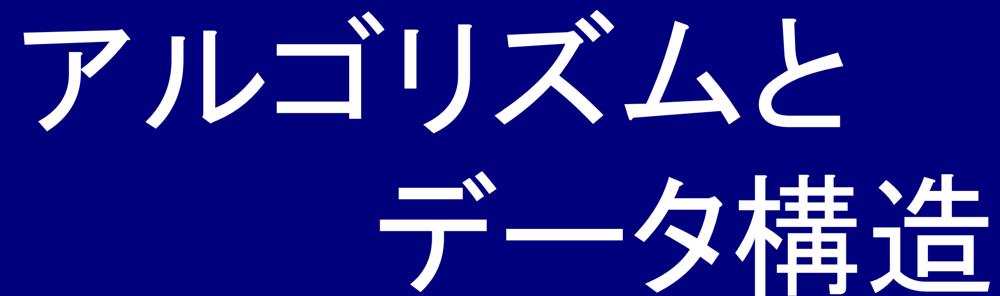


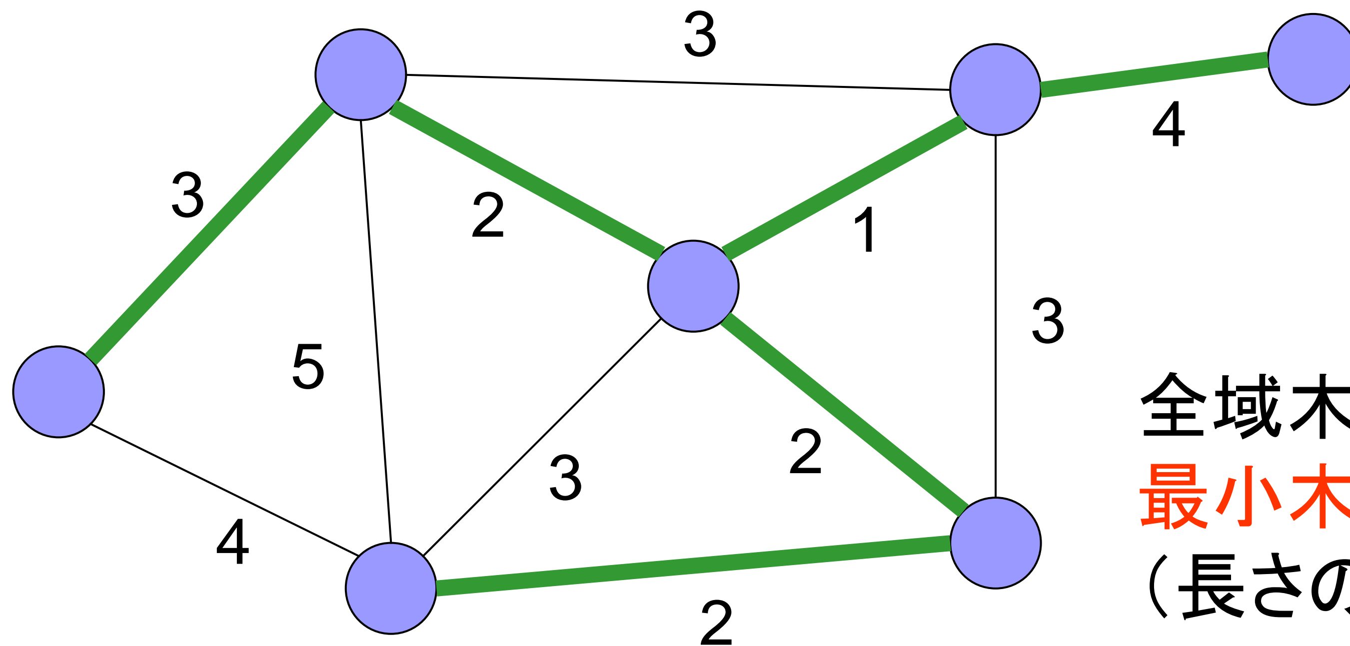
アルゴリズムとデータ構造

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

最小木問題

- 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)



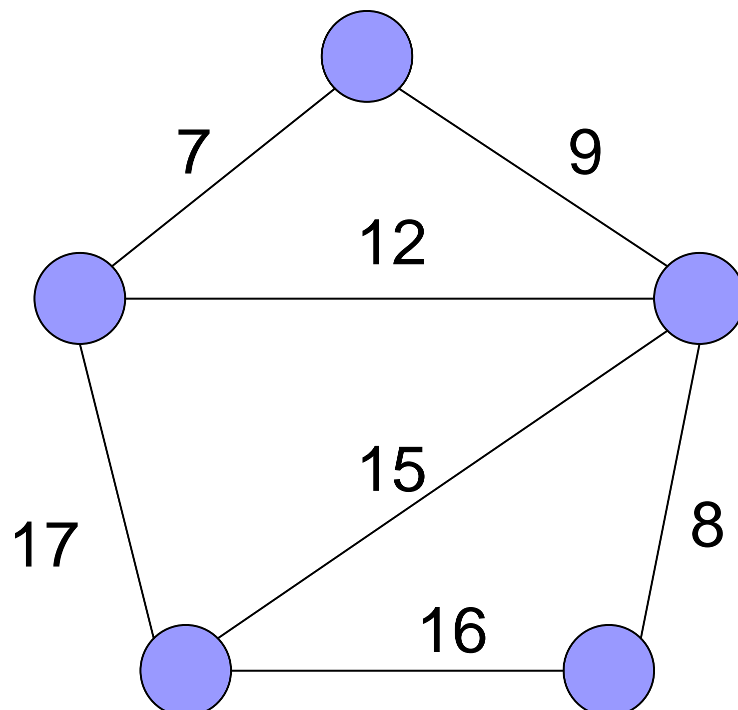
最小木を求めるアルゴリズム

- クラスカルのアルゴリズム（前回の講義）
  - 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（今日の講義）
  - 適当な節点 s を決める
  - 節点 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)

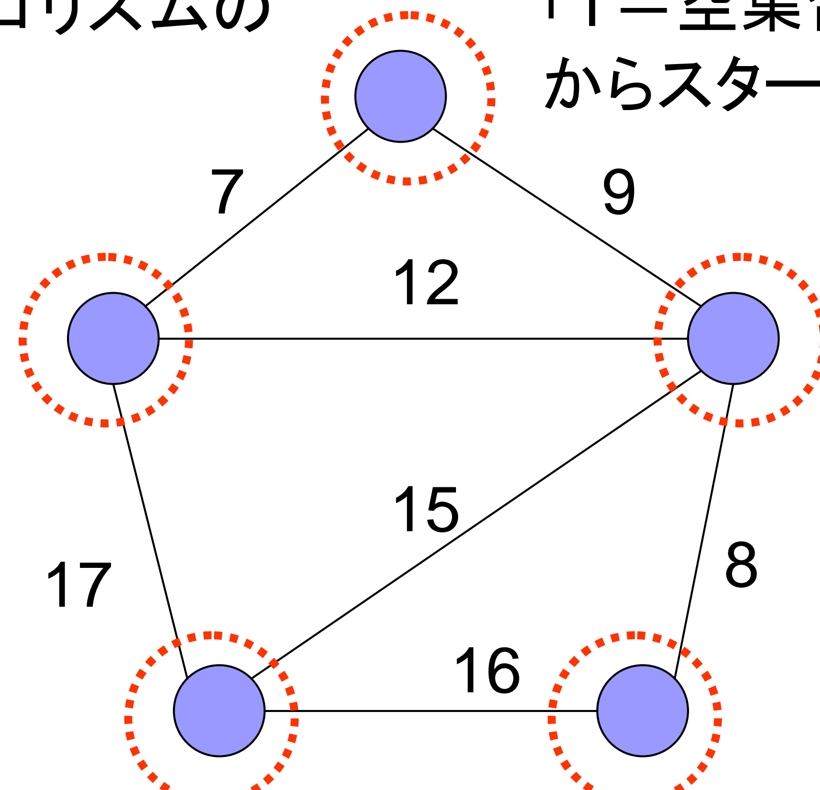
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

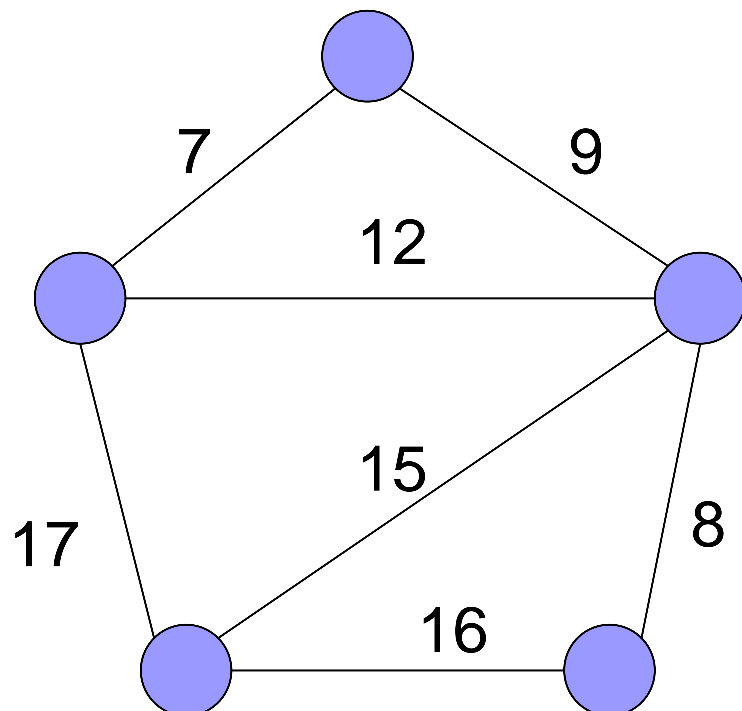
「T＝空集合」からスタート



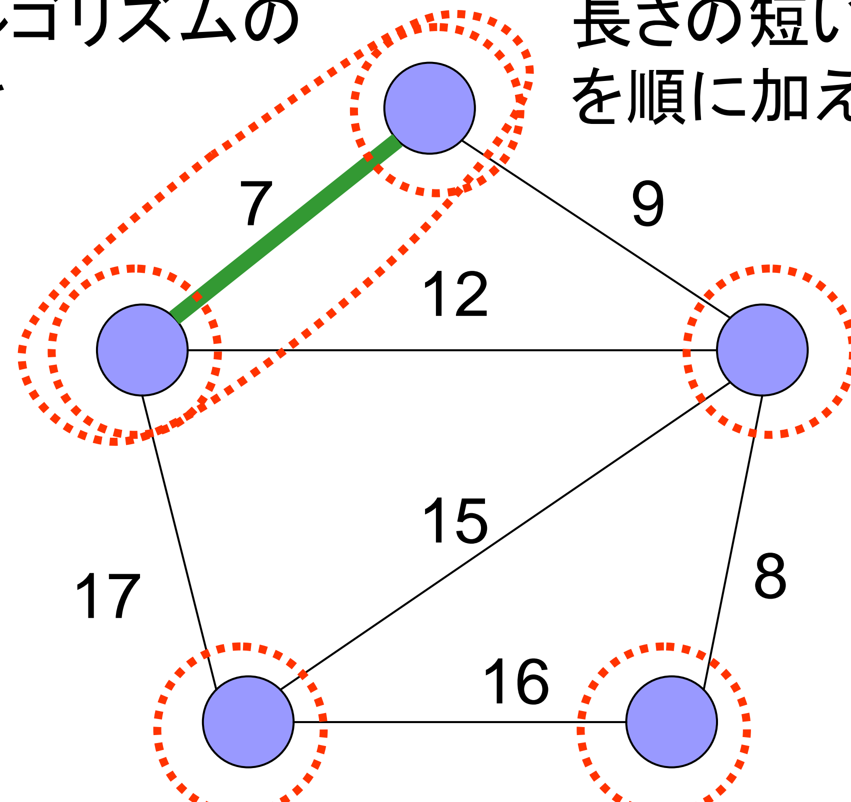
クラスカルのアルゴリズム(p.11)

