# 論文紹介

# Listing all the minimum spanning trees in an undirected graph

北大　情報知識ネットワーク研究室
B4 大泉翼

# 論文概要

- 最小全域木を求める効率的なアルゴリズムは多数存在
  - プリム法[1]: $O(m + n\log n)$
  - クラスカル法[2]: $O(m\log n)$

- 最小全域木を全て列挙するアルゴリズムを紹介
  - $ALL\_MST : O(N(mn + n^2\log n))$
  - $ALL\_MST_1 : O(Nmn)$
  - $ALL\_MST_2 : O(Nm\log n)$

[1] Dijkstra, E.W. (1959). A note on two problems in connection with graphs

[2] Joseph. B. Kruskal (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem

# 目次

- 準備・問題概要

- $ALL\_MST : O(N(mn + n^2 \log n))$
  - 愚直な分割法による列挙

- $ALL\_MST_1 : O(Nmn)$
  - cut-set を用いた効率の良い列挙方法

- $ALL\_MST_2 : O(Nm \log n)$
  - 上記アルゴリズムの改良

# 目次

- 準備・問題概要

- $ALL\_MST : O(N(mn + n^2 \log n))$
  - 愚直な分割法による列挙

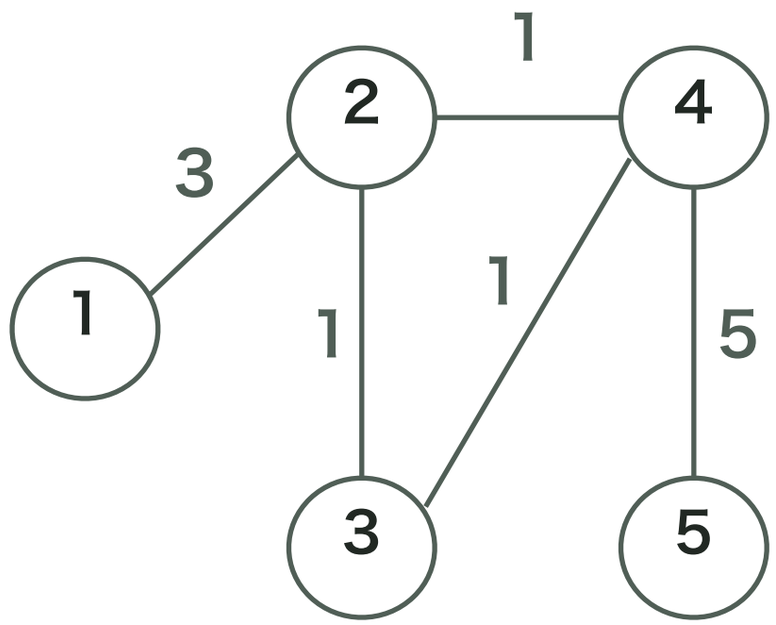
- $ALL\_MST_1 : O(Nmn)$
  - cut-set を用いた効率の良い列挙方法

- $ALL\_MST_2 : O(Nm \log n)$
  - 上記アルゴリズムの改良

# 準備（用語の定義）(1/2)

- 重み付き無向グラフとは $G = (V, E)$ と $w$ との組である
  - $V = (v_1, \ldots, v_n)$ を頂点集合, $E = (e_1, \ldots, e_m) \subseteq V \times V$ を辺集合とする.
  - $w : E \to \mathbb{N}$ を重み関数とし, $w(e)$ を $e$ の重みとする.

重み付き無向グラフの例



# 準備（用語の定義）(2/2)

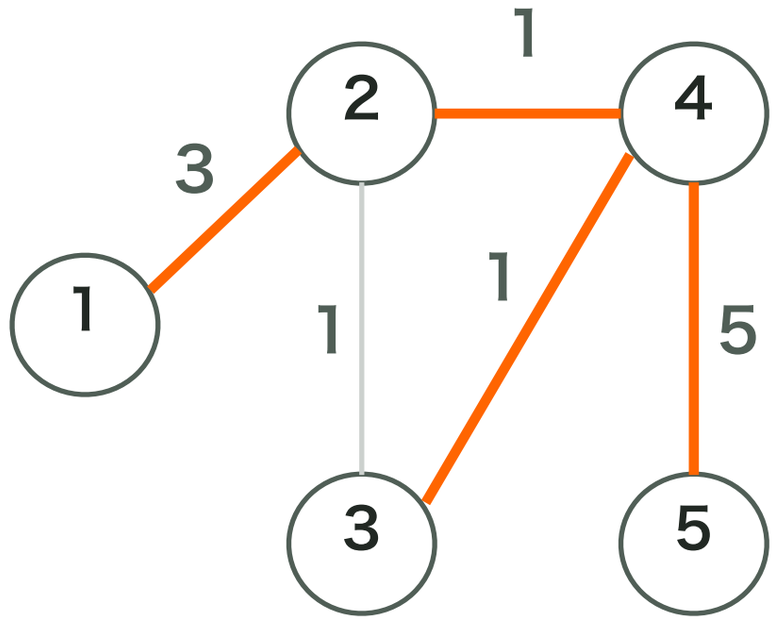
- グラフ $G$ の全域木とは，$G$ の頂点を全て含む部分グラフであり，閉路がなく連結なものである．

- 全域木 $T$ の重み $w(T)$ とは，$T$ の辺の重みの総和である．

- 最小全域木とは，重みが最小の全域木のことを指し，最小全域木の重みを $z^\star$ と書く．

**最小全域木の例**

$z^\star = 10$



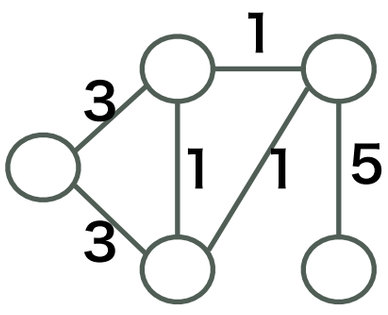
# 対象とする問題

問題 $P$: グラフ $G$ の最小全域木の列挙

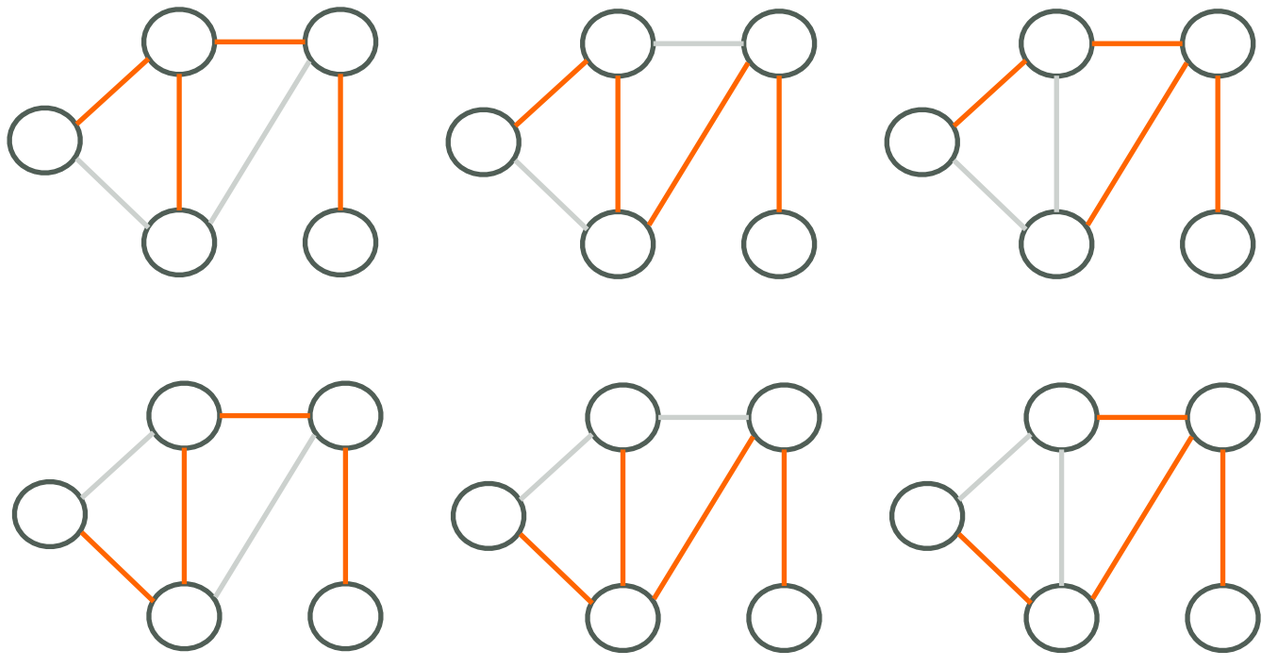
入力: 重み付き無向グラフ $G = (V, E), w$

出力: $G$ に含まれる全ての最小全域木

入力例




出力例



# 目次

- 準備・問題概要

- $ALL\_MST : O(N(mn + n^2 \log n))$
  - 愚直な分割法による列挙

- $ALL\_MST_1 : O(Nmn)$
  - cut-set を用いた効率の良い列挙方法

- $ALL\_MST_2 : O(Nm \log n)$
  - 上記アルゴリズムの改良

# 部分問題への分割

- $G$ の一つの最小全域木を $T = \{e^1, \ldots, e^{n-1}\}$ とする.

- 以下の部分問題を，全ての $i \in [1, n-1]$ に対して解くことで，最小全域木を全て列挙できる.

部分問題 $P(\{e^1, \ldots, e^{i-1}\}, \{e^i\})$:

$e^1, \ldots, e^{i-1}$ を含み，$e^i$ を含まない最小全域木の列挙

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $e^1$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $e^2$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $e^3$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

0: 含む

1: 含まない

# 部分問題への分割

- $G$ の一つの最小全域木を $T = \{e^1, \ldots, e^{n-1}\}$ とする.

- 以下の部分問題を，全ての $i \in [1, n-1]$に対して解くことで，最小全域木を全て列挙できる.

部分問題 $P(\{e^1, \ldots, e^{i-1}\}, \{e^i\})$:

$e^1, \ldots, e^{i-1}$ を含み，$e^i$ を含まない最小全域木の列挙

**$P(\{\}, \{e^1\})$ で列挙**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $e^1$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $e^2$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $e^3$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

**0: 含む**

**1: 含まない**

# 部分問題への分割

- $G$ の一つの最小全域木を $T = \{e^1, \ldots, e^{n-1}\}$ とする.

- 以下の部分問題を，全ての $i \in [1, n-1]$ に対して解くことで，最小全域木を全て列挙できる.

部分問題 $P(\{e^1, \ldots, e^{i-1}\}, \{e^i\})$:

$e^1, \ldots, e^{i-1}$ を含み，$e^i$ を含まない最小全域木の列挙

**$P(\{e^1\}, \{e^2\})$ で列挙**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $e^1$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $e^2$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $e^3$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

**0: 含む**

**1: 含まない**

# 部分問題への分割

- $G$ の一つの最小全域木を $T = \{e^1, \ldots, e^{n-1}\}$ とする.

- 以下の部分問題を，全ての $i \in [1, n-1]$ に対して解くことで，最小全域木を全て列挙できる.

部分問題 $P(\{e^1, \ldots, e^{i-1}\}, \{e^i\})$:

$e^1, \ldots, e^{i-1}$ を含み，$e^i$ を含まない最小全域木の列挙

**$P(\{e^1, e^2\}, \{e^3\})$ で列挙**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $e^1$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $e^2$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $e^3$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

**0: 含む**

**1: 含まない**

# 部分問題への分割

- $G$ の一つの最小全域木を $T = \{e^1, \ldots, e^{n-1}\}$ とする.

- 以下の部分問題を，全ての $i \in [1, n-1]$ に対して解くことで，最小全域木を全て列挙できる.

部分問題 $P(\{e^1, \ldots, e^{i-1}\}, \{e^i\})$:

$e^1, \ldots, e^{i-1}$ を含み，$e^i$ を含まない最小全域木の列挙

$T$ そのもの

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $e^1$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $e^2$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $e^3$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

0: 含む

1: 含まない

# 部分問題の一般化

- 全域木 $T$ が $(F, R)$-許容であるとは,

  $F \subseteq T$ かつ $R \cap T = \emptyset$ を満たすことである.

  - ただし，$F, R \subseteq E, F \cap R = \emptyset$.

- より一般的に, 部分問題を以下のように定義する.

部分問題 $P(F, R)$: グラフ $G$ の $(F, R)$-許容な最小全域木の列挙

**直観的な意味: $F$ に含まれる辺は必ず使い**

**$R$ に含まれる辺は絶対使わないような全域木の列挙**

# 最小全域木を一つ見つける

- $An\_MST(F, R)$: $(F, R)$-許容な最小全域木を

  1 つ見つけるアルゴリズム

  - 見つかった最小全域木を $T^\star(F, R)$，重さを $z^\star(F, R)$ とする.
  - $(F, R)$-許容な全域木がない場合，$z^\star(F, R) = \infty$ とする.

- *Prim* 法を応用することで，時間計算量 $O(m + n \log n)$ で達成可能

# アルゴリズムの概要

- 以下のアルゴリズムに対して $All\_MST(\emptyset, \emptyset)$

  を実行すると, 最小全域木を全列挙できる.

  - $F = \{e^1, \ldots, e^k\}, T^\star(F, R) = F \cup \{e^{k+1}, \ldots, e^{n-1}\}$ とする

ALGORITHM $All\_MST(F, R)$

// $(F, R)$-許容な最小全域木を全て列挙する

Step 1: $An\_MST(F, R)$ を実行し, $T^\star(F, R)$, $z^\star(F, R)$ を定める

Step 2: if $z^\star(F, R) > z^\star$ return

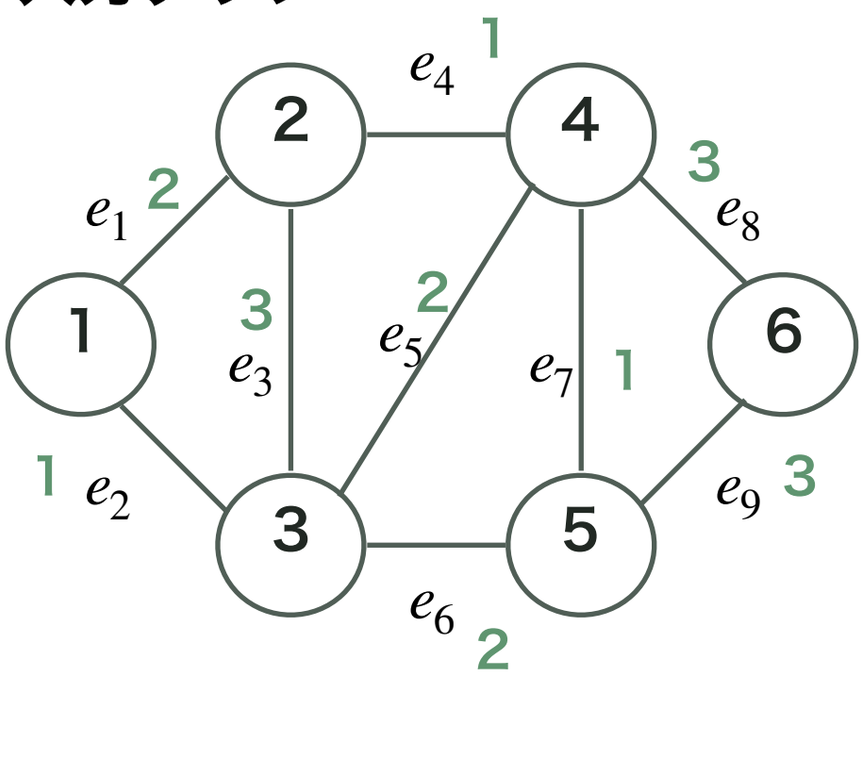
Step 3: $T^\star(F, R)$ を出力

Step 4: for $i \in [k+1, n-1]$ do

- $F_i \leftarrow F \cup \{e^{k+1}, \ldots, e^{i-1}\}$, $R_i \leftarrow R \cup \{e^i\}$
- $All\_MST(F_i, R_i)$ を再帰呼び出し

# アルゴリズムの実行例

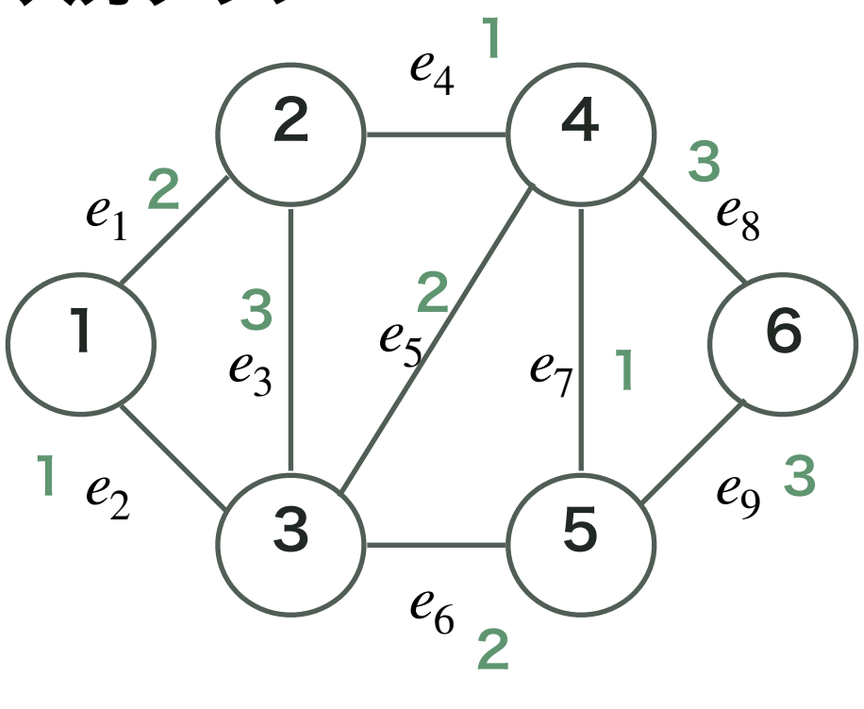
入力グラフ



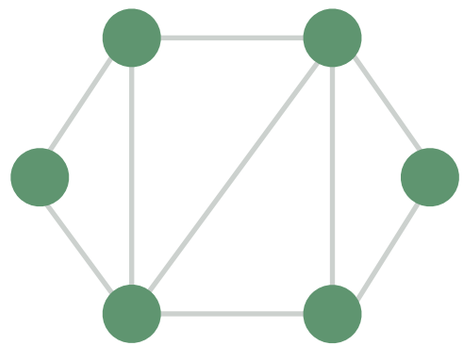

$P(\emptyset, \emptyset)$

# アルゴリズムの実行例

入力グラフ

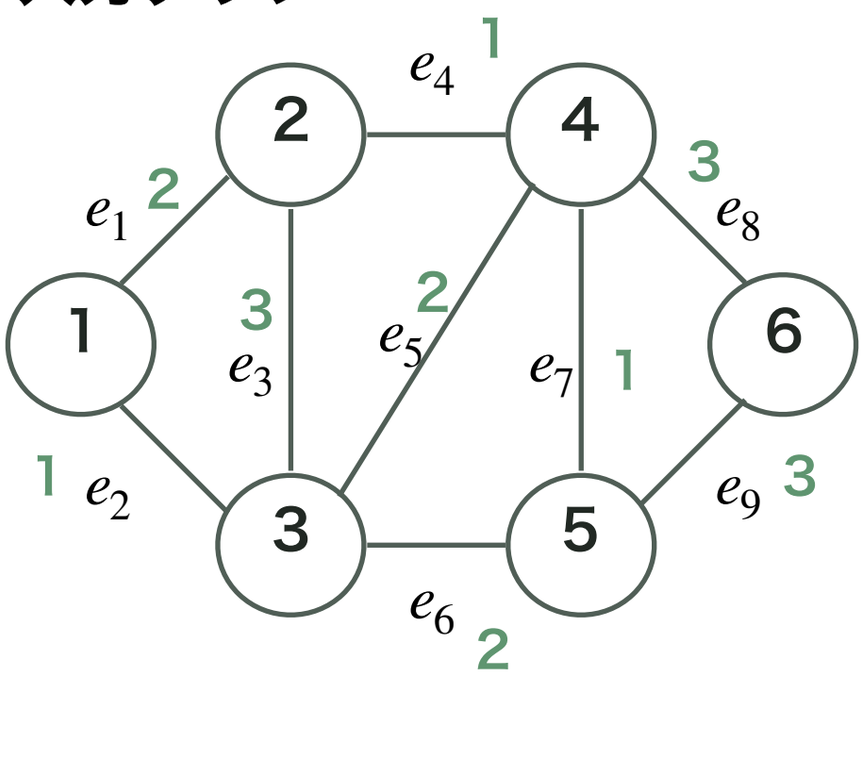



$P(\emptyset, \emptyset)$



# アルゴリズムの実行例

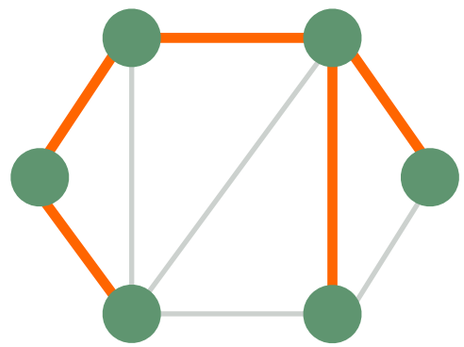
入力グラフ




最小全域木を一つ見つける

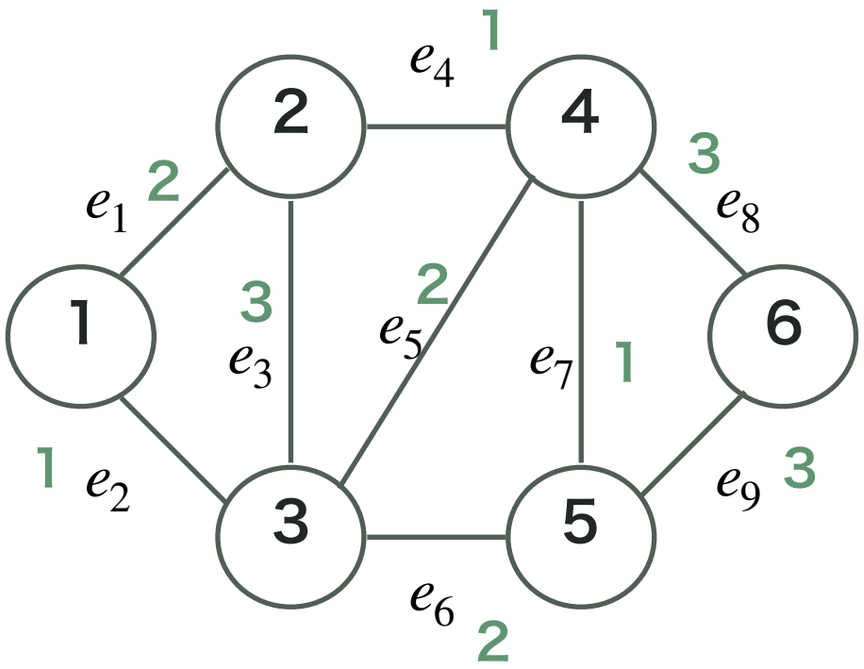
$P(\varnothing, \varnothing)$

$z^\star = 8$



# アルゴリズムの実行例

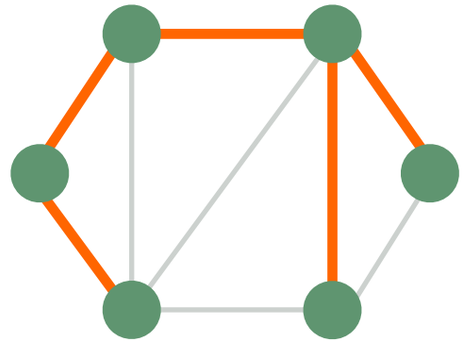
入力グラフ



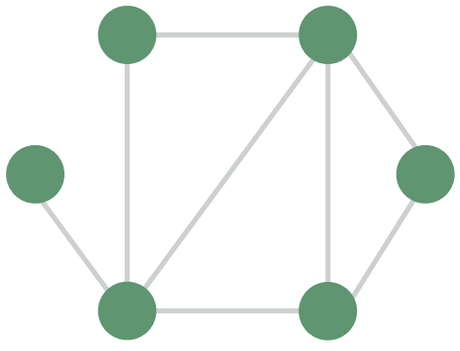

部分問題に分割

$P(\varnothing, \varnothing)$

$z^\star = 8$

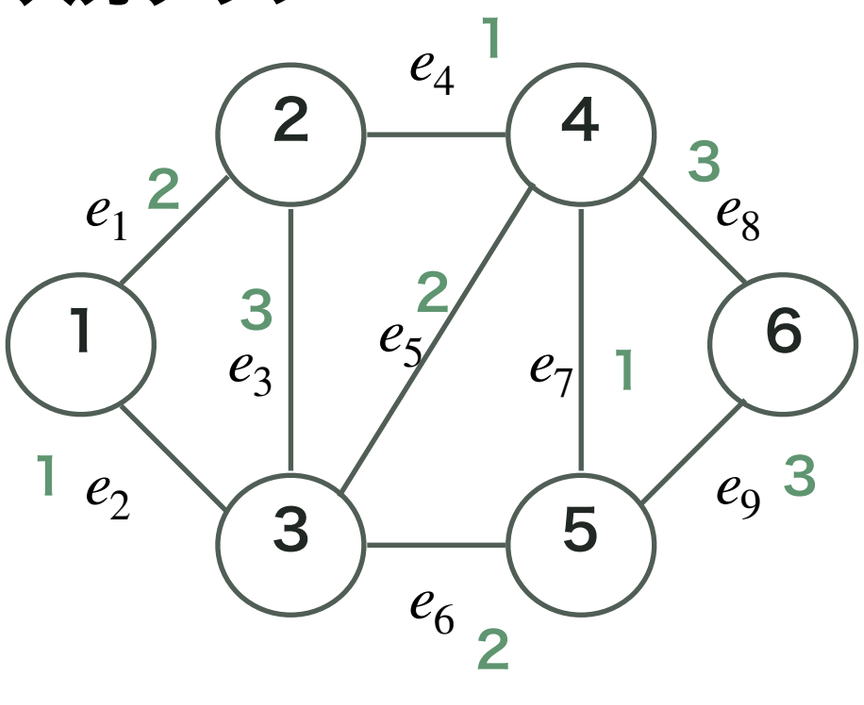


$P(\varnothing, \{e_1\})$



# アルゴリズムの実行例

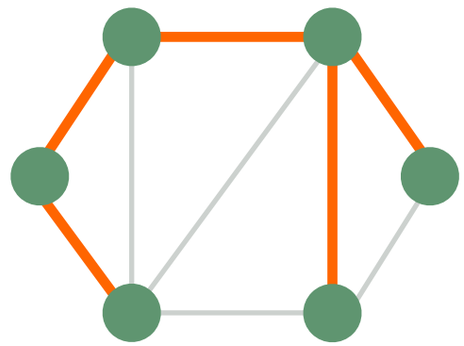
入力グラフ



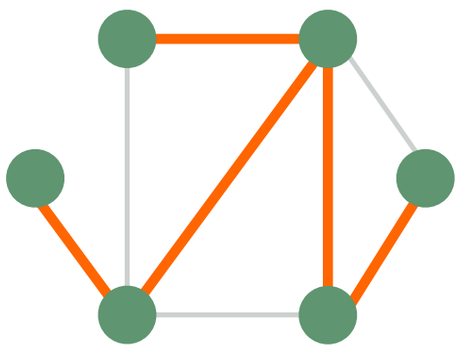

部分問題に分割

$P(\emptyset, \emptyset)$

$z^\star = 8$



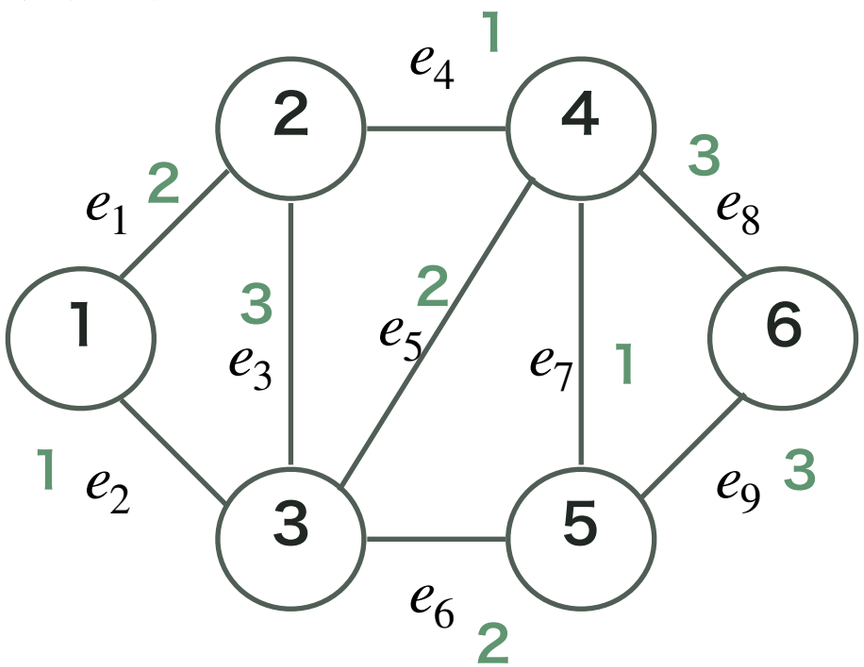
$P(\emptyset, \{e_1\})$ $z^\star = 8$



最小全域木を一つ見つける

# アルゴリズムの実行例

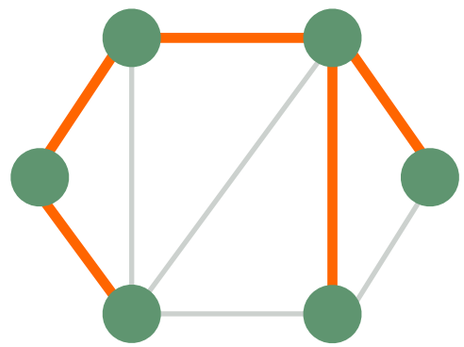
入力グラフ



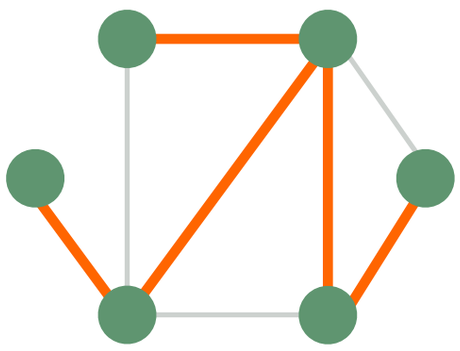

部分問題に分割

$P(\emptyset, \emptyset)$

$z^\star = 8$



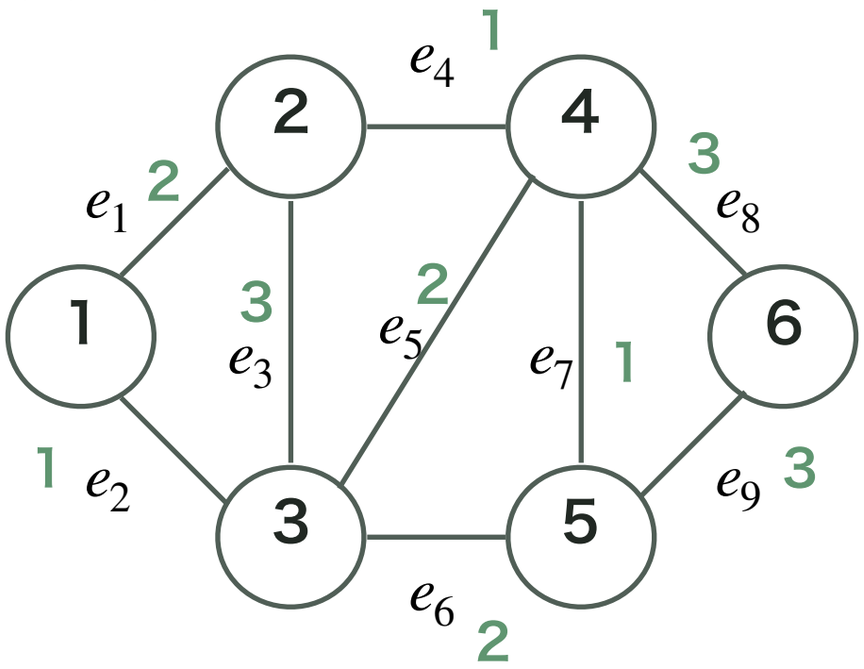
$P(\emptyset, \{e_1\})$ $z^\star = 8$



最小全域木を発見!

# アルゴリズムの実行例

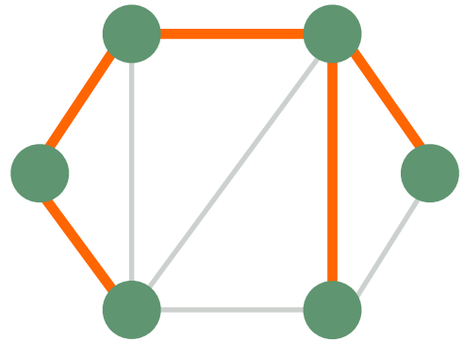
入力グラフ



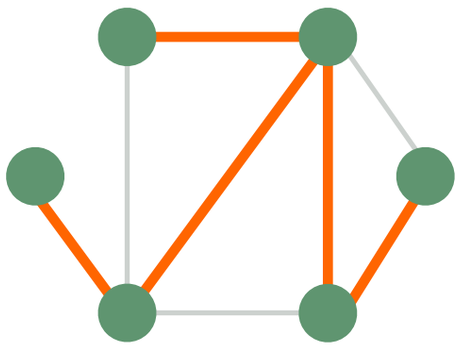

部分問題に分割

$P(\emptyset, \emptyset)$

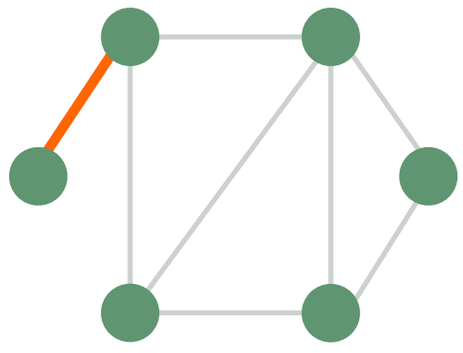
$z^\star = 8$



$P(\emptyset, \{e_1\})$

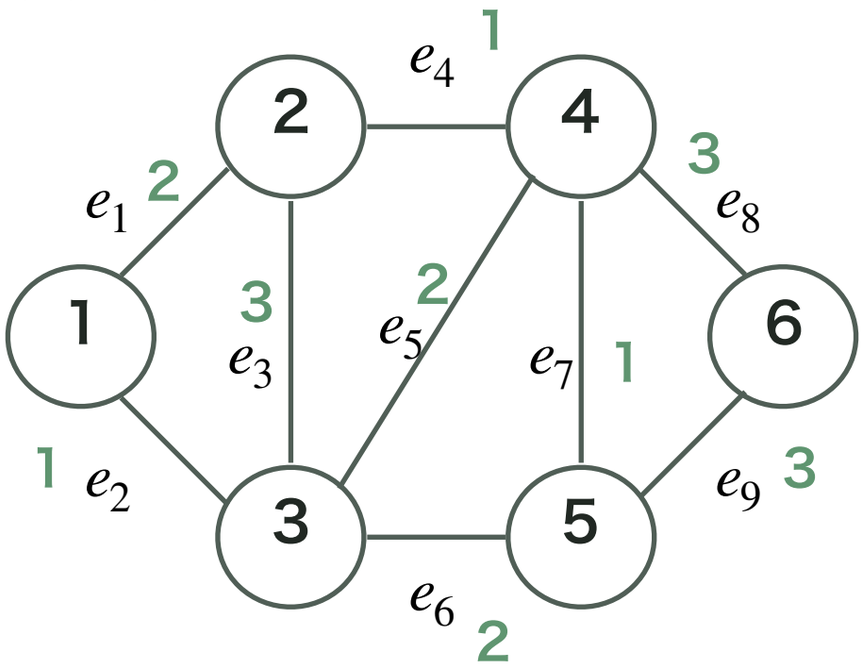


$P(\{e_1\}, \{e_2\})$



# アルゴリズムの実行例

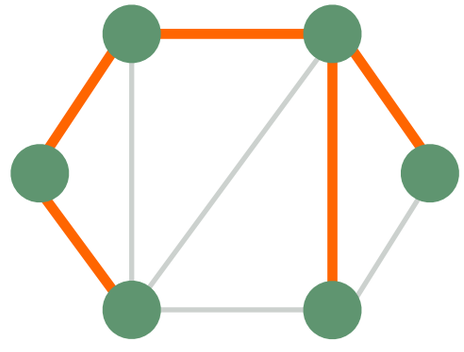
入力グラフ




部分問題に分割

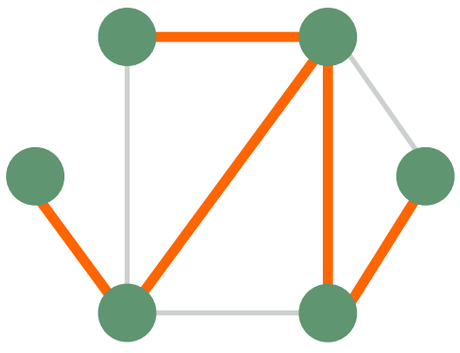
$P(\varnothing, \varnothing)$

$z^\star = 8$

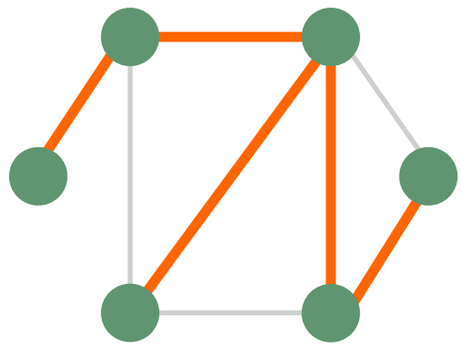


$z^\star = 9$

$P(\varnothing, \{e_1\})$



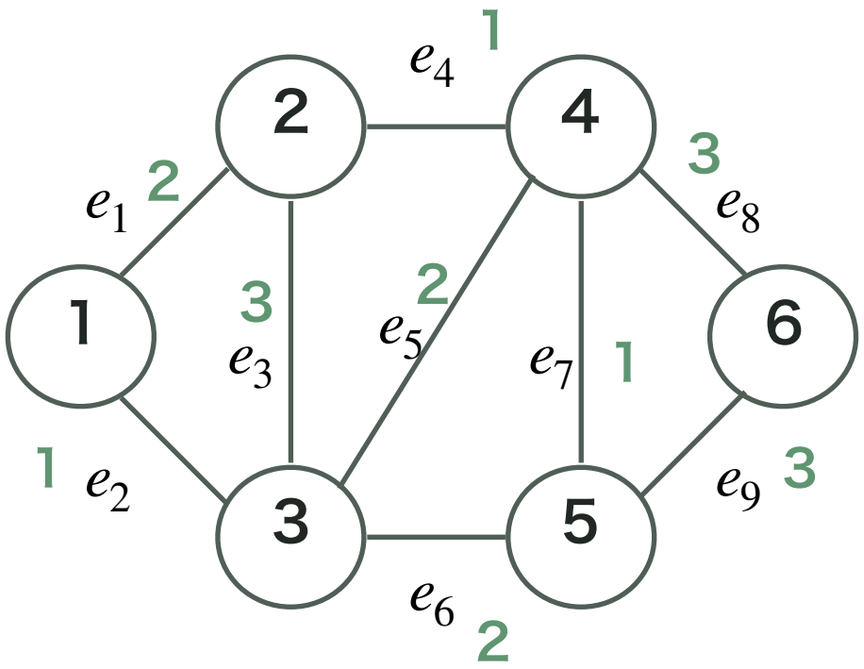
$P(\{e_1\}, \{e_2\})$



最小全域木を一つ見つける

# アルゴリズムの実行例

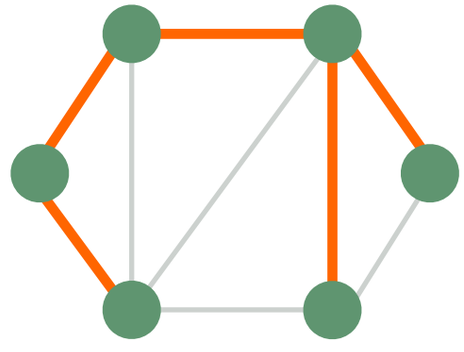
入力グラフ




部分問題に分割

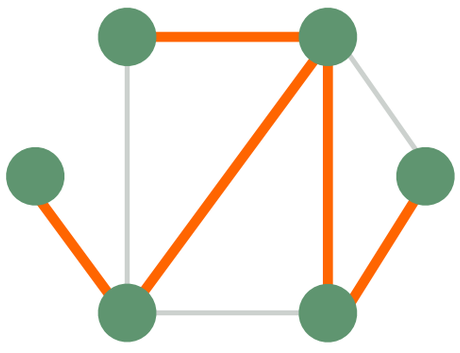
$P(\emptyset, \emptyset)$

$z^\star = 8$



$z^\star = 9$

$P(\emptyset, \{e_1\})$



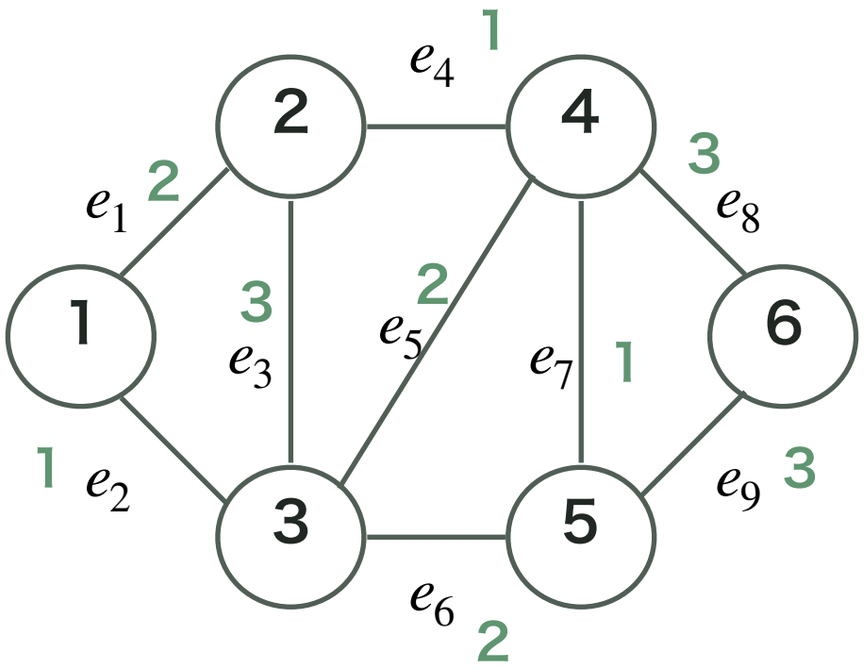
$P(\{e_1\}, \{e_2\})$



プログラム終了...

# アルゴリズムの実行例

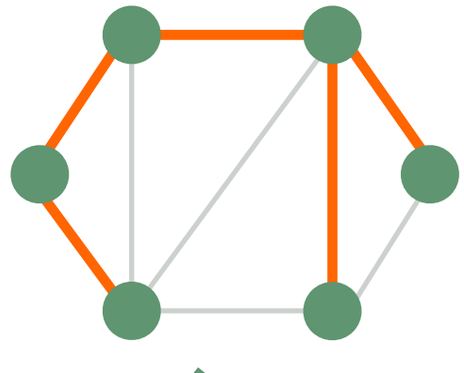
入力グラフ



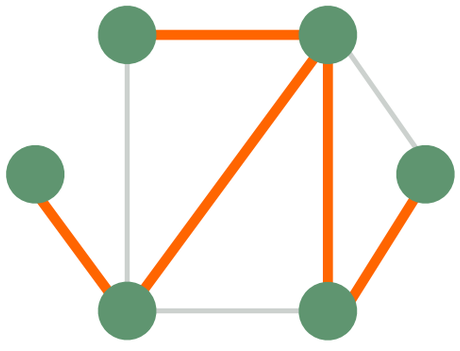

部分問題に分割

$P(\emptyset, \emptyset)$

$z^\star = 8$



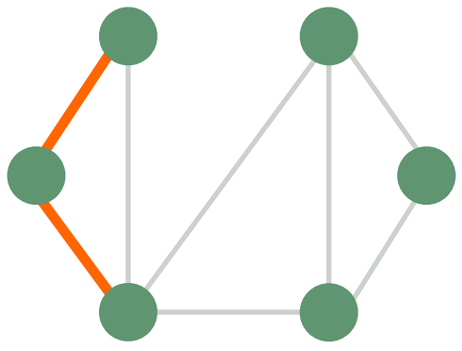
$P(\emptyset, \{e_1\})$



$P(\{e_1\}, \{e_2\})$

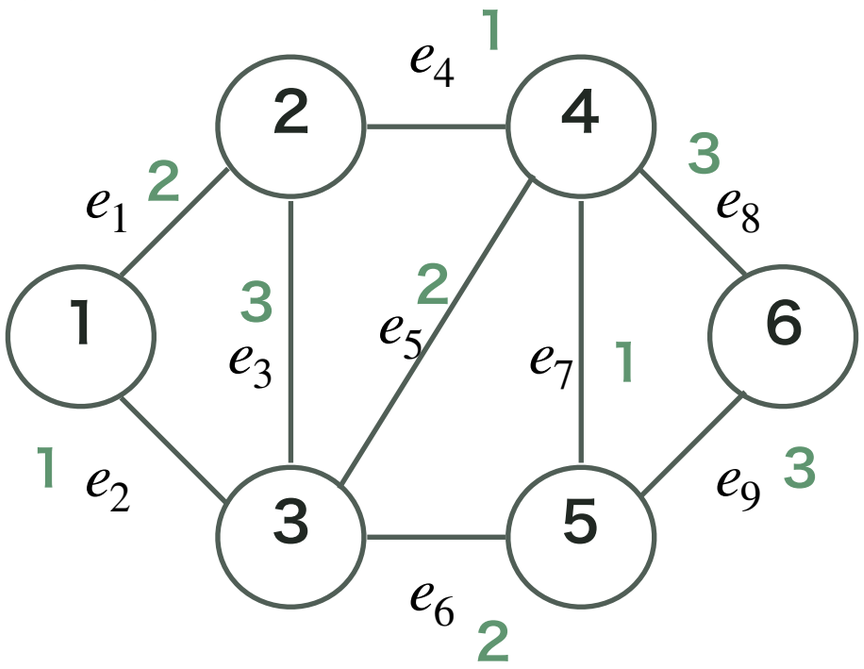


$P(\{e_1, e_2\}, \{e_4\})$



# アルゴリズムの実行例

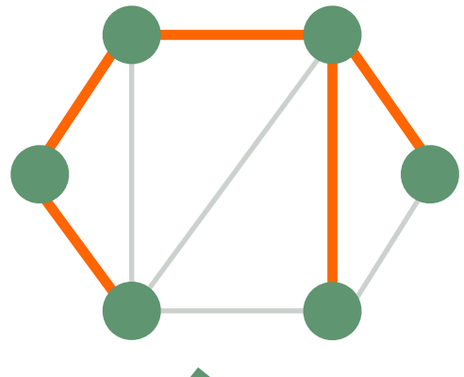
入力グラフ




部分問題に分割

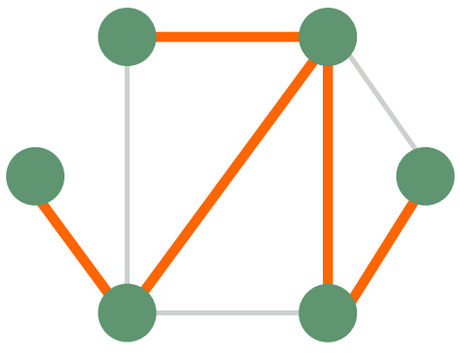
$P(\emptyset, \emptyset)$

$z^\star = 8$



$z^\star = 9$

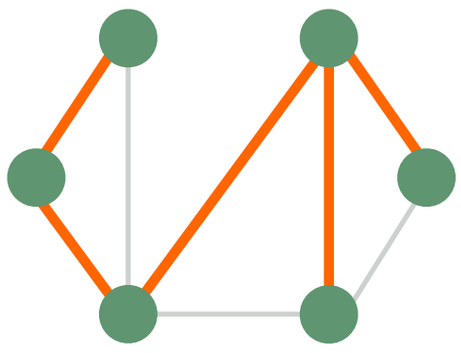
$P(\emptyset, \{e_1\})$



$P(\{e_1\}, \{e_2\})$



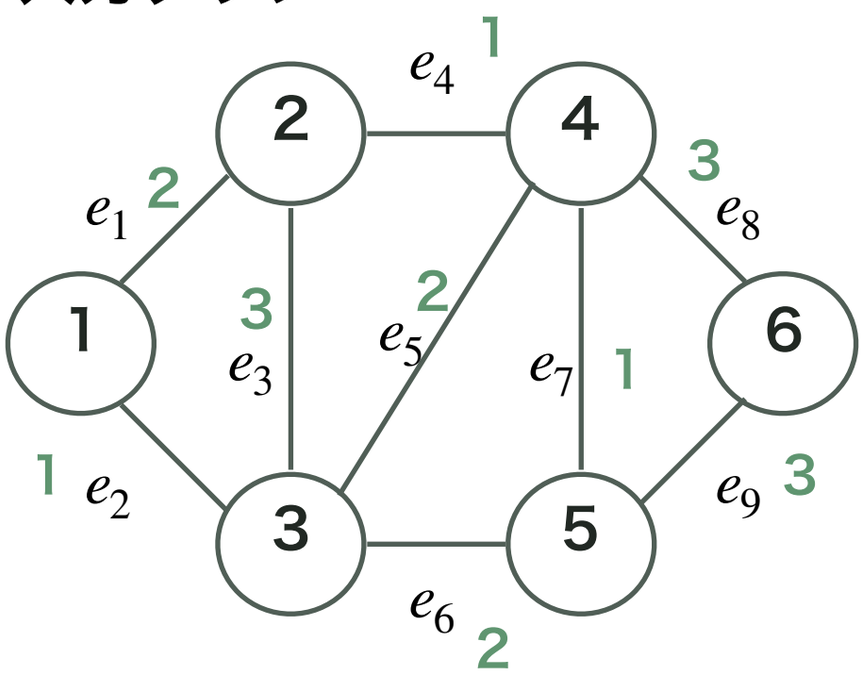
$P(\{e_1, e_2\}, \{e_4\})$



最小全域木を一つ見つける

# アルゴリズムの実行例

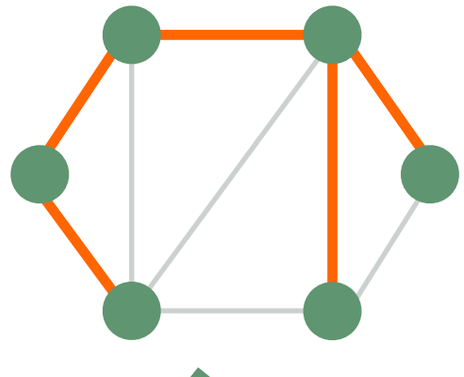
入力グラフ




部分問題に分割

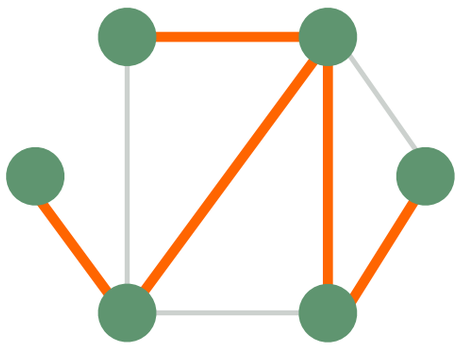
$P(\emptyset, \emptyset)$

$z^\star = 8$



$z^\star = 9$

$P(\emptyset, \{e_1\})$



$P(\{e_1\}, \{e_2\})$



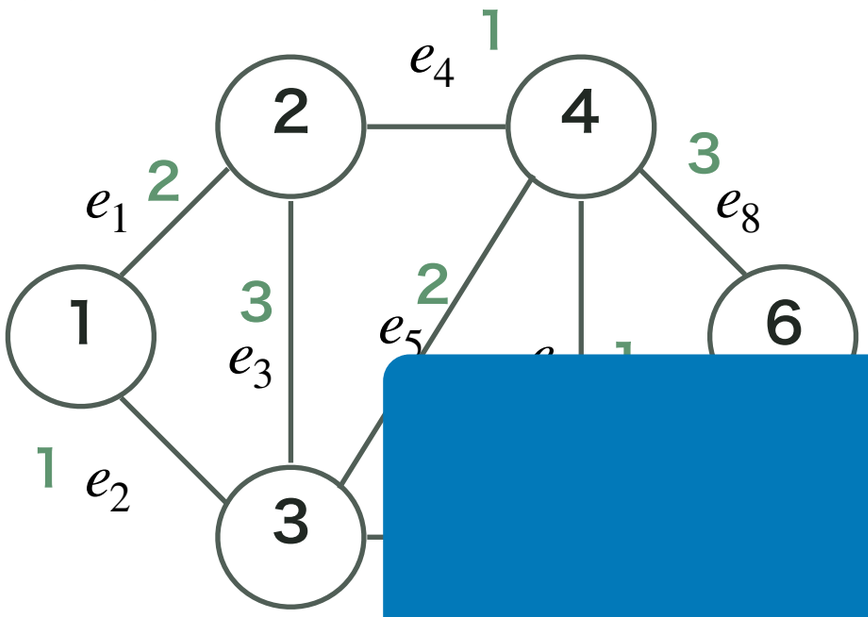
$P(\{e_1, e_2\}, \{e_4\})$



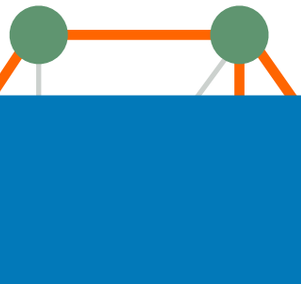
プログラム終了...

# アルゴリズムの実行例

入力グラフ

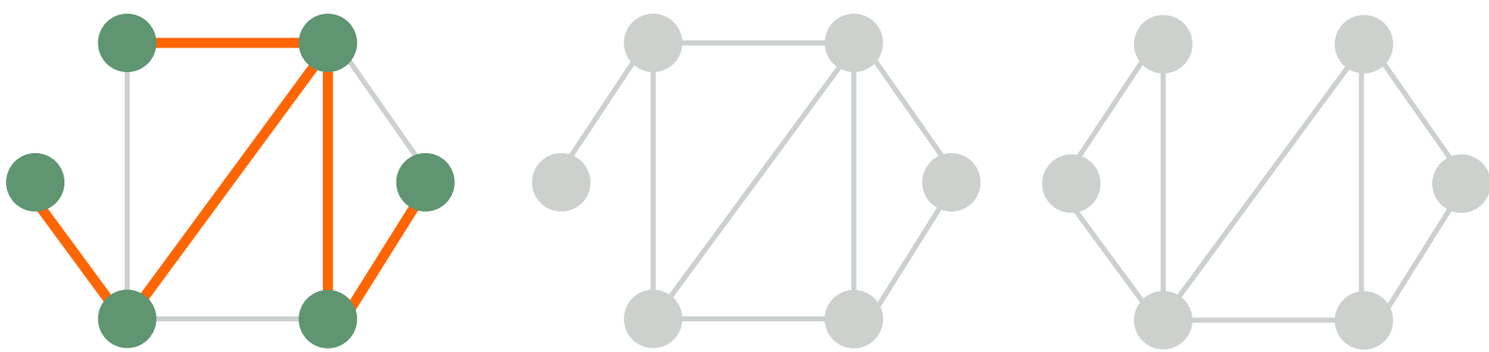



$P(\emptyset, \emptyset)$



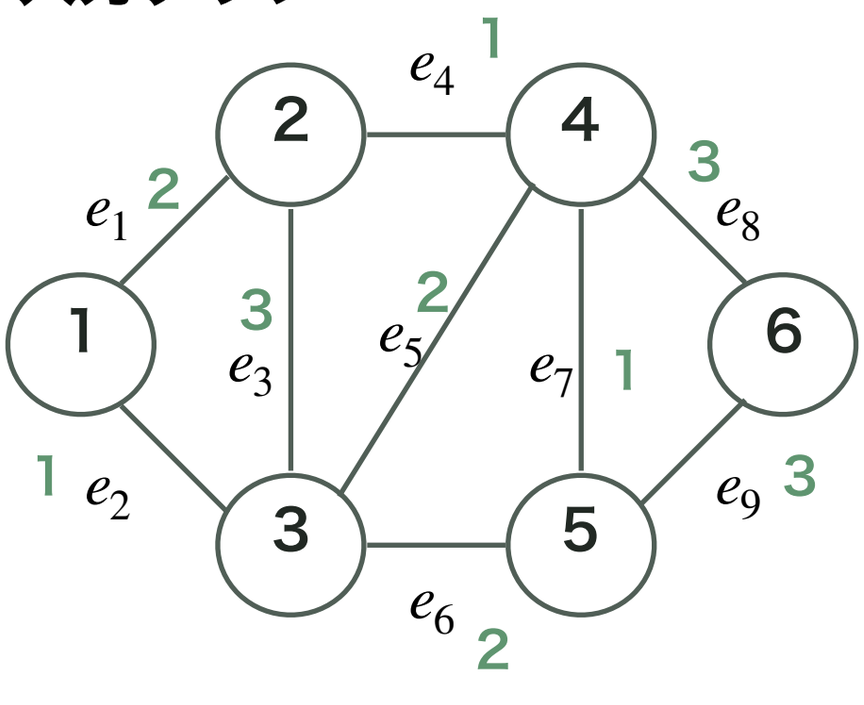
省略

$P(\emptyset, \{e_1\})$

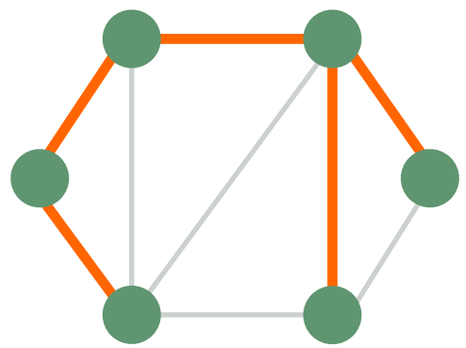


# アルゴリズムの実行例

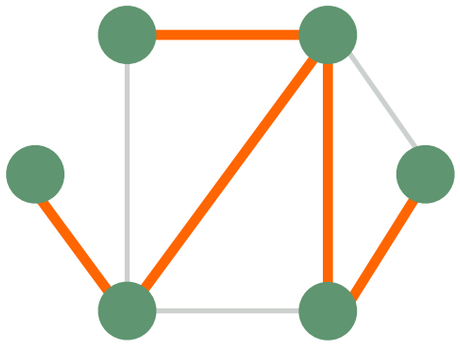
入力グラフ




$P(\varnothing, \varnothing)$



$P(\varnothing, \{e_1\})$



$P(\{e_1\}, \{e_2\})$



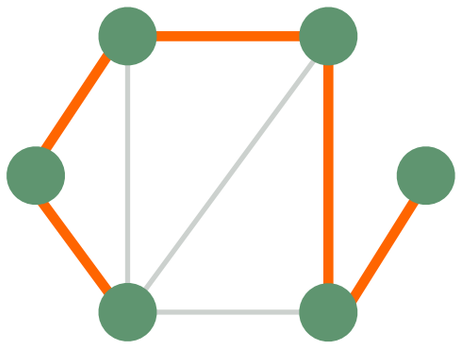
$P(\{e_1, e_2\}, \{e_4\})$



$P(\{e_1, e_2, e_4\}, \{e_7\})$

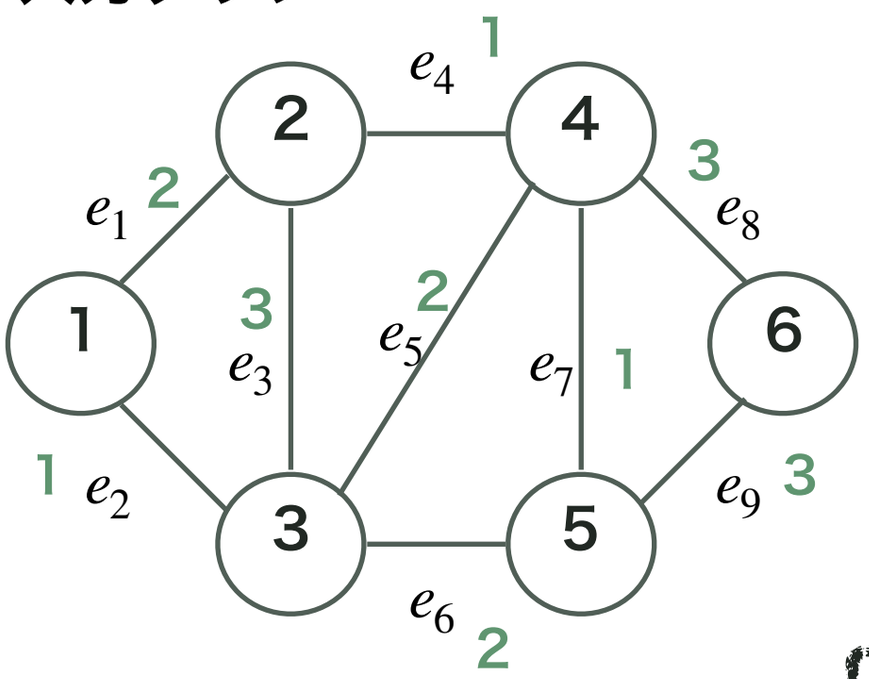


$P(\{e_1, e_2, e_4, e_7\}, \{e_8\})$

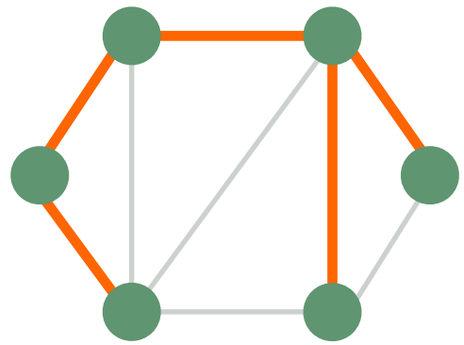


# アルゴリズムの実行例

入力グラフ

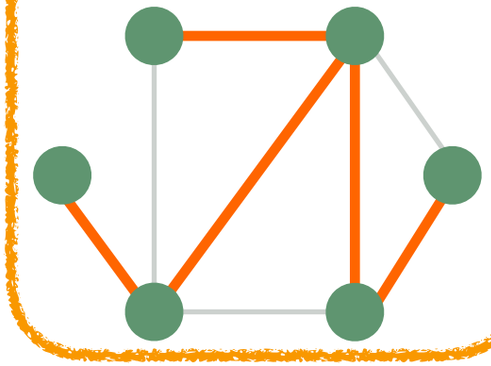



$P(\varnothing, \varnothing)$



**さらに部分問題に分割**

$P(\varnothing, \{e_1\})$



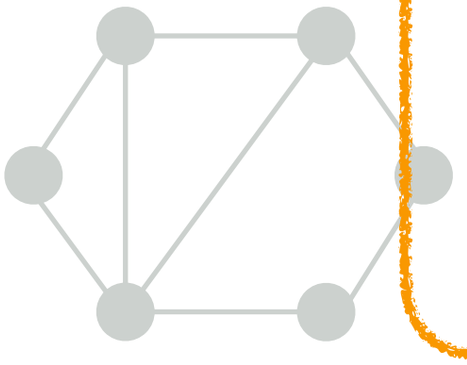
$P(\{e_1\}, \{e_2\})$



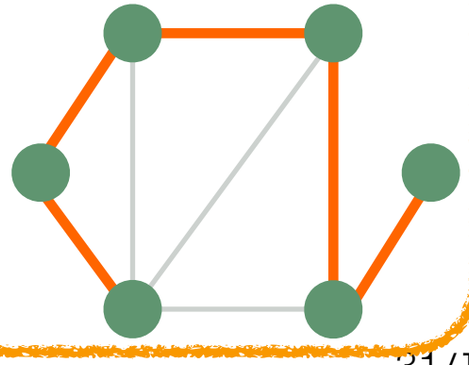
$P(\{e_1, e_2\}, \{e_4\})$



$P(\{e_1, e_2, e_4\}, \{e_7\})$



$P(\{e_1, e_2, e_4, e_7\}, \{e_8\})$



# 時間計算量の解析

- $All\_MST$ の時間計算量は $O(NnT_{MST}(n,m))$

ALGORITHM $All\_MST(F,R)$

// $(F,R)$-許容な最小全域木を全て列挙する

Step 1: $An\_MST(F,R)$ を実行し, $T^\star(F,R)$, $z^\star(F,R)$ を定める.

