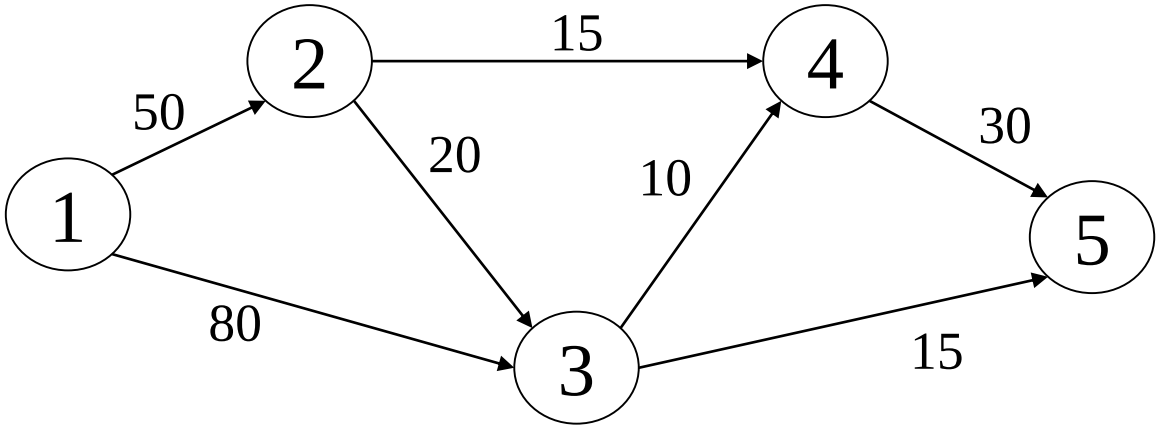
算法数理工学
第10回

定兼　邦彦

最短路問題

- 有向グラフ $G = (V, E)$ を考える
  - V: 節点集合, 節点数 n
  - E: 枝集合, 枝数 m
- グラフ G の各枝 $(i,j) \in E$ は長さ a_{ij} を持つ
- ある節点 $s \in V$ から別の節点 $t \in V$ への路の中で,
  最も長さの短いものを見つける問題を最短路問題という



重み無しグラフの場合

- 重み無し ⇔ 全ての枝の重みが 1
- 幅優先探索で求まる
  - O(n+m) 時間
- s からの距離が d の点の集合を V_d とする ($d \geq 0$)
  - $V_0 = \{s\}$
  - V_d = (V_{d-1}の点から枝1本で行ける点の集合)
    - ($V_0, V_1, \ldots, V_{d-1}$ に入っている点の集合)
- V_dを求めるには，V_{d-1} の各点 v に対し，v の隣接点でまだ訪れていない点を求めればいい
- V_dをキューで管理する

- 各頂点に色を付ける
  - 白: 未発見
  - 灰: 発見済みだが隣接リストの探索が未完了
  - 黒: 発見済みで隣接リストの探索も完了
- 初期状態: s 以外の全ての点 u に対し
  - $color[u]$ = 白
  - $d[u] = \infty$ (s からの距離)
  - $\pi[u]$ = NIL (最短路木での親)
- 始点 s については
  - $color[s]$ = 灰
  - $d[s] = 0$
  - $\pi[s]$ = NIL (最短路木での根)

BFS(G, s)

1. $Q \leftarrow \emptyset$ (空集合)
2. ENQUEUE(Q, s)
3. while $Q \neq \emptyset$
4. $u \leftarrow$ DEQUEUE(Q)
5. for each $v \in Adj[u]$
6. if $color[v]$ = 白
7. $color[v]$ = 灰
8. $d[v] \leftarrow d[u] + 1$
9. $\pi[v] \leftarrow u$
10. ENQUEUE(Q, v)
11. $color[u] \leftarrow$ 黒

計算時間の解析

- 初期化以外で色が白になる点は無い
- ある点がキューに入れられるとき，その色は白で，キューに入ると灰になる

  ⇒各点は高々1回だけキューに挿入・削除される
- 各点 u の隣接リストが走査されるのは u がキューから削除されるときのみ

  ⇒各点の隣接リストは高々1回だけ走査される

  ⇒隣接リストの走査にかかる時間は全体で $O(m)$
- 初期化の時間は $O(n)$
- 全体で $O(n+m)$

正当性の証明

- 定義: s から v への最短路距離 (shortest-path distance) $\delta(s, v)$ を, s から v へのパスの辺数の最小値と定義する. パスが無ければ $\delta(s, v) = \infty$.
- 補題1: 有向または無向グラフ G において,
  任意の辺 $(u, v) \in E$ に対し,
  $$\delta(s, v) \leq \delta(s, u) + 1$$
- 証明: u が s から到達可能ならば v へも到達可能で, s から v へのパスは s から u への最短路に (u, v) を追加したものか, それより短いものとなる.
  u が s から到達不可能ならば $\delta(s, u) = \infty$ なので不等式は成立する.

- 補題2 : BFSが停止した時，各点 $v \in V$ に対し $d[v] \geq \delta(s, v)$ が成立する.
- 証明 : ENQUEUEの操作回数に関する帰納法で証明する. 最初の s だけが挿入された段階では $d[s] = 0 = \delta(s, s)$, 全ての点 $v \in V - \{s\}$ に対し $d[v] = \infty \geq \delta(s, v)$ なので成立.
- 点 u の隣接点 v を探索する場合を考える.
  帰納法の仮定により $d[u] \geq \delta(s, u)$.
  $d[v]$ を更新すると

$$\begin{aligned} d[v] &= d[u] + 1 \\ &\geq \delta(s, u) + 1 \\ &\geq \delta(s, v) \quad \text{(補題1より)} \end{aligned}$$

  となり，ENQUEUE後も成り立つ

- 補題3: BFSの実行中で，キュー Q が点 $\langle v_1, v_2, \ldots, v_r \rangle$ を含むとする (v_1が先頭，v_rが末尾).
  このとき $d[v_r] \leq d[v_1]+1$ かつ
  $d[v_i] \leq d[v_{i+1}]$ $(i=1,2,\ldots,r-1)$
- 証明: キューの操作回数に関する帰納法を用いる.
  初期段階はキューは s だけを含むので成り立つ.
- キューの先頭 v_1 を削除すると v_2 が新たな先頭になる. 帰納法の仮定より $d[v_1] \leq d[v_2]$.
  このとき $d[v_r] \leq d[v_1]+1 \leq d[v_2]+1$ となり成立.
- キューに頂点 v を挿入するとき ($v_{r+1} = v$ となる)
  現在頂点 u の隣接リストを走査しているとする.

- $d[v_{r+1}] = d[v] = d[u]+1$(アルゴリズムの動作)
- u を削除する前のキューは$\langle u = v_1, v_2, \ldots, v_r \rangle$なので帰納法の仮定から $d[v_r] \leq d[u]+1$.
- 従って $d[v_r] \leq d[u]+1 = d[v] = d[v_{r+1}]$.
  また, $d[v_{r+1}] = d[u]+1 = d[v_1]+1$.
- よって, u を削除後のキューでも $d[v_{r+1}] \leq d[v_1]+1$.
- 定理: アルゴリズムBFSは始点 s から到達可能な全ての頂点 $v \in V$ を発見し, 停止した時全ての v に対して $d[v] = \delta(s, v)$ が成立する. さらに, s から到達可能な全ての $v \neq s$ に対し, s から v への最短路の1つは, s から $\pi[v]$ への最短路の後に辺 $(\pi[v], v)$ を連接したものである.

