アルゴリズムとデータ構造

第14回　グラフ構造

第14回のキーワード

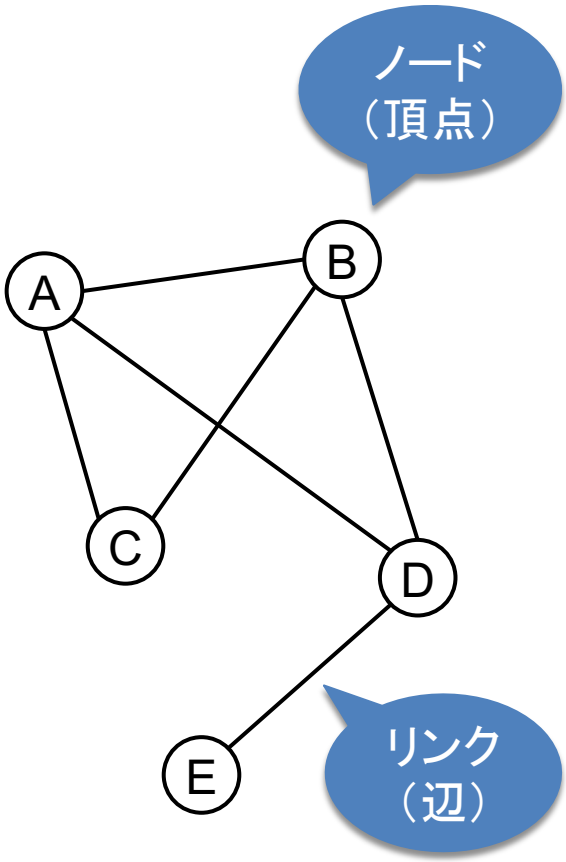
アルゴリズム関係

- グラフ構造（graph structure）
- 無向グラフ（undirected graph）
- 有向グラフ（directed graph）
- 重み付きグラフ（weighted graph）
- 隣接行列/リスト（adjacency matrix/list）
- ダイクストラ法（Dijkstra’s algorithm）

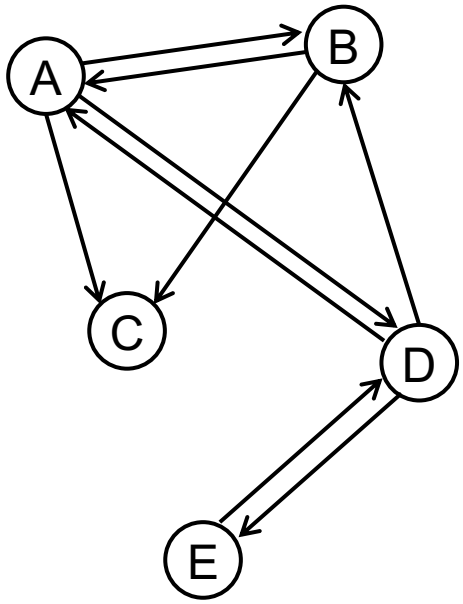
Java関係

- コレクションフレームワーク
- List, ArrayList, LinkedList
- Set, HashSet, TreeSet
- Map, HashMap, TreeMap
- 匿名クラス
- 関数型インタフェース
- ラムダ式
- forEachメソッド