Step 2: if $z^\star(F,R) > z^\star$ return.

Step 3: $T^\star(F,R)$ を出力

Step 4: for $i \in [k+1, n-1]$ do

$F_i \leftarrow F \cup \{e^{k+1}, \ldots, e^{i-1}\}$, $R_i \leftarrow R \cup \{e^i\}$

$All\_MST(F_i, R_i)$ を再帰呼び出し

# 時間計算量の解析

- $All\_MST$ の時間計算量は $O(NnT_{MST}(n,m))$

ALGORITHM $All\_MST(F,R)$

// $(F,R)$-許容な最小全域木を全て列挙する

Step 1: $An\_MST(F,R)$ を実行し, $T^\star(F,R)$, $z^\star(F,R)$ を定める.

Step 2: if $z^\star(F,R) > z^\star$ return.

Step 3: $T^\star(F,R)$ を出力

Step 4: for $i \in [k+1, n-1]$ do

$F_i \leftarrow F \cup \{e^{k+1}, \ldots, e^{i-1}\}$, $R_i \leftarrow R \cup \{e^i\}$

$All\_MST(F_i, R_i)$ を再帰呼び出し

**解 1 つにつき，$O(n)$ 回再帰呼び出しが行われる**

# 時間計算量の解析

- $All\_MST$ の時間計算量は $O(NnT_{MST}(n,m))$

ALGORITHM $All\_MST(F,R)$

// $(F,R)$-許容な最小全域木を全て列挙する

Step 1: $An\_MST(F,R)$ を実行し, $T^\star(F,R)$, $z^\star(F,R)$ を定める

Step 2: if $z^\star(F,R) > z^\star$ return

Step 3: $T^\star(F,R)$ を出力

Step 4: for $i \in [k+1, n-1]$ do

$F_i \leftarrow F \cup \{e^{k+1}, \ldots, e^{i-1}\}$, $R_i \leftarrow R \cup \{e^i\}$

$All\_MST(F_i, R_i)$ を再帰呼び出し

**解 1 つにつき，$O(n)$ 回再帰呼び出しが行われる**

**$All\_MST(F_i, R_i)$ が全体で呼ばれる回数は $O(Nn)$ 回**

# 目次

- 準備・問題概要
- $ALL\_MST : O(N(mn + n^2 \log n))$
  - 愚直な分割法による列挙
- $ALL\_MST_1 : O(Nmn)$
  - **cut-set を用いた効率の良い列挙方法**
- $ALL\_MST_2 : O(Nm \log n)$
  - 上記アルゴリズムの改良

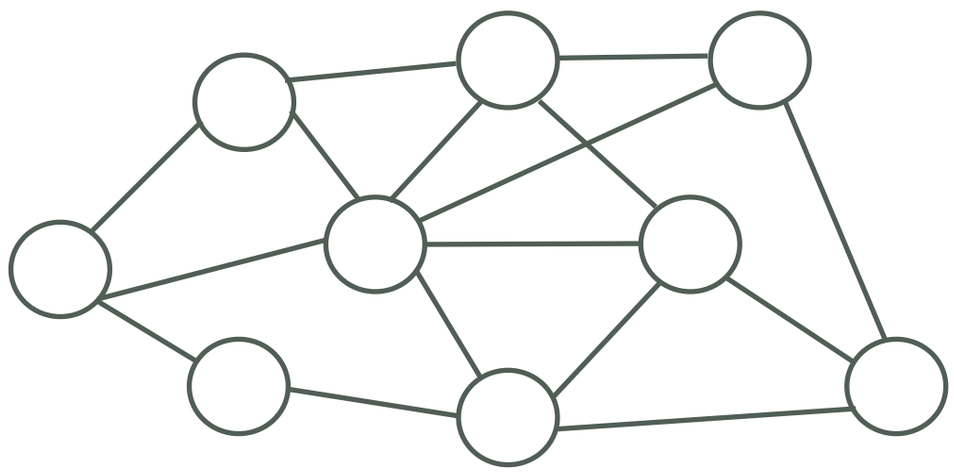
# カットセット

- 全域木 $T$ から $T$ 上の辺 $e$ を削除すると，

  $V$ は 2 つの連結成分 $V_1, V_2$ に分かれる．

- $e$ に対して，カットセット $Cut(e)$ を以下のように定める．

$$Cut(e) := \{e' \in E \mid e' \in (V_1 \times V_2) \cup (V_2 \times V_1)\}$$



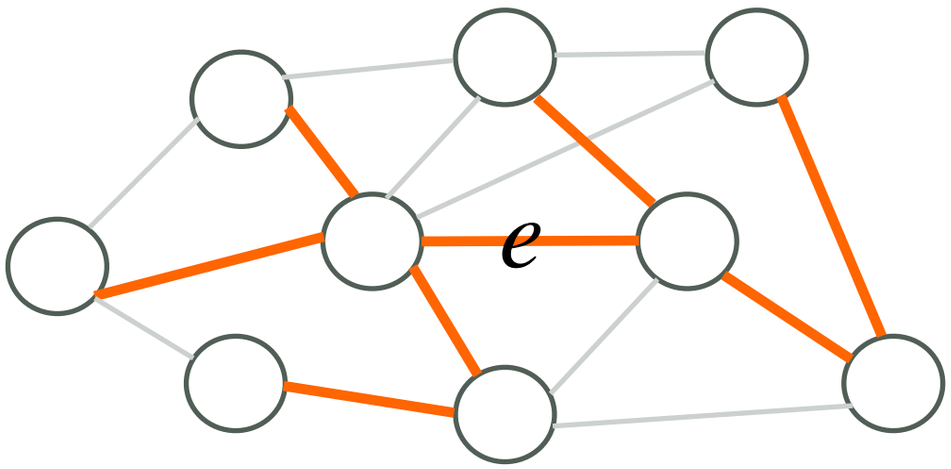
# カットセット

- 全域木 $T$ から $T$ 上の辺 $e$ を削除すると，

  $V$ は 2 つの連結成分 $V_1, V_2$ に分かれる．

- $e$ に対して，カットセット $Cut(e)$ を以下のように定める．

$$Cut(e) := \{e' \in E \mid e' \in (V_1 \times V_2) \cup (V_2 \times V_1)\}$$



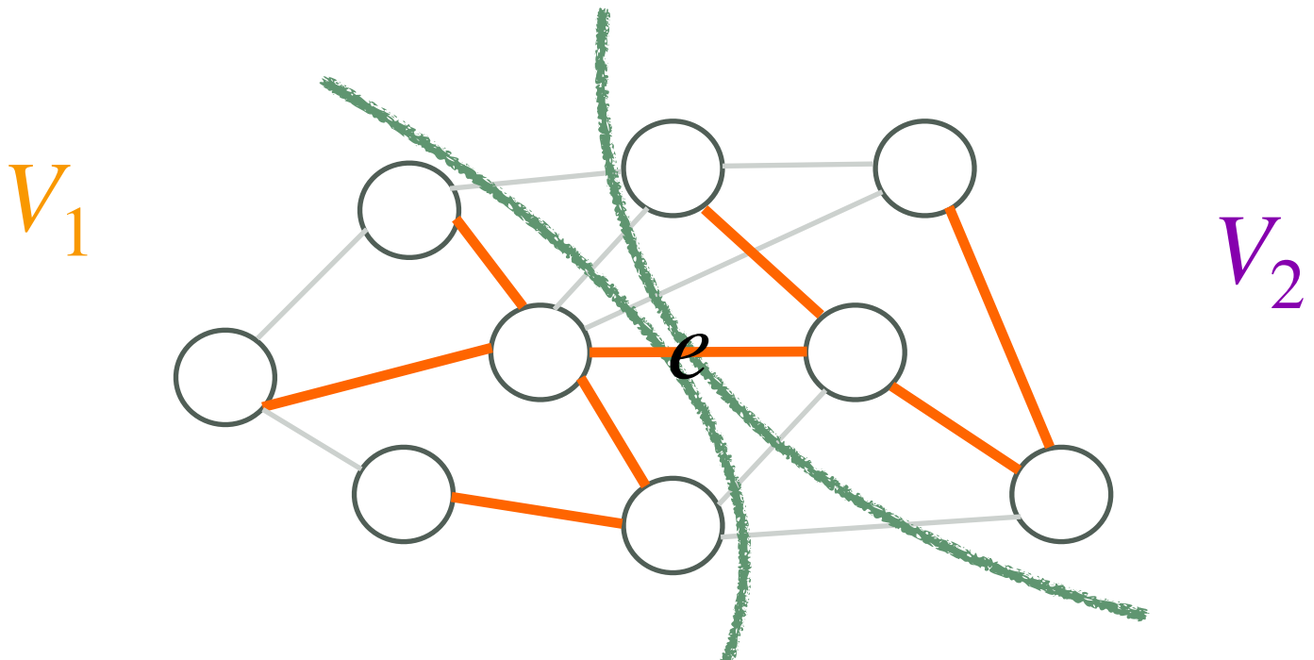
# カットセット

- 全域木 $T$ から $T$ 上の辺 $e$ を削除すると，

  $V$ は 2 つの連結成分 $V_1, V_2$ に分かれる．

- $e$ に対して，カットセット $Cut(e)$ を以下のように定める．

$$Cut(e) := \{e' \in E \mid e' \in (V_1 \times V_2) \cup (V_2 \times V_1)\}$$



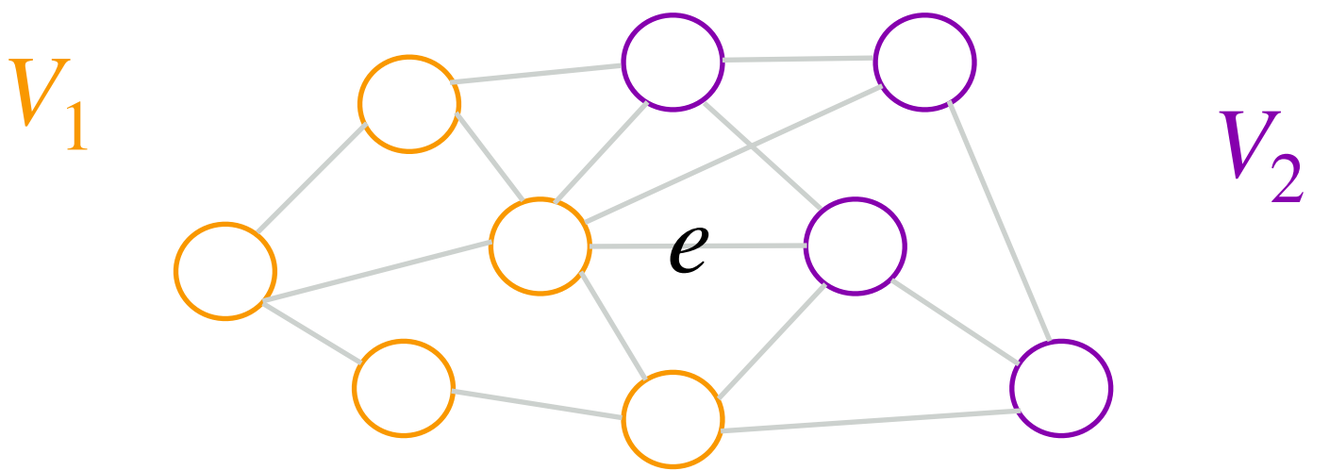
# カットセット

- 全域木 $T$ から $T$ 上の辺 $e$ を削除すると，

  $V$ は 2 つの連結成分 $V_1, V_2$ に分かれる．

- $e$ に対して，カットセット $Cut(e)$ を以下のように定める．

$$Cut(e) := \{e' \in E \mid e' \in (V_1 \times V_2) \cup (V_2 \times V_1)\}$$



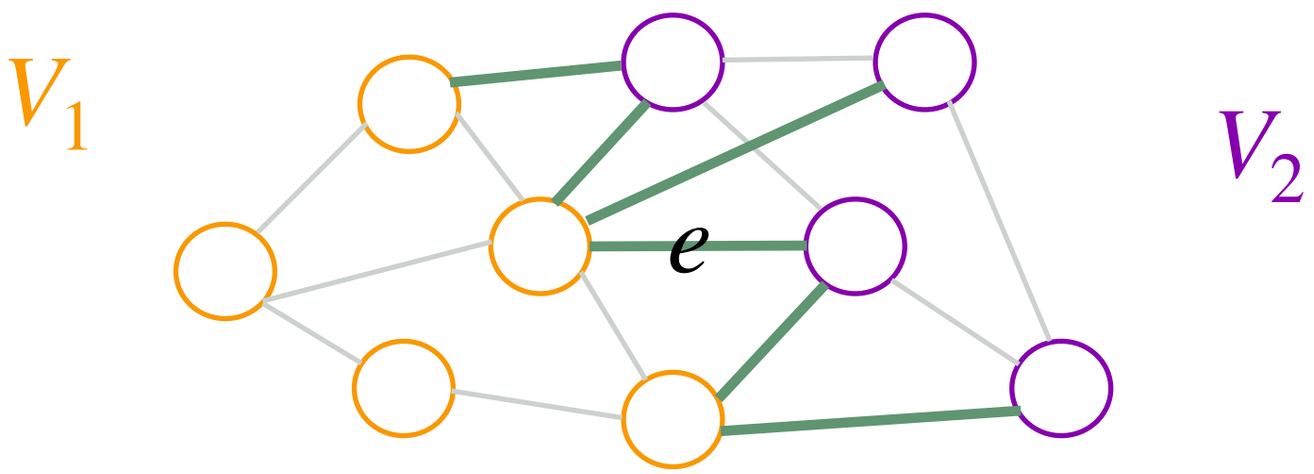
# カットセット

- 全域木 $T$ から $T$ 上の辺 $e$ を削除すると，

  $V$ は 2 つの連結成分 $V_1, V_2$ に分かれる．

- $e$ に対して，カットセット $Cut(e)$ を以下のように定める．

$$Cut(e) := \{e' \in E \mid e' \in (V_1 \times V_2) \cup (V_2 \times V_1)\}$$

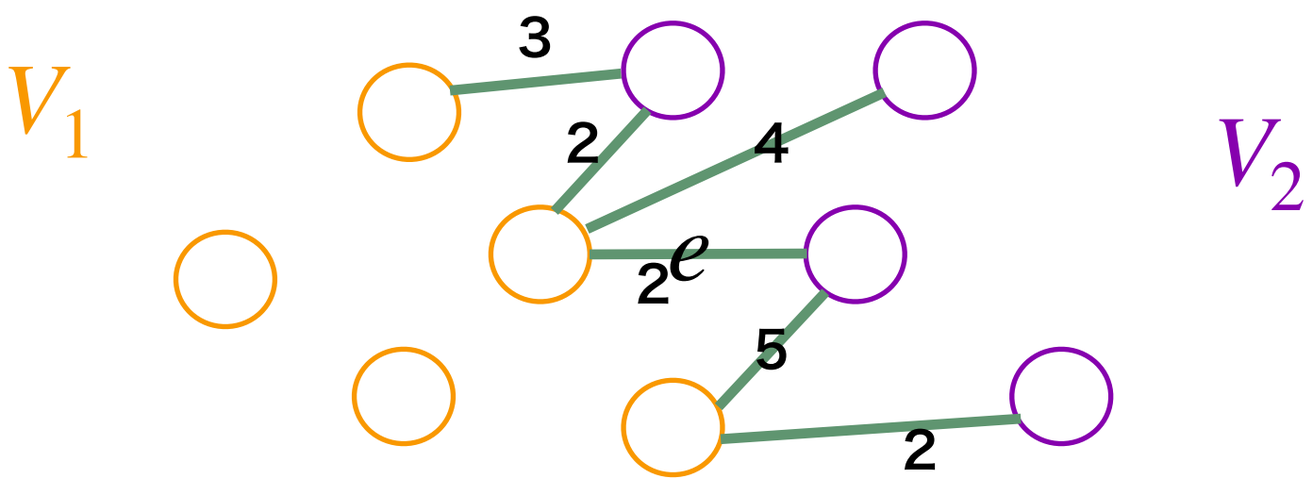


# 代替辺

- $T$ を全域木，$e \in T$ とする．
- ここで，$S(e)$ を以下のように定める．

$$S(e) := \{\tilde{e} \in Cut(e) \mid \tilde{e} \neq e, w(\tilde{e}) = w(e)\}$$

  - $S(e)$ の要素（の内の一つ）を $e$ の代替辺と呼ぶ

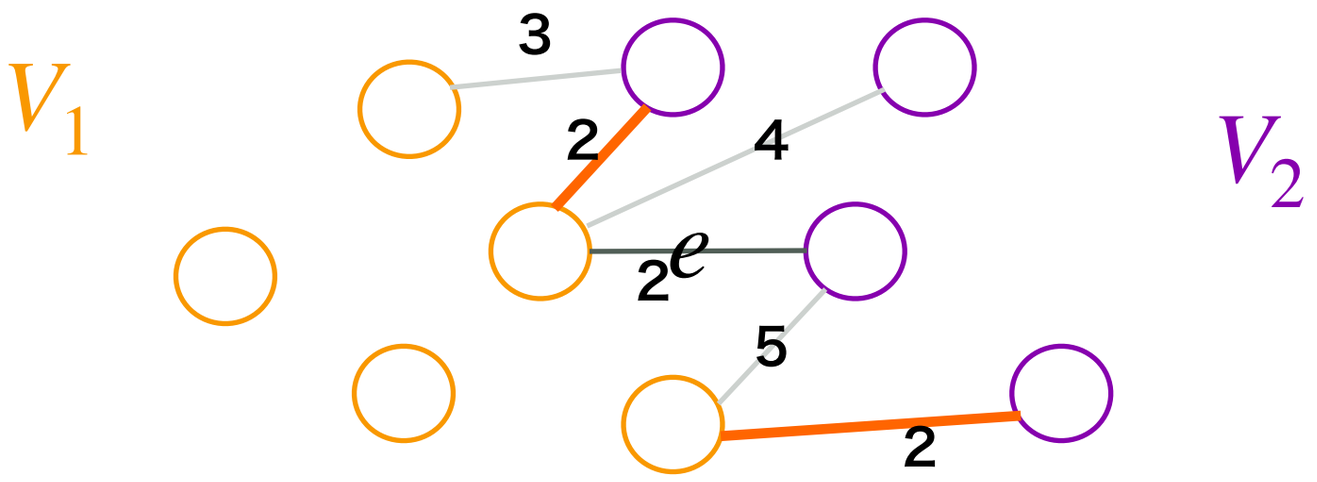


# 代替辺

- $T$ を全域木，$e \in T$ とする．
- ここで，$S(e)$ を以下のように定める．

$$S(e) := \{\tilde{e} \in Cut(e) \mid \tilde{e} \neq e, w(\tilde{e}) = w(e)\}$$

  - $S(e)$ の要素（の内の一つ）を $e$ の代替辺と呼ぶ

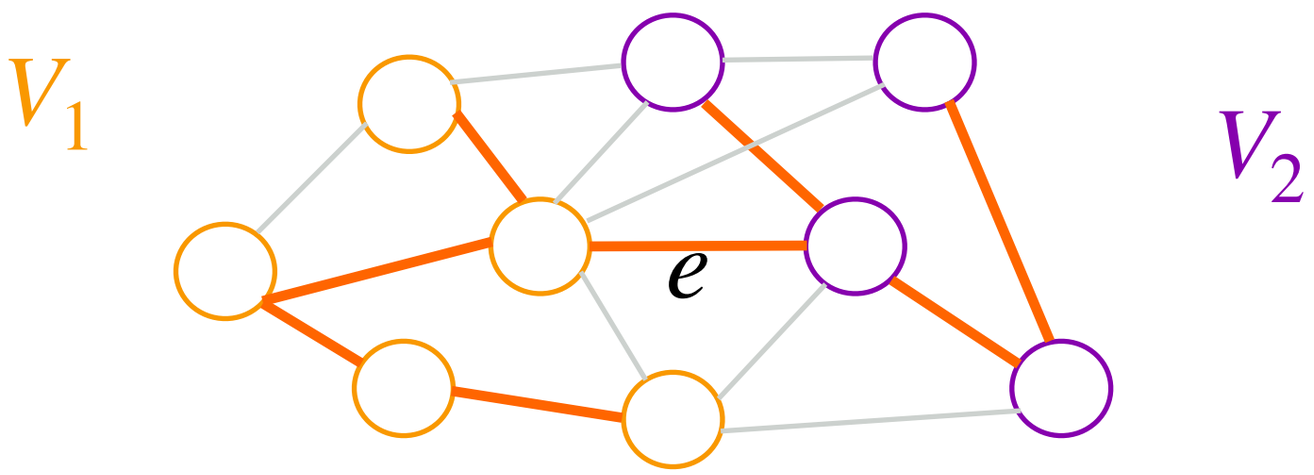


# アルゴリズムの構築（定理1）

[定理 1]

$T$ を最小全域木，$e \in T$ とする．

全ての $e' \in Cut(e)$ に対して，$w(e') \geq w(e)$ が成立．



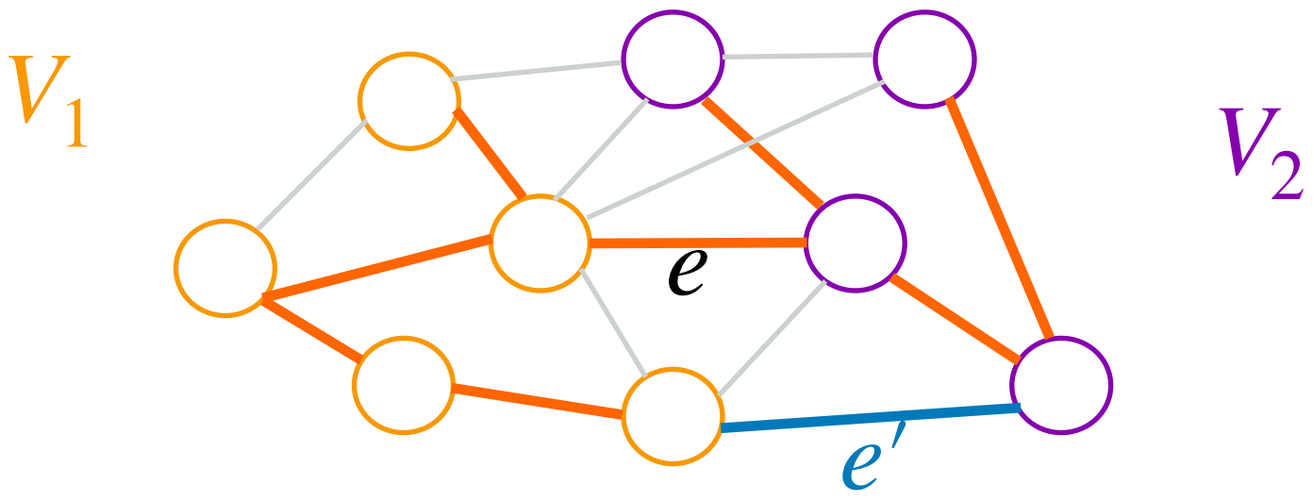
# アルゴリズムの構築（定理1）

[定理 1]

$T$ を最小全域木，$e \in T$ とする．

全ての $e' \in Cut(e)$ に対して，$w(e') \geq w(e)$ が成立．

- $w(e') < w(e)$ を満たす $e'$ が存在すると仮定する．



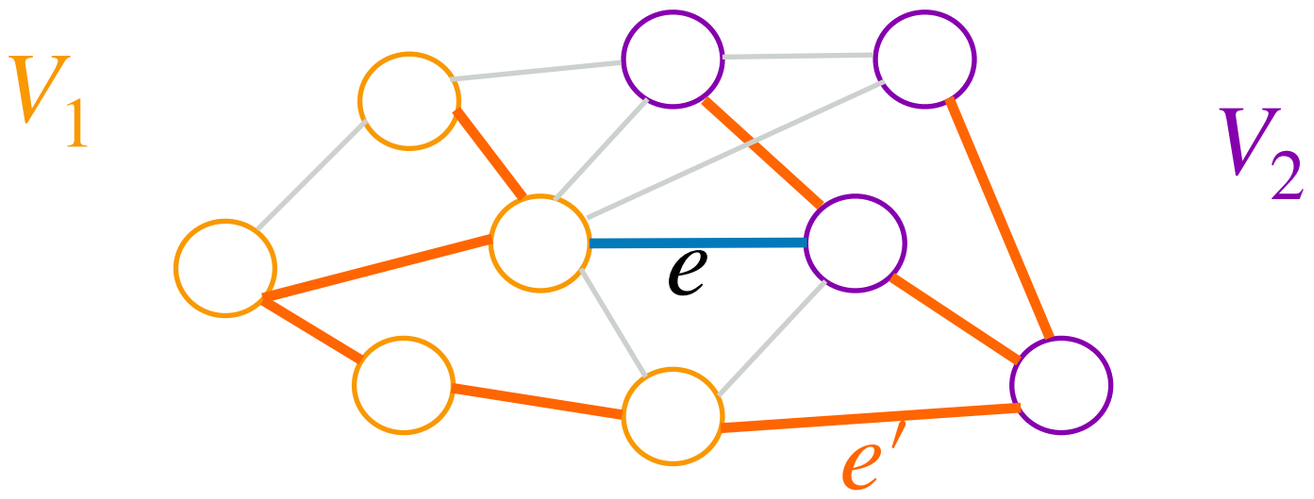
# アルゴリズムの構築（定理1）

[定理 1]

$T$ を最小全域木，$e \in T$ とする．

全ての $e' \in Cut(e)$ に対して，$w(e') \geq w(e)$ が成立．

- $w(e') < w(e)$ を満たす $e'$ が存在すると仮定する．
- $e$ と $e'$ を入れ替えると，$T$ より重みの小さい全域木が作れる



# アルゴリズムの構築（定理1）

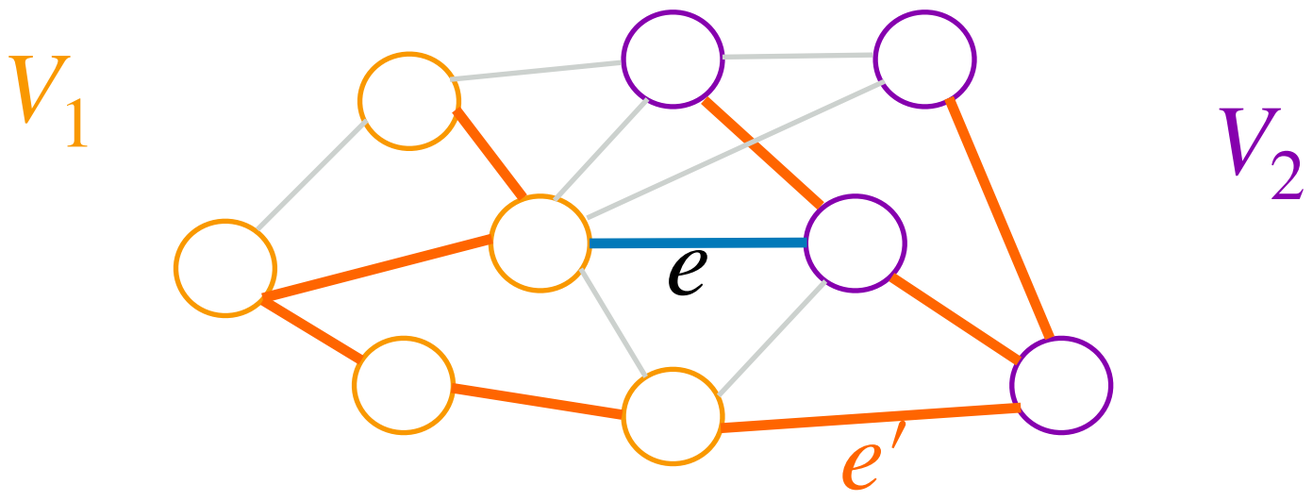
[定理 1]

$T$ を最小全域木，$e \in T$ とする．

全ての $e' \in Cut(e)$ に対して，$w(e') \geq w(e)$ が成立．

- $w(e') < w(e)$ を満たす $e'$ が存在すると仮定する．
- $e$ と $e'$ を入れ替えると，$T$ より重みの小さい全域木が作れる

  → $T$ の最小性に矛盾



# アルゴリズムの構築（定理2）

[定理 2]

$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 1 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

- **注意**
  - $T \cup \{e'\} \backslash \{e\}$ を単に $T \cup e' \backslash e$ と表記する
  - $G$ から $e$ を取り除いて得られるグラフを $G^{(e)}$ と表記する

# アルゴリズムの構築（定理2）

[定理 2]

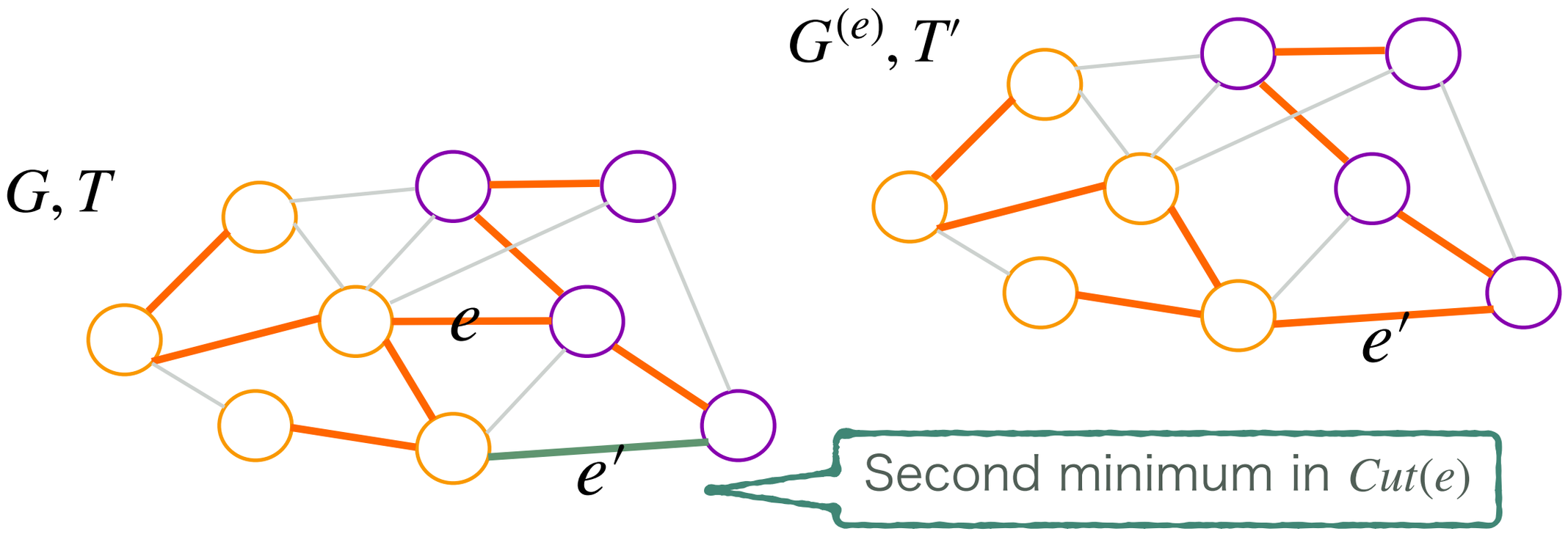
$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 2 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

- 直感的な意味

$G^{(e)}$ の最小全域木が $T'$ となる



# アルゴリズムの構築（定理2）

[定理 2]

$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 2 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

［証明］

$T'$ が $G^{(e)}$ における最小全域木でないと仮定する.

すなわち，$T'' := T' \cup c \backslash d$ が全域木で $w(c) < w(d)$ を満たす $c \notin T', d \in T'$ が存在.

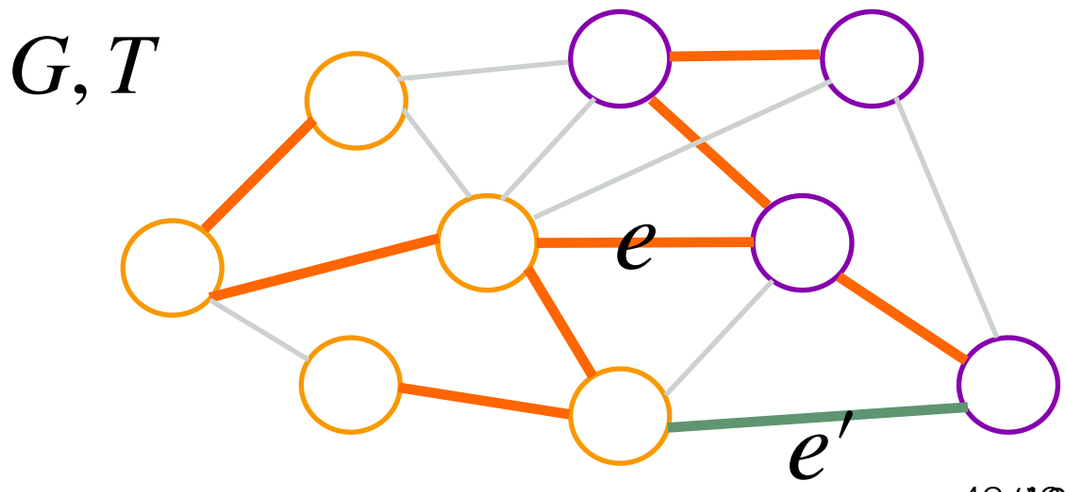
ここで，$c \in Cut(e)$ であることが言える.

$c \notin Cut(e)$ ならば，$\tilde{T} := T \cup c \backslash d$ が全域木かつ $w(\tilde{T}) < z^{\star}$ となり，

$T$ の最小性に反するからである.

これまでの議論より，以下の不等式が成り立つ.

$$w(e) \leq w(e') \leq w(c) < w(d) \cdots (1)$$



# アルゴリズムの構築（定理2）

[定理 2]

$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 2 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

背理法の仮定

［証明］

$T'$ が $G^{(e)}$ における最小全域木でないと仮定する.

すなわち，$T'' := T' \cup c \backslash d$ が全域木で $w(c) < w(d)$ を満たす $c \notin T', d \in T'$ が存在.

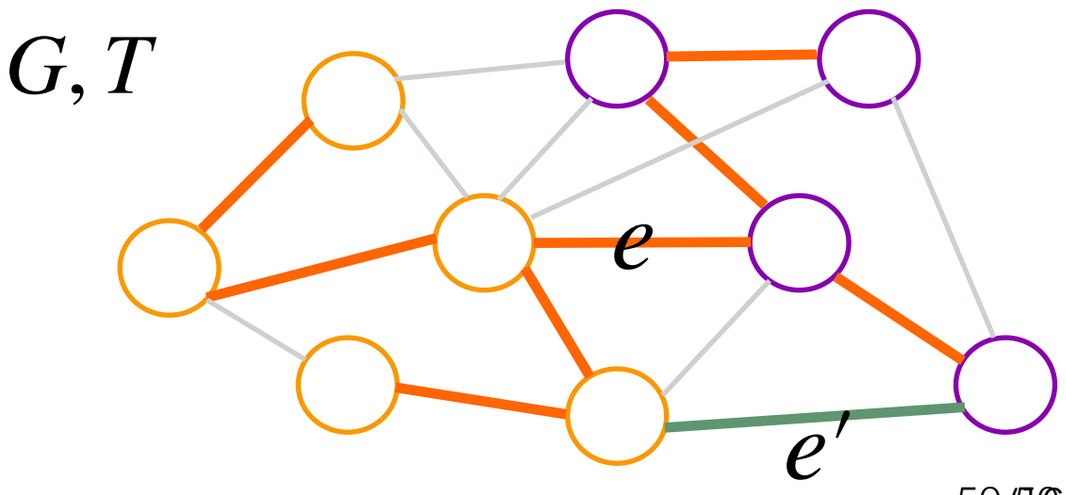
ここで，$c \in Cut(e)$ であることが言える.

$c \notin Cut(e)$ ならば，$\tilde{T} := T \cup c \backslash d$ が全域木かつ $w(\tilde{T}) < z^{\star}$ となり，

$T$ の最小性に反するからである.

これまでの議論より，以下の不等式が成り立つ.

$$w(e) \leq w(e') \leq w(c) < w(d) \cdots (1)$$



# アルゴリズムの構築（定理2）

[定理 2]

$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 2 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

$c \in Cut(e)$ の証明

［証明］

$T'$ が $G^{(e)}$ における最小全域木でないと仮定する.

すなわち，$T'' := T' \cup c \backslash d$ が全域木で $w(c) < w(d)$ を満たす $c \notin T', d \in T'$ が存在.

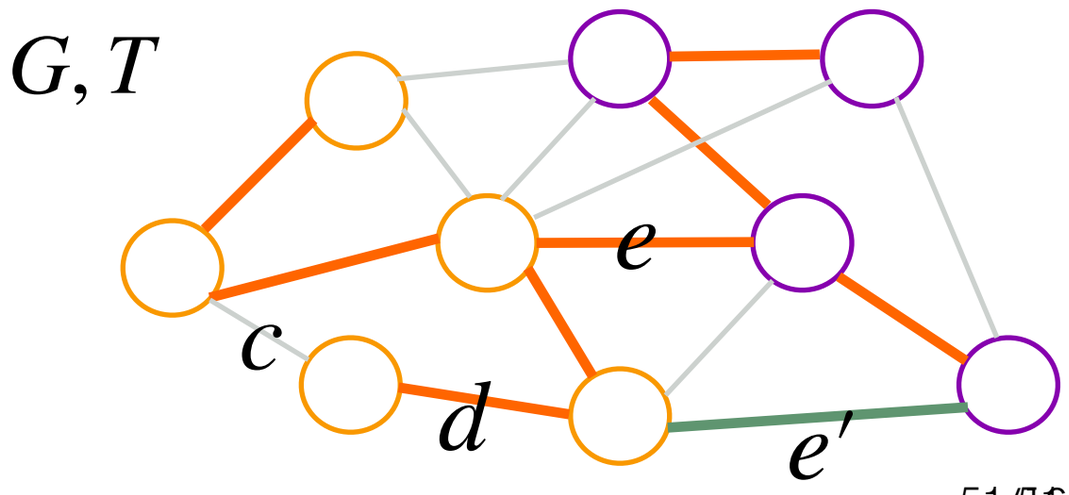
ここで，$c \in Cut(e)$ であることが言える.

$c \notin Cut(e)$ ならば，$\tilde{T} := T \cup c \backslash d$ が全域木かつ $w(\tilde{T}) < z^{\star}$ となり，

$T$ の最小性に反するからである.

これまでの議論より，以下の不等式が成り立つ.

$$w(e) \leq w(e') \leq w(c) < w(d) \cdots (1)$$



# アルゴリズムの構築（定理2）

[定理 2]

$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 2 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

［証明］

ここで，サイクル $T \cup \{e'\}, T \cup \{c\}, T' \cup \{c\}$ をそれぞれ $C_0, C_1, C_2$ とする

$C_2 = C_0 \oplus C_1$ かつ $d \in C_2$ であるので，$d \in C_0$ or $d \in C_1$ が言える

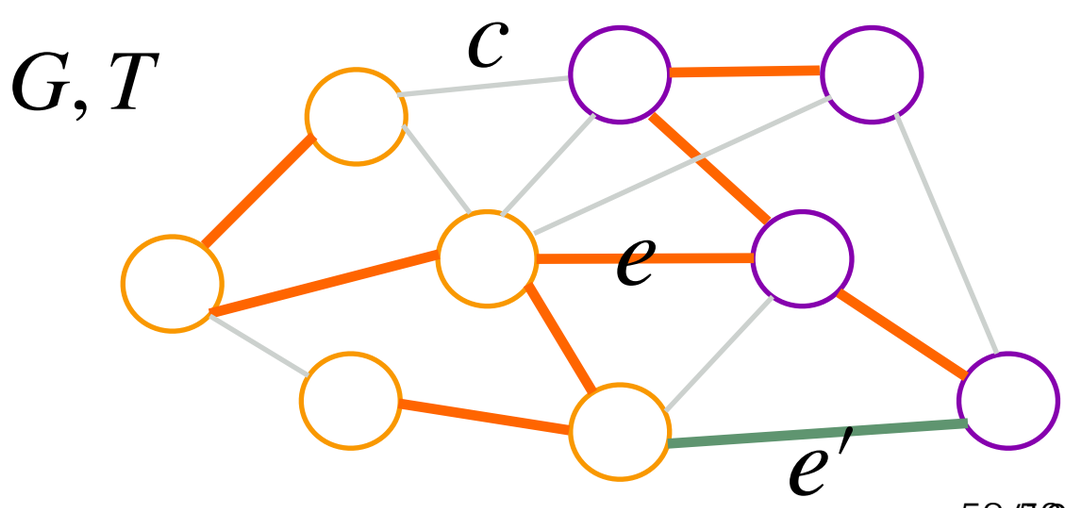
(1) $d \in C_0$ の時,

$\hat{T} = T \cup e' \backslash d$ とすると，$\hat{T}$ は全域木かつ $w(\hat{T}) < z^\star$ となり矛盾

(1) $d \in C_1$ の時,

$\hat{T} = T \cup c \backslash d$ とすると，同様に矛盾

よって，定理 2 が示された.



# アルゴリズムの構築（定理2）

[定理 2]

$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 2 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

［証明］

ここで，サイクル $T \cup \{e'\}, T \cup \{c\}, T' \cup \{c\}$ をそれぞれ $C_0, C_1, C_2$ とする

$C_2 = C_0 \oplus C_1$ かつ $d \in C_2$ であるので，$d \in C_0$ or $d \in C_1$ が言える

$C_0$

(1) $d \in C_0$ の時,

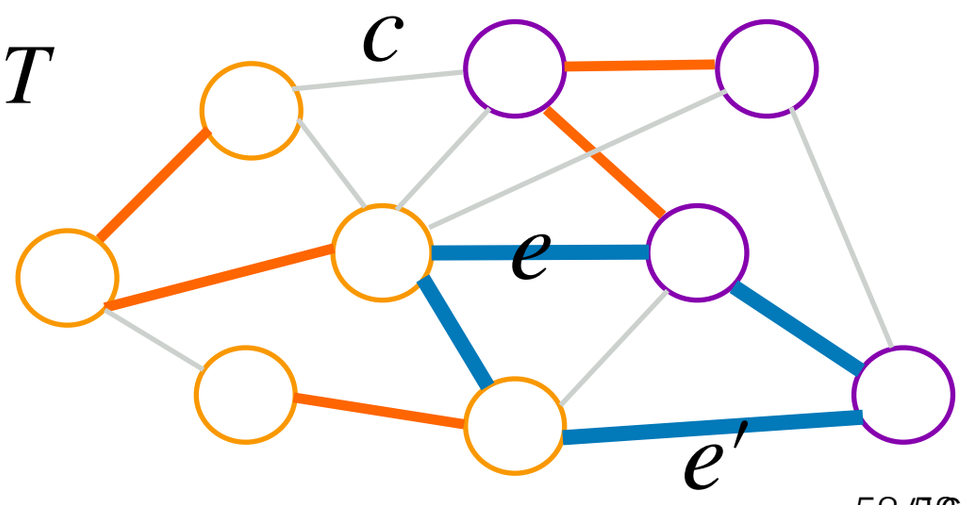
$\hat{T} = T \cup e' \backslash d$ とすると，$\hat{T}$ は全域木かつ $w(\hat{T}) < z^\star$ となり矛盾

(1) $d \in C_1$ の時,

$\hat{T} = T \cup c \backslash d$ とすると，同様に矛盾

よって，定理 2 が示された.

$G, T$



# アルゴリズムの構築（定理2）

[定理 2]

$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 2 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

［証明］

ここで，サイクル $T \cup \{e'\}, T \cup \{c\}, T' \cup \{c\}$ をそれぞれ $C_0, C_1, C_2$ とする

$C_2 = C_0 \oplus C_1$ かつ $d \in C_2$ であるので，$d \in C_0$ or $d \in C_1$ が言える

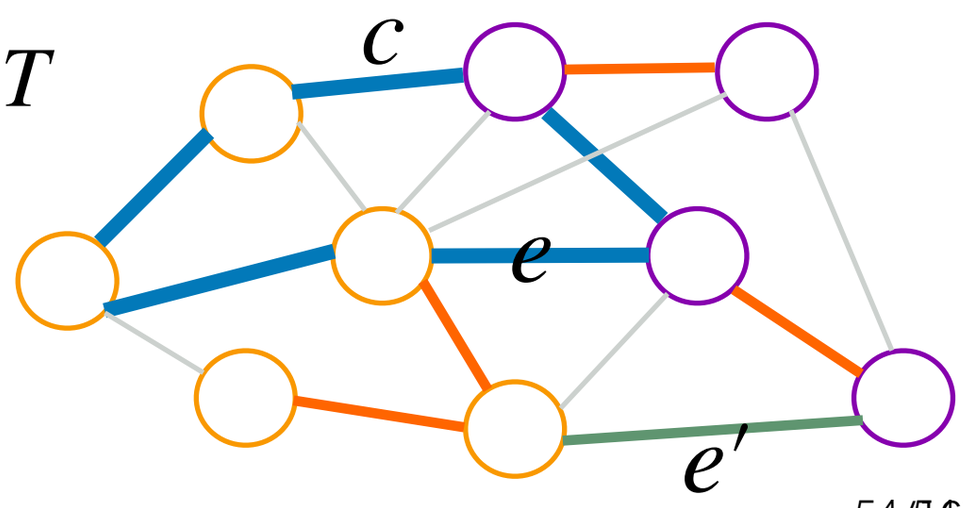
(1) $d \in C_0$ の時,

$\hat{T} = T \cup e' \backslash d$ とすると，$\hat{T}$ は全域木かつ $w(\hat{T}) < z^\star$ となり矛盾

(1) $d \in C_1$ の時,

$\hat{T} = T \cup c \backslash d$ とすると，同様に矛盾

よって，定理 2 が示された.

# アルゴリズムの構築（定理2）

[定理 2]

$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 2 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

［証明］

ここで，サイクル $T \cup \{e'\}, T \cup \{c\}, T' \cup \{c\}$ をそれぞれ $C_0, C_1, C_2$ とする

$C_2 = C_0 \oplus C_1$ かつ $d \in C_2$ であるので，$d \in C_0$ or $d \in C_1$ が言える

$C_2$

(1) $d \in C_0$ の時,

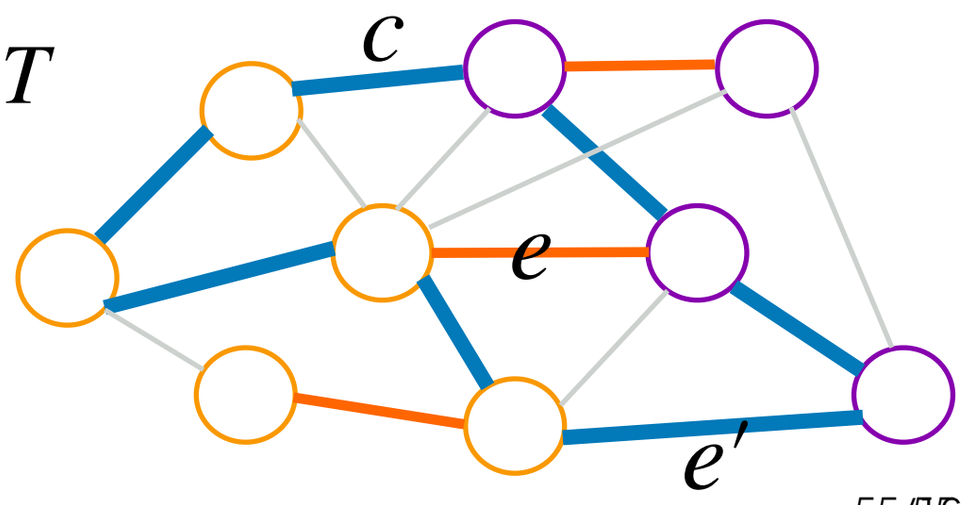
$\hat{T} = T \cup e' \backslash d$ とすると，$\hat{T}$ は全域木かつ $w(\hat{T}) < z^\star$ となり矛盾

(1) $d \in C_1$ の時,

$\hat{T} = T \cup c \backslash d$ とすると，同様に矛盾

よって，定理 2 が示された.

$G, T$



# アルゴリズムの構築（定理2）

[定理 2]

$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 2 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

［証明］

ここで，サイクル $T \cup \{e'\}, T \cup \{c\}, T' \cup \{c\}$ をそれぞれ $C_0, C_1, C_2$ とする

$C_2 = C_0 \oplus C_1$ かつ $d \in C_2$ であるので，$d \in C_0$ or $d \in C_1$ が言える

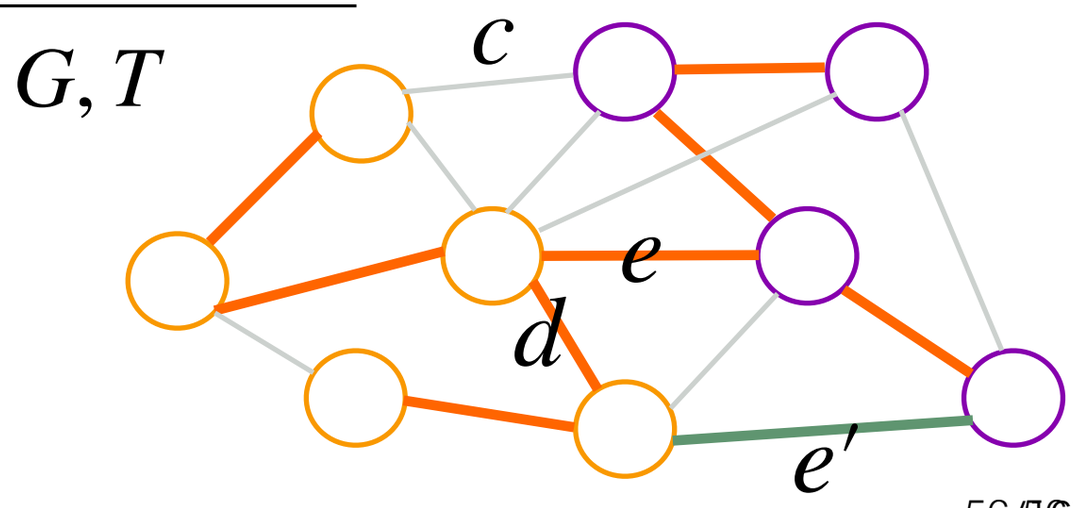
(1) $d \in C_0$ の時,

$\hat{T} = T \cup e' \backslash d$ とすると，$\hat{T}$ は全域木かつ $w(\hat{T}) < z^\star$ となり矛盾

(1) $d \in C_1$ の時,

$\hat{T} = T \cup c \backslash d$ とすると，同様に矛盾

よって，定理 2 が示された.



# アルゴリズムの構築（定理2）

[定理 2]

$T$ を最小全域木，$e \in T$ ,

$e'(\neq e)$ を $Cut(e)$ で 2 番目に重みの小さい辺とする.

この時，$T' := T \cup e' \backslash e$ は $G^{(e)}$ 上での最小全域木となる

［証明］

ここで，サイクル $T \cup \{e'\}, T \cup \{c\}, T' \cup \{c\}$ をそれぞれ $C_0, C_1, C_2$ とする

$C_2 = C_0 \oplus C_1$ かつ $d \in C_2$ であるので，$d \in C_0$ or $d \in C_1$ が言える

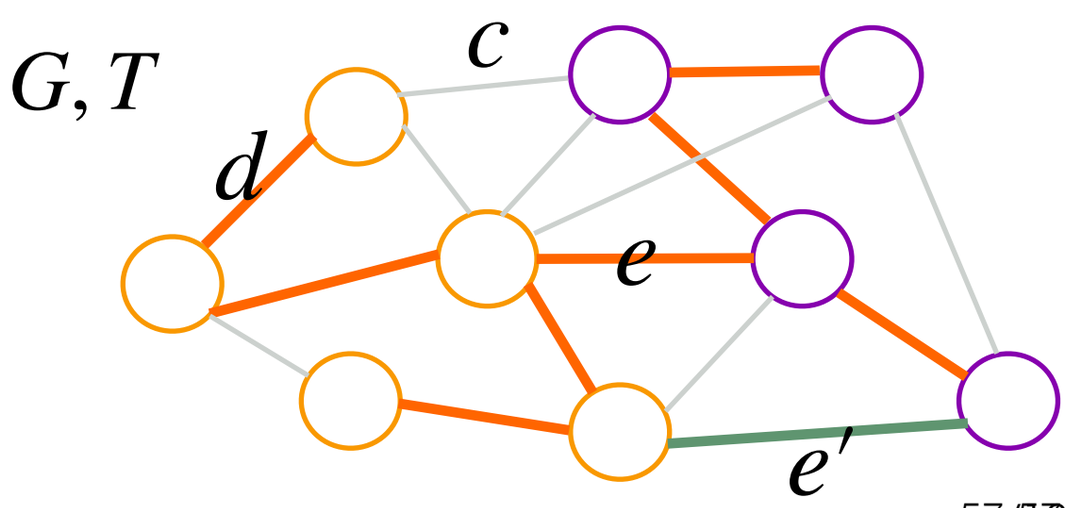
(1) $d \in C_0$ の時,

$\hat{T} = T \cup e' \backslash d$ とすると，$\hat{T}$ は全域木かつ $w(\hat{T}) < z^\star$ となり矛盾

(1) $d \in C_1$ の時,

$\hat{T} = T \cup c \backslash d$ とすると，同様に矛盾

よって，定理 2 が示された.



# アルゴリズムの構築（定理 3）

- 定理1, 2 より，以下の定理が導ける

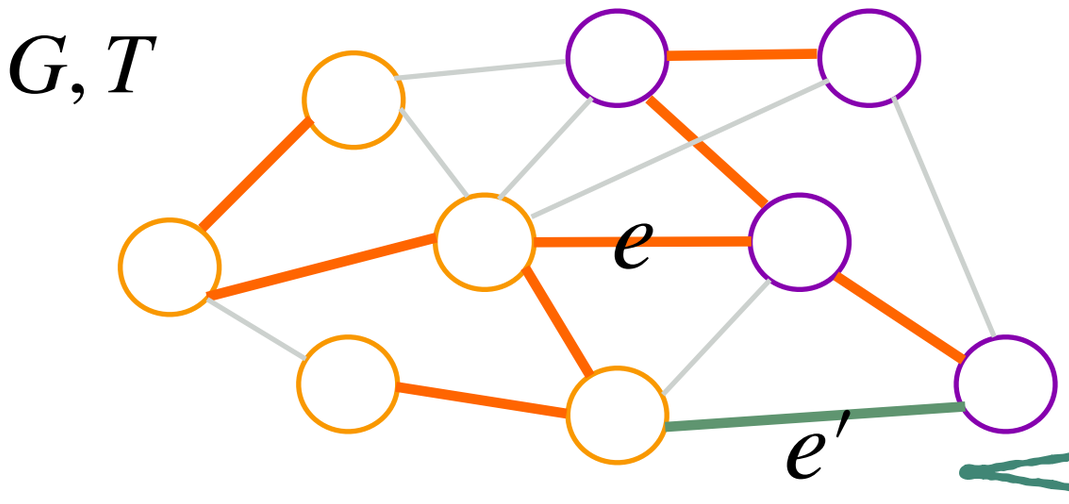
[定理 3]

$T$ を最小全域木，$e \in T$，とする.

もし $S(e)$ が空でない場合，その要素を $e'$ として

$T \cup e' \backslash e$ は最小全域木となる.

$S(e)$ が空な場合，$G^{(e)}$ に最小全域木は存在しない.




もし $w(e') = w(e)$ なら，

$T \cup e \backslash e'$ が最小全域木となる

Second minimum in $Cut(e)$

# アルゴリズムの概要

- 以下のアルゴリズムに対して $All\_MST1(\emptyset, \emptyset, T)$ を実行すると, 最小全域木を全列挙できる.
  - $F = \{e^1, \ldots, e^k\}, T = F \cup \{e^{k+1}, \ldots, e^{n-1}\}$ とする

ALGORITHM $All\_MST1(F, R, T)$

// $(F, R)$-許容な最小全域木を全て列挙する

Step 1: for $i \in [k+1, n-1]$ do

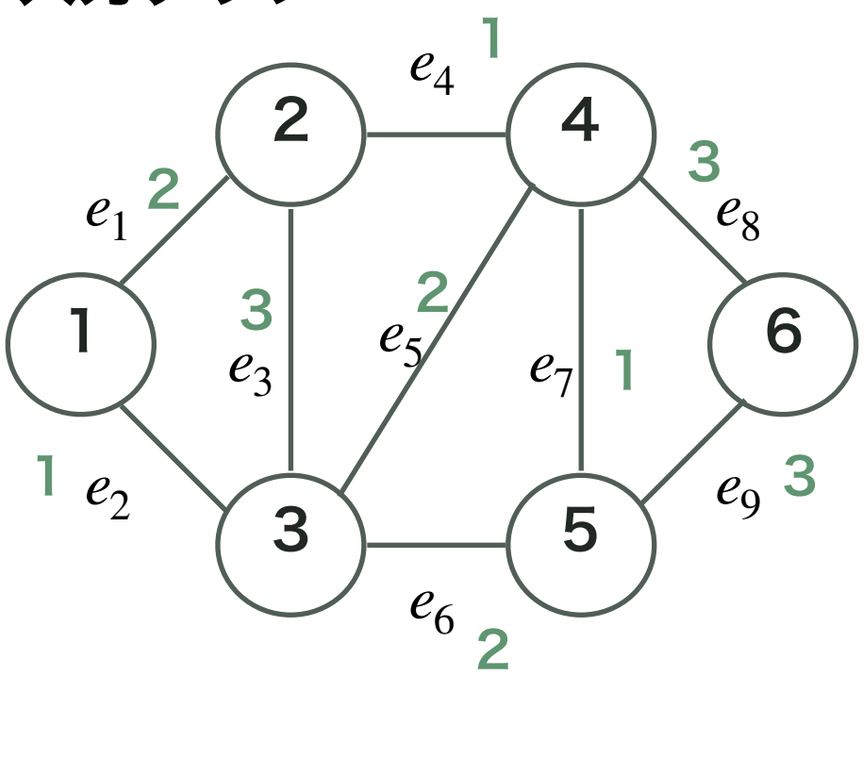
- 代替辺 $\tilde{e}^i \in S(e^i)$ を見つける.

