最適化 / Optimization

スタートアップゼミ 第4回

2024/4/25

M2 林由翔

最適化 / Optimization

「何らかの制約条件のもと，**目的関数を最小化 (or 最大化)**したい！」

Minimize or maximize objective function under some constraints!

例	制約条件	目的関数
不採算路線をめぐる地域交通計画		
デマンド交通※の配車計画 ※予約に合わせて運行スケジュールを決定する乗り合いの公共交通		
交通量観測地点の決定問題		

手法を考える前に，**制約条件と目的関数を厳密に定義することが重要**

最適化 / Optimization

「何らかの制約条件のもと，**目的関数を最小化 (or 最大化)**したい！」

Minimize or maximize objective function under some constraints!

例	制約条件	目的関数
不採算路線をめぐる地域交通計画	• 予算	• 便益の最大化
デマンド交通※の配車計画 ※予約に合わせて運行スケジュールを決定する乗り合いの公共交通	• 台数 • バッテリー容量	• 利用者の待ち時間最小化 • 運用コスト最小化
交通量観測地点の決定問題	• 機器台数	• 取得情報量の最大化

手法を考える前に，**制約条件と目的関数を厳密に定義することが重要**

推定でも登場する最適化 / Optimization in Estimation

- 最尤推定では，モデルの式形とデータから求められる「尤度関数」を最大化する．

MNLの対数尤度関数: $\sum_i \log P(y_i; x_i) = \sum_i \frac{\exp(V_{y_i})}{\sum_j \exp(V_j)}$

- 機械学習モデルの学習過程も最適化といえる

いわゆる最適化研究をやらない人でも，**最適化についての知識**があれば，計算速度や精度を既存手法より良くできるかも！

Optimization is also important for researchers working with estimation problems.

連続最適化と組合せ最適化

- 最適化問題は**連続最適化**と**組合せ最適化**に大別される
- 適用される解法が全く異なる

連続最適化
Continuous optimization

目的関数に登場する変数が連続量(実数など)

例: 線形計画問題 / Linear programming

$$\max(x_1 + x_2) \quad \text{目的関数}$$
$$s.t.$$
$$x_1 \geq 0, x_2 \geq 0 \quad \text{制約条件}$$
$$3x_1 + 4x_2 \leq 10$$

組合せ最適化 (離散最適化)
Combinatorial optimization

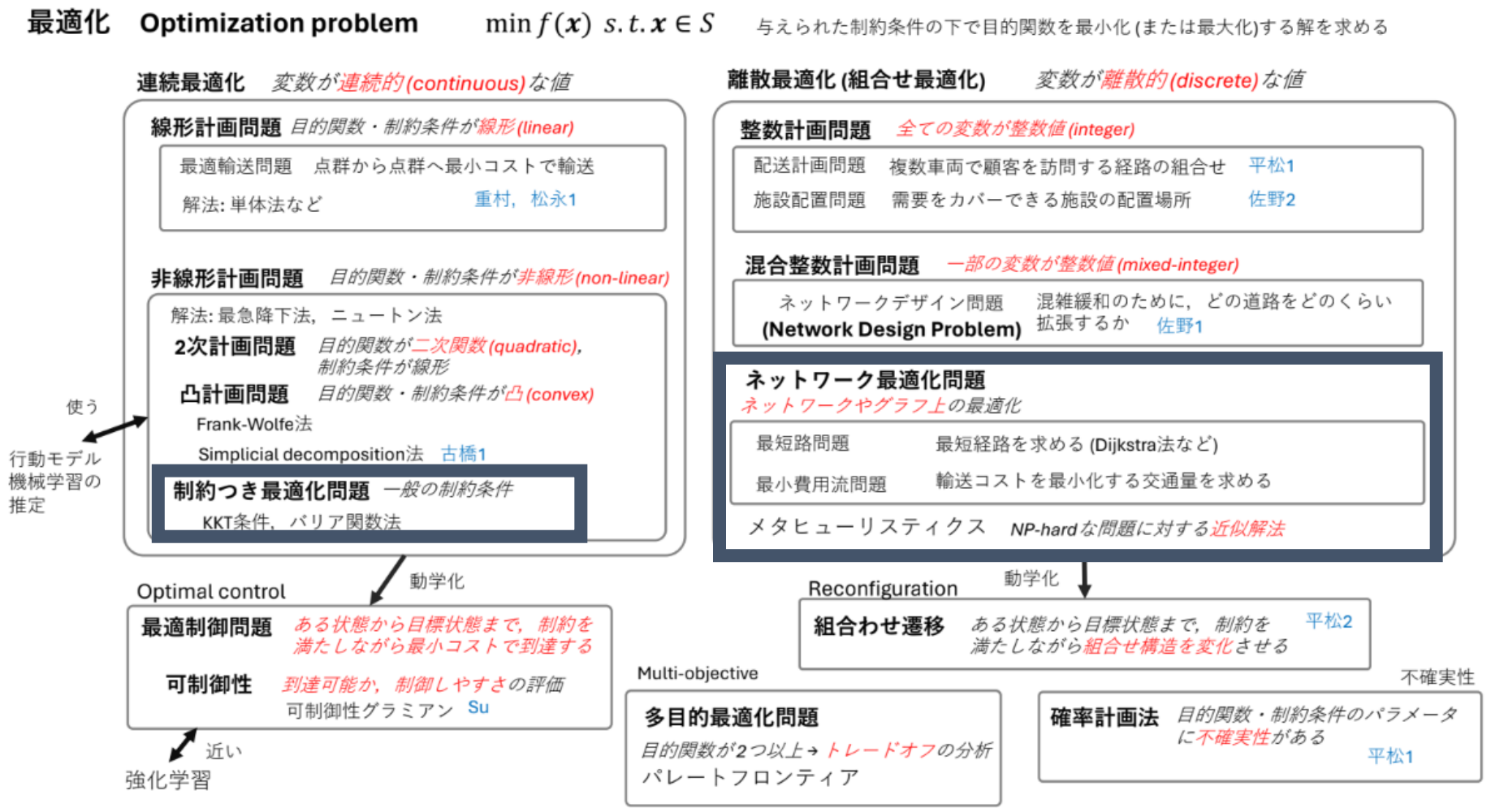
目的関数に登場する変数が離散量(整数など)

例: 最短経路問題 / Shortest path problem

どの辺を使うか？が0-1変数となる

今日扱う範囲 / Today's topics

今年度の理論談話会で扱わないトピックを重点的に扱います．



2024年理論談話会#1(増田さん発表)

連続最適化

Continuous optimization

連続最適化問題の解析解法と数値解法

- 目的関数の式形次第では，数式で最適解を導ける場合もある
  （=**解析解** / Analytical method）

  - 例) $f(x) = ax^2 + bx + c, a > 0$

  頂点は $\left(-\frac{b}{2a}, -\frac{b^2}{4a} + c\right)$ なので，$\arg\min f(x) = -\frac{b}{2a}$

  - 経済学分野の論文で多い印象？

- 解析解が不明なら，数値計算を繰り返すことで解を探す
  （**数値解法** / Numerical method）

解析解法 / Analytical method

解析解法は，局所解 (≒目的関数の傾きが0になる点) を求めるのが基本

- 最適解の必要条件を与えるが，十分条件ではない場合も
  →後述の数値解法と組み合わせて探索

KKT条件

非線形計画問題

$$\min_x f(x)\ s.t. g_i(x) \leq 0, h_J(x) = 0$$

の最適解$\bar{x}$の必要条件は，

$$\nabla f(\bar{x}) + \sum_i \lambda_i \nabla g_i(x) + \sum_j \mu_j \nabla h_j(\bar{x}) = 0$$

$$\lambda_i \geq 0, \lambda_i g_i(x) = 0$$

十分条件ではない

HJB方程式

最適制御問題

$$\min_{\mathrm{u}} \left\{ G(x(T)) + \int_0^T L\big(x(t), u(t)\big) dt \right\} s.t.\ \dot{x} = f(x, u, t)$$

の最適解$\bar{u}$の必要十分条件は，

$$V(x,t) + \min_u \{\nabla V(x,t) \cdot f(x,u,t) + L\big(x(t), u(t)\big)\} = 0$$

$$V(x,T) = G(x(T))$$

そもそも↑式が解析的に解けない

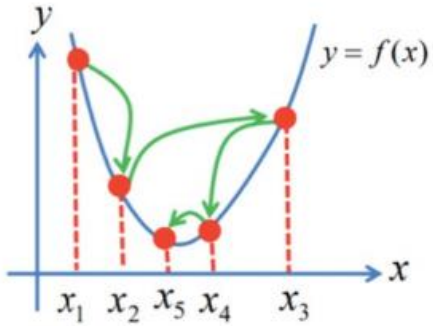
数値解法 (復習) / Numerical method

UE/FDの解法

■ **非線形最適化問題の一般的な解法**

ステップA：降下方向の探索
どの方向に向かえば目的関数が減少するか

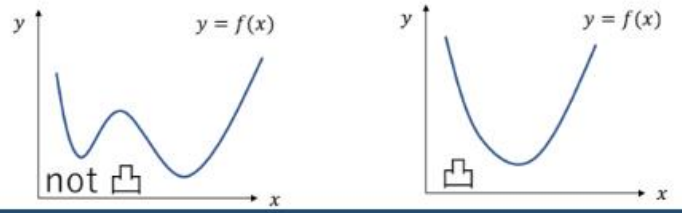
ステップB：ステップサイズの探索
降下方向にどこまで進めるか



ステップAとステップBを繰り返すことで局所最適解を見つける

（UE/FD-Primalのような）**凸計画問題**では，局所最適解が大域的最適解に一致する→解の一意性

目的関数が凸関数で，実行可能領域が凸集合であるような最適化問題



fが凸関数
⇔任意のa, bに対して，
$\lambda f(a) + (1-\lambda) f(b) \geq f(\lambda a + (1-\lambda) b)\ \forall \lambda \in [0,1]$

→関数上の任意の2点をとって線分を引いた時，その線が必ず元の関数より上に来る

2.2 UEの解法 30

2024年スタートアップゼミ#2(増田さん発表)

凸計画問題 / Convex programming problem

- 「局所解が1つしかない」問題であれば，**確実に大域解が得られる**．このような問題を**凸計画問題**と呼ぶ．
  - 利用者均衡配分は凸計画問題になる．
  - 確実に大域解が得られるだけでなく，黄金探索など効率的な解法が適用可能．
- **連続最適化では，目的関数の凸性を真っ先に確認すべし．**

準備：連続な凸関数

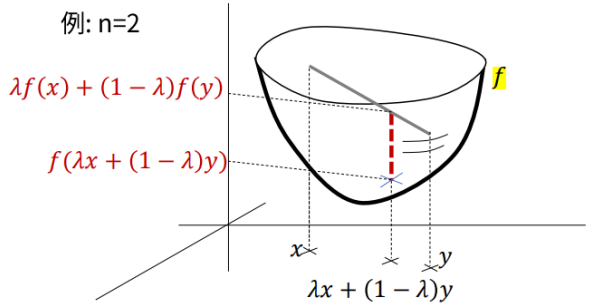
定義

$f: \mathbb{R}^n \to \overline{\mathbb{R}}$ が(連続な)**凸関数**であるとは，

N次元の実数ベクトル　　実数∩{+∞}

$$\forall x, y \in \mathbb{R}^n \ \forall \lambda \in [0,1]$$
$$\lambda f(x) + (1-\lambda) f(y) \geq f(\lambda x + (1-\lambda) y)$$

例: n=2



2023年理論談話会#15 (林発表)
http://bin.t.u-tokyo.ac.jp/rzemi23/file/15_hayashi.pdf

2023/6/7　Discrete Convex Analysis　6

非凸計画問題の数値解法 / Non-convex

- 非凸計画問題の場合，**探索をいかに効率化するか**と**局所解からいかに脱出するか**によって，無数の解法が存在する．
  - 探索の進行度に応じて探索範囲を変える Simulated Annealing (焼きなまし)
  - 複数の解の合成を繰り返して優秀なものを残す 遺伝的アルゴリズム
  - ……
- 理論的な性能保証がなく，経験的に良いとされる「ヒューリスティック」な解法がほとんど．

ハンズオン①：Nelder-Mead法

最尤推定で用いられるNelder-Mead法を体験してみよう

maximum likelihood estimation

パラメータ推定：最尤推定

尤度関数$L(\boldsymbol{\beta})$を最大化するパラメータ$\boldsymbol{\beta}$を求める

尤度関数は確率の積→値が小さくなりすぎてしまう

→**対数尤度を最大化**する問題に

$$LL(\boldsymbol{\beta}) = \sum_i P(y_i; \boldsymbol{x}_i)$$

最大化はライブラリで計算

2024年スタートアップゼミ#3(倉澤さん発表)

9 対数尤度関数の最大化

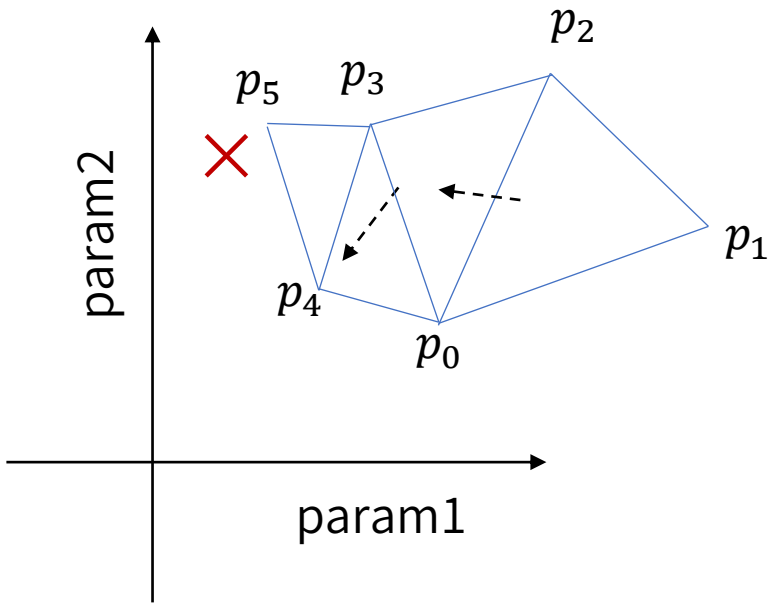
```
76  ##### 対数尤度関数 fr の最大化#####
77
78  ##パラメータ値の最適化
79  res <- optim(b0,fr, method = "Nelder-Mead", hessian = TRUE, control=list(fnscale=-1))
80
```

