グラフネットワーク
~フロー & カット~

アルゴ研 D2　井上 祐馬

内容

- 最大流と最小カットの定義
- 最大流 - 最小カット定理
- 最大流を求めるアルゴリズム (Ford-Fulkerson, Dinic)
- 最小費用流とその解き方 (SSP)
- 全域最小カット・最大カット

内容

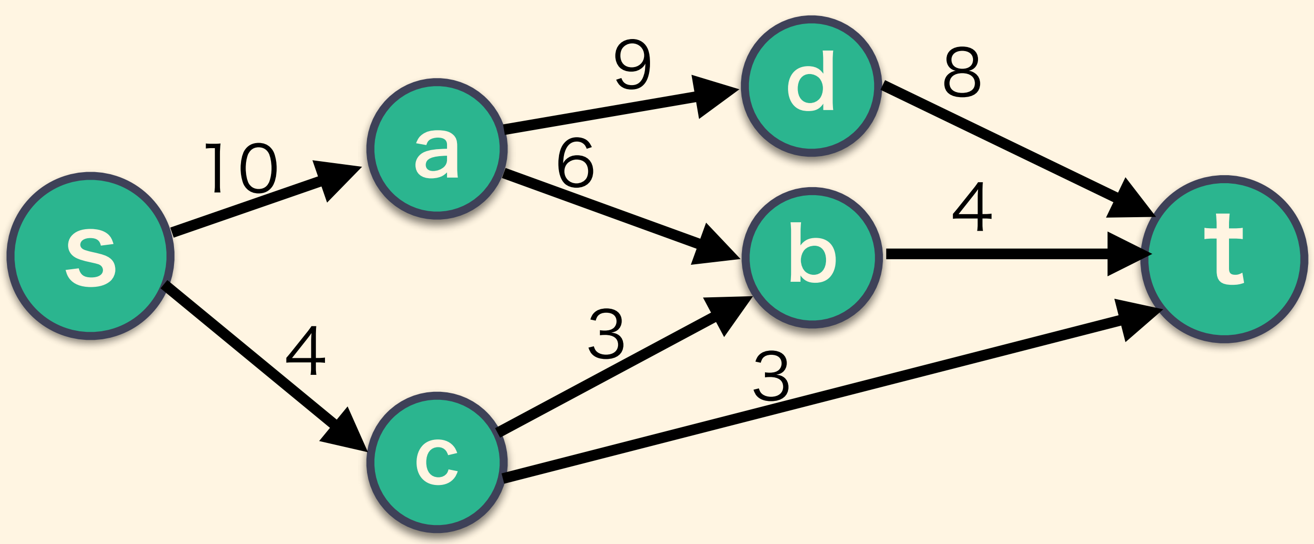
- 最大流と最小カットの定義 ← 蟻本
- 最大流 - 最小カット定理 ← 蟻本
- 最大流を求めるアルゴリズム (Ford-Fulkerson, Dinic)
- 最小費用流とその解き方 (SSP) ← 蟻本
- 全域最小カット・最大カット ← not 蟻本

内容

- **最大流と最小カットの定義**
- 最大流 - 最小カット定理
- 最大流を求めるアルゴリズム (Ford-Fulkerson, Dinic)
- 最小費用流とその解き方 (SSP)
- 全域最小カット・最大カット

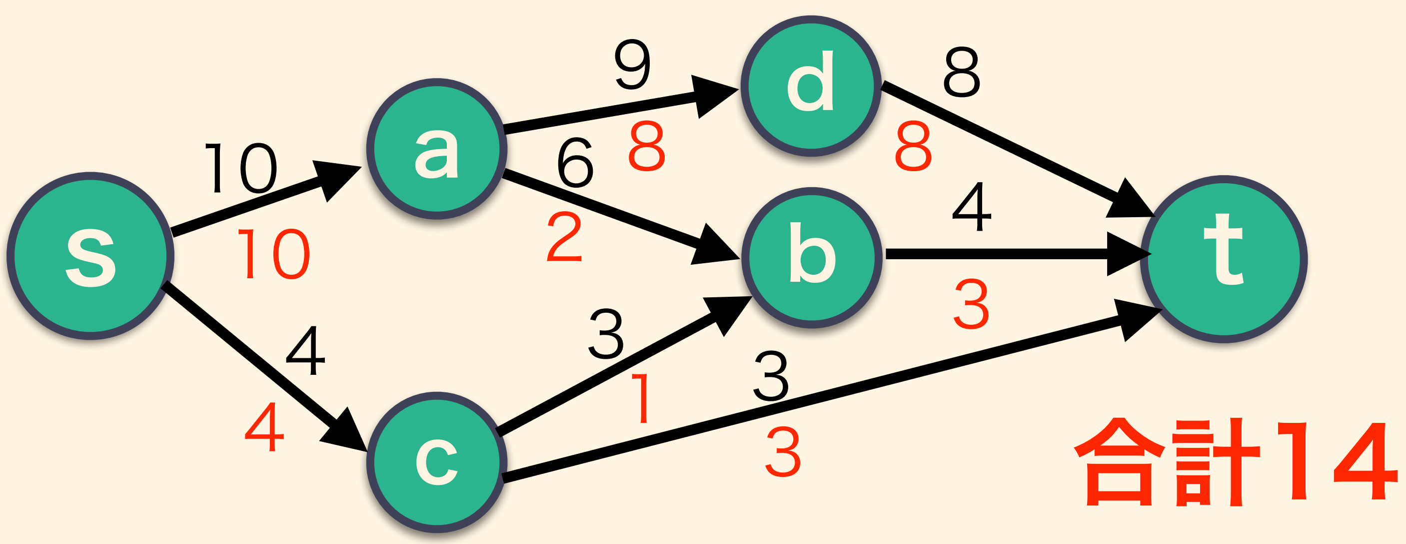
ネットワークフロー

- 辺に容量制約がついた有向グラフ G = (V,E) を考える
- 始点 (ソース) s から終点 (シンク) t へ、各辺の容量制約を守って**最大いくら t まで流せるか？**



ネットワークフロー

- 辺に容量制約がついた有向グラフ G = (V,E) を考える
- 始点 (ソース) s から終点 (シンク) t へ、各辺の容量制約を守って最大いくら t まで流せるか？

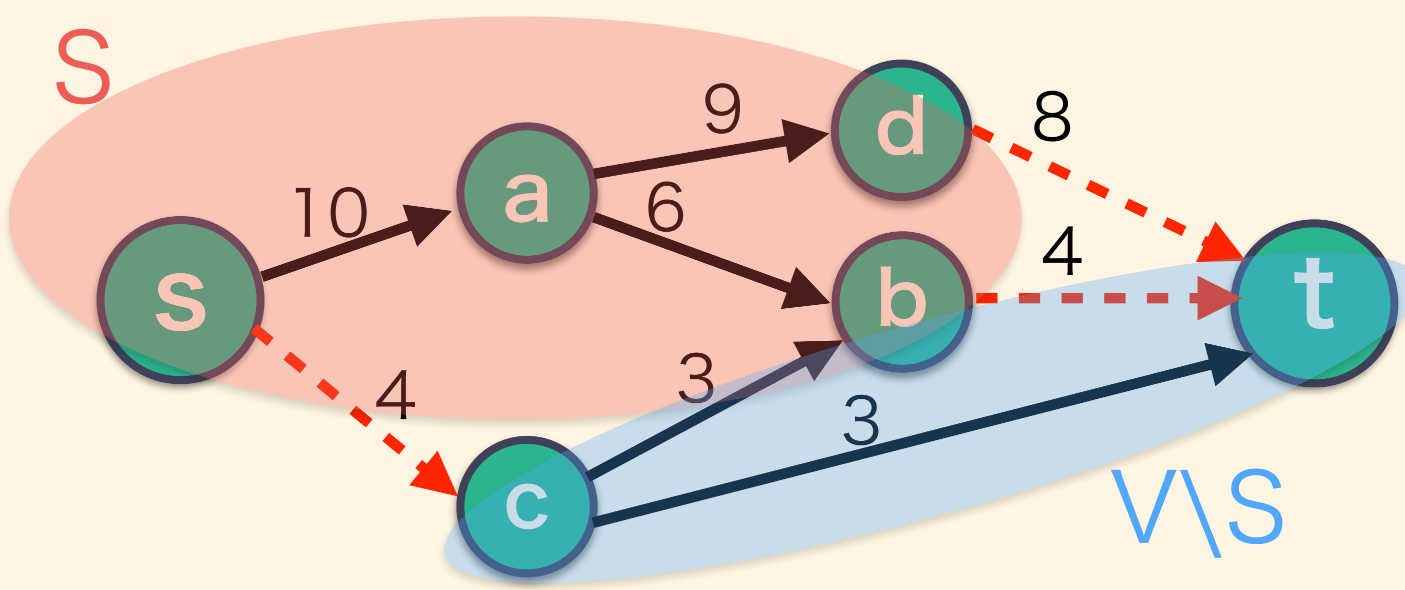


ネットワークフロー

- もっと数学的な定義: G = (V,E) の s-t フローとは、
  - 辺 $e \in E$ に対し、容量関数 $c(e)$、流量関数 $f(e)$を考える
  - $f(e)$ は以下の制約を満たす
    - $0 \leq f(e) \leq c(e)$
    - $v \in V \setminus \{s,t\}$ について、$\Sigma_{e \in \overset{\text{入辺}}{\delta\text{-}(v)}} f(e) = \Sigma_{e \in \overset{\text{出辺}}{\delta\text{+}(v)}} f(e)$
  - このとき、$\Sigma_{e \in \delta+(s)} f(e)\ (= \Sigma_{e \in \delta-(t)} f(e))$ を最大化せよ

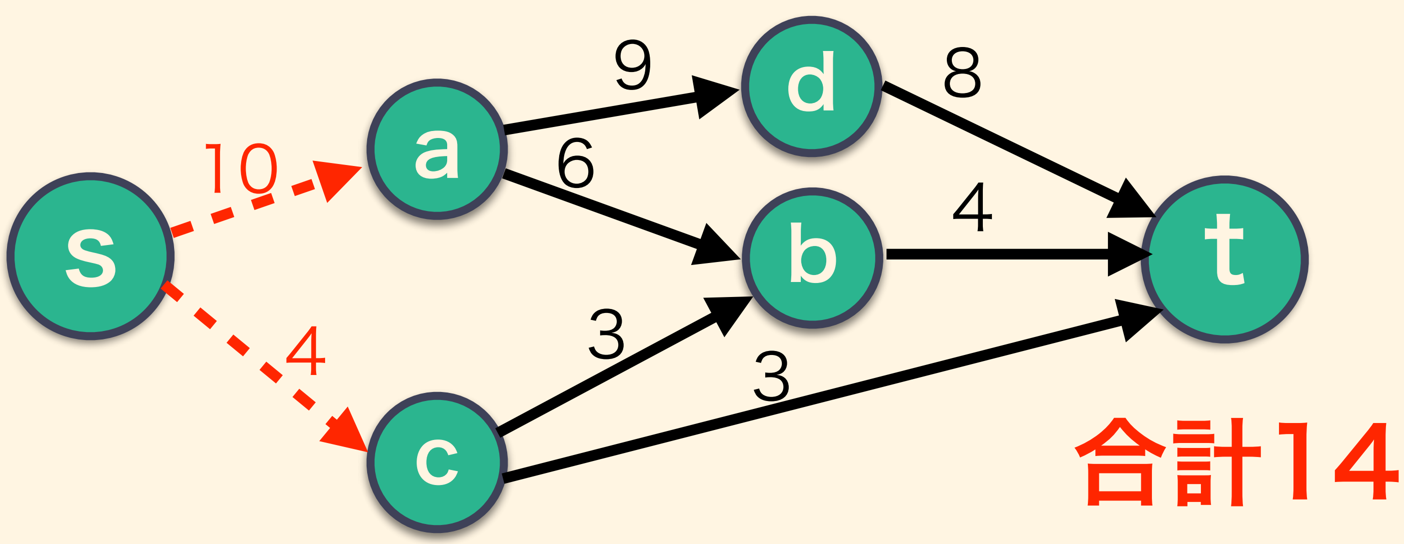
グラフカット

- (重み付き)有向グラフ G = (V,E) を考える
- 頂点集合 V の任意の分割 S, V\S に対し、S から V\S への辺の集合を**カットセット**と呼ぶ



グラフカット

- G の最小カットとは、任意のカットセットのうち重み和が最小のもの
- 特に、与えられた s, t に対し、s ∈ S, t ∈ V\S であるカットを s-t カットという

