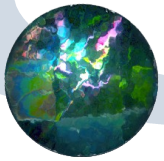
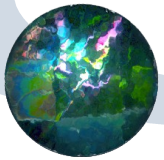
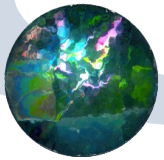


最小木問題
Minimum-tree Problem



ネットワーク(networks)

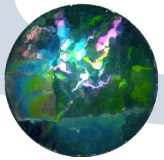
- 弧に長さ、重み、費用などの属性のあるグラフ
  - 都市とそれを結ぶ交通：距離、費用
  - コンピュータとネットワーク：帯域
  - 作業工程：所用時間、遅れ



最小木問題

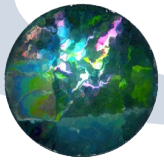
- 連結無向グラフ$G=(V,A)$
- 重み関数 $w:A \rightarrow R$
- G の木 $T \subseteq A$
- 重みが最小になる木を見付ける

$$w(T)=\sum_{a \in T} w(a)$$



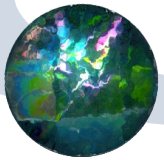
最小木問題の応用

- 油井（ゆせい）から精油所へパイプラインを引く
  - 最短のパイプラインで一カ所に原油を集める
- 中央から、最短時間で、各部署に指示をする



二つのアプローチ

- Kruskal法
  - 重み最小の弧を選ぶ
  - 途中は木になっていない
- Jarník-Primアルゴリズム
  - 始点から開始して、連結した頂点の数を増やす



貪欲アルゴリズム (Greedy algorithm, Kruskal algorithm)