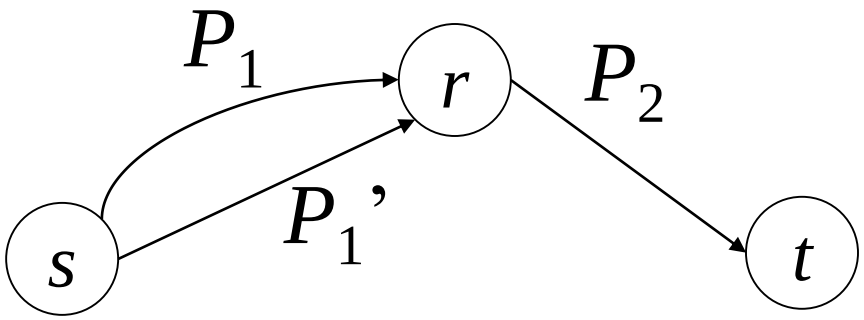
- 証明: 背理法で示す. 最短路長に等しくない d 値を受け取る頂点が存在すると仮定する. v を, そのような頂点の中で $\delta(s, v)$ が最小の頂点とする.
- 補題2より $d[v] \geq \delta(s, v)$ なので, $d[v] > \delta(s, v)$.
- v が s から到達不可能ならば $\delta(s, v) = \infty$ なので $d[v]$ は正しい値 (∞). よって v は到達可能.
- u を s から v への最短路上の点で v の直前のものとすると $\delta(s, v) = \delta(s, u) + 1$ である.
- $\delta(s, u) < \delta(s, v)$ であり, 仮定より最短路長が $\delta(s, v)$ より短い場合は $d[u] = \delta(s, u)$ なので $d[v] > \delta(s, v) = \delta(s, u) + 1 = d[u] + 1$ …(1)

- キューから u を削除する場合を考える.
  v の色は白，灰，黒のいずれかである.
- v が白の時
  - $d[v] = d[u] + 1$ を実行するので式 (1) に矛盾.
- v が黒の時
  - v は既にキューから削除されている
  - $d[v] \leq d[u]$ なので式 (1) に矛盾.
- v が灰の時
  - ある点 w を削除した時に v は灰に彩色された
  - $d[v] = d[w]+1$
  - w は u よりも先にキューから削除されたので $d[w] \leq d[u]$
  - よって $d[v] \leq d[u]+1$ となり式 (1) に矛盾.

- 以上より，全ての点 v に対して $d[v] = \delta(s, v)$.
- $\pi[v] = u$ ならば $d[v] = d[u] + 1$ が成り立つ．よって辺 $(\pi[v], v)$ を連接すると最短路が求まる.
- グラフ $G_\pi = (V_\pi, E_\pi)$ を次のように定義する
  - $V_\pi = \{v \in V \mid \pi[v] \neq \mathrm{NIL}\} \cup \{s\}$
  - $E_\pi = \{(\pi[v], v) \in E \mid v \in V_\pi - \{s\}\}$
- V_π は s から到達可能な全頂点
- 全ての $v \in V_\pi$ に対し，s から v に至る唯一のパスが G_π に存在し，これが G における s から v への最短路になっている
- G_π は幅優先探索木(breadth-first tree)と呼ばれる

最適性の原理

- 節点 s から t への最短路 P が得られているとする
- 路 P に含まれる節点を1つ任意に選び, r とする
- 路 P は s から r までの部分 P_1 と，r から t への部分 P_2 に分割できる. このとき, P_1 は s から r への最短路で, P_2 は r から t への最短路となる
  - もし P_1 より短い s から r への路 P_1'が存在するなら, P_1'と P_2 をつないだ路は P より短くなる
- このような性質を最適性の原理と呼ぶ



- ある節点 s からネットワークの他の全節点への最短路を求める問題を考える
- ダイクストラ法は，枝の長さに関する非負条件 $a_{ij} \geq 0 \ \ ((i,j) \in E)$ の仮定の下で，節点 s から各節点 $i \in V$ への最短路の長さの上限値 $d(i)$ を更新していく反復アルゴリズム
- 計算の途中では，$d(i)$ の値が s から i への真の最短路の長さに等しいことがわかっている節点が存在する．そのような節点の集合を S で表す
- $\overline{S}$ は S の補集合 $V - S$ を表す

ダイクストラ (Dijkstra) 法

(0) $S := \emptyset$, $\bar{S} := V$, $d(s) := 0$, $d(i) := \infty$ $(i \in V - \{s\})$
とおく

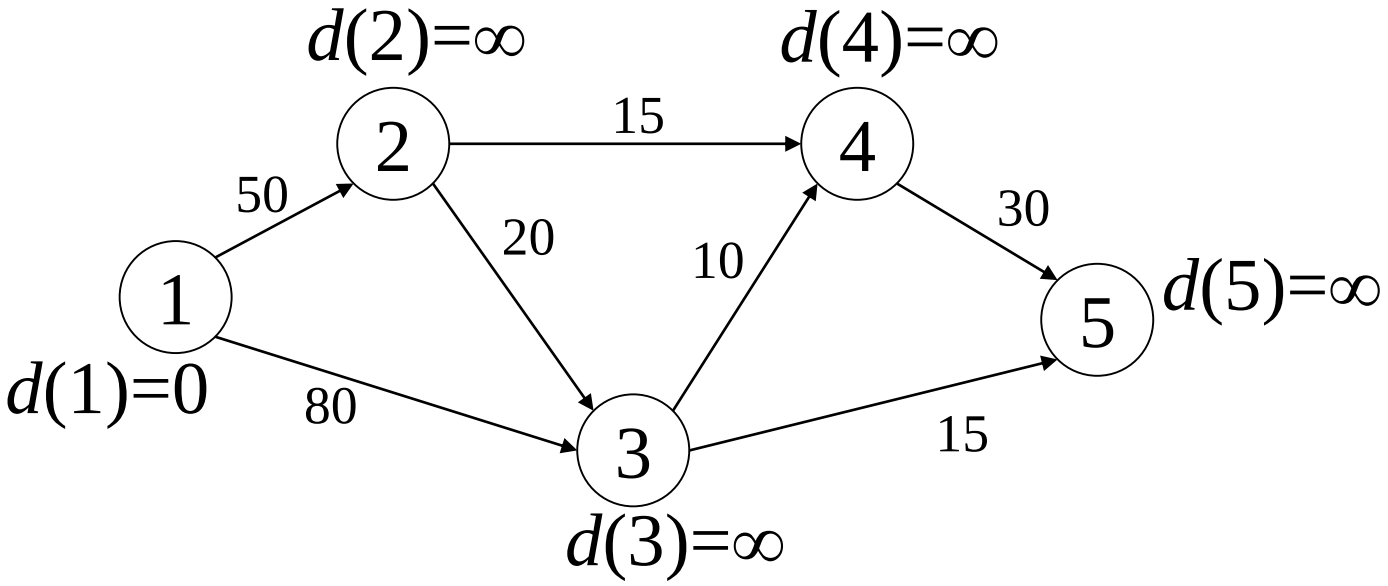
(1) $S = V$ なら計算終了. そうでないなら,
$d(v) = \min\left\{d(i) \mid i \in \bar{S}\right\}$ である節点 $v \in \bar{S}$ を選ぶ

(2) $S := S \cup \{v\}$, $\bar{S} := \bar{S} - \{v\}$ とし, $(v,j) \in E$ かつ $j \in \bar{S}$
であるような全ての枝 (v,j) に対して
$d(j) > d(v) + a_{vj}$ ならば $d(j) = d(v) + a_{vj}$, $p(j) := v$
とする. ステップ (1) に戻る

$p(i)$ は, s から i までの最短路において i の直前に
位置する節点を示すために用いられる

[初期化]