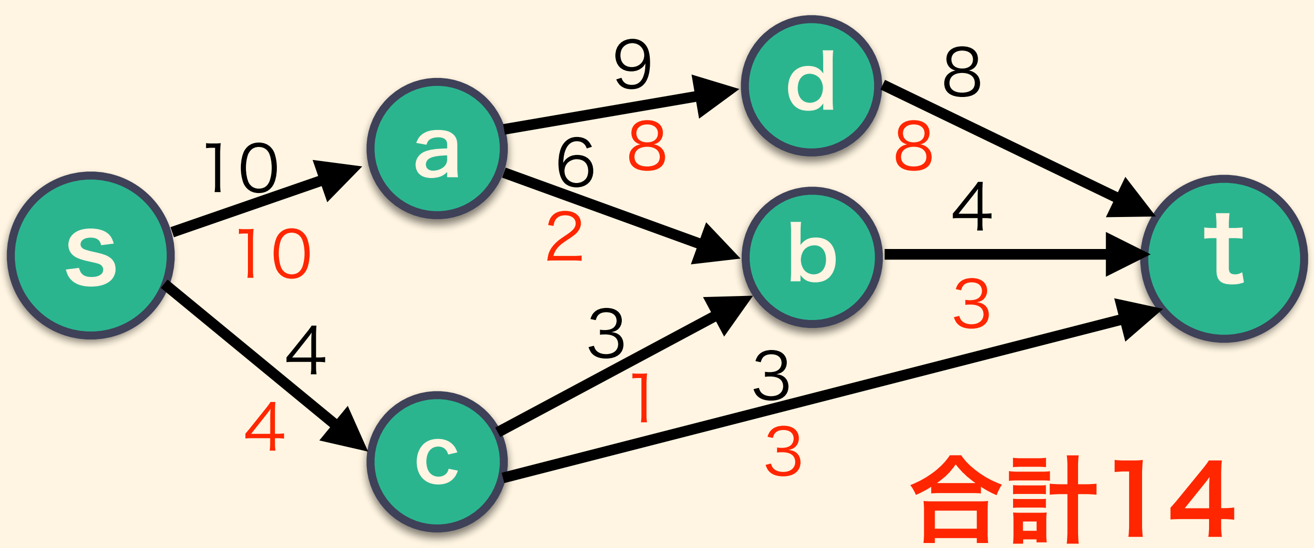


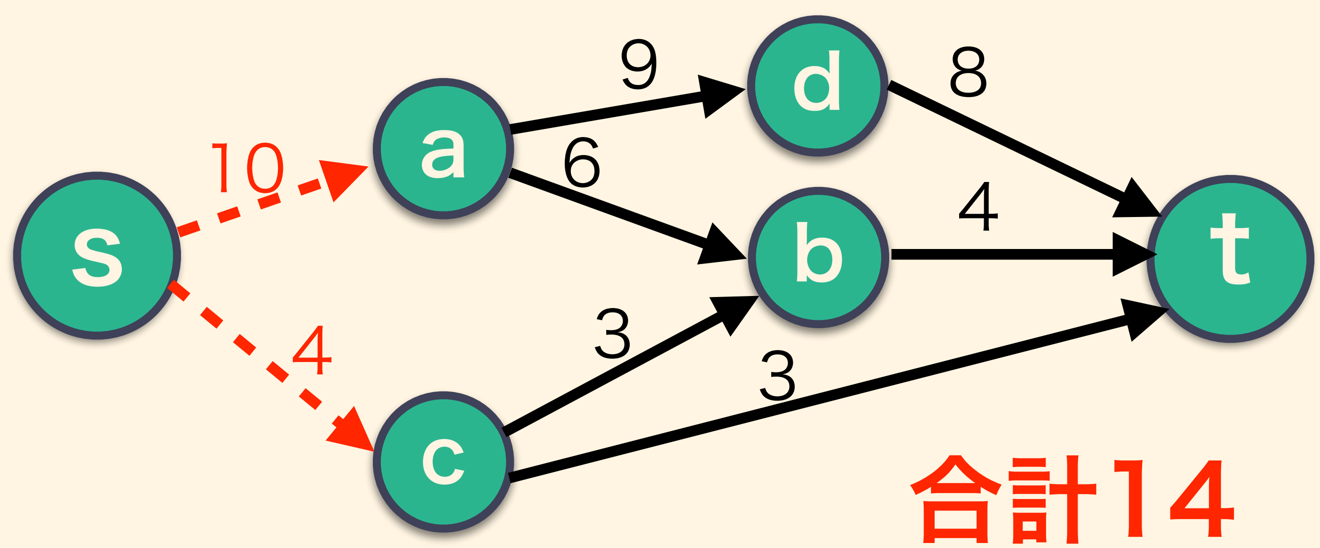
内容

- 最大流と最小カットの定義
- **最大流 - 最小カット定理**
- 最大流を求めるアルゴリズム (Ford-Fulkerson, Dinic)
- 最小費用流とその解き方 (SSP)
- 全域最小カット・最大カット

最大流・最小カット定理

- 辺に容量制約がついた有向グラフ G = (V,E) を考える
- 最大 s-t フローの流量 = 最小 s-t カットの容量和
- 証明は後で



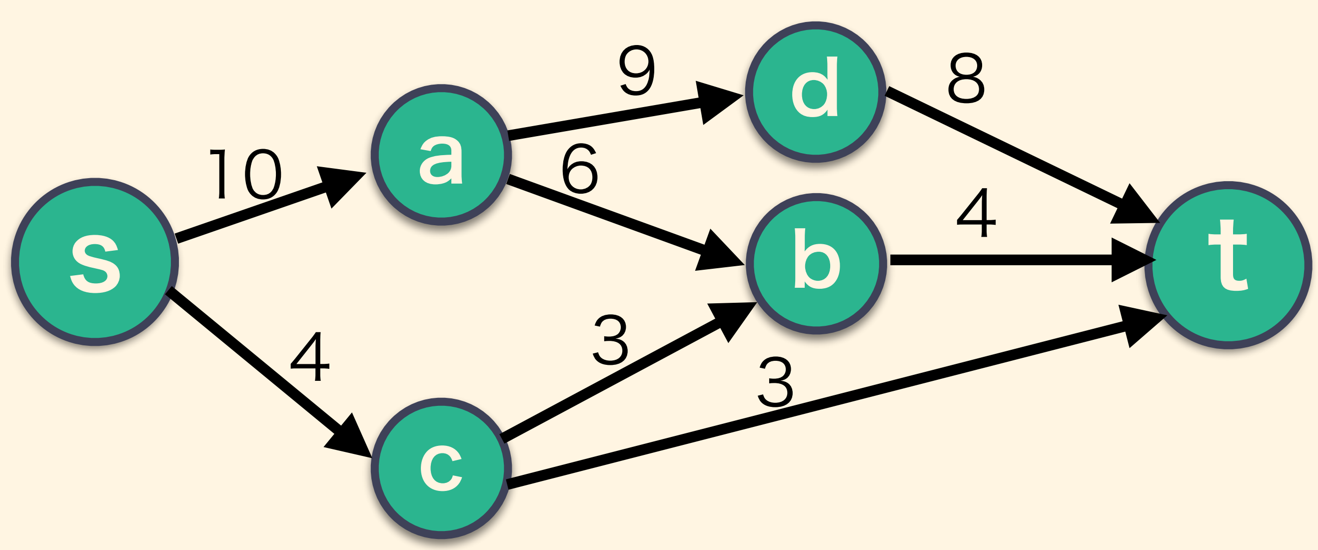


内容

- 最大流と最小カットの定義
- 最大流 - 最小カット定理
- 最大流を求めるアルゴリズム (Ford-Fulkerson, Dinic)
- 最小費用流とその解き方 (SSP)
- 全域最小カット・最大カット

