講義「アルゴリズムとデータ構造」

第13回　グラフとネットワークのアルゴリズム (2)

大学院情報科学研究院 情報理工学部門
情報知識ネットワーク研究室
喜田拓也

今日の内容

グラフとは （おさらい）

グラフGに対する全域木について

- 全域木（スパニング木）
- 最小全域木（最小スパニング木）

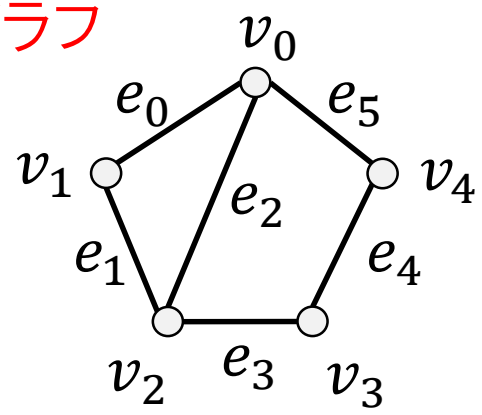
最小全域木を求めるアルゴリズムについて

- クラスカルのアルゴリズム
- クラスカルのアルゴリズムの高速化
- クラスカルのアルゴリズムが正しいことの証明
- 時間計算量が$O(m \log n)$であることの証明

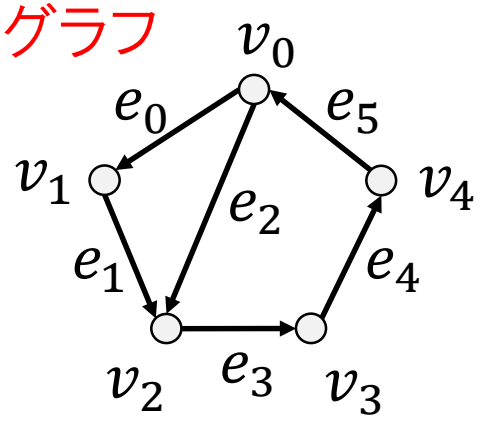
グラフとは（おさらい）

頂点(vertex)の集合Vと辺(edge)の集合$E \subseteq V \times V$の組(V, E)

