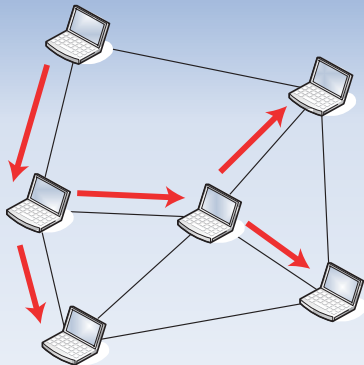


- 全域木 (スパニングツリー, Spanning Tree)
 - 連結無向グラフ $G = (V, E)$ の全域木
 - G の部分グラフ
 - V の全ての節点を連結した木
- 応用例: マルチキャスト, ブロードキャスト
 - 同一データを多数のユーザに配信



最小全域木

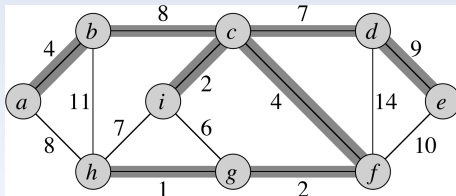
■ 全域木の辺集合 T

- T に含まれる辺の重みの合計 $w(T)$

$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

■ 最小全域木

- $w(T)$ が最小となる全域木
- 最小全域木を求めるアルゴリズム
 - **Kruskal** (クラスカル) のアルゴリズム
 - **Prim** (プリム) のアルゴリズム



成長法による最小全域木の生成

- 最小全域木を生成するための基礎アルゴリズム
 - 貪欲アルゴリズム
 - Kruskal, Prim のアルゴリズムは基礎アルゴリズムの具体的な実現方法

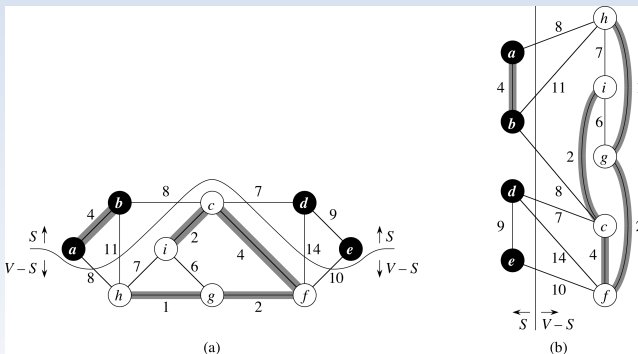
GENERIC-MST(G, w)

```
1:  $A = \emptyset$ 
2: while  $A$  は最小全域木ではない :
3:    $A$  に対して安全な辺  $(u, v)$  を発見する
4:    $A = A \cup \{(u, v)\}$ 
5: return  $A$ 
```

- ループ不変式 (ループの中で変化しない条件)
「各繰り返しの直前では、 A はある最小全域木の部分集合である」
- ループ不変式を維持しながら安全な辺を追加することにより最小全域木を求める
 - 安全な辺：ループ不変式に違反しない辺

成長法による最小全域木の生成

- 無向グラフ $G = (V, E)$ の**カット** $(S, V - S)$
 - $(u, v) \in E, u \in S, v \in V - S$ であるとき, (u, v) はカット $(S, V - S)$ と**交差**
- 辺集合 $A \subseteq E$
 - A のどの辺も $(S, V - S)$ と交差しないとき, $(S, V - S)$ は A を**尊重**する
- **軽い辺** : $(S, V - S)$ と交差する辺の中で重みが最小の辺



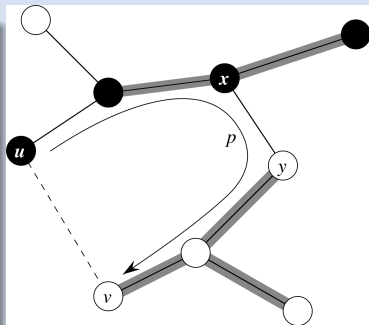
成長法による最小全域木の生成

定理 23.1

G のある最小全域木の (辺の) 部分集合 $A \subseteq E$, A を尊重する G のカットを $(S, V - S)$, $(S, V - S)$ と交差する軽い辺を (u, v) とする. (u, v) は A に対して安全である.

証明

- A を含む最小全域木 T
- $(u, v) \in T$ であるとき, (u, v) が安全であることは自明
- $(u, v) \notin T$ であるとき
 - 単純道 p と辺 (u, v) によって閉路ができる.
 $\Rightarrow p$ 上に $(S, V - S)$ と交差する $(x, y) \in T - A$ が少なくとも 1 つ存在する



最小全域木 T 上の辺だけを
表示

成長法による最小全域木の生成

証明 (続き)

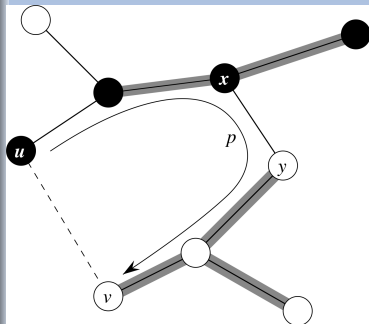
- (x, y) を削除すると T は二つの木に分断される
- $T' = T - \{(x, y)\} \cup \{(u, v)\}$ は全域木をなす
- $(x, y), (u, v)$ はカット $(S, V - S)$ と交差し, (u, v) は軽い辺であるから,

$$w(u, v) \leq w(x, y)$$

従って,

$$w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$$

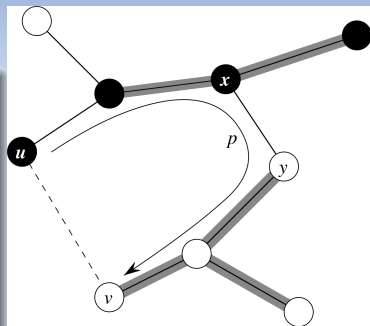
- T は最小全域木なので, $w(T) \leq w(T')$
 $\Rightarrow w(T') = w(T)$



最小全域木 T 上の辺だけを
表示

証明 (続き)

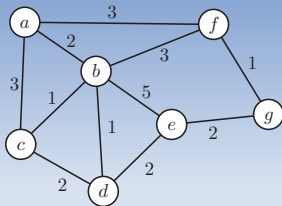
- 従って, T' も最小全域木である
- $A \subseteq T$ かつ $(x, y) \notin A$ より, $A \subseteq T'$
 $\Rightarrow A \cup \{(u, v)\} \subseteq T'$
つまり, A に (u, v) を加えた集合も最小
全域木 T' の部分集合である
 $\Rightarrow (u, v)$ は A に対して安全である.



最小全域木 T 上の辺だけを
表示

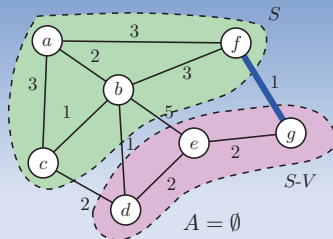
- 定理 23.1 は, *GENERIC-MST* アルゴリズムにおける安全な辺の発見方法を示す
- A を辺集合とするグラフ $G_A = (V, A)$
 - 複数の木を含む森を表す.
- 初期状態 $A = \emptyset$
 - A は辺を含まない $\Rightarrow G_A$ は全ての木が単一節点からなる森
- ループは, 安全な辺 (u, v) により異なる木を連結する
 - ループ回数: $|V| - 1$ 回
 - 安全な辺が発見される度に木の個数が 1 減る

成長法による最小全域木の生成

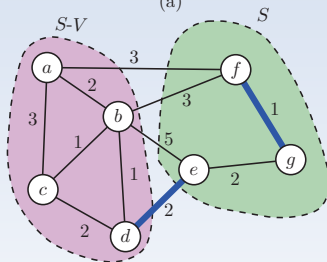


グラフ G

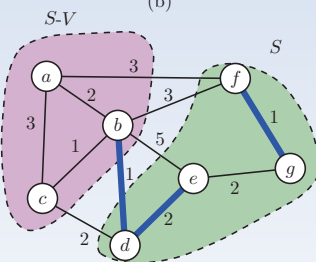
(a)



(b)

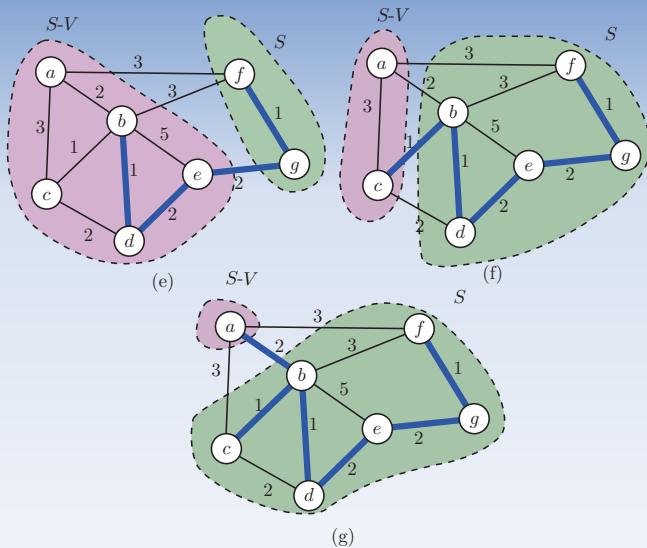


(c)



(d)

成長法による最小全域木の生成



系 23.2

$G = (V, E)$ を E の上で実数値重み関数 w が定義されている連結無向グラフとする. G のある最小全域木の部分集合を $A \subseteq E$, 森 $G = (V, A)$ の任意の連結成分 (木) を $C = (V_C, E_C)$ とする. (u, v) が C と G_A の他の連結成分を連結する軽い辺なら, (u, v) は A に対して安全である.