パラメータ"b0"に対して対数尤度関数"fr"を最大化し，その結果を"res"に出力します．

「多項ロジットモデル(MNL)の説明」
http://bin.t.u-tokyo.ac.jp/kaken/pdf/mnl.pdf

Nelder-Mead法とは

- 非凸計画問題に対して，経験的に良い解が得られる．
- 微分不能な目的関数でも適用可能．
- パラメータ空間上の多面体を少しずつ動かしていき，最適なパラメータを探しに行くイメージ．



Nelder-Mead法のアルゴリズム

Saša Singer and John Nelder (2009). Nelder-Mead algorithm

簡単のため目的関数fの最小化のみを考える．

fはm個のパラメータを含むとする．

1. $p_0 \leftarrow \left(\overbrace{0,0,\ldots,0}^{m}\right), p_i \leftarrow p_0 + e_i\ (1 \leq i \leq m)$ として$m+1$個の点を初期化する．
2. $p_0 \sim p_m$ を，$f(p)$の昇順にソートする．
3. 十分に収束するまで以下の**操作**を繰り返す．

操作

a. $m+1$個の点の重心$p_g \leftarrow \sum p_i/(m+1)$を求める．

b. 反射点$p_r \leftarrow p_g + (p_g - p_m)$ を求める．

c. $f(p_r) < f(p_0)$ なら**膨張操作**，$f(p_m) < f(p_r)$なら**収縮操作**，それ以外なら**反射操作**を行う．

d. 新たな$p_0 \sim p_m$ を，$f(p)$の昇順にソートする．

膨張操作

i. 膨張点$p_e \leftarrow p_g + 2(p_r - p_g)$ を求める．

ii. $f(p_e) < f(p_r)$ならば$p_m \leftarrow p_e$，そうでないならば$p_m \leftarrow p_r$ を代入する．

収縮操作

i. 収縮点$p_c \leftarrow p_g + 0.5(p_m - p_g)$ を求める．

ii. $f(p_c) < f(p_r)$ならば$p_m \leftarrow p_c$，そうでないならば$p_i \leftarrow p_i - 0.5(p_i - p_0)\ (0 \leq i \leq m)$ を代入する．

反射操作

i. $p_m \leftarrow p_r$を代入する．

Nelder-Mead method

Saša Singer and John Nelder (2009). Nelder-Mead algorithm

For simplicity, we consider only the minimization of the objective function f. Let f contain m parameters.

1. Initialize $m+1$ points: $p_0 \leftarrow \left(\overbrace{0,0,\ldots,0}^{m}\right)$, $p_i \leftarrow p_0 + e_i$ $(1 \le i \le m)$.
2. Sort $p_0 \sim p_m$ in ascending order by $f(p)$.
3. Repeat the following **operations** until a sufficiently convergent solution is obtained.

Operation

a. Find the center of gravity $p_g \leftarrow \sum p_i/(m+1)$.

b. Find the reflected point $p_r \leftarrow p_g + (p_g - p_m)$.

c. If $f(p_r) < f(p_0)$, do **expansion operation**; if $f(p_m) < f(p_r)$, do **contraction operation**; otherwise, do **reflection operation**.

d. Sort updated $p_0 \sim p_m$ in ascending order by $f(p)$.

Expansion operation

i. Find the expanded point $p_e \leftarrow p_g + 2(p_r - p_g)$.

ii. If $f(p_e) < f(p_r)$, substitute $p_m \leftarrow p_e$, otherwise substitute $p_m \leftarrow p_r$.

Contraction operation

i. Find the contracted point $p_c \leftarrow p_g + 0.5(p_m - p_g)$.

ii. If $f(p_c) < f(p_r)$, substitute $p_m \leftarrow p_c$, otherwise substitute $p_i \leftarrow p_i - 0.5(p_i - p_0)$ $(0 \le i \le m)$.

Reflection operation

i. Substitute $p_m \leftarrow p_r$.

人力で最尤推定をやってみよう / Let's try!

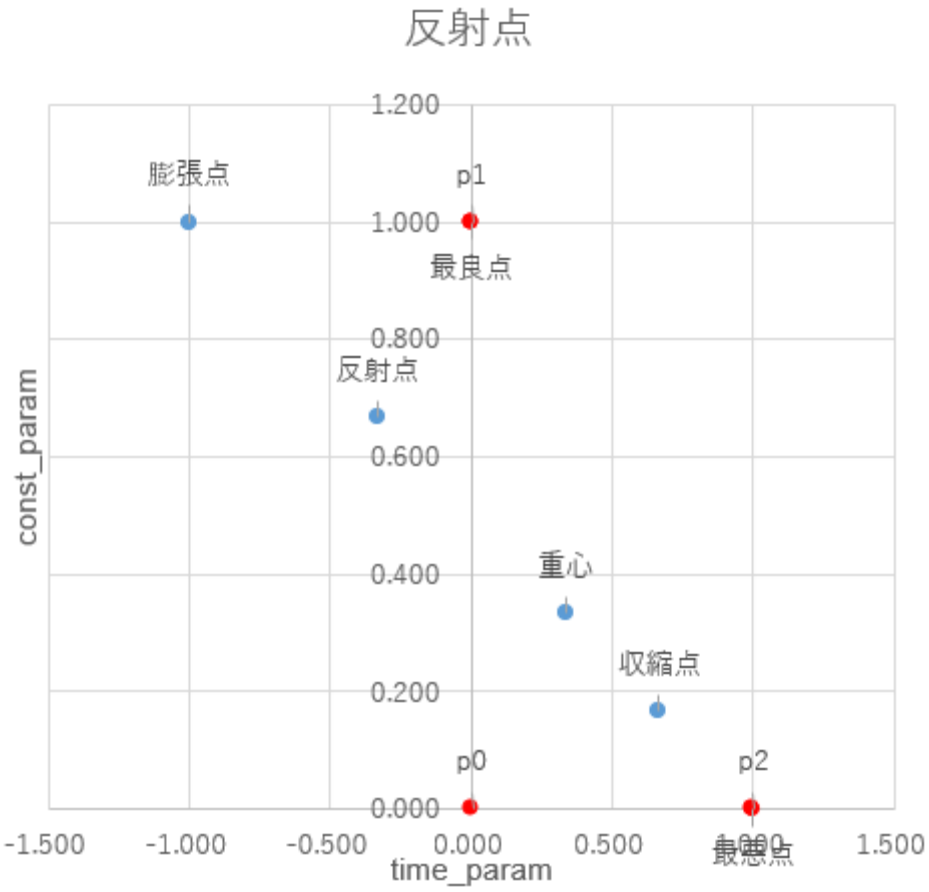
Excelシートを渡します．

尤度計算

time_param	const_param	対数尤度（符号反転）
0.000	0.000	69.315

↑をできるだけ小さく！

```
V_car = time_car * time_param / 100
V_train = time_train * time_param / 100 + const_param
train_prob = np.exp(v_train) / (np.exp(v_car) + np.exp(v_train))
```



人力で最尤推定をやってみよう

「54.0以下」のスコアを出してみてください / Let's aim for a score of 54.0 or lower.

※対数尤度に換算すると-54.0以上.

- 右クリック→**値のみ貼り付け** を使ってください.
  - use Paste Value Only
- scipyのNelder-Mead法では-53.37まで行きましたが，Excelだとexp関数の限界に達するのでそこまでいけないと思います.

Nelder-Mead法

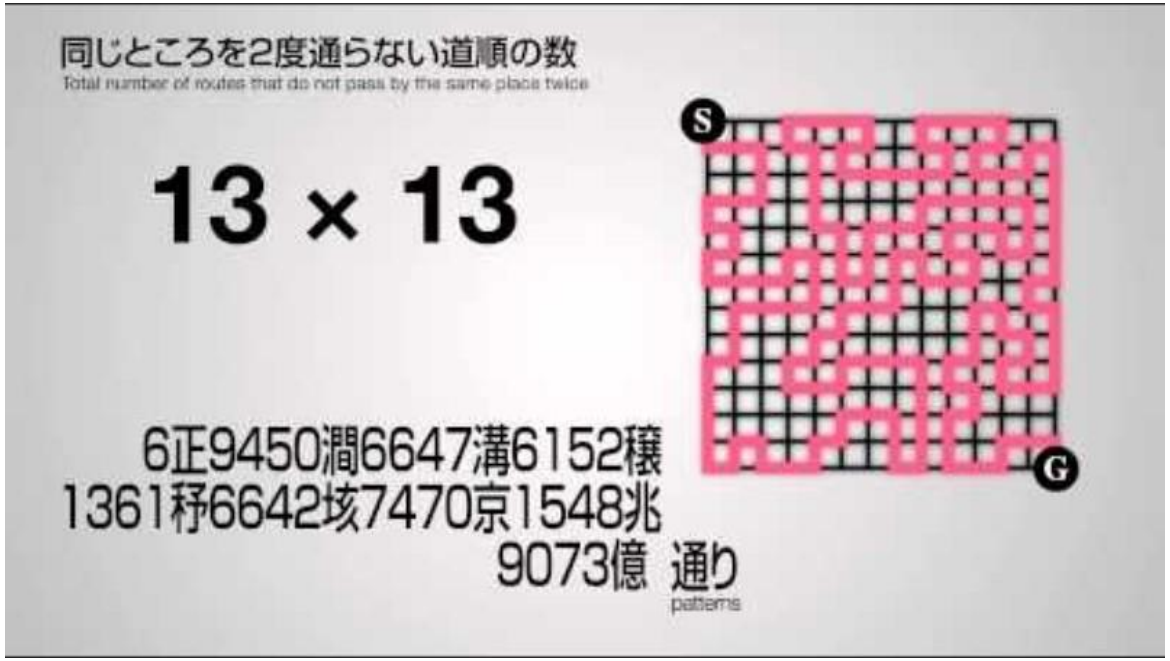
- 様々な連続最適化問題に対して良い結果を示すことが経験的に知られている
- 目的関数が凸の場合も含め，最適解に収束しない場合がある [McKinnon, 1998]

組合せ最適化（離散最適化）

Combinatorial optimization

組合せ最適化 / Combinatorial optimization

- 目的関数$f(x)$を最小化する $\hat{x}$ を求めたい
- xは離散量なので，「その気になれば」 xの候補を全列挙できるが……
- **効率的なアルゴリズムの設計とその評価が重要**



「フカシギの数え方」 同じところを2度通らない道順の数 – YouTube
https://www.youtube.com/watch?v=ge8vy4tc_kQ

計算量 / Time complexity

- 組合せ最適化のアルゴリズムには，その計算時間を理論的に見積もれるものが多い
- アルゴリズムの処理コスト「**時間計算量**」を見積もることで，CPUの性能に依らずアルゴリズムの性能を評価できる．
- 時間計算量は入力サイズの関数として表される．
  - 入力サイズ: グラフの頂点数, 辺数, 車両の数, など

$n \times m$ 行列 A の最小要素を求める

```
def minimum(n, m, A):
  ans = 0
  for i in range(n):
    for j in range(m):
      if A[i][j] < ans:
        ans = A[i][j]
  return ans
```

このアルゴリズムの時間計算量は
nm に比例する
The time complexity of this algorithm is proportional to

ソート済の長さnの数列Pから，x以上の最小の値を見つける

```
def bisect(n, P, x):
  l = 0
  r = n - 1
  while (r – l) > 1:
    m = (r + l) // 2
    if P[m] < x:
      l = m
    else:
      r = m
  return P[r]
```

$r - l$ の値が毎回必ず半分になるので $\log_2 n$ 回で終了する

このアルゴリズムの時間計算量は
$\log n$ に比例する
The time complexity of this algorithm is proportional to

オーダー記法 / Landau symbol

- 「比例する」という代わりに，**オーダー記法**を用いる．

$n \times m$ 行列 A の最小要素を求める

```
def minimum(n, m, A):
  ans = 0
  for i in range(n):
    for j in range(m):
      if A[i][j] < ans:
        ans = A[i][j]
  return ans
```

このアルゴリズムの時間計算量オーダーは$O(nm)$

ソート済の長さnの数列Pから，x以上の最小の値を見つける

```
def bisect(n, P, x):
  l = 0
  r = n - 1
  while (r – l) > 1:
    m = (r + l) // 2
    if P[m] < x:
      l = m
    else:
      r = m
  return P[r]
```

$r - l$ の値が毎回必ず半分になるので $\log_2 n$ 回で終了する

このアルゴリズムの時間計算量オーダーは$O(\log n)$

オーダー記法の厳密な運用

- O (big-O)
  - 「最悪の場合でも計算量はこれより良い」という評価に用いる
  - $f(n) \in O(g(n))$ であるとは，$\forall n_0\ \exists c\ \forall n \geq n_0\ 0 \leq f(n) \leq cg(n)$
- Ω (big-Omega)
  - 「計算量はこれより悪い」という評価に用いる
  - $f(n) \in \Omega(g(n))$ であるとは，$\forall n_0\ \exists c\ \forall n \geq n_0\ 0 \leq cg(n) \leq f(n)$
- Θ (big-Theta)
  - $f(n) \in \Theta(g(n))$ であるとは，$f(n) \in O\big(g(n)\big) \cap \Omega\big(g(n)\big)$

