アルゴリズムとデータ構造

６．２．１節：最小木
２．５節：集合族の併合

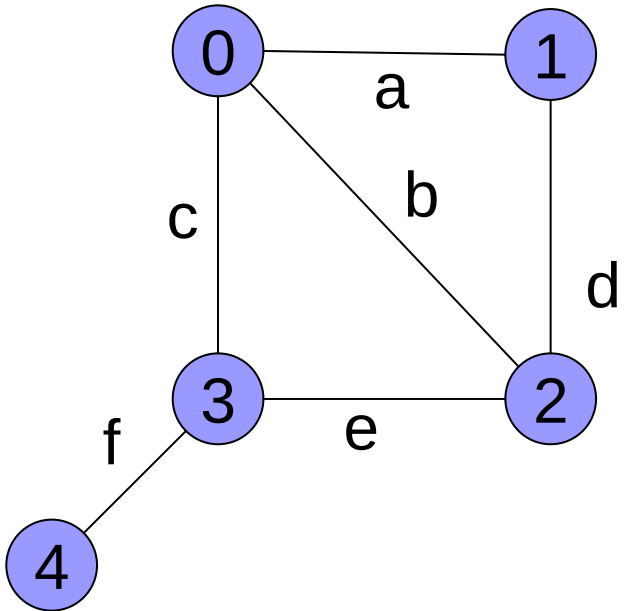
塩浦昭義
情報科学研究科　准教授
shioura@dais.is.tohoku.ac.jp

今日の講義の概要

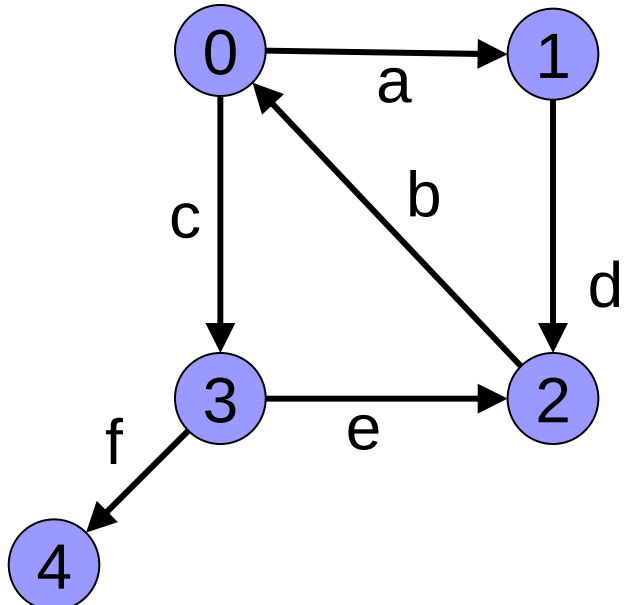
- グラフのデータ構造
- 最小木問題
- 最小木を求める2つのアルゴリズム
  - クラスカルのアルゴリズム
- アルゴリズムの計算時間の解析
  - 集合族の併合のためのデータ構造

無向グラフと有向グラフ

- 無向グラフ G=(V, E)
  - 頂点集合 V
  - 頂点の対を表す枝の集合 E
  - e=(u,v) ➔頂点 u, v は枝 e の端点



- 有向グラフ G=(V, E)
  - 頂点集合 V
  - 頂点の順序対を表す枝の集合 E
  - e=(u,v)➔頂点uは枝eの始点
    頂点vは枝eの終点



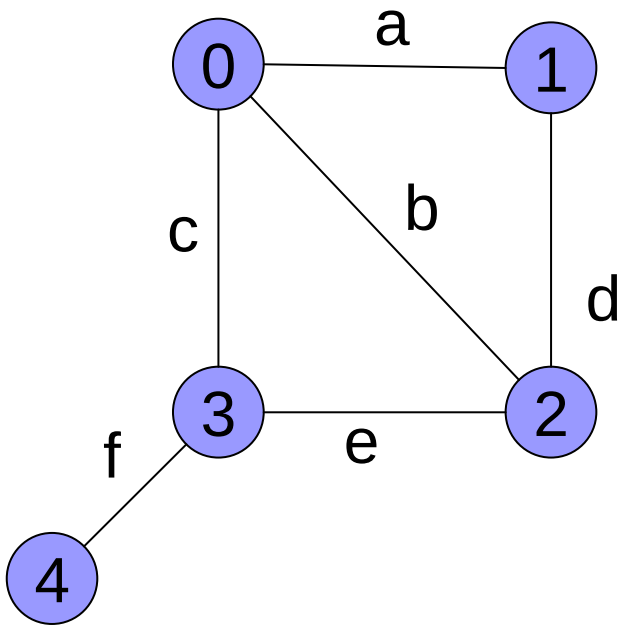
グラフのデータ構造

- グラフ G=(V, E) を表現するデータ構造
  - 接続行列 --- 領域計算量 O(mn)
  - 隣接行列--- 領域計算量 $O(n^2)$
    - 2次元配列を使う
    - 実現は簡単
    - 疎なグラフに対して無駄が多い
  - 隣接リスト--- 領域計算量 O(m+n)
    - 複数の連結リストを使う
    - 実現は少し複雑
    - どのグラフに対しても無駄がない

m: 枝の数
n: 頂点の数

無向グラフの接続行列

- 行列の行➔頂点集合，列➔枝集合　に対応
- 頂点 u は枝 e の
  - 端点である➔(u, e)の要素＝1
  - 端点ではない➔ (u, e)の要素＝0



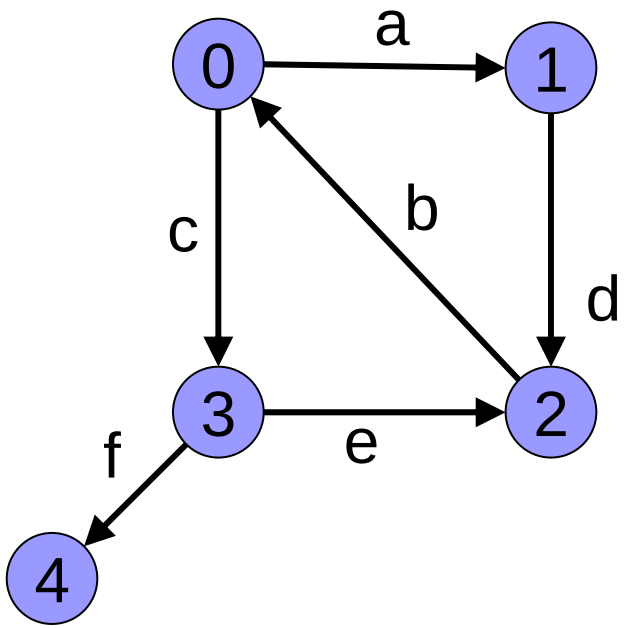
	a	b	c	d	e	f
0	1	1	1	0	0	0
1	1	0	0	1	0	0
2	0	1	0	1	1	0
3	0	0	1	0	1	1
4	0	0	0	0	0	1

行列の各列に「１」はちょうど2個

領域計算量
＝行列の大きさ
＝O(mn)

有向グラフの接続行列

- 行列の行➔頂点集合，列➔枝集合　に対応
- 頂点 u は枝 e の
  - 始点である➔(u, e)の要素＝1
  - 終点である➔(u, e)の要素＝−1
  - 端点ではない➔ (u, e)の要素＝0

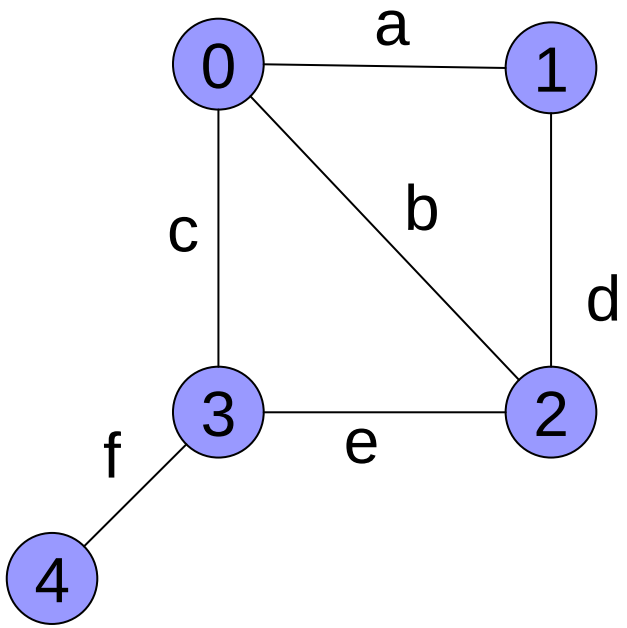


	a	b	c	d	e	f
0	1	-1	1	0	0	0
1	-1	0	0	1	0	0
2	0	1	0	-1	-1	0
3	0	0	-1	0	1	1
4	0	0	0	0	0	-1

領域計算量
＝行列の大きさ
＝O(mn)

無向グラフの隣接行列

- 行列の　行，列➔頂点集合　に対応
- 頂点 u, v の間に枝が
  - 存在している➔(u, v)の要素＝1
  - 存在していない➔ (u, v)の要素＝0



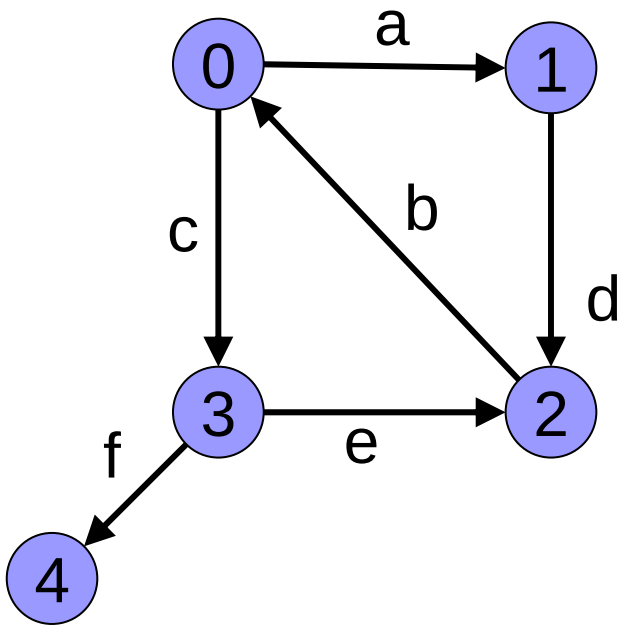
	0	1	2	3	4
0	0	1	1	1	0
1	1	0	1	0	0
2	1	1	0	1	0
3	1	0	1	0	1
4	0	0	0	1	0

「1」の数
＝2m個

領域計算量
＝行列の大きさ
＝$O(n^2)$

有向グラフの隣接行列

- 行列の　行，列➔頂点集合　に対応
- 頂点 u から v に向かう枝が
  - 存在している➔(u, v)の要素＝1
  - 存在していない➔ (u, v)の要素＝0



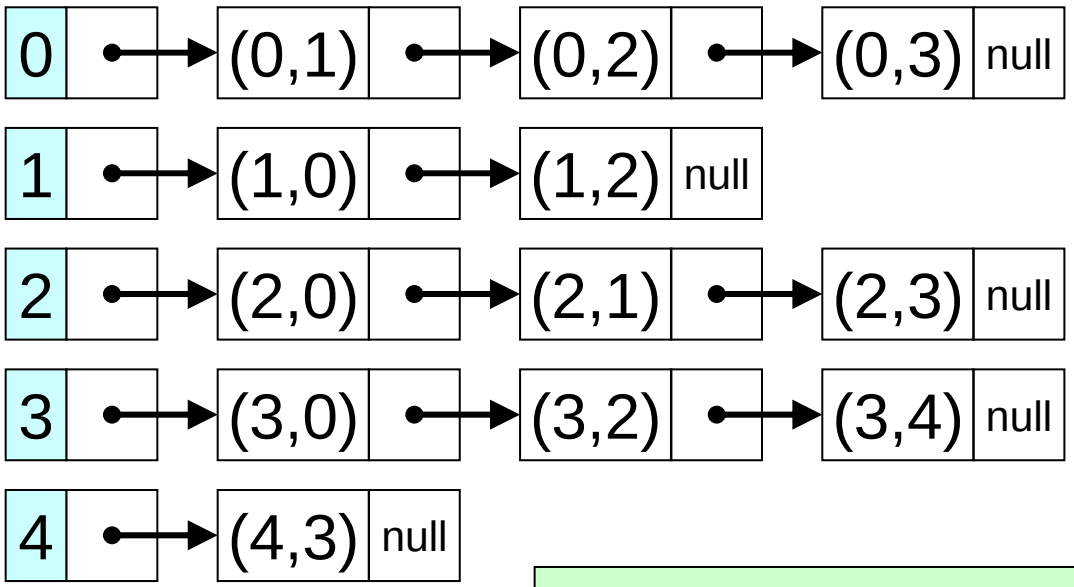
	0	1	2	3	4
0	0	1	0	1	0
1	0	0	1	0	0
2	1	0	0	0	0
3	0	0	1	0	1
4	0	0	0	0	0