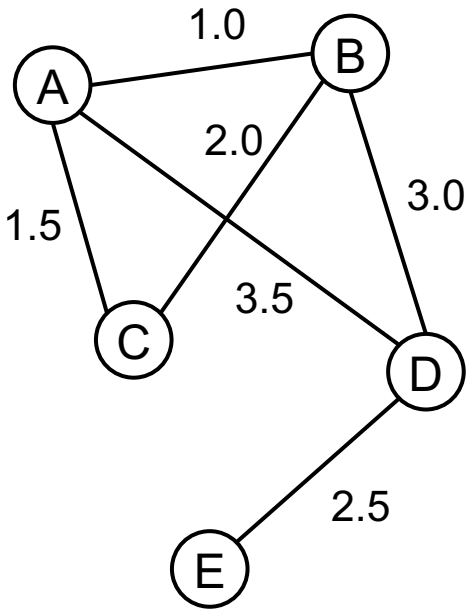
グラフ構造



無向グラフ



有向グラフ



重み付きグラフ

隣接行列

- ノード間の接続状態を要素とする行列

	A	B	C	D	E
A	0	1	1	1	0
B	1	0	1	1	0
C	1	1	0	0	0
D	1	1	0	0	1
E	0	0	0	1	0

無向グラフ

対称行列になる
$a_{ij} = a_{ji}$

	A	B	C	D	E
A	0	1	1	1	0
B	1	0	1	0	0
C	0	0	0	0	0
D	1	1	0	0	1
E	0	0	0	1	0

有向グラフ

一方通行があるので
$a_{ij} = a_{ji}$とは限らない

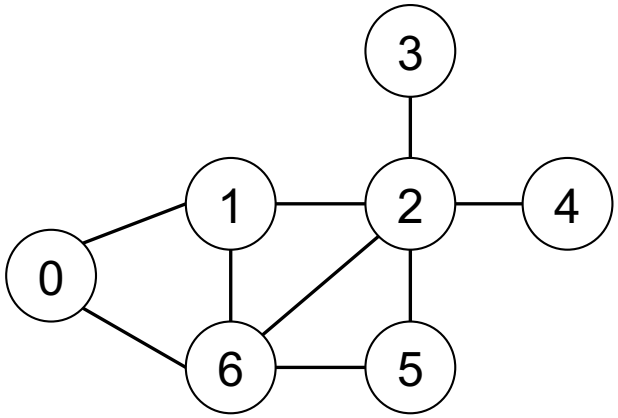
	A	B	C	D	E
A	∞	1.0	1.5	3.5	∞
B	1.0	∞	2.0	3.0	∞
C	1.5	2.0	∞	∞	∞
D	3.5	3.0	∞	∞	2.5
E	∞	∞	∞	2.5	∞

重み付きグラフ

リンクがない場合は
0または∞とする

確認問題

- 演習課題1
  - Javaの2次元配列で隣接行列を定義せよ

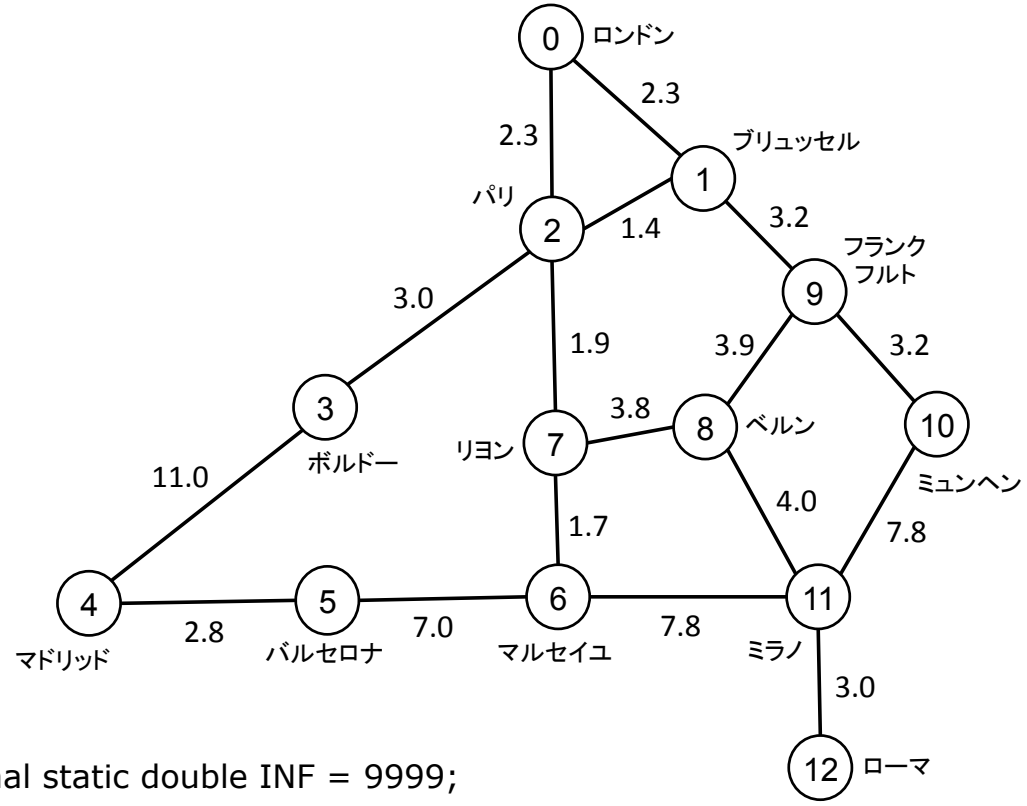



```
// 隣接行列：ノードiとjが連結していれば
// adj[i][j]==1，そうでなければ0を設定
private final static int[][] adj = {
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    }};
```

