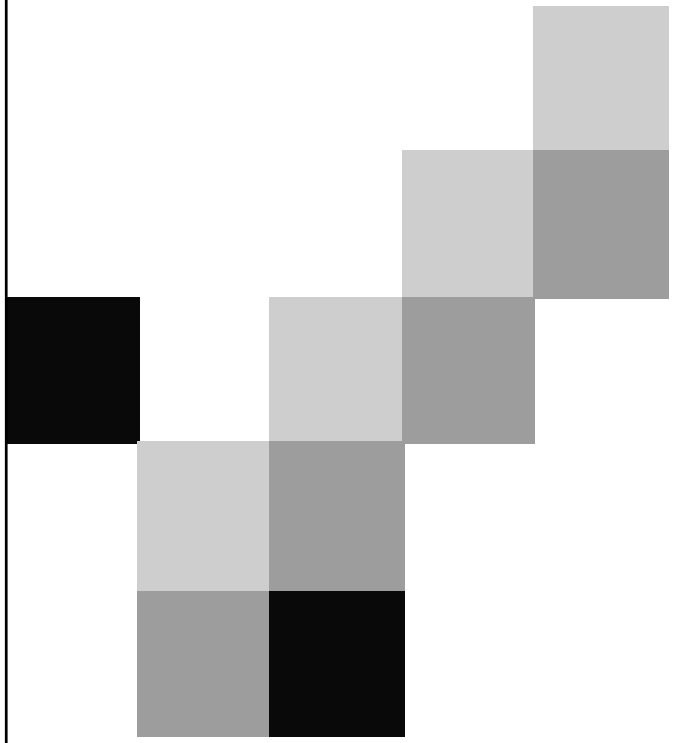


アルゴリズムとデータ構造

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

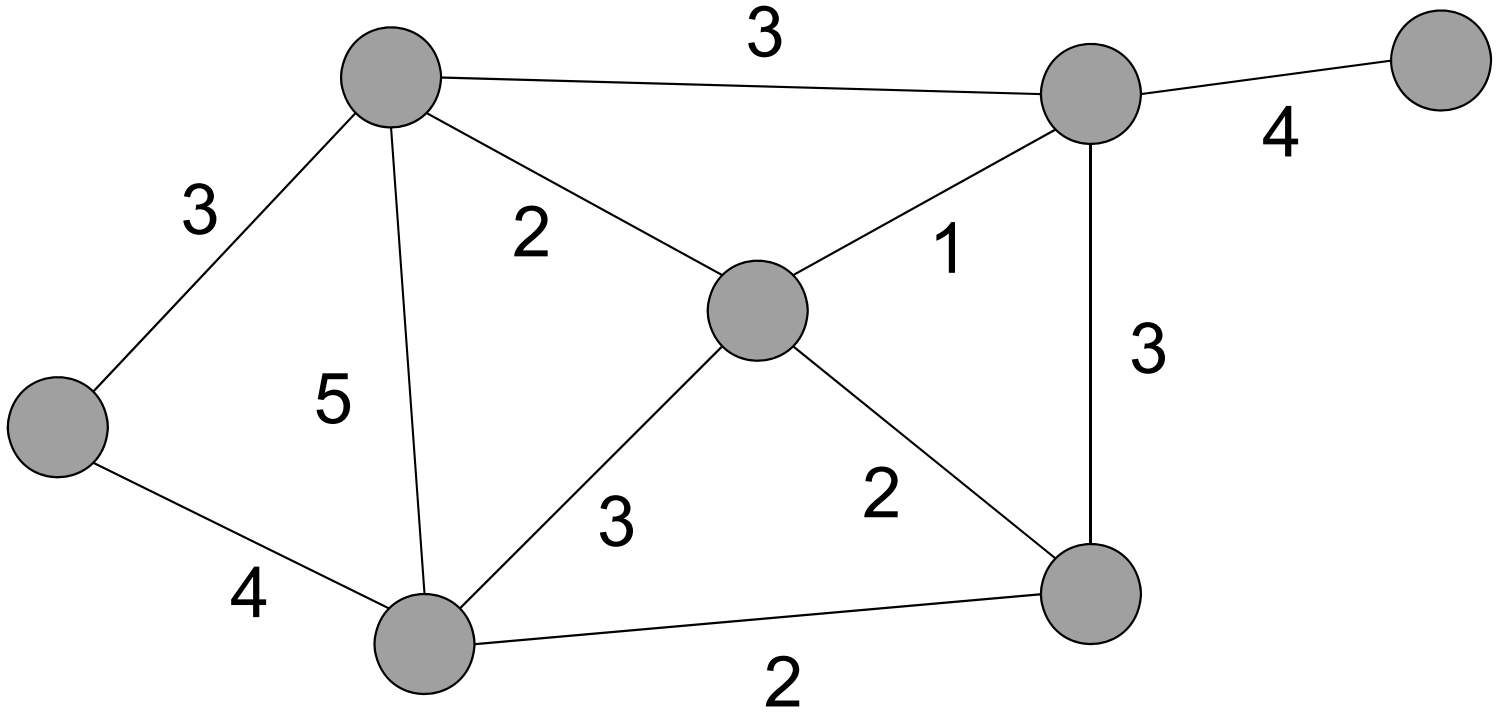
連絡事項

- 期末試験を1月21日に実施
- 今日は授業アンケートを行います

最小木問題(p.4)

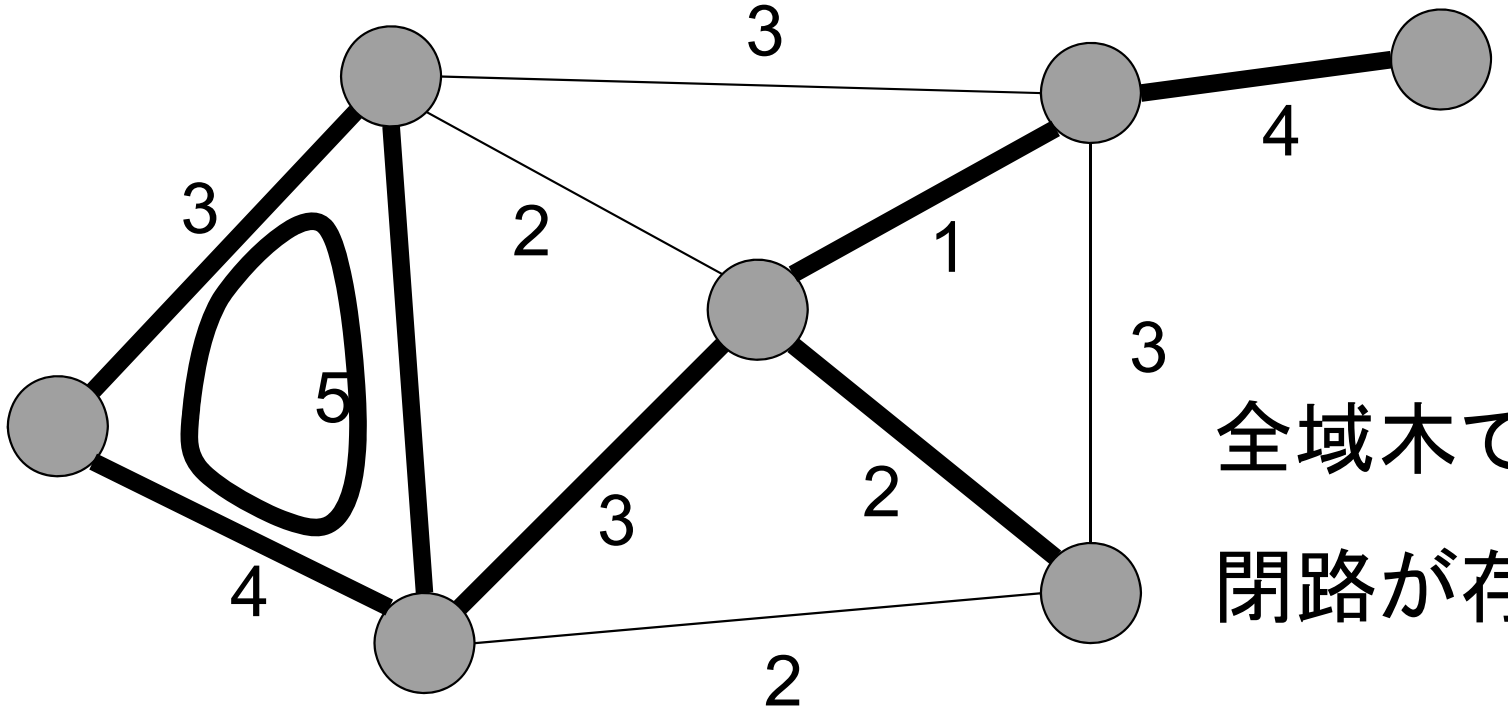
minimum spanning tree problem

- 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）

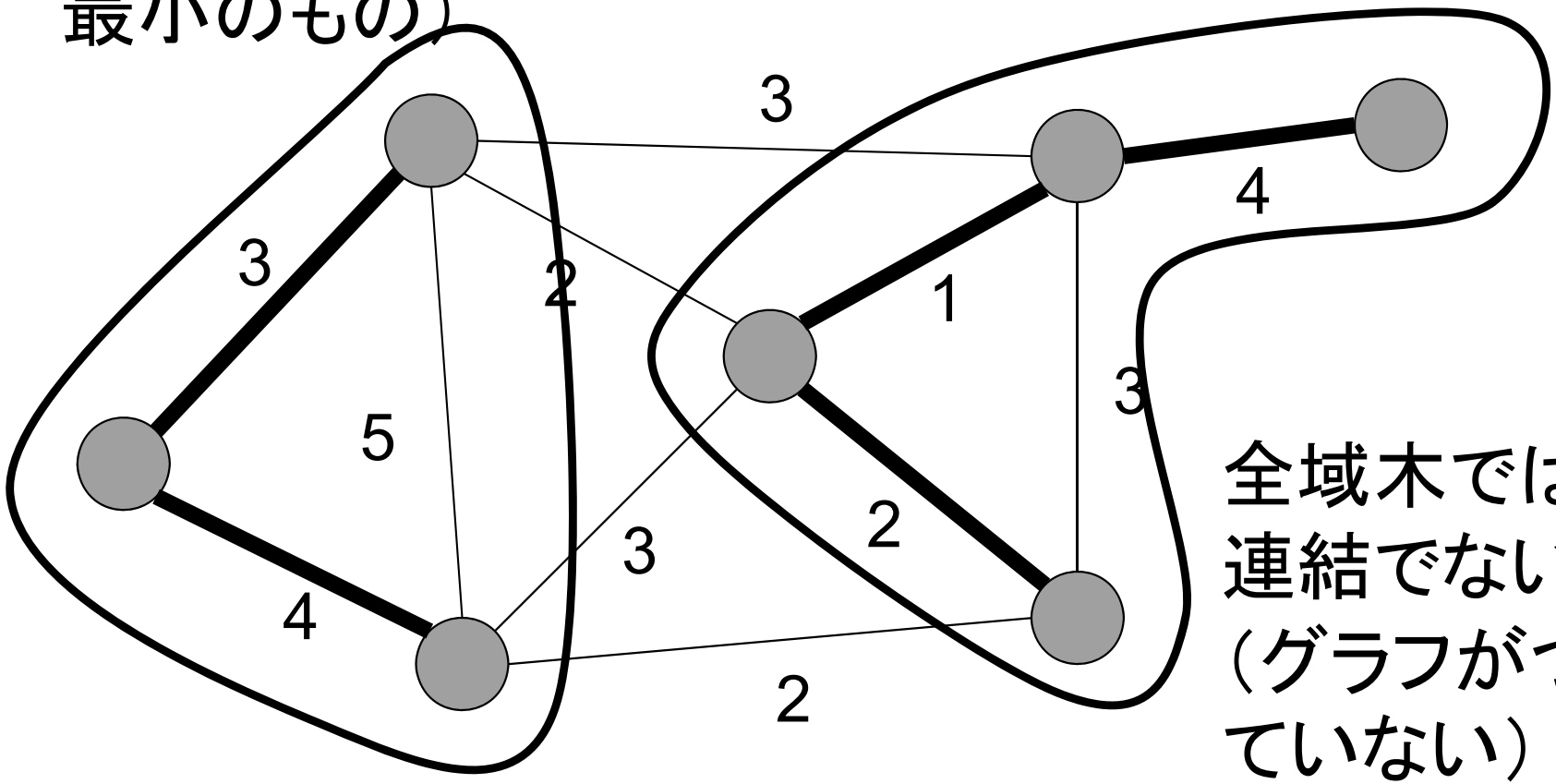


全域木ではない！

閉路が存在

最小木問題(p.4)

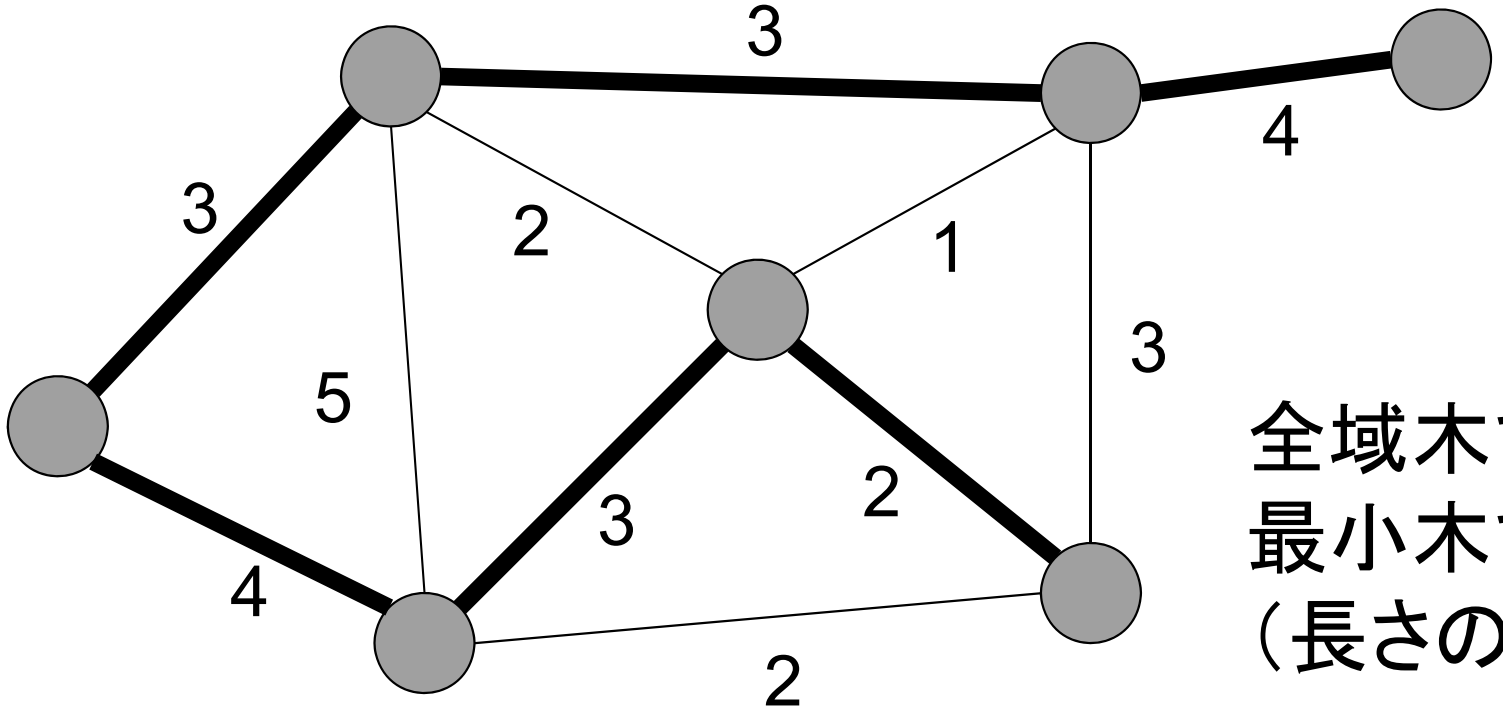
- 入力：無向グラフG=(V,E)，各枝の長さd(e) (e∈E)
- 出力：Gの最小木（Gの全域木で，枝の長さの和が最小のもの）



