最適化の基礎

スタートアップゼミ #3

2025年4月23日

増田 慧樹・平松 正吾

目次

最適化とは

与えられた**制約条件**の下で**目的関数**の値を最小（もしくは最大）にする解を求める問題

→「どう行動すればいいか」の意思決定を構造化

一般形

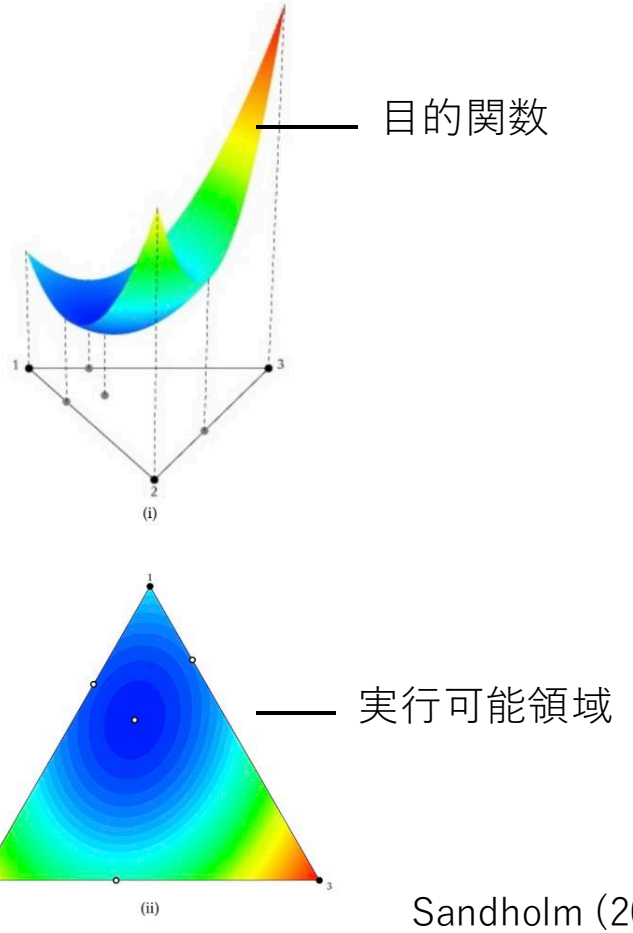
目的関数 objective function

最小化 $f(\boldsymbol{x})$

決定変数 decision variables

条件 $\boldsymbol{x} \in S$ — **実行可能領域** feasible region

制約条件 constraint



Sandholm (2010)

Figure 3.1.1: Graph (i) and contour plot (ii) of the potential function of 123 Coordination.

最適化のこころ

✅ **決定変数はなにか？**

- 離散 or 連続

✅ **目的関数はなにか？**

- 線形 or 非線形 or 2次関数

✅ **制約条件はなにか？**

↓

✅ **解法アルゴリズムとして何が使えそうか？**

- 厳密に解くor 近似的に解く

最適化　Optimization problem　$\min f(\boldsymbol{x})\ s.t.\ \boldsymbol{x} \in S$

与えられた制約条件の下で目的関数を最小化 (または最大化)する解を求める

連続最適化　*変数が連続的 (continuous) な値*

線形計画問題　*目的関数・制約条件が線形 (linear)*

最適輸送問題　点群から点群へ最小コストで輸送

解法: 単体法など

非線形計画問題　*目的関数・制約条件が非線形 (non-linear)*

解法: 最急降下法，ニュートン法

2次計画問題　*目的関数が二次関数 (quadratic)，制約条件が線形*

凸計画問題　*目的関数・制約条件が凸 (convex)*

Frank-Wolfe法

Simplicial decomposition法

制約つき最適化問題　*一般の制約条件*

KKT条件，バリア関数法

使う

行動モデル
機械学習
均衡配分

動学化

Optimal control

最適制御問題　*ある状態から目標状態まで，制約を満たしながら最小コストで到達する*

可制御性　*到達可能か，制御しやすさの評価*

可制御性グラミアン

関連

強化学習、Recursive logitモデル

離散最適化 (組合せ最適化)　*変数が離散的 (discrete) な値*

整数計画問題　*全ての変数が整数値 (integer)*

配送計画問題　複数車両で顧客を訪問する経路の組合せ

施設配置問題　需要をカバーできる施設の配置場所

混合整数計画問題　*一部の変数が整数値 (mixed-integer)*

ネットワークデザイン問題 **(Network Design Problem)**　混雑緩和のために，どの道路をどのくらい拡張するか

ネットワーク最適化問題

ネットワークやグラフ上の最適化

最短路問題　最短経路を求める (Dijkstra法など)

最小費用流問題　輸送コストを最小化する交通量を求める

メタヒューリスティクス　*NP-hard な問題に対する近似解法*

動学化

Reconfiguration

組合せ遷移　*ある状態から目標状態まで，制約を満たしながら組合せ構造を変化させる*

Multi-objective

多目的最適化問題

目的関数が2つ以上→ トレードオフの分析

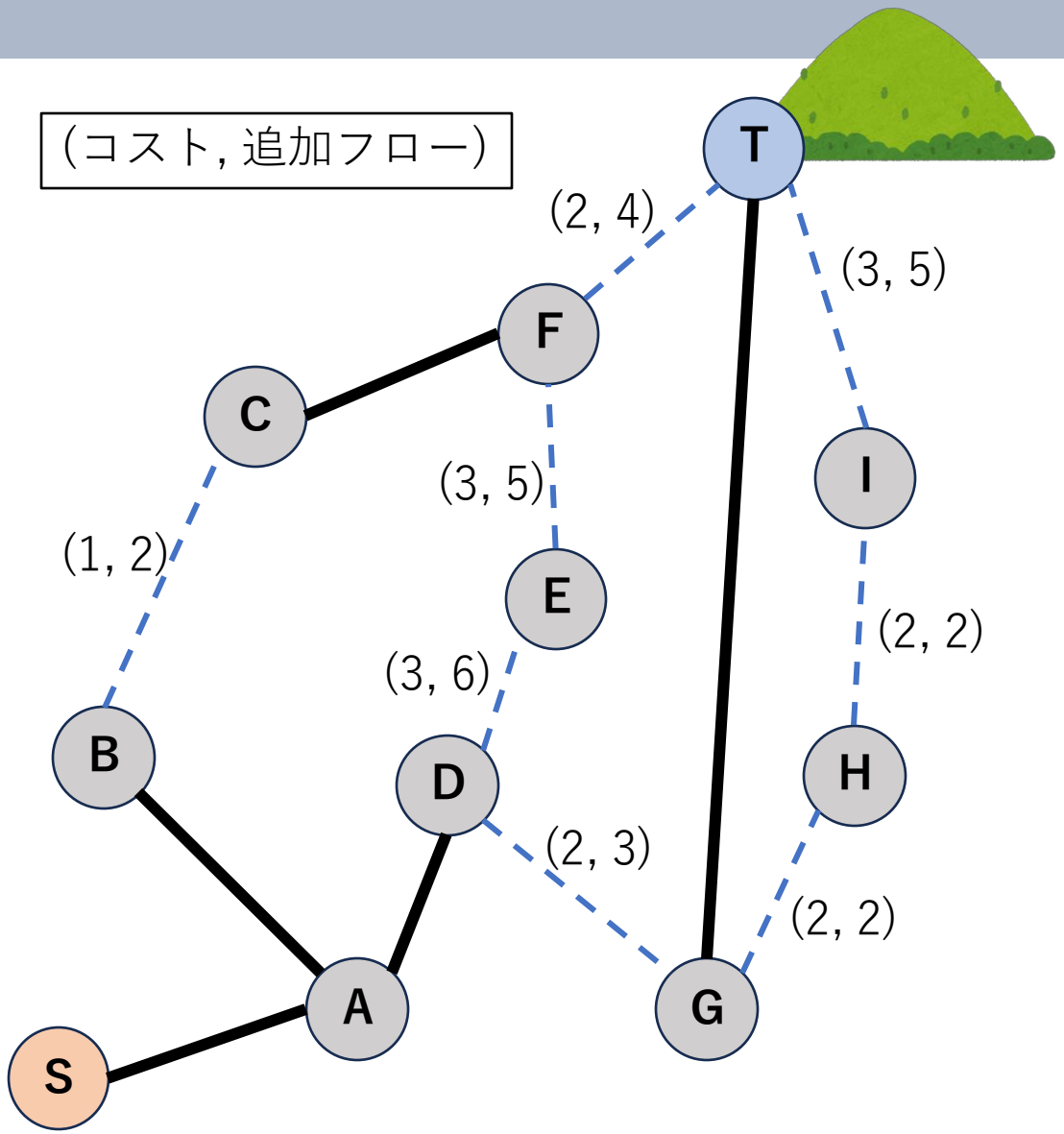
パレートフロンティア

不確実性

確率計画法　*目的関数・制約条件のパラメータに不確実性がある*

ハンズオン

- ある町では、災害時の円滑な避難のために、危険な道路（リンク）をあらかじめ強化する必要があります。
- しかし、予算や時間には限りがあるため、すべての道路を強化することはできません。
- 限られた予算で、どの道路リンクを優先的に強化すれば、避難者全体がよりスムーズに避難できるか？
- **予算10以内**で、強化によって追加されるフローが最大になるようなリンクの組み合わせを見つけてください。



定式化する

- 変数は？（何を決めたいか）
- 目的関数は？（どの指標を良くしたいか）
- 制約条件は？

定式化する

- 変数は？（何を決めたいか）

  → 各リンクが強化されるか否かを表す**バイナリ（二値）変数**　x_{ij}

- 目的関数は？（どの指標を良くしたいか）

  → 強化されたリンクのフローの合計を最大化

- 制約条件は？

  → リンクの強化コストの合計が10を超えない

リンクij間の追加フロー

$$\max_{x_{ij}} \sum_{i,j \in S} F_{ij} x_{ij}$$

リンクijの強化コスト

$$s.t. \quad \sum_{i,j \in S} c_{ij} x_{ij} \leq 10$$

$$x_{ij} \in \{0,1\}, i,j \in S$$

解いてみよう

- 制限時間2分で、目的関数を一番大きくする組み合わせを見つけてみよう
- 組合せ最適化の代表的な解法
  - 全列挙 (厳密解法)
  - 貪欲法 (近似解法)
  - 分枝限定法 (厳密解法)
  - ヒューリスティクス (発見的解法)

解法1: 全列挙

- 候補となるリンクは8本 → $2^8 = 256$回の探索で、全ての解をしらみつぶしに探せる

  （解）(D-E, D-G, E-F, F-T). 目的関数値: 6, コスト: 10.

- 問題が大きくなると、現実的な時間で列挙することは難しくなる (組合せ爆発)

2^{10}	1,024
2^{20}	1,048,576
2^{30}	1,073,741,824
2^{40}	1,099,511,600,000
2^{50}	112,589,990,000,000

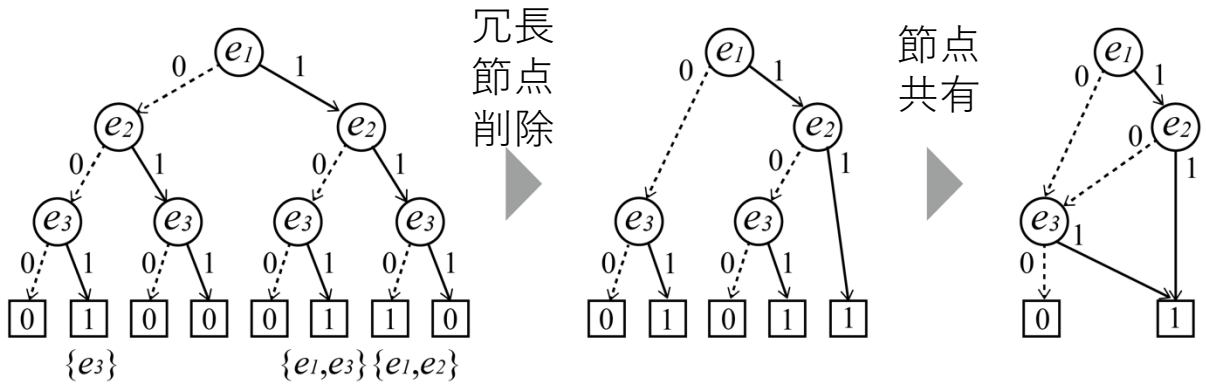
https://youtu.be/Q4gTV4r0zRs?si=41XY_KLdkGsvSlWu