計算量 $f(n)$	O	Ω	Θ
$n^2 + 5n + 1$	$f(n) \in O(n^2)$	$f(n) \in \Omega(n^2)$	$f(n) \in \Theta(n^2)$
n	$f(n) \in O(n^2)$	$f(n) \notin \Omega(n^2)$	$f(n) \notin \Theta(n^2)$
n^3	$f(n) \notin O(n^2)$	$f(n) \in \Omega(n^2)$	$f(n) \notin \Theta(n^2)$

初学者向けWeb記事などでOをΘと同一視しているものが多いが，別物なので要注意

計算量オーダーの目安

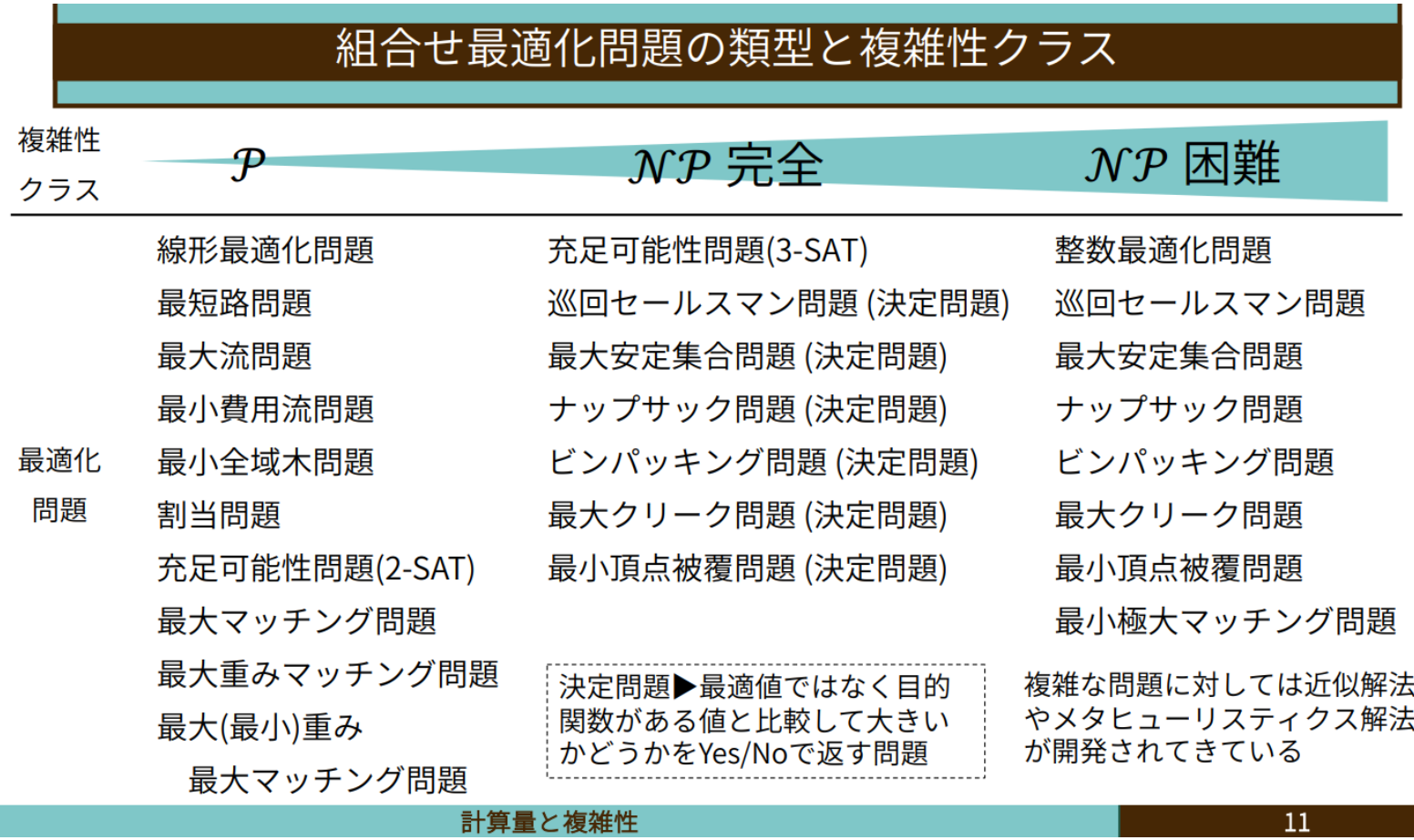
- 現代のコンピュータでは，1秒でだいたい 10^8 回くらいの四則演算ができるとされている．
  - Pythonだと遅くなって10^7回くらいになる
- 計算量オーダーを研究の計算時間見積もりに生かそう

Algorithm Time complexity	*N : Problem size that can be solved in 1 second*	
オーダー	1秒で実行できる問題サイズ N	参考例
$\Theta(N)$	10^8	線形探索
$\Theta(N \log N)$	10^7	ソート
$\Theta(N^2)$	10^4	ナップサック問題に対する動的計画法 (商品数N, 重さの総和Wで$O(NW)$)
$\Theta(N^3)$	3×10^2	行列積計算 Dinic法($G = (V, E)$の最大流計算が$O(V^2E)$)
$\Theta(2^N)$	22	bit DP

※経験上の目安です．

様々な組合せ最適化問題

- 問題ごとに有効なアルゴリズムが全く異なる



2023年スタートアップゼミ#4(黛さん発表) http://bin.t.u-tokyo.ac.jp/startup23/file/slide4_1.pdf

最短経路問題 / Shortest path problem

グラフ$G = (V, E)$で，ある頂点sから別の頂点tまでの最短経路を求める．

頂点の集合　辺の集合

定式化:

目的関数

$$\min \sum_{(i,j)\in E} c_{ij} x_{ij}$$

有向辺ijのコスト(長さ)

有向辺ijを使うか否かを表す01変数

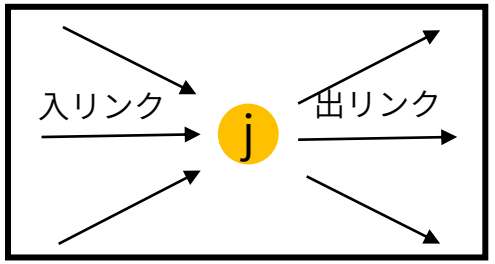
制約条件

$$s.t.$$

$$\forall j \in V \quad \sum_{k|(j,k)\in E} x_{jk} - \sum_{i|(i,j)\in E} x_{ij} = \begin{cases} 1\ (j = s) \\ -1\ (j = t) \\ 0\ (\text{otherwise}) \end{cases}$$

$$\forall (i,j) \in E\ x_{ij} \in \{0,1\}$$

出リンク数と入リンク数の差



最短経路問題 / Shortest path problem

グラフ$G = (V, E)$で，ある頂点sから別の頂点tまでの最短経路を求める．

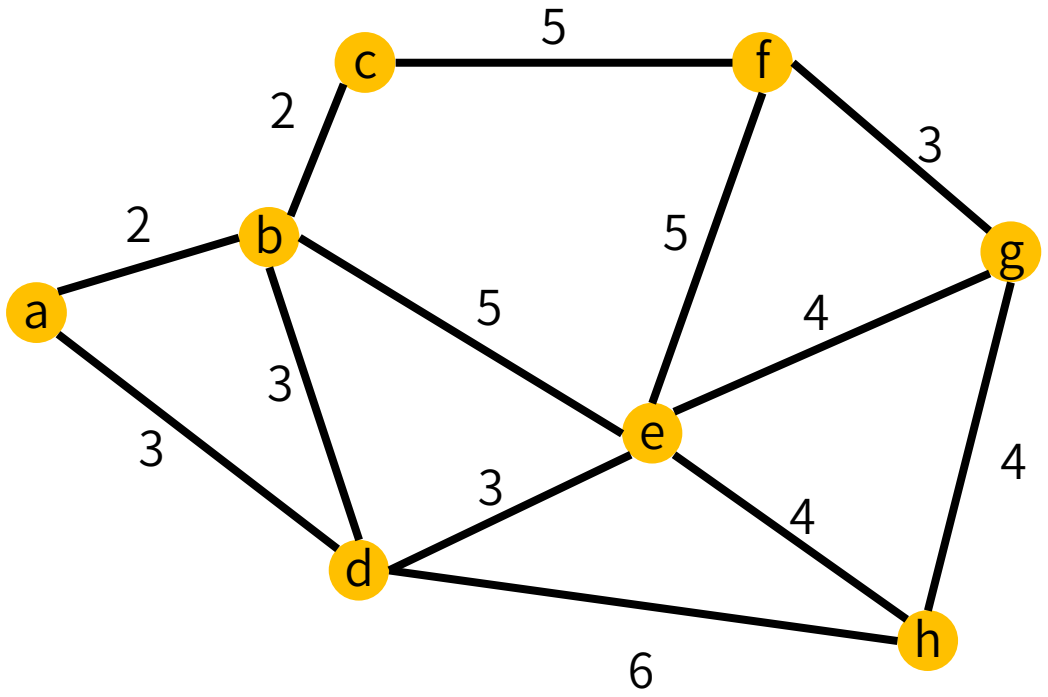
頂点の集合　辺の集合

解法:

- Dijkstra法
  - 1回のアルゴリズムで**すべての頂点までの最短路を同時に求められる．**
- A*法
  - 1始点・1終点に限る．点の地理的な位置など追加情報を必要とする．
- Bellman-ford法
  - コストが負の辺を含んでいても計算可能．

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める.



優先度付きキュー

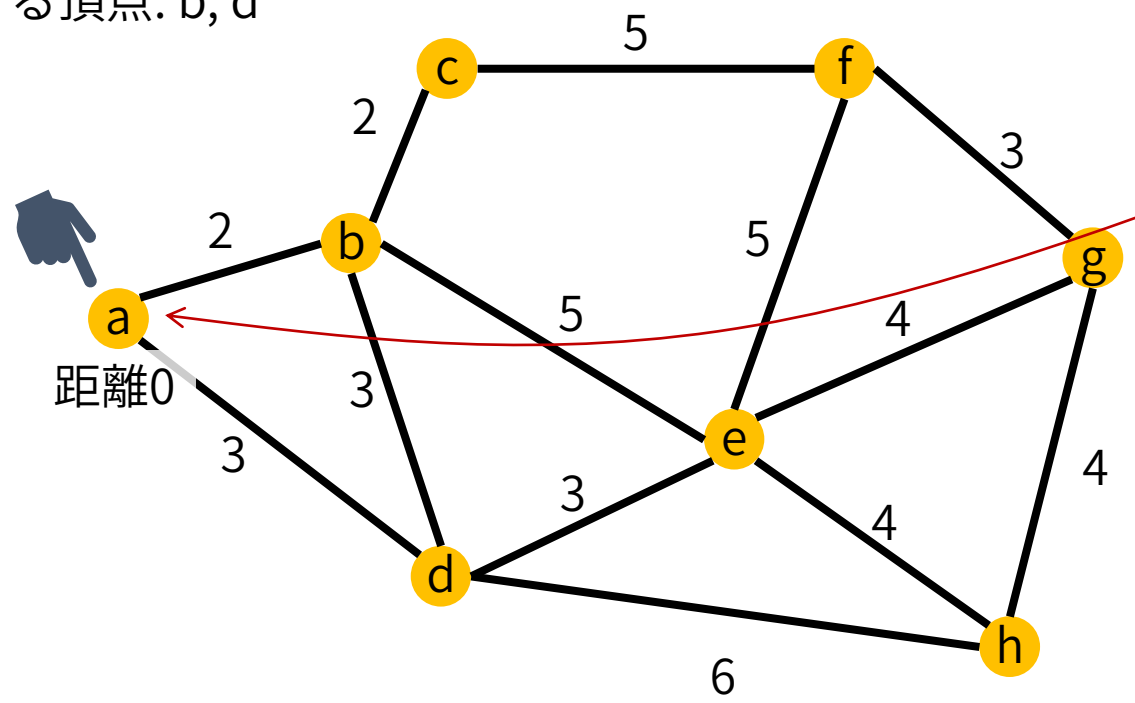
頂点	aからの距離
a	0

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
    heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

aに隣接する頂点: b, d



優先度付きキュー

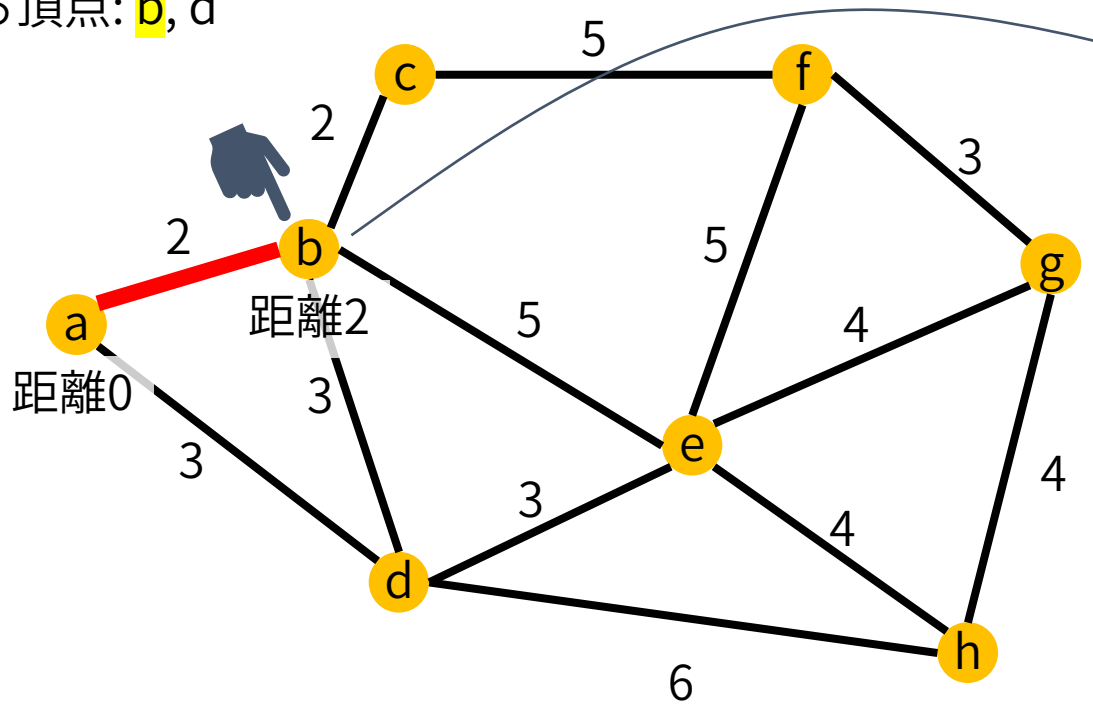
頂点	aからの距離
~~a~~	0

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
    heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