(0) $S = \phi, \bar{S} = \{1,2,3,4,5\}$



[反復1]

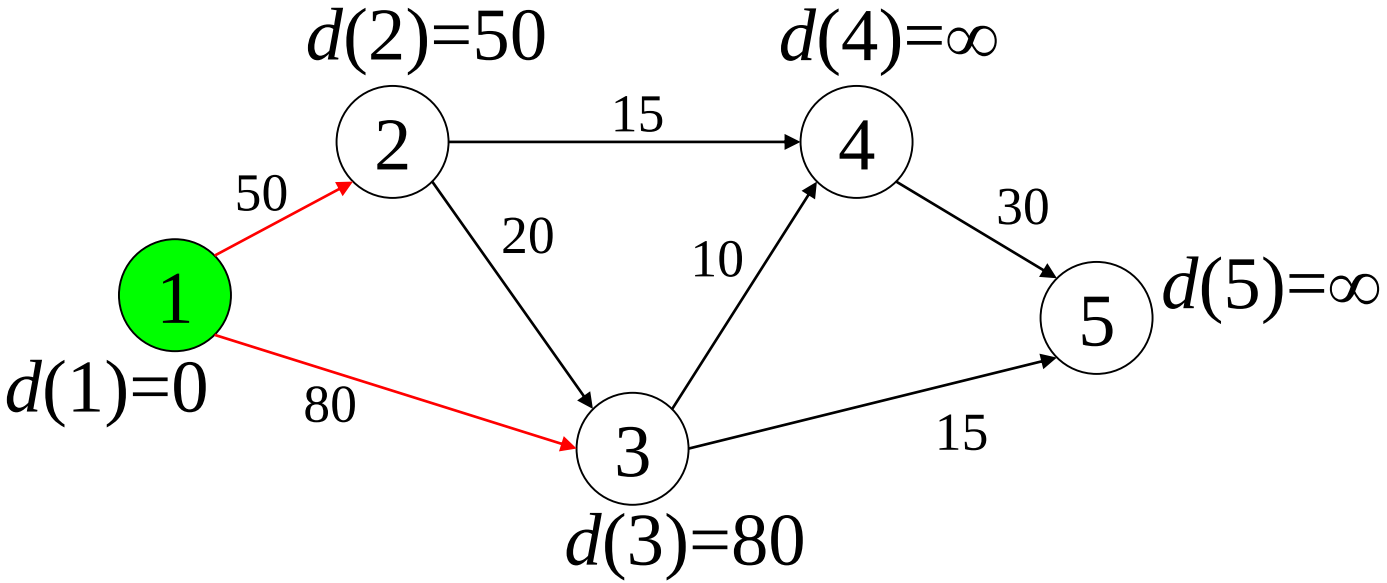
(1) $\bar{S} = \{1,2,3,4,5\}$

$\min\{d(i) \mid i \in \bar{S}\} = 0$ より $v = 1$

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

$d(2) = \infty > d(1) + a_{12} = 50$ より $d(2) = 50, p(2) = 1$

$d(3) = \infty > d(1) + a_{13} = 80$ より $d(3) = 80, p(3) = 1$



[反復2]

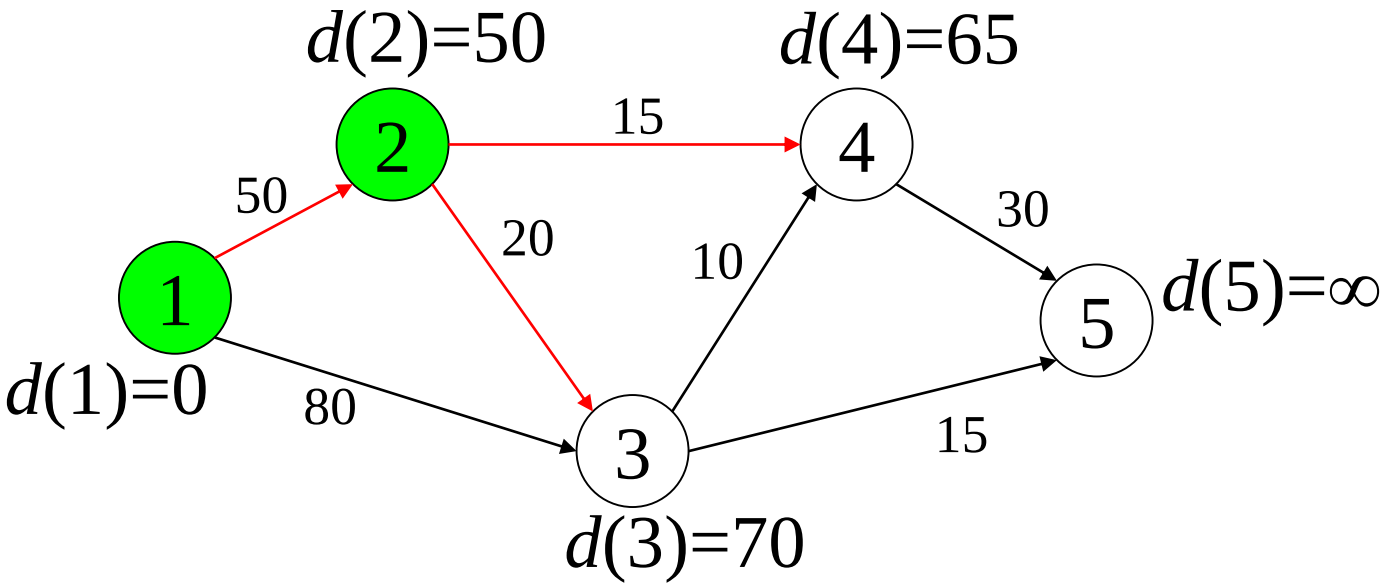
(1) $\bar{S} = \{2,3,4,5\}$

$\min\{d(i) \mid i \in \bar{S}\} = 50$ より $v = 2$

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

$d(3) = 80 > d(2) + a_{23} = 70$ より $d(3) = 70, p(3) = 2$

$d(4) = \infty > d(2) + a_{24} = 65$ より $d(4) = 65, p(4) = 2$



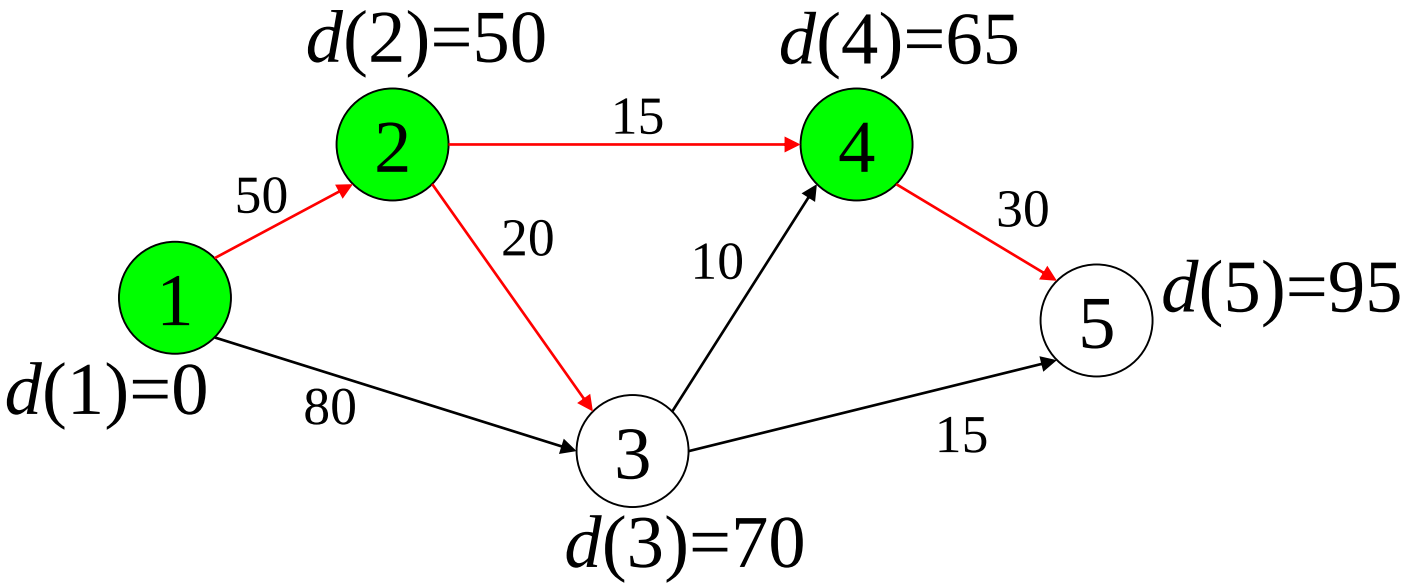
[反復3]