- 列挙の技術も進展

  → 実行可能領域の効率的な探索や、

  制約の絞り込み等の有用性

Zero-suppressed binary Decision Diagram (ZDD) →



解法2: 貪欲法 (greedy method)

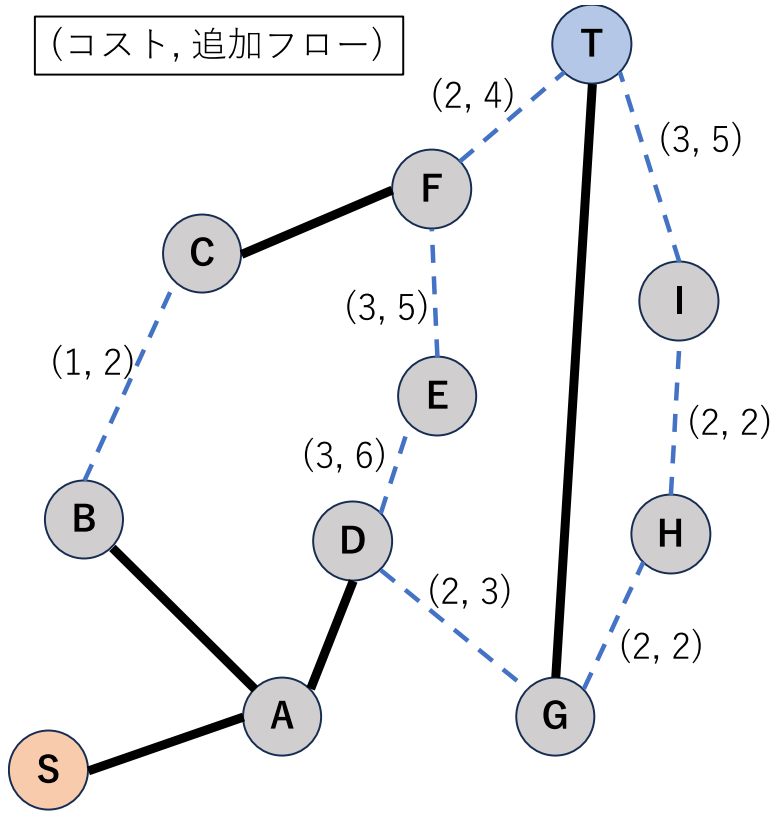
- 各段階で局所的な評価値がもっとも高い要素を選ぶ手法

1. すべての候補リンクについて「効果 ÷ コスト」（=効率）を計算
   例. リンク D-E → フロー増加6／費用3 → 効率 = 2.0
2. 効率が高い順に並べる
   → 「1円あたり多くの人が通れるリンク」を優先
3. 予算が尽きるまで順に選んでいく

B-C	2
D-E	2
F-T	2
E-F	1.67
I-T	1.67
D-G	1.5
G-H	1
H-I	1

上4つを選んで終了。
目的関数値: 17

最適解。目的関数値: 18



解法2: 貪欲法 (greedy method)

- 貪欲法はどれくらい真の最適解に近いか？→定量的な指標: **近似比率**
- 真の最適化をOPT, 近似アルゴリズムが出力する最適値をALGとしたとき、

  最大化問題なら、$ALG \geq R \cdot OPT\ (R \leq 1)$　　最小化問題なら、$ALG \leq R \cdot OPT\ (R \geq 1)$

  が成り立つとき、アルゴリズムは近似比率Rをもつという。
- 上記の貪欲法では、近似比率は任意の定数になるが、最後に以下のステップを加えることで、
  1/2-近似アルゴリズムになる。
  ＝ 最適値の1/2以上を出力することを保証

効果 (フロー増加)が最大のものを選び、貪欲法の解と比較し、大きい方を解とする

解法3: 分枝限定法

- 直接解くことが難しい問題をいくつかの子問題に分割する**分枝操作**と、
  最適解が得られる見込みのない子問題を見つける**限定操作**を繰り返し適用。

Step 1. 適当な方法で、元問題P_0の暫定解x'を決め、目的関数値zを計算。

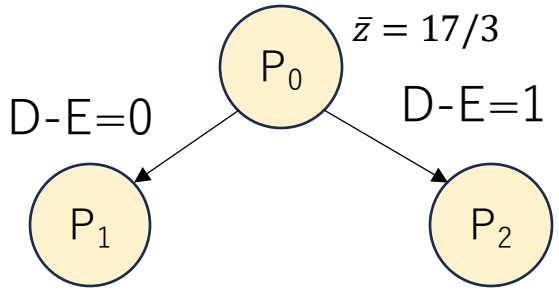
(B-C, D-E, D-G, G-H)=(0, 0, 0, 0). $z = 0$

Step 2. **<u>線形緩和問題</u>**を解く

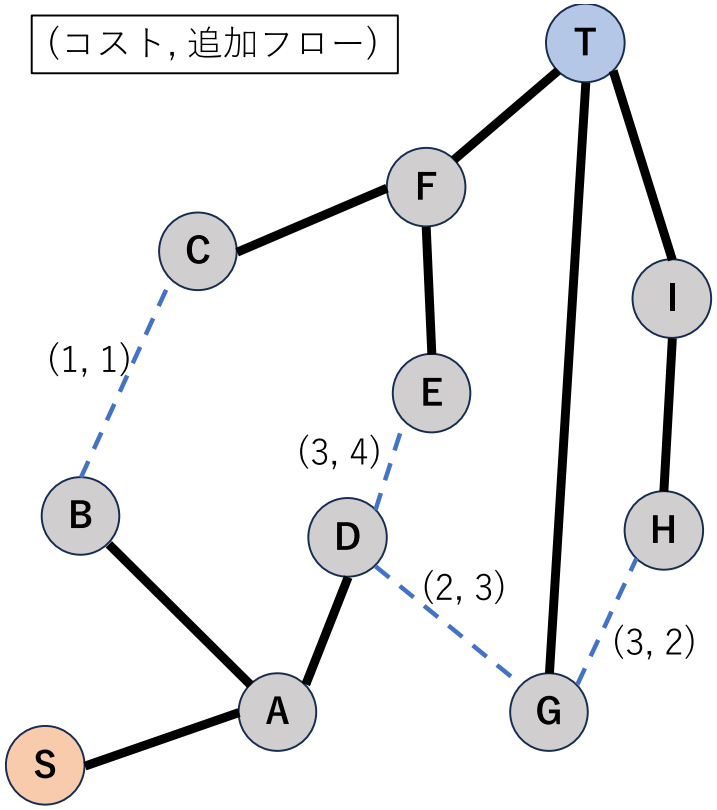
$x \in \{0,1\}$の制約を緩めて、$0 \leq x \leq 1$と連続値を取れるようにする
→ xのとれる領域が広がったので、最適値も改善されることが期待

(B-C, D-E, D-G, G-H)=(0, 2/3, 1, 0). $z = \frac{17}{3}$

Step 3. D-E=0 or 1で、子問題に分割 (**分枝操作**)



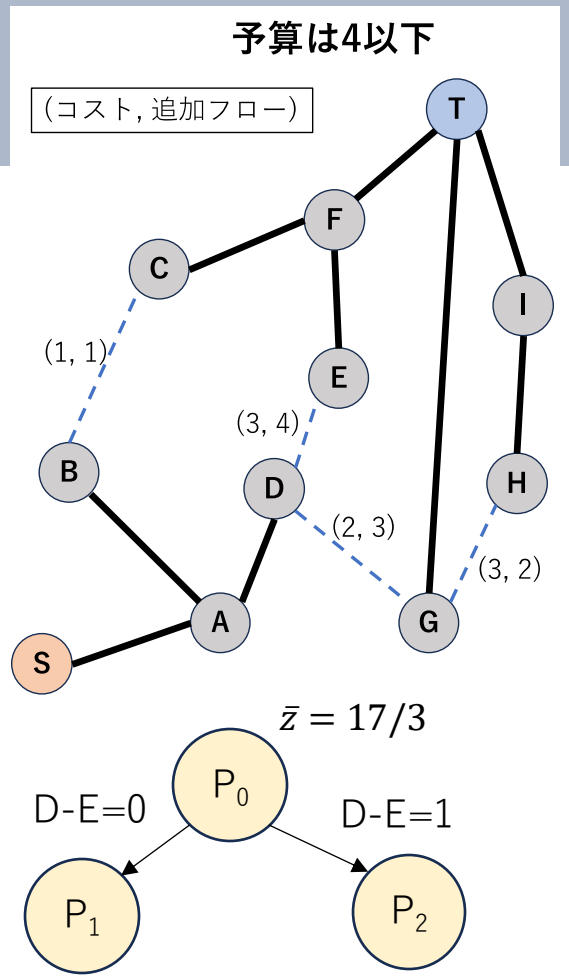
予算は4以下



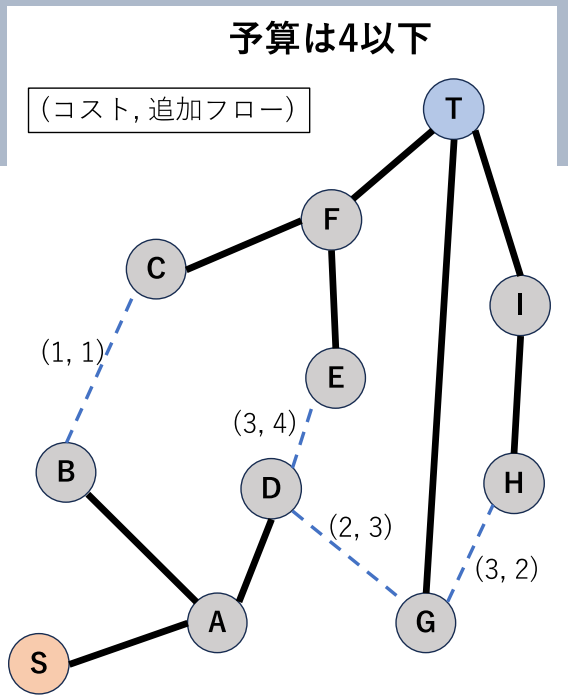
解法3: 分枝限定法

Step 2. P_1の**線形緩和問題**を解く

(B-C, D-E, D-G, G-H)=(_, 0, _, _). $z = \_$
固定



解法3: 分枝限定法



Step 2. P_1の**線形緩和問題**を解く

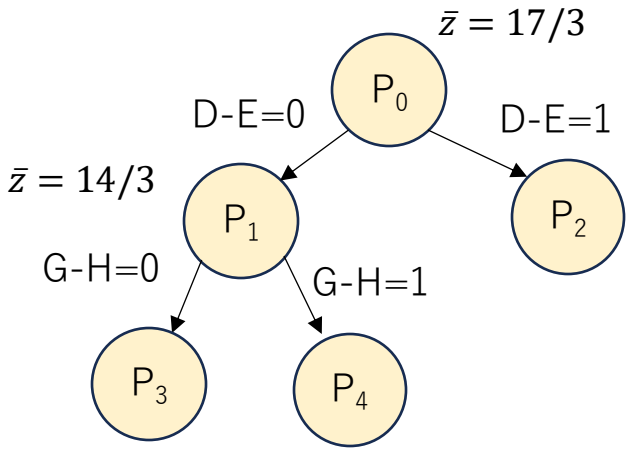
(B-C, D-E, D-G, G-H)=(_, 0, _, _). $z = \_$
固定

(B-C, D-E, D-G, G-H)=(1, 0, 1, 1/3). $z = \frac{14}{3}$

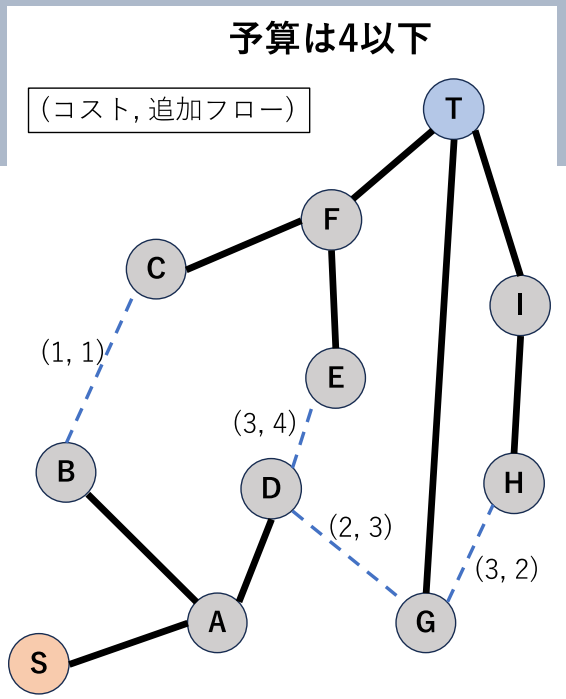
Step 3. G-H=0 or 1で、子問題に分割 (**分枝操作**)