aに隣接する頂点: b, d



優先度付きキュー

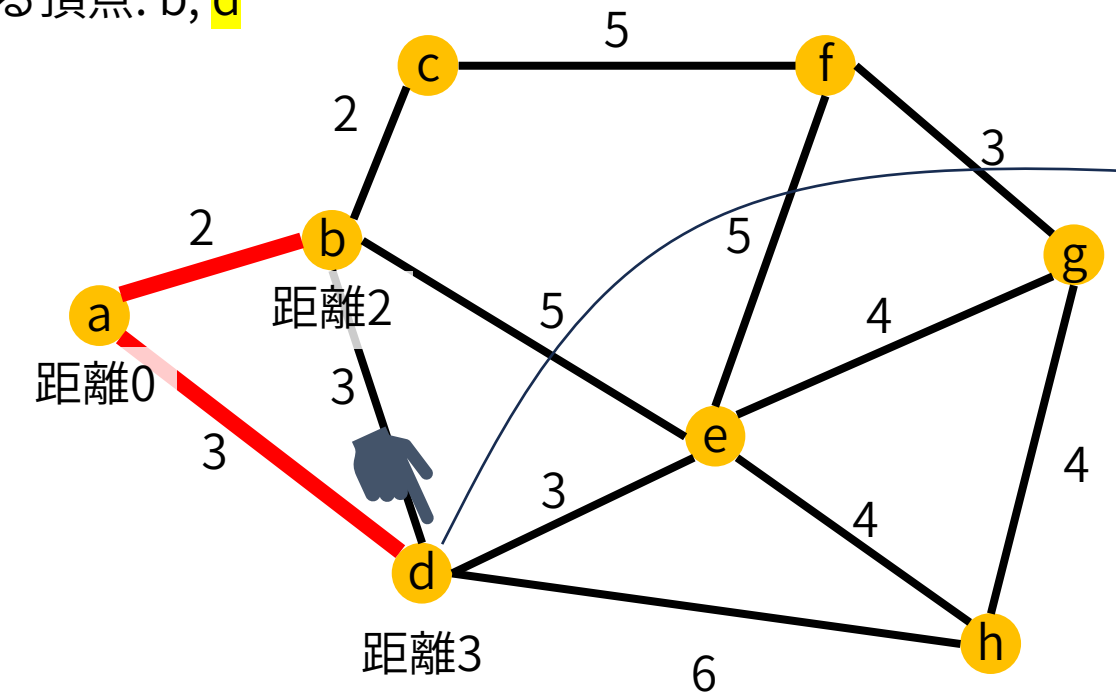
頂点	aからの距離
b	2

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
▶   heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

aに隣接する頂点: b, d



優先度付きキュー

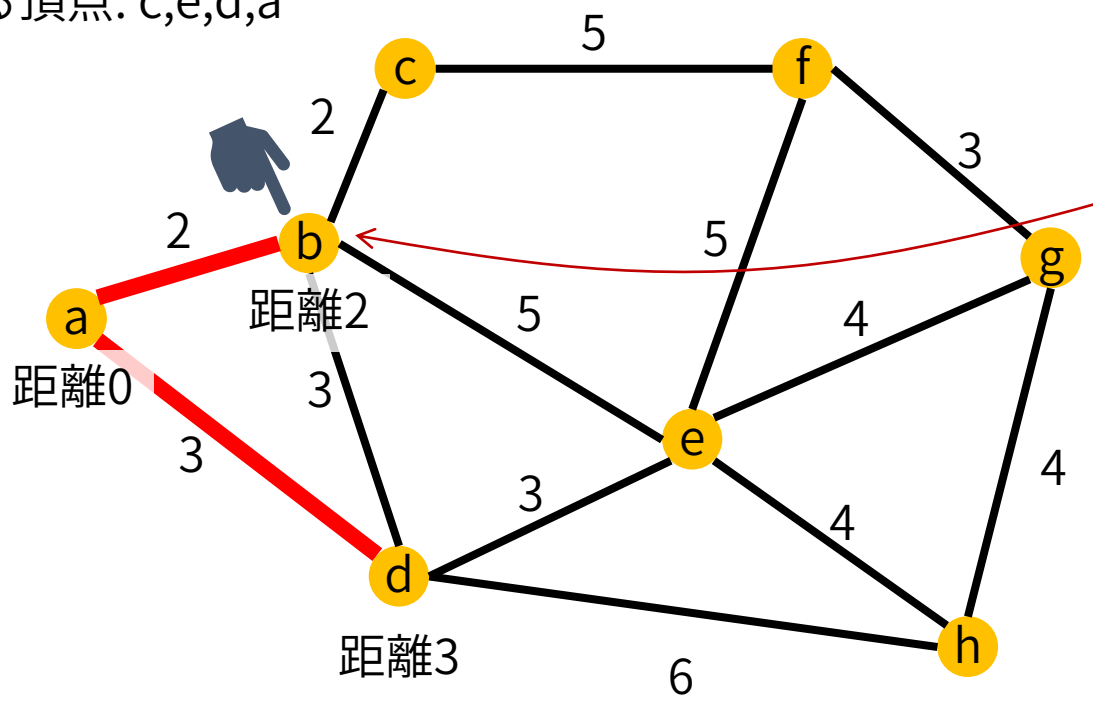
頂点	aからの距離
b	2
d	3

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
▶   heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

bに隣接する頂点: c,e,d,a



優先度付きキュー

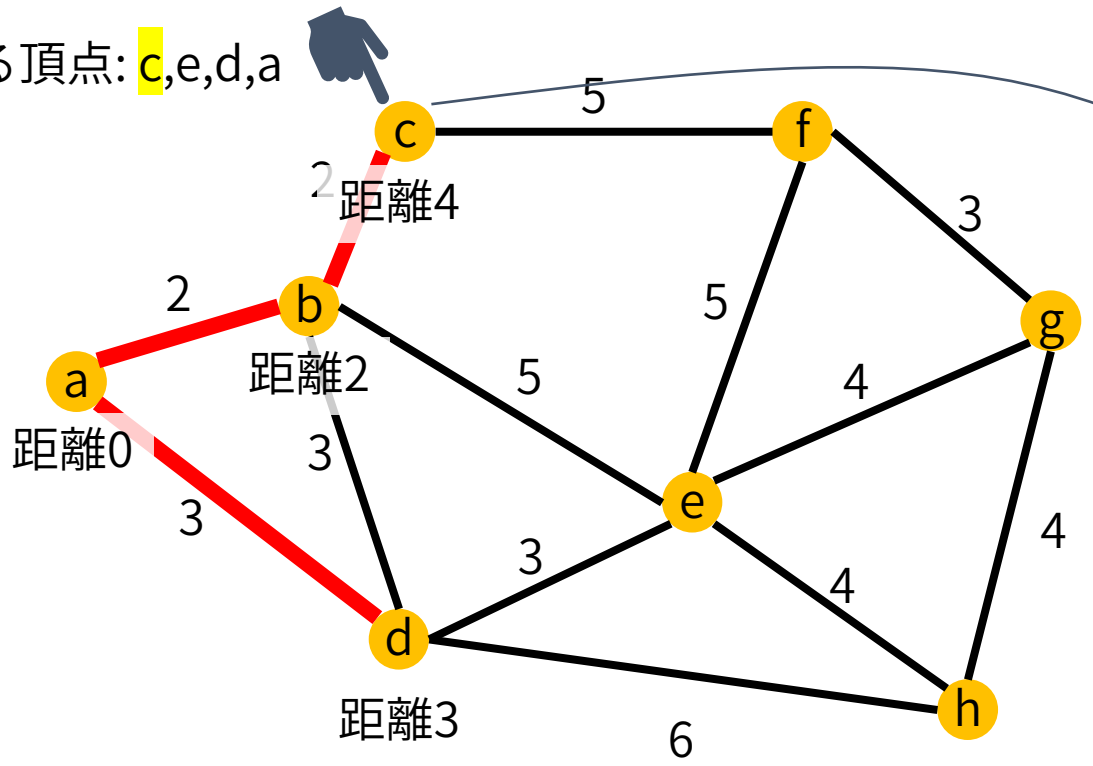
頂点	aからの距離
~~b~~	2
d	3

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
    heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

bに隣接する頂点: c,e,d,a



優先度付きキュー

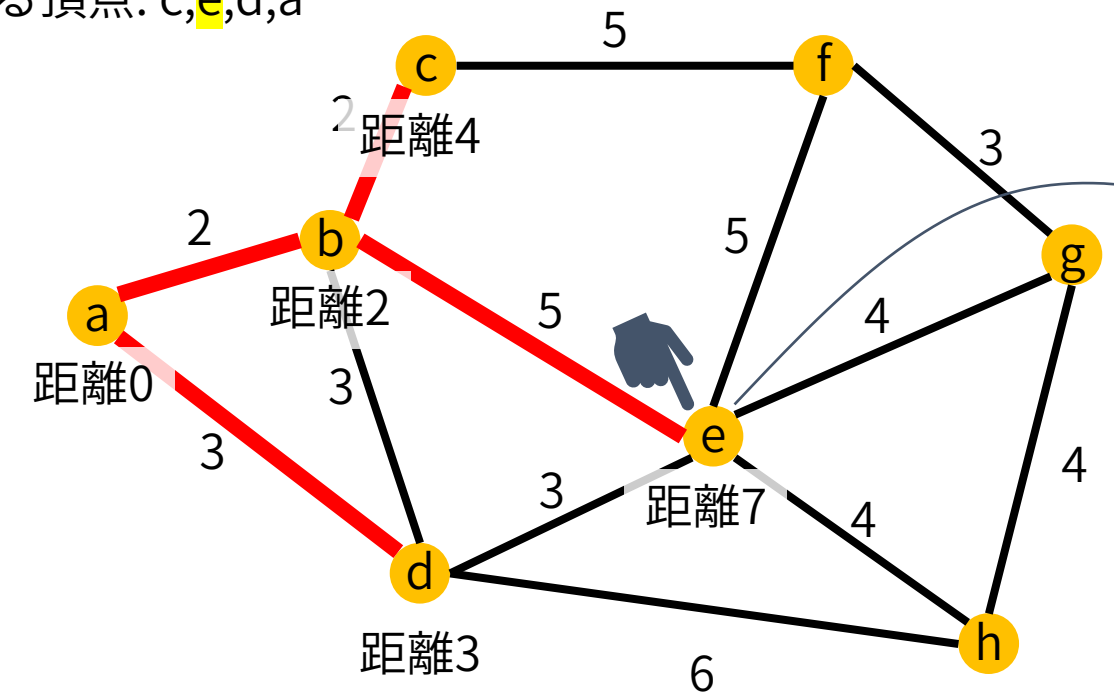
頂点	aからの距離
d	3
c	4

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
▶   heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

bに隣接する頂点: c,e,d,a



優先度付きキュー

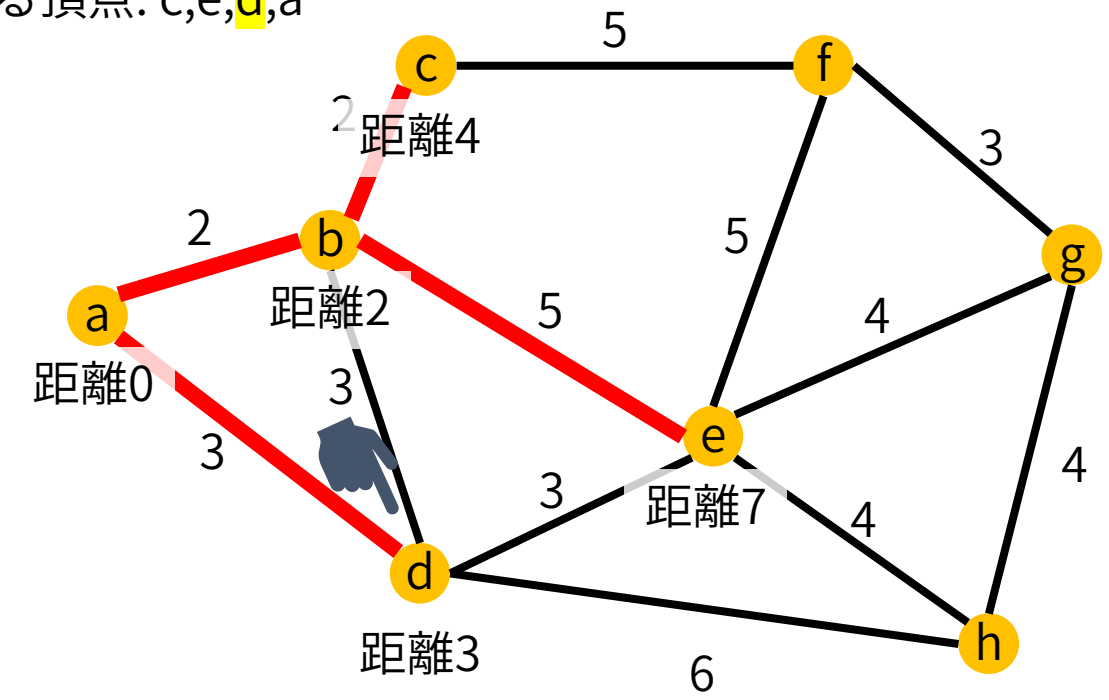
頂点	aからの距離
d	3
c	4
e	7

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
▶   heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

