アルゴリズム論

平成22年11月15日

最短路問題

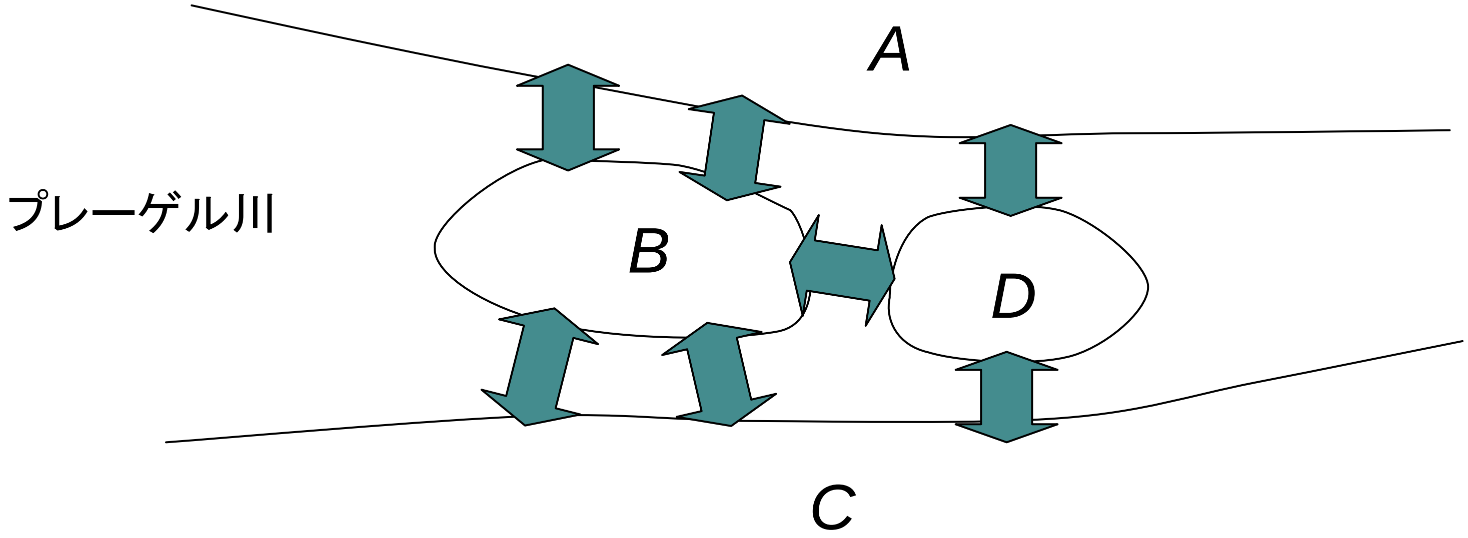
ダイクストラ：E. W. Dijkstra

- 単一始点最短路問題のアルゴリズム 1959年
  - 負の重みを許す単一始点最短路問題（負の重みの閉路があれば検出）Bellman-Ford法
  - 全点対最短路問題は Warshall-Floyd法 1962年
- いずれも多項式時間アルゴリズム
  - Dijkstra法：$O(n^2)$
  - Bellman-Ford法：$O(ne)$
  - Warshall-Floyd法：$O(n^3)$

今日の話題は最短路問題

グラフ理論の誕生
Königsberg’s Bridge Problem

ケーニヒスベルグに七つの橋がありました

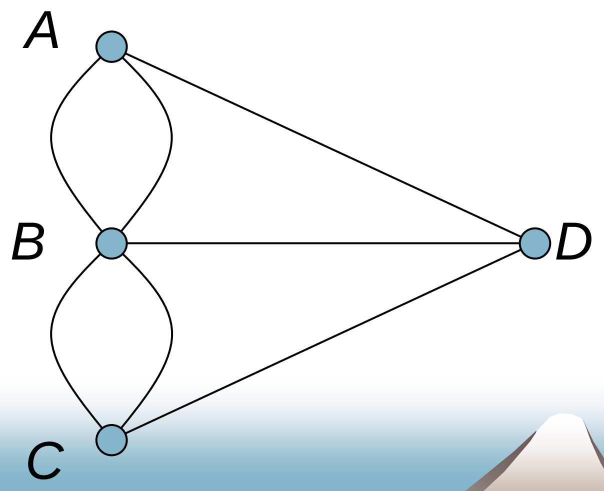


*A*にある家を出て，七つの橋を<u>ちょうど一回ずつ</u>渡り，
家へ戻ってくるにはどうすればよいか？

おいらが解決

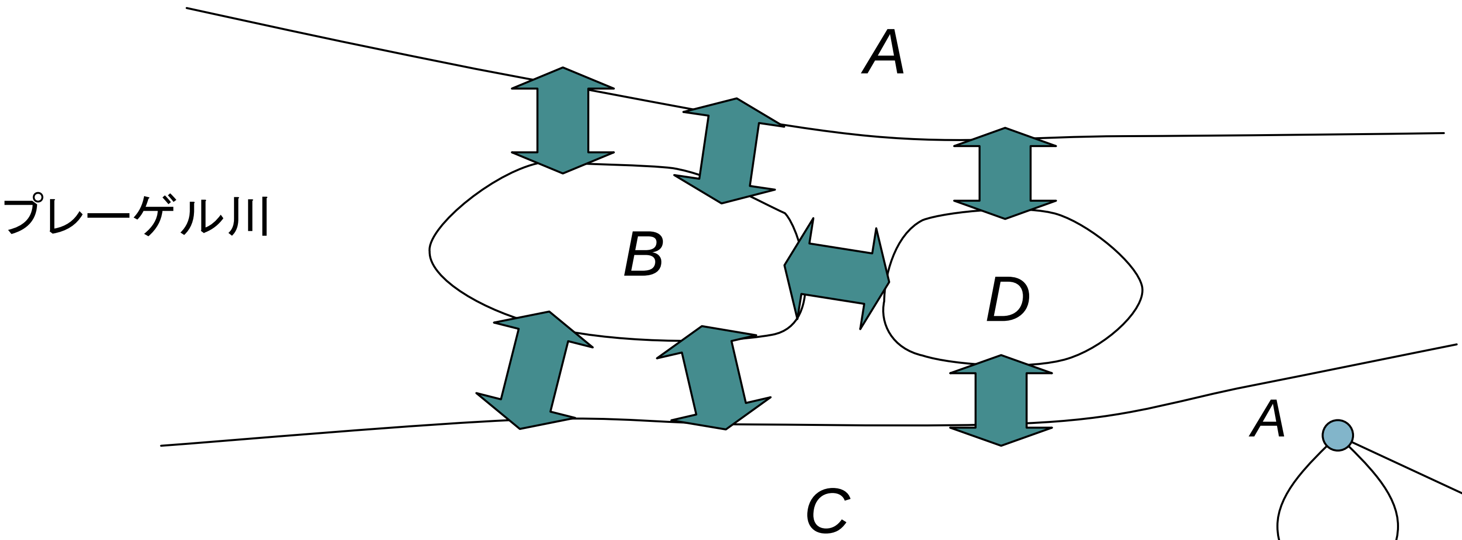
- Euler (1707-1782) became the father of graph theory when in 1736 he settled a famous unsolved problem of his day called the Königsberg's Bridge Problem.