Step 2. P_3の**線形緩和問題**を解く

(B-C, D-E, D-G, G-H)=(_, 0, _, 0). $z = \_$
固定



解法3: 分枝限定法



Step 2. P_1の**線形緩和問題**を解く

(B-C, D-E, D-G, G-H)=(_, 0, _, _). $z = \_$
固定

(B-C, D-E, D-G, G-H)=(1, 0, 1, 1/3). $z = \frac{14}{3}$

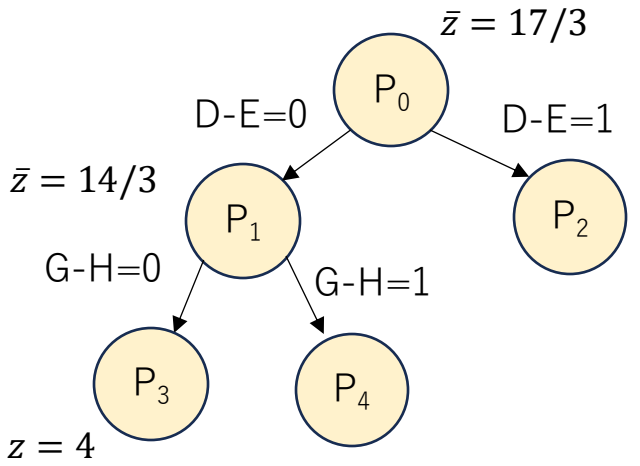
Step 3. G-H=0 or 1で、子問題に分割 (**分枝操作**)

Step 2. P_3の**線形緩和問題**を解く

(B-C, D-E, D-G, G-H)=(_, 0, _, 0). $z = \_$
固定

(B-C, D-E, D-G, G-H)=(1, 0, 1, 0). $z = 4$

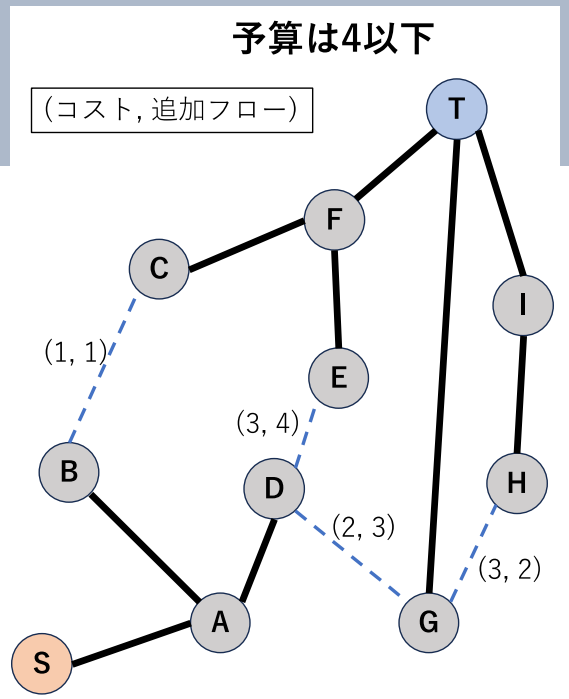
元の問題の実行可能解なので暫定値を$z' = 4$と更新

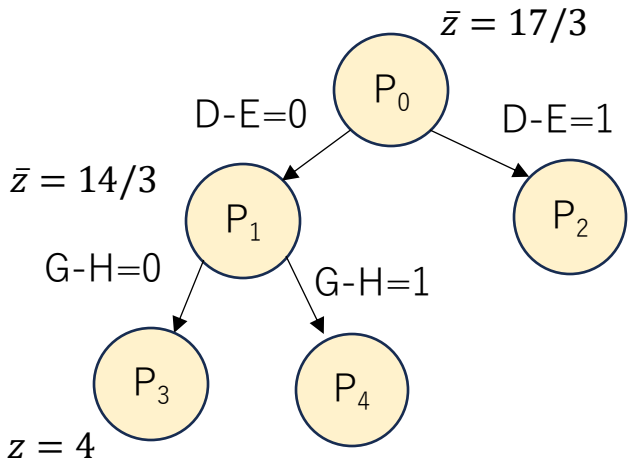


解法3: 分枝限定法

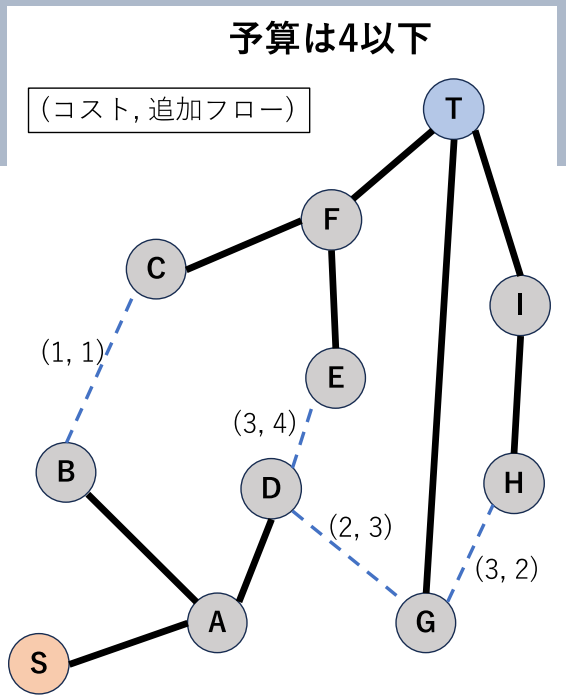
Step 2. P_4の**線形緩和問題**を解く

(B-C, D-E, D-G, G-H)=(_, 0, _, 1). $z = \_$
固定





解法3: 分枝限定法



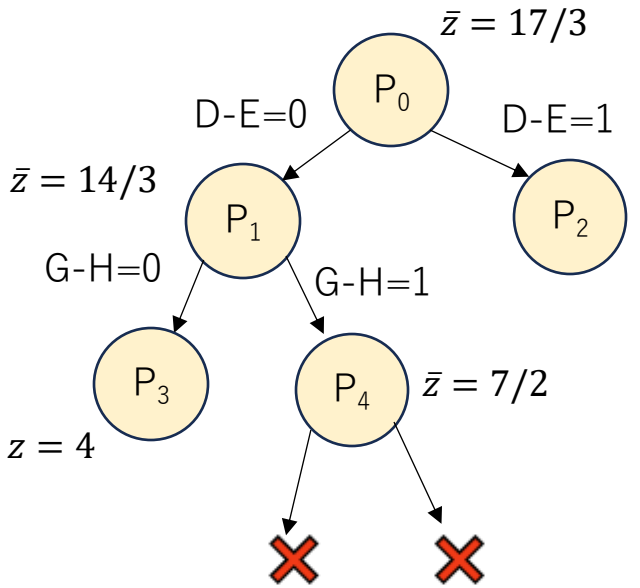
Step 2. P_4の**線形緩和問題**を解く

(B-C, D-E, D-G, G-H)=(_, 0, _, 1). $z = \_$
固定

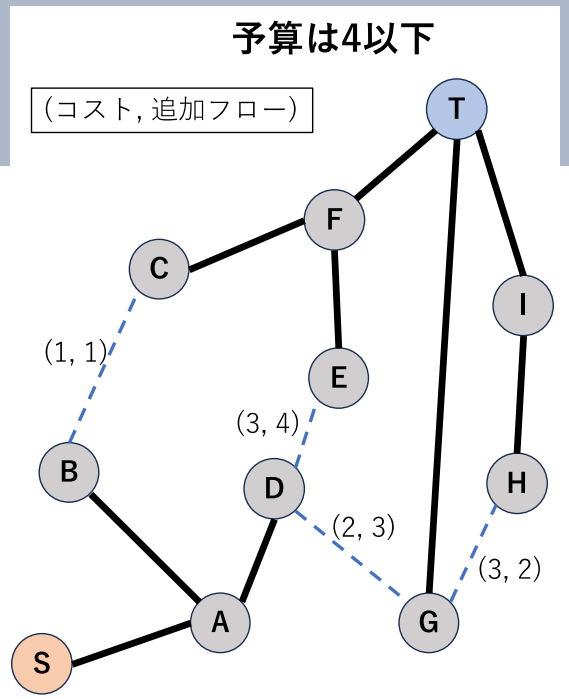
(B-C, D-E, D-G, G-H)=(0, 0, 1/2, 1). $z = \frac{7}{2}$

元問題の実行可能解の暫定値: $z' = 4 >$ 線形緩和問題P_4の値: $z = \frac{7}{2}$

P_4の値が暫定値を超えることはないので、展開しない (**限定操作**)



解法3: 分枝限定法



Step 2. P_4の**線形緩和問題**を解く

(B-C, D-E, D-G, G-H)=(_, 0, _, 1). $z = \_$
固定

(B-C, D-E, D-G, G-H)=(0, 0, 1/2, 1). $z = \frac{7}{2}$

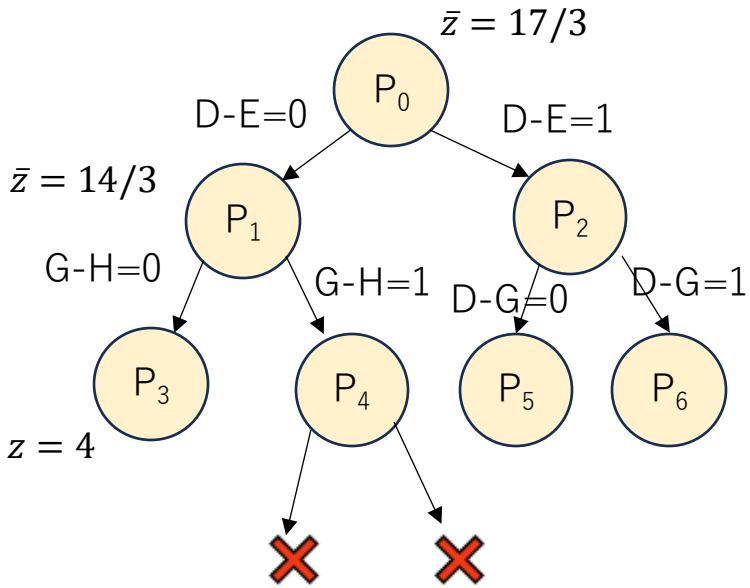
元問題の実行可能解の暫定値: $z' = 4 >$ 線形緩和問題P_4の値: $z = \frac{7}{2}$

P_4の値が暫定値を超えることはないので、展開しない (**限定操作**)

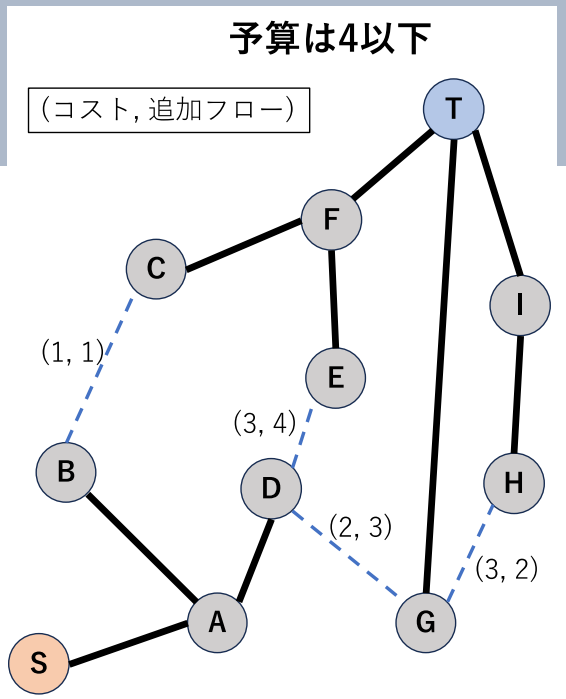
Step 2. P_2の**線形緩和問題**を解く

(B-C, D-E, D-G, G-H)=(0, 1, 1/2, 0). $z = \frac{11}{2}$

Step 3. D-G=0 or 1で、子問題に分割 (**分枝操作**)



