Scratchによる
アルゴリズム入門

第6回　最短経路問題

2017.04.03

鎌倉シチズンネット（KCN）

最短経路問題(1)

目的地への最短ルートを求める、ダイクストラの「最短経路問題」のアルゴリズムです（P.12以降の「補足説明」を参照）。
乗り換え案内のサイトやカーナビの経路選択、ルータというネットワーク機器（ OSPF というルーティングプロトコル）でも使われています。


Edsger W. Dijkstra
Turing award 1972

（ダイクストラは構造化プログラミングの提唱者としても知られています）

最短経路問題は、平成15年秋の基本情報技術者試験に出題されました。

平成15年 秋期　基本情報技術者 午後 問04

最短経路問題(2)

基本情報技術者試験のアルゴリズム問題では、アルゴリズムの記述に「擬似言語」と呼ばれる言語を使用しています。この問題で用いられている擬似言語とScratch の対応関係を以下に示します。

☆代入

擬似言語	Scratch
・ X ← 1	

☆繰り返し

擬似言語	Scratch
X ≦N	

最短経路問題(3)

☆条件判定

擬似言語　　Scratch

P[Y] = 0

☆その他	擬似言語	Scratch
二次元配列の参照	C [x, y]	C [(x - 1)*n + y] で代用
比較式	＝　、≠、≦	＝　、ではない、> ではない
論理演算子	and	かつ
出力用の関数	Output(　)	ネコにしゃべらせる
最大の数	∞	

最短経路問題(4)

☆配列の意味

C [i, j] ：地点 i から地点 j までの距離。直接行けない場合は9999（最大値）
Scratchには2次元配列がないため、C [(i - 1) * n + j] で代用

D [i] ： 地点1（始点）から地点 i までの最短距離

P [i] ： 地点 i が確定（処理済に）したとき 1 にする

S [i] ： 地点 i に至る経路上で地点 i の直前の地点

W [i] ：経路を出力する際に用いる作業用の配列

☆変数の意味

n： 地点の数（この例では 5 ）

x： 繰返しのカウンタ

y： 地点の番号

z： 始点からの最短距離

t： 処理済にする地点の番号

最短経路問題(5)

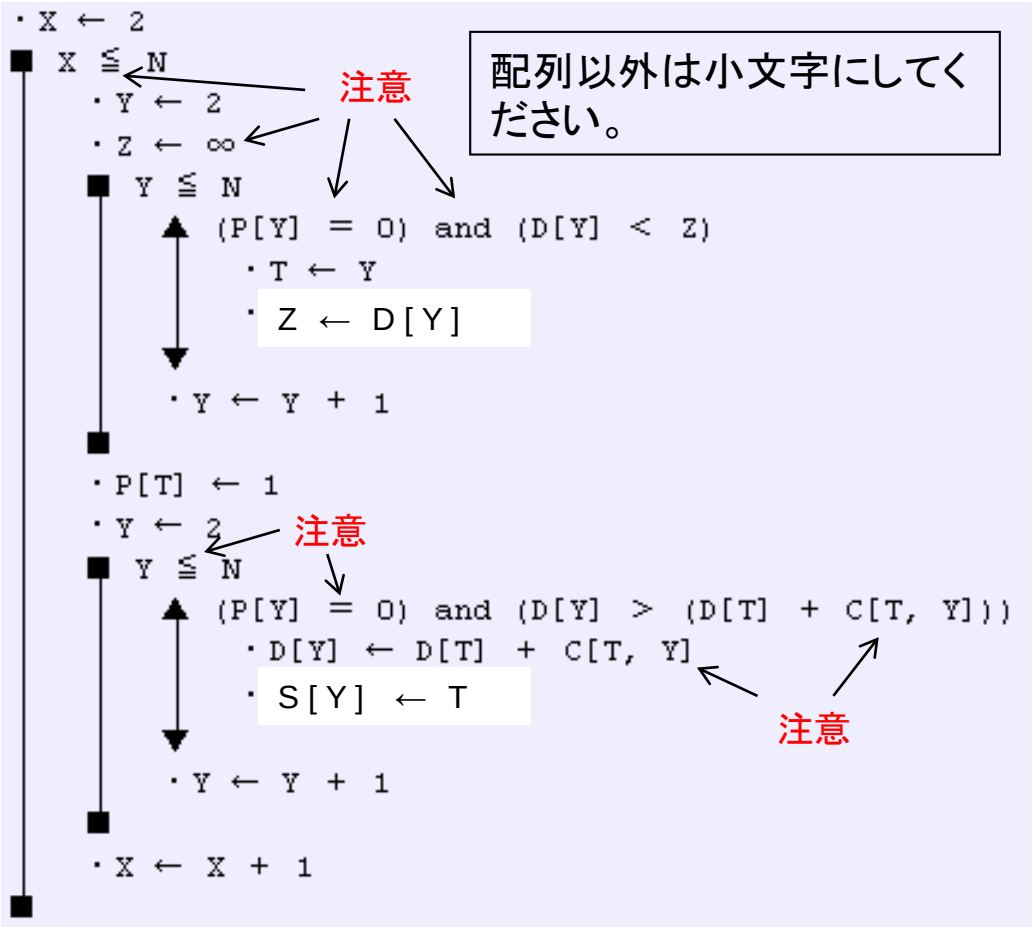
{ 初期設定 }

```
・x ← 1
■ x ≦ n                    /* 地点の数だけ以下の処理を繰り返す */
   ・D [ x ] ← C [ 1, x ]  /* 始点からの距離を設定する */
   ・P [ x ] ← 0           /* 未処理（未確定）の印をつける */
   ・S [ x ] ← 1           /* 経路上ひとつ前の地点を始点にする */
   ・x ← x + 1             /* 次の地点に移る */
■
・P [ 1 ] ← 1              /* 始点を処理済み（確定）にする  */
```