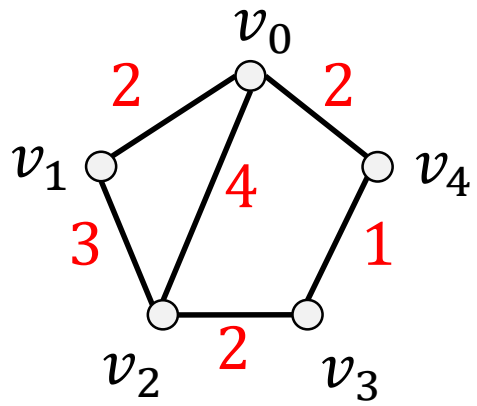
無向グラフ



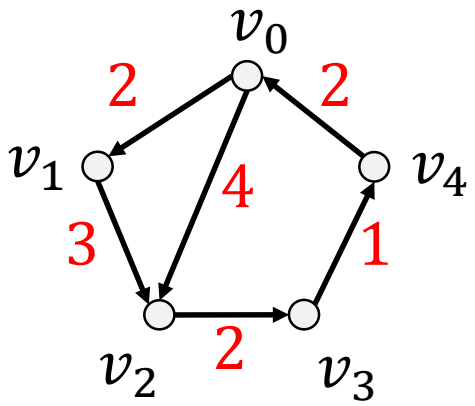
有向グラフ



無向ネットワーク

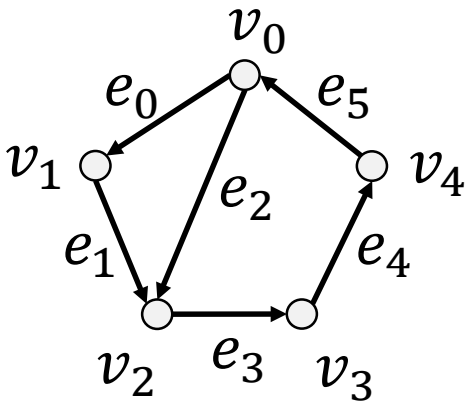


有向ネットワーク

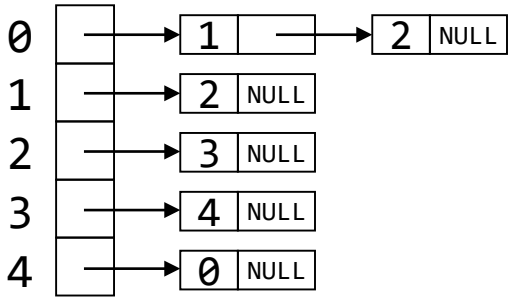


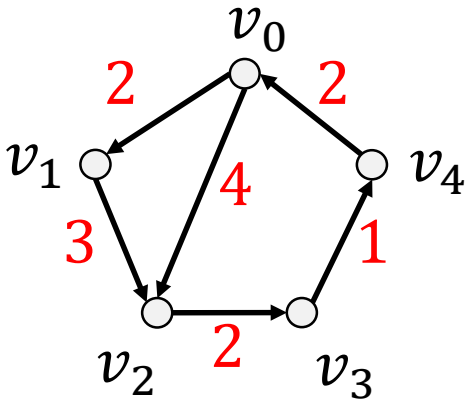
グラフの表現のしかた（おさらい）

隣接行列　　隣接リスト

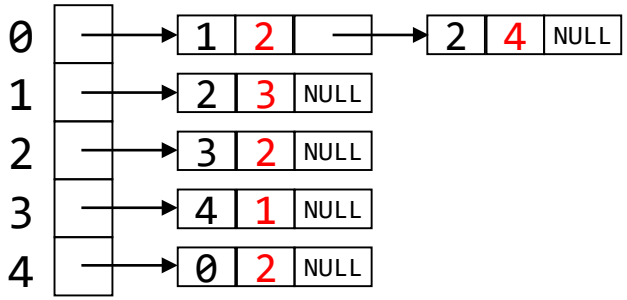


$$\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{ccccc} v_0 & v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$





$$\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{ccccc} v_0 & v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 2 & 4 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



※ 無向グラフの場合は，各辺を両方向の2辺からなる有向グラフとみなして表現する

グラフGに対する全域木(spanning tree)

定義：

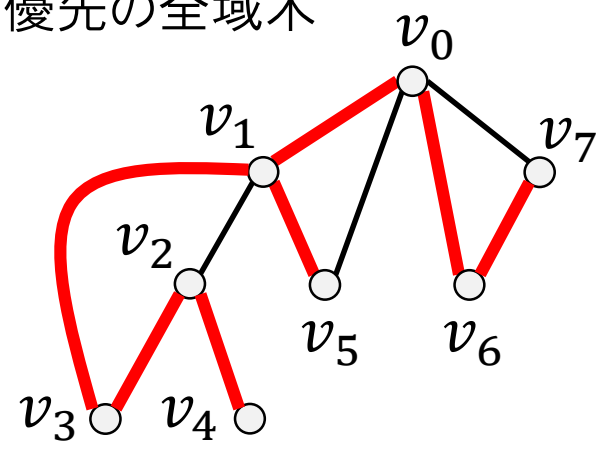
「グラフG'はグラフ$G = (V, E)$の全域木（スパニング木）である」

⇕

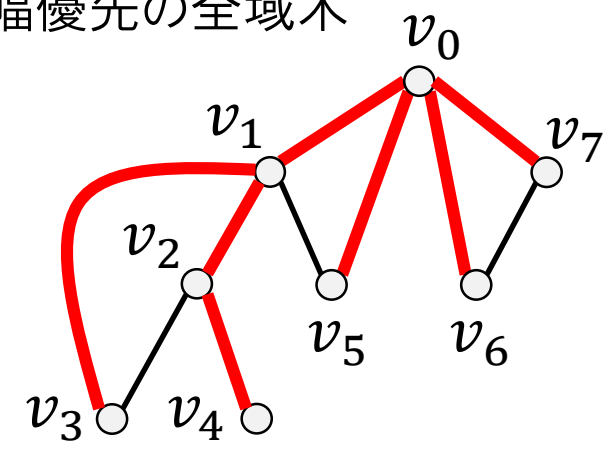
「G'はGのすべての頂点を含む部分グラフで木になっているもの」

【路(path)】 辺でつながった頂点の列
【閉路(cycle)】 自分に戻ってくるような路（長さは3頂点以上）
グラフGが**連結** ⇔ Gの任意の頂点対が路でつながっている
グラフGが**木** ⇔ Gは閉路のない連結グラフ

深さ優先の全域木



幅優先の全域木



ネットワークGの最小全域木(minimum spanning tree)