$T=\emptyset$

$w_{\max}$

while (T は G の極大木でない){

$w=w_{\max}$

forall($a\in A\setminus T$){

if($w(a)<w$)){

if($T\cup\{a\}$ は閉路を持たない){

$a_{\text{new}}=a$

$w=w(a)$

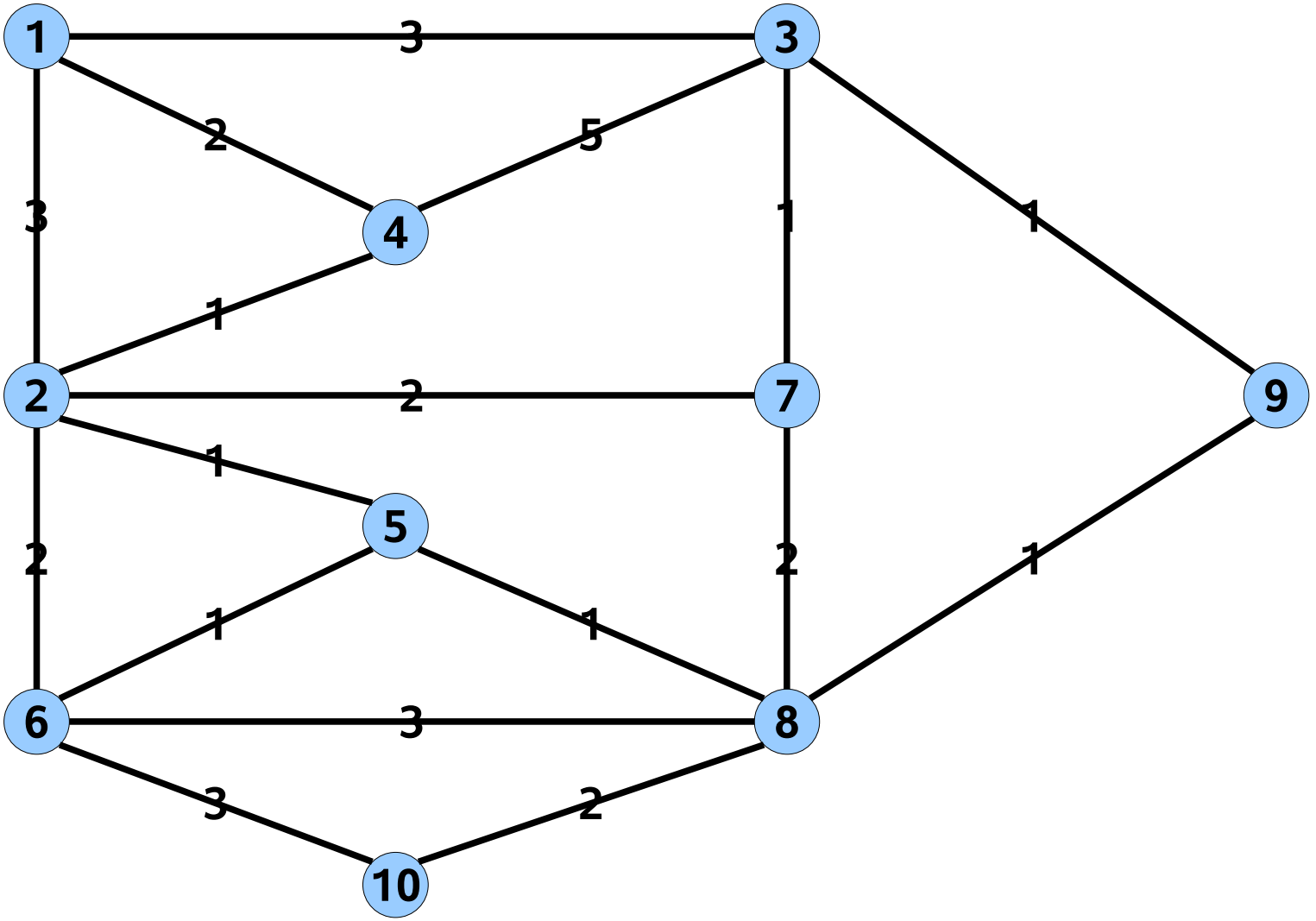
}

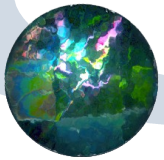
}

}

$T=T\cup\{a_{\text{new}}\}$

}


1
3
3
2
5
3
4
1
1
1
2
2
7
9
1
5
2
2
1
1
1
6
3
8
3
2
10

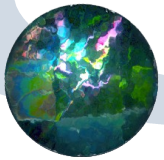


貪欲アルゴリズムが正しい理由

- 次の定理を証明すれば良い

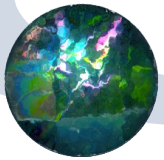
  貪欲アルゴリズム実行中で得られる T は、枝数$|T|$を持ち、サーキットを含まない枝集合のうちで、その重みが最小である。

- 要するに、アルゴリズムの各段階で最小の重みのグラフであることを示す。
- 証明では、上記を性質(*)と呼ぶことにする。

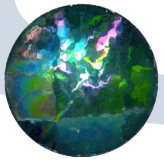


証明：数学的帰納法

- $T = \emptyset$ の時、自明
- 操作を i 回行って、次の弧を選択する直前にある枝集合 T が性質(*)を満たしているとする。次に選択された枝を $\boldsymbol{a}$ とする。
- $|T|+1$ 本の枝を持ちサーキットを有さない枝集合のうちで重みが最小のものを S とする。
  - $w(S)=w(T)+w(\boldsymbol{a})$ ならば性質(*)が成り立つ
  - $w(S)<w(T)+w(\boldsymbol{a})$ は矛盾する(起こりえない)

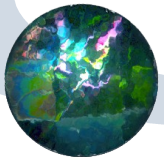


- T は$|T|$本の枝を持つサーキットの無い枝集合で重みが最小である。
  - $w(S \setminus \{a\}) \geq w(T)\ (a \in S)$
  - 従って $w(a) \leq w(\boldsymbol{a})$
- S とT はサーキットを含まず$|S|=|T|+1$
  - ある$a' \in S \setminus T$ に対して$T \cup \{a'\}$はサーキットを含まない
  - $w(a') < w(\boldsymbol{a})$は$\boldsymbol{a}$ の選び方に反する。
- よって矛盾する