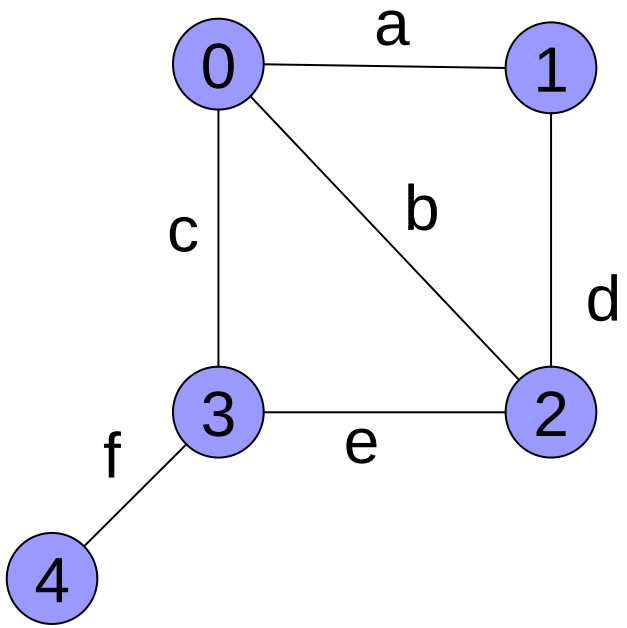
「1」の数
＝m個

領域計算量
＝行列の大きさ
＝$O(n^2)$

無向グラフの隣接リスト

- 各頂点に対し，接続する枝を連結リストで表現
  （双方向リストを使ってもよい）



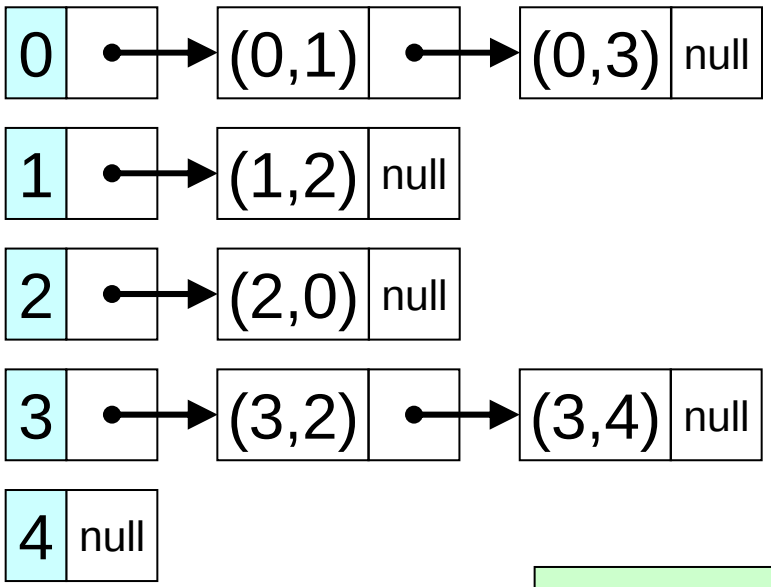


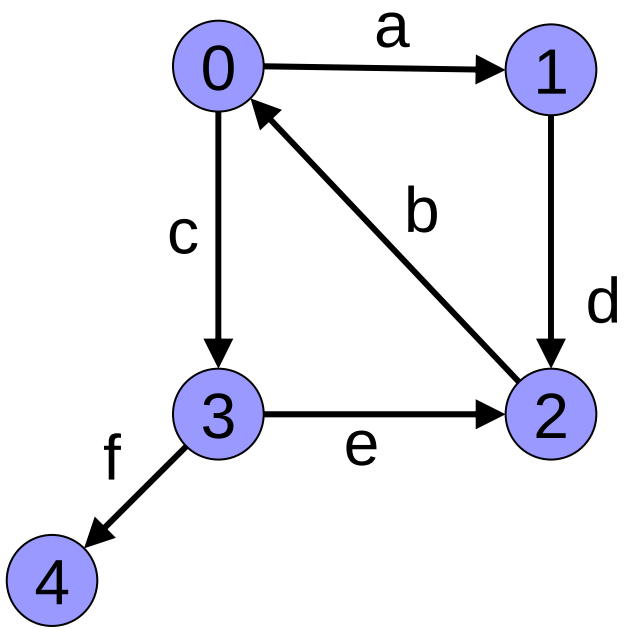
セルの数
＝2m個

領域計算量
=O（リストの数）＋O（リストのセルの数）
＝O(m+n)　※最適な領域計算量

有向グラフの隣接リスト

- 各頂点に対し，その頂点から出る枝を連結リストで表現（双方向リストを使ってもよい）



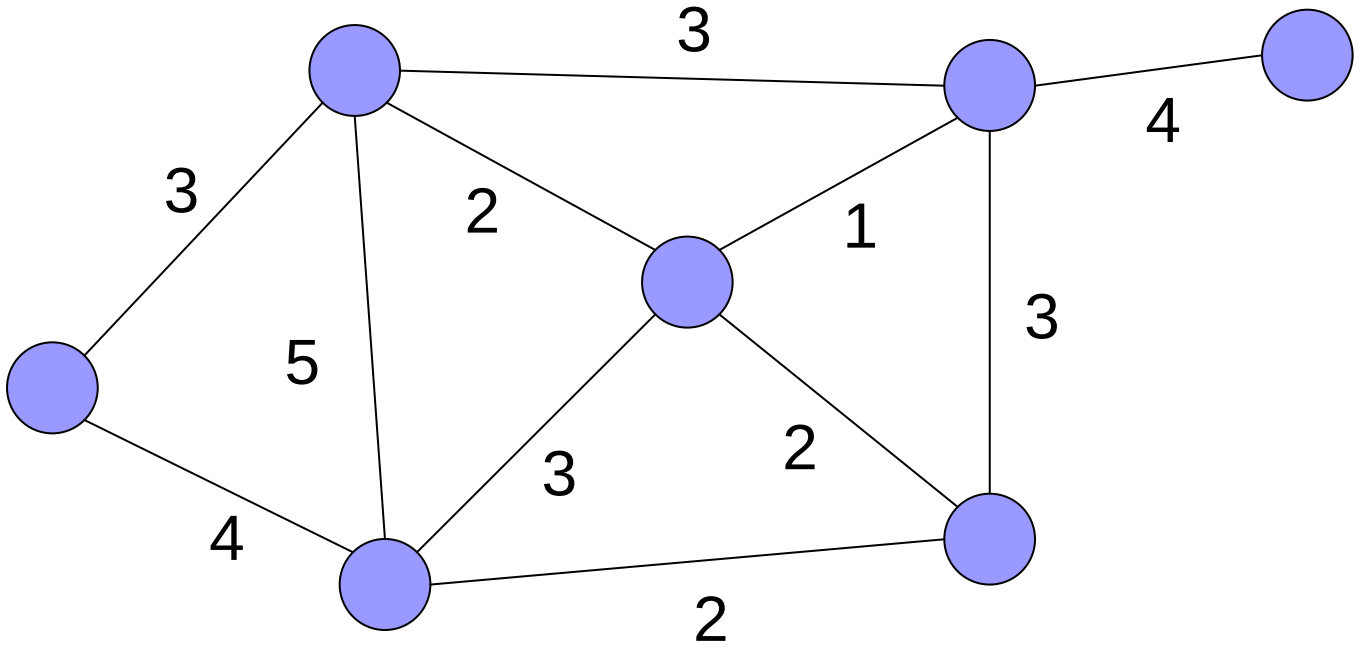


セルの数
＝m個

領域計算量
=O（リストの数）＋O（リストのセルの数）
＝O(m+n)　※最適な領域計算量

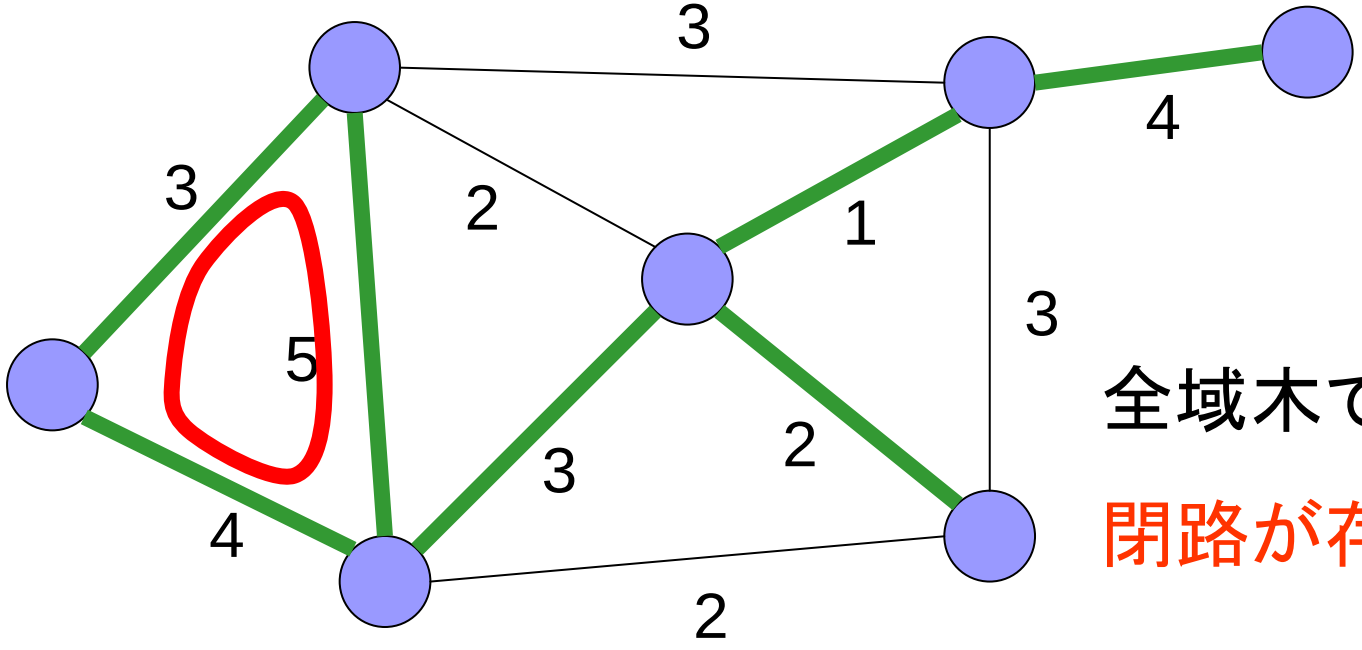
最小木問題(p.4)
minimum spanning tree problem

- 入力：無向グラフ G=(V,E)，各枝の長さ d(e) (e∈E)



最小木問題(p.4)