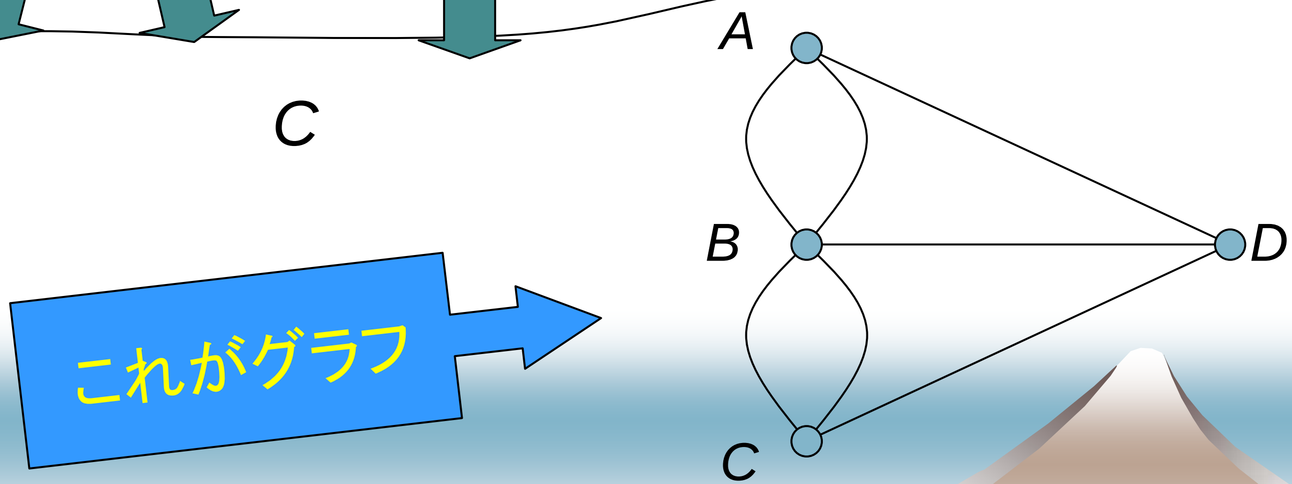
- 右の図は一筆書きできないから，そのような散歩道は存在しない。



問題の本質をグラフで表現

それぞれの橋はどの地点をつないでいるか



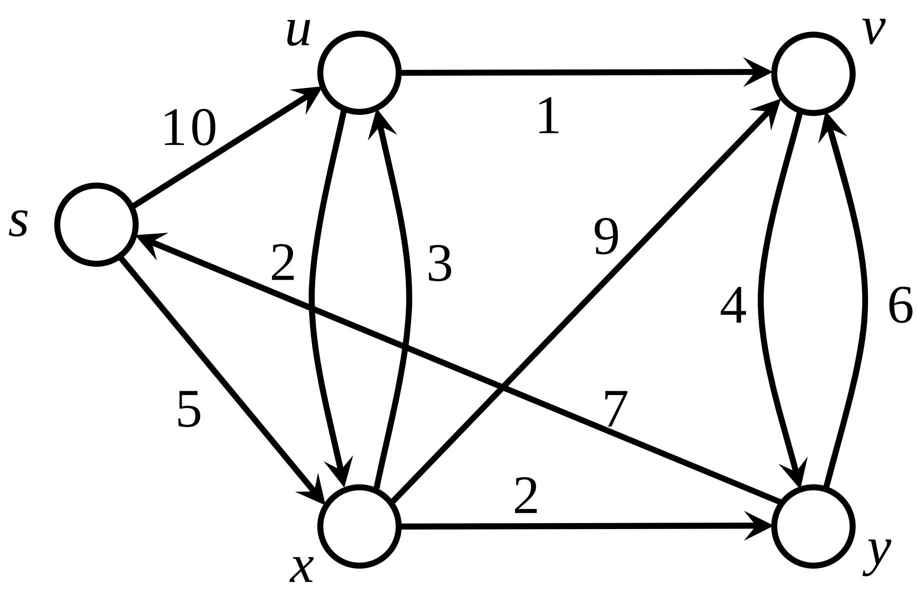


ネットワーク＝重み付きグラフ

有向グラフ $G=(V,E)$

各辺 $(u,v)\in E$ に

　重み $w(u,v)=$『長さ』



　経路の長さ

＝構成する辺の長さの和

$$w(<s,x,u,v>)=w(s,x)+w(x,u)+w(u,v)=5+3+1=9$$

$\delta(u,v):u$ から v への最短路の長さ

$$\delta(s,u)=8,\ \ \delta(s,v)=9,\ \ \delta(s,x)=5,\ \ \delta(s,y)=7$$

最短路問題

- 単一始点最短路問題　← 今日の話題
  single-source shortest-paths problem
- 単一目的地 SPP　single-destination SPP
- 単一点対 SPP　single-pair SPP

- 全点対間最短路問題
  all-pairs shortest-paths problem

頂点のラベル

$d[v]$: s から v への最短路の長さの推定値

```
dist = [INF  for k in range(N)]
```

N は頂点数，INF は大きな値

```
dist[s] = 0
```

$\pi[v]$: s から v への最短路の先行点の推定値

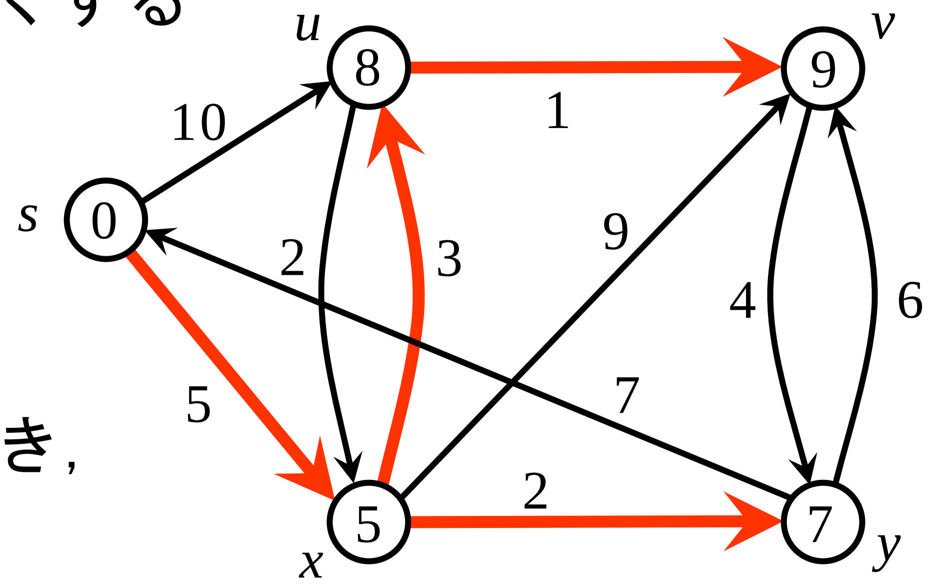
```
prev = [-1  for k in range(N)]
```

頂点のラベル

$d[v]$：頂点 v の中に書く

$\pi[v]=u$ のとき辺 (u,v) を太くする

根付き木：太い辺の集まり

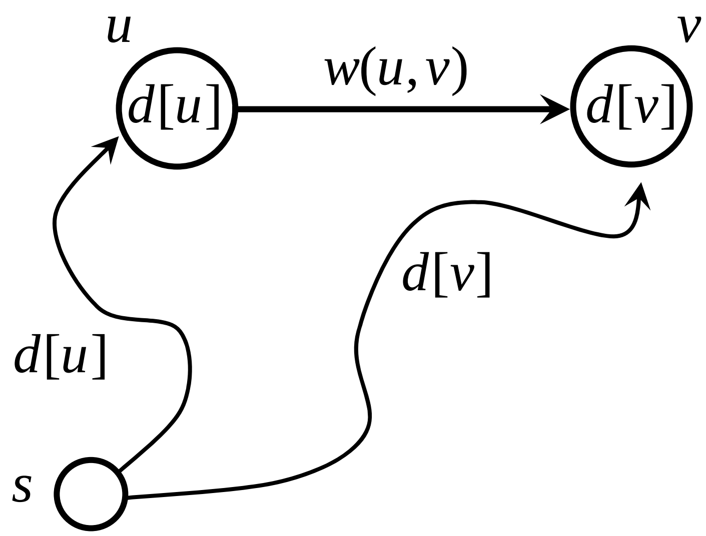


右の図は最終状態。このとき，

- $d[v]$ は最短路の長さ
- 各 v から $\pi[v]$ を逆にたどれば
  s からの最短路になっている

緩和操作

RELAX (u, v)
  if $d[v] > d[u] + w(u,v)$:
    $d[v] = d[u] + w(u,v)$
    $\pi[v] = u$

