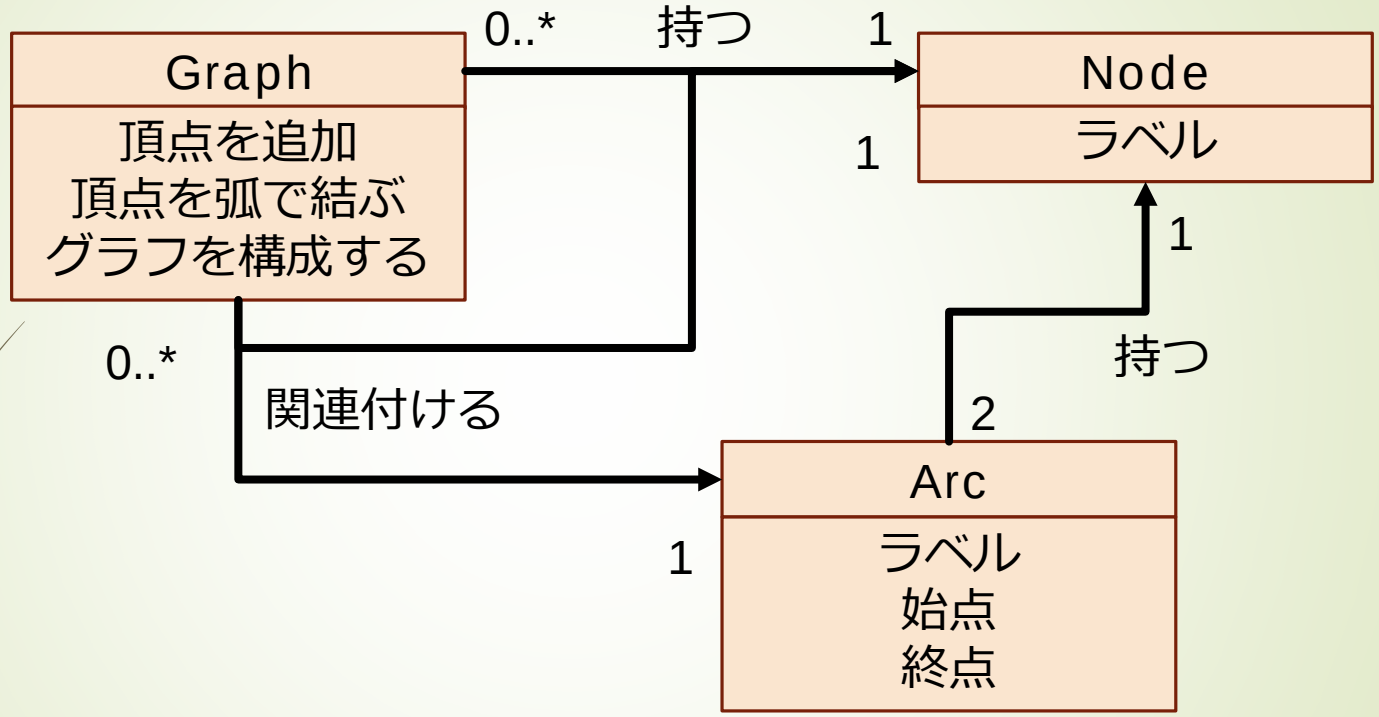
只木進一：工学系研究科

グラフ(Graph)

- 頂点(Node)を弧(Arc)で連結したものをグラフという
  - 弧には向きがあるとする：有向グラフ
- 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

- Graph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


Graph
頂点を追加
頂点を弧で結ぶ
グラフを構成する
0..*
持つ
1
Node
ラベル
1
1
持つ
0..*
関連付ける
2
Arc
1
ラベル
始点
終点

```
public abstract class Graph {

    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);
    }
```

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

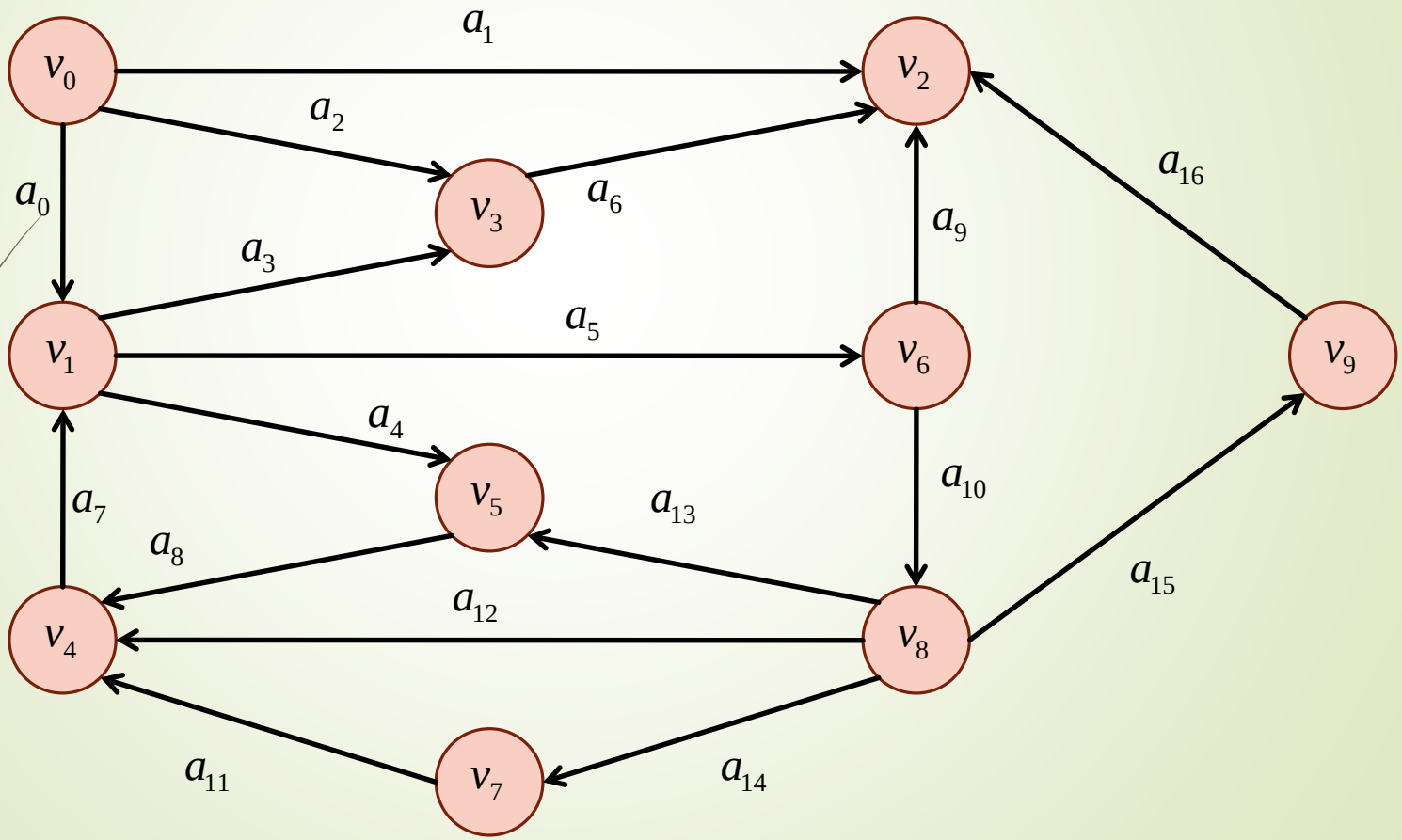
再帰的関数で表現

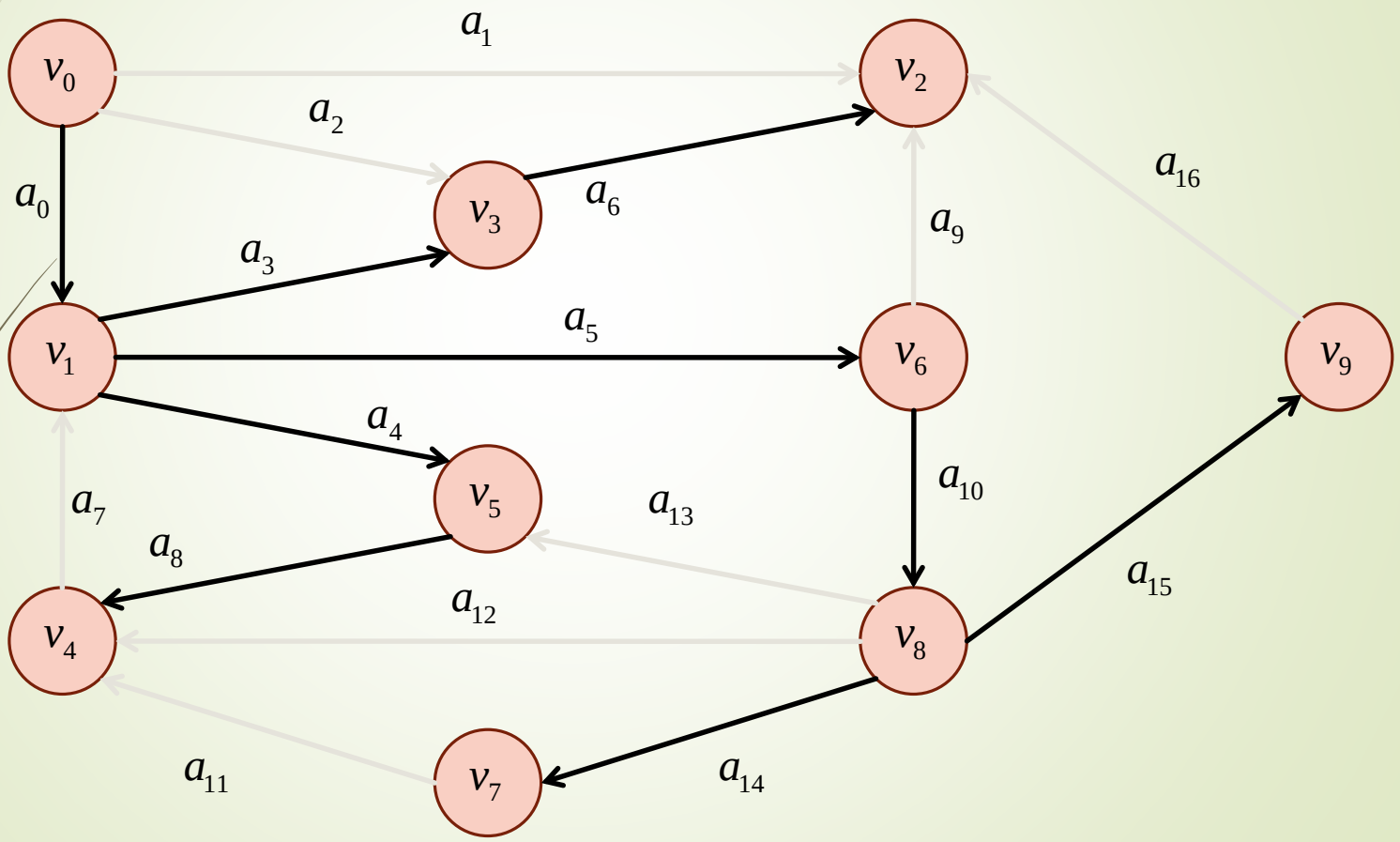
- L：既にチェックした点のリスト