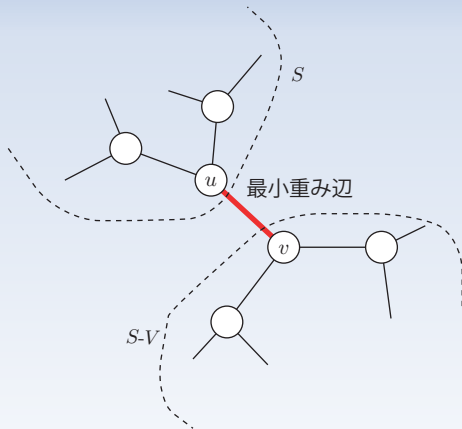
証明

- C と G_A に含まれる他の木は連結ではない
⇒ $(V_C, V - V_C)$ は A を尊重する.
- (u, v) はカット $(V_C, V - V_C)$ に対する軽い辺である.
- 定理 23.1 より (u, v) は A に対して安全である.

練習問題 23.1-1

(u, v) をグラフ $G = (V, E)$ の最小重み辺とする. (u, v) を含む G の最小全域木が存在することを示せ.

- 初期状態 $A = \emptyset$ を考える. 集合 S を u を含み v を含まないように設定する. このとき, (u, v) はカット $(S, V - S)$ に交差し, 最小重みであるため安全な辺として選択することができる.



- 互いに素な集合族のためのデータ構造
 - Kruskal のアルゴリズムのための準備
 - 第 21 章参照 (詳細は省く)
- 集合族：集合の集合

$$\mathcal{S} = \{S_1, S_2, \dots, S_k\}$$

- S_i ($i = 1, 2, \dots, k$) は集合
- 代表元
 - 各集合 S_i は代表元によって識別する
 - 例：最小要素
- 互いに素な集合 S_i, S_j ($i \neq j$)

$$S_i \cap S_j = \emptyset$$

互いに素な集合族のためのデータ構造

■ $MAKE-SET(x)$

- x を唯一の要素として持つ新しい集合を生成する

■ $UNION(x, y)$

- x を含む動的集合 S_x と y を含む動的集合 S_y を統合し、これらの和集合である新しい集合を生成する.
- 例: $S_x = \{a, b, x\}$, $S_y = \{c, d, e, y\}$

$$UNION(x, y) \Rightarrow \{a, b, c, d, e, x, y\}$$

■ $FIND-SET(x)$

- x を含む集合の代表元へのポインタを返す
- 例: 集合 $S_1 = \{v, w, x, y, z\}$

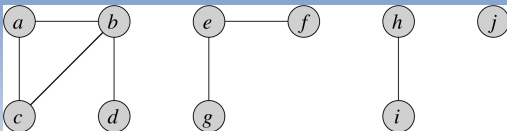
$$\begin{aligned} FIND-SET(v) &= FIND-SET(w) \\ &= FIND-SET(x) \\ &= FIND-SET(y) \\ &= FIND-SET(z) \end{aligned}$$

- 応用：無向グラフ $G = (V, E)$ の連結成分を求める

CONNECTED-COMPONENTS(G)

- 1: **for** 各頂点 $v \in G.V$:
- 2: $MAKE-SET(v)$
- 3: **for** 各辺 $(u, v) \in G.E$:
- 4: **if** $FIND-SET(u) \neq FIND-SET(v)$:
- 5: $UNION(u, v)$

互いに素な集合族のためのデータ構造



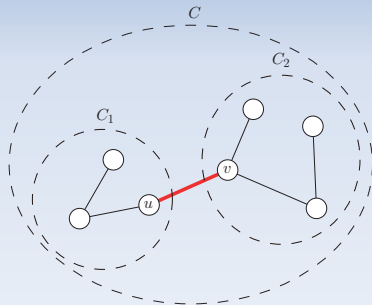
(a)

| Edge processed | Collection of disjoint sets | | | | | | | | | |
|----------------|-----------------------------|-------|-----|-----|---------|-----|-----|-------|-----|-----|
| initial sets | {a} | {b} | {c} | {d} | {e} | {f} | {g} | {h} | {i} | {j} |
| (b,d) | {a} | {b,d} | {c} | | {e} | {f} | {g} | {h} | {i} | {j} |
| (e,g) | {a} | {b,d} | {c} | | {e,g} | {f} | | {h} | {i} | {j} |
| (a,c) | {a,c} | {b,d} | | | {e,g} | {f} | | {h} | {i} | {j} |
| (h,i) | {a,c} | {b,d} | | | {e,g} | {f} | | {h,i} | | {j} |
| (a,b) | {a,b,c,d} | | | | {e,g} | {f} | | {h,i} | | {j} |
| (e,f) | {a,b,c,d} | | | | {e,f,g} | | | {h,i} | | {j} |
| (b,c) | {a,b,c,d} | | | | {e,f,g} | | | {h,i} | | {j} |

(b)

Kruskal のアルゴリズム

- 無向グラフ $G = (V, E)$ の辺を重みの昇順に並べる
 - 並べ替えた順に各辺を辺集合 A に加えるか判断する
- 木に属する節点を互いに素な集合族のためのデータ構造により表す
- まだ判定しない辺の中で最小の重みを持つ辺 (u, v)
 - (u, v) により接続される (互いに素な) 最小全域木の部分木 $C_1 = (V_{C_1}, E_{C_1})$, $C_2 = (V_{C_2}, E_{C_2})$
 - 系 23.2 より, (u, v) は安全な辺
 - $UNION(u, v)$ により C_1 と C_2 を C へと統合
 - 部分木 C は他の部分木とは互いに素

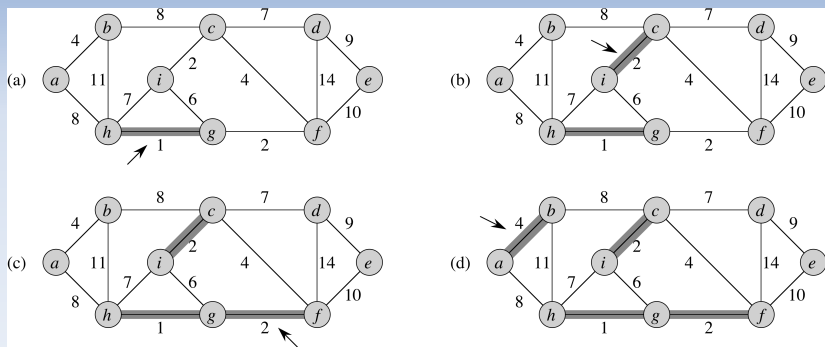


MST-KRUSKAL(G, w)

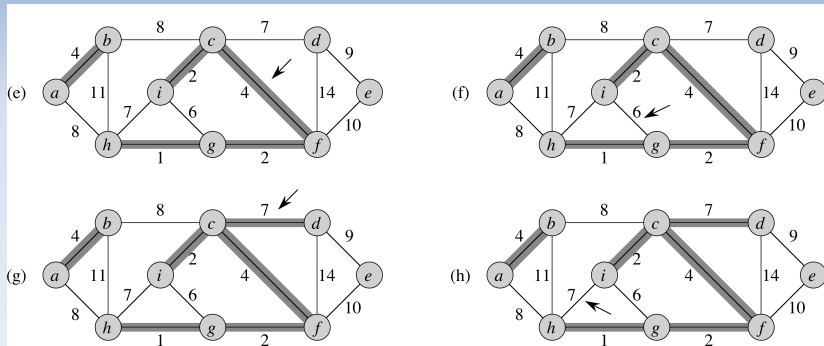
```
1:  $A = \emptyset$ 
2: for 各頂点  $v \in G.V$  :
3:   MAKE-SET( $v$ )
4: 重み  $w$  の非減少順で,  $G.E$  の各辺をソートする.
5: for 重みの非減少順で, 各辺  $(u, v) \in G.E$  :
6:   if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ ) :
7:      $A = A \cup \{(u, v)\}$ 
8:     UNION( $u, v$ )
9: return  $A$ 
```