解法3: 分枝限定法



Step 2. P_5の**線形緩和問題**を解く

(B-C, D-E, D-G, G-H)=(_, 1, 1, _). $z = \_$
固定

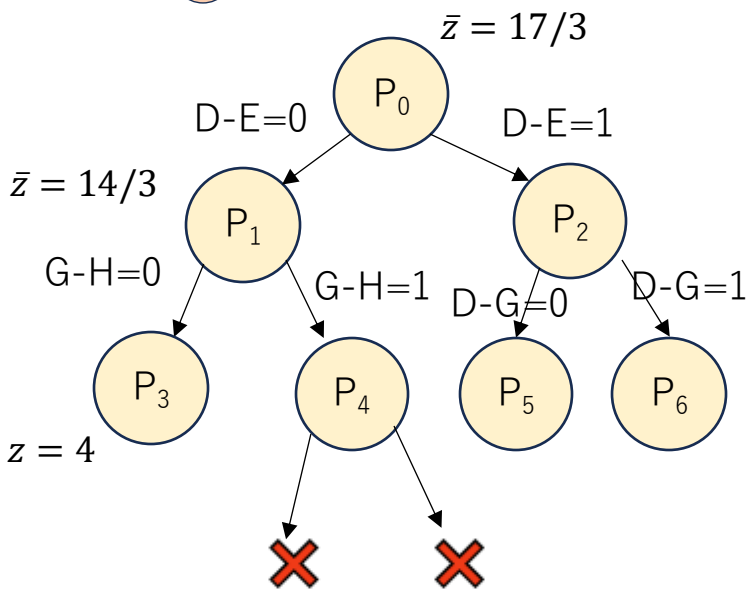
この時点で制約を満たさないので、展開しない (**限定操作**)

Step 2. P_6の**線形緩和問題**を解く

(B-C, D-E, D-G, G-H)=(1, 1, 0, 0). $z = 5$
固定

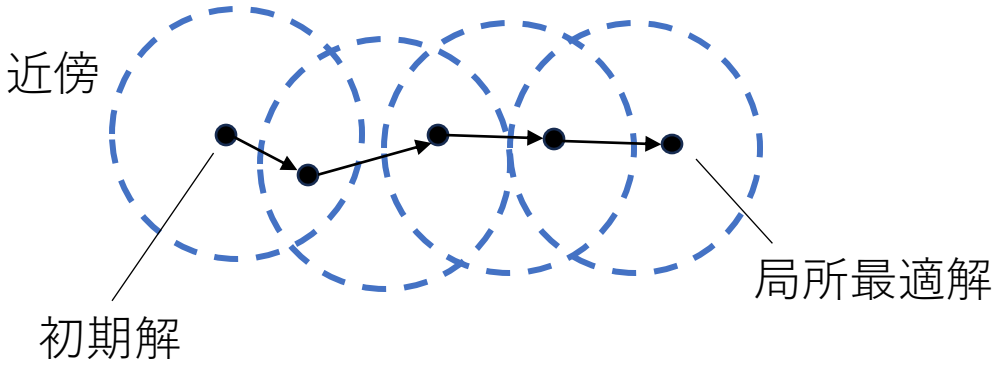
元の問題の実行可能解なので暫定値を$z' = 5$と更新

Step 4. 展開する節点がなくなったので、最適解(1,1,0,0)を出力して終わり

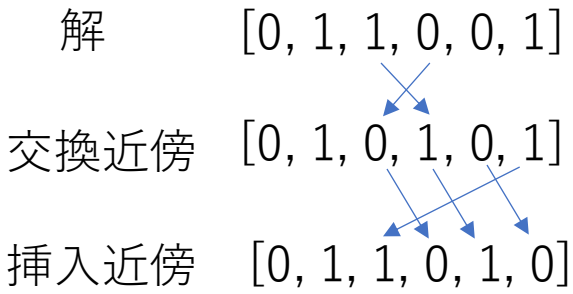


解法4: 発見的解法 (heuristics)

- 問題特有の知見に基づいて作られた、最適値に対する近似性能の保証はないが多くの問題例に対して質の高い実行可能解を出力するアルゴリズム
- 例: 局所探索法 (Local search method)
  適当な実行可能解$\boldsymbol{x} \in S$から始めて、現在の解を少し変形させて得られる解の集合$N(\boldsymbol{x}) \subset S$内に、
  $f(\boldsymbol{x}') > f(\boldsymbol{x})$を満たす解改善解$\boldsymbol{x}'$ があれば移動する手続きを繰り返す。
  $N(\boldsymbol{x})$を、近傍 (neighborhood)と呼ぶ。
- 近傍の定義、探索空間、解の評価、移動戦略など、自由度が大きい

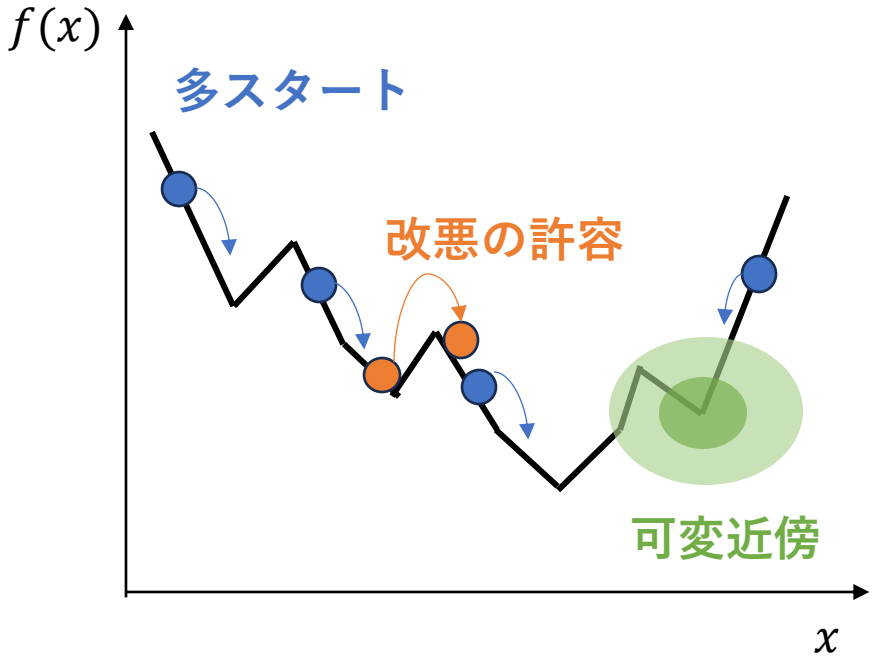


近傍の例



解法4: 発見的解法 (heuristics)

- 多くの組合せ最適化問題に対して適用できる発見的解法の一般的な手法
  - 過去の探索で得られた良い解の周辺を集中的に探索する**集中化**
  - 過去の探索とは異なる構造を持つ解を探索する**多様化**
    を組み合わせる
- 局所探索の改善
  - 多スタート・反復局所探索、可変近傍探索、大規模近傍探索
- 生物からヒントを得たもの
  - アリコロニー最適化、粒子群最適化
- その他
  - 遺伝的アルゴリズム、焼きなまし法



まとめ – 組合せ最適化へのアプローチ

- 全列挙 (厳密解法)
  - 組み合わせ爆発、ZDD
- 貪欲法 (近似解法)
  - 近似比率
- 分枝限定法 (厳密解法)
  - 子問題、線形緩和
- ヒューリスティクス (発見的解法)
  - 局所探索、近傍

連続最適化

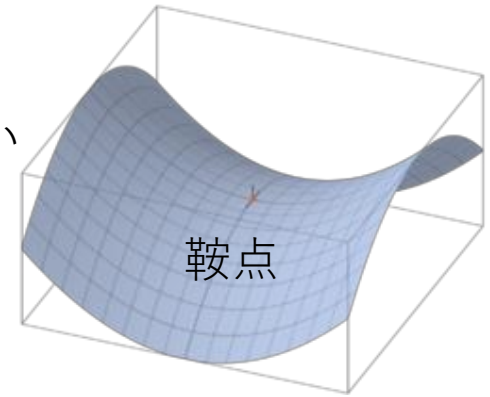
解析解

- 目的解の形によっては、数式の操作によって解を導ける場合がある（=**解析解**）
  → 停留点（傾きが0になる点）を求める（最適性の1次の必要条件）

  例）$f(x) = ax^2 + bx + c, a > 0$

  $f'(x) = 2ax + b = 0$より、$x = -\frac{b}{2a}$は最適性の1次の必要条件を満たす。

- 極大値や鞍点も、傾き=0を満たすので、 1次の条件は最適性の十分条件ではない



- 最適性の2次の条件も考えると、必要条件と十分条件が言える。
  - 点$\boldsymbol{x}^*$が局所最適解 → ヘッセ行列$\nabla^2 f(\boldsymbol{x}^*)$が半正定値
  - 点$\boldsymbol{x}^*$が停留点でヘッセ行列$\nabla^2 f(\boldsymbol{x}^*)$が正定値 →点$\boldsymbol{x}^*$が局所最適解

数値解

- 私たちが扱う問題は紙とペンでは解けないことが多くある

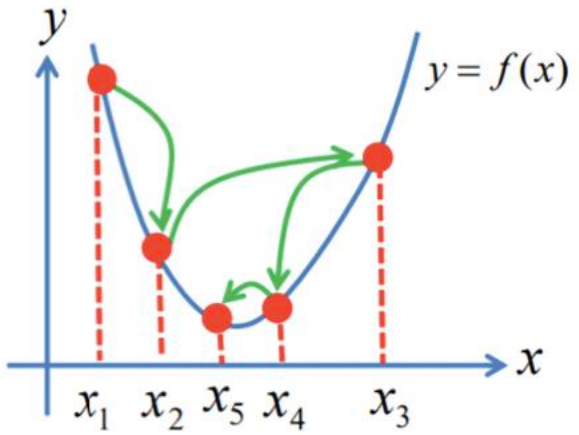
例）尤度最大化 $\max \mathrm{LL}(\boldsymbol{\theta}) = \sum_{i=1}^{n} y_i \log \frac{\exp(\boldsymbol{x}^T\boldsymbol{\theta})}{\sum \exp(\boldsymbol{x}^T\boldsymbol{\theta})}$

→ コンピュータで数値的に解く

制約なし非線形最適化問題の数値解法

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

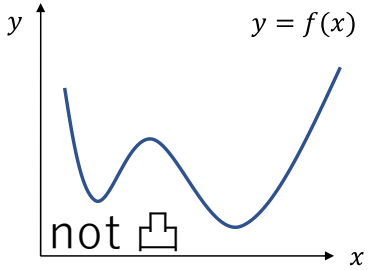


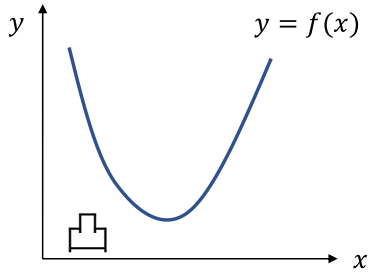
ステップAとステップBを繰り返すことで局所最適解を見つける

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

最急降下法

- 各反復において、 $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha \boldsymbol{d}\left(\boldsymbol{x}^{(k)}\right)$ と更新する。

  ステップ幅　探索方向

- 更新先での目的関数値は、$f\left(\boldsymbol{x}^{(k)} + \alpha \boldsymbol{d}\left(\boldsymbol{x}^{(k)}\right)\right) \approx f\left(\boldsymbol{x}^{(k)}\right) + \alpha \nabla f\left(\boldsymbol{x}^{(k)}\right)^T \boldsymbol{d}\left(\boldsymbol{x}^{(k)}\right)$

  - $\nabla f\left(\boldsymbol{x}^{(k)}\right)^T \boldsymbol{d}\left(\boldsymbol{x}^{(k)}\right) < 0$ならば、十分小さいステップ幅$\alpha$に対して、 $f\left(\boldsymbol{x}^{(k)} + \alpha \boldsymbol{d}\left(\boldsymbol{x}^{(k)}\right)\right) < f\left(\boldsymbol{x}^{(k)}\right)$

  - 最急降下法では、 $\boldsymbol{d}\left(\boldsymbol{x}^{(k)}\right) = -\nabla f\left(\boldsymbol{x}^{(k)}\right)$ (勾配の逆方向)と定める。

- 非線形最適化では、目的関数の形を工夫して、凸最適化に落とし込んで勾配降下法で解くことを考えてみる！

動学的最適化

最短経路問題

- 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

解法:

- Dijkstra法: 1回のアルゴリズムで全ての頂点までの最短経路を求められる
- A*法: 終点までの距離といったヒューリスティックを用いる
- Bellman-ford法: 負のコストを持つ辺があっても計算可能

