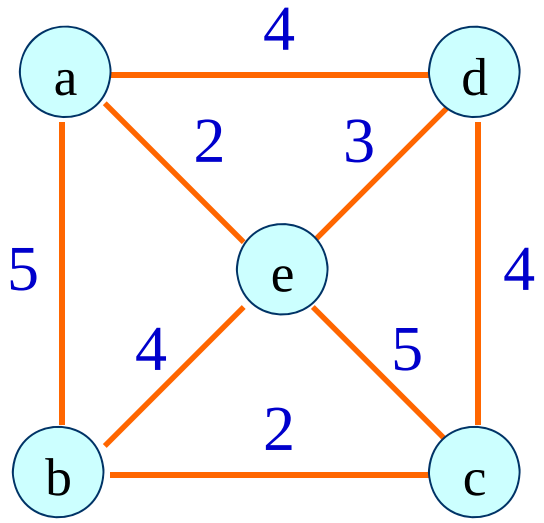
プリムのアルゴリズム



V1

{ a }

{ a e }

{ a e d }

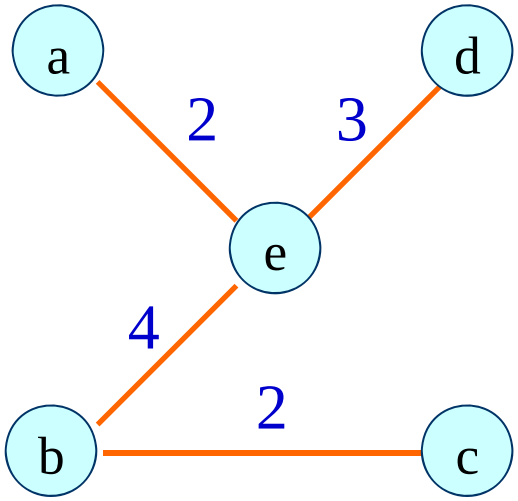
{ a e d b }

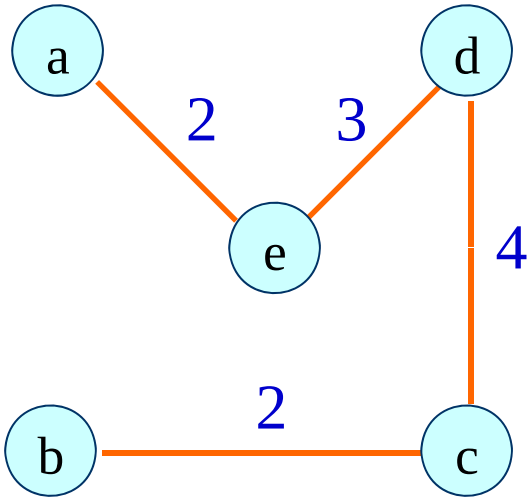
{ a e d b c }

{ a e d c }

{ a e d c b }

コスト４の，bとc，
どちらを選ぶか？





コスト最小の生成木を求めるアルゴリズム（１）

・プリム（Prim）のアルゴリズム

Ｖ＝｛０，１，・・・，ｎ－１｝，

Ｅ＝｛０，１，・・・，ｍ－１｝．

コスト最小の生成木を与える辺の集合 Ｅ1 を求める．

(1) Ｖ1 ＝｛０｝，Ｅ1 ＝φとする．

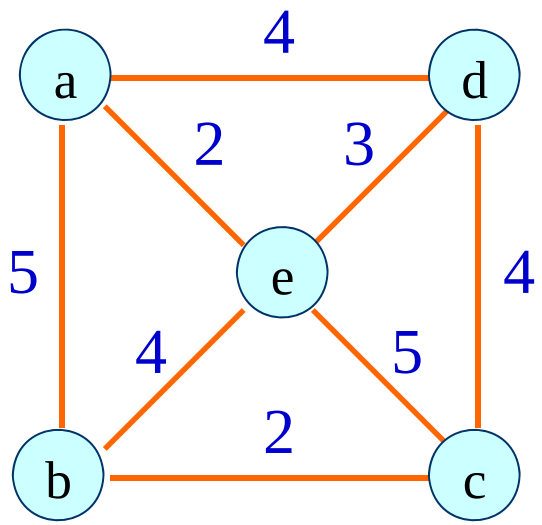
(2) Ｖ1 が Ｖ と等しくなるまで繰り返す．

(2-1) ｘ∈Ｖ1，ｙ∈Ｖ－Ｖ1 であるような，コスト最小の辺（ｘ，ｙ）を選ぶ．

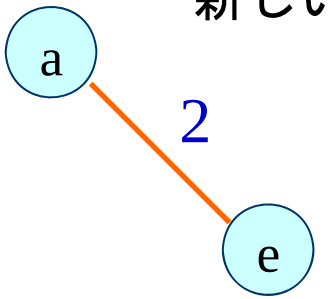
(2-2) Ｖ1 を，Ｖ1 ∪｛ｙ｝に更新する．