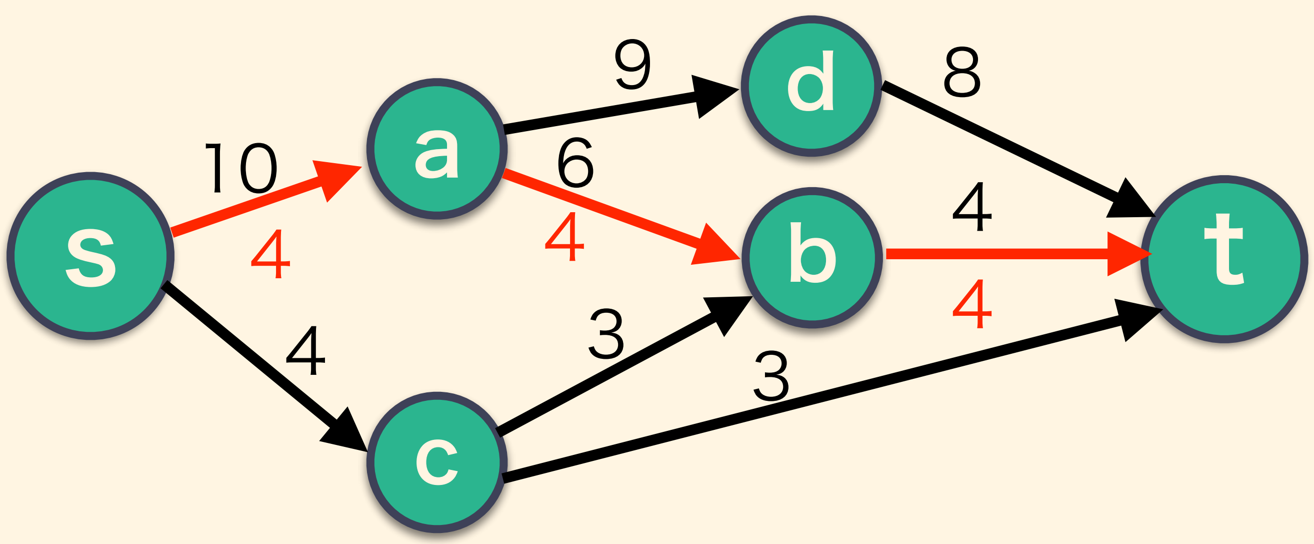
間違った貪欲法

- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す



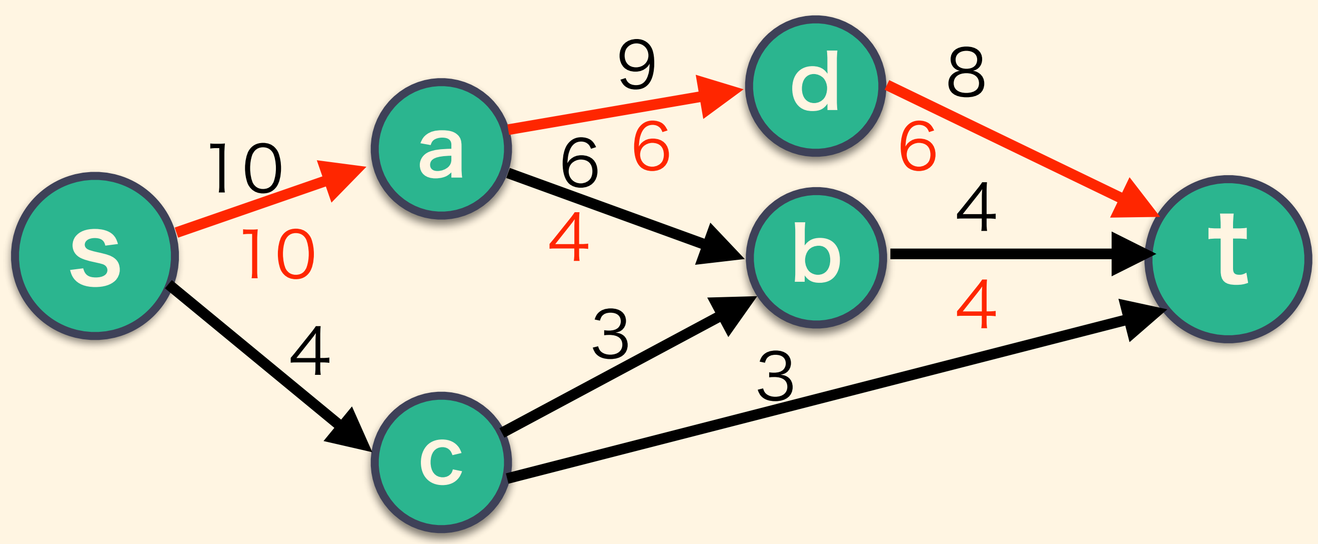
間違った貪欲法

- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す



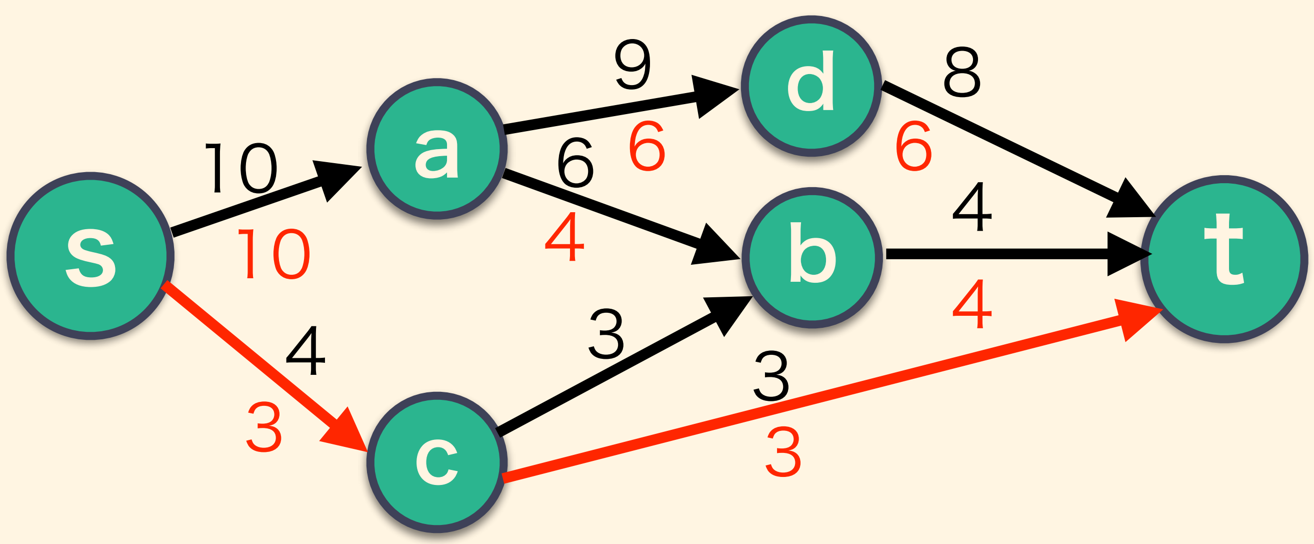
間違った貪欲法

- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す



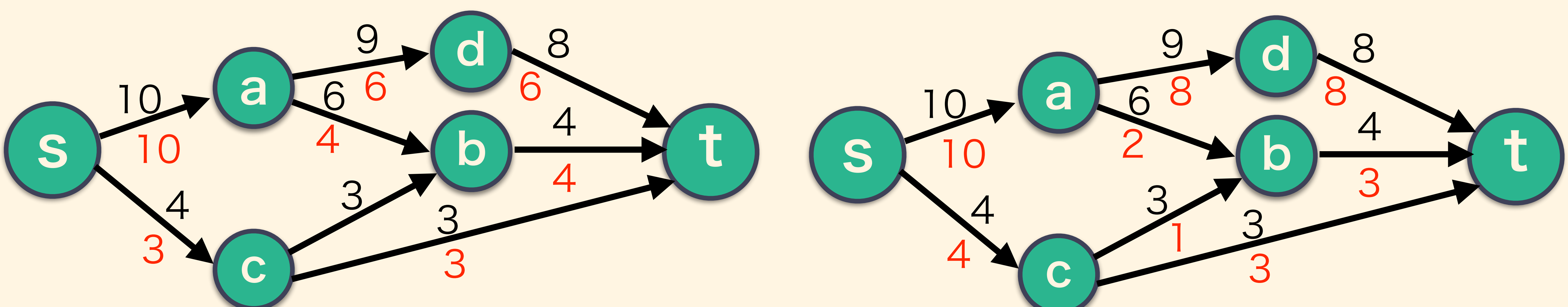
間違った貪欲法

- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す



間違った貪欲法

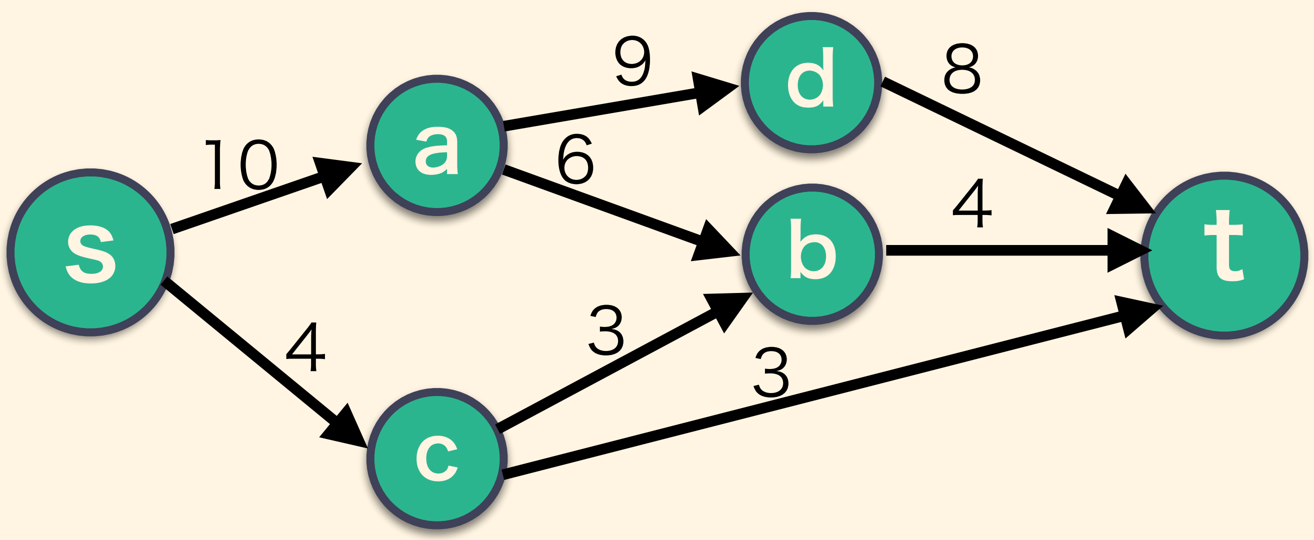
- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す
- 他のパスで流した方がよいところを勝手に先に使ってしまうのでダメ



合計13　<　合計14

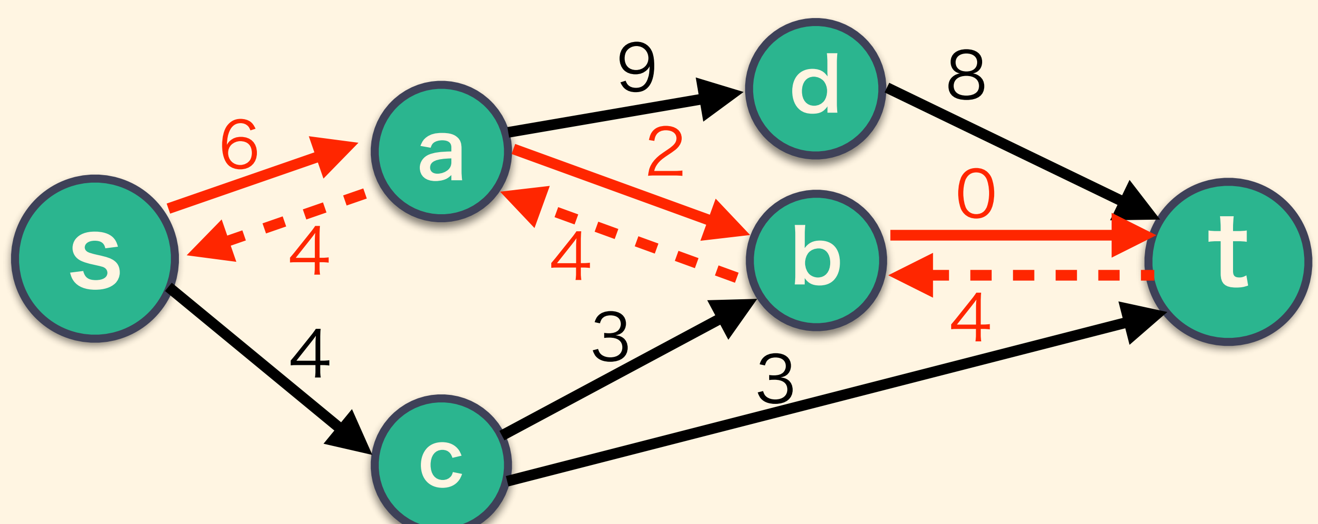
Ford-Fulkerson のアルゴリズム

- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す
- 流れている容量分、逆辺を追加して押し戻せるようにする
- 逆辺込みのグラフを残余グラフという



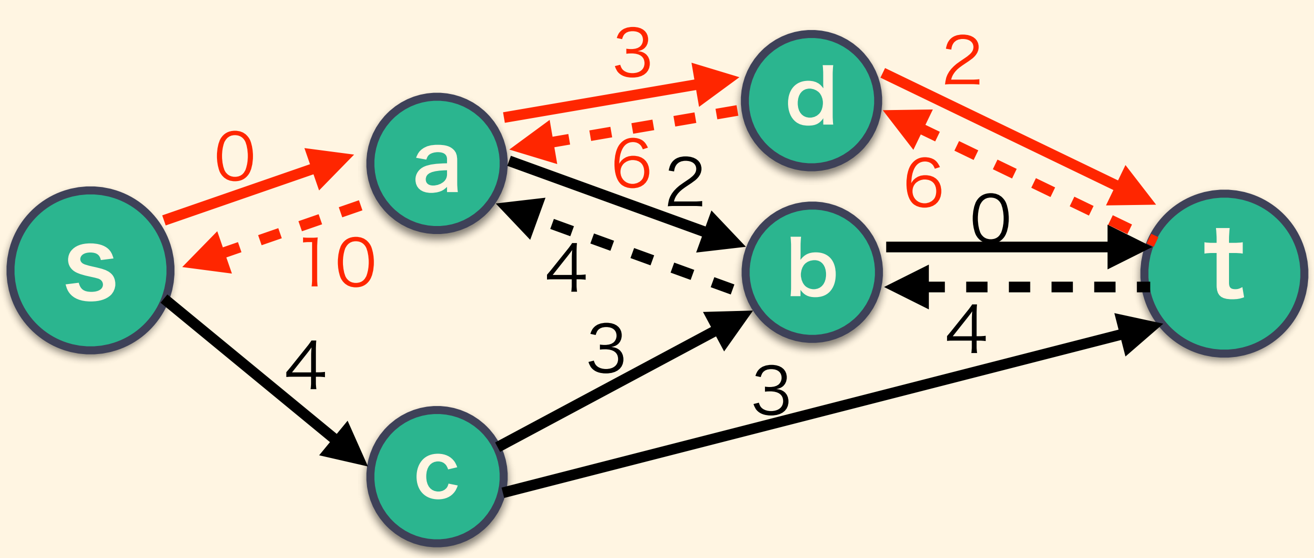
Ford-Fulkerson のアルゴリズム

- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す
- 流れている容量分、逆辺を追加して押し戻せるようにする
- 逆辺込みのグラフを残余グラフという



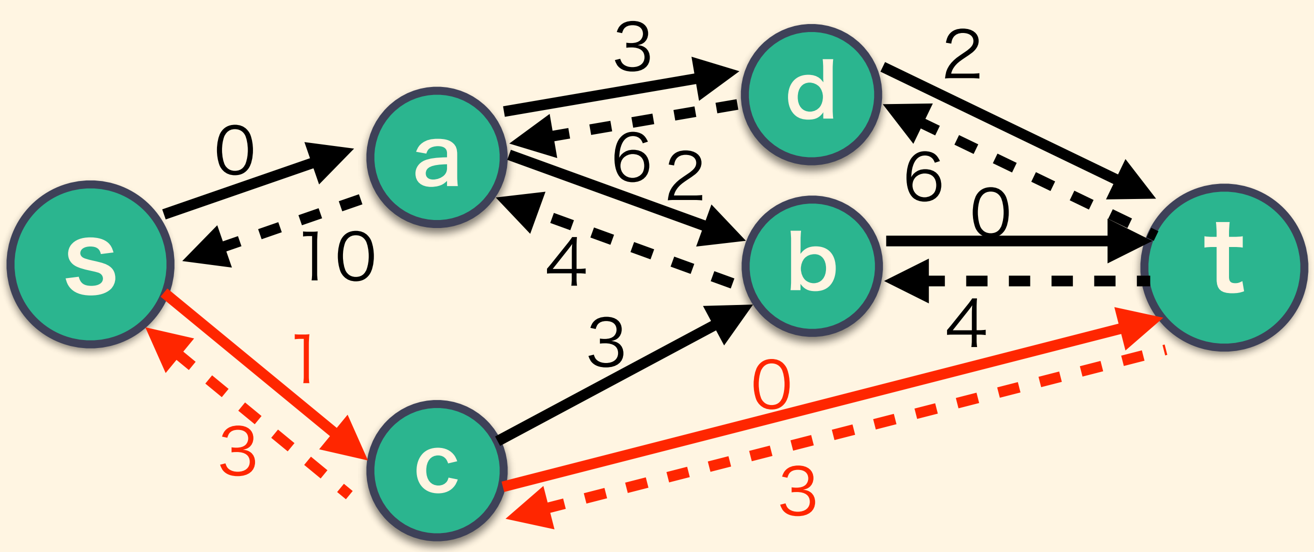
Ford-Fulkerson のアルゴリズム

- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す
- 流れている容量分、逆辺を追加して押し戻せるようにする
- 逆辺込みのグラフを残余グラフという