定式化する

- 変数は？（何を決めたいか）
- 目的関数は？（どの指標を良くしたいか）
- 制約条件は？

定式化する

- 変数は？（何を決めたいか）

  → 各リンクが最短経路に含まれるか否かの**バイナリ（二値）変数** x_e

- 目的関数は？（どの指標を良くしたいか）

  → 経路コストの最小化

- 制約条件は？

  → 始点と終点までグラフが接続していなければいけない

リンクij間の追加フロー

$$\min_{x_e} \sum_{e \in E} c_e x_e$$

$$s.t. \quad \sum_{e \in \delta^+(s)} x_e = 1$$

$\delta^+(s)$ sを始点とする辺集合

$$\sum_{e \in \delta^-(t)} x_e = 1$$

$\delta^-(t)$ tを終点とする辺集合

$$\sum_{e \in \delta^+(s)} x_e - \sum_{e \in \delta^-(t)} x_e = 1$$

$$x_e \in \{0, 1\},$$

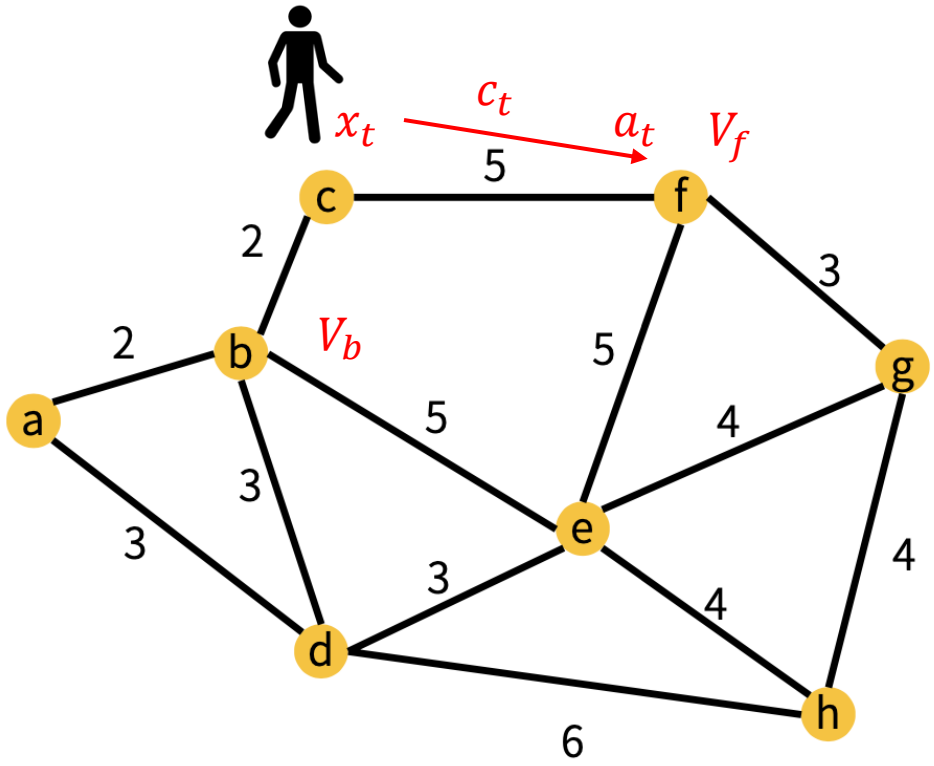
動学的な最適化問題

- **状態**：今どのノードにいるか x_t
- **行動**：どのノードに進むか a_t
- **遷移**：$x_{t+1} = f(x_t, a_t)$（＝次ノードへ移動）
- **各期のコスト**：$c_t(x_t, a_t)$（＝エッジの重み）
- **価値関数V**：各ノードから終点までのコスト

と捉えると、時間を含む動学的な最適化問題と言える。

ベルマン方程式: 最適性を表す

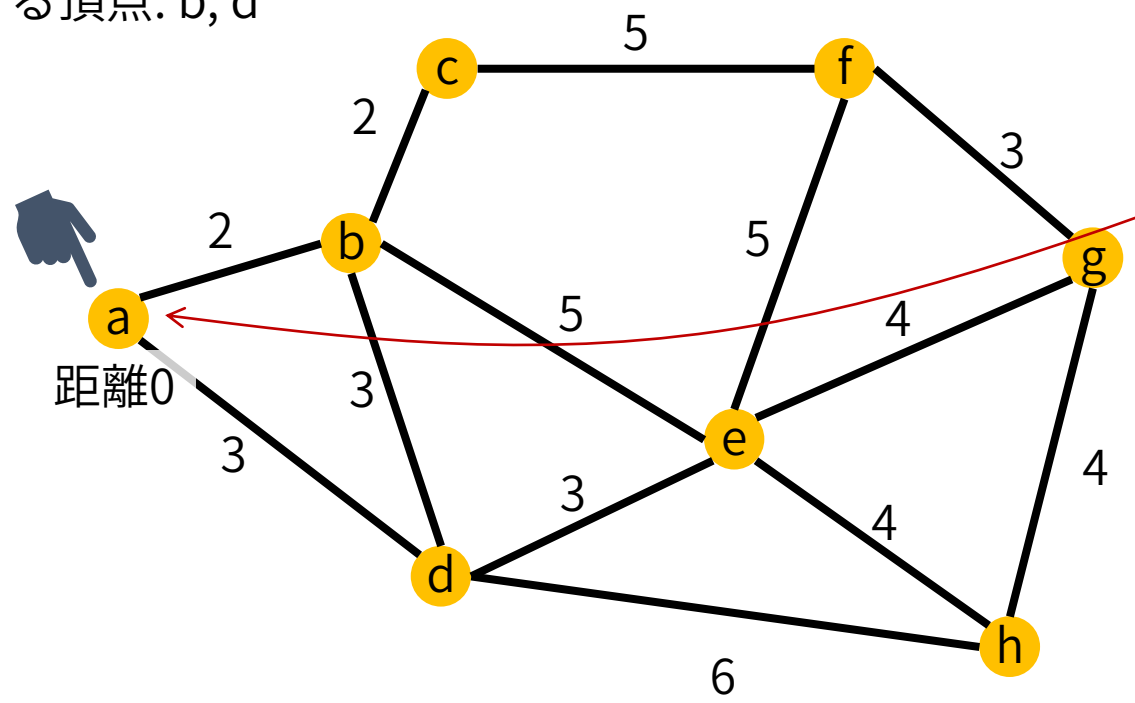
$$V(x_t) = \min_{j \in \text{neighbor}(x_t)} \{c_t(x_t, a_t) + V(x_{t+1})\}$$



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

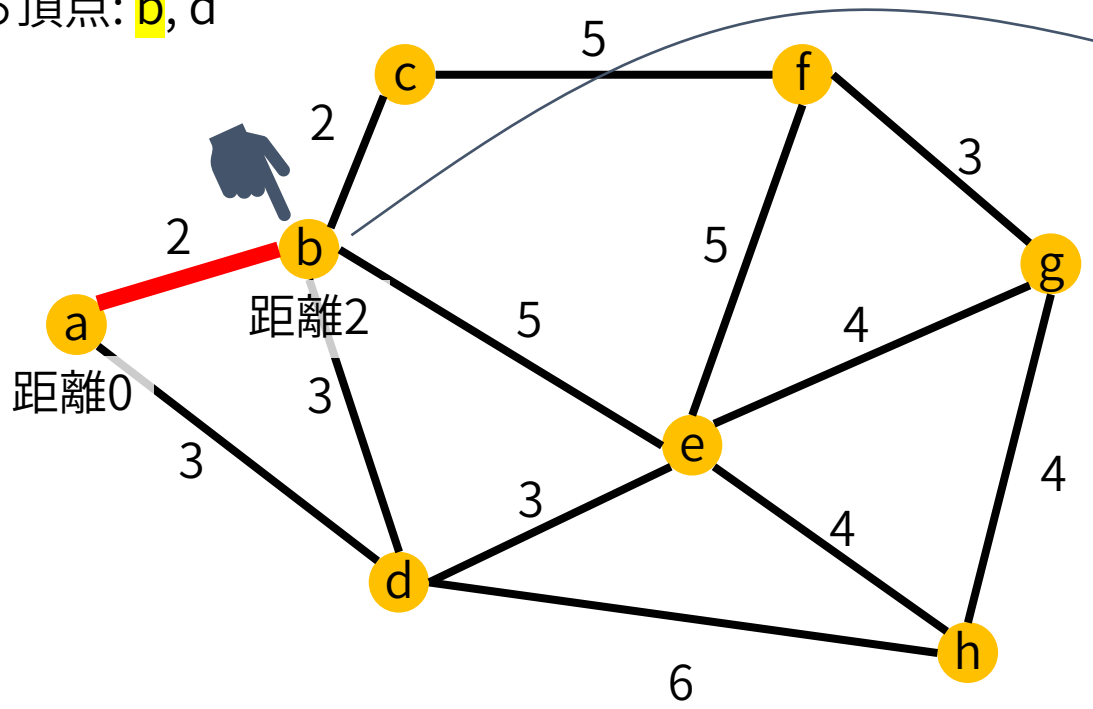
頂点	aからの距離
~~a~~	0

```
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

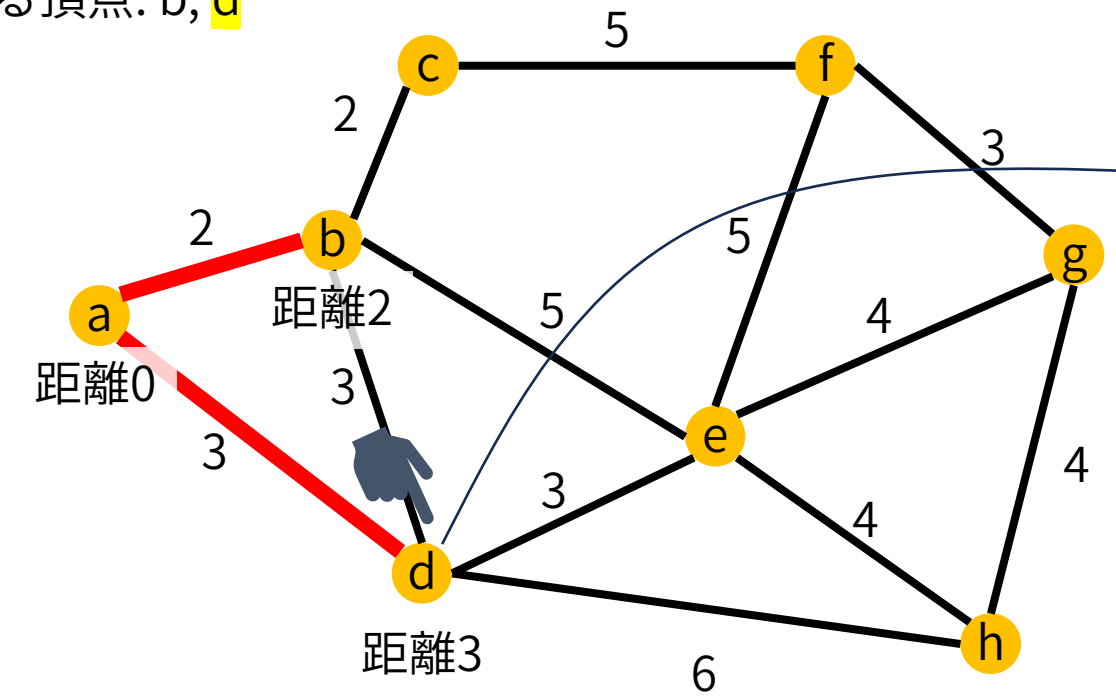
頂点	aからの距離
b	2