- 2行目, 3行目: 節点 v のみを要素として含む木を生成
- 6行目: *FIND-SET*(u) \neq *FIND-SET*(v) ならば, (u, v) を加えても

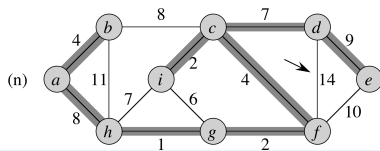
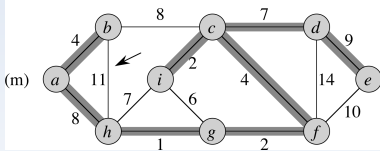
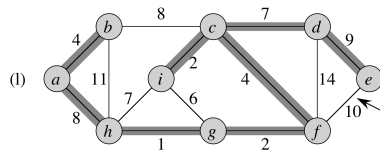
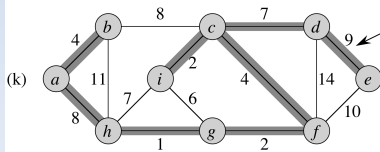
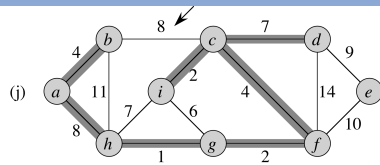
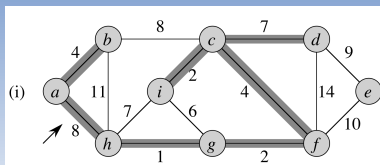
Kruskal のアルゴリズム



Kruskal のアルゴリズム

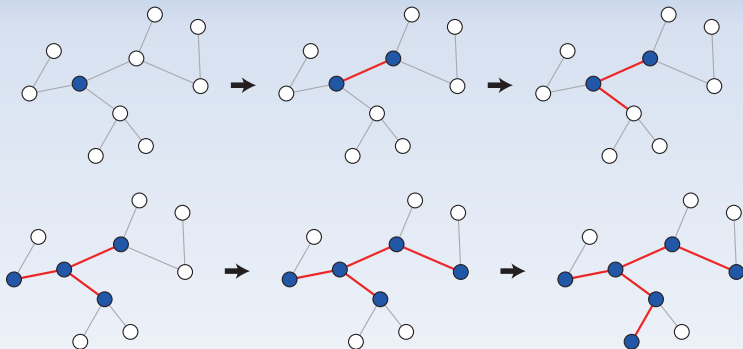


Kruskal のアルゴリズム

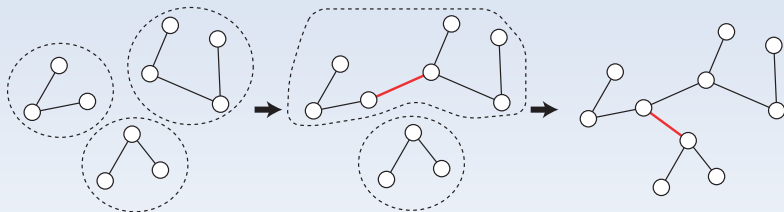


■ Prim のアルゴリズム

- 辺集合 A は常に一つの木を含む
- 各ステップで孤立点を A に加えていく
 - A と孤立点を連結する軽い辺を A に加える



- Kruskal のアルゴリズムと Prim のアルゴリズムの違い
- Kruskal のアルゴリズム
 - 辺集合 A には複数の非連結な木が含まれる (森)
 - 唯一の節点を含む場合 (孤立点) も木と考える
 - 各ステップで安全な辺を加えていくことにより, 異なる木を連結していく



MST-PRIM(G, w, r)

```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = \text{NIL}$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = \text{EXTRACT-MIN}(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```

- グラフ G , 重み w に対して, r を根 (開始点) として最小全域木を発見する
- 1~4 行目: 初期設定
 - r 以外の節点
 - $key = \infty$
 - $\pi = \text{NIL}$
 - 根
 - $key = \infty$
- 5 行目: Q は min 優先度付きキュー
 - キーの最小値を持つ要素が先頭になるようにソートされたキュー
 - ヒープソートにより実現

MST-PRIM(G, w, r)

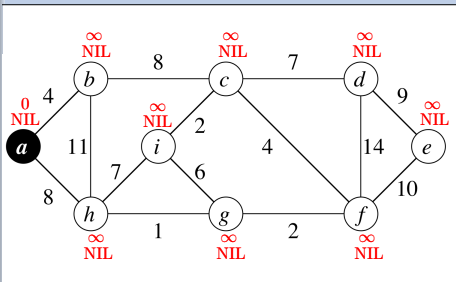
```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = \text{NIL}$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = \text{EXTRACT-MIN}(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:        $v.\pi = u$   
11:        $v.key = w(u, v)$ 
```

- 節点 v のキーは, v と木に属する頂点とを結ぶ辺の最小重み
- 7 行目: 最小のキーを持つ節点 u を取り出す
- 8 行目: u の各隣接節点 $v \in G.Adj[u]$ に対し,
 - 9~11 行目: v が Q の要素でかつ $w(u, v) < v.key$ ならば v の先行点とキーを更新する

Prim のアルゴリズム

$MST-PRIM(G, w, r)$

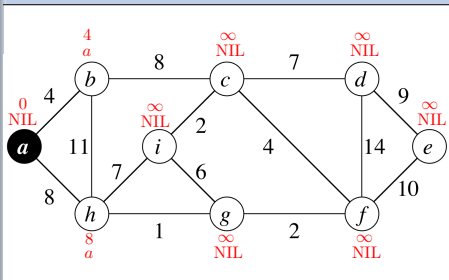
```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = NIL$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = EXTRACT-MIN(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```



Prim のアルゴリズム

MST-PRIM(G, w, r)

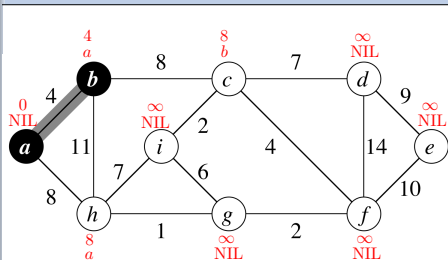
```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = \text{NIL}$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = \text{EXTRACT-MIN}(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```



Prim のアルゴリズム

$MST-PRIM(G, w, r)$

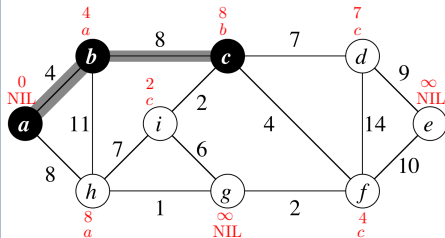
```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = NIL$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = EXTRACT-MIN(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```



Prim のアルゴリズム

$MST-PRIM(G, w, r)$

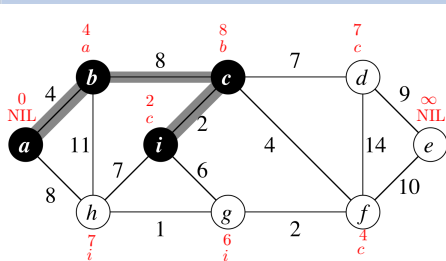
```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = NIL$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = EXTRACT-MIN(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```



Prim のアルゴリズム

$MST-PRIM(G, w, r)$

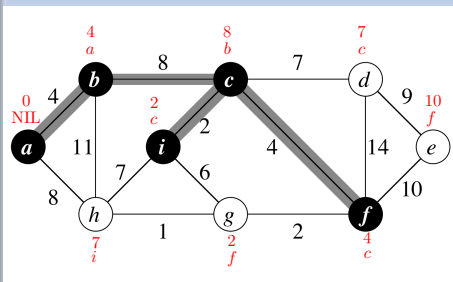
```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = \text{NIL}$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = \text{EXTRACT-MIN}(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```



Prim のアルゴリズム

$MST-PRIM(G, w, r)$

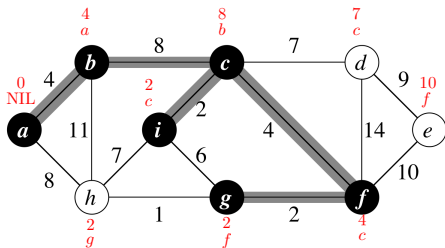
```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = NIL$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = EXTRACT-MIN(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```



Prim のアルゴリズム

$MST-PRIM(G, w, r)$

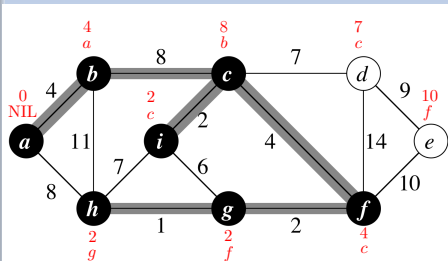
```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = NIL$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = EXTRACT-MIN(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```



