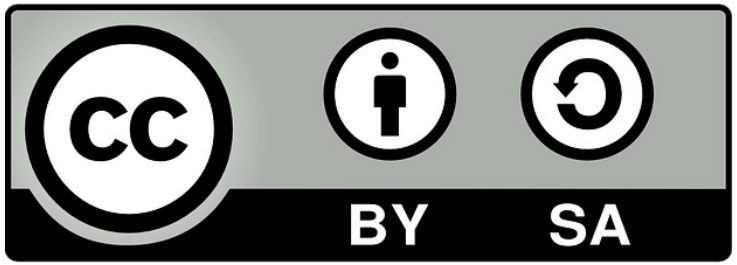
Algorithms (2024 Summer)
#13：グラフアルゴリズム4,
(最小費用流問題)
「難しい問題」とは，さいごに

矢谷 浩司
東京大学工学部電子情報工学科



3回目の授業アンケート

ご協力を何卒よろしくお願いいたします.

https://forms.gle/EWGxvUTDYMai71Sv5

これが最後のアンケートになります．授業全体を通しての感想などもお聞きしたいと思っていますので，どうぞよろしくお願いいたします.

UTAS上での授業アンケート（本学学生のみ）

こちらもどうぞよろしくお願いいたします．こちらは工学部で行っているものになりますので，こちらにもご意見いただけるとありがたいです．

今日のお題は3つ.