```
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

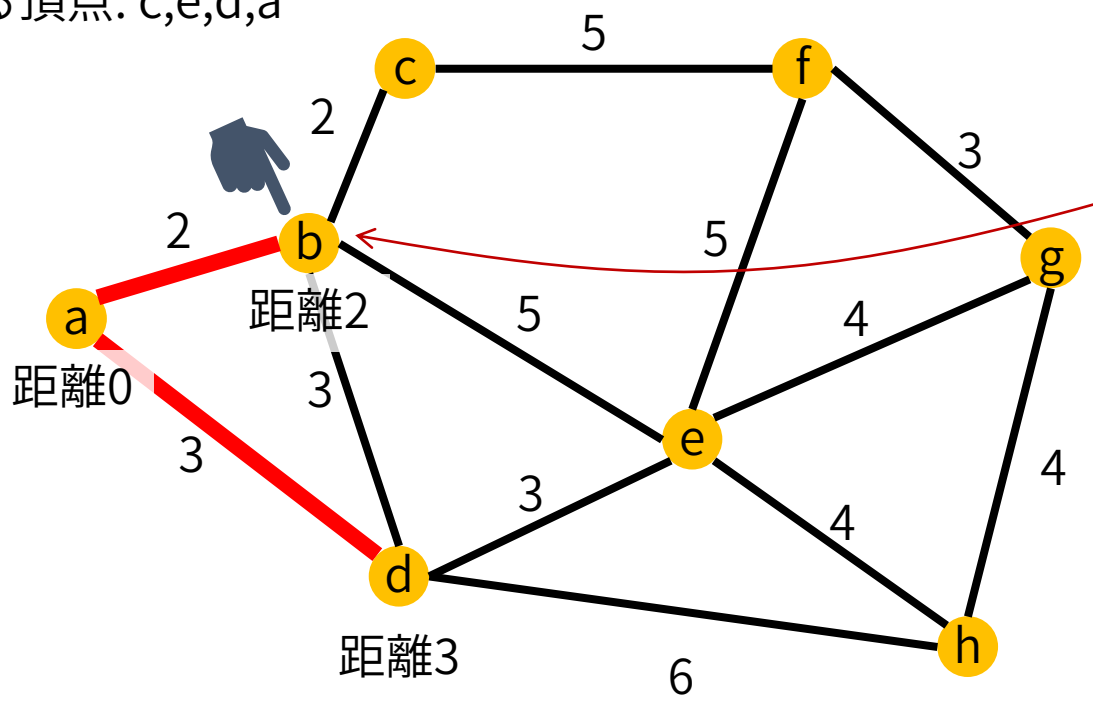
頂点	aからの距離
b	2
d	3

```
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

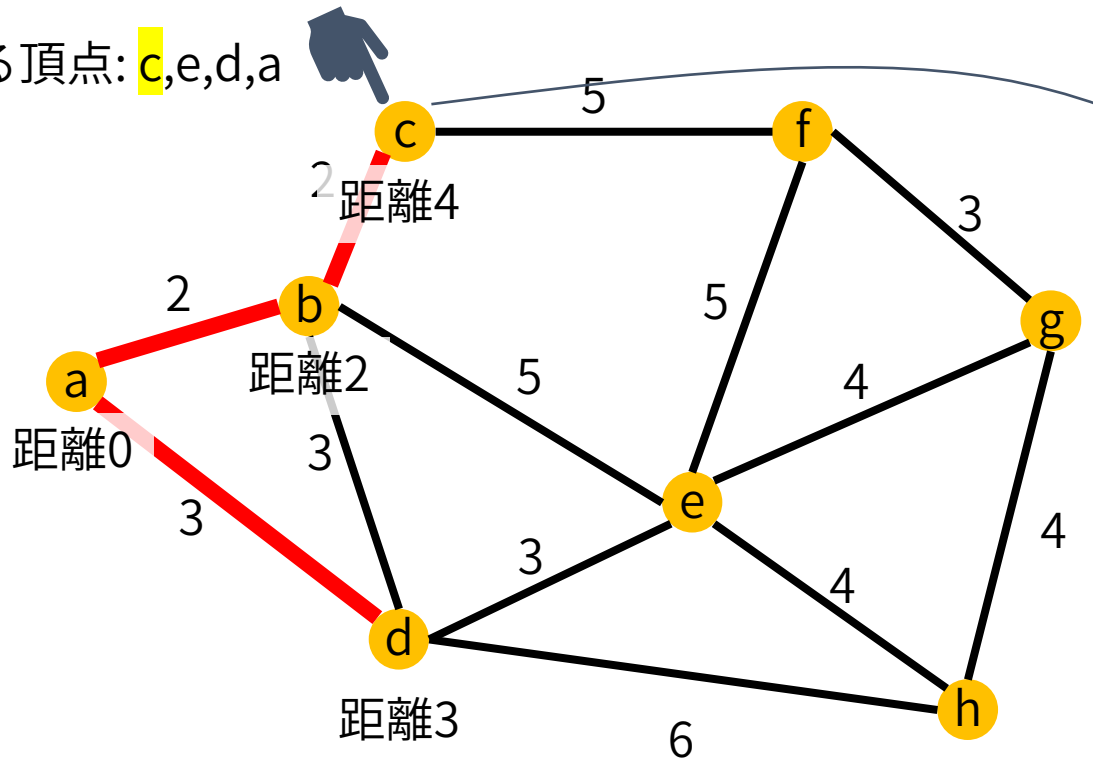
頂点	aからの距離
~~b~~	2
d	3

```
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

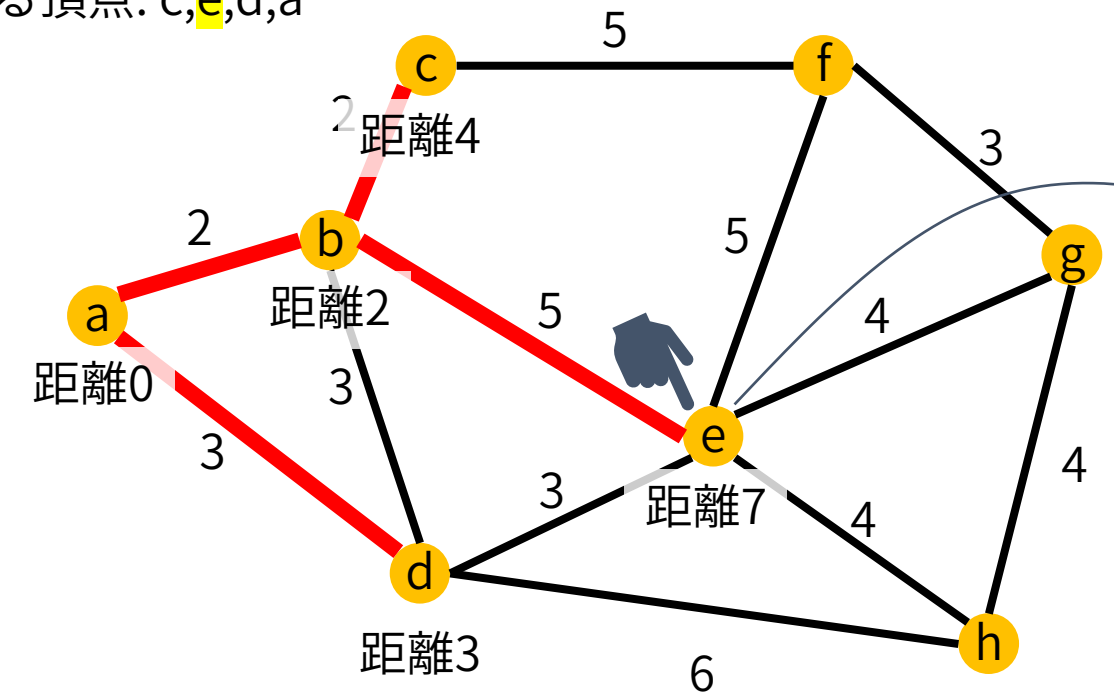
頂点	aからの距離
d	3
c	4

```
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

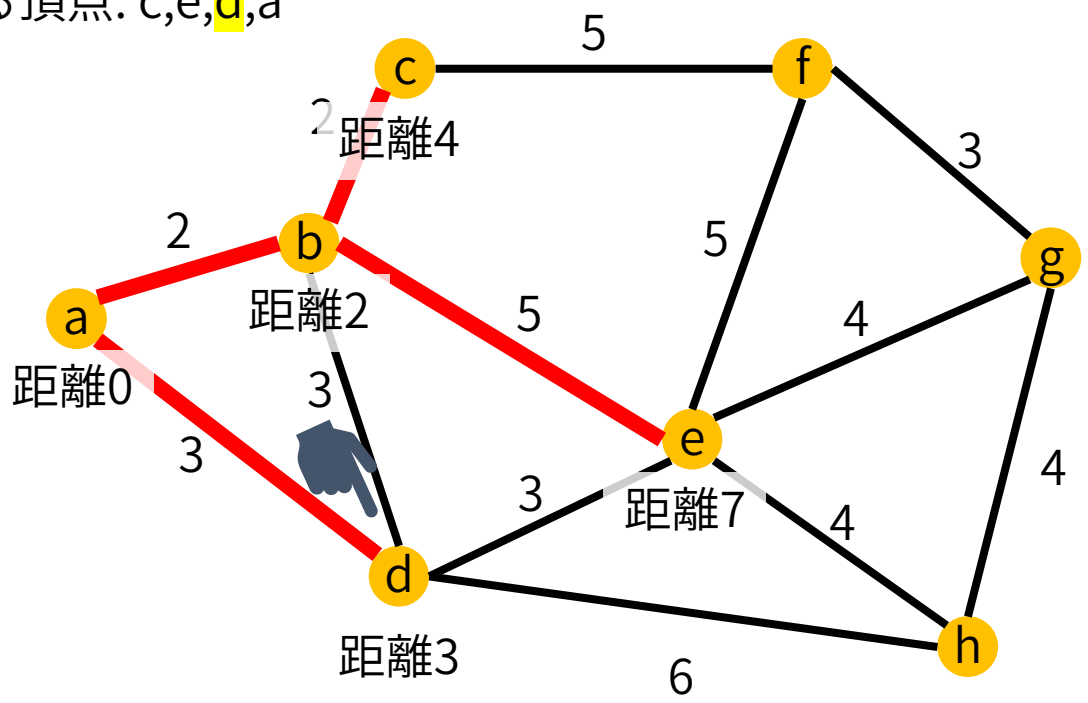
頂点	aからの距離
d	3
c	4
e	7

```
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

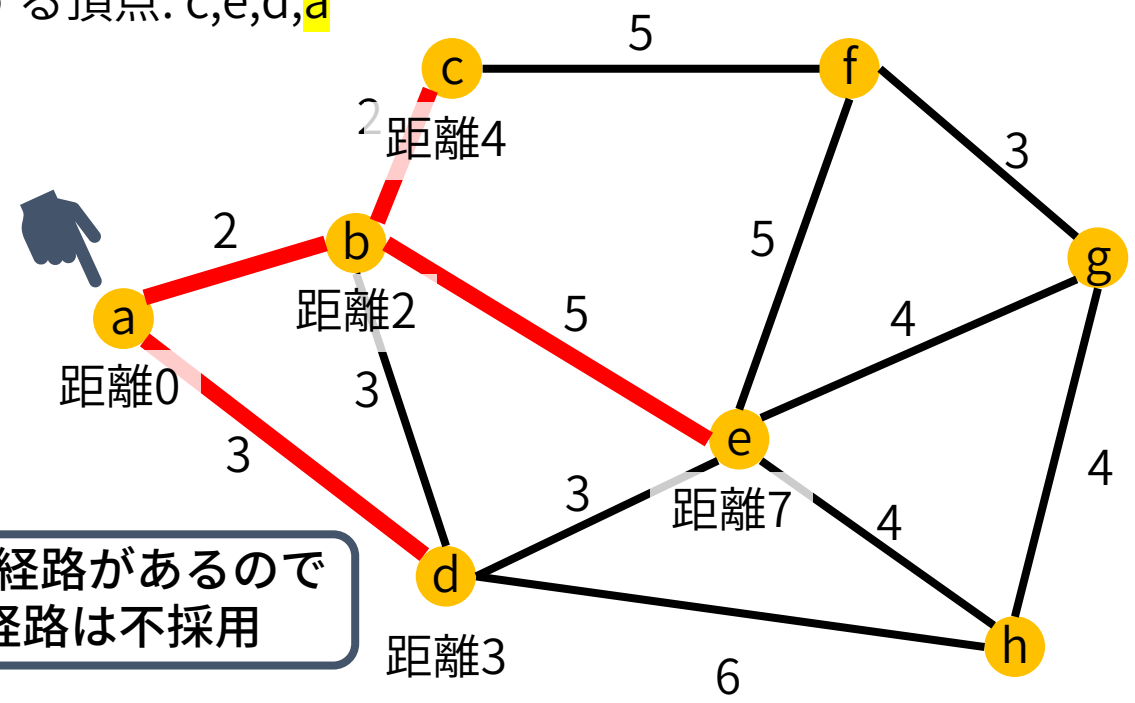
頂点	aからの距離
d	3
c	4
e	7

```
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

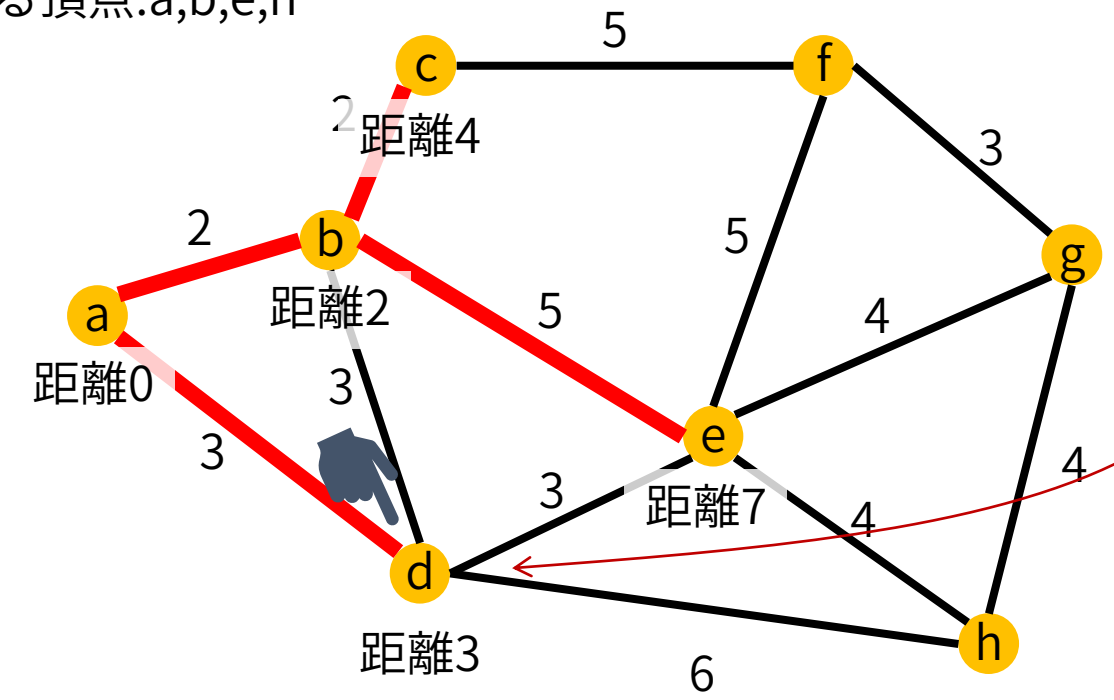
頂点	aからの距離
d	3
c	4
e	7

