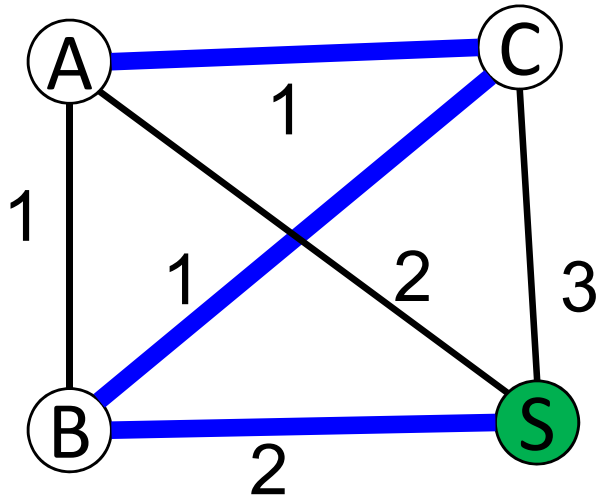
数理経済学
第６回
最小全域木問題

塩浦昭義
東京工業大学 経営工学系 准教授
shioura.a.aa@m.titech.ac.jp

応用：最小全域木構築の費用分担

- ネットワークを構築したい➔最小全域木を作る➔総費用4
  - Sは基地局（ソース）
  - その他の施設（A, B, C）の利用者（プレイヤー）で費用を負担
  - 費用をどう分担するか？



条件1：全プレイヤーの支払額の合計
＝最小全域木の総費用

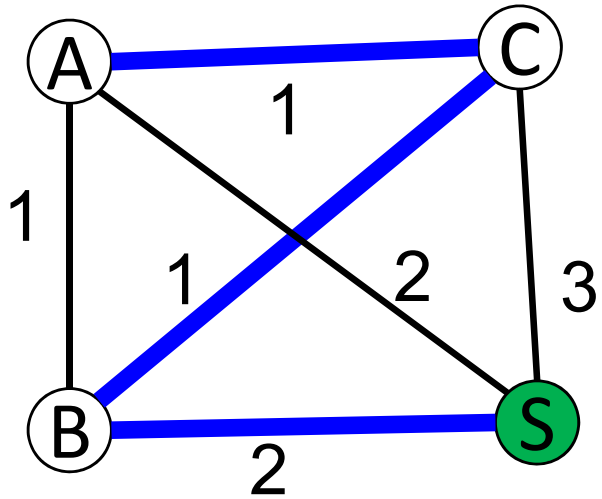
例：$(p_A, p_B, p_C) = (3,1,0)$

条件2：「公平な分担にする」

「公平な」費用分担を求める問題：最小全域木ゲーム
（協力ゲームの中の，特性関数形ゲームの特殊ケース）

応用：最小全域木構築の費用分担

- ネットワークを構築したい➔最小全域木を作る➔総費用4
  - Sは基地局（ソース）
  - その他の施設（A, B, C）の利用者（プレイヤー）で費用を負担
  - 費用をどう分担するか？



条件1：全プレイヤーの支払額の合計
＝最小全域木の総費用

例：$(p_A, p_B, p_C) = (3,1,0)$

コア：条件1，2を満たす費用分担

条件2：「公平な分担にする」← いろいろな定式化が可能

よく使われる条件：グループの支払額の合計
≦グループとソースに対する最小全域木の総費用

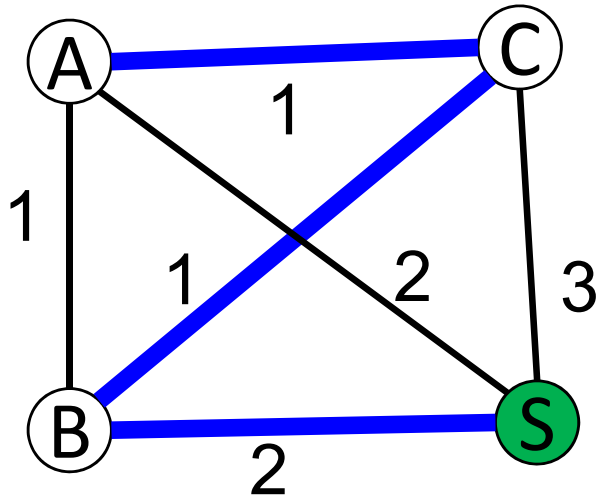
例：A の支払い = 3 ➔ A は単独でネットワークを作った方が費用小

最小全域木ゲームとコア

条件1: 全プレイヤーの支払額の合計＝最小全域木の総費用

条件2: グループの支払額の合計

≦グループとソースに対する最小全域木の総費用



$$p_A + p_B + p_C = 4,$$
$$p_A \leq 2,\ p_B \leq 2,\ p_C \leq 3,$$
$$p_A + p_B \leq 3,\ p_A + p_C \leq 3,\ p_B + p_C \leq 3$$