  - 初期値は空集合
- v：現在の頂点
- δ^+v：vを始点とする弧の集合

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 ∈ δ⁺v ) {
        w = ∂⁻a //wの終点
        if( w ∉ L ) {
            L ← L ∪ {w}
            search(w)
        }
    }
}
```


a_1
v_0
v_2
a_2
a_0
v_3
a_6
a_{16}
a_9
a_3
a_5
v_1
v_6
v_9
a_4
a_7
v_5
a_{13}
a_{10}
a_8
a_{15}
a_{12}
v_4
v_8
a_{11}
v_7
a_{14}


v_0
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
a_0
a_1
a_2
a_3
a_4
a_5
a_6
a_7
a_8
a_9
a_{10}
a_{11}
a_{12}
a_{13}
a_{14}
a_{15}
a_{16}

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

$L = \{v_0\}$

$Q \longleftarrow \delta^+ v_0$//push

while ($Q \neq \emptyset$){

  $a = Q$.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

    $Q \longleftarrow U$//push

  }

}

$L=\{v_0\}$ $Q=[a_0,a_1,a_2]$

$L=\{v_0,v_1\}$ $Q=[a_3,a_4,a_5,a_1,a_2]$

$L=\{v_0,v_1,v_3\}$ $Q=[a_6,a_4,a_5,a_1,a_2]$

$L=\{v_0,v_1,v_2,v_3\}$ $Q=[a_4,a_5,a_1,a_2]$

$L=\{v_0,v_1,v_2,v_3,v_5\}$ $Q=[a_8,a_5,a_1,a_2]$

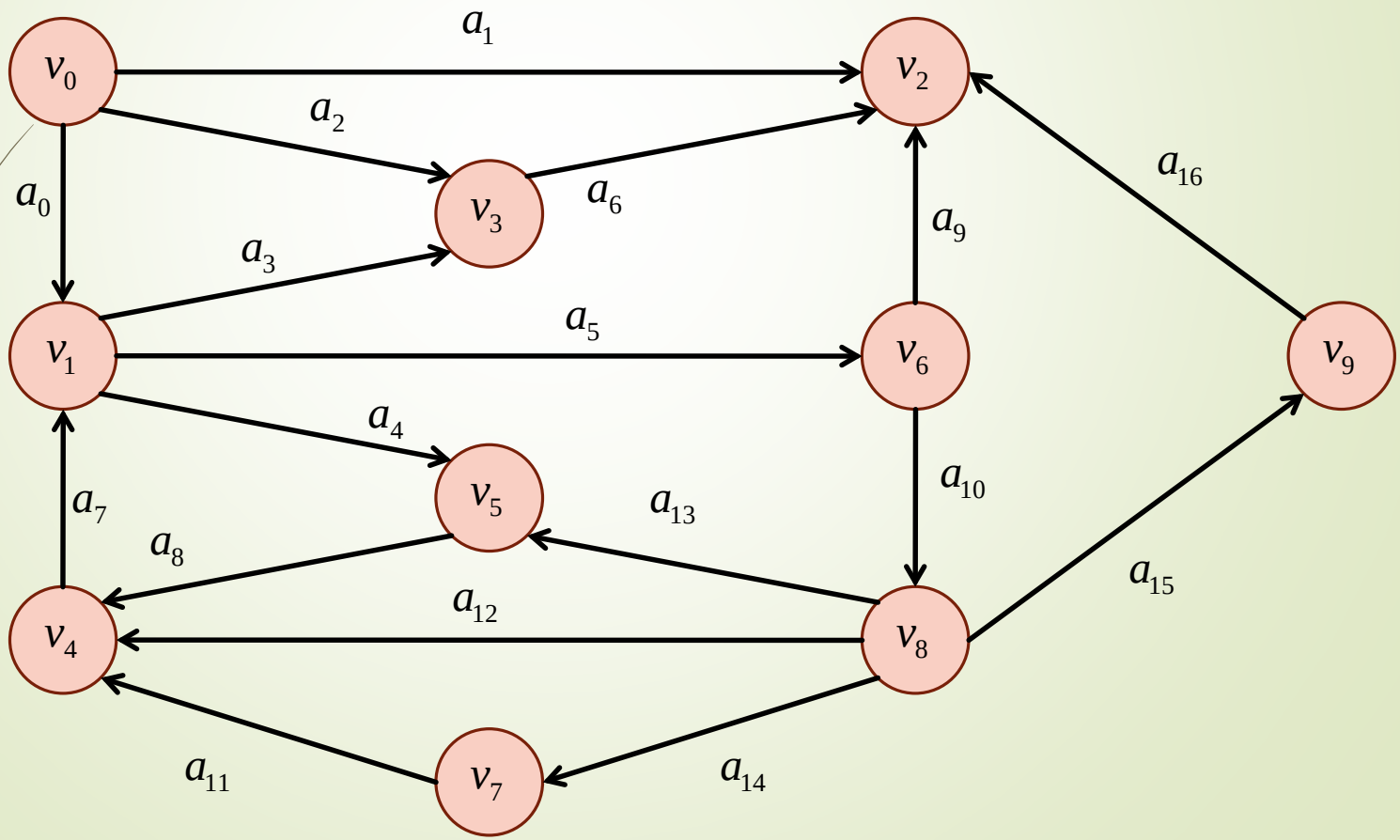
$L=\{v_0,v_1,v_2,v_3,v_4,v_5\}$ $Q=[a_5,a_1,a_2]$

$L=\{v_0,v_1,v_2,v_3,v_4,v_5,v_6\}$ $Q=[a_{10},a_1,a_2]$

$L=\{v_0,v_1,v_2,v_3,v_4,v_5,v_6,v_8\}$ $Q=[a_{14},a_{15},a_1,a_2]$

