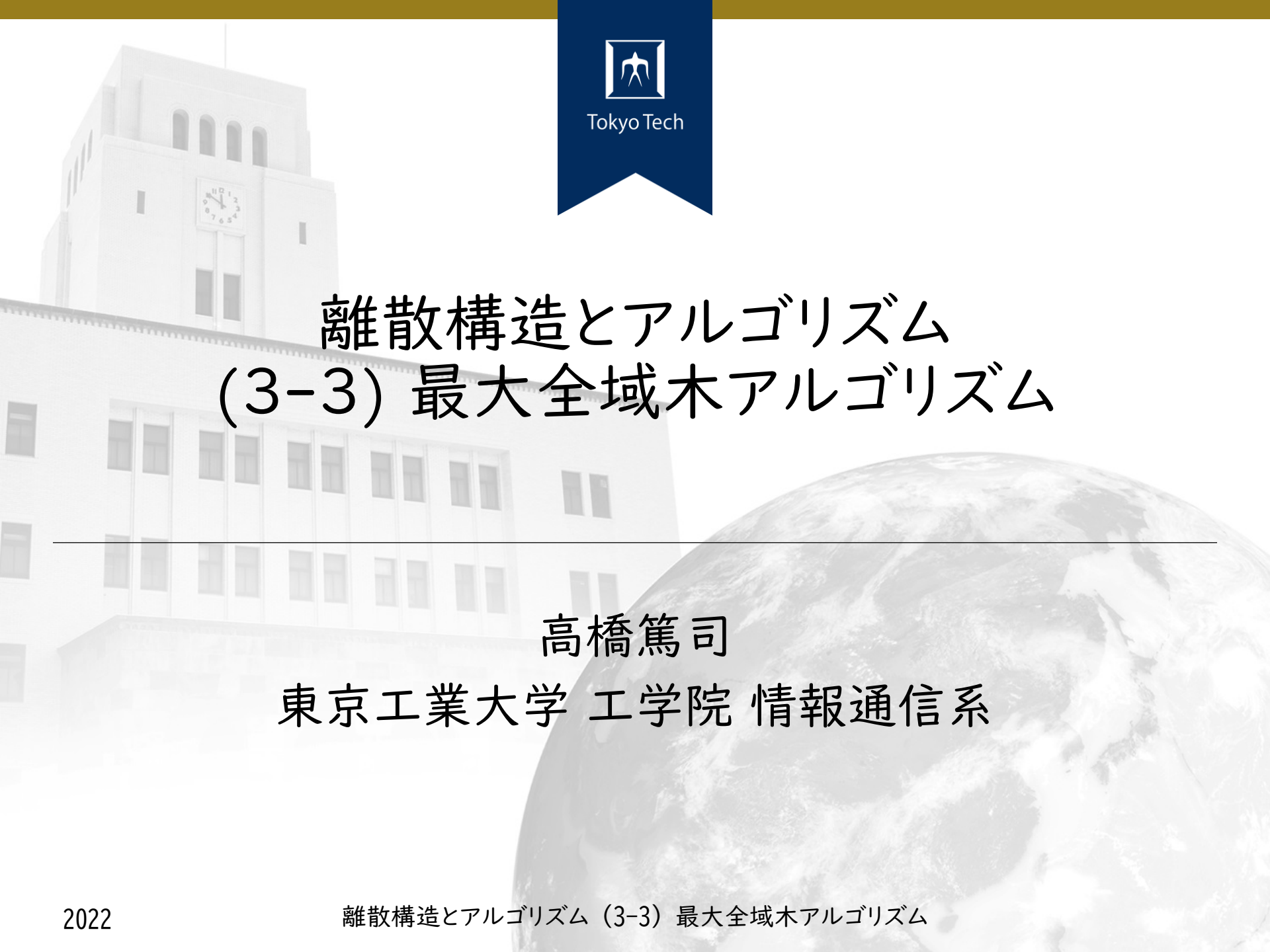


離散構造とアルゴリズム
(3-3) 最大全域木アルゴリズム

高橋篤司
東京工業大学 工学院 情報通信系

3-3（1）最大全域木アルゴリズム

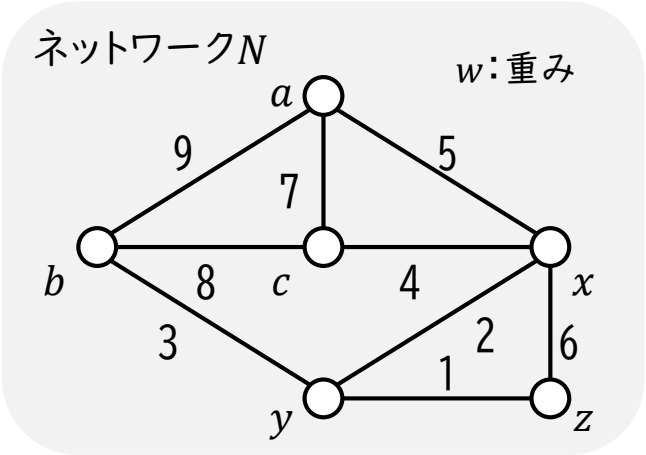
■ 最大全域木問題（maximum-spanning tree problem）

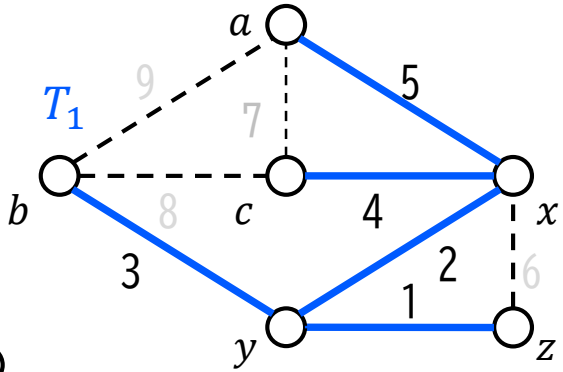
- 入力：連結グラフ$G = (V, E)$，重み関数$w: E \to \mathbb{R}$
- 質問：ネットワーク$N = (G, w)$の最大全域木を一つ示せ