（ 赤字 はScratch 変換時に注意すべき点 ）

最短経路問題(6)

{ 最短経路探索処理 }



下記の処理 ① ～ ④ を，全地点が処理済になるまで繰り返す。

① 処理済でない（P[Y] ＝ 0）すべての地点のうち，始点からの距離（D[Y] ）が最小である地点 T を選ぶ。

② 地点 T を処理済（P[T] ＝ 1）とする。

③ 処理済でない（P[Y] ＝ 0）すべての地点に対して，地点Tを経由した方が短ければ（D[T] ＋ C[T, Y] の値が D[Y] の値より小さければ），D[T] ＋ C[T, Y] の値で D[Y] を置き換える。

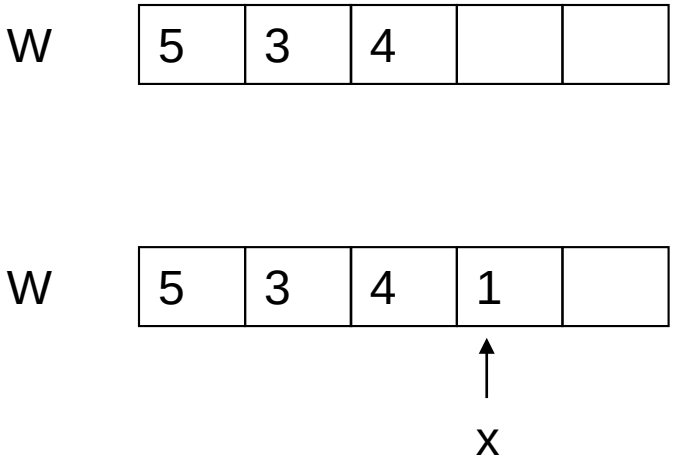
④ ③で D[Y] を置き換えた場合，S[Y] に T を代入する。

最短経路問題(7)

{ 最短経路表示処理 }

- x ← 1
- y ← n
- y ≠ 1
  - W [x] ← y
  - y ← S [y]
  - x ← x + 1
- W [x] ← y
- x > 0
  - Output (W [x])
  - x ← x - 1

配列 W に、終点から始点に向かって、1つ前の地点の番号を格納していく。



最短経路問題(8)