(2-3) Ｅ1 を，Ｅ1 ∪｛（ｘ，ｙ）｝に更新する．

クラスカルのアルゴリズム

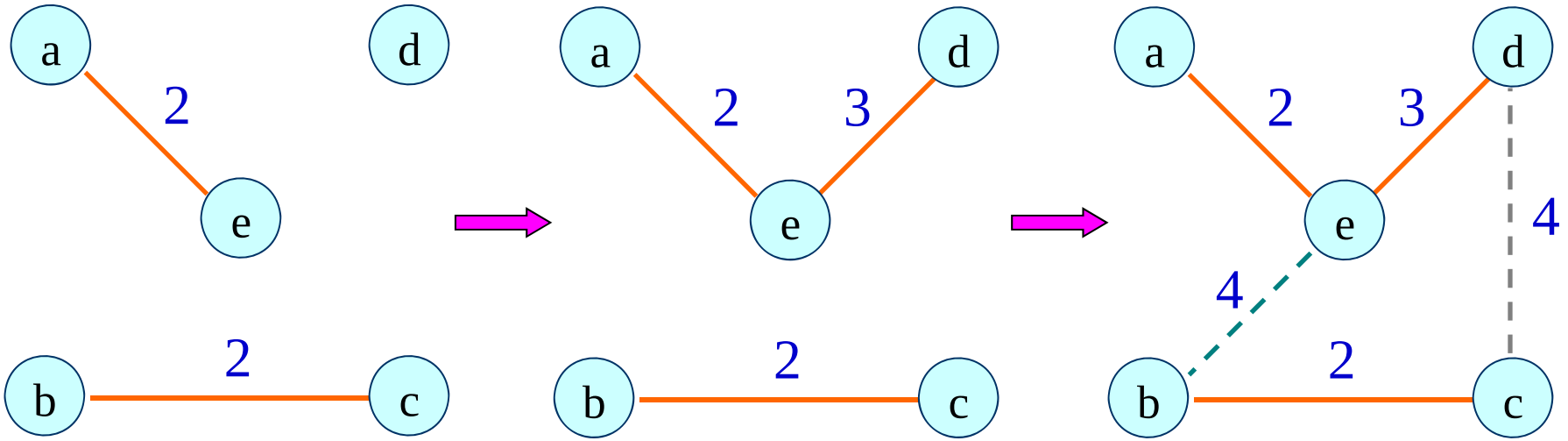


新しい連結



または

新しい連結



コスト最小の生成木を求めるアルゴリズム（２）

・クラスカル（Kruskal）のアルゴリズム

Ｖ＝｛０，１，・・・，ｎ－１｝，

Ｅ＝｛０，１，・・・，ｍ－１｝．

コスト最小の生成木を与える辺の集合　Ｅ1　を求める.

準備

Ｅ2：　木の一部になるかを調べる対象の辺の集合

連結集合：　辺の列により結ばれている節点の集合.　１つのみの節点の集合も，連結集合.

(1) Ｅ1＝φ，Ｅ2＝Ｅ とする.

(2) 各節点それぞれからなる連結集合を作る.

(3) 全ての節点が，１つの連結集合に含まれるようになるまで繰り返す.

(3-1) Ｅ2 から最小のコストの辺 ｚ を選ぶ.

(3-2) ｚ が異なる２つの連結集合に含まれる節点を結ぶのであれば,

(3-2-1) それら２つの連結集合の和集合を，新しい連結集合とて生成する.

(3-2-2) ２つの連結集合を削除する.

(3-2-3) Ｅ1 を，Ｅ1 ∪ {ｚ} に更新する.

(3-3) Ｅ2 を，Ｅ2 － {ｚ} に更新する.