グラフの探索

- 深さ優先探索
  - 訪問したノードにつながるリンクを優先してたどっていく
  - 始点から先へ先へと遠くへ探索していく
- 幅優先探索
  - 先に発見してたどっていないリンクを優先にたどっていく
  - 始点から探索範囲を段階的に広げていく
- グラフの探索における注意
  - 同じノードに複数の経路からたどり着く可能性がある
  - 各ノードが訪問済みなのか記憶しておく必要がある

重み付きグラフの例




```
private final static double INF = 9999;

private final static double[][] adj = {
   { INF, 2.3, 2.3, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF },  // ロンドン

   /* 途中11行省略 */

   { INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, 3.0, INF }   // ローマ
};
```

最短経路問題

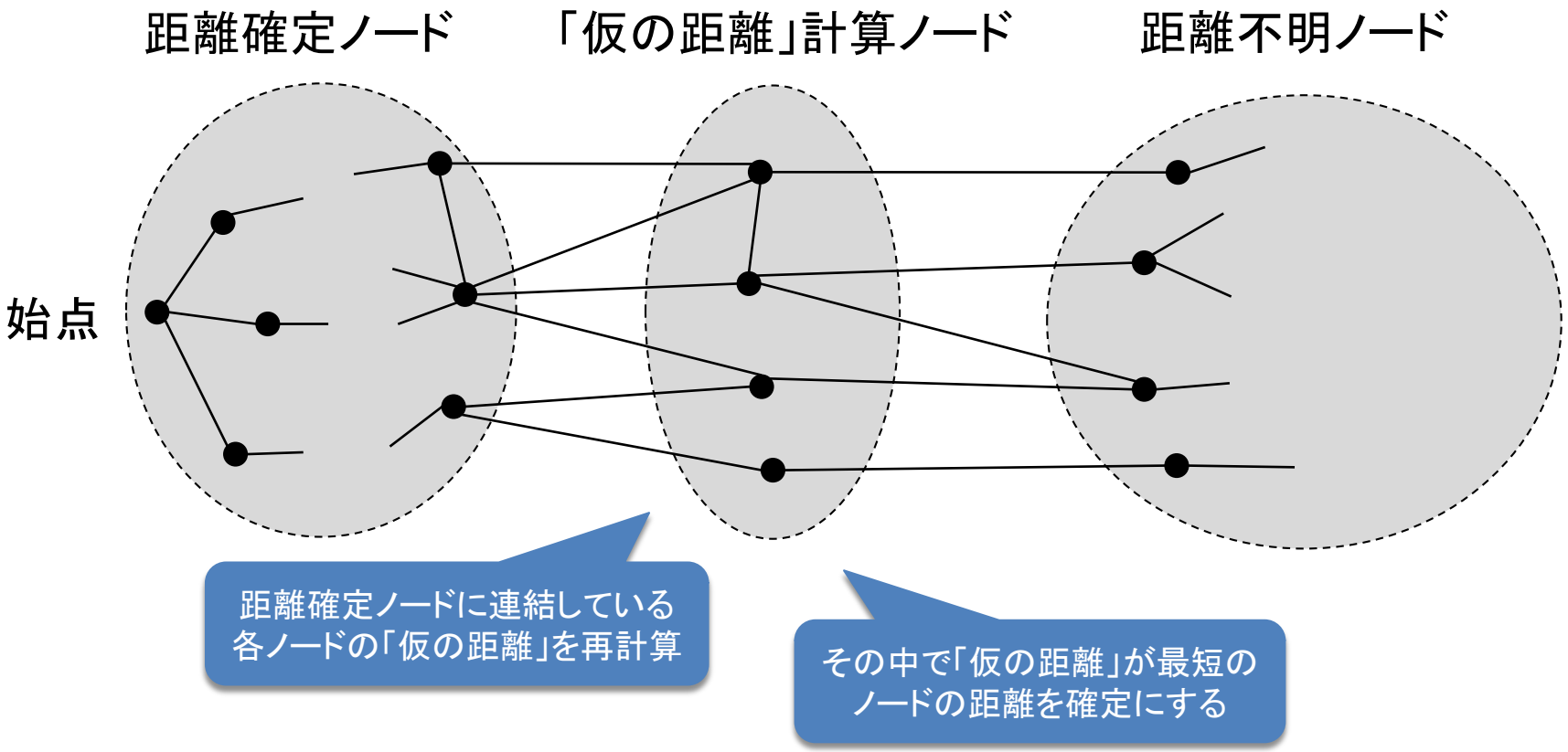
- ダイクストラ法
  - 重み付きグラフで，始点から全ノードへの最短経路を求める
  - 理論的に，2点間だけの最短経路を求める確実な方法はない