Prim のアルゴリズム

$MST-PRIM(G, w, r)$

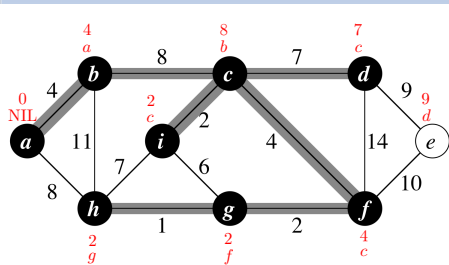
```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = NIL$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = EXTRACT-MIN(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```



Prim のアルゴリズム

$MST-PRIM(G, w, r)$

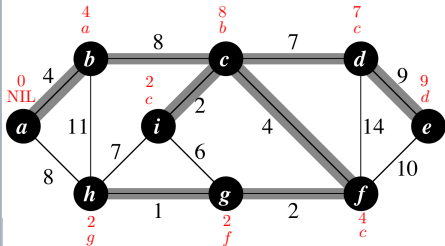
```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = NIL$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = EXTRACT-MIN(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```



Prim のアルゴリズム

$MST-PRIM(G, w, r)$

```
1: for 各  $u \in G.V$  :  
2:    $u.key = \infty$   
3:    $u.\pi = NIL$   
4:  $r.key = 0$   
5:  $Q = G.V$   
6: while  $Q \neq \emptyset$  :  
7:    $u = EXTRACT-MIN(Q)$   
8:   for 各  $v \in G.Adj[u]$  :  
9:     if  $v \in Q$  かつ  $w(u, v) < v.key$  :  
10:       $v.\pi = u$   
11:       $v.key = w(u, v)$ 
```



データ構造とアルゴリズム II

単一始点最短路問題

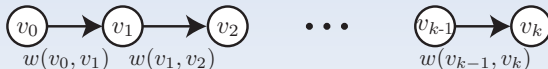
松田 崇弘

システムデザイン学部情報科学科

2019 年 6 月 21 日

- 有向グラフ $G = (V, E)$
- 辺に対する重み関数 $w : E \rightarrow \mathbb{R}$
 - 辺 $(u, v) \in E$ の重み $w(u, v)$
- 道 (経路) $p = \langle v_0, v_1, \dots, v_k \rangle$
 - p の重み $w(p)$

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i)$$



- 節点 u から節点 v への経路 p が存在するとき, u は v へ到達可能であると言い, $u \overset{p}{\rightsquigarrow} v$ と表す.

最短路問題

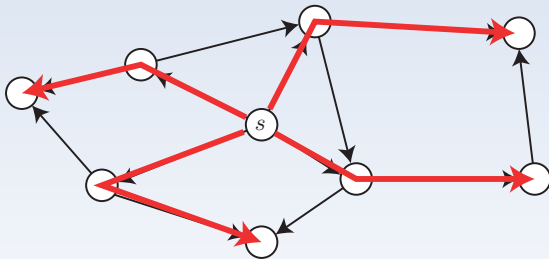
- 節点 u から節点 v への最短路重み $\delta(u, v)$

$$\delta(u, v) = \begin{cases} \min\{w(p) \mid u \overset{p}{\rightsquigarrow} v\} & u \text{ から } v \text{ への道が存在するとき} \\ \infty & \text{それ以外の場合} \end{cases}$$

- u から v への最短路: $w(p) = \delta(u, v)$ となる p

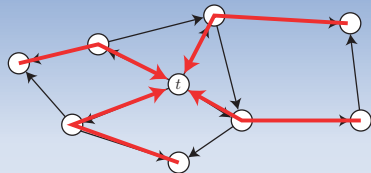
- 単一起点最短路問題

- グラフ $G = (V, E)$ と始点 $s \in V$ が与えられたとき, s から各節点 $v \in V$ への最短路を求める問題

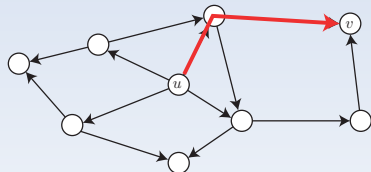


最短路問題

- 派生問題
- 単一目的地最短路問題
 - 各頂点 v から与えられた目的地 t までの最短路を求める問題
- 単一点対最短路問題
 - 与えられた2節点 u, v に対して, u から v への最短路を求める問題
- 全点对最短路問題
 - 全ての節点对 (u, v) に対して, u から v への最短路を求める問題



単一目的地最短路問題



単一点対最短路問題

最短路問題

■ 最短路の部分構造最適性

補題 24.1：最短路の部分道は最短路

節点 v_0 から v_k への最短路を $p = \langle v_0, v_1, \dots, v_k \rangle$, $0 \leq i \leq j \leq k$ を満たす任意の i と j に対して、頂点 v_i から頂点 v_j への p の部分道を $p_{i,j} = \langle v_i, v_{i+1}, \dots, v_j \rangle$ とする。このとき、 $p_{i,j}$ は v_i から v_j への最短路である。



- ある節点对間の最短路は、その経路上にある節点对の最短路を含んでいる。

証明

- 経路 p を $v_0 \xrightarrow{p_{0,i}} v_i \xrightarrow{p_{i,j}} v_j \xrightarrow{p_{j,k}} v_k$ と分割すると次式が得られる。

$$w(p) = w(p_{0,i}) + w(p_{i,j}) + w(p_{j,k})$$

証明 (続き)

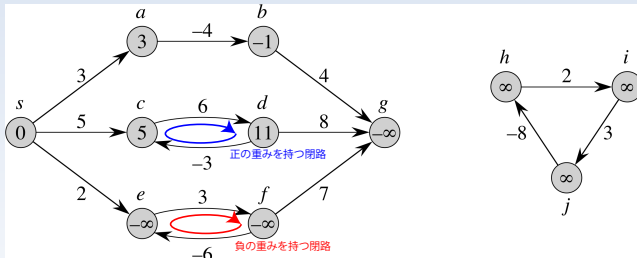
- $w_{p'_{i,j}} < w(p_{i,j})$ となる経路 $p'_{i,j}$ が存在すると仮定すると, 経路 $v_0 \xrightarrow{p_{0,i}} v_i \xrightarrow{p'_{i,j}} v_j \xrightarrow{p_{j,k}} v_k$ の重みは

$$w(p_{0,i}) + w(p'_{i,j}) + w(p_{j,k}) < w(p_{0,i}) + w(p_{i,j}) + w(p_{j,k}) = w(p)$$

となるため, v_0 から v_k までの経路で重みが $w(p)$ よりも小さいものが存在することになり, $w(p)$ が最短路であるという仮定に矛盾する.

最短路問題

- 負の重みを持つ辺が存在する場合
- 始点 s から経路 v への経路
 - 経路上に負の重みを持つ閉路が存在しなければ経路の重みを求めることができる.
 - 経路上に負の重みを持つ閉路が存在する場合, $\delta(s, v) = -\infty$ とする.
- 例:
 - $\delta(s, e) = \delta(s, f) = \delta(s, g) = -\infty$
 - $\delta(s, h) = \delta(s, i) = \delta(s, j) = \infty$
 - 負の閉路があっても到達できないので ∞ とする



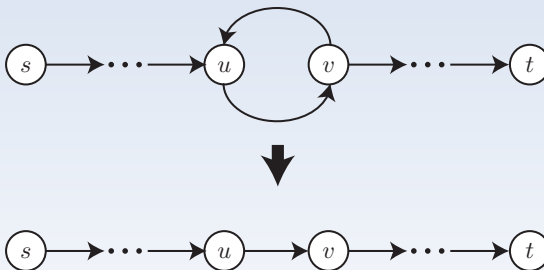
最短路問題

■ 最短路は閉路を含むか？

- 負の重みを持つ閉路は含まない
- 正の重みを持つ閉路 \Rightarrow 閉路の部分を削除することによって、より小さい重みを持つ閉路を作ることができる.
- 重み 0 の閉路が存在する場合、閉路を削除して同じ重みを持つ経路を考える.