$n = |V|, m = |E|$

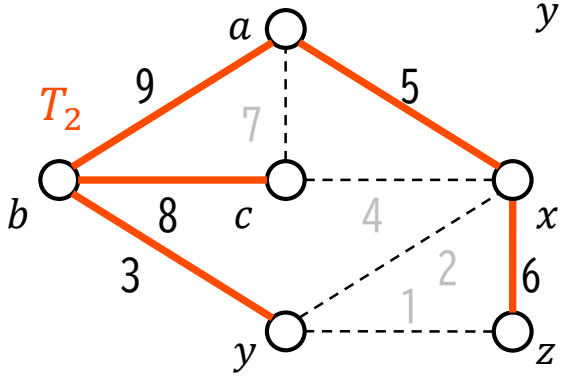
■ 最大全域木

- 全域木Tの中で$\sum_{e \in E(T)} w(e)$が最大





$w(T_1) = 15$



$w(T_2) = 31$

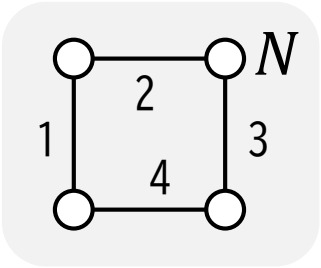
最大全域木アルゴリズム（全列挙）

■ 最大全域木問題（maximum-spanning tree problem）

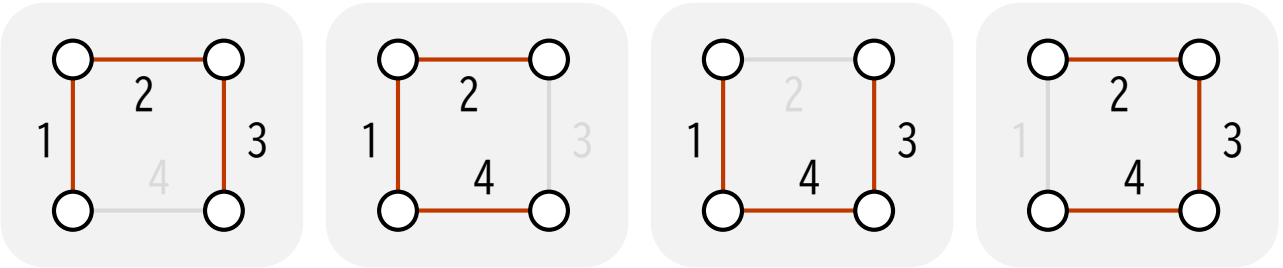
- 入力：連結グラフ$G = (V, E)$，重み関数$w: E \to \mathbb{R}$
- 質問：ネットワーク$N = (G, w)$の最大全域木を一つ示せ

$n = |V|, m = |E|$

■ すべての全域木を列挙したら？



最大全域木



最大全域木アルゴリズム（全列挙）

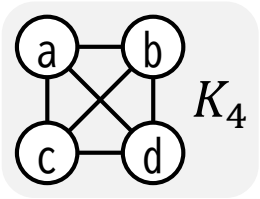
■ 最大全域木問題（maximum-spanning tree problem）

- 入力：連結グラフ$G = (V, E)$，重み関数$w: E \to \mathbb{R}$
- 質問：ネットワーク$N = (G, w)$の最大全域木を一つ示せ

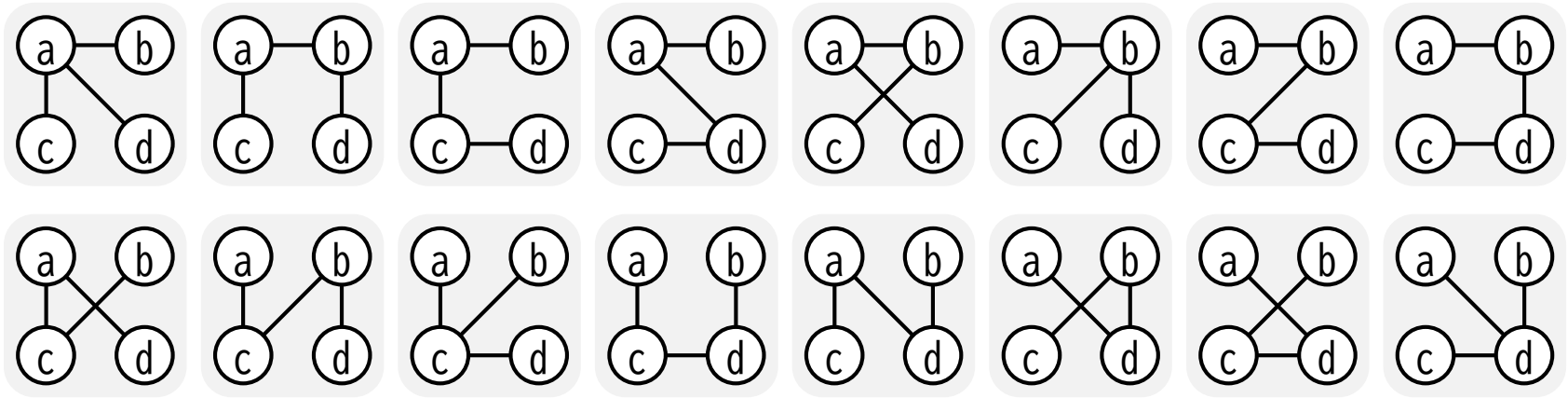
$n = |V|, m = |E|$

■ すべての全域木を列挙したら？

- K_nの異なる全域木の数：n^{n-2}



➢ K_4の全域木



最大全域木アルゴリズム（Kruskal）

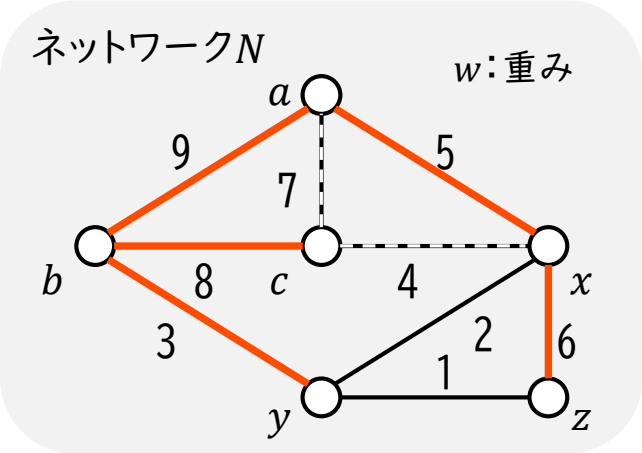
■ 最大全域木問題（maximum-spanning tree problem）

- 入力：連結グラフ$G = (V, E)$，重み関数$w: E \to \mathbb{R}$
- 質問：ネットワーク$N = (G, w)$の最大全域木を一つ示せ

$n = |V|, m = |E|$

■ 重たい辺から順に選んで木を構成

- 重たい辺でも，選ぶと木でなくなるなら...あきらめる



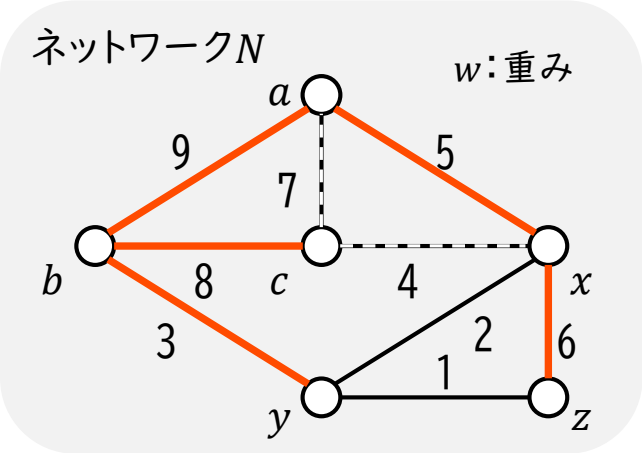
最大全域木アルゴリズム（Kruskal）

■ Algorithm 3.8 (Kruskal)