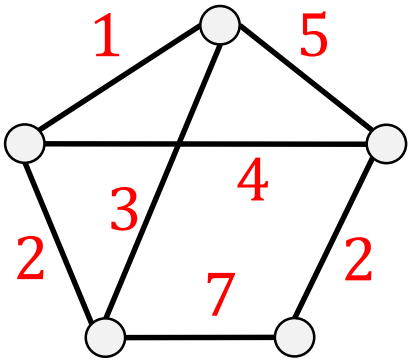
定義：

グラフG'はネットワーク（重み付きグラフ）$G = (V, E)$の
最小全域木（最小スパニング木）である

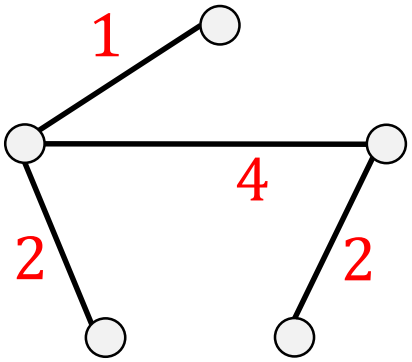
⇕

G'はGの全域木で含まれる辺の重みの総和が最小のもの

ネットワークG



Gの最小スパニング木



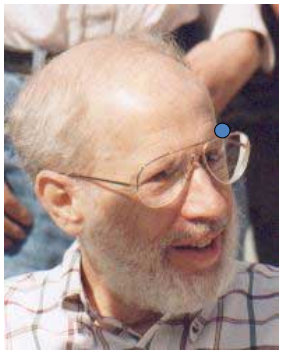
応用例として，通信網の設計，ガス・水道の配管設計などがある

最小全域木を求めるアルゴリズム

最小全域木(MST)を求める代表的なアルゴリズム

- クラスカル(Kruskal)のアルゴリズム

  1956年にJoseph Kruskal（ジョセフ・クラスカル）によって提案された．重みが小さい辺から選択し，現在の解に追加していくという方針で求める．



Joseph B. Kruskal
1928-2010 , USA

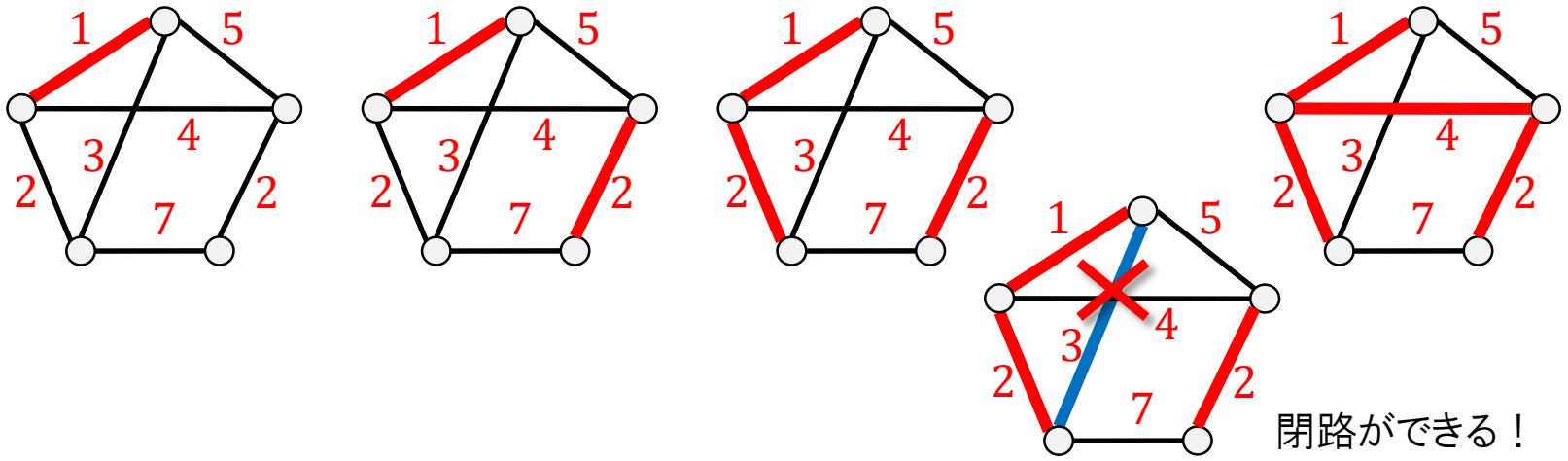
- プリム(Prim)のアルゴリズム

  1930年にVojtěch Jarník（ヤルニーク）が開発．
  それとは独立に1957年にPrimにより開発された．
  また，1959年にはDijkstraにより再発見された．
  DJP法，Jarník法，Jarník-Prim法とも呼ばれる．
  最小全域木が構成されている連結成分の範囲を徐々に広げていくという方針で求める．

Robert Clay Prim
(1921-?), USA

クラスカルのアルゴリズム

[考え方] 解候補Tを空集合から始めて，重みが小さい辺から選択し，現在の解候補Tに加えていく．ただし，選択することにより閉路ができる場合は選択しない．



クラスカルのアルゴリズムの解候補Tがもつ性質

- Tは閉路を含まない
- Tは，森になっている（必ずしも連結しているとは限らない）

※ グラフGが森　⇔　Gは閉路のないグラフ

クラスカルのアルゴリズム（疑似コード）

Procedure MST-Kruskal(G: グラフ, w: 重み)