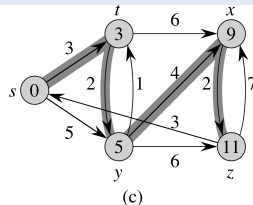
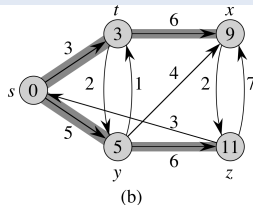
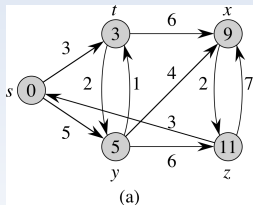
\Rightarrow 最短路は閉路を含まない考える

■ 節点数 $N = |V|$ のグラフの最短経路長は、高々 $N - 1$



最短路問題

- 有向グラフ $G = (V, E)$
- $s \in V$ を根とする **最短路木** $G' = (V', E')$
 - G' は G の有向部分グラフ
 - $V' \subseteq V, E' \subseteq E$
- G' が満たす条件
 - V' は G において、 s から到達可能な頂点の集合である
 - G' は s を根とする根付き木である
 - 全ての $v \in V'$ に対して、 G' における s から v への唯一の単純道が G における s から v への最短路である。



- 節点 $v \in V$ の属性
 - d : 始点 s からの距離
 - π : 節点 v の先行点
- 初期化
 - 先行点を NIL とする
 - $v \in V - \{s\}$ の始点 s からの距離を無限大とする

INITIALIZE-SINGLE-SOURCE(G, s)

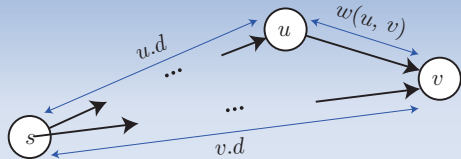
for 各頂点 $v \in G.V$:

$v.d = \infty$

$v.\pi = \text{NIL}$

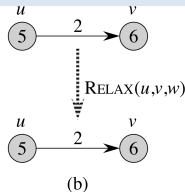
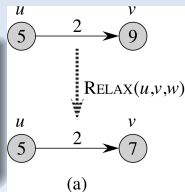
$s.d = 0$

- 辺 (u, v) の緩和 (relaxing)
 - u を経由して s から v への既知の最短経路を改善できるかを判定
 - 改善できるのであれば, $v.d$ と $v.\pi$ を更新



RELAX(u, v, w)

if $v.d > u.d + w(u, v)$:
 $v.d = u.d + w(u, v)$
 $v.\pi = u$



■ 最短路と緩和の性質 (証明省略, 第 24.5 節)

■ 三角不等式

- 任意の辺 $(u, v) \in E$ に対して $\delta(s, v) \leq \delta(s, u) + w(u, v)$ が成立する

■ 上界性

- 全ての頂点 $v \in V$ に対して $v.d \geq \delta(s, v)$ が常に成立する. (緩和によって経路が更新される場合) $v.d$ が値 $\delta(s, v)$ に達するとその後は変化しない.

■ 無経路性

- 頂点 s から v への道がなければ, $v.d = \delta(s, v) = \infty$ が常に成立する.

■ 経路緩和性

- $p = \langle v_0, v_1, \dots, v_k \rangle$ が $s = v_0$ から v_k への最短路で, p の辺が $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$ の順に緩和されたとき, $v_k.d = \delta(s, v_k)$ が成立する. この性質は他の任意の緩和とは無関係に成立する.

■ 先行点部分グラフ性

- 全ての $v \in V$ に対して $v.d = \delta(s, v)$ が成立すれば, 先行点部分グラフは s を根とする最短路木である.

- 単一始点最短路問題を解くアルゴリズム
 - 負の重みを持つ辺の存在を許す
 - 負の重みを持つ閉路が存在する場合、解が存在しないと出力

BELLMAN-FORD(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2: for  $i = 1$  to  $|G.V| - 1$  :
3:   for 各辺  $(u, v) \in G.E$  :
4:     RELAX( $u, v, w$ )
5: for 各辺  $(u, v) \in G.E$  :
6:   if  $v.d > u.d + w(u, v)$  :
7:     return FALSE
8: return TRUE
```

- 1 行目：初期化
- 2~4 行目：グラフの各辺の緩和を $|G.V| - 1$ 回繰り返す
- 5~8 行目：負の重みを持つ閉路の判定

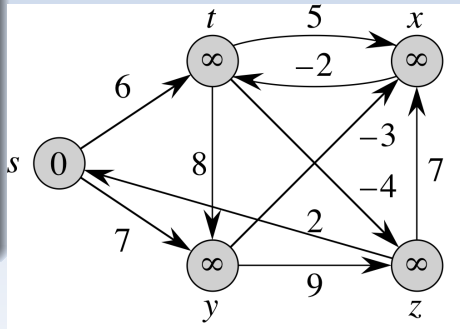
Bellman-Ford アルゴリズム

BELLMAN-FORD(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2: for  $i = 1$  to  $|G.V| - 1$  :
3:   for 各辺  $(u, v) \in G.E$  :
4:     RELAX( $u, v, w$ )
5: for 各辺  $(u, v) \in G.E$  :
6:   if  $v.d > u.d + w(u, v)$  :
7:     return FALSE
8: return TRUE
```

■ 辺の走査順

$(t, x), (t, y), (t, z), (x, t), (y, x),$
 $(y, z), (z, x), (z, s), (s, t), (s, y),$



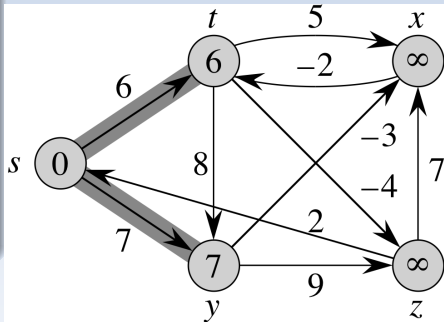
Bellman-Ford アルゴリズム

BELLMAN-FORD(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2: for  $i = 1$  to  $|G.V| - 1$  :
3:   for 各辺  $(u, v) \in G.E$  :
4:     RELAX( $u, v, w$ )
5: for 各辺  $(u, v) \in G.E$  :
6:   if  $v.d > u.d + w(u, v)$  :
7:     return FALSE
8: return TRUE
```

■ 辺の走査順

$(t, x), (t, y), (t, z), (x, t), (y, x),$
 $(y, z), (z, x), (z, s), (s, t), (s, y),$



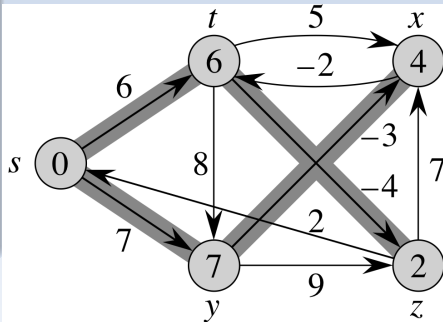
Bellman-Ford アルゴリズム

BELLMAN-FORD(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2: for  $i = 1$  to  $|G.V| - 1$  :
3:   for 各辺  $(u, v) \in G.E$  :
4:     RELAX( $u, v, w$ )
5: for 各辺  $(u, v) \in G.E$  :
6:   if  $v.d > u.d + w(u, v)$  :
7:     return FALSE
8: return TRUE
```

■ 辺の走査順

$(t, x), (t, y), (t, z), (x, t), (y, x),$
 $(y, z), (z, x), (z, s), (s, t), (s, y),$



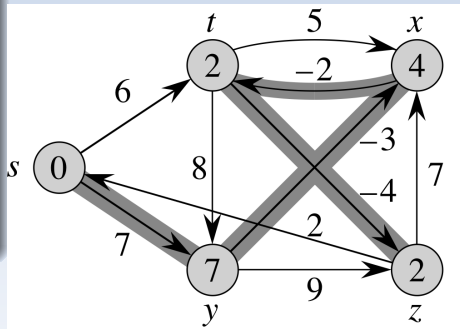
Bellman-Ford アルゴリズム

BELLMAN-FORD(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2: for  $i = 1$  to  $|G.V| - 1$  :
3:   for 各辺  $(u, v) \in G.E$  :
4:     RELAX( $u, v, w$ )
5: for 各辺  $(u, v) \in G.E$  :
6:   if  $v.d > u.d + w(u, v)$  :
7:     return FALSE
8: return TRUE
```

■ 辺の走査順

$(t, x), (t, y), (t, z), (x, t), (y, x),$
 $(y, z), (z, x), (z, s), (s, t), (s, y),$

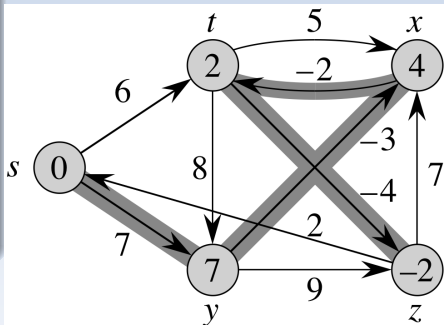


BELLMAN-FORD(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2: for  $i = 1$  to  $|G.V| - 1$  :
3:   for 各辺  $(u, v) \in G.E$  :
4:     RELAX( $u, v, w$ )
5: for 各辺  $(u, v) \in G.E$  :
6:   if  $v.d > u.d + w(u, v)$  :
7:     return FALSE
8: return TRUE
```

■ 辺の走査順

$(t, x), (t, y), (t, z), (x, t), (y, x),$
 $(y, z), (z, x), (z, s), (s, t), (s, y),$



補題 24.2

G は始点 s から到達可能な負閉路を含まないと仮定する.