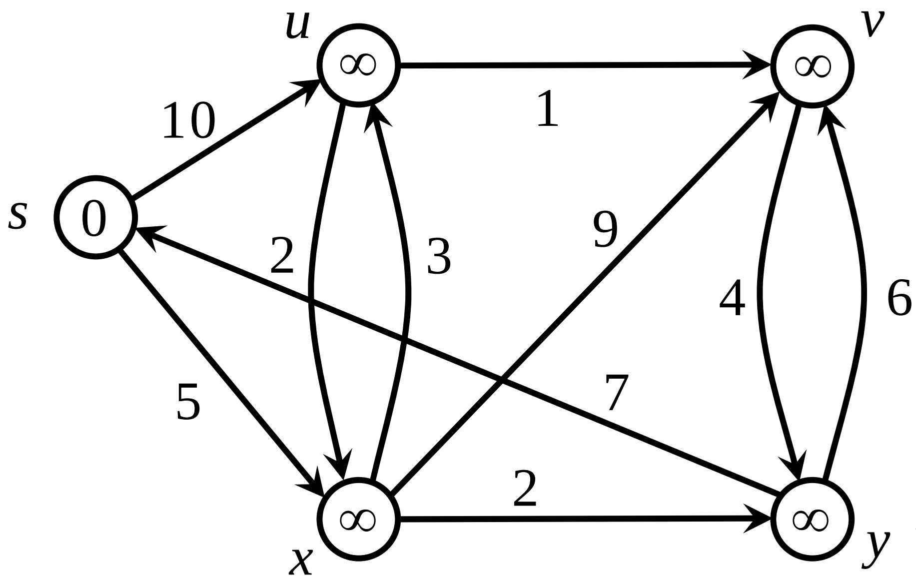


RELAX (u,v) は，辺 (u,v) を通ることで，
これまでの推定値を改善できるかを調べる

Dijkstra 法の例題

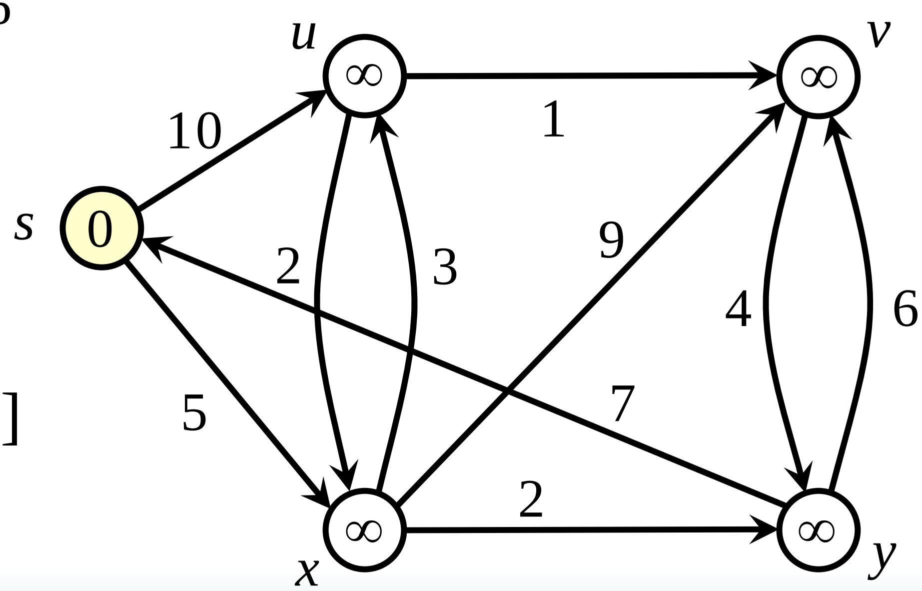
○ キュー Q の要素

○ 集合 S の要素



初期状態

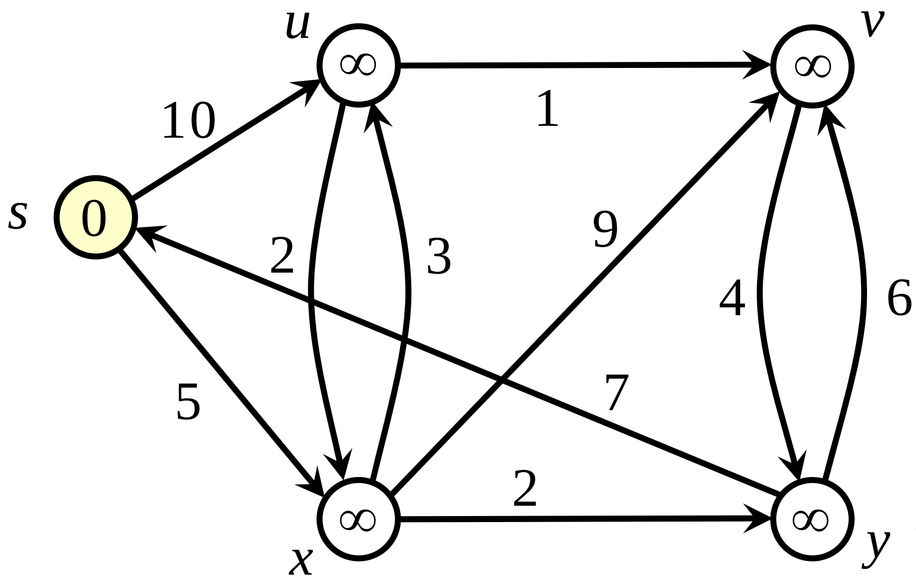
$$Q=[\boxed{s:0,}u:\infty,v:\infty,x:\infty,y:\infty]$$