- 入力：無向グラフG=(V,E)，各枝の長さd(e) (e∈E)
- 出力：Gの最小木（Gの全域木で，枝の長さの和が最小のもの）

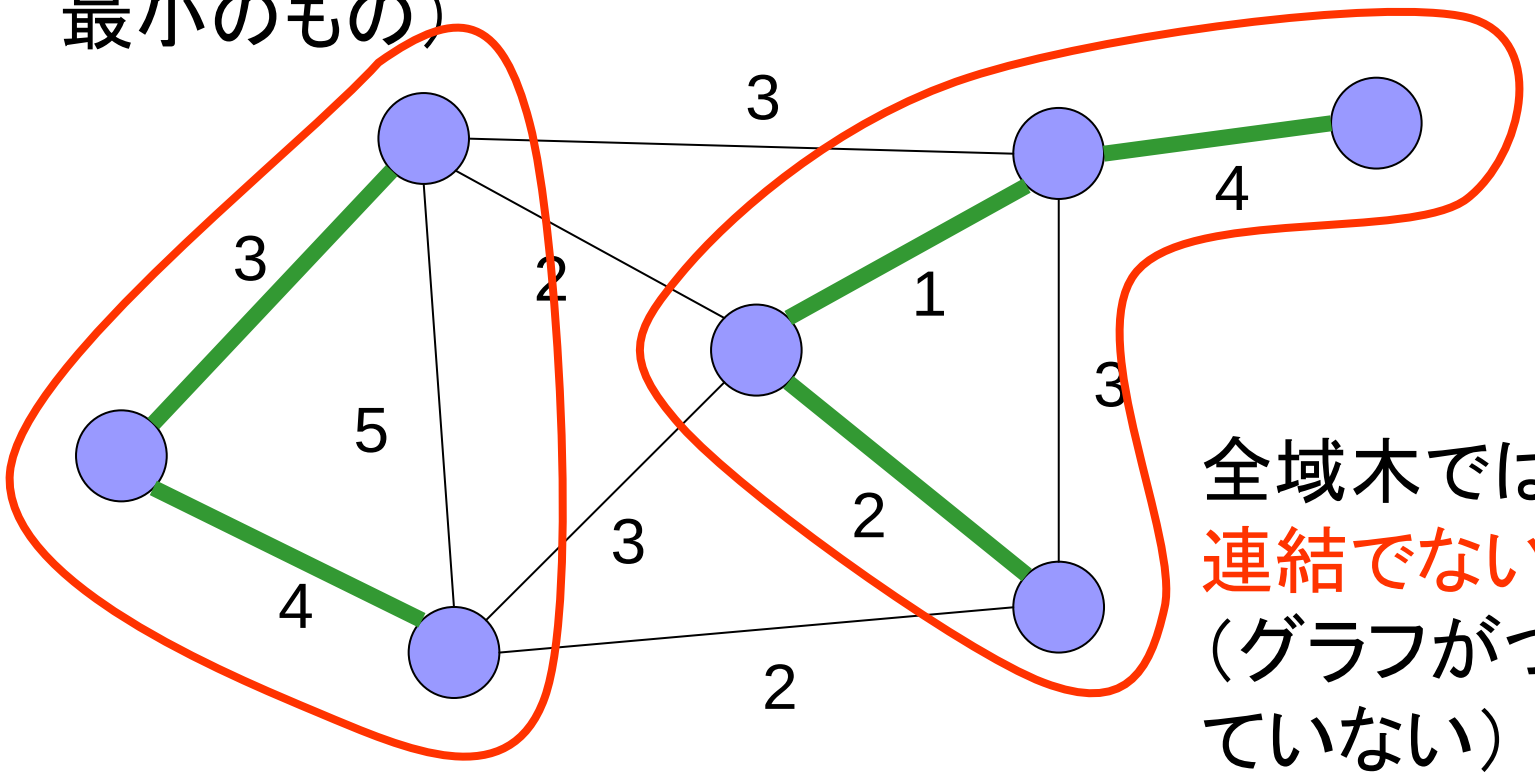


全域木ではない！

閉路が存在

最小木問題(p.4)

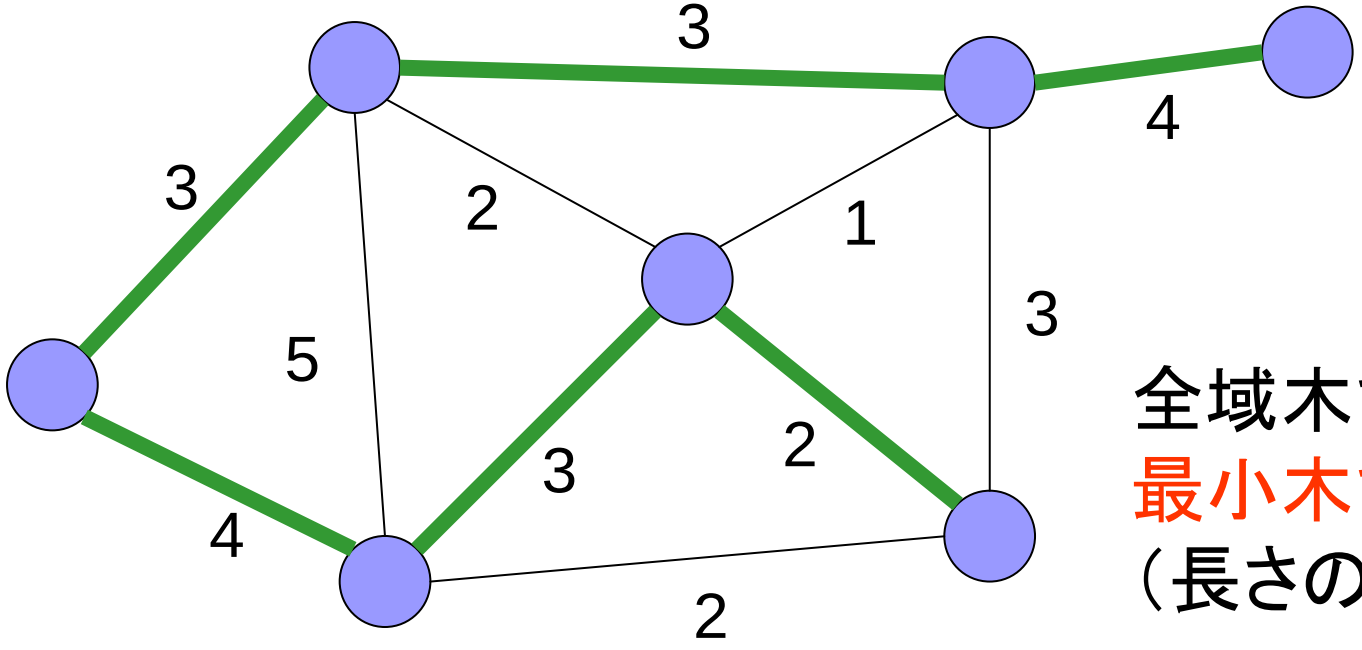
- 入力：無向グラフG=(V,E)，各枝の長さd(e) (e∈E)
- 出力：Gの最小木（Gの全域木で，枝の長さの和が最小のもの）



全域木ではない！
連結でない
（グラフがつながっていない）

最小木問題(p.4)

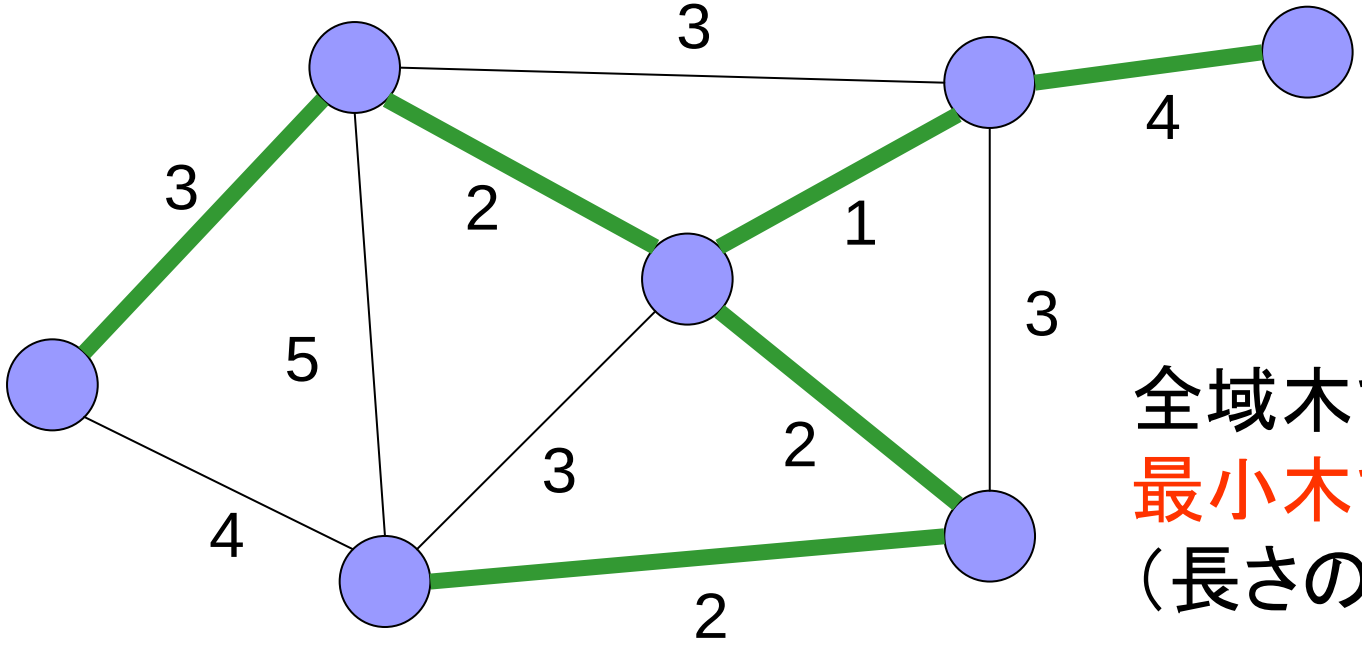
- 入力：無向グラフG=(V,E)，各枝の長さd(e) (e∈E)
- 出力：Gの最小木（Gの全域木で，枝の長さの和が最小のもの）



全域木であるが，
最小木でない
（長さの和＝19）

最小木問題(p.4)

- 入力：無向グラフG=(V,E)，各枝の長さd(e) (e∈E)
- 出力：Gの最小木（Gの全域木で，枝の長さの和が最小のもの）



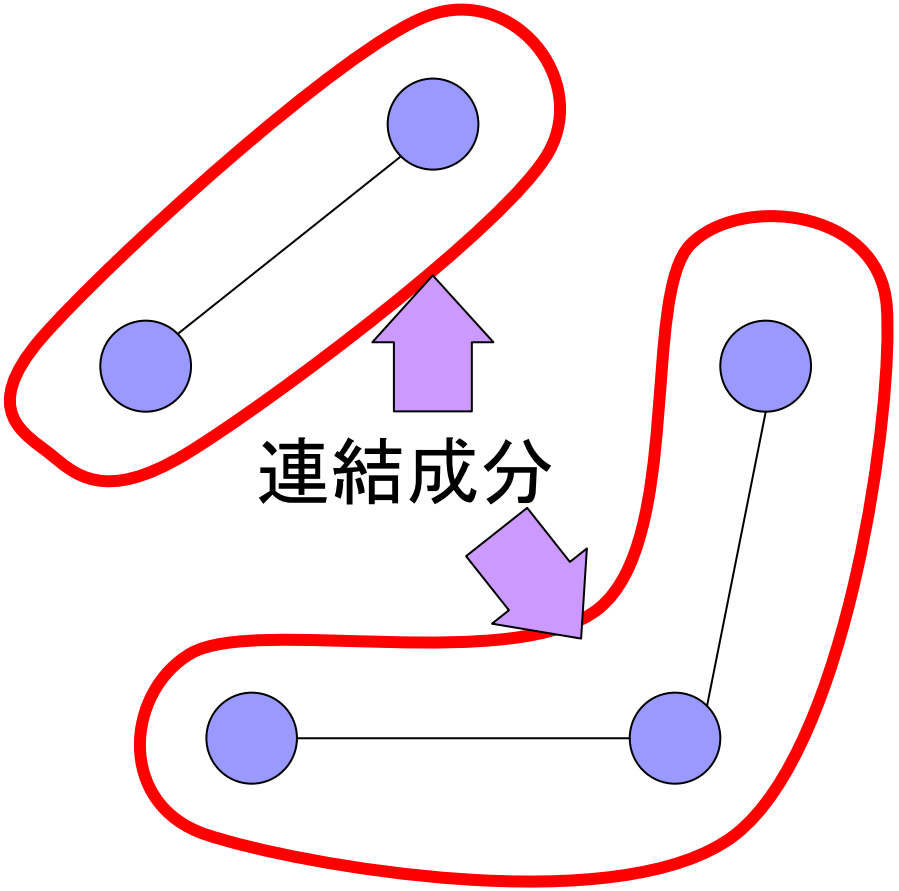
全域木であり，
最小木である
（長さの和＝14）

最小木を求めるアルゴリズム

- クラスカルのアルゴリズム
  - 長さの短い順に枝を加える
  - 閉路が出来ないようにするため，同じ連結成分を結ぶ枝は除外
- プリムのアルゴリズム（次回の講義）
  - 適当な頂点s を決める
  - 頂点 s を含む連結成分 P と，P に含まれない頂点を結ぶ枝の中で長さ最小のものを繰り返し加える