BELLMAN-FORD の第 2~4 行の **for** 文を $|V| - 1$ 回繰り返した後, s から到達可能な全ての頂点 v に対して, $v.d = \delta(s, v)$ が成立する.

- 負閉路を含まない場合には, 第 2~4 行の **for** 文を繰り返すことにより, 最短経路が求まる.

証明

- s から v への最短路を $p = \langle v_0 = s, v_1, \dots, v_k = v \rangle$ とする.
- p は高々 $|V| - 1$ の辺を持つ.
 - $\Rightarrow k \leq |V| - 1$
 - \Rightarrow 2 行目の **for** 文内の各繰り返して全ての辺が緩和される
 - $\Rightarrow (v_{i-1}, v_i)$ は i 回目の繰り返して緩和される辺の一つ
 - \Rightarrow 経路緩和性から $v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$


Bellman-Ford アルゴリズム

■ 経路緩和性 (再掲)

- $p = \langle v_0, v_1, \dots, v_k \rangle$ が $s = v_0$ から v_k への最短路で, p の辺が $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$ の順に緩和されたとき, $v_k.d = \delta(s, v_k)$ が成立する. この性質は他の任意の緩和とは無関係に成立する.
- つまり, 最短路上の辺を始点 s から順番に緩和していけば最短路の重みを求めることができる.

■ 補題 24.2 の証明の解釈

- 最短路上の辺の順番に緩和していかななくても, $|V| - 1$ 回繰り返せば, 始点 s から順番に緩和したと見なすことができる.
- Bellman-Ford のアルゴリズムは緩和する辺の順番に依存しない
- 例

| | |
|--------|--|
| |  |
| | (s, v_1) (v_1, v_2) (v_2, v_3) (v_3, v) |
| 繰り返し 1 | 3 1 2 4 |
| 繰り返し 2 | 7 5 6 8 |
| 繰り返し 3 | 11 9 10 12 |
| 繰り返し 4 | 15 13 14 16 |

緩和する順番

系 24.3

各頂点 $v \in V$ に対して、 s から v への道が存在するための必要十分条件は、*BELLMAN-FORD* を G に適用したとき、終了時に $v.d < \infty$ が成立することである。

証明

- 「 s から v への道が存在 $\Rightarrow v.d < \infty$ 」
 s から v へ経路が存在するとき、経路上の辺 (u, v) が緩和される。
従って、 $v.d$ は必ず $v.d < \infty$ となる。
- 「 $v.d < \infty \Rightarrow s$ から v への道が存在」の証明
 \Leftrightarrow 「 s から v への道が存在しない $\Rightarrow v.d = \infty$ 」 (対偶)
 s から v への道が存在しない場合、無経路性より $v.d = \infty$ となる。

定理 24.4 (Bellman-Ford アルゴリズムの正当性)

- G が s から到達可能な負経路を含まなければ, アルゴリズムは TRUE を返し,
 - (A) 全ての頂点 $v \in V$ に対して $v.d = \delta(s, v)$
 - (B) 先行点部分グラフ G_π は s を根とする最短路木
- (C) G が s から到達可能な負経路を含めば, アルゴリズムは FALSE を返す.

証明

- G が始点 s から到達可能な負経路を含まないと仮定
 - v が s から到達可能ならば, 補題 24.2 から $v.d = \delta(s, v)$ が成立する.
 - v が s から到達可能でなければ, 無経路性から $v.d = \delta(v, d) = \infty$ が成立する.
 - \Rightarrow (A) は正しい
 - (A) の主張と先行点部分グラフ性から G_π は最短路木である.
 - \Rightarrow (B) は正しい

証明 (続き)

- (A) が成り立つとき、アルゴリズム終了時に全ての辺 $(u, v) \in E$ に対して次式が成り立つ.

$$\begin{aligned} v.d &= \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) \text{ (三角不等式より)} \\ &= u.d + w(u, v) \end{aligned}$$

従って、FALSE を返すことはない. 従って、アルゴリズムは TRUE を返す.

- G が始点 s から到達可能な負経路 $c = \langle v_0, v_1, \dots, v_k \rangle$ を含むと仮定する. このとき次式が成り立つ.

$$\sum_{i=1}^k w(v_{i-1}, v_i) < 0, \quad v_0 = v_k$$

証明 (続き)

- Bellman-Ford アルゴリズムが TRUE を返すと仮定する。
三角不等式より,

$$\begin{aligned}\delta(s, v_i) &= v_i.d \\ &\leq \delta(s, v_{i-1}) + w(v_{i-1}, v_i) \\ &= v_{i-1}.d + w(v_{i-1}, v_i) \\ \Rightarrow v_i.d &\leq v_{i-1}.d + w(v_{i-1}, v_i)\end{aligned}$$

経路 c に沿って, この不等式を加えると,

$$\begin{aligned}\sum_{i=1}^k v_i.d &\leq \sum_{i=1}^k (v_{i-1}.d + w(v_{i-1}, v_i)) \\ &= \sum_{i=1}^k v_{i-1}.d + \sum_{i=1}^k w(v_{i-1}, v_i)\end{aligned}$$

証明 (続き)

- $v_0 = v_k$ より,

$$\sum_{i=1}^k v_i.d = \sum_{i=1}^k v_{i-1}.d$$

- 系 24.3 より, $v_i.d$ は有限である. 従って,

$$0 \leq \sum_{i=1}^k w(v_{i-1}, v_i)$$

これは, $\sum_{i=1}^k w(v_{i-1}, v_i) < 0$ の仮定と矛盾する.
従って, (C) は正しい.