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)

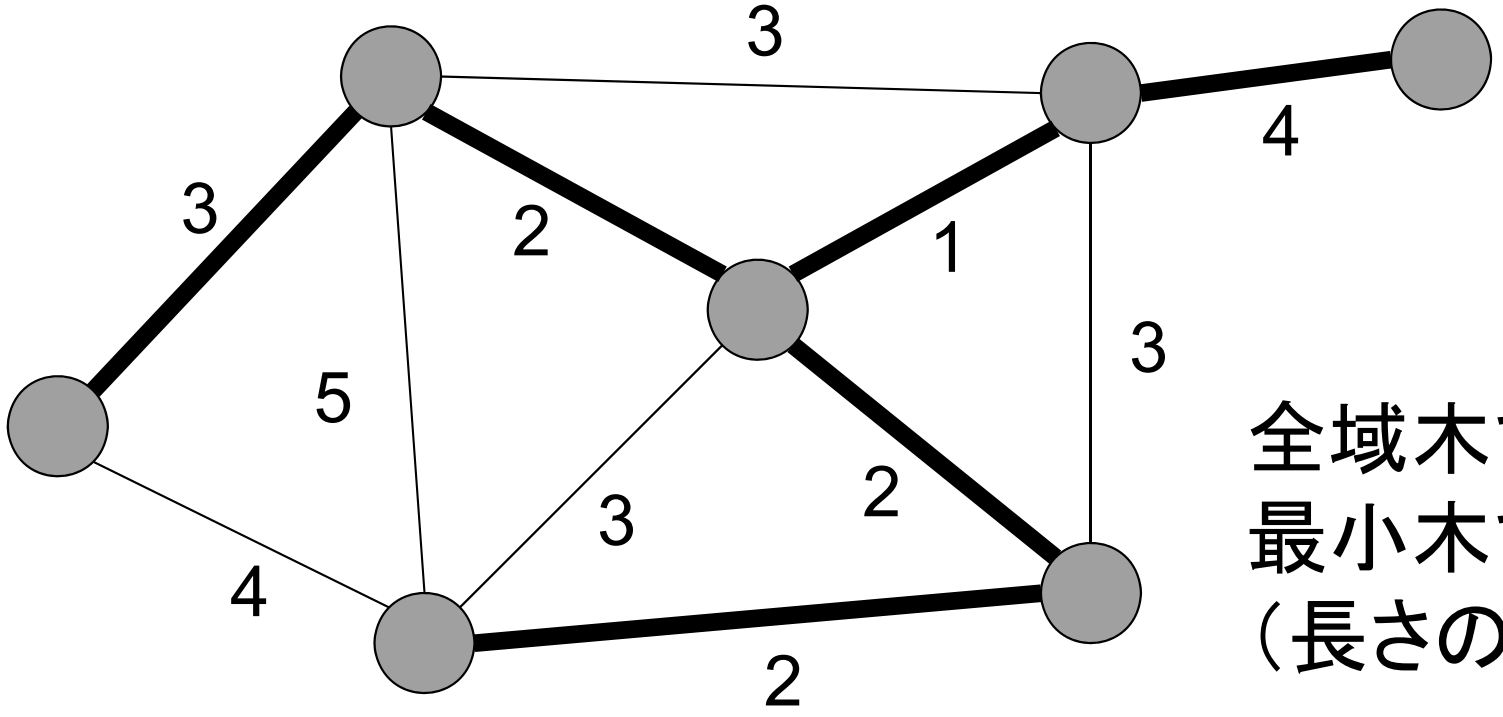
- 入力：無向グラフG=(V,E)，各枝の長さd(e) (e∈E)
- 出力：Gの最小木（Gの全域木で，枝の長さの和が最小のもの）



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)

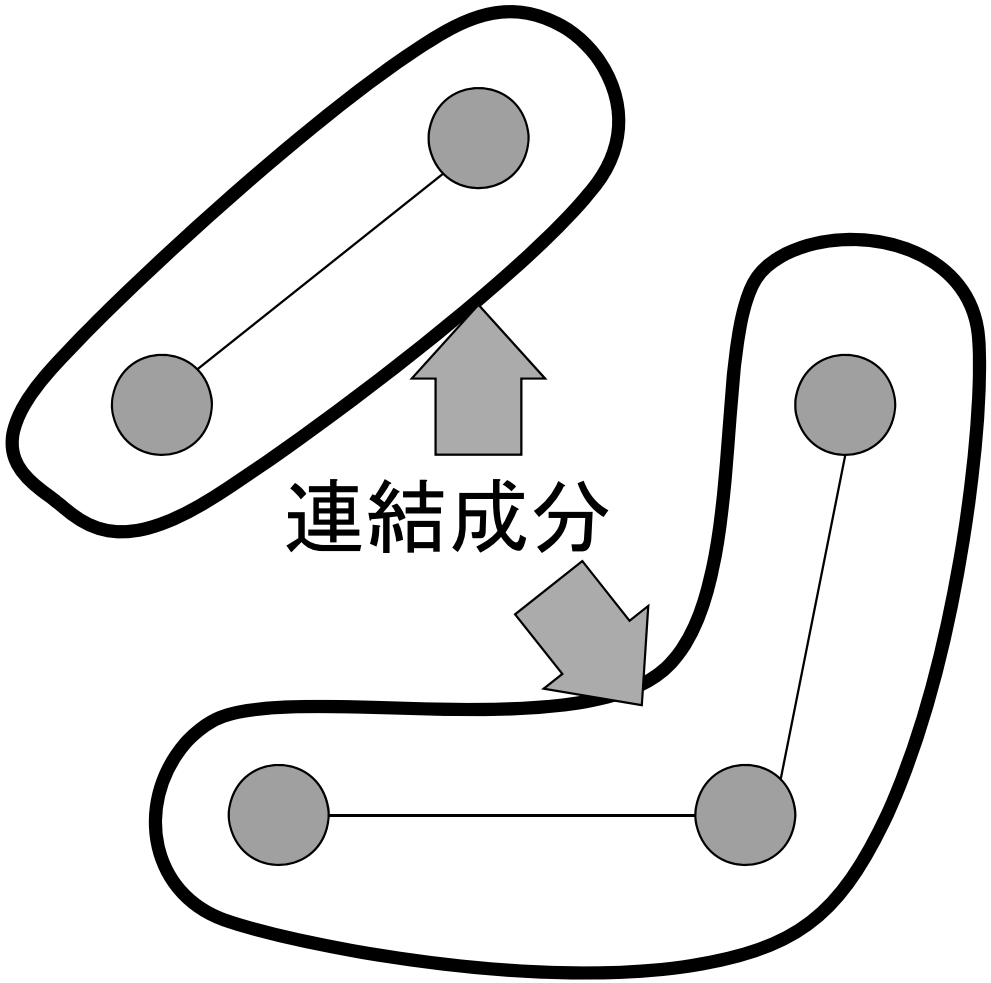
- 入力：無向グラフG=(V,E)，各枝の長さd(e) (e∈E)
- 出力：Gの最小木（Gの全域木で，枝の長さの和が最小のもの）



全域木であり，
最小木である
（長さの和＝14）

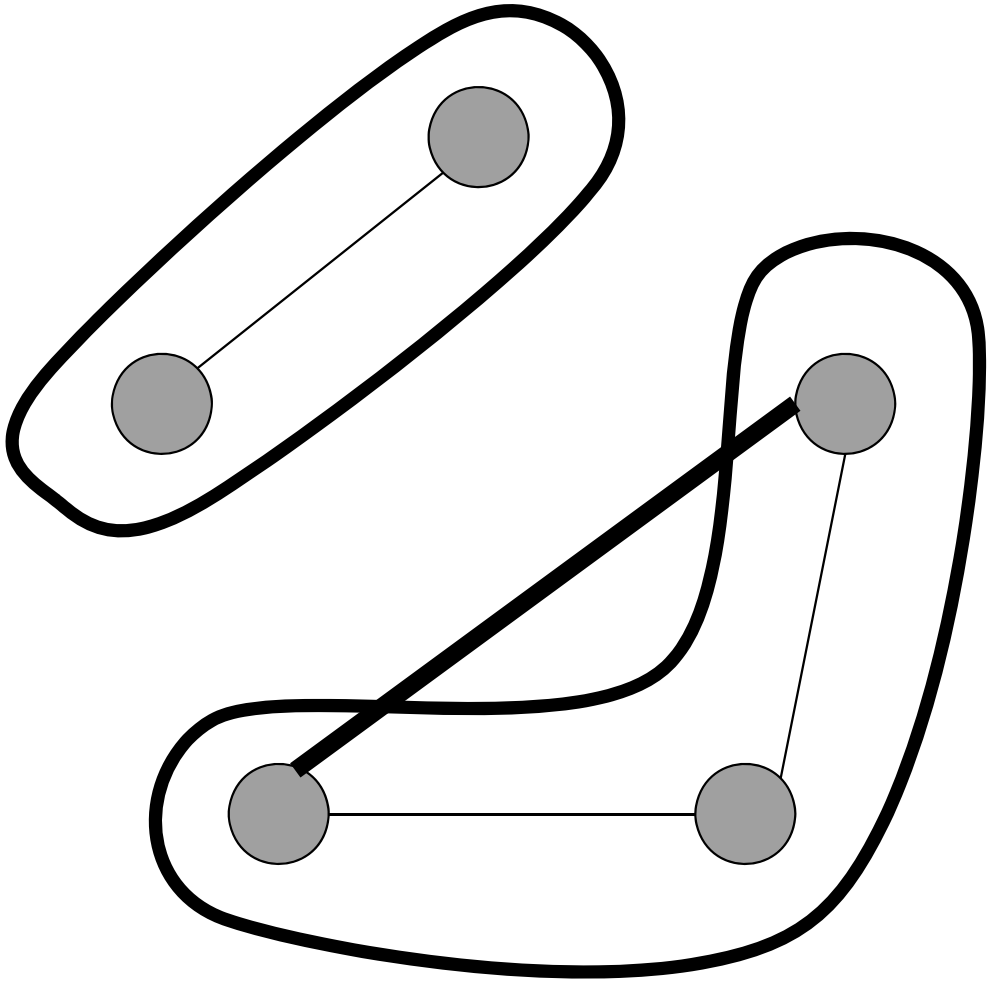
グラフの連結成分

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

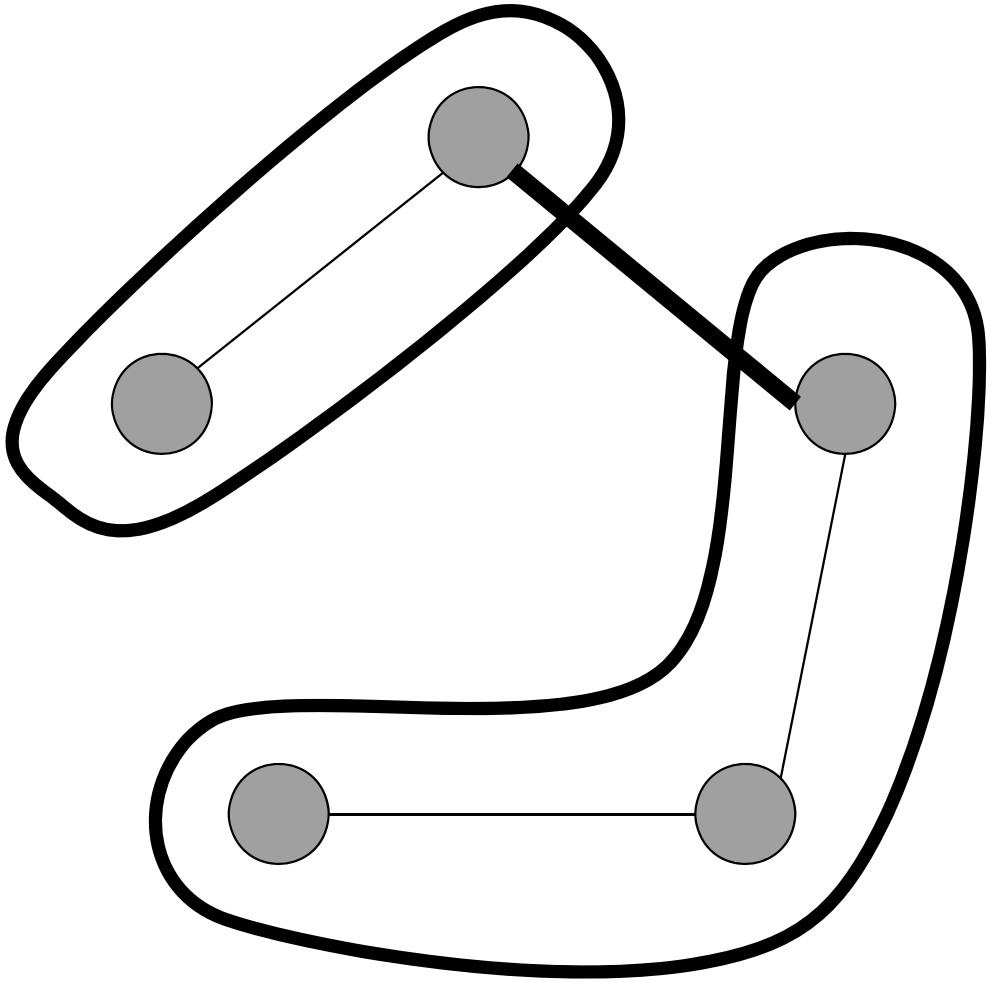
グラフの連結成分＝枝で結ばれている頂点の集合



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

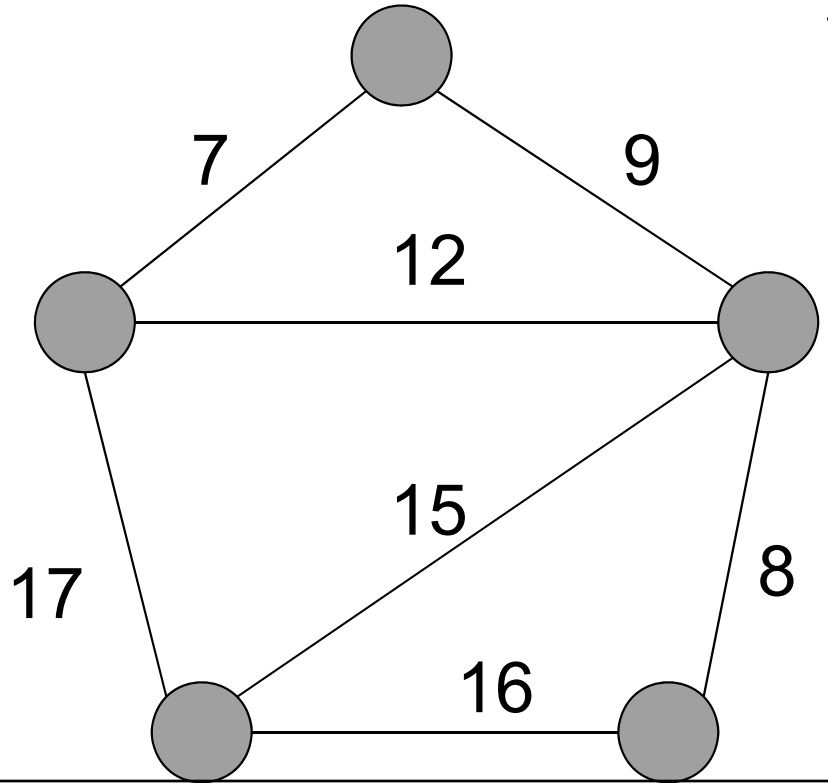
最小木を求めるアルゴリズム

- 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
  - 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