最小費用流問題

コンピュータにとって「難しい問題」とは？
　　そういう問題にぶち当たったときは？

さいごに

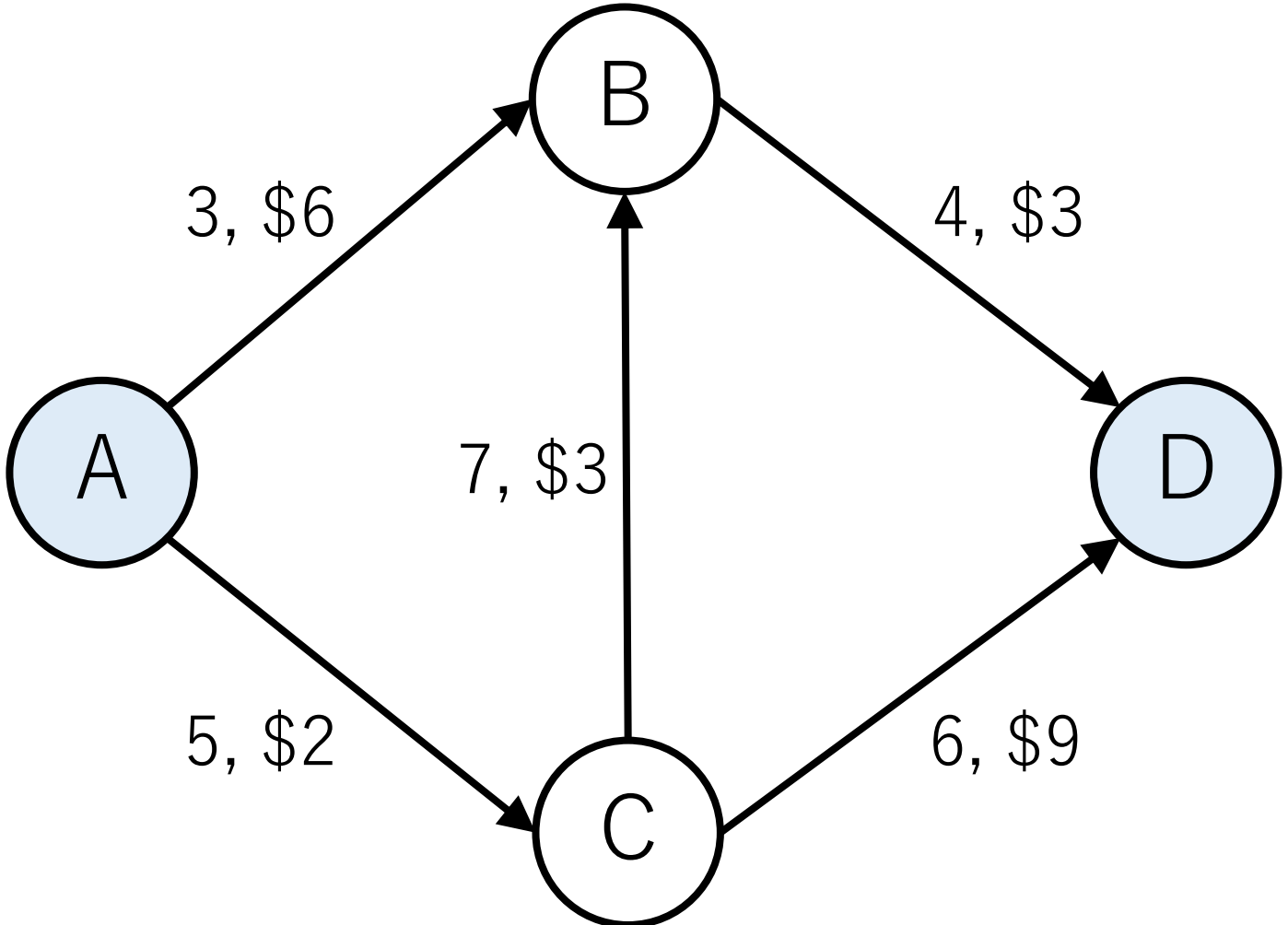
最小費用流

「ある流量fを開始ノードから終了ノードに流す時，最小のコストはいくらか．」

経路に容量だけでなく，コスト（流量1に対する費用）が定義されている．

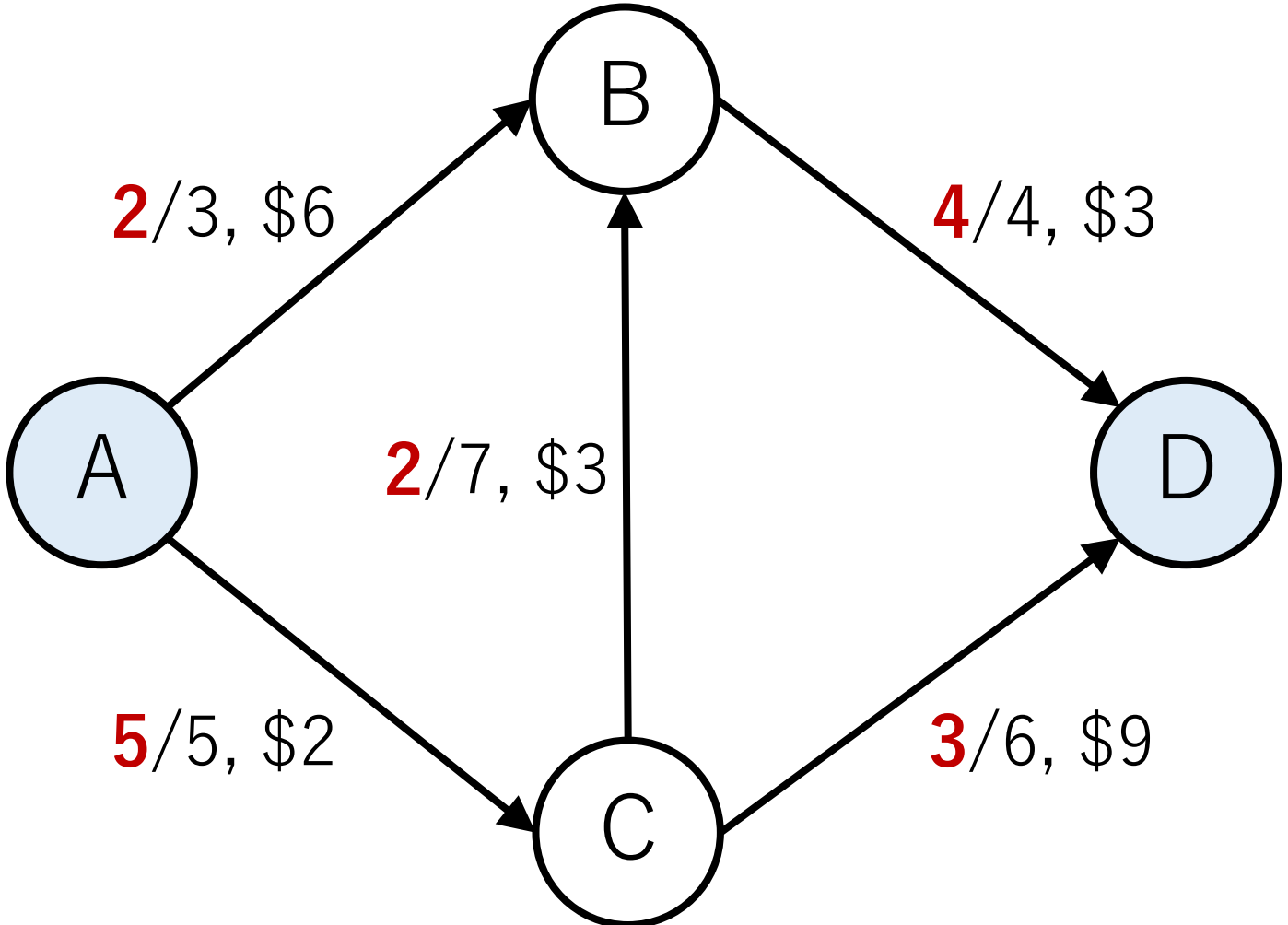
最小費用流

7流す時の最小費用は？



最小費用流

7流す時の最小費用は？-> 67



プライマル・デュアル法（Primal-Dual）

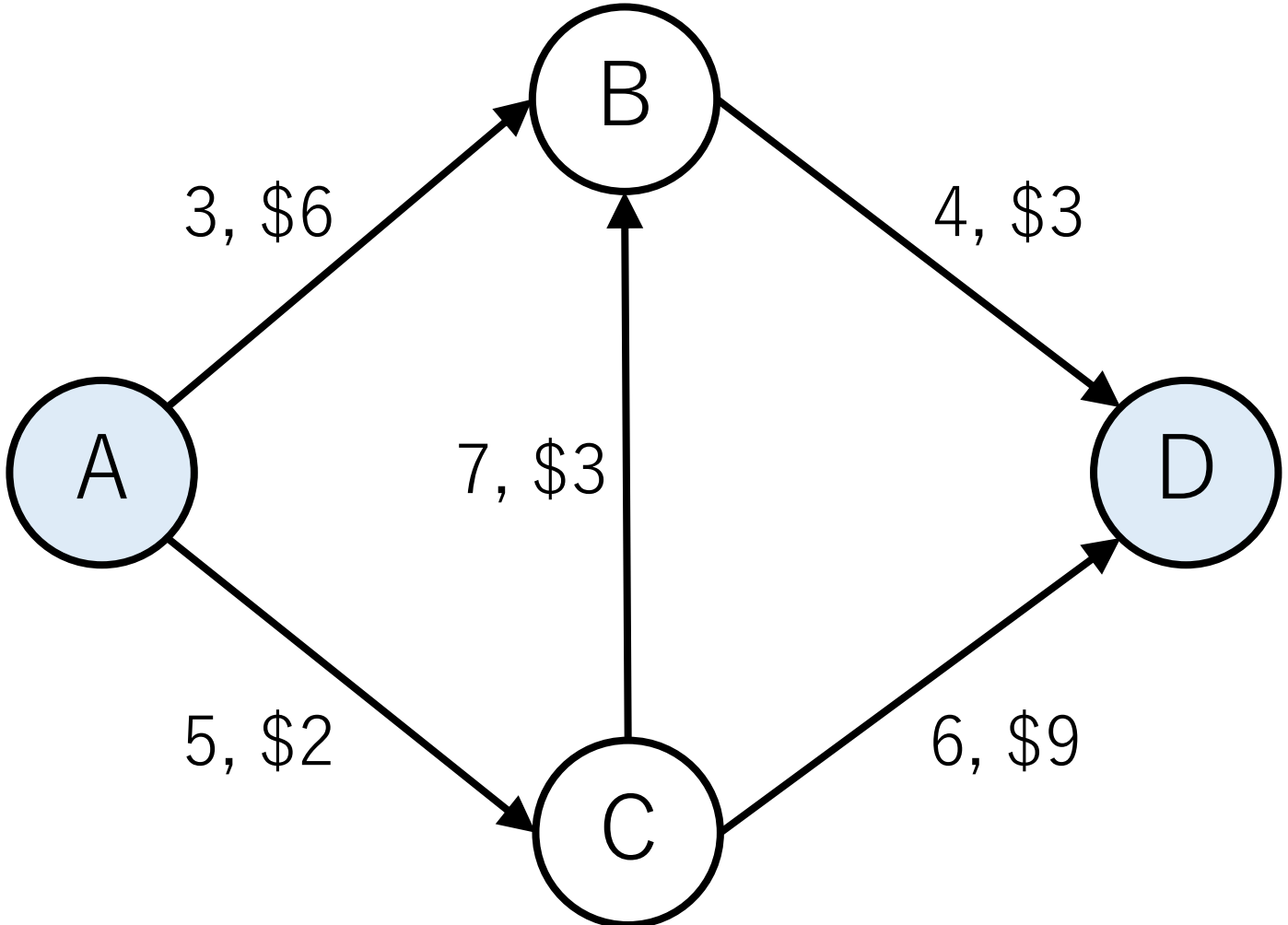
基本はフォード・ファルカーソン法と同じ．逆方向の経路も考える．

経路の探索において，その時点での最小コストの経路を求めて，そこに流せるだけ流す．

最短経路はベルマン・フォード法などを使う．
（負の経路が存在するため）

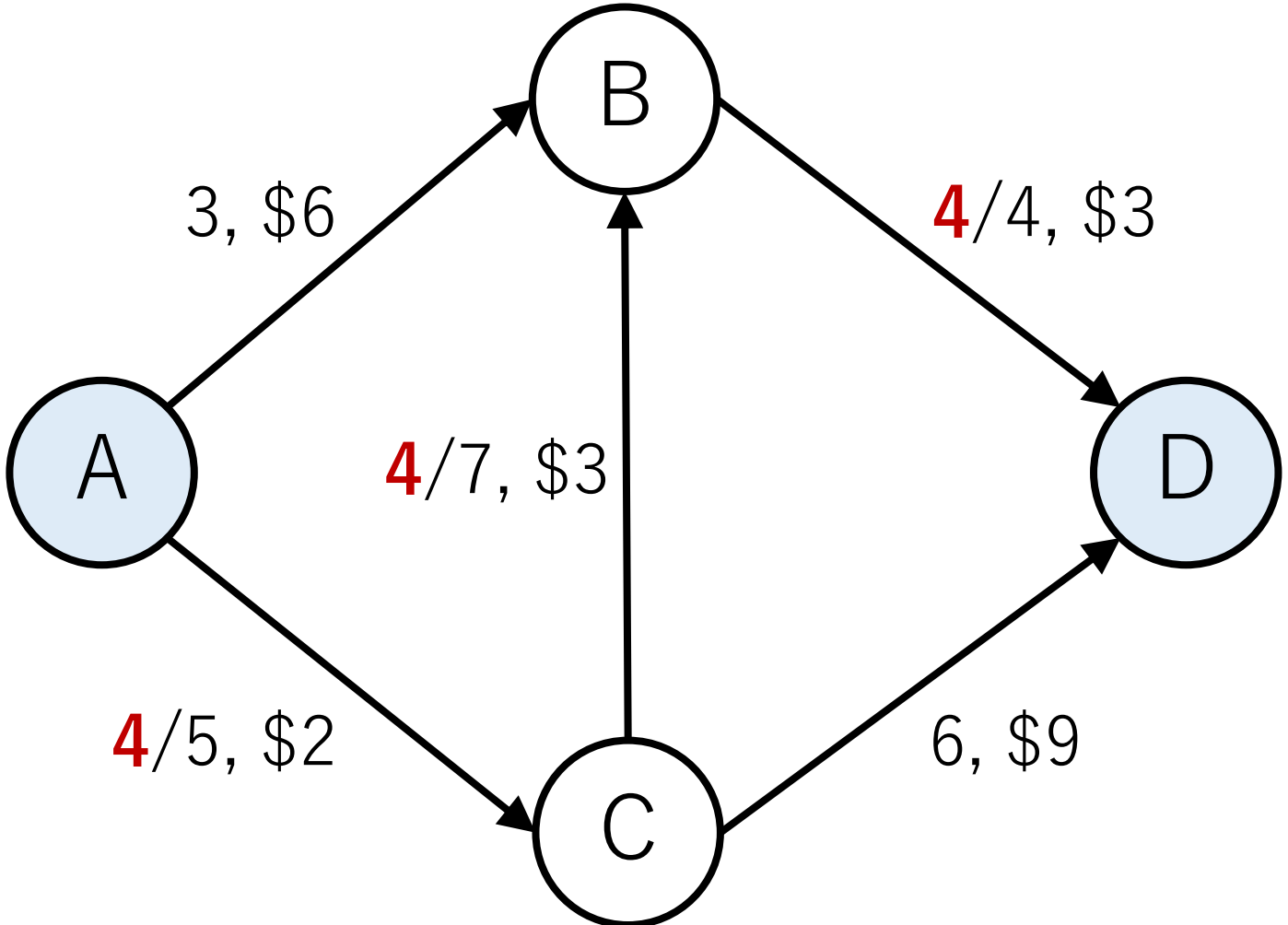
最小費用流の求め方例

7流す時の最小費用は？



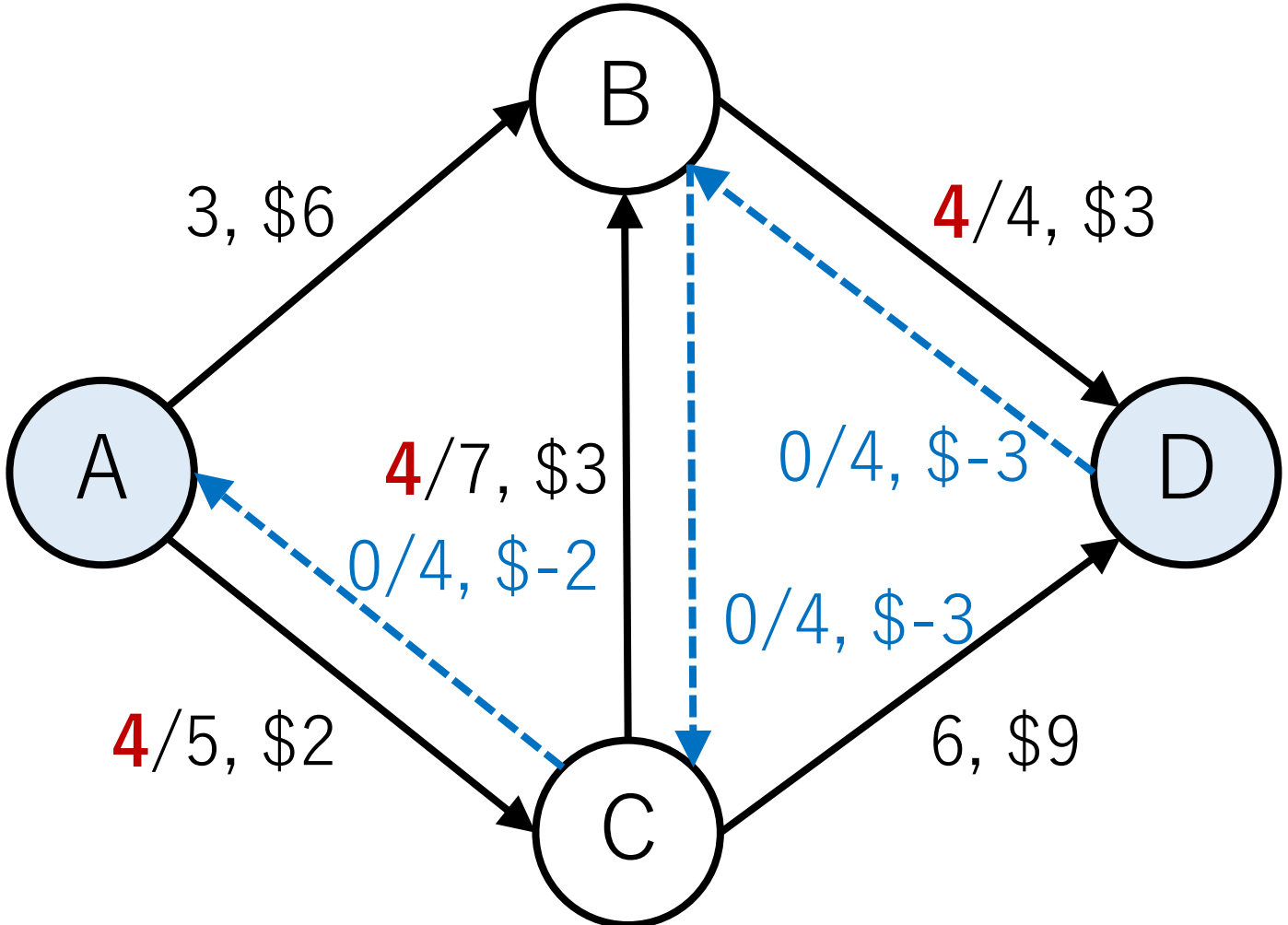
最小費用流

A->C->B->Dが最小費用経路$8/流量で4流せる.



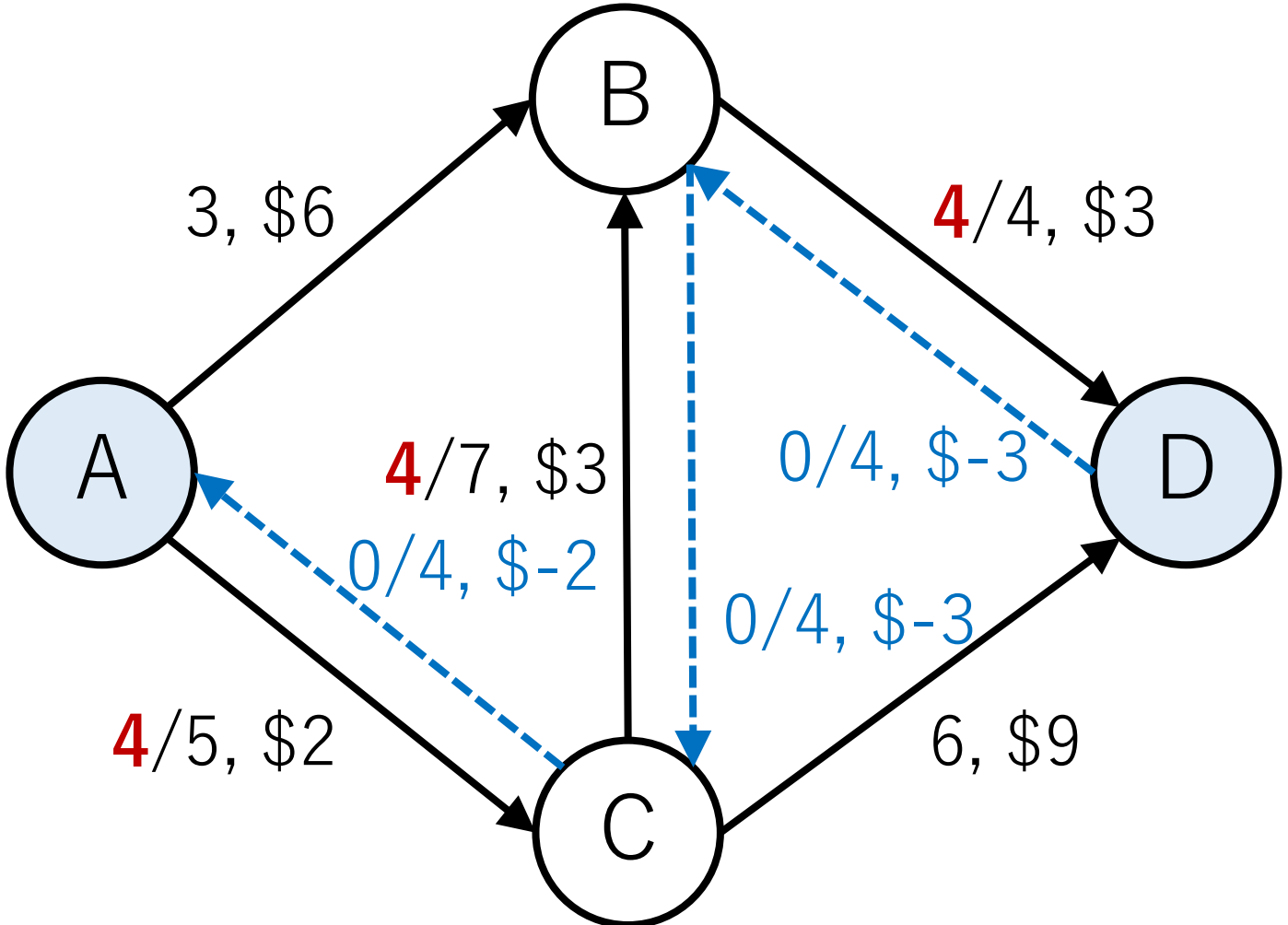
最小費用流

逆の経路を設定する．逆の経路のコストは順方向の負の値.



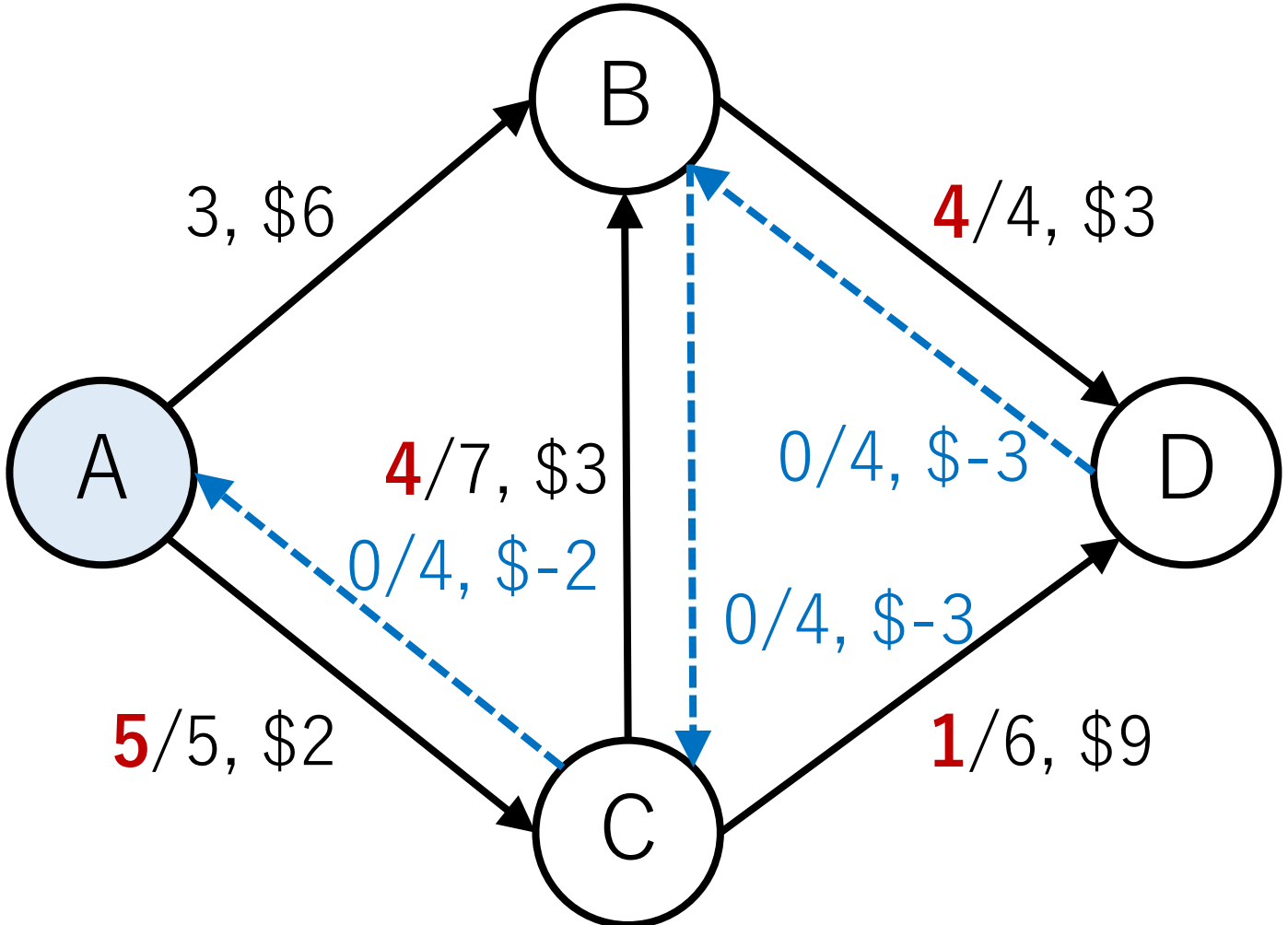
最小費用流

A->C->Dは1流せて$11/流量．A->B->C->Dは3流せて$12/流量.