BELLMAN-FORD(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2: for  $i = 1$  to  $|G.V| - 1$  :
3:   for 各辺  $(u, v) \in G.E$  :
4:     RELAX( $u, v, w$ )
5: for 各辺  $(u, v) \in G.E$  :
6:   if  $v.d > u.d + w(u, v)$  :
7:     return FALSE
8: return TRUE
```

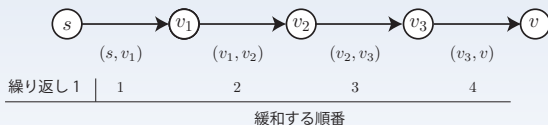
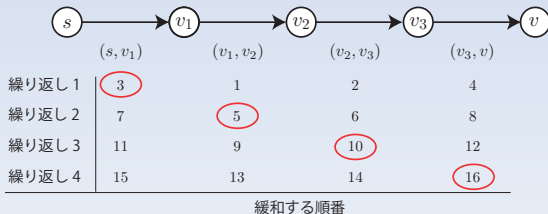
- Bellman-Ford アルゴリズムの実行時間 $O(VE)$ ($O(|V||E|)$)
 - 1 行目: $\Theta(V)$
 - 2~4 行目: $O(VE)$
 - 5~7 行目: $O(E)$
- Bellman-Ford アルゴリズムは、緩和する辺の順番には依存せず実行可能
⇒ 冗長な繰り返しも存在する

有向非巡回グラフにおける単一始点最短路

■ 経路緩和性

- 経路 $p = \langle v_0 = s, v_1, \dots, v_k \rangle$ に対して、始点に近い方から緩和していけば $v_k.d = \delta(s, v_k)$ が得られる。

- 経路緩和性より、始点 s からの始まる任意の経路に対して、始点 s に近い方から緩和できるように辺の順番を決めることができれば、冗長な繰り返しを減らせる



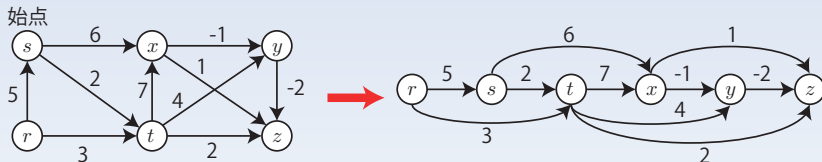
有向非巡回グラフにおける単一起点最短路

■ 有向非巡回グラフの場合

- トポロジカルソートにより、任意の $(u, v) \in E$ に対して、 $(u \text{ の番号}) \leq (v \text{ の番号})$ となるように節点に番号付けができる。
⇒ 任意の経路に対して、始点 s に近い方の辺から緩和を実行可能

- 例： $(r, s), (r, t), (s, t), (s, x), (t, x), (t, y), (t, z), (x, y), (x, z), (y, z)$ の順に緩和

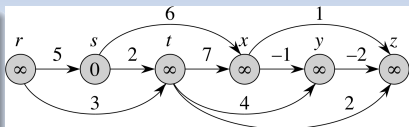
- 注意： s から r への経路は存在しない



有向非巡回グラフにおける単一始点最短路

DAG-SHORTEST-PATHS(G, w, s)

- 1: G の頂点をトポロジカルソートする.
- 2: *INITIALIZE-SINGLE-SOURCE*(G, s)
- 3: **for** トポロジカルソート順に, 各頂点 u :
- 4: **for** 各頂点 $v \in G.Adj[u]$:
- 5: $RELAX(u, v, w)$

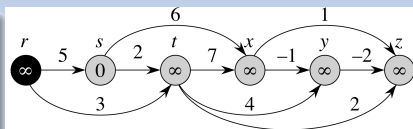


- トポロジカルソート後の状態

有向非巡回グラフにおける単一起点最短路

DAG-SHORTEST-PATHS(G, w, s)

- 1: G の頂点をトポロジカルソートする.
- 2: *INITIALIZE-SINGLE-SOURCE*(G, s)
- 3: **for** トポロジカルソート順に, 各頂点 u :
- 4: **for** 各頂点 $v \in G.Adj[u]$:
- 5: *RELAX*(u, v, w)

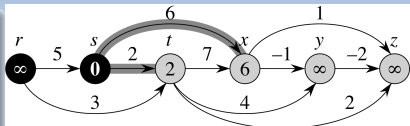


- 節点 r から出る辺を緩和
 - $s.d = 0$ は更新なし.
 - $t.d = \infty$, $r.d + w(r, t) = \infty$ より, $t.d$ も更新なし.

有向非巡回グラフにおける単一始点最短路

DAG-SHORTEST-PATHS(G, w, s)

- 1: G の頂点をトポロジカルソートする.
- 2: *INITIALIZE-SINGLE-SOURCE*(G, s)
- 3: **for** トポロジカルソート順に, 各頂点 u :
- 4: **for** 各頂点 $v \in G.Adj[u]$:
- 5: *RELAX*(u, v, w)



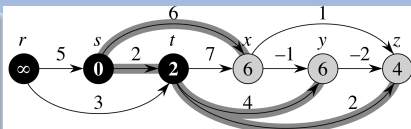
■ 節点 s から出る辺を緩和

- $t.d (= \infty) > s.d + w(s, t) = 2$ より, $t.d = 2$, $t.\pi = s$ に更新
- $x.d (= \infty) > s.d + w(s, x) = 6$ より, $x.d = 6$, $x.\pi = s$ に更新

有向非巡回グラフにおける単一起点最短路

DAG-SHORTEST-PATHS(G, w, s)

- 1: G の頂点をトポロジカルソートする.
- 2: *INITIALIZE-SINGLE-SOURCE*(G, s)
- 3: **for** トポロジカルソート順に, 各頂点 u :
- 4: **for** 各頂点 $v \in G.Adj[u]$:
- 5: *RELAX*(u, v, w)



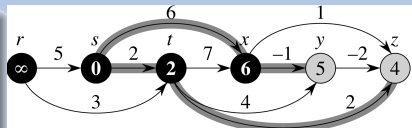
■ 節点 t から出る辺を緩和

- $x.d (= 6) < t.d + w(t, x) = 9$ より, $x.d = 6$ は更新せず
- $y.d (= \infty) > t.d + w(t, y) = 6$ より, $y.d = 6$, $y.\pi = t$ に更新
- $z.d (= \infty) > t.d + w(t, z) = 4$ より, $z.d = 4$, $z.\pi = t$ に更新

有向非巡回グラフにおける単一始点最短路

DAG-SHORTEST-PATHS(G, w, s)

- 1: G の頂点をトポロジカルソートする.
- 2: *INITIALIZE-SINGLE-SOURCE*(G, s)
- 3: **for** トポロジカルソート順に, 各頂点 u :
- 4: **for** 各頂点 $v \in G.Adj[u]$:
- 5: *RELAX*(u, v, w)



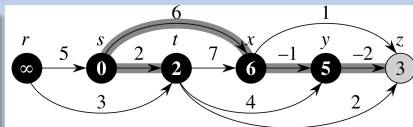
■ 節点 x から出る辺を緩和

- $y.d (= 6) < x.d + w(x, y) = 5$ より, $y.d = 5$, $y.\pi = x$ に更新
- $z.d (= 4) > x.d + w(x, z) = 7$ より, $z.d = 4$ は更新せず

有向非巡回グラフにおける単一起点最短路

DAG-SHORTEST-PATHS(G, w, s)

- 1: G の頂点をトポロジカルソートする.
- 2: *INITIALIZE-SINGLE-SOURCE*(G, s)
- 3: **for** トポロジカルソート順に, 各頂点 u :
- 4: **for** 各頂点 $v \in G.Adj[u]$:
- 5: $RELAX(u, v, w)$

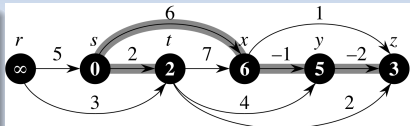


- 節点 y から出る辺を緩和
 - $z.d (= 4) > y.d + w(y, z) = 3$ より, $z.d = 3$, $z.\pi = y$ に更新

有向非巡回グラフにおける単一始点最短路

DAG-SHORTEST-PATHS(G, w, s)

- 1: G の頂点をトポロジカルソートする.
- 2: *INITIALIZE-SINGLE-SOURCE*(G, s)
- 3: **for** トポロジカルソート順に, 各頂点 u :
- 4: **for** 各頂点 $v \in G.Adj[u]$:
- 5: $RELAX(u, v, w)$



■ アルゴリズム終了

Dijkstra のアルゴリズム

- 全ての辺の重みが非負である場合に単一始点最短路問題を解く
 - Bellman-Ford のアルゴリズムよりも高速
 - 貪欲法

DIJKSTRA(G, w, s)

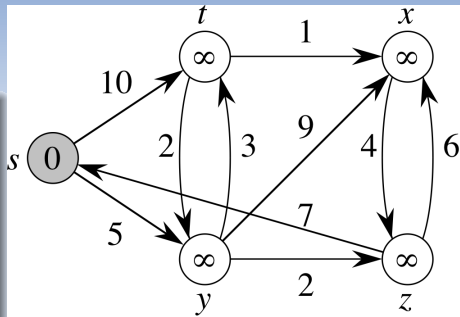
```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2:  $S = \emptyset$ 
3:  $Q = G.V$ 
4: while  $Q \neq \emptyset$  :
5:    $u = \text{EXTRACT-MIN}(Q)$ 
6:    $S = S \cup \{u\}$ 
7:   for 各頂点  $v \in G.Adj[u]$  :
8:     RELAX( $u, v, w$ )
```

- Q : min 優先度付きキュー
 - d 値 (距離) の昇順に優先度を付ける
- 5 行目 : 最小の距離を持つ節点 u を取り出す
- 7~8 行目 : u と u の隣接節点との辺を緩和する

Dijkstra のアルゴリズム

DIJKSTRA(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2:  $S = \emptyset$ 
3:  $Q = G.V$ 
4: while  $Q \neq \emptyset$  :
5:    $u = \text{EXTRACT-MIN}(Q)$ 
6:    $S = S \cup \{u\}$ 
7:   for 各頂点  $v \in G.Adj[u]$  :
8:     RELAX( $u, v, w$ )
```

