

01. 並び替えのアルゴリズム

例えば、次のような表では、どのような仕組みで「昇順」「降順」への並び替えが行われるのだろうか。

送信者 ▼	件名 ▼	受信日時 ▼
宮下武	次回会合について	6/1/2017 10:05
川田美香	スケジュールのご確認	5/30/2017 18:01
野崎順子	昨日のお礼	5/30/2017 9:39
浅田酒店	ご注文の品について	5/29/2017 11:45
赤井直紀	Re: アルゴリズムの質問	5/25/2017 13:22
秋山裕子	ご協力のお願い	5/24/2017 12:57
浅田酒店	新酒入荷しました	5/21/2017 16:10
安達裕也	セミナーのお知らせ	5/20/2017 15:02

◆ 基本選択法（選択ソート）

- 最小選択法
- 最大選択法

【最小選択法の実装例】

1. 比較基準値を決める。
2. 最初の数値を比較基準値とし、n個の中から最も小さい数字を探し、それと入れ替える。
3. 次に、残りのn-1個の中から最も小さい数字を探し、それを2番目の数字と入れ替える。
4. この処理をn-1回繰り返す。

```
function minSelectSort(Array $array): Array
{
    // 比較基準値を固定し、それ以外の数値と比べていく。
    for($i = 0; $i < count($array)-1; $i++){

        // 比較基準値を仮の最小値として定義。
        $min = $array[$i];

        // 比較基準値の位置を定義
        $position = $i;

        // 比較基準値の位置以降で、数値を固定し、順番に評価していく。
        for($j = $position + 1; $j < count($array); $j++){

            // 比較基準値の位置以降に小さい数値があれば、比較基準値と最小値を更新。
            if($min > $array[$j]){
                $position = $j;
                $min = $array[$j];
            }
        }
    }
}
```

```

    }

    // 比較基準値の位置が更新されていなかった場合、親のfor文から抜ける。
    if($i == $position){
        break;
    }

    // 親のfor文の最小値を更新。
    $array[$i] = $min;

    // 次に2番目を比較基準値とし、同じ処理を繰り返していく。
}
return $array;
}

```

```

// 実際に試してみる。
$array = array(10,2,12,7,16,8,13)
$result = selectSort($array);
var_dump($result);

// 昇順に並び替えられている。
2, 7, 8, 10, 12, 13, 16

```

【アルゴリズム解説】

データ中の最小値を求め、次にそれを除いた部分の中から最小値を求める。この操作を繰り返していく。

02



数列を線形探索し、最小値を探します。最小値1が見つかりました（線形探索は、3-1節で解説しています）。



03



最小値の1を列の左端の6と交換し、ソート済みにします。なお、最小値がすでに左端であった場合には、何の操作も行いません。



05



2を左から2番目の6と交換し、ソート済みにします。





ソートが完了しました。

◆ クイックソート

【実装例】

1. 適当な値を基準値（Pivot）とする（※できれば中央値が望ましい）
2. Pivotより小さい数を前方、大きい数を後方に分割する。
3. 二分割された各々のデータを、それぞれソートする。
4. ソートを繰り返し実行する。

```
function quickSort(Array $array): Array
{
    // 配列の要素数一つしかない場合、クイックソートする必要がないので、返却する。
    if (count($array) <= 1) {
        return $array;
    }

    // 一番最初の値をPivotとする。
    $pivot = array_shift($array);

    // グループを定義
    $left = $right = [];

    foreach ($array as $value) {
        if ($value < $pivot) {
            // Pivotより小さい数は左グループに格納
            $left[] = $value;
        } else {
            // Pivotより大きい数は右グループに格納
            $right[] = $value;
        }
    }
}
```

```
// 処理の周回ごとに、結果の配列を結合。
return array_merge
(
    // 左のグループを再帰的にクイックソート。
    quickSort($left),

    // Pivotを結果に組み込む。
    array($pivot),

    // 右のグループを再帰的にクイックソート。
    quickSort($right)
);
}
```

```
// 実際に試してみる。
$array = array(6, 4, 3, 7, 8, 5, 2, 9, 1);
$result = quickSort($array);
var_dump($result);

// 昇順に並び替えられている。
1, 2, 3, 4, 5, 6, 7, 8
```

【アルゴリズム解説】

適当な値を基準値（Pivot）とし、それより小さな値のグループと大きな値のグループに分割する。同様に、両グループの中でPivotを選び、二つのグループに分割する。グループ内の値が一つになるまで、この処理を繰り返していく。