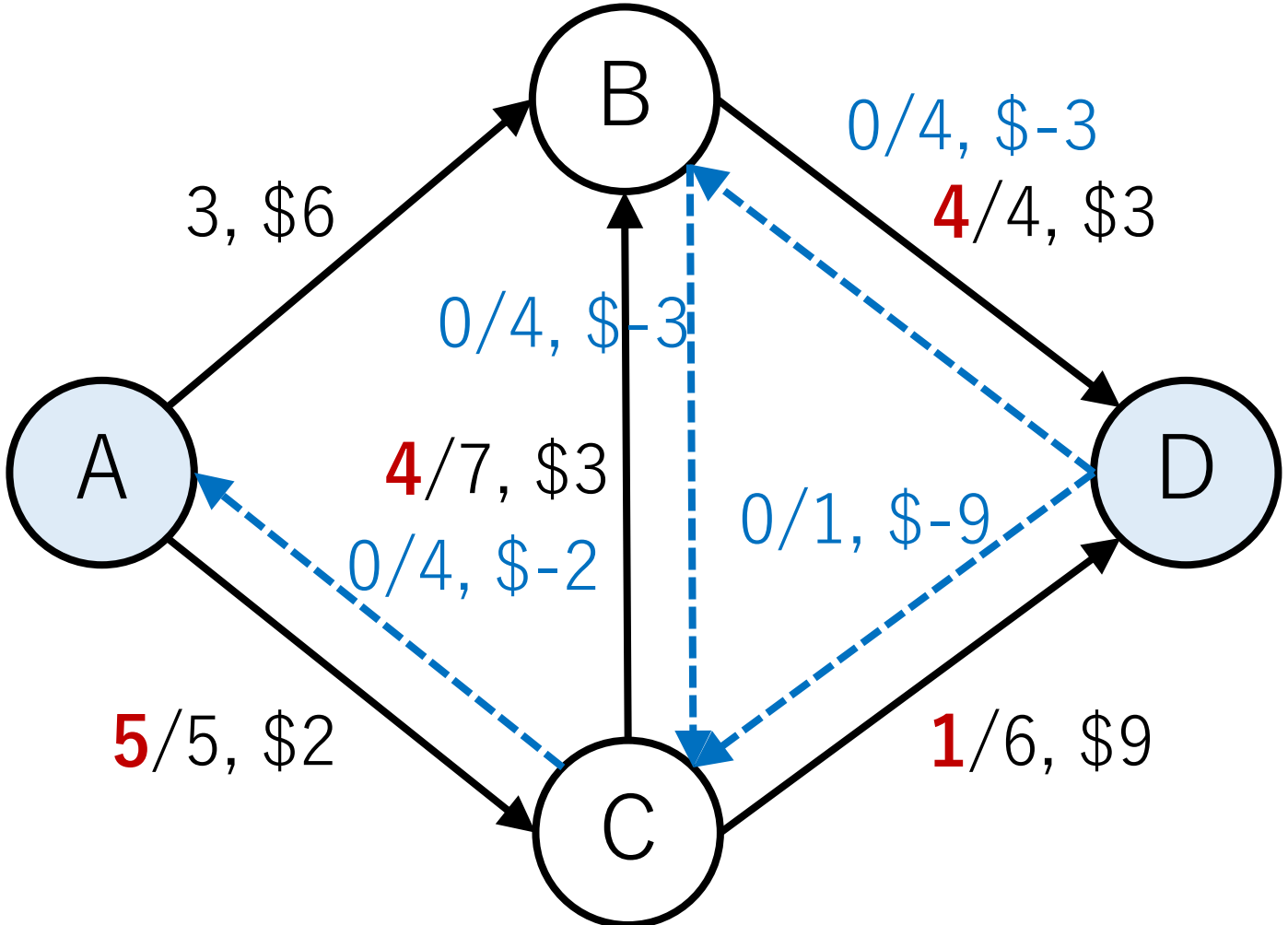
最小費用流

よって，A->C->Dに1流す．



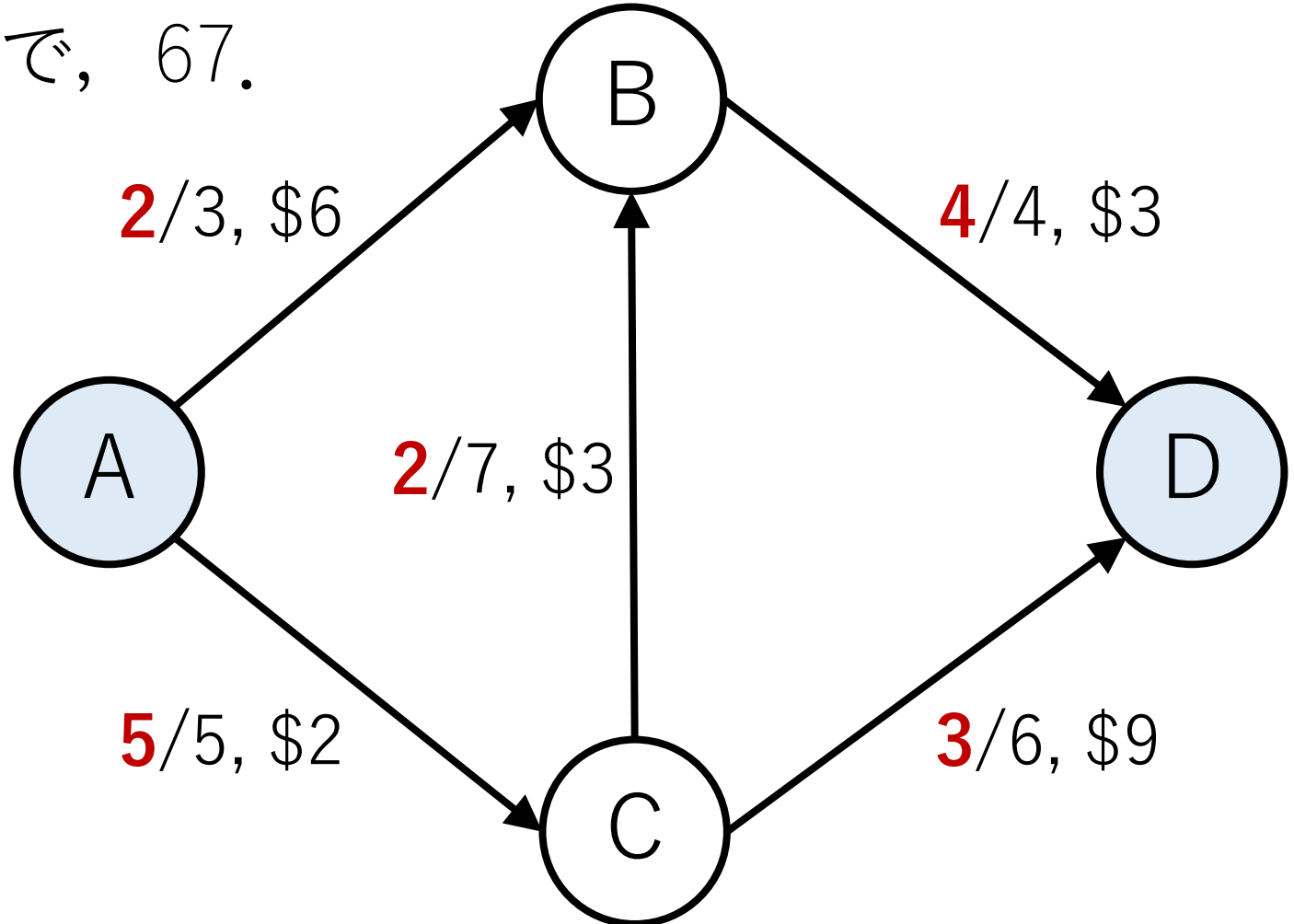
最小費用流

次にA->B->C->Dに2を流す．これは6-3+9=$12/流量.



最小費用流

逆経路の流れが全て相殺されることを確認し，終了.
32+11+24で，67.



プライマル・デュアル法の実装例

```
# 各変数の定義
node_num = 4, F = 7, inf = float('inf')
# 最初と最後をそれぞれ開始，終了ノードとする
#{（次のノード），[（容量），（単位当たりコスト）]}
graph =[
{1: [3, 6], 2: [5, 2]},                  # ノードA
{0: [0, -6], 2: [0, -3], 3:  [4, 3]},  # ノードB
{0: [0, -2], 1: [7, 3], 3: [6, 9]},    # ノードC
{1: [0, -3], 2: [0, -9]}                 # ノードD
]
```

プライマル・デュアル法の実装例

```
# g：グラフ（形式は前ページのスライド参照），
# V：ノードの数，f：指定された流すべきフロー量
def PrimalDual(g, V, f):
        totalCost = 0    # コストの総額
        cur_flow = 0     # 今までに考えたフローの総量
```

プライマル・デュアル法の実装例

```
def PrimalDual(g, f):
    ...
    while True:       # ベルマンフォードを行う
        cost = [inf]*V
        cost[0] = 0
        route = [-1]*V  # 最短経路を保持するリスト
        flag = True
```

プライマル・デュアル法の実装例

```
def PrimalDual(g, f):
        …
        while True:    # ベルマンフォードを行う
                …
                while flag:
                        flag = False, cur_node = 0
                        for i in range(V):
                                for dest_n, e in graph[i].items():
                                        if e[0] > 0 and cost[i] + e[1] < cost[dest_n]:
                                                flag = True
                                                cost[dest_n] = cost[i] + e[1]
                                                route[dest_n] = i  # 経路を記録
                if cost[V-1] == inf: return -1
```

プライマル・デュアル法の実装例

```
def PrimalDual(g, f):
    ...
    if cost[V-1] == inf: return -1

    # 今回の経路で流せる量を求める準備
    min_c = inf
    cur_n = V-1
    prev_n = route[cur_n]
```

プライマル・デュアル法の実装例

```
def PrimalDual(g, f):
    ...
    # 経路を逆にたどりながら，流せる限界の量を求める
    while prev_n != -1:
        if min_c > graph[prev_n][cur_n][0]:
            min_c = graph[prev_n][cur_n][0]
        cur_n = prev_n
        prev_n = route[cur_n]
```

プライマル・デュアル法の実装例

```
def PrimalDual(g, f):
    ...
    # 目標のフローを超えるときは，ギリギリまでにする
    if cur_flow + min_c >= f:
        totalCost += cost[V-1]*(f - cur_flow)
        return totalCost
    else: #  そうでなければ，今の容量目一杯流す
        totalCost += cost[V-1]*min_c
        cur_flow += min_c
```

プライマル・デュアル法の実装例

```
def PrimalDual(g, f):
    …
    # 経路を逆にたどりながら，容量の更新
    cur_n = V-1, prev_n = route[cur_n]
    while prev_n != -1:
        graph[prev_n][cur_n][0] -= min_c    # 順方向
        graph[cur_n][prev_n][0] += min_c    # 逆方向
        cur_n = prev_n, prev_n = route[cur_n]
```

プライマル・デュアル法の実行例

```
print(PrimalDual(graph, F))

======出力結果======
67
```

プライマル・デュアル法の実行例

与えられたグラフ

```
[
{1: [3, 6], 2: [5, 2]},
{0: [0, -6], 2: [0, -3], 3: [4, 3]},
{0: [0, -2], 1: [7, 3], 3: [6, 9]},
{1: [0, -3], 2: [0, -9]}
]
```

1回目（A->C->B->D）
経路が終わったあと

```
[
{1: [3, 6], 2: [1, 2]},
{0: [0, -6], 2: [4, -3], 3: [0, 3]},
{0: [4, -2], 1: [3, 3], 3: [6, 9]},
{1: [4, -3], 2: [0, -9]}
]
```

（次のノード），[（容量），（単位当たりコスト）]で並んでいる．

プライマル・デュアル法の計算量

ノード数$|V|$，エッジ数$|E|$，設定流量をFとする．

最小コスト経路探索のためにはベルマン・フォード法を使うと$O(|V||E|)$必要．（隣接リストを使う場合）

さらに，流量を1ずつしか増やせないのが最悪のケース．

よって，$O(F|V||E|)$．

プライマル・デュアル法の計算量

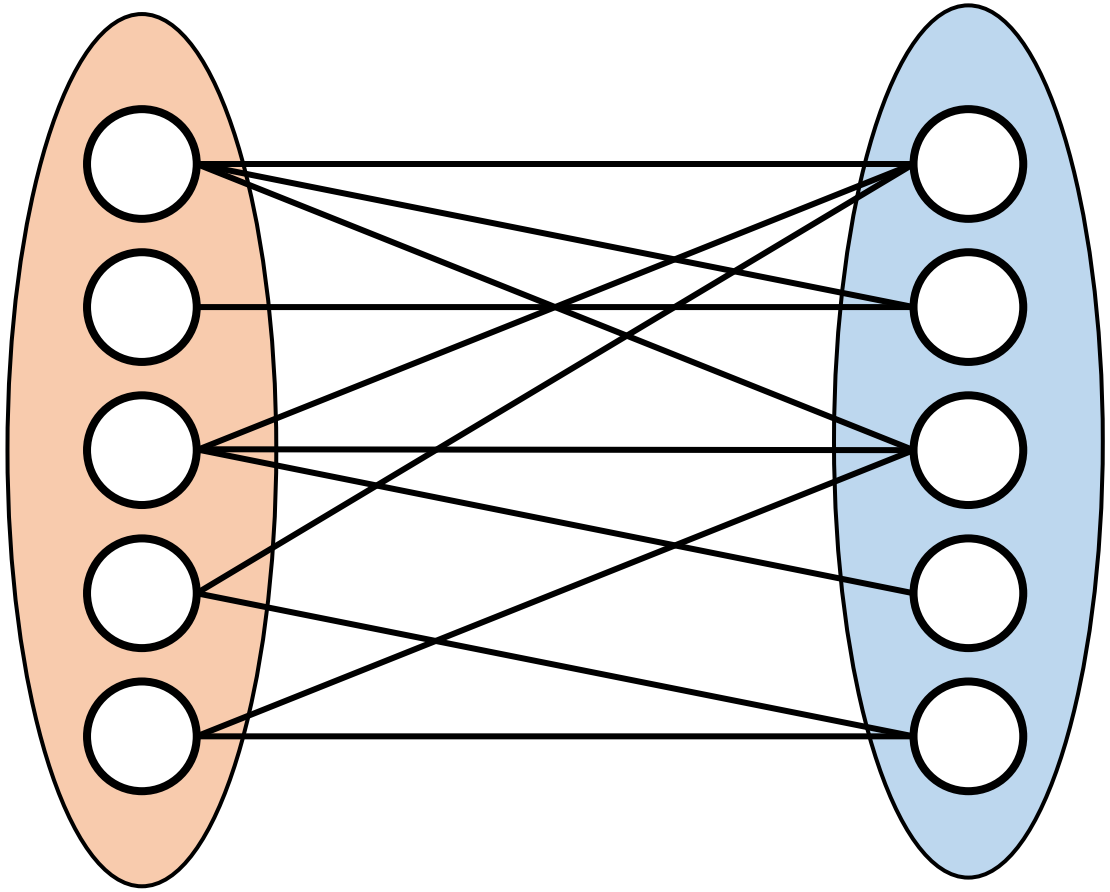
ポテンシャルと呼ばれる工夫をするとダイクストラ法を使うことができる.

この場合，$O(F|V|^2)$.

ヒープを使うダイクストラ法を使えば，計算量を$O(F|E|\log|V|)$に減らせる.

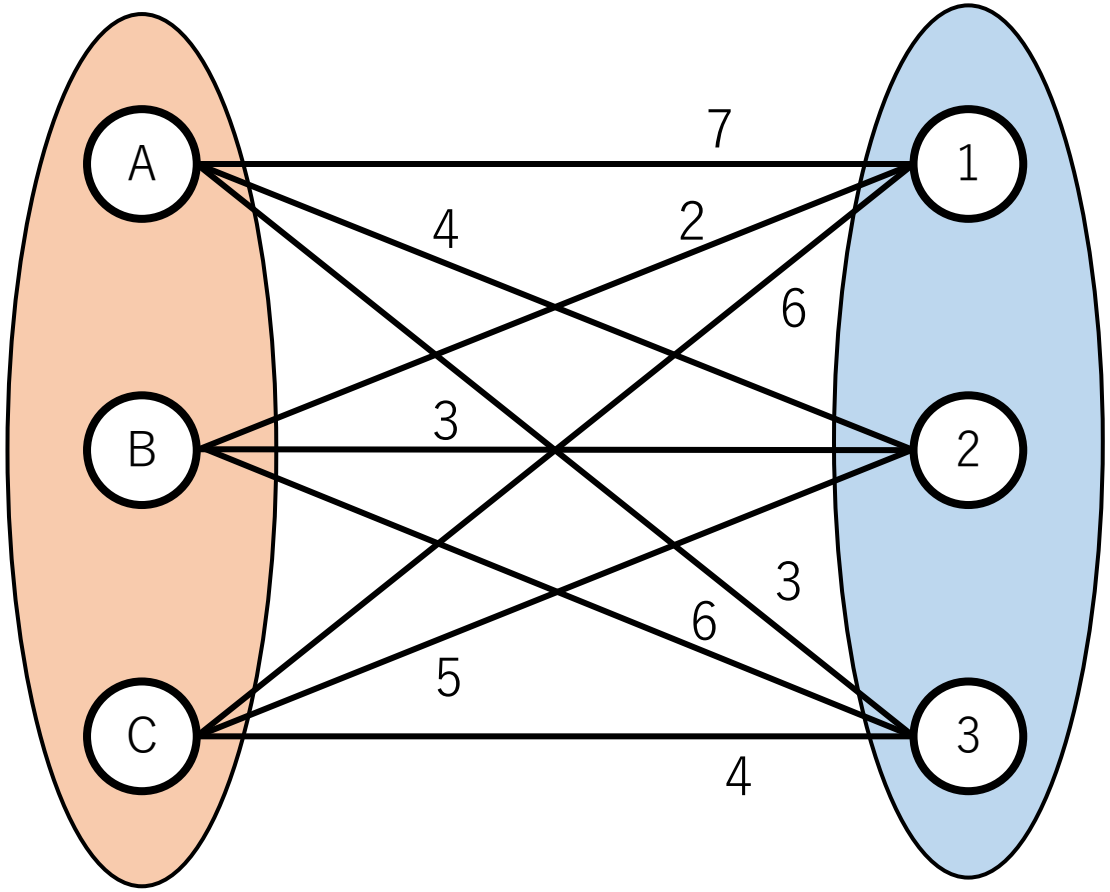
二部グラフ（bipartite graph）

頂点集合を2つの部分集合に分割でき，各集合内の頂点同士の間には辺が無いグラフ.



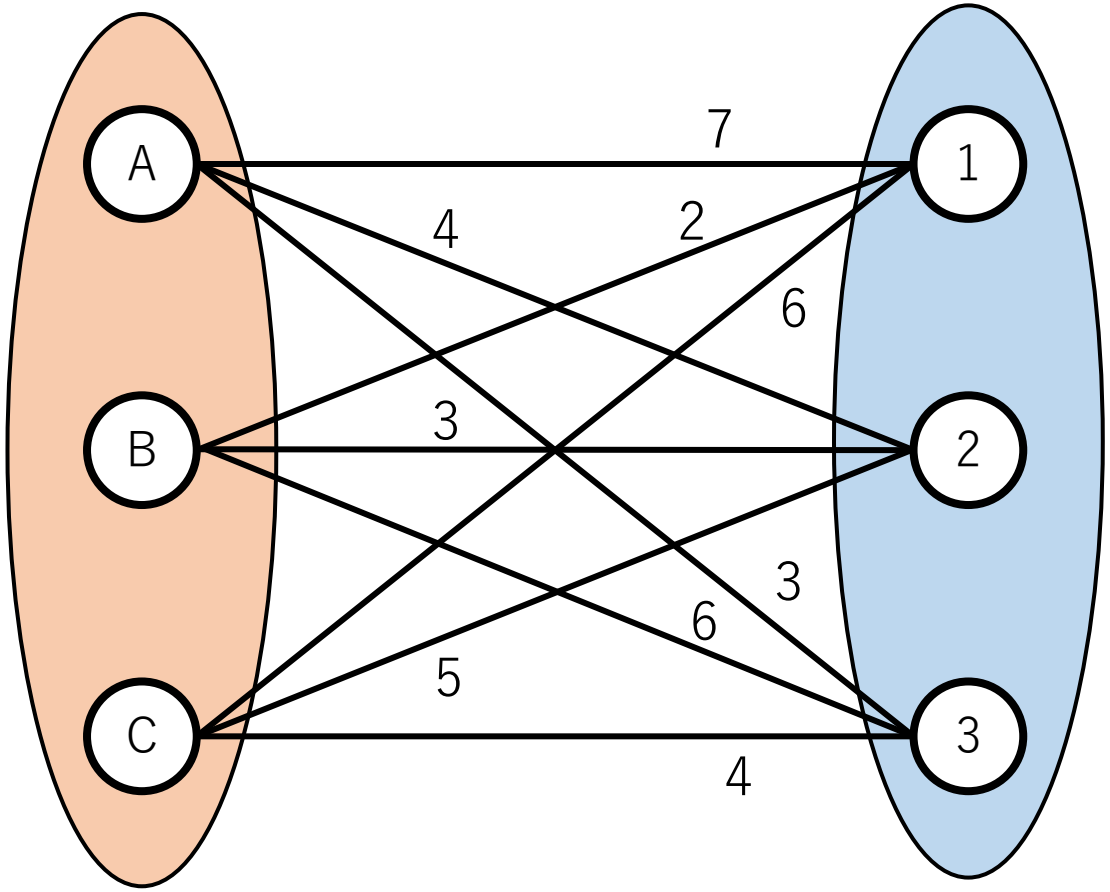
重み付き二部グラフの最大マッチング問題

左右のグループをつなぐ辺の重みの総和を最大にする.
どの2辺も共通のノードを持たない.