$L=\{v_0,v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8\}$ $Q=[a_{15},a_1,a_2]$

$L=\{v_0,v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8,v_9\}$ $Q=[a_1,a_2]$



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$Q = [r]$

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

$L = \varnothing$
$Q = [v_0]$

$L = \{v_0\}$
$Q = [v_1, v_2, v_3]$

$L = \{v_0, v_1\}$
$Q = [v_2, v_3, v_5, v_6]$

$L = \{v_0, v_1, v_2, v_3\}$
$Q = [v_5, v_6]$

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

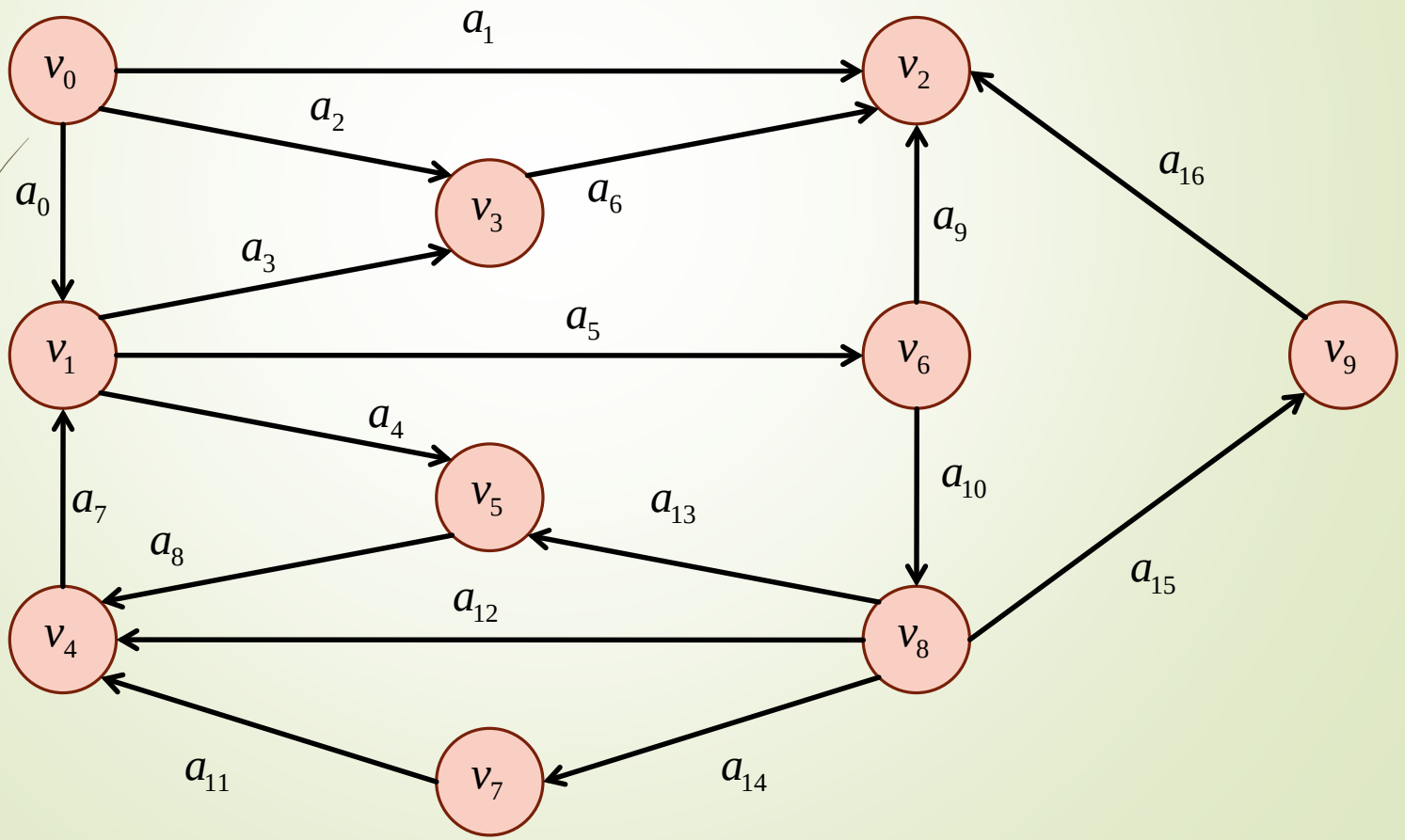