- 入力：連結グラフ$G=(V,E)$，重み関数 $w: E \to \mathbb{R}$
- 出力：ネットワーク$N=(G,w)$の最大全域木

Step 0: Sort E so that $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_m)$
Step 1: Set $i=1$ and $X=\emptyset$
Step 2: If $G\langle X \cup \{e_i\}\rangle$ contains no cycle, then set $X = X \cup \{e_i\}$
Step 3: If $|X| = n-1$, then output $G\langle X\rangle$, and halt
Step 4: Set $i = i+1$ and return to Step 2

$G\langle X\rangle$ ： X以外の辺を除去したGの部分グラフ



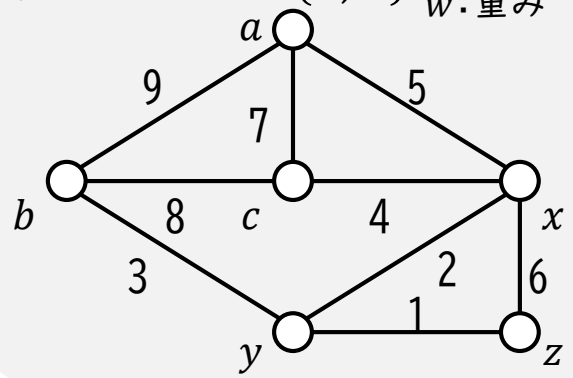
最大全域木アルゴリズム（Kruskal）

■ Kruskalアルゴリズム

- 入力：連結グラフ$G=(V,E)$，重み関数$w: E \to \mathbb{R}$
- 出力：ネットワーク$N=(G,w)$の最大全域木T

$n=|V|, m=|E|$

0. 初期操作
   - 辺を重みの降順に整列（ソーティング）
   - $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_m)$
1. 初期設定
   - $i := 1$，$X := \emptyset$
2. 閉路判定
   - 辺e_iの追加でXに閉路が生じない
     ⇒ $X := X \cup \{e_i\}$
3. 終了判定
   - $|X| = n-1 \implies G\langle X \rangle$出力し終了
4. 更新
   - $i := i+1$
   - Step 2へ戻る



辺の追加？

$e_1=(a,b)$：

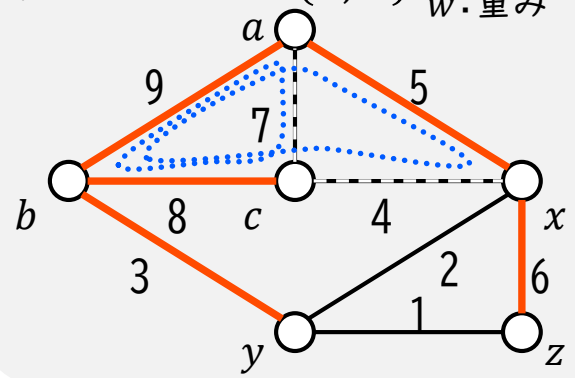
最大全域木アルゴリズム（Kruskal）

■ Kruskalアルゴリズム

- 入力：連結グラフ$G=(V,E)$，重み関数$w: E \to \mathbb{R}$
- 出力：ネットワーク$N=(G,w)$の最大全域木T

$n=|V|, m=|E|$

0. 初期操作
   - 辺を重みの降順に整列（ソーティング）
   - $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_m)$
1. 初期設定
   - $i \coloneqq 1$，$X \coloneqq \emptyset$
2. 閉路判定
   - 辺e_iの追加でXに閉路が生じない
     $\Rightarrow$ $X \coloneqq X \cup \{e_i\}$
3. 終了判定
   - $|X|=n-1 \implies G\langle X\rangle$出力し終了
4. 更新
   - $i \coloneqq i+1$
   - Step 2へ戻る



辺の追加？

$e_1=(a,b)$：OK
$e_2=(b,c)$：OK
$e_3=(a,c)$：NG
$e_4=(x,z)$：OK
$e_5=(a,x)$：OK
$e_6=(c,x)$：NG
$e_7=(b,y)$：OK

最大全域木アルゴリズム（Kruskal）

■ 補題（Lemma 3.1）：Algorithm 3.8(Kruskal)は
最大全域木問題を解く

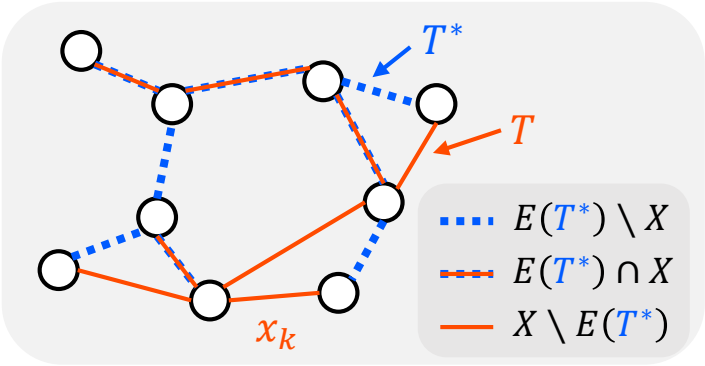
□ 証明

- $X = \{x_1, x_2, \ldots, x_{n-1}\}$
  - $w(x_1) \geq w(x_2) \geq \cdots \geq w(x_{n-1})$

➢ 出力$T(= G\langle X \rangle)$
  - 閉路を含まないGの全域部分グラフ
    ⇒ Gの全域木

✓ TはGの最大全域木であることを示す
  - 「Tは最大全域木ではない」と仮定

➢ 最大全域木T^*
  - Xと共通部分が最大の最大全域木
  - $E(T^*) = \{x_1, x_2, \ldots, x_{k-1}, \ldots\}$
    - x_1からx_{k-1}まで一致 ⇒ kが最大
    - $k < n-1$ （∵仮定）



最大全域木アルゴリズム（Kruskal）

■ 補題（Lemma 3.1）：Algorithm 3.8(Kruskal)は
最大全域木問題を解く

□ 証明（cont.）

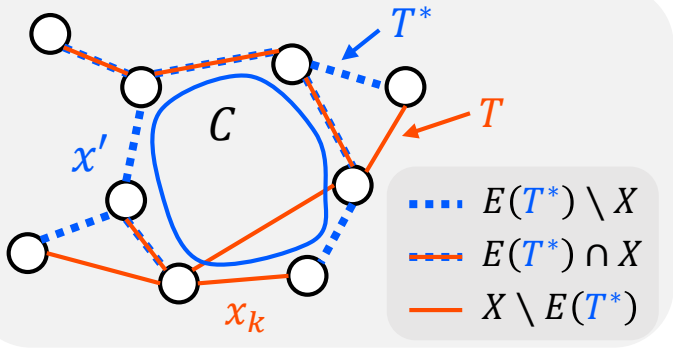
- $X = \{x_1, x_2, \ldots, x_{n-1}\}$

➢ 出力$T(= G\langle X\rangle)$

➢ 最大全域木T^*

- Xと共通部分が最大の最大全域木
- $x_k \notin E(T^*)$
- $T^* \cup \{x_k\}$は一意的な閉路Cを持つ
- $\exists x' \in E(C) \setminus X$