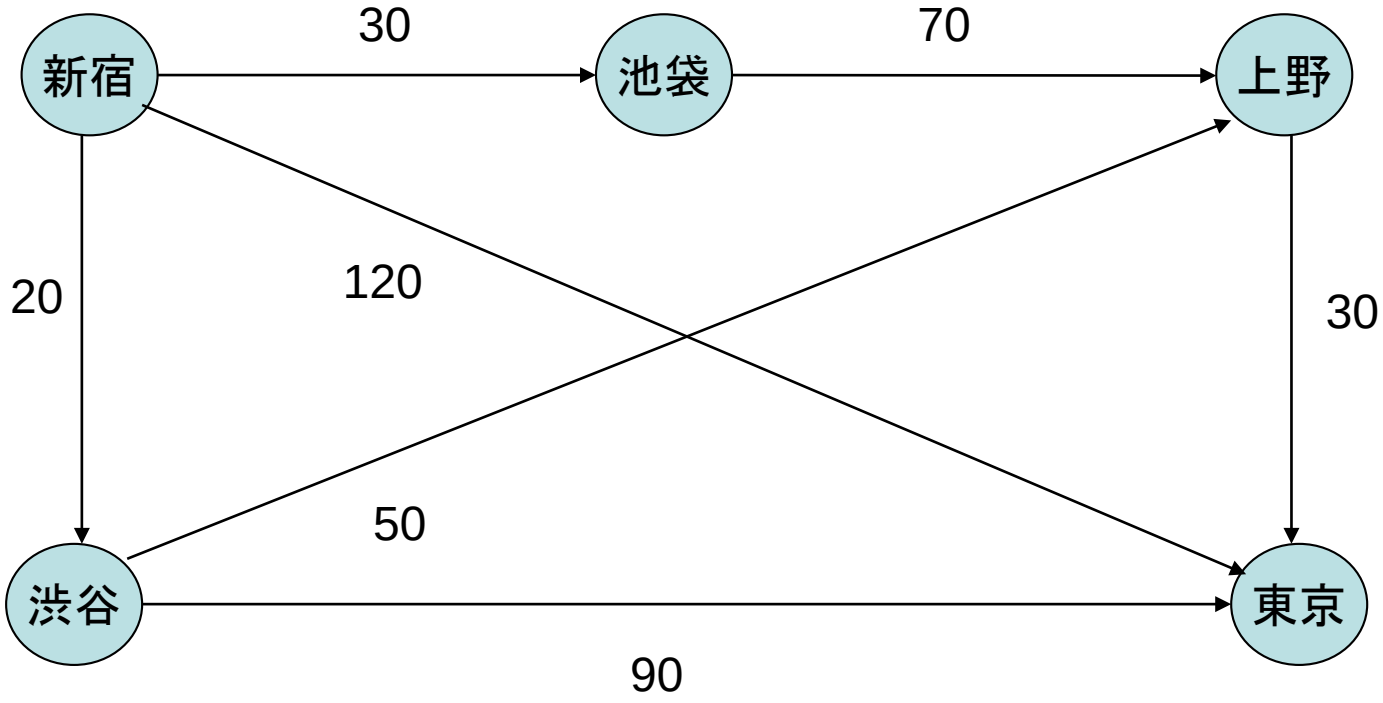
[問題]

次のプロジェクトをリミックスして、最短経路問題の処理を完成してください。
（最短経路の表示処理が未完成になっています。）
https://scratch.mit.edu/projects/153760361/

補足説明(1)