$L = \{v_0, v_1, v_2, v_3, v_5, v_6\}$
$Q = [v_4, v_8]$

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

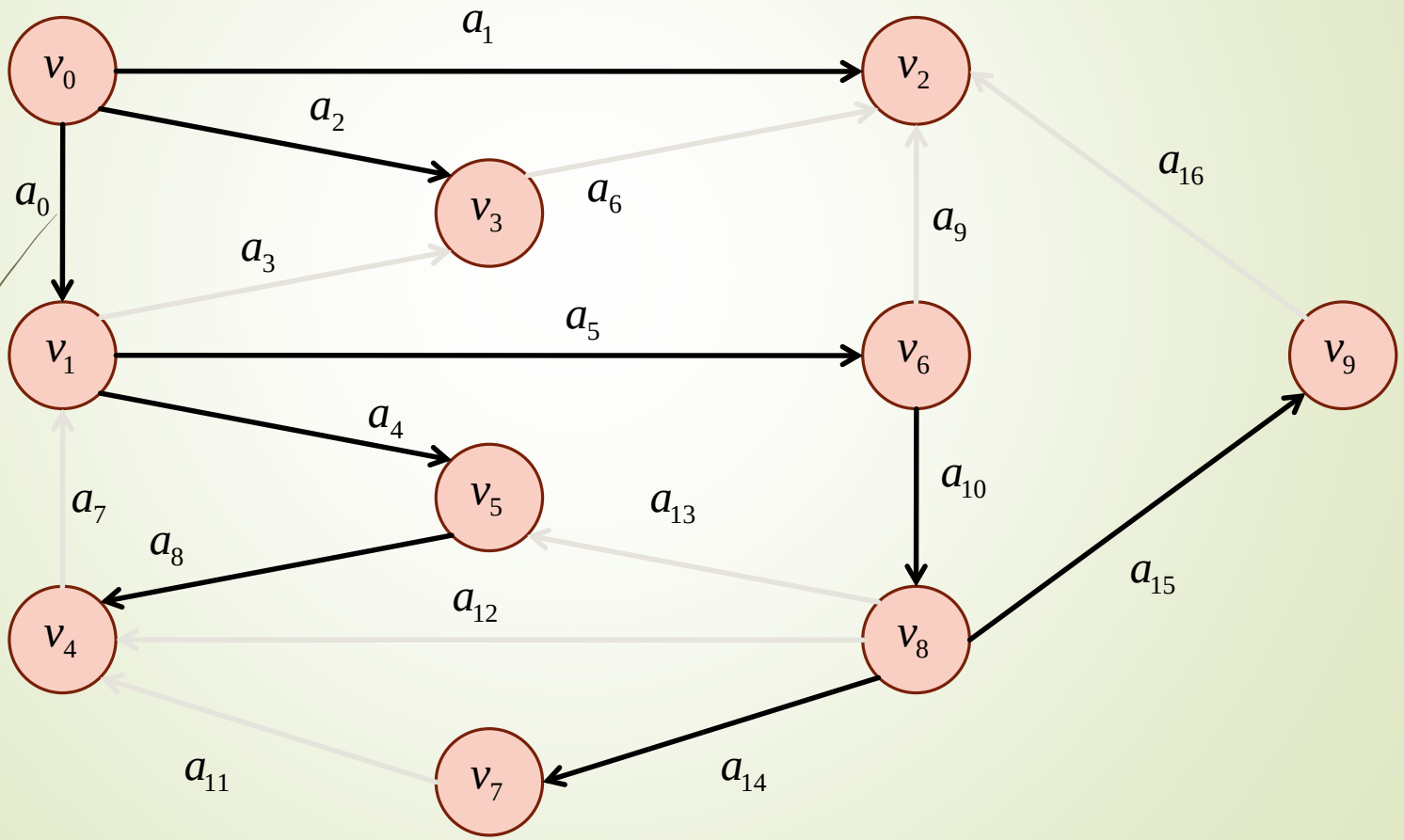
$L = \{v_0, v_1, v_2, v_4, v_3, v_5, v_6, v_8\}$
$Q = [v_7, v_9]$

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

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



結果

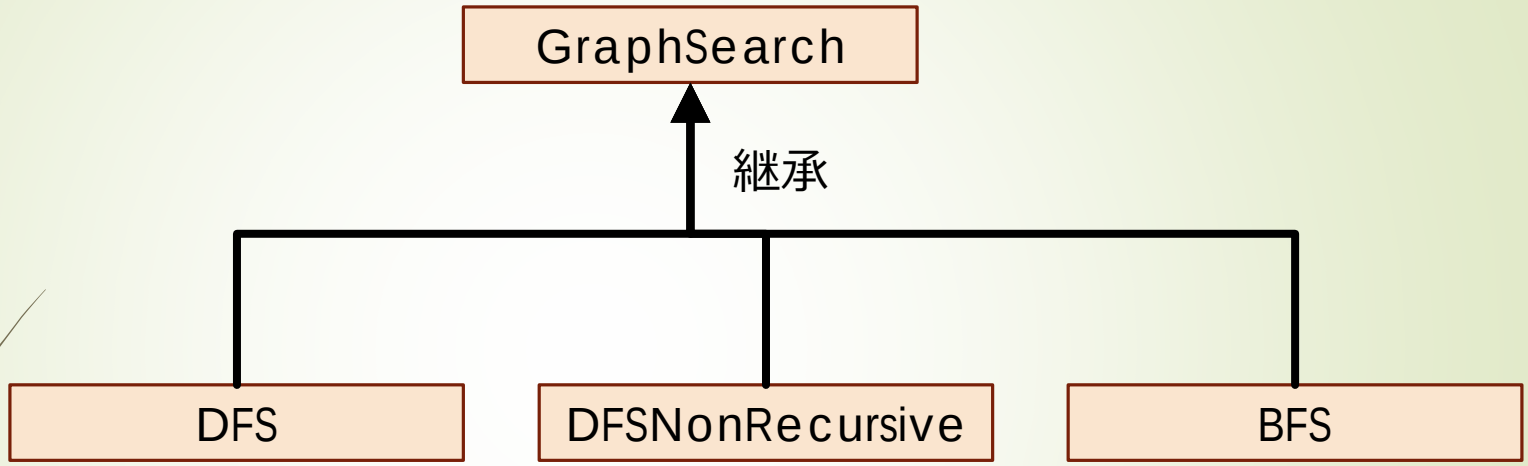

v_0
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
a_0
a_1
a_2
a_3
a_4
a_5
a_6
a_7
a_8
a_9
a_{10}
a_{11}
a_{12}
a_{13}
a_{14}
a_{15}
a_{16}

クラス設計
三つの探索の共通部分

```
bstract 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);
}
```


GraphSearch
継承
DFS
DFSNonRecursive
BFS

DFS

```
public List<Arc> doSearch(Node start) {
   initialize();  nodeList.add(start);
   searchSub(start);
   return arcList;
}

protected void searchSub(Node v) {
   List<Arc> arcs = graph.getArcs(v);
   if (arcs == null) return;
   arcs.stream().forEach((a) -> {
      Node w = a.end;
      if (!nodeList.contains(w)) {
         arcList.add(a);   nodeList.add(w);
         searchSub(w);
      }
   });
}
```

BFS

```
    public List<Arc> doSearch(Node start) {
        initialize();  searchSub(start);
        return arcList;
    }
    protected void searchSub(Node start) {
        LinkedList<Node> queue = new LinkedList<>();
        queue.add(start);
        while (!queue.isEmpty()) {
            Node v = queue.poll();
            List<Arc> aList = graph.getArcs(v);
            if (aList != null) {
                aList.stream().forEach(
                        a -> {  Node w = a.end;
                            if (!nodeList.contains(w) && !queue.contains(w)) {
                                queue.add(w);  arcList.add(a);  }
                        });
            }
            nodeList.add(v);
}}
```