$$S=\{s\},\ u=s$$

$$Q=[u:\infty,v:\infty,x:\infty,y:\infty]$$

Dijkstra 法の例題

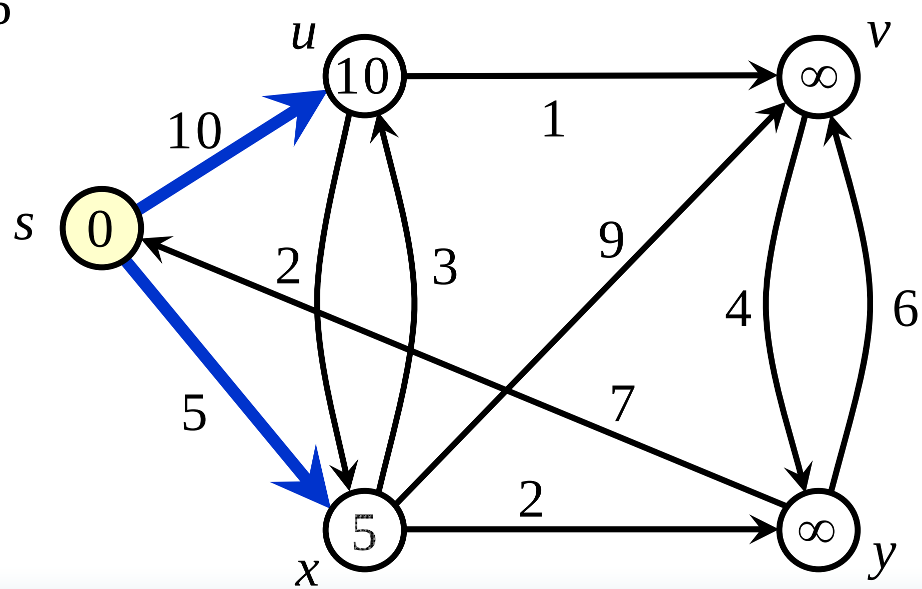


$$S = \{s\},\ u = s$$

$$Q = [u:\infty, v:\infty, x:\infty, y:\infty]$$

○ キュー Q の要素

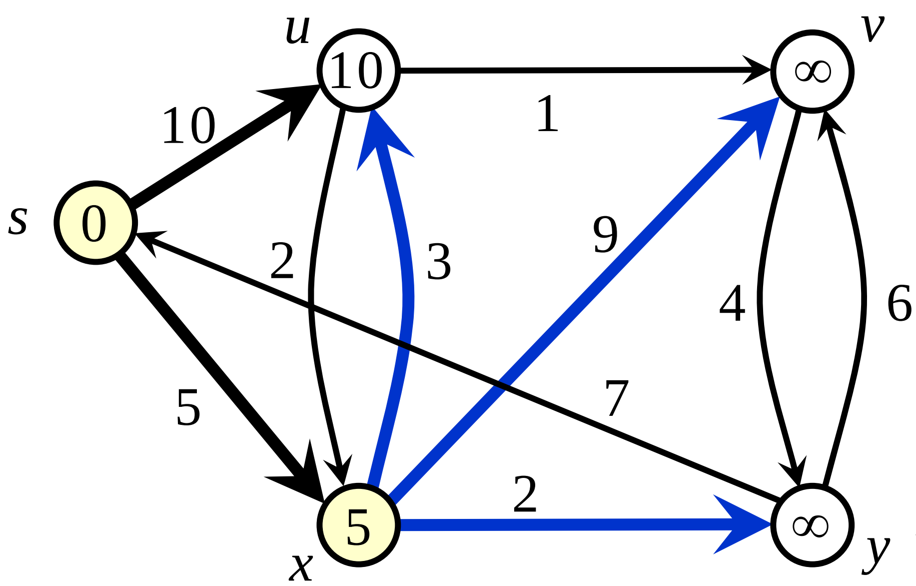
● 集合 S の要素



after RELAX(s, x), (s, u)

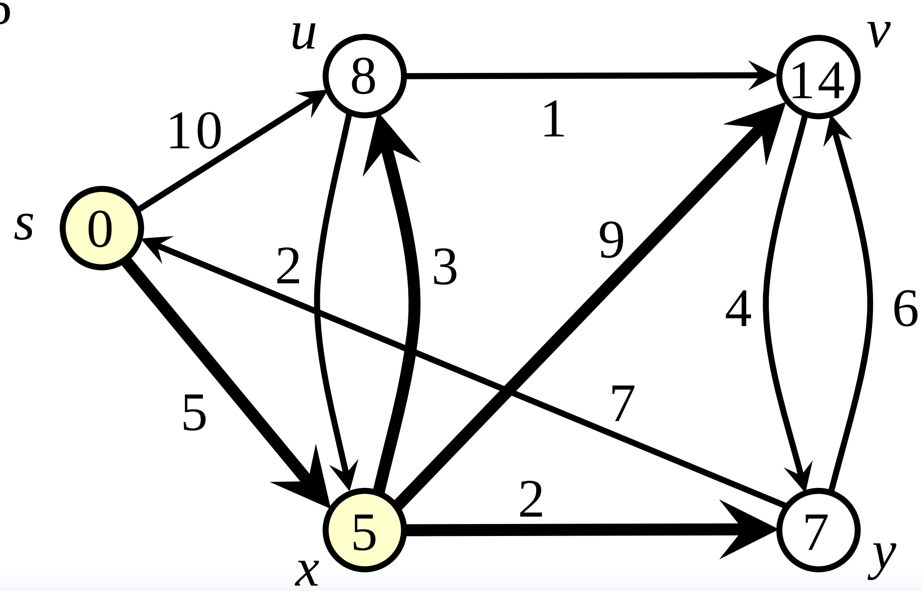
Dijkstra 法の例題

○ キュー Q の要素
○ 集合 S の要素



$Q=[\boxed{x:5,}u:10, v:\infty, y:\infty]$

RELAX (x, u) , (x, v) , (x, y)