～　ダイクストラ(E.W.Dijkstra)のアルゴリズム　～

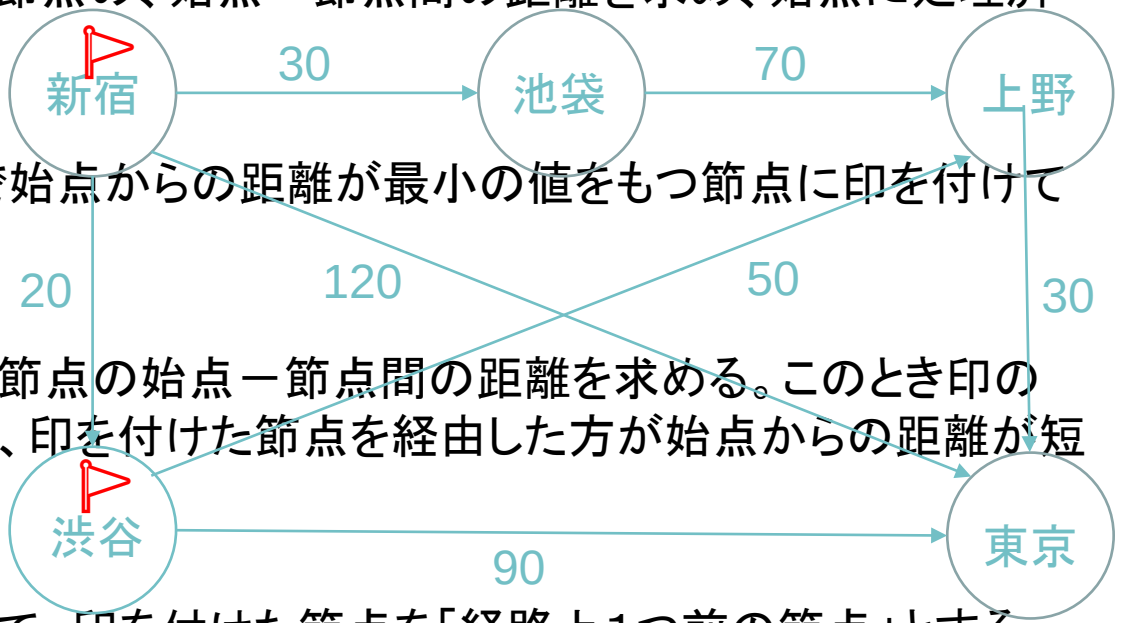
「新宿から東京まで最短何分(あるいは最低何円)で行けるか？」



補足説明(2)

～　ダイクストラ(E.W.Dijkstra) のアルゴリズム　～

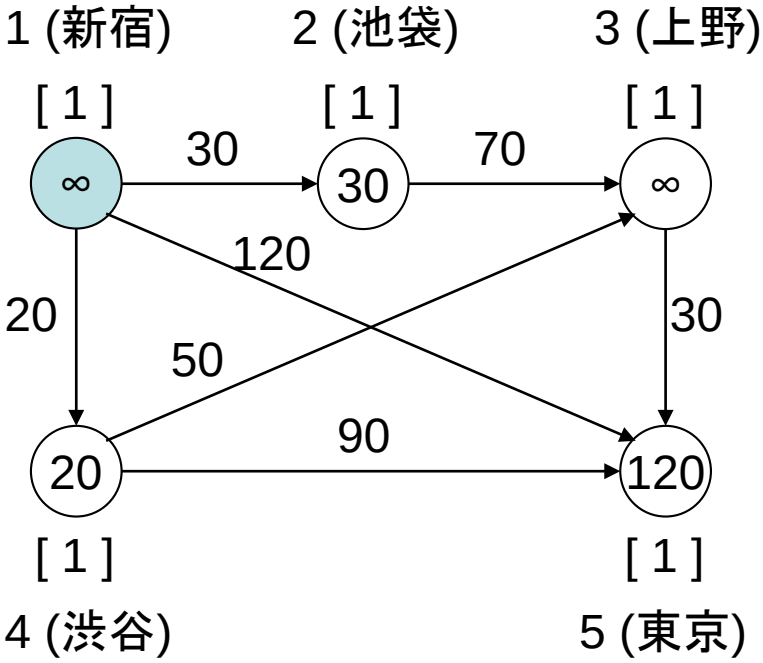
1. 始点に直接つながっている節点の、始点－節点間の距離を求め、始点に処理済みの印を付けて確定する。
2. 処理済みでない節点の中で始点からの距離が最小の値をもつ節点に印を付けて確定する。
3. 印を付けた節点につながる節点の始点－節点間の距離を求める。このとき印の付いていない節点について、印を付けた節点を経由した方が始点からの距離が短ければ距離を更新する。
4. 距離を更新した節点について、印を付けた節点を「経路上1つ前の節点」とする。



上記2～4をすべての節点に印が付くまで繰り返すと、各節点に得られる値が、始点からの最短距離となる。

補足説明(3)