クラスカルのアルゴリズム(p.11)

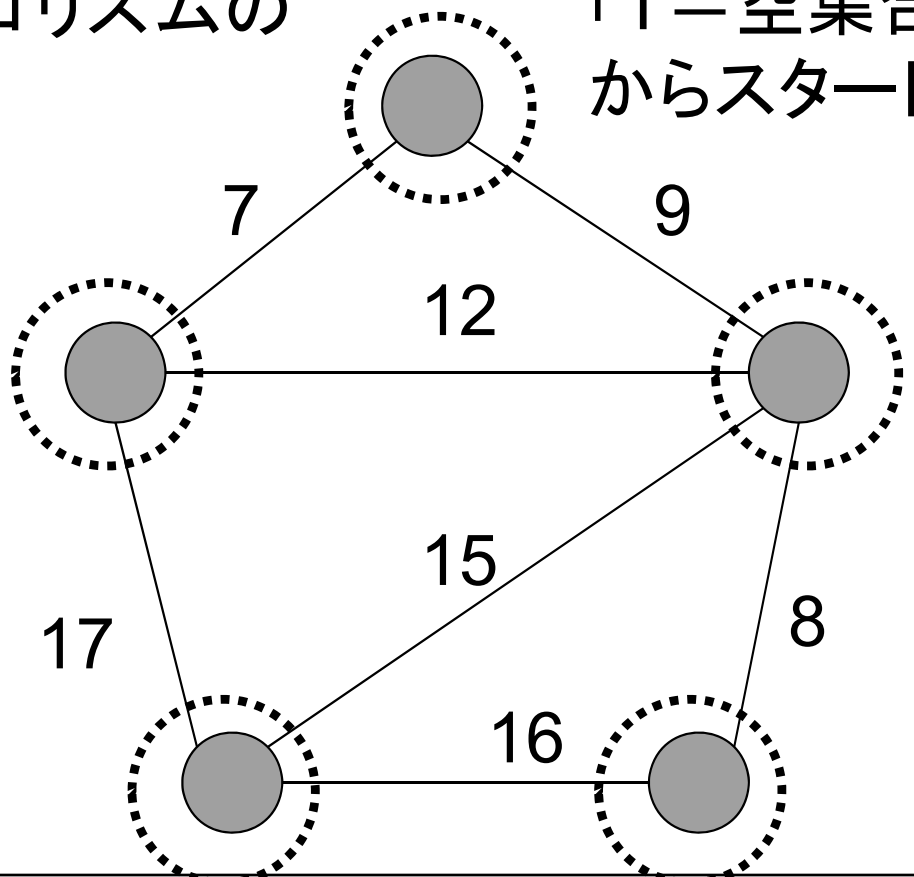
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

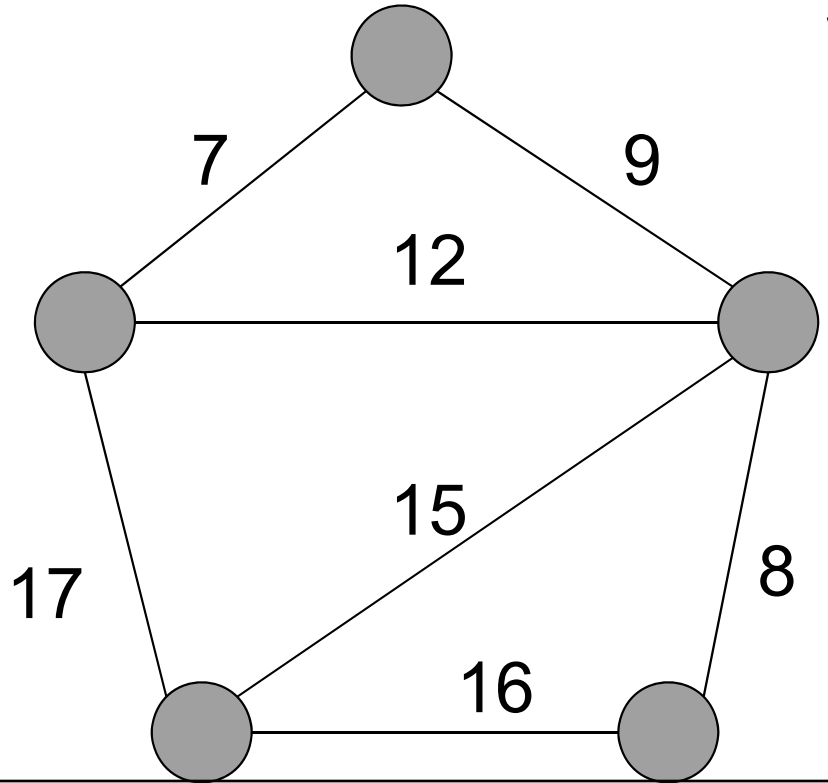
「T＝空集合」からスタート



クラスカルのアルゴリズム(p.11)

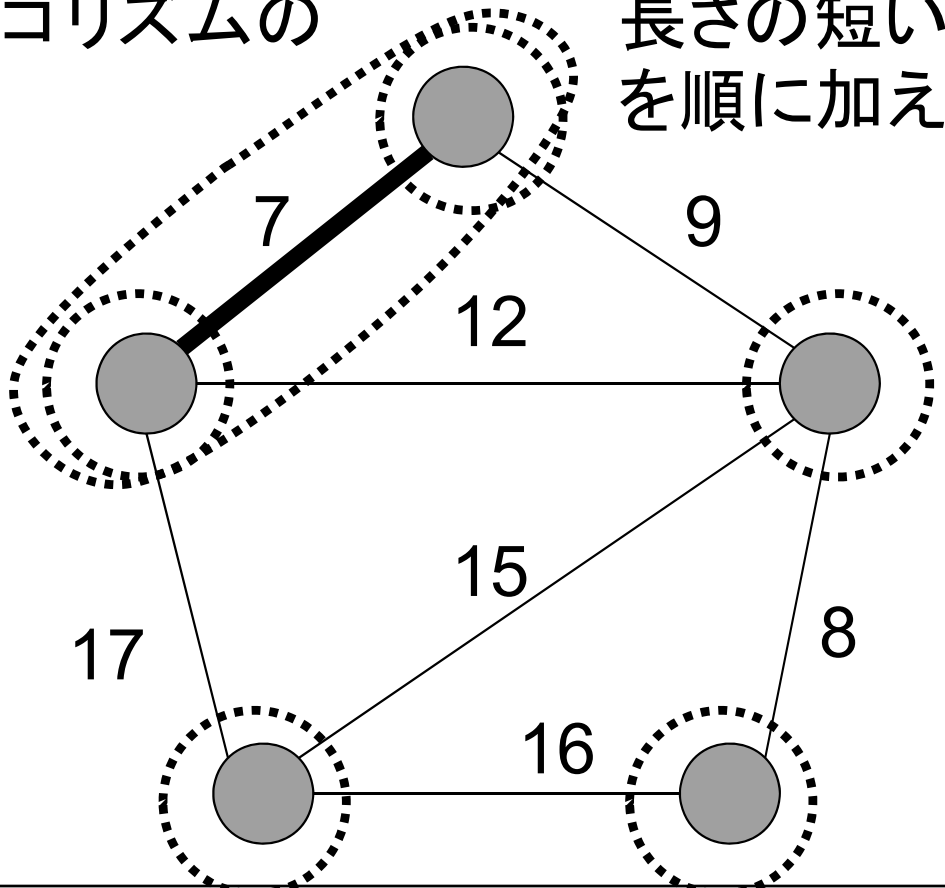
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

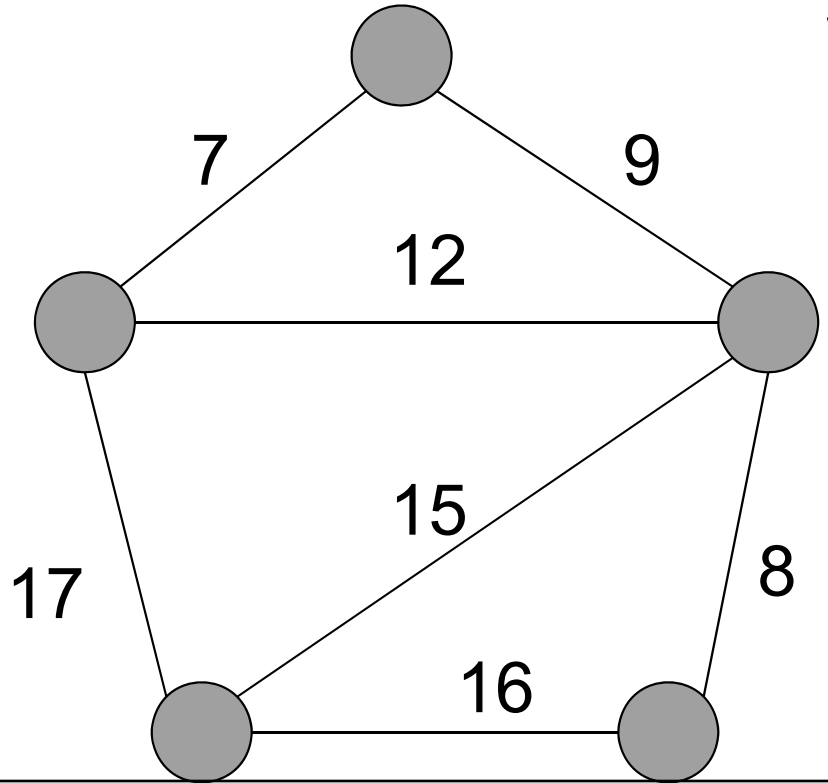
長さの短い枝を順に加える



クラスカルのアルゴリズム(p.11)

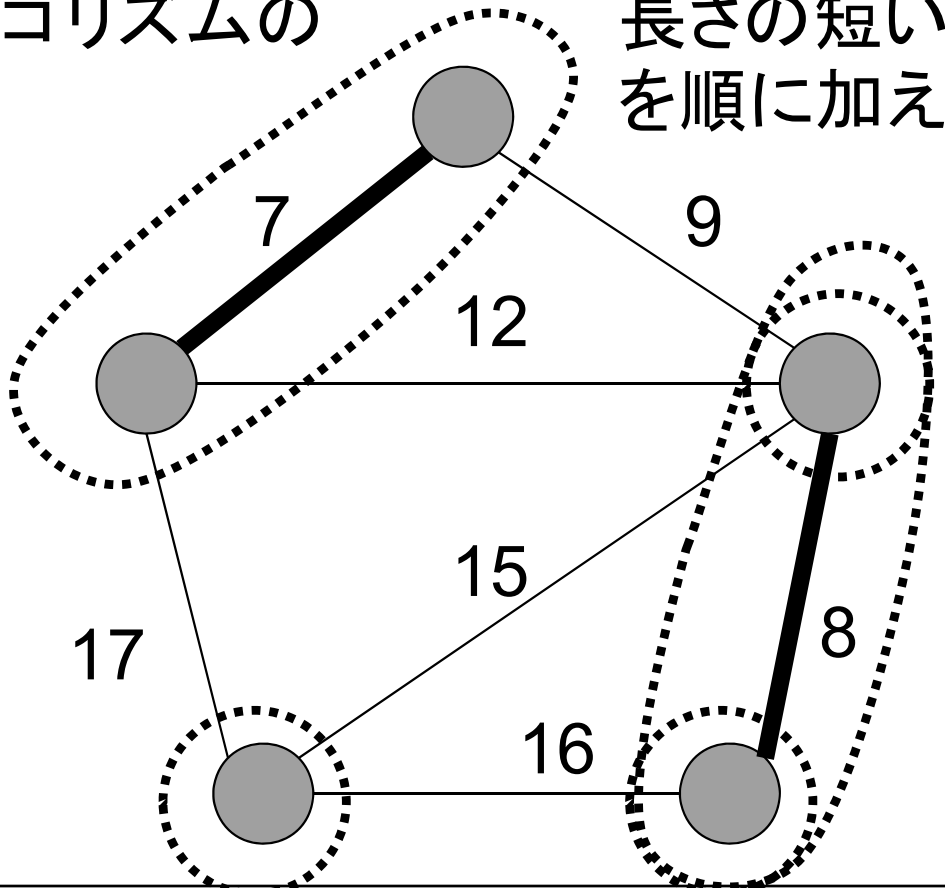
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

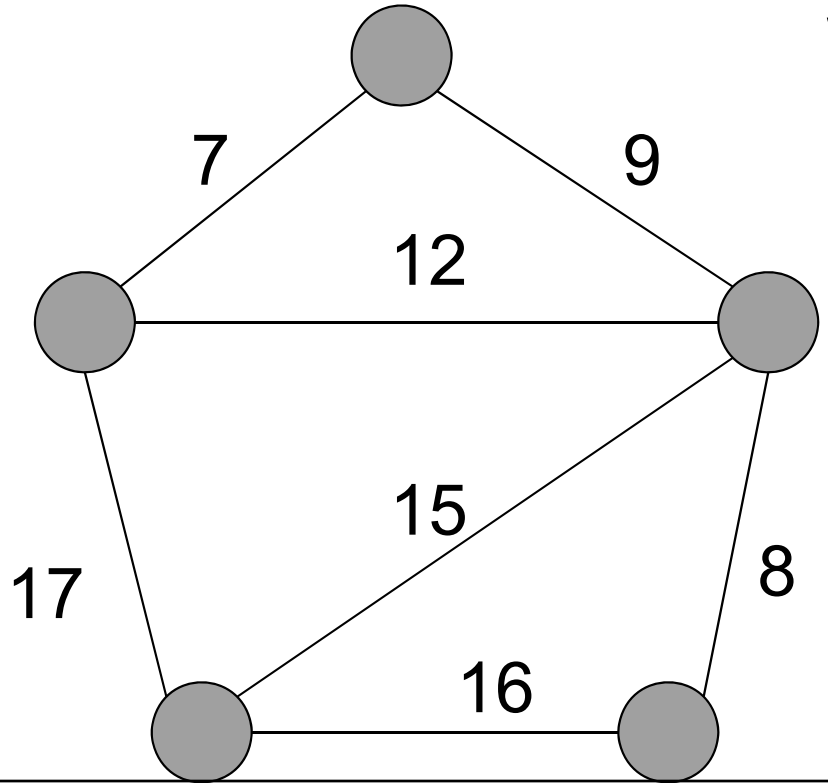
長さの短い枝を順に加える



クラスカルのアルゴリズム(p.11)