1: $T \leftarrow \phi$;　// 解Tは最初は空

2: 辺の配列eを重みの小さい順$e[0], \ldots, e[m-1]$に整列する;

3: for $i \leftarrow 0, 1, \ldots, m-1$ do

4: 　　if ($T \cup \{e[i]\}$が閉路を含まない) then

5: 　　$T \leftarrow T \cup \{e[i]\}$;

6: 　　end if

7: end for

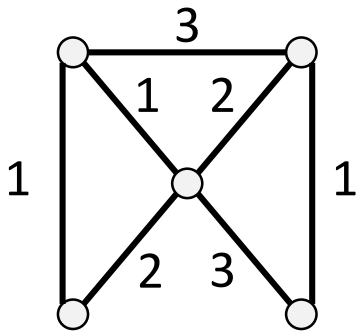
6: 最小全域木 Tを出力する;

このチェックは
意外に難しい！

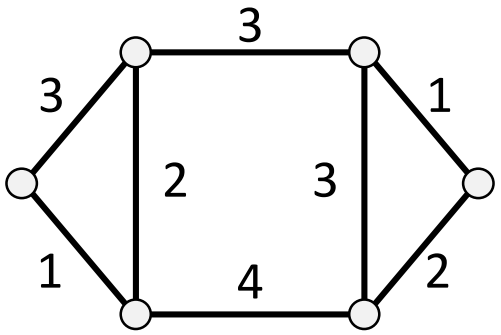
単純にやると毎回$O(n)$時間
かかる（全体では$O(mn)$）
（ヒント：$e[i]$の一方の頂点
からTを探索をする）

