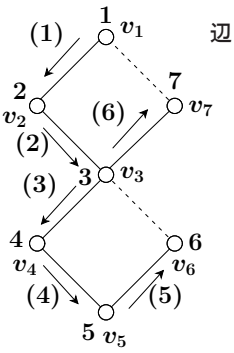
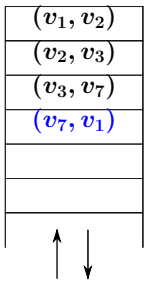
－－－－－－－－－－－－－－－－－－－－

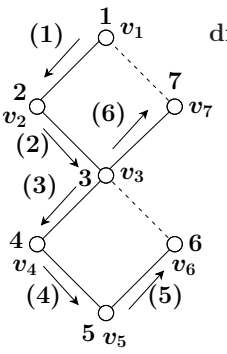


辺 (v_7, v_1) を見た.

v	$L[v]$
v_1	1
v_2	2
v_3	3
v_4	3
v_5	3
v_6	3
v_7	1

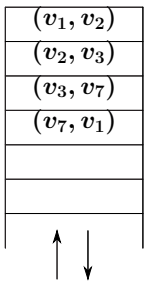


－－－－－－－－－－－－－－－－－－－－



dfsb(v_2) まで終了.

v	$L[v]$
v_1	1
v_2	1
v_3	1
v_4	3
v_5	3
v_6	3
v_7	1



$L[v_2] \geq dfnum[v_1]\ (= 1)$ が成立するので，(v_1, v_2) 以降の（つまり，全ての）辺をポップ.

－－－－－－－－－－－－－－－－－－－－

- 以上のアルゴリズムは，スタックの操作の部分を除いて，基本的に深さ優先探索を行っている．深さ優先探索は $O(n+m)$ 時間で実行できる $(n = |V|, m = |E|)$.
- スタックには，各辺が初めて調べられたときにのみプッシュされる．⇒ スタックの操作にかかる時間の合計は $O(m)$.

連結なグラフでは，$m \geq n - 1$ が成立する.
⇒ アルゴリズムの時間計算量は $O(m)$.

－－－－－－－－－－－－－－－－－－－－

8.3　グリーディ法

最適化問題（optimization problem）
　ある評価基準のもとで最も良い解を見つけるというタイプの問題

グリーディ法（greedy algorithm）
　アルゴリズムの基本的設計技法の一つで，貪欲算法とも呼ばれる．最適化問題を解くとき，計算の各段階で最も利益の大きい部分解を選んでいき，それらの部分解を組み合わせたものを最終的な解とする.

－－－－－－－－－－－－－－－－－－－－

［例］

- 郵便料金を何種類かの切手で支払う.
- 持っている切手の種類は 10 円, 50 円, 100 円の 3 種類とする.
- 使用する切手の枚数を最小にしたい.

－－－－－－－－－－－－－－－－－－－－

例えば，料金が 160 円のとき，10 円切手を 16 枚使うよりも，10 円，50 円，100 円の切手を 1 枚ずつ使う方がよい.

⇓

「料金をこえない範囲で，金額が最大の切手を選択する」ことを繰り返すとうまくいきそうに思える.
これは，グリーディ法.

－－－－－－－－－－－－－－－－－－－－

支払うべき料金が 270 円 とする.

切手の種類：10 円，50 円，100 円

残金	270	→	170	→	70	→	20	→	10	→	0
選択		100		100		50		10		10	

これは最適解.

切手の種類が 10 円，80 円，100 円
のときは，最適解が得られない.