注意

- ある弧を選択した際に、それが閉路を作らないことの確認が必要
  - 深さ優先、幅優先の探索アルゴリズムが必要



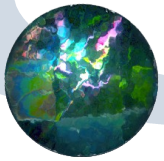
最大補木を求める方法

- 最小木を求めるために、重み最大の補木$A \setminus T$を求める

1. T=Aとする
2. Tがspanning tree ならば停止する

そうでないならば、$T \setminus \{a\}$がG のspanning treeを含み、かつ重みが最大である枝aを選ぶ

3. $T \leftarrow T \setminus \{a\}$とし2へ戻る

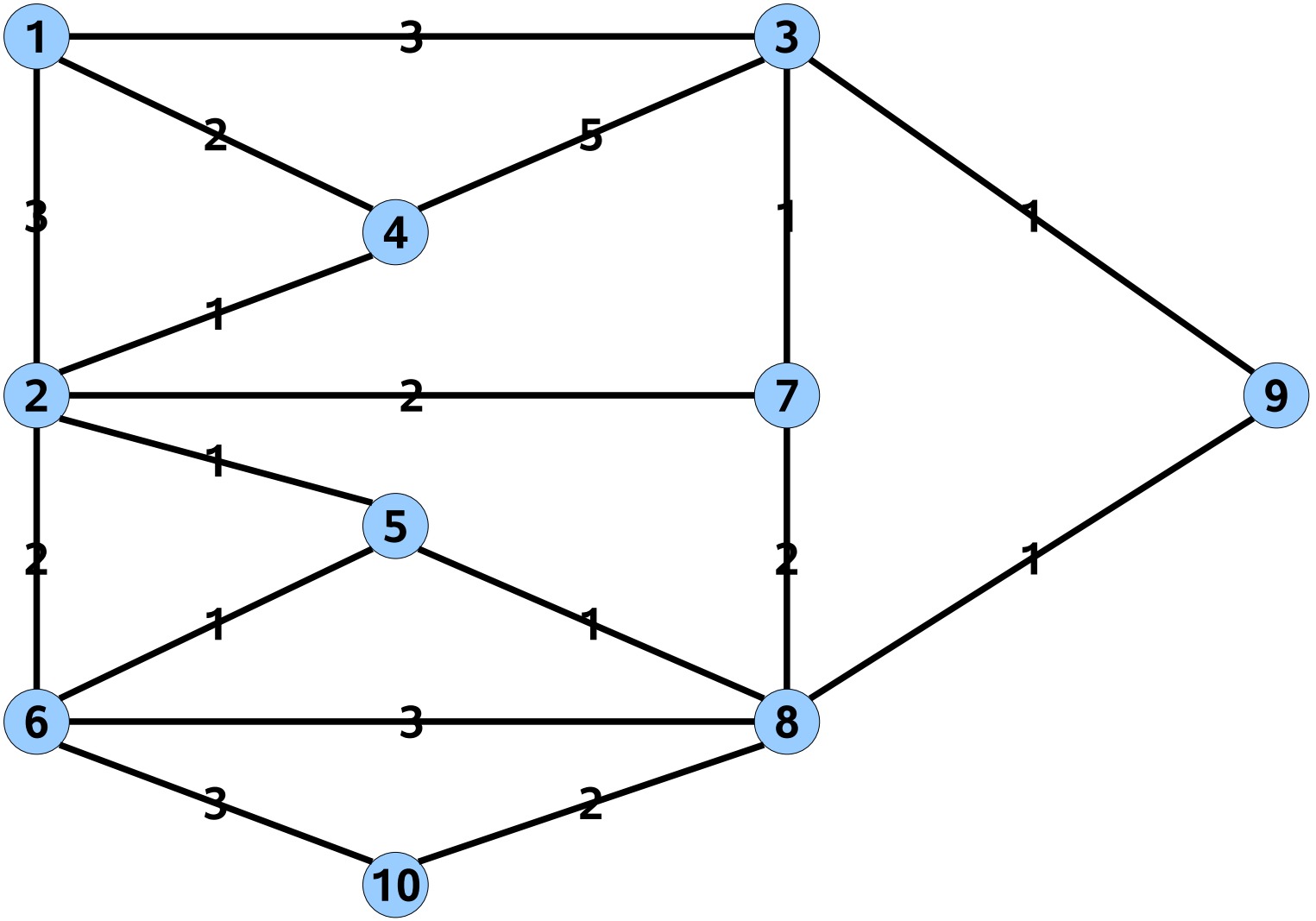


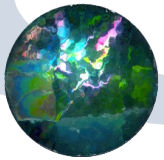
Jarník-Primアルゴリズム

1. 任意の点$v \in V$を選び、$U=\{v\}$、$T=\emptyset$とする。
2. $U=V$、つまりTがGのspanning treeならば停止