bに隣接する頂点: c,e,d,a



既に距離3の最短経路があるので
距離2+3=5の経路は不採用

優先度付きキュー

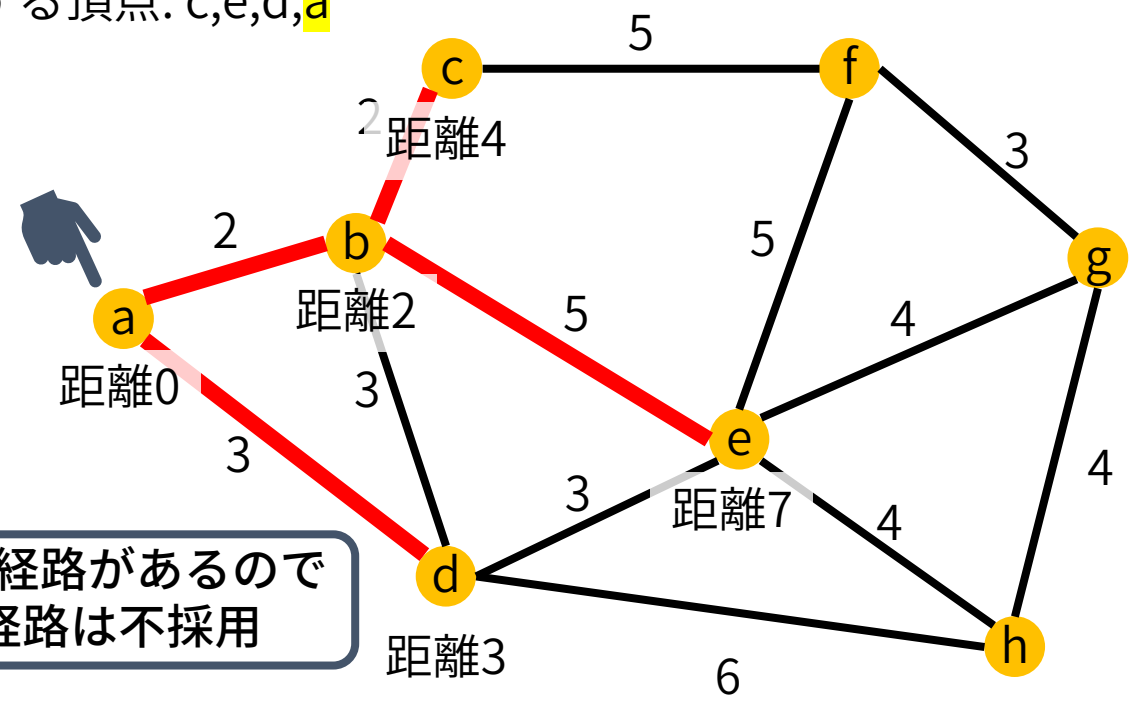
頂点	aからの距離
d	3
c	4
e	7

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
    heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

bに隣接する頂点: c,e,d,a



既に距離0の最短経路があるので
距離2+2=4の経路は不採用

人間からしたら自明だが，
プログラムは書いたとおりにしか動かない

優先度付きキュー

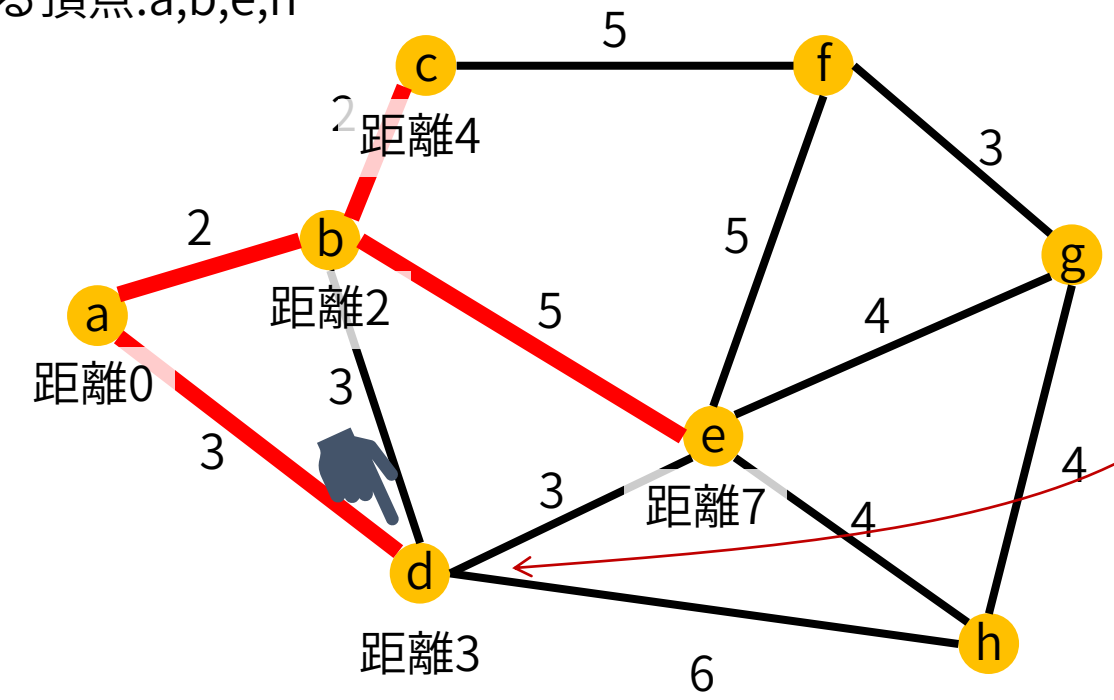
頂点	aからの距離
d	3
c	4
e	7

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
    heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

dに隣接する頂点:a,b,e,h



優先度付きキュー

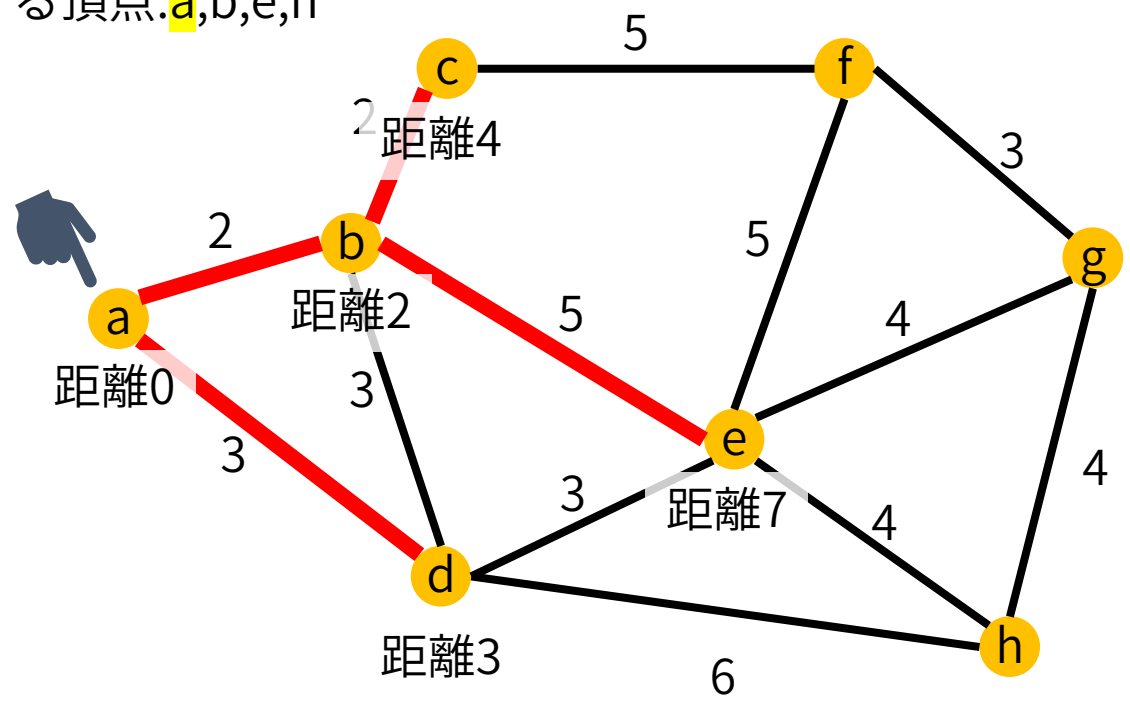
頂点	aからの距離
~~d~~	3
c	4
e	7

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
    heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

dに隣接する頂点:a,b,e,h



優先度付きキュー

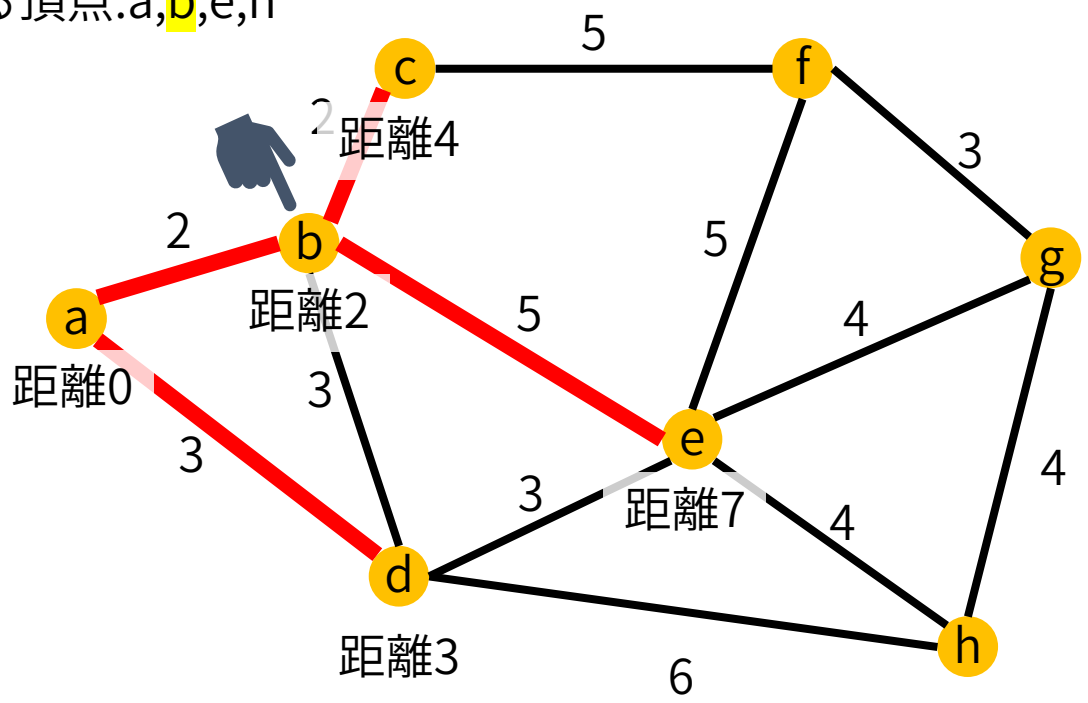
頂点	aからの距離
c	4
e	7

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
    heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

dに隣接する頂点:a,b,e,h



優先度付きキュー

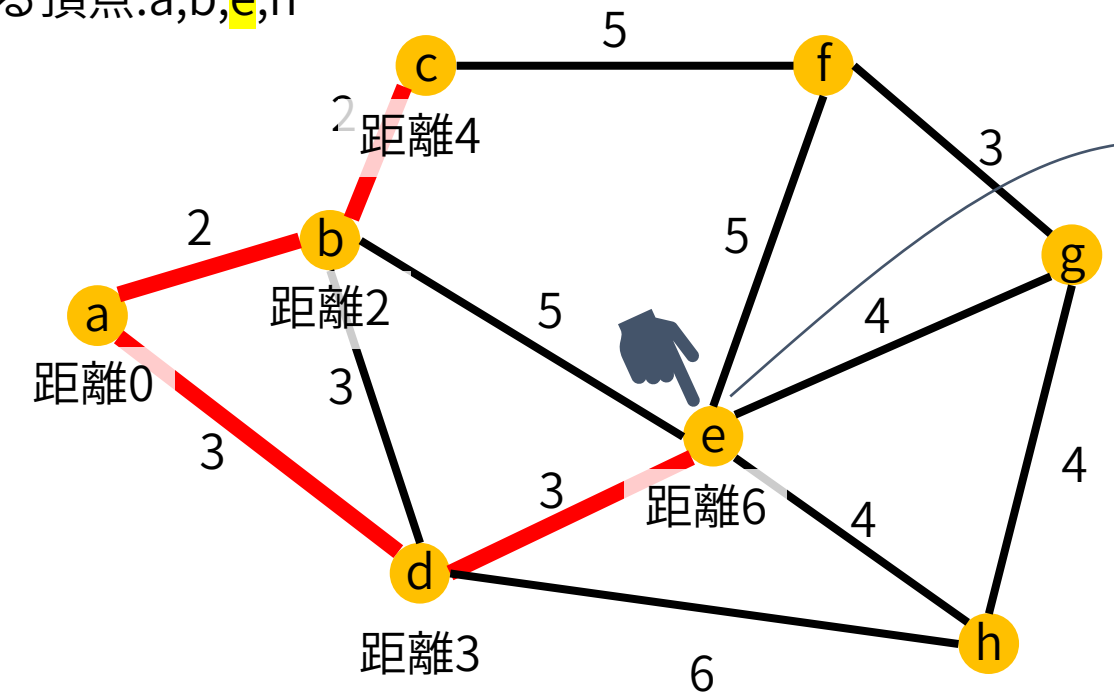
頂点	aからの距離
c	4
e	7

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
    heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