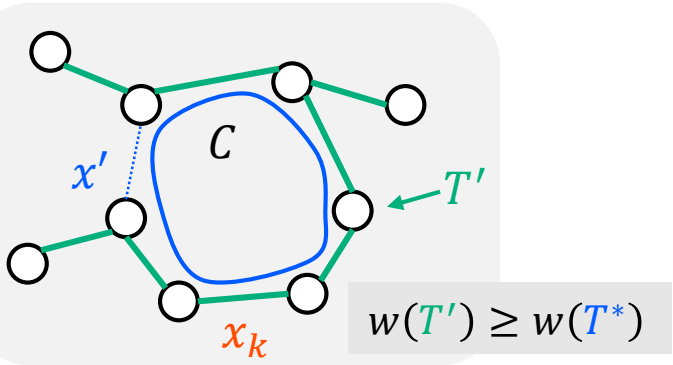
$$X\colon\ w(x_1), w(x_2), \ldots, w(x_{k-1}), w(x_k), \ldots, w(x_{n-1})$$
$$\text{IV}$$
$$E(T^*)\colon w(x_1), w(x_2), \ldots, w(x_{k-1}), w(x'), \ldots$$



➢ $T' = (T^* + \{x_k\}) - \{x'\}$ ： 全域木

- $\{x_1, x_2, \ldots, x_{k-1}, x'\}$ $(\subseteq E(T^*))$に閉路なし
- $w(x') > w(x_k)$ならば ⇒
  アルゴリズムはx'をXに加える ⇒ 矛盾

$\therefore\ w(x_k) \geq w(x')$ ⇒ $w(T') \geq w(T^*)$



最大全域木アルゴリズム（Kruskal）

■ 補題（Lemma 3.1）：Algorithm 3.8(Kruskal)は
最大全域木問題を解く

□ 証明（cont.）

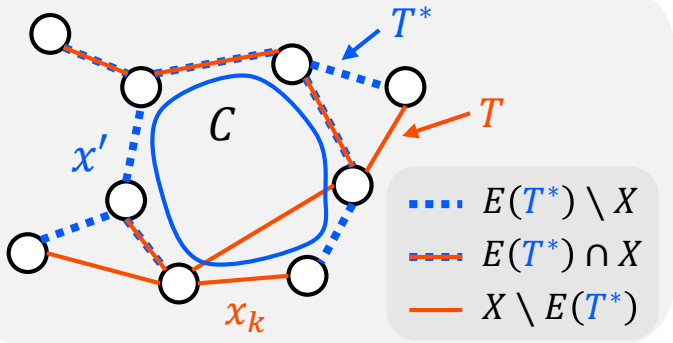
- $X = \{x_1, x_2, \ldots, x_{n-1}\}$

➢ 出力$T(= G\langle X\rangle)$

➢ 最大全域木T^*

- Xと共通部分が最大の最大全域木
- $x_k \notin E(T^*)$
- $T^* \cup \{x_k\}$は一意的な閉路Cを持つ
- $\exists x' \in E(C) \setminus X$

$$X\colon\ w(x_1), w(x_2), \ldots, w(x_{k-1}), w(x_k), \ldots, w(x_{n-1})$$
$$\text{IV}$$
$$E(T^*)\colon w(x_1), w(x_2), \ldots, w(x_{k-1}), w(x'), \ldots$$



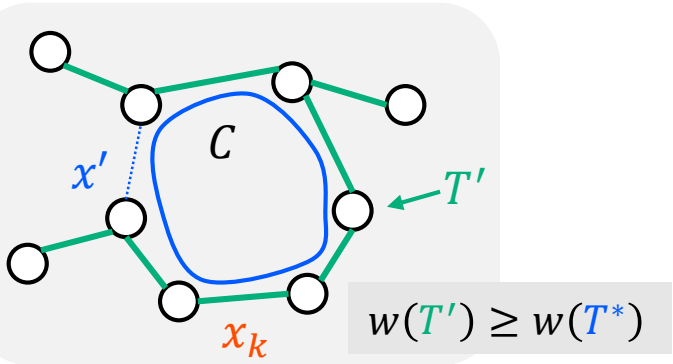
➢ $T' = (T^* + \{x_k\}) - \{x'\}$ ： 全域木

- T^*よりXと共通部分が多い最大全域木

$\Rightarrow T^*$の定義に反する $\Rightarrow k = n$

$\therefore \quad T = T^*$

✓「$T = G\langle X\rangle$は最大全域木」■



最大全域木アルゴリズム（Kruskal）

■ Kruskalアルゴリズム
- 入力：連結グラフ$G=(V,E)$，重み関数$w: E \to \mathbb{R}$
- 出力：ネットワーク$N=(G,w)$の最大全域木T

$n=|V|, m=|E|$

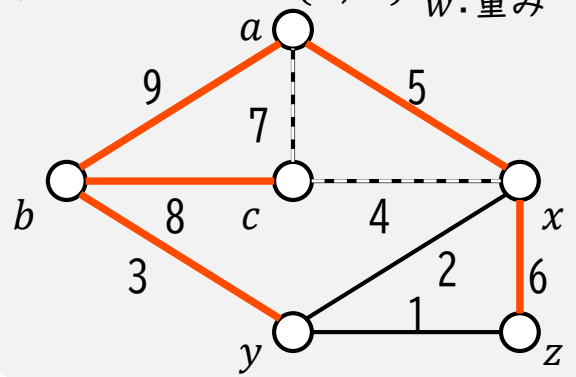
0. 初期操作
- 辺を重みの降順に整列（ソーティング）
- $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_m)$ $O(m \log m)$

1. 初期設定 $O(1)$
- $i := 1$，$X := \emptyset$

2. 閉路判定 閉路判定計算量$\times O(m) = ?$
- 辺e_iの追加でXに閉路が生じない
  $\Rightarrow X := X \cup \{e_i\}$

3. 終了判定 $O(1) \times O(m) + O(n) = O(m) + O(n)$
- $|X| = n-1 \implies G\langle X \rangle$出力し終了

4. 更新 $O(1) \times O(m) = O(m)$
- $i := i+1$
- Step 2へ戻る



辺の追加？
- $e_1=(a,b)$：OK
- $e_2=(b,c)$：OK
- $e_3=(a,c)$：NG
- $e_4=(x,z)$：OK
- $e_5=(a,x)$：OK
- $e_6=(c,x)$：NG
- $e_7=(b,y)$：OK

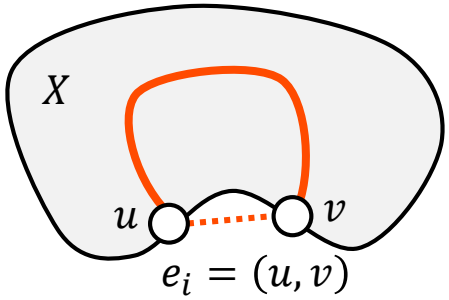
全体の計算量 ： $O(m \log m) + O(?) \times O(m) + O(n)$