そうでないならば、Uと$V \setminus U$を結ぶ枝aのうち重み最小のものを選ぶ。aの$V \setminus U$側の端点をwとする

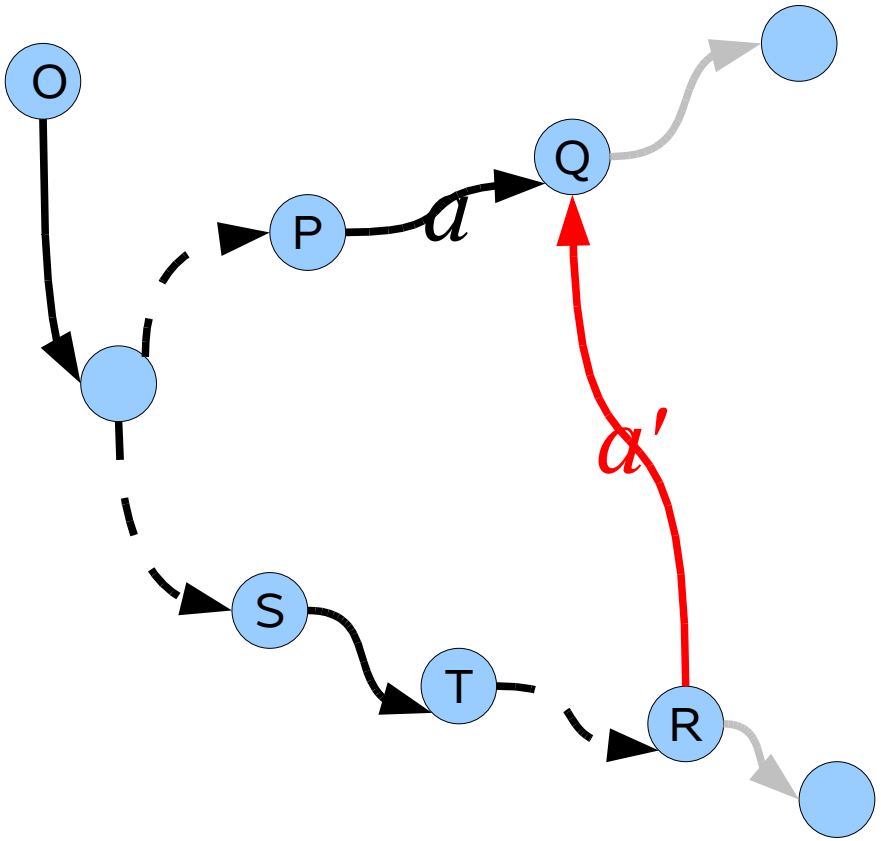
- 3. $U \leftarrow U \cup \{w\}$、$T \leftarrow T \cup \{a\}$として、2へ戻る