Step 2: for $i \in [k+1, n-1]$, if $\tilde{e}^i$ exists do

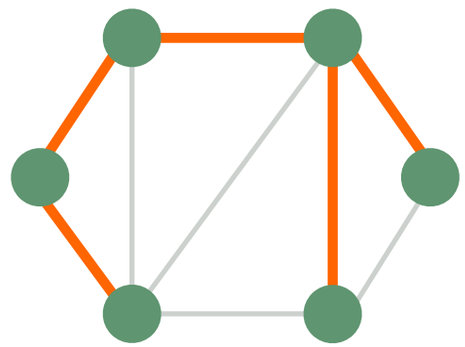
- $T_i \leftarrow T \cup \tilde{e}^i \backslash e^i$, $T_i$ を出力
- $F_i \leftarrow F \cup \{e^{k+1}, \ldots, e^{i-1}\}, R_i \leftarrow R \cup \{e^i\}$
- $All\_MST1(F_i, R_i, T_i)$ を再帰呼び出し

# アルゴリズムの実行例

入力グラフ

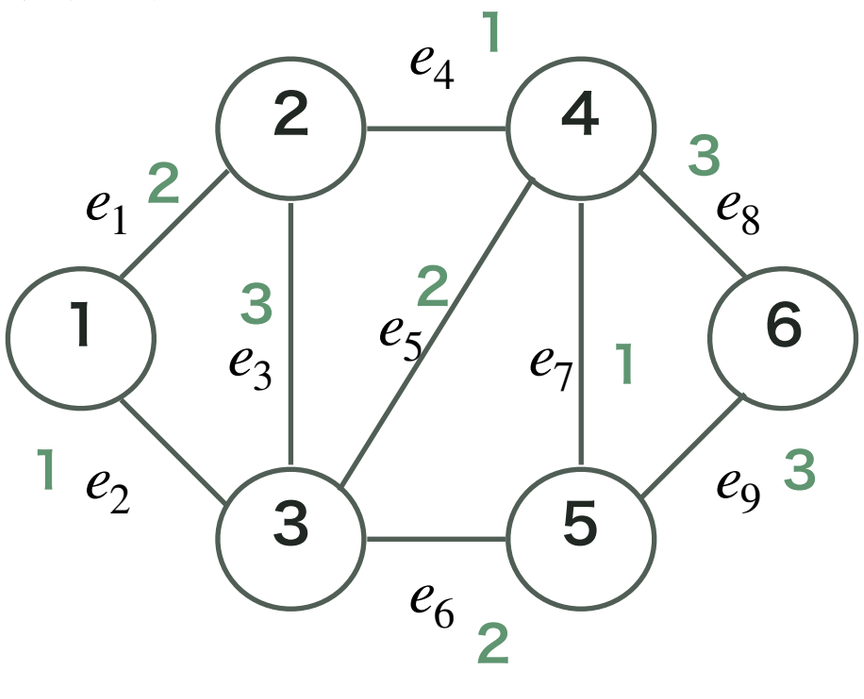



$P(\emptyset, \emptyset)$

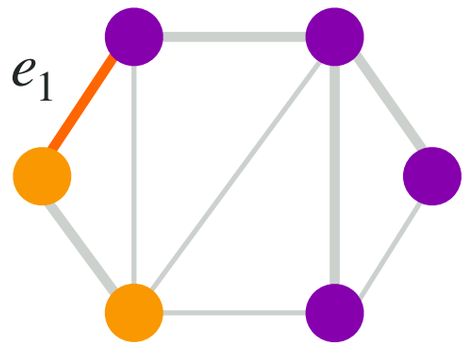


# アルゴリズムの実行例

入力グラフ



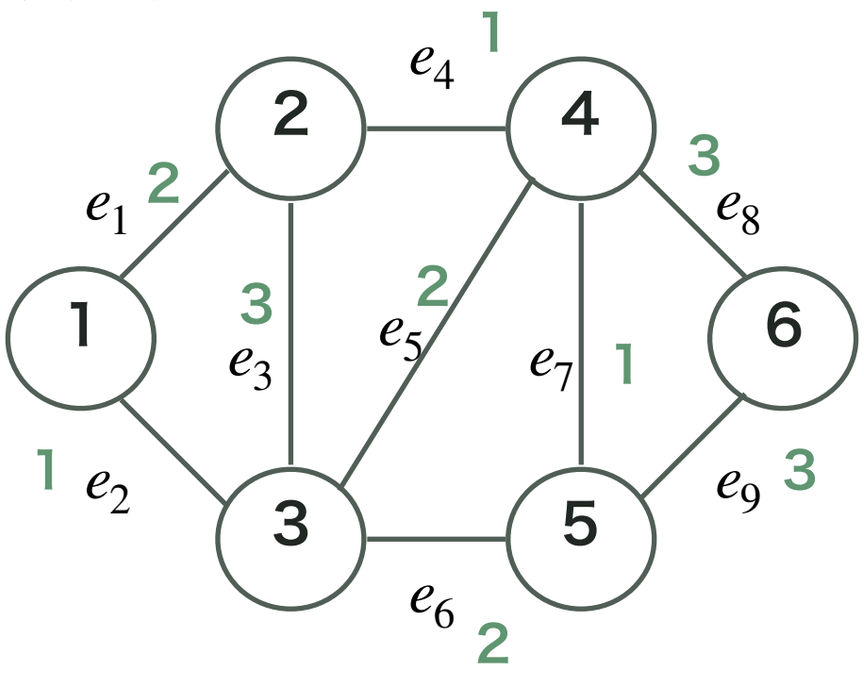

$P(\emptyset, \emptyset)$



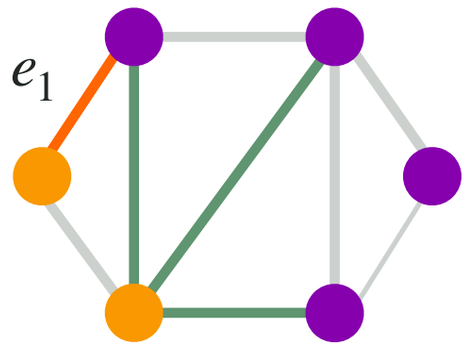

$e_1$ の代替辺を探す

# アルゴリズムの実行例

入力グラフ



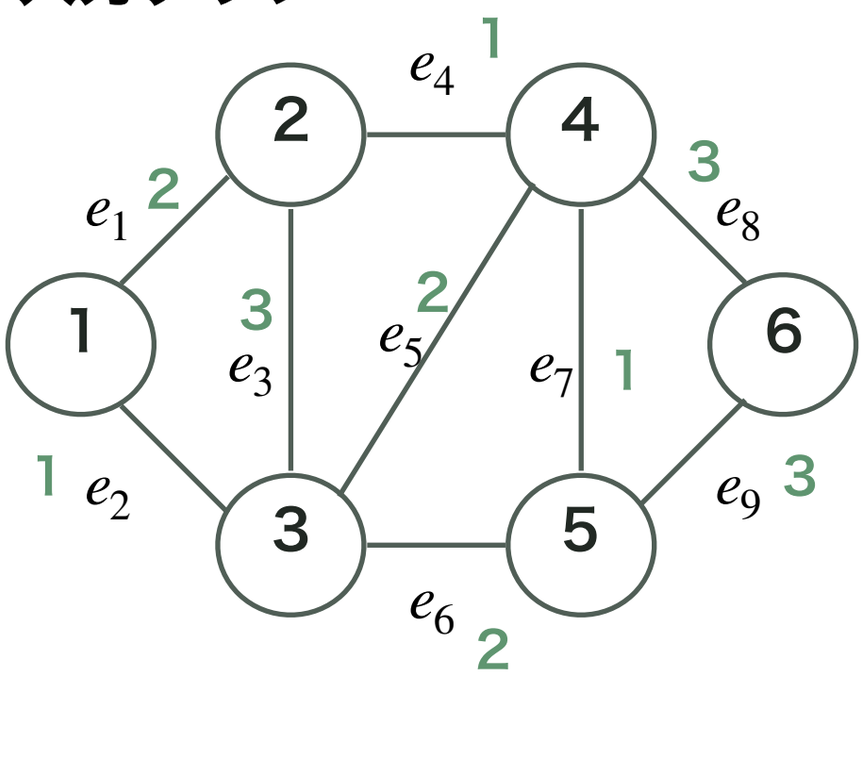

$P(\emptyset, \emptyset)$



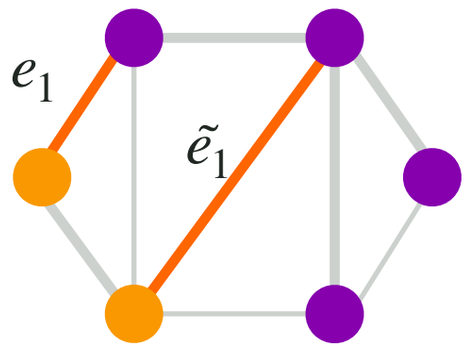

$e_1$ の代替辺を探す

# アルゴリズムの実行例

入力グラフ



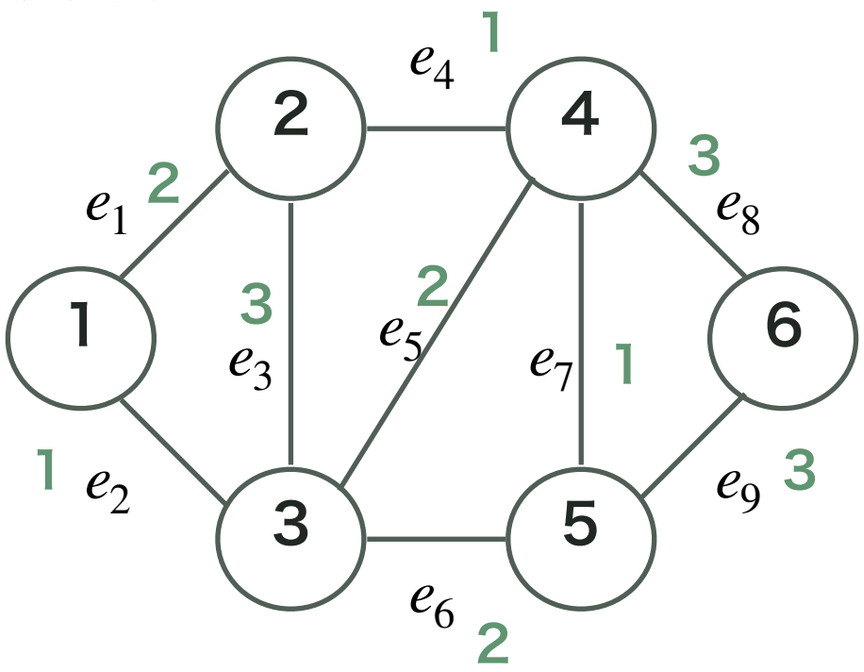

$P(\varnothing, \varnothing)$



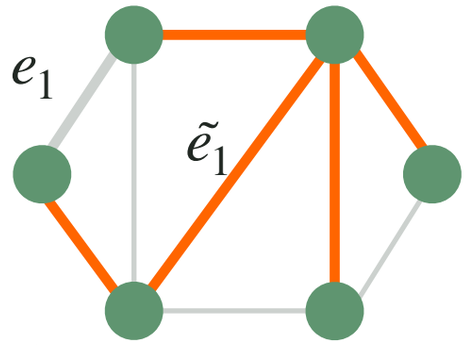

$e_1$ の代替辺が見つかる

# アルゴリズムの実行例

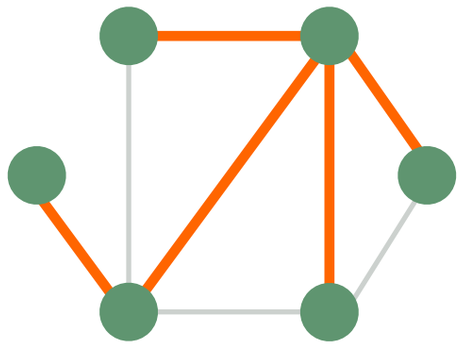
入力グラフ




$P(\varnothing, \varnothing)$



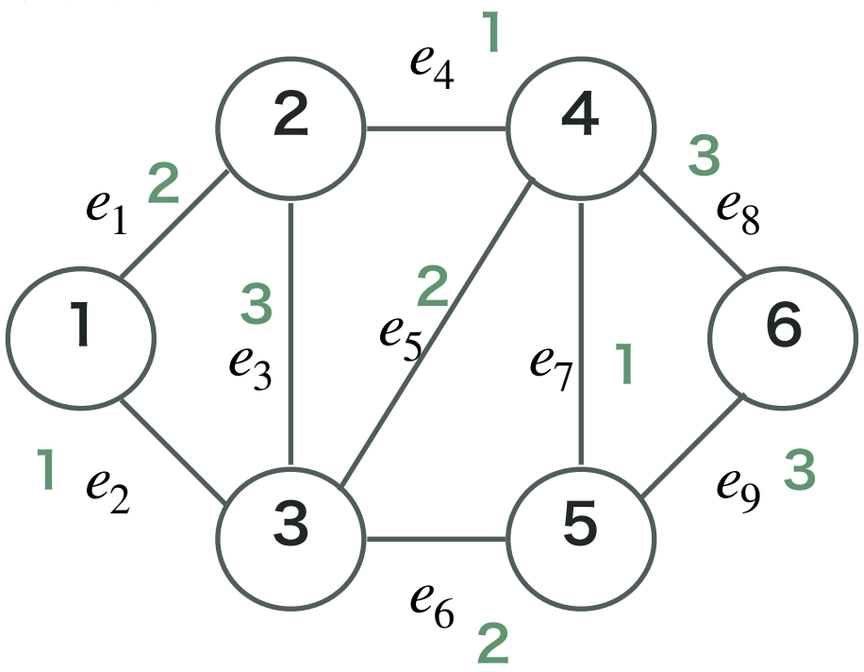

$P(\varnothing, \{e_1\})$



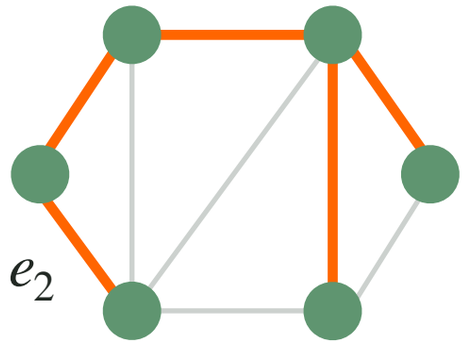
$T'$ を出力して部分問題へ

# アルゴリズムの実行例

入力グラフ

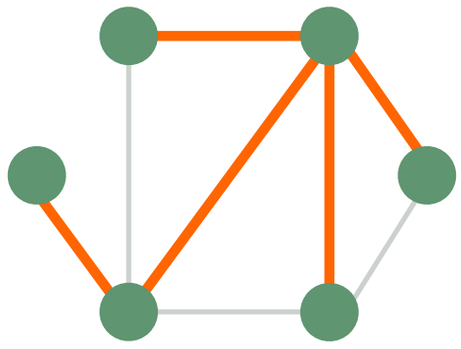



$P(\emptyset, \emptyset)$



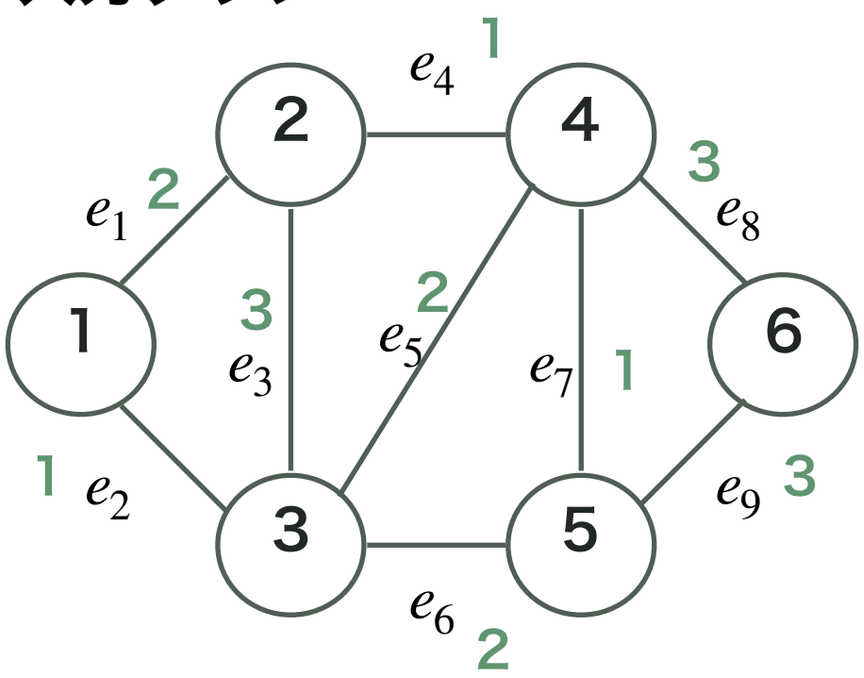

$e_2$ の代替辺を探すが. . .

$P(\emptyset, \{e_1\})$

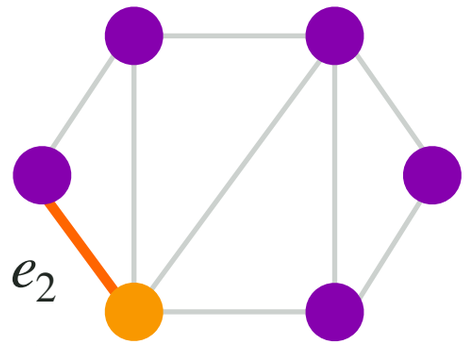


# アルゴリズムの実行例

入力グラフ

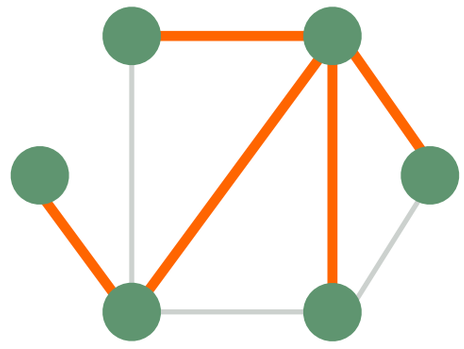



$P(\emptyset, \emptyset)$



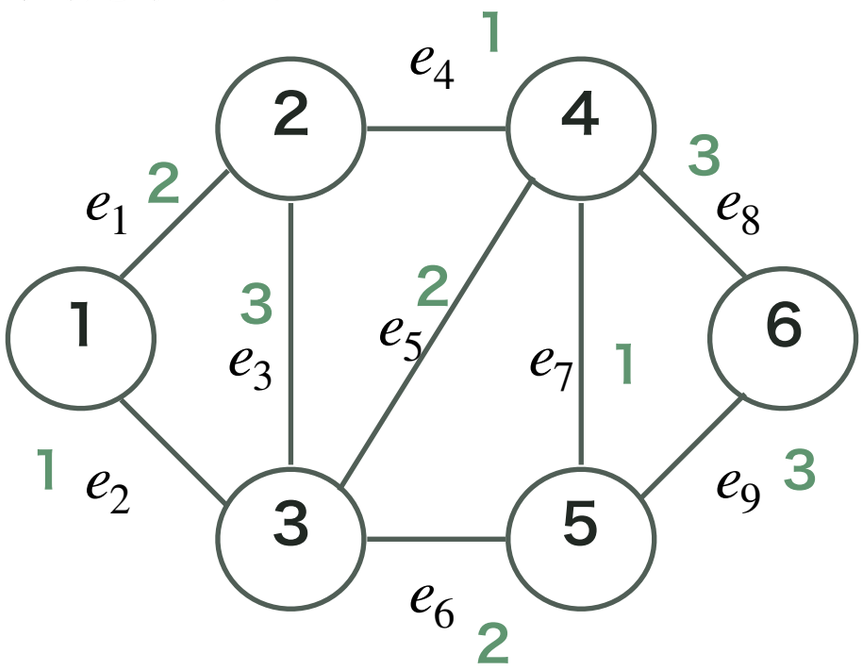

$e_2$ の代替辺を探すが. . .

$P(\emptyset, \{e_1\})$

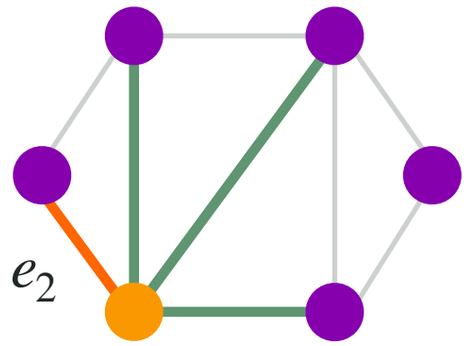


# アルゴリズムの実行例

入力グラフ

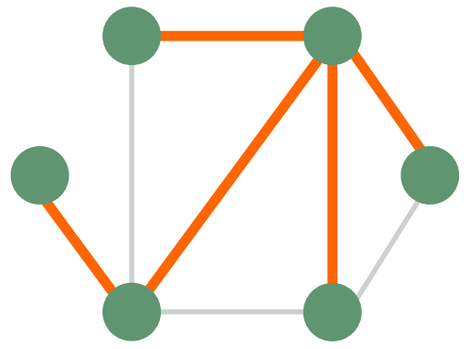



$P(\emptyset, \emptyset)$



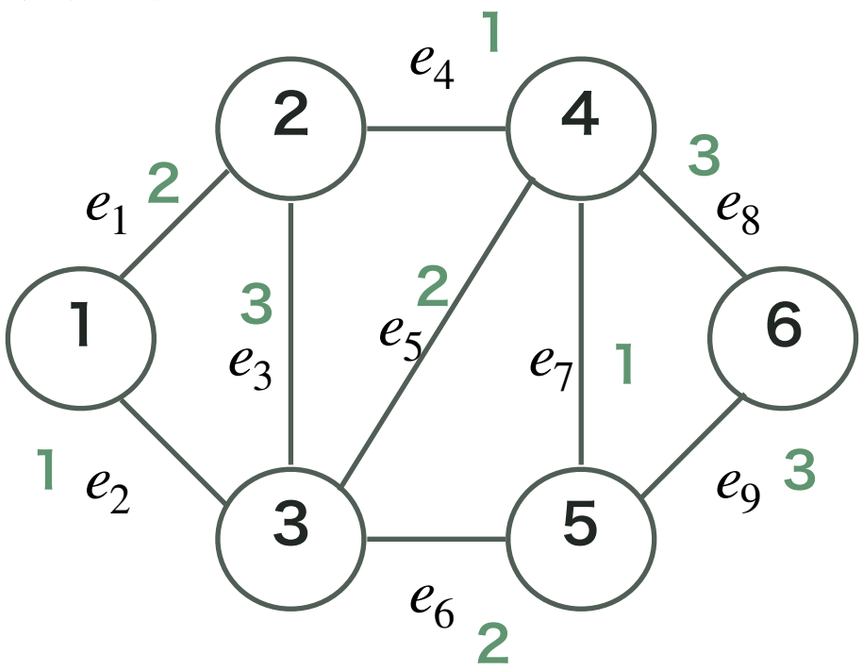

$e_2$ の代替辺を探すが. . .