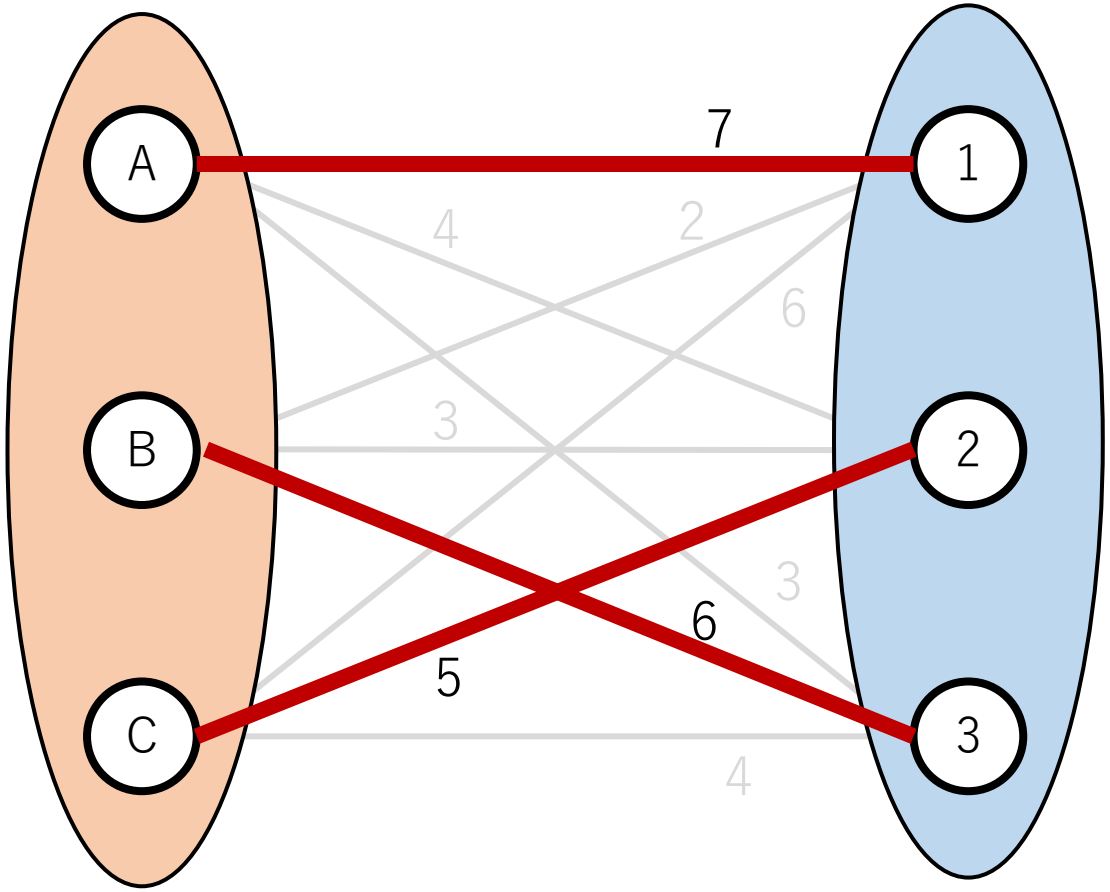
重み付き二部グラフの最大マッチング問題

例）希望度合いがついたペア組み．数字が大きいほどペアになりたい度がより高いことを表す．



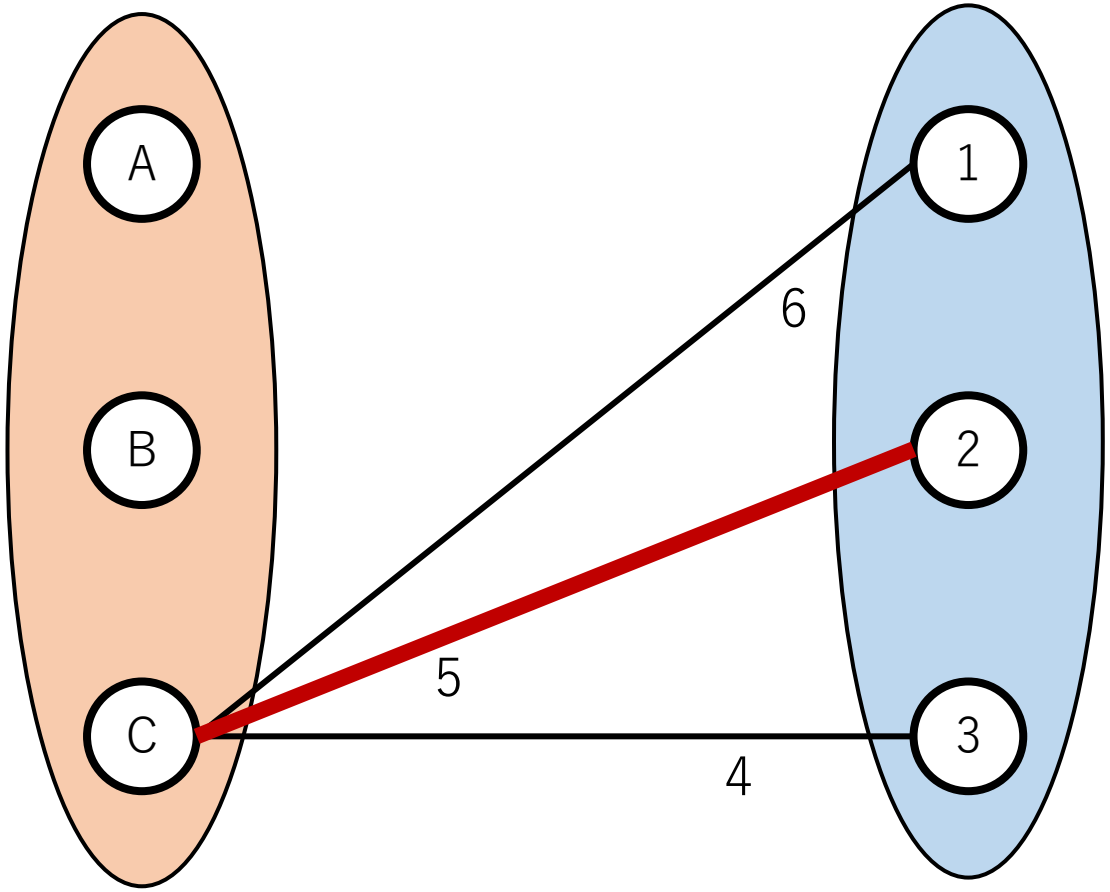
重み付き二部グラフの最大マッチング問題

この場合は以下のようにつないで18.



重み付き二部グラフの最大マッチング問題

個人レベルでは必ずしも最大にならない.



重み付き二部グラフの最大マッチング問題

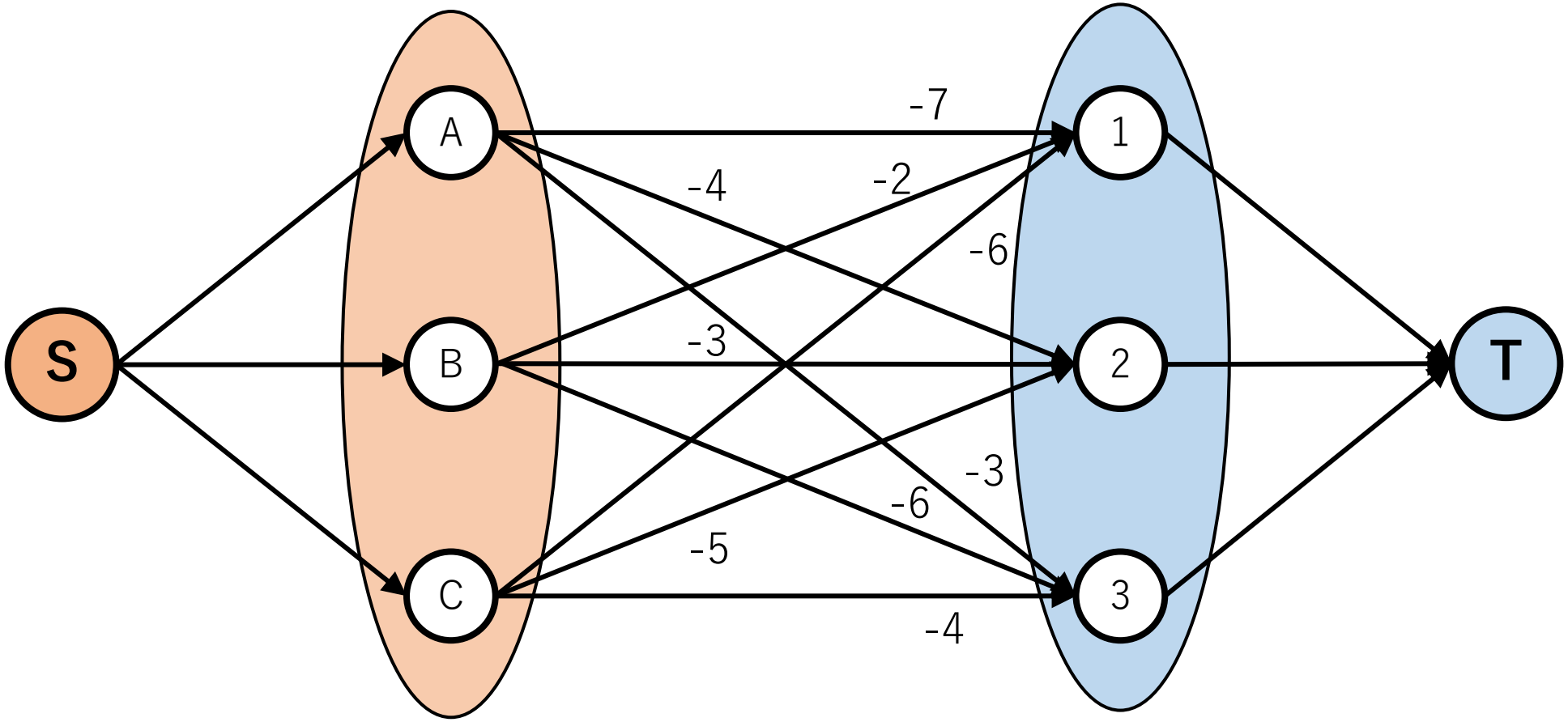
これを最小費用流問題に置き換えて考える.

辺の重みを負にして，辺のコストと見立てる.

元々の重みが大きければ大きいほど，辺のコストは小さくなる.

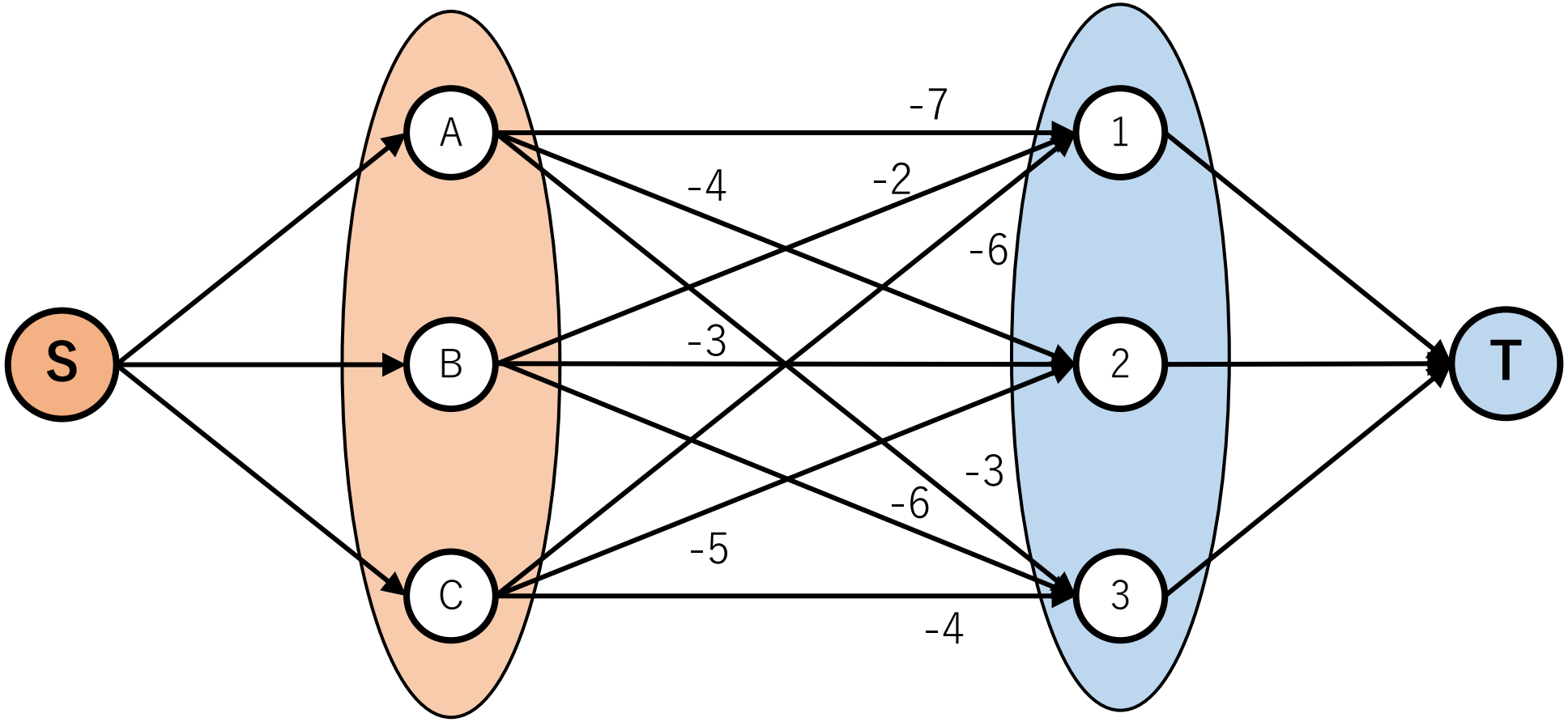
重み付き二部グラフの最大マッチング問題

仮想のSとTを用意し，さらに辺の費用を重みの負の値で設定．全ての辺の容積は1.



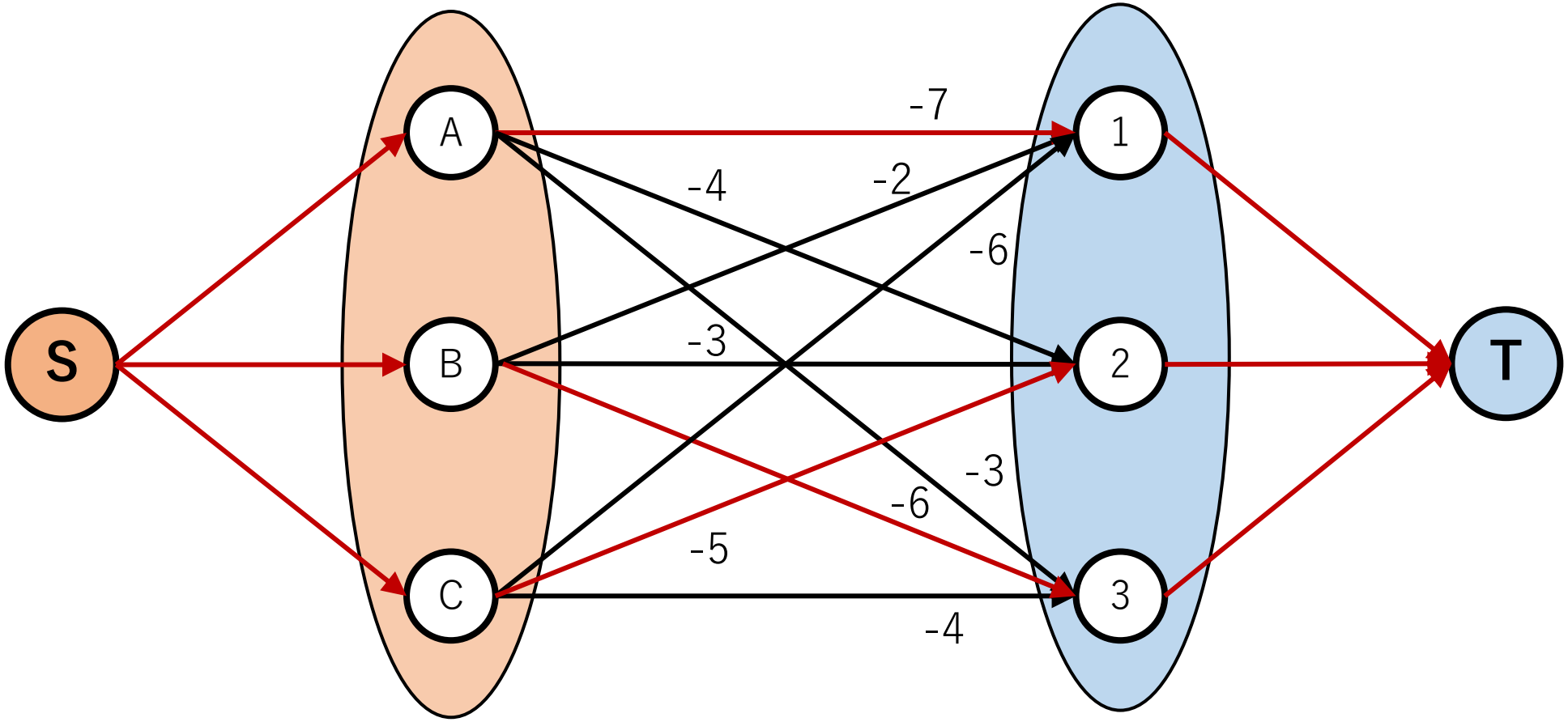
重み付き二部グラフの最大マッチング問題

ノード数（この場合3）分流す時の最小費用流を求める.