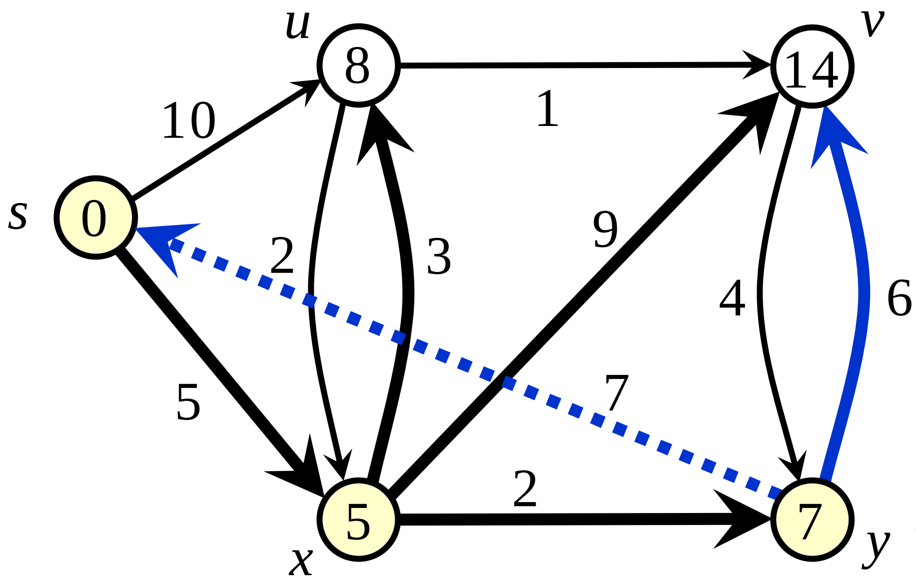
after RELAXs

$Q=[y:7, u:8, v:14]$

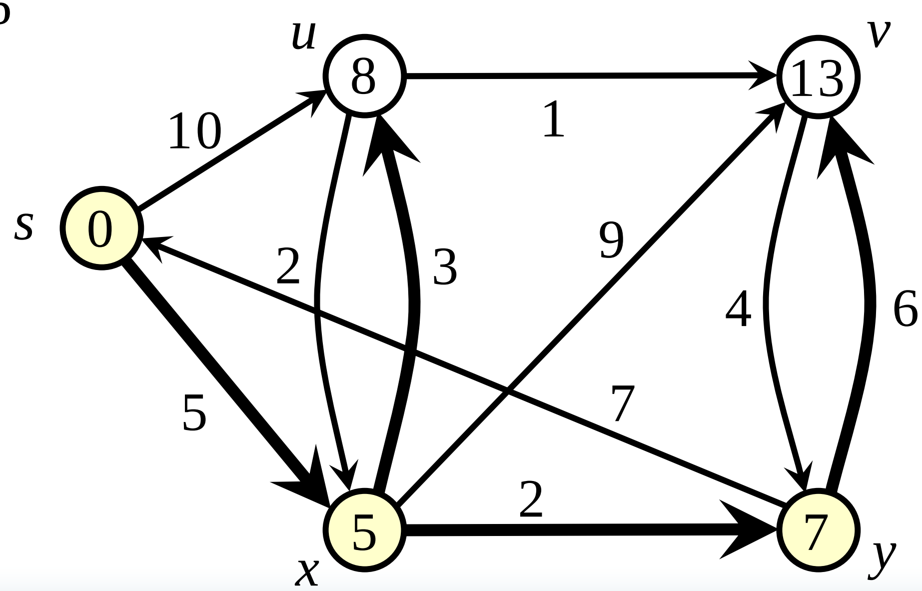
Dijkstra 法の例題

キュー Q の要素

集合 S の要素

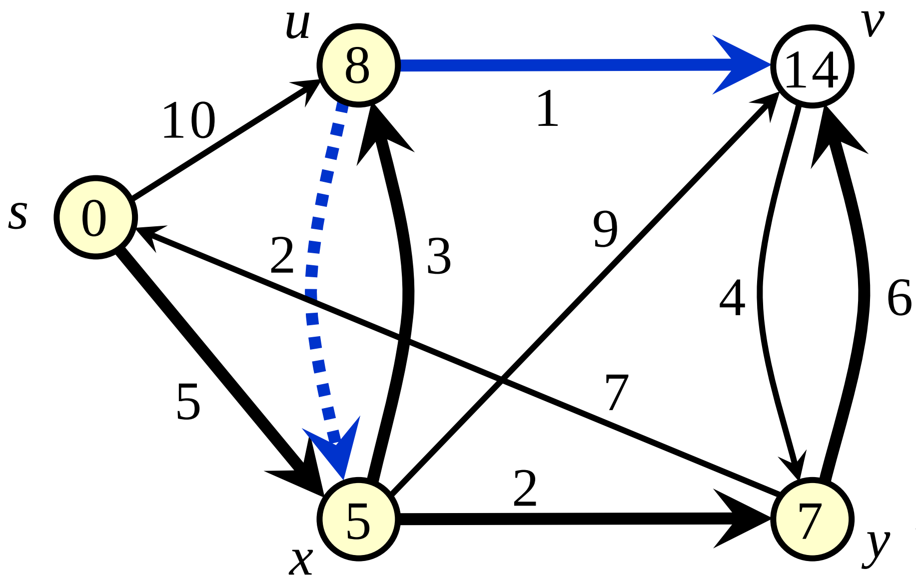


before RELAXs

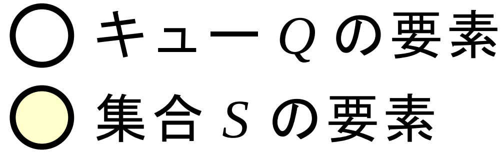


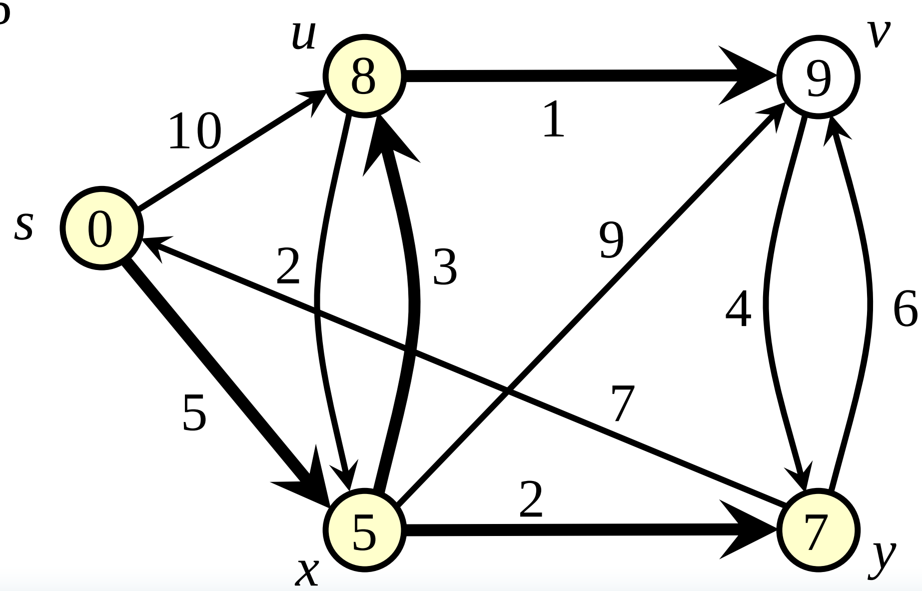
after RELAXs

Dijkstra 法の例題



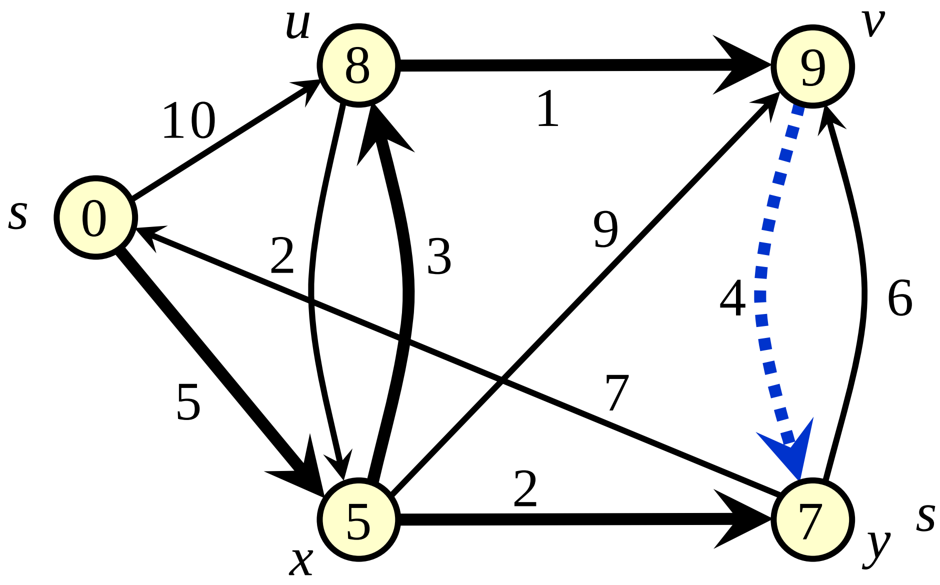
before RELAXs





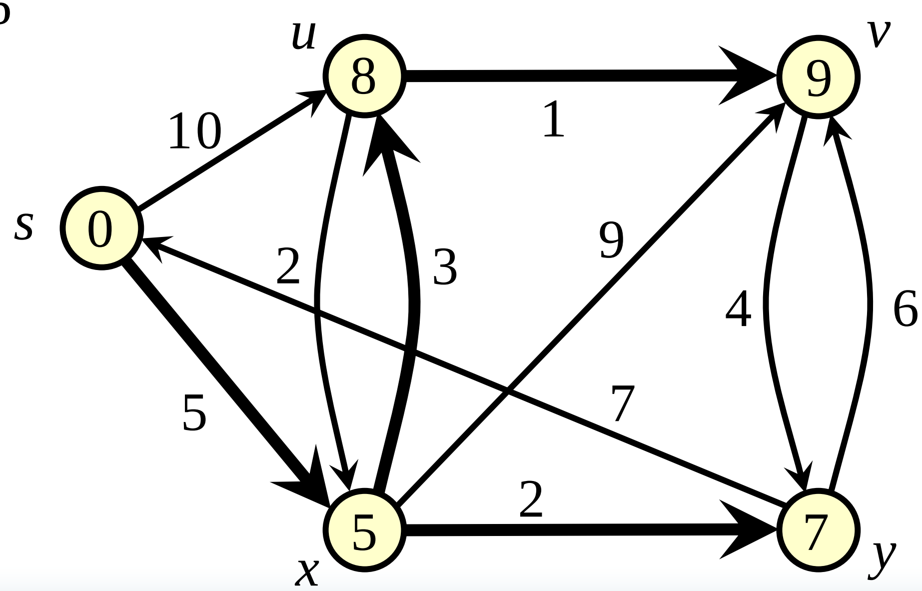
after RELAXs

Dijkstra 法の例題



before RELAX

キュー Q の要素
集合 S の要素



最終状態

Dijkstra 法

- 辺の重みが非負の単一始点最短路問題

```
def Dijkstra(adjL, s):  # adjL はグラフの隣接情報, s は始点
    queue = range(len(adjL))
    dist = [INF for k in queue]   # INF = 999999  グローバル定数
    prev = [ -1 for k in queue]
    dist[s] = 0
    while queue != []:            # queue にある頂点は未確定
        u = popMin(queue, dist)   # queue にある dist の最小頂点
        for v, weight in adjL[u]:
            if v not in queue:  continue
            if dist[v] > dist[u] + weight:        # RELAX(v, u)
                dist[v], prev[v] = (dist[u] + weight, u)
    return (dist, prev)
```