Step 2：閉路判定の計算量は？

最大全域木アルゴリズム（Kruskal）

■ 辺を追加したとき閉路が生じるか否かの判定方法は？

- 辺$e_i = (u, v) \notin X$をXに追加
- Xでu, v間に路があると閉路が生じる

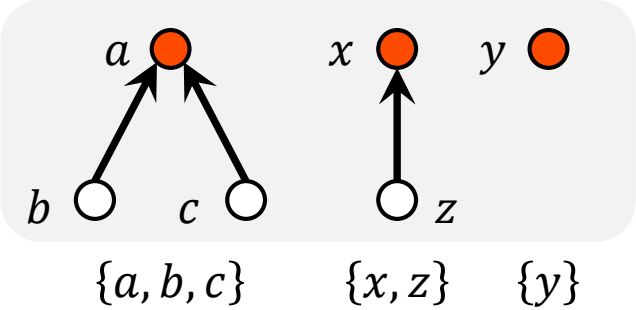


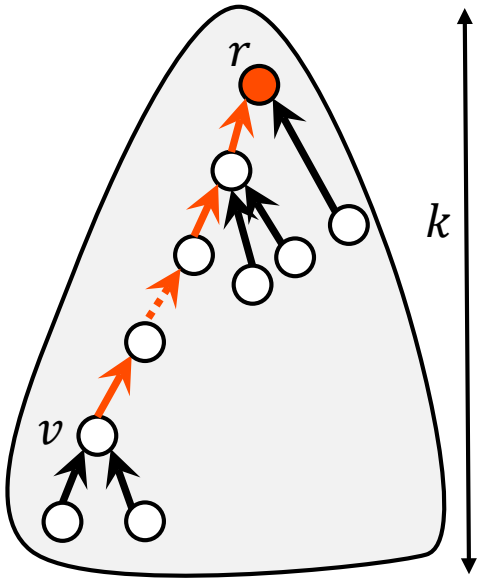
- Dijkstraを利用（$\mathrm{dis}_X(u, v)$ は有限か?）$\Longrightarrow O(n^2)$
  - $O(m \log m) + O(n^2) \times O(m) + O(n) = O(n^2 m)$
- BFS(DFS)を利用（u, v はXで連結か?）$\Longrightarrow O(n + m)$
  - $O(m \log m) + O(n + m) \times O(m) + O(n) = O(m^2)$
- Union-Findを利用 $\Longrightarrow O(\alpha)$
  - $O(m \log m) + O(\alpha) \times O(m) + O(n) = O(m \log m)$
  - Kruskalアルゴリズムの計算量
    - Step 0（整列）が支配的

α ： アッカーマン関数$A(n, n)$の逆関数（ほぼ定数）

3-3（2）合併発見手法（union-find）

■ 互いに素な集合の族(集合の分割)を根付き木で表現





根付き木の高さ：k

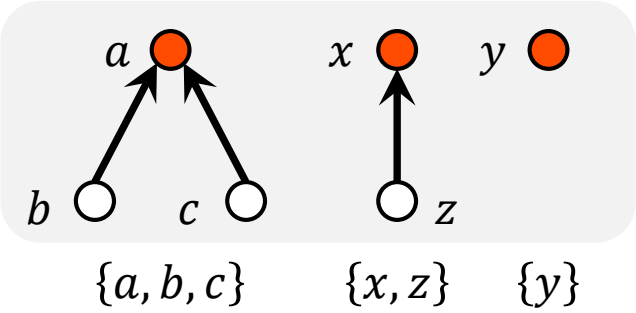
■ 発見操作 find(v) ： O(k)
- 集合の代表点（根）を見つける

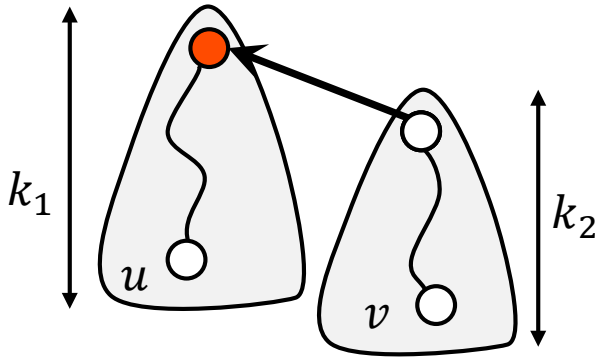
➢ 2要素が同じ集合に属すか？
- Yes ⇒ 代表点が同じ
- No ⇒ 代表点が異なる

✓ O(k)で確認可能（発見操作を2回実行）

合併発見手法（union-find）

■ 互いに素な集合の族(集合の分割)を根付き木で表現





■ 合併操作 union(u, v) ： O(k)

- 集合の合併（点uと点vの属す集合）

1. find(u)とfind(v)でそれぞれ代表点（根）を見つける
2. 異なれば一方を他方の子に
   - 合併後の木の高さk ： $\max(k_1, k_2 + 1)$ or $\max(k_1 + 1, k_2)$

要素数：n

■ 不適切に合併 ⇒ 木の高さが増大 ⇒ 最悪で木の高さO(n)

- Kruskal の計算量はO(nm)になり、効果があまりない

合併発見手法（union-find）

■ 高速化のための工夫

➢ 木の高さをなるべく

- Union
  - ✓ 要素数が少ない集合を子に
    ⇒ 木の高さkは高々$O(\log n)$
  - ➢ Kruskal での閉路判定 $O(m \log n)$
    - 辺の整列（ソーティング）$O(m \log m)$より計算量小
- Find
  - ✓ 根までの路の上の点⇒根の子に
  - ✓ 木の高さは$O(\alpha)$