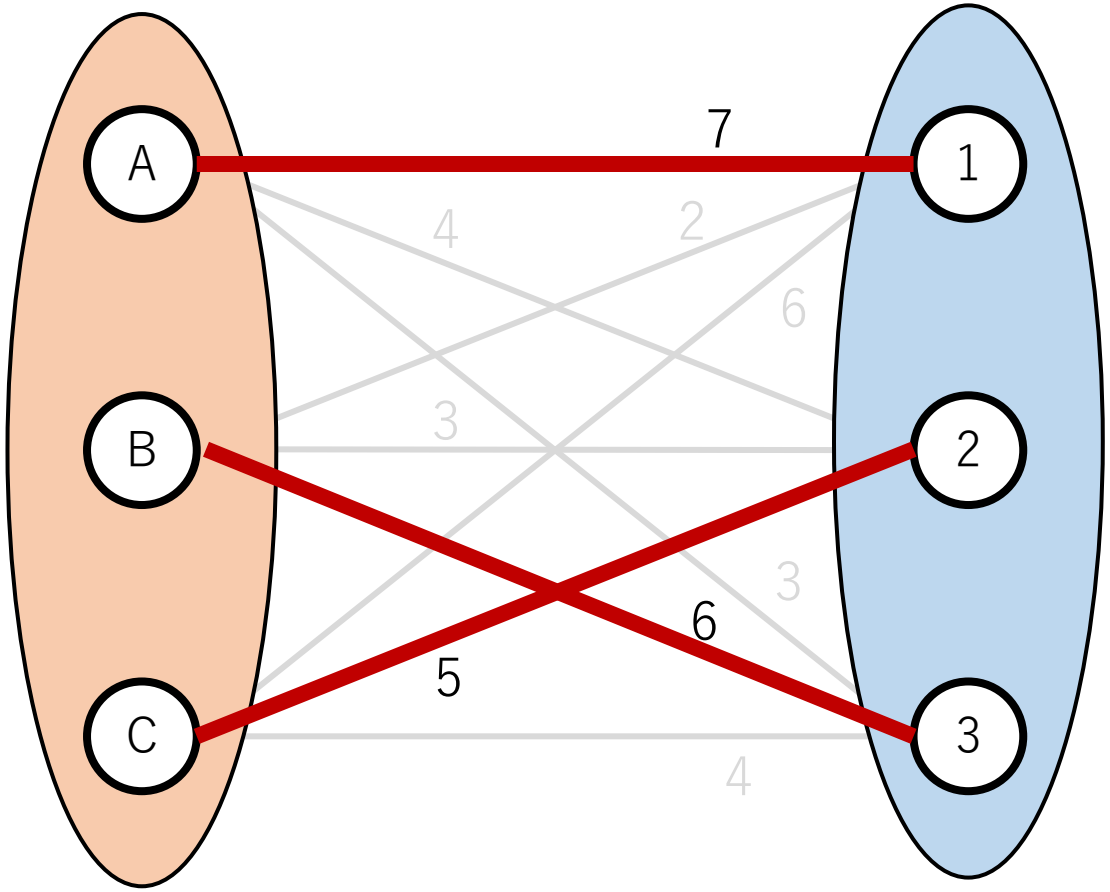
重み付き二部グラフの最大マッチング問題

ノード数（この場合3）分流す時の最小費用流を求める.



重み付き二部グラフの最大マッチング問題

SとTを削除すれば，最大マッチングになっている．



重み付き二部グラフの最大マッチング問題

重み付き二部グラフの最大マッチングには,

最大重み最大マッチング

マッチングの数を最大にしたうえで重みの和を最大化する

最大重みマッチング

マッチングの数は任意で重みの和を最大化

などがある.

先程の説明は最大重みマッチングになっている.

二部グラフのマッチング問題

全体最適を目指すアルゴリズムとなる.

全体最適のために場合によっては大きく不利益を被る個人が出てくることがある.

お互いに現在よりも好ましい組が存在しないようなマッチング方法もある（安定マッチング，ゲール・シャプレイのアルゴリズム）.

駒場での進学選択で使われています. 😊

http://www.c.u-tokyo.ac.jp/zenki/news/kyoumu/2016guidance_algorithm1128.pdf

つづいて,

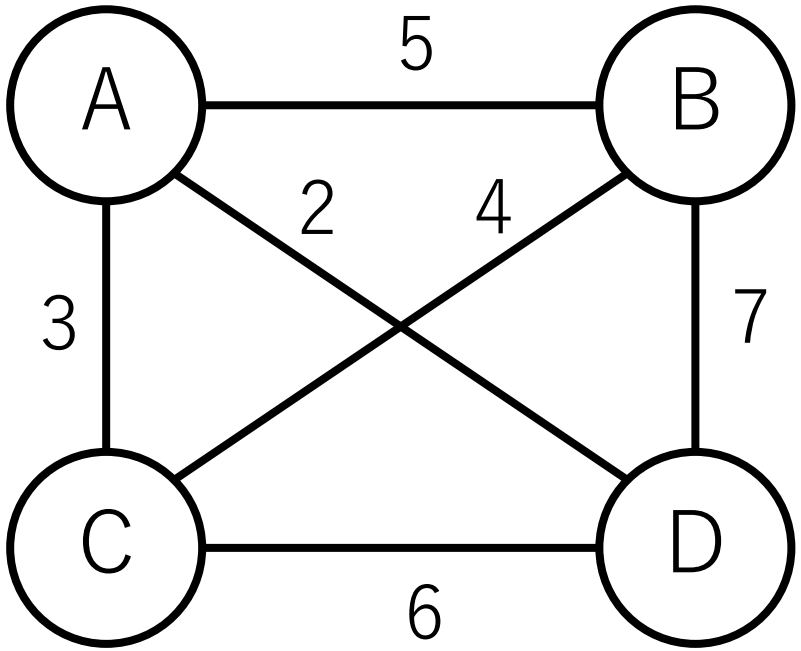
最小費用流問題

コンピュータにとって「難しい問題」とは？
そういう問題にぶち当たったときは？

さいごに

巡回セールスパーソン問題

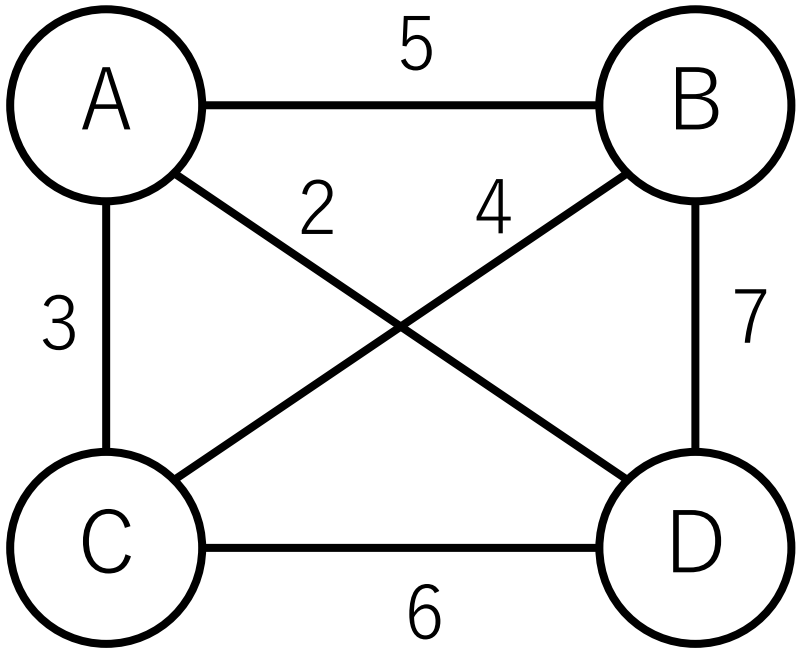
「複数の都市を全て必ず1回だけ通るように巡回し，
開始都市に戻ってくる時，最短の経路はなにか？」



巡回セールスパーソン問題

「複数の都市を全て必ず1回だけ通るように巡回し，
開始都市に戻ってくる時，最短の経路はなにか？」

A->C->B->D->A（逆でも可）と回って，16.



巡回セールスパーソン問題

この問題が解けると嬉しいことがいっぱいありそう！

指定した観光地を最も安い金額で周るツアーの設計
配達物を届けるための経路選択

などなど．．．

巡回セールスパーソン問題

「複数の都市を**全て必ず1回だけ通る**ように巡回し，
開始都市に戻ってくる時，最短の経路はなにか？」

単なる最短経路（全て必ず通る必要はない）であれば，
なんとかなるんだが．．．

巡回セールスパーソン問題

「複数の都市を**全て必ず1回だけ通る**ように巡回し，
開始都市に戻ってくる時，最短の経路はなにか？」

任意の2つの都市が連結している（完全連結グラフ）場合，
全探索するなら都市の順列を考えて，$O(n!)$.

巡回セールスパーソン問題

「複数の都市を**全て必ず1回だけ通る**ように巡回し，
開始都市に戻ってくる時，最短の経路はなにか？」

任意の2つの都市が連結している（完全連結グラフ）場合，
全探索するなら都市の順列を考えて，$O(n!)$.

コンピュータにとっての「難しい問題」

コーディングが複雑（コードに落とすのが難しい），
という意味ではない.

現時点において「効率的に」解くアルゴリズムが
知られていない問題.

「難しい問題」とはどんなものか？

以下のスライドでは，直感的な理解を重視して，ざっくりした説明をしています．

> チューリングマシンとかの説明を完全に無視しております．．．

厳密な解説は他の講義や資料におまかせしたいと思いますので，興味ある人はぜひご自身で探してみてください．

決定問題（判定問題）

ある入力に対してyesかnoの答えを求める問題.

答えの出力にかかる時間が入力に依存しない.

「与えられた整数が素数であるかどうか．」
「与えられた文字列が回文かどうか．」

P問題とNP問題

P = Polynomial（多項式）Time Solvable

多項式時間で解くアルゴリズムが存在する判定問題.

多項式時間：$O(n), O(n^k), O(\log n)$などなど.
指数関数よりは時間のかからないものを指す.

P問題とNP問題

P = Polynomial（多項式）Time Solvable

アルゴリズムの世界でいう「易しい問題」.
（現実的にはめちゃ遅いものも含まれるが．．．）

「効率的に」解くことのできる方法が知られている
問題とされる.

P問題とNP問題

NP = Non-deterministic Polynomial（非決定的多項式）
Time Solvable

Non-polynomialではないことに注意！

出力がyesとなる証拠があった時，その証拠を確認する
多項式時間のアルゴリズムが存在する判定問題.

P問題とNP問題

NP = Non-deterministic Polynomial（非決定的多項式）

「多項式時間で問題を解くアルゴリズムが存在しない」
ではない！

無限に並列計算できるなら，動かしている中のある組み合わせが，yesの出力になることはあり得るので，多項式時間で問題を絶対に解けないというわけではない.

P問題とNP問題

P問題は必ずNP問題でもある．ただし，その逆は必ずしも成立しない．

P問題：多項式時間で答えを出す方法がある

NP問題：多項式時間で答えを確認する方法がある

NPだけどPではない問題は，「難しい問題」として扱われる．

多項式時間還元（polynomial-time reduction）

ある問題から，別の問題へ変換することが多項式時間で可能であること．

「多項式時間変換」，「多項式時間帰着」などとも呼ばれる．

多項式時間多対一還元と多項式時間Turing還元がある．

多項式時間多対一還元

ある判定問題Aともう1つの判定問題Bにおいて，以下のような関係が成立する時，「判定問題Aは判定問題Bに**多項式時間多対一還元可能**である」という．

- ある多項式時間アルゴリズムYがあり，問題Aの入力を問題Bのある入力へと置き換える．
- 答えがyesとなるAの入力xの場合，Bの入力$Y(x)$の出力もyesとなる．
- 答えがnoとなるAの入力xの場合，Bの入力$Y(x)$の出力もnoとなる．

多項式時間多対一還元

判定問題Aを多項式で解く方法がわからないが，判定問題Bは多項式で解く方法が知られているとする．

この時，判定問題Aを判定問題Bに多項式時間で還元できれば，判定問題Bを解くことで判定問題Aを解くことができる．

すなわち，全体として多項式時間で処理できる．

この逆は必ずしも成立しない．

多項式時間多対一還元

多項式時間還元は，問題の「難しさの度合い」を議論するときに役に立つ.

判定問題Aの入力は多項式時間アルゴリズムYによって，判定問題Bの入力へとマッピングされる.

ただし，このマッピングは一対一とは限らない.

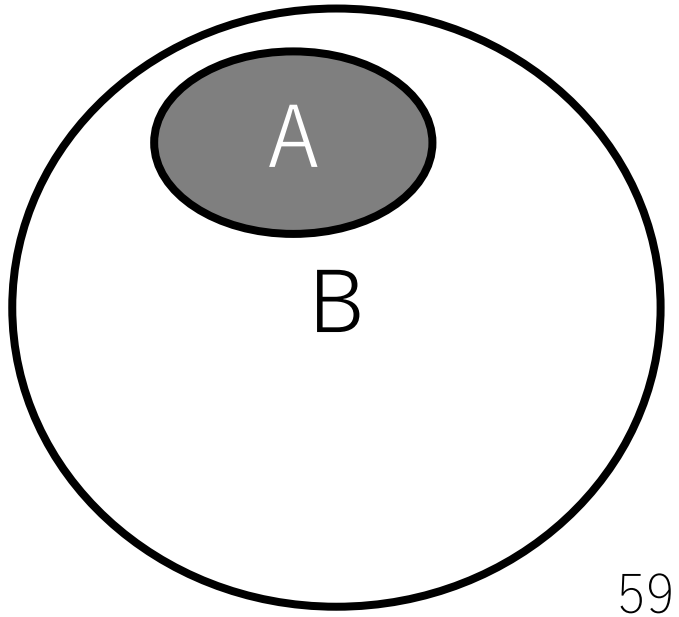
判定問題Aのいくつかの入力は判定問題Bのある1つの入力にマッピングされることもある.

多項式時間多対一帰着

つまり，判定問題Bから見たときには，判定問題Aは
判定問題Bが扱う入力スペースの一部しか扱わない問題，
というふうに見える．

たまたま入力スペースの
全部になることはある．

よって，判定問題Bは判定問題Aよりも
「広範囲の問題」を扱っており，「判定
問題Bは判定問題Aと同等かそれよりも
難しい」といえる．



多項式時間帰着

多項式時間帰着は問題を解く方法をみつけるために使える
ほか，多項式時間で解けないことを確認するためにも使える.

もし，判定問題Aが多項式時間で解けないとわかったとする.

すると，判定問題Bは判定問題Aと同等かそれよりも難しい
ので，判定問題Bも多項式時間で解けない，ということが
わかる.

多項式時間Turing還元

ある問題Pともう1つの問題Qにおいて，以下のような関係が成立する時，「問題Pは問題Qに**多項式時間**Turing**還元可能**である」という．

- 問題Pを解くために，問題Qを解くアルゴリズムZを利用する．
- アルゴリズムZ，および付随するその他の処理が多項式時間であり，かつ有限回実行されるものである．（Zや他の処理が有限回なら何回呼ばれてもよい．）

Turing還元は決定問題でなくても良い．

多項式時間Turing還元

Turing還元も他対一還元と同じように，「問題の難しさ」を扱う議論に利用できる．

他対一還元はTuring還元の特別なケース，とみることもできる．

NP完全（NP-complete）

全てのNP問題を多項式時間還元できるNP問題.

NP問題の中でも最も難しい問題とされる.

このようなNP完全問題が存在することは知られている.

NP困難（NP-hard）

どんなNP問題からでも多項式時間還元できる問題.

決定問題でなくても良い.

NP困難な問題自身はNPに属していなくても良い.

NP困難であり，かつNPに属する問題はNP完全となる.

NP困難（NP-hard）

NP困難は，NPと同等かそれ以上に難しいとされる．

NP（完全）「よりも困難」と覚えると良い．

NP完全，NP困難かがわかる意義

効率的に解けないことがわかる悪い知らせ？

NP完全，NP困難かがわかる意義

効率的に解けないことがわかる悪い知らせ？

世の中の他の人も効率的に解けない.