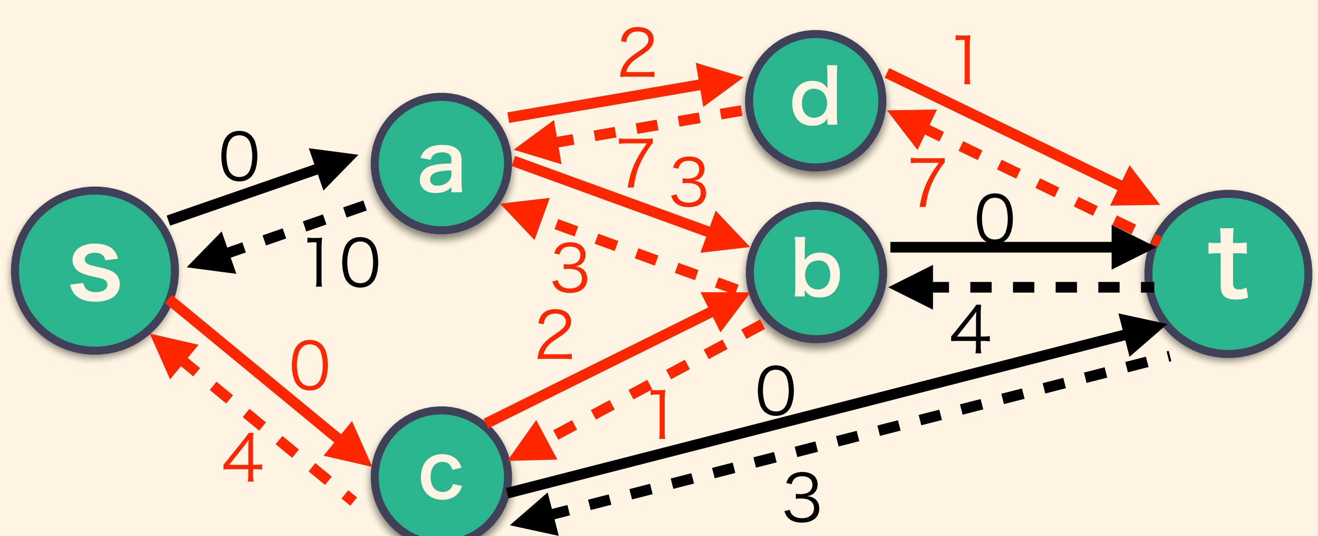
Ford-Fulkerson のアルゴリズム

- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す
- 流れている容量分、逆辺を追加して押し戻せるようにする
- 逆辺込みのグラフを残余グラフという



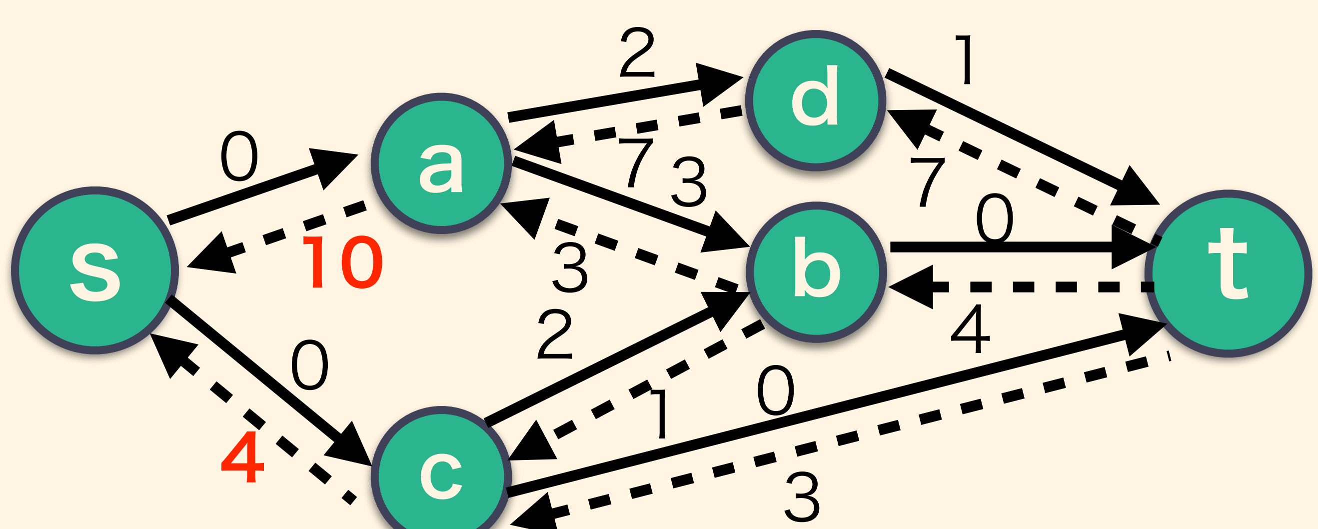
Ford-Fulkerson のアルゴリズム

- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す
- 流れている容量分、逆辺を追加して押し戻せるようにする
- 逆辺込みのグラフを残余グラフという



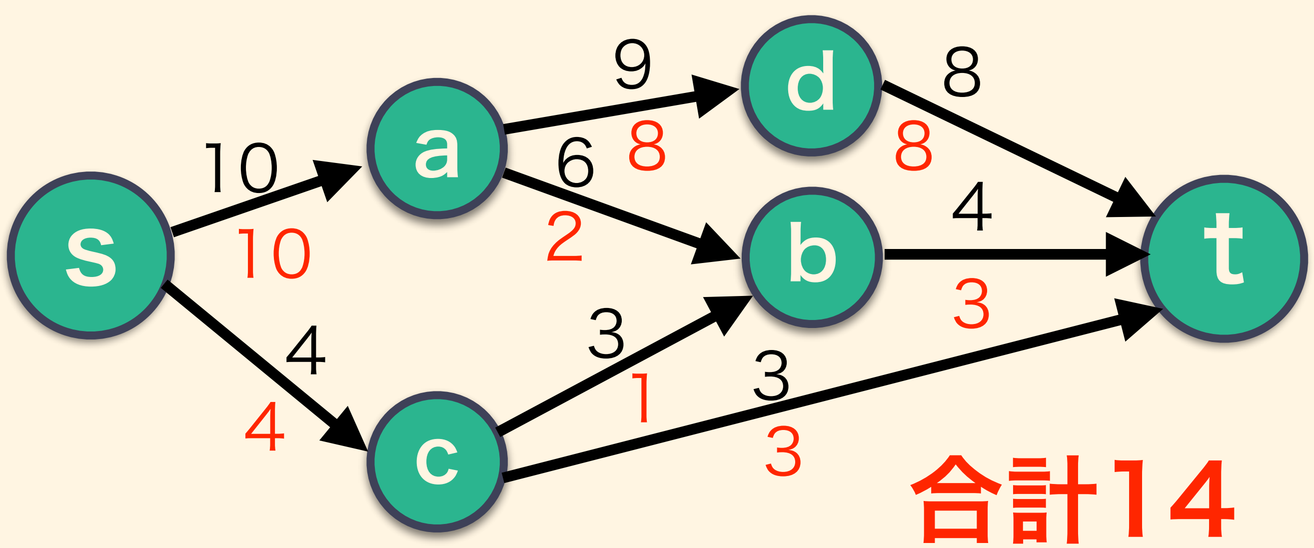
Ford-Fulkerson のアルゴリズム

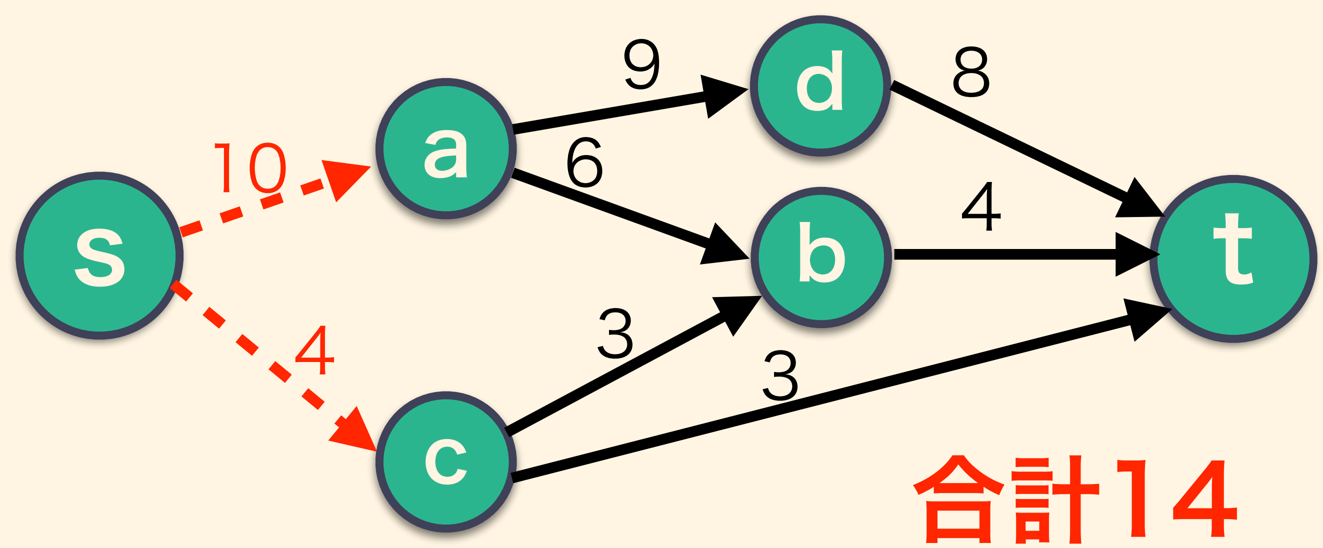
- 容量が余っている辺からなる s-t パスがあれば流せるだけ流す
- 流れている容量分、逆辺を追加して押し戻せるようにする
- 逆辺込みのグラフを残余グラフという