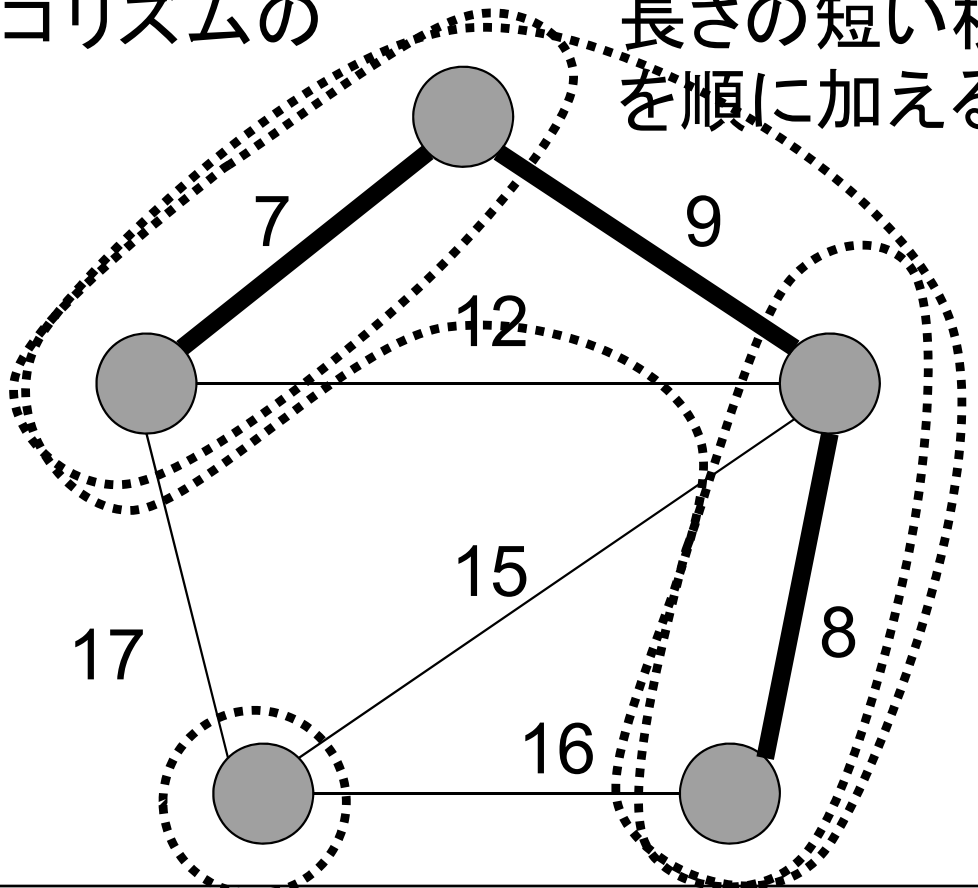
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

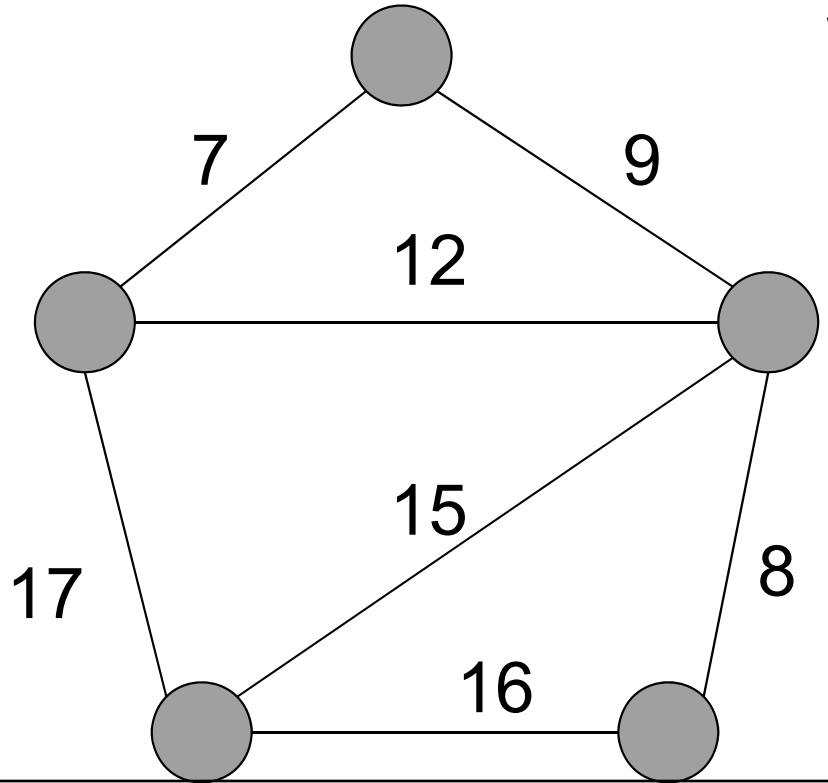
長さの短い枝を順に加える



クラスカルのアルゴリズム(p.11)

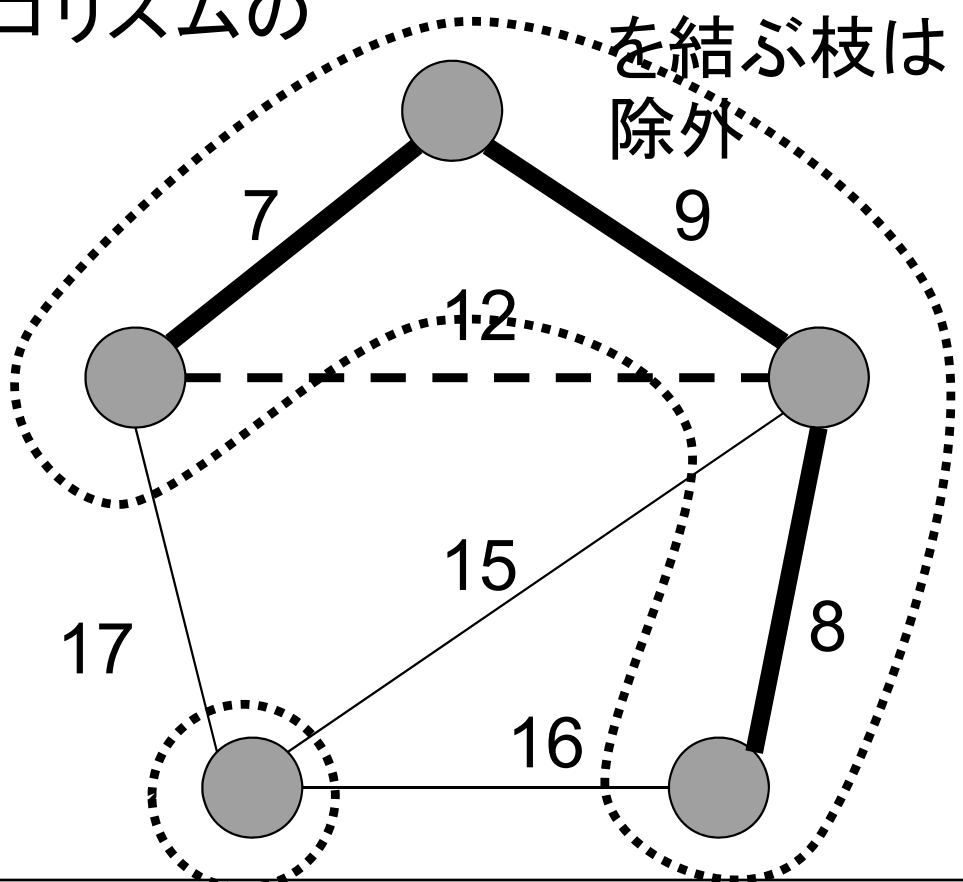
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

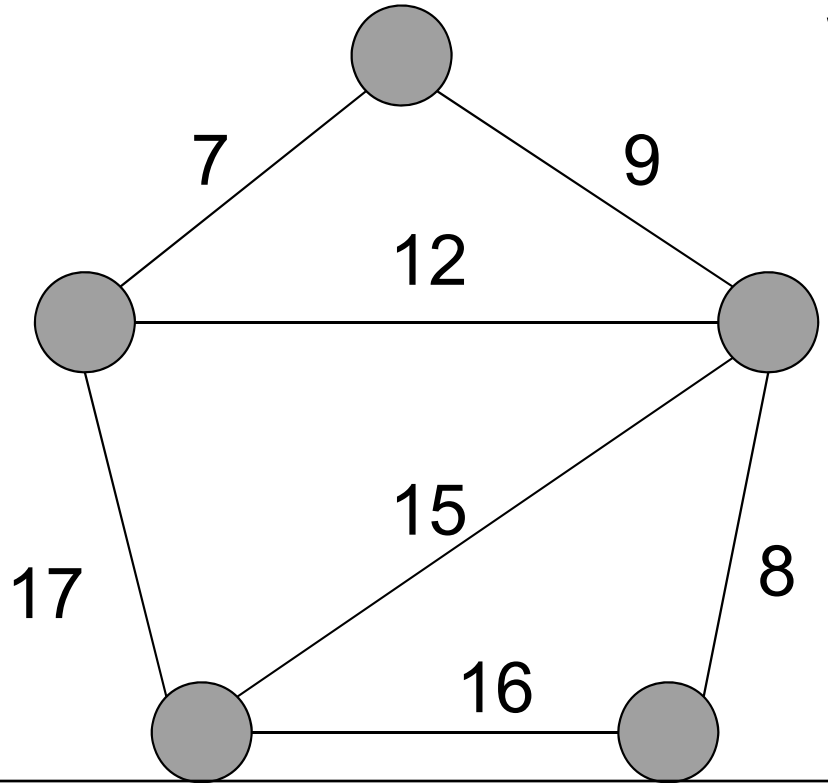
同じ連結成分を結ぶ枝は除外



クラスカルのアルゴリズム(p.11)

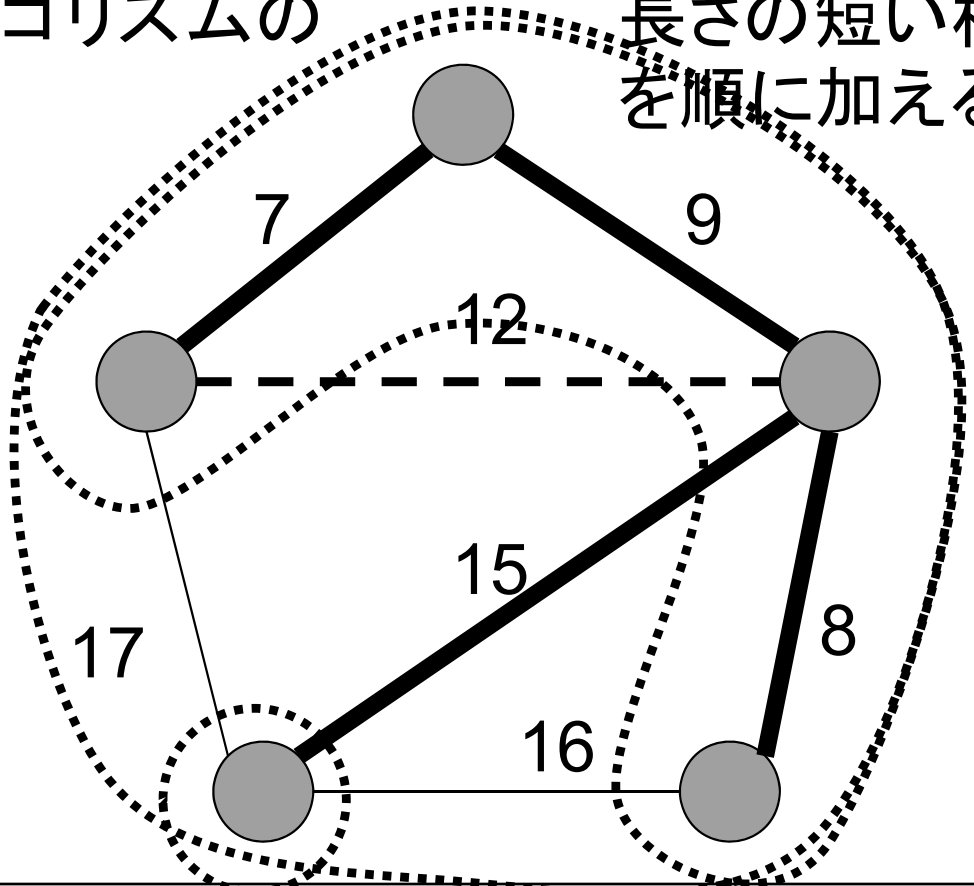
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

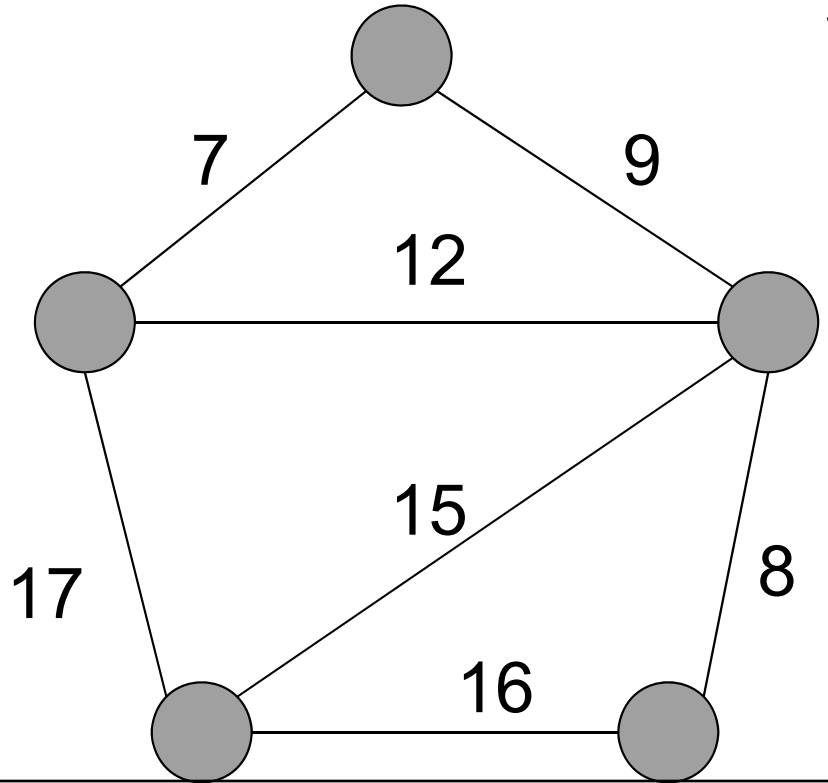
長さの短い枝を順に加える



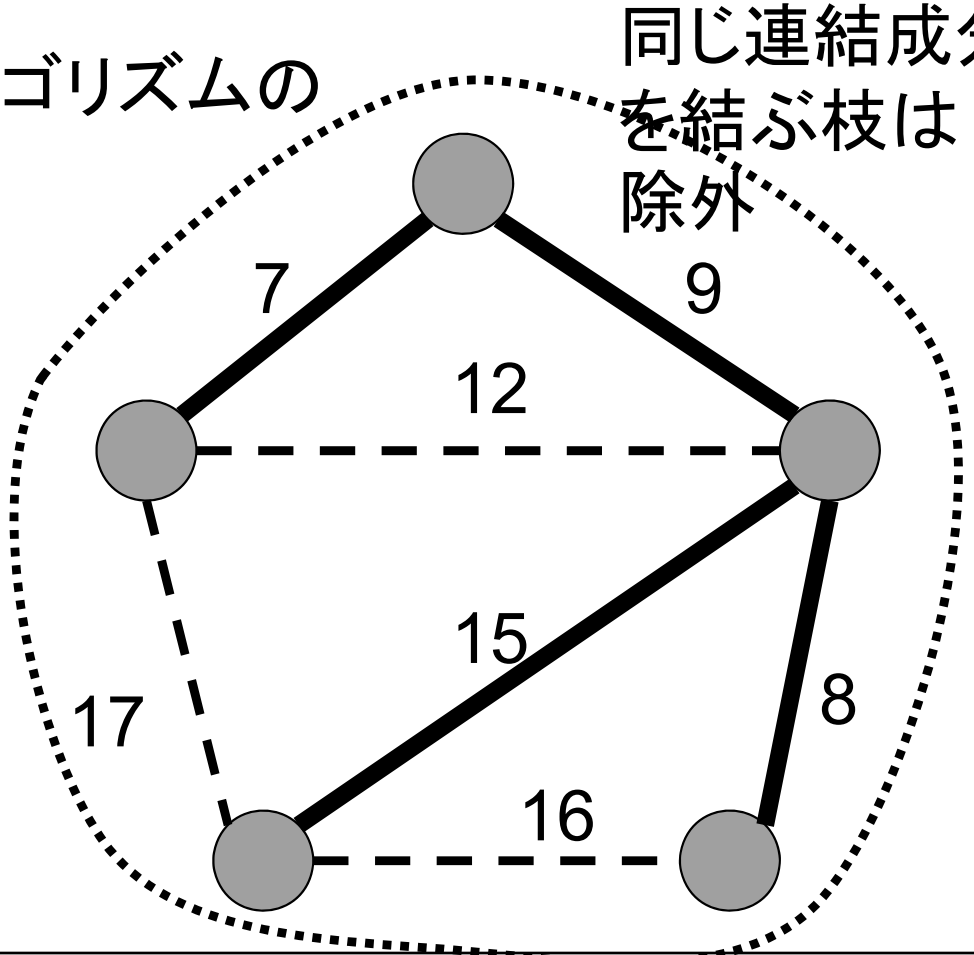
クラスカルのアルゴリズム(p.11)