欲張り法（貪欲法：　greedy method）：

局所的な１変数の値が評価値の変化に及ぼす貢献度を調べ，最適化問題を解決する方法.

コスト最小の生成木を求めるアルゴリズム

局所的に各辺に注目し，コストを変数として，コストの値が「最小コスト」の評価に及ぼす貢献度を調べ，その辺を「選ぶ」，「選ばない」を判断し，解く.

↓

欲張り法.

最短経路（有向グラフ）

無向グラフに適用すると，コスト最小生成木が得られる.

出発点からの最短経路を求める.

有向連結グラフ　Ｇ＝（Ｖ，Ｅ）.

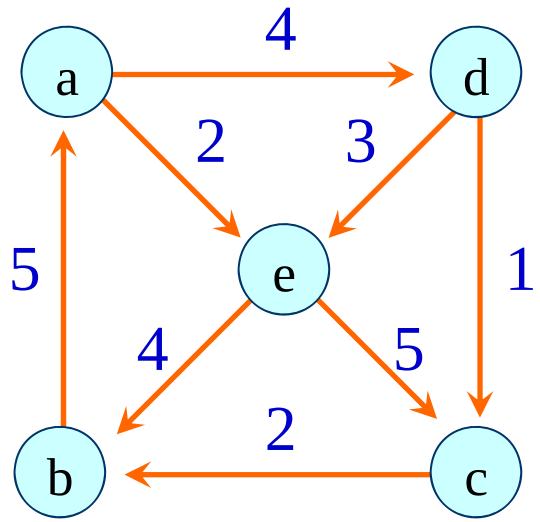
辺に非負のコストが与えられている.

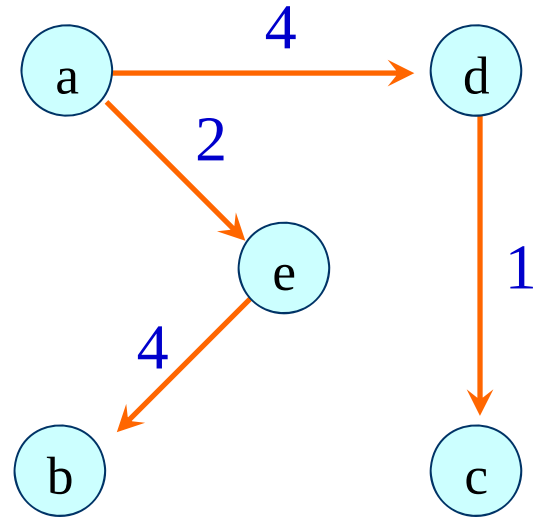
ホーム（出発点）としての節点　ｈ∈Ｖ　を与える.

ｈ　から，Ｖ　の各節点にいたる，コスト最小の，辺の列を見つける.

最短経路では，必ずしも閉路は要求されない.

ダイクストラのアルゴリズム　（ホーム ｈ は 0）





V1	C（ｎｉ）			
	b	c	d	e
{ a }	∞	∞	4	2
{ a　e }	6	7	4	–
{ a　e　d }	6	5	–	–
{ a　e　d　c }	6	–	–	–
{ a　e　d　c　b }				

ダイクストラ（Dijkstra）のアルゴリズム　（欲張り法）

Ｖ＝｛０，１，・・・，ｎ－１｝，ホーム ｈ∈Ｖ.

(1) Ｖ1 ＝ ｛ｈ｝ とする.

(2) ｈからｎｉ ∈ Ｖ－Ｖ1 へのコスト Ｃ（ｎｉ）を求める.

ｎｉ と ｈ が隣接する時は，ｎｉ と ｈ の間のコスト.

ｎｉ と ｈ が隣接しない時は，無限大相当値.

(3) Ｖ1 ＝ Ｖ となるまで繰り返す.

(3-1) Ｖ－Ｖ1 から，Ｃ（ｎｉ）が最小の節点ｘを選ぶ.

(3-2) Ｖ1 を，Ｖ1 ∪ ｛ｘ｝に更新する.

(3-3) Ｖ－Ｖ1 の各節点 ｎｉ に対して繰り返す.

(3-3-1) Ｃ（ｘ）＋（辺（ｘ，ｎｉ）のコスト）

と，Ｃ（ｎｉ）の，小さい方の値で，

Ｃ（ｎｉ）を更新する.