$P(\emptyset, \{e_1\})$

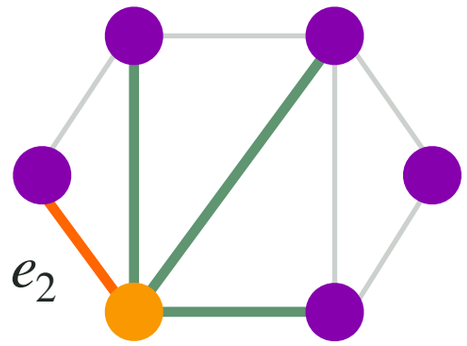


# アルゴリズムの実行例

入力グラフ

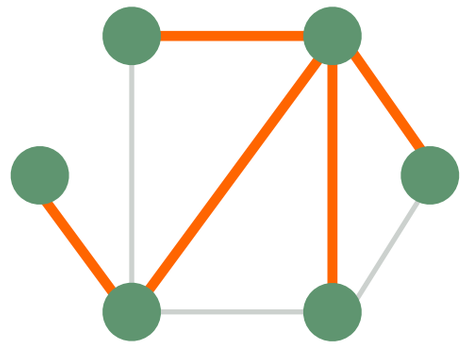



$P(\emptyset, \emptyset)$



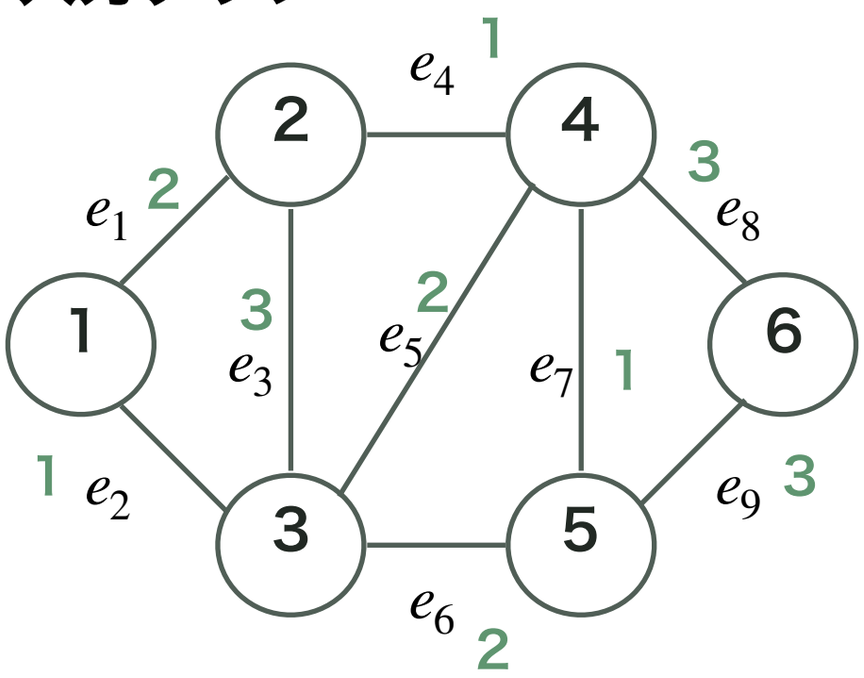

$e_2$ の代替辺は存在しない

$P(\emptyset, \{e_1\})$

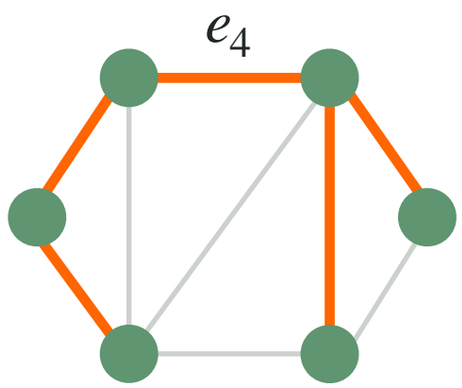


# アルゴリズムの実行例

入力グラフ

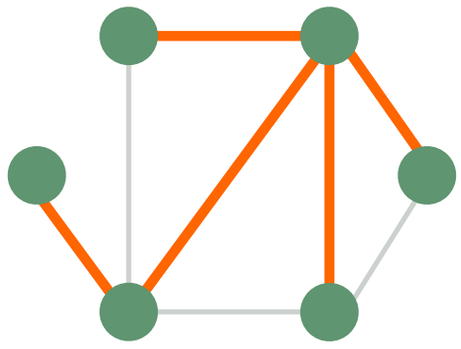



$P(\emptyset, \emptyset)$



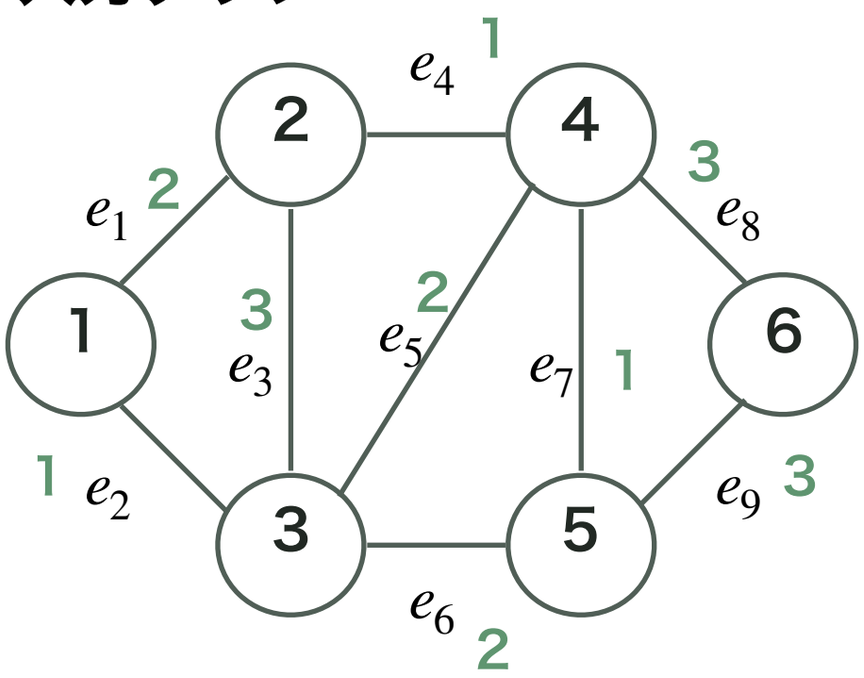

$e_4$ の代替辺を探す

$P(\emptyset, \{e_1\})$

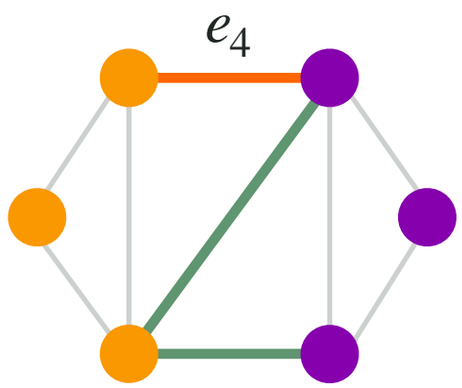


# アルゴリズムの実行例

入力グラフ

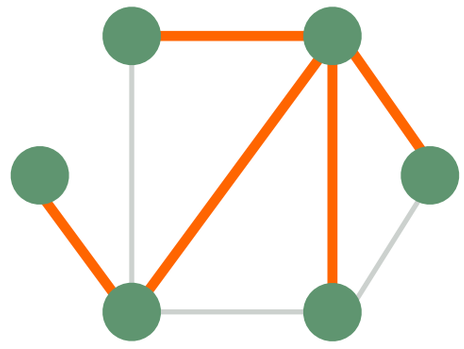



$P(\varnothing, \varnothing)$



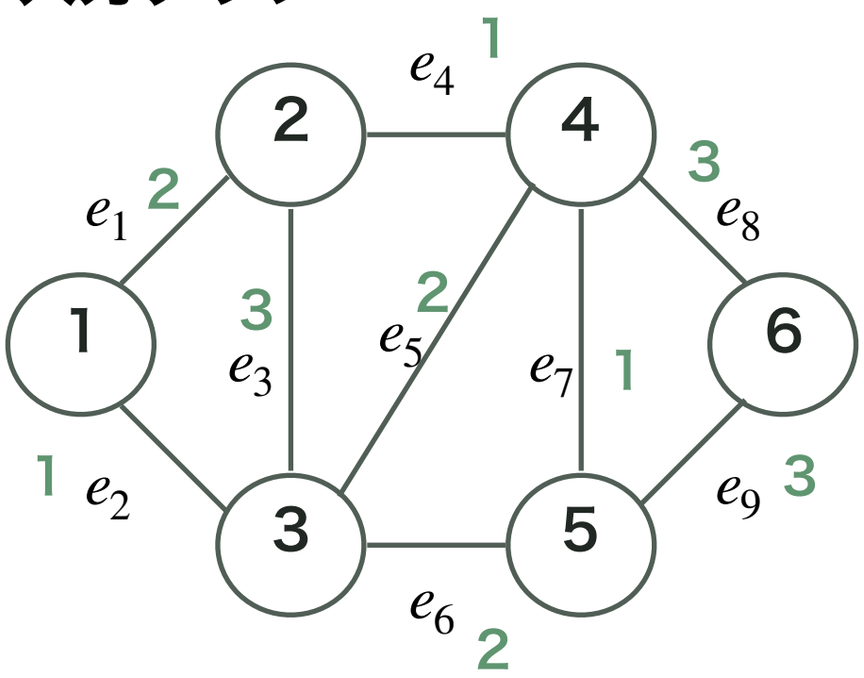

$e_4$ の代替辺を探す

$P(\varnothing, \{e_1\})$

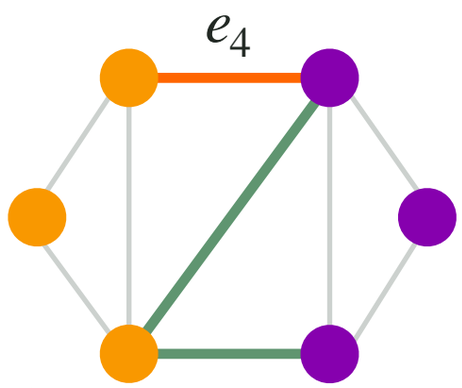


# アルゴリズムの実行例

入力グラフ

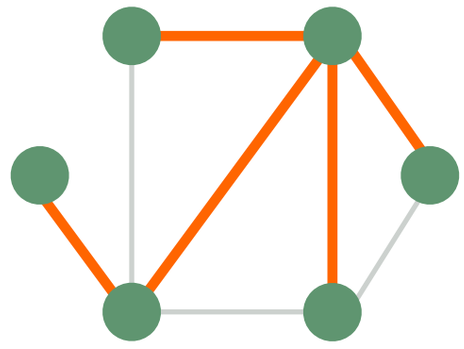



$P(\varnothing, \varnothing)$



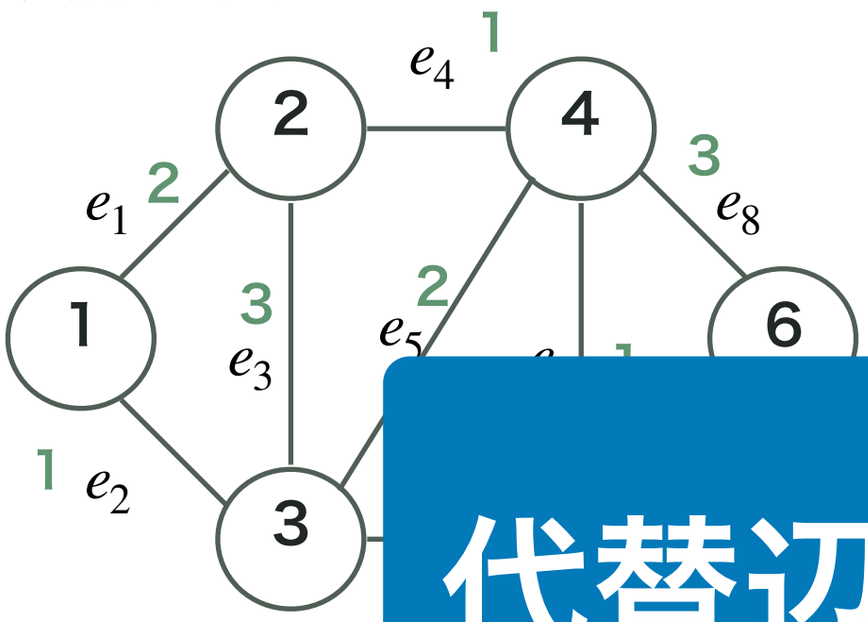

$e_4$ の代替辺は存在しない

$P(\varnothing, \{e_1\})$

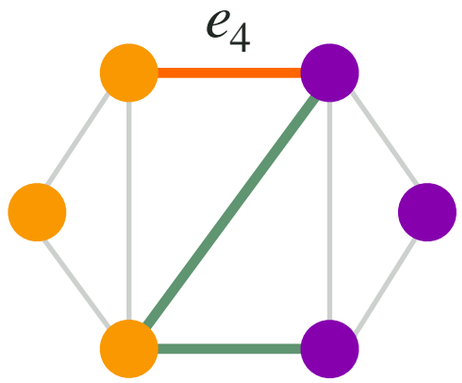


# アルゴリズムの実行例

入力グラフ




$P(\emptyset, \emptyset)$

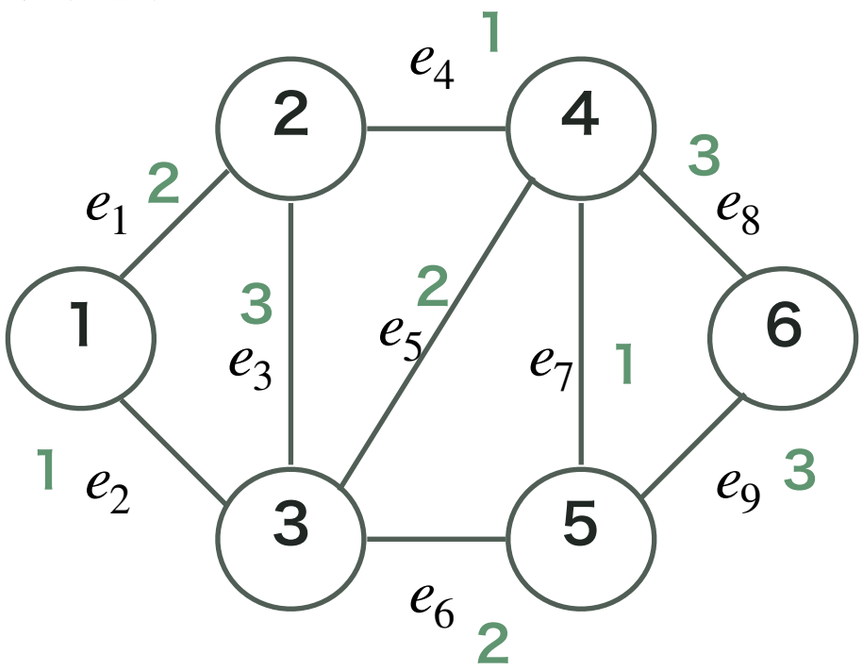



は存在しない

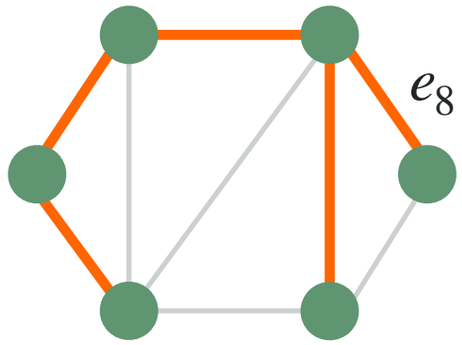
代替辺が存在しない辺は省略

# アルゴリズムの実行例

入力グラフ

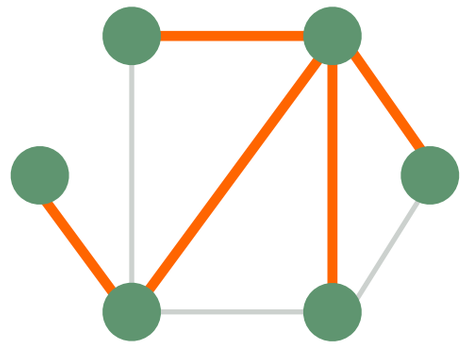



$P(\varnothing, \varnothing)$



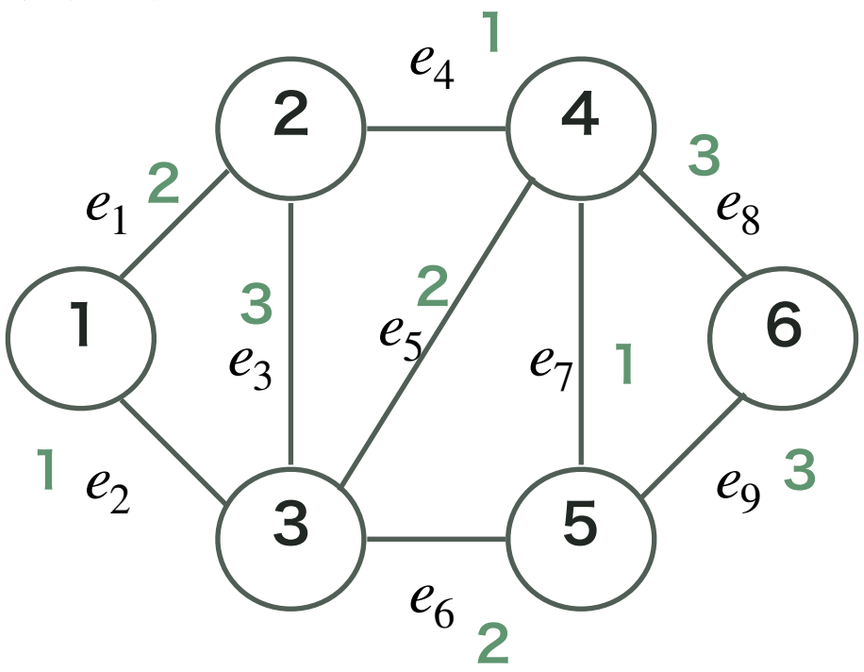

$e_8$ の代替辺を探す

$P(\varnothing, \{e_1\})$

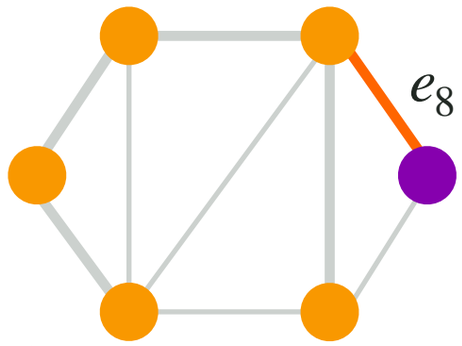


# アルゴリズムの実行例

入力グラフ

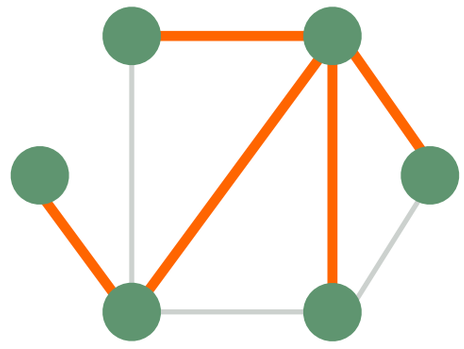



$P(\varnothing, \varnothing)$



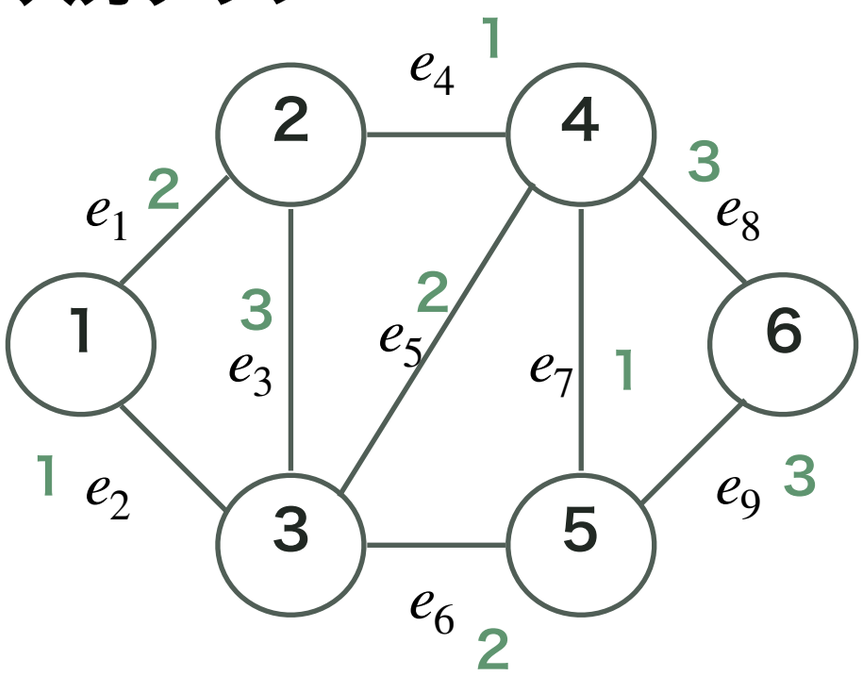

$e_8$ の代替辺を探す

$P(\varnothing, \{e_1\})$

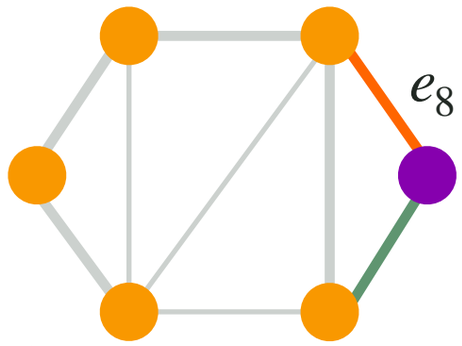


# アルゴリズムの実行例

入力グラフ

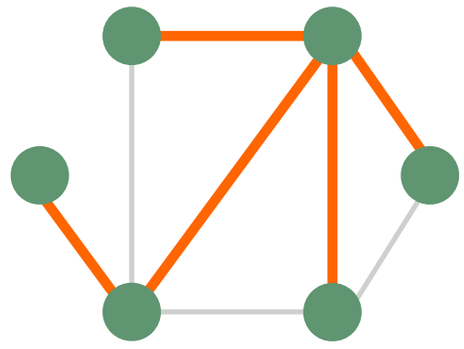



$P(\varnothing, \varnothing)$



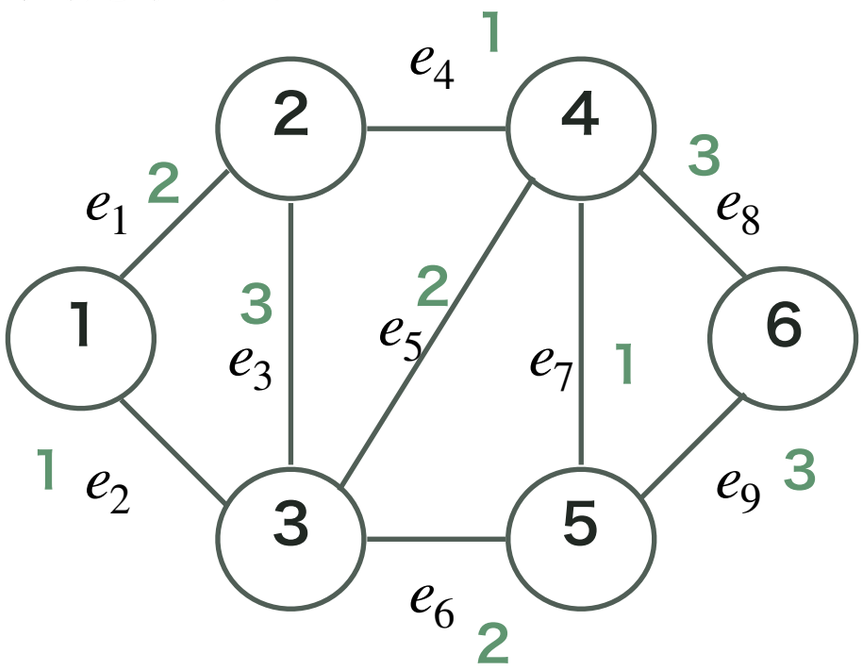

$e_8$ の代替辺を探す

$P(\varnothing, \{e_1\})$

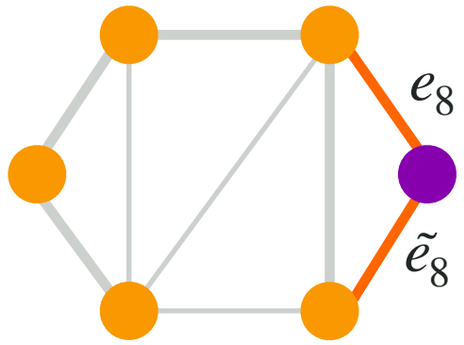


# アルゴリズムの実行例

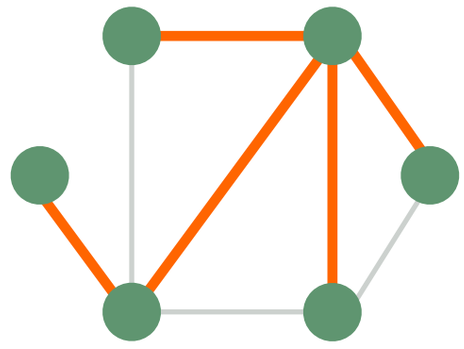
入力グラフ




$P(\emptyset, \emptyset)$



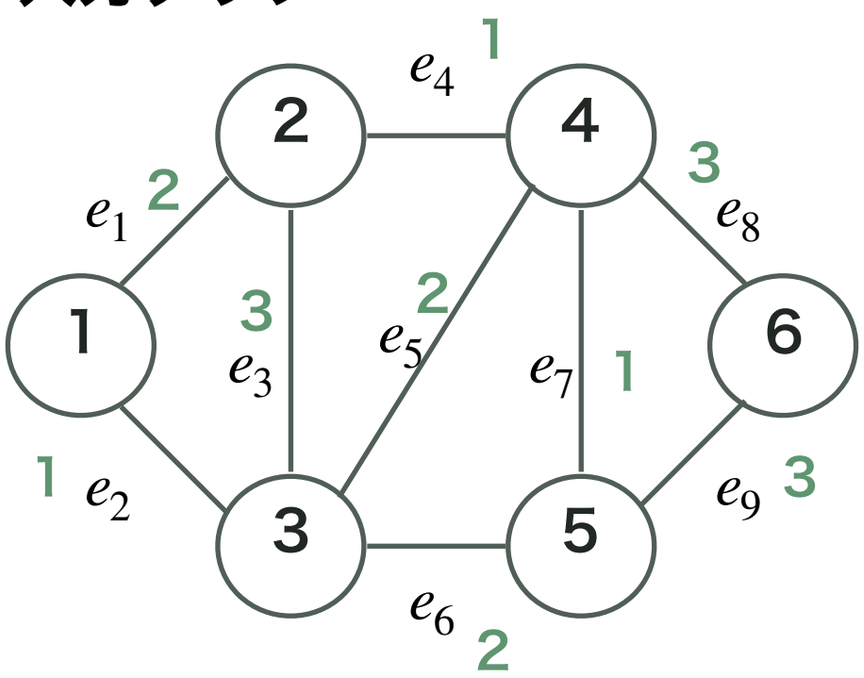

$P(\emptyset, \{e_1\})$



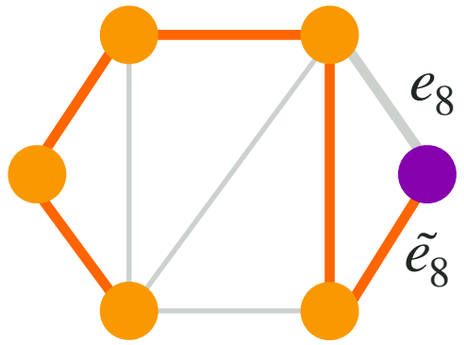
$e_8$ の代替辺が見つかる

# アルゴリズムの実行例

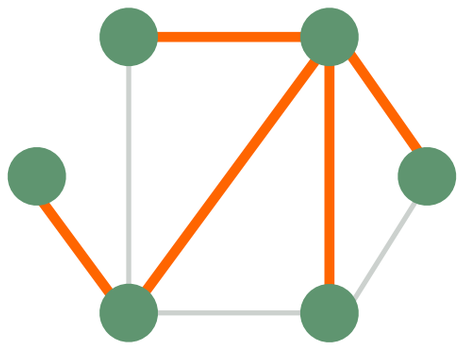
入力グラフ



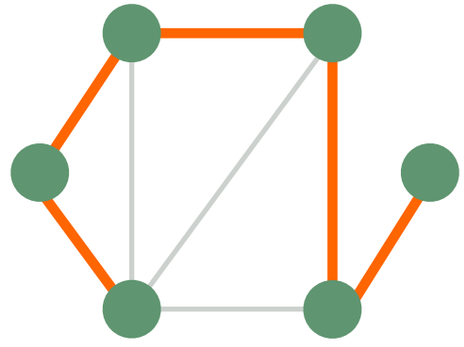

$P(\emptyset, \emptyset)$




$P(\emptyset, \{e_1\})$

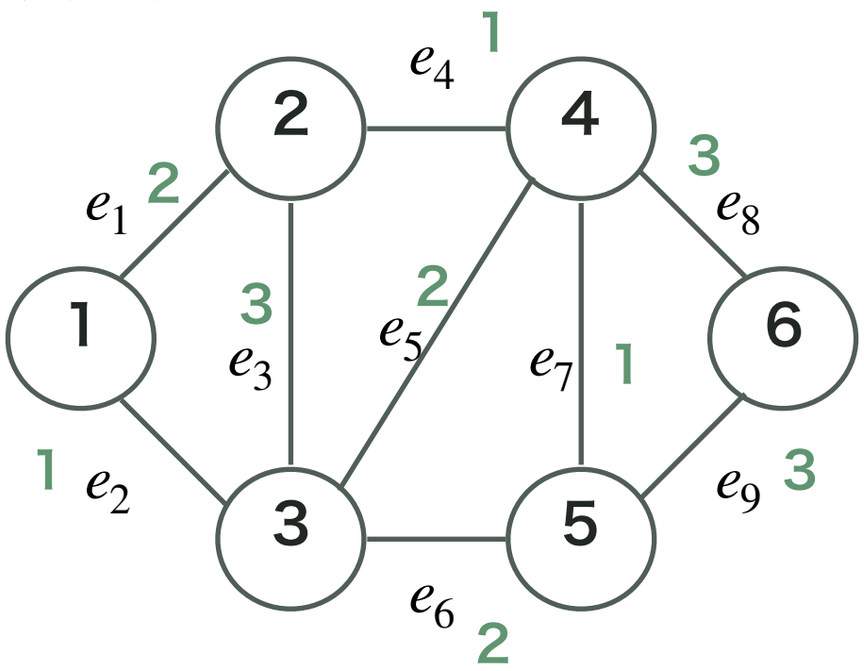


$P(\{e_1, e_2, e_4, e_7\}, \{e_8\})$

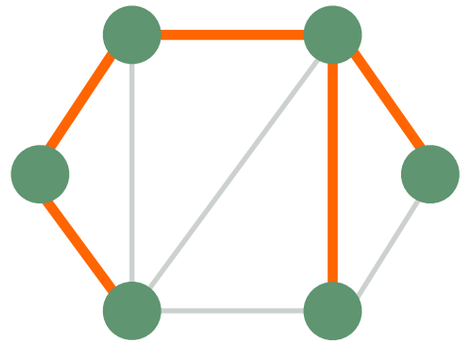


# アルゴリズムの実行例

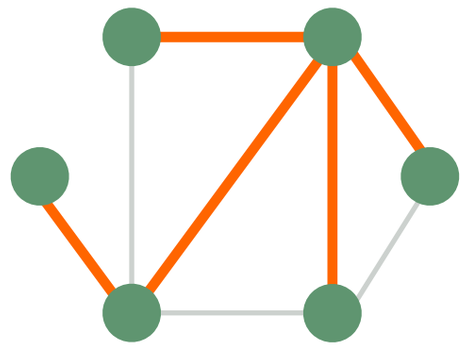
入力グラフ



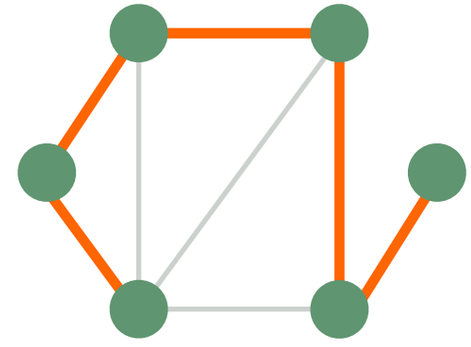

$P(\emptyset, \emptyset)$



$P(\emptyset, \{e_1\})$

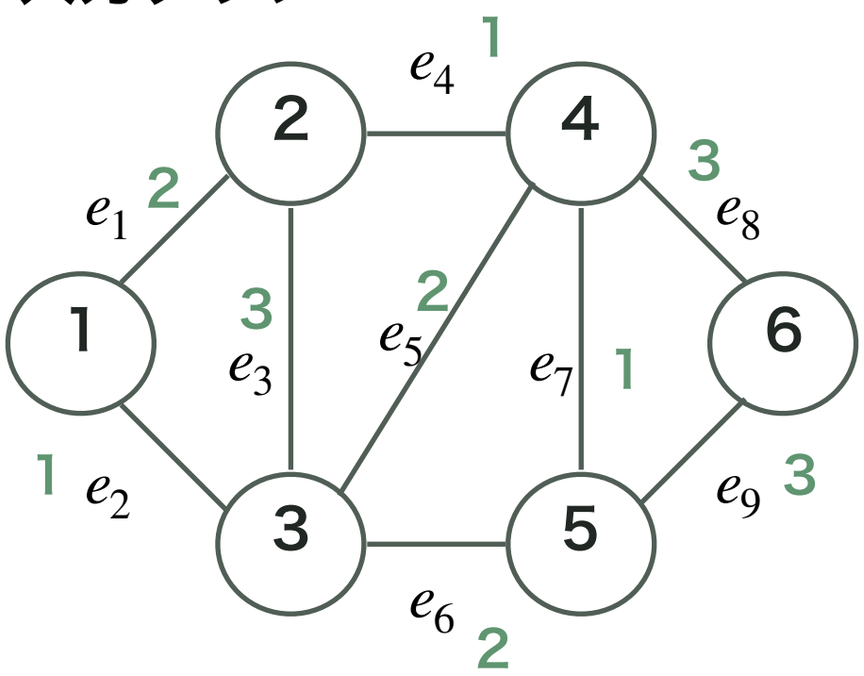


$P(\{e_1, e_2, e_4, e_7\}, \{e_8\})$

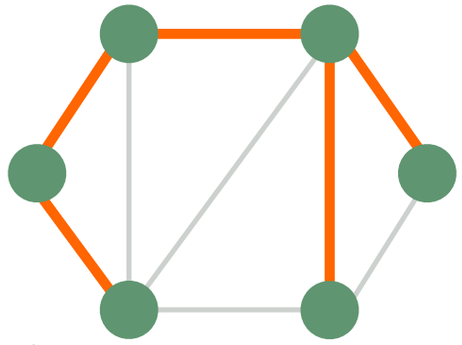


# アルゴリズムの実行例

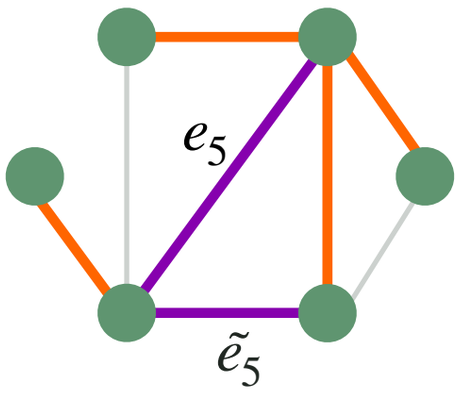
入力グラフ



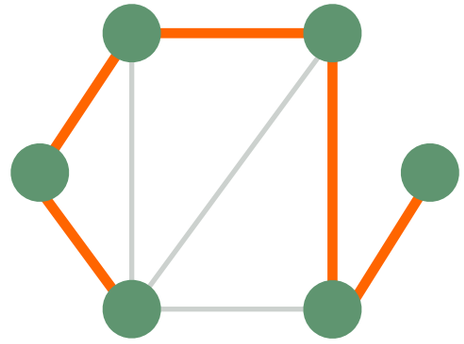

$P(\emptyset, \emptyset)$



$P(\emptyset, \{e_1\})$

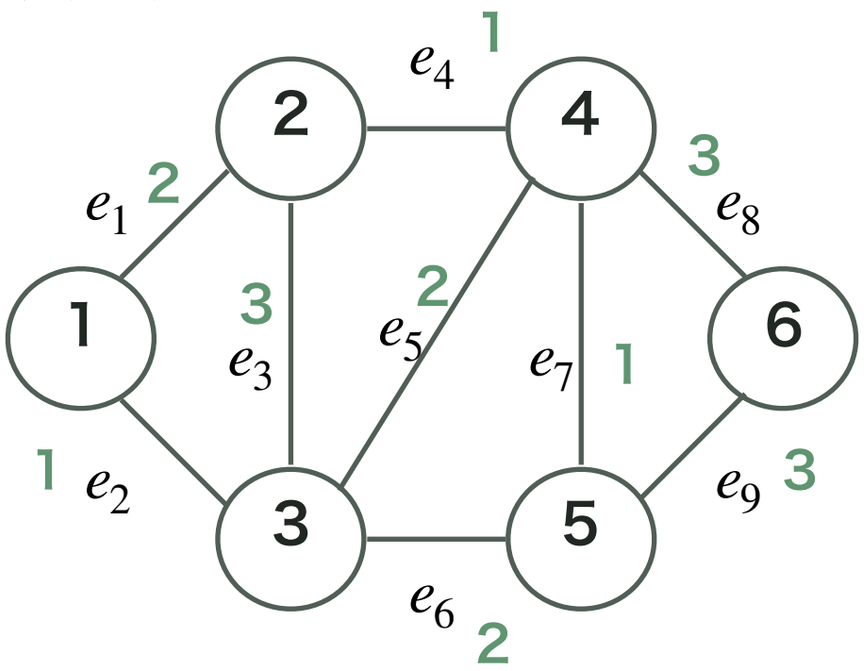



$P(\{e_1, e_2, e_4, e_7\}, \{e_8\})$

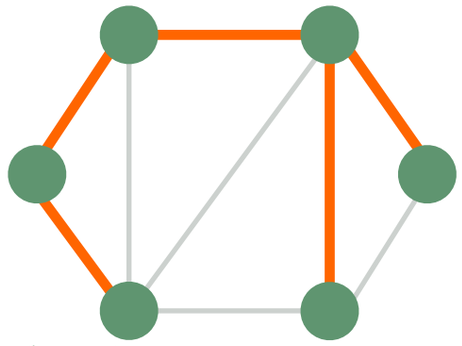


# アルゴリズムの実行例

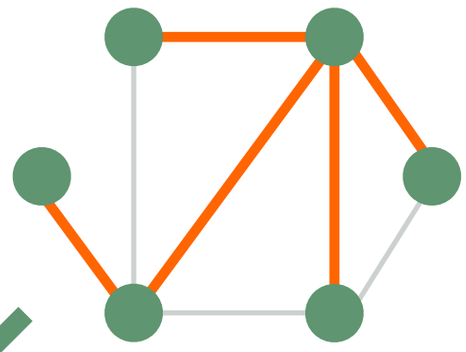
入力グラフ



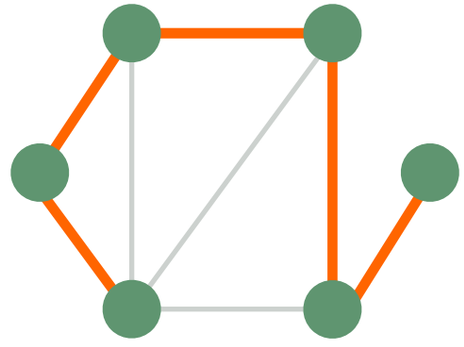

$P(\emptyset, \emptyset)$



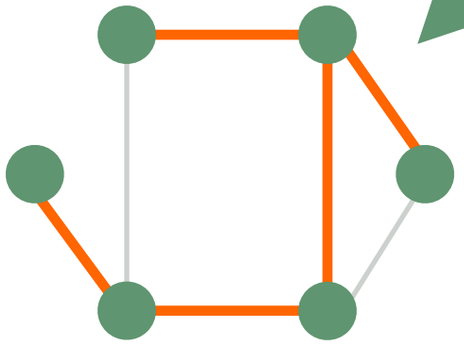
$P(\emptyset, \{e_1\})$



$P(\{e_1, e_2, e_4, e_7\}, \{e_8\})$

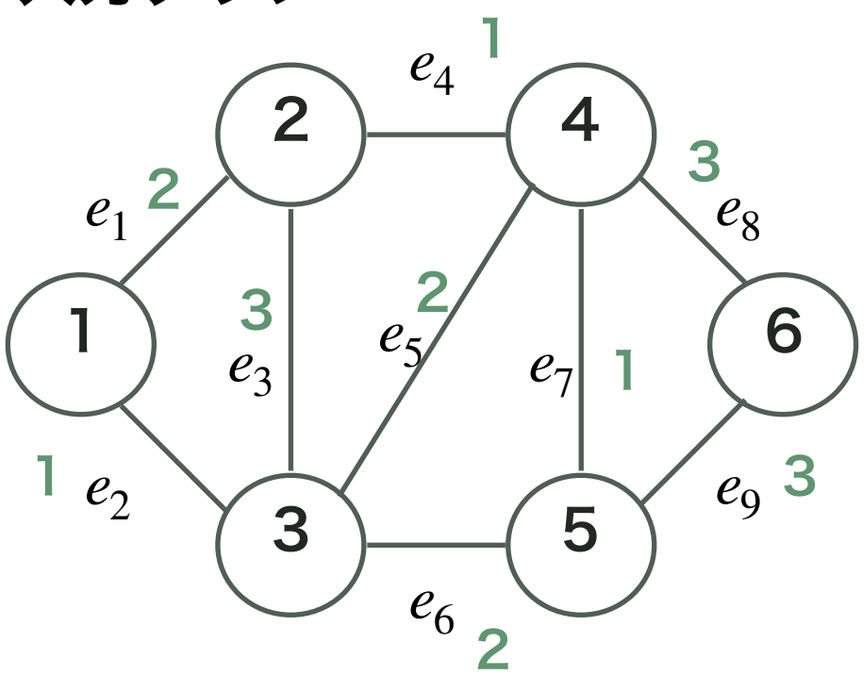


$P(\{e_2\}, \{e_1 . e_5\})$

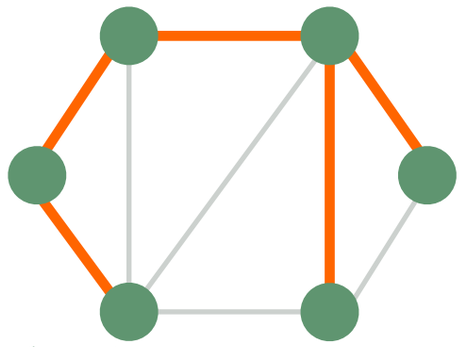


# アルゴリズムの実行例

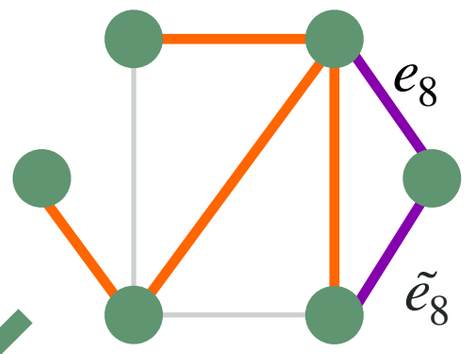
入力グラフ



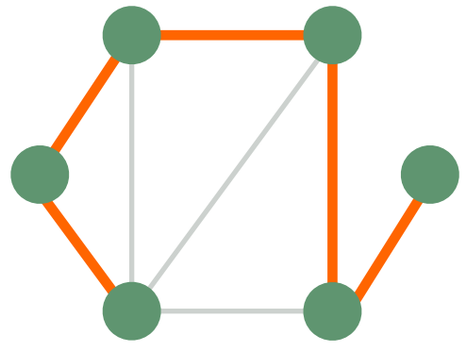

$P(\emptyset, \emptyset)$



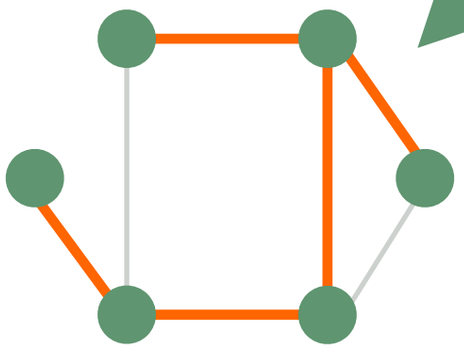
$P(\emptyset, \{e_1\})$




$P(\{e_1, e_2, e_4, e_7\}, \{e_8\})$

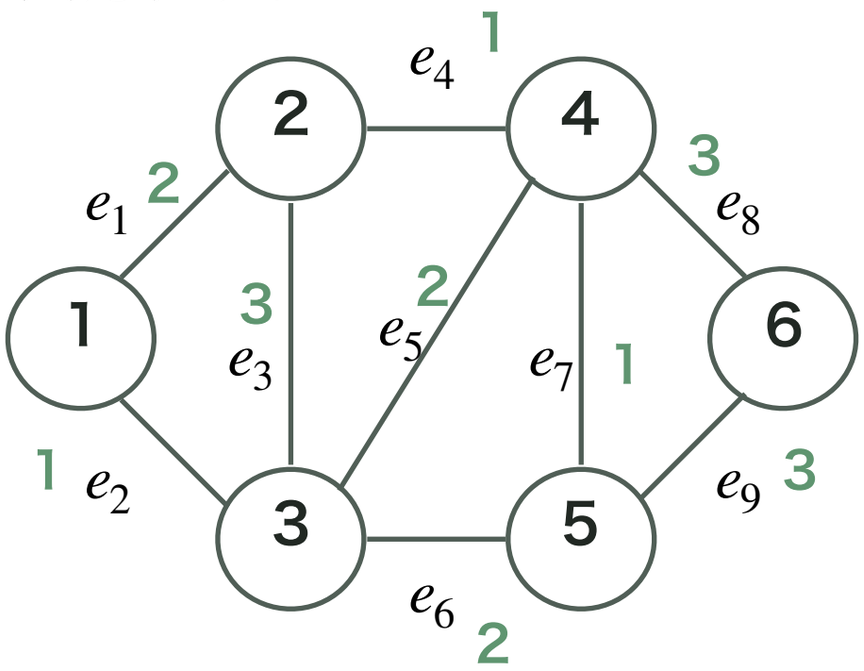


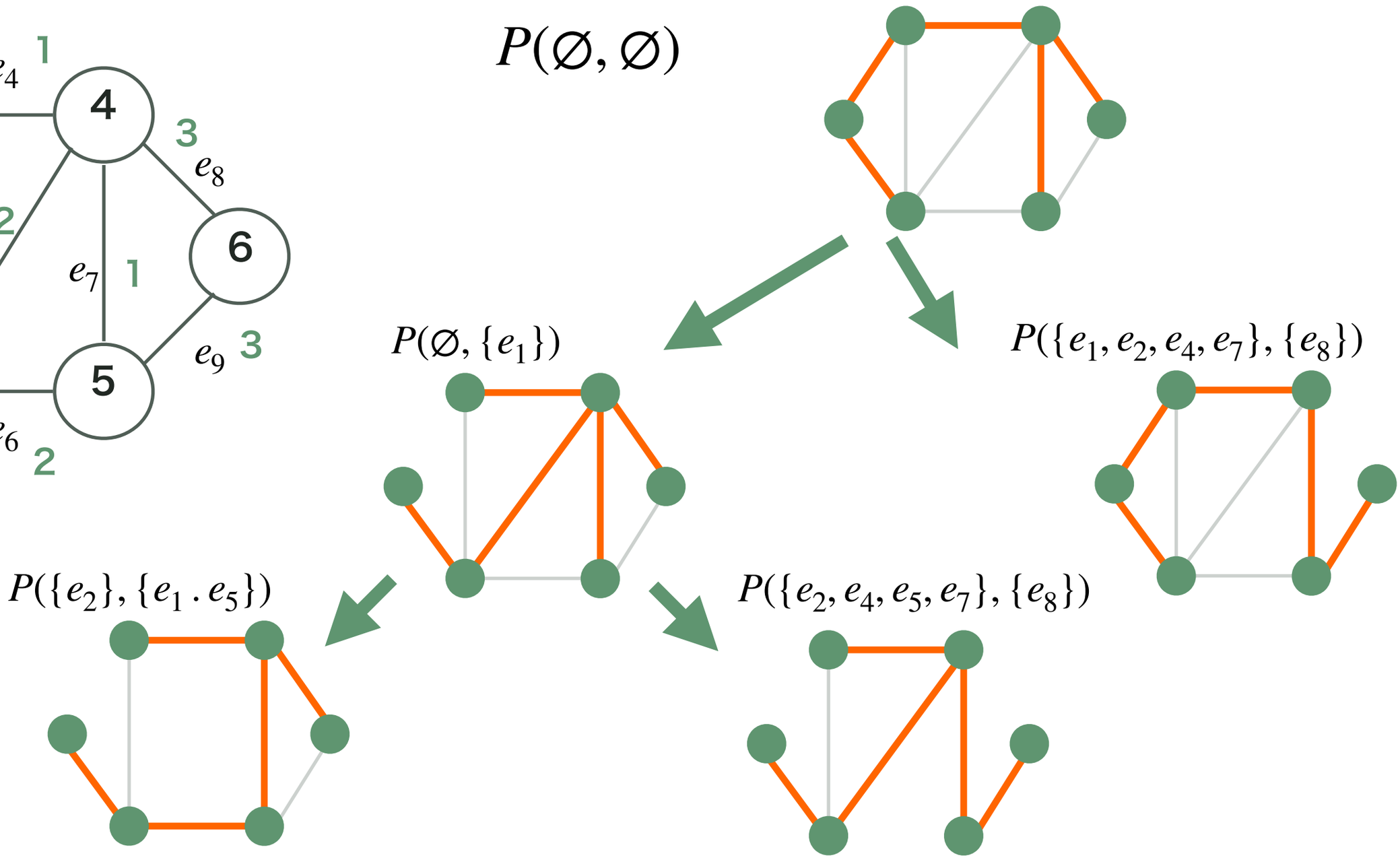
$P(\{e_2\}, \{e_1 . e_5\})$



# アルゴリズムの実行例

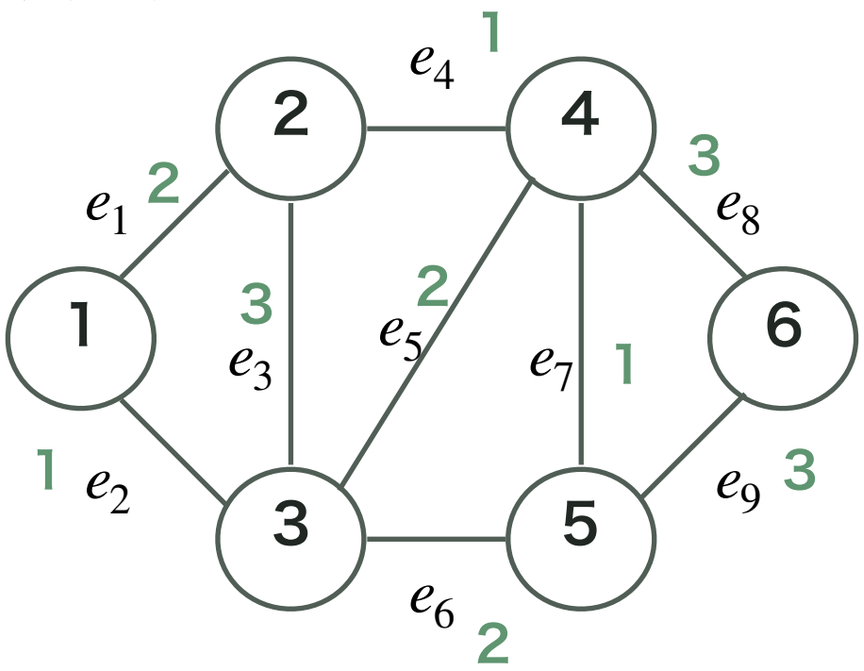
入力グラフ



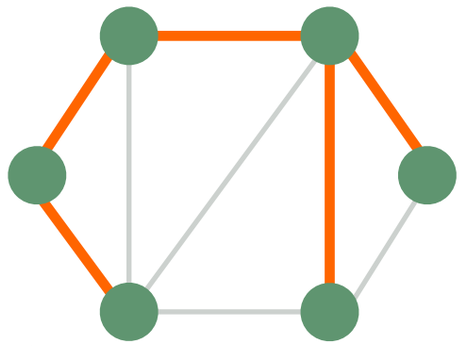



# アルゴリズムの実行例

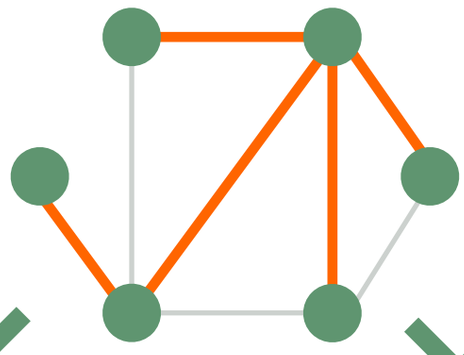
入力グラフ



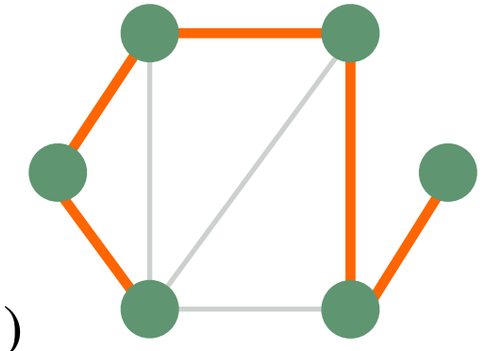

$P(\emptyset, \emptyset)$



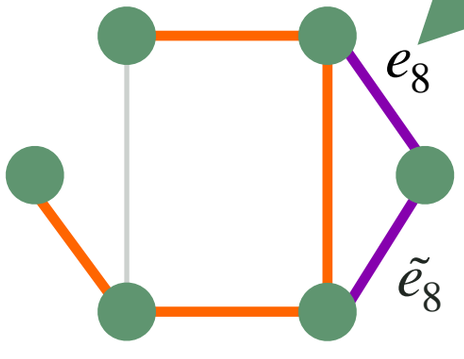
$P(\emptyset, \{e_1\})$



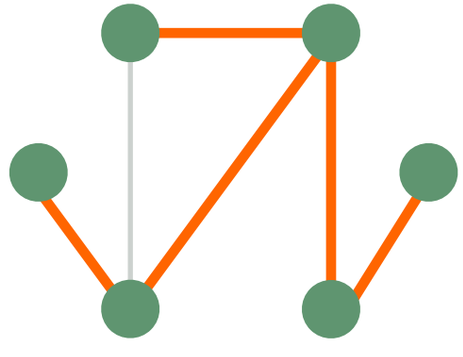
$P(\{e_1, e_2, e_4, e_7\}, \{e_8\})$



$P(\{e_2\}, \{e_1 . e_5\})$

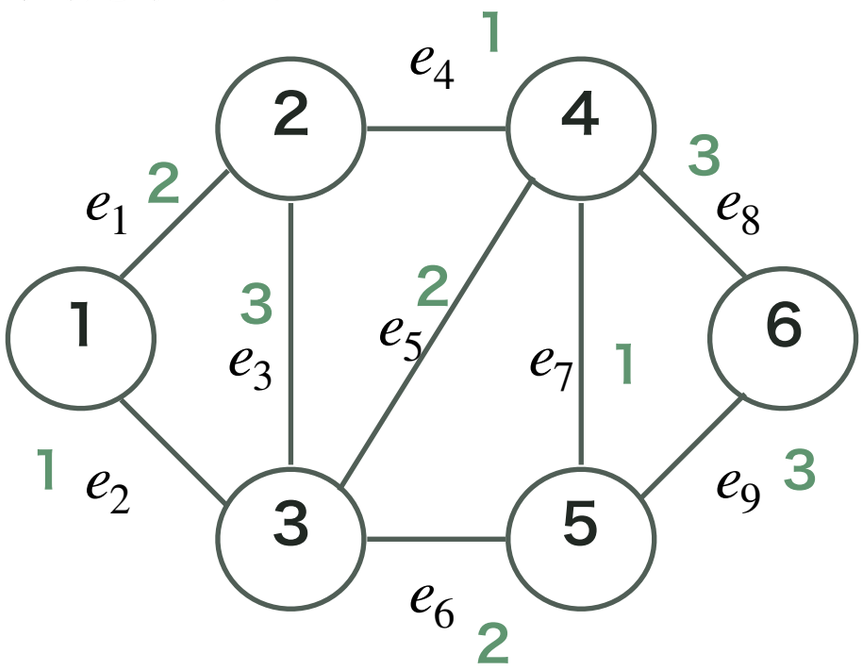



$P(\{e_2, e_4, e_5, e_7\}, \{e_8\})$

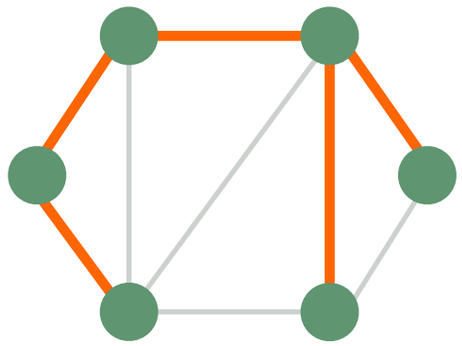


# アルゴリズムの実行例

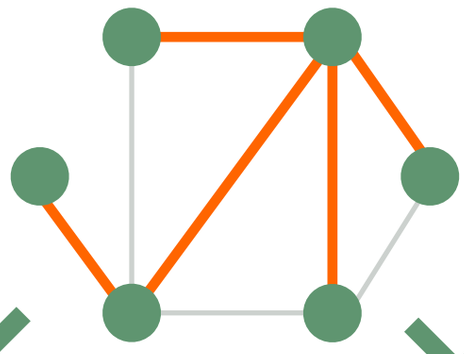
入力グラフ



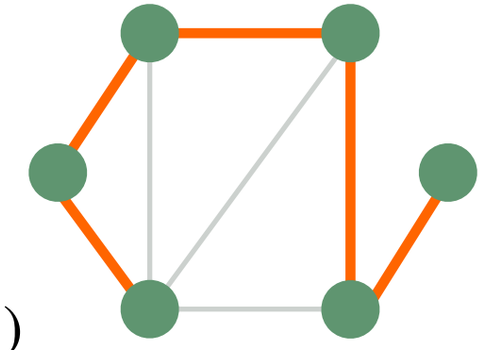

$P(\emptyset, \emptyset)$



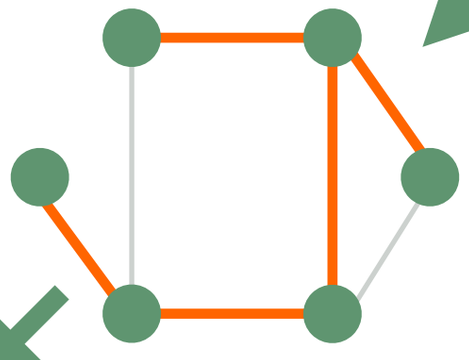
$P(\emptyset, \{e_1\})$



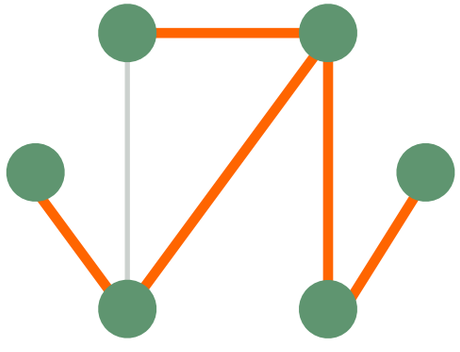
$P(\{e_1, e_2, e_4, e_7\}, \{e_8\})$

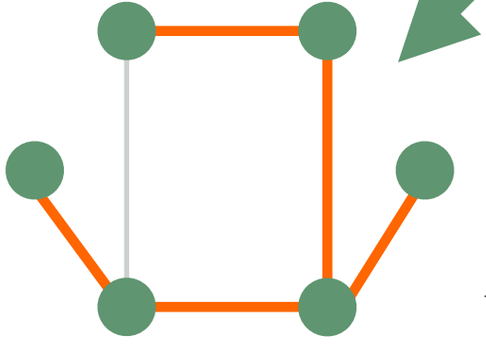


$P(\{e_2\}, \{e_1 . e_5\})$



$P(\{e_2, e_4, e_5, e_7\}, \{e_8\})$





$P(\{e_2, e_4, e_6, e_7\}, \{e_1 . e_5 . e_8\})$

# 時間計算量解析

- $All\_MST_1$ の時間計算量は $O(Nnm)$

ALGORITHM $All\_MST_1(F, R, T)$

// $(F, R)$-許容な最小全域木を全て列挙する

Step 1: for $i \in [k+1, n-1]$ do

- 代替辺 $\tilde{e}^i \in S(e^i)$ を見つける.

Step 2: for $i \in [k+1, n-1]$, if $\tilde{e}^i$ exists do

- $T_i \leftarrow T \cup \tilde{e}^i \backslash e^i$, $T_i$ を出力
- $F_i \leftarrow F \cup \{e^{k+1}, \ldots, e^{i-1}\}, R_i \leftarrow R \cup \{e^i\}$
- $All\_MST_1(F_i, R_i, T_i)$ を再帰呼び出し

# 時間計算量解析

- $All\_MST_1$ の時間計算量は $O(Nnm)$

ALGORITHM $All\_MST_1(F, R, T)$

// $(F, R)$-許容な最小全域木を全て列挙する

Step 1: for $i \in [k+1, n-1]$ do

- 代替辺 $\tilde{e}^i \in S(e^i)$ を見つける.

Step 2: for $i \in [k+1, n-1]$, if $\tilde{e}^i$ exists do

- $T_i \leftarrow T \cup \tilde{e}^i \backslash e^i$, $T_i$ を出力
- $F_i \leftarrow F \cup \{e^{k+1}, \ldots, e^{i-1}\}, R_i \leftarrow R \cup \{e^i\}$
- $All\_MST_1(F_i, R_i, T_i)$ を再帰呼び出し

**ここで $O(nm)$ かかる**

# 時間計算量解析

- $All\_MST_1$ の時間計算量は $O(Nnm)$

ALGORITHM $All\_MST_1(F, R, T)$

// $(F, R)$-許容な最小全域木を全て列挙する

Step 1: for $i \in [k+1, n-1]$ do

- 代替辺 $\tilde{e}^i \in S(e^i)$ を見つける.

Step 2: for $i \in [k+1, n-1]$, if $\tilde{e}^i$ exists do

- $T_i \leftarrow T \cup \tilde{e}^i \backslash e^i$, $T_i$ を出力
- $F_i \leftarrow F \cup \{e^{k+1}, \ldots, e^{i-1}\}, R_i \leftarrow R \cup \{e^i\}$
- $All\_MST_1(F_i, R_i, T_i)$ を再帰呼び出し

**ここで $O(nm)$ かかる**

**ここは $O(1)$**

# 時間計算量解析

- $All\_MST_1$ の時間計算量は $O(Nnm)$

ALGORITHM $All\_MST_1(F, R, T)$

// $(F, R)$-許容な最小全域木を全て列挙する

Step 1: for $i \in [k+1, n-1]$ do

- 代替辺 $\tilde{e}^i \in S(e^i)$ を見つける.

Step 2: for $i \in [k+1, n-1]$, if $\tilde{e}^i$ exists do

- $T_i \leftarrow T \cup \tilde{e}^i \backslash e^i$, $T_i$ を出力
- $F_i \leftarrow F \cup \{e^{k+1}, \ldots, e^{i-1}\}, R_i \leftarrow R \cup \{e^i\}$
- $All\_MST_1(F_i, R_i, T_i)$ を再帰呼び出し

**ここで $O(nm)$ かかる**

**ここは $O(1)$**

**必ず解を 1 つ出力する**

# 目次

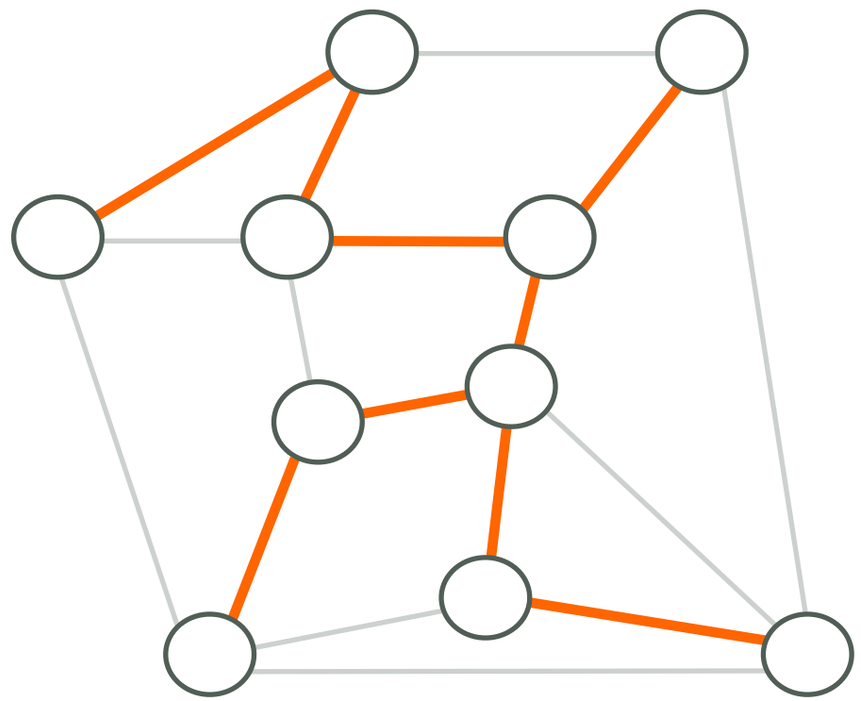
- 準備・問題概要
- $ALL\_MST : O(N(mn + n^2 \log n))$
  - 愚直な分割法による列挙
- $ALL\_MST_1 : O(Nmn)$
  - cut-set を用いた効率の良い列挙方法
- $ALL\_MST_2 : O(Nm \log n)$
  - 上記アルゴリズムの改良

# 計算量改善の概要

- 前章では，各 $e^i \in T$ に対して 代替辺 $\tilde{e}^i$ を見つけるのに $O(m)$ かかり，全体で $O(nm)$ かかっていた.

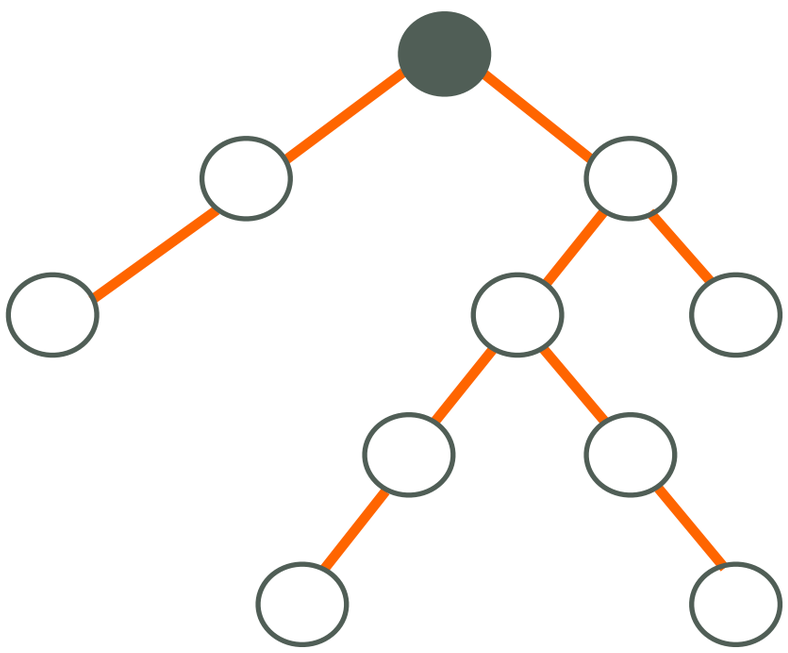
- 代替辺になり得る辺の集合を $AVL$ 木で管理し，全ての $e^i$ に対する代替辺 $\tilde{e}^i$ を $O(m \log n)$ で構築.

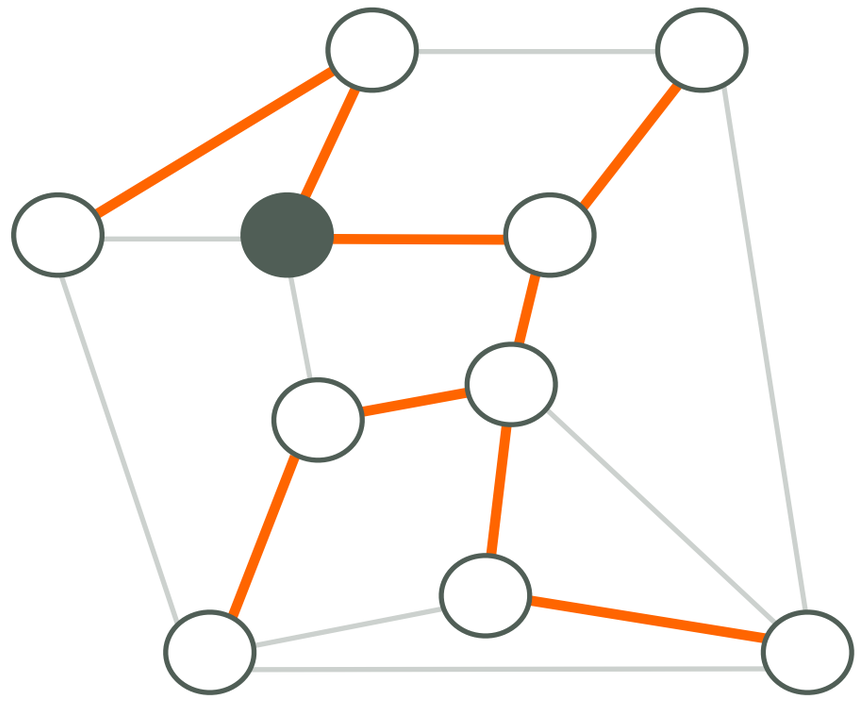
  - $AVL$ 木: 集合に対して
    値の挿入，削除，検索が
    $O(\log k)$ で達成可能なデータ構造
    - $k :=$ 集合のサイズ



# 根付き木の帰りがけ DFS

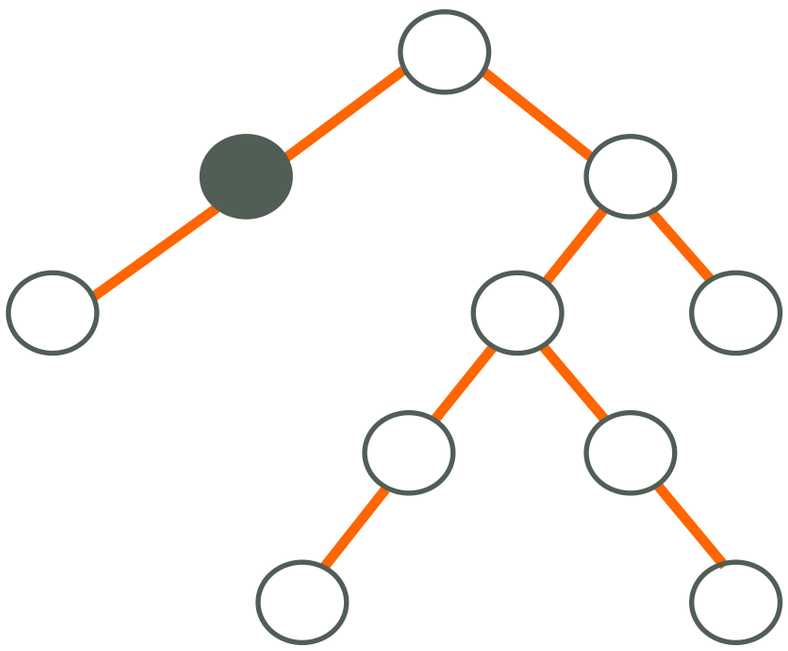
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

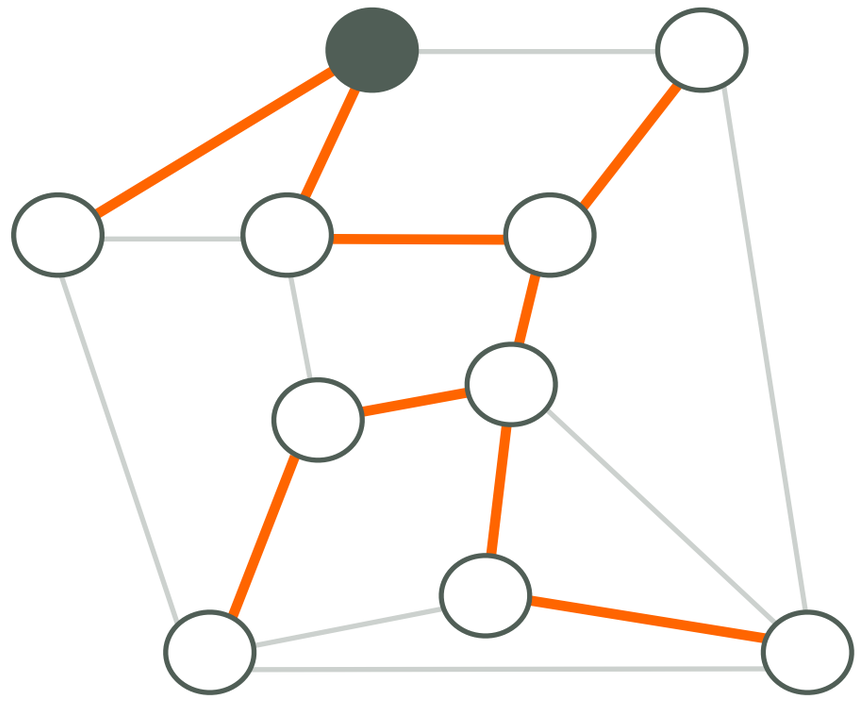




# 根付き木の帰りがけ DFS

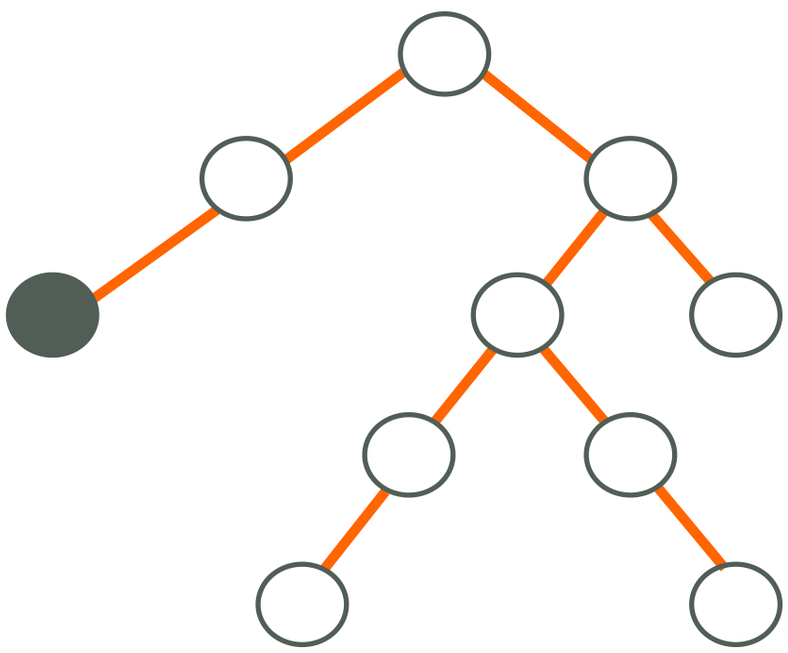
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

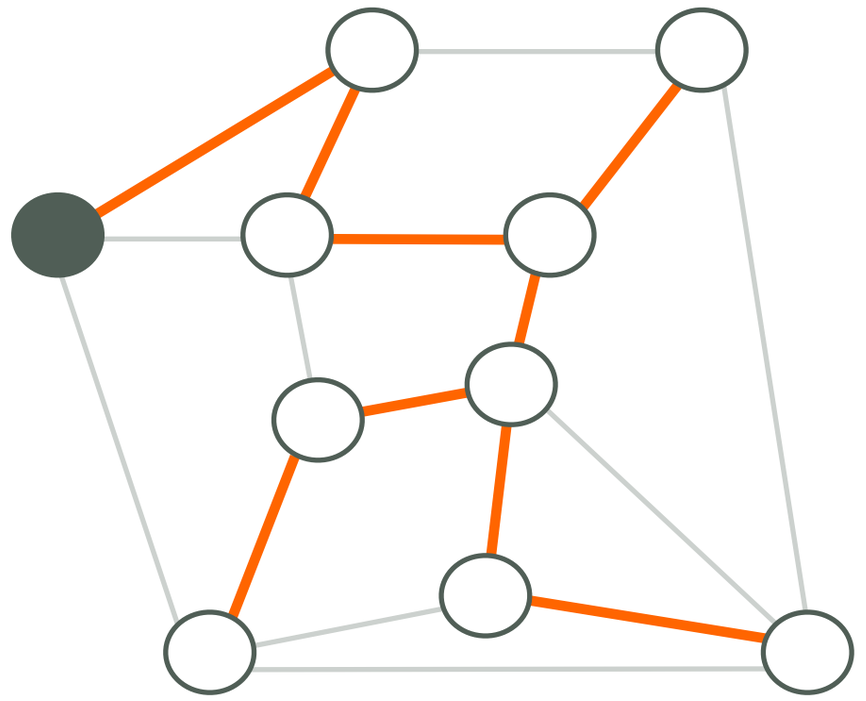




# 根付き木の帰りがけ DFS

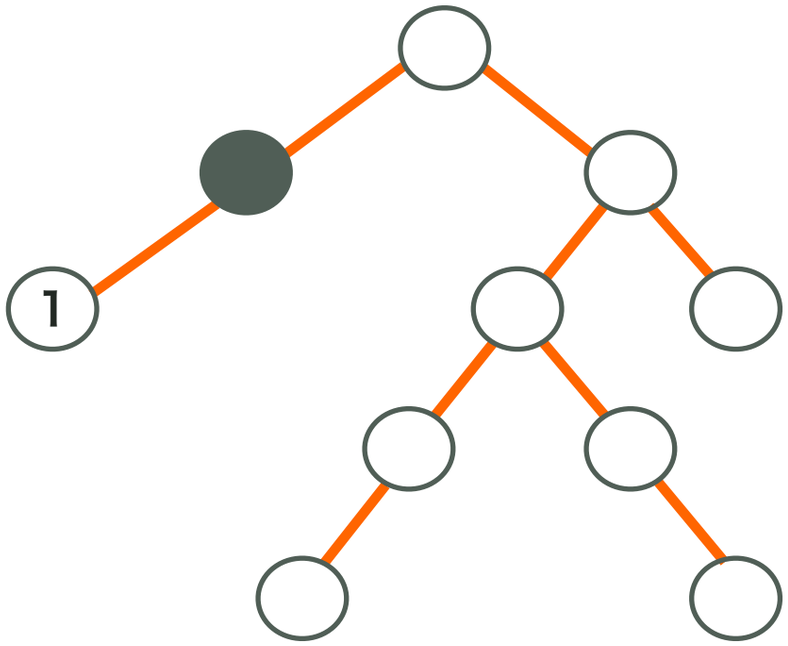
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

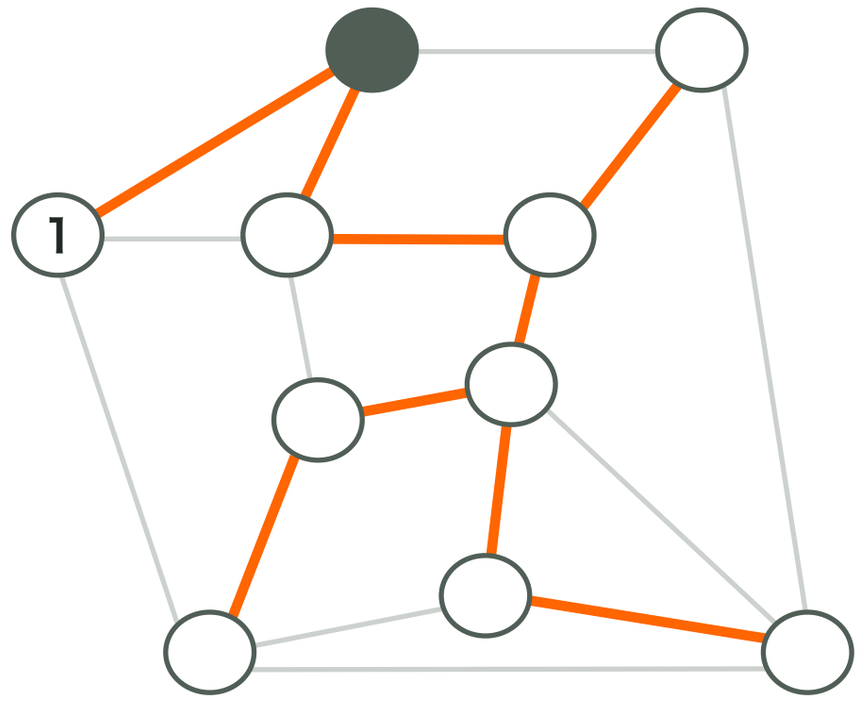




# 根付き木の帰りがけ DFS

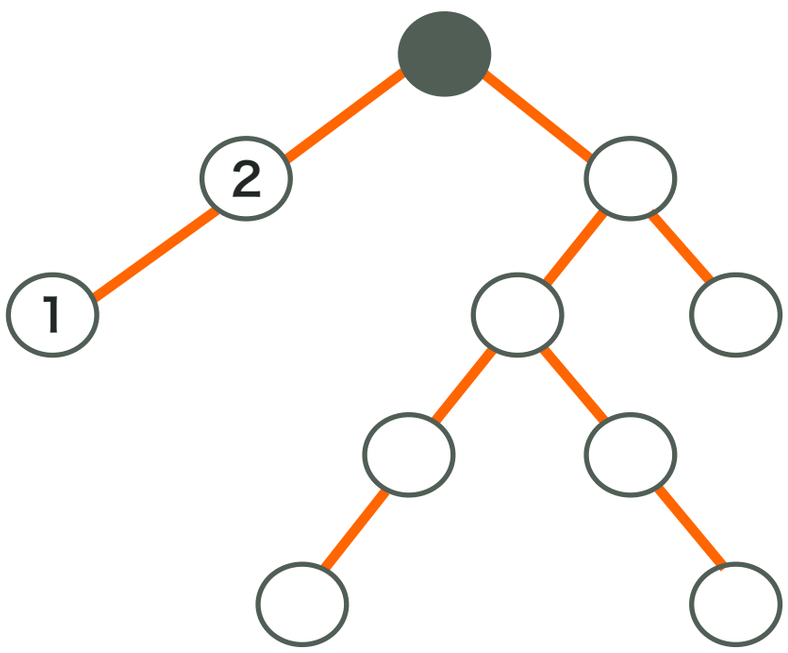
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

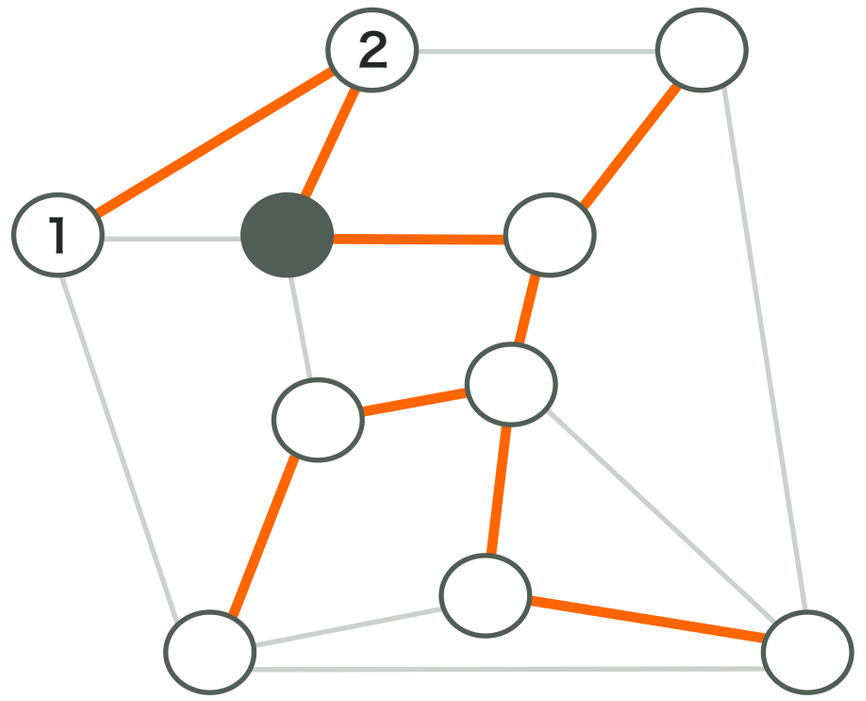


# 根付き木の帰りがけ DFS

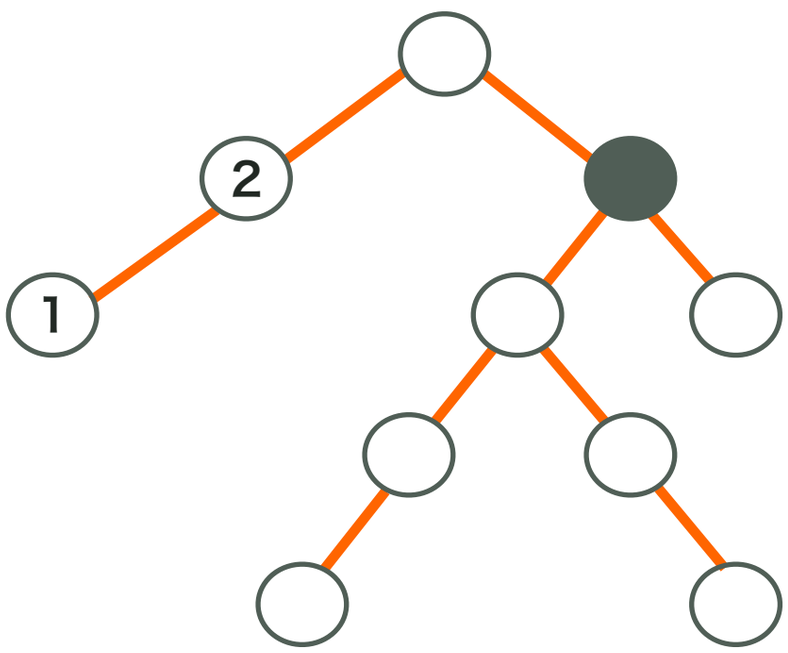
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

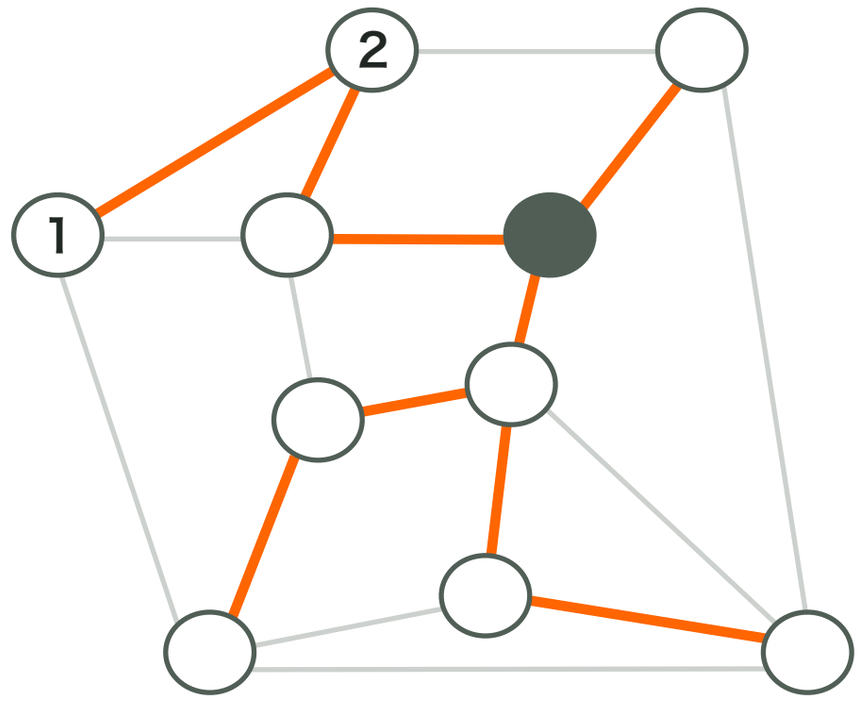


# 根付き木の帰りがけ DFS

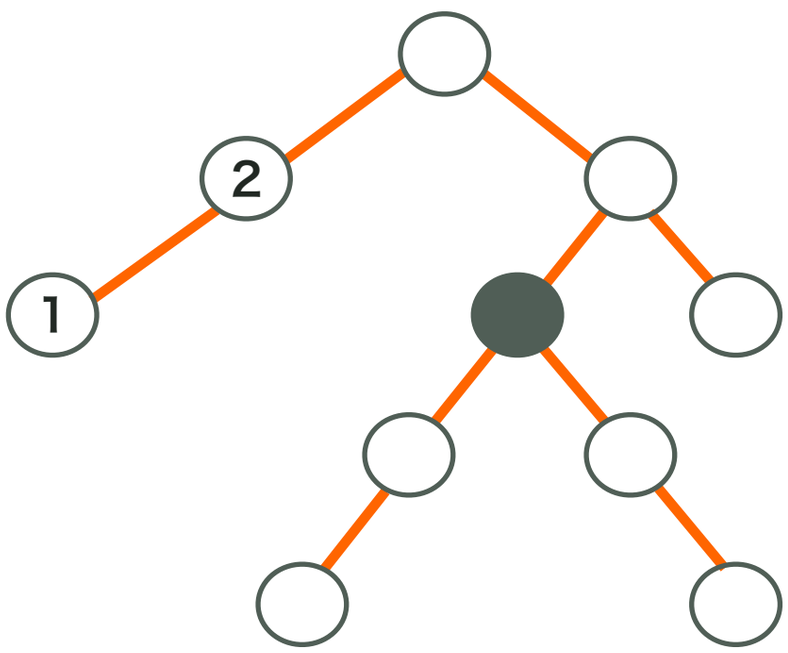
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

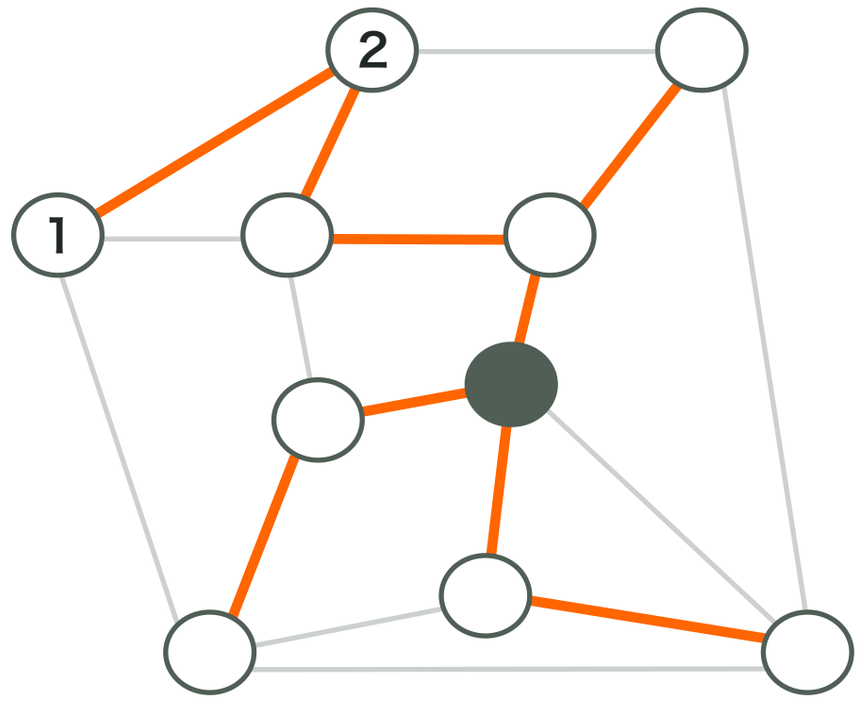


# 根付き木の帰りがけ DFS

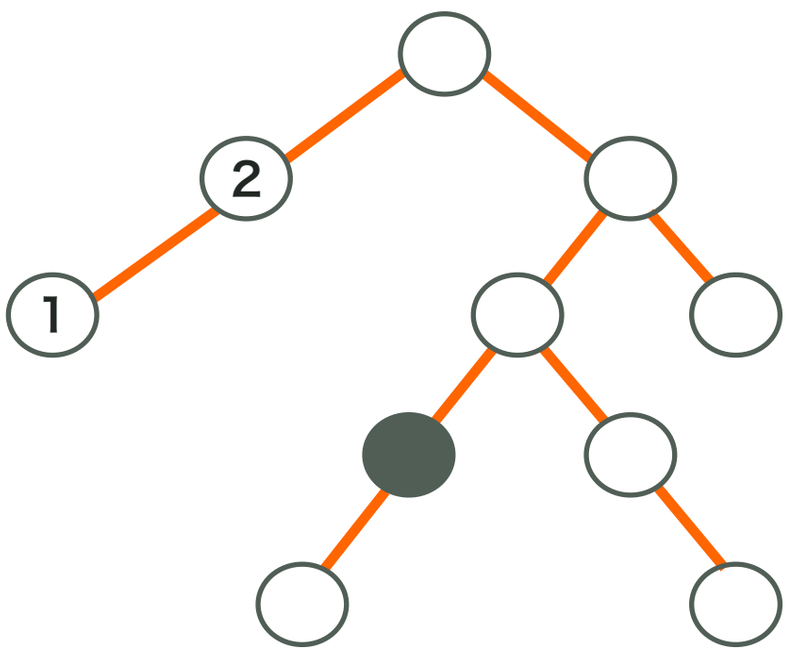
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

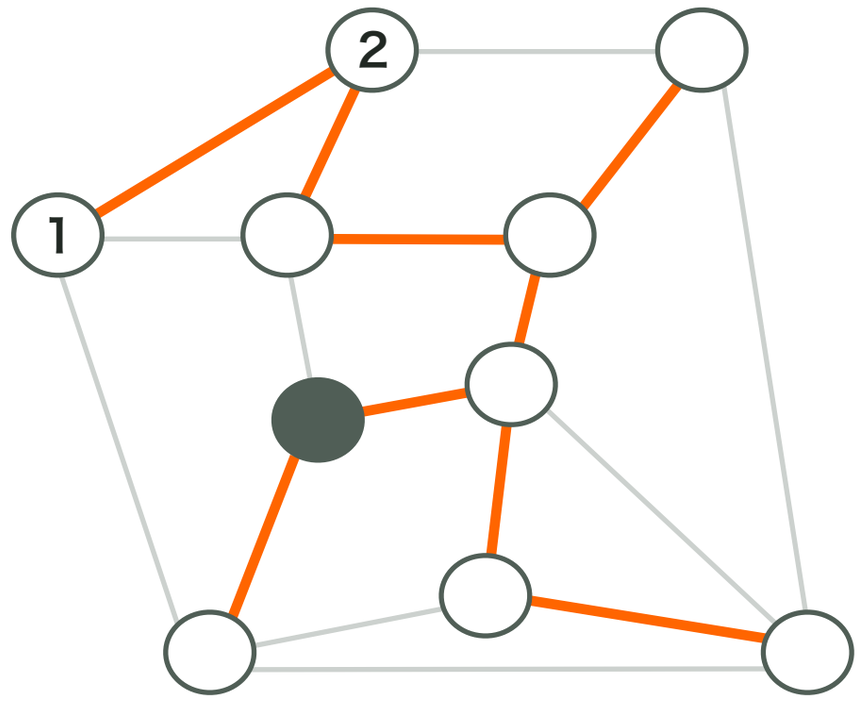


# 根付き木の帰りがけ DFS

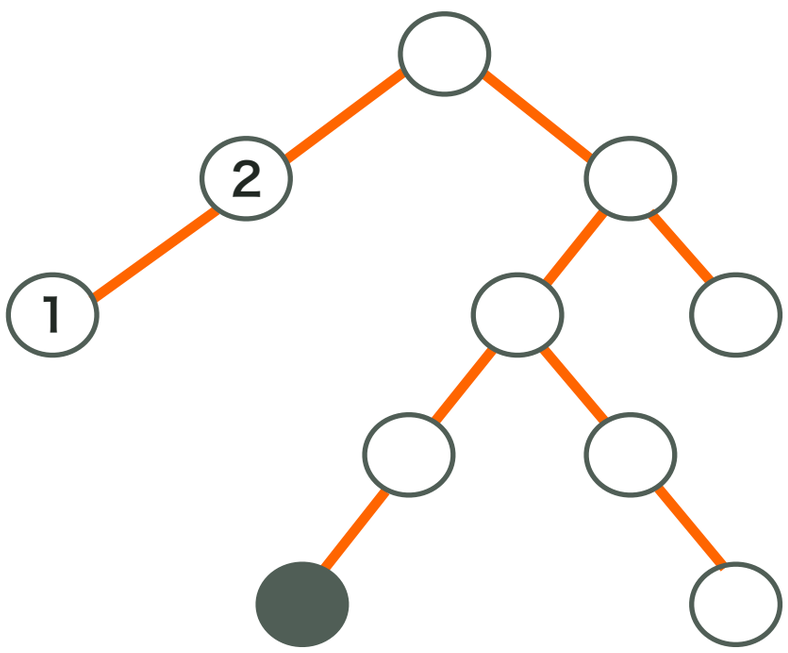
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