```
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

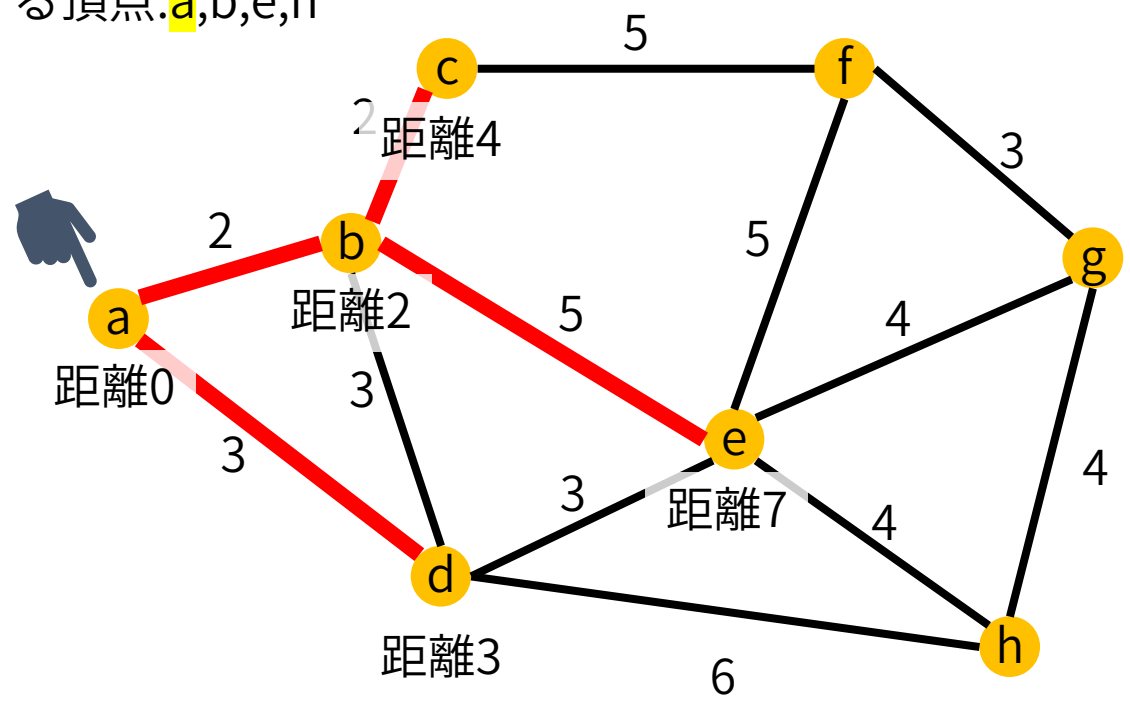
頂点	aからの距離
~~d~~	3
c	4
e	7

```
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

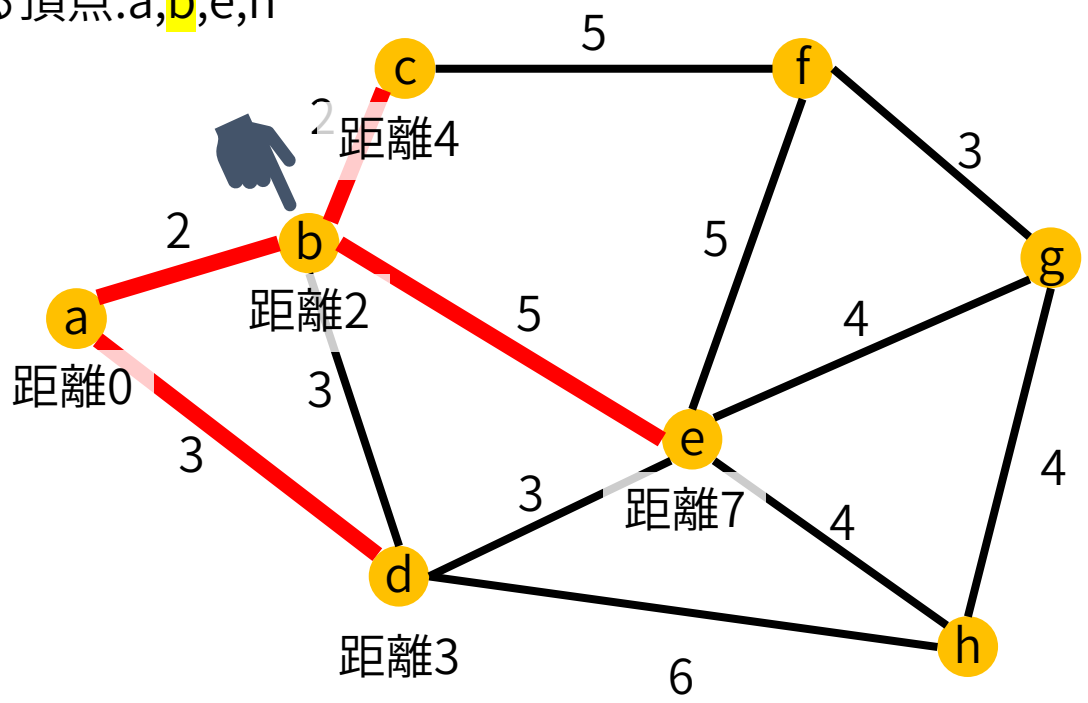
頂点	aからの距離
c	4
e	7

```
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

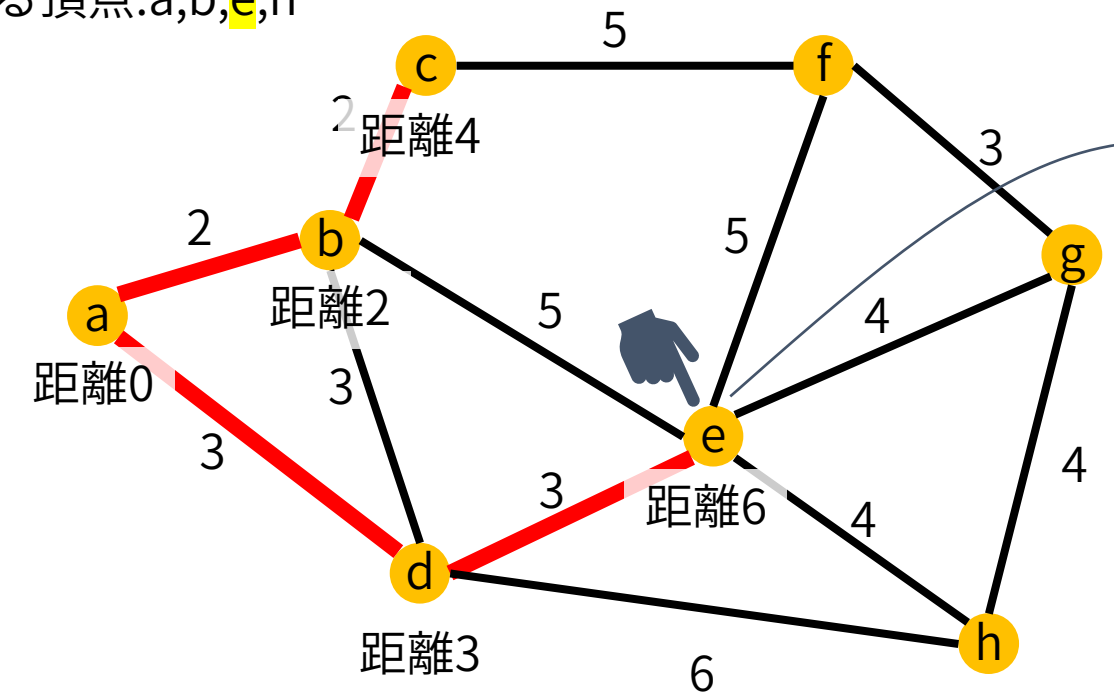
頂点	aからの距離
c	4
e	7

```
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

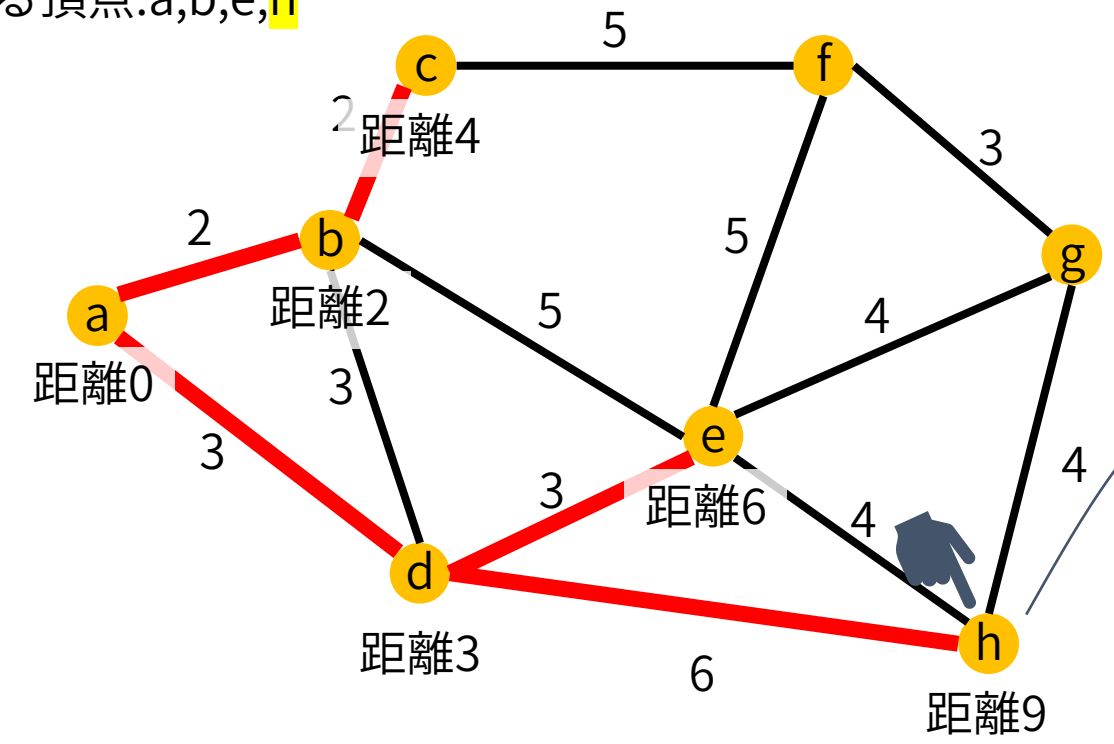
頂点	aからの距離
c	4
e	6
e	7