-> 落ち込む必要なし！

気持ちを切り替えて，「少しでもマシ」な方に向かうことができないか，を考えれば良い.

積極的に部分点を取りに行くイメージ.

NP完全，NP困難の例をみてみよう．

どんな問題がNP完全，NP困難であるかを知っていれば，「この問題はやばそうだぞ」というセンスを養うことができる．

NP完全：充足可能性問題（SAT）

入力：n個のBoolean変数に対する論理式

出力：論理式全体を真にする変数の組み合わせは存在するか？

例：$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3)$という論理式において真になる変数の組み合わせはあるか？

答え：yes．例えば，$x_1 = T, x_2 = F, x_3 = F.$

NP完全：充足可能性問題（SAT）

もし変数の組み合わせがわかれば，それを実際に代入して真になるかどうかを確認するのは，多項式時間でできる．

それを見つけようとすると単純には$O(2^n)$となる．．．

Steve Cook先生によりSATがNP完全であることが証明されている（世界で一番最初にNP完全であることが示された）．

NP完全：部分和問題

「n個の整数a[0]，a[1]，…，a[n-1]が与えられたとき，そのいくつかを組み合わせて総和がMにできるかどうかを判定せよ．」

ん？どこかでみたような．．．？

矢谷式DPの考え方：部分和問題の場合（再掲）

#4 コード化

2通りのパターンがある.

a[i]を入れる：dp[i][j] = dp[i-1][j-a[i]]

a[i]を入れない：dp[i][j] = dp[i-1][j]

この内，どちらか一方でもTrueならdp[i][j]もTrue.
そうでなければ，False.

（ただし，j-a[i]は0以上．そうでない場合，自動的に dp[i][j] = dp[i-1][j] ）

矢谷式DPの考え方：部分和問題の場合（再掲）

#4 コード化

2通りのパターンがある．

a[i]を入れる：dp[i][j] = dp[i-1][j-a[i]]

a[i]を入れない：dp[i][j] = dp[i-1][j]

この内，どちらか一方でもTrueならdp[i][j]もTrue.
そうでなければ，False.

（ただし，j-a[i]は0以上．そうでない場合，自動的に dp[i][j] = dp[i-1][j] ）

解けてるやん！

部分和問題ってDPで解けるのでは？

DPを使えば$O(nM)$なので，多項式なのでは？

実際それなりの時間で解ける？

部分和問題ってDPで解けるのでは？

DPを使えば$O(nM)$なので，多項式なのでは？
　　実際それなりの時間で解ける？

$M = 2^n$のような場合には，結局$O(n2^n)$となり，そのままでは多項式時間で解くことにならない.

このようなものは擬多項式時間（Pseudo-polynomial time）アルゴリズムと呼ばれている.

部分和問題ってDPで解けるのでは？

第1回目の講義でも説明したとおり，計算量の議論は入力が$O(n)$規模であることを前提としている.

指数的に変化するような場合などは考えない.

一方Mは入力の個数ではなく，こちらが勝手に決めることのできる数字（今回の例では部分和の値）．よって，これを2^nなどにしてしまうと，入力のサイズに応じて計算量が指数的に増加してしまうことになる.

NP完全：クリーク問題

「与えられたグラフにおいて，頂点数がkのクリークが存在するか」

クリーク：与えられたグラフの部分集合において，その頂点同士が全て互いに繋がっている（完全グラフになっている）もの.

NP完全：頂点被覆問題

「与えられたグラフが頂点の数kとなる頂点被覆が存在するか」

頂点被覆：与えられたグラフGのノードの部分集合Vを取り出したとき，Gの任意の辺において，その2つのノードの少なくとも1つはVに含まれているとき，VはGの頂点被覆という．

NP完全：ハミルトン閉路問題

「与えられたグラフにおいて，全ての頂点を一度だけ通る閉路が存在するか」

グラフに向きがあるときは有向ハミルトン閉路問題，向きがないときは無向ハミルトン閉路問題と呼ばれる．

NP困難：巡回セールスパーソン問題

「与えられたグラフにおいて，全ての頂点を一度だけ通る最短距離の閉路は何か」

もし判定問題（距離がLより短い閉路があるか）であれば，NP完全として扱える．

ハミルトン閉路問題は巡回セールスマン問題に多項式時間で帰着できる．

NP困難：ナップサック問題

DPの回で扱いましたが，こちらもNP困難.

部分和問題と同じく，擬多項式時間.

NP困難：スーパーマリオ

「ステージの最初からゴールまで辿り着けるかどうか」

ファミコンのゲームのいくつかは同様にNP困難であることが証明されている.

Classic Nintendo Games are (Computationally) Hard

Greg Aloupis, Erik D. Demaine, Alan Guo, Giovanni Viglietta

We prove NP-hardness results for five of Nintendo's largest video game franchises: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokemon. Our results apply to generalized versions of Super Mario Bros. 1-3, The Lost Levels, and Super Mario World; Donkey Kong Country 1-3; all Legend of Zelda games; all Metroid games; and all Pokemon role-playing games. In addition, we prove PSPACE-completeness of the Donkey Kong Country games and several Legend of Zelda games.

https://arxiv.org/abs/1203.1895

https://www.technologyreview.com/s/427197/super-mario-bros-proved-np-hard/

NP完全，NP困難

以上の問題の多くは「組み合わせ最適化問題」と呼ばれるカテゴリに入る．

「XXXをうまく選んで，YYYになるか」などのような問題は雲行きが怪しい可能性あり．

ただ，世の中の現実問題はこういうものが結構多い．

NP完全，NP困難にぶち当たったとき

最新の研究を調べてみよう！

巡回セールスパーソン問題もDPを使うと，$O(2^n n^2)$に
計算量を落とせるほか，さらなる手法が研究されている．

Waterloo大学の研究者によるConcorde algorithmなどがある．
http://www.math.uwaterloo.ca/tsp/

ちょっとややこしいこととして，

P，NP，NP完全，NP困難は，「問題」の相対的な困難さをクラス分けしたもの．

これらは「アルゴリズムの計算量」の議論を直接行うものではないので，計算量に関してはアルゴリズムごとの議論が必要となります．

巡回セールスパーソン問題のアルゴリズムが$O(2^n n^2)$になるからといって，他のNP困難の問題が同様の計算量で解けるとは限らない．

NP完全，NP困難にぶち当たったとき

近似アルゴリズムで行く方針に切り替え.

最適解でなくても，そこそこいい解なら，そこそこ
十分なことも.

様々な方法が知られており，また問題ごとに特化した
アルゴリズムも存在する.

最近傍法

手元にある選択肢のうち，とりあえず一番良いものを選んでいく．

巡回セールスパーソン問題ならば，「今いる都市から，未訪問の都市のうち最短距離の都市へと移動する．」

巡回セールスパーソン問題の場合は後の方で長い距離の移動が発生することが多くなり，結果的に非効率になることが多い．

局所探索法

現在までに見つかっている解を少しずつ改良していくイメージ.

#1：解をとりあえず見つける（最近傍法で見つける，ランダムに生成するなど）.

#2：現在の解に「ちょっと変更したもの」を考える.

#3：変更したものの方が良ければ，現在の解を置き換える.

#4：#2，#3を何度か繰り返す.

局所探索法

#1：解をとりあえず見つける（最近傍法で見つける，ランダムに生成するなど）．

#2：現在の解に「**ちょっと変更したもの**」を考える．

#3：変更したものの方が良ければ，現在の解を置き換える．

#4：#2，#3を何度か繰り返す．

「ちょっと変更したもの」＝近傍解と呼ばれる．近傍，近傍解は手法によって色々な定義がある．（巡回セールスパーソン問題でいれば，例えばある2つの都市の訪問順序を入れ替えるなど．）

局所探索法の例

山登り法：現在考えられる全ての近傍解のうち，最も良いものと比較し，必要に応じて入れ替える．

焼きなまし法：ある近傍解と設定した確率に応じて入れ替える（一時的に悪くなる可能性がある）．

タブーサーチ：現在考えられるいくつかの近傍解のうち，最も良いものと必ず入れ替える．ただし，入れ替え後はある期間再度入れ替えできない．

NP完全，NP困難かがわかる意義（再掲）

効率的に解けないことがわかる悪い知らせ？

世の中の他の人も効率的に解けない.

-> 落ち込む必要なし！

気持ちを切り替えて，「少しでもマシ」な方に向かうことができないか，を考えれば良い.

積極的に部分点を取りに行くイメージ.

みなさま，最後までお付き合いいただき
誠にありがとうございました！

累積和，しゃくとり法

ユークリッドの互除法，素数判定，エラトステネスの篩，繰り返し自乗法，剰余をとる場合の四則演算

キュー，スタック，連結リスト，二分木，ヒープ，セグメント木，BIT

線形探索，二分探索，二分探索木，平衡木，ハッシュ

ボゴソート，挿入ソート，ツリーソート，バブルソート，シェーカーソート，シェルソート，クイックソート，マージソート，ヒープソート，バケットソート

文字列照合での力任せ法，KMP法，BM法，BMH法，ローリングハッシュ

動的計画法（フィボナッチ数，コイン問題，ナップサック問題，レーベンシュタイン距離，DTW，貰うDP vs 配るDP）

グラフデータ構造，BFS，DFS，オイラーツアー，LCA，橋の検出

ダイクストラ，ベルマンフォード，SPFA，ワーシャルフロイド

クラスカル法，Union-Find木，プリム法，トポロジカルソート

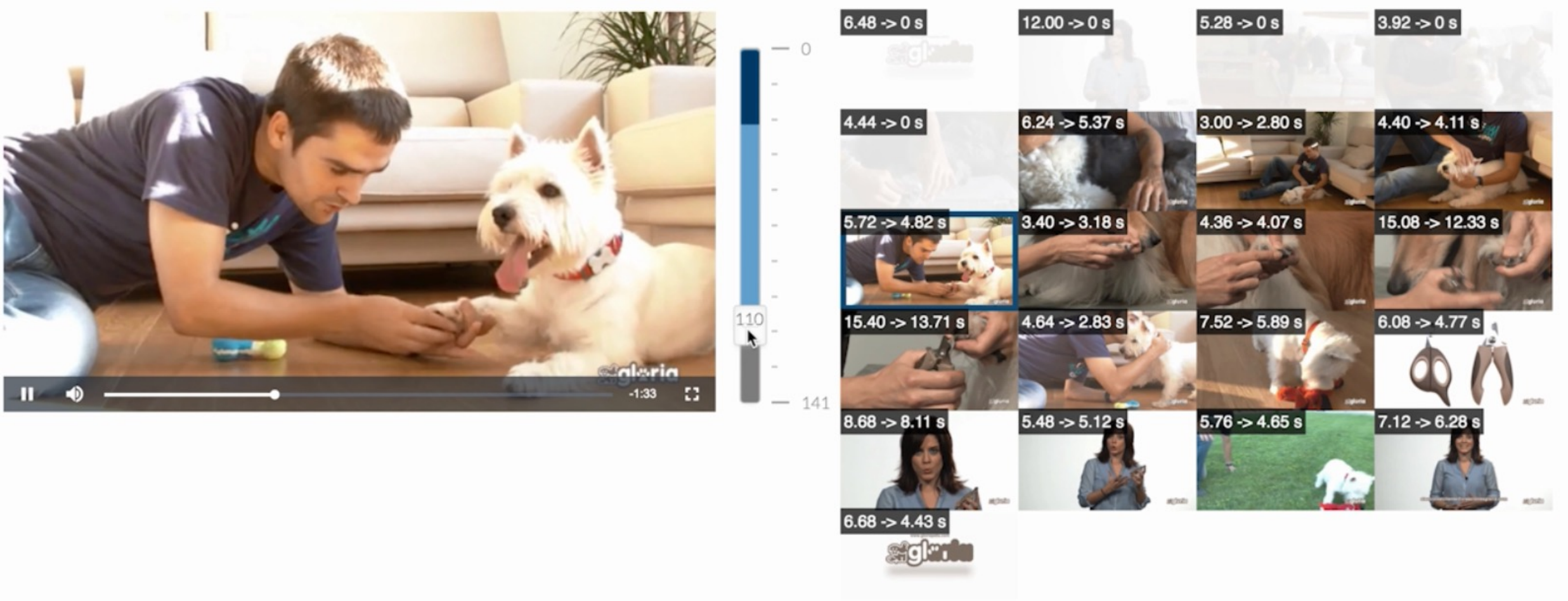
フォードファルカーソン，プライマルデュアル，二部グラフのマッチング

計算論（P，NP，NP完全，NP困難）

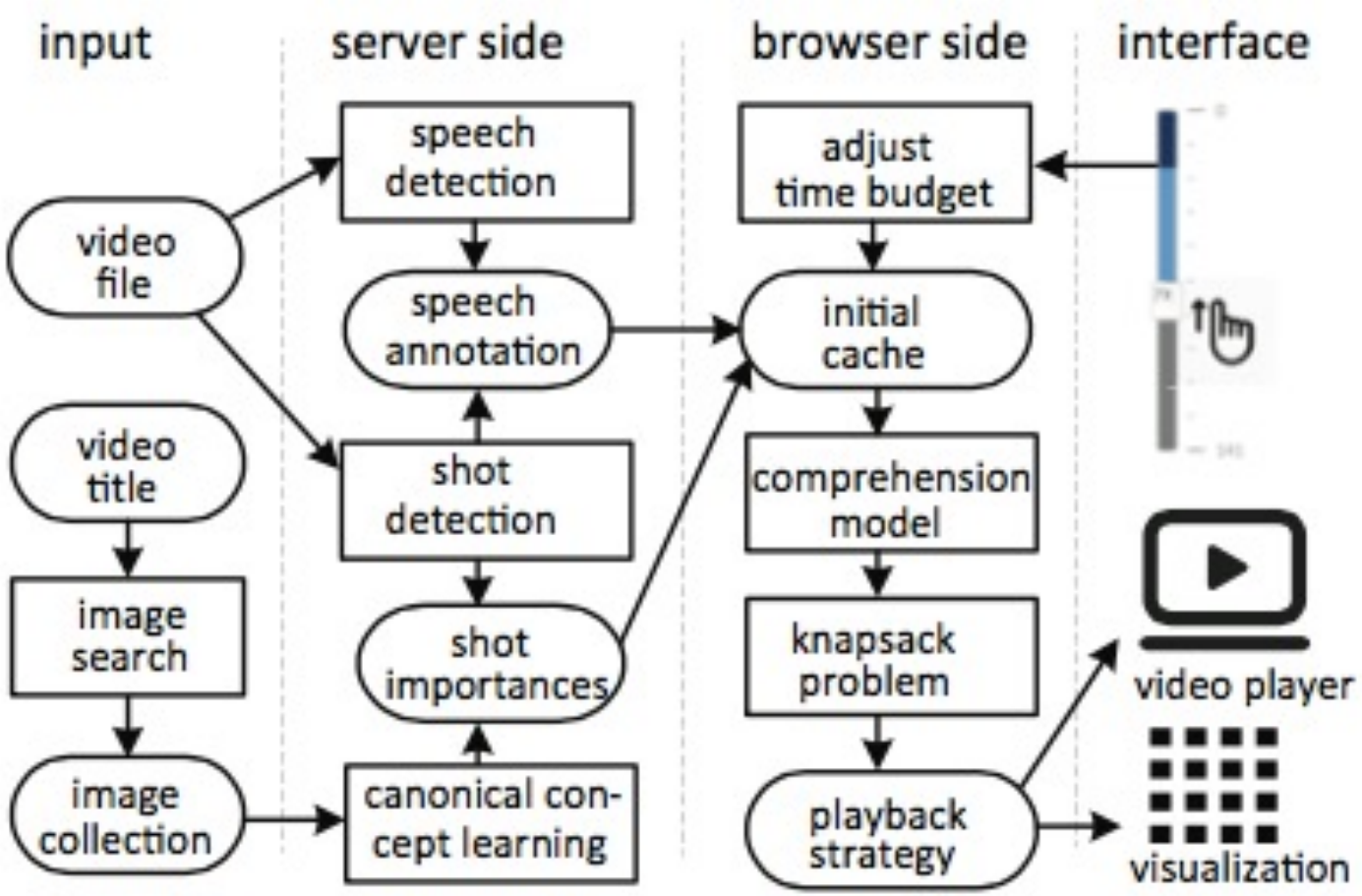
何ができるの？

ElasticPlay: 視聴時間を指定できる動画再生インタフェース

Haojian Jin, Yale Song, Koji Yatani. ElasticPlay: Interactive Video Summarization with Dynamic Time Budgets. In Proceedings of ACM MM 2017 (oral presentation).

