グラフアルゴリズム
閉路の列挙

オブジェクト指向プログラミング特論

2020年度

只木進一：工学系研究科

今日のサンプルプログラム

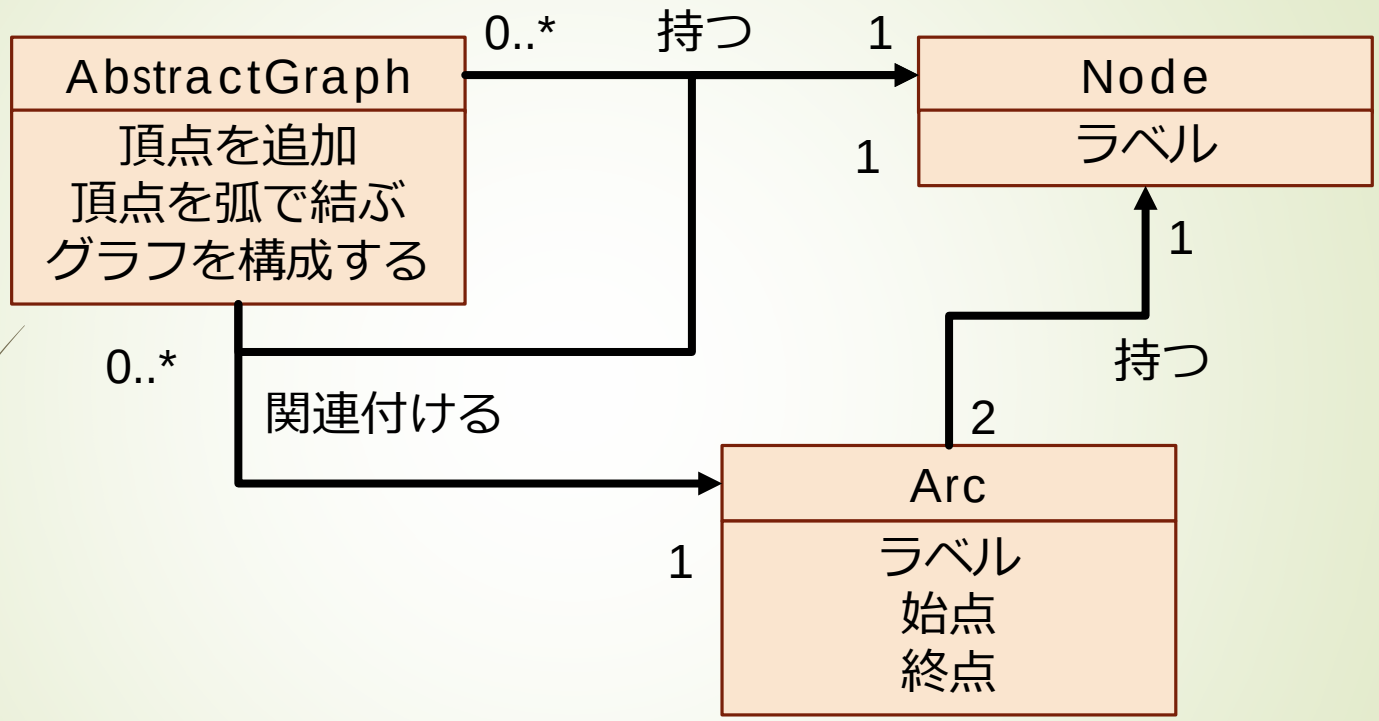
- https://github.com/oop-mc-saga/GraphAlgorithm

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


AbstractGraph
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
ラベル
始点
終点
1

```
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);
    }
```

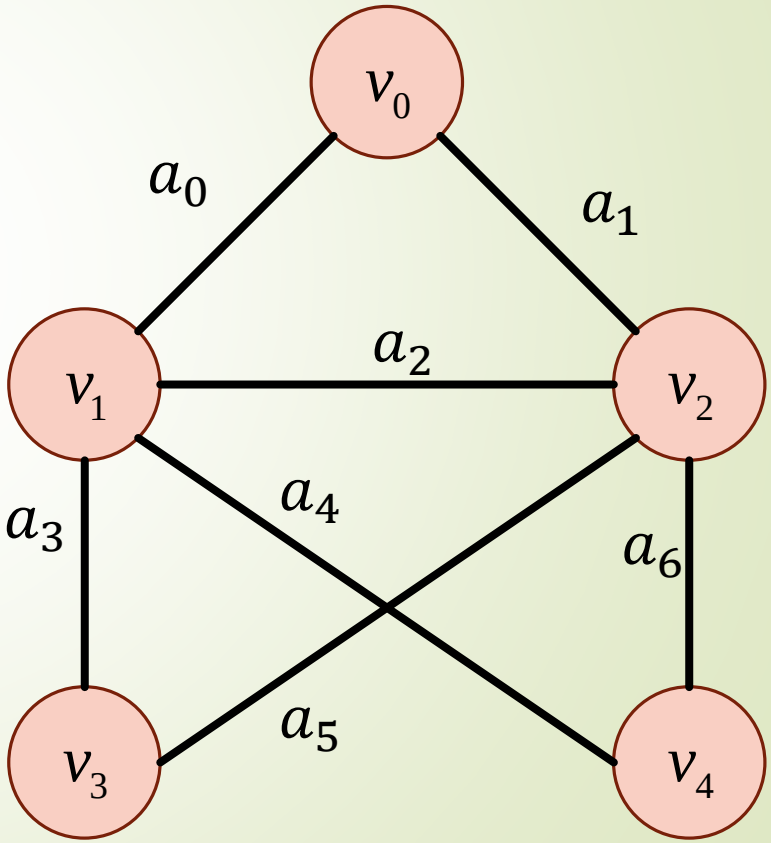
```
public void addArc(Node from, Node to, String label) {
    Arc arc = new Arc(from, to, label);
    if (!node2arc.containsKey(from)) {
        List<Arc> arcList = Utils.createList();
        node2arc.put(from, arcList);
    }
    node2arc.get(from).add(arc);
    if (!directed) {
        if (!node2arc.containsKey(to)) {
            List<Arc> arcList = Utils.createList();
            node2arc.put(to, arcList);
        }
        node2arc.get(to).add(arc);
    }
}
```