- 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

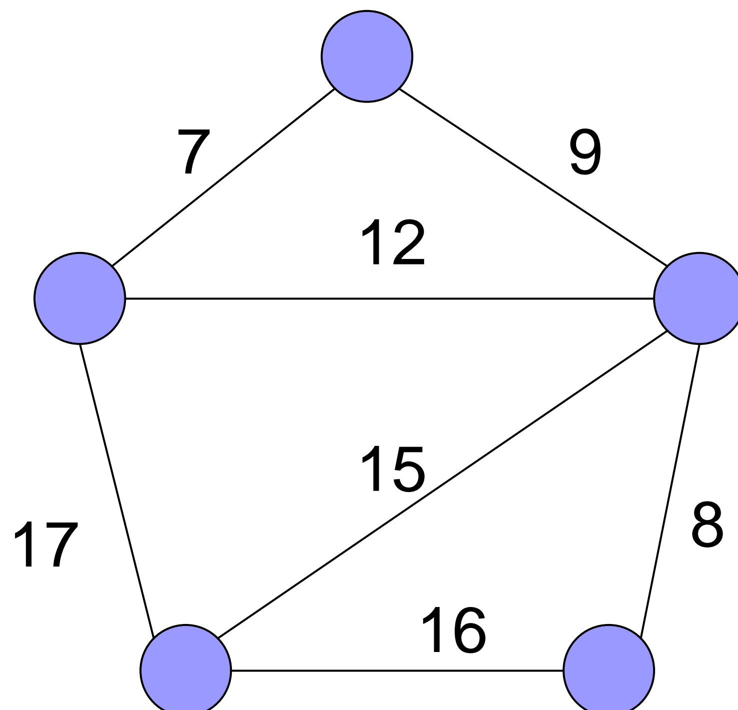


長さの短い枝を順に加える

クラスカルのアルゴリズム(p.11)

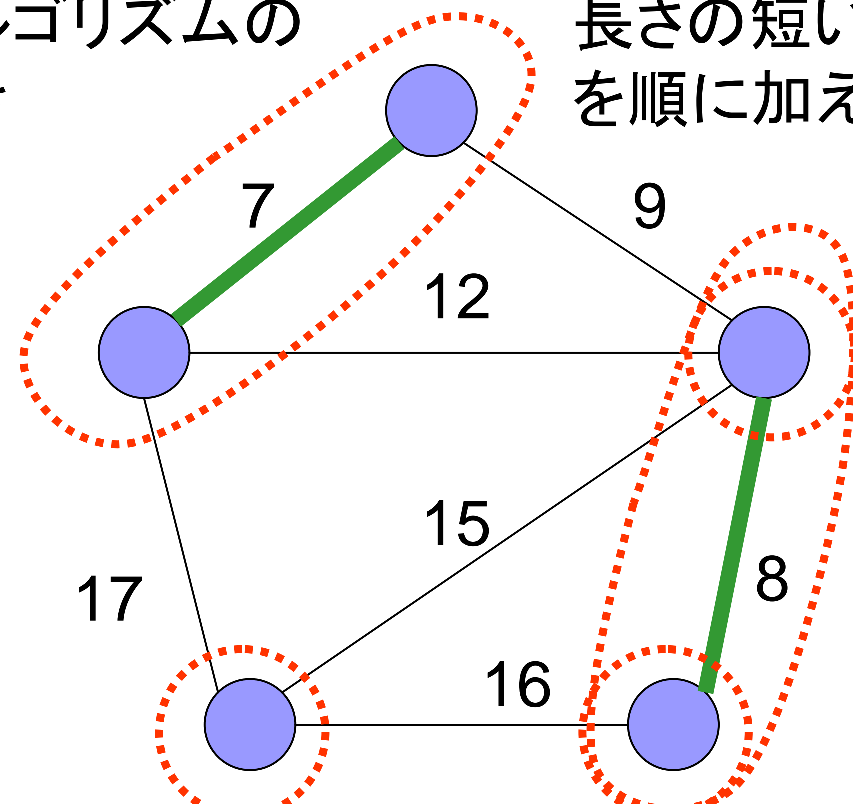
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

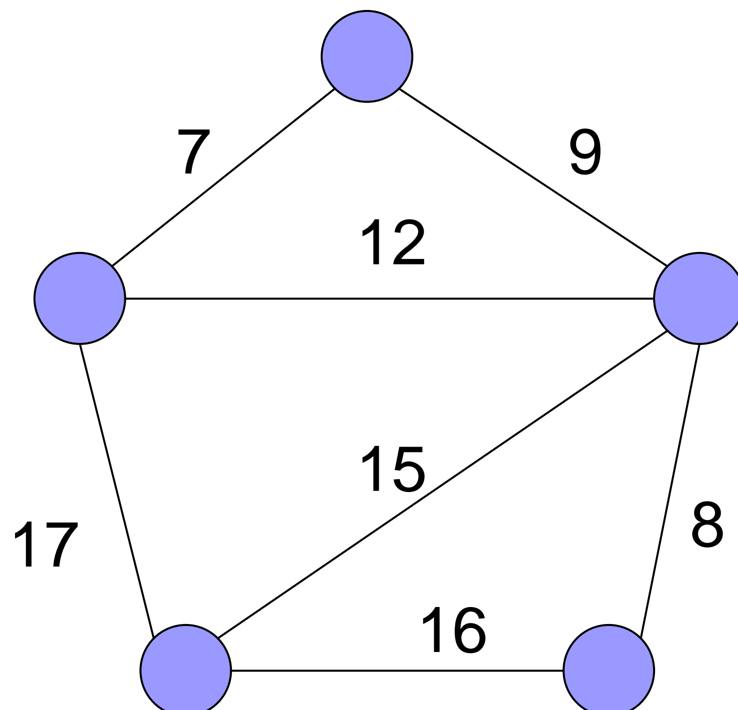
長さの短い枝を順に加える



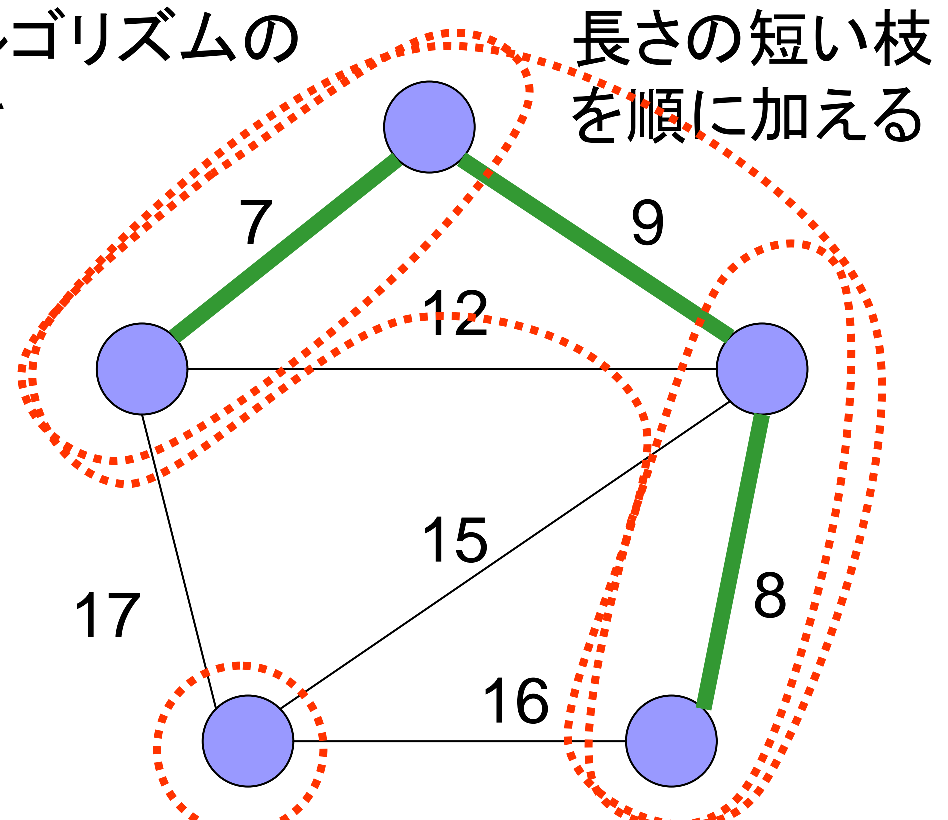
クラスカルのアルゴリズム(p.11)

- 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



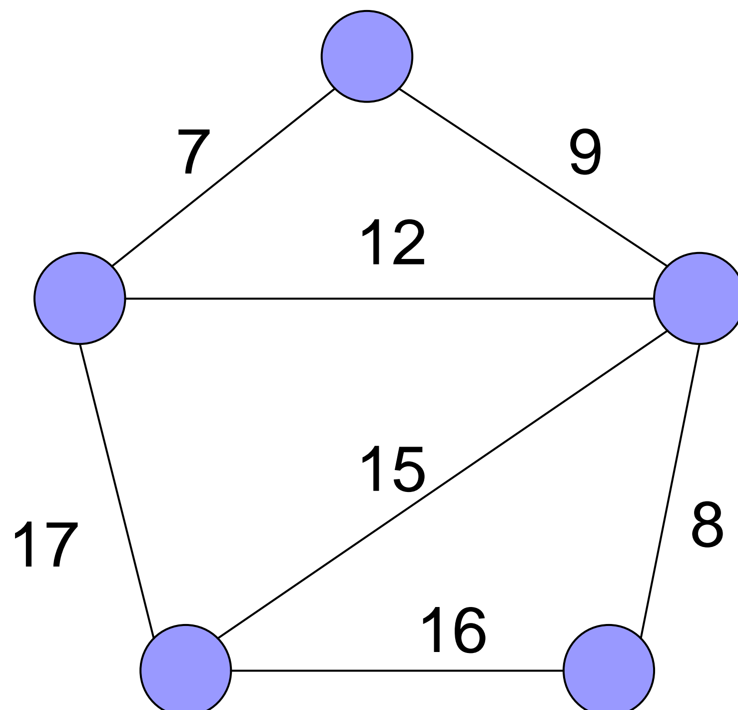
アルゴリズムの動き



クラスカルのアルゴリズム(p.11)