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

$\min\{d(i) \mid i \in \bar{S}\} = 65$ より $v = 4$

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

$d(5) = \infty > d(4) + a_{45} = 95$ より $d(5) = 95$, $p(5) = 4$



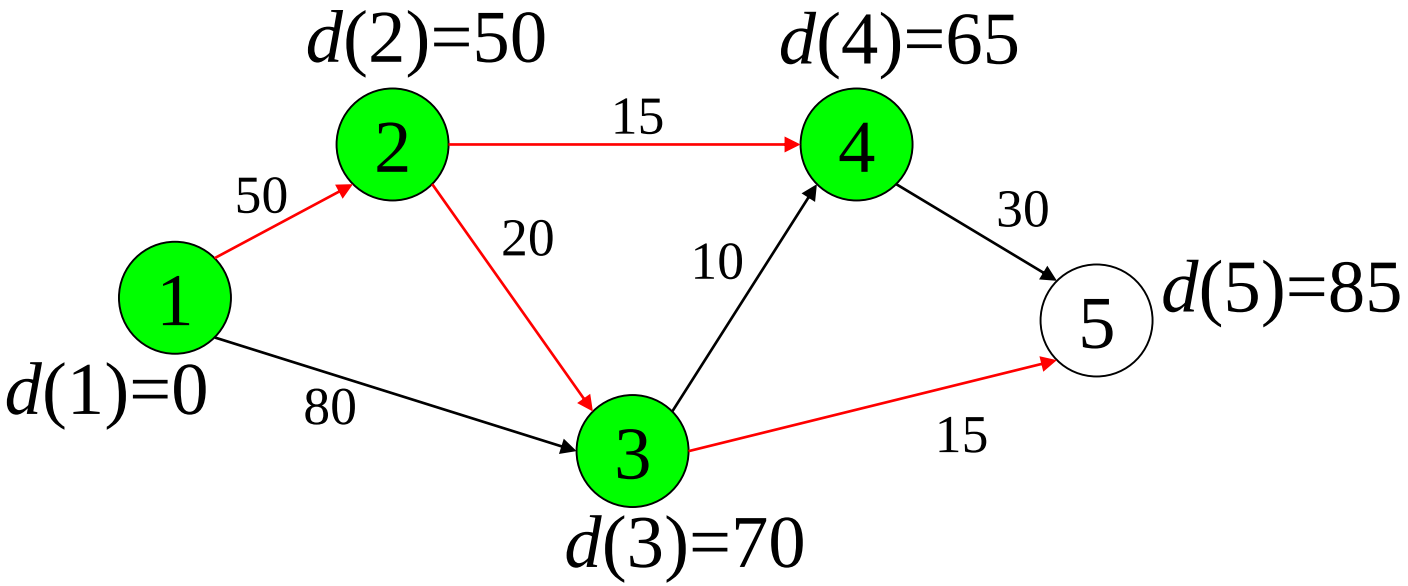
[反復4]

(1) $\bar{S} = \{3,5\}$

$\min\{d(i) \mid i \in \bar{S}\} = 70$ より $v = 3$

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

$d(5) = 95 > d(3) + a_{35} = 85$ より $d(5) = 85, p(5) = 3$



[反復5]