```
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

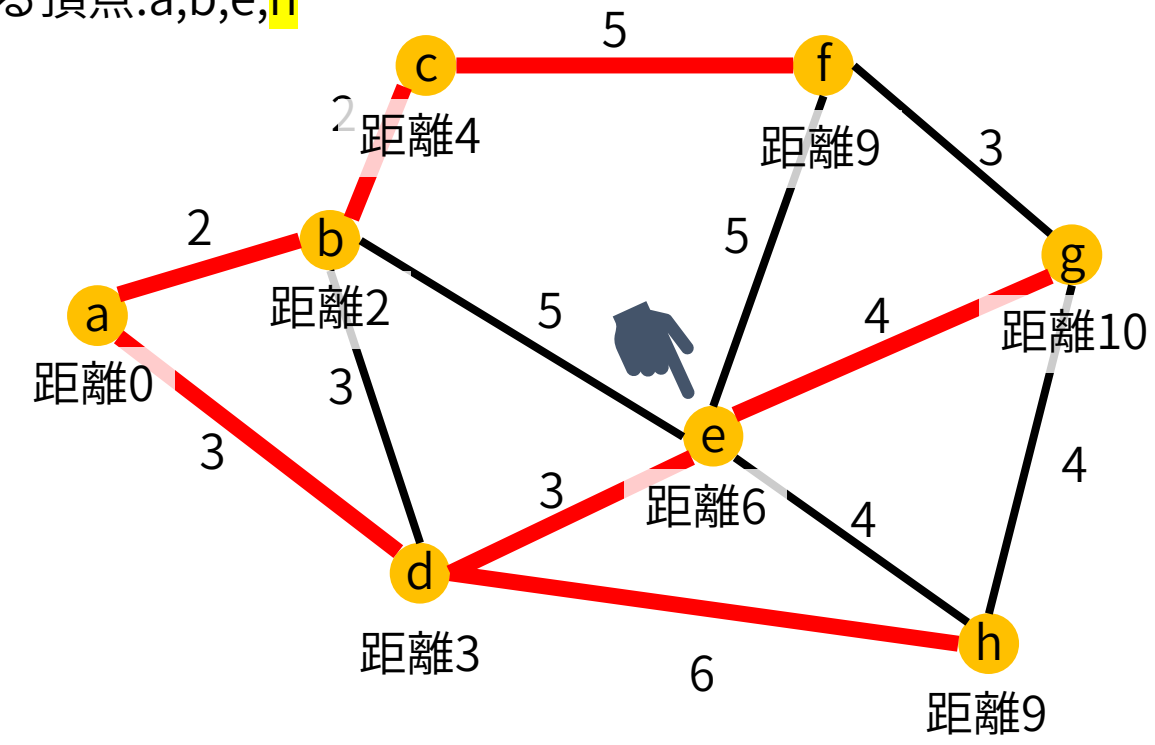
頂点	aからの距離
c	4
e	6
e	7
h	9

```
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

スタートアップゼミ#3
最適化 課題

2025/4/23

交通・都市・国土学研究室
平松正吾

Basic Problem #1: カーシェア車両へのマッチング

各利用者とカーシェア乗り場までの距離が与えられています.

利用者の移動距離が最小になるような，利用者（1, 2, …, 10）と乗車するカーシェア車両（a, b, c, …, l）の組み合わせを，2通りの方法で求めてください.

＜ルール＞

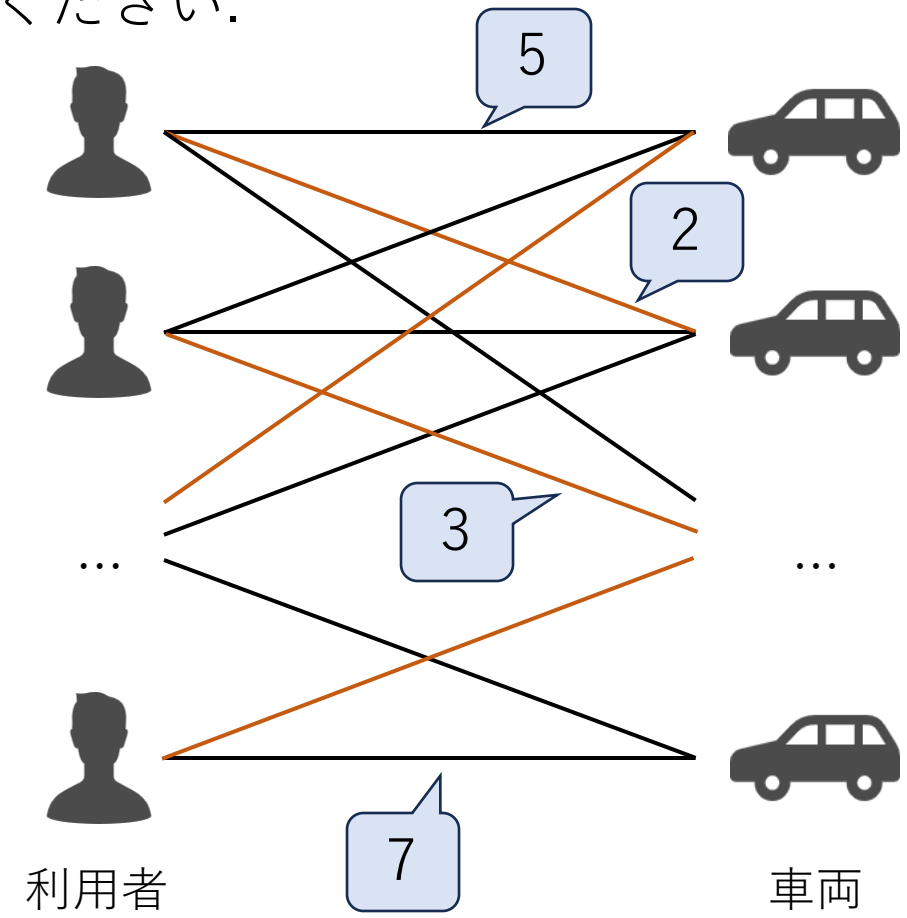
- 全利用者がカーシェア車両に乗車できるようにする
- 利用者は高々1つのカーシェア車両にしか乗れない
- 1台のカーシェア車両には高々1人しか乗れない

①線形計画法に落とし込む

※pulpで解く場合のヒントコードをドライブに載せているので参考にしてもいいです.

②？？？

※ネットワークを工夫し、有名問題に帰着させよう



有名問題とは？

組合せ最適化問題の類型と複雑性クラス

複雑性クラス	$\mathcal{P}$	$\mathcal{NP}$ 完全	$\mathcal{NP}$ 困難
最適化問題	線形最適化問題 最短路問題 最大流問題 最小費用流問題 最小全域木問題 割当問題 充足可能性問題(2-SAT) 最大マッチング問題 最大重みマッチング問題 最大(最小)重み 最大マッチング問題	充足可能性問題(3-SAT) 巡回セールスマン問題 (決定問題) 最大安定集合問題 (決定問題) ナップサック問題 (決定問題) ビンパッキング問題 (決定問題) 最大クリーク問題 (決定問題) 最小頂点被覆問題 (決定問題)	整数最適化問題 巡回セールスマン問題 最大安定集合問題 ナップサック問題 ビンパッキング問題 最大クリーク問題 最小頂点被覆問題 最小極大マッチング問題

決定問題▶最適値ではなく目的関数がある値と比較して大きいかどうかをYes/Noで返す問題

複雑な問題に対しては近似解法やメタヒューリスティクス解法が開発されてきている

Advanced Problem #1: 身近な最適化問題

関心のある問題を最適化問題として立式し、解いてみてください.

＜ヒント / Hint＞

- 目的関数をどう書く？最大化する？最小化する？
- 目的関数はコスト？収入？それとも？
- 何が制約になる？
- 変数をどう設定するか？
  - 連続変数？離散変数？0-1変数（Binary変数）？
  - 1次元の変数？2次元？より高次元？

- 立式する上で…
  - まずは変数を定義しよう。うまく定義できるかどうかが非常に重要です。
  - その変数を用いて、目的関数、制約条件を立式してみましょう。立式が難しければ、変数定義を見直してみましょう。
- 解法もさまざまに考えてみてください！

Advanced Problem #1の例: 徳島県阿南市の避難計画

阿南市領家地区の日開野団地周辺の避難所配置
および避難計画を考えてください.

なお，既存の避難所の場所と収容人数，新たな避難所候補の場所と収容人数，対象敷地の大まかな人口の分布は与えられています.

目的関数

・全員の避難先までの移動距離の最大値の最小化

制約条件

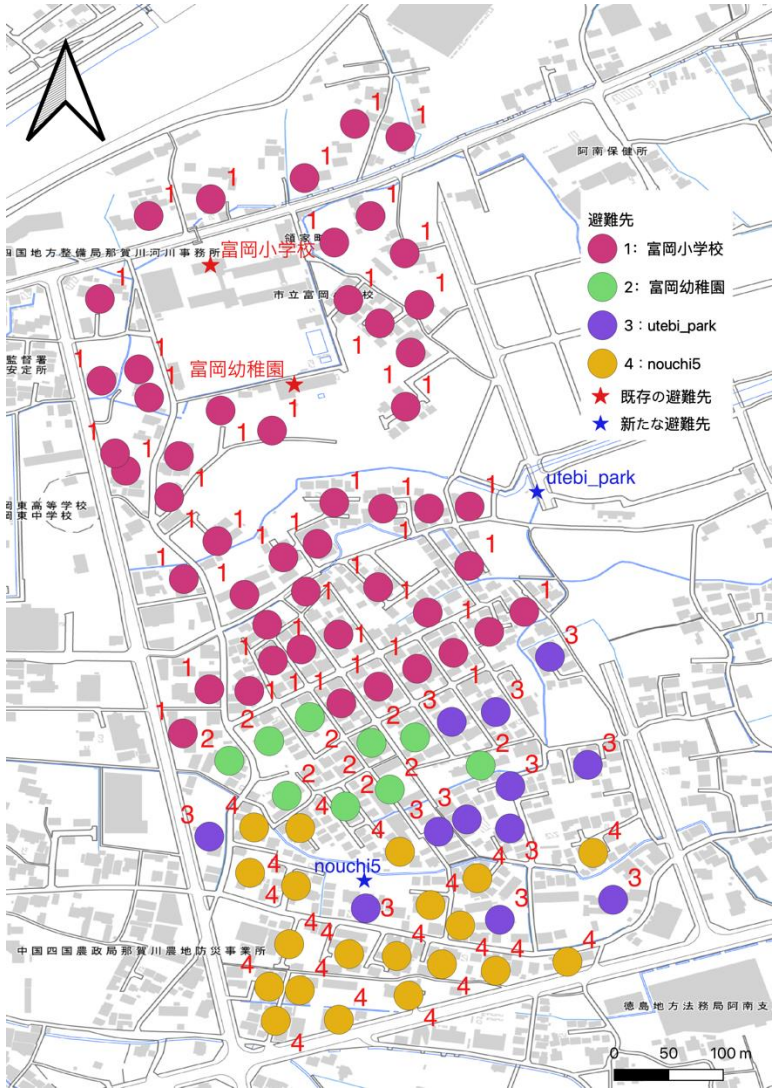
・各住民は1つの避難先に振り分けられる

・設置されない避難所候補には住民が振り分けられない

・新たに設置される避難先の数に上限をかける

・各避難先に避難する住民の総数はキャパシティを超えない

→これを解いたコードがipynbファイルに入っています。



Advanced Problem #2: 罰金付きスケジューリング問題

右の表のように〆切(due)と報酬(reward)が定められたいくつかのタスクがあります。

得られる報酬が最大になるように、 0時から6時までの時間のタスク処理スケジュールを求めてください。

ただし、以下の条件を満たすこと：

- 各タスクを処理するのに1時間がかかる
- 2つのタスクを同時に処理することはできない
- 各タスクがdue時までに（例えばjob aなら2時までに）完了された場合に報酬を獲得することができる。

また、この問題には**貪欲法が存在します**。しかし、条件が変わると貪欲法が存在しなくなります。この点についても考察してみてください！

job	due	reward
a	2	4
b	6	8
c	4	6
d	3	9
e	3	2
f	1	5
g	5	6
h	5	7
i	2	4
j	1	2

まとめ

✅ **決定変数はなにか？**

- 離散 or 連続

✅ **目的関数はなにか？**

- 線形 or 非線形 or 2次関数

✅ **制約条件はなにか？**

↓

✅ 問題規模や厳密・近似の要請に合わせた解法の選択

✅ より性質の良い問題構造 (凸)に直せたり、別の有名問題に帰着できないか