02



基準となる数（ピボット）を、数列の中からランダムに1つ選びます。今回は4が選ばれました。



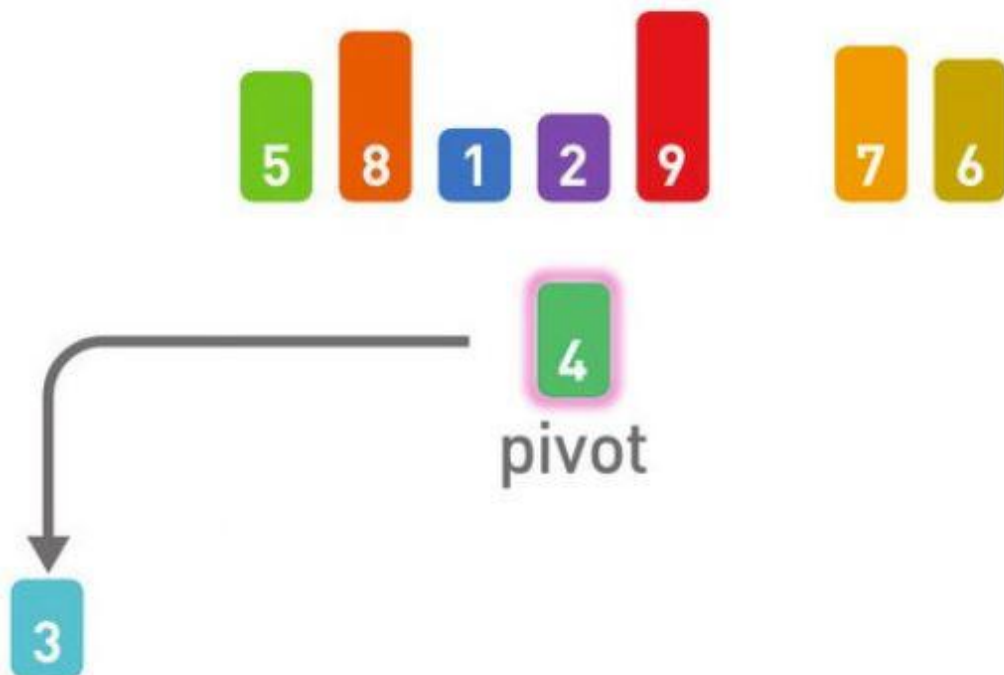
03



ピボット以外の各数字を、ピボットと比較していきます。ピボットより小さい数字は左に、大きい数字は右に移動します。



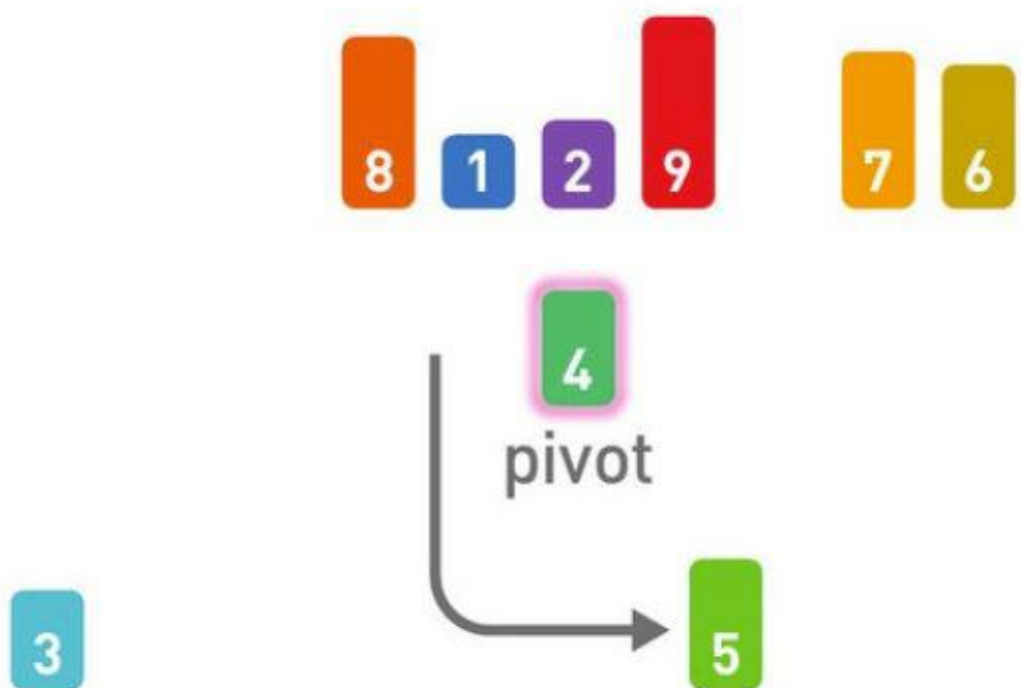
05



$3 < 4$ なので、3は左に移動します。



07



5 > 4なので、5は右に移動します。



08



他の数字も同様に比較し、移動させていくと、このようになります。



10



したがって、左側と右側をそれぞれ独立にソートすれば、全体のソートが完成します。



13

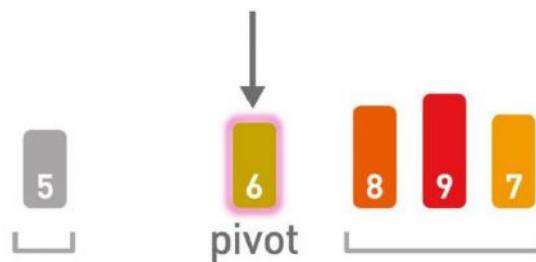


A yellow bar with the number '6' inside, highlighted with a pink glow. Below it is the word 'pivot'.

ピボットの6とそれぞれの数字を比較し、小さければ左へ、大きければ右へ移動します。



15



先ほどと同様、左右を独立にソートすれば、この部分のソートが完成します。しかし、左側は5のみなので、すでにソート済みです。これ以上やる必要はありません。右側はこれまでと同様に、ピボットを選んでいきます。



16



8がピボットとして選ばれました。



17



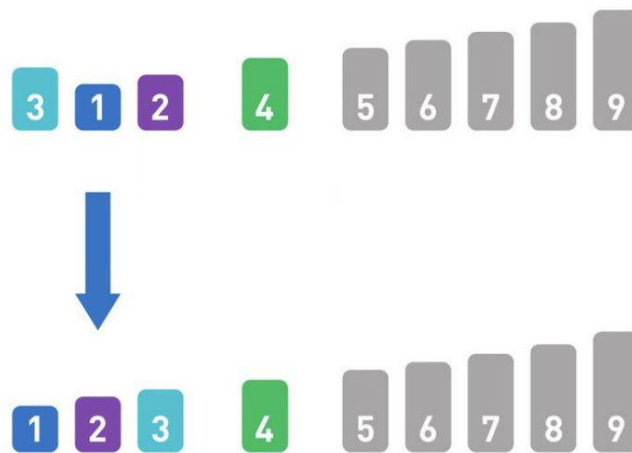
9と7が8と比較されて、左右に振り分けられました。8の左右はどちらも1つの数字しかないので、これで終わりです。これで、7 8 9がソート済みになりました。