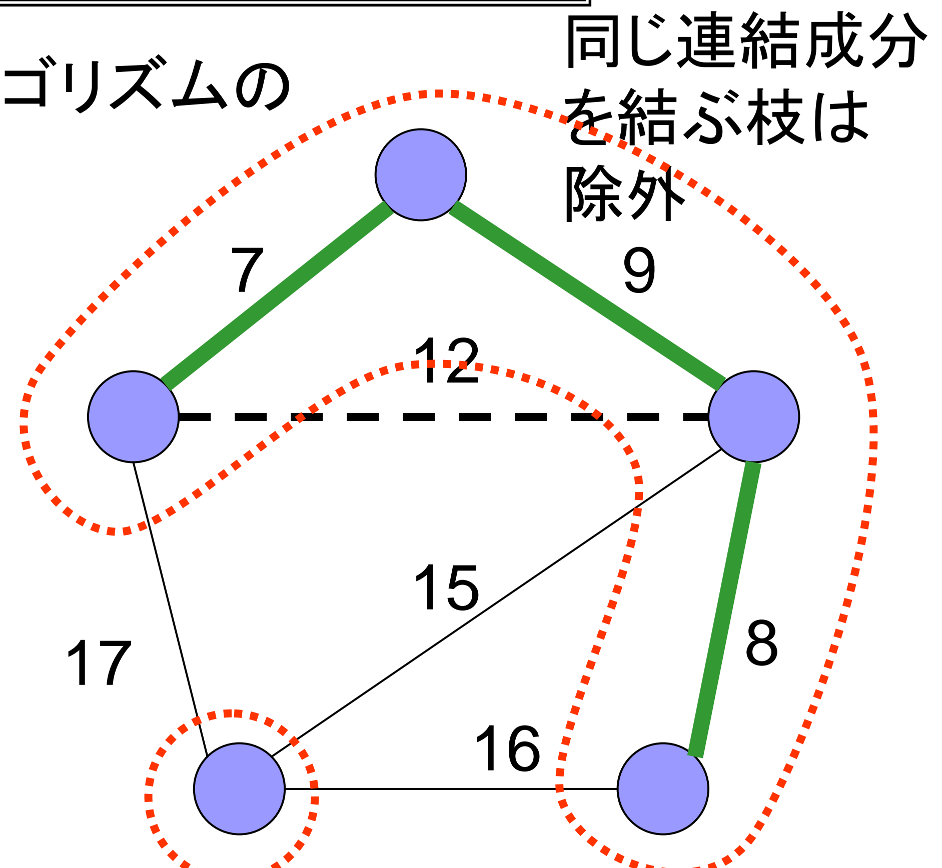
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

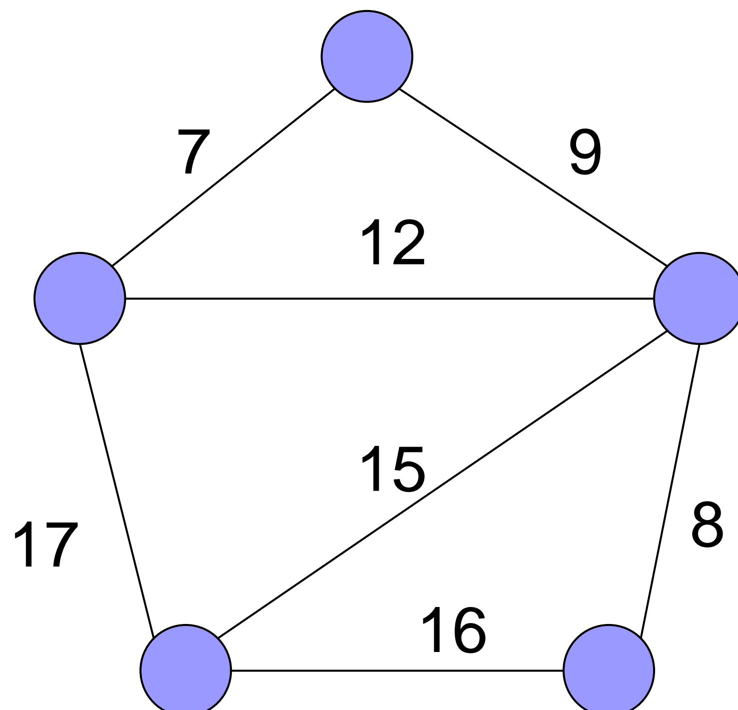
同じ連結成分を結ぶ枝は除外



クラスカルのアルゴリズム(p.11)

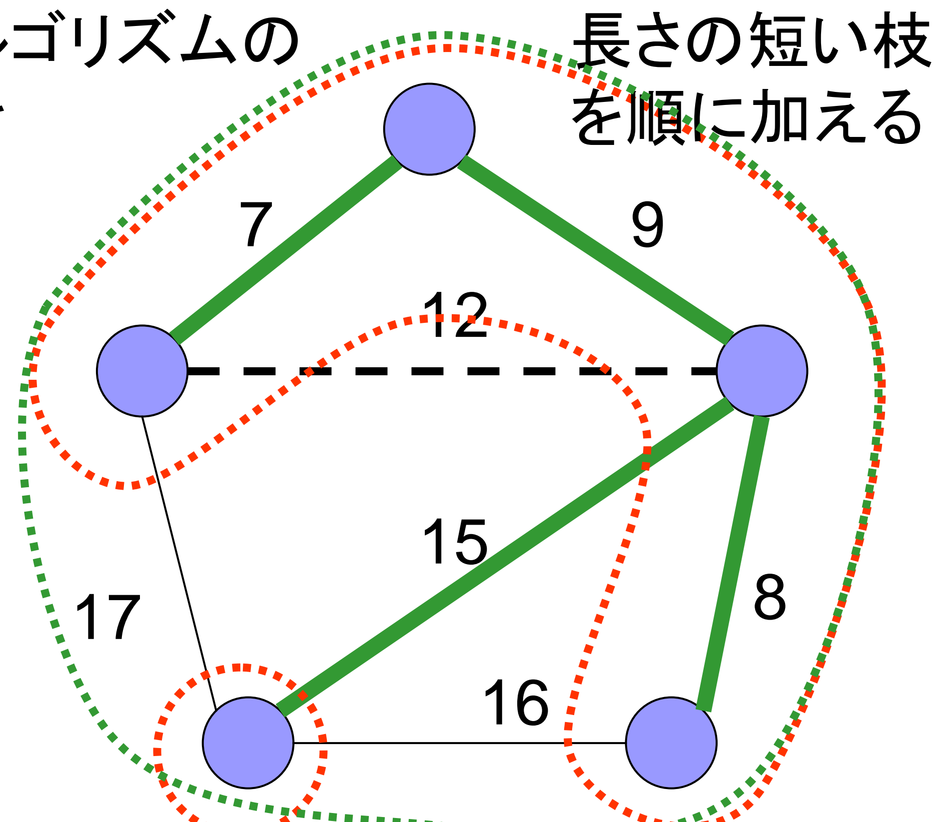
- 長さの短い順に枝を加える
- ただし，同じ連結成分を結ぶ枝は除外

入力の無向グラフ



アルゴリズムの動き

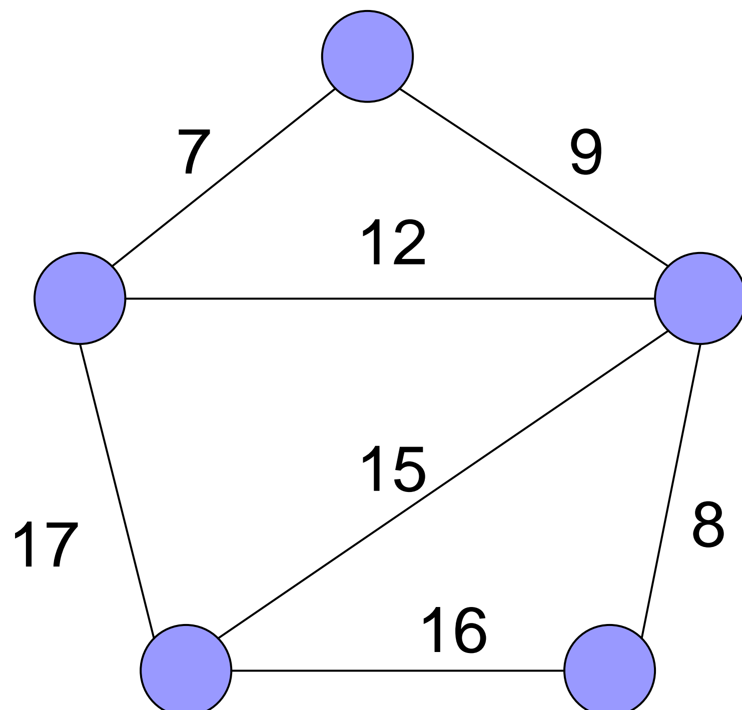
長さの短い枝を順に加える



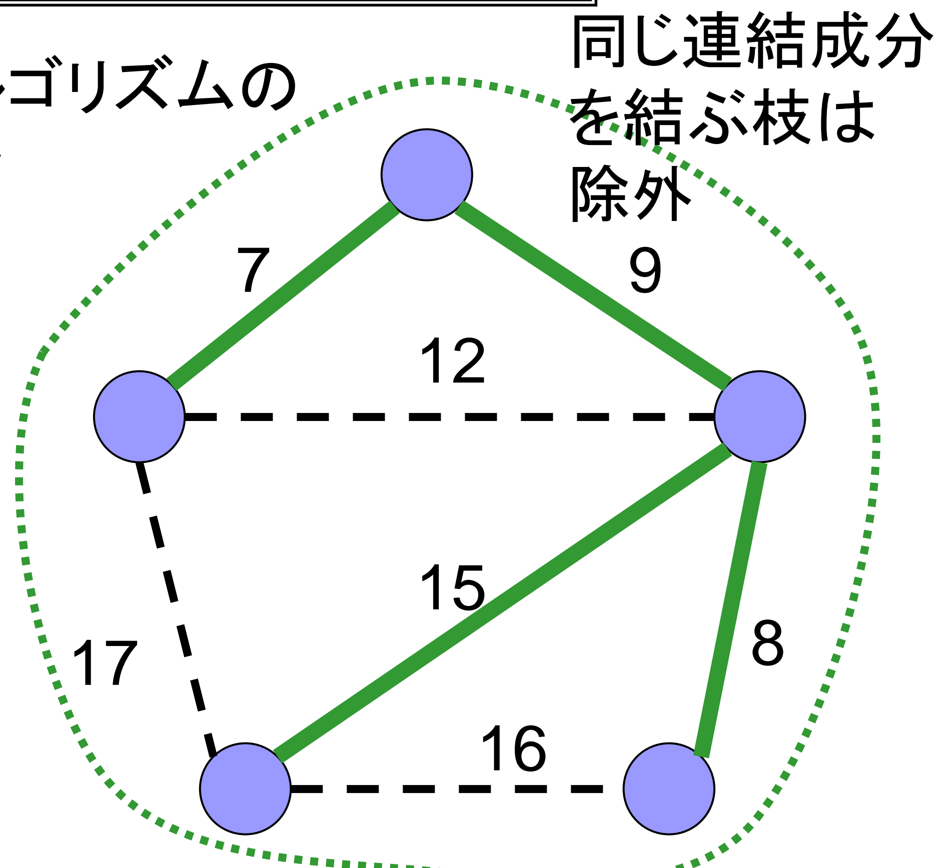
クラスカルのアルゴリズム(p.11)