最大流・最小カット定理の証明

- 証明は2つ
  - フローの流量 ≤ カットの容量
  - 最小カットの容量 ≤ 最大流の流量

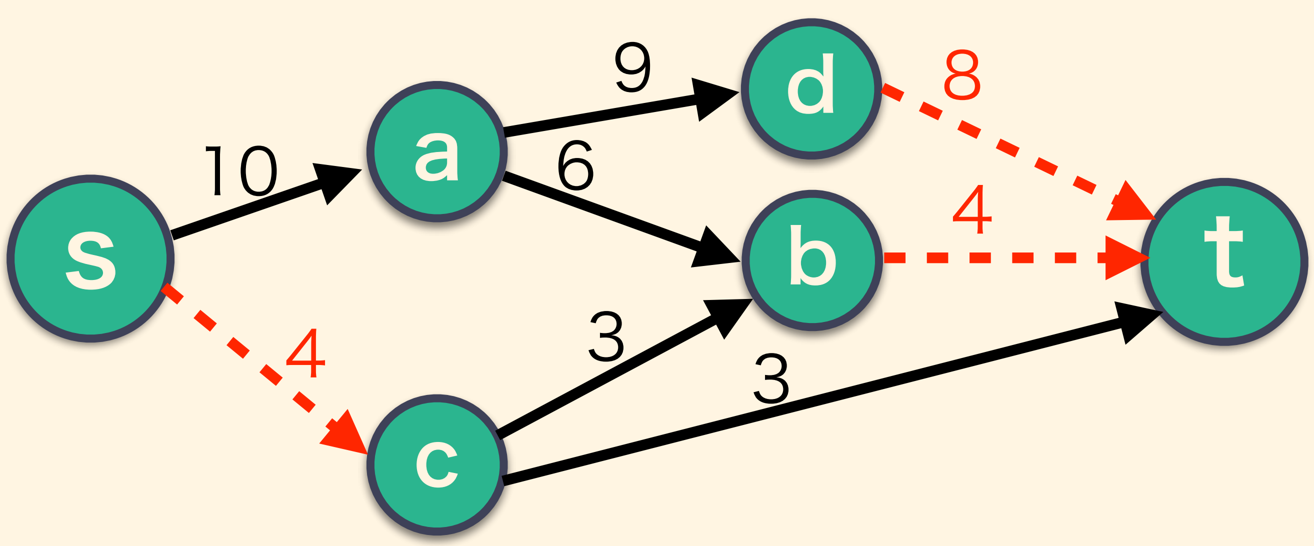




1. フロー ≤ カット

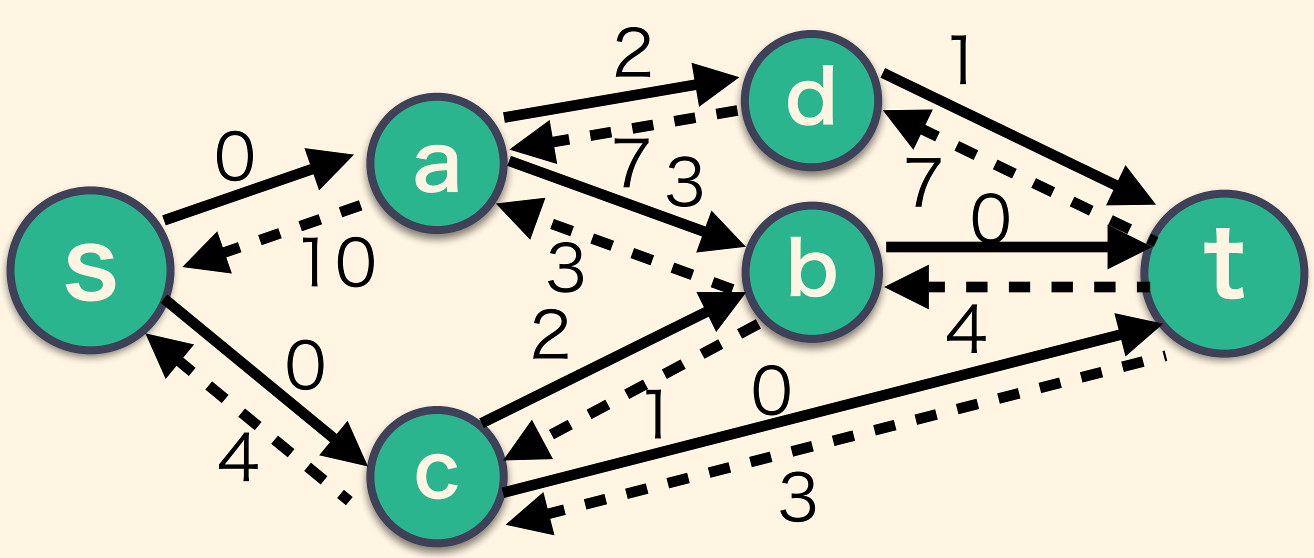
- カットされていると s-t パスがないためフローが流せない

→ カットをやめても最大でカットの容量しか流せない



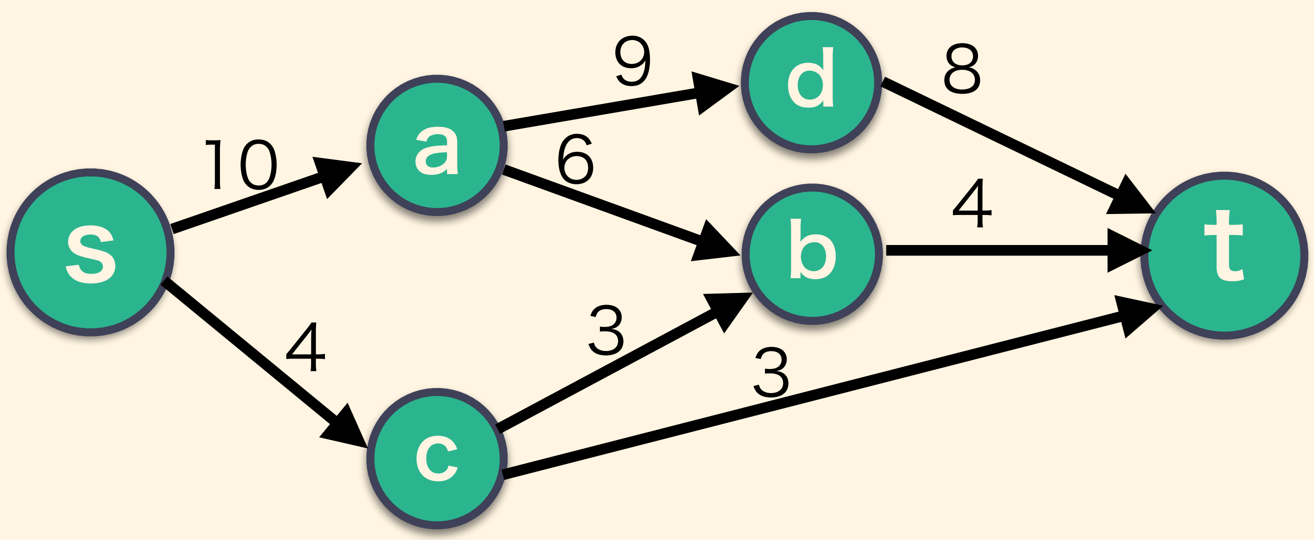
2. 最小カット ≤ 最大流

- 最大流 ⇔ 残余グラフで s-t パスがない = カット
- **最大流の流量 = カットの容量 ≧ 最小カットの容量**
  - S から V\S への辺の流量 = 容量
  - V\S から S への辺の流量 = 0 (= 逆辺容量も0)



Ford-Fulkerson のアルゴリズム

- 残余グラフに s-t パスが存在する
  →めいっぱい流した後、逆辺を張って
  残余グラフを更新
- 以上をパスがある限り繰り返す
- O(|F| |E|) : |F| = 最大流量、|E| = 辺数



Ford-Fulkerson の実装 (グラフ構築)

```
struct edge{
  int to, cap, rev;
  edge(int t, int c, int r){
    to = t; cap = c; rev = r;
  }
};

void add_edge(int from, int to, int cap, vector< vector<edge> > &g){
  g[from].push_back( edge(to,cap,g[to].size()) );
  g[to].push_back( edge(from,0,g[from].size()-1) );
}
```

Ford-Fulkerson の実装 (本体)

```
int dfs(int v,int t, int f, vector< vector<edge> > &g, vector<int> &used){
  if(v==t)return f;
  used[v] = 1;
  for(edge &e : g[v]){
    if(!used[e.to] && e.cap>0){
      int d = dfs(e.to,t,min(f,e.cap),g,used);
      if(d>0){
        e.cap -= d;
        g[e.to][e.rev].cap += d;
        return d;
      }
    }
  }
  return 0;
}

int max_flow(int s, int t, vector< vector<edge> > g){
  int flow = 0;
  while(1){
    vector<int> used(g.size(),0);
    int f = dfs(s,t,INF,g,used);
    if(f==0)return flow;
    flow += f;
  }
}
```

フローいろいろ

- 無向グラフのとき
  - 双方向に辺を張ればよい
- 複数始点、複数終点がある
  - 新たな始点、終点1つ用意して、それらにつなぐ
- 複数始点、複数終点かつ、始点・終点ペアが決まっている
  - 多品種フローというNP完全問題、Ford-Fulkersonでは解けない
- 最小流量制約がある
  - 蟻本を読もう！
- もっと高速に解きたい
  - Dinic のアルゴリズム $O(|E|\,|V|^2)$ が知られている ← 蟻本を読もう！
  - Push-relabel アルゴリズム $O(|V|^3)$ が知られている ← not 蟻本

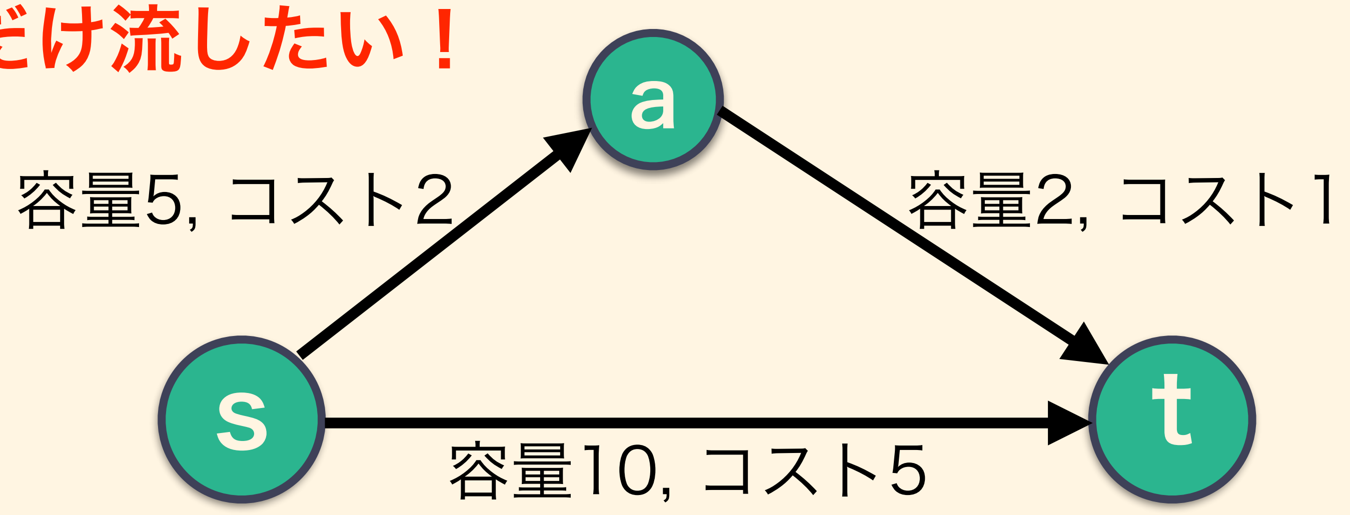
内容

- 最大流と最小カットの定義
- 最大流 - 最小カット定理
- 最大流を求めるアルゴリズム (Ford-Fulkerson, Dinic)
- 最小費用流とその解き方 (SSP)
- 全域最小カット・最大カット

最小費用流

- 辺に容量制約、コストがついた有向グラフ G = (V,E) を考える
- 辺に流す流量1につき、辺についたコストがかかる
- 始点 (ソース) s から終点 (シンク) t へ、各辺の容量制約を守って流量 f まで t へ流すとき、最小のコスト和はいくらか？

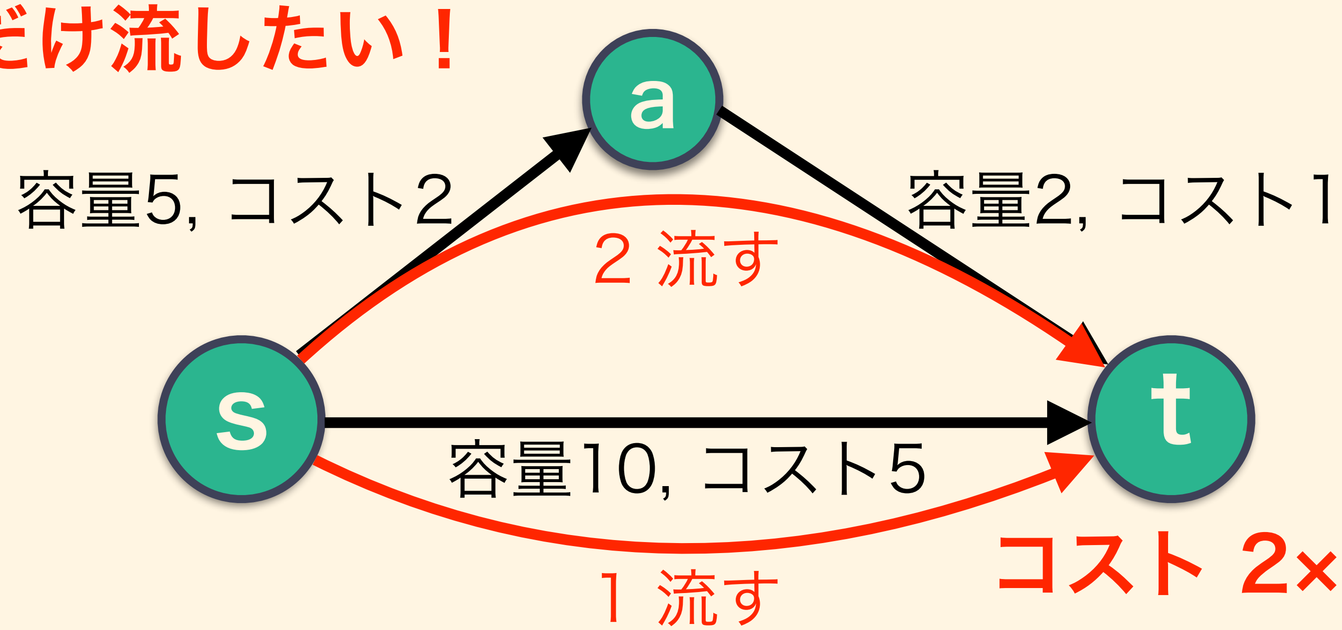
流量 f = 3 だけ流したい！



最小費用流

- 辺に容量制約、コストがついた有向グラフ G = (V,E) を考える
- 辺に流す流量1につき、辺についたコストがかかる
- 始点 (ソース) s から終点 (シンク) t へ、各辺の容量制約を守って流量 f まで t へ流すとき、最小のコスト和はいくらか？