その結果、最初のピボットの4の右側は、ソートが終了します。





左側も同様にソートしていくと、全体のソートが完了します。

◆ 基本交換法（バブルソート）

隣り合ったデータの比較と入替えを繰り返すことによって、小さな値のデータを次第に端のほうに移していく方法。

01



数列の右端に天秤を置き、天秤の左右の数字を比較します。比較した結果、右の数字の方が小さければ入れ替えます。



03



比較が完了すると天秤を1つ左に移動し、同様に数字を比較します。今回は $4 < 6$ なので、数字は入れ替えません。



05



並べ替えを繰り返し、天秤が左端に到達しました。一連の操作で、数列の中で最も小さい数字が左端に移動したことになります。



07



天秤を右端に戻します。先ほどと同様の操作を、天秤が左から2番目に到達するまで繰り返します。



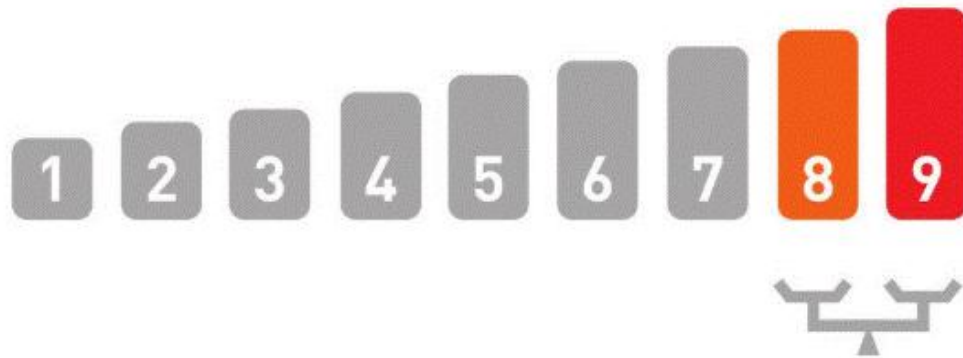
09



天秤を右端に戻します。同様の操作を、すべての数字がソート済みになるまで繰り返します。



12



ソートが完了しました。

◆ 基本挿入法（挿入ソート）

既に整列済みのデータ列の正しい位置に、データを追加する操作を繰り返していく方法。

◆ ヒープソート

◆ シェルソート

02. 配列内探索のアルゴリズム

◆ 線形探索法

今回は、配列内で「6」を探す。

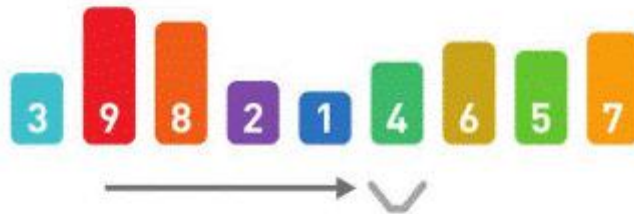
02



まず、配列の左端の数字を調べます。6と比較し、一致すれば探索を終了します。一致しなければ、1つ右の数字を調べます。



04



6が見つかるまで比較を繰り返します。



05



6が見つかったので探索を終了します。

◆ 二分探索法

前提として、ソートによって、すでにデータが整列させられているとする。今回は、配列内で「6」を探す。

02



まず、配列の真ん中にある数を調べます。この場合は5になります。



03



5と、探索する数である6を比較します。



04



必要のなくなった数字は候補から外します。



05



残った配列の真ん中にある数を調べます。この場合は7になります。



07



必要のなくなった数字は候補から外します。



08

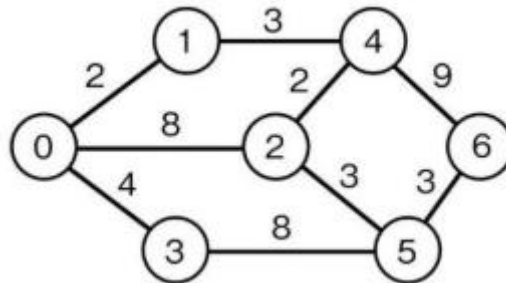


残った配列の真ん中にある数を調べます。この場合は6になります。

03. グラフ探索のアルゴリズム（難し過ぎて記入途中）

◆ ダイクストラ法による最良優先探索

【実装例】



地点間の距離を表で表す。ただし、同地点間の距離は『0』、隣り合わない地点間の距離は『-1』とする。

	0	1	2	3	4	5	6
0	0	2	8	4	-1	-1	-1
1	2	0	-1	-1	3	-1	-1
2	8	-1	0	-1	2	3	-1
3	4	-1	-1	0	-1	8	-1
4	-1	3	2	-1	0	-1	9
5	-1	-1	3	8	-1	0	3
6	-1	-1	-1	-1	9	3	0



```
// 各地点間の距離を二次元の連想配列で定義
$matrix = array(
    'P0' => array(0, 2, 8, 4, -1, -1, -1),
    'P1' => array(2, 0, -1, -1, 3, -1, -1),
    'P2' => array(8, -1, 0, -1, 2, 3, -1),
    'P3' => array(4, -1, -1, 0, -1, 8, -1),
    'P4' => array(-1, 3, 2, -1, 0, -1, 9),
    'P5' => array(-1, -1, 3, 8, -1, 0, 3),
    'P6' => array(-1, -1, -1, -1, 9, 3, 0)
);
```

```
// 各地点間の距離、出発地点、開始地点を引数にとる。
public function bestFirstSearchByDijkstra(
    Array $matrix,
```

```

Int $startPoint,
Int $goalPoint
)
{
    // 地点数を定数で定義
    define('POINT_NUMBER', count($matrix));

    if($startPoint < self::POINT_NUMBER
        || self::POINT_NUMBER < $goalPoint){
        throw new Exception('存在しない地点番号は設定できません。') ;
    }

    // 出発地点を定数で定義
    define('START_POINT', $startPoint);

    // 到着地点を定数で定義
    define('GOAL_POINT', $goalPoint));

    // 無限大の定数のINFを使いたいが、定数は上書きできないため、代わりに-1を使用。
    // 各頂点に対して、最短ルート地点番号、地点間距離の初期値、最短距離確定フラグを設定。
    for($i = 0; $i < self::POINT_NUMBER; $i ++){
        $route[$i] = -1;
        $distance[$i] = -1;
        $fixFlg[$i] = false;
    }

    // 【別の書き方】
    // $cost = array_fill(0, self::POINT_NUMBER, -1);
    // $distance = array_fill(0, self::POINT_NUMBER, -1);
    // $fix = array_fill(0, self::POINT_NUMBER, false);

    // 出発地点から出発地点への距離をゼロとする。
    $distance[self::START_POINT] = 0;

    //
    while(true) {
        $i = 0;

        while($i < self::POINT_NUMBER){
            if(!$fixFlg[$i]){
                break 1;
            }
            $i += 1;
        }

        if($i === self::POINT_NUMBER){
            break 1;
        }

        for($j = $i + 1; $j < self::POINT_NUMBER; $j ++){
            if(!$fixFlg[$i] && $distance[$j] < $distance[$i]){
                $i = $j;
            }
        }
    }