コア: 条件1，2を満たす費用分担

コアの例: $(p_A, p_B, p_C) = (2,1,1), (1,2,1)$ など

定理 (Bird 1976): 最小全域木ゲームにおいて，コアは常に存在

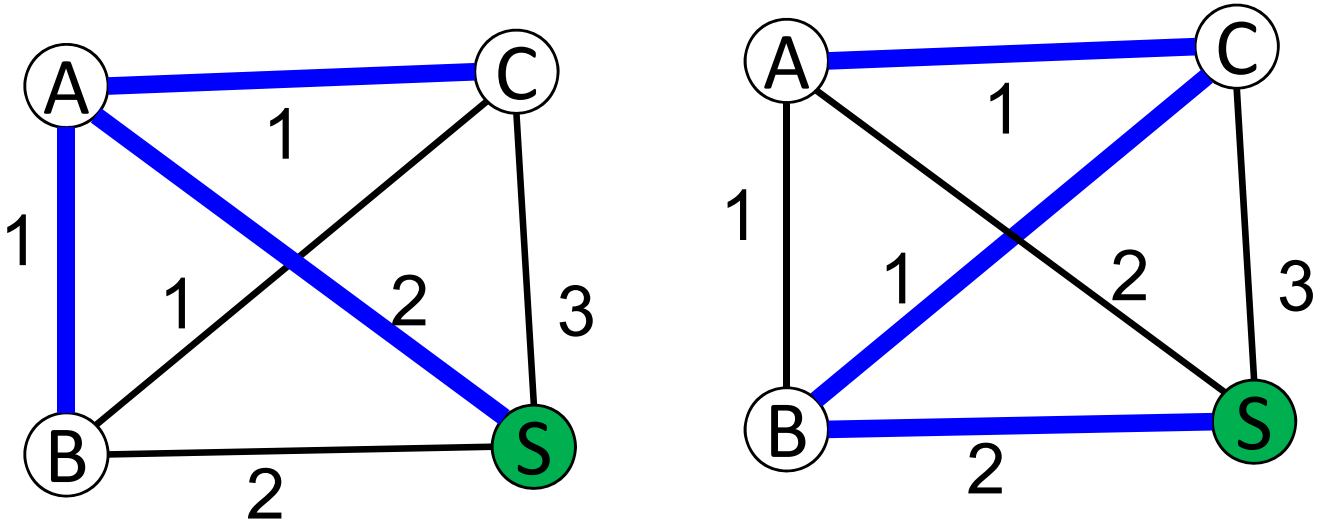
証明: 「Bird 分担」とよばれる費用分担がコアであることを示す

Bird 分担

Bird 分担の定義

- ソース s を根として，プリムのアルゴリズムにより最小全域木を求める
- $V = \{v_1, v_2, \dots, v_n\}$ の順に頂点が追加されたと仮定
- $(v_i, f(v_i))$: v_i が追加されたときの枝（ここで $f(v_i) \in \{v_1, \dots, v_{i-1}\}$）
- v_i の支払額 p_{v_i} =枝$(v_i, f(v_i))$の費用

Bird分担の例: $(p_A, p_B, p_C) = (2,1,1), (1,2,1)$



※Bird分担以外のコアも存在しうる

Bird分担がコアであることの証明

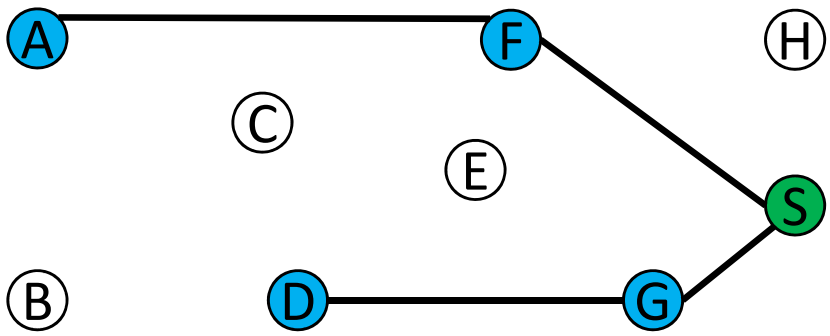
「条件2:グループの支払額の合計
≦グループとソースに対する最小全域木の総費用」
が成り立つことをチェックすれば良い

プレイヤーのグループ W に注目

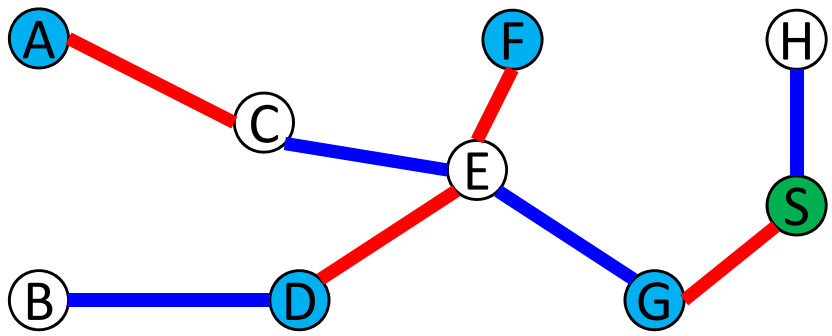
$\alpha = W \cup \{s\}$ に対する最小全域木T'の総費用

β =グループ W の支払額の合計= $\sum_{v \in W} c(v, f(v))$

$\gamma = \sum_{v \in V \setminus W} c(v, f(v))$ ➔プリムによる最小全域木の総費用= $\beta + \gamma$



T'の総費用α



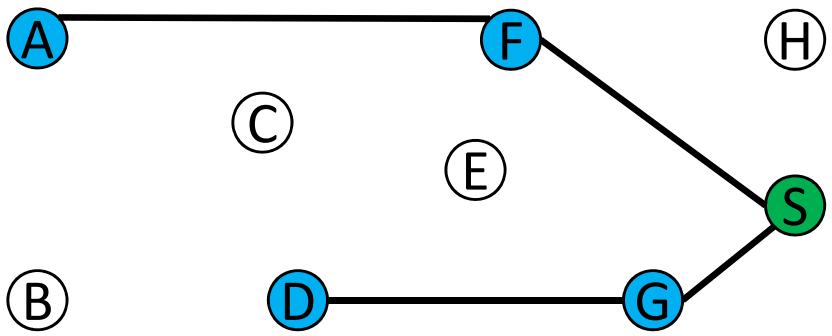
最小全域木の総費用= $\beta + \gamma$

参考資料:岡本吉央さんのスライド
http://jpn.nec.com/rd/datamining/rd/datamining/images/nec_datamining_seminar7.pdf

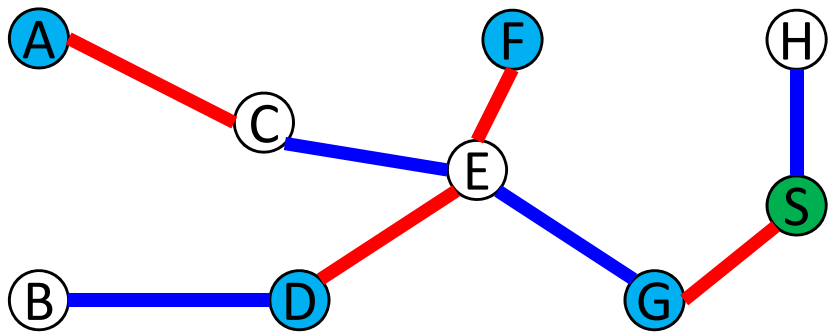
Bird分担がコアであることの証明（続き）

T' に枝集合$\{(v, f(v)|v \in N \setminus W\}$を追加
➔全域木になる（なぜ？）

この全域木の総費用$\alpha + \gamma$≧最小全域木の総費用＝$\beta + \gamma$
$\therefore \alpha \geq \beta$ が成立