- 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



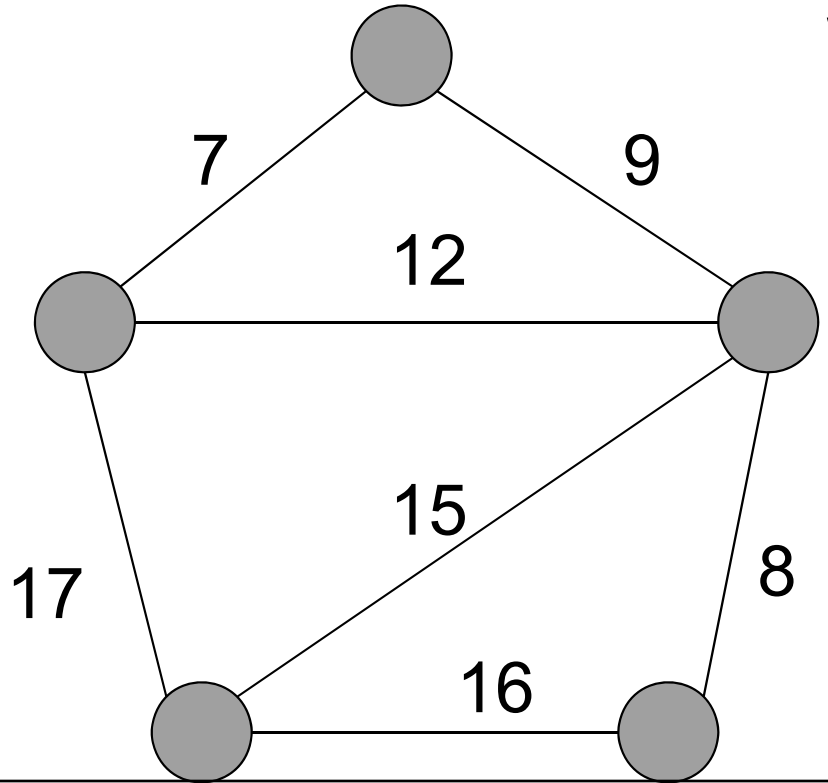
アルゴリズムの動き



クラスカルのアルゴリズム(p.11)

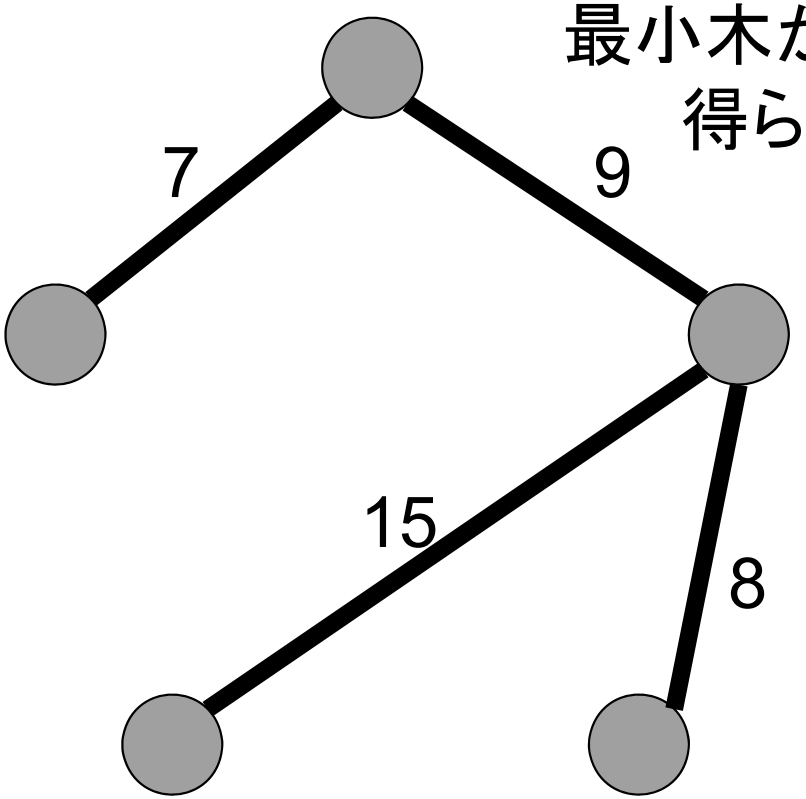
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



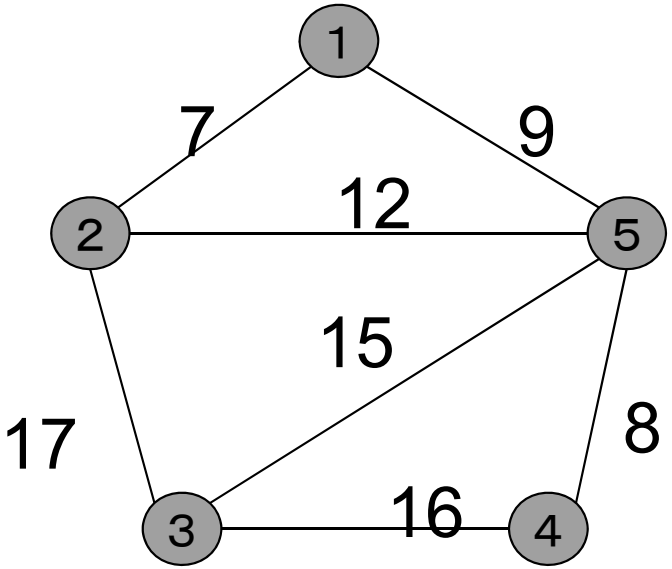
アルゴリズムの動き

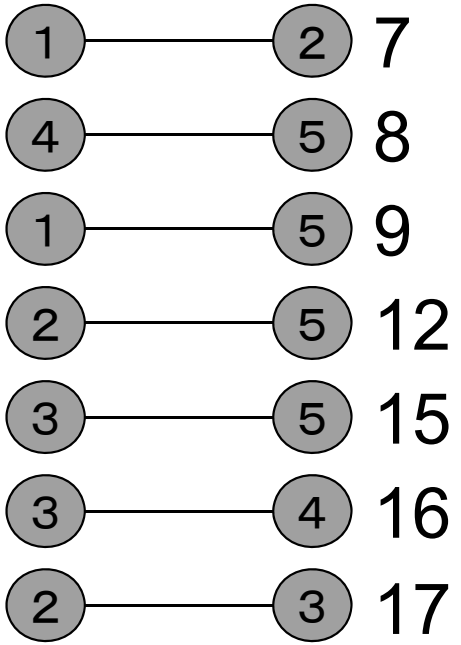
アルゴリズム終了
最小木が
得られた



クラスカルのアルゴリズムの計算時間

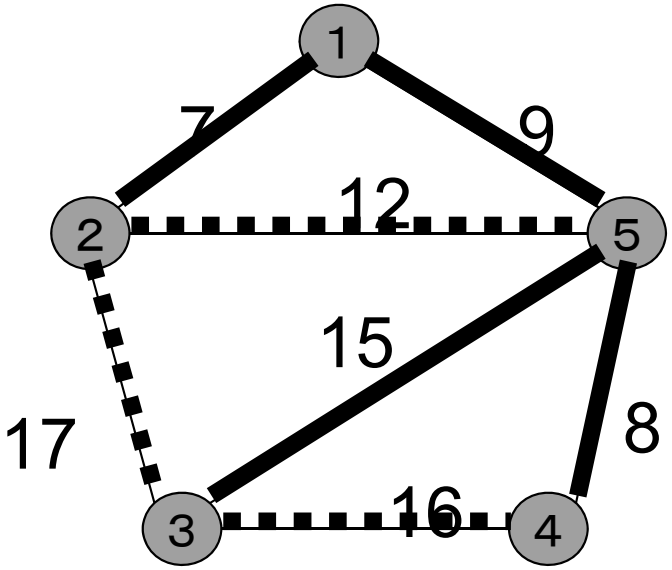
- アルゴリズムの実装方法
  - □枝を長さの短い順にソート --- O(m log m) = O(m log n)





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}

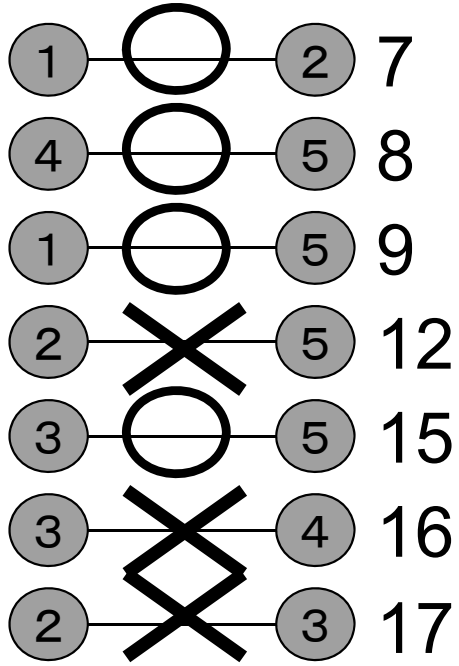
{1, 2}, {4, 5}, {3}

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

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

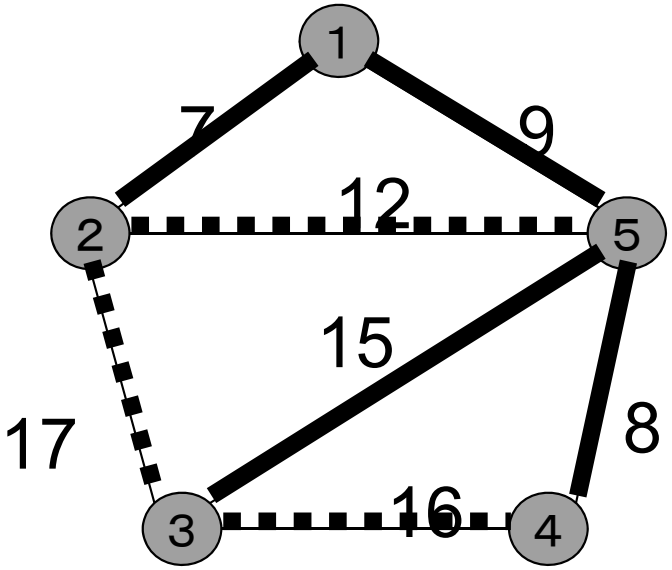
{1, 2, 4, 5, 3}

{1, 2, 4, 5, 3}



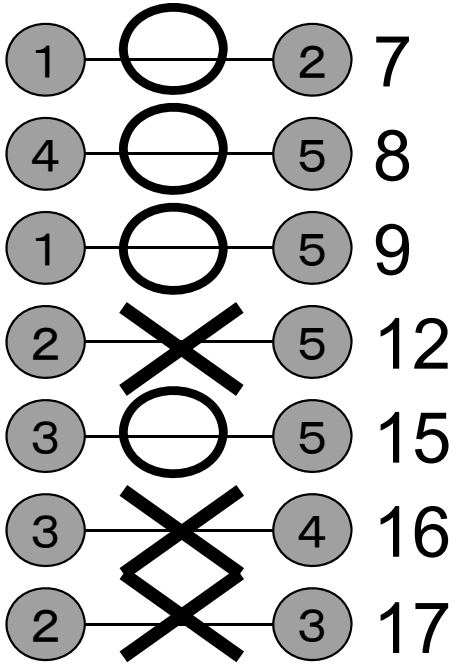
クラスカルのアルゴリズムの計算時間

- 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



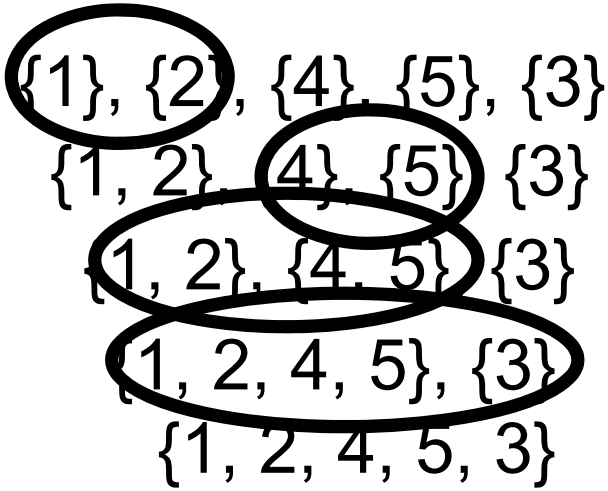
注意！
枝を加える度に
連結成分は
変化する

集合族の併合のためのデータ構造を利用する



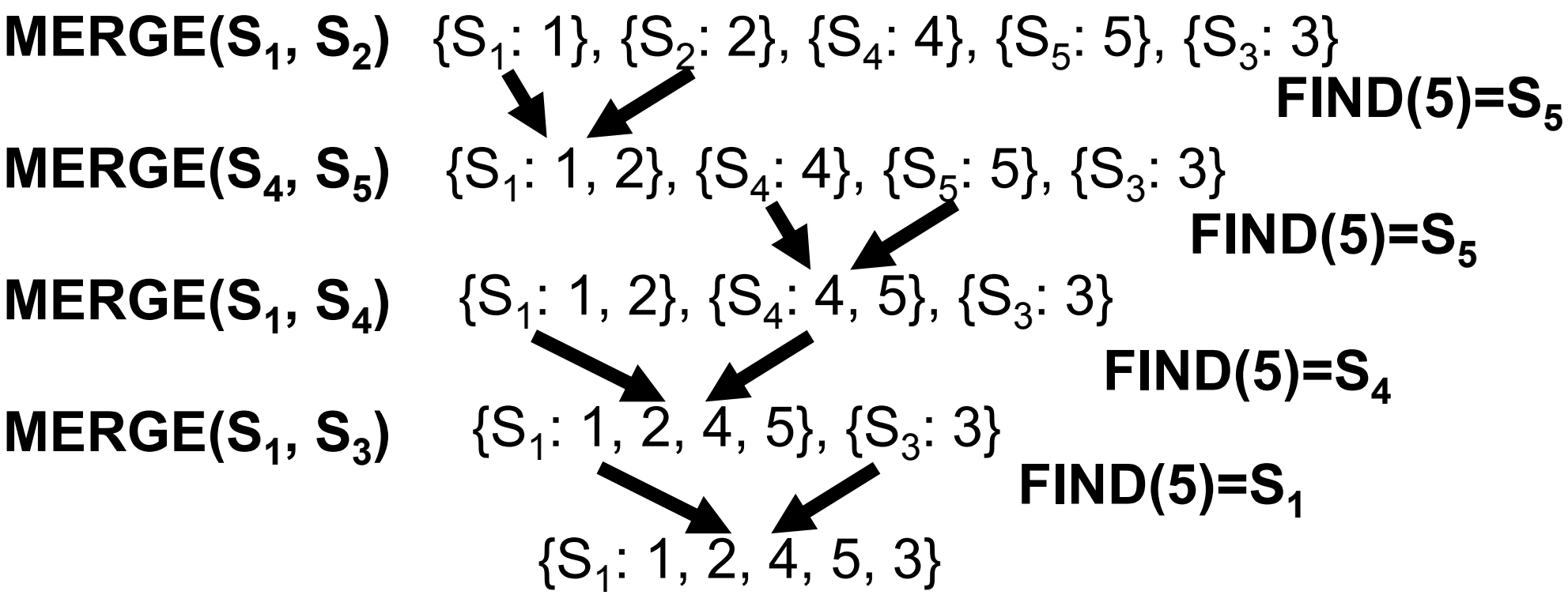
集合族の併合のためのデータ構造

- 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



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)