```

今の自分には、これ以上は難しい...

未来の俺、頑張ってくれ...

```
}
```

【最短経路探索処理の解説】

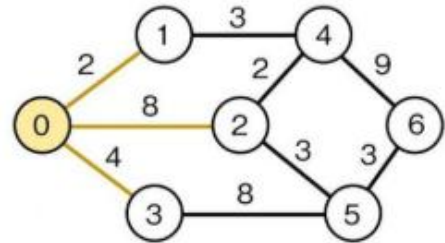
\$startPoint = 0

\$goalPoint = 6

とした時、出発地点（0）から1ステップ行ける地点までの距離（pDist）を取得し、確定させる。

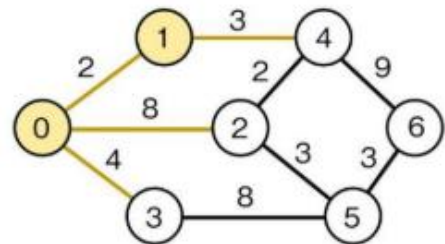
・繰り返し 1 回目

sPoint	0	1	2	3	4	5	6
添字=(=地点)	0	1	2	3	4	5	6
pFixed	true	false	false	false	false	false	false
↓確定							
添字=(=地点)	0	1	2	3	4	5	6
pDist	0	2	8	4	∞	∞	∞
pRoute	0	0	0	0	0	0	0



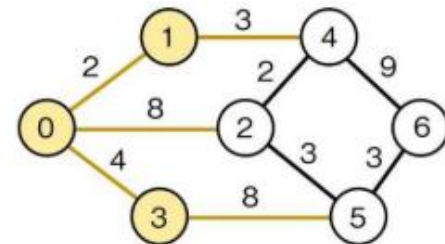
・繰り返し 2 回目 (未確定の地点のうち、距離が最小のものは地点 1)

sPoint	1						
添字 (=地点)	0	1	2	3	4	5	6
pFixed	true	true	false	false	false	false	false
		↓確定					
添字 (=地点)	0	1	2	3	4	5	6
pDist	0	2	8	4	5	∞	∞
pRoute	0	0	0	0	1	0	0



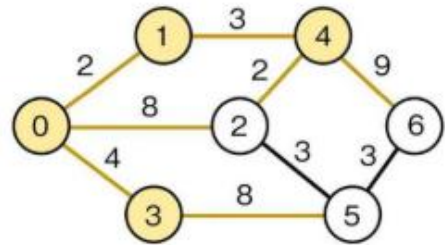
・繰り返し 3 回目 (未確定の地点のうち、距離が最小のものは地点 3)

sPoint	3						
添字 (=地点)	0	1	2	3	4	5	6
pFixed	true	true	false	true	false	false	false
	↓確定						
添字 (=地点)	0	1	2	3	4	5	6
pDist	0	2	8	4	5	12	∞
pRoute	0	0	0	0	1	3	0



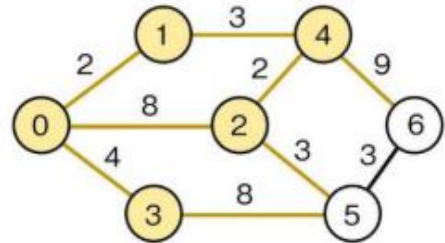
・繰り返し 4 回目 (未確定の地点のうち、距離が最小のものは地点 4)

sPoint	4						
添字=(=地点)	0	1	2	3	4	5	6
pFixed	true	true	false	true	true	false	false
					↓確定		
添字=(=地点)	0	1	2	3	4	5	6
pDist	0	2	7	4	5	12	14
pRoute	0	0	4	0	1	3	4



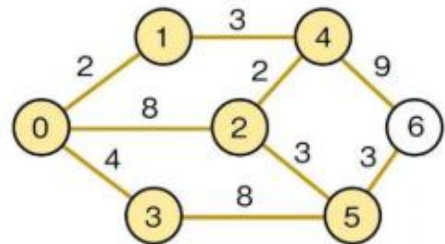
・繰り返し 5 回目 (未確定の地点のうち、距離が最小のものは地点 2)

sPoint	2						
添字=(=地点)	0	1	2	3	4	5	6
pFixed	true	true	true	true	true	false	false
					↓確定		
添字=(=地点)	0	1	2	3	4	5	6
pDist	0	2	7	4	5	10	14
pRoute	0	0	4	0	1	2	4



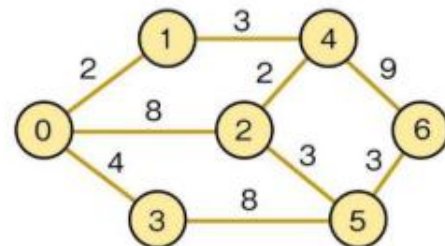
・繰り返し 6 回目 (未確定の地点のうち、距離が最小のものは地点 5)

sPoint	5						
添字=(=地点)	0	1	2	3	4	5	6
pFixed	true	true	true	true	true	true	false
						↓確定	
添字=(=地点)	0	1	2	3	4	5	6
pDist	0	2	7	4	5	10	13
pRoute	0	0	4	0	1	2	5