T'の総費用α

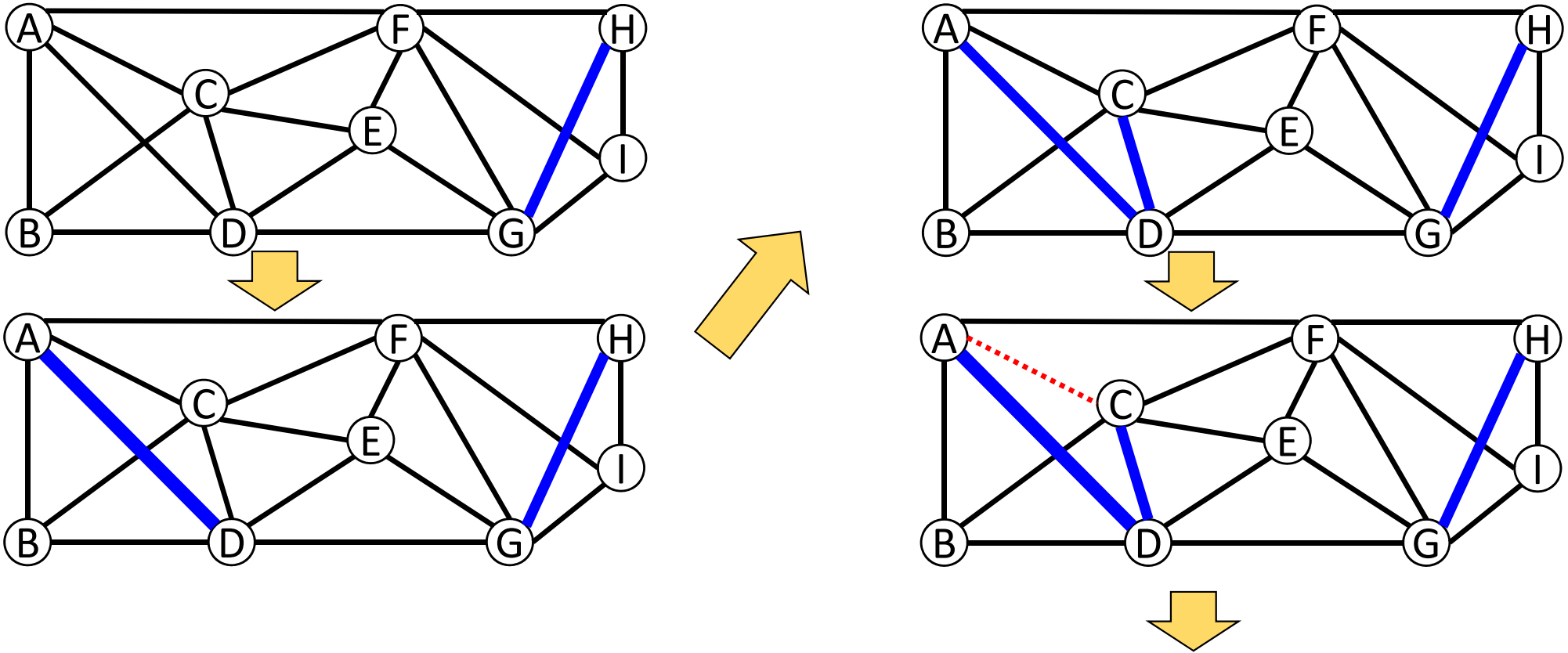


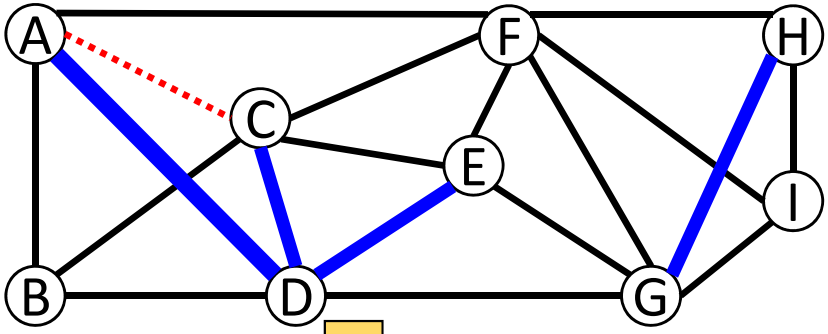
最小全域木の総費用＝ $\beta + \gamma$

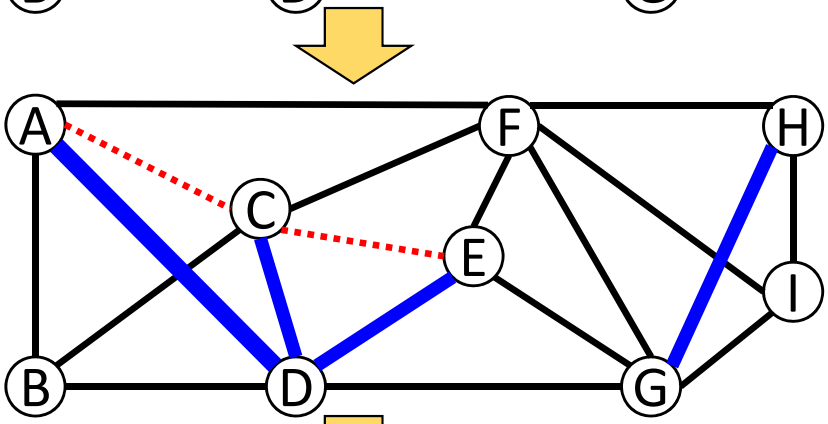
参考資料：岡本吉央先生のスライド
http://jpn.nec.com/rd/datamining/rd/datamining/images/nec_datamining_seminar7.pdf

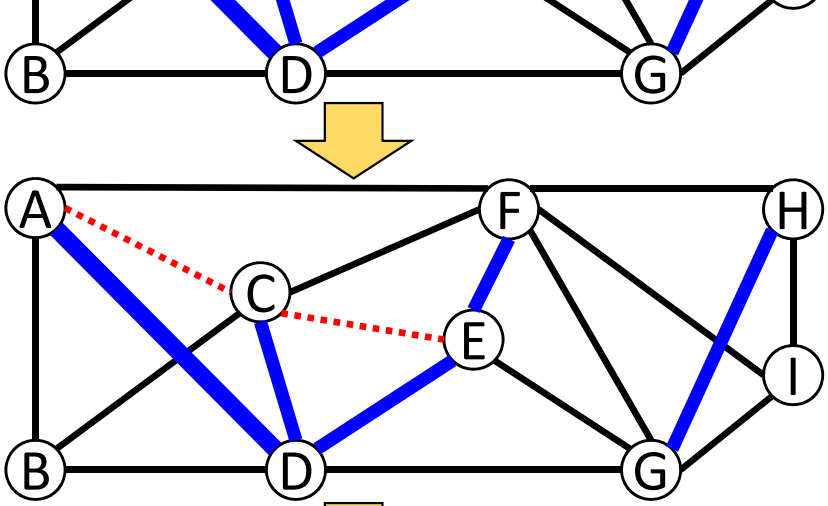
全域木の求め方 その2

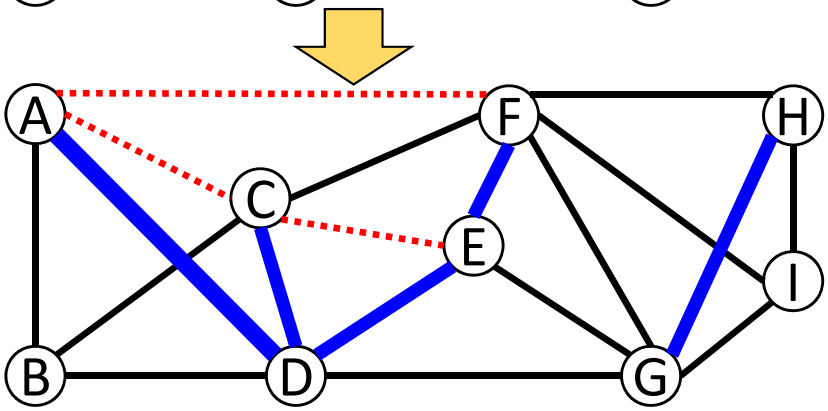
- 枝を1つずつ，適当な順番でＴに追加してみる
- 追加しても閉路ができない ➔ そのまま追加
- 追加したら閉路ができた ➔ その枝（もしくは閉路の別の枝を削除）
- アルゴリズム終了時にＴは連結になる ➔ 定義より，Ｔは全域木
  - 途中で閉路はできないことに注意