グラフの定義

```
1.  public class Graph1 extends AbstractGraph {
2.
3.      private final int n = 5;
4.
5.      public Graph1(String title) {
6.          super(title);
7.          setDirected(false);
8.      }
9.
10.     @Override
11.     public void construct() {
12.         for (int i = 0; i < n; i++) {
13.             Node nn = new Node(String.valueOf(i));
14.             this.addNode(nn);
15.         }
16.         int k=0;
17.         addArc(nodes.get(0), nodes.get(1), "a_"+String.valueOf(k));k++;
18.         addArc(nodes.get(0), nodes.get(2), "a_"+String.valueOf(k));k++;
19.         addArc(nodes.get(1), nodes.get(2), "a_"+String.valueOf(k));k++;
20.          …
21.     }
22. }
```

Euler閉路(Euler Circle)

- 無向グラフが対象
- 一筆書き
  - 全ての弧を一度ずつ経由して始点に戻る道
- 全ての点の次数が偶数



Euler閉路の列挙の方針
全てのEuler閉路を見つける

- 使用した弧の列を管理する
- 深さ優先で、一つの閉路を見つける
  - 使った弧の一覧を保持
- 分岐点まで戻って、他の閉路を見つける
- 上記を繰り返す

Euler閉路の列挙アルゴリズム

```
1.  search(v, A_Euler) {
2.      if( (v == r) ∧ (|A| == |A_Euler|) ) {
3.          見つかったEuler閉路を保存
4.      } else {
5.          forall( a ∈ δv ) {
6.              if( a ∉ A_Euler ) {
7.                  A'_Euler = A_Euler ∪ {a}
8.                  w = ∂a \ {v}
9.                  search( w, A'_Euler)
10.             }
11.         }
12.     }
13. }
```

A_{Euler}：既に経由した弧の列、初期値はø
r：始点

Euler閉路の列挙アルゴリズム

```
1.  search(v, A_Euler) {
2.      if( (v == r) ∧ (|A| == |A_Euler|) ) {
3.          見つかったEuler閉路を保存
4.      } else {
5.          forall( a ∈ δv ) {
6.              if( a ∉ A_Euler ) {
7.                  A'_Euler = A_Euler ∪ {a}
8.                  w = ∂a \ {v}
9.                  search( w, A'_Euler)
10.     }
11.         }
12.     }
13. }
```

A_{Euler}：既に経由した弧の列、初期値はø
r：始点

```
1.  public class Euler {

2.      final private List<List<Arc>> curcuitList;//閉路のリスト
3.      final private AbstractGraph g;//対象となる無向グラフ
4.      final private Node start;//始点
5.      final private int L;//弧の総数

6.      public Euler(AbstractGraph g, Node start) {
7.          curcuitList = Utils.createList();
8.          this.start = start;
9.          this.g = g;
10.         L = g.getArcs().size();
11.     }
```

```
1.  public List<List<Arc>> startEnumerate() {
2.          List<Arc> aEuler = Utils.createList();
3.          enumerateSub(start, aEuler);
4.          return curcuitList;
5.      }
```

```
1.      private void enumerateSub(Node v, List<Arc> aEuler) {
2.          if (v == start && aEuler.size() == L) {
3.              //見つけた閉路を保存
4.              curcuitList.add(aEuler);
5.              return;
6.          }
7.          //vを始点とする全ての弧について調べる
8.          List<Arc> arcList = g.getArcs(v);
9.          if (arcList != null) {
10.             for (Arc a : arcList) {
11.                 if (!aEuler.contains(a)) {
12.                     List<Arc> newList = Utils.createList();
13.                     newList.addAll(aEuler);
14.                     newList.add(a);
15.                     Node w = a.getAnotherEnd(v);
16.                     enumerateSub(w, newList);
17.                 }
18.             }
19.         }
20.     }
```

[a_0, a_2, a_5, a_3, a_4, a_6, a_1]
[a_0, a_2, a_6, a_4, a_3, a_5, a_1]
[a_0, a_3, a_5, a_2, a_4, a_6, a_1]
[a_0, a_3, a_5, a_6, a_4, a_2, a_1]
[a_0, a_4, a_6, a_2, a_3, a_5, a_1]
[a_0, a_4, a_6, a_5, a_3, a_2, a_1]
[a_1, a_2, a_3, a_5, a_6, a_4, a_0]
[a_1, a_2, a_4, a_6, a_5, a_3, a_0]
[a_1, a_5, a_3, a_2, a_6, a_4, a_0]
[a_1, a_5, a_3, a_4, a_6, a_2, a_0]
[a_1, a_6, a_4, a_2, a_5, a_3, a_0]
[a_1, a_6, a_4, a_3, a_5, a_2, a_0]