練習問題: 最小全域木を求めてみよう！

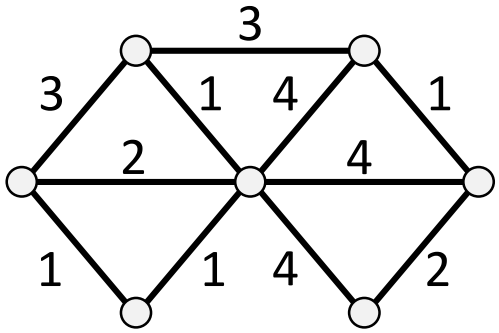
問1



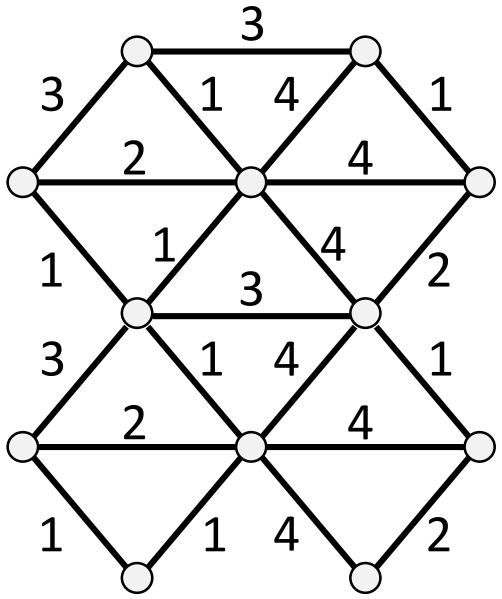
問2



問3



問4

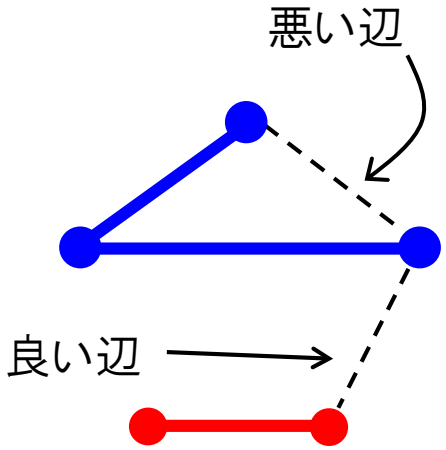


クラスカルのアルゴリズムの高速化

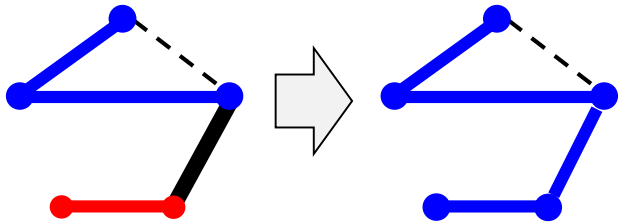
解決法

Tの連結成分毎に，頂点に異なる「色」を塗っておいて，新しく加えた辺$e[i]$の両端の色を比べることでチェックできる！

「色」のグループを表すのには，
Union-Findデータ構造を使う

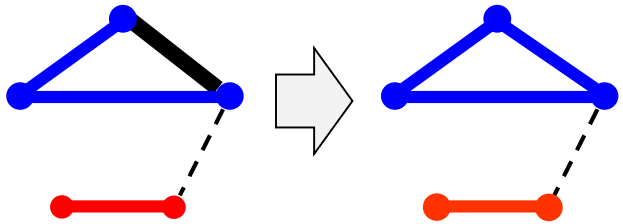


両端が異なる色のときは，辺を加えても閉路はできない



辺を加えたら色を塗り直す

両端が同じ色のときは，辺を加えると閉路ができる



単純に塗り直すと毎回$O(n)$時間かかる

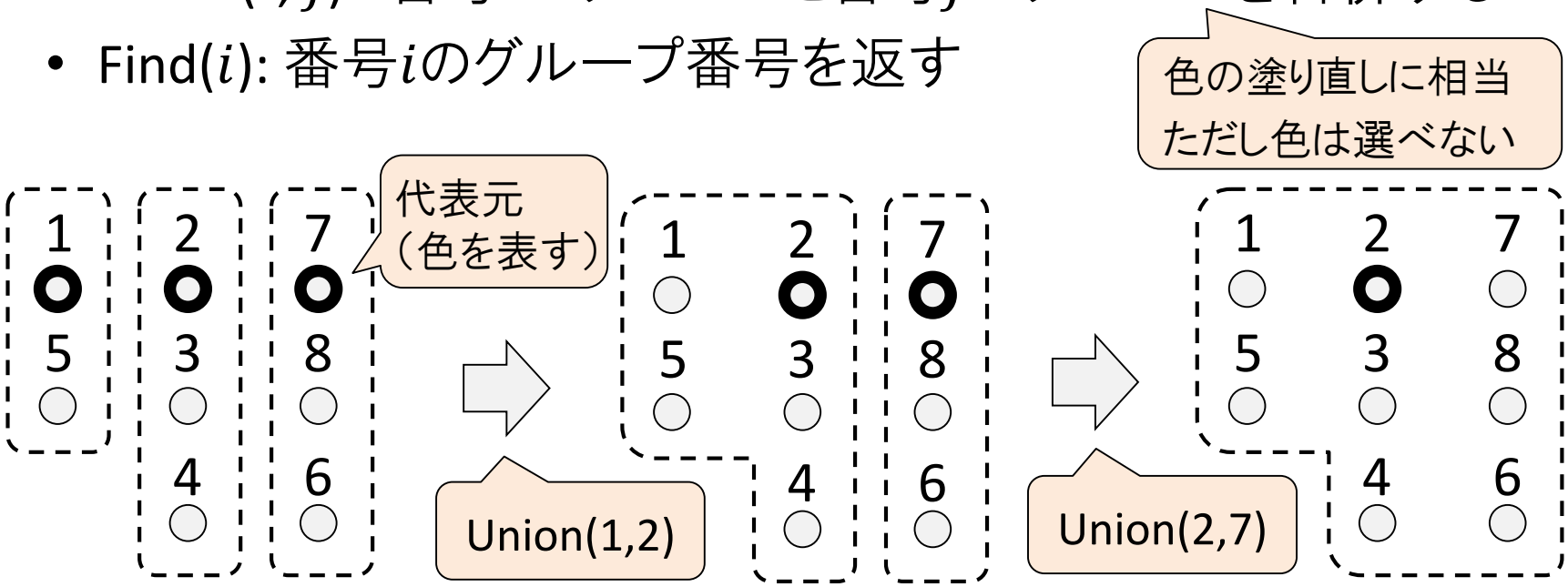
Union-Findで$O(\log n)$時間に

Union-Find データ構造

「色」(グループ番号)をもつグループ分けを管理するデータ構造

次の二つの演算を毎回$O(\log n)$時間で実行できる