― 初期状態 ―



・1, 2, 3, ... : 地点の番号

・矢印の数字 : 地点間の距離

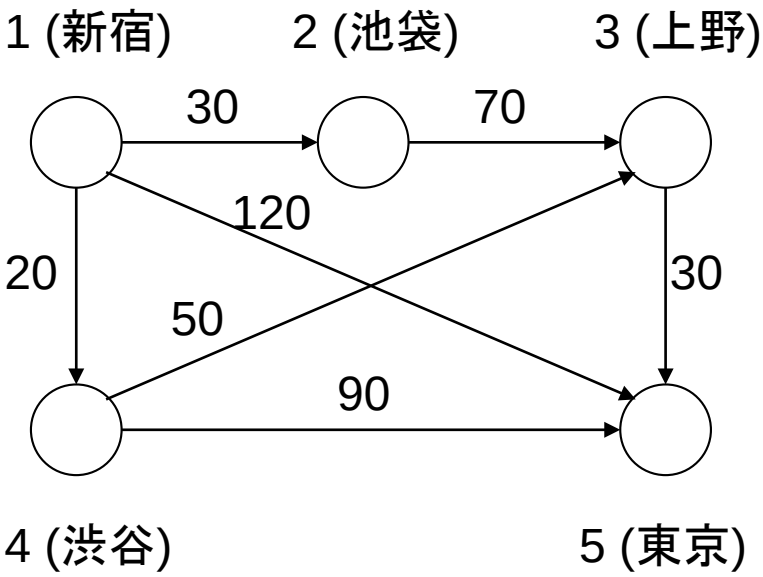
・〇の中の数字 : 出発地からの距離 → 配列D

・[ ] の中の数字 : 経路上1つ前の地点 → 配列S

・●: 処理済 → 配列P (値が1)

補足説明(4)

― 地点間の距離の表現 ―



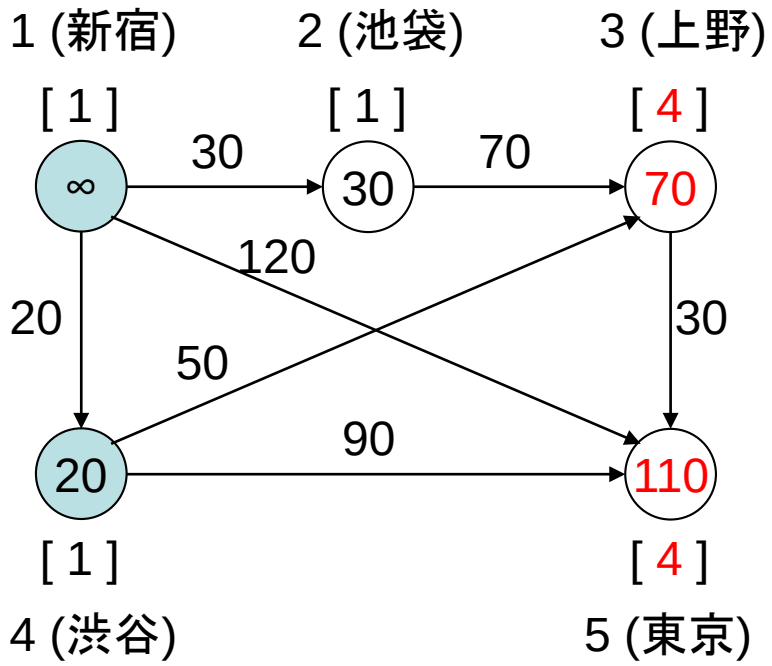
2次元配列 C

from ↑ \ to →	1	2	3	4	5
1	∞	30	∞	20	120
2	∞	∞	70	∞	∞
3	∞	∞	∞	∞	30
4	∞	∞	50	∞	90
5	∞	∞	∞	∞	∞

自分自身および直接の経路がない地点間には9999を入れておく。

補足説明(5)