グラフの情報

G1 = [[1, 2, 1], [1, 4, 5], [1, 8, 3], [2, 3, 2], [2, 4, 3], [3, 4, 1], [3, 5, 5],
[4, 8, 2], [4, 5, 2], [5, 6, 6], [5, 7, 3], [6, 3, 1], [7, 6, 4], [8, 7, 8]]

G1 の頂点名リスト : [1, 2, 3, 4, 5, 6, 7, 8]

G1 の adjL : 頂点番号は 0 ～ 7

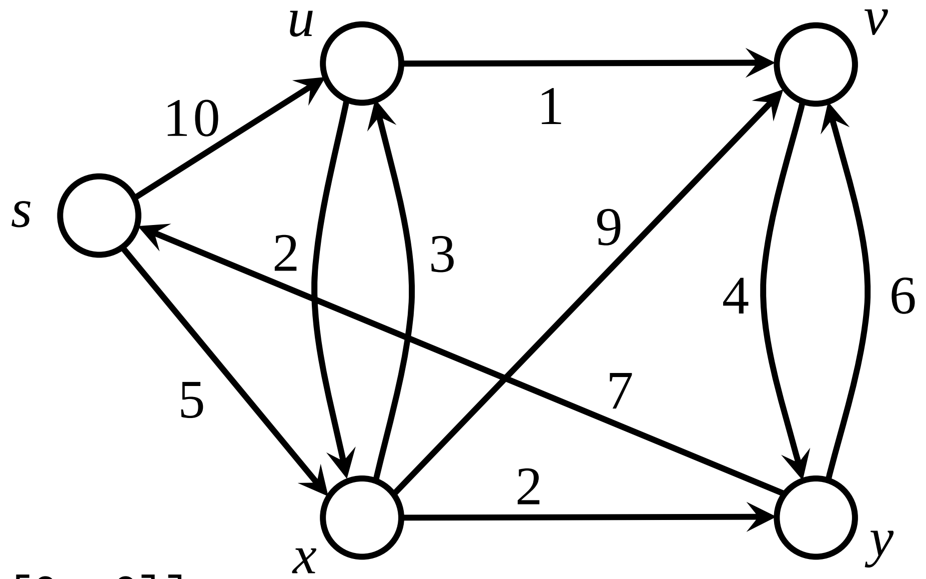
```
[[[1, 1], [3, 5], [7, 3]],    [[2, 2], [3, 3]],
 [[3, 1], [4, 5]],            [[7, 2], [4, 2]],
 [[5, 6], [6, 3]],            [[2, 1]],
 [[5, 4]],                    [[6, 8]]            ]
```

adjL[1] = [[2, 2], [3, 3]] ← $w(2, 3) = 2, w(2, 4) = 3$

adjL[7] = [[6, 8]] ← $w(8, 7) = 8$

グラフの情報

```
G2 = [['s','u',10],['s','x',5],['u','v',1],['u','x',2],
      ['v','y',4],['x','u',3],['x','v',9],['x','y',2],
      ['y','s',7],['y','v',6]]
```



G2 の頂点名リスト：

```
['s', 'u', 'v', 'x', 'y']
```

G2 の adjL：

```
[[[1, 10], [3, 5]],    [[2, 1], [3, 2]],
 [[4, 4]],             [[1, 3], [2, 9], [4, 2]],
 [[0, 2], [2, 6]]   ]
```

枝情報 → 隣接情報

```
def edge2adj(edgeG):
    """ グラフの枝情報 G から隣接リスト adjL を作る """
    vrtxL = []
    for u, v, w in edgeG:     # edgeG の各要素を (u, v, w) とする
        if u not in vrtxL:    vrtxL.append(u)
        if v not in vrtxL:    vrtxL.append(v)
    vrtxL.sort()
    adjL = [[] for k in range(len(vrtxL))]
    for u, v, weight in edgeG:
        adjL[vrtxL.index(u)].append([vrtxL.index(v), weight])
    return (adjL, vrtxL)
```

Dijkstra 法

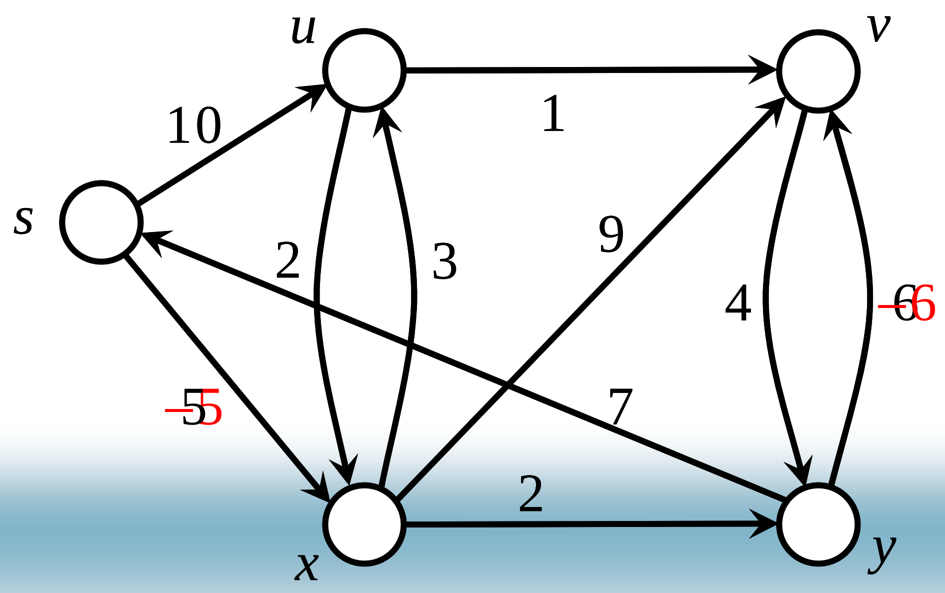
```
INF = 999999

def Dijkstra(adjL, s):  # adjL はグラフの隣接情報, s は始点
    queue = range(len(adjL))
    dist = [INF for k in queue]
    prev = [ -1 for k in queue]
    dist[s] = 0
    while queue != []:                  # queue にある頂点は未確定
        u = popMin(queue, dist)         # queue にある dist の最小頂点
        for v, weight in adjL[u]:
            if v not in queue:  continue
            if dist[v] > dist[u] + weight:          # RELAX(v, u)
                dist[v], prev[v] = (dist[u] + weight, u)
    return (dist, prev)
```

queue から最小値を pop する

```
def popMin(queue, dist):
    indMin, distMin = (0, dist[queue[0]])  # 仮の最小値
    for k in range(1, len(queue)):
        if distMin > dist[queue[k]]:
            indMin, distMin = (k, dist[queue[k]])
    return queue.pop(indMin)
#  queue.pop(indMin) は，queue[indMin] を消去し，
#                           かつ，消去した値を返す
```

Bellman-Ford 法

- 単一始点最短路問題
- 辺の重みが負であってもよい
- 負の重みをもつ閉路はない



Bellman-Ford 法

$|V(G)| = n,\ |E(G)| = e$

とすると $O(ne)$ の時間で

実行できる

```
def BellmanFord(edgeIn, adjL, s):
    N = len(adjL)
    dist = [INF for k in range(N)]
    prev = [ -1 for k in range(N)]
    dist[s] = 0
    for k in range(N - 1):
        for u, v, weight in edgeIn:
            if dist[v] > dist[u] + weight:
                dist[v] = dist[u] + weight
                prev[v] = u
    for u, v, weight in edgeIn:
        if dist[v] > dist[u] + weight:  # 負の重みの閉路がある
            return ([], [])
    return (dist, prev)
```