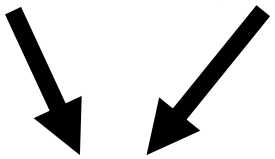
    ∴ 2m 回繰り返す（m=グラフの枝数）

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

- 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

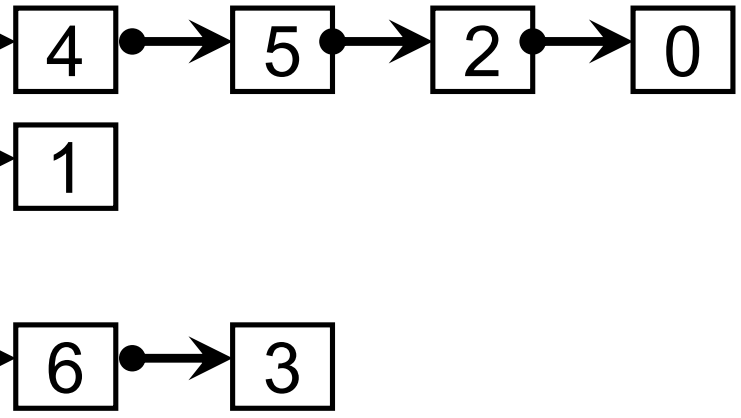
集合族の併合のデータ構造：配列＋リストによる実現

- 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

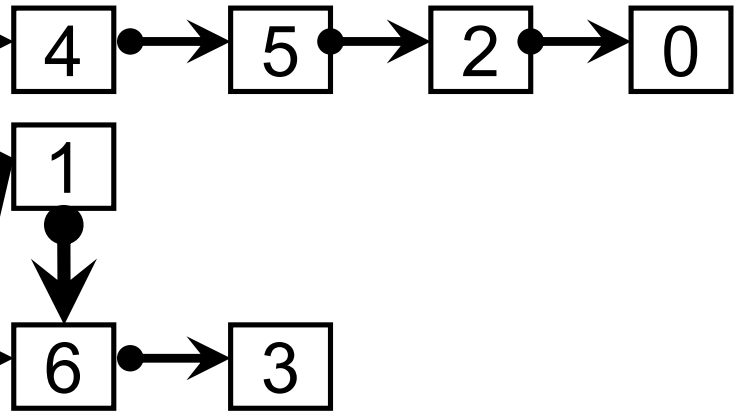
$\{S_0: 0, 2, 4, 5\}, \{S_1: 1\}, \{S_3: 3, 6\}$

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：
$O(|S_1|)$

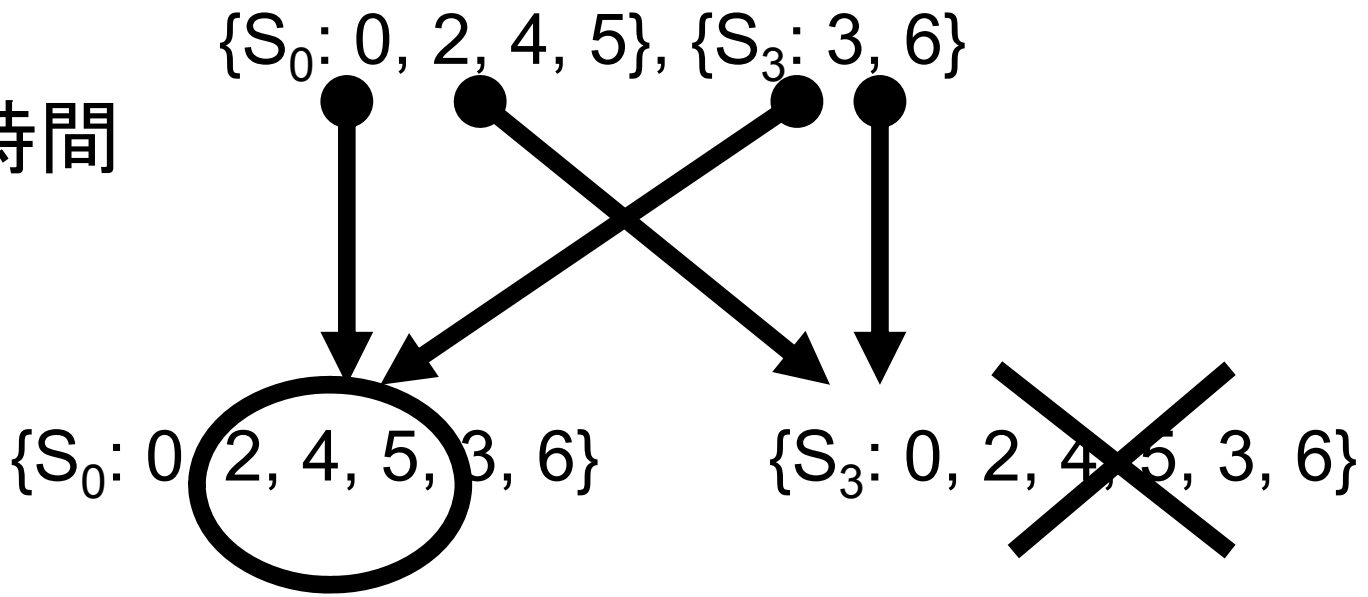
集合の名前	集合の大きさ	先頭要素へのポインタ
S_0	4	
S_1	0	NULL
S_2	0	NULL
S_3	3	

(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

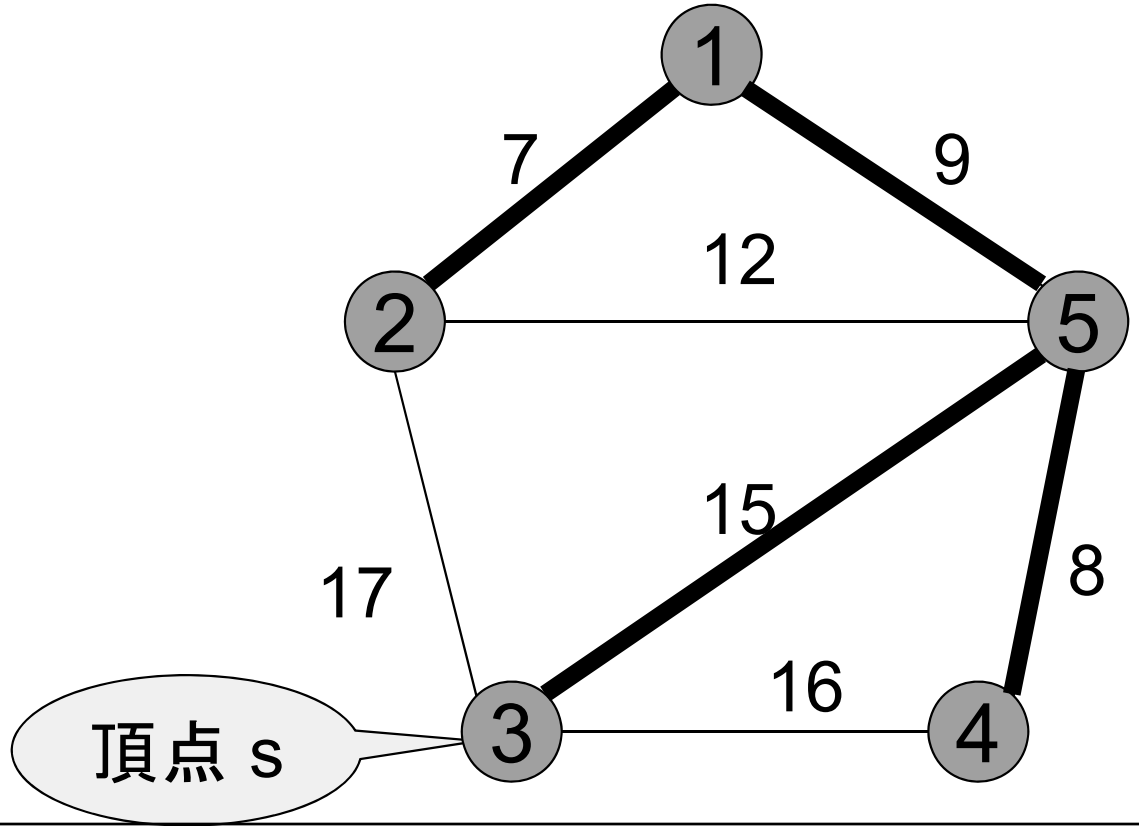
      FINDの回数：2m回 --- O(m)時間

      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)

プリムのアルゴリズム

- アルゴリズムの大まかな流れ
  - 最初は一つの頂点 s からスタート
  - 次々に枝を加え，ひとつの木を大きくしていく

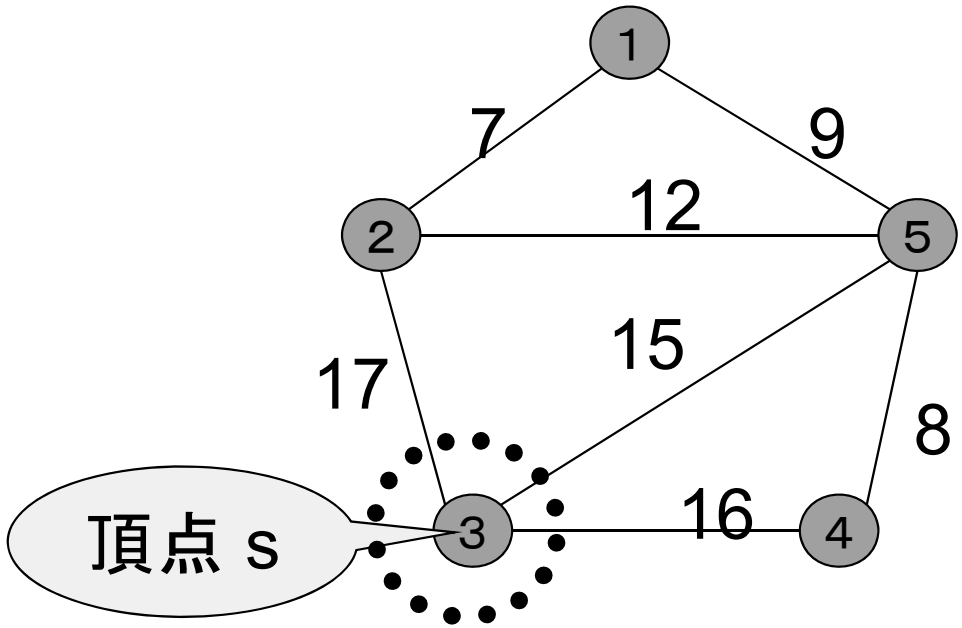


プリムのアルゴリズムの手順

- アルゴリズムの具体的な手順

  Step 0: 頂点 s を適当に選び，最初の木 T は頂点 s のみからなる，枝のない木とする

$T = \emptyset$



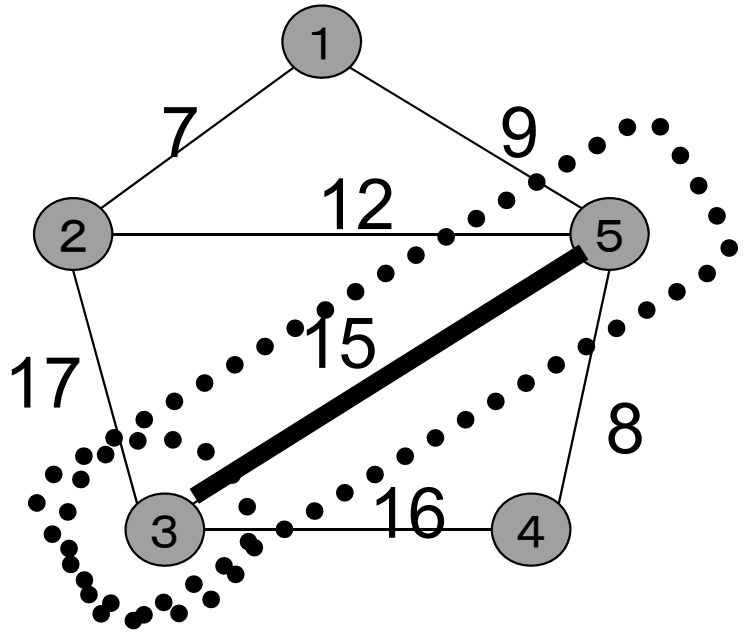
プリムのアルゴリズムの手順

- アルゴリズムの具体的な手順

  Step 1: 現在の木 T と，木の外側を結ぶ枝の中で
  　　長さ最小のものを選び，Tに加える.

  Step 2: 木 T が全域木ならば終了． そうでない場合はStep 1へ.

$T = \{(3,5)\}$



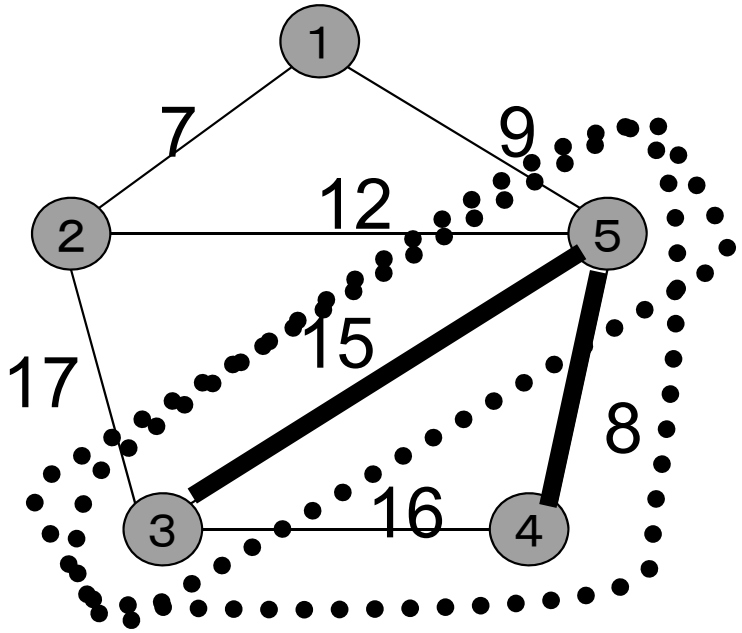
プリムのアルゴリズムの手順

- アルゴリズムの具体的な手順

  Step 1: 現在の木 T と，木の外側を結ぶ枝の中で
  　　長さ最小のものを選び，Tに加える.