・繰り返し 7 回目 (未確定の地点は地点 6)

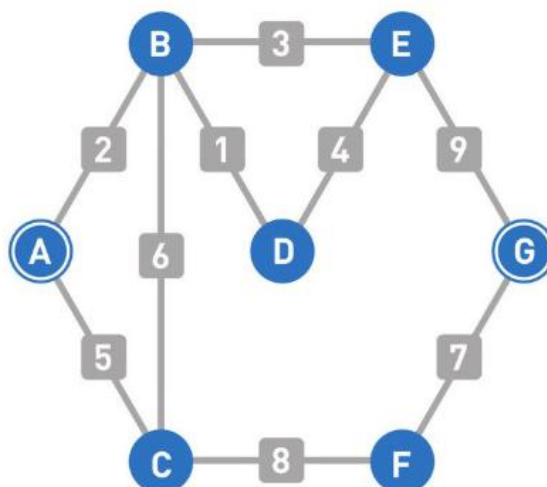
sPoint	6						
添字=(=地点)	0	1	2	3	4	5	6
pFixed	true	true	true	true	true	true	true
							↓確定
添字=(=地点)	0	1	2	3	4	5	6
pDist	0	2	7	4	5	10	13
pRoute	0	0	4	0	1	2	5



【アルゴリズム解説】

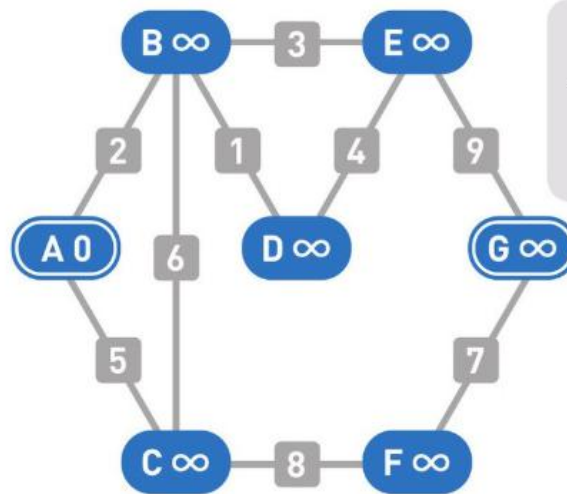
正のコストの経路のみの場合、用いることができる方法。

01



ここではAを始点、Gを終点としてダイクストラ法を説明します。

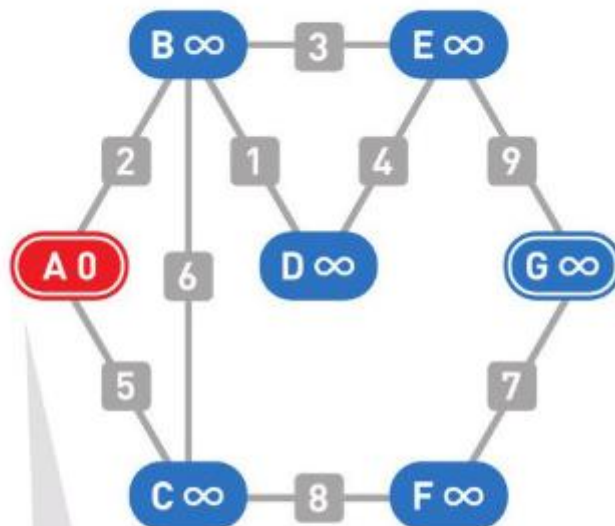
02



このコストは、ベルマン-フォード法のとくと同じく、最短路の暫定コストを表すものです。

はじめに各頂点のコストの初期値を設定します。始点は0、それ以外の頂点は無限大(∞)に設定します。

03



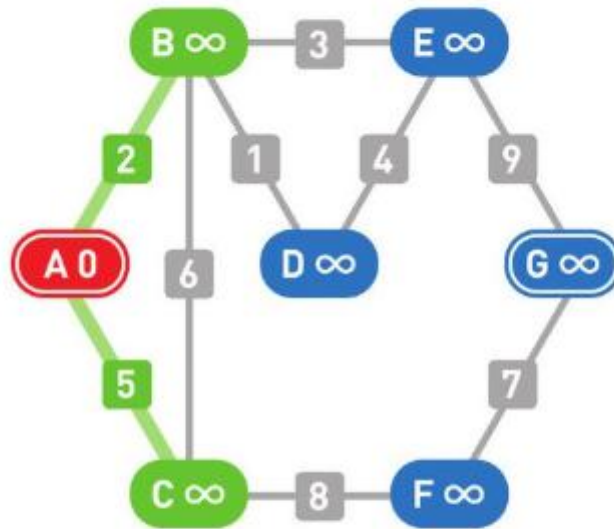
現在いる頂点を赤で示すことにします。

始点からスタートします。



04

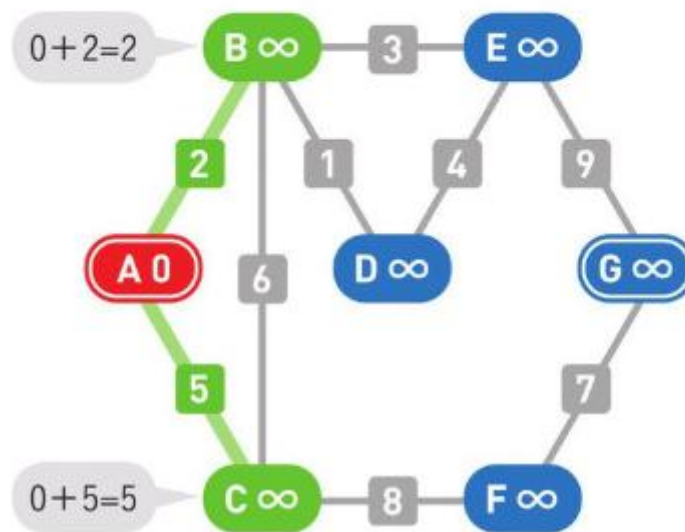
候補は緑で表します。



現在いる頂点から辿ることのできる、探索済みでない頂点を探します。見つけた頂点は、次に辿る候補にします。この場合、頂点Bと頂点Cが候補となります。