- Union(i, j): 番号iのグループと番号jのグループを合併する
- Find(i): 番号iのグループ番号を返す



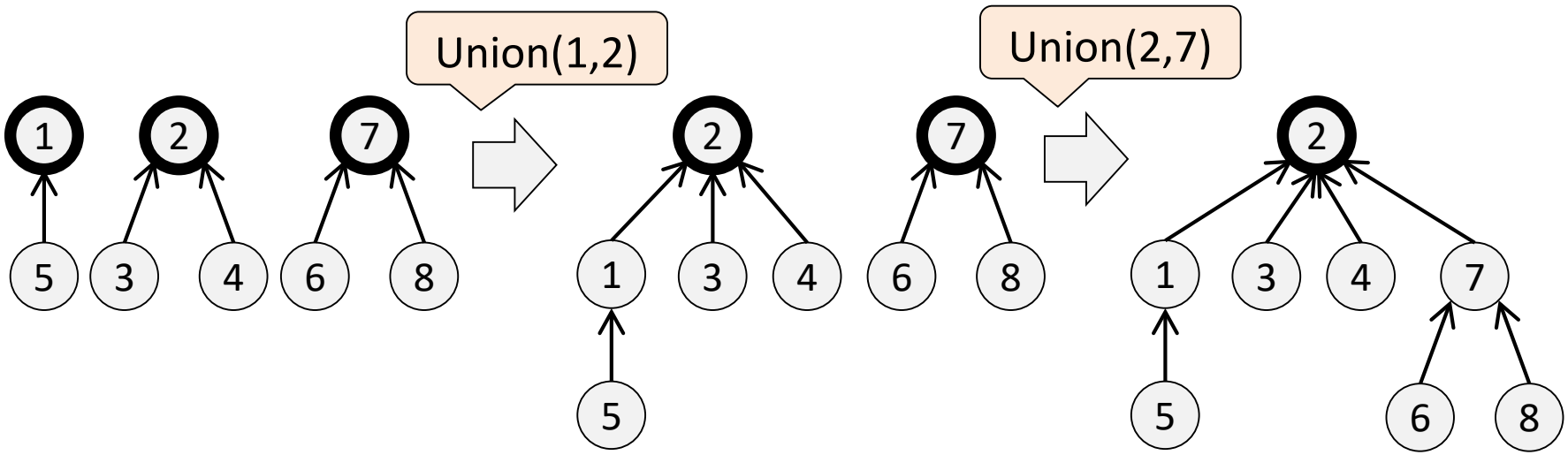
Find(1) → 1
Find(4) → 2
Find(6) → 7

Find(1) → 2
Find(4) → 2
Find(6) → 7

Find(1) → 2
Find(4) → 2
Find(6) → 2

Union-Find データ構造の実現方法

集合のグループ分けを，木のあつまり（森）で表現する
各グループは，要素番号を頂点ラベルとする一つの木に対応する
グループの色は，対応する木の根がもつ要素番号とする



Find(i)：番号iの頂点から親をたどり，根の番号を返す

Union(i,j)：番号iと番号jが属する木をマージする．ただし，高さが低い方の木の根を高い方の木の根の子とする（高さが同じ場合はどちらでもよい）

どちらも$O(\log n)$（nは要素数）

※ Find(i)の証明はそれほど自明ではない

クラスカルのアルゴリズムのベースとなる補題

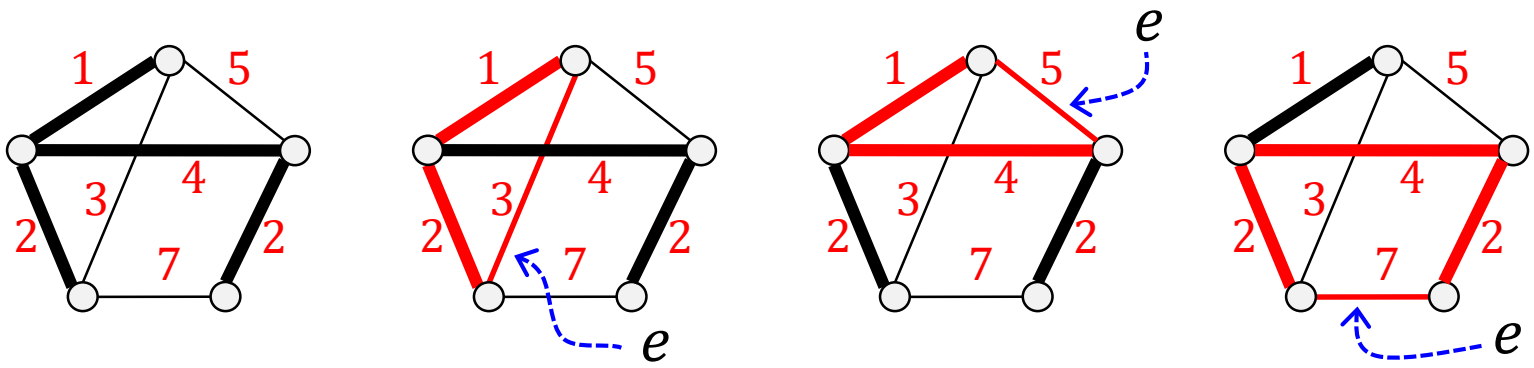
【補題】 (V, E)を連結無向グラフとし，$S \subseteq E$とする．このとき，以下の条件は，部分グラフ(V, S)が(V, E)の最小全域木であるための必要十分条件である．