1
3
3
2
5
3
4
1
1
1
2
2
7
9
1
5
2
2
1
1
1
6
3
8
3
2
10

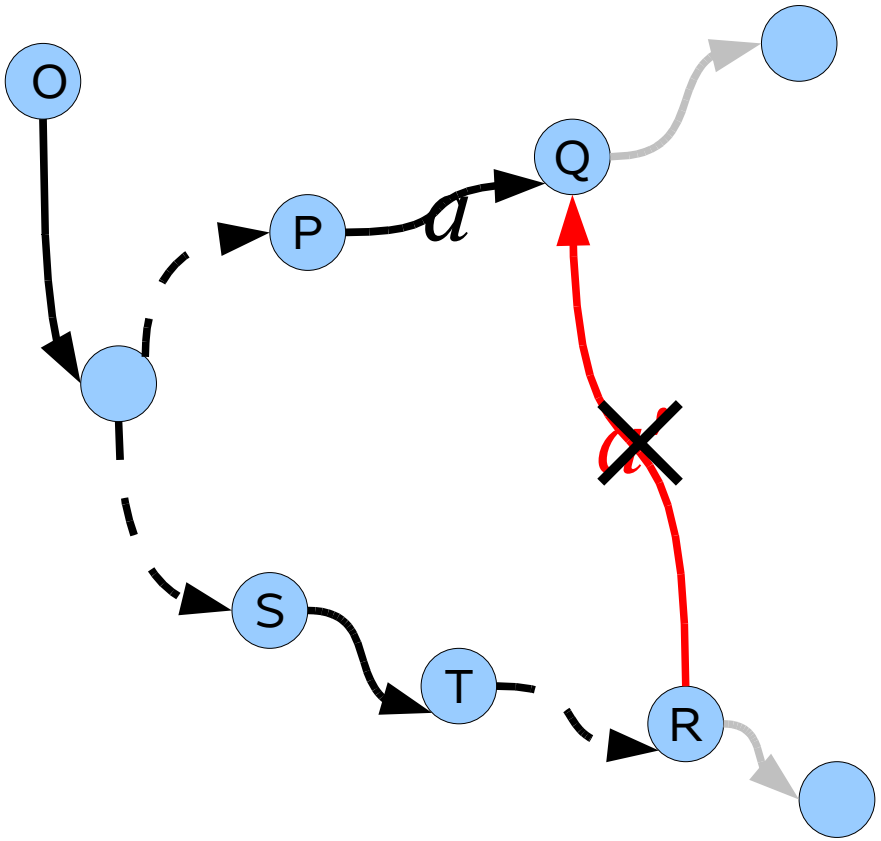


Jarník-Primアルゴリズムが正しいこと

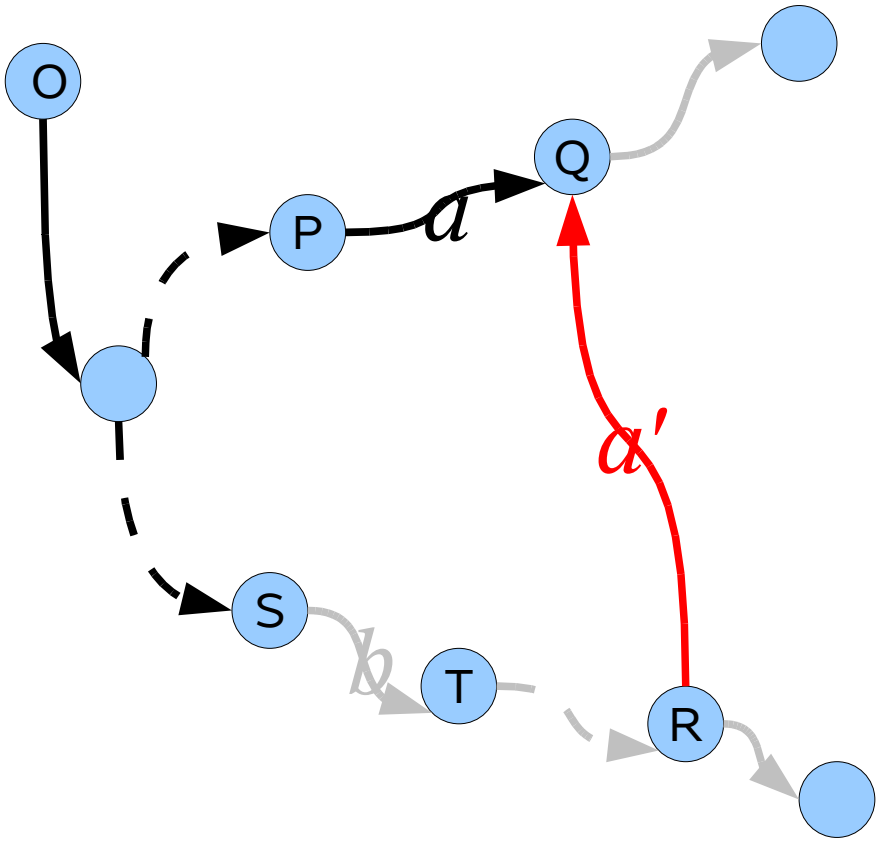
- Jarník-Primアルゴリズム実行中の木Tは、Uによって誘導されるGの部分グラフ$G(U)$における最小木になっている。
- 証明
  - Tのある枝aをTに含まれない枝a'に置き換えることで、より小さい木ができる$(w(a')<w(a))$ ことを仮定して、矛盾を導く。