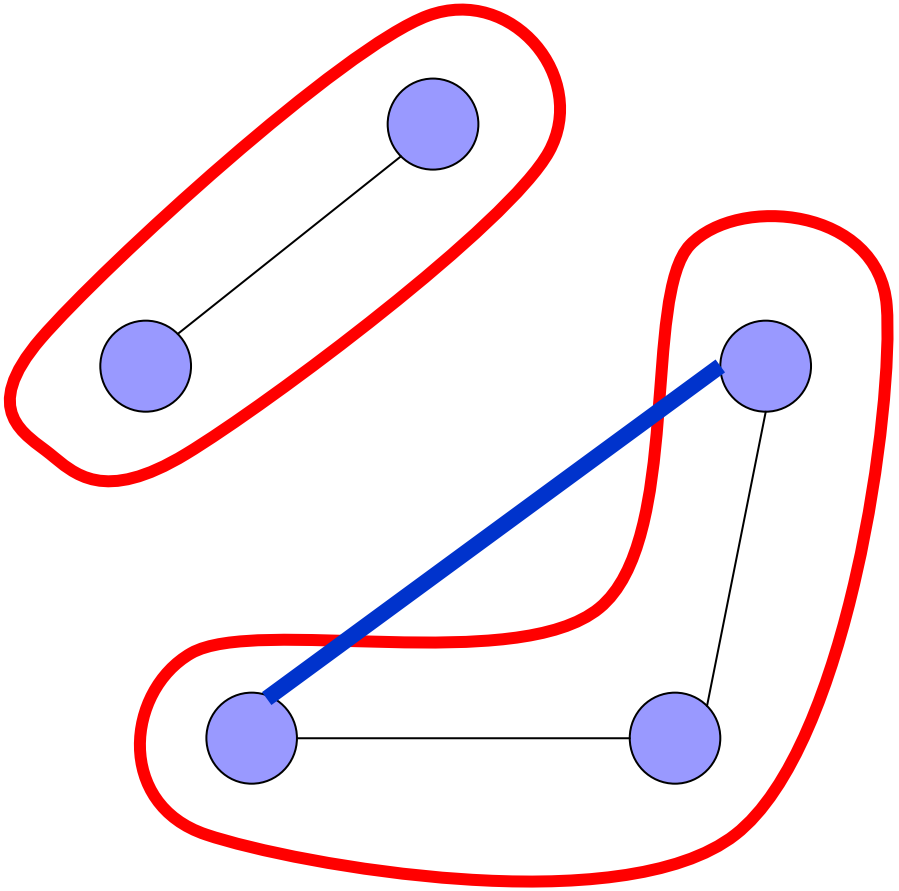
グラフの連結成分

グラフの連結成分＝枝で結ばれている頂点の集合



グラフの連結成分

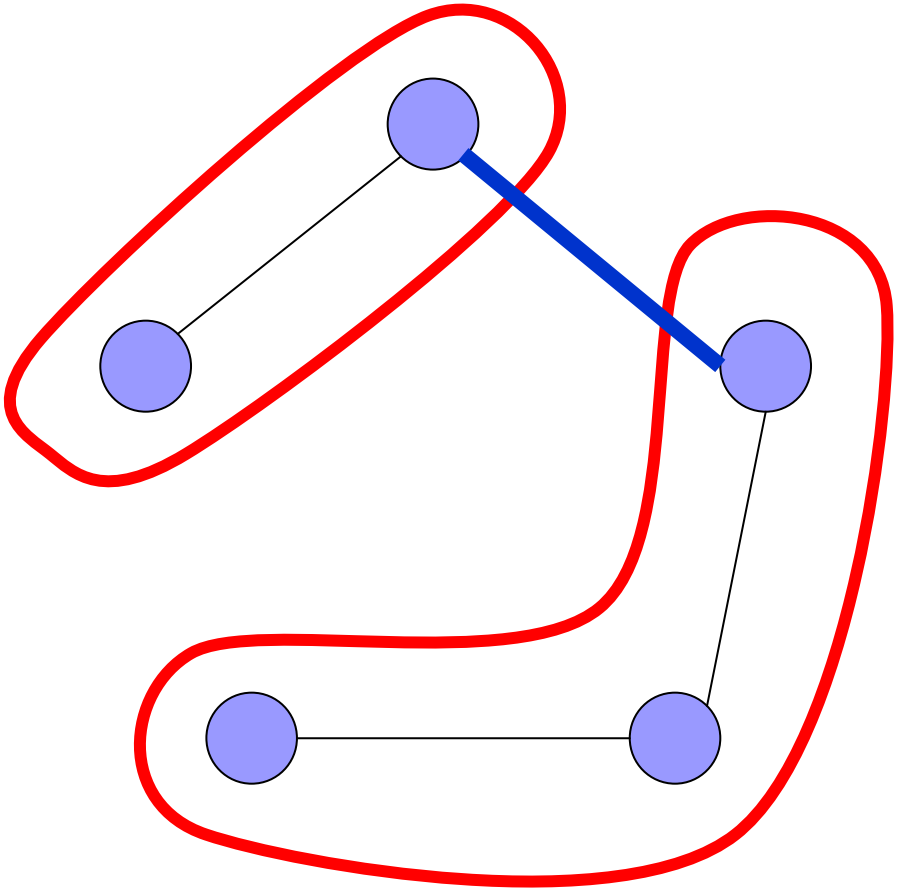
グラフの連結成分＝枝で結ばれている頂点の集合



同じ連結成分に
含まれる頂点を枝で結ぶ
➔閉路が出来る

グラフの連結成分

グラフの連結成分＝枝で結ばれている頂点の集合



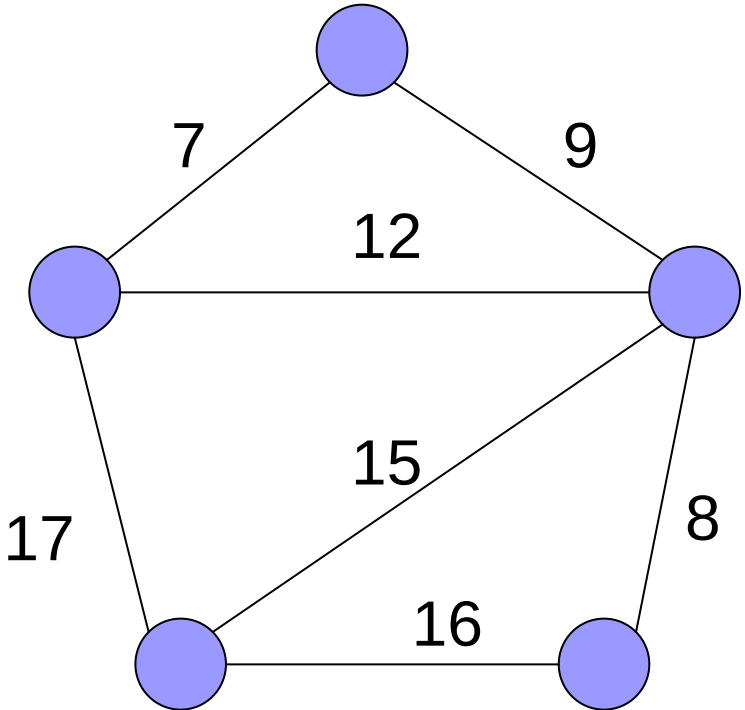
異なる連結成分に
含まれる頂点を枝で結ぶ
➔閉路は出来ない

異なる連結成分を結ぶ枝
を繰り返し加えていく
➔ 全域木が得られる

クラスカルのアルゴリズム(p.11)

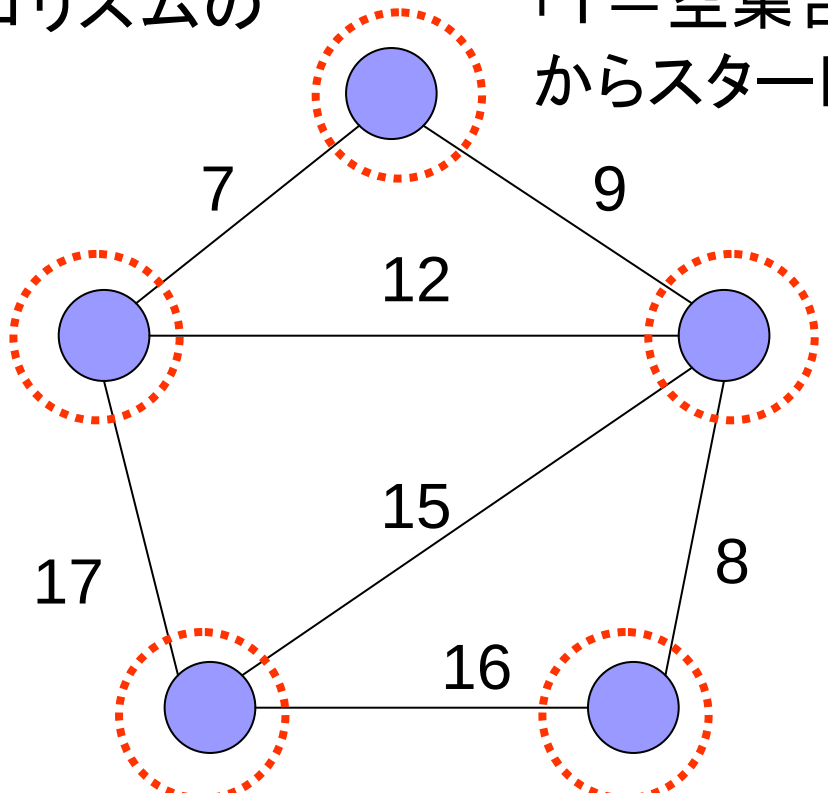
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

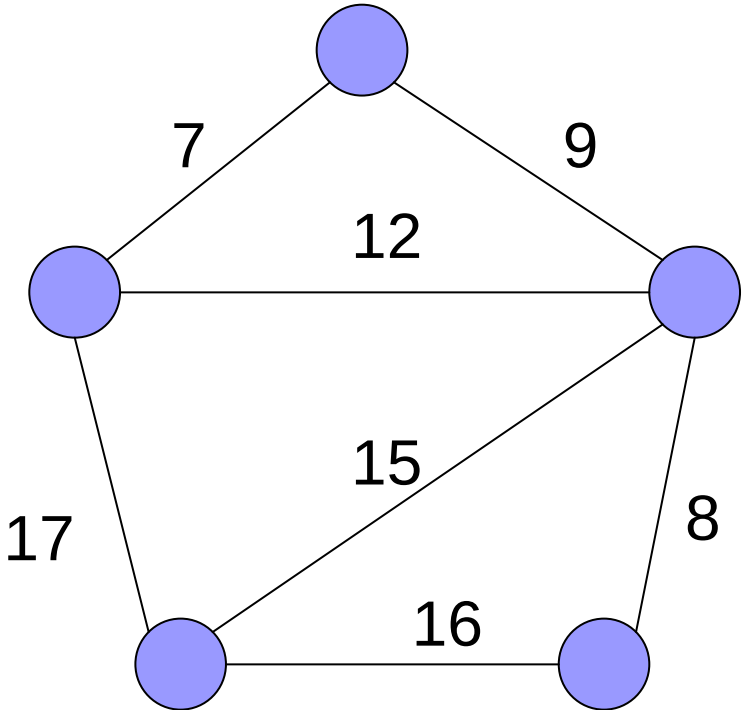
「T＝空集合」からスタート



クラスカルのアルゴリズム(p.11)