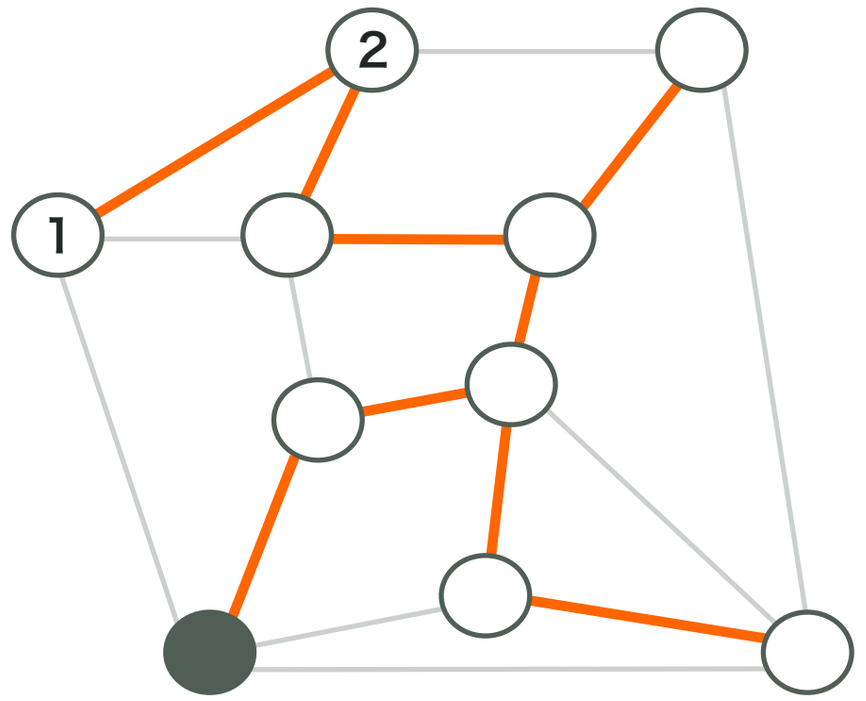


# 根付き木の帰りがけ DFS

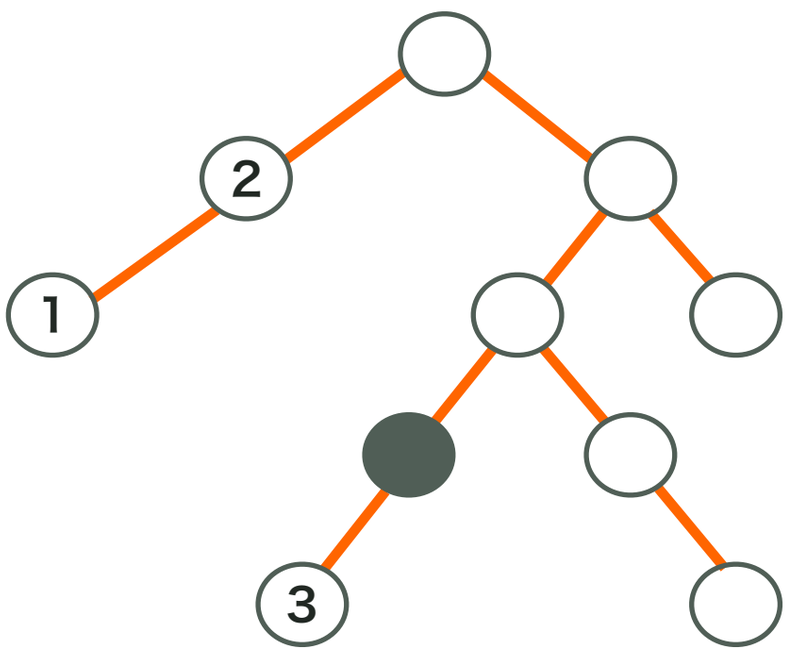
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

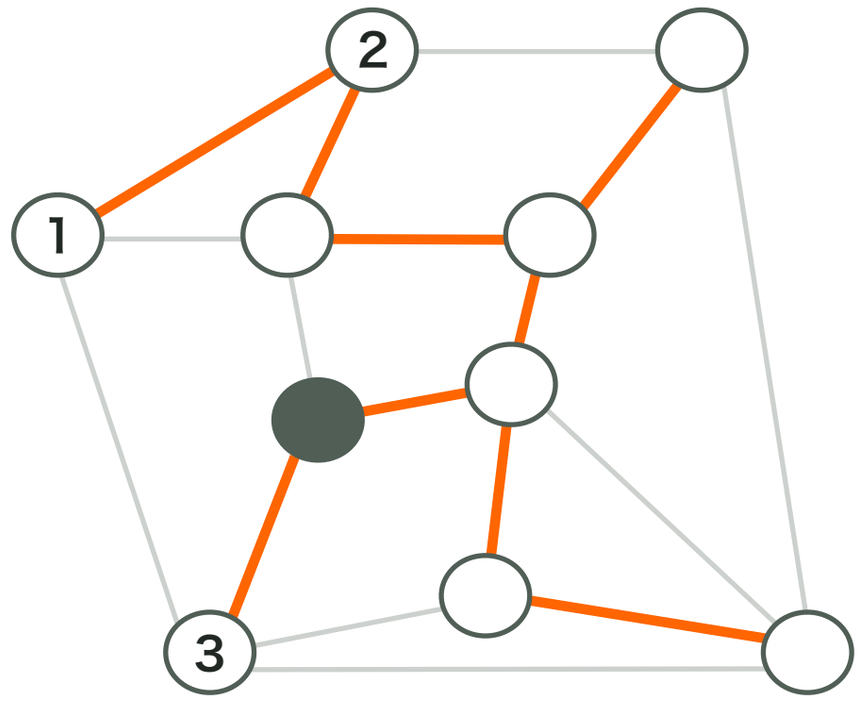


# 根付き木の帰りがけ DFS

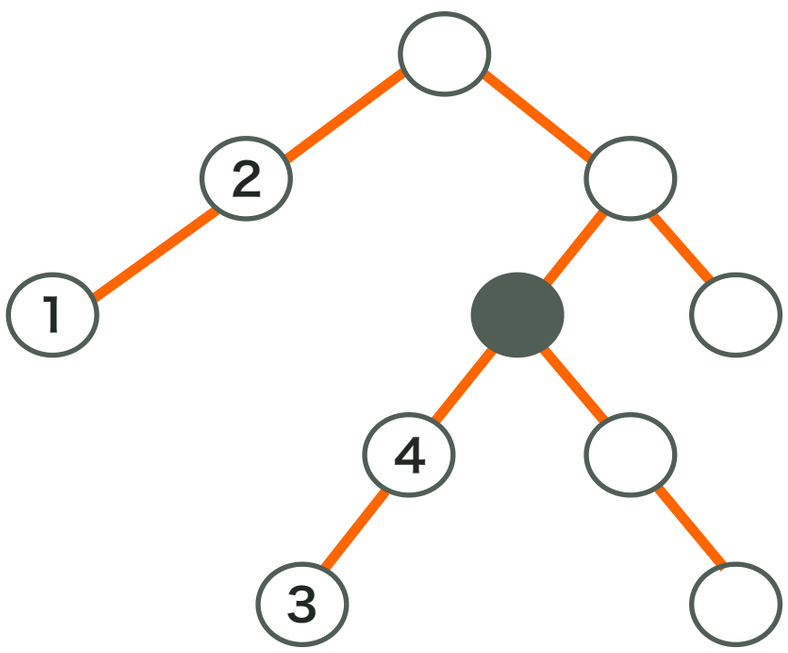
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

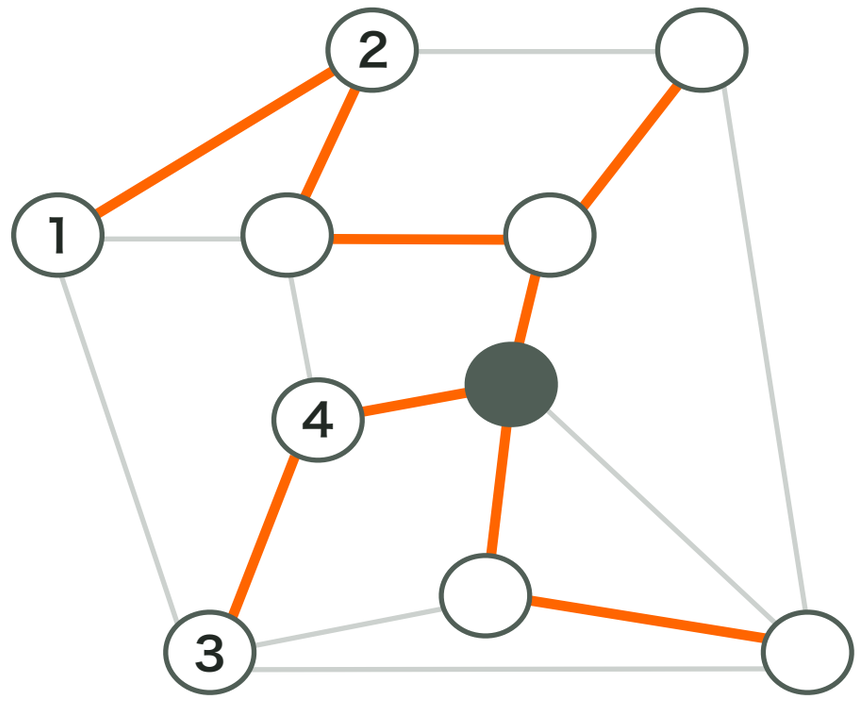


# 根付き木の帰りがけ DFS

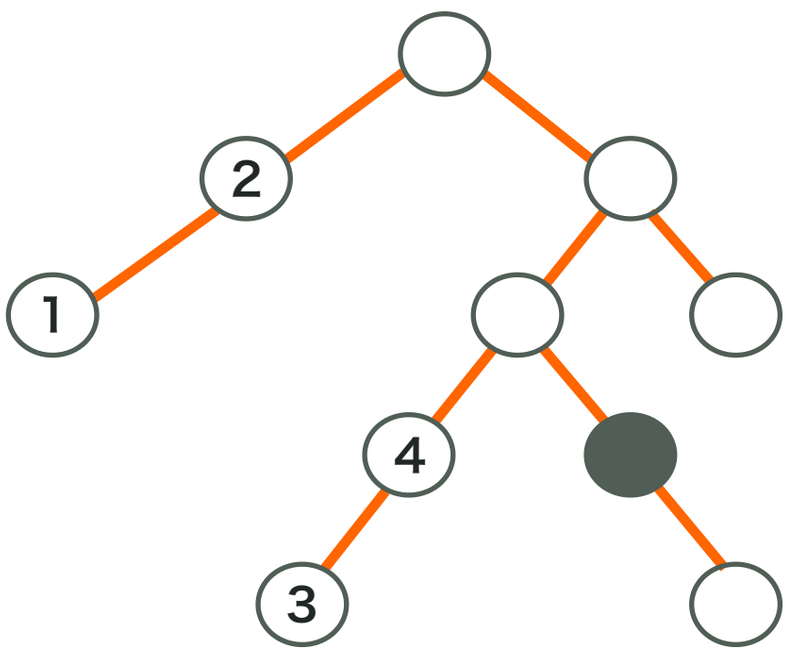
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

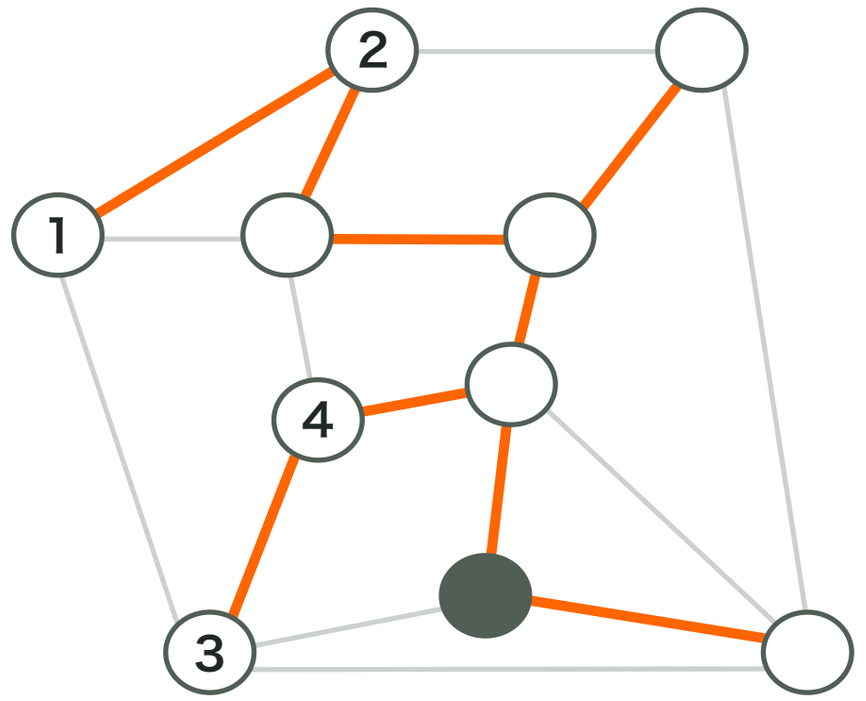


# 根付き木の帰りがけ DFS

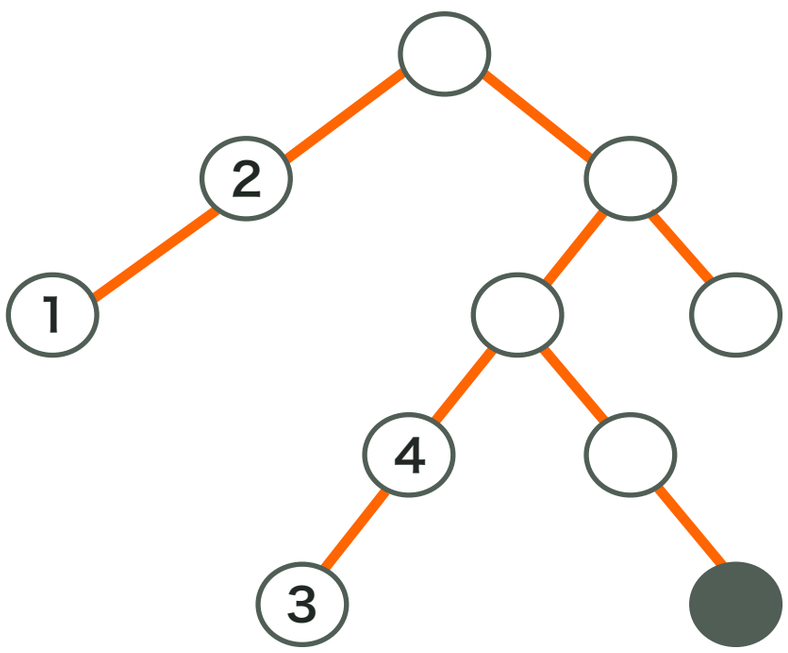
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

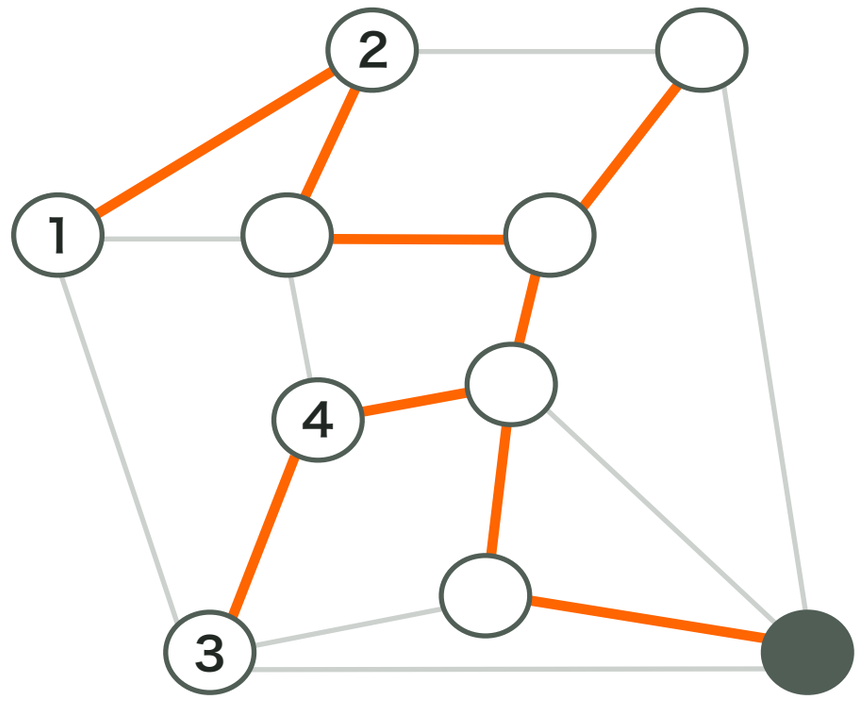


# 根付き木の帰りがけ DFS

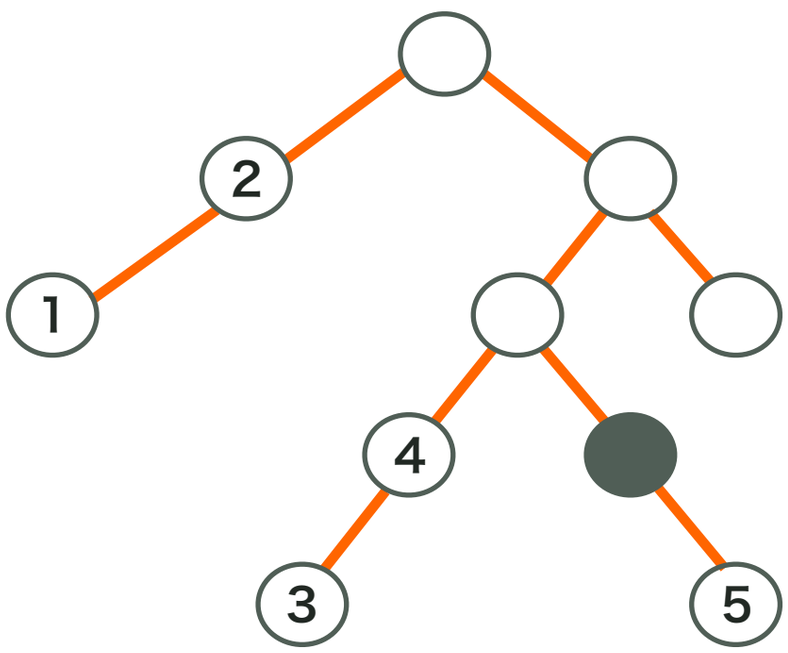
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

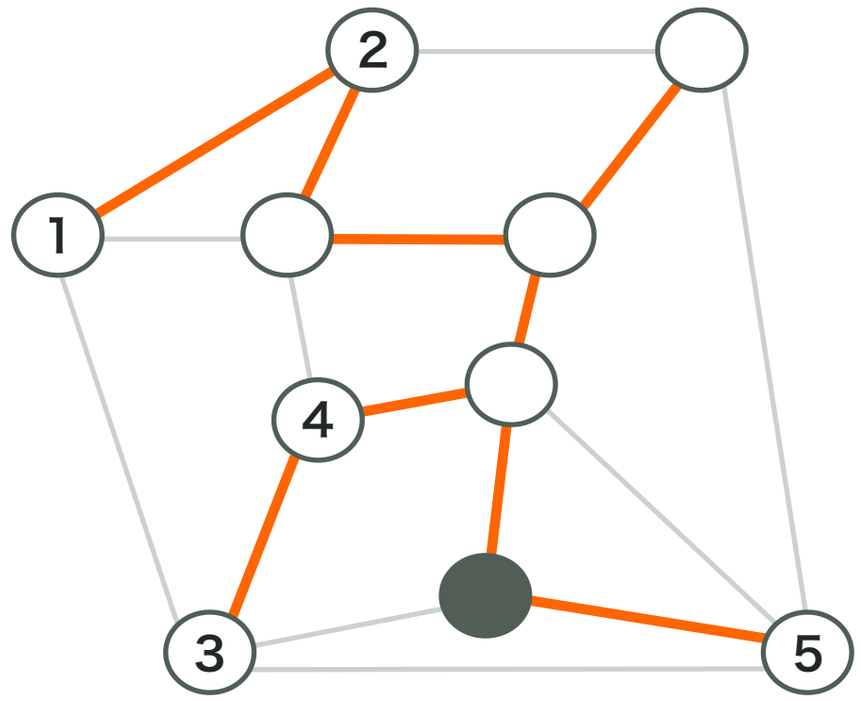


# 根付き木の帰りがけ DFS

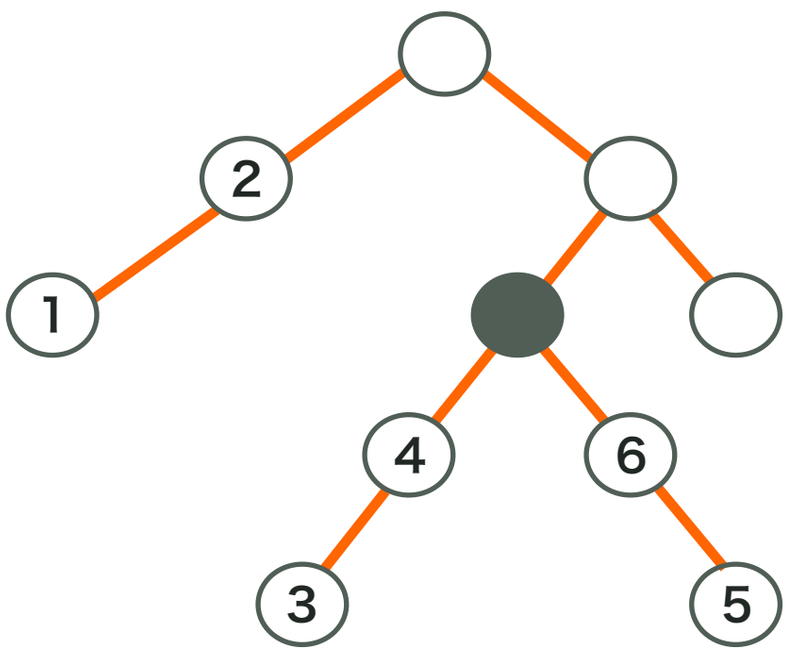
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i \mid i = 1, \ldots, n\}$ とする

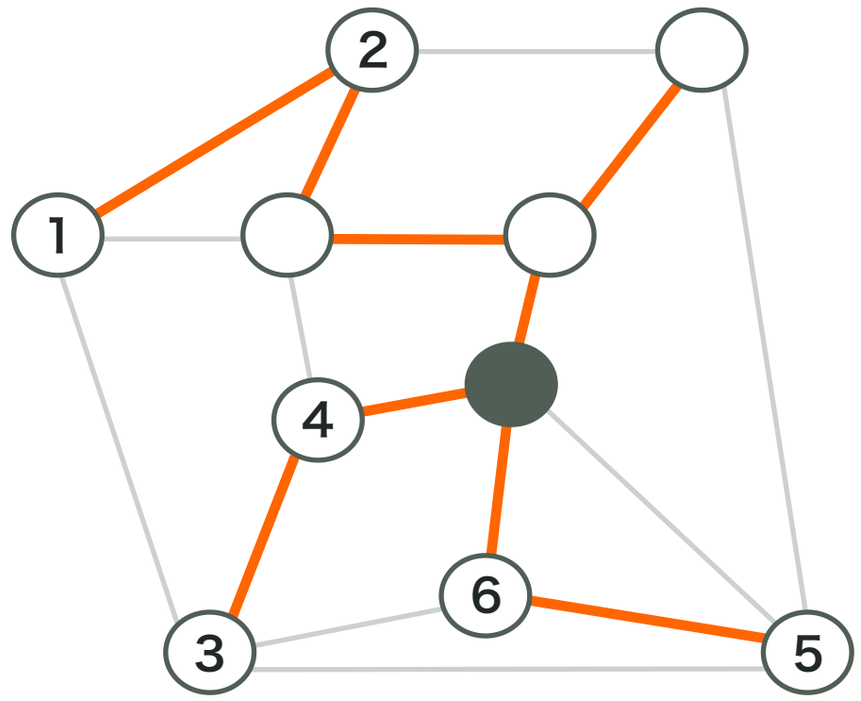


# 根付き木の帰りがけ DFS

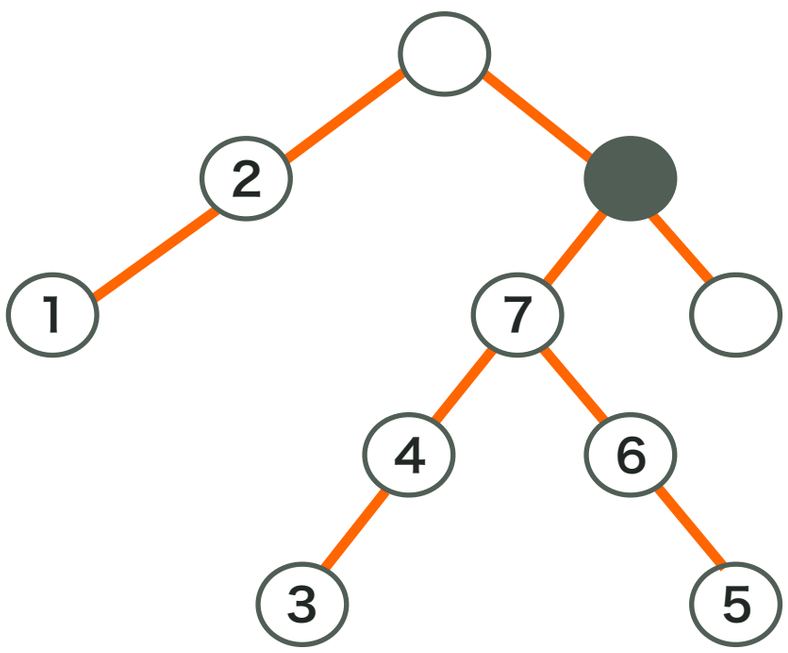
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i \mid i = 1, \ldots, n\}$ とする

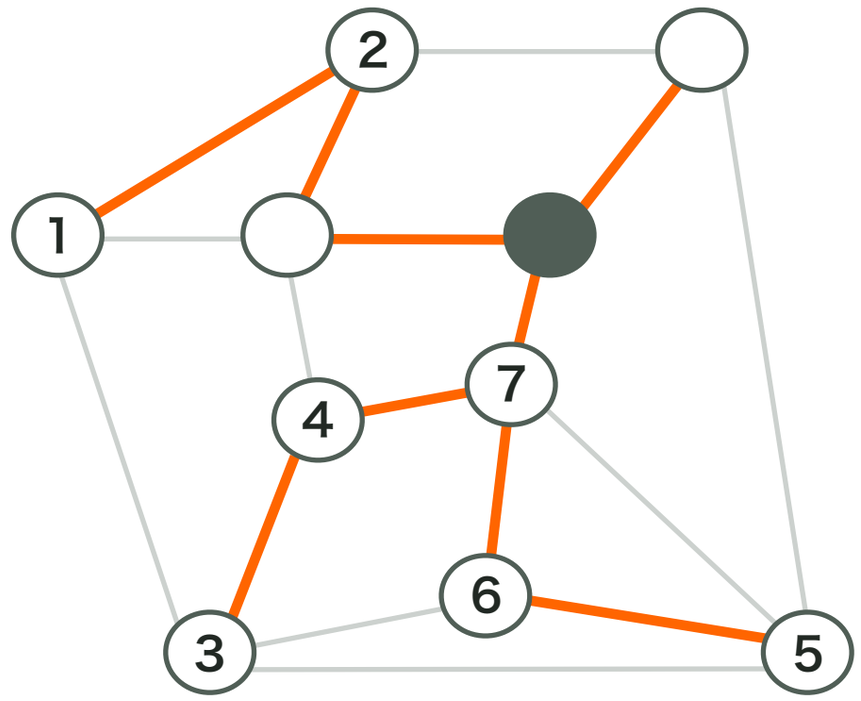


# 根付き木の帰りがけ DFS

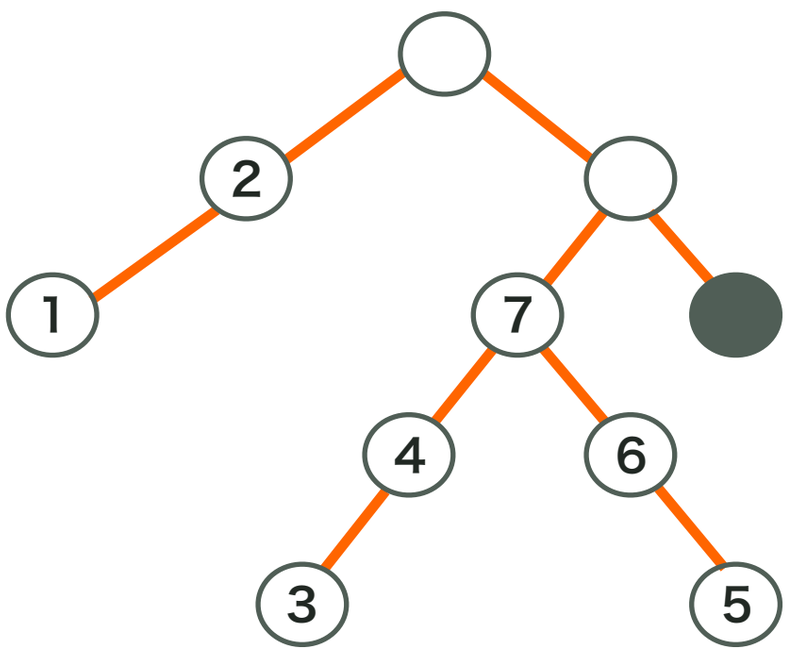
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

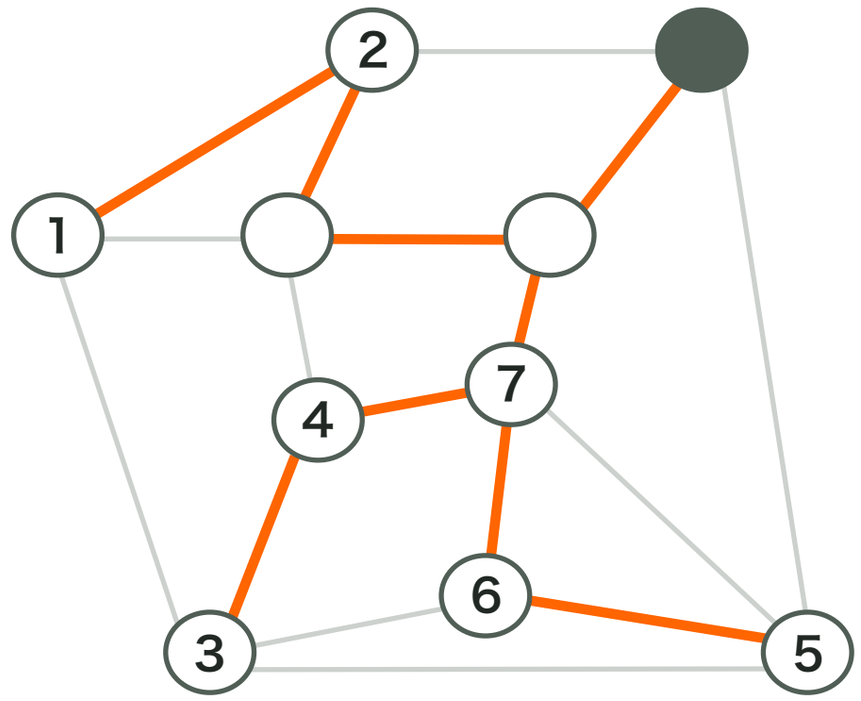


# 根付き木の帰りがけ DFS

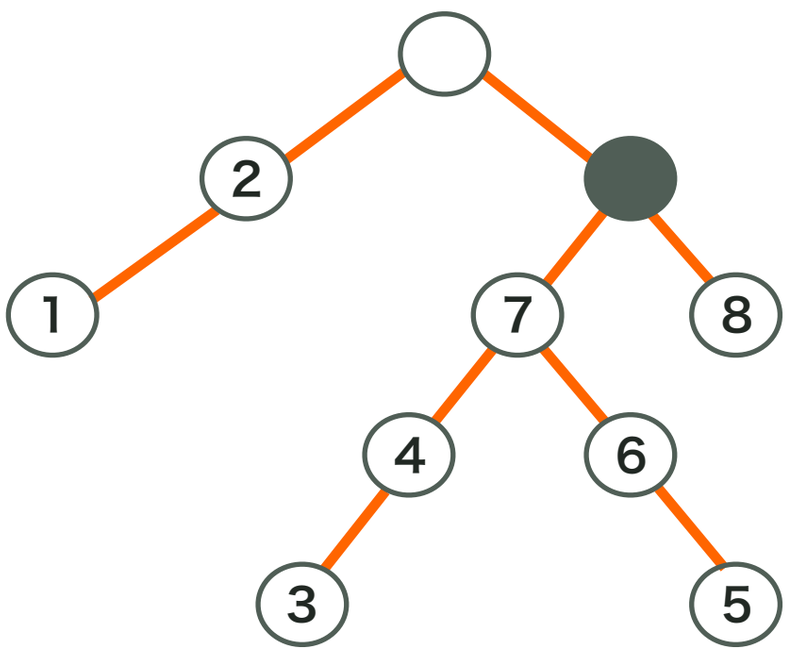
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

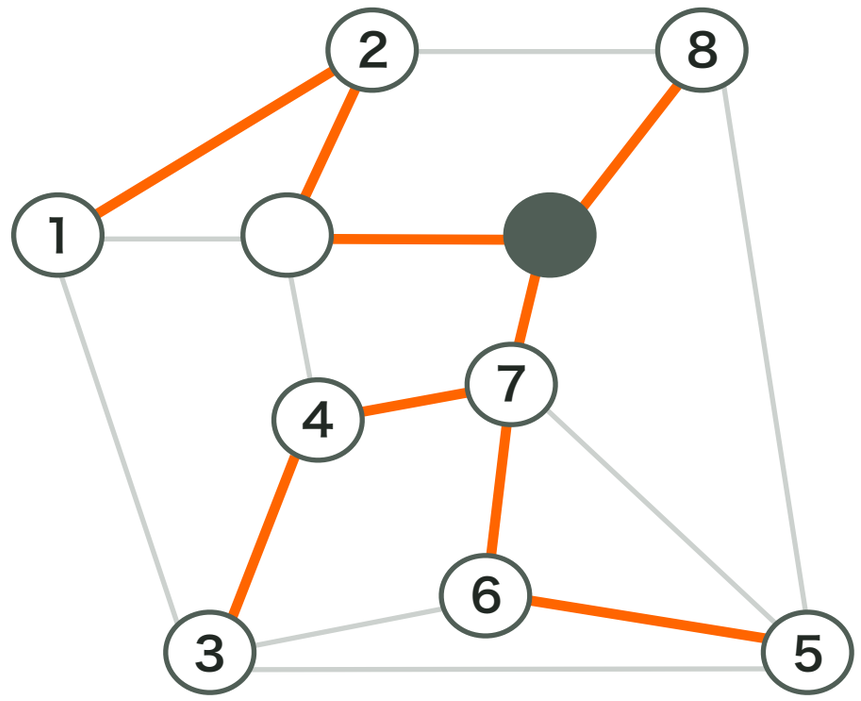


# 根付き木の帰りがけ DFS

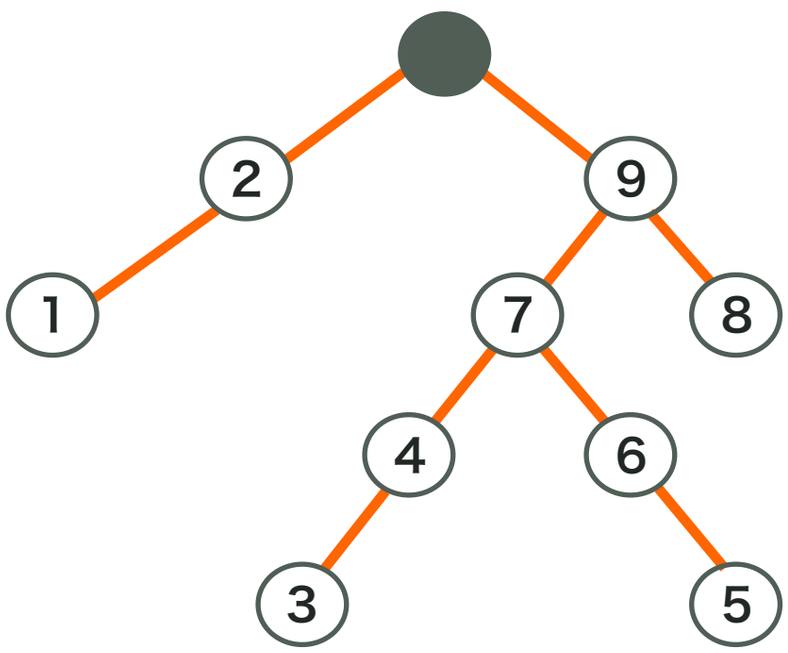
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i \mid i = 1, \ldots, n\}$ とする

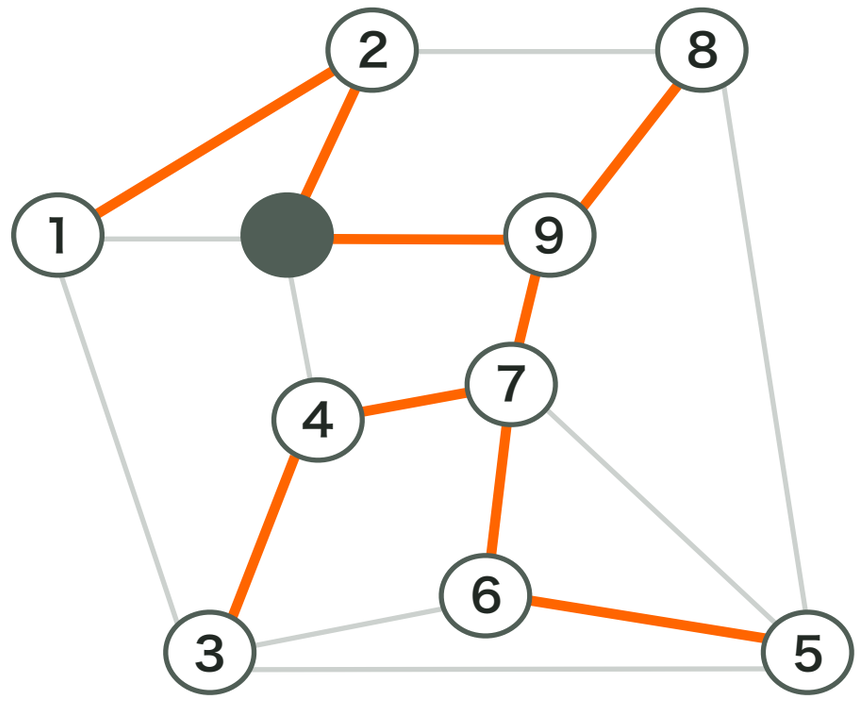


# 根付き木の帰りがけ DFS

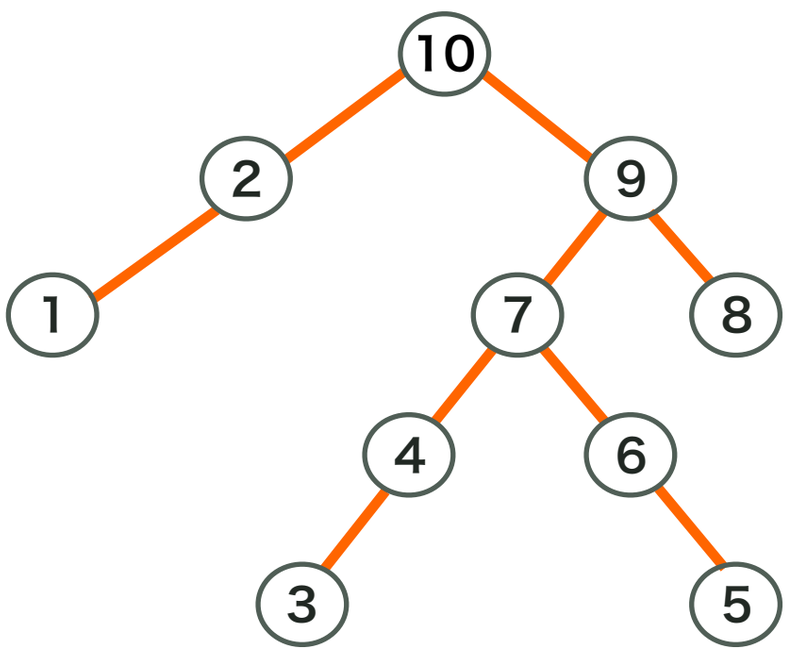
- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

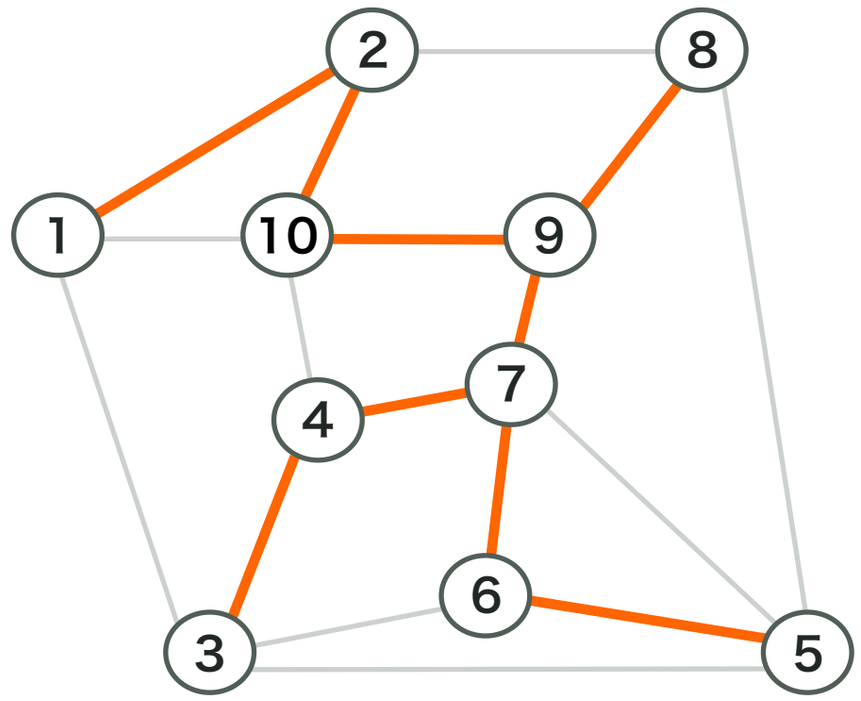


# 根付き木の帰りがけ DFS

- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1, \ldots, n\}$ とする

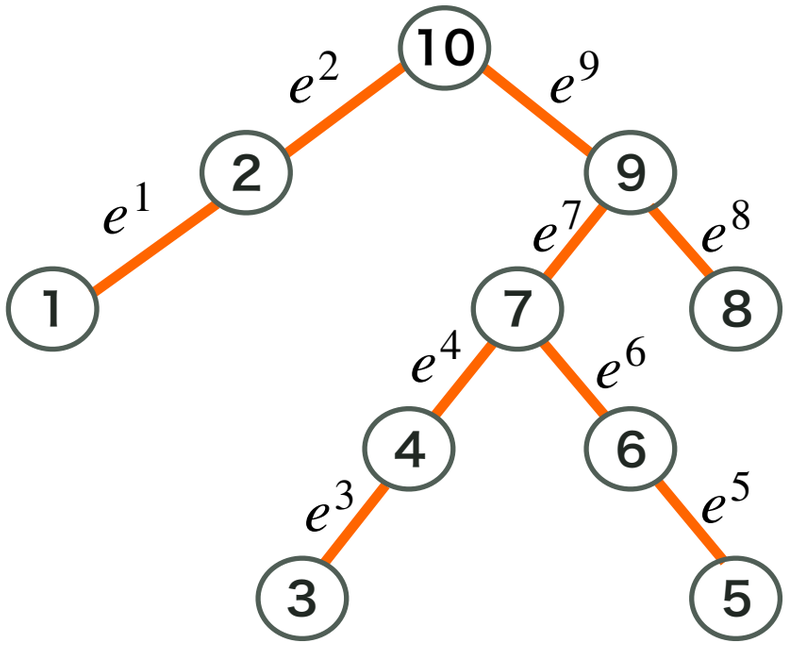


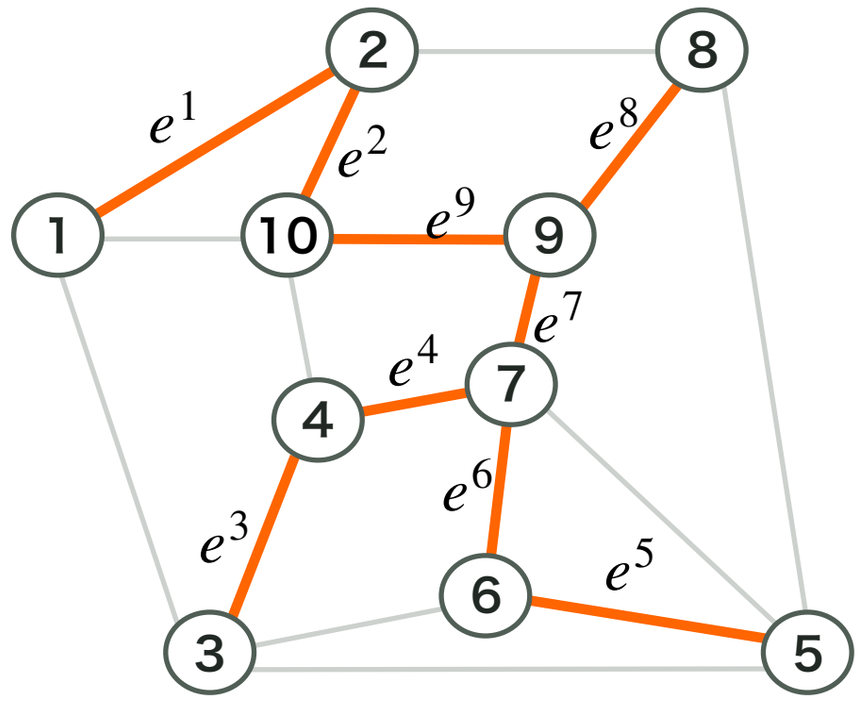



# 根付き木の帰りがけ DFS

- 最小全域木 $T$ に対して，適当な頂点を根と定める．
  根からDFS を行い，帰りがけ順に番号を振り直す．
  - 振り直された番号を $\{v^i | i = 1,\ldots,n\}$ とする

- 頂点 $v^l$ から，その親に向かう辺を $e^l$ と表記する．

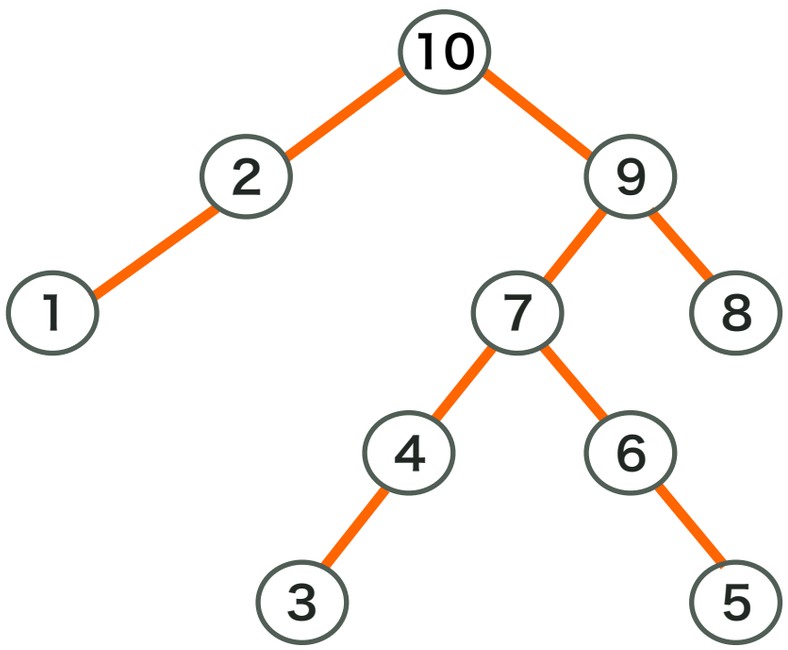


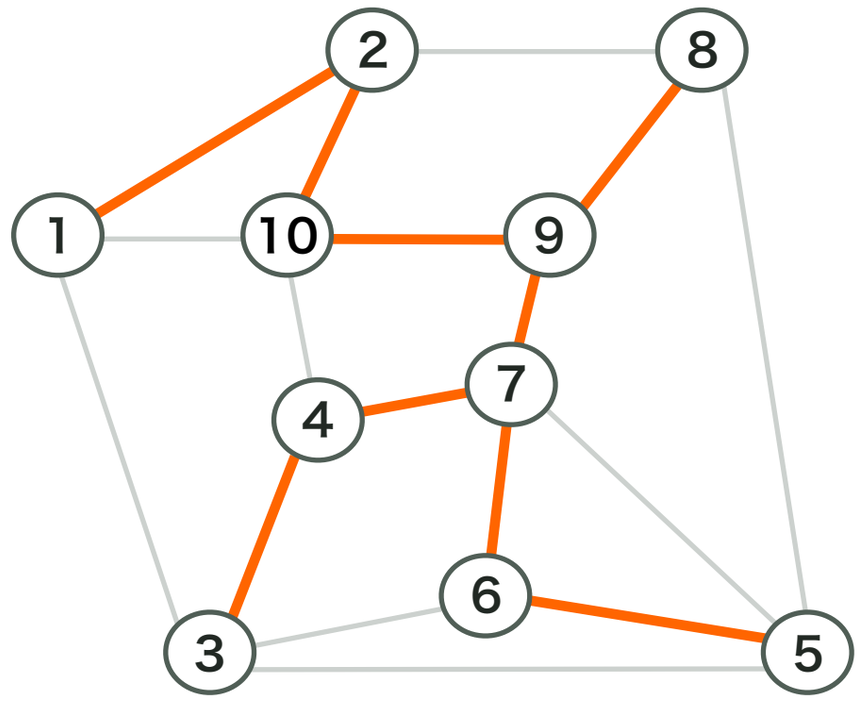



# インターバル

- 頂点 $v^i$ に対して，インターバル $[\underline{\sigma}_i, \overline{\sigma}_i]$ を以下のように定める．

  $j \in [\underline{\sigma}_i, \overline{\sigma}_i] \Leftrightarrow v^j$ は $v^i$ の子孫である

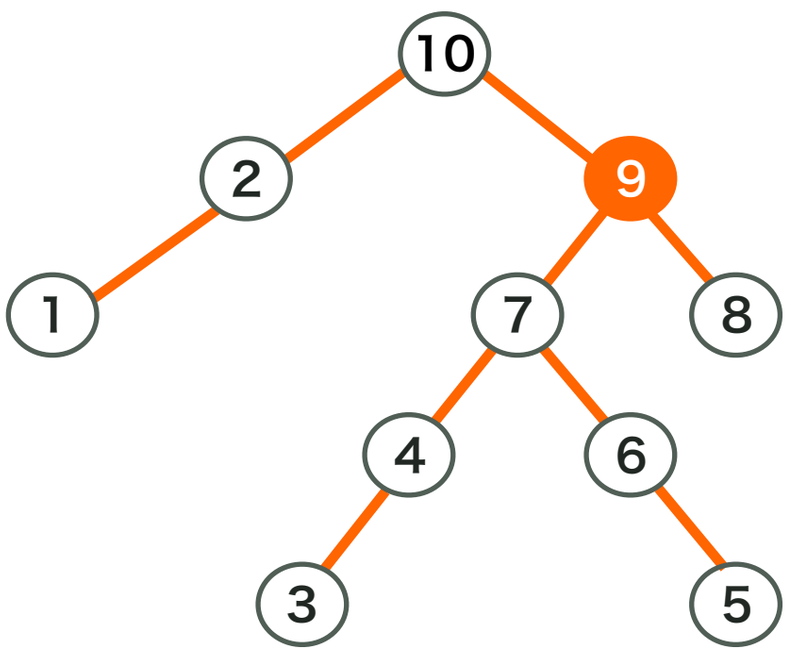


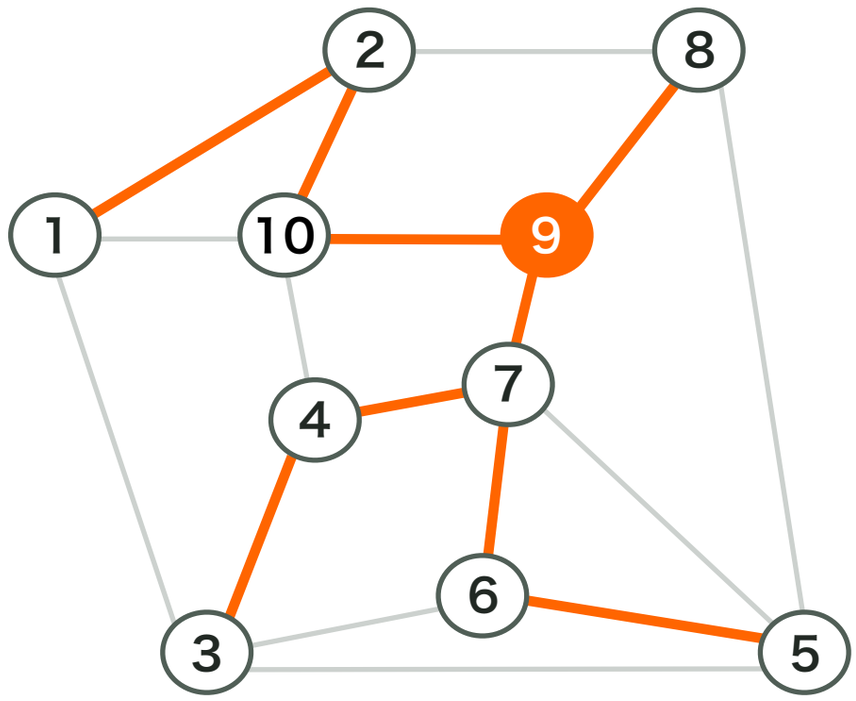
# インターバル

- 頂点 $v^i$ に対して，インターバル $[\underline{\sigma}_i, \overline{\sigma}_i]$ を以下のように定める．

  $j \in [\underline{\sigma}_i, \overline{\sigma}_i] \Leftrightarrow v^j$ は $v^i$ の子孫である



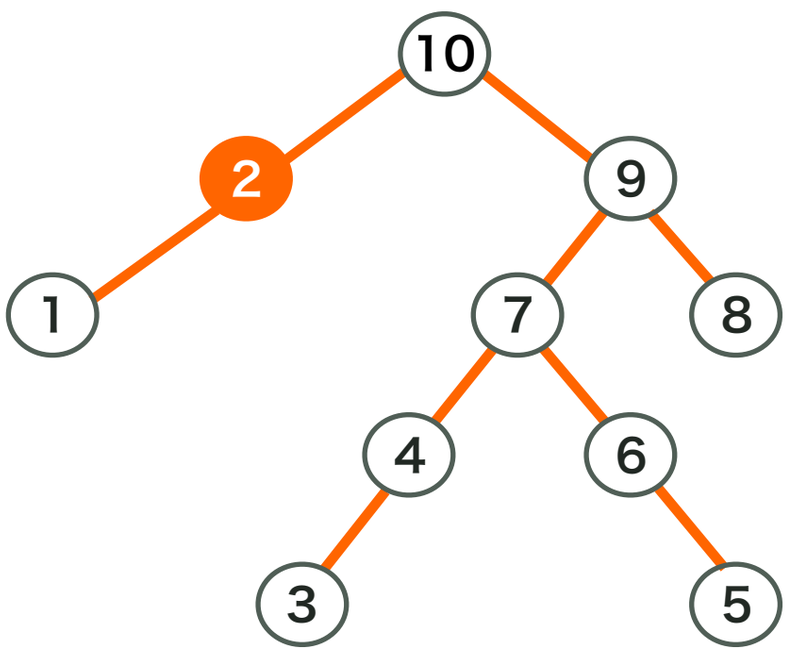

$$[\underline{\sigma}_9, \overline{\sigma}_9] = [3,9]$$



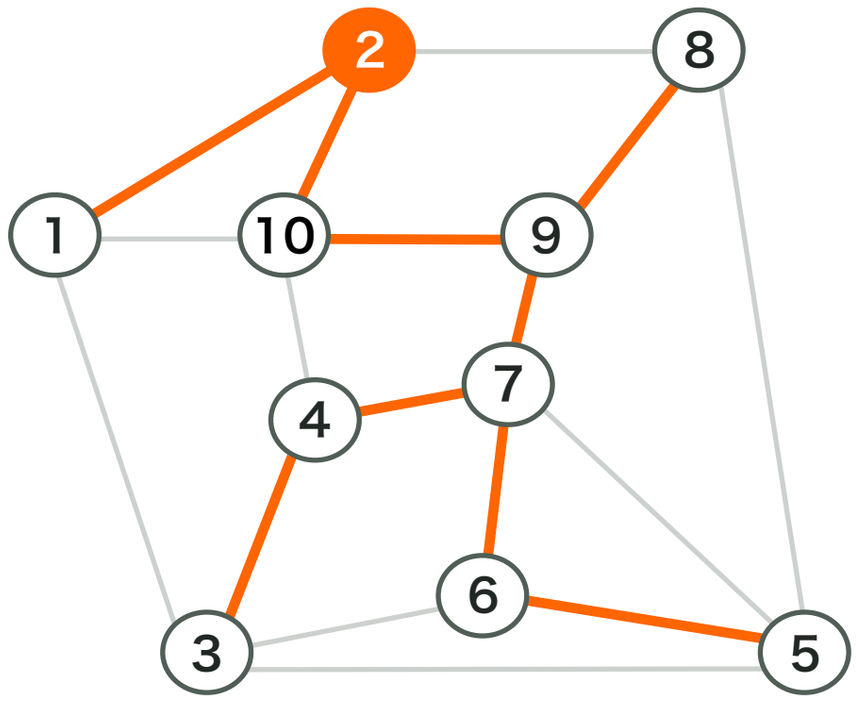
# インターバル

- 頂点 $v^i$ に対して，インターバル $[\underline{\sigma}_i, \overline{\sigma}_i]$ を以下のように定める．

  $j \in [\underline{\sigma}_i, \overline{\sigma}_i] \Leftrightarrow v^j$ は $v^i$ の子孫である



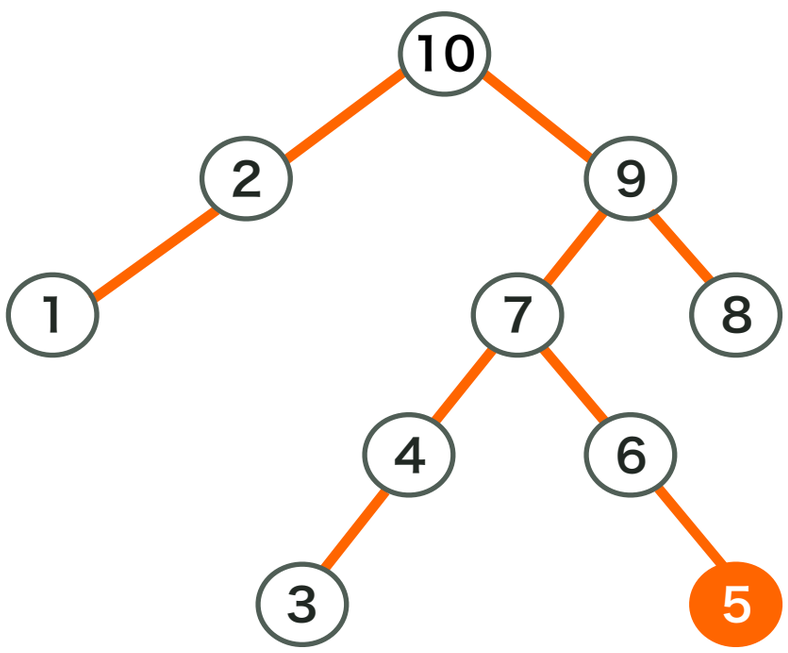

$$[\underline{\sigma}_2, \overline{\sigma}_2] = [1,2]$$



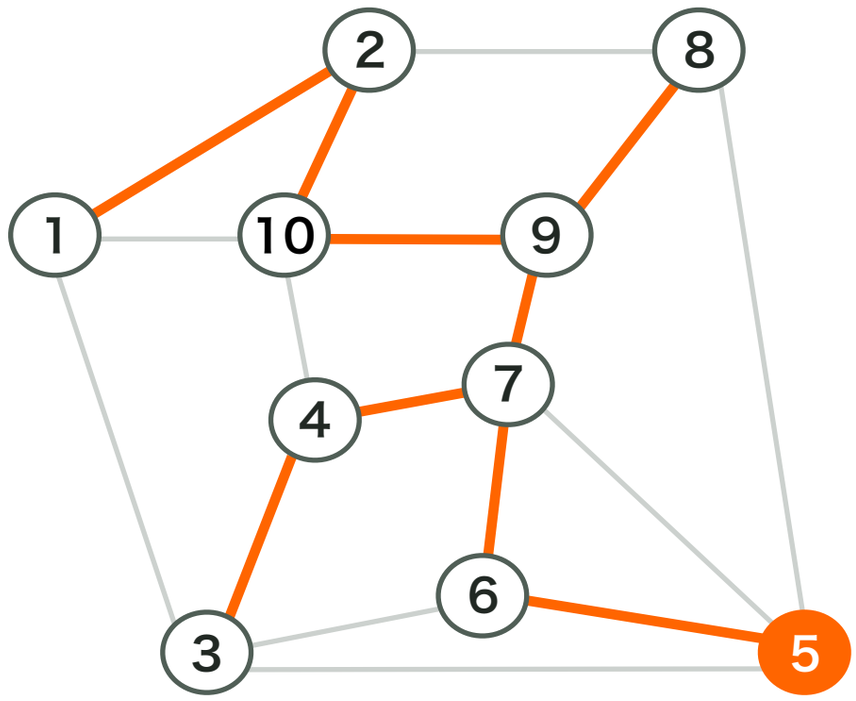
# インターバル

- 頂点 $v^i$ に対して，インターバル $[\underline{\sigma}_i, \overline{\sigma}_i]$ を以下のように定める．

  $j \in [\underline{\sigma}_i, \overline{\sigma}_i] \Leftrightarrow v^j$ は $v^i$ の子孫である