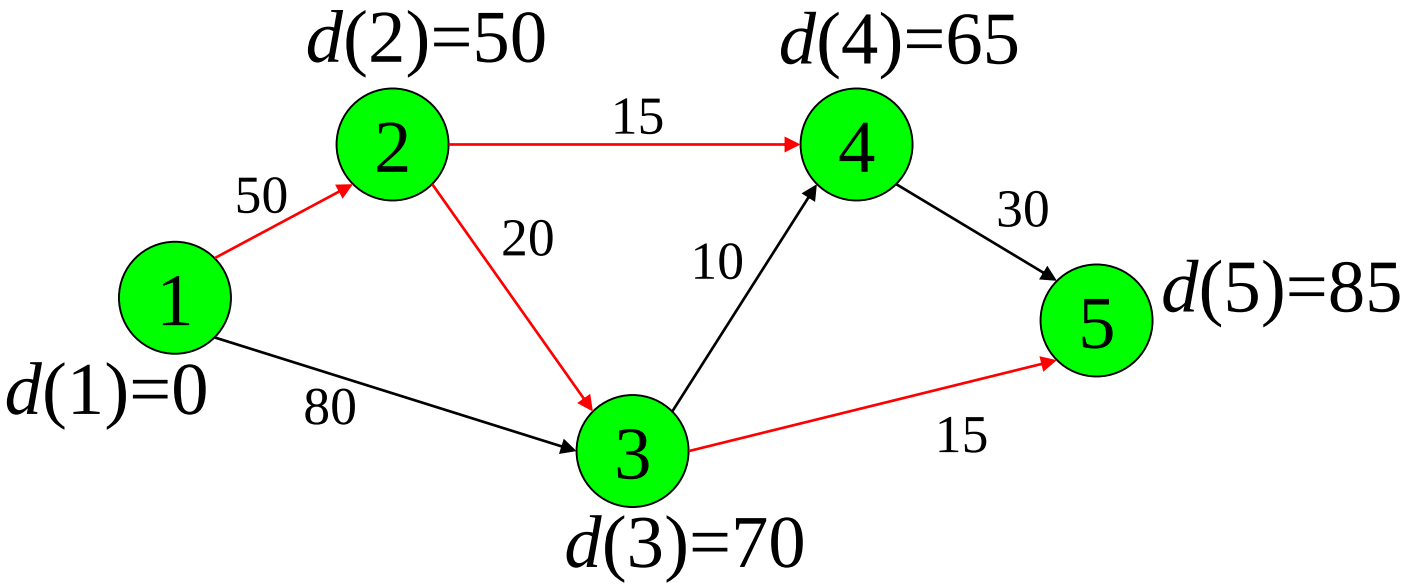
(1) $\bar{S} = \{5\}$

自動的に $v = 5$

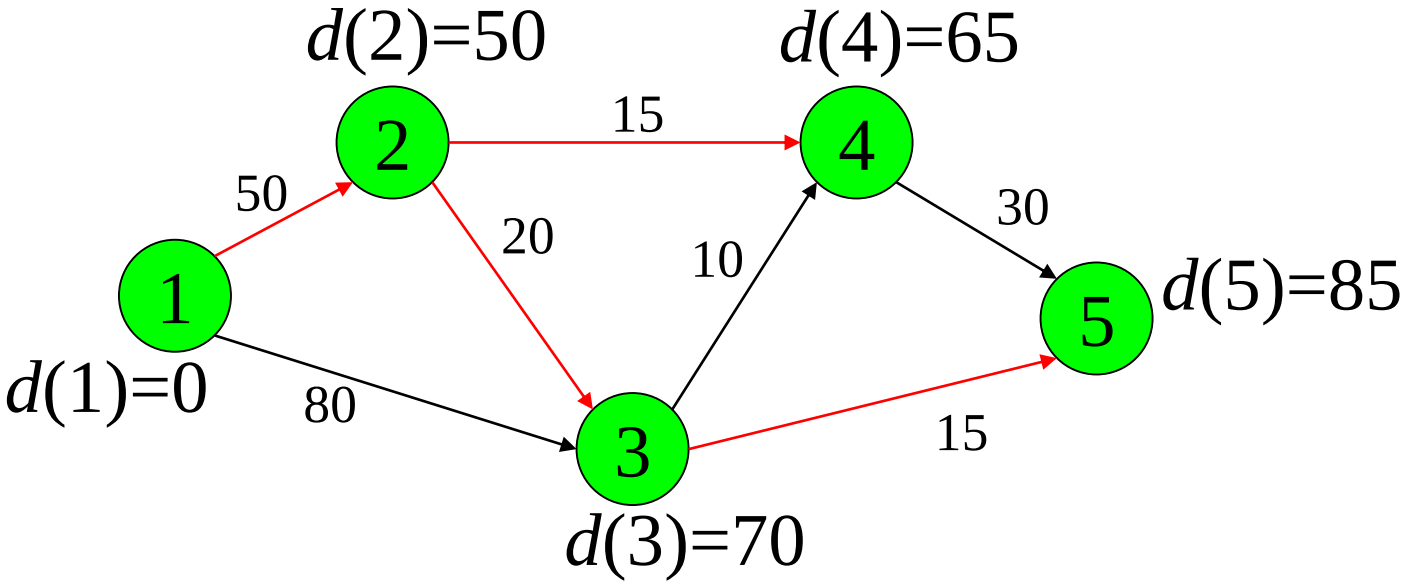
(2) $S = \{1,2,3,4,5\}, \bar{S} = \phi$

[反復6]

(1) $S = V$ なので終了



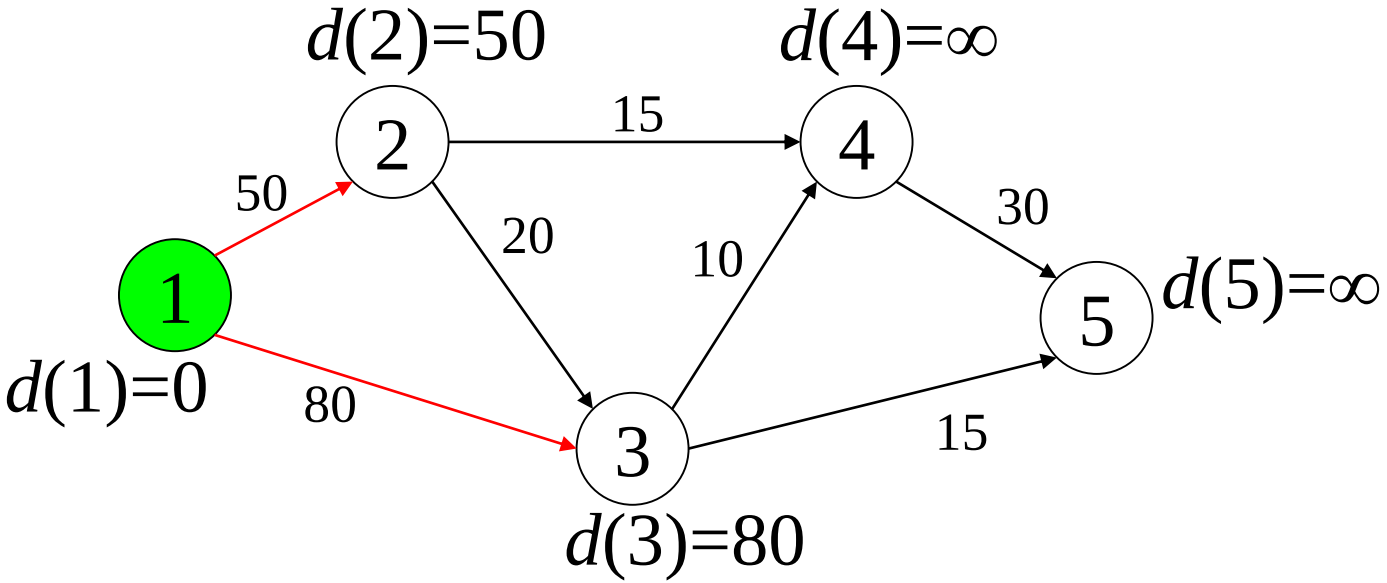
- アルゴリズムが終了したときの $d(i)$ は，節点 1 から i への最短路の長さを与えている
- 枝の集合 $\{(p(i),i)\}$ が各節点への最短路を表す
  これを最短路木と呼ぶ



アルゴリズムの正当性の証明

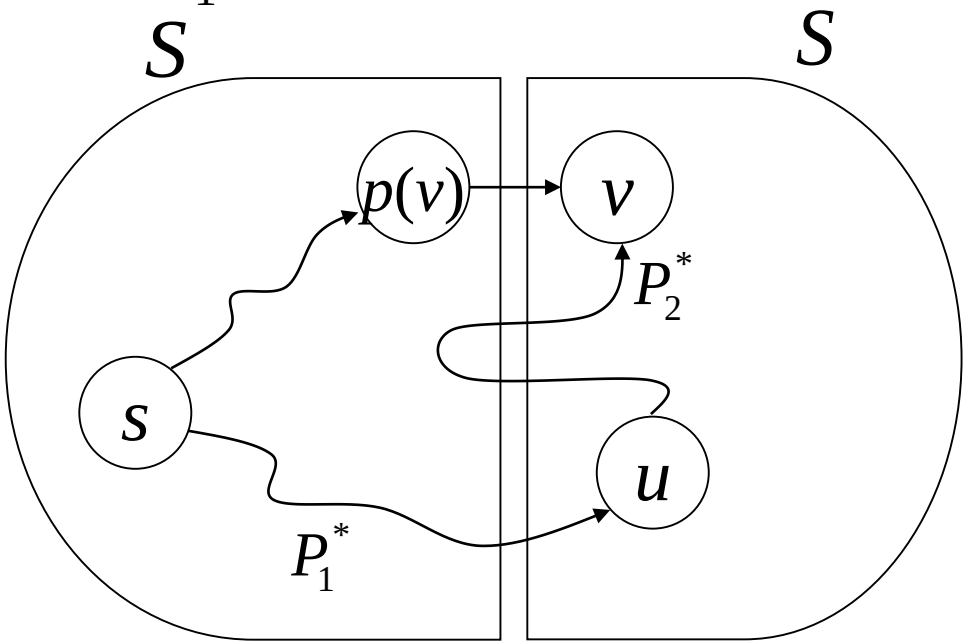
- ダイクストラ法で，S は出発点 s からの最短路の長さがわかっている節点の集合であることを確認
- 以下のことを帰納法で示す
  - 全ての i に対し，$d(i)$ = [S に含まれる節点のみを経由して s から i に至る最短路の長さ] (3.3)
    (S に含まれる節点のみを経由するのでは到達できない場合は $d(i) = \infty$)
  - $i \in S$ ならば，$d(i)$ = [s から i への最短路の長さ]