$N-1$ 回 繰返す

枝情報・経路情報

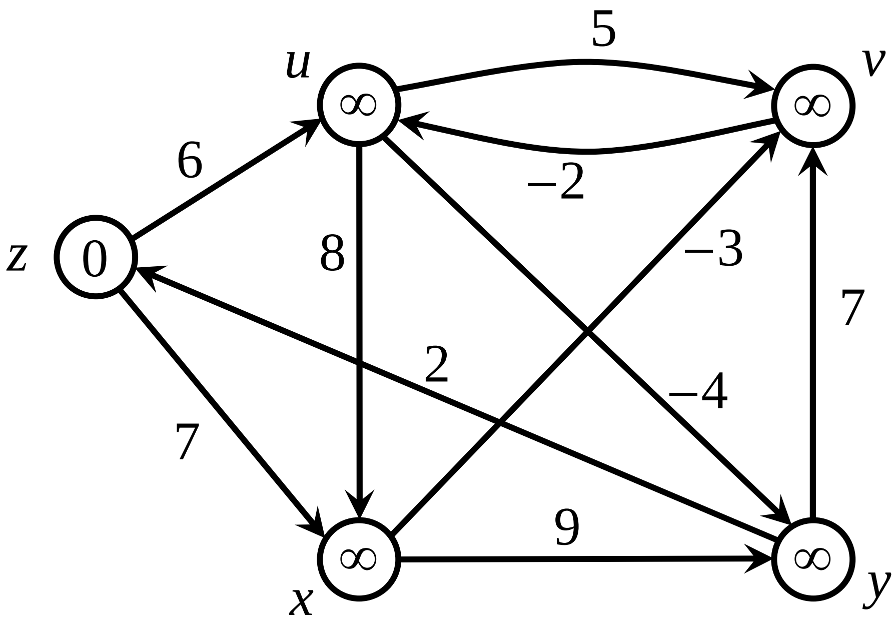
◎ 枝情報：edgeG → edgeIn

```
edgeIn = []
for u, v, w in edgeG:
    edgeIn.append([vrtxL.index(u), vrtxL.index(v), w])
```

◎ s から t への経路情報：prevL → path

```
last = prevL[t]            # last は頂点番号。t は終点番号
path = [vrtxL[last]]       # vrtxL[last] はその頂点名
while last != s:           # s は始点番号
    last = prevL[last]
    path.append(vrtxL[last])    # path には頂点名を入れる
path.reverse()
path.append(vrtxL[t])      # vrtxL[t] は終点名
```

Bellman-Ford 法の例題



頂点 z が始点
辺は辞書式順序で走査する

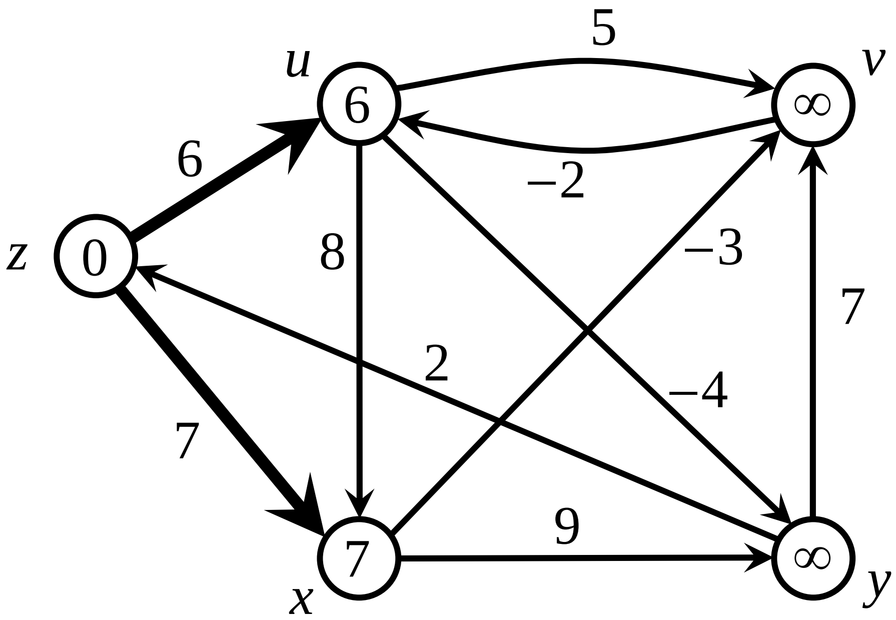
$(u,v),(u,x),(u,y),(v,u),$
$(x,v),(x,y),(y,v),(y,z),$
$(z,u),(z,x)$