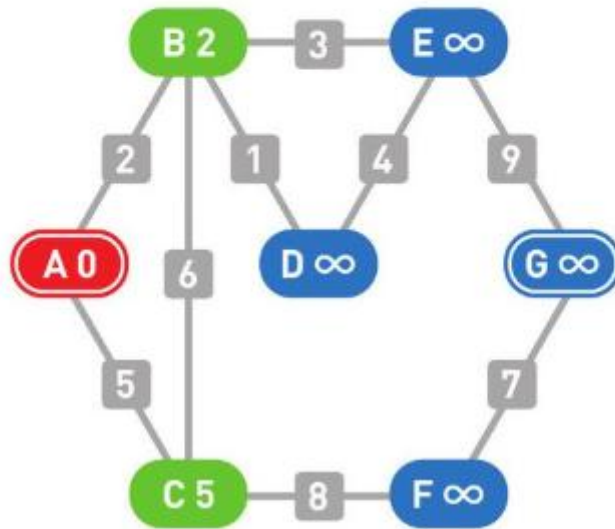
05



候補の各頂点のコストを計算します。計算方法は、「現在いる頂点のコスト+現在いる頂点から候補の頂点へ辿るコスト」になります。例えば、頂点Bの場合、現在いる始点Aのコストが0なので、 $0+2$ で2となります。同様にCのコストは $0+5$ で5となります。



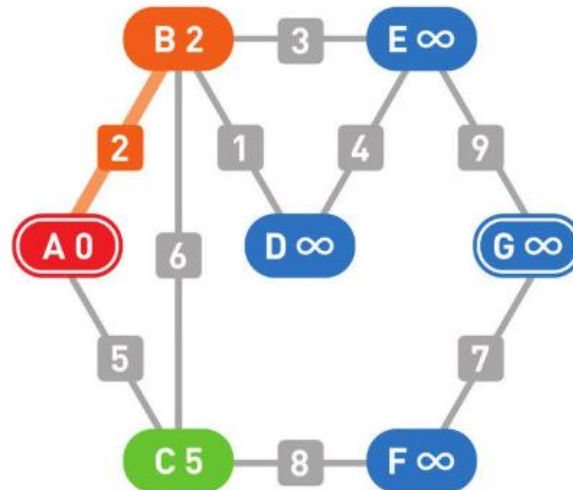
06



計算した結果が現在のコスト値より小さければ、コストを新しい値で更新します。頂点B、Cの現在のコストは無限大であり、計算した結果の方が小さいため、それぞれ新しい値で更新しました。



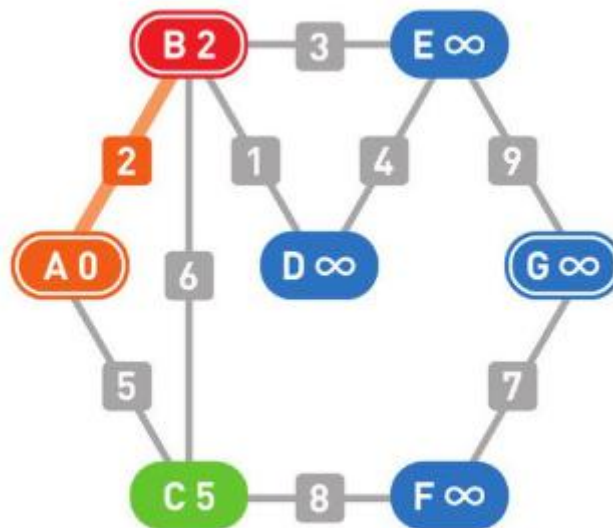
07



候補の頂点の中から、コストが最も小さい頂点を選択します。この場合は頂点Bになります。この時点で、選択した頂点Bへの経路A-Bが、始点から頂点Bに至る最短路として決定しました。なぜなら、他の経路を使う場合には必ず頂点Cを経由する必要があり、その結果、現状の経路よりもコストが高くなるからです。



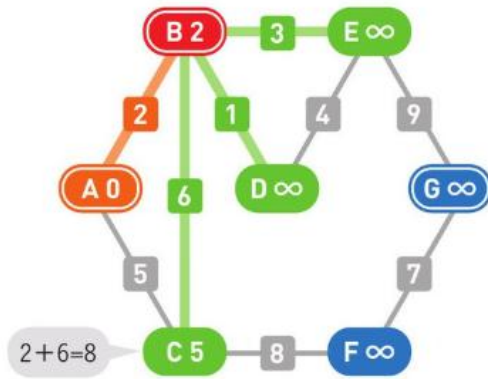
08



最短路が決定した頂点Bに移動します。



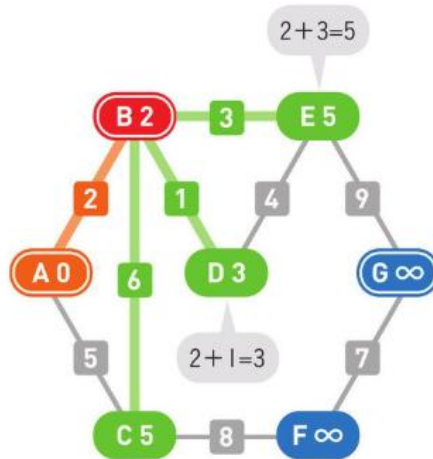
10



先ほどと同様の計算方法で、候補の各頂点のコストを計算します。頂点Bから辿った場合の頂点Cのコストは $2+6$ で8となりますが、現状の5の値の方が小さいので更新しません。