流量 f = 3 だけ流したい！



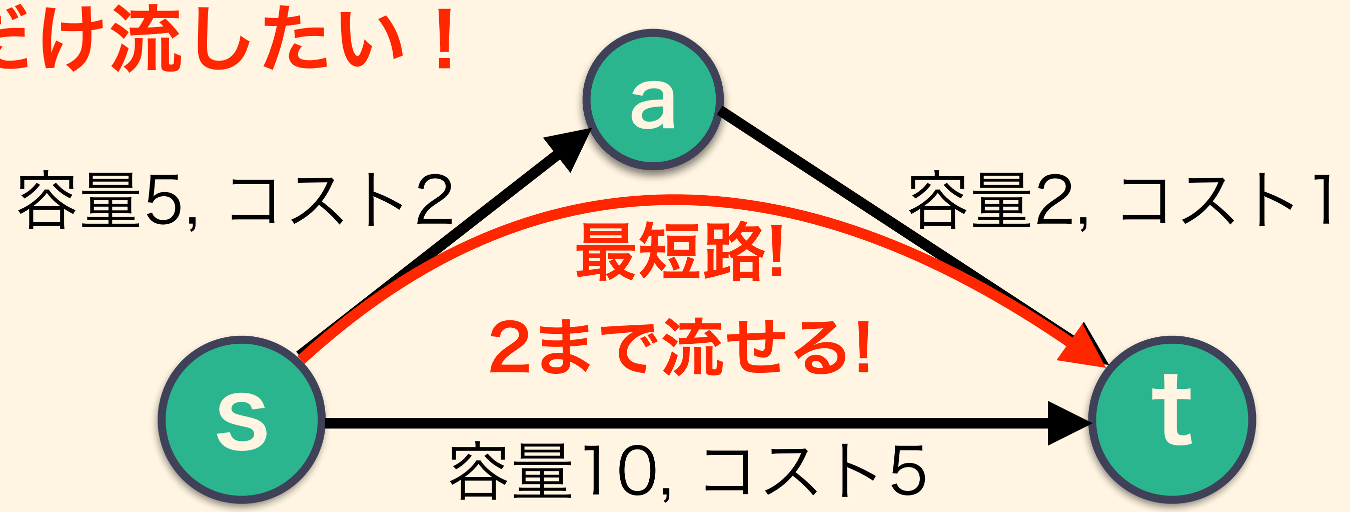
コスト 2×(2+1) + 1×5 =11

最小！！！

SSP (Successive Shortest Path)

- 容量が余っている辺からなる s-t 最短路に流せるだけ流す
- 流れている容量分、逆辺を追加して押し戻せるようにする
- 逆辺のコストは正負を逆転する

流量 f = 3 だけ流したい！



SSPの正当性

- コンテスタントがじゃぶじゃぶ最小費用流を流したくなるような射幸心を煽りまくる証明

SSPの正当性

帰納法 : f_x が x 流す時の最小費用流

→ 最短路にさらに1だけ流したf_{x+1}は？

背理法: f_{x+1}が最小費用流でないと仮定

→ もっとコストの小さいフロー f'_{x+1}が存在

- $f'_{x+1} - f_x$ は s-t パスといくつかの閉路
- f'_{x+1} は f_{x+1} よりコストが小さい → 負の閉路が存在

→ f_x に負の閉路があった → 補題に矛盾 → f_{x+1}は最小費用流

- 補題: 最小費用流の残余グラフは負の閉路を持たない

↑ 閉路に流しまくった方がコストが下がる

SSPのアルゴリズム

- 残余グラフに s-t パスが存在する
  → s-t 最短路を求めて目一杯流した後、
  逆辺を張って残余グラフを更新
- 以上をパスがある限り繰り返す
- 基本的に Ford-Fulkerson の dfs を最短路アルゴリズムに変えるだけ
- O(|F| ×最短路計算量) : |F| = 最大流量
- 逆辺が負値になるので Bellman-Ford (O(VE)) を使うのが基本
  - ポテンシャル法の利用で Dijkstra (O(E+VlogV)) が使える ← 割愛

SSP (Bellman-Ford) の実装 (グラフの構築)

```
struct edge{
  int to, cap, cost, rev;
  edge(int t, int c, int v, int r){
    to = t; cap = c; cost = v; rev = r;
  }
};

void add_edge(int from, int to, int cap, int cost, vector< vector<edge> > &g){
  g[from].push_back( edge(to,cap,cost,g[to].size()) );
  g[to].push_back( edge(from,0,-cost,g[from].size()-1) );
}
```

SSP (Bellman-Ford) の実装 (最短路部分)

```
//Bellman-Ford based: O(FVE) , 入力: 隣接リスト
int min_cost_flow(int s, int t, int f, vector< vector<edge> > g){
  int n = g.size(), res = 0;

  while(f>0){
    //最短路 (復元付き) を求める
    vector<int> d(n,INF), pv(n), pe(n);
    d[s] = 0;

    bool update = true;
    while(update){
      update = false;
      for(int v=0;v<n;v++){
        for(int i=0;i<(int)g[v].size();i++){
          edge &e  = g[v][i];
          if(e.cap>0 && d[e.to] > d[v] + e.cost){
            d[e.to] = d[v] + e.cost;
            pv[e.to] = v; pe[e.to] = i;
            update = true;
          }
        }
      }
    }
```

SSP (Bellman-Ford) の実装 (グラフ更新部分)

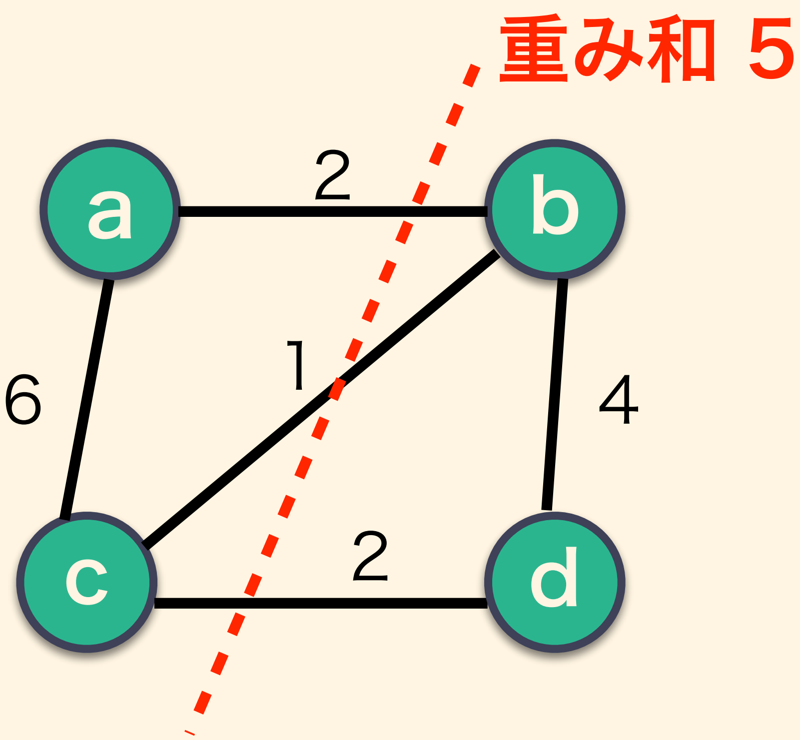
```
    // f 流すには容量が足りない
    if(d[t]==INF)return -1;

    //流せる流量の上限を求める
    int x = f;
    for(int v=t; v!=s; v=pv[v]){
      x = min(x, g[pv[v]][pe[v]].cap);
    }
    f -= x;
    res += x*d[t];

    //実際に流す
    for(int v=t; v!=s; v=pv[v]){
      edge &e = g[pv[v]][pe[v]];
      e.cap -= x;
      g[e.to][e.rev].cap += x;
    }
  }
  return res;
}
```

内容

- 最大流と最小カットの定義
- 最大流 - 最小カット定理
- 最大流を求めるアルゴリズム (Ford-Fulkerson, Dinic)
- 最小費用流とその解き方 (Primal-Dual)
- 全域最小カット・最大カット

全域最小/最大カット

- 正の重み付き無向グラフ G = (V,E) を考える
- 任意の非空集合 S ⊂ V におけるカットのうち、重み和最小 (最大) のものを全域最小 (最大) カットという
  - s, t の縛りがない



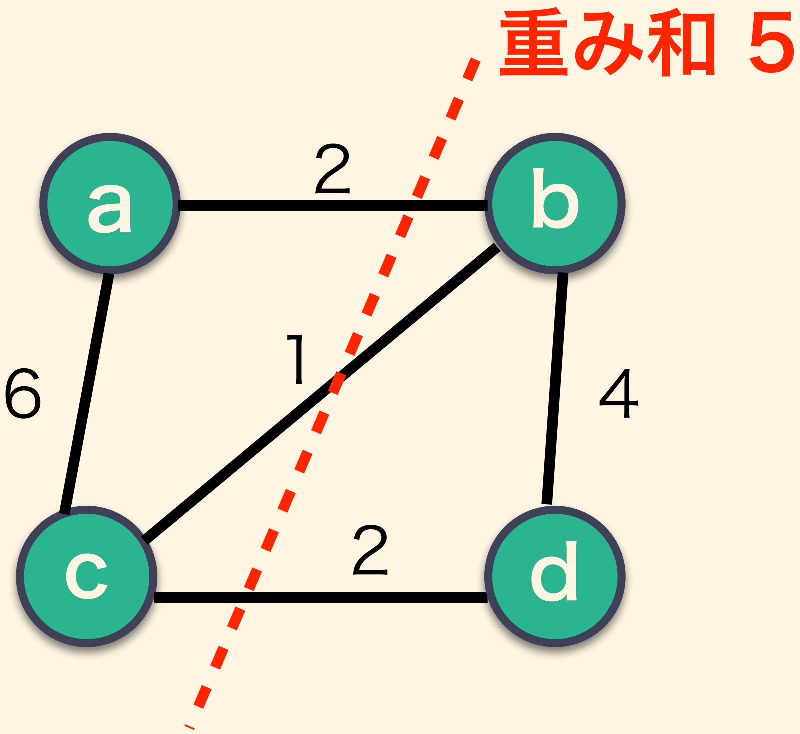
全域最小/最大カット

- 残念なお知らせ:

  (全域) 最大カットはNP完全

- 朗報:

  全域最小カットは $O(VE + V^2 \log V)$ で解ける



Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

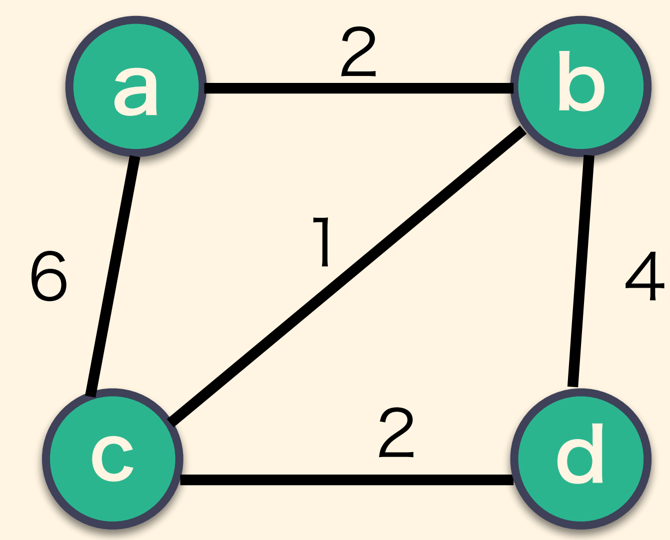
1. 適当な頂点1つをリスト L に入れる
2. まだ L に入れてない頂点がある限り、
   - L の頂点との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との間の辺重み和を W とおく
4. $L_{|V|}$ と $L_{|V|-1}$ を1つの頂点にまとめて G' とする
5. min(W, MinCostCut(G')) が答え

Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = ( )