- 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



集合族の併合：クラスカルのアルゴリズムの場合

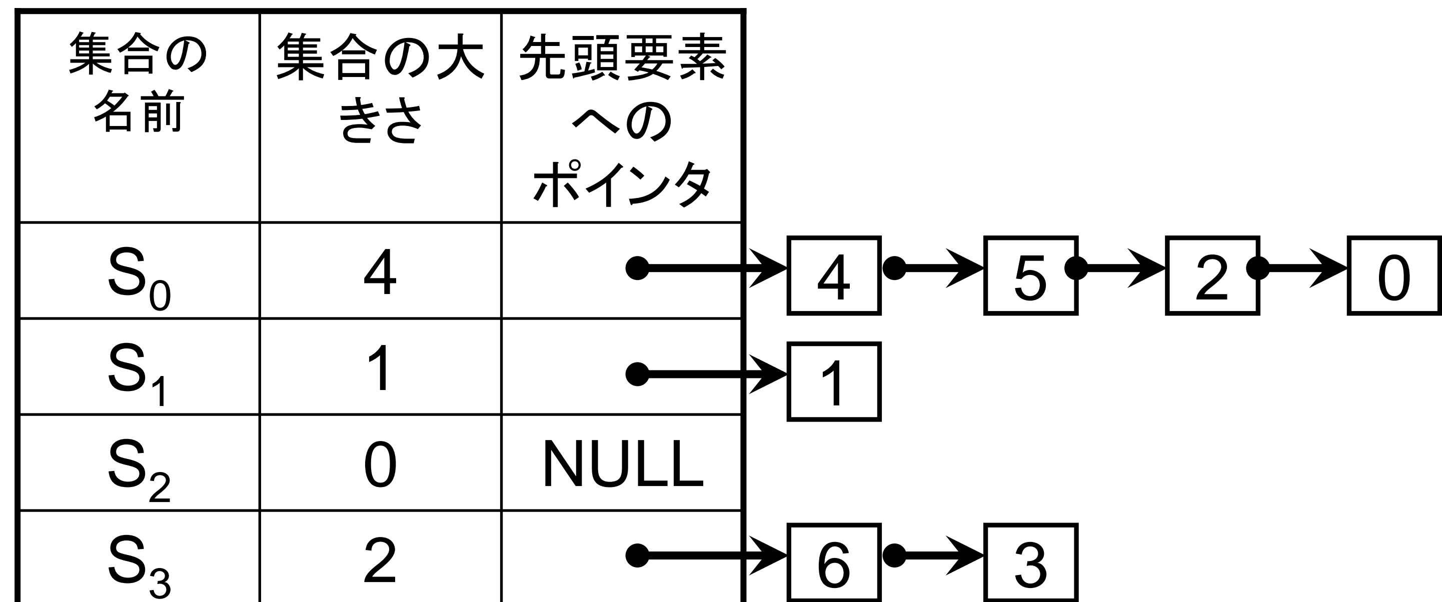
- 集合族 $S_1, S_2, \ldots, S_t$ に対する2つの操作
  - **MERGE(S_i, S_k)**: 集合S_iとS_kを併合，名前をS_iもしくはS_kとする
  - **FIND(x)**: 要素 x を含む集合の名前を返す
- クラスカルのアルゴリズムでは. . .
  - **MERGE(S_i, S_k)**: 枝を追加➔連結成分を併合
    ∴ n-1 回繰り返す（n =グラフの節点数）
  - **FIND(x)**: 現在の枝 (u, v) に対し，
    両端点が同じ連結成分に含まれる⇔ FIND(u) = FIND(v)
    ∴ 2m 回繰り返す（m=グラフの枝数）

集合族の併合のデータ構造:配列+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	→ 4 → 5 → 2 → 0
S_1	1	→ 1
S_2	0	NULL
S_3	2	→ 6 → 3

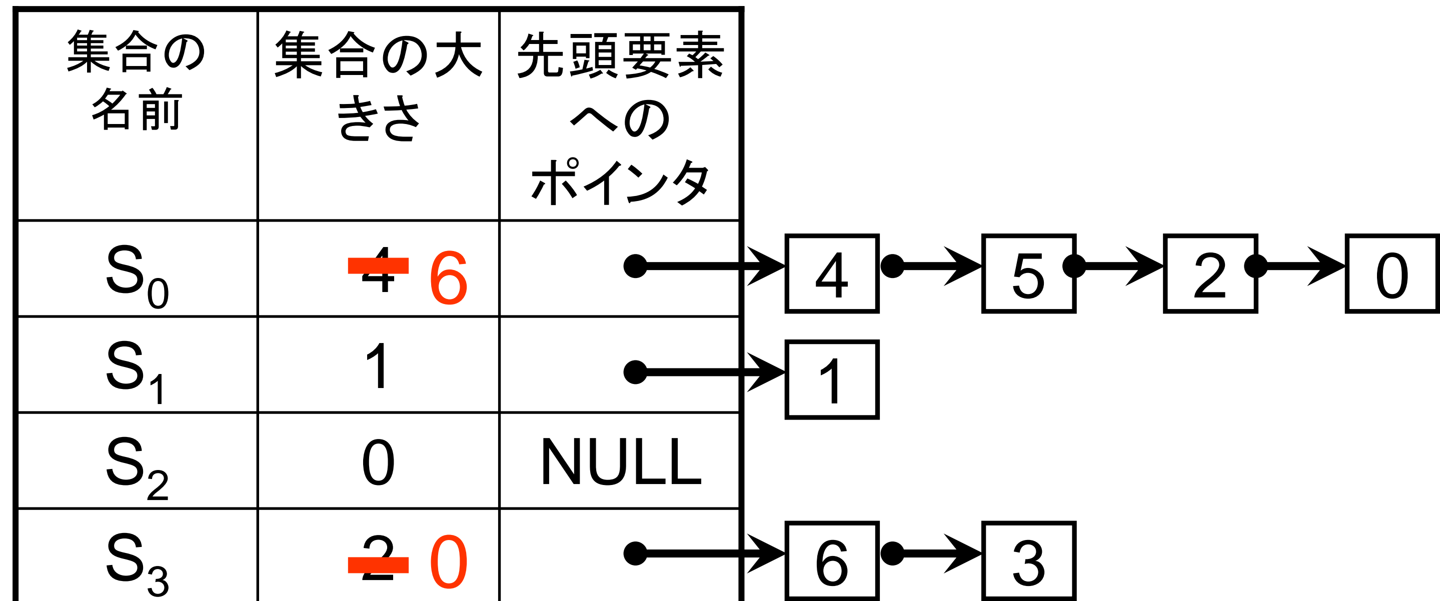
集合族の併合のデータ構造：配列＋リストによる実現

- S_0とS_3を併合し，名前をS_0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	0	~~3~~ 0	0	0	~~3~~ 0

①S_3の各要素のset_nameを変更
$O(|S_3|)$時間
S_0, S_3の大きさを変更
$O(1)$時間



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

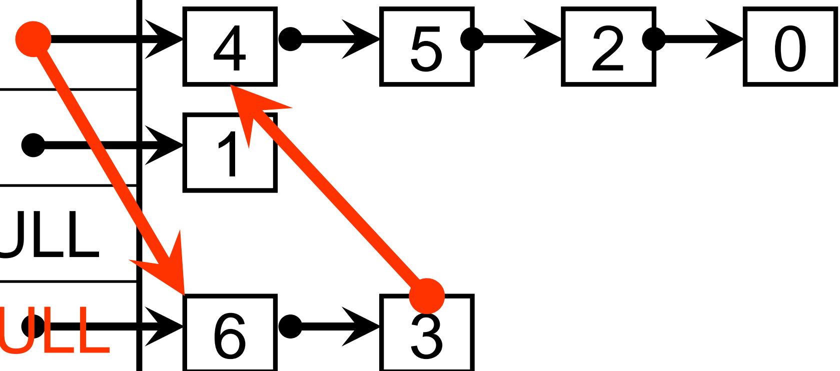
要素名	0	1	2	3	4	5	6
set_name	0	1	0	0	0	0	0

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

(1) S_3の最後の要素からS_0の先頭へポインタを張る
(2) S_0の先頭要素をS_3の先頭にする

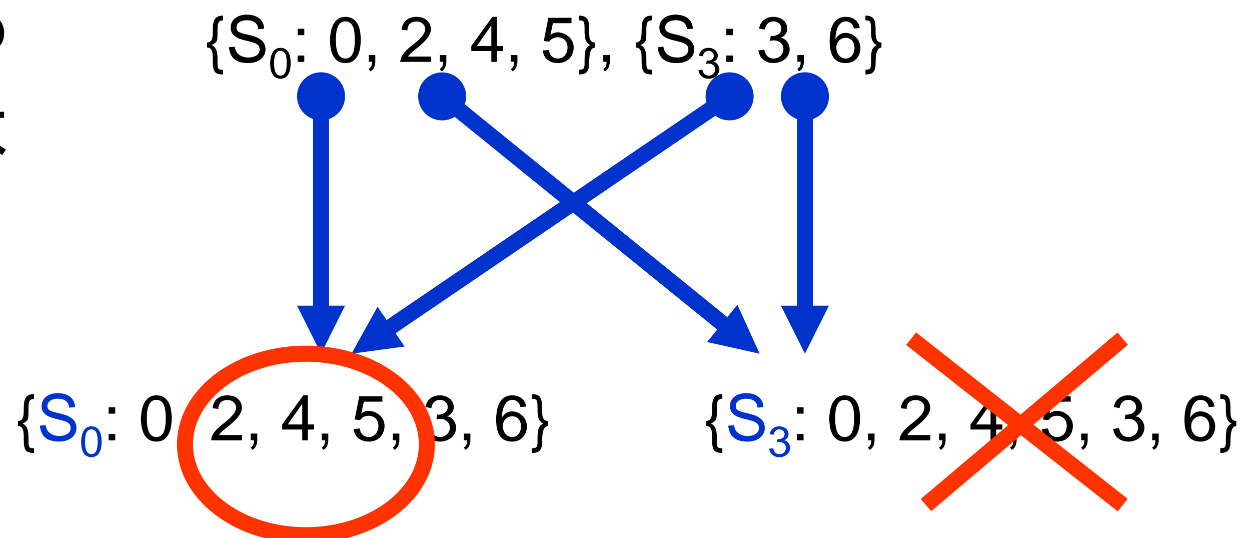
②S_0, S_3のポインタの付け替え
計算時間: $O(|S_3|)$

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



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
  ➔併合に要する時間の合計は
  $O(N \log_2 N)$