― 1回目の処理 ―



・処理済でない地点のうち、出発地からの距離が最小の渋谷を処理済にする

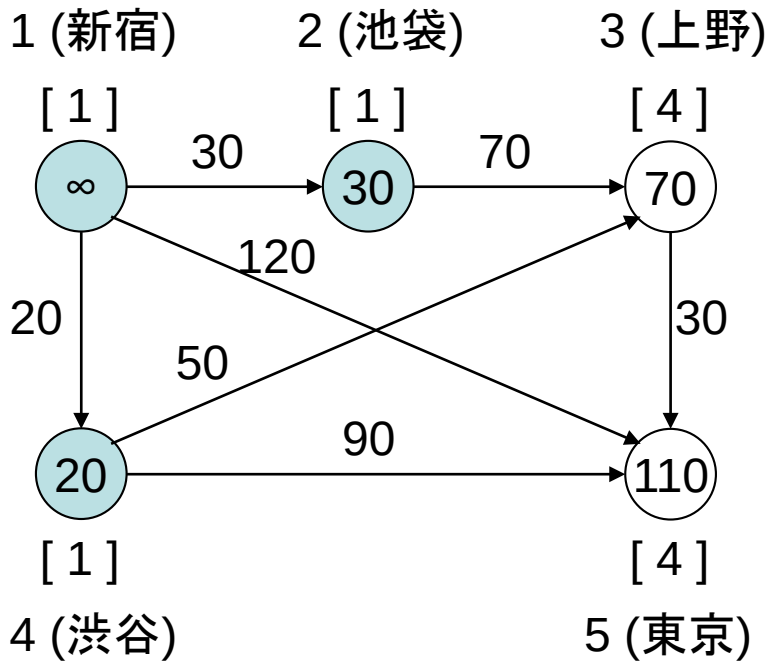
・渋谷を経由した方が出発地からの距離が短くなるものは距離を更新する

・更新した地点の1つ前の地点([ ] の中)を渋谷にする

(赤字は更新個所)

補足説明(6)

― 2回目の処理 ―

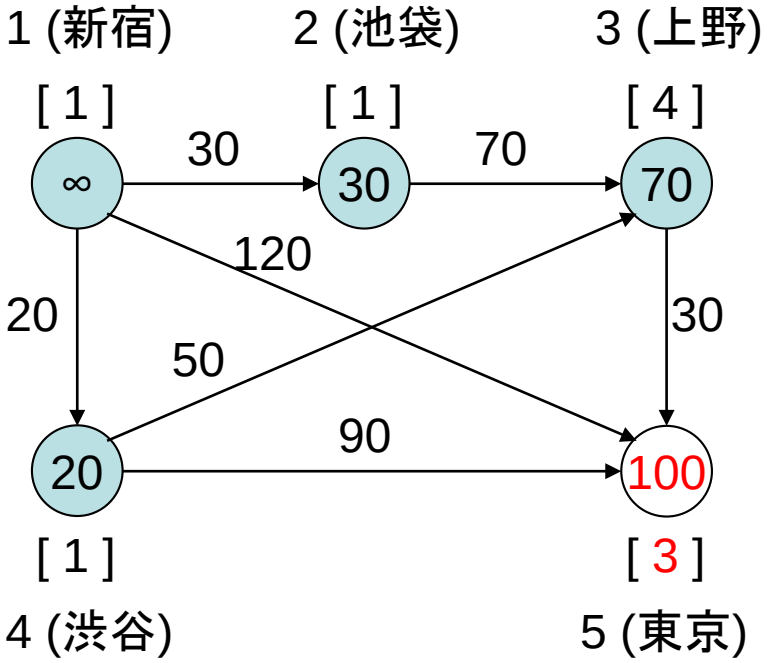


・処理済でない地点のうち、出発地からの距離が最小の池袋を処理済にする

・池袋を経由した方が出発地からの距離が短くなるものは距離を更新する （なし）

補足説明(7)

― 3回目の処理 ―



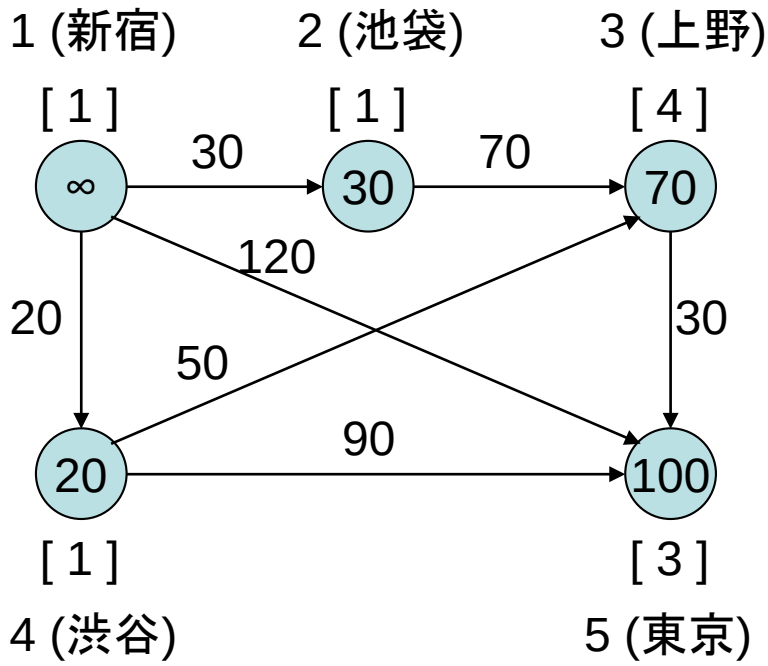
・処理済でない地点のうち、出発地からの距離が最小の上野を処理済にする

・上野を経由した方が出発地からの距離が短くなるものは距離を更新する

・更新した地点の1つ前の地点([ ] の中)を上野にする

補足説明(8)