  Step 2: 木 T が全域木ならば終了． そうでない場合はStep 1へ.

$T = \{(3,5), (5,4)\}$

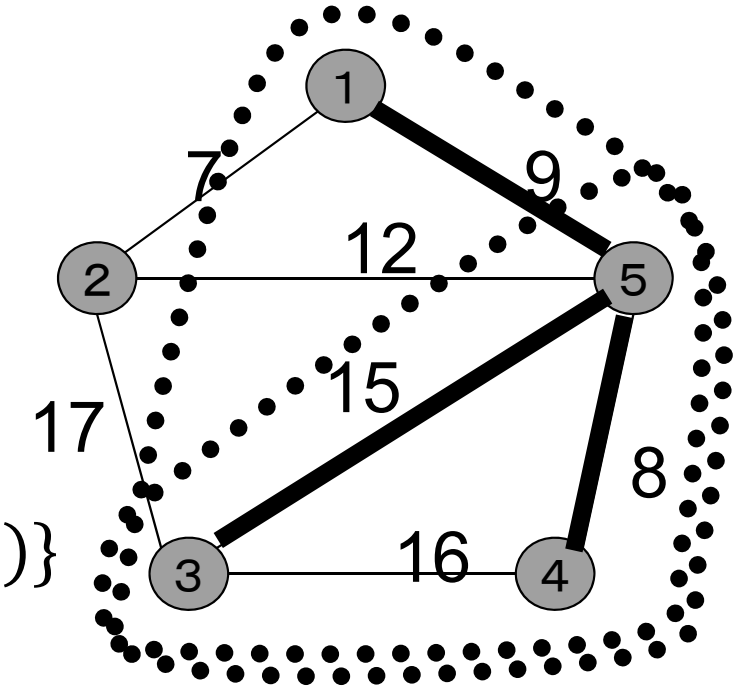


プリムのアルゴリズムの手順

- アルゴリズムの具体的な手順

  Step 1: 現在の木 T と，木の外側を結ぶ枝の中で
  　　長さ最小のものを選び，Tに加える.

  Step 2: 木 T が全域木ならば終了． そうでない場合はStep 1へ.



$T = \{(3,5), (5,4), (5,1)\}$

プリムのアルゴリズムの手順

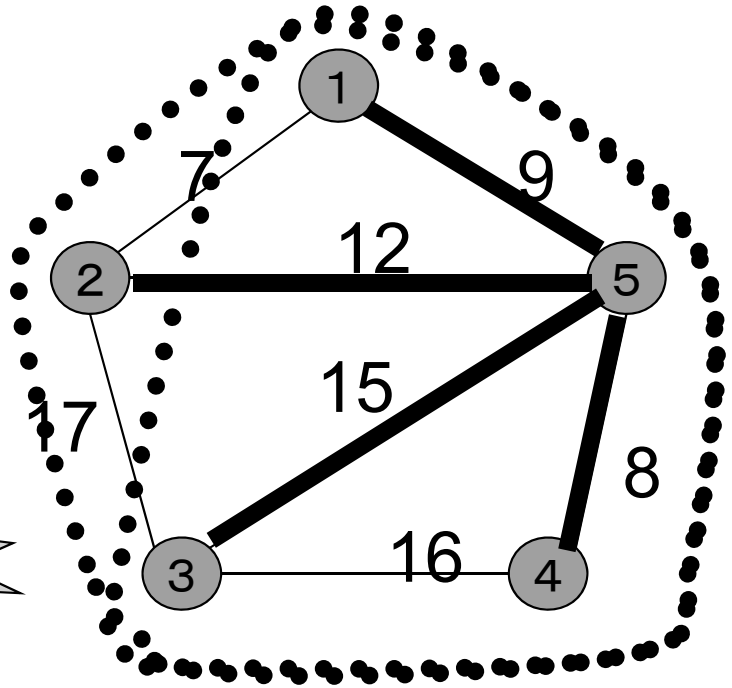
- アルゴリズムの具体的な手順

  Step 1: 現在の木 T と，木の外側を結ぶ枝の中で
  　　長さ最小のものを選び，Tに加える.

  Step 2: 木 T が全域木ならば終了．そうでない場合はStep 1へ.

$T = \{(3,5), (5,4), (5,1), (1,2)\}$

終了

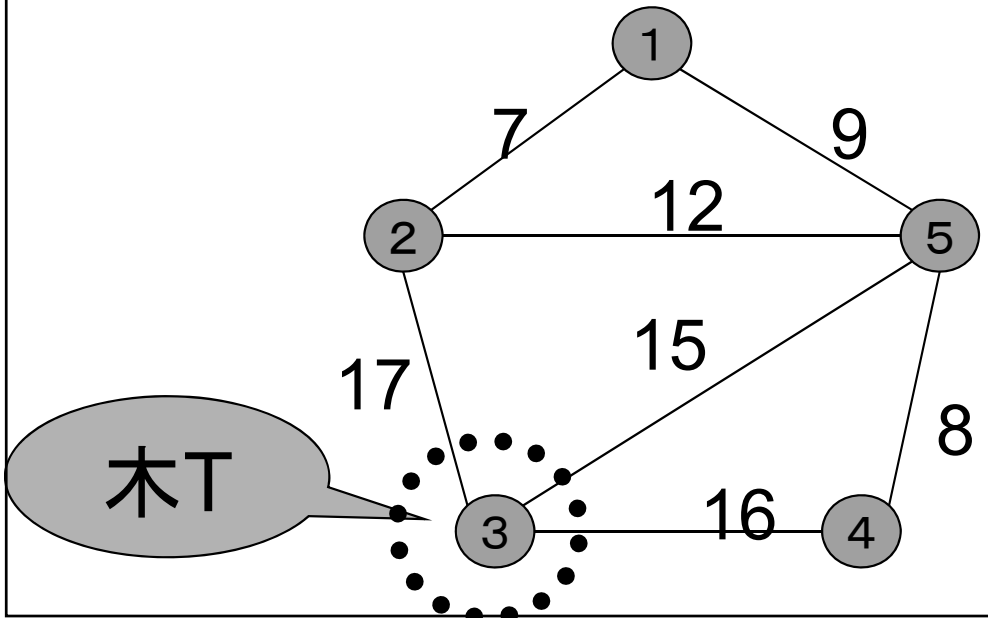


注意：反復回数は必ず n-1 回で終了
➔ T は全域木

プリムのアルゴリズムの実装方法

- Step 1: 現在の木 T と，木の外側を結ぶ枝の中で
  長さ最小のものを選び，Tに加える.

  どうやって実現するか ---- ヒープを利用する
  - ヒープに格納する要素：木に含まれない頂点 v
  - 各要素のデータ：頂点 vから木 T に接続する枝の最小の長さ
    （及び対応する枝）



長さ最小

ヒープの中の要素

頂点 1, 最小長さ∞ (枝なし)

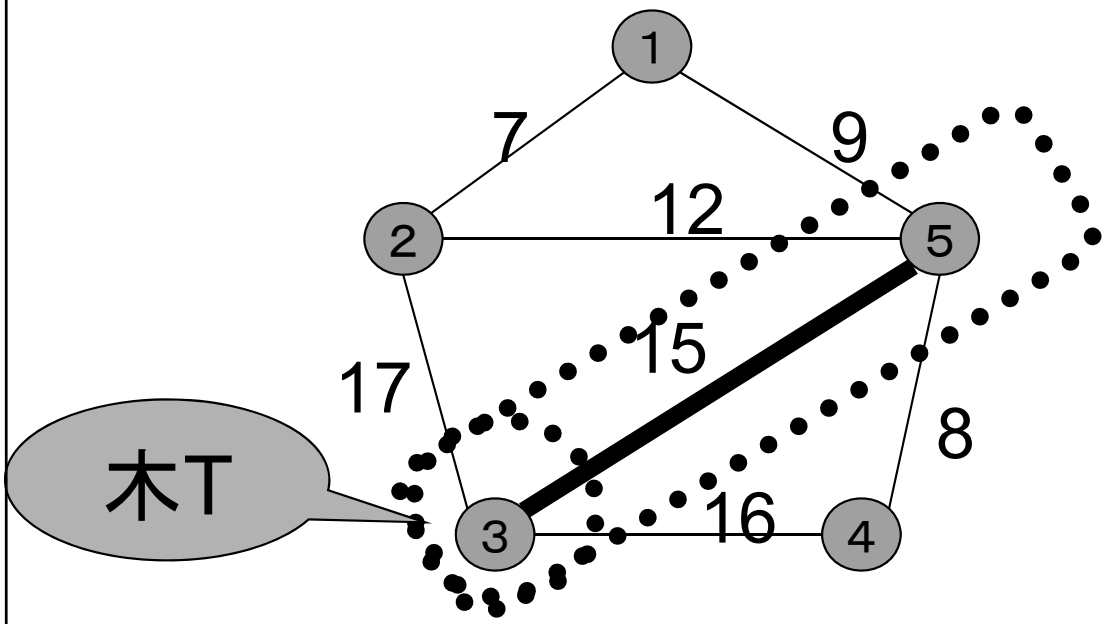
頂点 2, 最小長さ17 (枝(2,3))

頂点 4, 最小長さ16 (枝(4,3))

頂点 5, 最小長さ15 (枝(5,3))

プリムのアルゴリズムの実装方法

- 枝が追加されたときのヒープの更新方法
  - Tに追加した頂点をヒープから削除



ヒープの中の要素

頂点 1, 最小長さ∞ (枝なし)

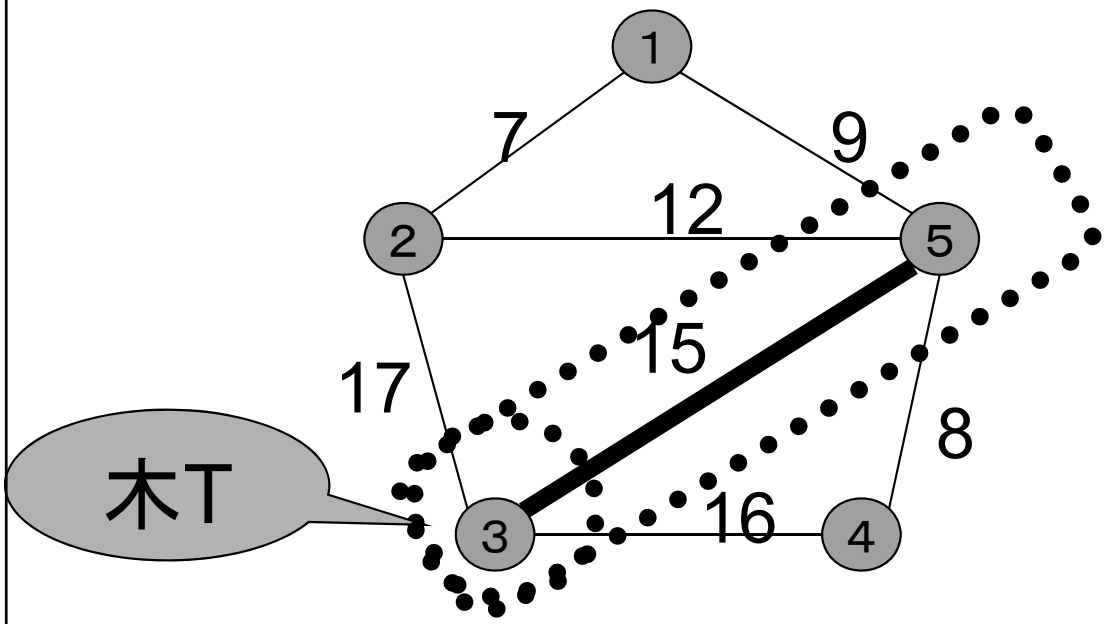
頂点 2, 最小長さ17 (枝(2,3))

頂点 4, 最小長さ16 (枝(4,3))

頂点 5, 最小長さ15 (枝(5,3))

プリムのアルゴリズムの実装方法

- 枝が追加されたときのヒープの更新方法
  - □ Tに追加した頂点をヒープから削除
  - □ Tに追加した頂点に接続する枝を使って，ヒープの各要素の最小長さを(必要があれば)更新
    ※最小長さが更新されるとき，必ず減少



ヒープの中の要素

頂点 1, 最小長さ∞ (枝なし)
最小長さ9 (枝(1,5))
頂点 2, 最小長さ17 (枝(2,3))
最小長さ12 (枝(2,5))
頂点 4, 最小長さ16 (枝(4,3))
最小長さ8 (枝(4,5))

プリムのアルゴリズムの実装方法

- ヒープの維持に必要な計算時間
  - 木Tとその外側を結ぶ最小長さの枝を求める
    --- ヒープの中で最小の要素を求める➔ O(1)
  - 木Tに追加した頂点 v の削除
    --- ヒープの中で要素を一つ削除➔ O(log n)
  - ヒープの各要素の最小長さを更新
    --- ヒープの中の要素のデータを減少させる➔ O(log n)
    更新が必要な要素数≦追加した頂点 v の次数 d(v)
    ∴O(d(v) log n)
- アルゴリズム全体で O(m log n) 時間を要する