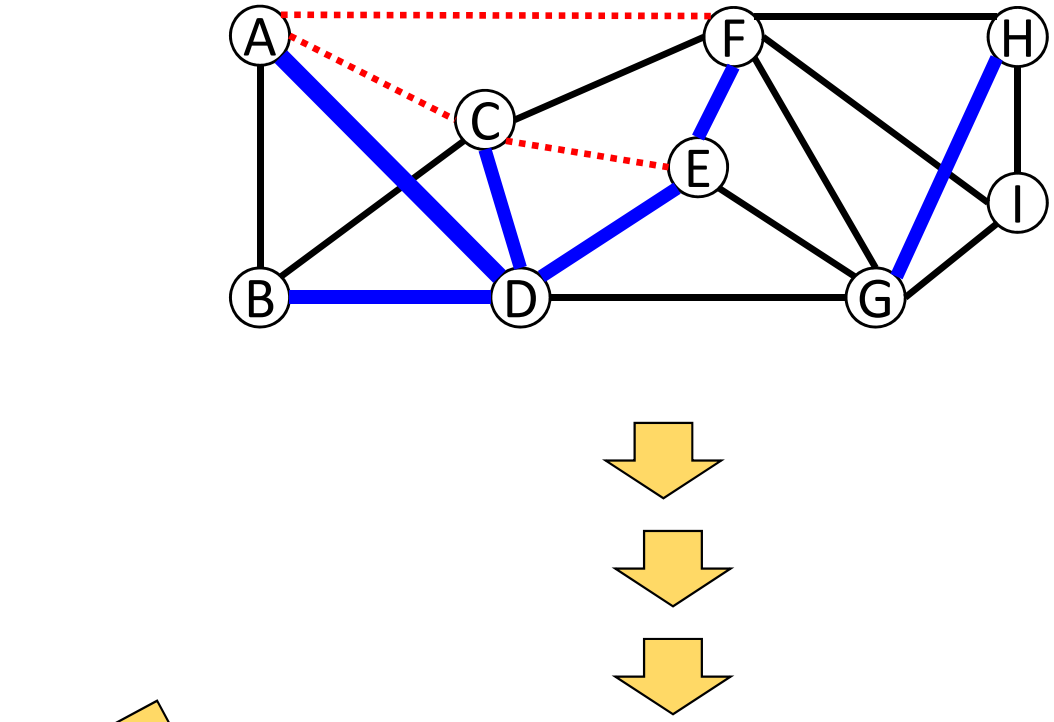












全域木は
閉路を含まない
枝集合のうち，
枝数が最大

閉路と全域木

全域木の求め方
その2の正当性

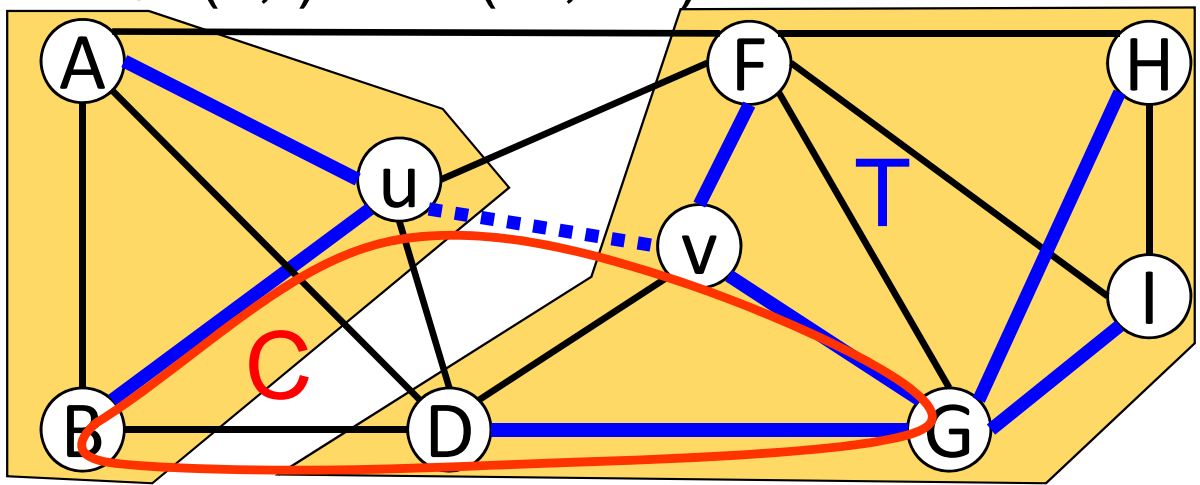
全域木を求める際，閉路の枝は1つ以上捨てる必要あり（定義より）
さらに，どの枝を捨てても良い

命題: C: 任意の閉路，(u,v): 閉路の中の任意の枝
➔(u,v)を含まない全域木 T が存在

証明（概略）: 全域木 T を任意に選ぶ.

- T が (u,v) を含まない: ただちに証明終了
- T が (u,v) を含む場合:
  T - (u,v) は 2つの連結成分 (V’, T’), (V’’, T’’) をもつ
  一方，P = C - (u,v) は u と v をつなぐ路である

∴ P に含まれるある枝(s,t) に 対し，(s,t) ∈ E(V’, V’’) 成立
T* = T - (u,v) + (s,t) とする
→ T* は連結，閉路を含まない
∴T* は全域木