- [反復1] 終了後 $S = \{s\}$, $d(s) = 0$
- S の点 s では，s から s への最短距離は 0 で，$d(s)$ と等しい
- Sの点から1本の枝で到達できる $\bar{S}$ の点(2 と 3)では $d(i)$ = [S に含まれる節点のみを経由して s から i に至る最短路の長さ] が成り立つ
- それ以外の点では $d(i) = \infty$ で，命題は成立

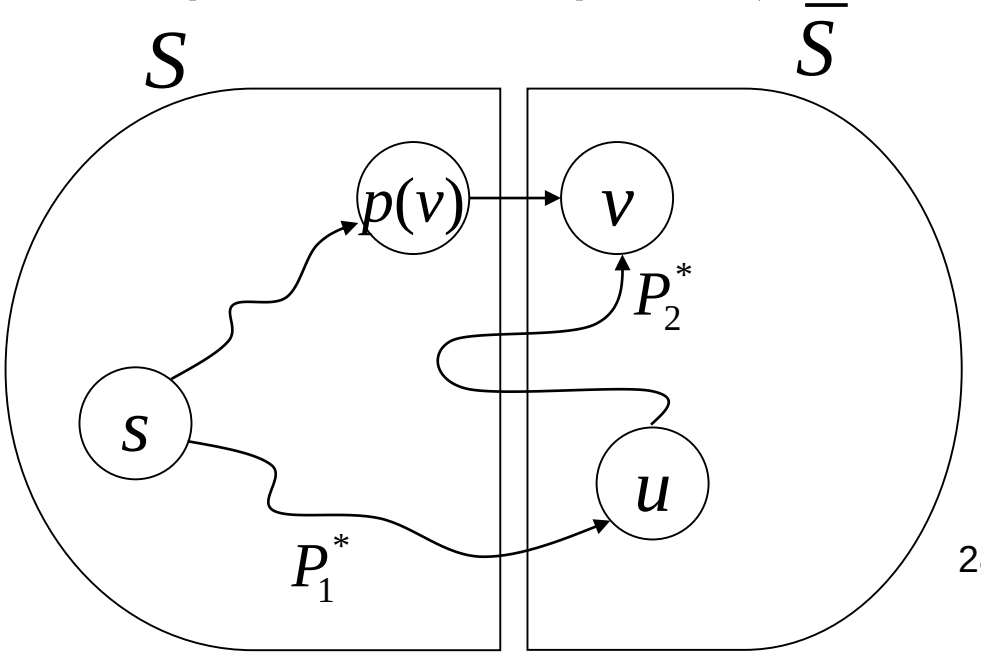


- 帰納法の仮定:ある反復に入った時点で命題成立
- ステップ(1)で $v \in \overline{S}$ が選ばれたときに，v に対し
  $d(v)$ = [S に含まれる節点のみを経由して s から v に至る最短路の長さ]
  = [s から v への最短路の長さ] (3.4)
  それ以外の点に対して (3.3) が成り立つことを示す
- (3.4)を背理法で示す．s からv への最短路をP^*とし，その長さが $d(v)$ と異なるとする
- アルゴリズムの構成法より，各節点 i に対し，$d(i)$ は s から i へのある路の長さを表しているので，上の仮定は $d(v) >$ [路 P^* の長さ] を意味する

- $v \in \overline{S}$なので，(3.3) より，s から v への真の最短路 P^* は，途中で $\overline{S}$ の点を少なくとも1つ経由する
- P^* において初めて現れる $\overline{S}$ の点を u とすると，P^* は P_1^* と P_2^* に分解できる
- 最適性の原理より，P_1^* は s から u への最短路
- P_1^* は S の点のみを経由しているので，(3.3) より
  $d(u)$ = [路 P_1^* の長さ] が言える



- 各枝の長さは非負なので,
  [路 P^* の長さ] $\geq$ [路 P_1^* の長さ]
- よって, $d(v) > d(u)$ $(d(v) >$ [路 P^* の長さ] より)
- ところが, $d(v) = \min\{d(i) \mid i \in \bar{S}\}$ と $u \in \bar{S}$ より
  $d(v) \leq d(u)$ となり, 矛盾
- よって, $d(v)$は s から v への最短路の長さに等しい
- (3.3) より,その路は S の節点のみを経由するので,
  (3.4) が成り立つ



- 反復が終了した時点で (3.3) が保たれていることを示す
- 反復終了時の S を $S^+ = S \cup \{v\}$ と書く
- アルゴリズムのステップ(2) を実行したとき

$$j \in \overline{S^+} \Rightarrow d(j) = \text{[}S^+ \text{に含まれる節点のみを経由して } s \text{ から } j \text{ に至る最短路の長さ]}$$

を示す

- S^+ に含まれる節点のみを経由して s から j に至る最短路は次の3つの場合が考えられる

(a) v を経由しない，つまり S の節点のみを経由

(b) j に到達する直前に v を経由

(c) v を経由し，その後さらに S の節点をいくつか通って j に到達

- (c) はありえない．なぜなら，そのような路が最短路の場合，j の直前の節点を k とすると，最適性の原理より，その路の k までの部分は sから kの最短路
- k はそれ以前の反復で S に入っているので，S の節点のみを経由する s から k への最短路が存在するはず
- よって，s から j への最短路は(a) か (b) のどちらか
- ステップ(2)で，$d(j) > d(v) + a_{vj}$ ならば $d(j)$ を $d(v) + a_{vj}$ で置き換えるが，これは (a) よりも(b)の方が路が短いときに最短路を置き換えることに対応
- 以上より

$$j \in \overline{S^+} \Rightarrow d(j) = \begin{array}{l}[S^+ \text{に含まれる節点のみを経由して} \\ s \text{から} j \text{に至る最短路の長さ}]\end{array}$$

- 次の反復に入ったときも命題が成り立つことが示された
- 各反復が終了するたびに，$\bar{S}$ のどれか1つの節点が S に入るので，アルゴリズムの反復の回数は節点数 n に等しい
- ステップ(1) の計算量は各反復で $O(\log n)$，全体で $O(n \log n)$
- ステップ(2) では，ネットワークの各枝はアルゴリズムを通して高々1回だけ処理されるのでステップ(2)の計算量は全体で $O(m \log n)$　(m: 枝数)
- 以上より，ダイクストラ法の計算量は $O(m \log n)$

- 注: 上のアルゴリズムを実行するには, ヒープからの削除を実現する必要がある.
  - サイズ n の配列を用いて, v の格納されているヒープの場所を管理する

- d の更新を挿入と削除ではなく, フィボナッチヒープのdecrease_keyで行うと, 総計算量は
  $O(m + n \log n)$

最小木問題

- 節点集合 $V = \{1,2,\ldots,n\}$ と枝集合 $\{e_1,e_2,\ldots,e_m\}$ をもつ連結無向グラフ $G = (V,E)$ に対して，G と同じ節点集合 V を持ち，E の部分集合 T を枝集合とするグラフ $G' = (V,T)$ を考える
- 各節点 $i \in V$ は少なくとも1つの枝 $e_j \in T$ の端点になっているとする
- G' が閉路を含まない連結グラフ，すなわち木になっているならば，G' を G の全域木という
- 各枝 $e_i \in T$ に長さ a_i が与えられているとき，$\sum_{e_i \in T} a_i$ を全域木の長さと定義する
- 長さ最小の全域木を求める問題

貪欲 (greedy) アルゴリズム

- アルゴリズムの各ステップで，その時点で最善と思われる選択を常に行うアルゴリズム
- 局所的に最善な選択が大局的に最善な解を導く場合には最適解を与える
- 貪欲アルゴリズムで解ける問題は比較的簡単な問題といえる