条件： 部分グラフ(V, S)は(V, E)の全域木で，すべての$e \in E - S$に対して，

$$w(e) \geq w(e') \text{ for all } e' \in C_S(e)$$

$E - S$は差集合（$E \setminus S$とも書く）

ただし，$w(e)$は辺eの重みを表し，$C_S(e)$は辺eとSの辺によって作られる閉路に含まれるTの辺集合を表す．



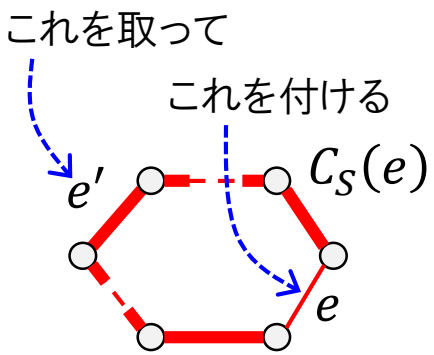
※ 太線：S，　細線：$E - S$，　赤色太線：$C_S(e)$

補題の証明(前半：⇒の証明)

「(V,S)は最小全域木」⇒「$w(e) \geq w(e')$ for all $e' \in C_S(e)$」

(V,S)が最小全域木なのに条件を満たさないと仮定する.

すると，仮定より，ある$e \in E - S$が存在し，ある$e' \in C_S(e)$に対して$w(e) < w(e')$が成り立つ.



$(V, S \cup \{e\} - \{e'\})$は$(V,E)$の全域木であり，
(V,S)よりも辺の重みの和が小さい.
これは(V,S)が最小全域木であることに矛盾する.

よって，(V,S)が最小全域木であれば必ず条件を満たす.

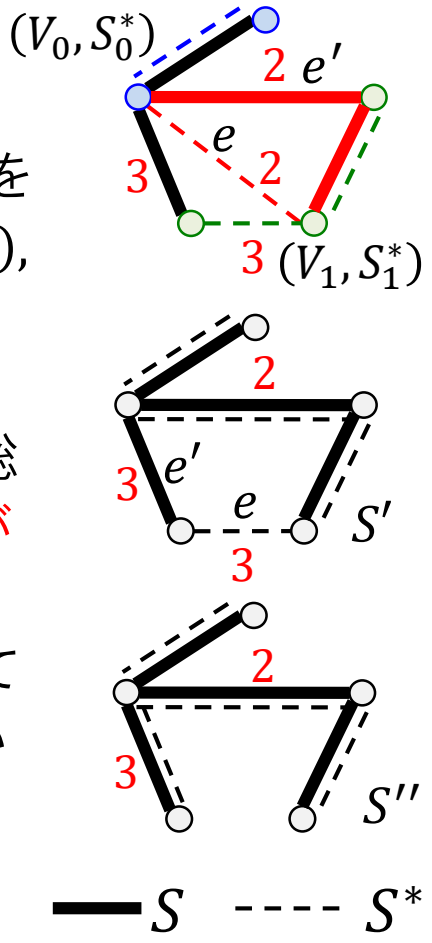
はいりほ～

補題の証明(後半:⇐の証明)

「(V,S)は最小全域木」⇐「$w(e) \geq w(e')$ for all $e' \in C_S(e)$」

最小全域木の一つを(V,S^*)とする. 条件を満たす(V,S)の辺の重みの和は (V,S^*) の辺の重みの和に等しいことを示す.

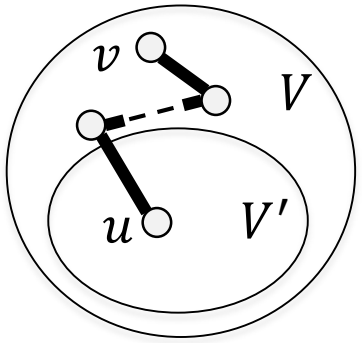
$e \in S^* - S$が存在したとする. eにより分かれる二つの連結成分を(V_0,S_0^*), (V_1,S_1^*)とすると, $e' \in C_S(e)$で二つの連結成分(V_0,S_0^*), (V_1,S_1^*)をまたぐ辺$e' \in S$が存在する.
(V,S)は条件を満たしており, $w(e) \geq w(e')$が成り立つので, $S' = S^* \cup \{e'\} - \{e\}$とすれば, S'の重みの総和はS^*の重みの総和以下になる. S^*は最小全域木なので, S'もS^*と重みの総和が等しい最小全域木であり, $|S^* - S| > |S' - S|$を満たす. ただし, $|S|$は集合Sの要素数を表すものとする. したがって, S'をS^*として同じ操作を繰り返すと$S' = S$となり, SもS^*と重みの総和が等しい最小全域木であることが示される.
【証明終わり】