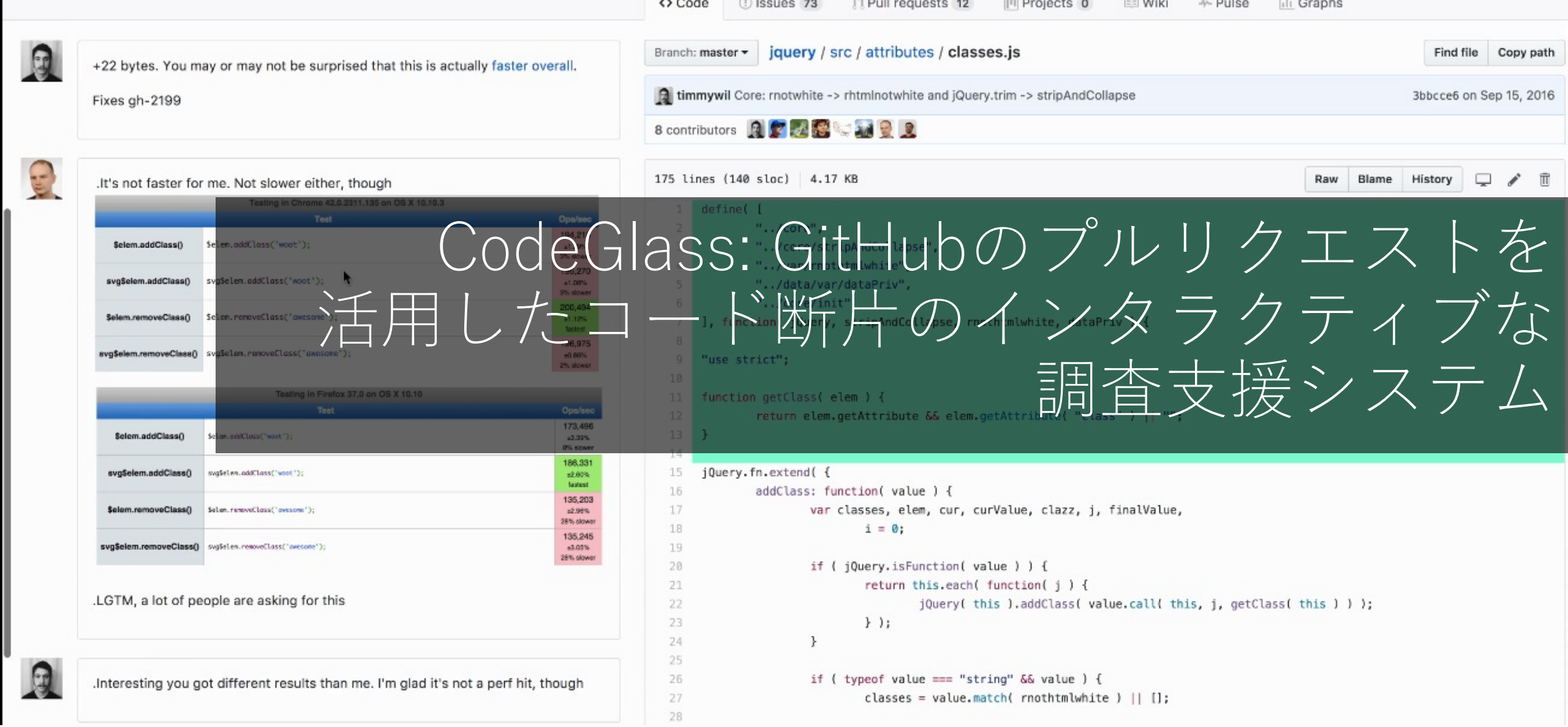


早送りと取捨選択を組み合わせたビデオの加速再生．ユーザは何秒でビデオを見終えたいか，を設定するだけで自動的に再生方針を決定する．



動画を音声がある部分とそうでない部分に分け，さらに重要度を推定し，重要度の低い部分は削除，また音声がない部分はより高速に早送りするようにして，ユーザが指定した再生時間に合わせた再生方法を自動的に生成.

CodeGlass: GitHubのプルリクエストを活用したコード断片のインタラクティブな調査支援システム

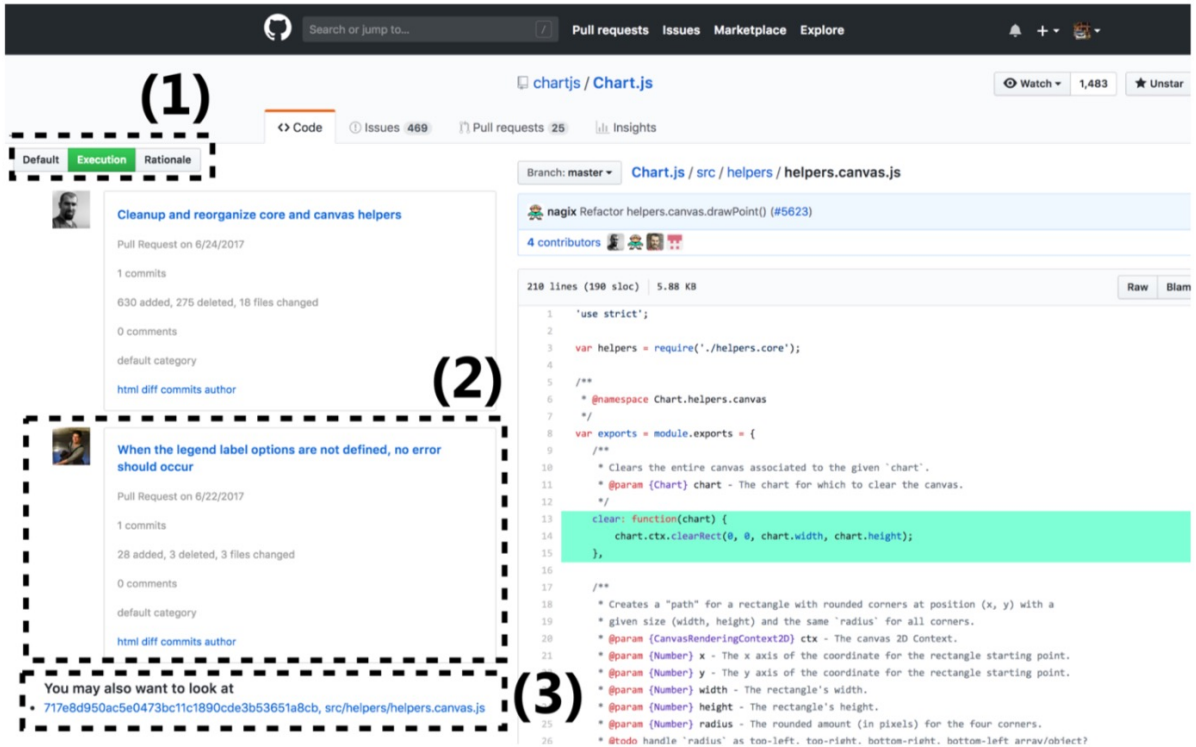


柴藤大介，有薗拓也，宮崎章太，矢谷浩司．CodeGlass: GitHubのプルリクエストを活用したコード断片のインタラクティブな調査支援システム．インタラクション2019（フルペーパー）．

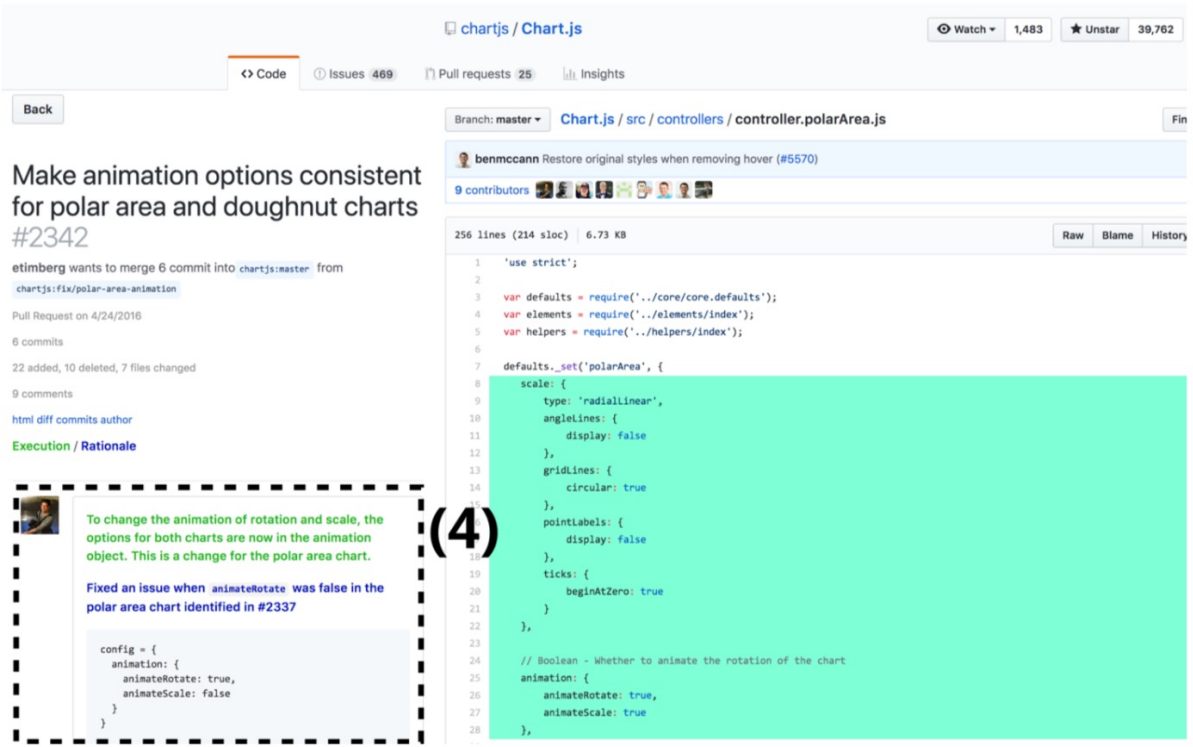
柴藤大介．「Interactive Piece-level Code Examination Using Diffs and Review Comments」東京大学工学部電子情報工学科卒業論文，2016年．優秀卒業論文賞受賞

CodeGlass: GitHubのプルリクエストを活用したコード断片のインタラクティブな調査支援システム

柴藤大介，有薗拓也，宮崎章太，矢谷浩司
IIS-Lab，東京大学



(a) 関連するプルリクエストの一覧画面.



(b) プルリクエストの詳細画面.

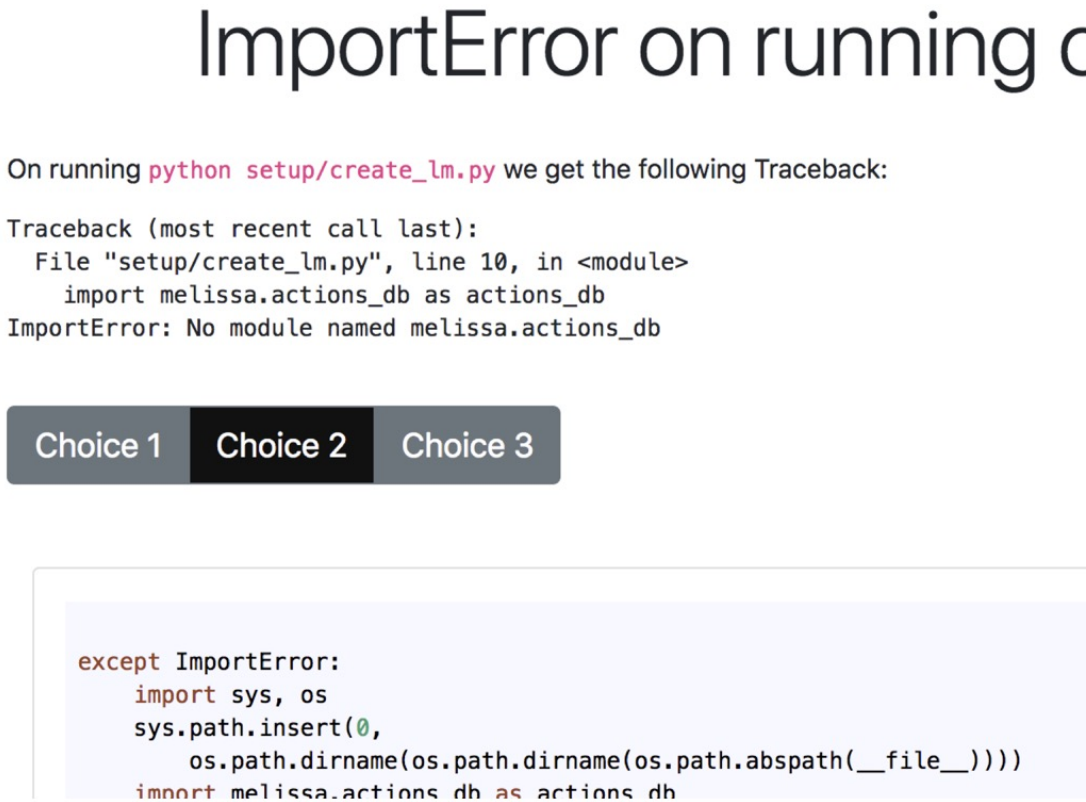
(1)時系列順，実装内容に関する情報が多い順，開発背景に関する情報が多い順に，関連するプルリクエストをソート可能.

(2)各プルリクエストの概要情報.

(3)選択されたコード断片と一致する可能性があるコード断片が過去のバージョンで複数ある場合には，そのバージョンにおけるソースコードへのリンクが表示.

(4)プルリクエストの詳細情報．実装内容に分類される文章は緑色で，開発背景に分類される文章は青色で表示される.

RealCode: GitHub上にある実コード変更をプログラミング課題に転用するシステム



柴藤大介，矢谷浩司．「GitHubのプルリクエストを用いたプログラミング課題自動生成システムの実現可能性に関する検討」第80回情報処理学会全国大会，2018年3月．学生奨励賞受賞

問題文: Issueの説明文

コードの解答例: 実際のコード変更

解答の説明文: Pull request内の説明文

存在するPull requestを利用して，実世界のコード変更をプログラミング課題として利用.

3種類のインタフェース（フリップカード，選択式，穴埋め式）で課題を提供．

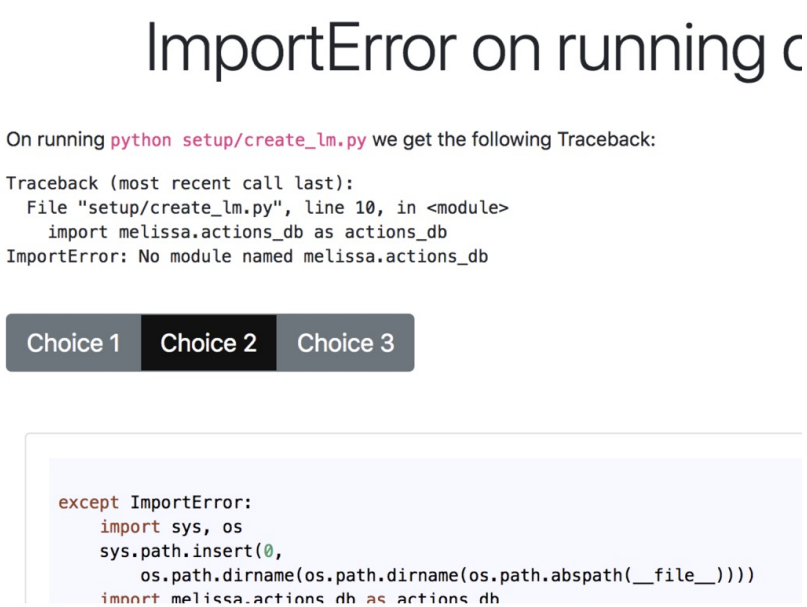
On running `python setup/create_lm.py` we get the following Traceback:

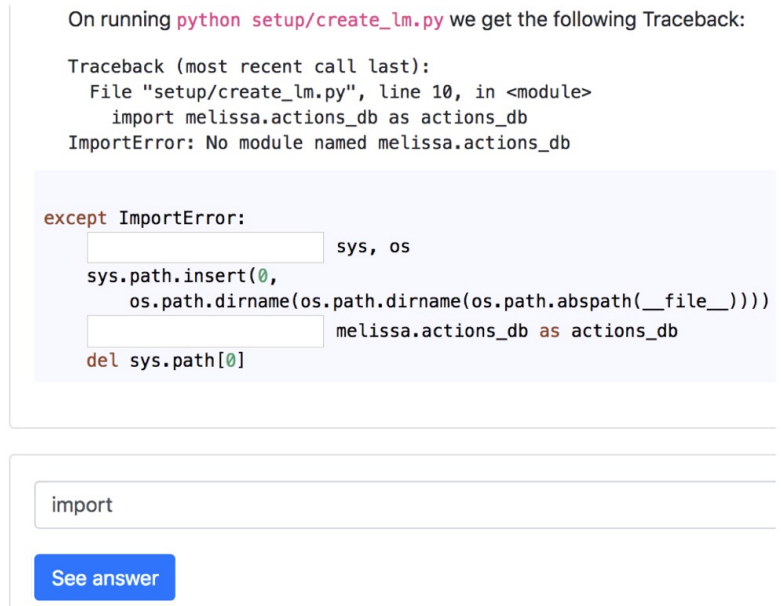
```
Traceback (most recent call last):
  File "setup/create_lm.py", line 10, in <module>
    import melissa.actions_db as actions_db
ImportError: No module named melissa.actions_db
```