- アルゴリズム
  - 基本方針：距離が未確定のノードから，始点に最も近いものを1つずつ選んで距離を確定し，全ノードが確定したら終わり

1. 最初は，全ノードの距離を未確定に設定する
2. 全ノードが確定するまで以下を繰り返す
   - ① 未確定のノードの中から，始点からの「仮の距離と経路」が最短になるノード v を選択する（最初は始点が選択される）
   - ② 始点から v までの距離と経路を確定済みにする
   - ③ v から直接連結するノードの「仮の距離と経路」を更新する

ダイクストラ法

- 「仮の距離」が最短のノードを選んで確定していく
  - 重みなしグラフの場合，幅優先探索と同様になる



Javaコレクションクラス

クラス	内部構造	親インタフェース	インタフェースの説明と代表的なメソッド
ArrayList<E>	配列	List<E>	データを一列に並べて格納する（第6回，第10回で説明）
LinkedList<E>	連結リストの一種		
HashSet<E>	ハッシュテーブル	Set<E>	要素が重複しないようにデータを格納する（集合） add(data), contains(data), remove(data), size(), isEmpty()
TreeSet<E>	2分探索木の一種		
HashMap<K, V>	ハッシュテーブル	Map<K, V>	キーと値のペアでデータを格納する（写像，辞書，連想配列） put(key, value), get(key), remove(key), isEmpty(), size(), containsKey(key), entrySet(), keySet(), values()
TreeMap<K, V>	2分探索木の一種		
Map.Entry<K, V>			Map<K, V>の各要素を表す内部クラス getKey(), getValue()

親インタフェースがデータ構造に対する抽象的な操作方法を提供する
なるべく親インタフェースを介して用いると具体的な実装を変更可能になる

匿名クラスとラムダ式

- 匿名クラス
  - 抽象クラスかインタフェースを継承/実装した“使い捨て”のクラスを定義し，そのインスタンスを生成する方法
  - class文を使わないので，クラスに名前がつかない
  - 形式： new 親クラス名 { クラスメンバの定義 }

または，親インタフェース名

- ラムダ式
  - 一般的な用語としては，関数を式として定義する記法
  - Javaの場合，メソッドが1つしかないインタフェースを実装した匿名クラスのインスタンスを生成する式
  - 形式： (引数並び) -> { 唯一のメソッドの処理 }