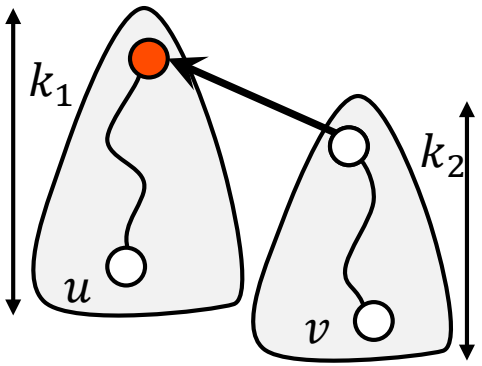
$n = |V|, m = |E|$

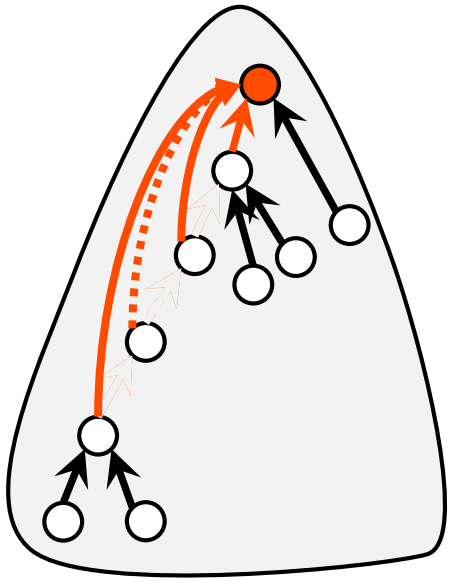
α ： アッカーマン関数$A(n,n)$の逆関数（ほぼ定数）

■ Union-Find一回あたりの計算量

⇒ $O(\alpha)$

- Kruskal での閉路判定
  - ■ 全体でほぼ$O(m)$
    - 整列（ソーティング）に比べ十分小さい





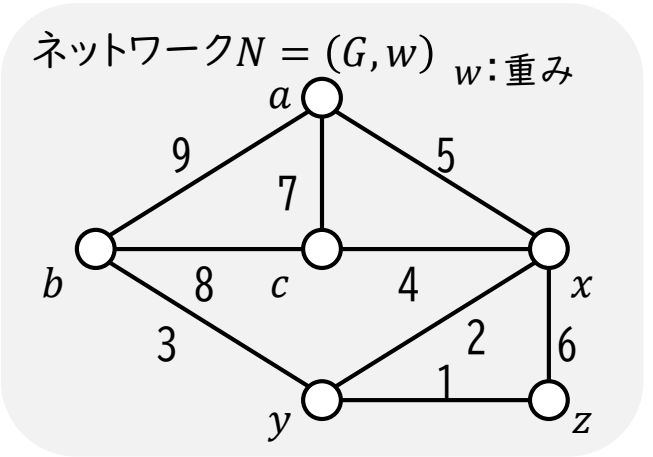
最大全域木アルゴリズム（Kruskal+UnionFind）

■ Kruskalアルゴリズム　$O(m \log m)$

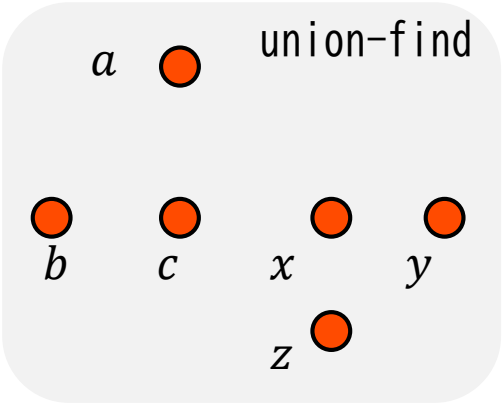
- 入力：連結グラフ$G = (V, E)$，重み関数$w: E \to \mathbb{R}$
- 出力：ネットワーク$N = (G, w)$の最大全域木T

$n = |V|, m = |E|$

0. 初期操作
   - 辺を重みの降順に整列（ソーティング）
   - $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_m)$
1. 初期設定
   - $i := 1$，$X := \emptyset$
2. 閉路判定
   - 辺e_iの追加でXに閉路が生じない
     ⇒ $X := X \cup \{e_i\}$
3. 終了判定
   - $|X| = n - 1 \implies G\langle X \rangle$出力し終了
4. 更新
   - $i := i + 1$
   - Step 2へ戻る



辺の追加？
$e_1 = (a, b)$ ：



連結成分の代表点（根付き木の根）

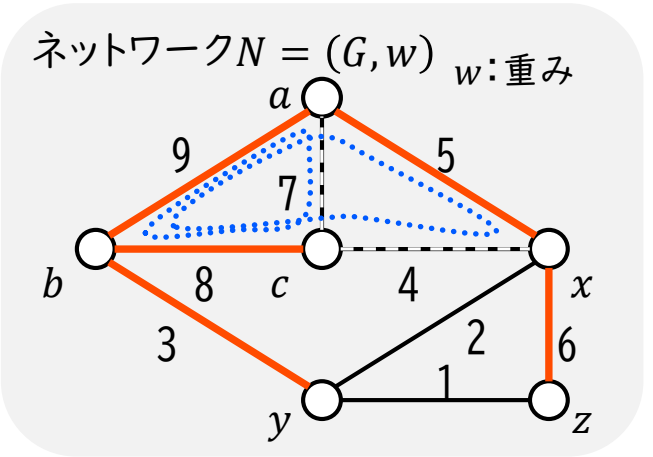
最大全域木アルゴリズム（Kruskal+UnionFind）

■ Kruskalアルゴリズム　$O(m \log m)$

- 入力：連結グラフ$G = (V, E)$，重み関数$w: E \to \mathbb{R}$
- 出力：ネットワーク$N = (G, w)$の最大全域木T

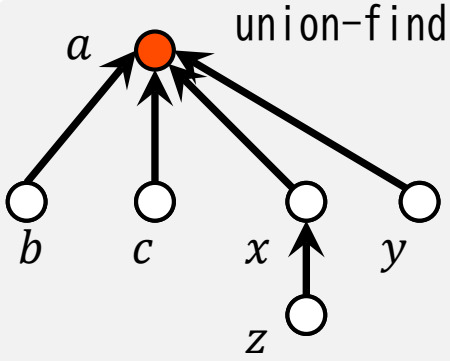
$n = |V|, m = |E|$

0. 初期操作
   - 辺を重みの降順に整列（ソーティング）
   - $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_m)$
1. 初期設定
   - $i := 1$，$X := \emptyset$
2. 閉路判定
   - 辺e_iの追加でXに閉路が生じない
     $\Rightarrow$ $X := X \cup \{e_i\}$
3. 終了判定
   - $|X| = n - 1 \implies G\langle X \rangle$出力し終了
4. 更新
   - $i := i + 1$
   - Step 2へ戻る



辺の追加？

$e_1 = (a, b)$ ：OK
$e_2 = (b, c)$ ：OK
$e_3 = (a, c)$ ：NG
$e_4 = (x, z)$ ：OK
$e_5 = (a, x)$ ：OK
$e_6 = (c, x)$ ：NG
$e_7 = (b, y)$ ：OK



連結成分の代表点（根付き木の根）

最大全域木アルゴリズム (Kruskal)

■ 定理 (Theorem 3.9): Algorithm 3.8(Kruskal)は
最大全域木問題を$O(m \log m)$で解く

■ 重たい辺から順に選んで木を構成
- 重たい辺でも,選ぶと木でなくなるなら...あきらめる

✓ 辺の整列が計算量で支配的

■ Algorithm 3.8 (Kruskal)
- 入力: 連結グラフ$G = (V, E)$, 重み関数$w: E \to \mathbb{R}$
- 出力: ネットワーク$N = (G, w)$の最大全域木

Step 0: Sort E so that $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_m)$
Step 1: Set $i = 1$ and $X = \emptyset$
Step 2: If $G\langle X \cup \{e_i\}\rangle$ contains no cycle, then set $X = X \cup \{e_i\}$
Step 3: If $|X| = n - 1$, then output $G\langle X\rangle$, and halt
Step 4: Set $i = i + 1$ and return to Step 2