クラスカルのアルゴリズムの計算時間

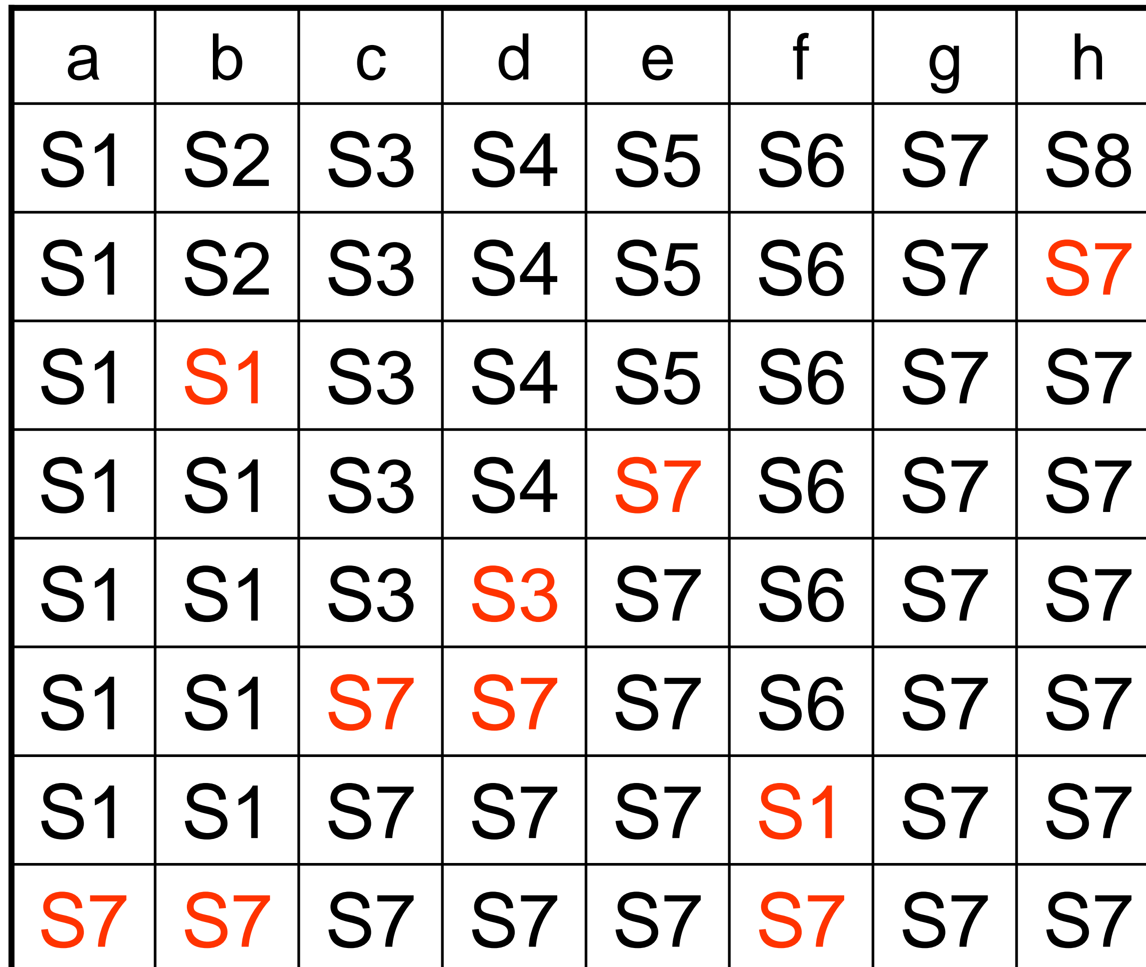
- アルゴリズムの実装方法
  - 枝を長さの短い順にソート --- O(m log m) = O(m log n)
  - 各枝の両端の節点が同じ連結成分に含まれるか否かチェック.
    - 同じ連結成分➔枝を除去
    - 異なる連結成分➔枝を加える

      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)

併合の例



a	b	c	d	e	f	g	h
S1	S2	S3	S4	S5	S6	S7	S8
S1	S2	S3	S4	S5	S6	S7	S7
S1	S1	S3	S4	S5	S6	S7	S7
S1	S1	S3	S4	S7	S6	S7	S7
S1	S1	S3	S3	S7	S6	S7	S7
S1	S1	S7	S7	S7	S6	S7	S7
S1	S1	S7	S7	S7	S1	S7	S7
S7	S7	S7	S7	S7	S7	S7	S7

S7とS8を併合➔S7

S1とS2を併合➔S1

S5とS7を併合➔S7

S3とS4を併合➔S3

S3とS7を併合➔S7

S1とS6を併合➔S1

S1とS7を併合➔S7

集合族の併合のデータ構造:配列+リストによる実現

主張:併合の工夫:常に大きい集合に小さい集合を併合

➔併合に必要な計算時間の合計は $O(N \log_2 N)$

証明:

- 併合に要する計算時間

  $= \Sigma_j$ (要素 j を含む集合の名前が更新された回数)
- 要素 j を含む集合の名前が1回更新される

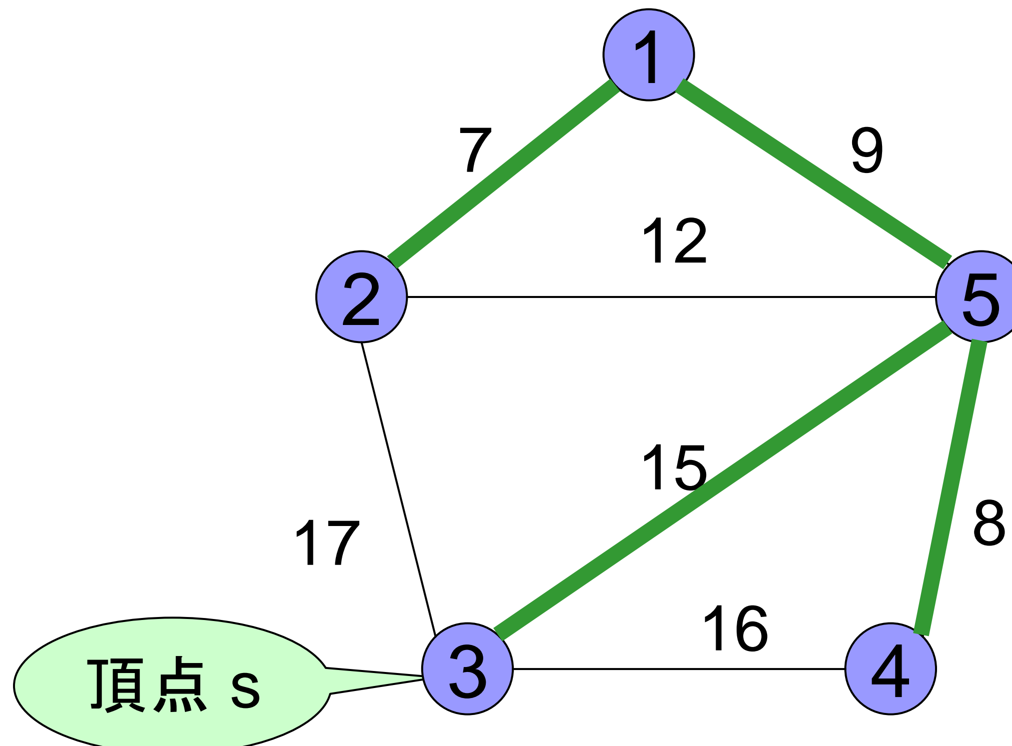
  ➔ 要素 j を含む集合の大きさは倍以上になる

  ∴要素 j を含む集合の名前の更新回数は高々 $\log_2 N$ 回
- 以上を纏めると,

  併合に要する計算時間= $O(N \log_2 N)$

プリムのアルゴリズム

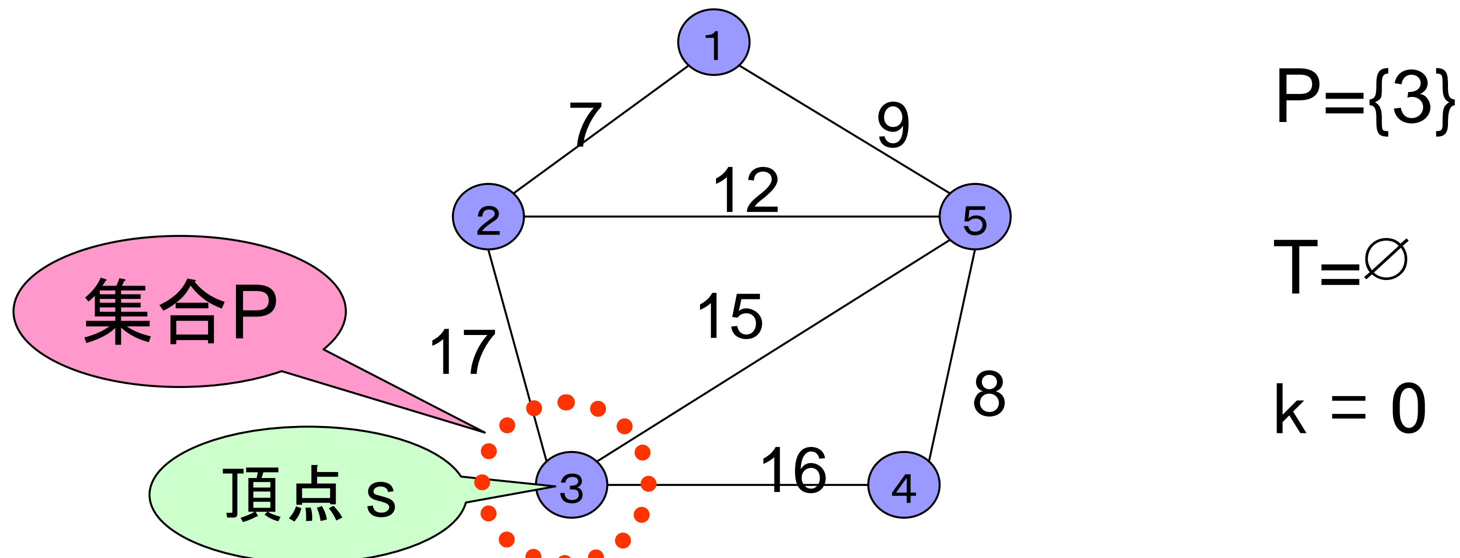
- アルゴリズムの大まかな流れ
  - 最初は一つの頂点 s からスタート
  - 次々に枝を加え，ひとつの木を大きくしていく



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

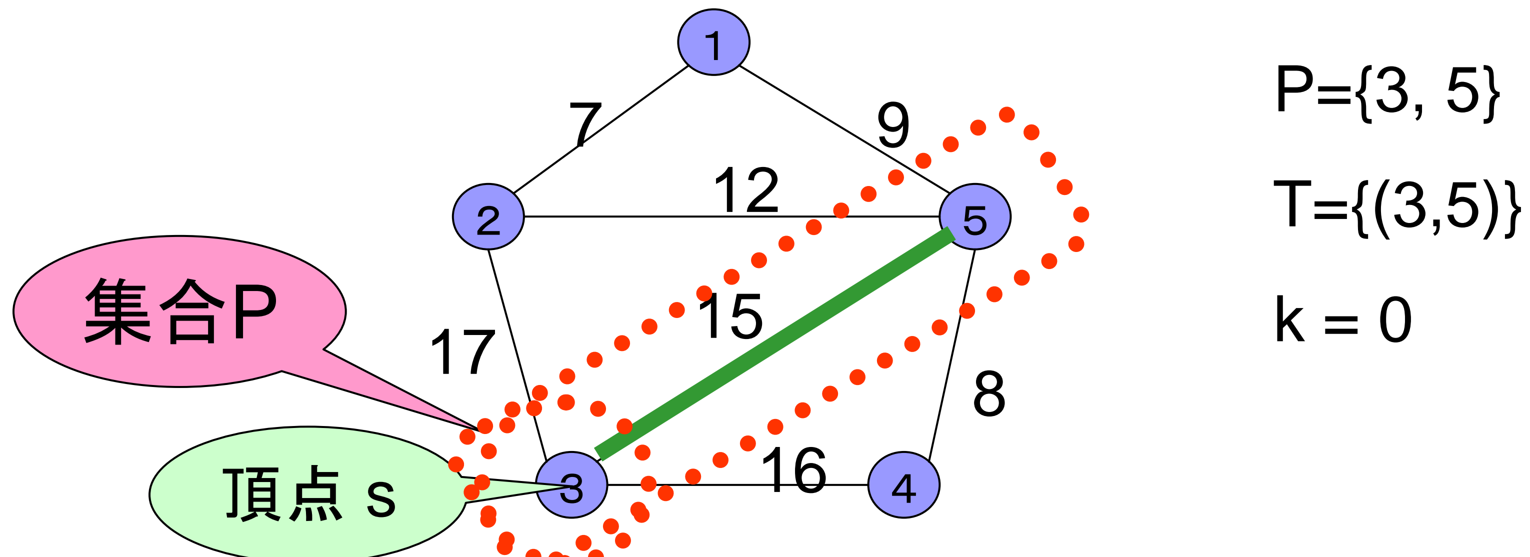
  Step 0: 頂点 s を適当に選び，P := {s}, T := ∅, k := 0 とおく.