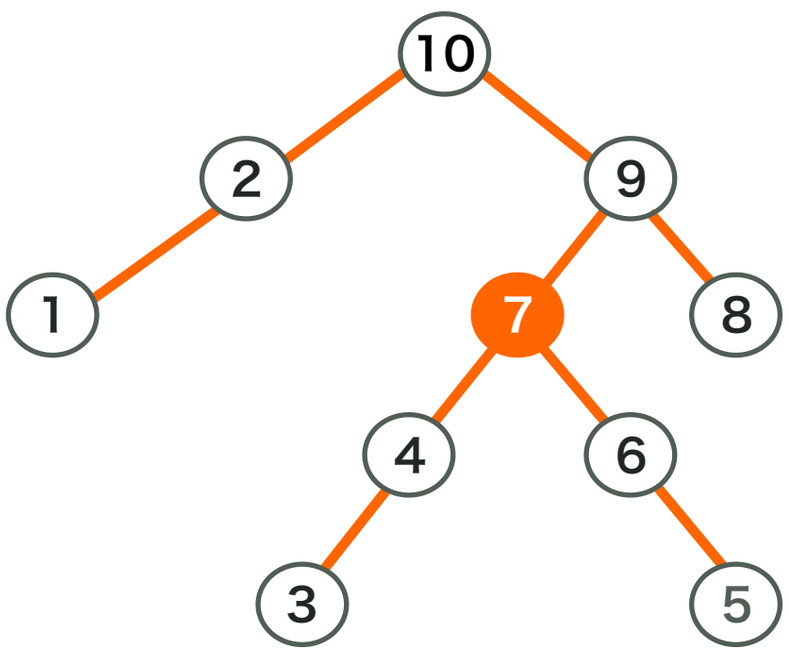

$$[\underline{\sigma}_5, \overline{\sigma}_5] = [5,5]$$



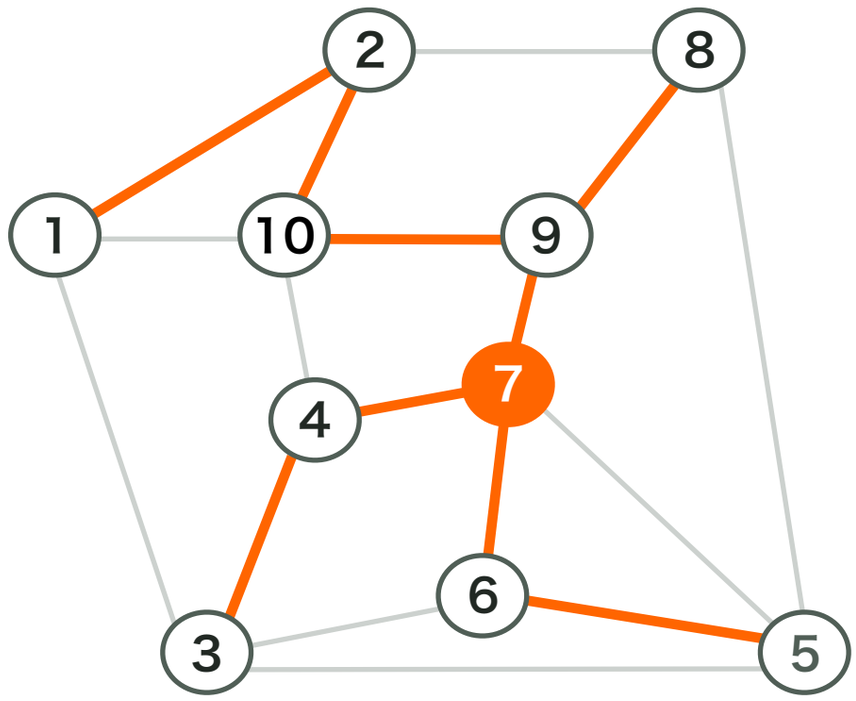
# インターバル

- 頂点 $v^i$ に対して，インターバル $[\underline{\sigma}_i, \overline{\sigma}_i]$ を以下のように定める．

  $j \in [\underline{\sigma}_i, \overline{\sigma}_i] \Leftrightarrow v^j$ は $v^i$ の子孫である




$$[\underline{\sigma}_7, \overline{\sigma}_7] = [3,7]$$

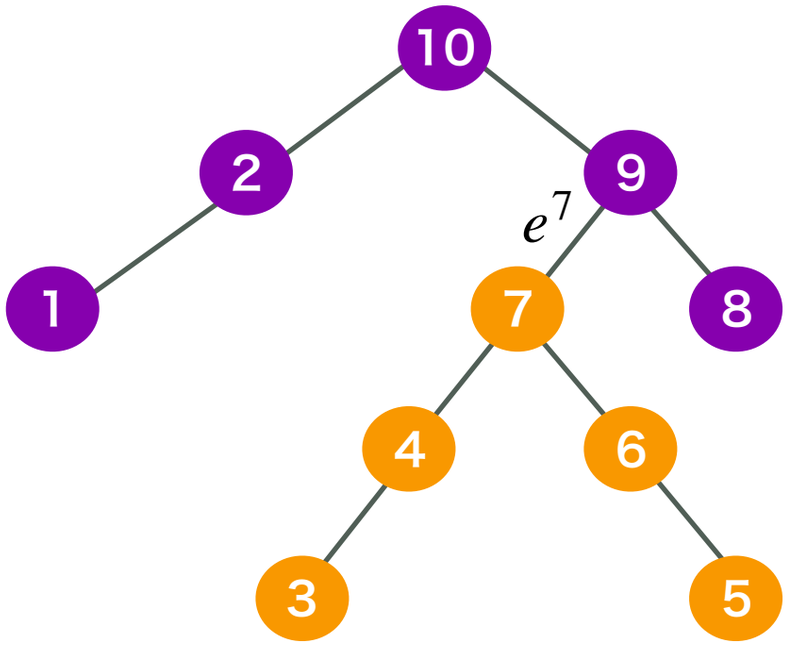


# 計算量削減のアイディア

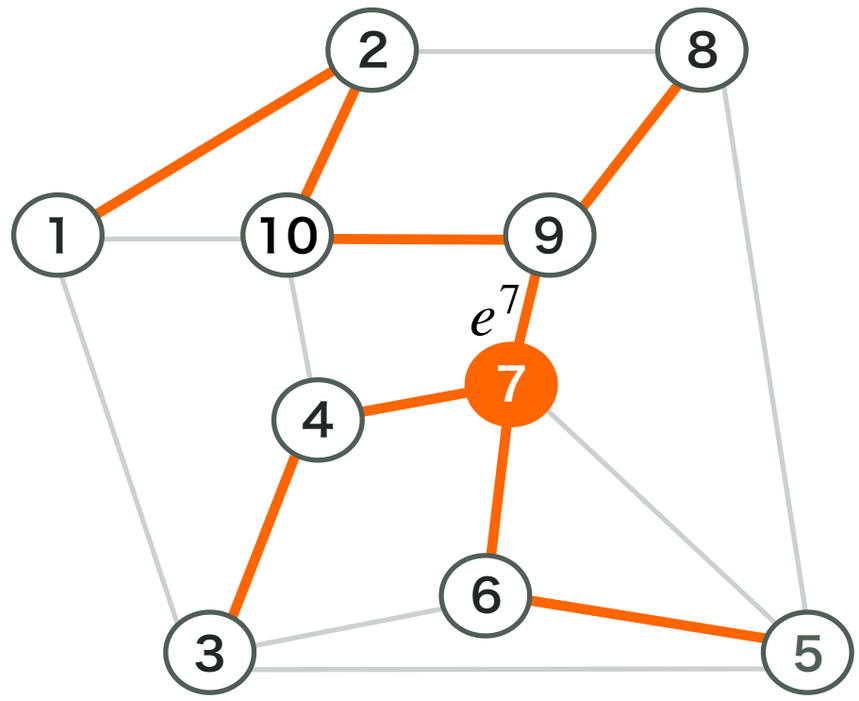
- 辺 $e^i$ の代替辺 $\{v, v'\}$ は，以下の条件を必ず満たす．

  $v \in [\underline{\sigma}_i, \overline{\sigma}_i]$ かつ $v' \in [\underline{\sigma}_i, \overline{\sigma}_i]$

  - 例えば，$e^7$ の代替辺の候補は [3,7] に含まれる頂点から含まれない頂点への辺に限る．

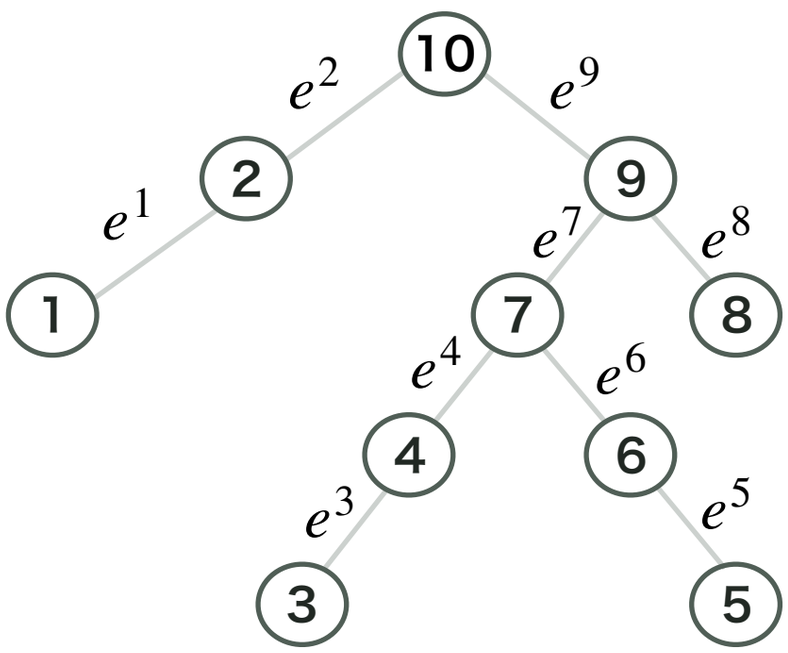


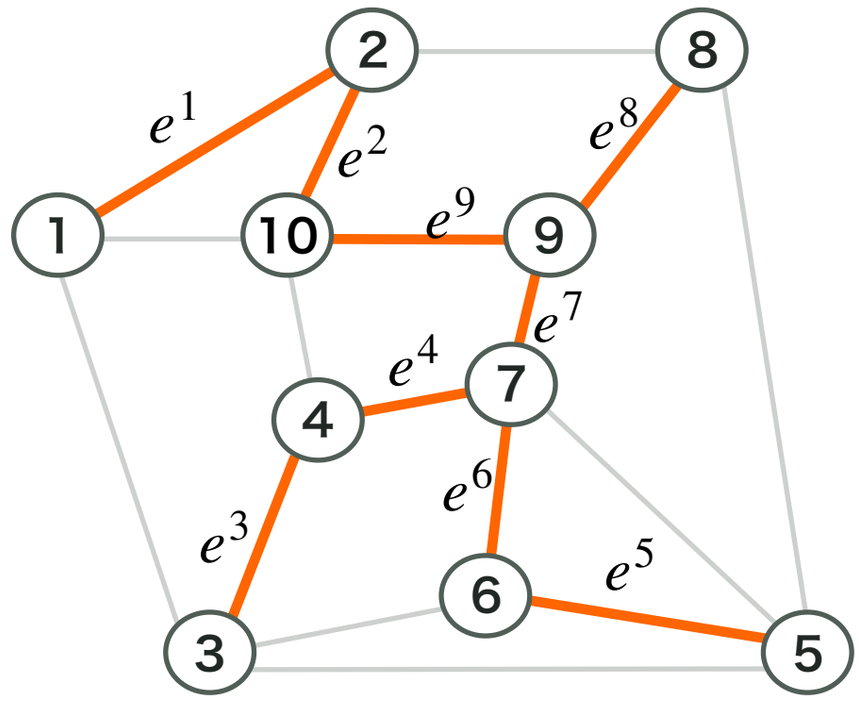

$$[\underline{\sigma}_7, \overline{\sigma}_7] = [3,7]$$



# 計算量削減のアイディア

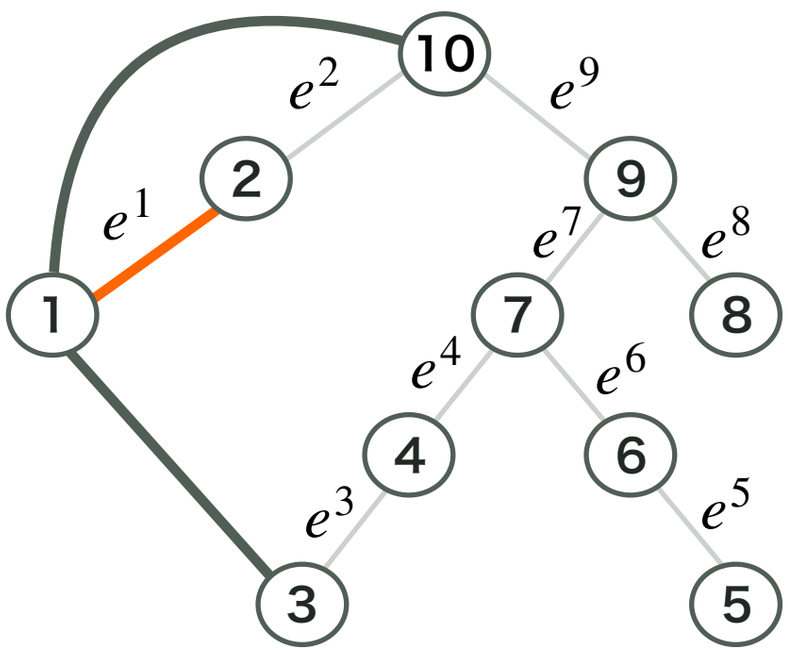
- $(w, v, v') \in Z \times V \times V$ を要素とする集合 $Q$ を $AVL$ 木を用いて管理.
  - $Q$ は今後，代替辺の候補になる辺の集合
  - $e^i$ に接続する辺を，$i$ に対して順番に処理
    - 処理: 集合 $Q$ に対して，挿入 or 削除 or 検索

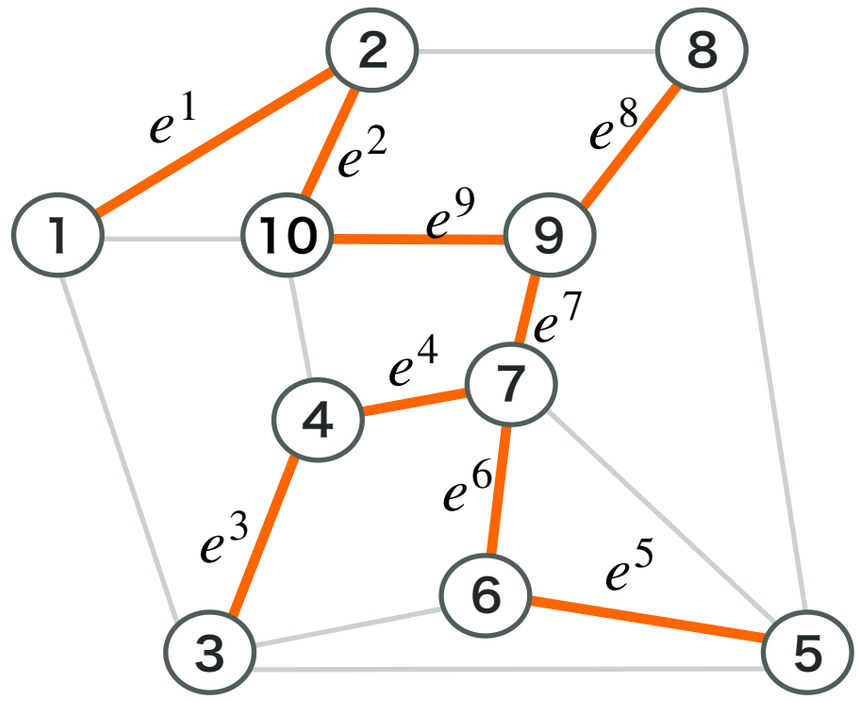


# 計算量削減のアイディア

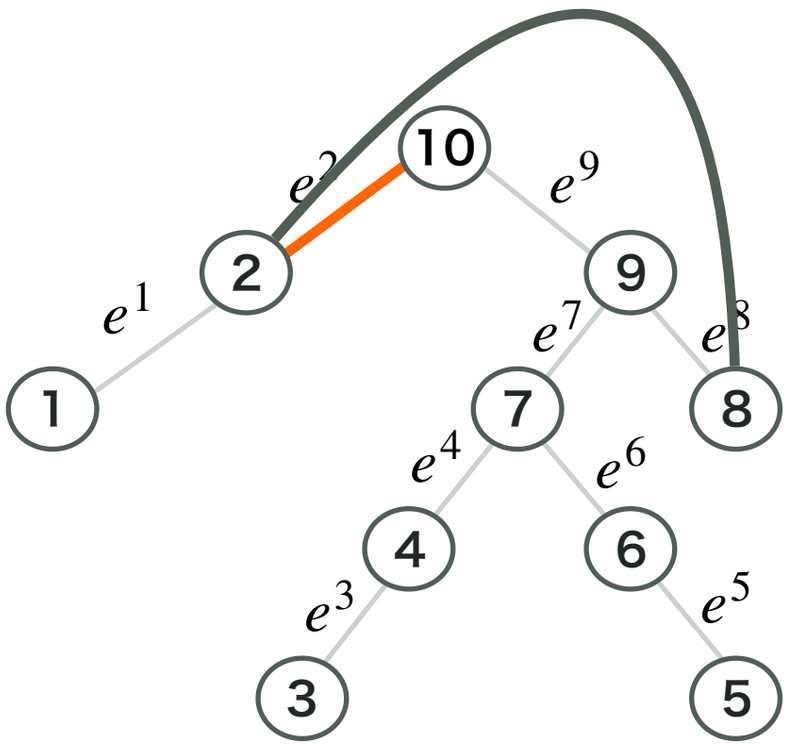
- $(w, v, v') \in Z \times V \times V$ を要素とする集合 $Q$ を $AVL$ 木を用いて管理.
  - $Q$ は今後，代替辺の候補になる辺の集合
  - $e^i$ に接続する辺を，$i$ に対して順番に処理
  - 処理: 集合 $Q$ に対して，挿入 or 削除 or 検索

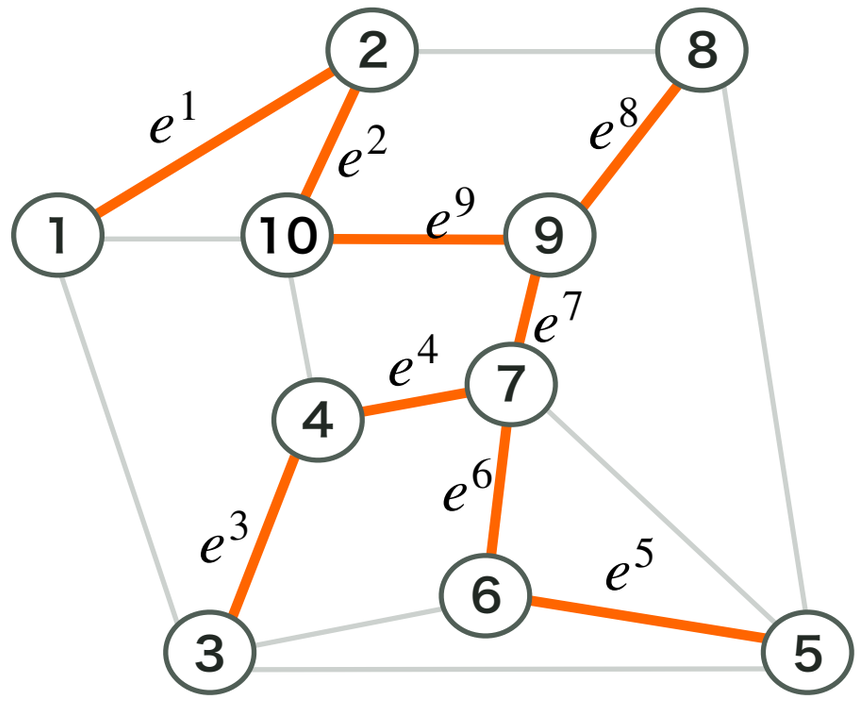


# 計算量削減のアイディア

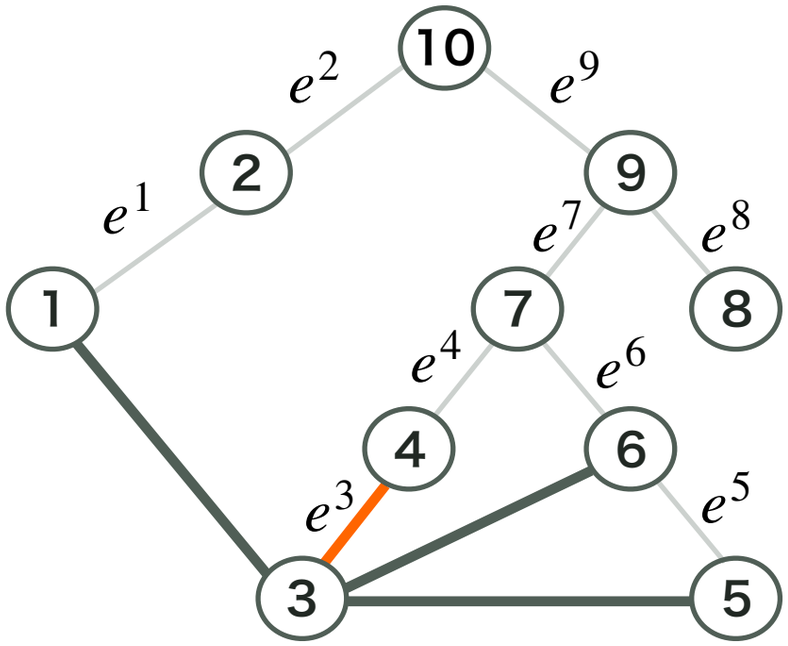
- $(w, v, v') \in Z \times V \times V$ を要素とする集合 $Q$ を $AVL$ 木を用いて管理.
  - $Q$ は今後，代替辺の候補になる辺の集合
  - $e^i$ に接続する辺を，$i$ に対して順番に処理
    - 処理: 集合 $Q$ に対して，挿入 or 削除 or 検索

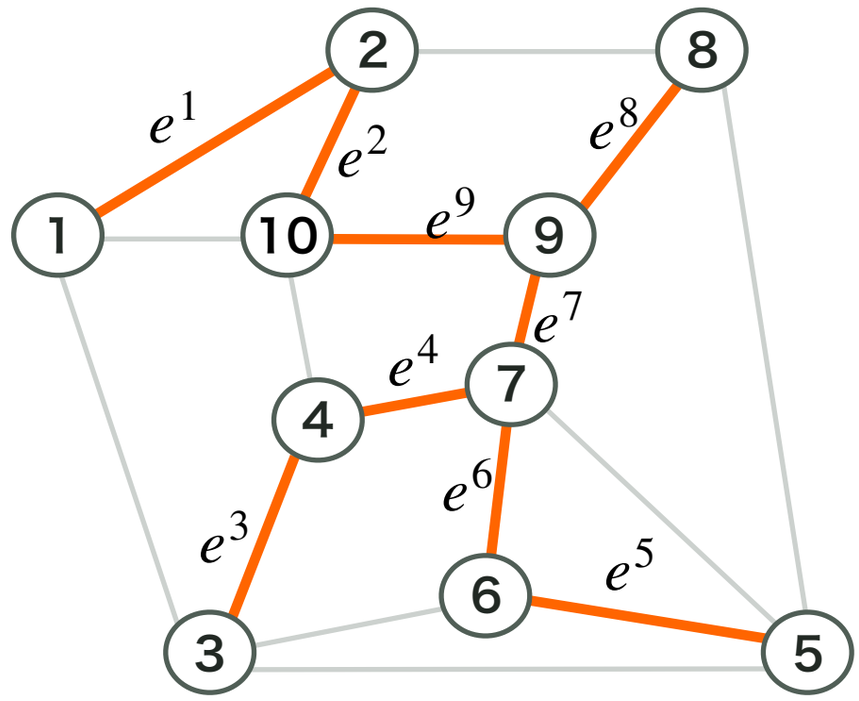


# 計算量削減のアイディア

- $(w, v, v') \in Z \times V \times V$ を要素とする集合 $Q$ を $AVL$ 木を用いて管理.
  - $Q$ は今後，代替辺の候補になる辺の集合
  - $e^i$ に接続する辺を，$i$ に対して順番に処理
  - 処理: 集合 $Q$ に対して，挿入 or 削除 or 検索



# アルゴリズムの概要

ALGORITHM $Substitute(F, R, T)$

Step 1: $Q \leftarrow \varnothing$

Step 2: for $i \in [1, n-1]$:

(1) for $\forall e = (v^i, v^j) \in E^i \backslash (R \cup T)$: // (辺の追加)

(a) if $j \in [\underline{\sigma}_i, \bar{\sigma}_i]$: $Q$ から $e$ を削除する

(b) else: $Q$ に $e$ を追加する

(2) if $e^i \notin F$: // (代替辺の検索)

(a) $w = w(e^i)$ かつ $i' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ を満たす $(w, i', j')$ を $Q$ から検索

(b) if $(w, i', j')$ が $j' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$(w, i', j')$ を $Q$ から削除し，(2-a) へ

(c) if $(w, i', j')$ が $j' \notin [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$\tilde{e}^i \leftarrow (v^{i'}, v^{j'})$

# アルゴリズムの概要

ALGORITHM $Substitute(F, R, T)$

Step 1: $Q \leftarrow \varnothing$

Step 2: for $i \in [1, n-1]$:

(1) for $\forall e = (v^i, v^j) \in E^i \backslash (R \cup T)$: // (辺の追加)

(a) if $j \in [\underline{\sigma}_i, \bar{\sigma}_i]$: $Q$ から $e$ を削除する

(b) else: $Q$ に $e$ を追加する

(2) if $e^i \notin F$: // (代替辺の検索)

(a) $w = w(e^i)$ かつ $i' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ を満たす $(w, i', j')$ を $Q$ から検索

(b) if $(w, i', j')$ が $j' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$(w, i', j')$ を $Q$ から削除し，(2-a) へ

(c) if $(w, i', j')$ が $j' \notin [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$\tilde{e}^i \leftarrow (v^{i'}, v^{j'})$

**代替辺の候補から外れる**

# アルゴリズムの概要

ALGORITHM $Substitute(F, R, T)$

Step 1: $Q \leftarrow \varnothing$

Step 2: for $i \in [1, n-1]$:

(1) for $\forall e = (v^i, v^j) \in E^i \backslash (R \cup T)$: // (辺の追加)

(a) if $j \in [\underline{\sigma}_i, \bar{\sigma}_i]$: $Q$ から $e$ を削除する

(b) else: $Q$ に $e$ を追加する

**代替辺の候補として追加**

(2) if $e^i \notin F$: // (代替辺の検索)

(a) $w = w(e^i)$ かつ $i' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ を満たす $(w, i', j')$ を $Q$ から検索

(b) if $(w, i', j')$ が $j' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$(w, i', j')$ を $Q$ から削除し，(2-a) へ

(c) if $(w, i', j')$ が $j' \notin [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$\tilde{e}^i \leftarrow (v^{i'}, v^{j'})$

# アルゴリズムの概要

ALGORITHM $Substitute(F, R, T)$

Step 1: $Q \leftarrow \varnothing$

Step 2: for $i \in [1, n-1]$:

(1) for $\forall e = (v^i, v^j) \in E^i \backslash (R \cup T)$: // (辺の追加)

(a) if $j \in [\underline{\sigma}_i, \bar{\sigma}_i]$: $Q$ から $e$ を削除する

(b) else: $Q$ に $e$ を追加する

(2) if $e^i \notin F$: // (代替辺の検索)

(a) $w = w(e^i)$ かつ $i' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ を満たす $(w, i', j')$ を $Q$ から検索

(b) if $(w, i', j')$ が $j' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$(w, i', j')$ を $Q$ から削除し，(2-a) へ

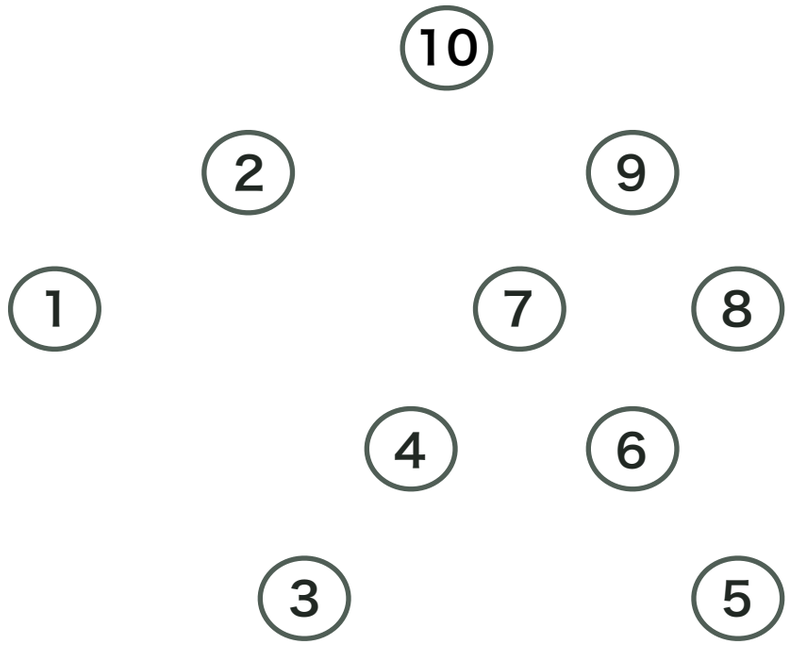
(c) if $(w, i', j')$ が $j' \notin [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

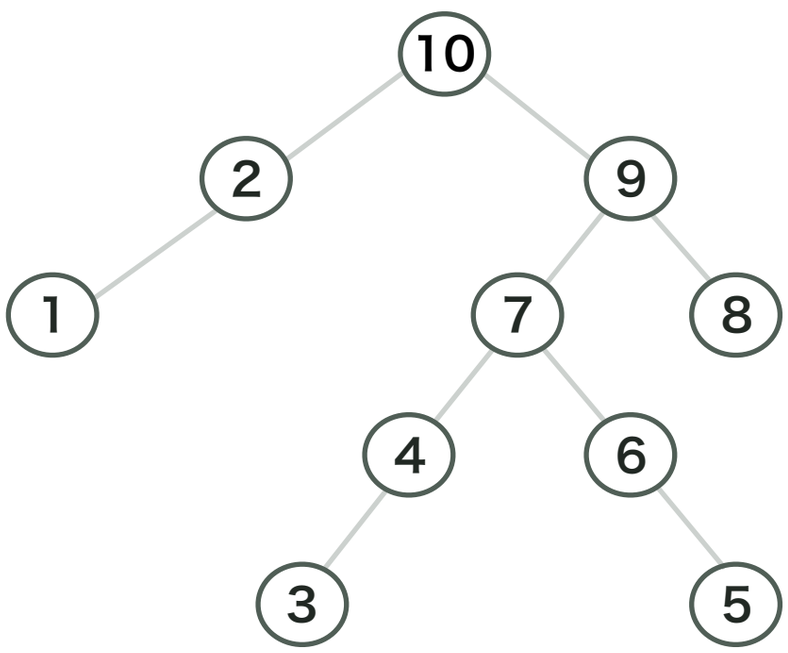
$\tilde{e}^i \leftarrow (v^{i'}, v^{j'})$

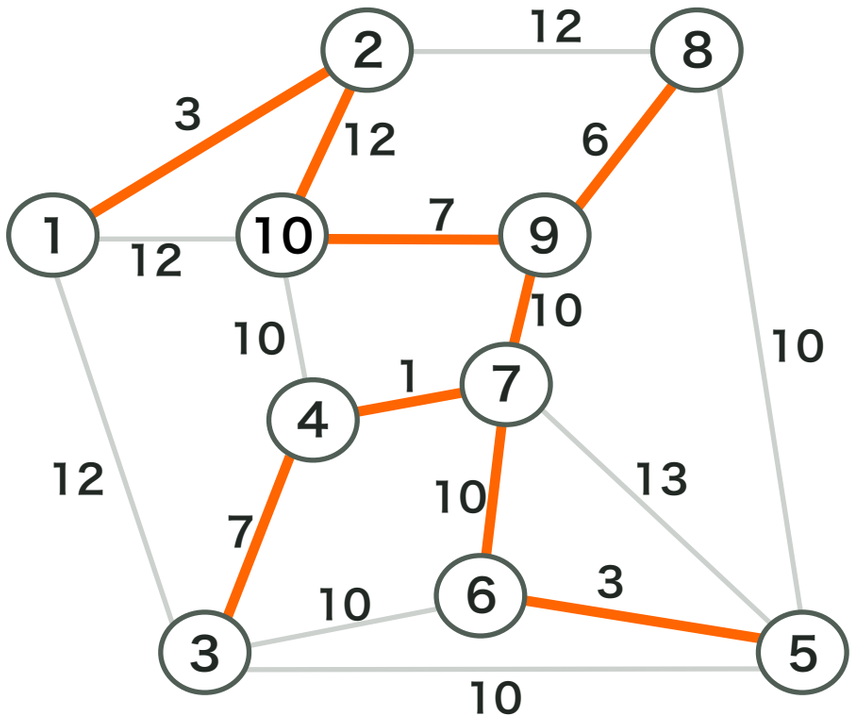
**代替辺が見つかる**

# 代替辺を順に構築

$Q$ に含まれる辺を
図示する

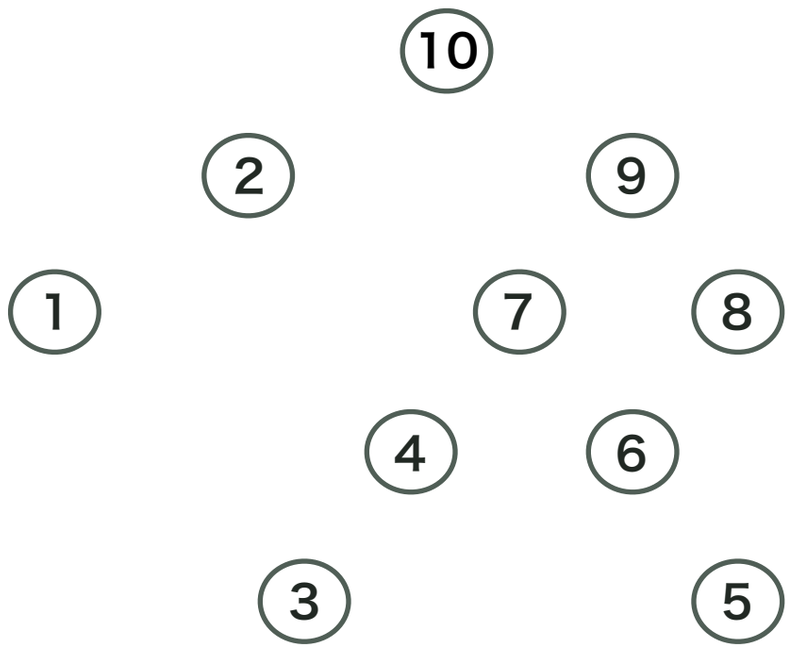


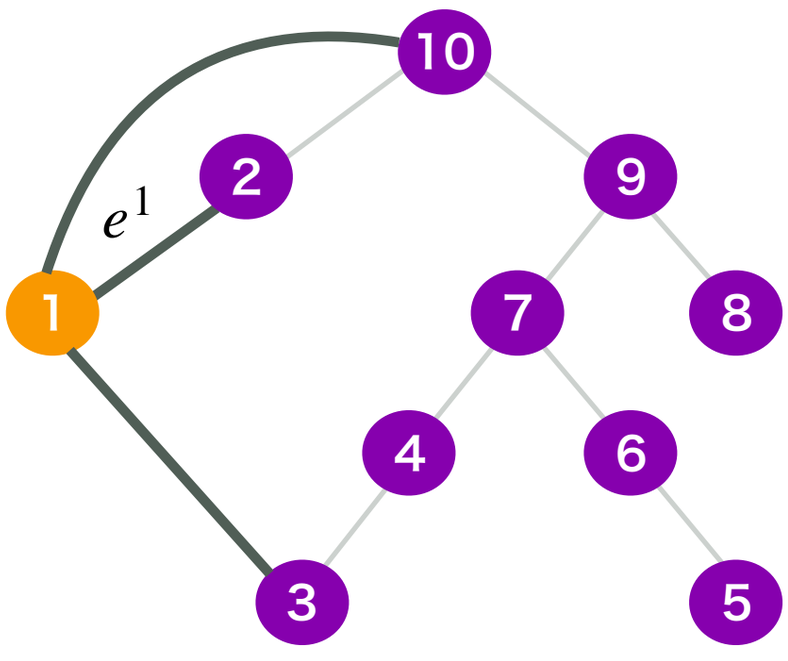



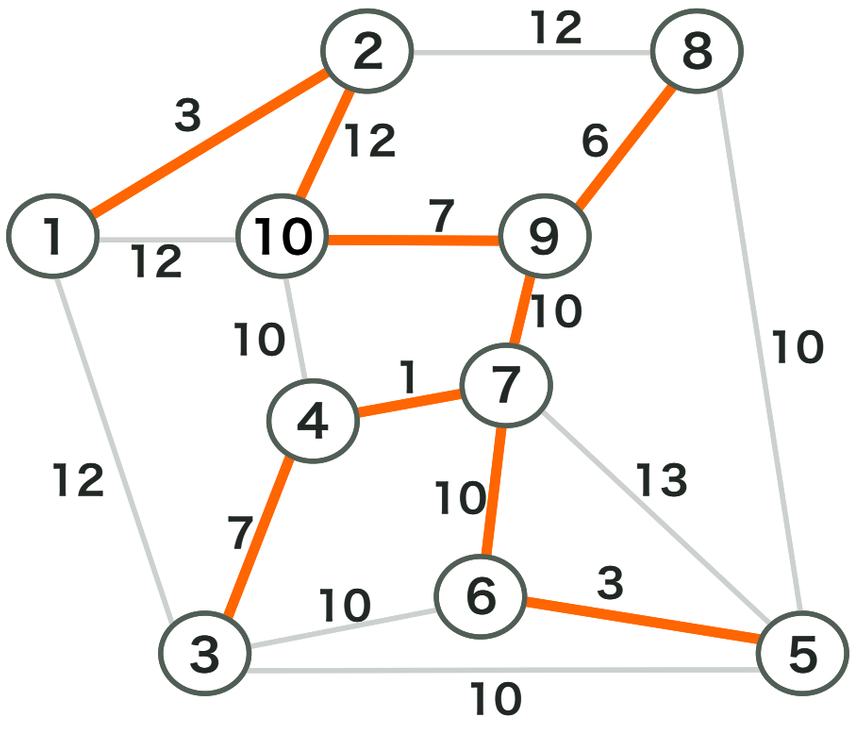



# 代替辺を順に構築

辺 $e^1$ に接続する辺を見る

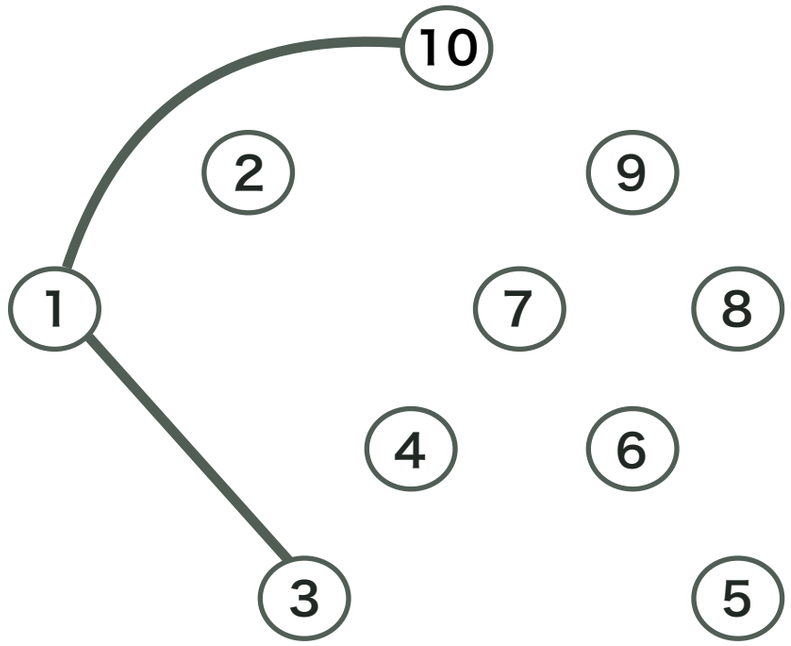


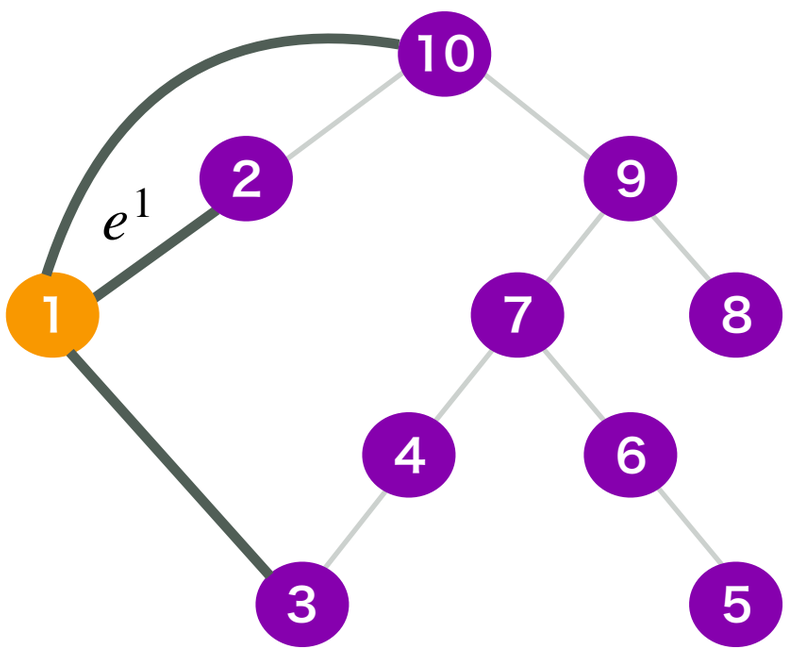



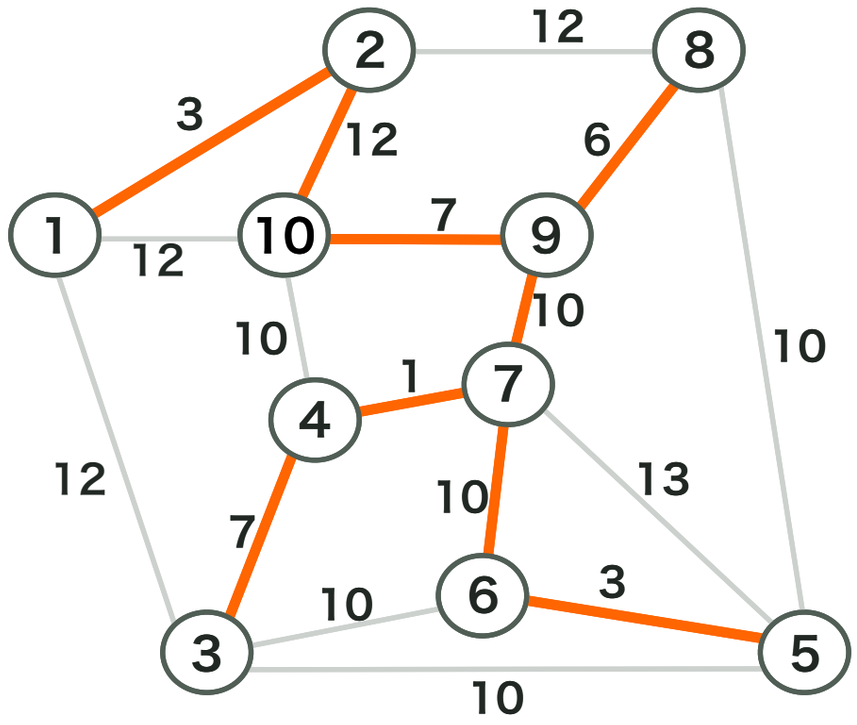



# 代替辺を順に構築

２ つの辺を<u>候補として追加</u>



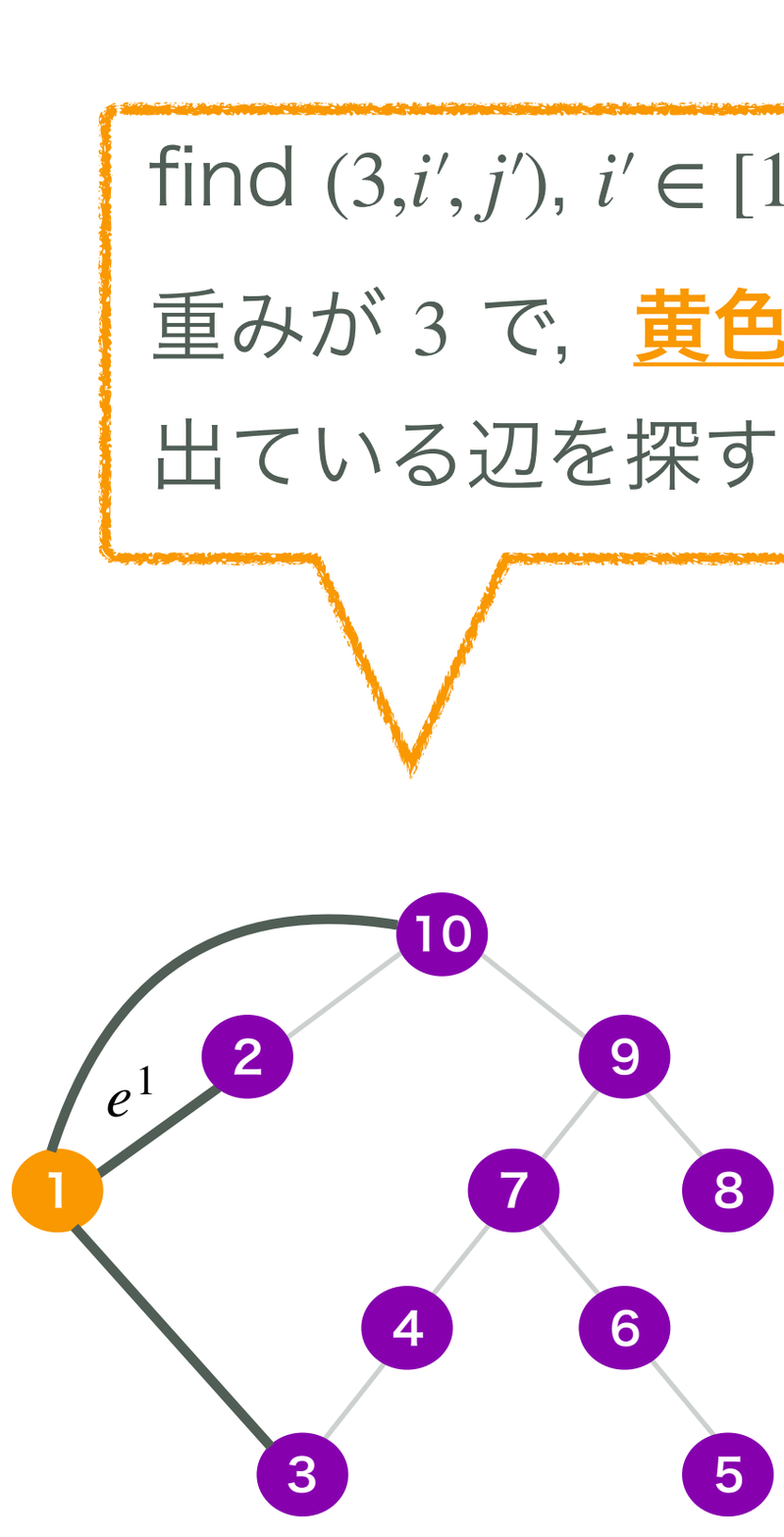



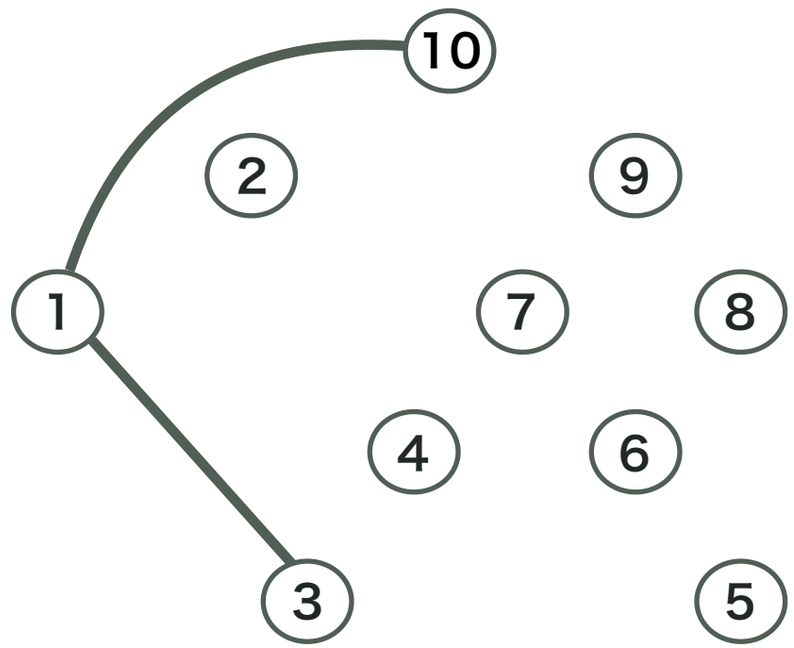



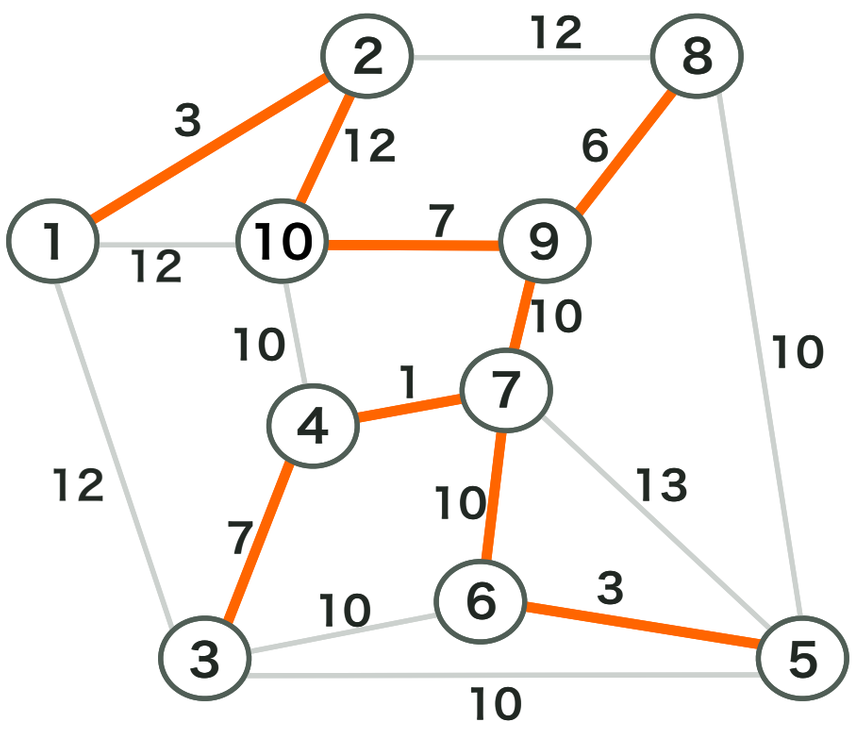
# 代替辺を順に構築

find $(3, i', j')$, $i' \in [1,1]$

重みが 3 で，**黄色**の頂点から
出ている辺を探す



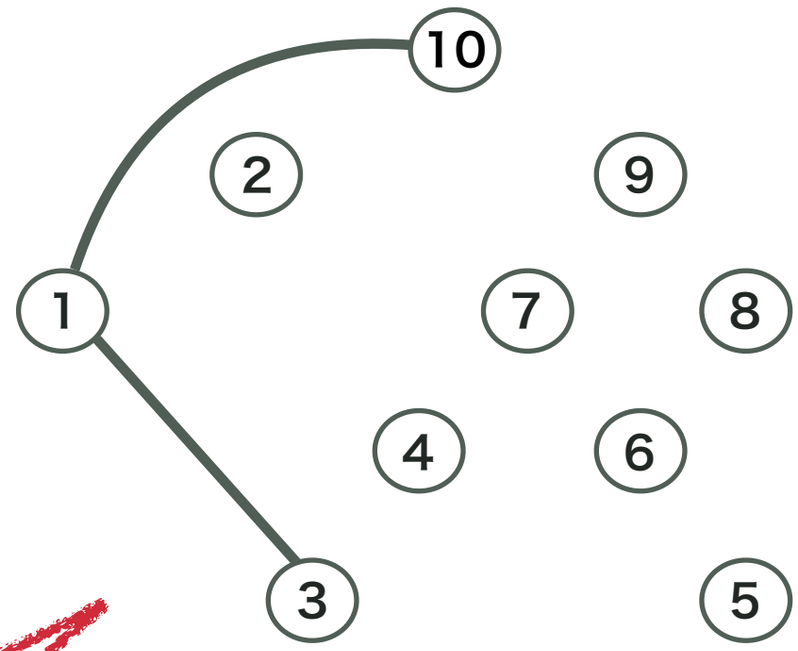



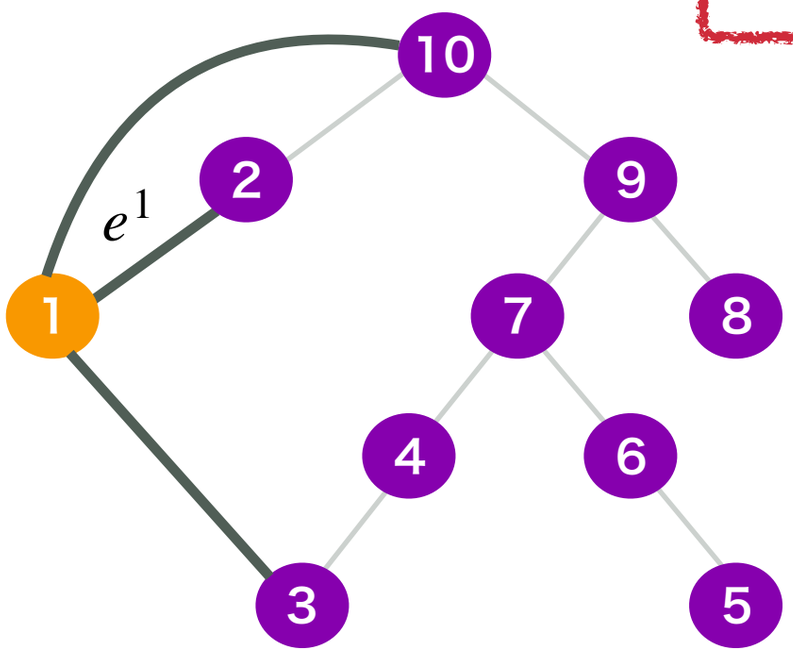
# 代替辺を順に構築

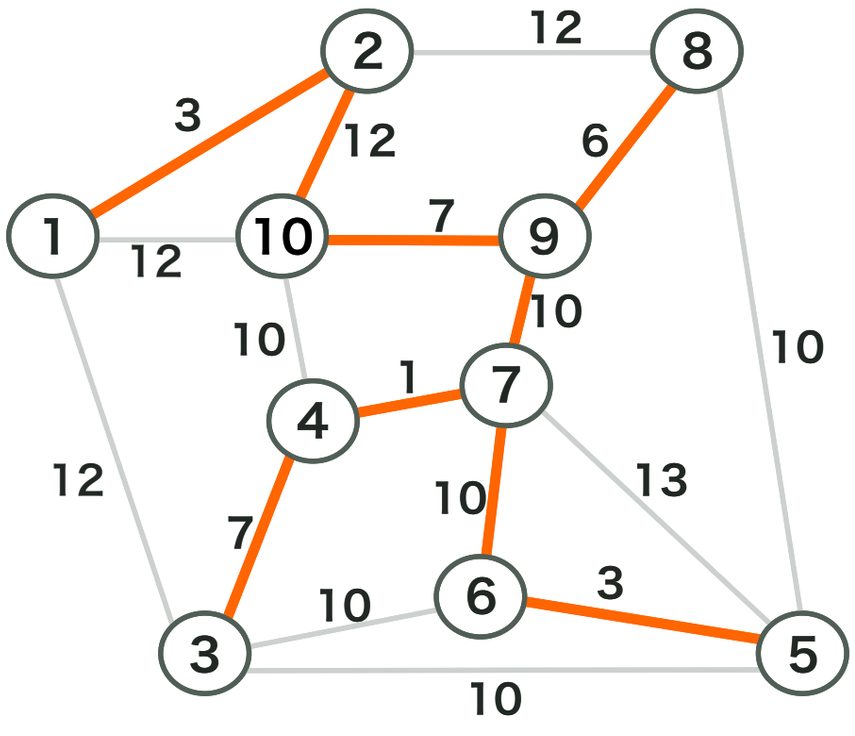
find $(3, i', j')$, $i' \in [1,1]$

重みが 3 で，**黄色**の頂点から
出ている辺を探す

が，見つからず...