dに隣接する頂点:a,b,e,h



優先度付きキュー

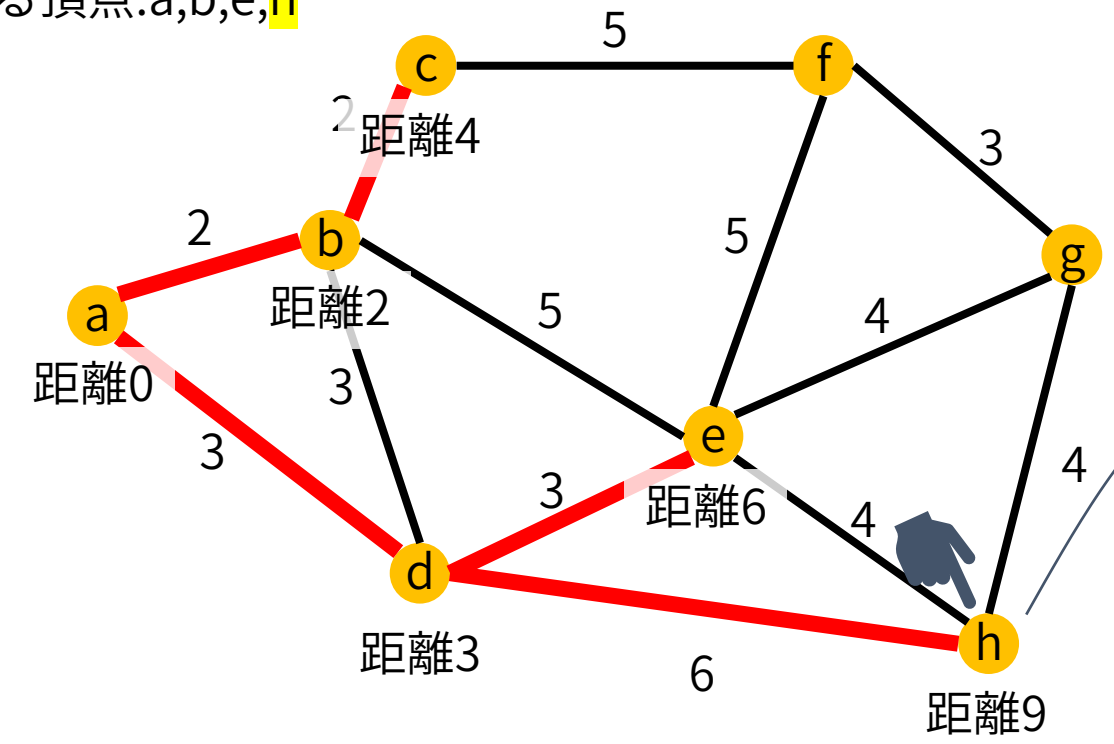
頂点	aからの距離
c	4
e	6
e	7

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
▶   heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

dに隣接する頂点:a,b,e,h



優先度付きキュー

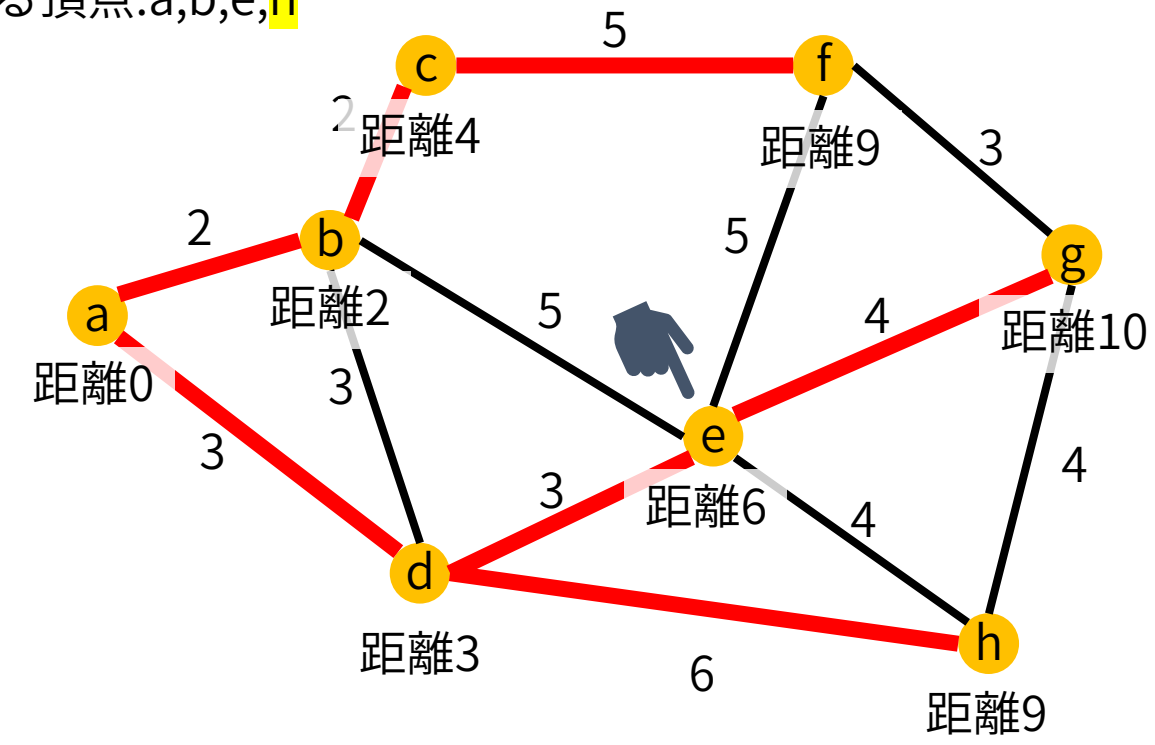
頂点	aからの距離
c	4
e	6
e	7
h	9

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
▶   heappush(q, (new_dist, next_node))
```

Dijkstra法

頂点aからほかの各頂点までの最短経路を求める．

dに隣接する頂点:a,b,e,h



優先度付きキュー

頂点	aからの距離
~~e~~	7
h	9
f	9
g	10

```
while len(q) != 0:
  tmp_dist, node = heappop(q)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
    heappush(q, (new_dist, next_node))
```

※頂点eが二重にキューに入っても二つ目以降はここで捨てられるのでOK

Dijkstra法の簡易的な計算量解析

- 優先度付きキュー(heap)へのデータの挿入と最小値の取得は，heapのサイズをNとして$\Theta(\log N)$でできる．
- heapへの操作は，高々辺の数しか実行しない．（=操作回数のオーダーが $O(E)$）
  - heapのサイズもE 個以下．

従って，全体の計算量は $O(E \log E)$ で上から抑えられる．

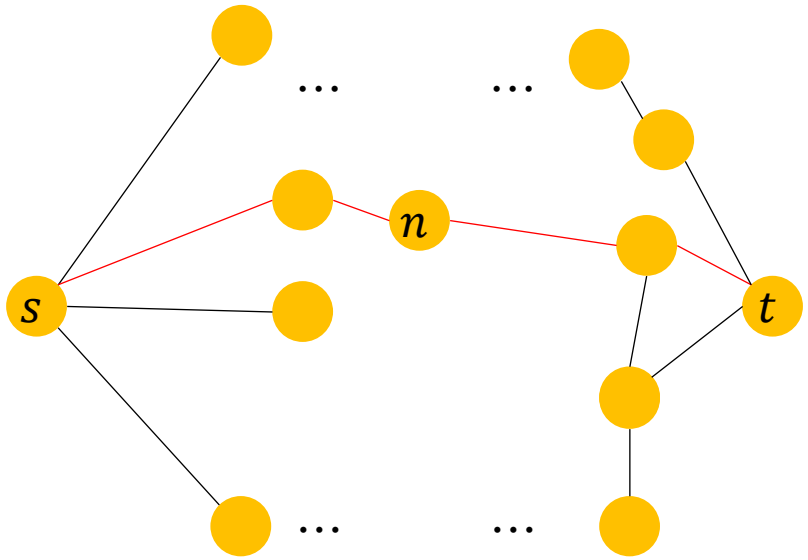
※うまく実装すると $\Theta(E \log V)$ にできるので，一般的な教科書では**Dijkstra法の計算量オーダーは**$\Theta(E \log V)$ とされる．