残金	270	→	170	→	70	→	60	→	···	→	0
選択		100		100		10		10		10	

（最適解：100 円，80 円＊２，10 円）

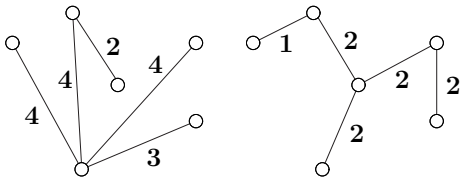
－－－－－－－－－－－－－－－－－－－－

- このように，グリーディ法は必ずしも最適解を求めるわけではない．
- しかし，グリーディ法は単純で高速であることが多いため，他の手法と組み合わせることにより有効なアルゴリズムになる場合がある．
- グリーディ法によって，常に最適解が求まる問題も存在する．
  ··· 最小スパニング木問題，（始点が一つ指定されている場合の）最短路問題

－－－－－－－－－－－－－－－－－－－－

7.4　最小スパニング木

複数の点を，最小のコストでつなぎたい（2 点をつなぐコストは，それぞれ決まっている）．



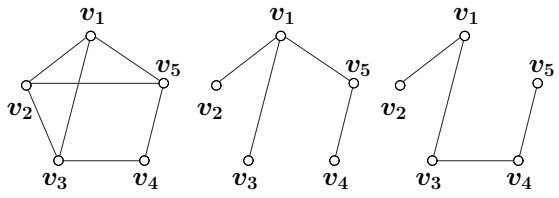
コスト 17　　**コスト 9**

この問題を，グラフの問題として定式化．

－－－－－－－－－－－－－－－－－－－－

$G = (V, E)$：連結な無向グラフ．$n = |V|$，$m = |E|$ とする．

G のスパニング木：すべての頂点を含む G の部分グラフで，木になっているもの

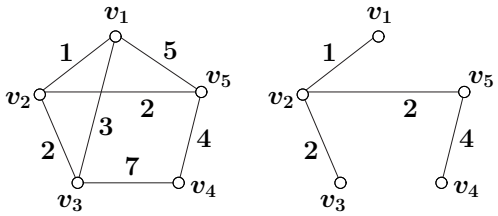


グラフ G　　**スパニング木の例**

－－－－－－－－－－－－－－－－－－－－

各辺 $e \in E$ に重さ $w(e)$ が付けられているとする．

最小スパニング木：辺の重さの総和が最小となるスパニング木



グラフ G　　**最小スパニング木**

－－－－－－－－－－－－－－－－－－－－

クラスカル（Kruskal）のアルゴリズム

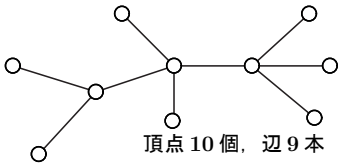
最小スパニング木を求めるアルゴリズムの一つ．グリーディ法による．

- 最小スパニング木（の辺集合）T を段階的に構成していく．
- T の辺として，重さの小さいものをできるだけ採用したい．
  $\Rightarrow$ G の辺を，重さの小さいものから順に T に加えていく．ただし，閉路は作らないようにする．

－－－－－－－－－－－－－－－－－－－－

準備（復習）

性質１： 任意の木 (V_T, E_T) に対して，$|E_T| = |V_T| - 1$ が成立する．



性質２： 無向グラフ $H = (V_H, E_H)$ に対して，$|E_H| = |V_H| - 1$ であるとする．このとき，次の (1)∼(3) は互いに同値である．

(1) H は木である．

(2) H は連結である．

(3) H は閉路をもたない．

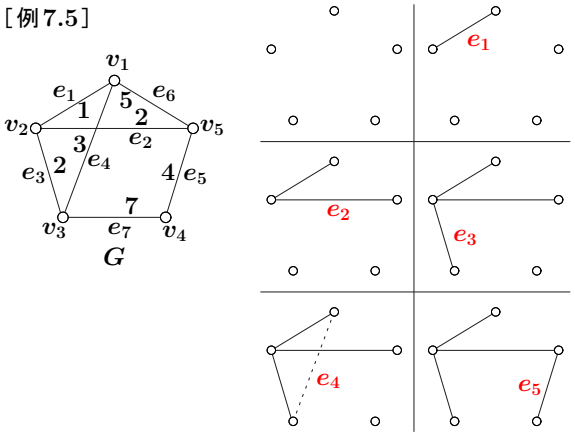
－－－－－－－－－－－－－－－－－－－－

関数 kruskal()