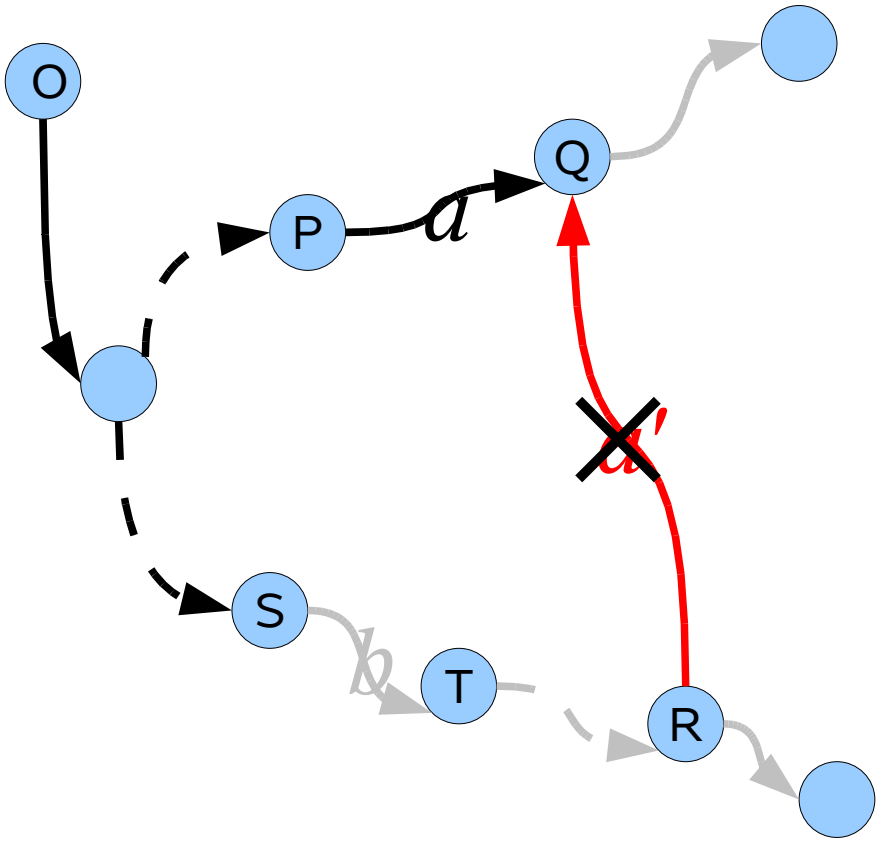
- Oを根とする木Tにおいて、弧aの代わりにa'としたほうが、重みが小さくなると仮定する。
  - $w(a')<w(a)$



- 弧aによって、Qを追加する際に、既に、点PとRが存在したとする
- 弧aによってQを接続したことは、つまり
  - $w(a) \leq w(a')$
- 仮定と矛盾



- 弧aによって、Qを追加する際に、点Pは存在していたが、もう一方では点Sまでであったと仮定



- Tよりも先にQを追加した
  - $w(a) \leq w(b)$
- 次にRを追加しなかった
  - $w(b) \leq w(a')$
- $w(a) \leq w(b) \leq w(a')$
  - 矛盾

```java
/*
 * Kruskal.java
 *
 * Created on 2007/04/30, 18:00
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package graphAnalysis;

/**
 * Kruskal Algorithm for finding minimum tree
 * @author tadaki
 */


public abstract class AbstractSearchMinimumTree {

    /**
     * graph  とtree の頂点の対応関係
     * 元のグラフ内頂点からTreeの頂点へ
     */
    protected java.util.HashMap<graphLib.Vertex,graphLib.Vertex> mapVertex=null;
    /* Tree で使用した弧の一覧 */
    protected java.util.Vector<graphLib.Arc> arcList=null;
    /* Tree で使用した頂点の一覧 */
    protected java.util.Vector<graphLib.Vertex> vertexList=null;
    protected graphLib.Network network=null;
    protected graphLib.Network tree=null;
    protected double maxWeight=0.;
    protected final boolean debug=true;
    /** Creates a new instance of Kruskal */
    public AbstractSearchMinimumTree(graphLib.Network network) {
        this.network=network;
        getMaxWeight();
    }

    /* 弧の重みの和を求める */
    protected void getMaxWeight(){
        int k=network.getNumArcs();
        double w=0.;
        for(int i=0;i<k;i++){
            graphLib.Arc aa=network.getArc(i);
            w+=aa.getWeight();
        }
        maxWeight=w;
    }

    /* 頂点対応関係の作成 */
    protected void mkVertexMap(){
        mapVertex = new java.util.HashMap<graphLib.Vertex,graphLib.Vertex>();
        tree= new graphLib.Network("Minimum Tree for "+network.toString());
        for(graphLib.Vertex v: network.getVertexes()){
            graphLib.Vertex w=new graphLib.Vertex(v.toString());
            w.setPoint(v.getPoint());
            mapVertex.put(v,w);
            tree.addVertex(w);
        }
        tree.setDirected(network.isDirected());
    }

    /* 探索実行
     * @return 探索結果のネットワーク */
    abstract public graphLib.Network search();

}
```