```
while len(q) != 0:
  tmp_dist, node = heappop(q)  ← Θ(log N)
  if dist[node] != tmp_dist:
    continue
  for next_node, cost in G[node]:
    new_dist = dist[node] + cost
    if dist[next_node] <= new_dist:
      continue
    dist[next_node] = new_dist
    heappush(q, (new_dist, next_node))  ← Θ(log N)
```

ハンズオン②：最短経路探索の応用パズル

- 工学の学生としては，**既知のアルゴリズムを応用**することで自分の研究に生かしたい．
- 以下の問題を，dijkstra法を使って$O(E \log V)$で解く方法を考えてください． [全3問]

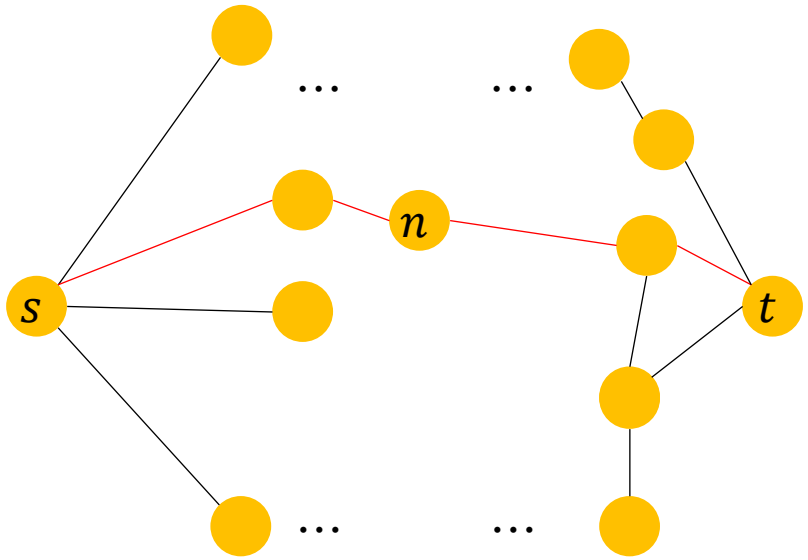
Q1. 無向グラフ$G(V, E)$において，ある頂点sから別の頂点tへの経路を考えます．経路の途中で頂点nを通る，最短の経路を見つけてください．



ハンズオン②：最短経路探索の応用パズル

- 工学の学生としては，**既知のアルゴリズムを応用**することで自分の研究に生かしたい．
- 以下の問題を，dijkstra法を使って$O(E \log V)$で解く方法を考えてください． [全3問]

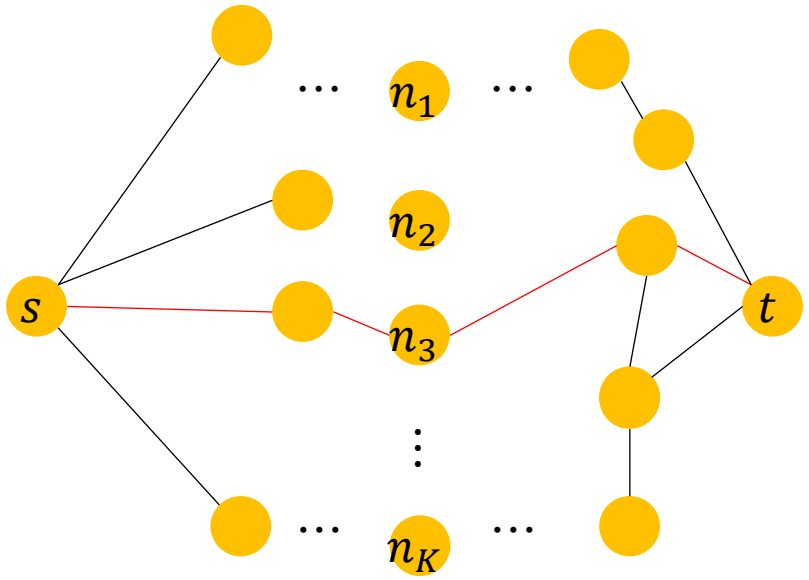
Q1. 無向グラフ$G(V, E)$において，ある頂点sから別の頂点tへの経路を考えます．経路の途中で頂点nを通る，最短の経路を見つけてください．



ハンズオン②：最短経路探索の応用パズル

- 工学の学生としては，**既知のアルゴリズムを応用**することで自分の研究に生かしたい．
- 以下の問題を，dijkstra法を使って$O(E \log V)$で解く方法を考えてください． [全3問]

Q2. 無向グラフ$G(V, E)$において，ある頂点sから別の頂点tへの経路を考えます．経路の途中で頂点$n_1, n_2, n_3, \ldots, n_K$のどれかを少なくとも1つ通る，最短の経路を見つけてください．

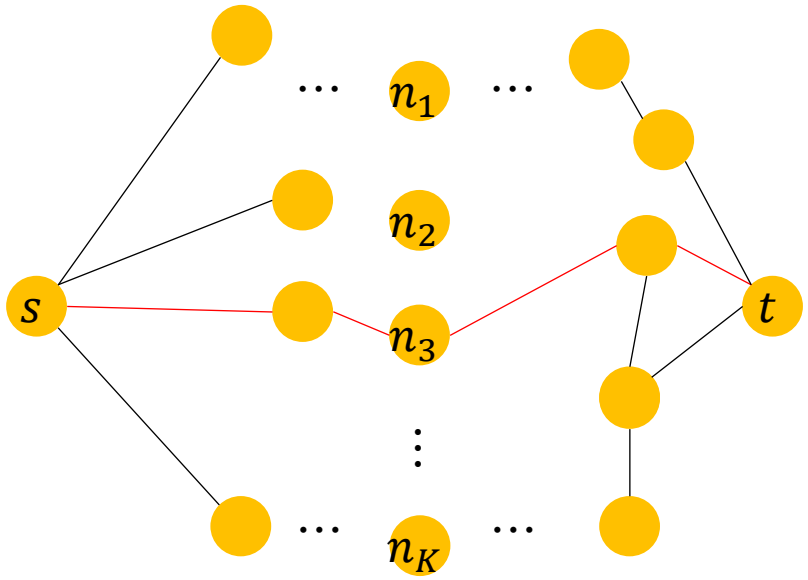


※dijkstra法を$K+1$回実行すると，計算量オーダーは$\Omega(KE \log V)$ になってしまいます．

ハンズオン②：最短経路探索の応用パズル

- 工学の学生としては，**既知のアルゴリズムを応用**することで自分の研究に生かしたい．
- 以下の問題を，dijkstra法を使って$O(E \log V)$で解く方法を考えてください． [全3問]

Q2. 無向グラフ$G(V, E)$において，ある頂点sから別の頂点tへの経路を考えます．経路の途中で頂点$n_1, n_2, n_3, \ldots, n_K$のどれかを少なくとも1つ通る，最短の経路を見つけてください．

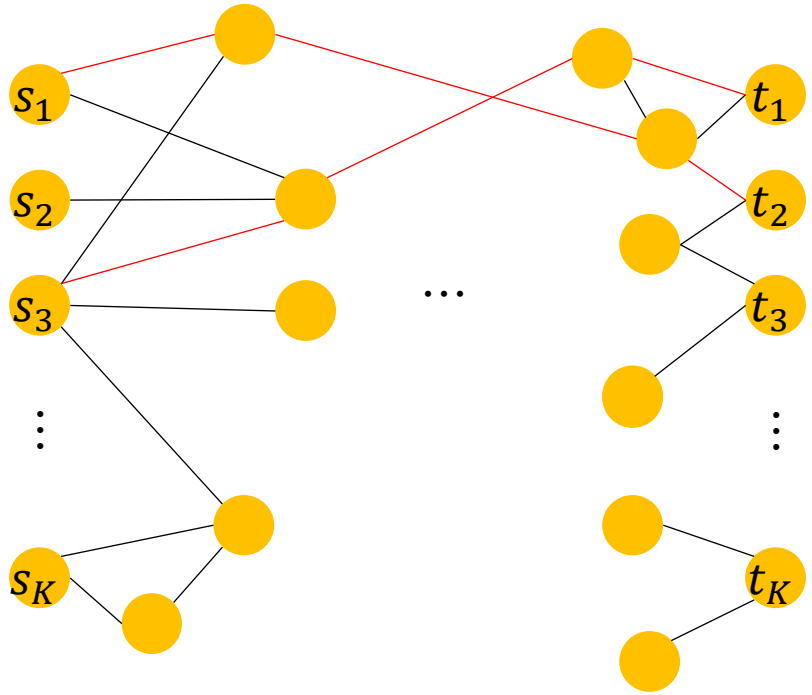


※dijkstra法を$K+1$回実行すると，計算量オーダーは$\Omega(KE \log V)$ になってしまいます．

ハンズオン②：最短経路探索の応用パズル

- 工学の学生としては，**既知のアルゴリズムを応用**することで自分の研究に生かしたい．
- 以下の問題を，dijkstra法を使って$O(E \log V)$で解く方法を考えてください． [全3問]

Q3. 無向グラフ$G(V, E)$において，複数の始点$s_1, s_2, \ldots, s_K$から複数の終点$t_1, t_2, \ldots, t_K$までのそれぞれの最短経路を考えます．これらの最短経路の中から，全体でK^2本ある最短経路の中で最も短いものを1つ見つけてください．

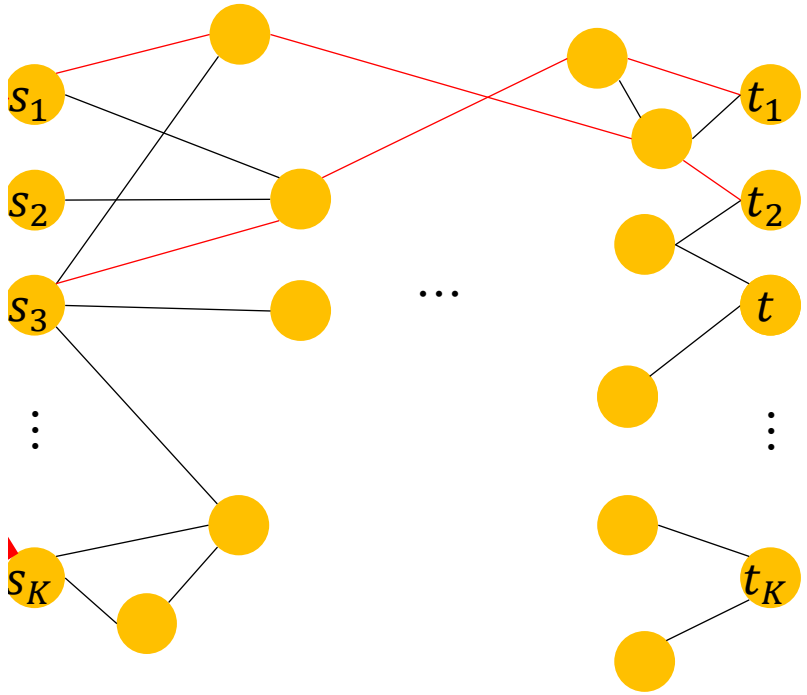


※dijkstra法をK^2回実行すると，計算量オーダーは$\Omega(K^2 E \log V)$になってしまいます．

ハンズオン②：最短経路探索の応用パズル

- 工学の学生としては，**既知のアルゴリズムを応用**することで自分の研究に生かしたい．
- 以下の問題を，dijkstra法を使って$O(E \log V)$で解く方法を考えてください． [全3問]

Q3. 無向グラフ$G(V, E)$において，複数の始点$s_1, s_2, \ldots, s_K$から複数の終点$t_1, t_2, \ldots, t_K$までのそれぞれの最短経路を考えます．これらの最短経路の中から，全体でK^2本ある最短経路の中で最も短いものを1つ見つけてください．

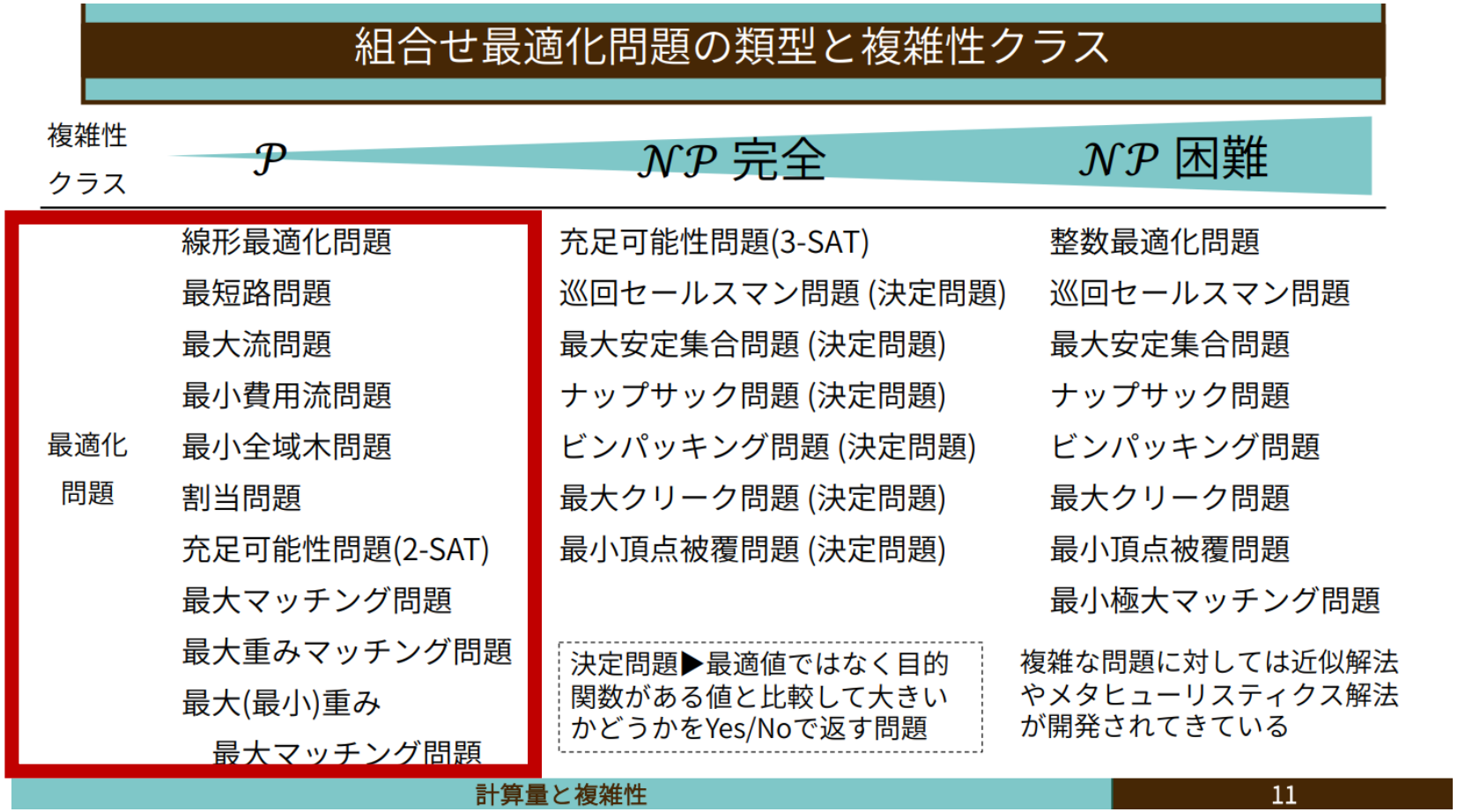


※dijkstra法をK^2回実行すると，計算量オーダーは$\Omega(K^2 E \log V)$になってしまいます．

発展：解きやすい組合せ最適化問題の背景

- (復習) 連続最適化では「凸計画問題」ならば解きやすい
- 組合せ最適化問題では→「離散凸」性(Murota, 1998)を満たす問題ならば解きやすい

discrete convex



2023年スタートアップゼミ#4(黛さん発表) http://bin.t.u-tokyo.ac.jp/startup23/file/slide4_1.pdf

L♮凸関数とM♮凸関数

- 定義域が格子点$\mathbb{Z}^n$上となるような凸関数「離散凸関数」を考える．

discrete convex function

定義 (Murota, 1998)
$g: \mathbb{Z}^n \to \mathbb{R}$が**L♮凸関数**であるとは，

$$\forall p, q \in \mathbb{Z}^n \; g(p) + g(q) \geq g\left(\left\lceil \frac{p+q}{2} \right\rceil\right) + g\left(\left\lfloor \frac{p+q}{2} \right\rfloor\right)$$

定義域が整数になるように
切り上げ/切り捨て

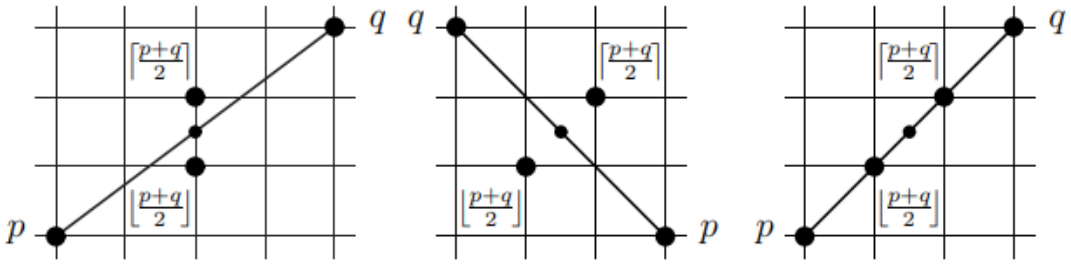


Figure 1: Discrete midpoint convexity

定義 (Murota, 1998)
$f: \mathbb{Z}^n \to \mathbb{R}$が**M♮凸関数**であるとは，

$$\forall x, y \in \mathbb{Z}^n \; \forall i \in \mathrm{supp}^+(x-y)$$
$$\exists j \in \mathrm{supp}^-(x-y) \cup \{0\}$$
$$s.t.$$
$$f(x) + f(y) \geq f\left(x - \chi_i + \chi_j\right) + f\left(y + \chi_i - \chi_j\right)$$

ただし $\mathrm{supp}^+(z) = \{i | z_i > 0\}, \mathrm{supp}^-(z) = \{i | z_i < 0\}$,
χ は基本ベクトル [0,0,1,0,0,0]のようなベクトル
特例として$\chi_0 = \mathbf{0}$

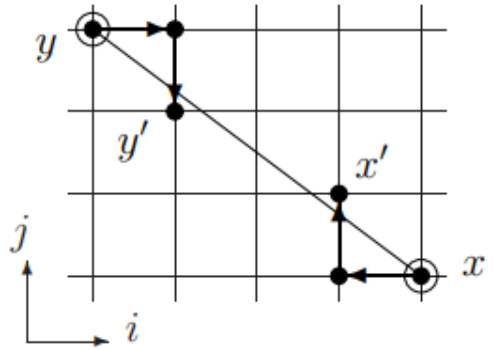


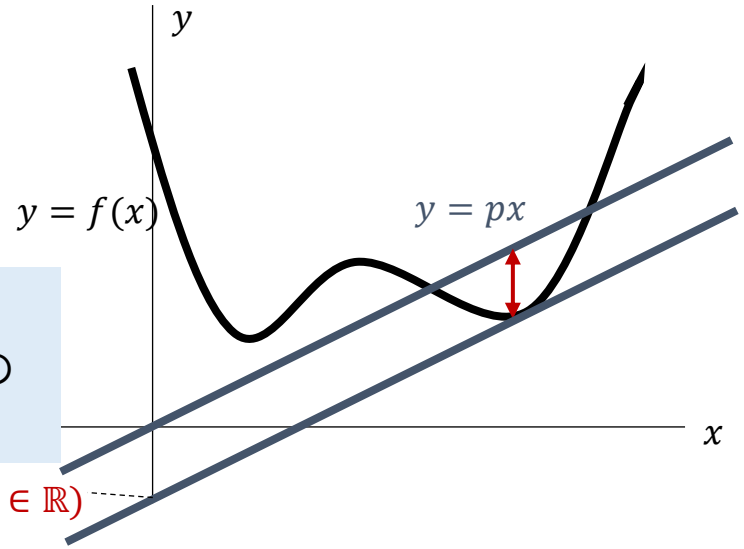
Figure 3: Nearer pair in the definition of M♮-convex functions