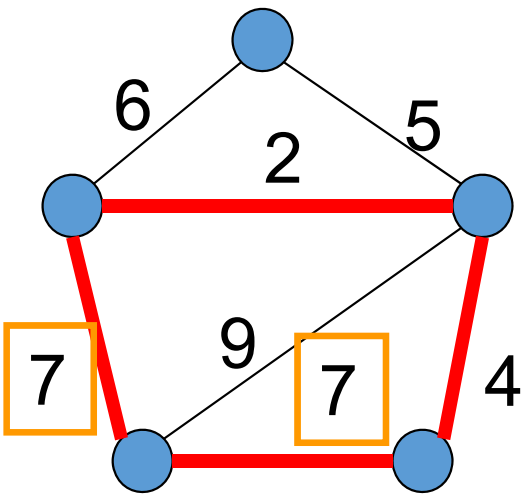


閉路と最小全域木

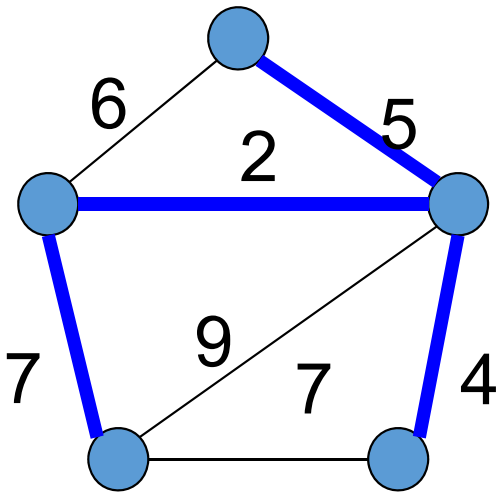
任意の閉路の枝のうち，少なくとも1つは全域木に含まれない
閉路の中で費用最大の枝は不要？　← Yes!

定理: C: 任意の閉路，(u,v): 閉路の中で費用最大の枝
➔(u,v)を含まない最小全域木T*が存在

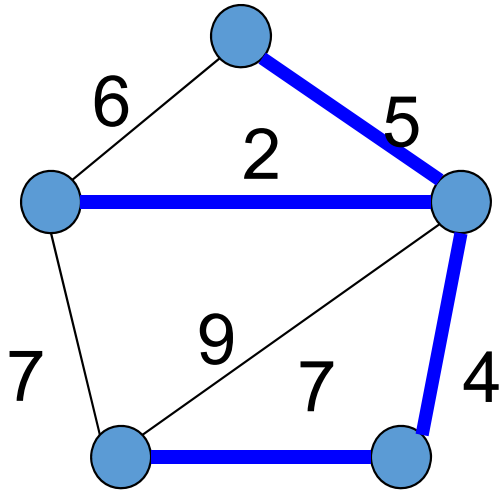
証明はあとで.



閉路C



最小全域木
その1



最小全域木
その2

閉路と最小全域木

定理: C: 任意の閉路, (u,v): 閉路の中で費用最大の枝
➔(u,v)を含まない最小全域木T*が存在

証明(概略): 最小全域木 T* を任意に選ぶ.

- T* が (u,v) を含まない: ただちに証明終了
- T* が (u,v) を含む場合:

  T* - (u,v) は 2つの連結成分 (V', T'), (V'', T'') をもつ

  P = C - (u,v) は u と v を結ぶ路なので,

  (s,t) ∈ E(V', V'') なる枝 (s,t) を含む

  T** = T* - (u,v) + (s,t) とする

→ T** は連結, 閉路を含まない ∴T** は全域木

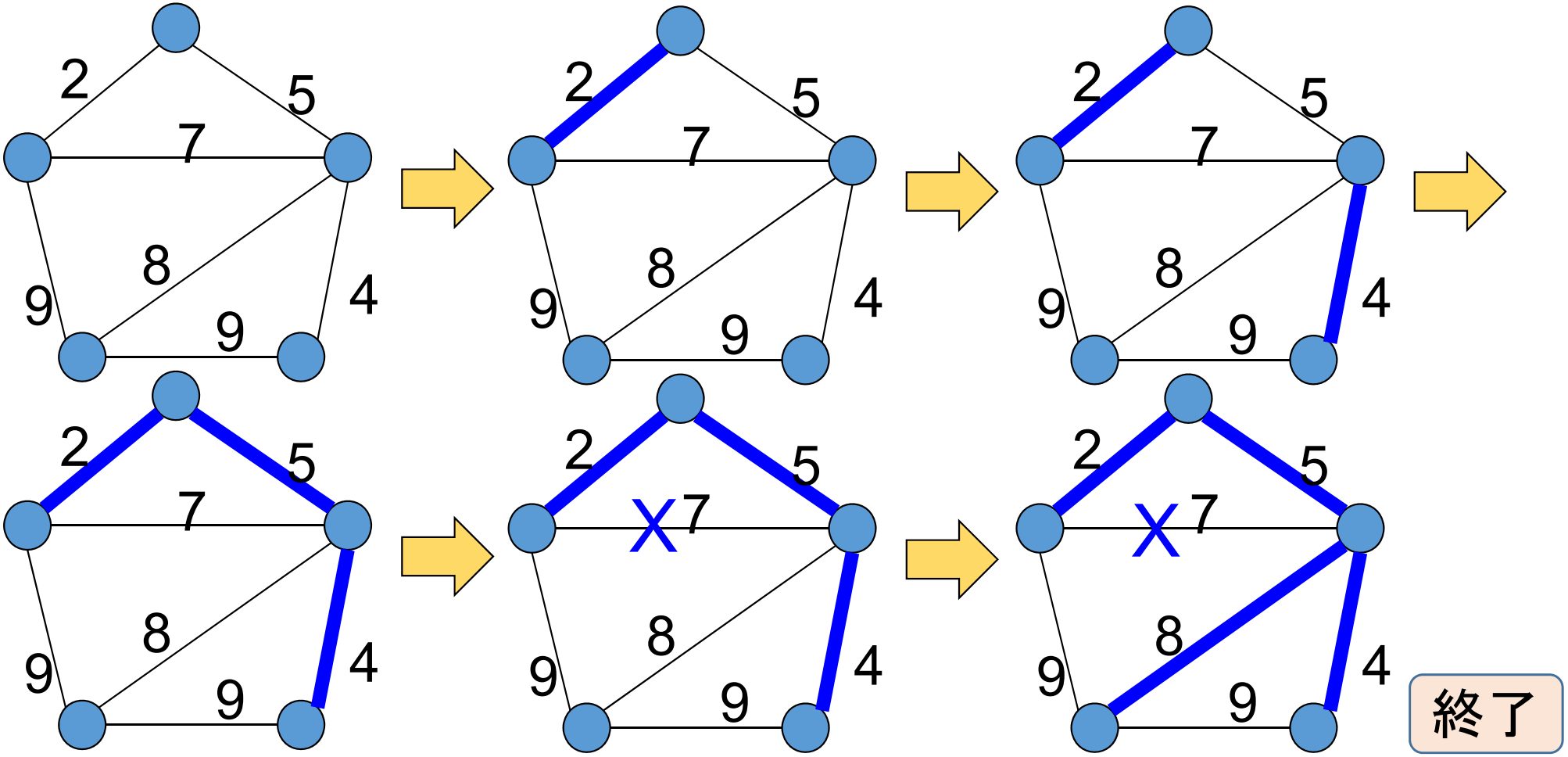
さらに T** の費用 - T* の費用 = (s,t)の費用 - (u,v)の費用≦0