$\infty > \infty + 5,\ \infty > \infty - 5$ などは不成立

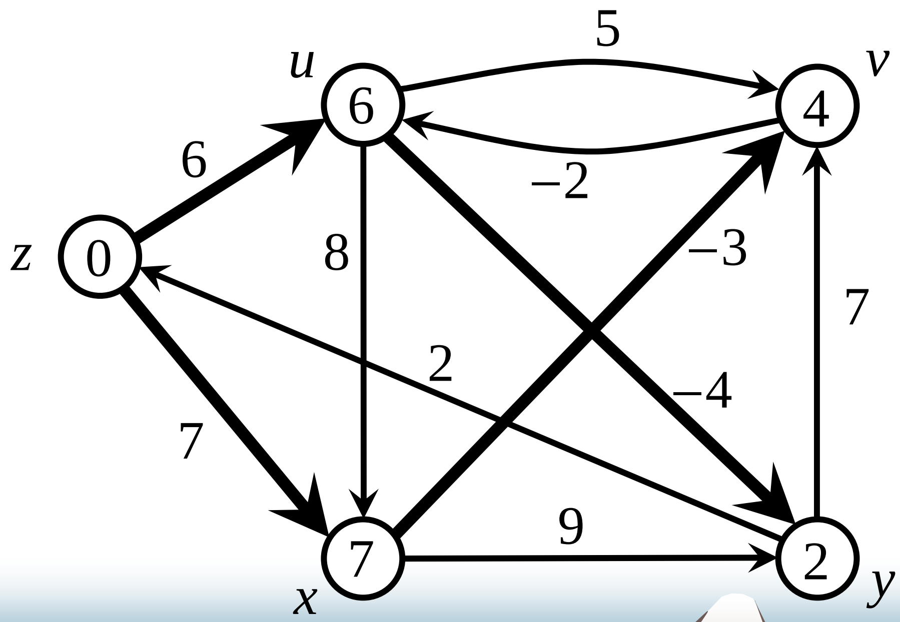
Bellman-Ford 法の例題

$$(u,v),(u,x),(u,y),(v,u),(x,v),(x,y),(y,v),(y,z),(z,u),(z,x)$$

１回目の走査が終了した状態



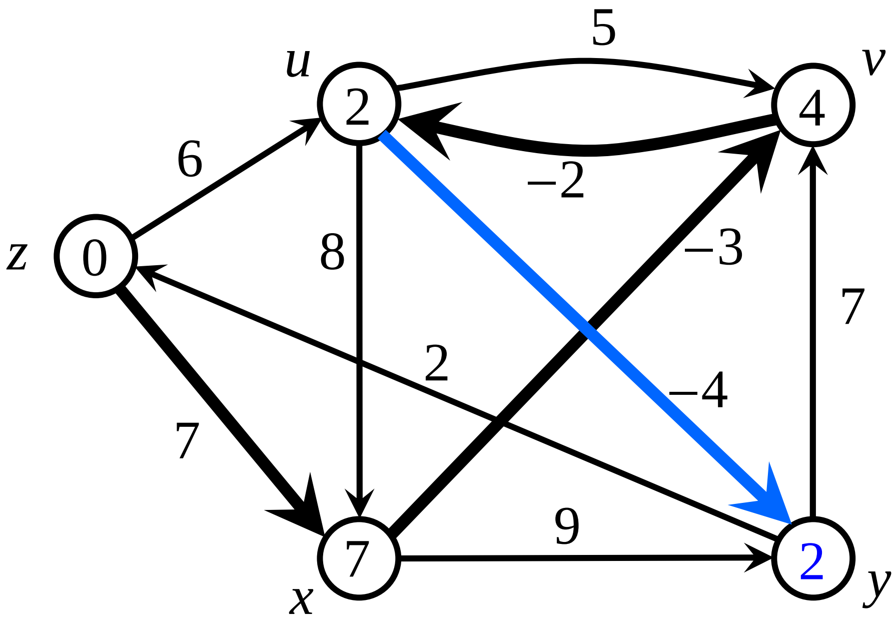
２回目の走査が終了した状態



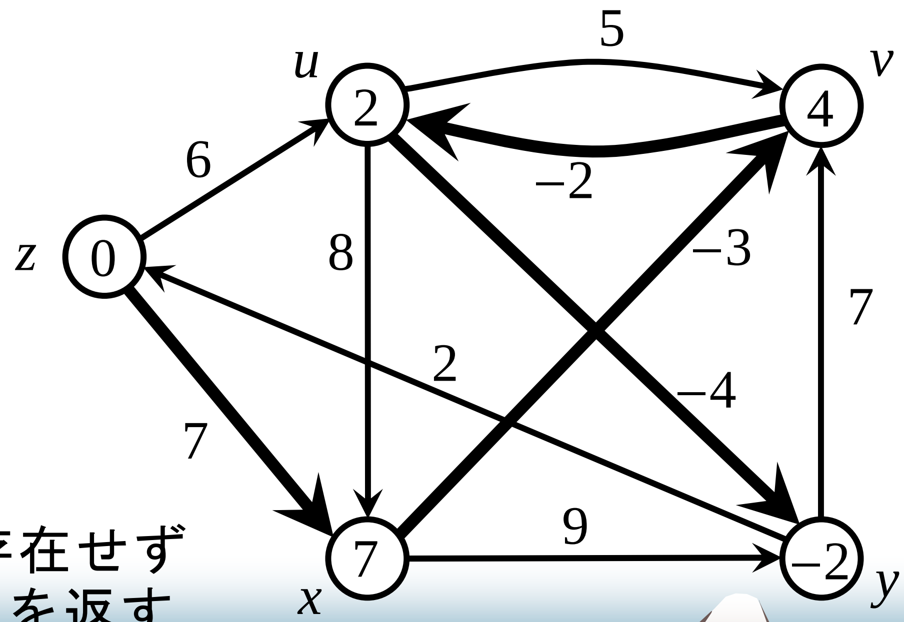
Bellman-Ford 法の例題

$$(u,v),(u,x),(u,y),(v,u),(x,v),(x,y),(y,v),(y,z),(z,u),(z,x)$$

３回目の走査が終了した状態



４回目の走査が終了した
最終状態



負の閉路は存在せず
dist と prev を返す

課題 sp1

- ［課題 sp1］単一始点最短路問題を解くプログラムを作成せよ。長さが非負なら Dijkstra, そうでなければ BellmanFord を使うこと。
  - グラフ情報：
    ```
    G2 = [['s','u',10],['s','x',-5],['u','v',1],
          ['u','x',2],['v','y',4],['x','u',3],
          ['x','v',9],['x','y',2],['y','s',7],
          ['y','v',6]]
    ```
  - 始点を指定：例えば 's'
  - 出力：各頂点への距離のリスト dist
    各頂点の先行点のリスト prev

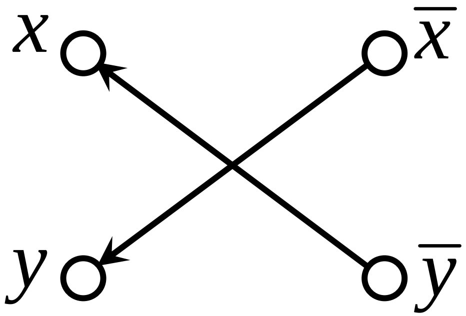
課題 sp2

- ［課題 sp2］グラフ情報と始点および終点を指定したら，その最短路の長さおよび道順を出力するプログラムを作れ。
  条件は［課題 sp1］に準ずる。
  - 実行例：

```
>>> howFar(G1, 1, 8)
(3, [1, 8])
>>> howFar(G1, 1, 7)
(9, [1, 2, 4, 5, 7])
>>> howFar(G2, 's', 'v')
(9, ['s', 'x', 'u', 'v'])
>>> howFar(G3, 'z', 'y')
(-2, ['z', 'x', 'v', 'u', 'y'])
```

2充足可能性問題

$$
\begin{aligned}
[x \vee y = 1] &\equiv [x = 0 \Rightarrow y = 1] \wedge [y = 0 \Rightarrow x = 1] \\
&\equiv [\overline{x} = 1 \Rightarrow y = 1] \wedge [\overline{y} = 1 \Rightarrow x = 1]
\end{aligned}
$$

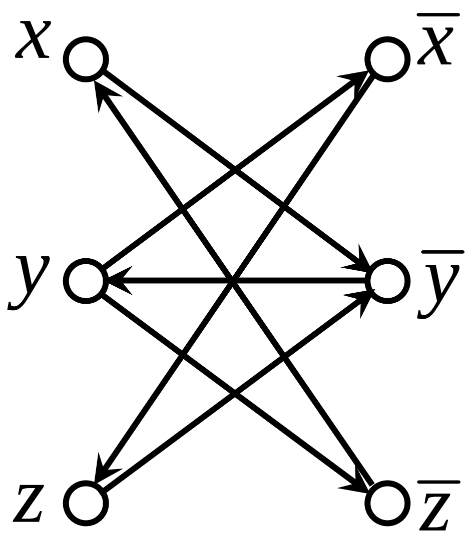


$$x = x \vee x$$

$$
\begin{aligned}
[x \vee x = 1] &\equiv [\overline{x} = 1 \Rightarrow x = 1] \wedge [\overline{x} = 1 \Rightarrow x = 1] \\
&\equiv [\overline{x} = 0] \equiv [x = 1]
\end{aligned}
$$

例えば

$$y(\bar{x}\vee\bar{y})(\bar{y}\vee\bar{z})(z\vee x)$$



x から $\bar{x}$ への道があり
かつ, $\bar{x}$ から x への道がある

$\Downarrow$

充足不可能

$x \Rightarrow \bar{x},\ \bar{x} \Rightarrow x$ のとき：充足不可能

$x \not\Rightarrow \bar{x},\ \bar{x} \Rightarrow x$ のとき：$x=1$

$x \Rightarrow \bar{x},\ \bar{x} \not\Rightarrow x$ のとき：$x=0$

$x \not\Rightarrow \bar{x},\ \bar{x} \not\Rightarrow x$ なら：
x の値は don't care

課題 sp3

- ［課題 sp3］２充足可能性問題 2SAT を解くプログラム twoSAT を Dijkstra を用いて実現せよ。
  - 実行例：

```
>>> A = [[10], [21, -3], [-21, 3], [-10, 3]]
>>> twoSAT(A)
3: 1
10: 1
21: 1
>>> A.append([-10,-21])
>>> A
[[10], [21, -3], [-21, 3], [-10, 3], [-10, -21]]
>>> twoSAT(A)
3: impossible
```