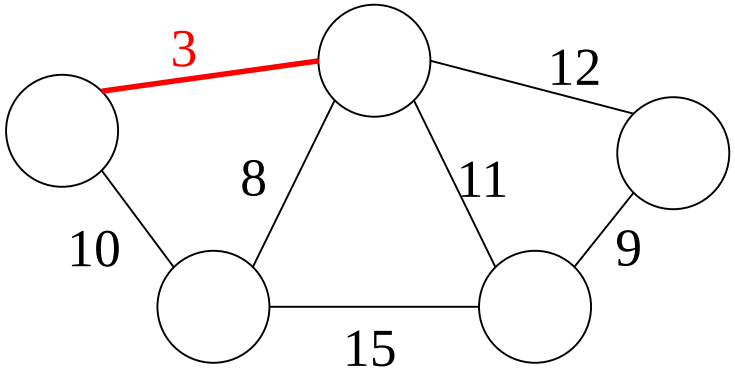
クラスカル法

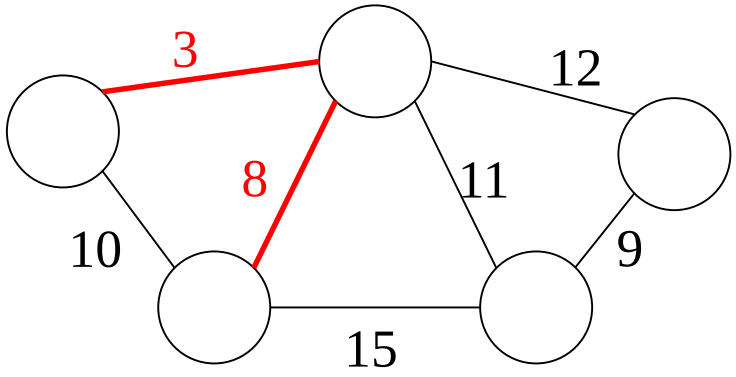
- 枝を長さの短い順に1つずつ選ぶ
- それを前に選んだ枝の集合に付け加えたときに閉路を生じないならば，その枝を付け加える

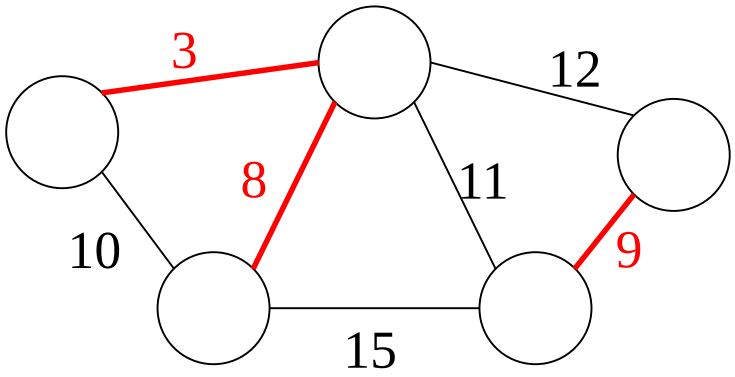
(0) グラフ G の枝を短い順に並べ，枝の番号を付け替える．$A = \{e_1\}, k = 2$ とおく．

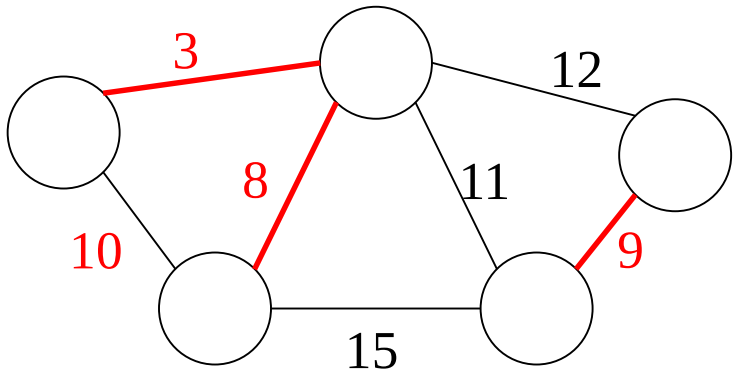
(1) 枝集合 $A \cup \{e_k\}$ が閉路を含まないならば，
$A := A \cup \{e_k\}$ とする

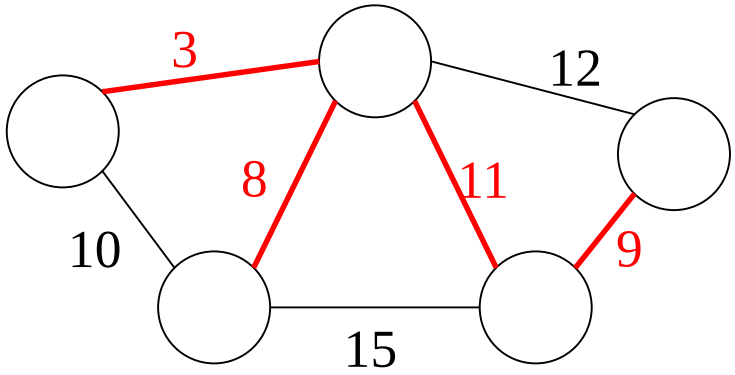
(2) A が G の全域木になっていれば計算終了．
さもなければ $k := k + 1$ としてステップ(1)へ戻る


3
12
8
11
10
9
15


3
12
8
11
10
9
15


3
12
8
11
10
9
15


3
12
8
11
10
9
15


3
12
8
11
10
9
15

- クラスカル法の計算途中では，枝集合 T は一般に複数の木の集まりになっている．このような集合を森と呼ぶ．
- どの枝を付け加えても閉路ができてしまうような森を，全域森と呼ぶ．
- クラスカル法の正当性を背理法で証明する．
- 得られた全域木を $T = \{e_{l_1}, e_{l_2}, \ldots, e_{l_{n-1}}\}$ とし，枝は $e_{l_1}, e_{l_2}, \ldots, e_{l_{n-1}}$ の順に付け加えられたとする．
- $a_{l_1} \leq a_{l_2} \leq \cdots \leq a_{l_{n-1}}$ が成り立つ
- T より長さの短い全域木 $T^* = \{e_{q_1}, e_{q_2}, \ldots, e_{q_{n-1}}\}$ が別に存在したとする．
  $a_{q_1} \leq a_{q_2} \leq \cdots \leq a_{q_{n-1}}$ とする

- T^* の長さは T よりも短いので，$a_{q_i} < a_{l_i}$ となる i が存在する.
- そのような i の中で最小のものを r とすると

$$\begin{cases} a_{q_1} \geq a_{l_1} \\ \vdots \\ a_{q_{r-1}} \geq a_{l_{r-1}} \\ a_{q_r} < a_{l_r} \end{cases}$$

- 枝集合 $\tilde{T} = \{e_{l_1}, \cdots, e_{l_{r-1}}, e_{q_1}, \cdots, e_{q_r}\}$ からなる部分グラフ $\tilde{G}$ を考える.
  ただし，$\tilde{T}$ には枝が重複して含まれている可能性もある. また，一般に $\tilde{G}$ は連結とは限らない.