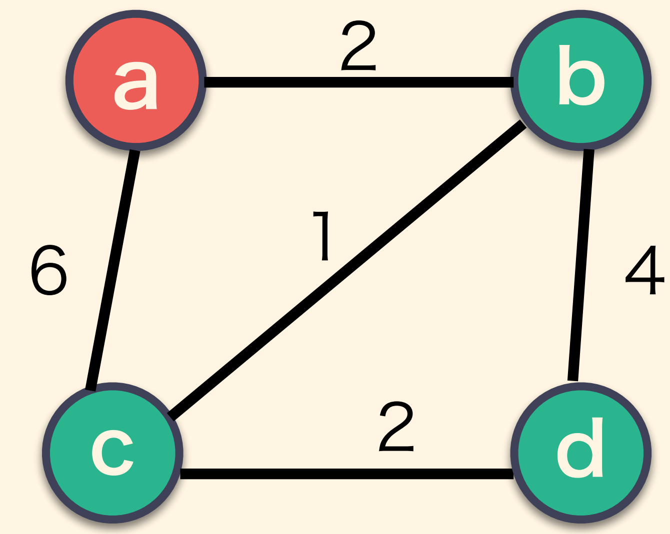


Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a)

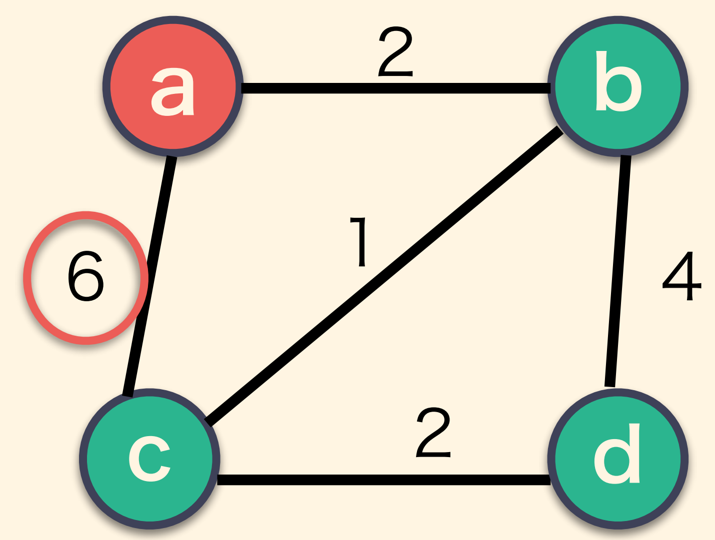


Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a)

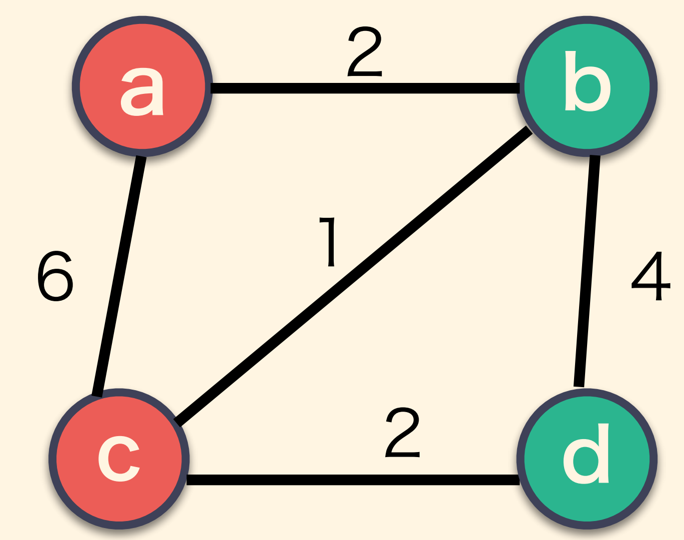


Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, c)

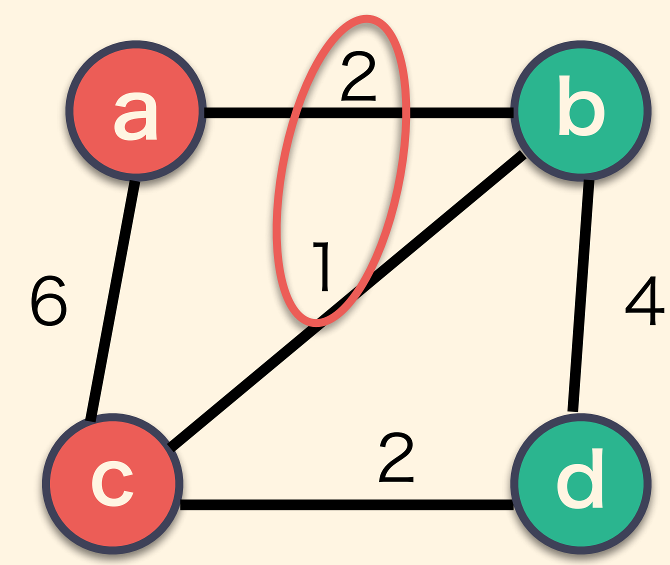


Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, c)

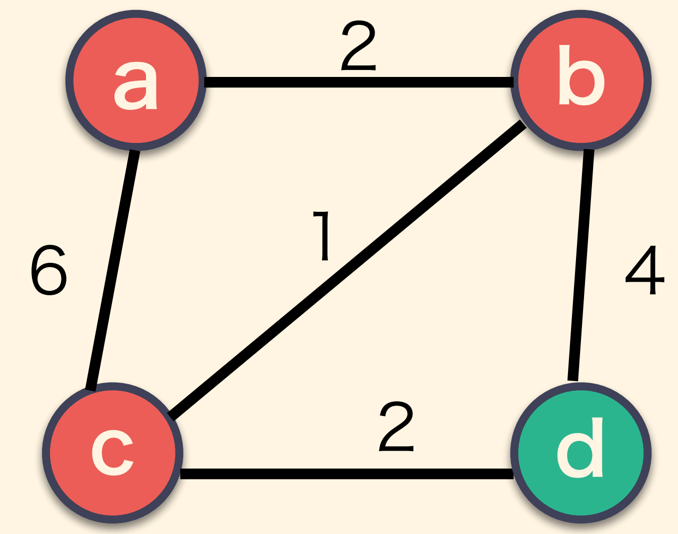


Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, c, b)

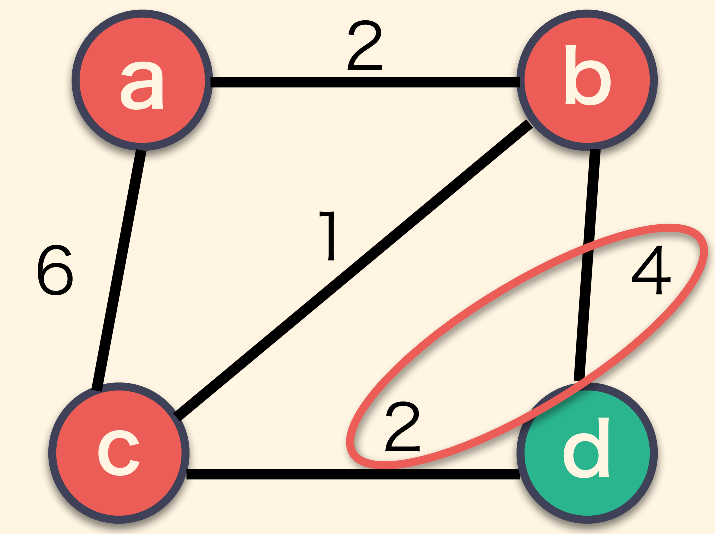


Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, c, b)

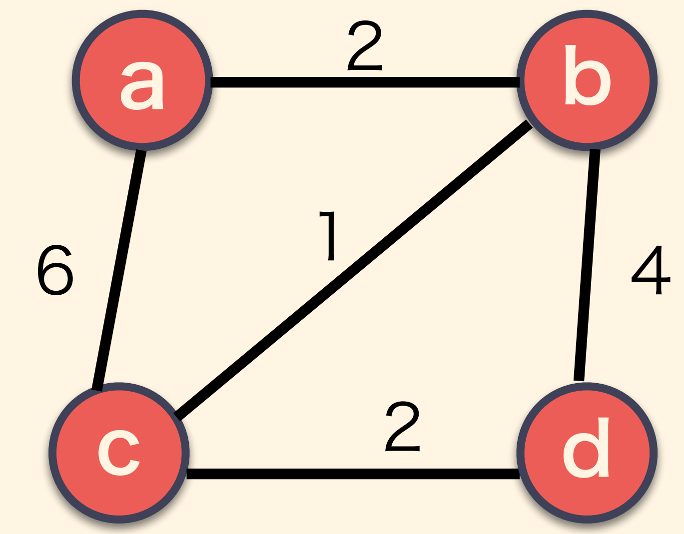


Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, c, b, d)

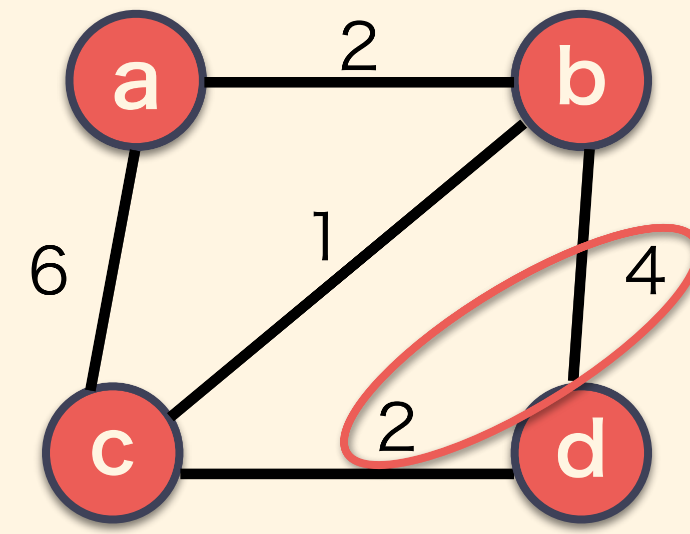


Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, c, b, d)



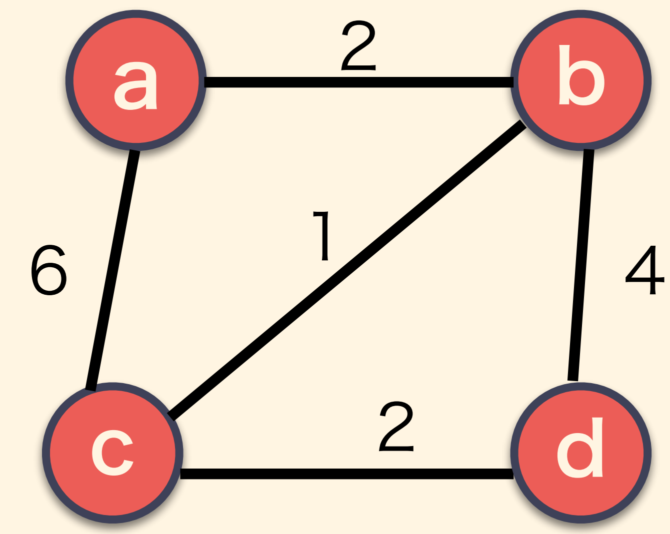
W = 6

Stoer-Wagner のアルゴリズム

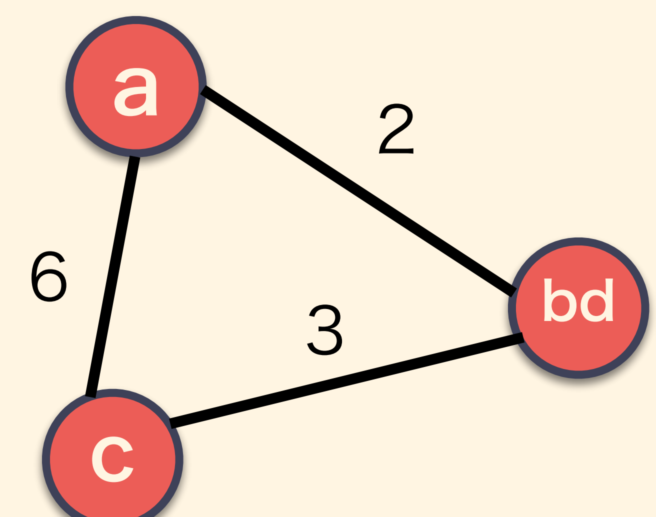
MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, c, b, d)