∴ T** も最小全域木, (u,v)を含まない

最小全域木を求めるアルゴリズムその2：クラスカルのアルゴリズム

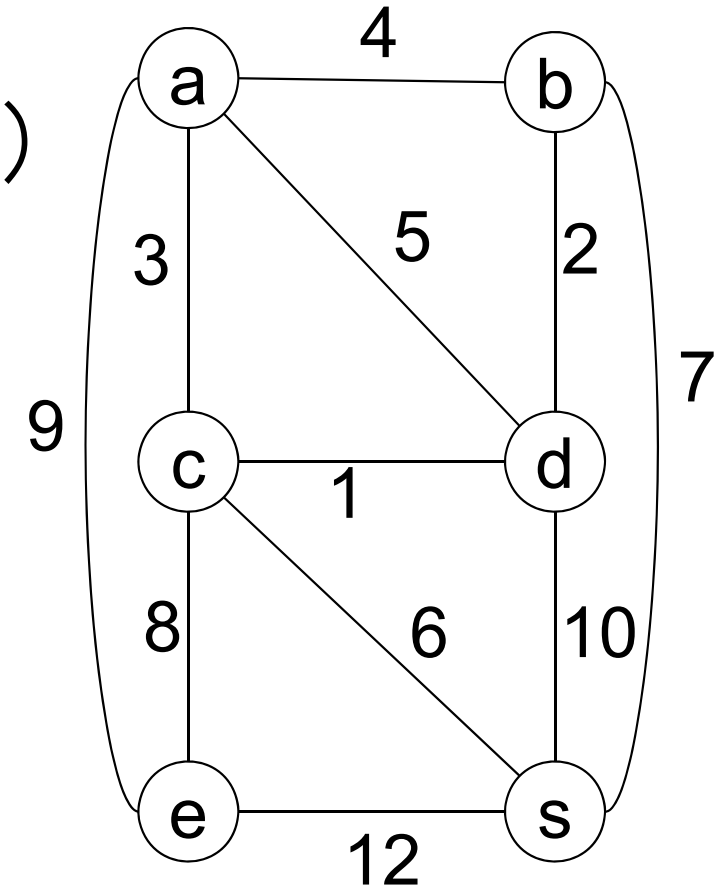
クラスカル(1956)

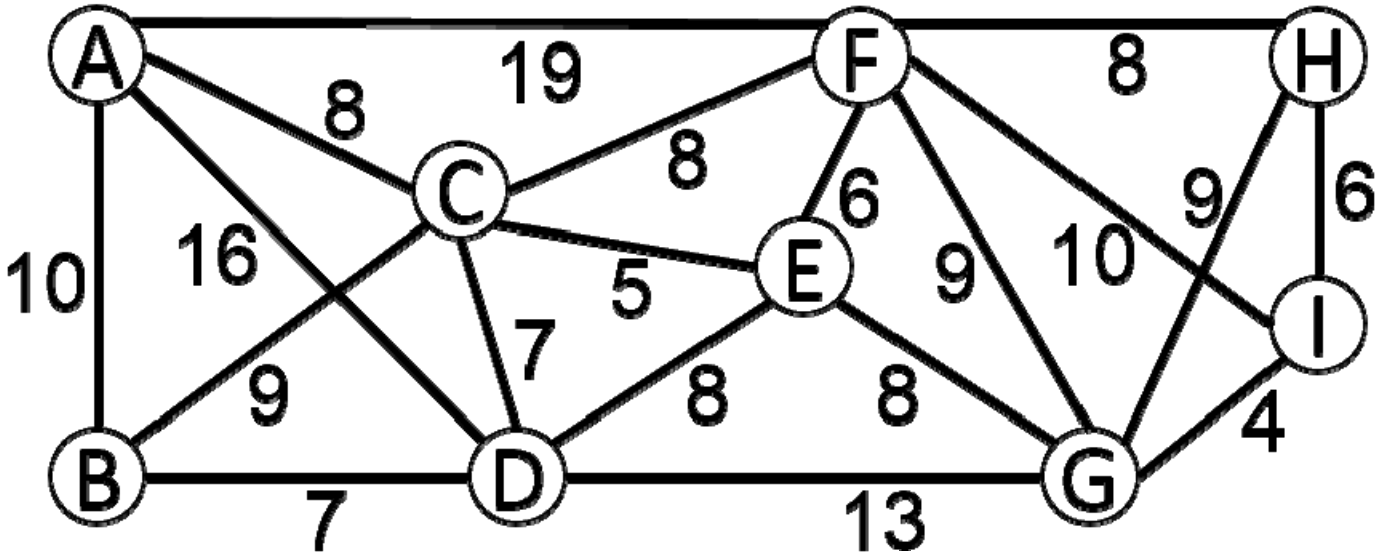
- 費用の小さい順に枝をTに追加
- ただし，閉路が出来る場合は追加しない



演習問題（提出の必要なし）

問題1：次の2種類のグラフの最小全域木を，クラスカルのアルゴリズムを使って求めよ.





演習問題（提出の必要なし）

問題2：

(1) スライド4，5のグラフにおいて，Bird分担以外のコアを求めよ.
　　そのようなコアで，全て整数値のものが存在するか？

(2) コア全体の集合を3次元の図で表現せよ.

問題3：Tは全域木, (u,v) は T に含まれる枝とする.

(1) 全域木 T から (u,v) を削除したとき，ちょうど2つの連結成分 (V’, T’), (V’’, T’’) が得られることを証明せよ

（ヒント：T - (u,v) において，任意の頂点は u または v に繋がっていることを示す）

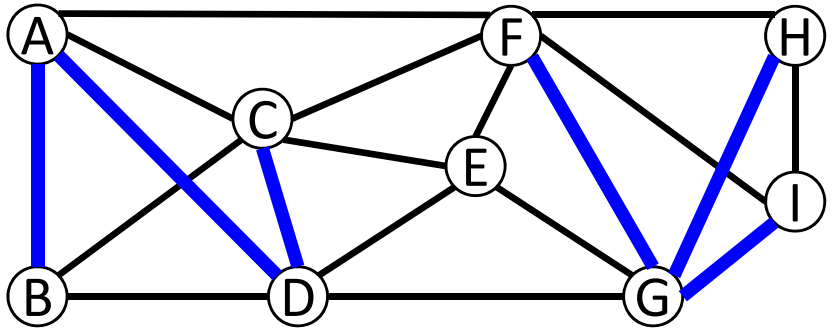
(2) カットセット E(V’, V’’) に含まれる任意の枝を (s,t) とする.
　　T* = T - (u,v) + (s,t) とおく.