ヒープ

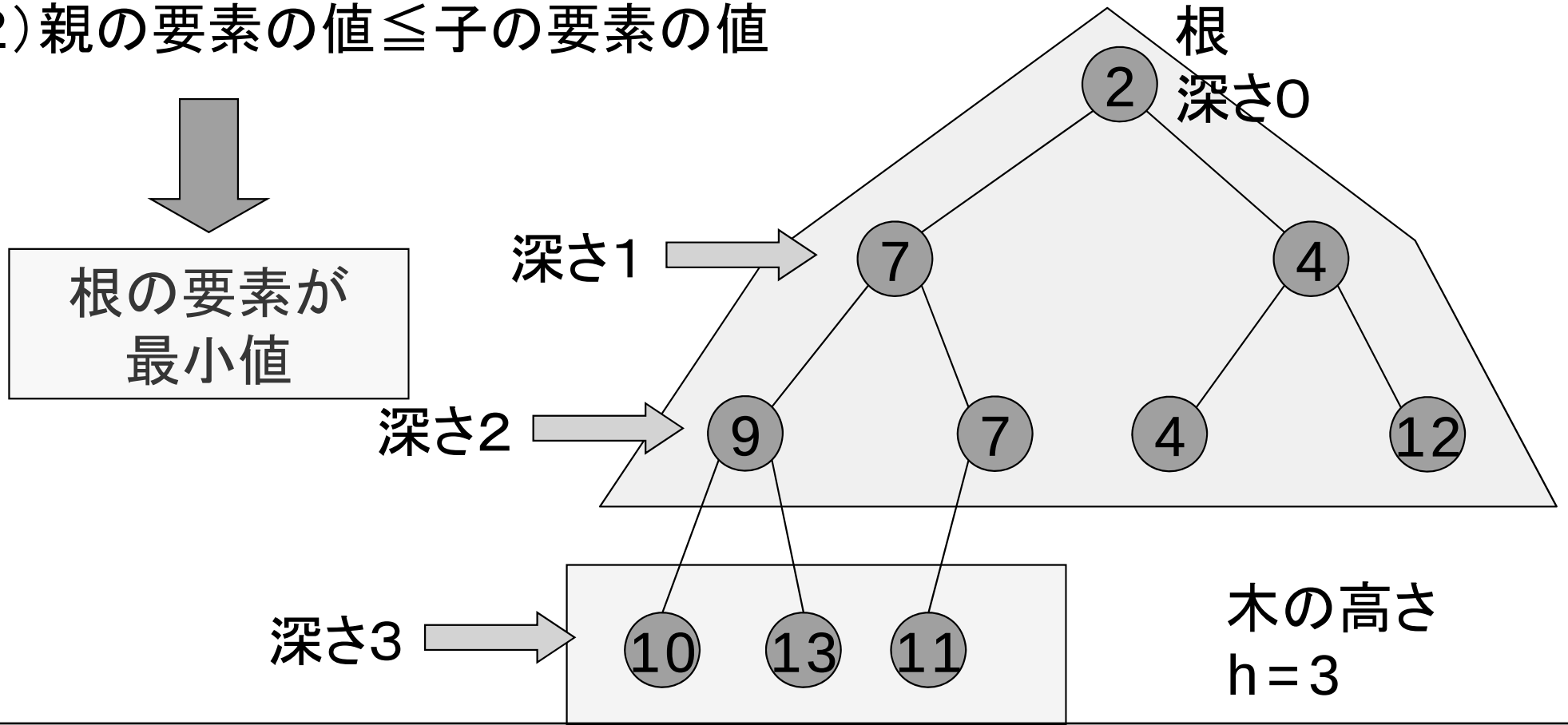
- 次の条件を満たす2分木をヒープと呼ぶ

（1）木の高さが h ➔ 深さ h-1 までは完全2分木

深さ h では，左側に葉が詰められている

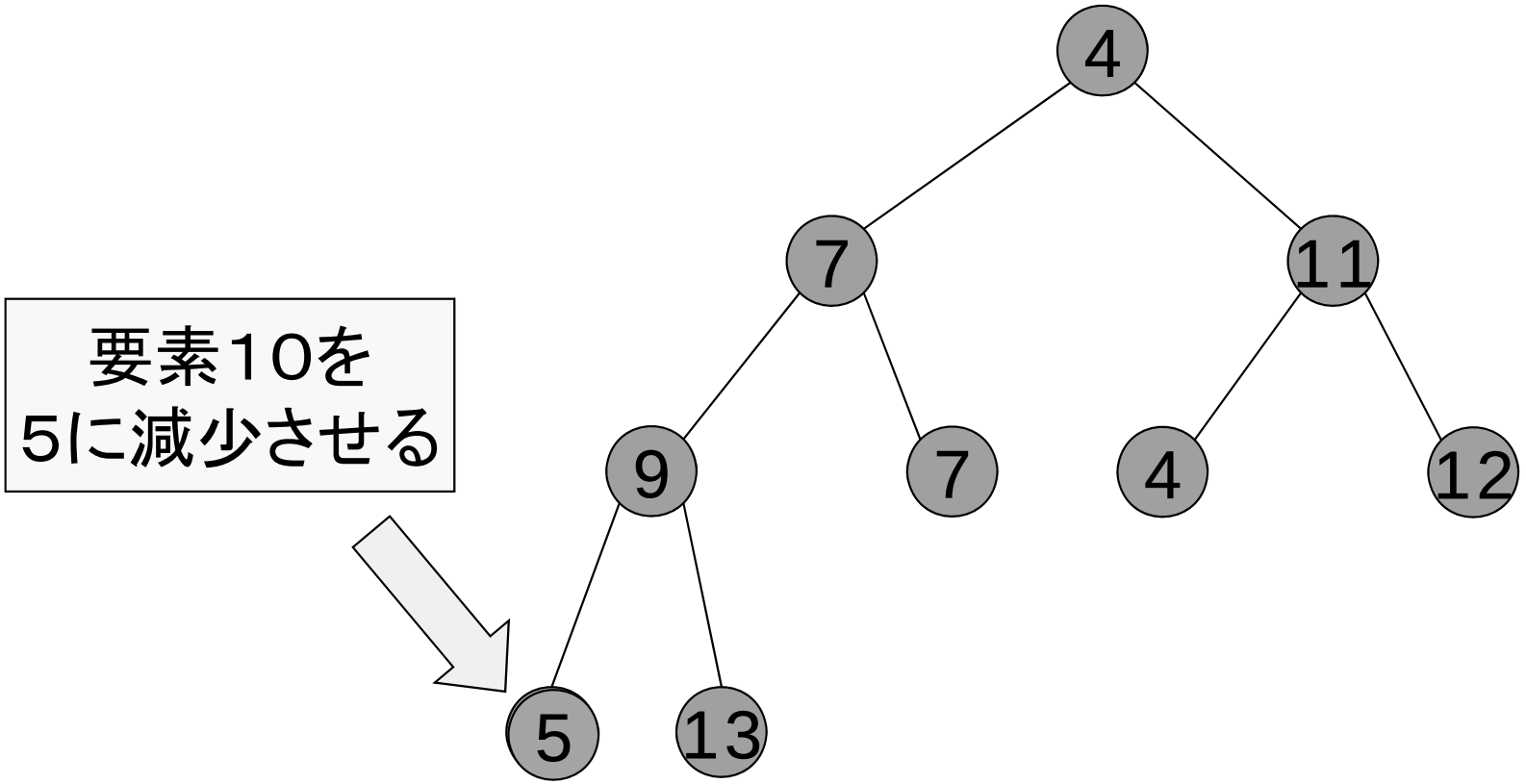
（2）親の要素の値≦子の要素の値

木の高さ
$= \lfloor \log_2 n \rfloor$



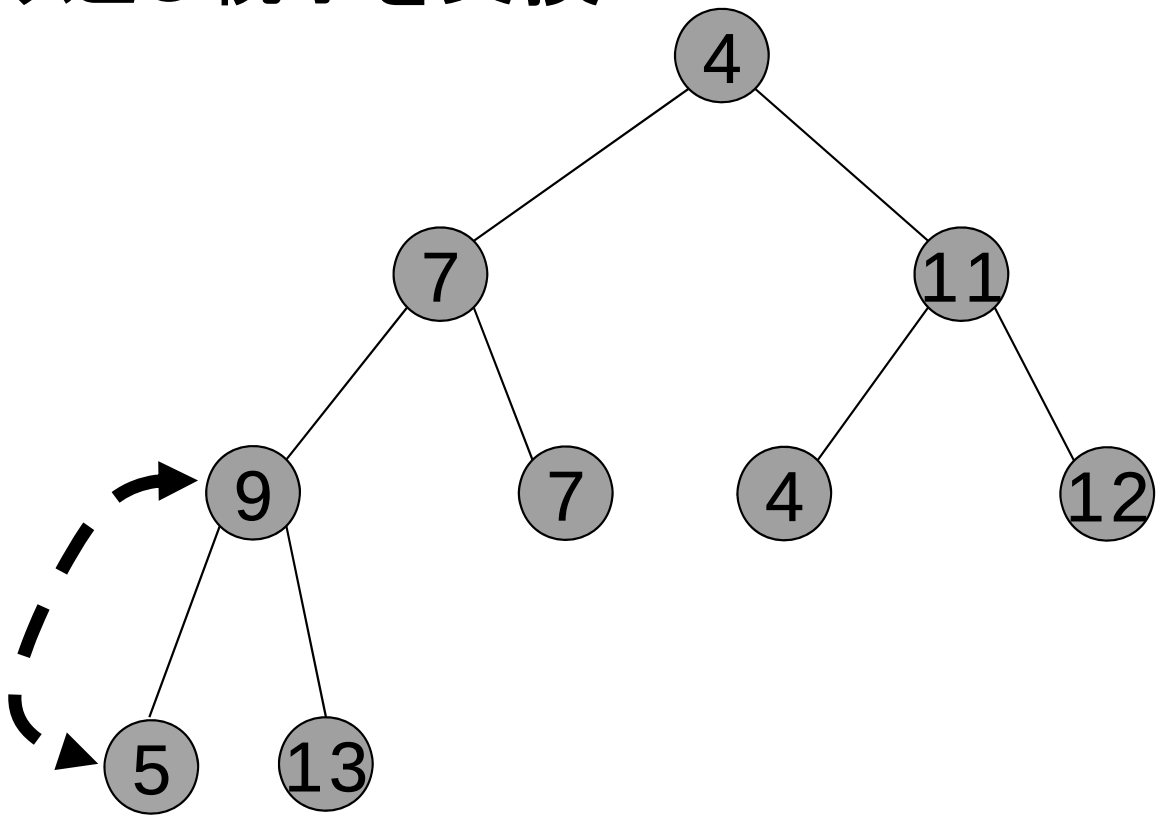
ある要素を減少させる

新しい要素を挿入したときと同じやり方で
ヒープを更新する



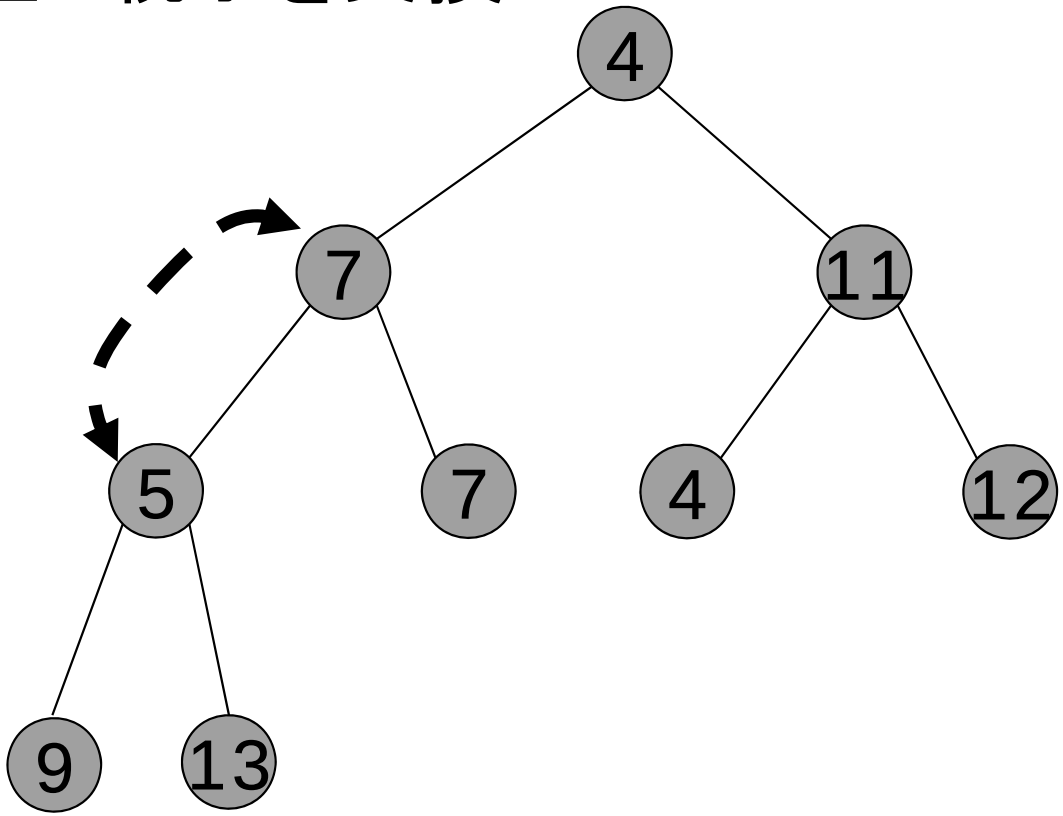
ある要素を減少させる

（1）減少させた要素を親と比較し，
「親の要素＞子の要素」が
成り立つときは，繰り返し親子を交換



ある要素を減少させる

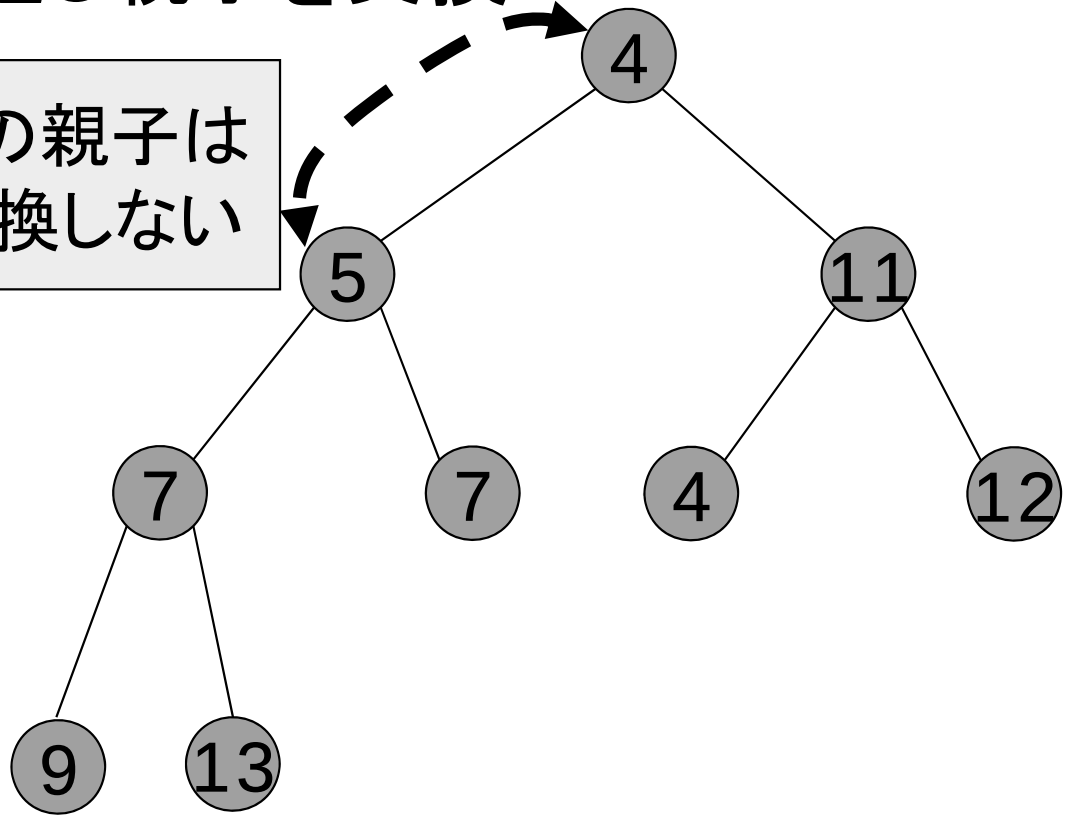
（1）減少させた要素を親と比較し，
「親の要素＞子の要素」が
成り立つときは，繰り返し親子を交換



ある要素を減少させる

（1）減少させた要素を親と比較し，
「親の要素＞子の要素」が
成り立つときは，繰り返し親子を交換

この親子は
交換しない



ステップ（1）が
行なわれる度に
新しい要素は
一つ上に上がる
➔反復回数≦木の高さ
➔時間計算量は$O(\log n)$