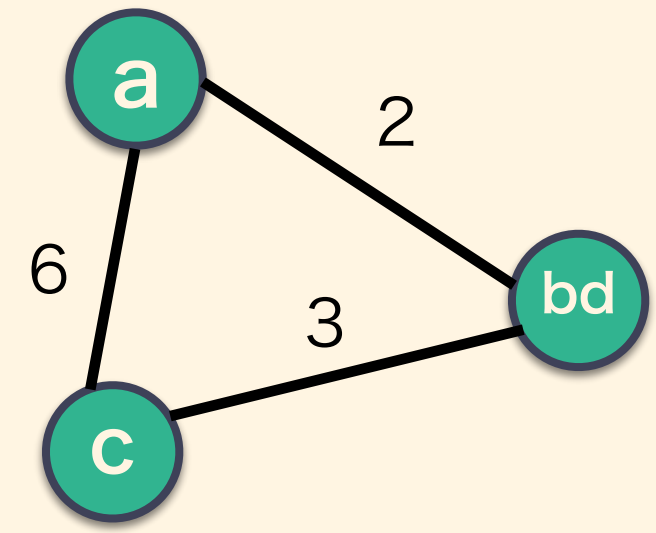
W = 6



Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え



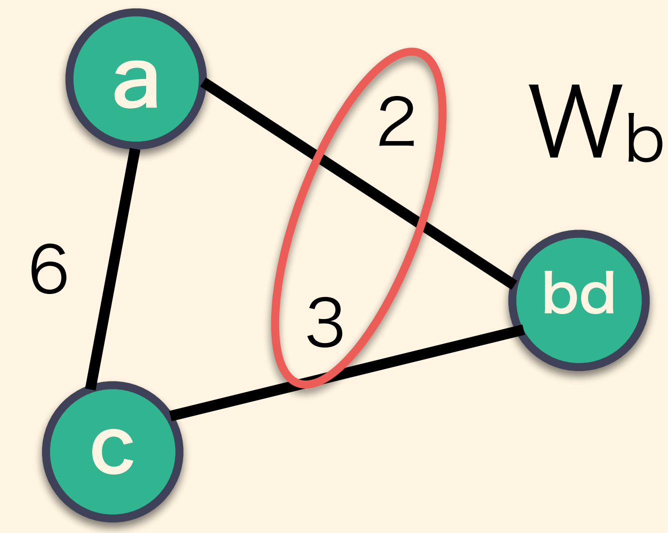
$W_d = 6$

Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, c, bd)



$W_{bd} = 5$

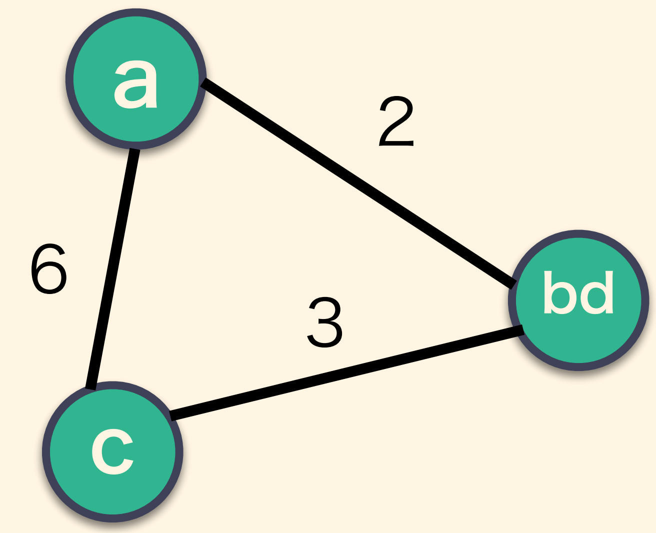
$W_d = 6$

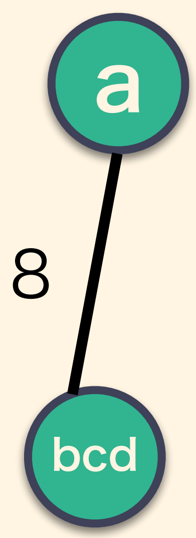
Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, c, bd)





$W_{bd} = 5$

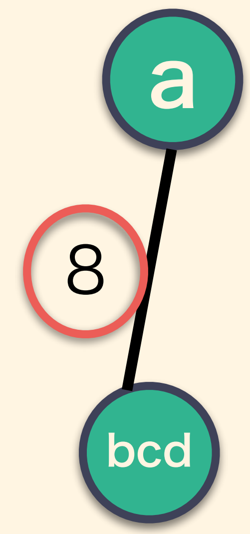
$W_d = 6$

Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, bcd)



$W_a = 8$

$W_{bd} = 5$

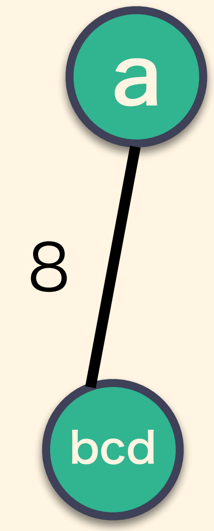
$W_d = 6$

Stoer-Wagner のアルゴリズム

MinCostCut (G = (V, E)):

1. 適当な頂点をリスト L に入れる
2. L に入れてない頂点がある限り、
   - L との辺重み和が最大の頂点を L に加える
3. $L_{|V|}$ と他の頂点との辺重み和を W
4. $L_{|V|}$ と $L_{|V|-1}$ を1つにまとめて G'
5. min(W, MinCostCut(G')) が答え

L = (a, bcd)



$W_a = 8$

最小全域カット = 5

$W_{bd} = 5$

$W_d = 6$

Stoer-Wagner の正当性

- 最大隣接順序 (Maximum Adjacent Order) で頂点を並べた最後の2頂点を s, t とする
- 以下の2つの定理が成り立てば正しい
  1. t から他の全頂点への辺和は最小 s-t カットに等しい
  2. G の最小カットは、G'の最小カットか最小 s-t カットかのいずれか

Stoer-Wagner の正当性

1. t から他の全頂点への辺和は最小 s-t カットに等しい

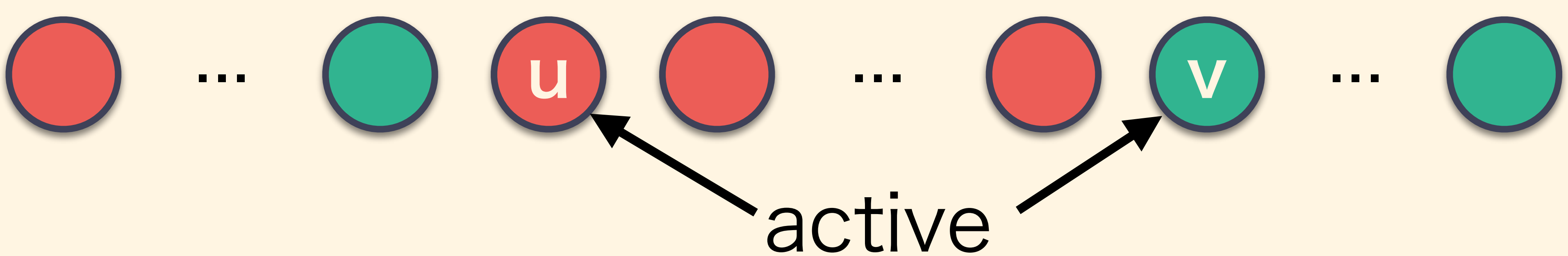
- 任意の s-t カット C について、$w(A_t, t) \leq w(C_t)$ を言えばよい
  - A_v : MA Order で v より前の頂点集合
  - $w(A,v)$: Aとvの間の辺の重み和
  - C_v : Cのうち、$A_v \cup \{v\}$ 間の辺のみの集合
  - $w(C)$: 辺集合 C に含まれる辺の重み和

Stoer-Wagner の正当性

1. t から他の全頂点への辺和は最小 s-t カットに等しい

- active な頂点に関する帰納法
  - v が active : カット C 中で v と1つ前の頂点が違う集合に分けられている
  - vの直前の active な頂点u が $w(A_u,u) \leq w(C_u)$ と仮定

    → $w(A_v,v) \leq w(C_v)$ が言えればよい



Stoer-Wagner の正当性

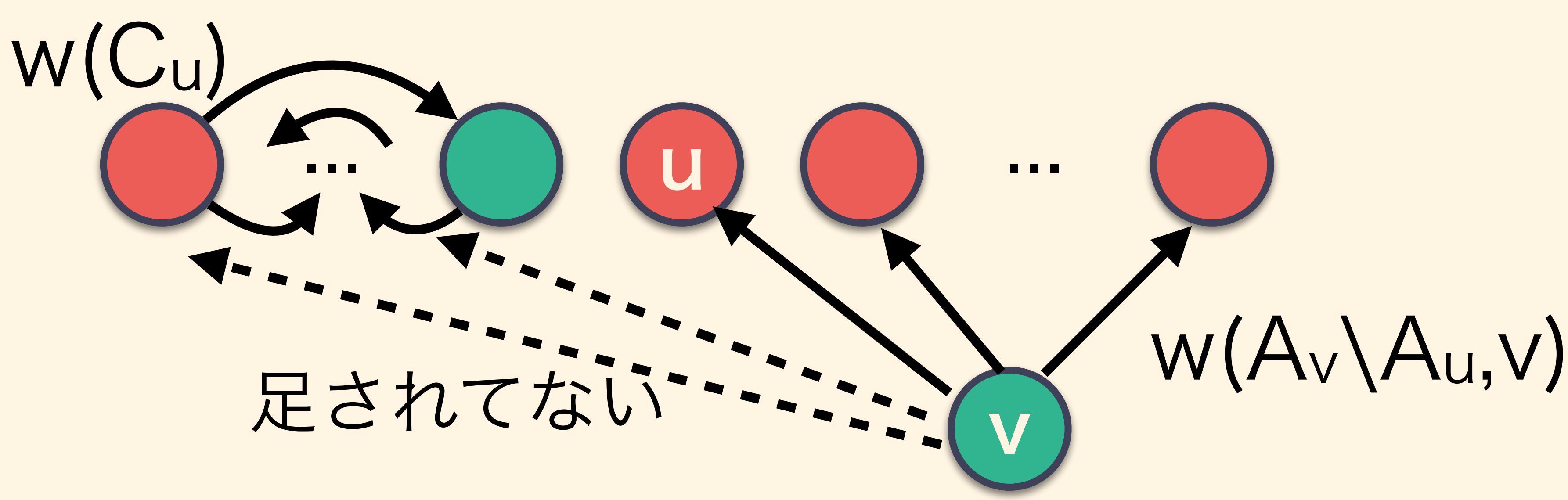
1. t から他の全頂点への辺和は最小 s-t カットに等しい

- 帰納法 : $w(A_v,v) \le w(C_v)$ を言いたい
  - $w(A_v,v) = w(A_u,v) + w(A_v \backslash A_u,v)$ ← 集合分けただけ

    $\le w(A_u,u) + w(A_v \backslash A_u,v)$ ← MA Order

    $\le w(C_u) + w(A_v \backslash A_u,v)$ ← 帰納法仮定

    $\le w(C_v)$ ← 下図



Stoer-Wagner の正当性

2. G の最小カットは、G'の最小カットか最小 s-t カットかのいずれか

- まあ自明
  - G'の最小カット = s と t を分けない最小カット
  - つまり、最小カットは s と t を分ける最小カットか分けない最小カットかのいずれか、と言ってるだけ

Stoer-Wagnerの計算量

- MinCostCut の呼び出しはちょうど|V|-1回
- 計算量は MA 順序の計算と頂点縮約に依存
  - MA 順序: ダイクストラっぽくできる
    $\rightarrow$ $O(|V|^2)$ か普通のヒープで$O(|E|\log|V|)$
    (フィボナッチヒープを使って $O(|E| + |V|\log|V|)$)
  - 縮約: 適当にやっても$O(|E|)$
- 全体で $O(V^3)$ or $O(|V||E| \log|V|)$

Stoer-Wagner の実装

```
//入力: n×n隣接行列
int global_min_cut(vector< vector<int> > g){
  int n = g.size(), res = INF;
  vector<int> redV(n);
  rep(i,n)redV[i] = i;

  for(int rem=n;rem>1;rem--){
    //calc MA order
    int u=0, v=0, cut=0;
    vector<int> w(rem,0);
    for(int i=0;i<rem;i++){
      u = v; v = max_element(w.begin(), w.end()) - w.begin();
      cut = w[v]; w[v] = -1;
      for(int p=0;p<rem;p++){
        if(w[p]>=0)w[p] += g[redV[v]][redV[p]];
      }
    }

    //merge graph
    rep(i,rem){
      g[redV[u]][redV[i]] += g[redV[v]][redV[i]];
      g[redV[i]][redV[u]] += g[redV[i]][redV[v]];
    }
    redV.erase(redV.begin() + v);

    //update min_cut
    res = min(res, cut);
  }
  return res;
}
```

まとめ

- グラフネットワークに対する問題とアルゴリズム
  - 最大流 = 最小カット: Ford-Fulkerson O(FE)
  - 最小費用流 : SSP O(FVE)
  - 全域最小カット: Stoer-Wagner $O(V^3)$ or O(VElogV)
- まだまだいろいろあります (そのうち)