クラスカルのアルゴリズムが正しいことの証明

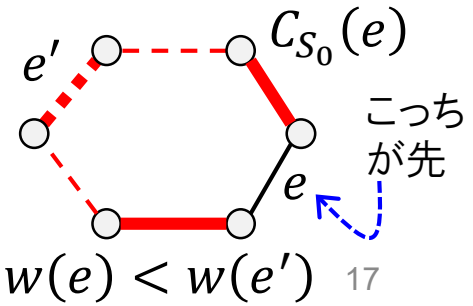
（証明） クラスカルのアルゴリズムにより求めた辺の集合をS_0とする．このとき，補題の条件が成り立っていることを示せばよい．

まず，(V, S_0)が全域木であることを示す．(V, S_0)は明らかに閉路を持たない．いま，S_0に含まれる辺の端点の集合をV'とし，$V' = V$を示す．$V' \neq V$と仮定すると，ある頂点$v \in V - V'$が存在する．(V, E)は連結であるので，ある頂点$u \in V'$が存在して，V'の頂点を経由しないでvへ到達する路が存在する．その路に属する辺をS_0に加えても閉路はできない．これはアルゴリズムの「閉路ができない限り辺を加える」という動作に矛盾する．よって$V' = V$となり，(V, S_0)は全域木であることが示された．



次に，すべての$e \in E - S_0$，すべての$e' \in C_{S_0}(e)$に対して$w(e) \geq w(e')$が成り立つことを示す．ある$e \in E - S_0$が存在して，ある$e' \in C_{S_0}(e)$に対して$w(e) < w(e')$が成り立ったと仮定する．すると，MST-Kruskalアルゴリズムの4行目において$S \cup \{e\}$が閉路を持つか否かチェックするときには，Sにはまだe'は含まれていないので，$S \cup \{e\}$は閉路を持たないことになりS_0がeを含んでいないことに矛盾する．

よって，補題の条件が成り立つ．したがって，(V, S_0)は最小全域木である．【証明終わり】



クラスカルのアルゴリズムの最悪時間計算量は$O(m \log n)$

（証明） 頂点数n，辺数mのグラフGに対して，アルゴリズムMST-Kruskalにかかる時間を考える.

1行目は明らかに$O(1)$時間でできる.

2行目の辺のソートは，m個のソートなので$O(m \log m)$であるが，$m \leq n^2$なので$O(m \log n)$.

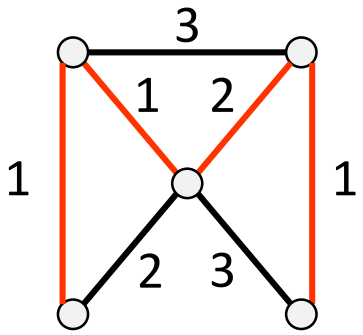
3～7行目のforループでは，各繰り返しにつき，高々$n-1$回のUnionと，高々$2m$回のFindを実行する．UnionとFindは$O(\log n)$時間でできるから，$O(n+m) \times O(\log n) = O\big((n+m)\log n\big)$時間．連結グラフで考えているので$n-1 \leq m$である．すなわち，forループ全体は$O(m \log n)$時間で実行できる.

したがって，全体の計算時間は$O(m \log n)$で実行できる.

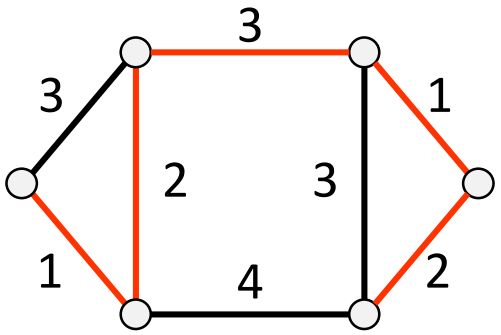
【証明終わり】

練習問題：最小全域木を求めてみよう！

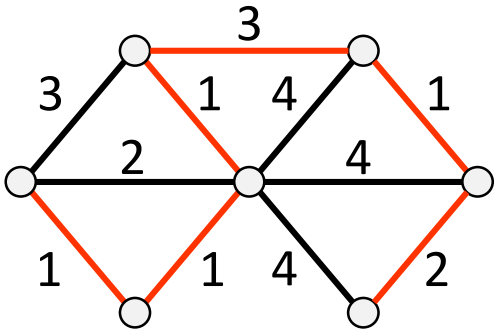
問1



問2



問3



問4