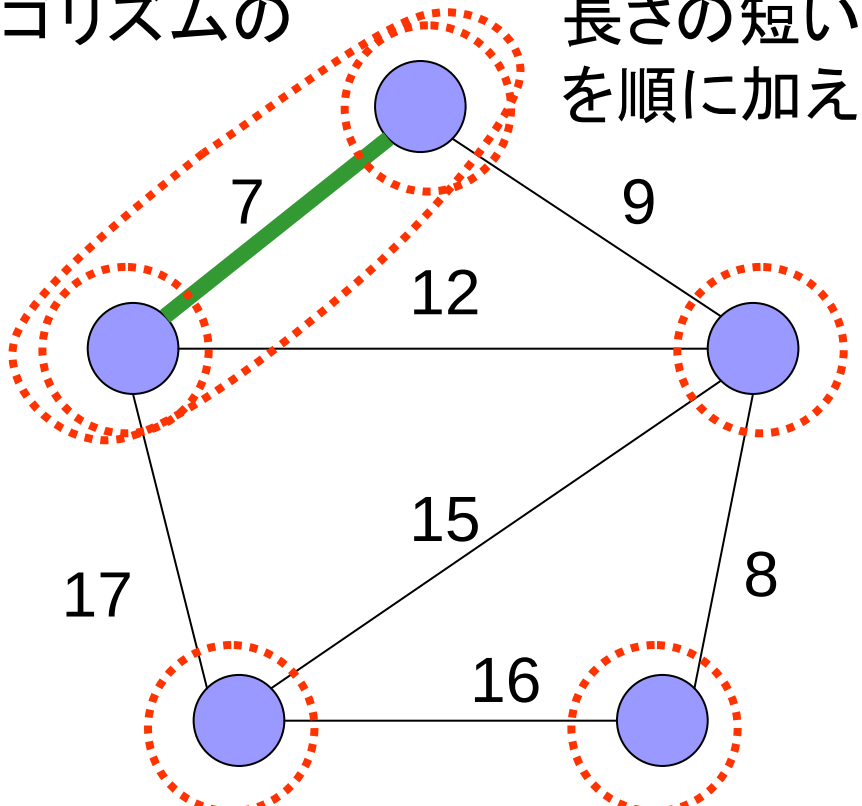
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

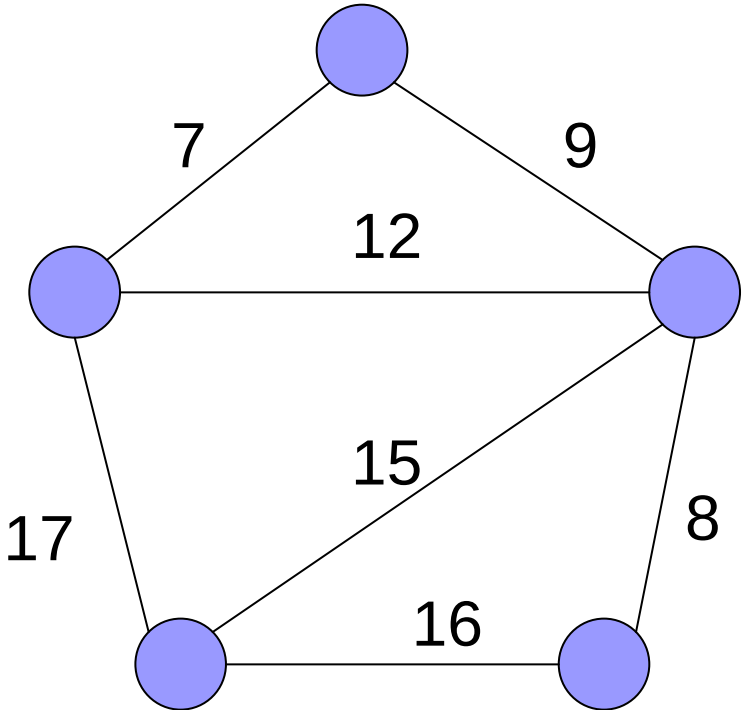
長さの短い枝を順に加える



クラスカルのアルゴリズム(p.11)

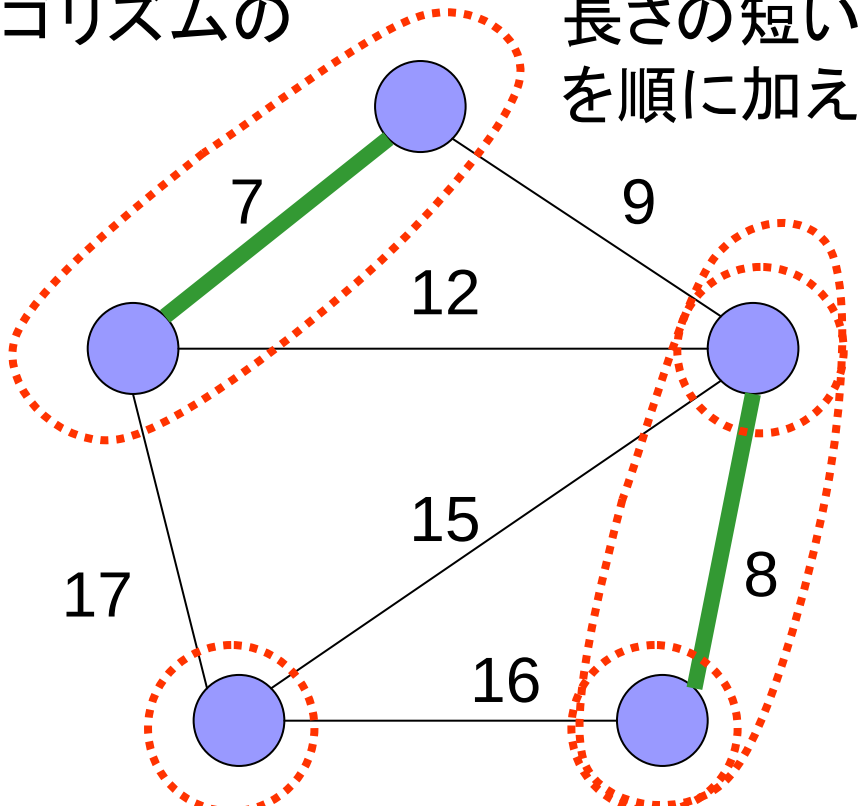
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

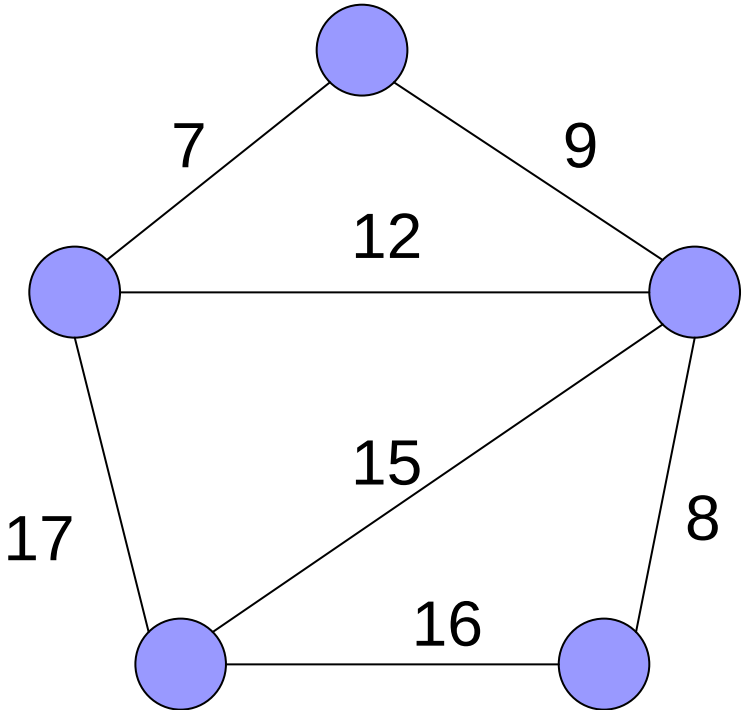
長さの短い枝を順に加える



クラスカルのアルゴリズム(p.11)

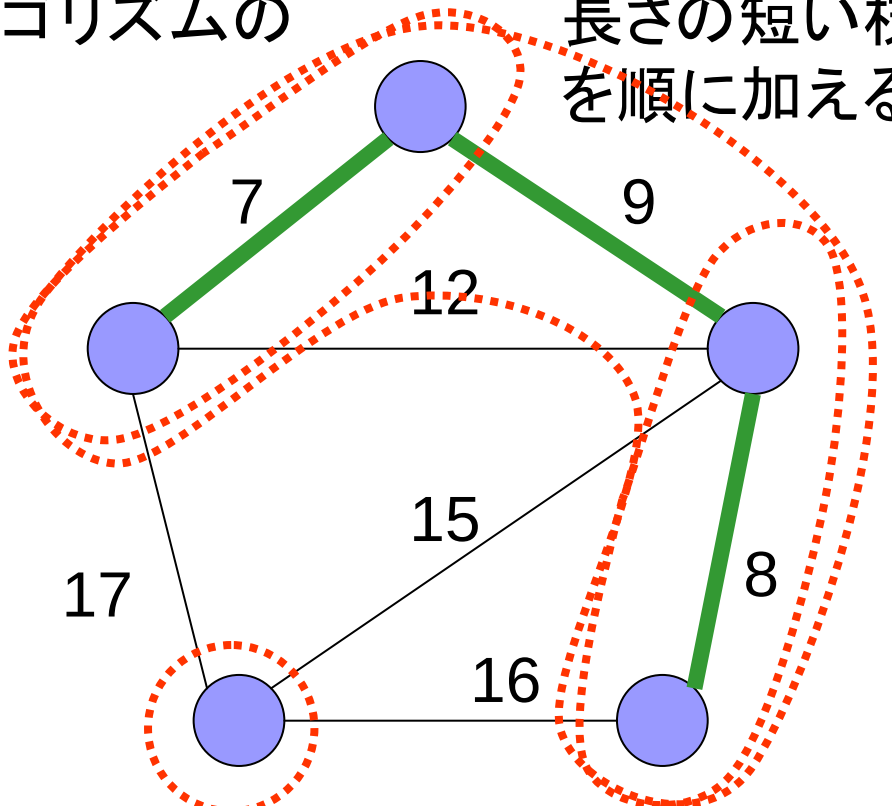
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

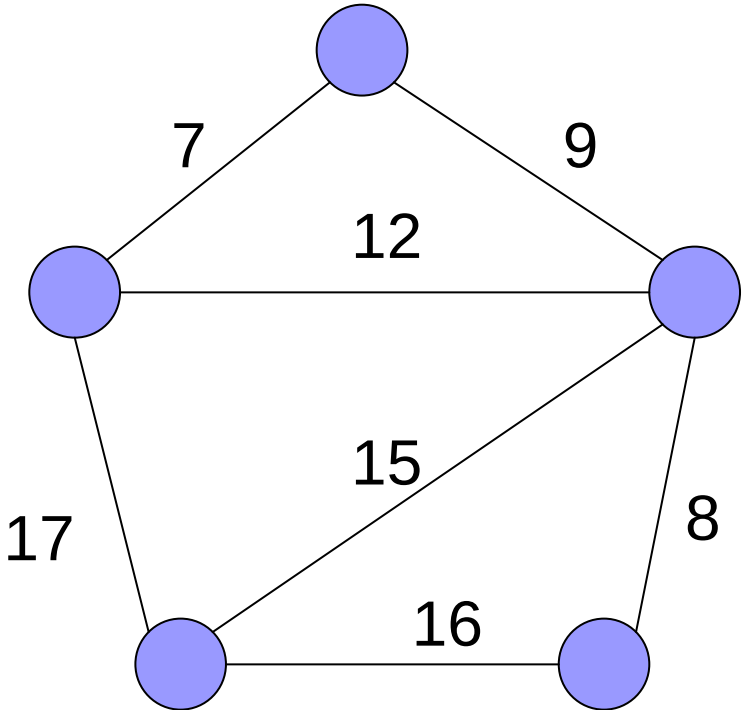
長さの短い枝を順に加える



クラスカルのアルゴリズム(p.11)