3-3（3）最小全域木アルゴリズム

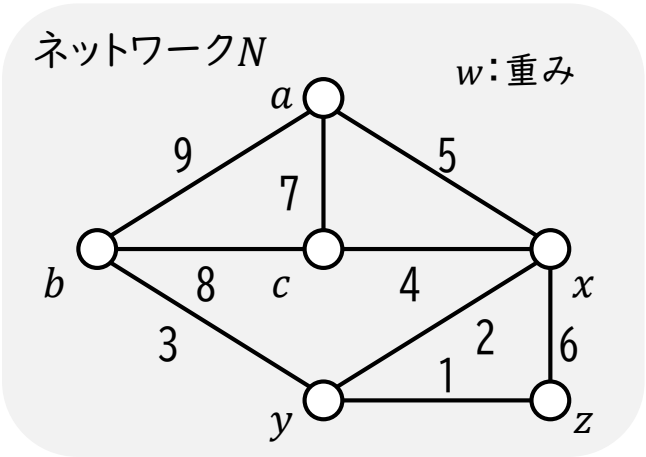
■ 最小全域木問題（minimum-spanning tree problem）

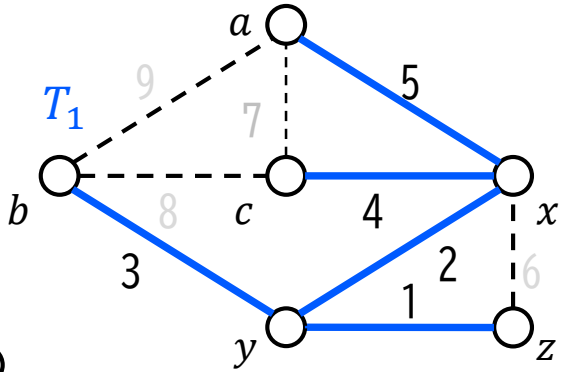
- 入力：連結グラフ$G = (V, E)$，重み関数$w: E \to \mathbb{R}$
- 質問：ネットワーク$N = (G, w)$の最小全域木を一つ示せ

$n = |V|, m = |E|$

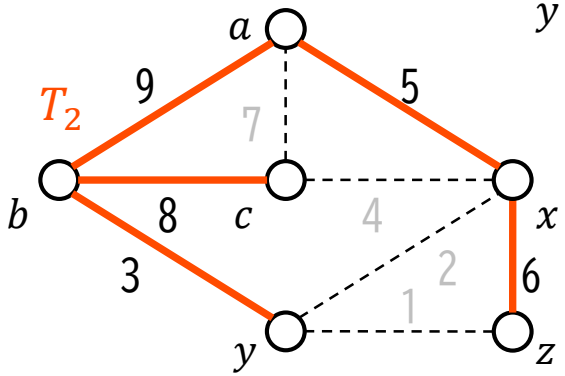
■ 最小全域木

- 全域木Tの中で$\sum_{e \in E(T)} w(e)$が最小





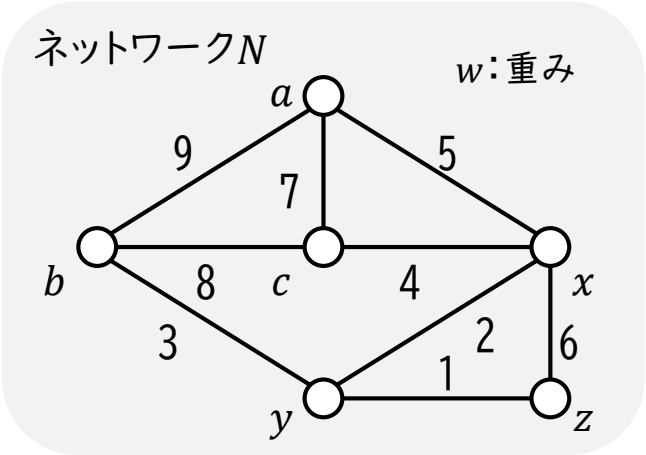
$w(T_1) = 15$

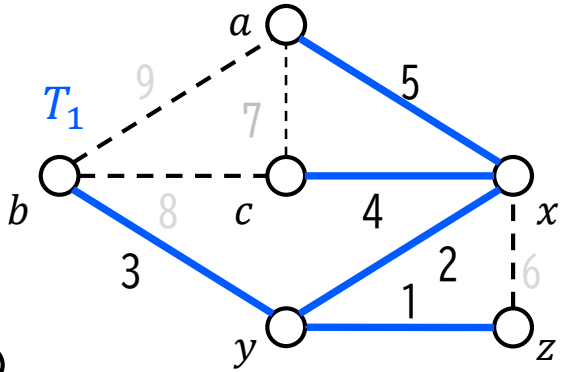


$w(T_2) = 31$

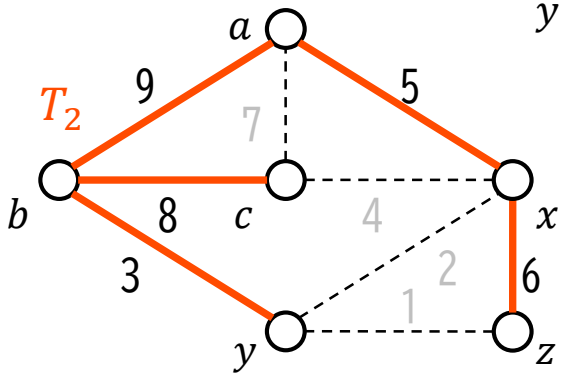
最小全域木問題の難しさ

- ✓ Kruskalアルゴリズムで$O(m \log m)$で解ける
  - 初期操作で辺を重みの昇順に整列(ソーティング)させる
- ■ 最大(小)全域木問題
  - 最適性の原理が成立
  - 辺重みの正負に無関係
    - 辺重みの大小関係で解が定まる





$w(T_1) = 15$



$w(T_2) = 31$

最小全域木アルゴリズム (Kruskal)

■ 定理: Algorithm 3.9(Kruskal)は
最小全域木問題を$O(m \log m)$で解く

■ 軽い辺から順に選んで木を構成
- 軽い辺でも,選ぶと木でなくなるなら...あきらめる

✓ 辺の整列が計算量で支配的

■ Algorithm 3.9 (Kruskal)
- 入力: 連結グラフ$G = (V, E)$, 重み関数$w: E \to \mathbb{R}$
- 出力: ネットワーク$N = (G, w)$の最小全域木