Legendre-Fenchel 変換

定義: Legendre-Fenchel変換 *convex conjugation*

$f: \mathbb{Z}^n \to \mathbb{R}$ (凸とは限らない)の**凸共役** $f^{\cdot}: \mathbb{R}^n \to \mathbb{R}$とは，

$$f^{\cdot}(p) = \sup(\langle p, x\rangle - f(x) | x \in \mathbb{Z}^n)$$



$f^{\cdot}(p)$は

「fに下側で接する傾きpの直線のy切片」に対応

離散凸関数の共役定理

- **適切なL$^\natural$凸関数**$g: \mathbb{Z}^n \to \mathbb{Z}$にLegendre-Fenchel 変換を1回行うと**M$^\natural$凸関数** $f: \mathbb{Z}^n \to \mathbb{Z}$になる．
- **適切なM$^\natural$凸関数**$f: \mathbb{Z}^n \to \mathbb{Z}$にLegendre-Fenchel 変換を1回行うと**L$^\natural$凸関数** $g: \mathbb{Z}^n \to \mathbb{Z}$になる．
- fの値域が$\mathbb{Z}$であれば，Legendre-Fenchel 変換を2回行うとfに戻る．

例1) 最短経路問題 / Shortest path problem

有向グラフ $G(V,E)$，リンクコスト $c(e)$ が与えられたとき，頂点sから全頂点への最短路を同時に求める最短路問題とは以下の通り．

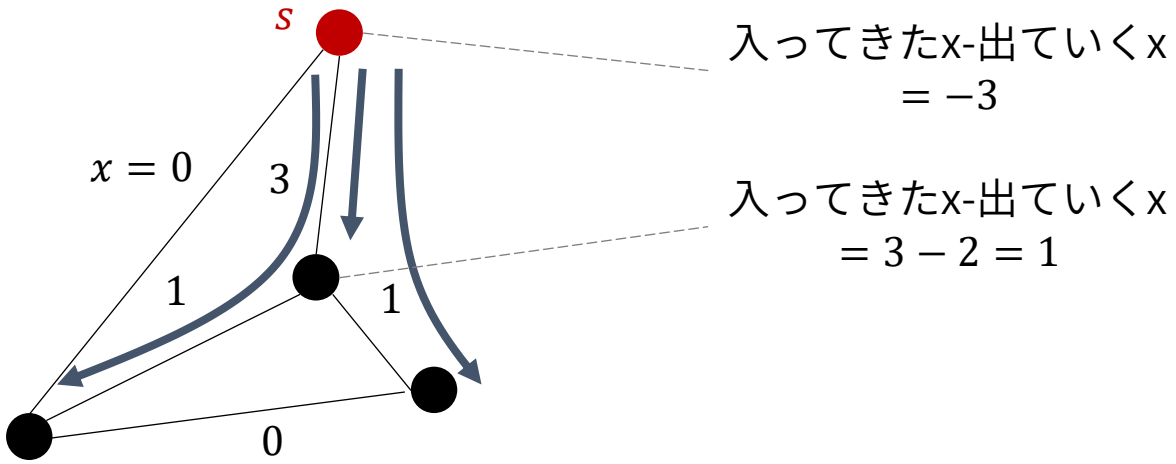
$$\min \sum_{e \in E} c(e)x(e)$$

$$s.t.\ \forall v \in V$$

$$\sum_{(u,v) \in E, u \in V} x(u,v) - x(v,u) = \begin{cases} -(n-1) & (v = s) \\ 1 & (otherwise) \end{cases}$$

x(e): リンクeを何回通るか

※全頂点への最短路を求めるにもかかわらず目的関数はひとつ！



最短経路を求めるアルゴリズムはダイクストラ法により$\Theta(E \log V)$が達成されており，高速に解ける．

例1) 最短経路問題 / Shortest path problem

有向グラフ $G(V,E)$，リンクコスト $c(e)$ が与えられたとき，頂点sから全頂点への最短路を同時に求める．

前の定式化のFenchel双対をとると，

$p: \mathbb{Z}^V \to \mathbb{Z}$の関数を決定する問題

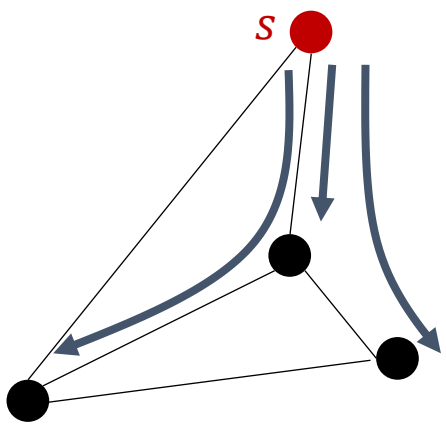
$$\max \sum_{v \in V \setminus \{s\}} p(v) - \cancel{p(s)}$$

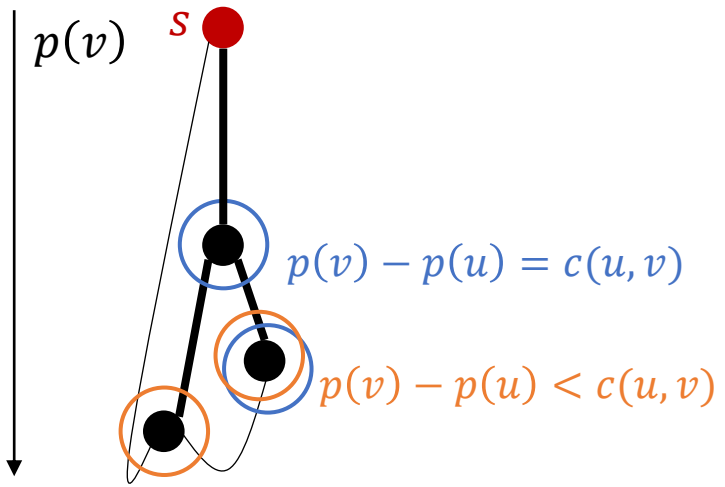
※$p(s) = 0$ と固定してよい

$$s.t.$$

$$\forall (u,v) \in E \quad p(v) - p(u) \leq c(u,v)$$

制約条件あり





紐という制約条件付きで
重力に従って垂らす(=距離最大化)
と解釈できる

例1) 最短経路問題 / Shortest path problem

有向グラフ $G(V,E)$，リンクコスト $c(e)$ が与えられたとき，頂点sから全頂点への最短路を同時に求める．

前の定式化のFenchel双対をとると，

$p: \mathbb{Z}^V \to \mathbb{Z}$の関数を決定する問題

$$\max \sum_{v \in V \setminus \{s\}} p(v) - \cancel{p(s)}$$

※$p(s) = 0$ と固定してよい

$$s.t.$$

$$\forall (u,v) \in E \;\; p(v) - p(u) \leq c(u,v)$$

制約条件あり

$g: \mathbb{Z}^V \to \underline{\mathbb{Z}}$を

$$g(x) = \begin{cases} \sum_{v \in V \setminus \{s\}} p(v) \;\; (p \in S) \\ -\infty \;\; (\text{otherwise}) \end{cases}$$

とする．gの実行可能領域は

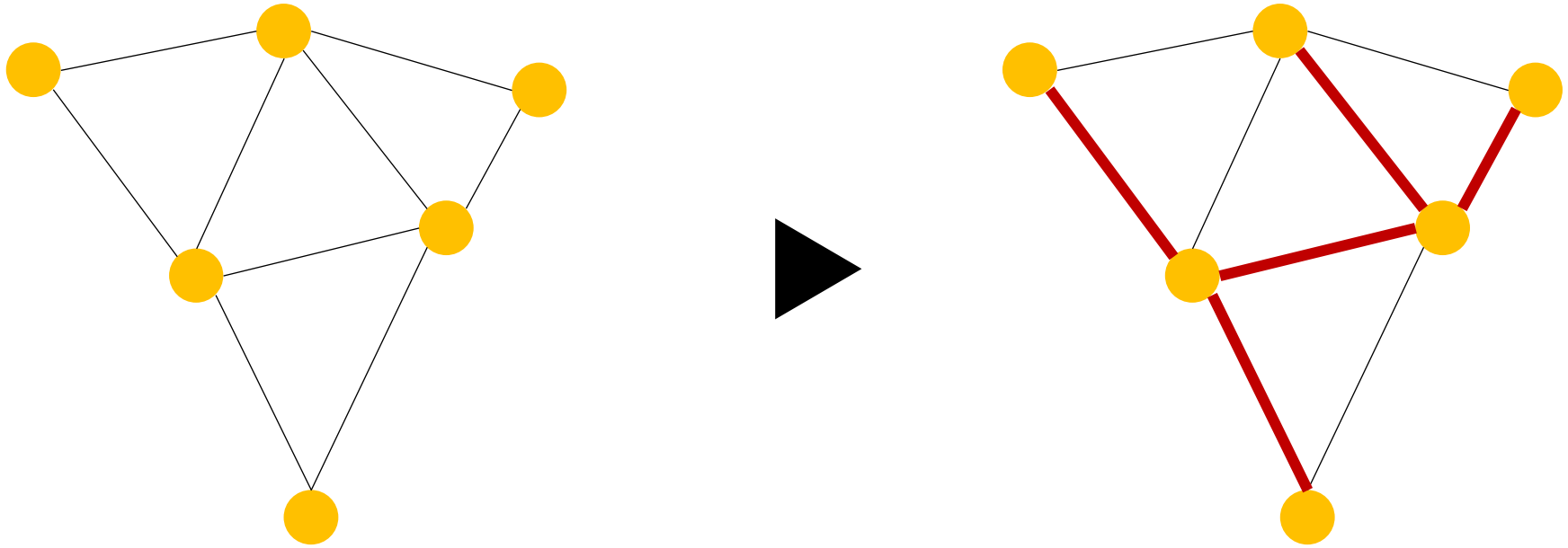
$$S = \{p \in \mathbb{Z}^n \mid p(v) - p(u) \leq c(u,v) \;\; (\forall (u,v) \in E)\}$$

Sは**L$^\natural$凹集合**，gは**L$^\natural$凹関数**なので，

双対問題は**L$^\natural$凹集合上のL$^\natural$凹関数最大化**となり，効率的に解ける

例2) 最小全域木問題 / Minimum spanning tree problem

無向連結グラフ $G(V,E)$，リンクコスト $c(e)$ が与えられたとき，総コストを最小化するように全域木（すべての頂点を含む木構造）をつくる．



グラフG

Gの全域木

最小全域木を求めるアルゴリズムはプリム法により$\Theta(E \log V)$，クラスカル法により$\Theta(E\ \alpha(V))$ が達成されており，高速に解ける．

例2) 最小全域木問題 / Minimum spanning tree problem

無向連結グラフ $G(V,E)$，リンクコスト $c(e)$ が与えられたとき，総コストを最小化するように全域木（すべての頂点を含む木構造）をつくる．

集合 $S \subseteq \{0,1\}^E$ を

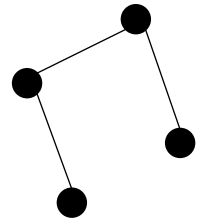
長さEの01ベクトル

$$S = \{e_X | X\text{は閉路を含まないリンク集合}\}$$

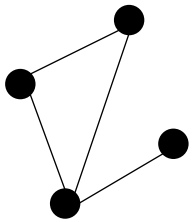
とする．

$f: \mathbb{Z}^E \to \overline{\mathbb{R}}$ を

$$f(x) = \begin{cases} \sum_{e \in X} c(e) \ (x \in S) \\ +\infty \ (\text{otherwise}) \end{cases}$$



閉路を含まない



閉路を含む

とすると，f はM$^\natural$凸関数．

→最小木問題は枝数=$|V|-1$という制約条件付きでのM$^\natural$凸最小化になる．

凸集合条件ではないが，効率的に解ける．
マトロイド基と関連．

まとめ

- 最適化問題は推定や機械学習の中でも登場する．
- 最適化問題は，連続最適化問題と組合せ最適化問題（=離散最適化）に大きく分けられる．
- 効率的なアルゴリズムを評価するために計算量オーダーが有用である．

- Optimization problems also appear in estimation and machine learning.
- Optimization problems can be broadly divided into continuous and combinatorial (=discrete) optimization problems.
- Time complexity is useful to evaluate efficient algorithms.