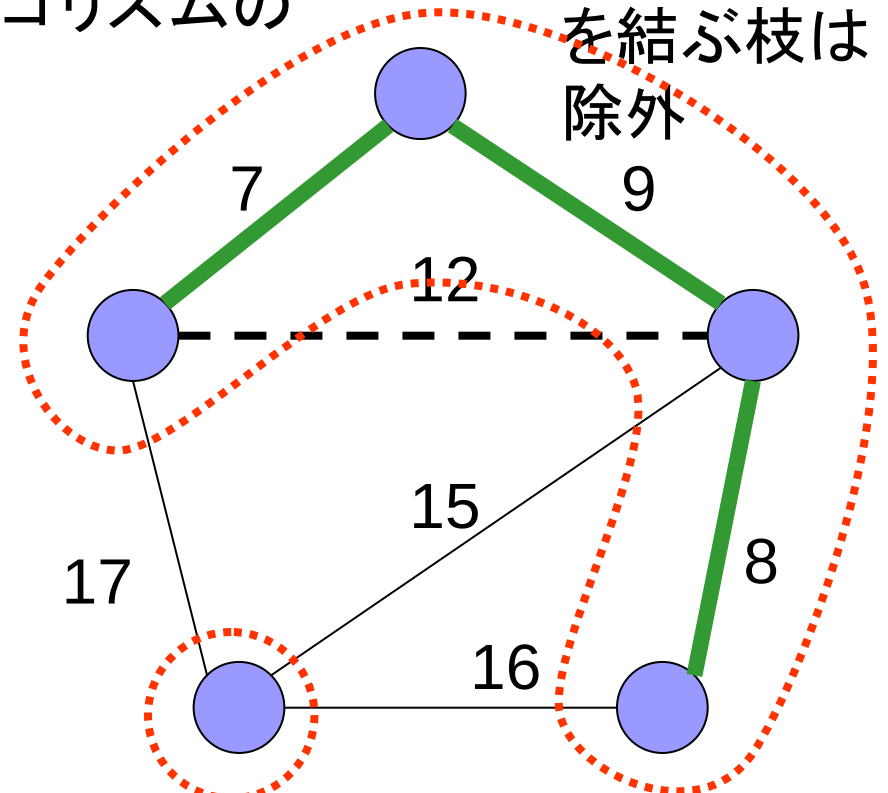
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

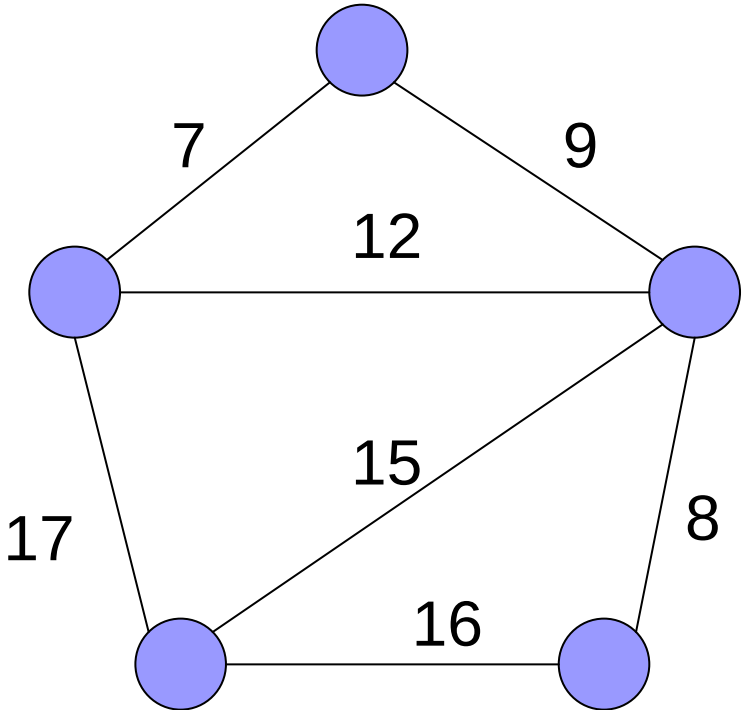
同じ連結成分を結ぶ枝は除外



クラスカルのアルゴリズム(p.11)

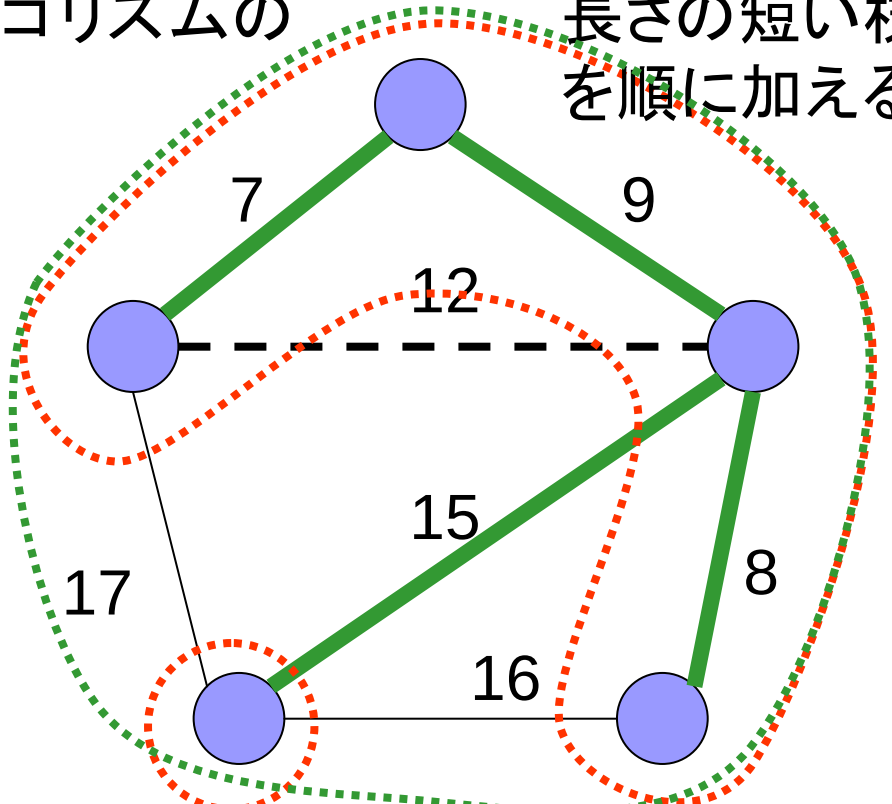
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

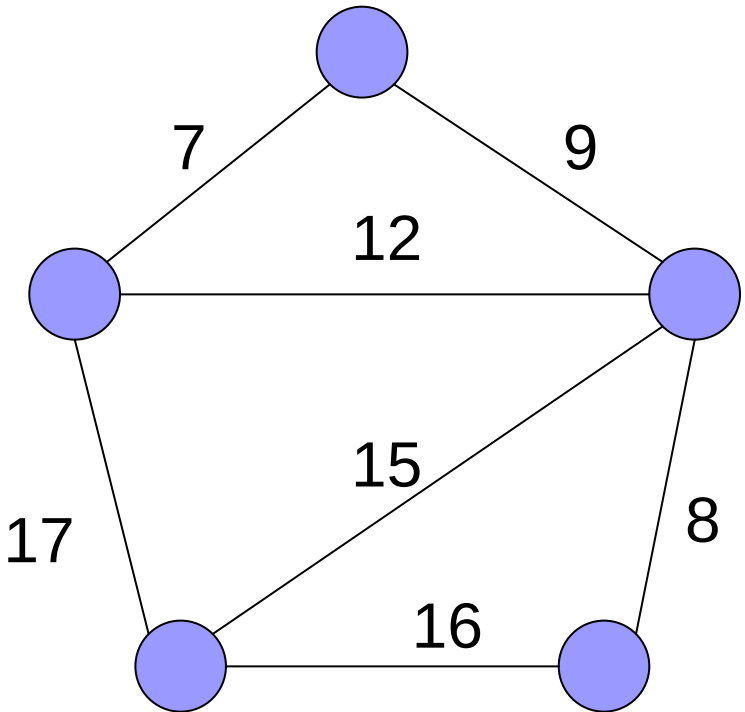
長さの短い枝を順に加える



クラスカルのアルゴリズム(p.11)

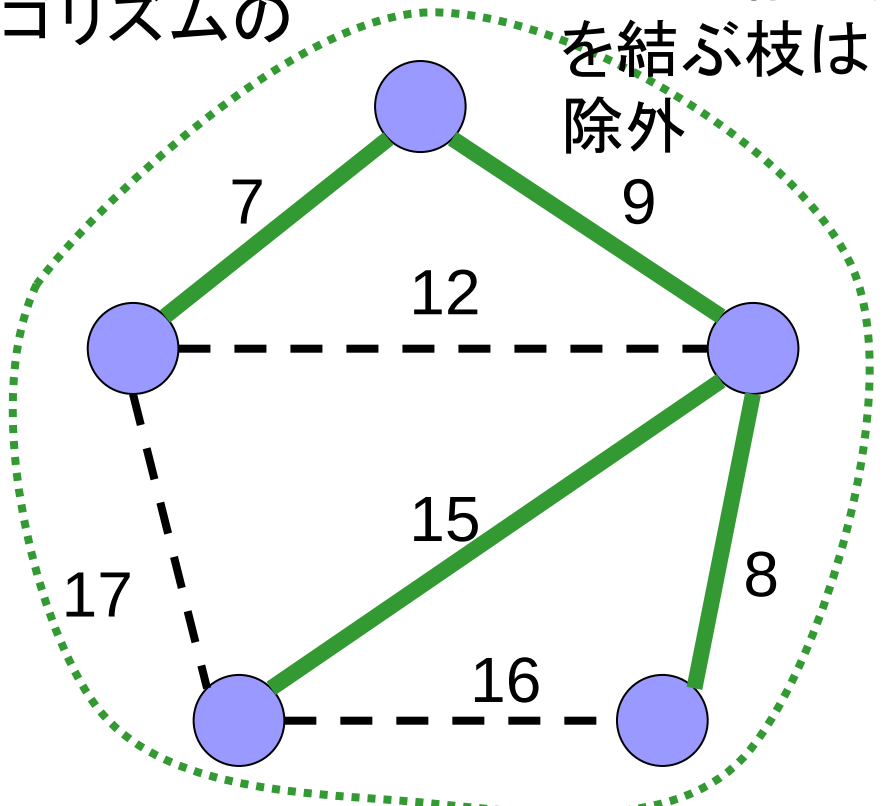
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