(2-a) T* は連結であることを示せ（ヒント：(s,t) は2つの連結成分の間の枝）

(2-b) T* は閉路を含まないことを示せ（ヒント：T - (u,v) には閉路は含まれない）

発展：全域木と貪欲アルゴリズム

- 定義：グラフの森 ～閉路を含まない枝集合
- グラフの全域木 ←→ 枝数が最大の森



全域木の求め方２ --- 枝数最大の森を求めるアルゴリズム

- 枝を次々に加える
- ただし，閉路ができたらその枝は除外

このように，
良さそうなものから順番に選んで加えていくアルゴリズムを
　　貪欲（どんよく）アルゴリズム　とよぶ

一次独立ベクトルと貪欲アルゴリズム

行列の列ベクトルの集合の中から，一次独立なものを選ぶ.
そのベクトルの数を最大にしたい.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 2 & 0 & 1 & 1 \end{pmatrix}$$

○ ○ × ○ × ○

最大サイズの一次独立ベクトル集合を求めるアルゴリズム

- ベクトルを次々に加える
- ただし，これまで選んだ幾つかのベクトルと一次従属になったら，
  そのベクトルは除外

これも貪欲アルゴリズム

重み最大全域木と貪欲アルゴリズム

- グラフの各枝 (u,v) に重み w(u,v) が与えられている
- 枝の総重みが最大の全域木を求めたい
  → クラスカルのアルゴリズムの修正版により計算可能

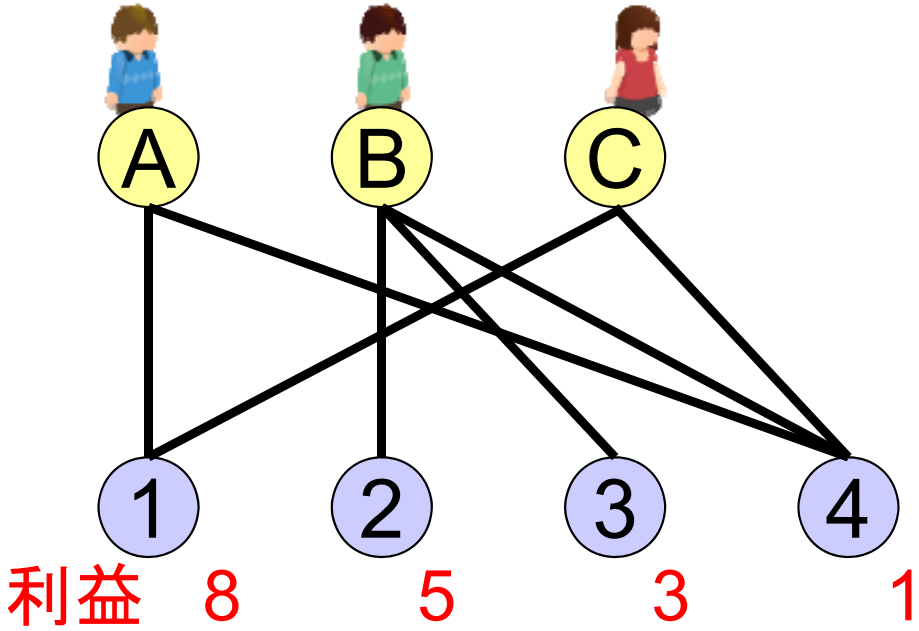
最大重み全域木を求めるアルゴリズム

- 枝を重い順に次々に加える
- ただし，閉路ができたらその枝は除外

正当性の証明は，最小全域木のときと全く同じ

仕事の割当問題と貪欲アルゴリズム

- 労働者集合と仕事集合が与えられている
- 各労働者は処理可能な仕事が限られている → 二部グラフ
- 各労働者には高々ひとつの仕事を割当可
- 各仕事は高々一人の労働者に割当可 → マッチング
- 仕事 j を処理 → 利益 p(j) を得る
- 処理可能な仕事の集合のうち，総利益最大のもの選びたい