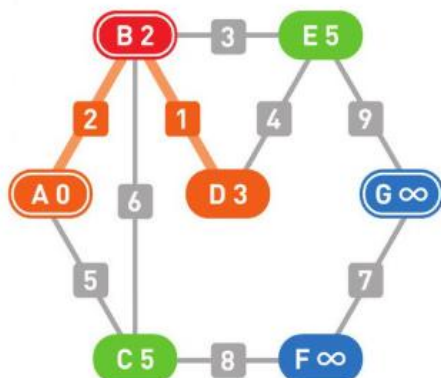
11



残りの頂点DとEのコストを更新しました。



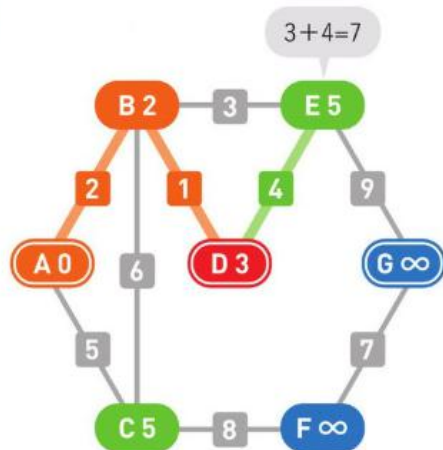
12



候補の頂点の中から、コストが最も小さい頂点を選択します。この場合は頂点Dになります。この時点で、選択した頂点Dへの経路A-B-Dが、始点から頂点Dに行く最短路として決定しました。



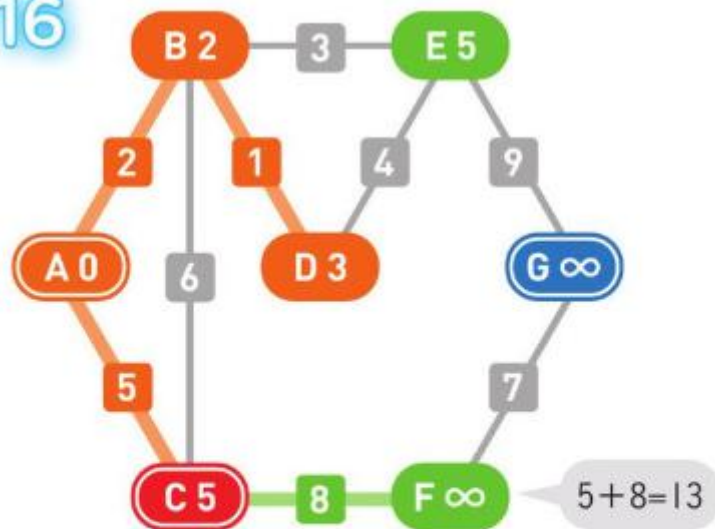
14



同様の操作を、終点Gに辿り着くまで繰り返します。Dに移動してEのコストを計算しますが、更新されません ($3+4=7$ になるため)。現在の候補はCとEで、どちらも同じ値 (5) ですから、どちらを選択してもかまいません。