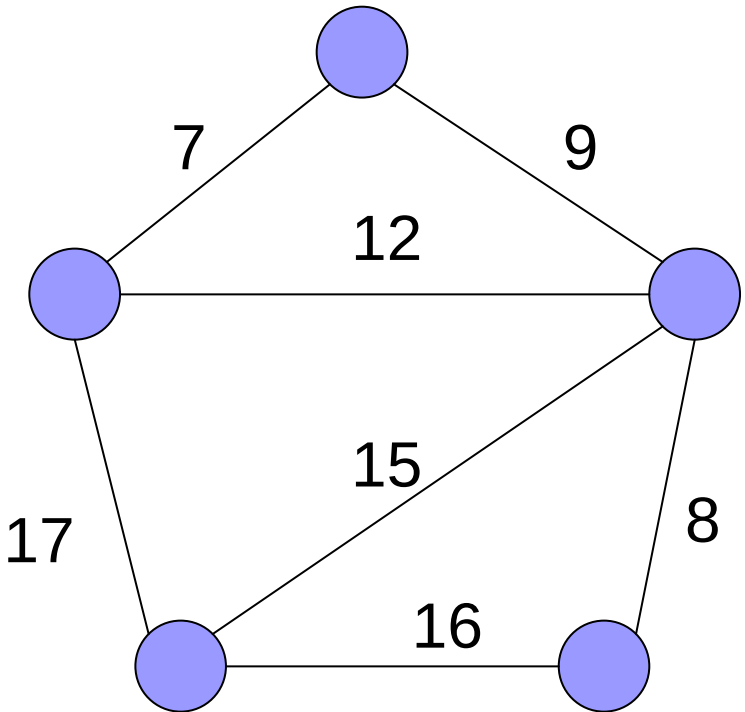
同じ連結成分を結ぶ枝は除外



クラスカルのアルゴリズム(p.11)

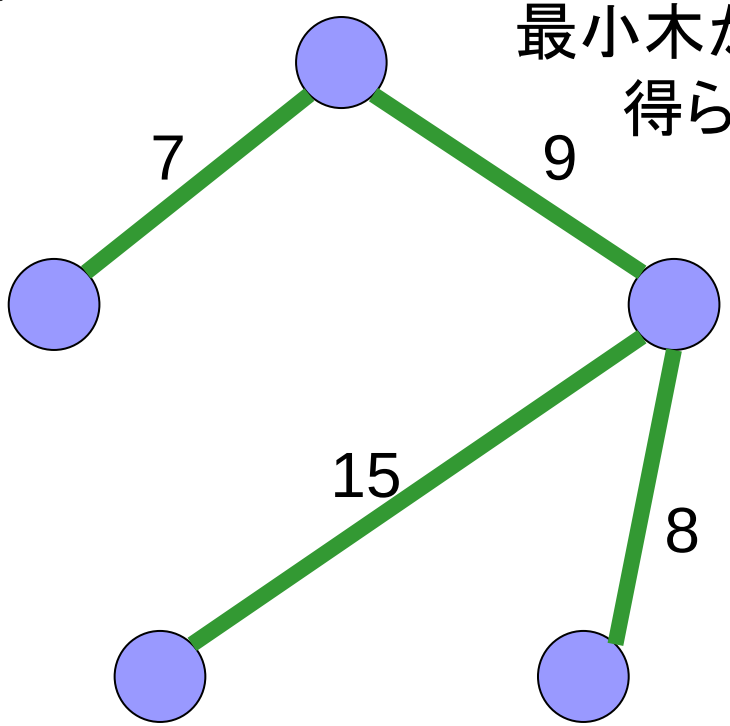
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



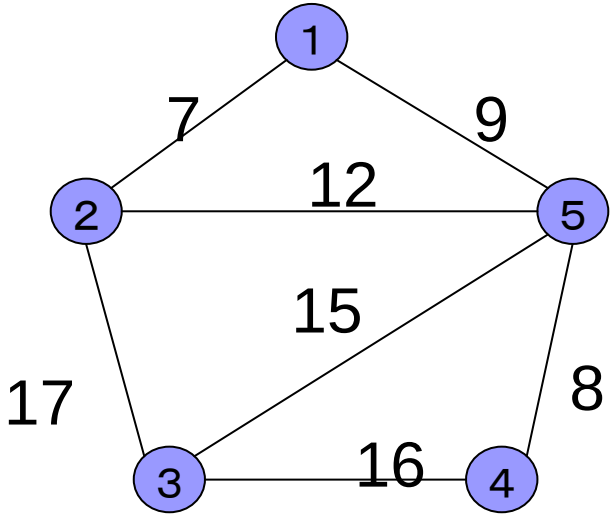
アルゴリズムの動き

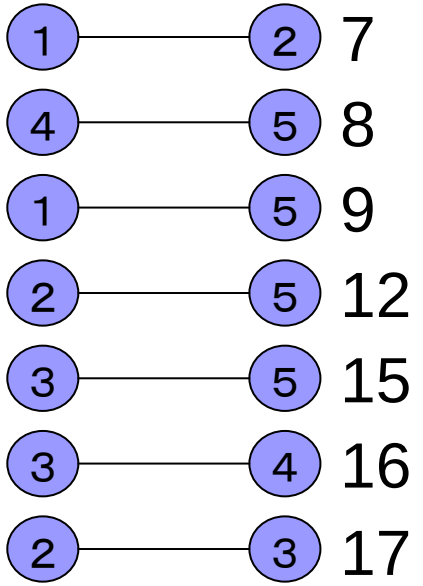
アルゴリズム終了
最小木が
得られた



クラスカルのアルゴリズムの計算時間

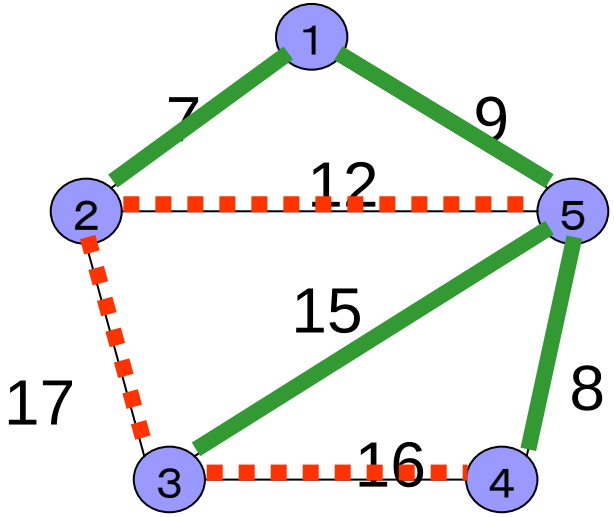
- アルゴリズムの実装方法
  - 枝を長さの短い順にソート --- O(m log m) = O(m log n)





クラスカルのアルゴリズムの計算時間

- アルゴリズムの実装方法
  - 枝を長さの短い順にソート --- O(m log m) = O(m log n)
  - 各枝の両端の節点が同じ連結成分に含まれるか否かチェック.
    - 同じ連結成分➔枝を除去
    - 異なる連結成分➔枝を加える



{1}, {2}, {4}, {5}, {3}

{1, 2}, {4}, {5}, {3}

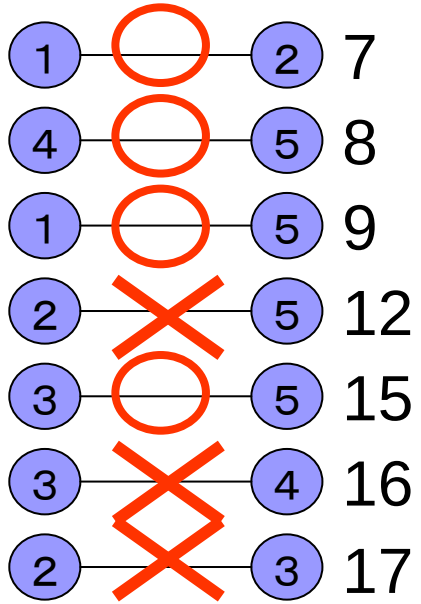
{1, 2}, {4, 5}, {3}

{1, 2, 4, 5}, {3}

{1, 2, 4, 5}, {3}

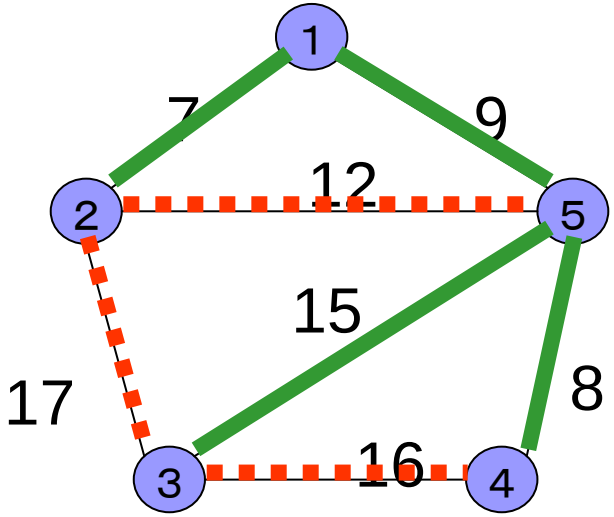
{1, 2, 4, 5, 3}

{1, 2, 4, 5, 3}



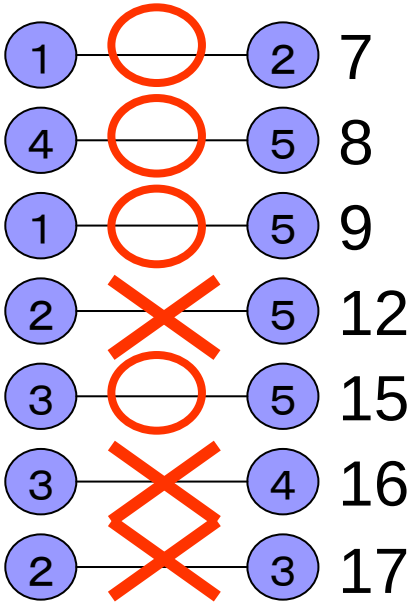
クラスカルのアルゴリズムの計算時間

- アルゴリズムの実装方法
  - 枝を長さの短い順にソート --- O(m log m) = O(m log n)
  - 各枝の両端の節点が同じ連結成分に含まれるか否かチェック.
    - 同じ連結成分➔枝を除去
    - 異なる連結成分➔枝を加える



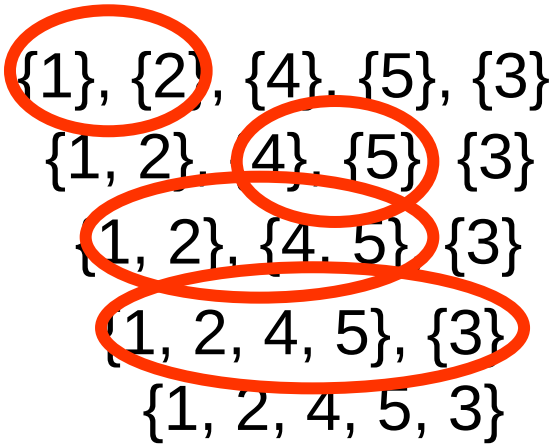
注意！
枝を加える
度に
連結成分は
変化する

集合族の併合の
ためのデータ構造
を利用する



集合族の併合のためのデータ構造

- 互いに素な複数の集合を繰り返し併合(merge)するプロセスを考える



集合族の併合

- 集合族 $S_1, S_2, \ldots, S_t$ に対する2つの操作
  - **MERGE(S_i, S_k)**: 集合S_iと S_k を併合，名前を S_i もしくは S_k とする
  - **FIND(x)**: 要素 x を含む集合の名前を返す

MERGE(S_1, S_2) $\{S_1: 1\}, \{S_2: 2\}, \{S_4: 4\}, \{S_5: 5\}, \{S_3: 3\}$

FIND(5)=S_5

MERGE(S_4, S_5) $\{S_1: 1, 2\}, \{S_4: 4\}, \{S_5: 5\}, \{S_3: 3\}$

FIND(5)=S_5

MERGE(S_1, S_4) $\{S_1: 1, 2\}, \{S_4: 4, 5\}, \{S_3: 3\}$

FIND(5)=S_4

MERGE(S_1, S_3) $\{S_1: 1, 2, 4, 5\}, \{S_3: 3\}$

FIND(5)=S_1

$\{S_1: 1, 2, 4, 5, 3\}$

集合族の併合：クラスカルのアルゴリズムの場合

- 集合族 $S_1, S_2, \ldots, S_t$ に対する2つの操作
  - **MERGE(S_i, S_k)**: 集合S_iと S_k を併合，名前を S_i もしくは S_k とする
  - **FIND(x)**: 要素 x を含む集合の名前を返す
- クラスカルのアルゴリズムでは．．．
  - **MERGE(S_i, S_k)**: 枝を追加➔連結成分を併合