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

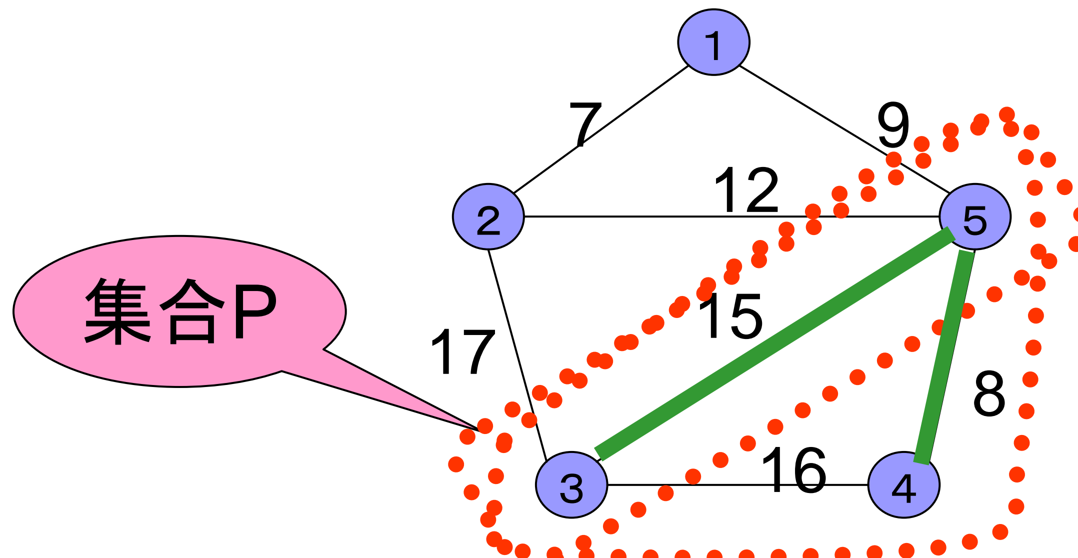
Step 0: 頂点 s を適当に選び，P := {s}, T := ∅, k := 0 とおく.
Step 1: u∈P, v∈V−P なる枝 (u, v) の中で
長さ最小のものを選び，Tに加える.
P := P∪{v}, k := k + 1 とおく.
Step 2: k = n−1 ならば終了．k < n−1 ならばStep 1へ.



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，P := {s}, T := ∅, k := 0 とおく.
Step 1: u∈P, v∈V−P なる枝 (u, v) の中で
長さ最小のものを選び，Tに加える.
P := P∪{v}, k := k + 1 とおく.
Step 2: k = n−1 ならば終了．k < n−1 ならばStep 1へ.



P={3, 5, 4}

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

k = 2

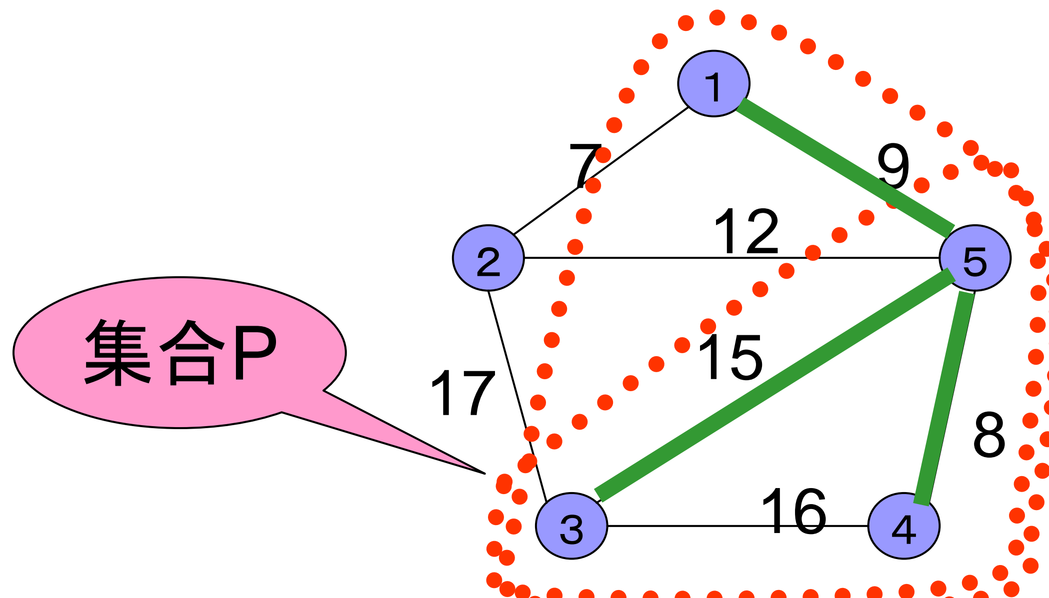
プリムのアルゴリズムの手順

- アルゴリズムの具体的な手順

Step 0: 頂点 s を適当に選び，P := {s}, T := ∅, k := 0 とおく.

Step 1: u∈P, v∈V−P なる枝 (u, v) の中で
　長さ最小のものを選び，Tに加える.
　P := P∪{v}, k := k + 1 とおく.

Step 2: k = n−1 ならば終了．k < n−1 ならばStep 1へ.



P={3, 5, 4, 1}

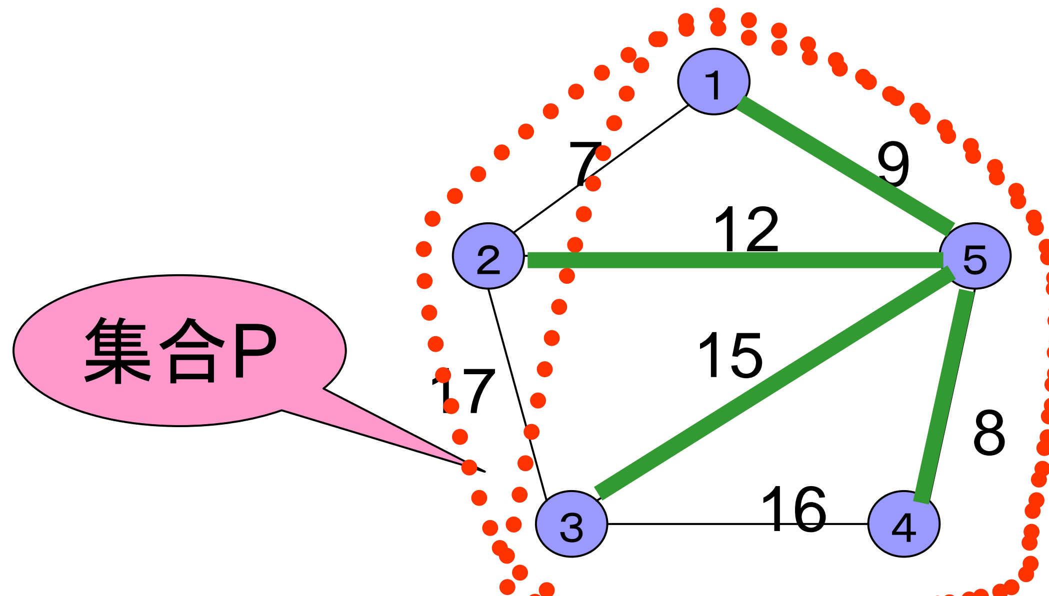
T={(3,5), (5,4), (5,1)}

k = 3

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，P := {s}, T := ∅, k := 0 とおく．
Step 1: u∈P, v∈V−P なる枝 (u, v) の中で
長さ最小のものを選び，Tに加える．
P := P∪{v}, k := k + 1 とおく．
Step 2: k = n−1 ならば終了．k < n−1 ならばStep 1へ．



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

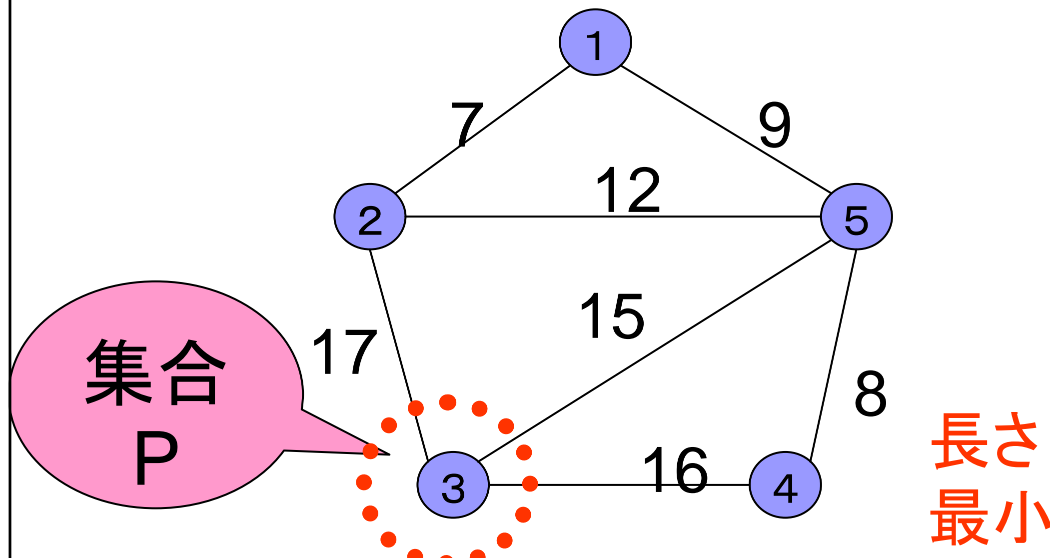
T={(3,5), (5,4), (5,1), (1,2)}

k = 4

プリムのアルゴリズムの実装方法

- Step 1: u∈P, v∈V–P なる枝 (u, v) の中で
  長さ最小のものを選び. . .