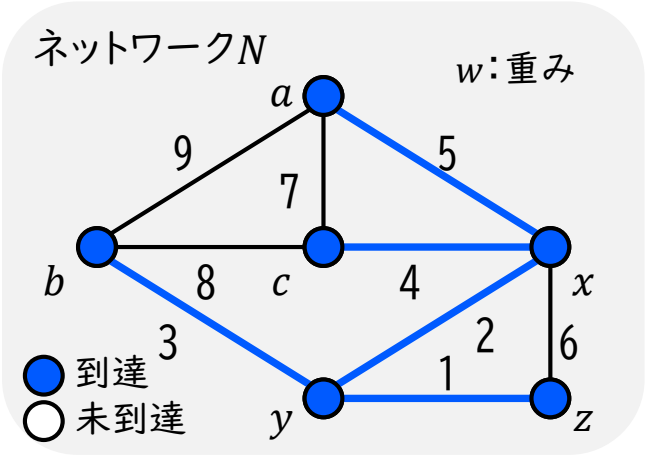
Step 0: Sort E so that $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$
Step 1: Set $i = 1$ and $X = \emptyset$
Step 2: If $G\langle X \cup \{e_i\}\rangle$ contains no cycle, then set $X = X \cup \{e_i\}$
Step 3: If $|X| = n - 1$, then output $G\langle X\rangle$, and halt
Step 4: Set $i = i + 1$ and return to Step 2

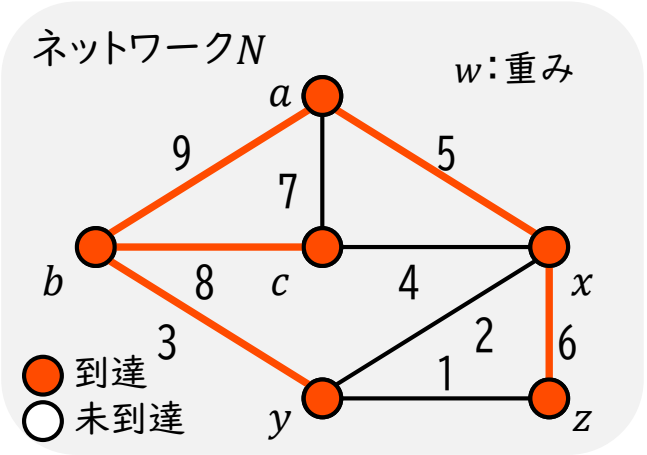
最小（大）全域木アルゴリズム (Prim)

- 定理：Algorithm 3.9p(Prim)は
  最小（大）全域木問題を$O(n^2)$で解く

- ある点から出発し，辺を選んで木を成長させる
  - 到達点と未到達点を接続する
    - 最も軽い辺を選ぶ（最小全域木）
    - 最も重い辺を選ぶ（最大全域木）

✓ Dijkstraとアルゴリズムの基本構成は同じ（評価関数が異なる）

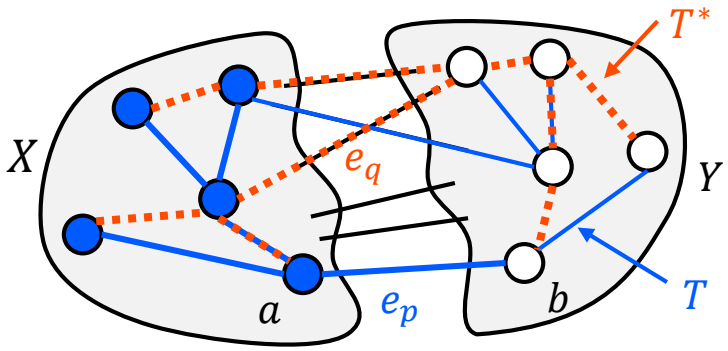




最小全域木アルゴリズム (Prim)

- Algorithm 3.9p(Prim)は最小全域木問題を解く
- 正当性
  - アルゴリズムの出力T
  - 最小全域木T^*
    - Tと共通部分が最大
  - アルゴリズムが$e_p = (a, b)$を選択
    - その時点の到達点集合X, 未到達点集合Y
    - $e_p = (a, b) \notin E(T^*)$と仮定
    - Xの点とYの点を接続するT^*の(a, b)−路上の辺e_q
      - $w(e_p) \leq w(e_q)$
    - $T' = (T^* + \{e_p\}) - \{e_q\}$
      - $w(T') \leq w(T^*)$
        - Tとの共通部分がT^*より多い最小全域木

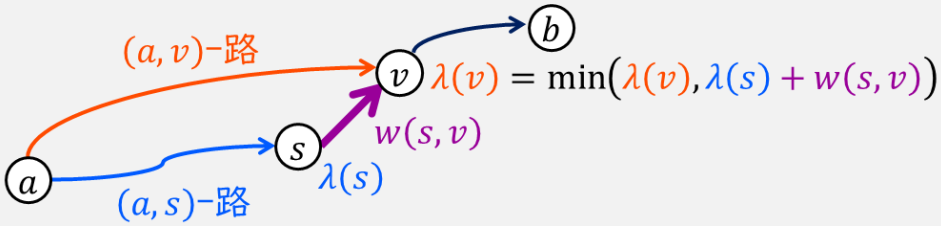
$\therefore e_p \in E(T^*)$ ■



最小（大）全域木アルゴリズム (Prim)

■ Algorithm 3.9p（Prim：最小全域木）
- 入力：連結グラフ$G=(V,E)$，重み関数$w: E \to \mathbb{R}$
- 出力：ネットワーク$N=(G,w)$の最小全域木

Step 0: Set $\lambda(v)=\infty$, $p(v)=v$ $(\forall v \in V(G))$, $X=\emptyset$, $Y=V(G)$
Step 1: Set $s=a$
Step 2: Set $Y=Y\setminus\{s\}$
Step 3: If $|X|=n-1$, then output $G\langle X\rangle$, and halt
Step 4: Set $\lambda(v)=\min\{\lambda(v), w(s,v)\}$ for every $v \in Y$.
Set $p(v)=s$ if the second term is larger
Step 5: Select s such that $\lambda(s)=\min\{\lambda(v)|v \in Y\}$,
and set $X=X\cup\{(p(s),s)\}$, and return to Step 2



■ ラベル更新(Step 4)
- Dijkstra（最短路）：$\lambda(v)=\min\{\lambda(v),\lambda(s)+w(s,v)\}$
- Prim（最小全域木）：$\lambda(v)=\min\{\lambda(v),w(s,v)\}$
- Prim（最大全域木）：$\lambda(v)=\max\{\lambda(v),w(s,v)\}$
  - 初期化：$\lambda(v)=-\infty, \forall v \in V(G)$　選択：ラベル値最大

最短路と最小全域木

■ 最短路問題 (shortest-path problem)
- 入力
  - 連結グラフ$G = (V, E)$, 重み関数$w: E \to \mathbb{R}^+$ ← 重みは非負
  - 始点a $(\in V)$, 終点b $(\in V)$
- 質問:ネットワーク$N = (G, w)$の最短(a, b)-路を一つ示せ

■ 最小全域木問題 (minimum-spanning tree problem)
- 入力
  - 連結グラフ$G = (V, E)$, 重み関数$w: E \to \mathbb{R}$ ← 重みは任意
- 質問:ネットワーク$N = (G, w)$の最小全域木を一つ示せ

■ 問題により難しさの性質は異なる
- 部分問題は,容易な場合がある
- 部分問題が難しければ,その部分問題を含む問題は難しい
- 質問に条件追加の場合は,容易になる場合も難しくなる場合もある