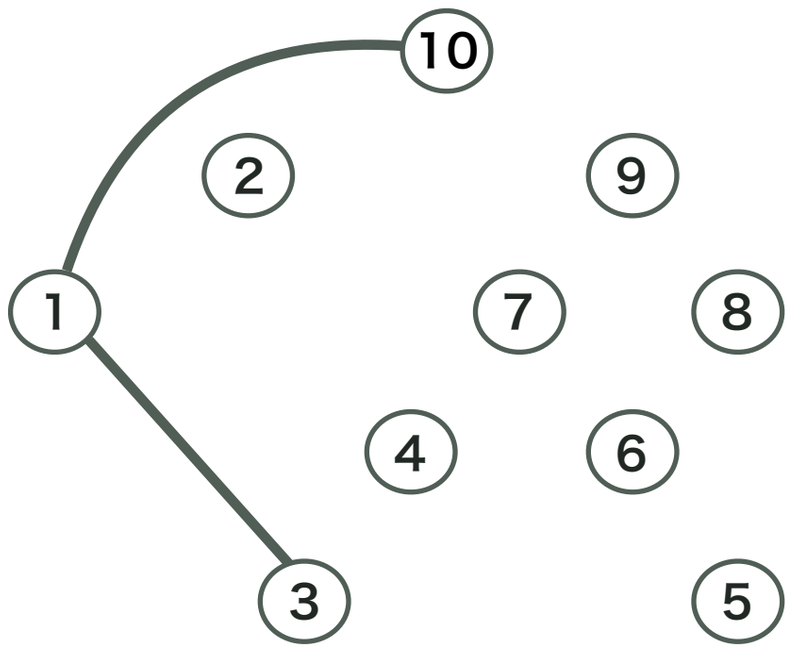


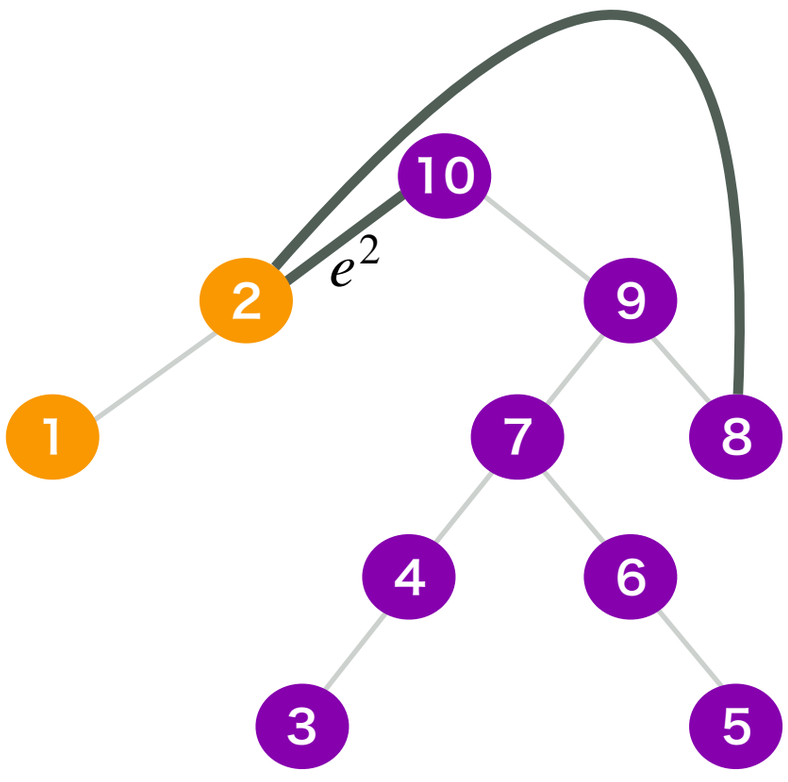



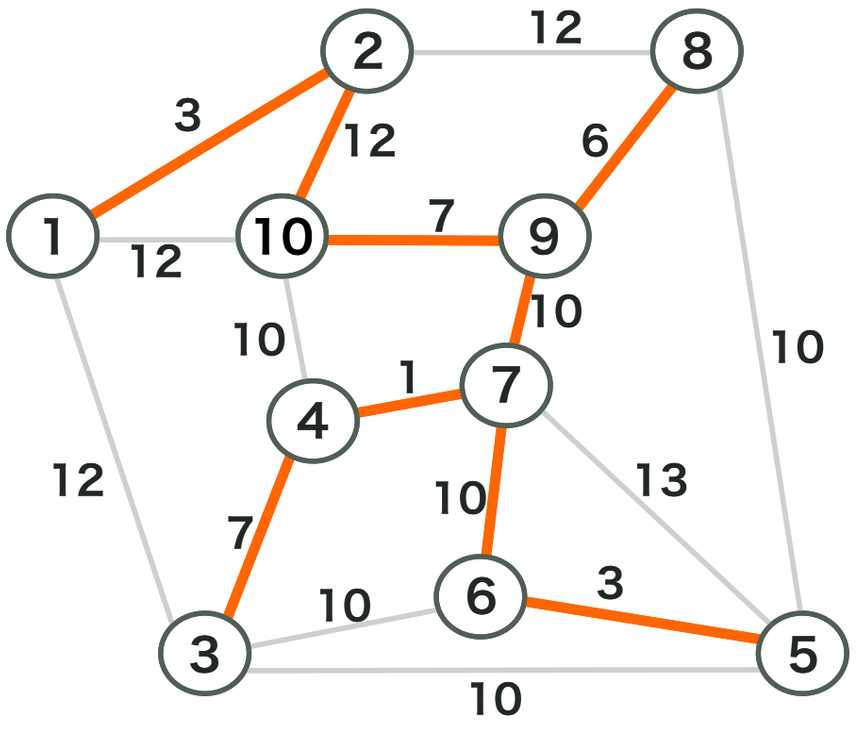



# 代替辺を順に構築

辺 $e^2$ に接続する辺を見る

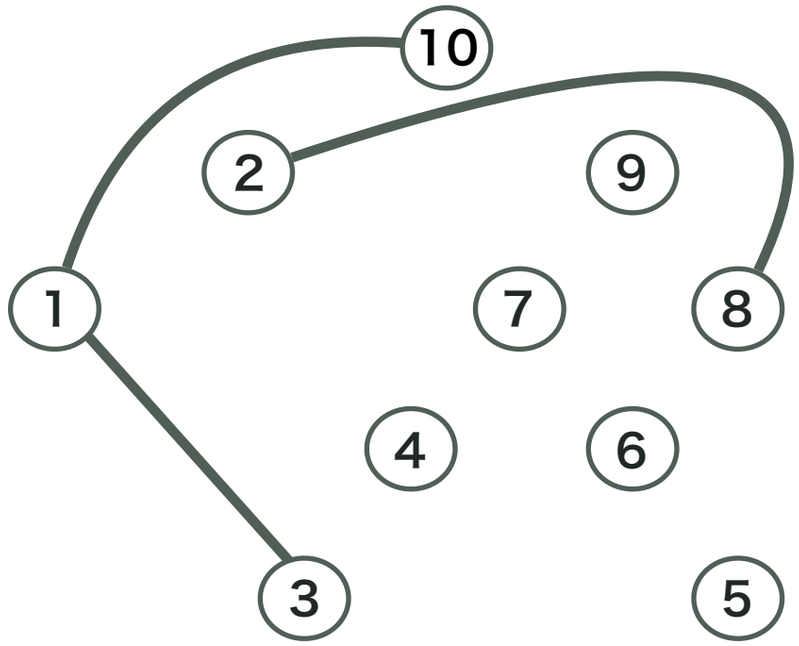


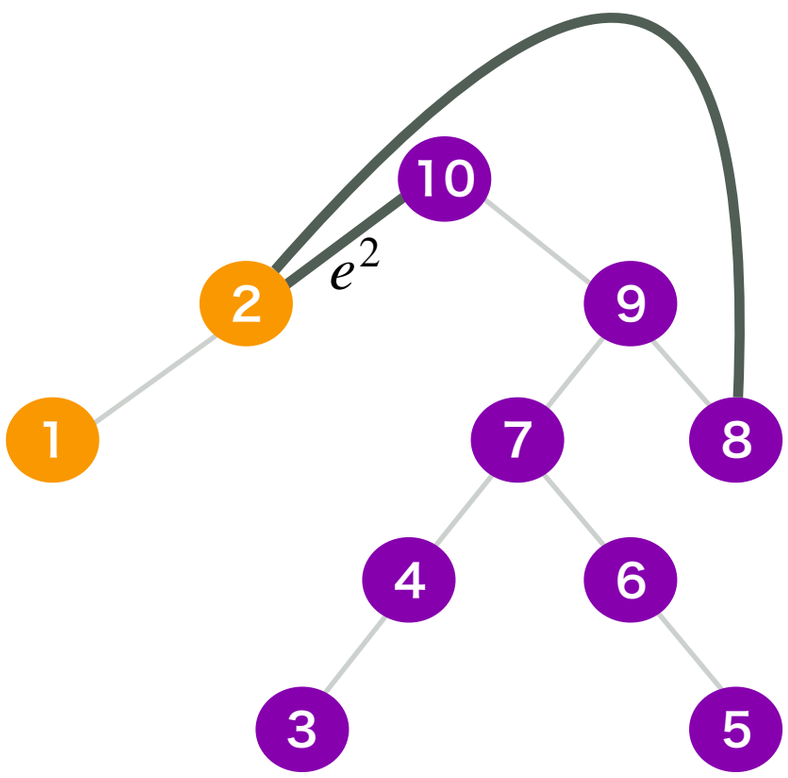



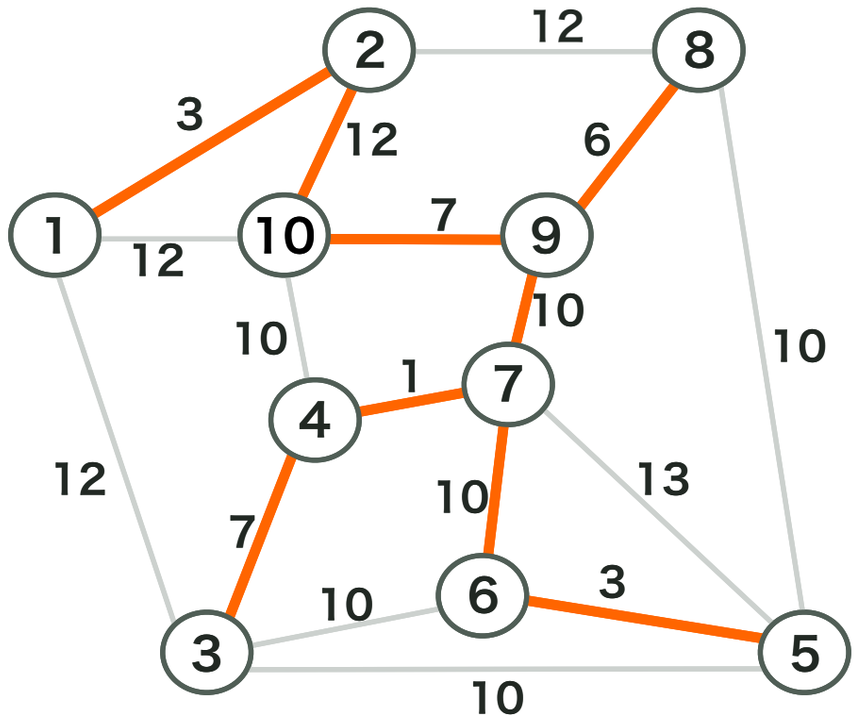



# 代替辺を順に構築

1 つの辺を**候補として追加**



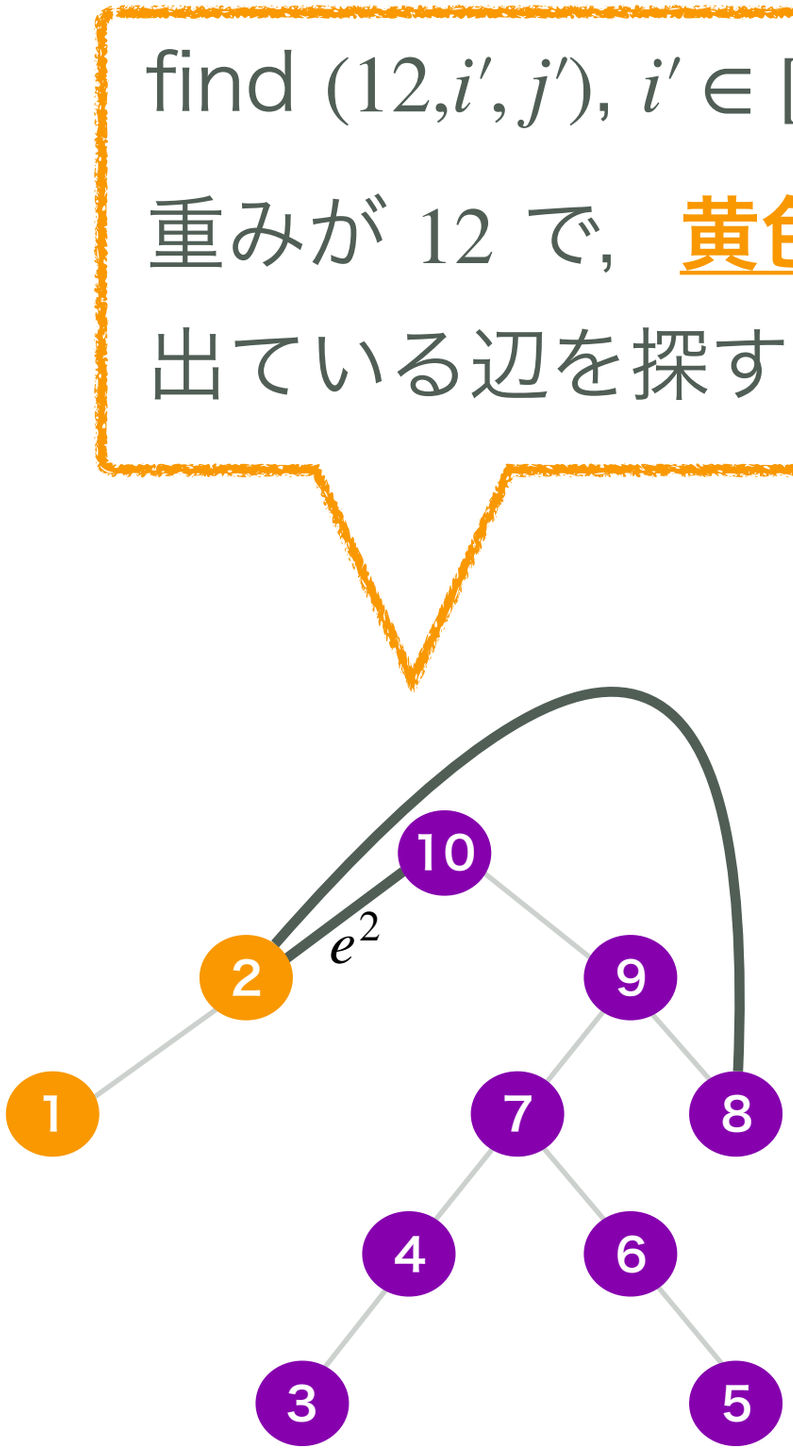



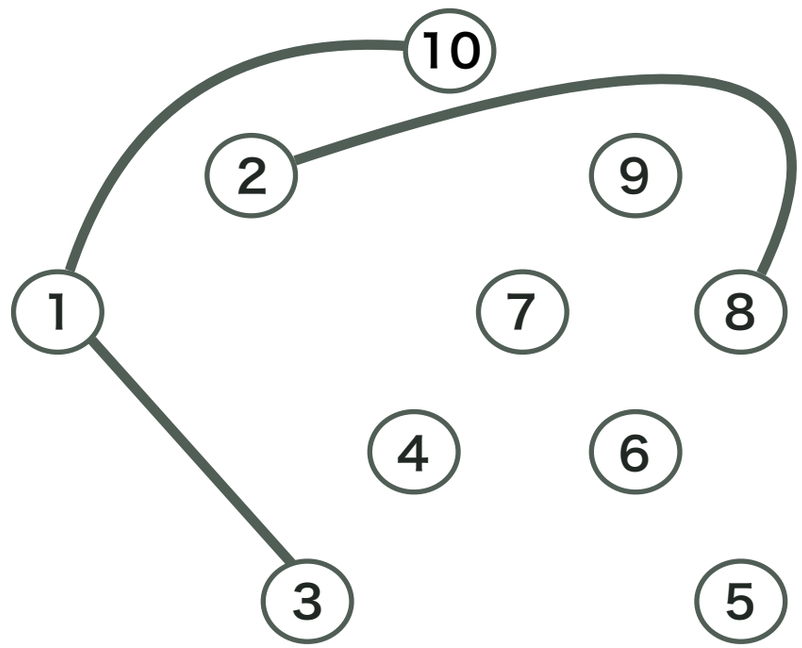



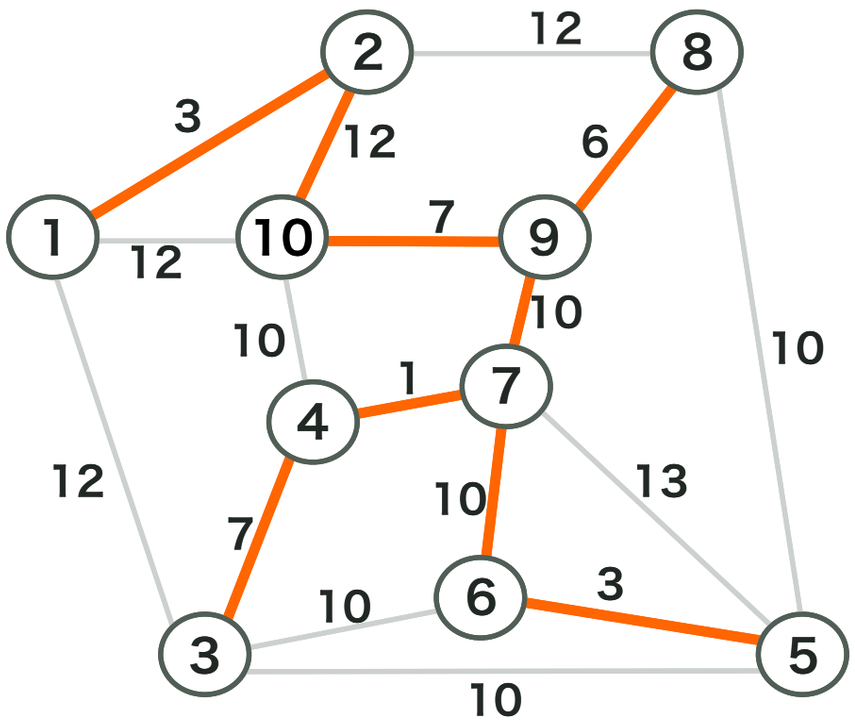
# 代替辺を順に構築

find $(12, i', j')$, $i' \in [1,2]$

重みが 12 で，**黄色**の頂点から出ている辺を探す



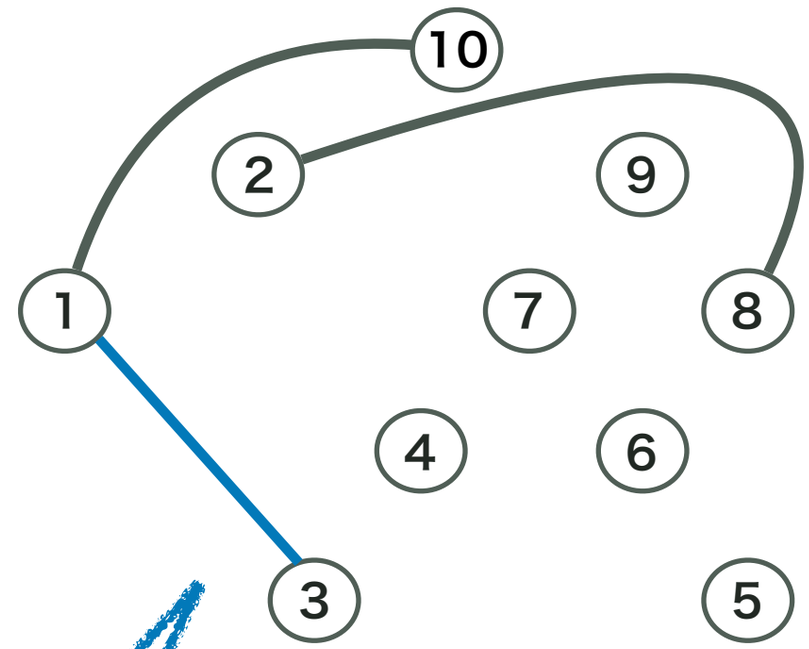



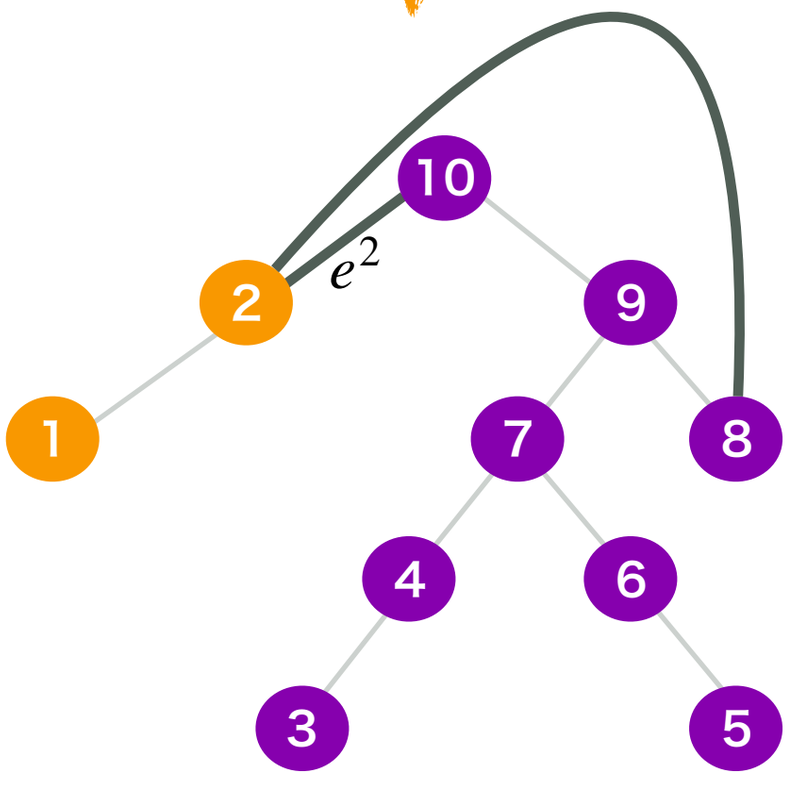
# 代替辺を順に構築

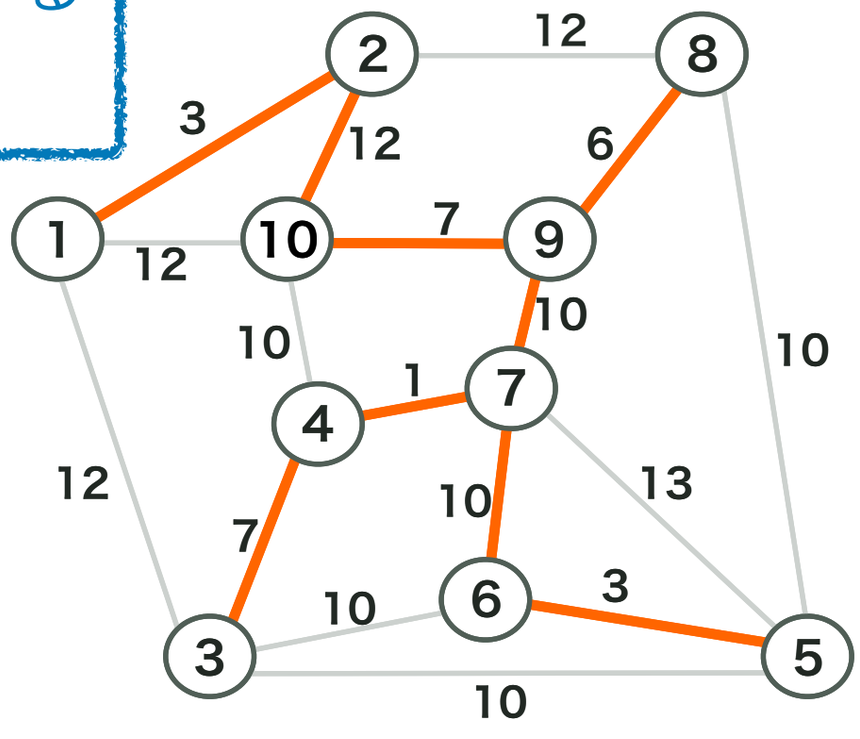
find $(12, i', j')$, $i' \in [1,2]$

重みが 12 で，**黄色**の頂点から出ている辺を探す



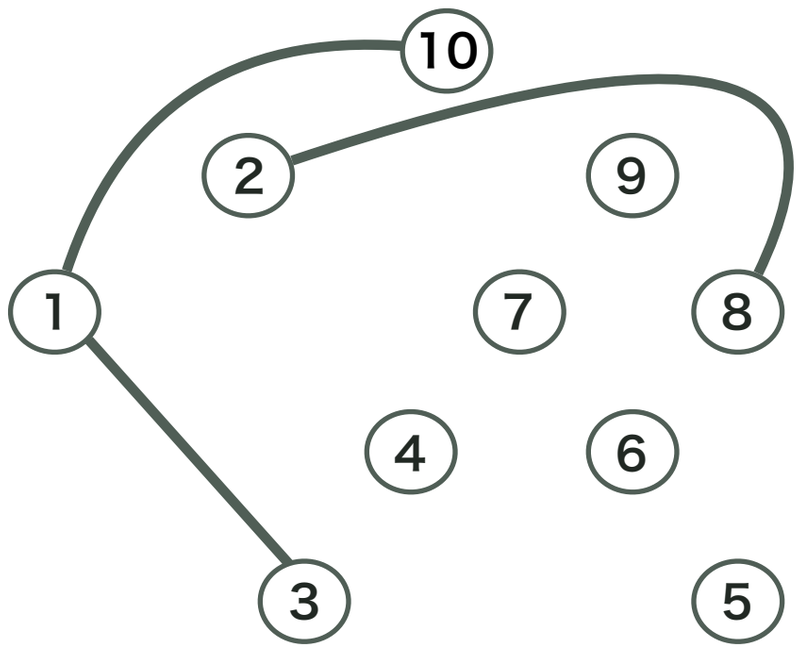

(12,1,3) が見つかる

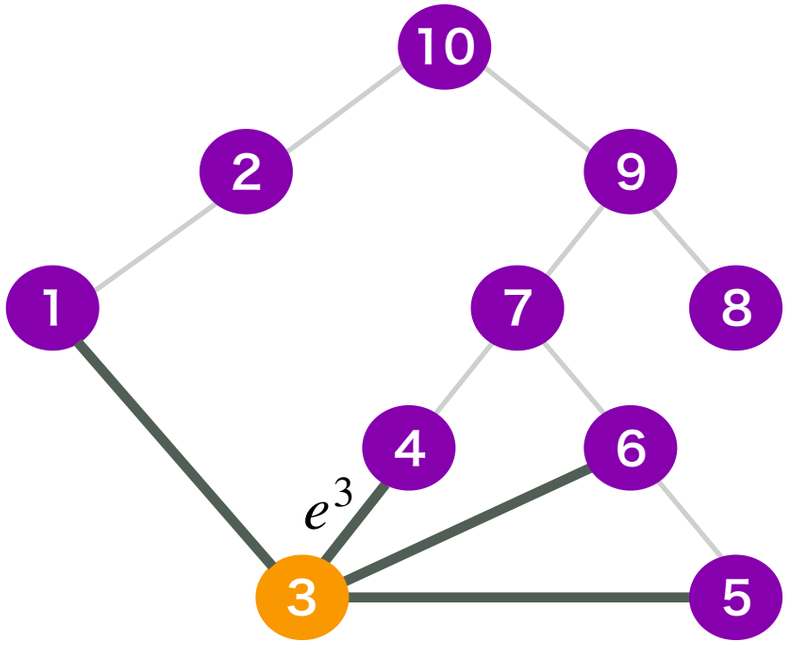


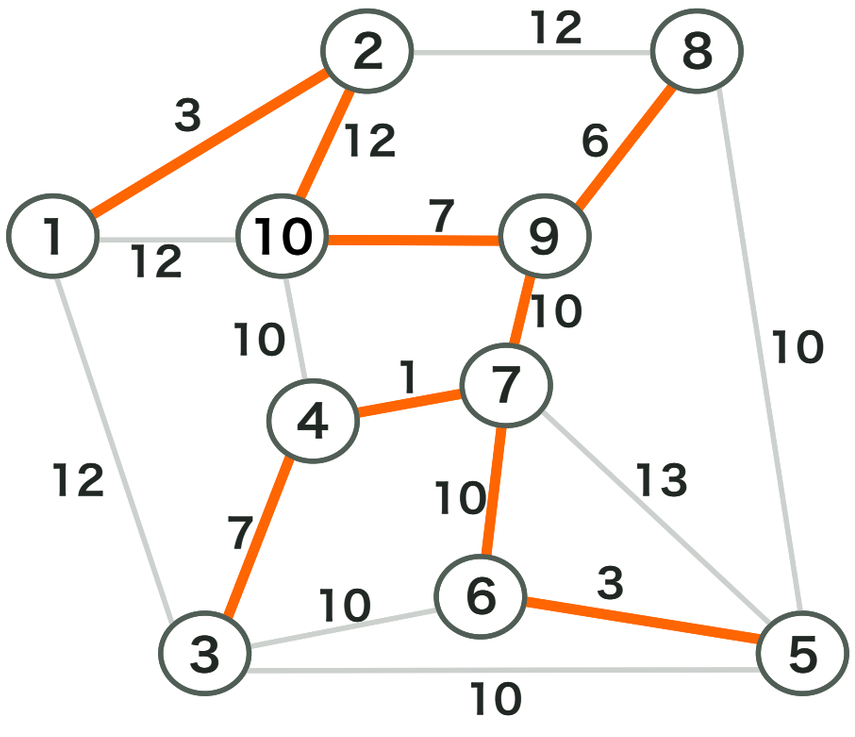



# 代替辺を順に構築

辺 $e^3$ に接続する辺を見る

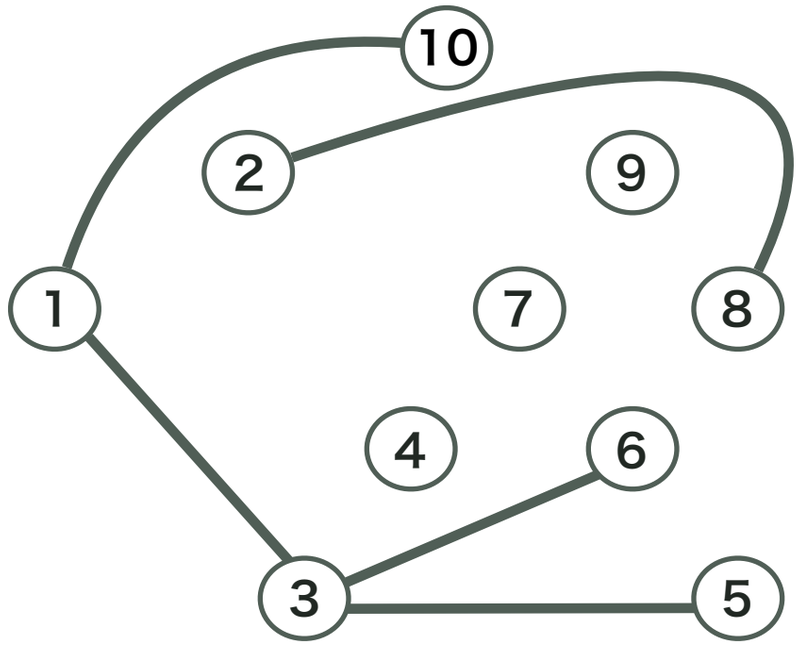


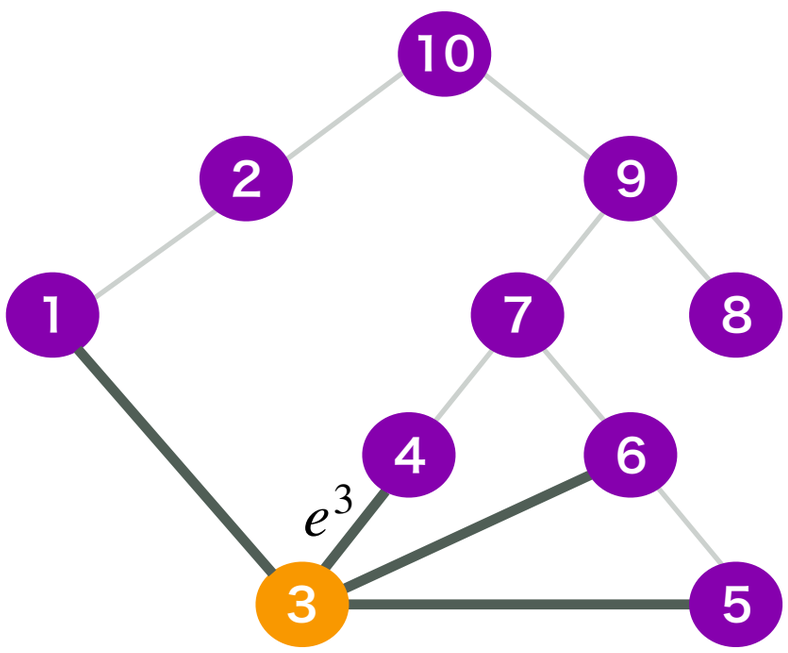



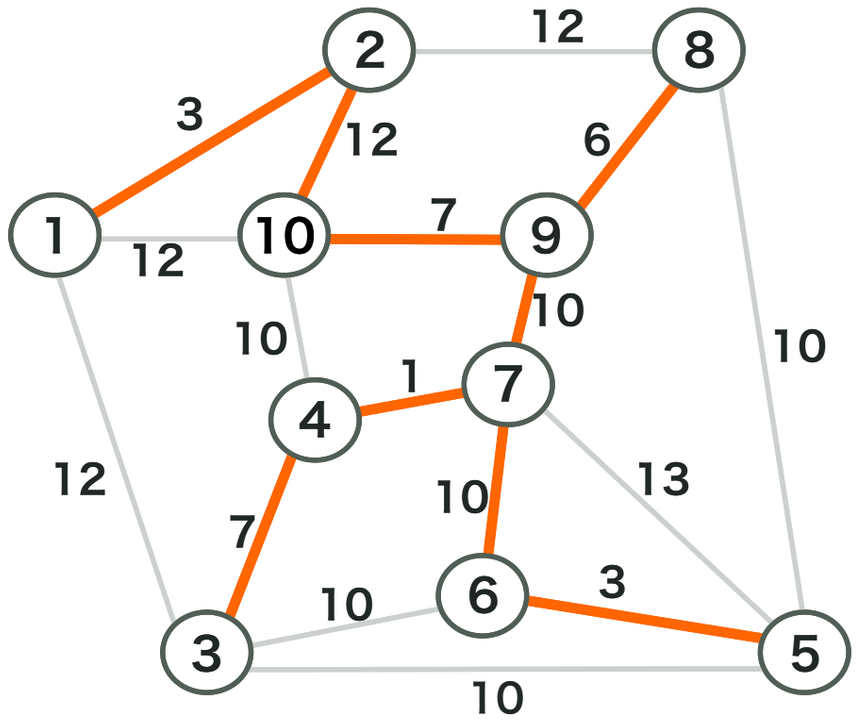



# 代替辺を順に構築

２ つの辺を<u>候補として追加</u>



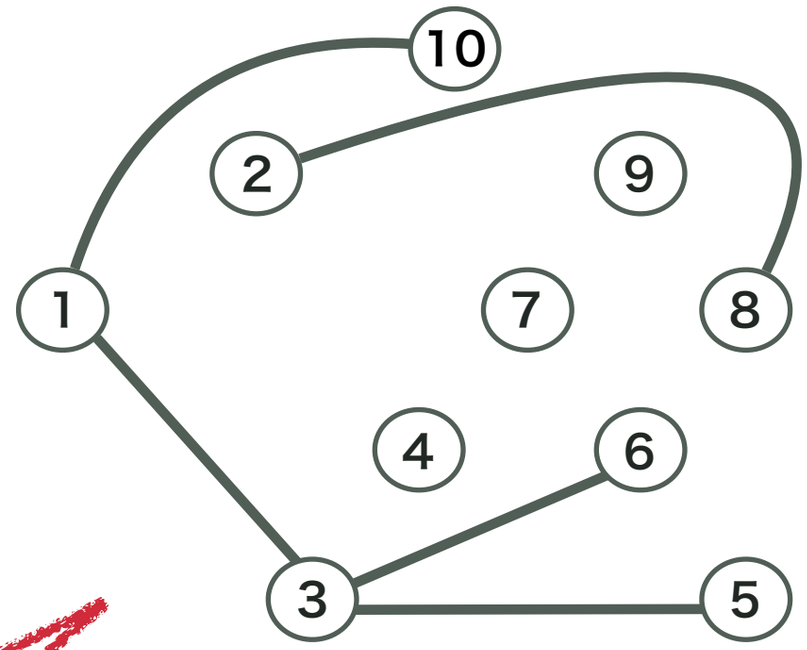



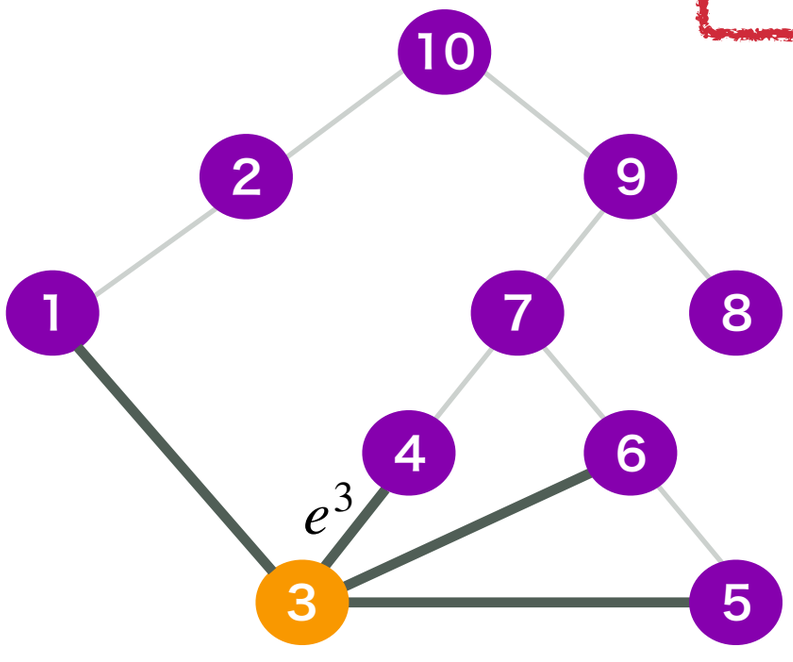
# 代替辺を順に構築

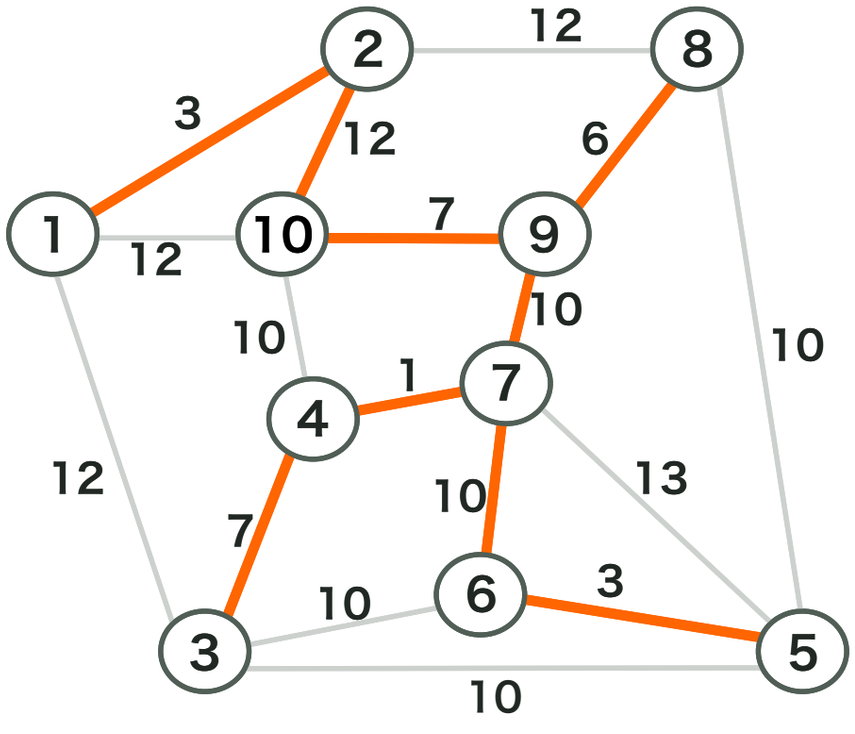
find $(7, i', j')$, $i' \in [3,3]$

重みが 3 で，**黄色**の頂点から出ている辺を探す

が，見つからず...

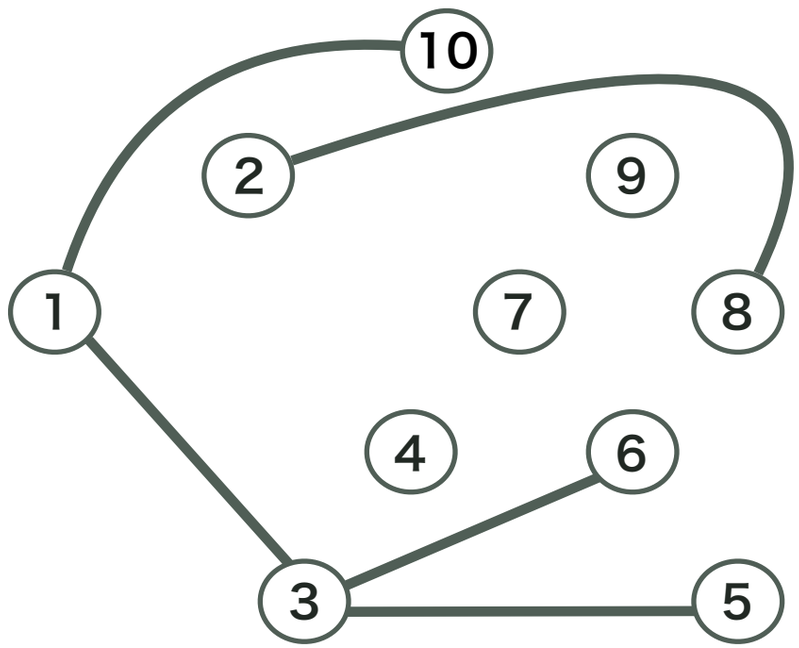


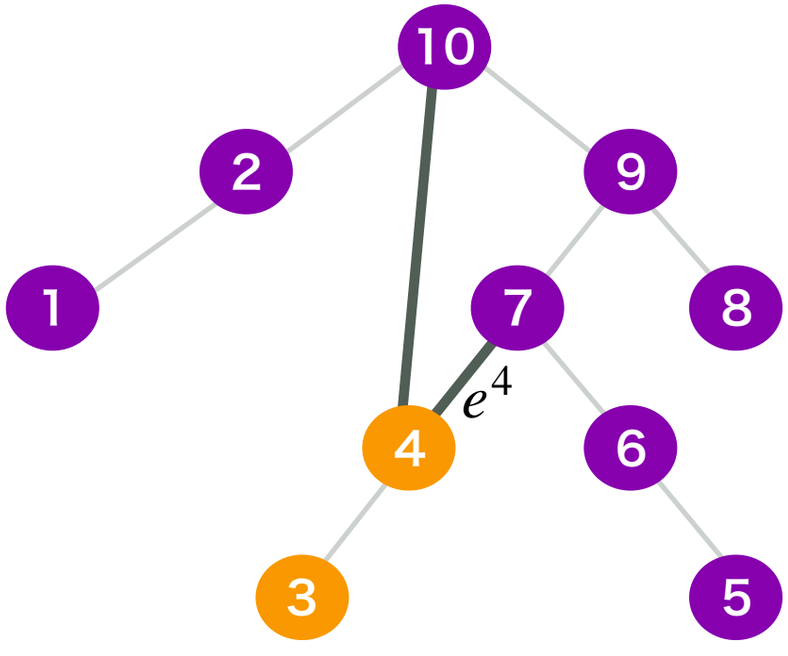



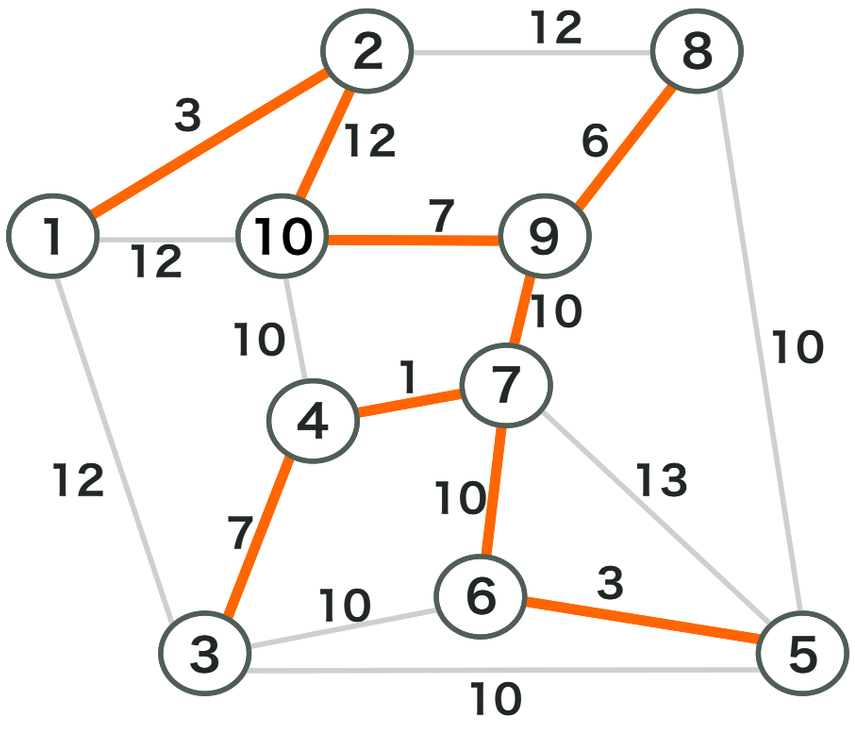



# 代替辺を順に構築

辺 $e^4$ に接続する辺を見る

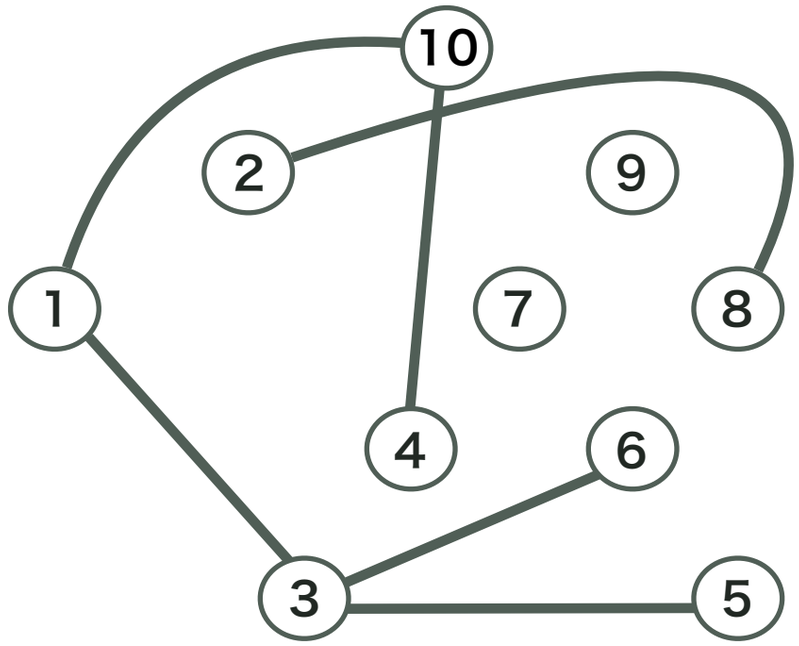


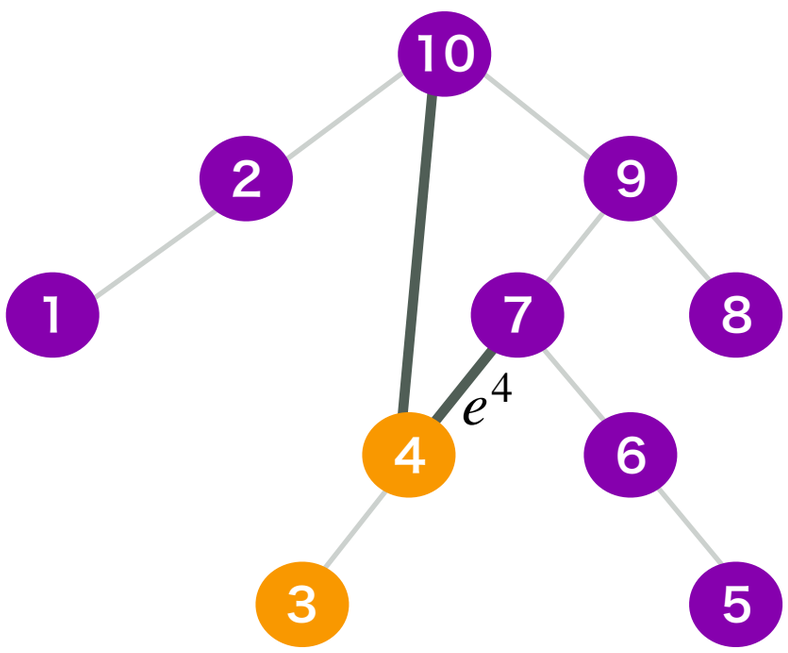



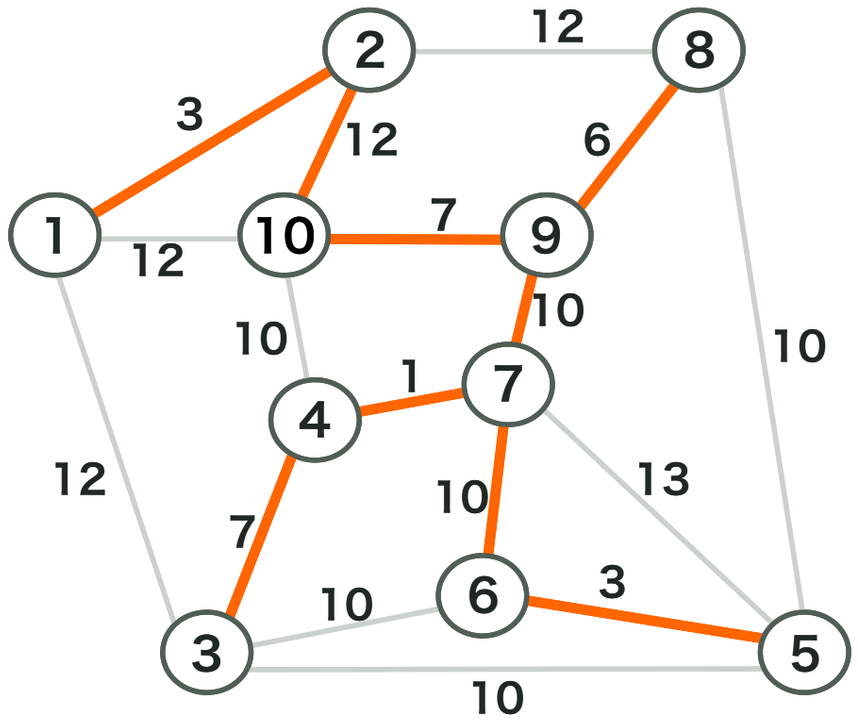



# 代替辺を順に構築

1 つの辺を<u>候補として追加</u>



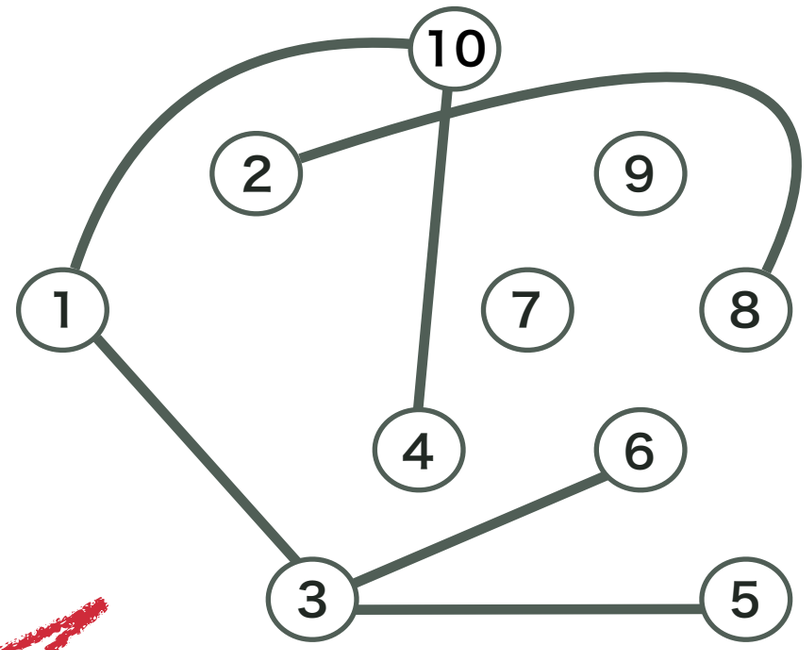



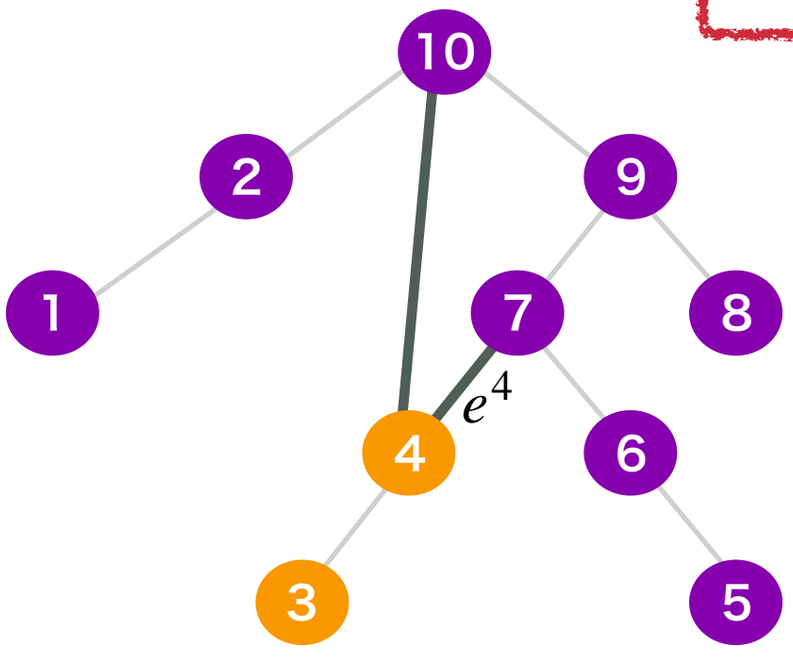
# 代替辺を順に構築

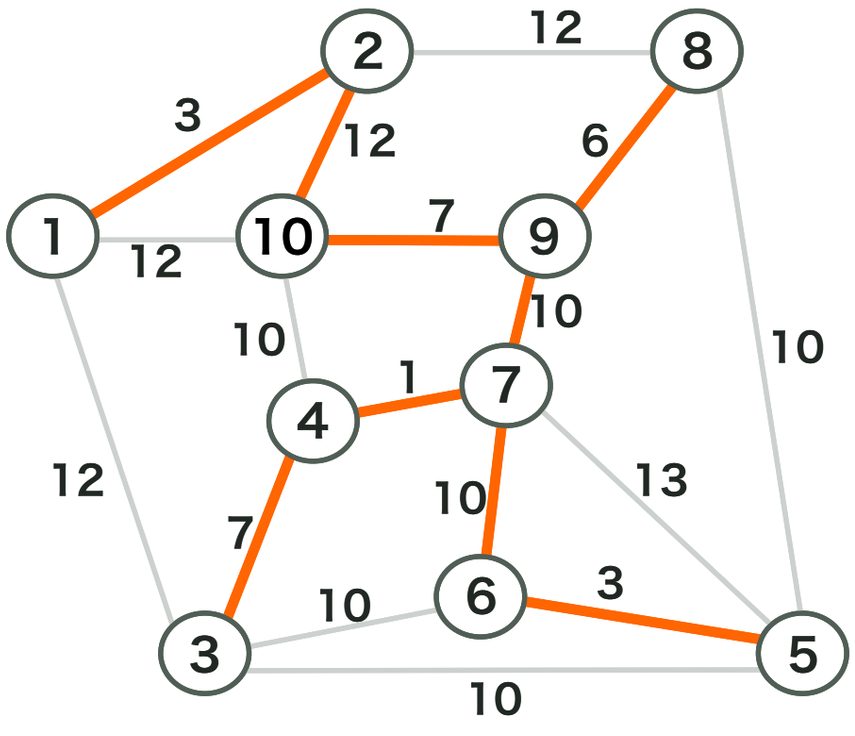
find $(1, i', j')$, $i' \in [3,4]$

重みが 1 で，**黄色**の頂点から
出ている辺を探す

が，見つからず...