Click to flip

```
except ImportError:
    import sys, os
    sys.path.insert(0,
        os.path.dirname(os.path.dirname(os.path.abspath(__file__)
    import melissa.actions_db as actions_db
    del sys.path[0]
```

Click to flip





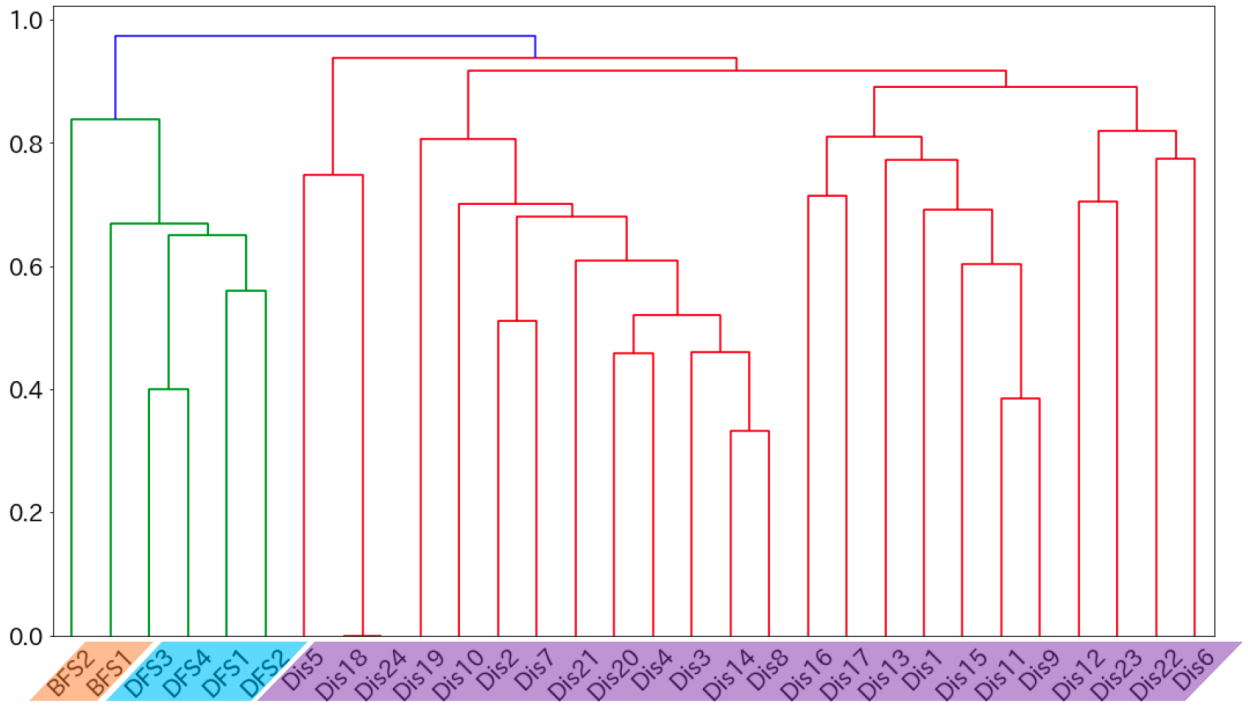
実開発を体験できるプログラミング課題を提供できる可能性を確認.

「あんまりこういうエラーが発生してそれにどう対処すべきか、みたいな問題を経験したことがないから、なんか新鮮で面白そう。」

「開発の背景というか現状把握に時間がかかるけど、話にストーリーがあって面白い。（~ would be great, というセリフを見て）何かのサービスを改善したい、から問題が始まってるのは、学習のモチベーション的にいい。」

「コード内にコメントがたくさんあって、なんとなくこう可読性というか人に見られることを意識していていいね。」

変数の値の変化を利用したソースコード分析手法



鈴木 凌斗，矢谷 浩司． 「変数の値の変化を利用したソースコード分析手法の提案」 ヒューマンインタフェースサイバーコロキウム2020．2020年10月．

変数の振る舞いの類似性が処理の類似性に関連すると考えたコードの分類手法.

2つのコードにおいて同じ入力を与え，コード内の変数がどのように変化するかを記録.

2つのコード間においてどの変数同士が最もよく似ているかを判定するために二部グラフのマッチングを利用.

コードの分析に変数の振る舞いを利用.

アルゴリズムの知識は，計算理論に限らず，
信号処理，データ処理，ソフトウェア工学，
HCIなど，色んな場面でとっても役に立つ！

新しい応用先をぜひ一緒に考えましょう！😊

色んな所で腕試ししてみよう！

競技プログラミング

与えられた課題に対して制限時間内に設定された要件を満たすようなコードを書き，提出する．

AtCoder: https://atcoder.jp/

AOJ: http://judge.u-aizu.ac.jp/onlinejudge/

ACM ICPC: https://icpc.iisf.or.jp/

などなど．

競技プログラミング

形式的にはこの講義でやったExtra課題のようなもの.

Pythonに限らずいろんなプログラミング言語で参加可能.

国内のコンテストは日本語でも参加可能.

過去のTAさん曰く,

「この授業を通してアルゴリズムに惹かれてatcoderにも挑戦しているのですが、この授業を一通り受講してある程度練習したら目安としてどのくらいのレベルまで狙えるとかありますかね…？」

→AtCoderであればここでの知識のみであっても水色中盤あたりにはいくのではないかなと思っています

→水色(上位15%)くらい狙えるのかなと思います。Extra課題の難易度が平均的にそれくらい(上位15%が解けるかどうか、くらい)になってると思います.

ぜひチャレンジしてみてください！

この講義で勉強したことを踏まえれば解くことのできる問題はそれなりにあるはず.

パズル的要素も多いですが，考えたものをコードに落とし込むエクササイズとしては，とても良いと思います.

ただし，競技プログラミングだけが生きていく道ではないので，自分にあった付き合い方を見出してもらうのが，自分の興味を持続させるコツかと思います.

期末試験

日時：7/31 12:55**集合，**13:05**開始，**14:40**頃解散予定**

試験時間：80分

会場：工学部2号館4階241，242教室を予定.

後日，座席を指定しますので詳細をお待ち下さい.

持ち込みは認めません.

【重要】期末試験受験者調査

期末試験を受験する予定の人は全員，以下のアンケートで回答をしてください．

https://forms.gle/7XdDFwtKsboLkWS68

回答期限：7/12 24:00

このアンケートに回答していない場合，期末試験の受験を認めないことがあります．

期末試験

受験の際には座席が指定されています．自分の席に間違いなく着席してください．

試験集合時間（12:55）までに必ず集合してください．

13:30まで遅刻を認めますが，試験時間の延長はありません．それ以降は入室を認められません．

期末試験に準備すべきもの

筆記用具，学生証.

持ち込み不可.

電子機器，スマートデバイスは全て体から外し，自分のかばんにしまっておいてください.

期末試験の試験範囲

授業1回目から13回目の最小費用流問題まで.

「コンピュータにとって「難しい問題」とは？」（計算論）と「さいごに」の内容は試験範囲に含まれません.

期末試験の前提

授業で説明した内容，用語，実装例，計算量を前提に出題をしますので，スライド・講義動画を使って十分に復習をしておいてください．

Extra課題レベルの難易度の問題は想定していません．

次に皆さんとお会いするのは

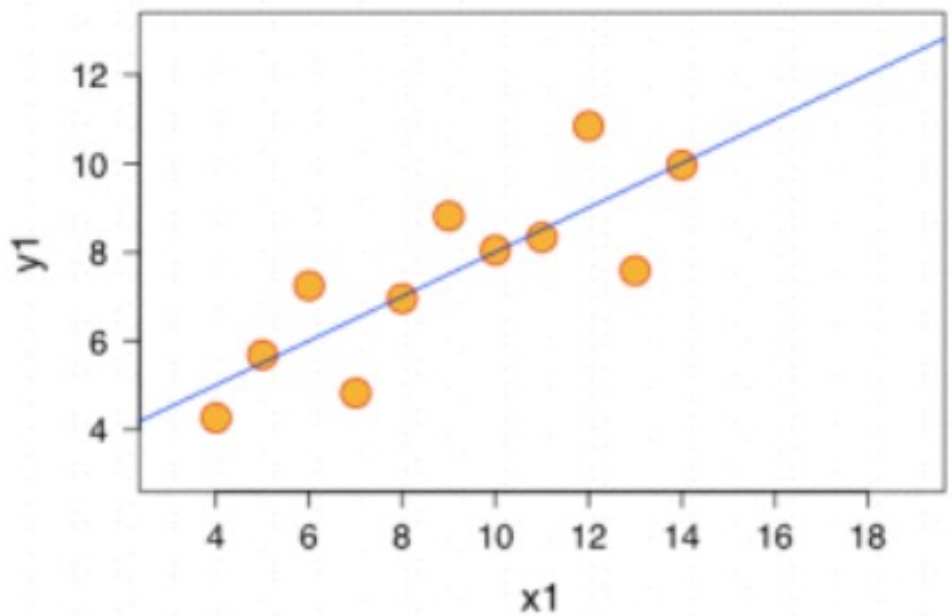
後期実験「情報可視化とデータ解析」になります！
https://iis-lab.org/infovis

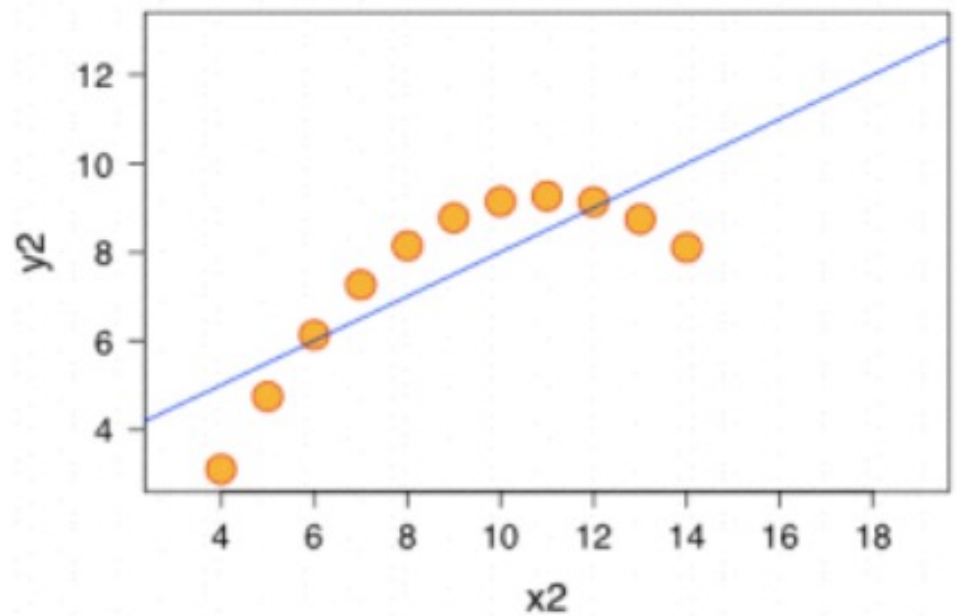
何が違うでしょうか？

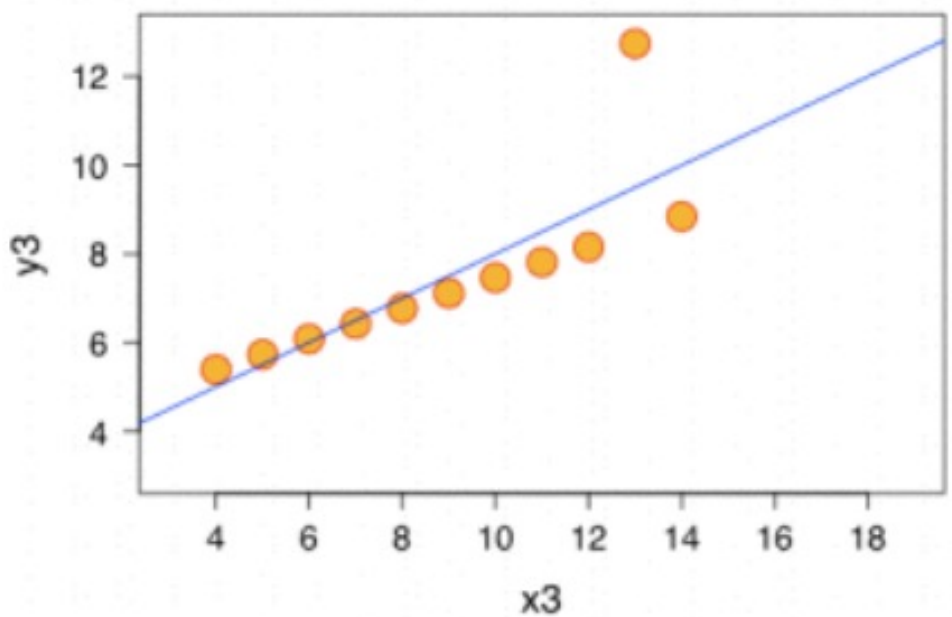
Anscombe's Quartet: Raw Data

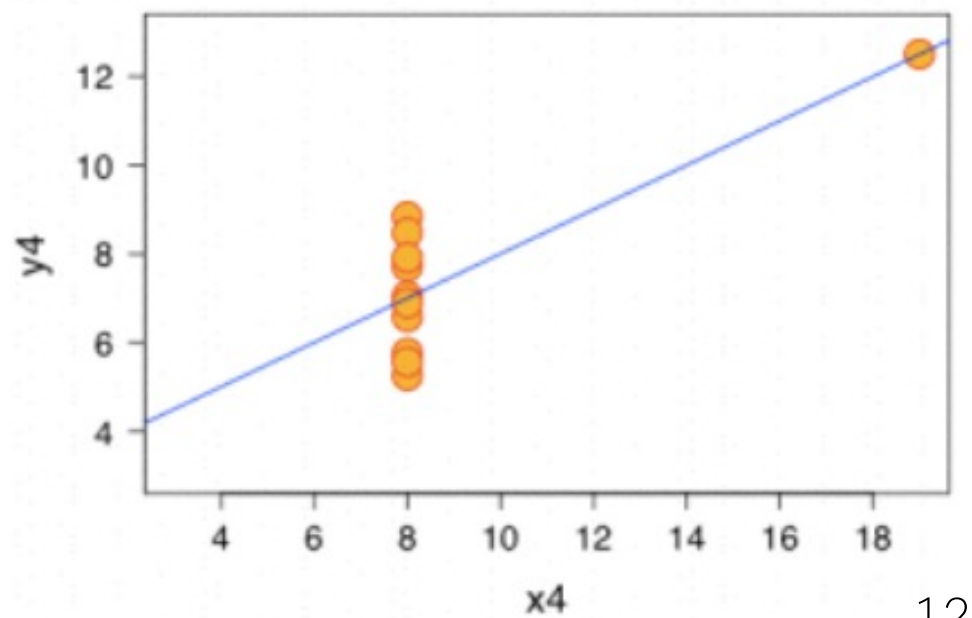
	I		II		III		IV	
	x	y	x	y	x	y	x	y
	10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
	8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
	13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
	9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
	11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
	14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
	6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
	4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
	12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
	7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
	5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89
mean	9.0	7.5	9.0	7.5	9.0	7.5	9.0	7.5
var.	10.0	3.75	10.0	3.75	10.0	3.75	10.0	3.75
corr.	0.816		0.816		0.816		0.816	

[Anscombe 1973]


y1
x1
4
6
8
10
12
14
16
18


y2
x2
4
6
8
10
12
14
16
18


y3
x3
4
6
8
10
12
14
16
18


y4
x4
4
6
8
10
12
14
16
18

次に皆さんとお会いするのは

後期実験「情報可視化とデータ解析」になります！
https://iis-lab.org/infovis

ユーザ（人間）がデータを見て分析したり，物事の判断をしたりするためにどのように可視化を設計するかを学んでいきたいと思います．

また，実世界のデータを使って皆さんなりの可視化システムを作っていただきます．😄

昨年度からの変更

JavaScriptとD3を用いてシステムを作ってもらいますが，
ChatGPTをフル活用してプロトタイピングをしてもらう
予定です．😄

コードを書きまくるというよりも，生成AIを駆使して
インタラクティブなシステムの反復的なプロトタイピング
を高速に行い，改良をしていくプロセスを体験してもらう
予定ですので，お楽しみに．

授業のアンケートにご協力ください！

ぜひ皆さんのご意見をお聞かせください！
https://forms.gle/EWGxvUTDYMai71Sv5

来年度のTAさんも募集しておりますので，我こそはという方はぜひ！😊

一緒にExtra問題を考えたり，受講生のサポートをしましょうー.

slackでDM，もしくはメールを下さい.

UTAS上での授業アンケート（本学学生のみ）

こちらもどうぞよろしくお願いいたします．こちらは工学部で行っているものになりますので，こちらにもご意見いただけるとありがたいです．

それでは期末試験で.

Good luck! 🙂