  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：V–Pに含まれる節点v
  - 各要素のデータ：節点 vから Pに接続する枝の最小の長さ（及び対応する枝）



ヒープの中の要素

節点 1, 最小長さ∞ (枝なし)

節点 2, 最小長さ17 (枝(2,3))

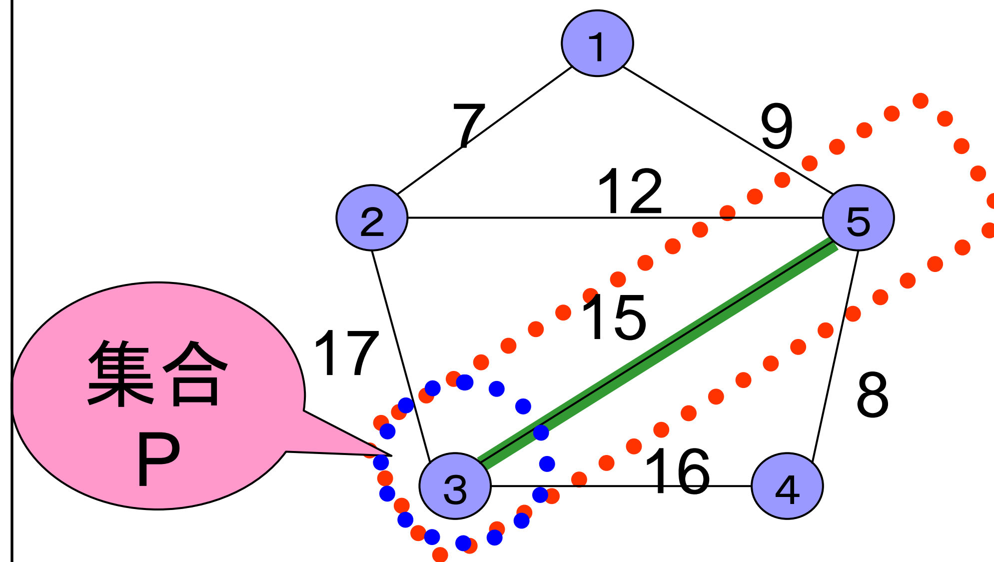
節点 4, 最小長さ16 (枝(4,3))

長さ最小

節点 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
  - Pに追加した頂点をヒープから削除



ヒープの中の要素

節点 1, 最小長さ∞ (枝なし)

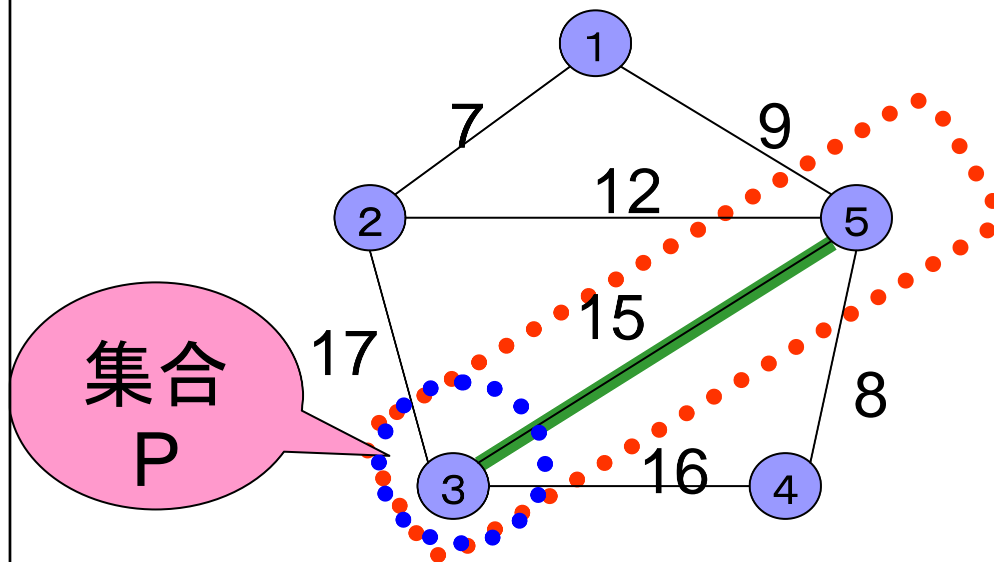
節点 2, 最小長さ17 (枝(2,3))

節点 4, 最小長さ16 (枝(4,3))

節点 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
  - Pに追加した頂点をヒープから削除
  - P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

節点 1, 最小長さ∞ (枝なし)

最小長さ9 (枝(1,5))

節点 2, 最小長さ17 (枝(2,3))

最小長さ12 (枝(2,5))

節点 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

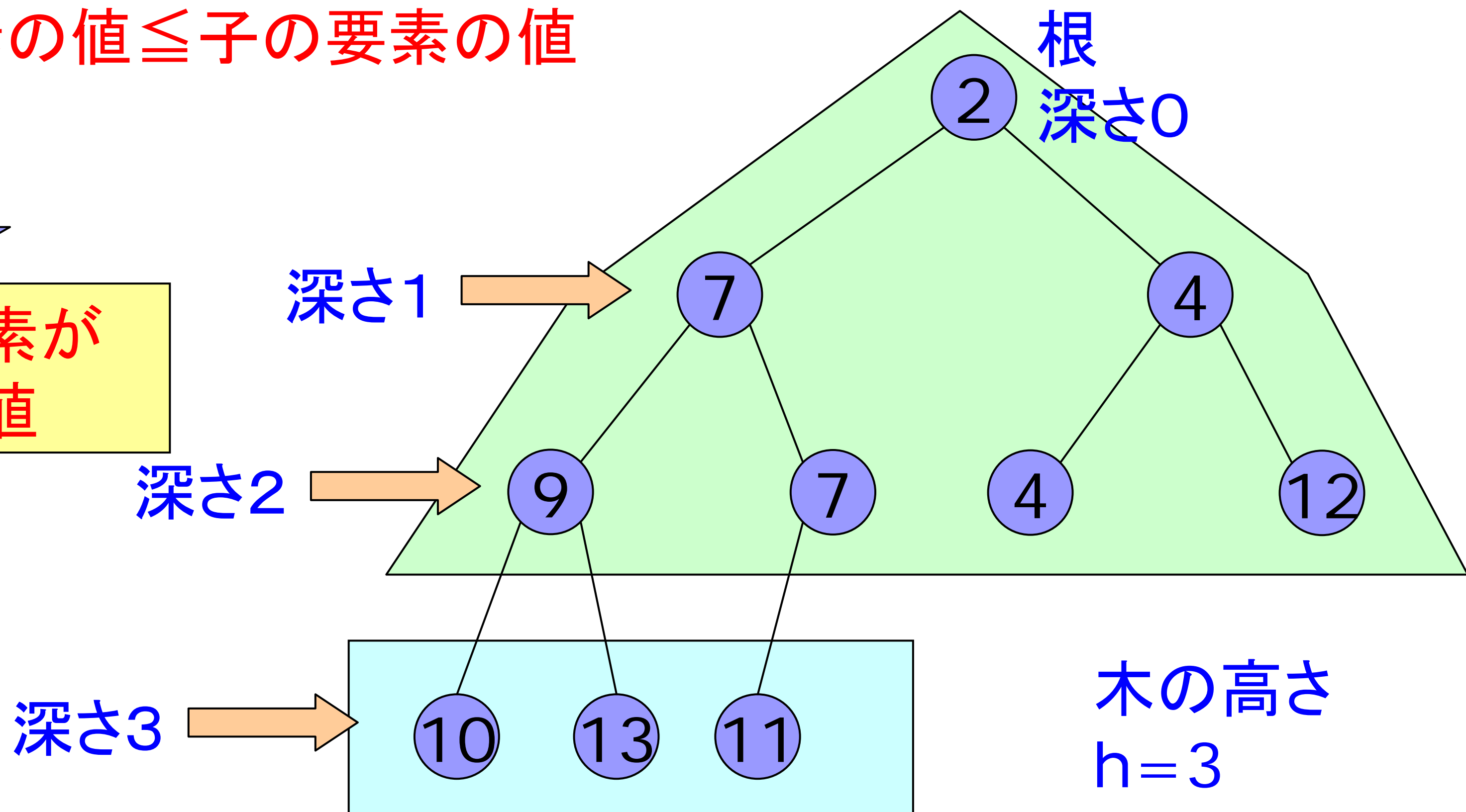
- ヒープの維持に必要な計算時間
  - PとV－Pを結ぶ最小長さの枝を求める
    --- ヒープの中で最小の要素を求める➔ O(1)
  - Pに追加した頂点 v の削除
    --- ヒープの中で要素を一つ削除➔ O(log n)
  - ヒープの各要素の最小長さを更新
    --- ヒープの中の要素のデータを減少させる➔ O(log n)
    更新が必要な要素数≦追加した頂点 v の次数 d(v)
    ∴O(d(v) log n)
- アルゴリズム全体で O(m log n) 時間を要する

ヒープ

- 次の条件を満たす2分木をヒープと呼ぶ

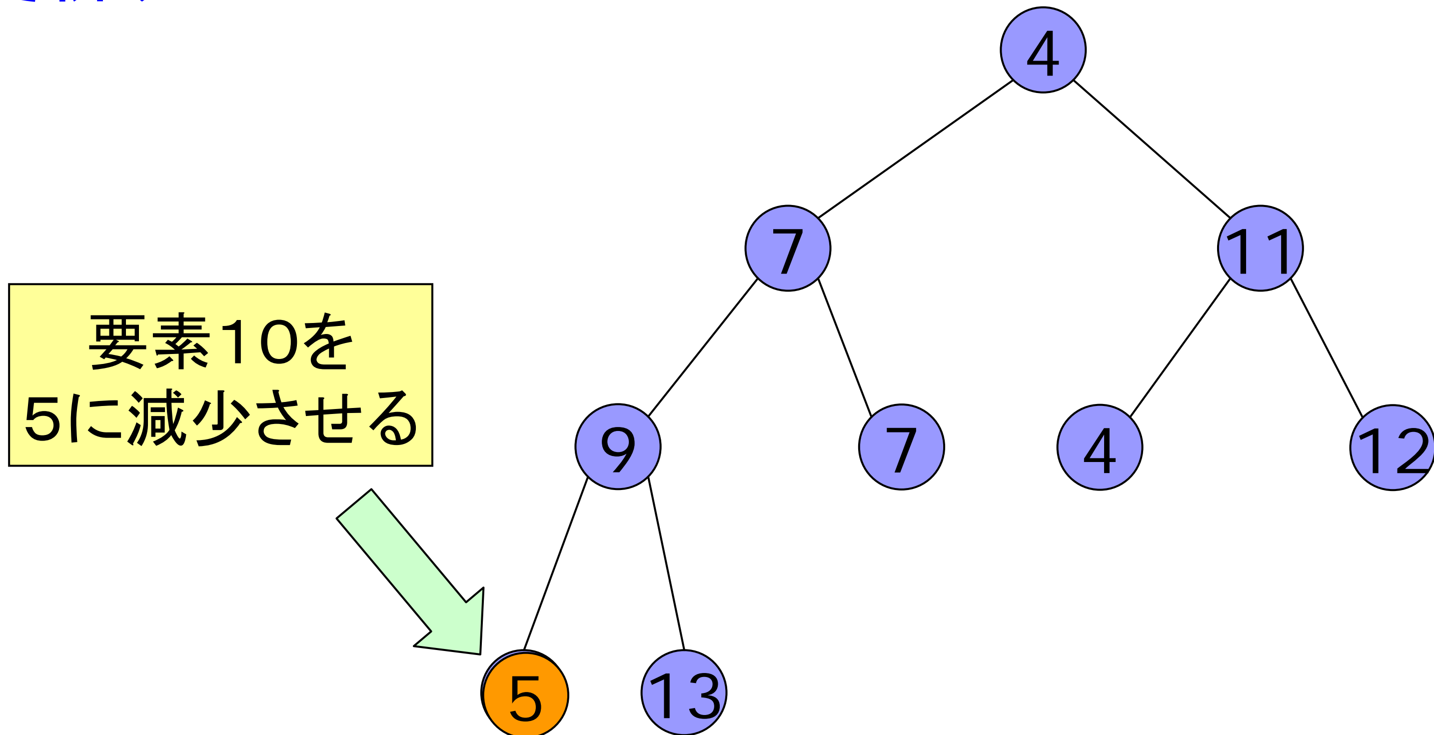
（1）木の高さが h ➔ 深さ h-1 までは完全2分木
深さ h では，左側に葉が詰められている