    ∴ n-1 回繰り返す（n =グラフの節点数）
  - **FIND(x)**: 現在の枝 (u, v) に対し，

    両端点が同じ連結成分に含まれる➔➔ FIND(u) = FIND(v)

    ∴ 2m 回繰り返す（m=グラフの枝数）

集合族の併合のデータ構造：配列による簡単な実現

- 全要素数 N の大きさの配列 set_name を使う
- 要素 j に対し set_name[j] =（j を含む集合の名前）と定義

$\{S_1: 1, 2\}, \{S_4: 4\}, \{S_5: 5\}, \{S_3: 3\}$

MERGE(S_4, S_5)

$\{S_1: 1, 2\}, \{S_4: 4, 5\}, \{S_3: 3\}$

MERGEのとき，set_name の
全ての値を調べる必要あり

➔ 1回のMERGEにつきO(N)時間
1回のFINDはO(1)時間

要素名	1	2	3	4	5
set_name	1	1	3	4	5

要素名	1	2	3	4	5
set_name	1	1	3	4	4

set_name[j]=5 ならば
set_name[j]=4 と
置き換える

クラスカルのアルゴリズムの計算時間（その１）

- アルゴリズムの実装方法
  - 枝を長さの短い順にソート --- O(m log m) = O(m log n)
  - 各枝の両端の節点が同じ連結成分に含まれるか否かチェック.
    - 同じ連結成分➔枝を除去
    - 異なる連結成分➔枝を加える

      集合族の併合のデータ構造として配列を使用:

      FINDの回数:2m回 --- O(m) 時間

      MERGEの回数: n-1回 --- $O(n^2)$時間

クラスカルのアルゴリズムの計算時間
$= O(m \log n + m + n^2) = O(m \log n + n^2)$

集合族の併合のデータ構造を改良すると，計算時間を改善できる➔来週

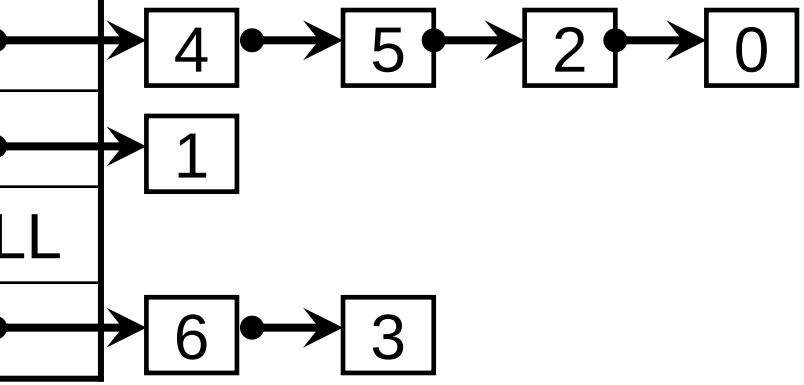
集合族の併合のデータ構造：配列＋リストによる実現

- 配列set_nameに加え，各集合をリストで表現

要素名	0	1	2	3	4	5	6
set_name	0	1	0	3	0	0	3

$\{S_0: 0, 2, 4, 5\}, \{S_1: 1\}, \{S_3: 3, 6\}$

集合の名前	集合の大きさ	先頭要素へのポインタ
S_0	4	
S_1	1	
S_2	0	NULL
S_3	2	



集合族の併合のデータ構造：配列＋リストによる実現

- S_1とS_3を併合し，名前をS_3とするときの実行例

$\{S_0: 0, 2, 4, 5\}, \{S_1: 1\}, \{S_3: 3, 6\}$

要素名	0	1	2	3	4	5	6
set_name	0	~~1~~ 3	0	3	0	0	3

①S_1の各要素のset_nameを変更，
S_1, S_3の大きさを変更
計算時間：
$O(|S_1|)$

集合の名前	集合の大きさ	先頭要素へのポインタ
S_0	4	→ 4 → 5 → 2 → 0
S_1	~~1~~ 0	→ 1
S_2	0	NULL
S_3	~~2~~ 3	→ 6 → 3

集合族の併合のデータ構造：配列＋リストによる実現

- S_1とS_3を併合し，名前をS_3とするときの実行例

$\{S_0: 0, 2, 4, 5\}, \{S_1: 1\}, \{S_3: 3, 6\}$

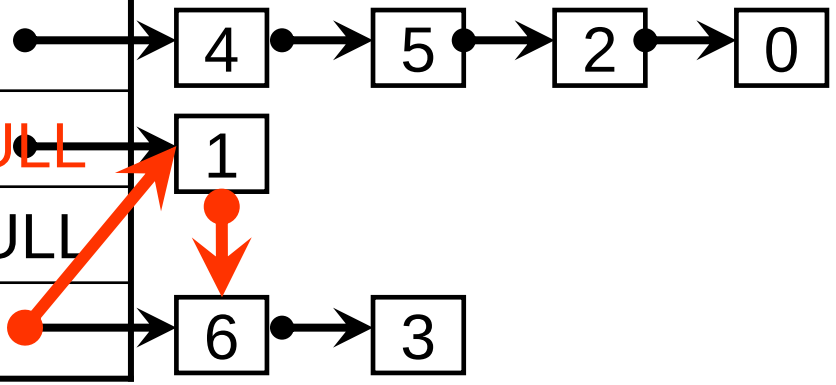
要素名	0	1	2	3	4	5	6
set_name	0	3	0	3	0	0	3

②S_1, S_3のポインタの付け替え

計算時間：
$O(|S_1|)$

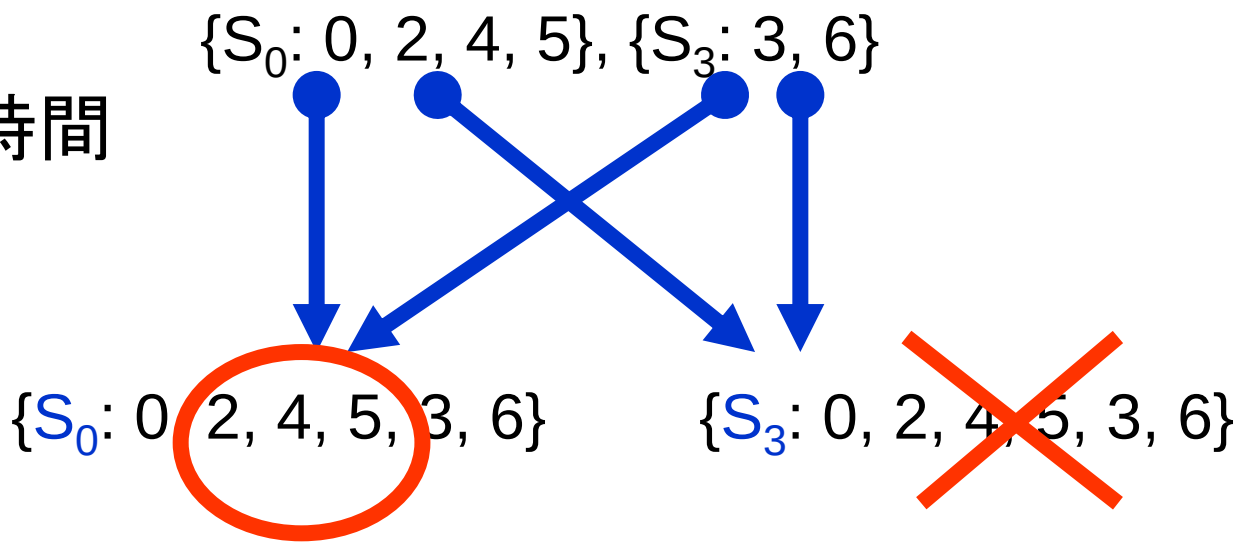
集合の名前	集合の大きさ	先頭要素へのポインタ
S_0	4	
S_1	0	NULL
S_2	0	NULL
S_3	3	

(1) S1の最後の要素からS3の先頭へポインタを張る
(2) S1の先頭要素をS3の先頭にする



集合族の併合のデータ構造：配列＋リストによる実現

- 併合1回の計算時間＝O(併合される集合の大きさ)
  ➔何も工夫しないと，N回の併合では最悪$O(N^2)$時間
  FIND は1回あたりO(1)時間
- 併合の工夫：常に大きい集合に小さい集合を併合
  ➔N回の併合で
  $O(N \log_2 N)$時間



クラスカルのアルゴリズムの計算時間

- アルゴリズムの実装方法
  - 枝を長さの短い順にソート --- O(m log m) = O(m log n)
  - 各枝の両端の節点が同じ連結成分に含まれるか否かチェック.
    - 同じ連結成分➔枝を除去
    - 異なる連結成分➔枝を加える

      FINDの回数：2m回 --- O(m)時間

      MERGEの回数： n-1回 --- O(n $\log_2$ n)時間

クラスカルのアルゴリズムの計算時間＝O(m log n)

レポート問題

問1：左下の行列はある無向グラフの隣接行列を表す．
（1）この隣接行列が表す無向グラフを書きなさい．
（2）この無向グラフに対する接続行列，および隣接リストを書きなさい．
（3）このグラフの最小木をクラスカルのアルゴリズムによって求めなさい．計算の過程についても省略せずに書くこと．
なお，各枝の重みは右下の行列によって与えられる．

	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	0	0	1	0	1
2	1	0	0	1	1	1
3	1	1	1	0	0	1
4	1	0	1	0	0	1
5	0	1	1	1	1	0

	0	1	2	3	4	5
0		4	3	5	9	
1				2		7
2				1	8	6
3						10
4						12
5						

レポート問題

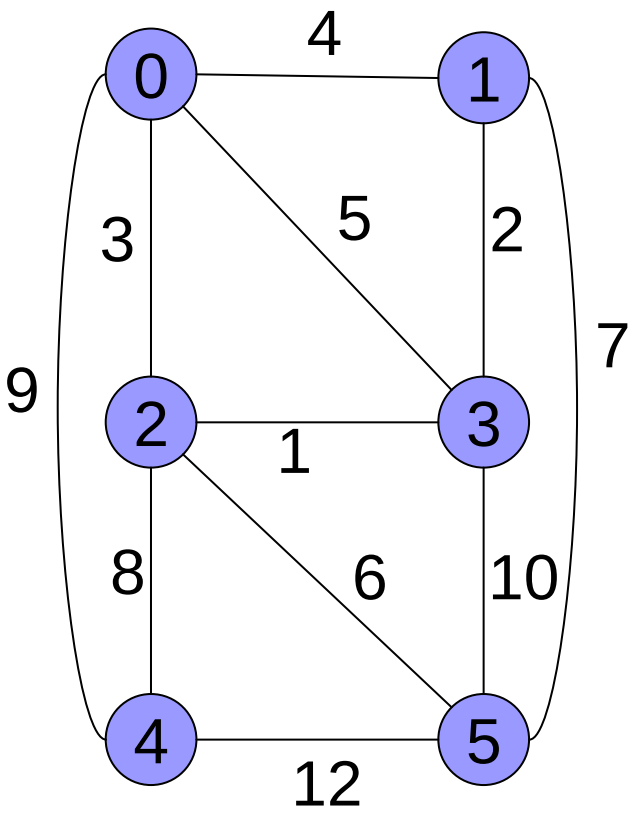
（1）無向グラフの2頂点u, vが与えられたとき，枝(u,v)が存在するか否かを判定したい．

接続行列，隣接行列，隣接リスト（連結リストの場合）のそれぞれを使った場合のアルゴリズムと時間計算量と説明せよ．

（2）頂点 u が与えられたとき，u に接続する枝をすべて求めたい．

接続行列，隣接行列，隣接リスト（連結リストの場合）のそれぞれを使った場合のアルゴリズムと時間計算量と説明せよ．

レポート問題の解答例：無向グラフの図



接続行列，隣接リストは省略

レポート問題の解答例：最小木を求める過程