重さの非減少順に辺を並べた列を $e_1, e_2, \cdots, e_m$ とする；
$i = 0;\ T = \emptyset;$
while $(|T| \neq n-1)\{$
i++;
if ($T \cup \{e_i\}$ が閉路をもたない)
$T = T \cup \{e_i\};$
}

－－－－－－－－－－－－－－－－－－－－

［例 7.5］



－－－－－－－－－－－－－－－－－－－－

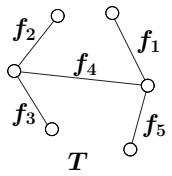
アルゴリズムの正当性

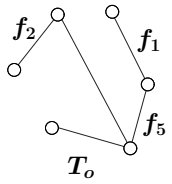
T：クラスカルのアルゴリズムで得られるスパニング木（の辺集合）

$f_1, f_2, \cdots, f_{n-1}$：$T$ の辺（T に加えられた順）

T_o：最小スパニング木のうち，T の辺を最も多く含むもの（の辺集合）

<u>$T \neq T_o$ であると仮定する．</u>

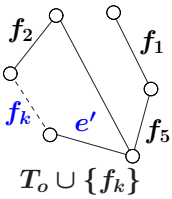




f_k：$f_1, f_2, \cdots, f_{n-1}$ の辺のうち，T_o に含まれない最初のもの

$T_o \cup \{f_k\}$ は一つの閉路を含む．

e'：その閉路上の辺で，T の辺でない任意のもの



$\{f_i \mid 1 \leq i < k\} \cup \{e'\}$ は閉路を含まない．

$\Rightarrow w(f_k) \leq w(e')$.

$T_o \cup \{f_k\} - \{e'\}$ は，連結で $n-1$ 本の辺を含む．

$\Rightarrow$ この辺集合もスパニング木を構成している．

$w(f_k) \leq w(e')$ であるから，$T_o \cup \{f_k\} - \{e'\}$ の辺の重さの和は T_o より大きくない．

$\Rightarrow T_o$ は最小スパニング木であるから，
$T_o \cup \{f_k\} - \{e'\}$ も最小スパニング木．

$T_o \cup \{f_k\} - \{e'\}$ は，T_o よりも，T の辺を 1 本多く含む．$\cdots$ <u>T_o の定義に矛盾．</u>

以上より，$T_o = T$．よって，T は最小スパニング木である．

－－－－－－－－－－－－－－－－－－－－

クラスカルのアルゴリズムの時間計算量

(1) 辺のソーティング $\cdots$ マージソートかヒープソートを使えば，$O(m \log n)$ 時間で可能．
（$m \leq n(n-1)/2$ より，$\log m = O(\log n)$ であることに注意）

(2) $T \cup \{e_i\}$ が閉路をもつかの判定 $\cdots$ e_i の両端点が，T において同じ連結成分に属するかどうかを判定すればよい．深さ優先探索あるいは幅優先探索を使えば，1 回当たり $O(n)$ 時間で実行可能．

(3) while ループ内の処理は，最大で m 回実行される．(2) の判定を除けば，このループ内の処理は，1 回当たり $O(1)$ 時間で実行可能．

以上より，クラスカルのアルゴリズムは，
$$O(\max\{m \log n, mn\}) = O(mn)$$
時間で実行することができる．

－－－－－－－－－－－－－－－－－－－－

データ構造の工夫により，全体の時間計算量を $O(m \log n)$ 時間に減らすことができる．
（Union-Find のための木構造を用いる．説明略．）

また，最小スパニング木を $O(n \log n + m)$ 時間で求めるアルゴリズムも存在する．
（プリムのアルゴリズムとフィボナッチヒープを用いる．説明略．）

－－－－－－－－－－－－－－－－－－－－