（2）親の要素の値≦子の要素の値



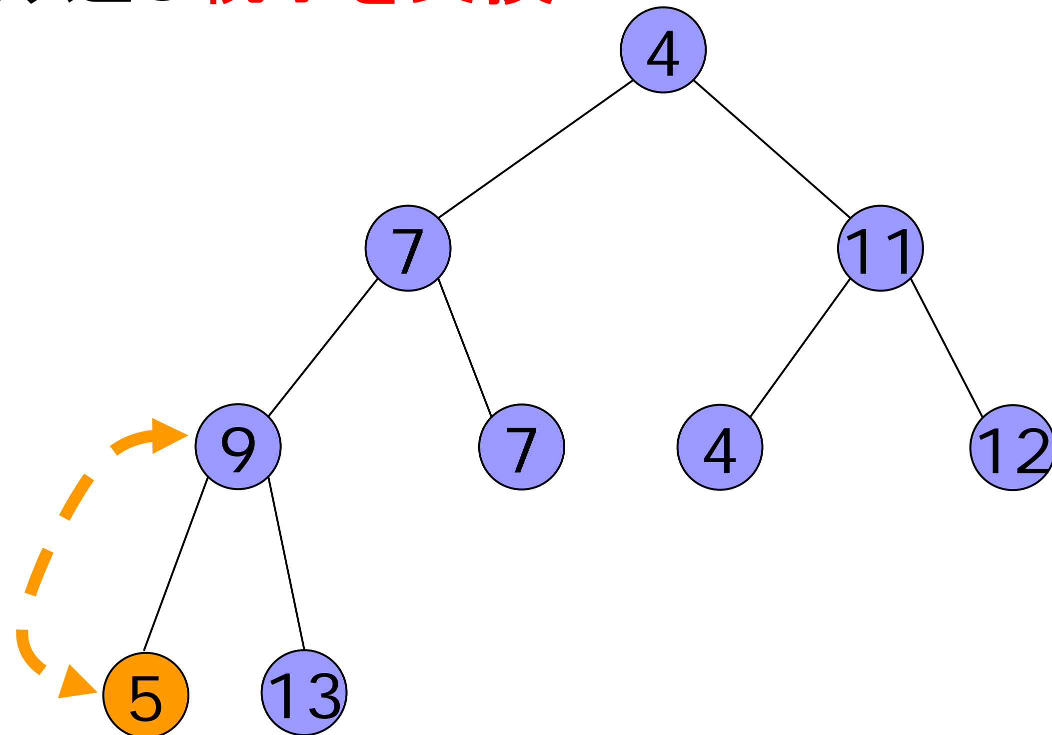
ある要素を減少させる

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



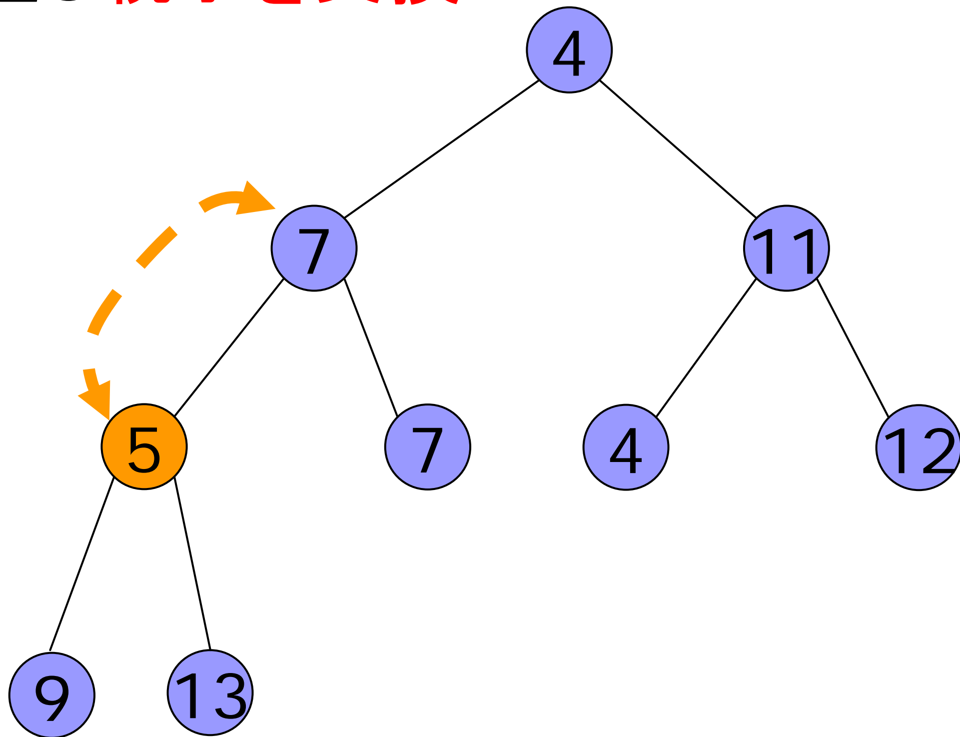
ある要素を減少させる

（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



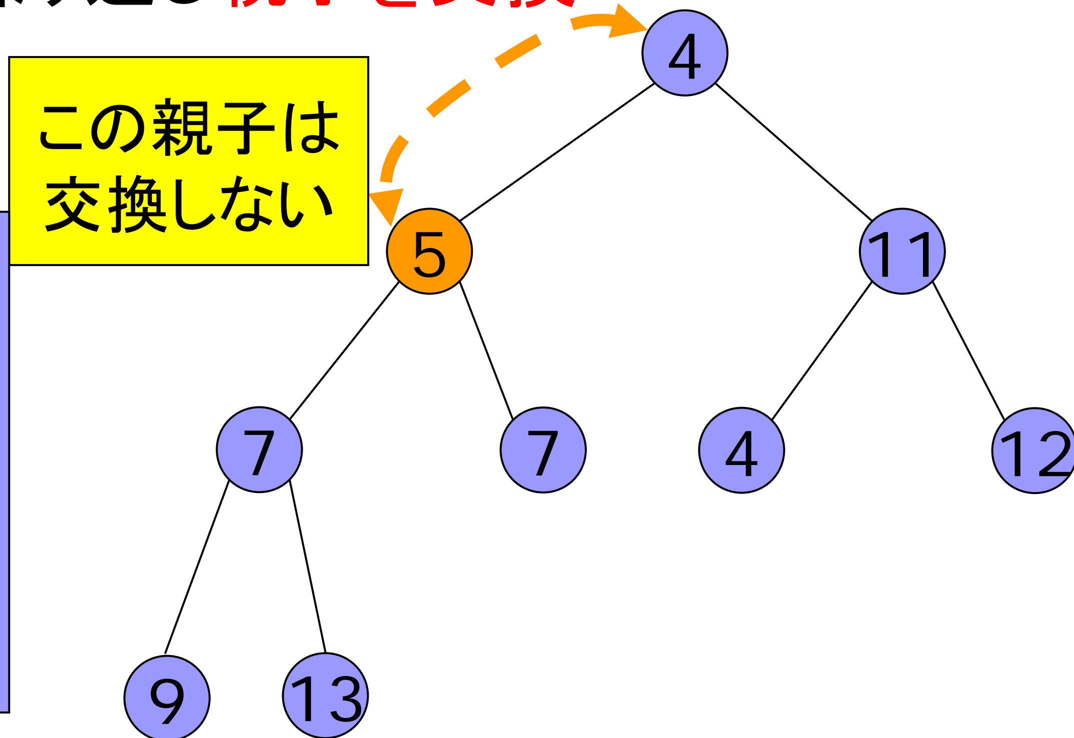
ある要素を減少させる

（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



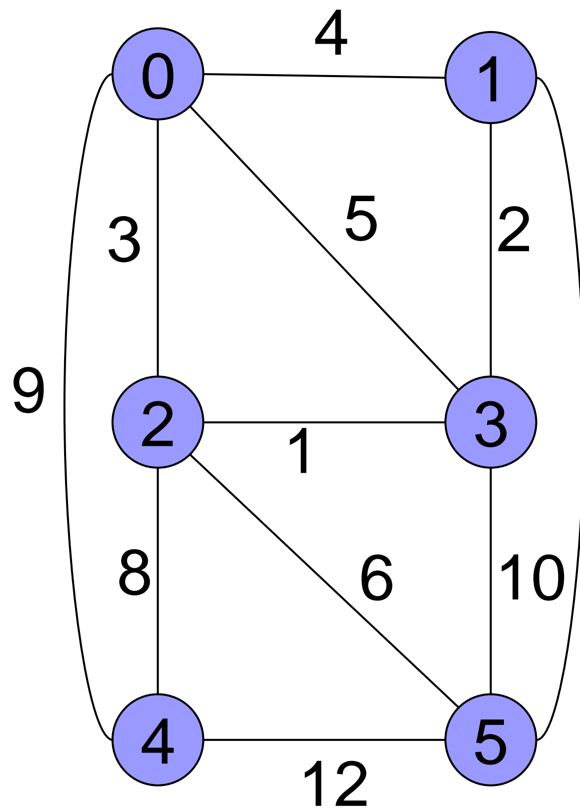
ある要素を減少させる

(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



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$O(\log n)$

レポート問題



（1）左のグラフの最小木を，プリムのアルゴリズムを用いて計算しなさい．ただし，最初の頂点sとして，頂点4を使うこと．
レポートには，答えを書くだけでなく，計算の過程も書くこと．
（2）プリムのアルゴリズムのStep1において長さ最小の枝を選ぶときに，ヒープを使わず単純な方法を用いたときの時間計算量を解析しなさい．
（3）無向グラフにおいて
$\Sigma \{d(v) \mid v \in V\} = 2m$ が成り立つことを証明しなさい．