― 4回目の処理 ―



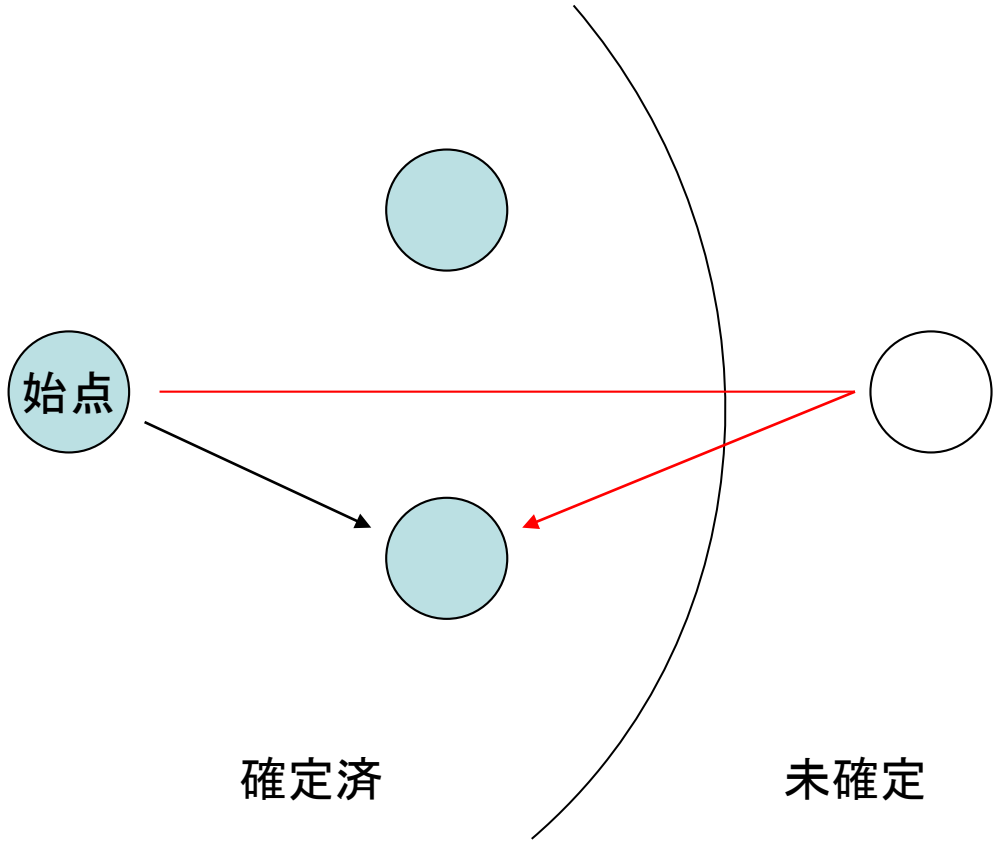
・処理済でない地点のうち、出発地からの距離が最短の東京を処理済にする

{ 最短経路の出力処理 }

・目的地の東京から[ ]の中を出発地までたどり(5 → 3 → 4 → 1)、逆順に表示すれば最短経路が出力される。

補足説明(9)

ー　なぜこれで最短経路が求まるのか　ー



始点からの距離が短い順に確定しているので、後から確定する節点を経由した方が短くなることはない。

赤の経路が黒の経路より短くなることはない。

確定済みの経路は後から確定する節点の影響を受けない。

補足説明（10）

☆ダイクストラ法の計算量：　　$O(n^2)$　（隣接行列を用いた場合、n：地点数）

1つの駅を確定するのに n 回の繰り返しで処理できるので、計算量は n ×n のオーダになる。

☆全経路比較を行う場合の計算量：　O(n!)　（隣接行列を用いた場合、n：地点数）

全経路の数は最大で (n - 1) 個の数の順列の個数となるため、

(n - 1) ×(n - 2)×・・・× 1 となり、計算量は n! のオーダになる。

最短経路問題(9)

[解答例]

ダイクストラ法による最短経路問題
https://scratch.mit.edu/projects/153736583/