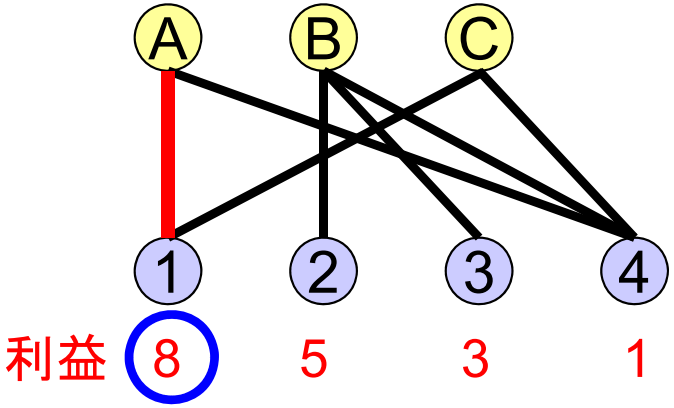


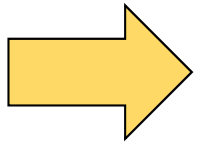
仕事①，③は
労働者A, B によって
処理可能
➔ 利益 8 + 3

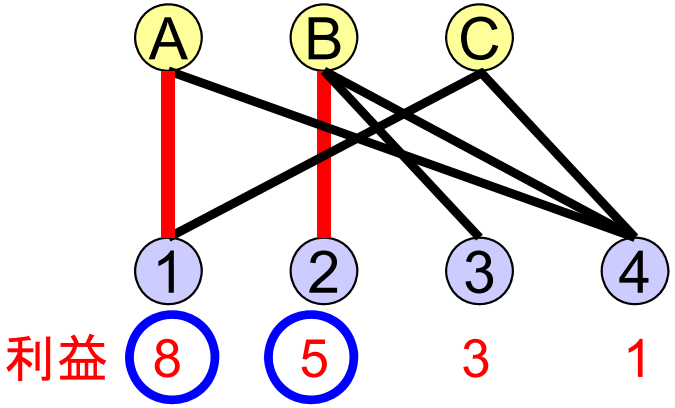
仕事の割当問題と貪欲アルゴリズム

利益最大の仕事集合を求めるアルゴリズム

- 仕事を利益の大きい順に次々に加える
- ただし，労働者との間にマッチングが存在しないならば
  その仕事は除外







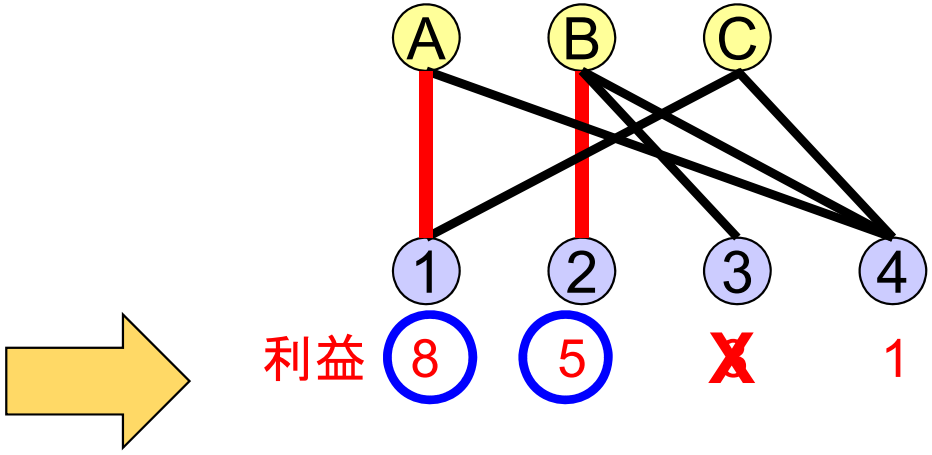
仕事①はAとの間に
マッチングをもつ
➔①追加

仕事①，②は
A, Bとの間に
マッチングをもつ
➔②追加

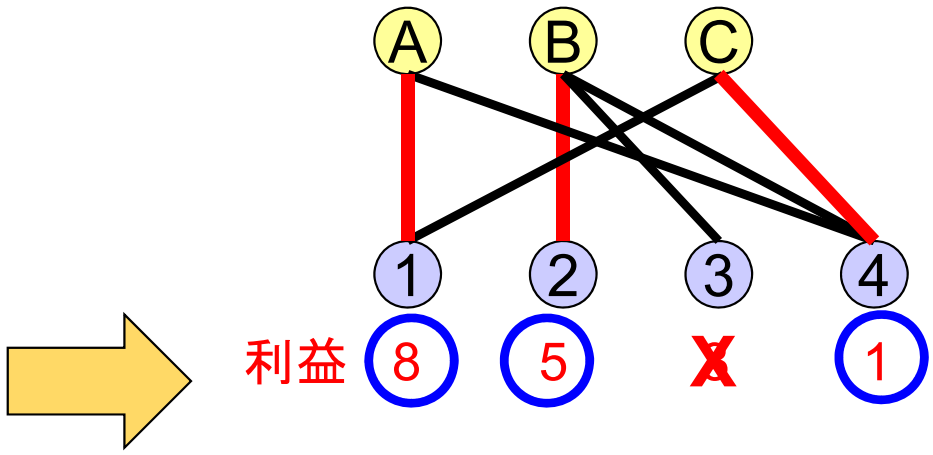
仕事の割当問題と貪欲アルゴリズム

利益最大の仕事集合を求めるアルゴリズム

- 仕事を利益の大きい順に次々に加える
- ただし，労働者との間にマッチングが存在しないならば
  その仕事は除外



仕事①，②，③は
A, B, Cとの間に
マッチングをもたない
➔③は除外



仕事①，②，④は
A, B, Cとの間に
マッチングをもつ
➔④追加

貪欲アルゴリズムの一般形

異なる問題に同様のアルゴリズムが適用可能

N = {1,2,..., n} --- 追加可能な要素の集合

一般的な貪欲アルゴリズム

- 重みなどの大きい順に要素を次々に追加する
- ただし，追加したときの集合が「実行可能集合」でなければ
  その要素は除外

実行可能集合の例

- (最大重み)全域木問題: 実行可能集合 ＝ 森
- 一次独立ベクトルの問題:実行可能集合 ＝ 一次独立ベクトル集合
- 仕事の割当問題: 実行可能集合 ＝ マッチングでカバーできる
  仕事の集合

これらの実行可能集合には共通の性質 → マトロイド

マトロイドの定義

一般的な貪欲アルゴリズムで
最適解の計算が可能

- 前述の実行可能集合は，いずれも以下の性質を満たす

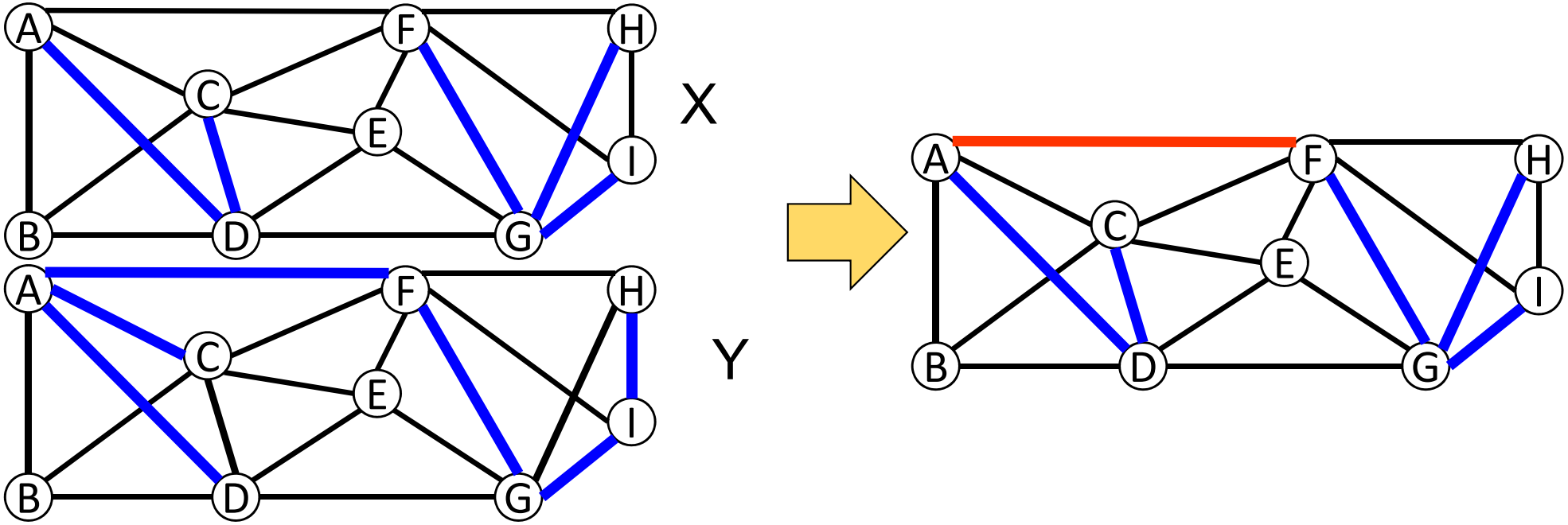
(P1) 空集合は実行可能集合

(P2) 集合 X が実行可能集合のとき，その部分集合 Y は実行可能集合

(P3) 実行可能集合X, Y に対し |X| < |Y| ならば，

ある要素 j∈ Y - X が存在して，X + j も実行可能集合

マトロイド：これらの性質を満たす離散構造（実行可能集合の集まり）



マトロイドの利点

マトロイドのように抽象的な概念を扱う利点

- 個々の問題のもつ離散構造の本質（重要なポイント）が見えてくる
- マトロイドに対して新たな結果を得る

  ➔個々の問題に対して適用可能

  - 似たような定理・アルゴリズムを個々の問題に対して示す必要がなくなる