- 命題：枝集合 $\{e_{l_1},\cdots,e_{l_{r-1}}\}$ は部分グラフ $\tilde{G}$ の全域森である．
- 証明：枝集合 $\{e_{l_1},\cdots,e_{l_{r-1}}\}$ に枝を付け加える段階で，$e_{q_1},\cdots,e_{q_r}$ ではなくそれより長い e_{l_r} が選ばれているということは，$e_{q_1},\cdots,e_{q_r}$ のどれを追加しても閉路ができることを意味する．(証明終)

- 枝集合 $\{e_{l_1},\cdots,e_{l_{r-1}}\}$ が $\tilde{G}$ の全域森であるため，$\tilde{G}$ の任意の森は r 本以上の枝を含まない
- 一方，$\{e_{q_1},\cdots,e_{q_r}\}$ は G に対する全域木 T^* の枝の一部であり，$\tilde{G}$ でも森になっている．しかしその枝数は r であるため，矛盾．
- 以上より，クラスカル法で得られた T は最小木．

アルゴリズムの効率的な実現

- 以下の操作を効率的に実現する必要がある

  枝集合 $A \cup \{e_k\}$ が閉路を含まないならば,

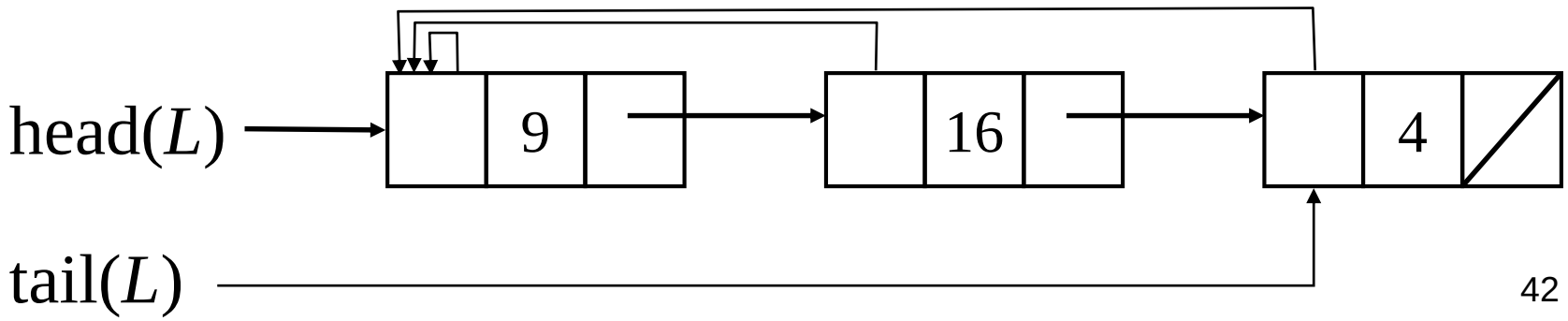
  $A := A \cup \{e_k\}$ とする
- $e_k = (u,v)$ の両端点 u, v が A の異なる連結成分に属している $\Leftrightarrow A \cup \{e_k\}$ は閉路を含まない
- 「互いに素な集合のためのデータ構造」を用いる

互いに素な集合のためのデータ構造

- 互いに素な動的集合の集合$S=\{S_1, S_2, \ldots, S_k\}$を扱う
- make_set(x):唯一の要素 x をもつ新しい集合を作る
  - x がすでに別の集合に含まれていてはいけない
- union(x, y): x と y を含む集合 S_x と S_y を合併し,
  それらの和集合を作る. 元の集合はSから取り除く.
- find_set(x): x を含む集合の代表元へのポインタを返す
- make_setの回数 n と全操作の総実行回数 m で
  評価する.
  - unionの回数は n–1 以下

連結リストによる表現

- 各集合を連結リストで表現する
- リストの先頭のオブジェクトがその集合の代表元の役割をする
- 各オブジェクトは集合の要素，次のオブジェクトへのポインタ，代表元に戻るポインタを持つ
- 各リストは代表元へのポインタ head とリストの最後の要素へのポインタ tail を持つ



- make_set(x) と find_set(x) は簡単で O(1) 時間
- union(x,y) は?
- x のリストを y のリストの最後に追加する場合
  - x のリストにあった各オブジェクトの代表元へのポインタを書き換える必要がある
  - x のリストの長さに比例する時間がかかる
- $\Theta(m^2)$ 時間かかる長さ m の操作列が存在
  - 初めに n 回の make_set を実行
  - 次に n–1 回の union を実行 ($m = 2n$–1)

- このアルゴリズムはどこが悪いのか
  - union(x,y) で常に x を y の末尾に追加しているので, x が長いと時間がかかる
- x と y の短いほうを長いほうの末尾に追加すれば計算量を抑えられる?
  - 長さ $n/2$ 同士のリストの併合は $\Theta(n)$ 時間かかるので最悪計算量は同じ
- ならし計算量(amortized time complexity)で評価
  - m 回の操作全体での計算量で評価する
- 定理: 長さ m の make_set, union, find_set 操作の列の実行の計算量は, その中の n 回が make_set のとき, $O(m + n \log n)$ である.

証明: n 個の各要素に対し，代表元へのポインタが更新される回数の上界を求める.
ある要素 x において，その代表元ポインタが更新されるとき，x は常に小さい方の集合に属している.
従って，最初に x の代表元ポインタが更新された時，得られる集合のサイズは2以上.
代表元ポインタが k 回更新された後に得られる集合は，少なくとも 2^k 個の要素を持っている.
要素は n 個しかないので，$k \leq \log n$.

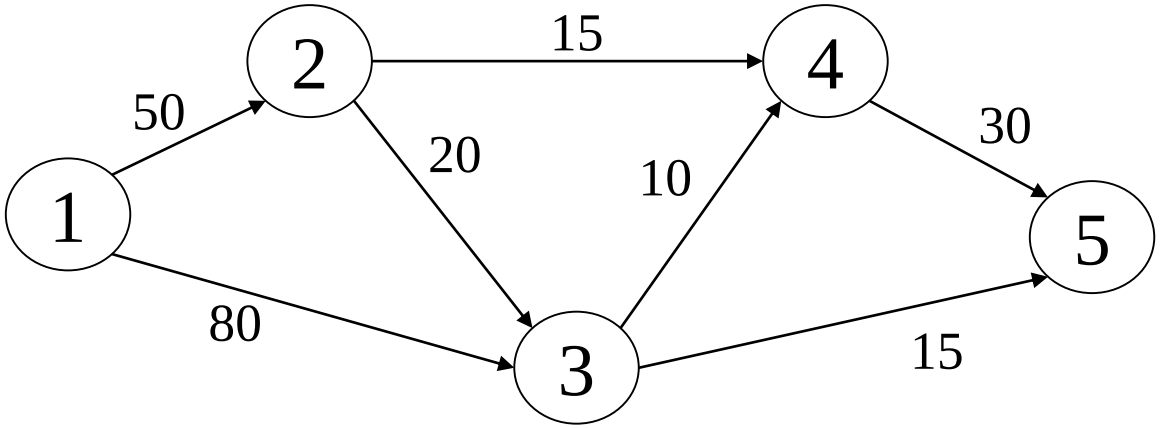
(続き)

n 個のオブジェクトを更新するのにかかる総時間は $O(n \log n)$.

make_set と find_set 操作は1回あたり $O(1)$,

m 回の操作全体で $O(m)$.

従って，列全体に対する総時間計算量は $O(m + n \log n)$.



クラスカルのアルゴリズム

mst_kruskal(G, w)

1. $A := \emptyset$
2. for $v \in V(G)$
3. 　　make_set(v)
4. 重み w の順に E の辺をソートする
5. for $(u,v) \in E$ (重みの小さい順)
6. 　　if find_set(u) $\neq$ find_set(v) then
7. 　　$A := A \cup \{(u,v)\}$
8. 　　union(u,v)
9. return A

クラスカル法の計算量

- 辺のソート: $O(m \log m)$
- make_set: n 回
- find_set: $2m$ 回
- union: $n-1$ 回
- 素な集合の操作全体で $O(m + n \log n)$
- $m = O(n^2)$ より
  クラスカル法の計算量は $O(m \log n)$