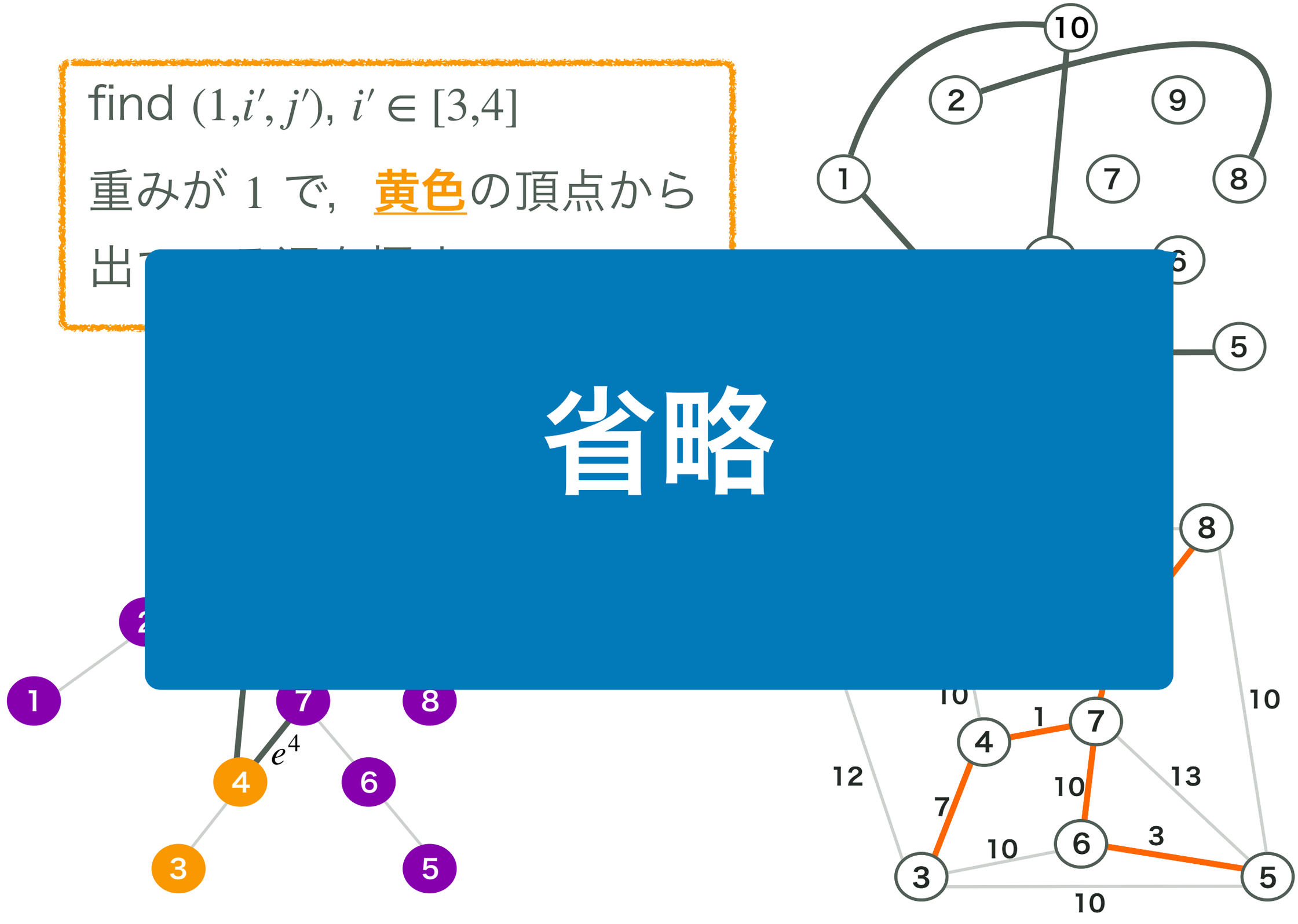
# 代替辺を順に構築

find $(1, i', j')$, $i' \in [3,4]$

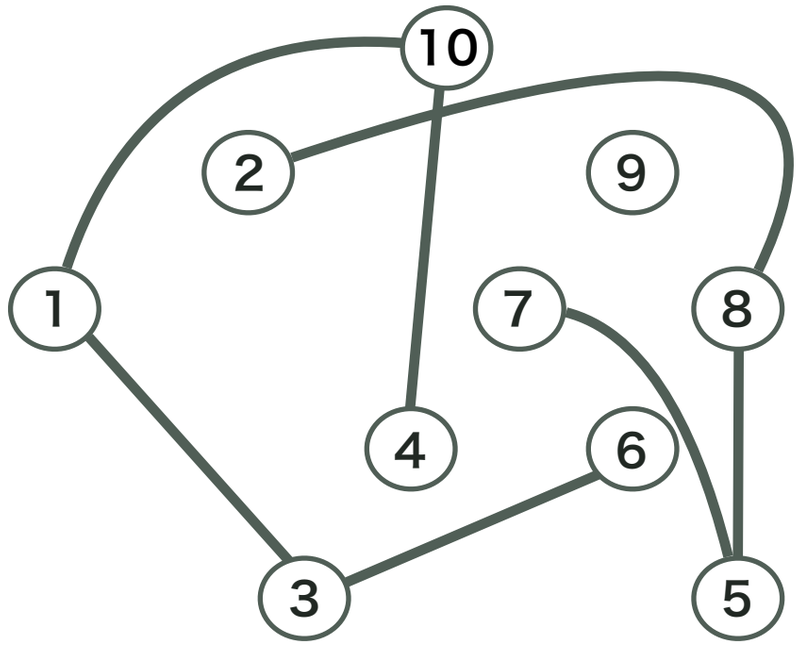
重みが 1 で，**黄色**の頂点から

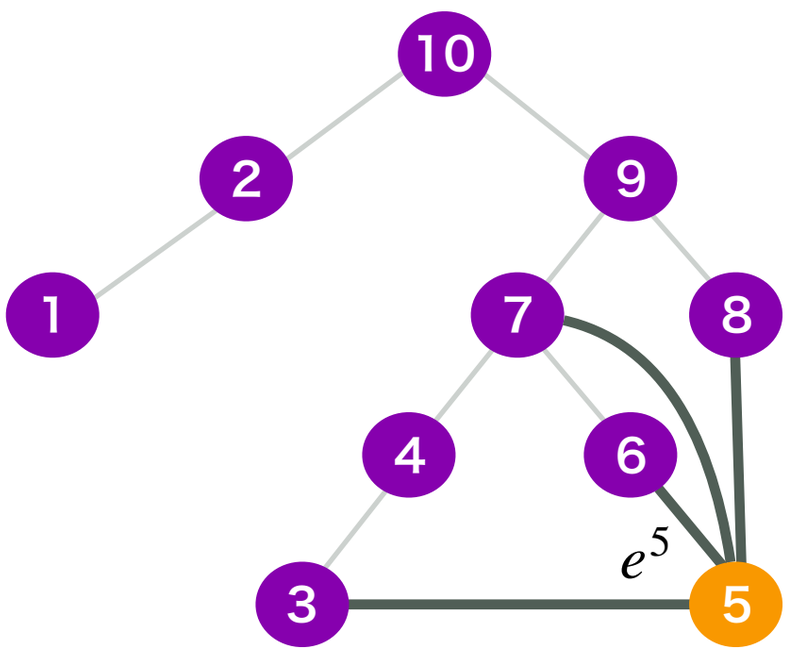
出

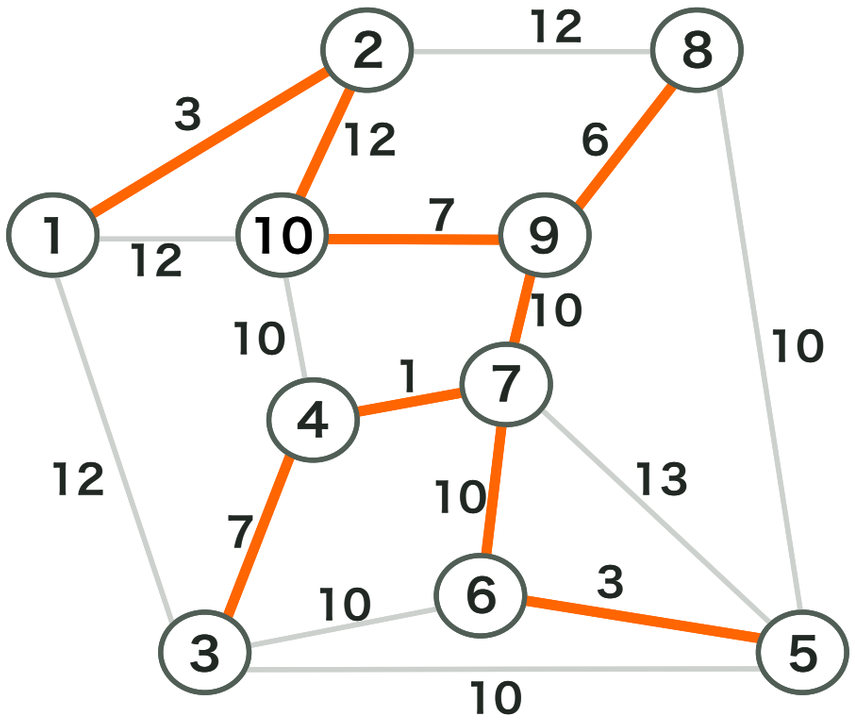
**省略**



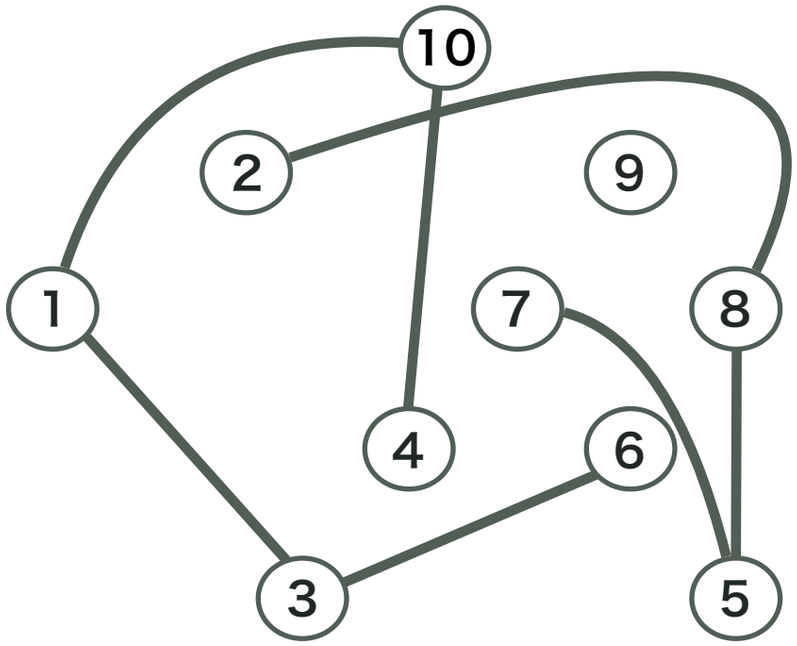
# 代替辺を順に構築

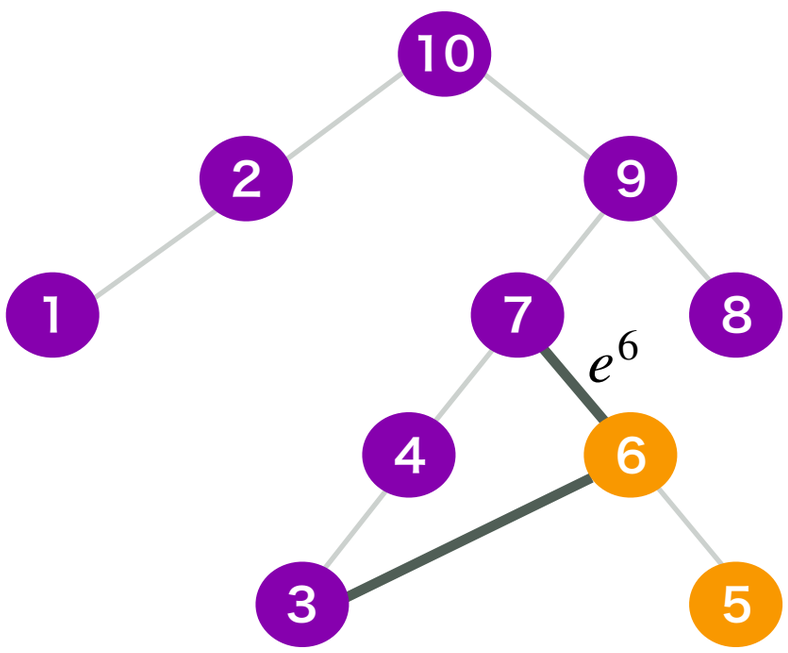

10
2
9
1
7
8
4
6
3
5

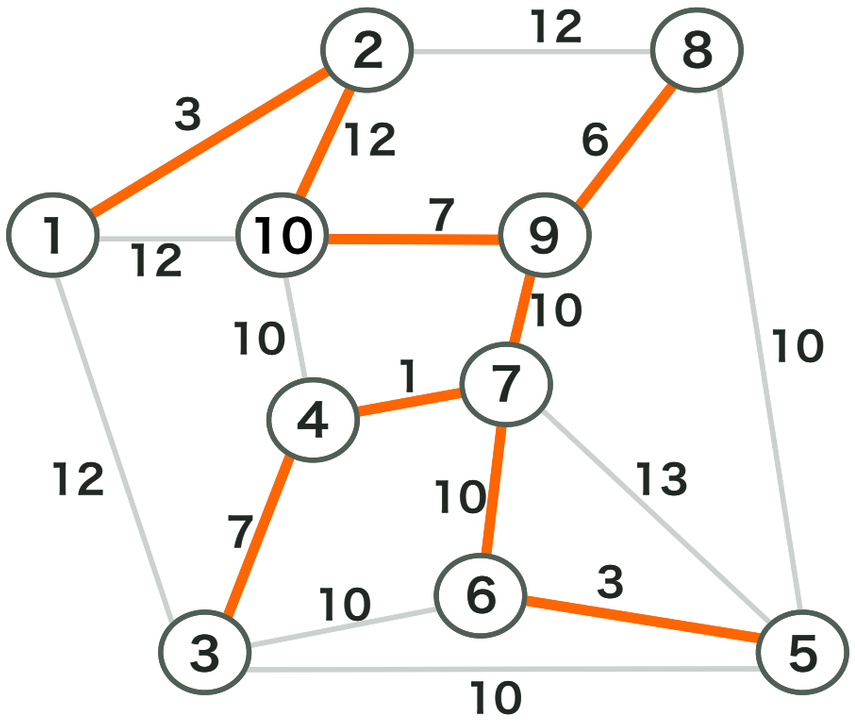

10
2
9
1
7
8
4
6
$e^5$
3
5


2
12
8
3
12
6
1
10
7
9
12
10
10
10
1
7
4
12
13
10
7
6
3
10
3
5
10

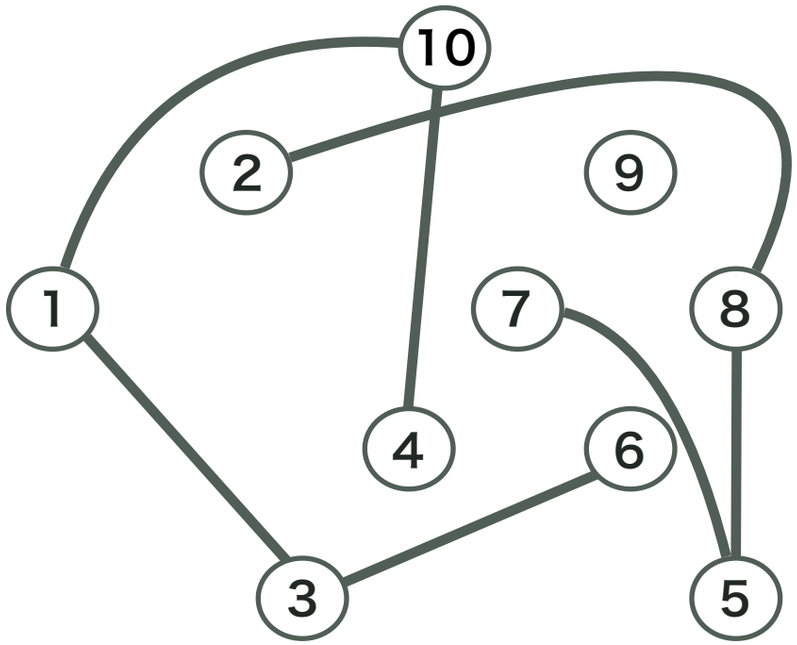
# 代替辺を順に構築

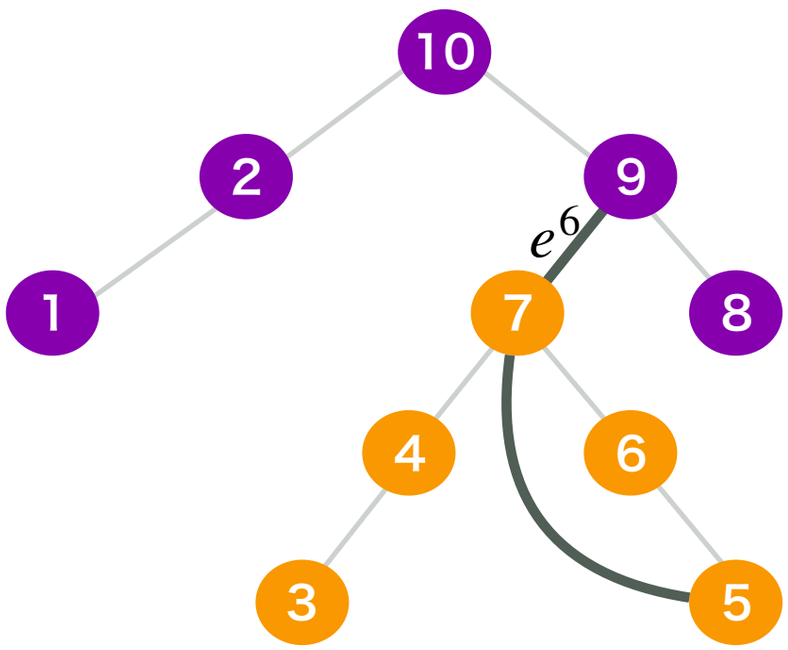

10
2
9
1
7
8
4
6
3
5

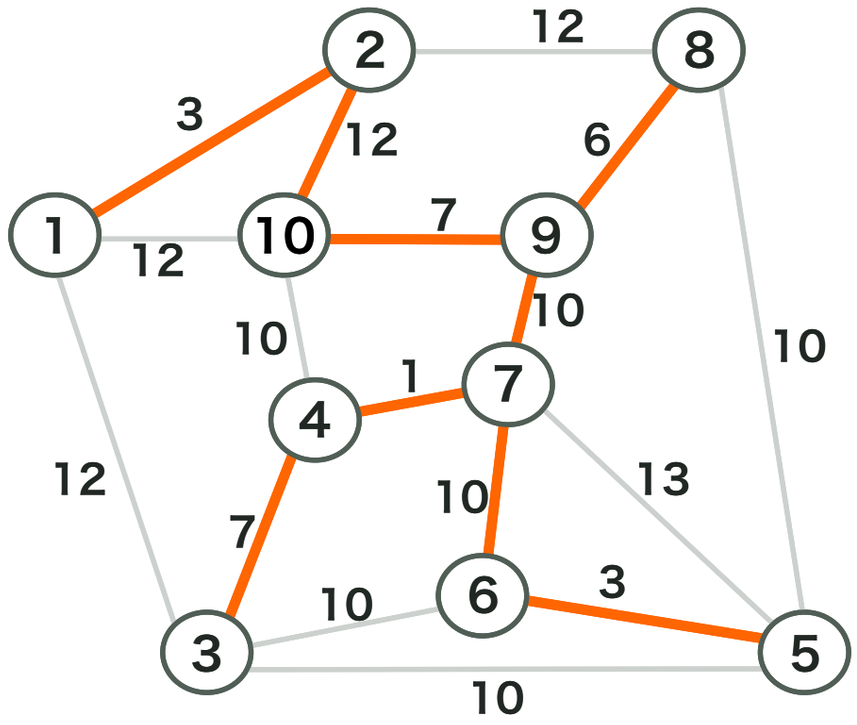

10
2
9
1
7
8
$e^6$
4
6
3
5


2
12
8
3
12
6
1
10
7
9
12
10
10
10
1
7
4
12
13
10
7
6
3
10
3
5
10

# 代替辺を順に構築

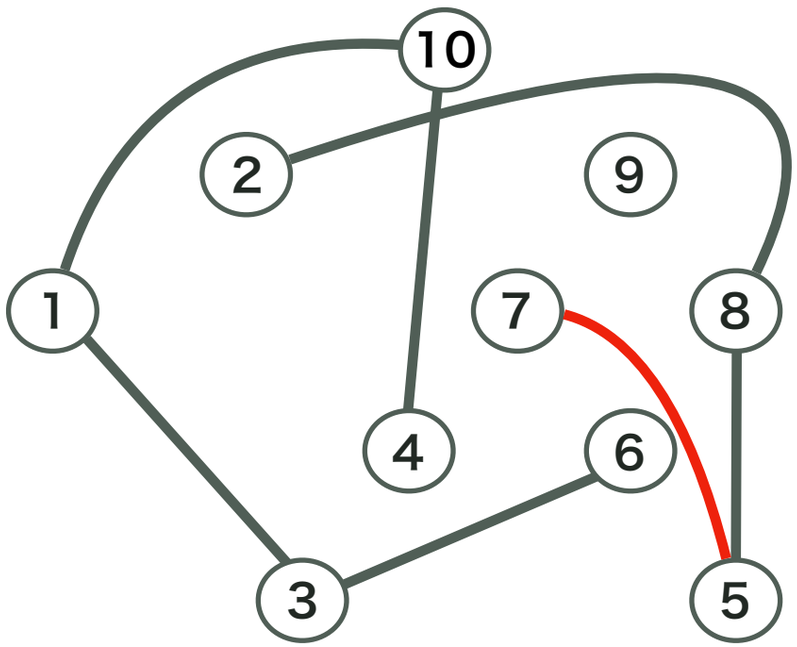

10
2
9
1
7
8
4
6
3
5

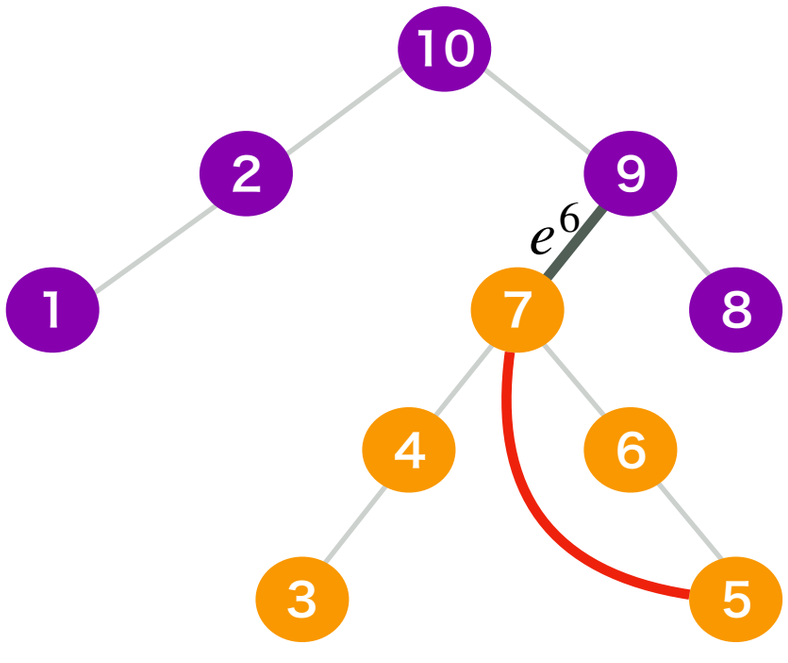

10
2
9
$e^6$
1
7
8
4
6
3
5

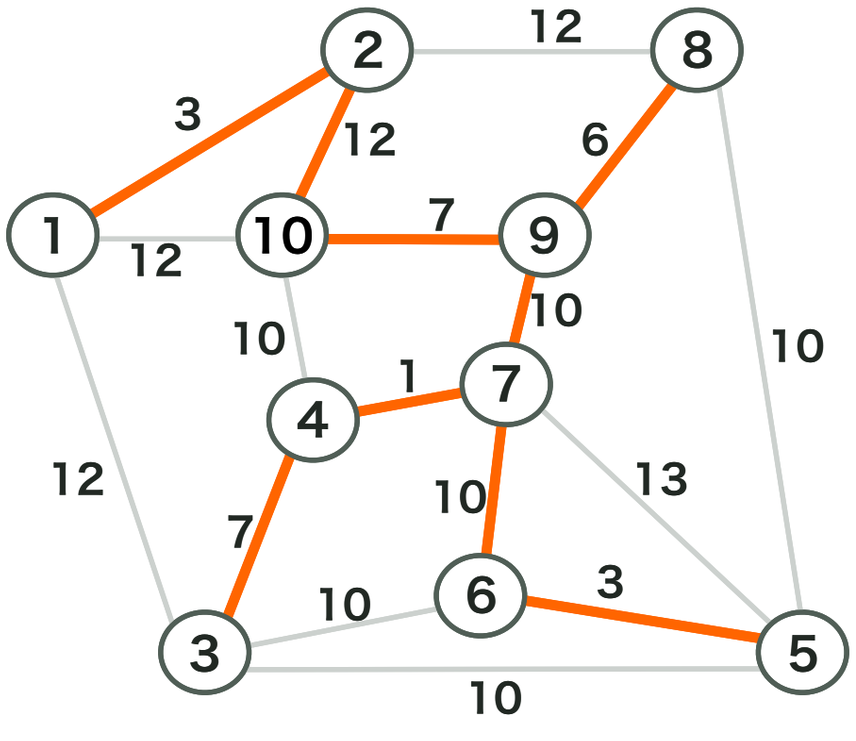

2
12
8
3
12
6
1
10
7
9
12
10
10
1
7
10
4
12
13
10
7
6
3
10
3
5
10

# 代替辺を順に構築

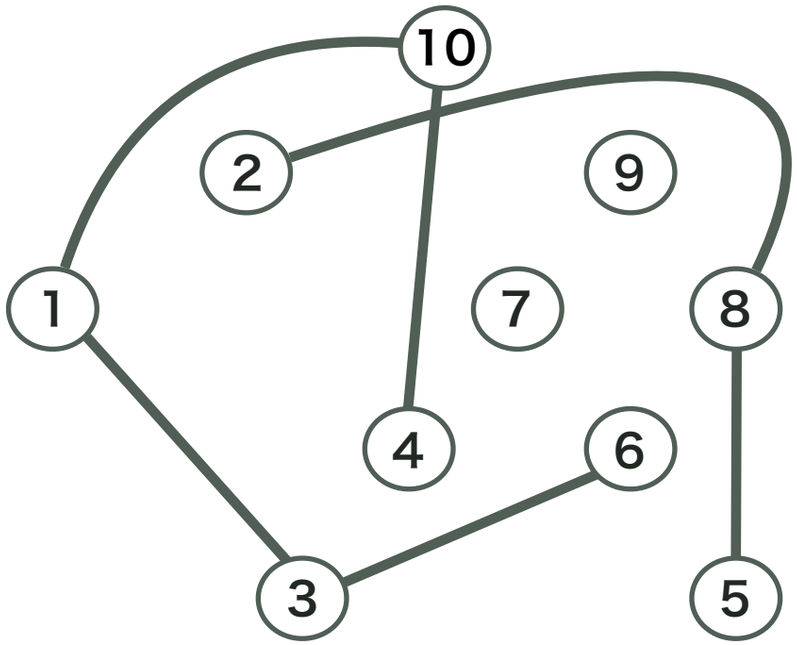
この辺は
**代替辺の候補から削除される**

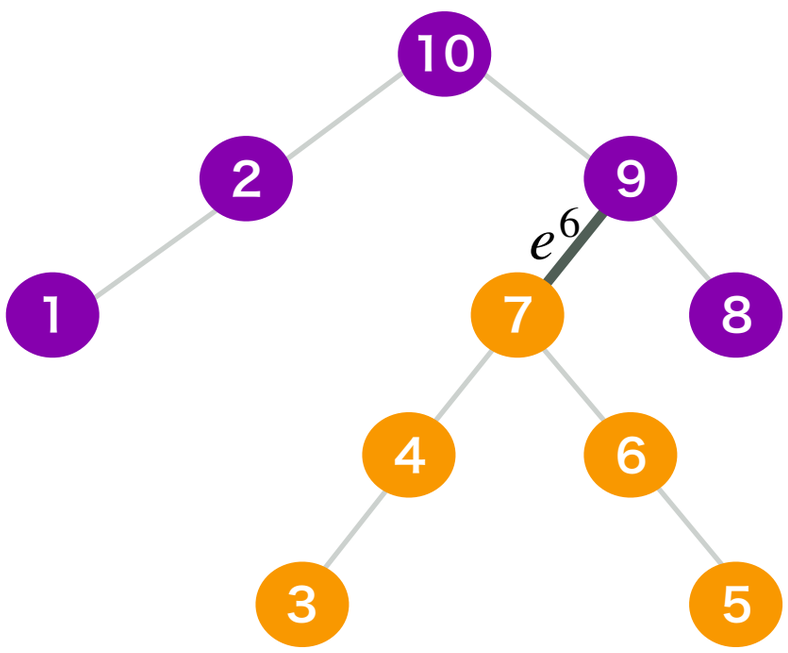


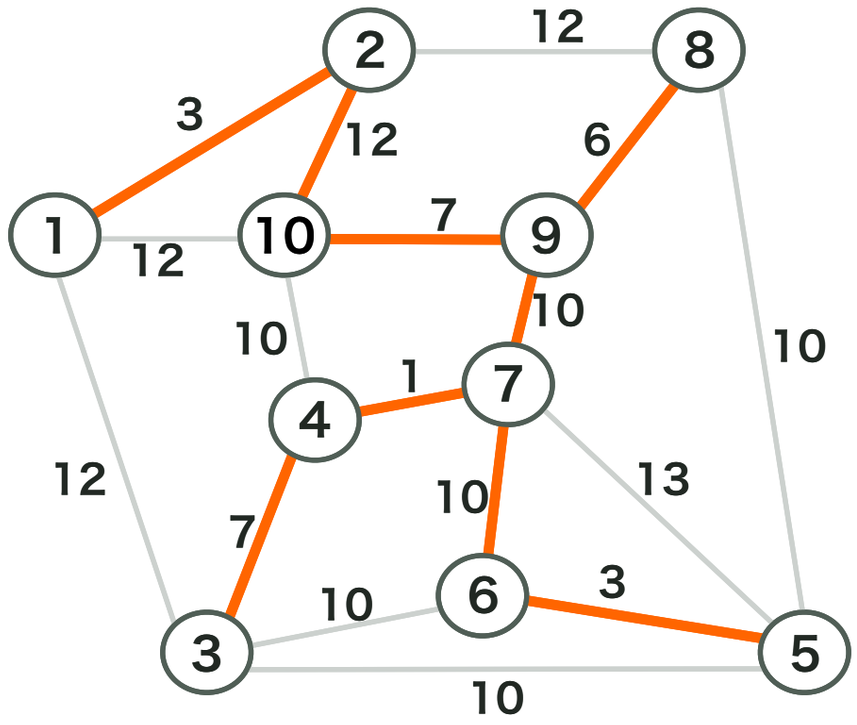



# 代替辺を順に構築


10
2
9
1
7
8
4
6
3
5

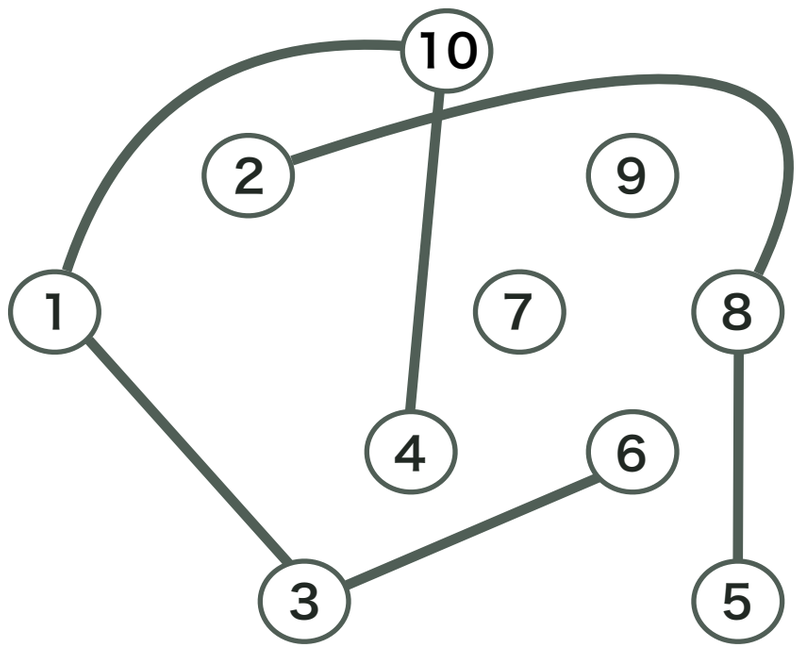

10
2
9
$e^6$
1
7
8
4
6
3
5

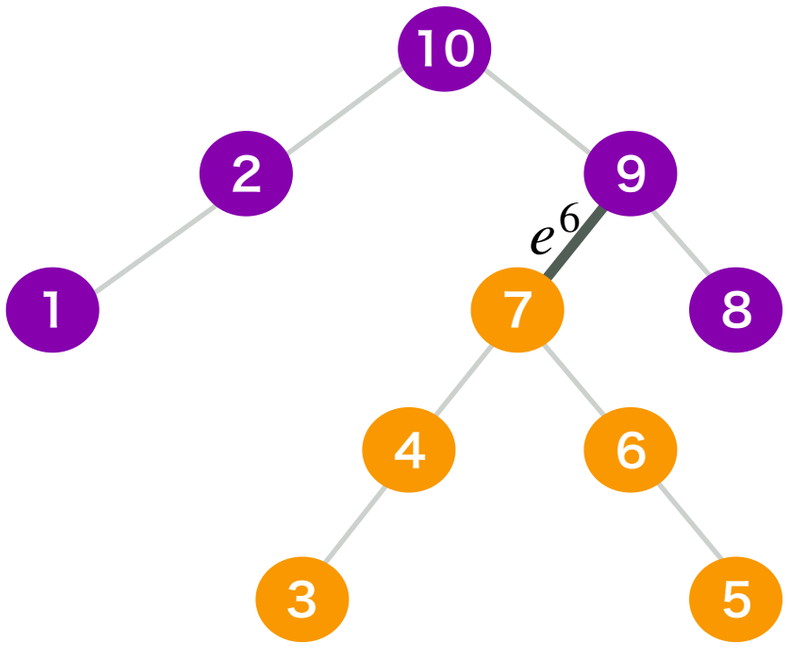

2
12
8
3
12
6
1
10
7
9
12
10
10
1
7
10
4
12
13
10
7
6
3
10
3
5
10

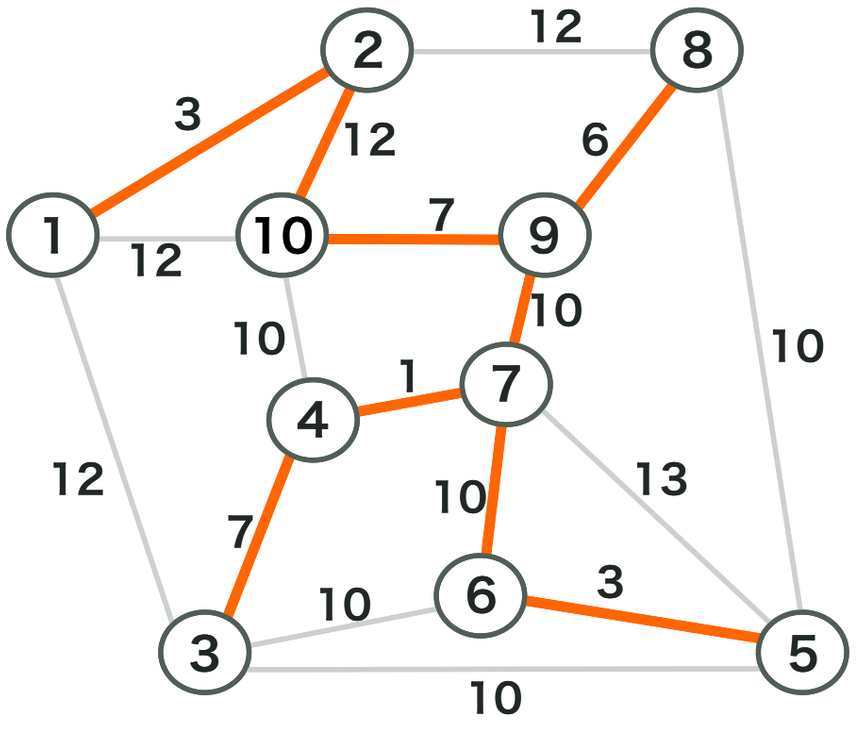
# 代替辺を順に構築

find $(10, i', j')$, $i' \in [3,7]$

重みが 3 で，**黄色**の頂点から
出ている辺を探す



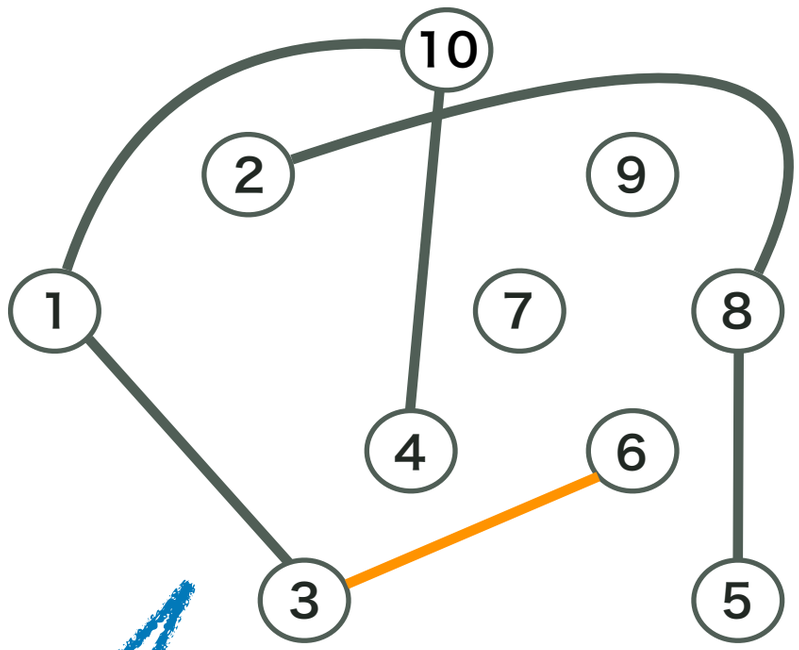



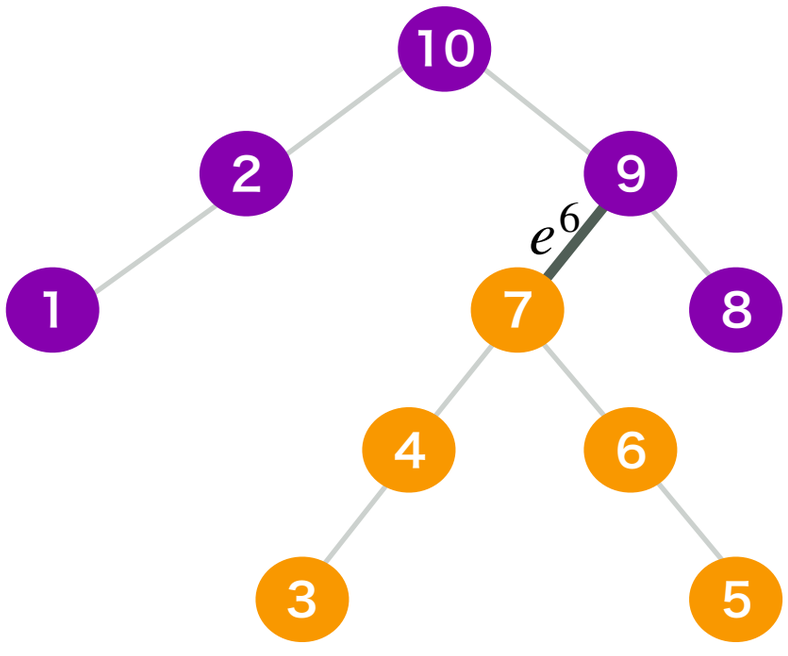
# 代替辺を順に構築

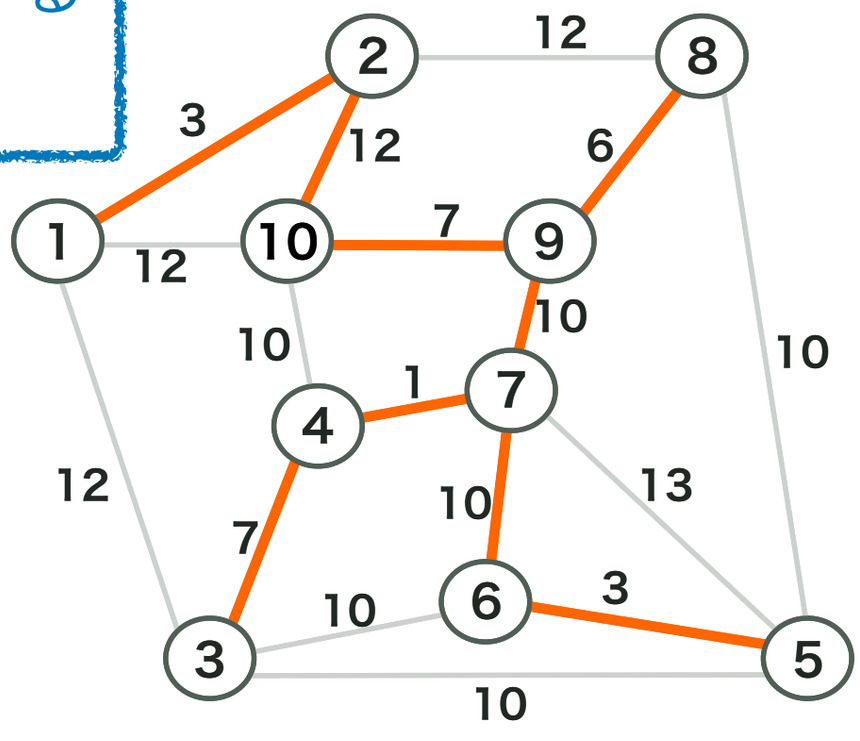
find $(10,i',j')$, $i' \in [3,7]$

重みが 3 で，<u>黄色</u>の頂点から出ている辺を探す




(10,3,6) が見つかる

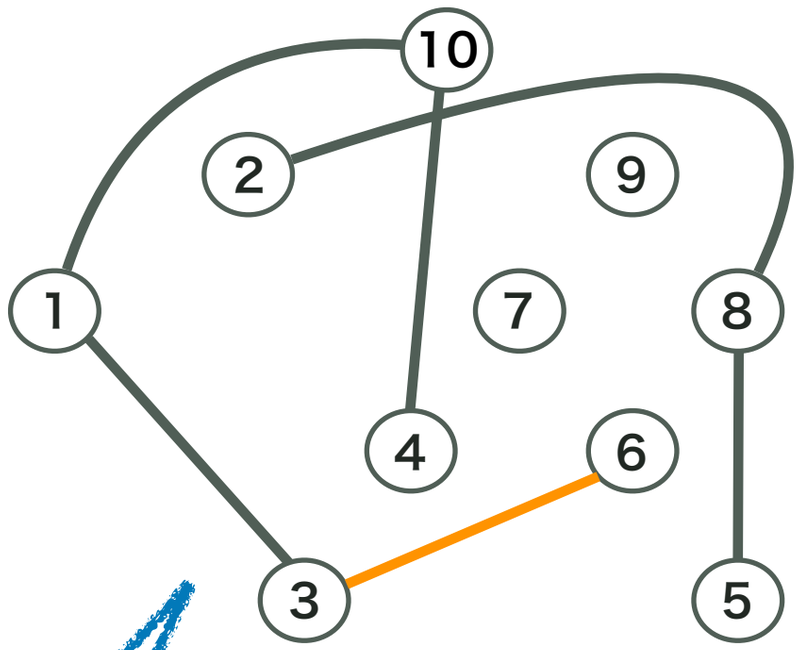


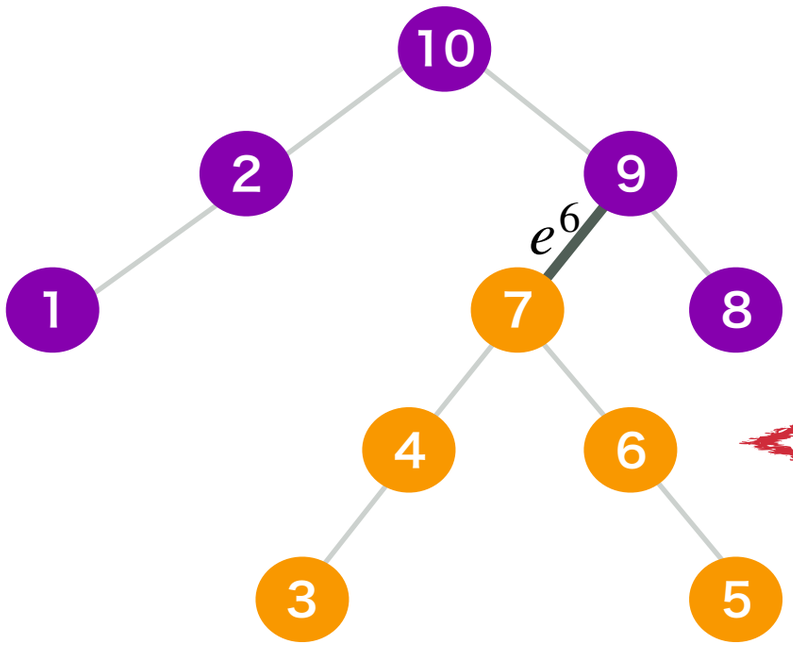
# 代替辺を順に構築

find $(10,i',j')$, $i' \in [3,7]$

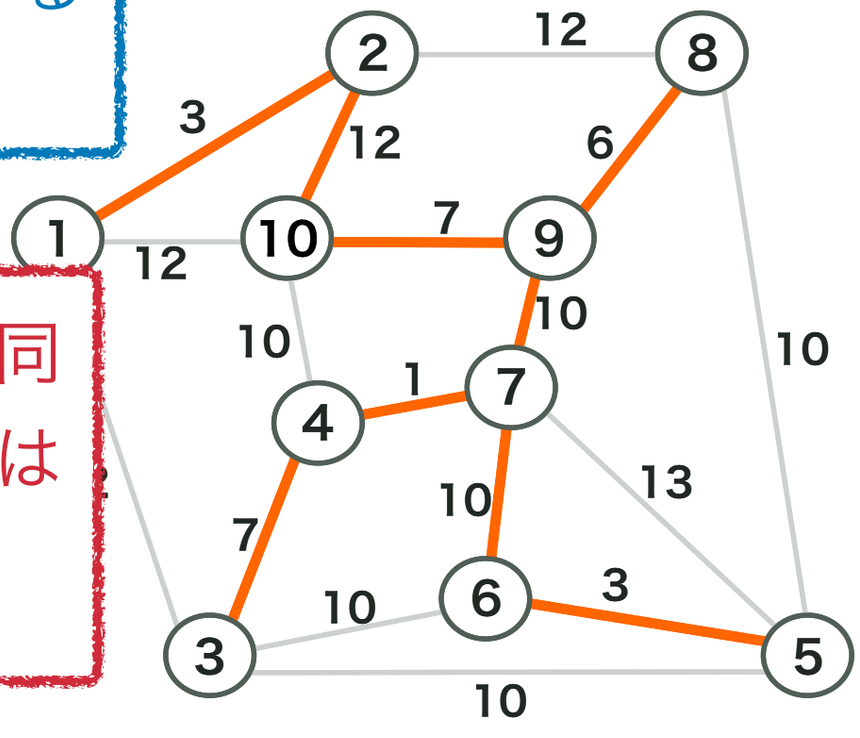
重みが 3 で，**黄色**の頂点から出ている辺を探す




(10,3,6) が見つかる



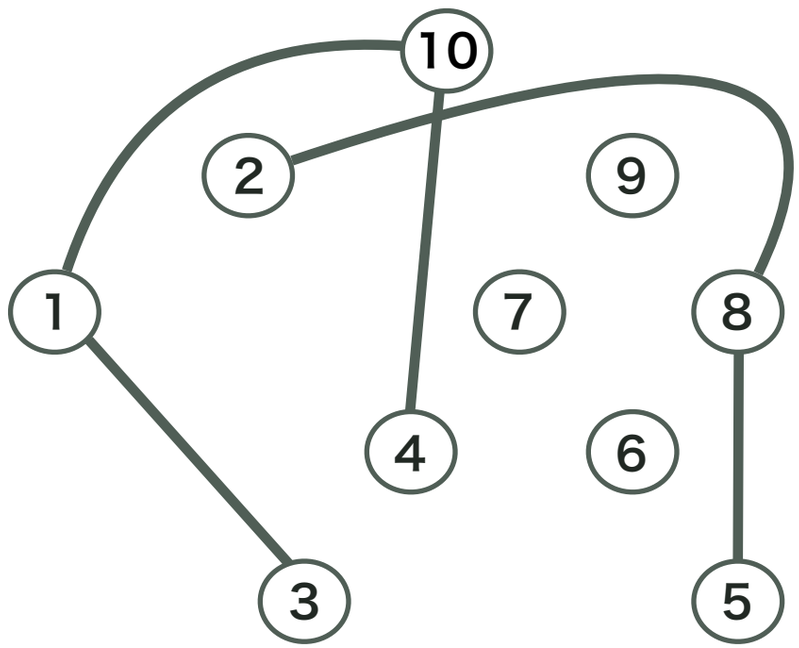

が，頂点の色が同色なのでこの辺は削除!!

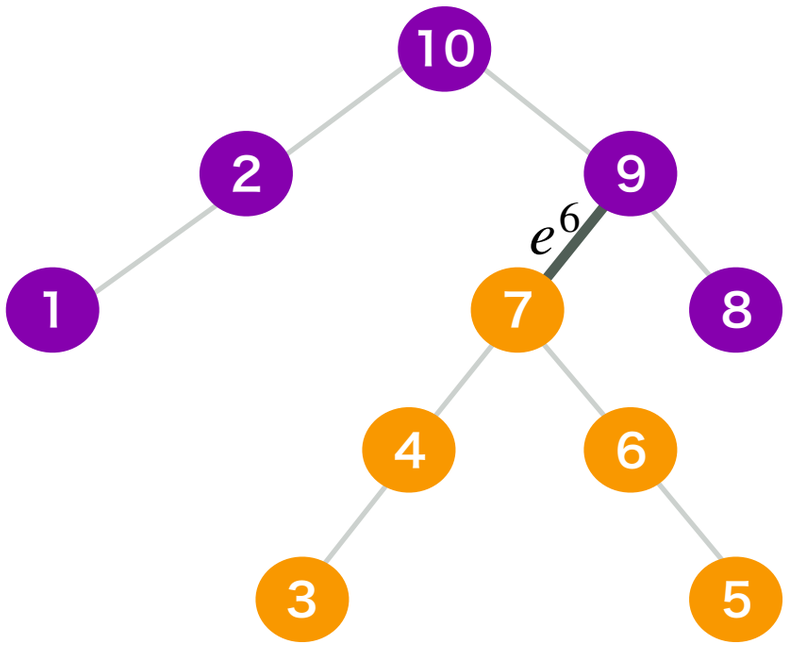


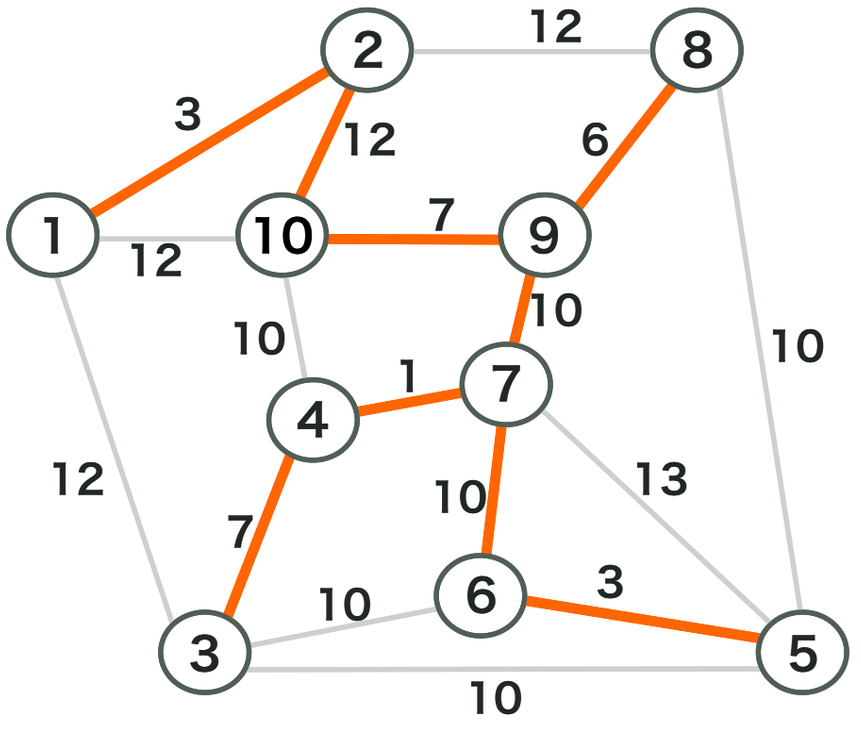
# 代替辺を順に構築

find $(10, i', j')$, $i' \in [3,7]$

重みが 3 で，**黄色**の頂点から
出ている辺を探す



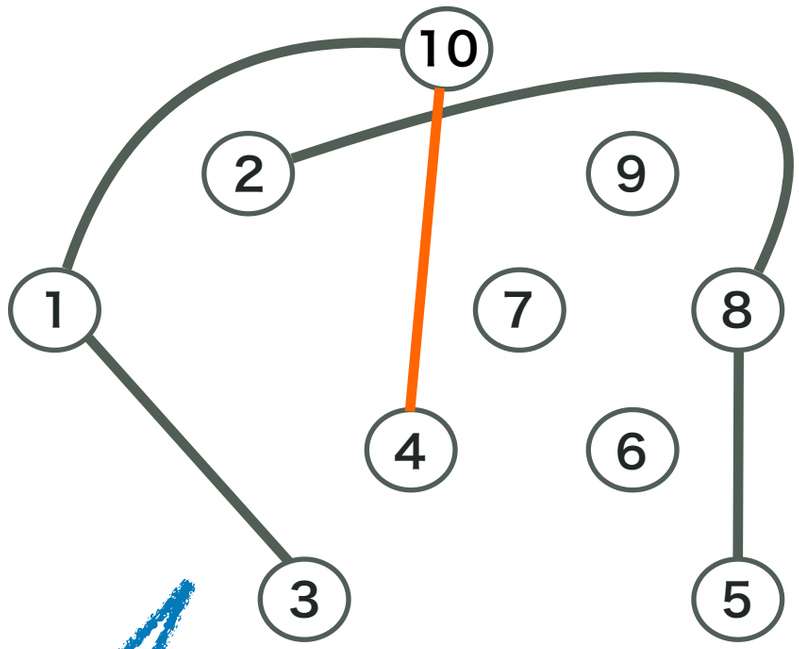



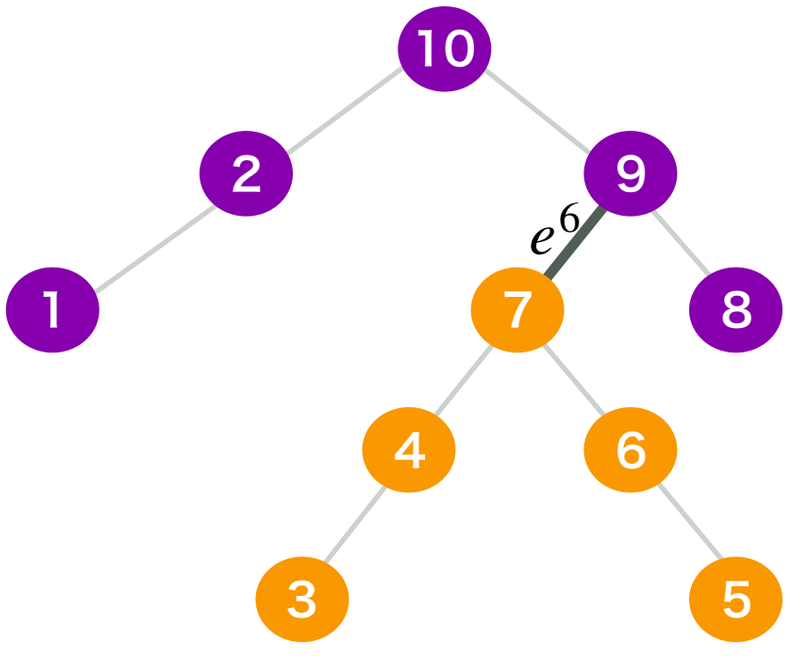
# 代替辺を順に構築

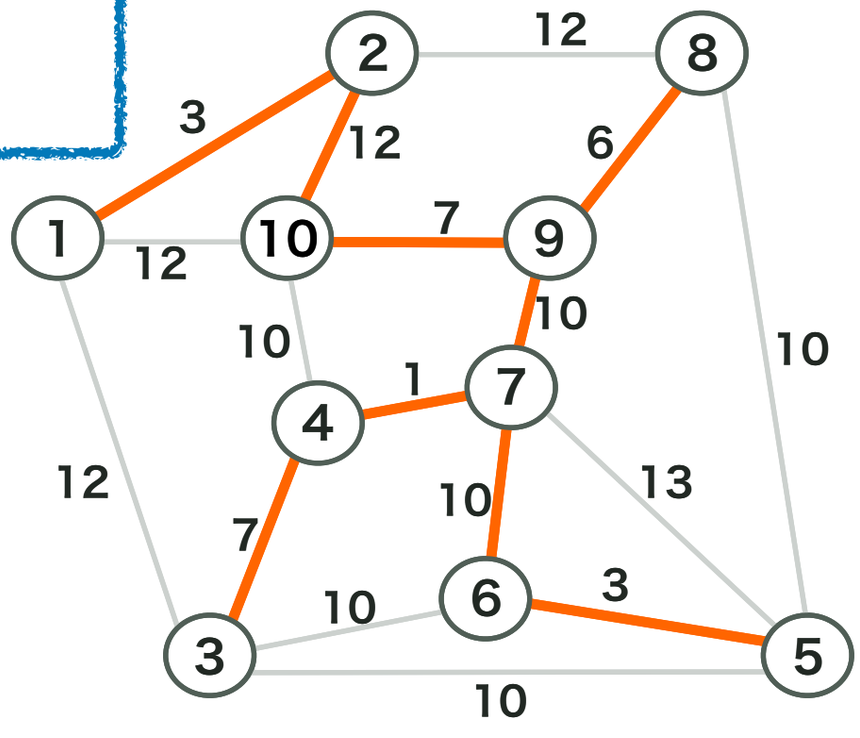
find $(10, i', j')$, $i' \in [3,7]$

重みが 3 で，**黄色**の頂点から出ている辺を探す




(10,4,10) が見つかる



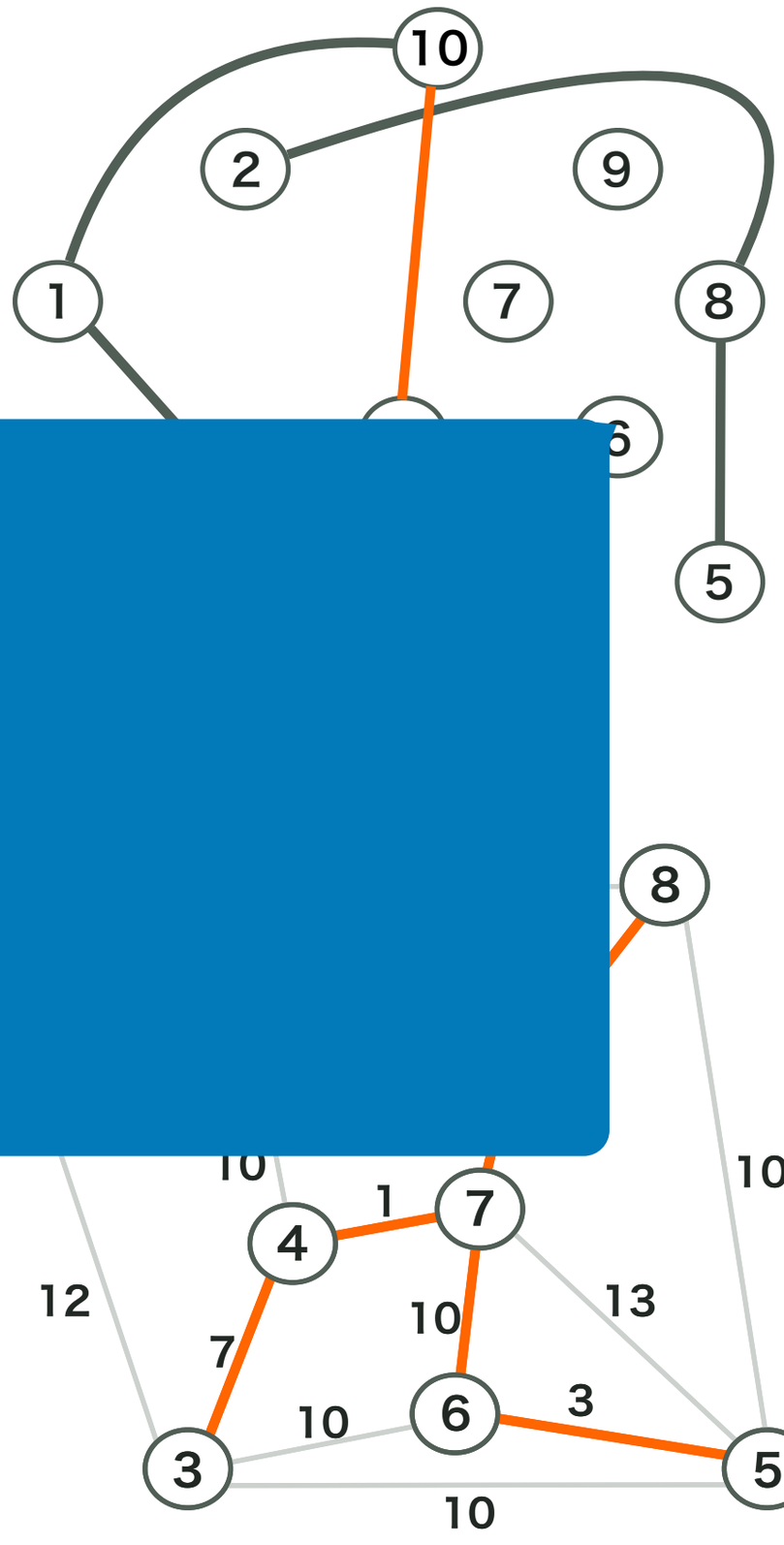
# 代替辺を順に構築

find $(10, i', j')$, $i' \in [3,7]$

重みが 3 で，<u>黄色</u>の頂点から

出

**省略**



# 計算量解析

ALGORITHM $Substitute(F, R, T)$

Step 1: $Q \leftarrow \varnothing$

Step 2: for $i \in [1, n-1]$:

(1) for $\forall e = (v^i, v^j) \in E^i \backslash (R \cup T)$: // (辺の追加)

(a) if $j \in [\underline{\sigma}_i, \bar{\sigma}_i]$: $Q$ から $e$ を削除する

(b) else: $Q$ に $e$ を追加する

(2) if $e^i \notin F$: // (代替辺の検索)

(a) $w = w(e^i)$ かつ $i' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ を満たす $(w, i', j')$ を $Q$ から検索

(b) if $(w, i', j')$ が $j' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$(w, i', j')$ を $Q$ から削除し，(2-a) へ

(c) if $(w, i', j')$ が $j' \notin [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$\tilde{e}^i \leftarrow (v^{i'}, v^{j'})$

# 計算量解析

ALGORITHM $Substitute(F, R, T)$

Step 1: $Q \leftarrow \varnothing$

Step 2: for $i \in [1, n-1]$:

(1) for $\forall e = (v^i, v^j) \in E^i \backslash (R \cup T)$: // (辺の追加)

(a) if $j \in [\underline{\sigma}_i, \bar{\sigma}_i]$: $Q$ から $e$ を削除する

(b) else: $Q$ に $e$ を追加する

(2) if $e^i \notin F$: // (代替辺の検索)

(a) $w = w(e^i)$ かつ $i' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ を満たす $(w, i', j')$ を $Q$ から検索

(b) if $(w, i', j')$ が $j' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$(w, i', j')$ を $Q$ から削除し，(2-a) へ

(c) if $(w, i', j')$ が $j' \notin [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$\tilde{e}^i \leftarrow (v^{i'}, v^{j'})$

$Q$ のサイズは高々 $m$ なので，

各種操作は $O(\log m) = O(\log n)$

# 計算量解析

ALGORITHM $Substitute(F, R, T)$

Step 1: $Q \leftarrow \varnothing$

Step 2: for $i \in [1, n-1]$:

(1) for $\forall e = (v^i, v^j) \in E^i \backslash (R \cup T)$: // (辺の追加)

(a) if $j \in [\underline{\sigma}_i, \bar{\sigma}_i]$: $Q$ から $e$ を削除する

(b) else: $Q$ に $e$ を追加する

(2) if $e^i \notin F$: // (代替辺の検索)

(a) $w = w(e^i)$ かつ $i' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ を満たす $(w, i', j')$ を $Q$ から検索

(b) if $(w, i', j')$ が $j' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$(w, i', j')$ を $Q$ から削除し，(2-a) へ

(c) if $(w, i', j')$ が $j' \notin [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$\tilde{e}^i \leftarrow (v^{i'}, v^{j'})$

辺の追加は $O(m)$ 回

それに伴い，辺の削除も $O(m)$ 回

# 計算量解析

辺の検索をするたびに,
代替辺が決定するか辺が削除されるので
$O(m)$ 回

ALGORITHM $Substitute(F, R, T)$

Step 1: $Q \leftarrow \varnothing$

Step 2: for $i \in [1, n-1]$:

(1) for $\forall e = (v^i, v^j) \in E^i \backslash (R \cup T)$: // (辺の追加)

(a) if $j \in [\underline{\sigma}_i, \bar{\sigma}_i]$: $Q$ から $e$ を削除する

(b) else: $Q$ に $e$ を追加する

(2) if $e^i \notin F$: // (代替辺の検索)

(a) $w = w(e^i)$ かつ $i' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ を満たす $(w, i', j')$ を $Q$ から検索

(b) if $(w, i', j')$ が $j' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$(w, i', j')$ を $Q$ から削除し, (2-a) へ

(c) if $(w, i', j')$ が $j' \notin [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$\tilde{e}^i \leftarrow (v^{i'}, v^{j'})$

# 計算量解析

ALGORITHM $Substitute(F, R, T)$

Step 1: $Q \leftarrow \varnothing$

Step 2: for $i \in [1, n-1]$:

(1) for $\forall e = (v^i, v^j) \in E^i \backslash (R \cup T)$: // (辺の追加)

(a) if $j \in [\underline{\sigma}_i, \bar{\sigma}_i]$: $Q$ から $e$ を削除する

(b) else: $Q$ に $e$ を追加する

(2) if $e^i \notin F$: // (代替辺の検索)

(a) $w = w(e^i)$ かつ $i' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ を満たす $(w, i', j')$ を $Q$ から検索

(b) if $(w, i', j')$ が $j' \in [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$(w, i', j')$ を $Q$ から削除し，(2-a) へ

(c) if $(w, i', j')$ が $j' \notin [\underline{\sigma}_i, \bar{\sigma}_i]$ で見つかった場合:

$\tilde{e}^i \leftarrow (v^{i'}, v^{j'})$

よって，全体で $O(m \log n)$ 時間

# まとめ

- 最小全域木を列挙するアルゴリズムを 3 つ紹介した
  - $ALL\_MST : O(N(mn + n^2 \log n))$
  - $ALL\_MST_1 : O(Nmn)$
  - $ALL\_MST_2 : O(Nm \log n)$