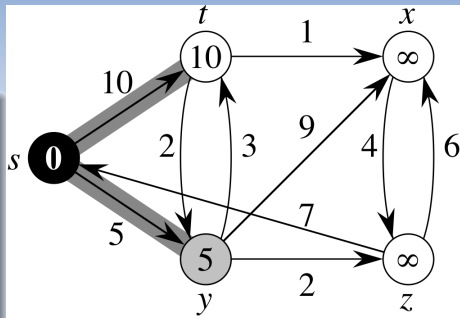


- 灰色の節点は Q より取り出した節点
- 黒色の節点は S の要素

Dijkstra のアルゴリズム

DIJKSTRA(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2:  $S = \emptyset$ 
3:  $Q = G.V$ 
4: while  $Q \neq \emptyset$  :
5:    $u = \text{EXTRACT-MIN}(Q)$ 
6:    $S = S \cup \{u\}$ 
7:   for 各頂点  $v \in G.Adj[u]$  :
8:     RELAX( $u, v, w$ )
```

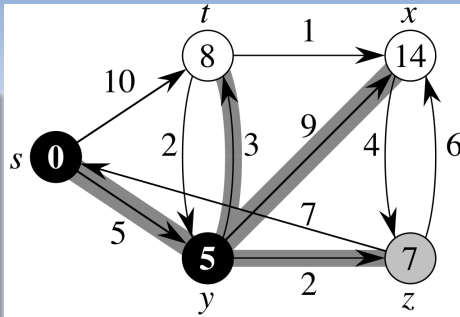


- 灰色の節点は Q より取り出した節点
- 黒色の節点は S の要素

Dijkstra のアルゴリズム

DIJKSTRA(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2:  $S = \emptyset$ 
3:  $Q = G.V$ 
4: while  $Q \neq \emptyset$  :
5:    $u = \text{EXTRACT-MIN}(Q)$ 
6:    $S = S \cup \{u\}$ 
7:   for 各頂点  $v \in G.Adj[u]$  :
8:     RELAX( $u, v, w$ )
```

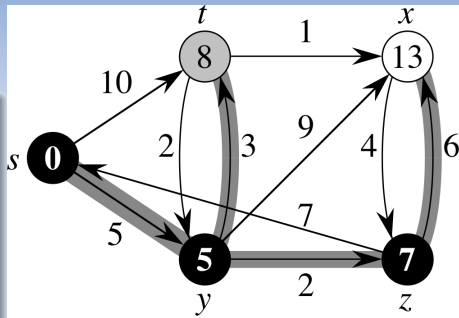


- 灰色の節点は Q より取り出した節点
- 黒色の節点は S の要素

Dijkstra のアルゴリズム

DIJKSTRA(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2:  $S = \emptyset$ 
3:  $Q = G.V$ 
4: while  $Q \neq \emptyset$  :
5:    $u = \text{EXTRACT-MIN}(Q)$ 
6:    $S = S \cup \{u\}$ 
7:   for 各頂点  $v \in G.Adj[u]$  :
8:     RELAX( $u, v, w$ )
```

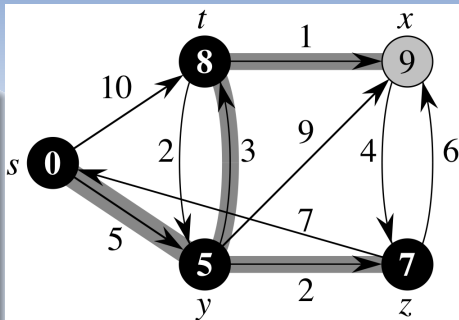


- 灰色の節点は Q より取り出した節点
- 黒色の節点は S の要素

Dijkstra のアルゴリズム

DIJKSTRA(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2:  $S = \emptyset$ 
3:  $Q = G.V$ 
4: while  $Q \neq \emptyset$  :
5:    $u = \text{EXTRACT-MIN}(Q)$ 
6:    $S = S \cup \{u\}$ 
7:   for 各頂点  $v \in G.Adj[u]$  :
8:     RELAX( $u, v, w$ )
```

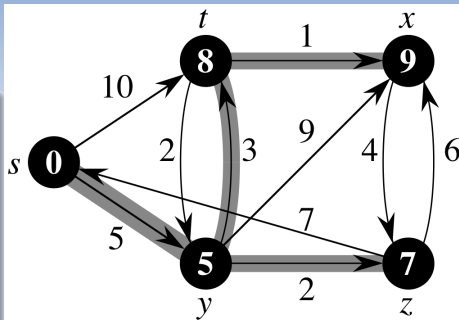


- 灰色の節点は Q より取り出した節点
- 黒色の節点は S の要素

Dijkstra のアルゴリズム

DIJKSTRA(G, w, s)

```
1: INITIALIZE-SINGLE-SOURCE( $G, s$ )
2:  $S = \emptyset$ 
3:  $Q = G.V$ 
4: while  $Q \neq \emptyset$  :
5:    $u = \text{EXTRACT-MIN}(Q)$ 
6:    $S = S \cup \{u\}$ 
7:   for 各頂点  $v \in G.Adj[u]$  :
8:     RELAX( $u, v, w$ )
```



- 灰色の節点は Q より取り出した節点
- 黒色の節点は S の要素

定理 24.6 (Dijkstra のアルゴリズムの正当性)

非負の重み関数 w と始点 s を持つ重み付き有向グラフ $G = (V, E)$ 上の Dijkstra のアルゴリズムの実行は停止し、停止した時点ですべての頂点 $u \in V$ に対して $u.d = \delta(s, u)$ が成立する。

証明

- ループ不変式が成り立つことを示す
「第 4~8 行の **while** 文の各繰返しの開始直前において、各頂点 $v \in S$ に対して $v.d = \delta(s, v)$ が成立する。」
- 頂点 $u \in V$ が集合 S に挿入された時点で、 $u.d = \delta(s, u)$ であることを示す
⇒ 上界性より一度 $u.d = \delta(s, u)$ が成り立つとその後は変化しない。
上界性: 全ての頂点 $v \in V$ に対して $v.d \geq \delta(s, v)$ が常に成立する。(緩和によって経路が更新される場合) $v.d$ が値 $\delta(s, v)$ に達するとその後は変化しない。

証明 (つづき)

- 初期条件: $S = \emptyset$
 - ループ不変式は真
- ループ内条件
 - 頂点 u :
 - $EXTRACT-MIN(Q)$ により取り出された頂点
 - 集合 S に挿入したとき $u.d \neq \delta(s, u)$ である最初の頂点
 - 始点 s , $s.d = \delta(s, s) = 0$ より, u ではない.
 $\Rightarrow u$ を挿入する直前では $S \neq \emptyset$
 - s から u への経路が存在しない場合, $u.d = \delta(s, u) = \infty$ より,
 $u.d \neq \delta(s, u)$ の仮定と矛盾する.
 $\Rightarrow s$ から u への経路が存在する場合を考える.

証明 (つづき)

■ ループ内条件 (続き)

■ s から u への最短路

$$p = s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$$

$$\blacksquare s, x \in S$$

$$\blacksquare y, u \in V - S$$

■ x は y の p 上の先行点

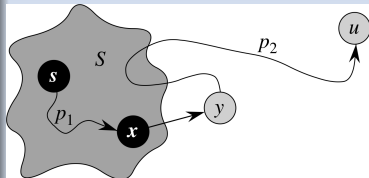
■ u は $u.d \neq \delta(s, u)$ となる初めての節点 なので, $x.d = \delta(s, x)$ が成り立つ

\Rightarrow 収束性より, $y.d = \delta(s, y)$

■ 辺の重みは非負なので,

$$\delta(s, y) \leq \delta(s, u)$$

■ つまり, $y \overset{p_2}{\rightsquigarrow} u$ 上で重みが減ることは
ない



証明 (つづき)

■ ループ内条件 (続き)

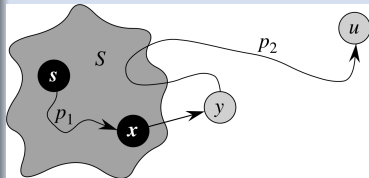
■ 従って,

$$y.d = \delta(s, y) \leq \delta(s, u) \leq u.d$$

(上界性 より)

- u, y は共に $V - S$ の要素 (つまり, Q の中にある) であり, かつ u は *EXTRACT-MIN* で取り出された節点なので, $u.d \leq y.d$ である. 従って, $y.d = \delta(s, y) = \delta(s, u) = u.d$

- つまり, $u.d = \delta(s, u)$ となるが, これは $u.d \neq \delta(s, u)$ であるという仮定と矛盾する.
- 従って, u を S に挿入したときには, $u.d = \delta(s, u)$ である.



証明 (続き)

■ 終了条件

- 終了時点では $Q = \emptyset$ である. $S = V$ なので, すべての頂点 $u \in V$ に対して $u.d = \delta(s, u)$ である.

- 1 頂点 z を始点として、図 24.4 に示す有向グラフ上で Bellman-Ford アルゴリズムを実行せよ。各走査では図と同じ順序で辺を緩和し、各走査後の d と π の値を求めよ。また、辺 (z, x) の重みを 4 に変更し、 s を始点としてこのアルゴリズムを再度実行せよ。
- 2 図 24.5 に示した有向非巡回グラフに対して、頂点 r を始点として用いて *DAG-SHORTEST-PATHS* を実行してみよ。
- 3 図 24.2 に示した有向グラフに対して、頂点 s を始点にした場合と z を始点にした場合のそれぞれについて、Dijkstra のアルゴリズムを実行し、最終的に得られる d と π の値を求めよ。