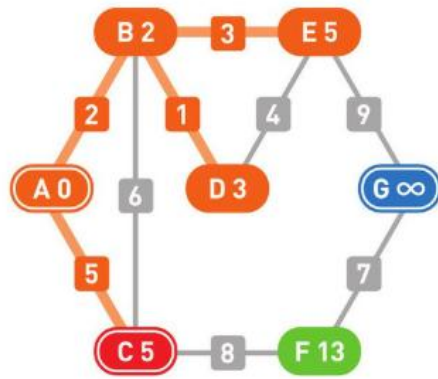
16



Cへと移動しました。今度はFが新たに候補になり、Fのコストが13に更新されます。候補はEの5とFの13なので……



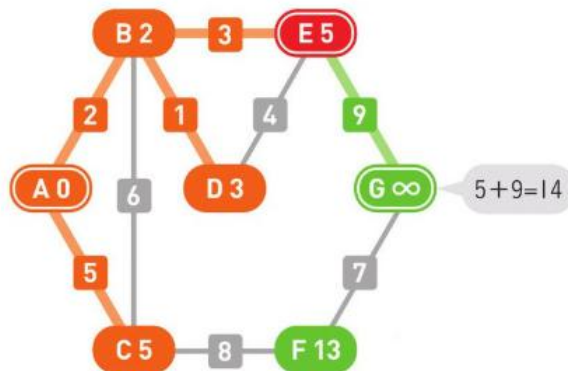
17



小さい方のEが選択され、Eへの最短路が決定します。



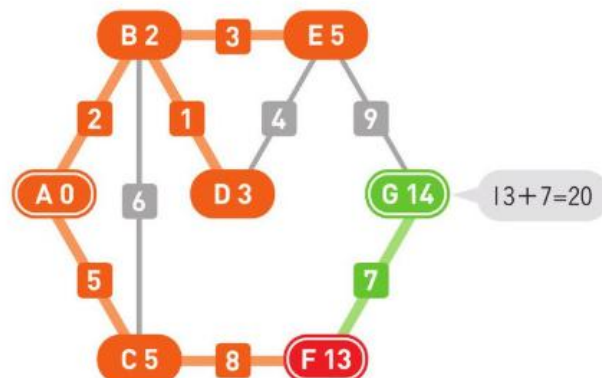
18



Eへ移動しました。Gが新たに候補になり、Gのコストが14に更新されます。候補はFの13と、Gの14ですから、小さい方のFが選択され、Fへの最短路が決定します。

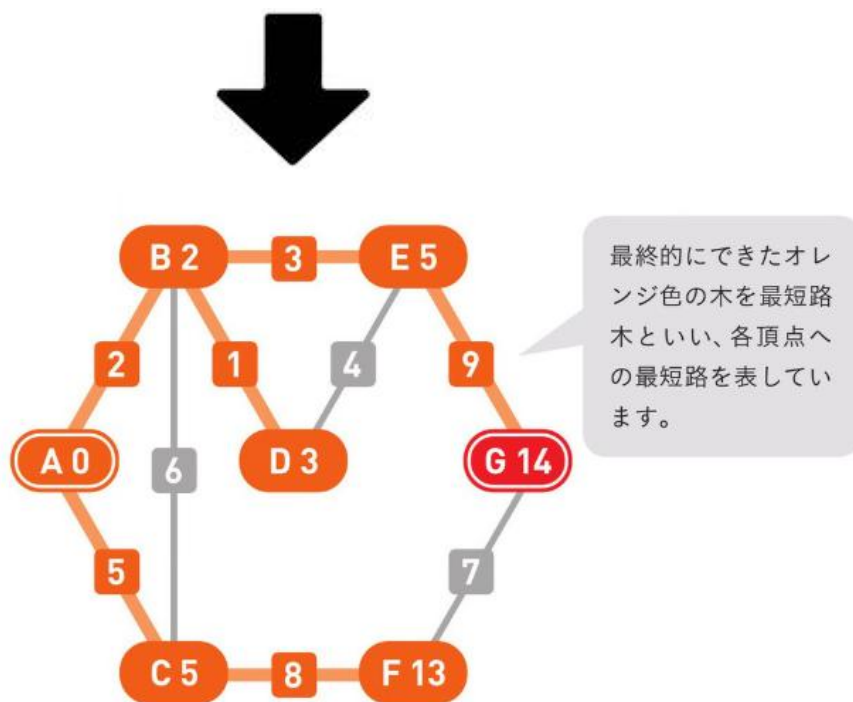


19



Fへ移動しました。Gの値を計算すると $13 + 7$ で20ですが、現状の14の方が小さいので、Gのコストは更新されません。候補はGの14のみなので、Gが選択され、Gへの最短路が決定しました。

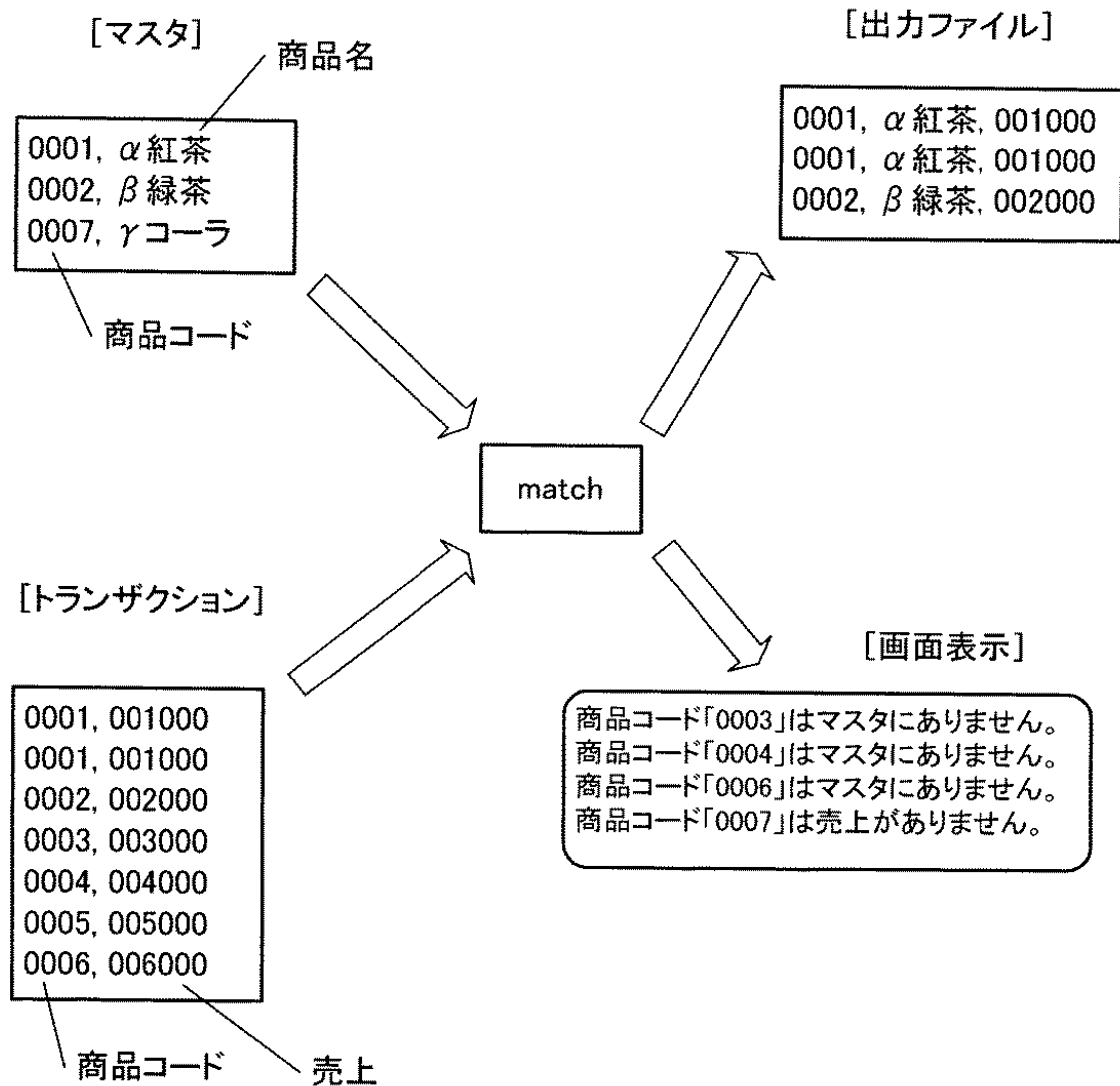
20



終点Gに辿り着いたので、探索を終わります。

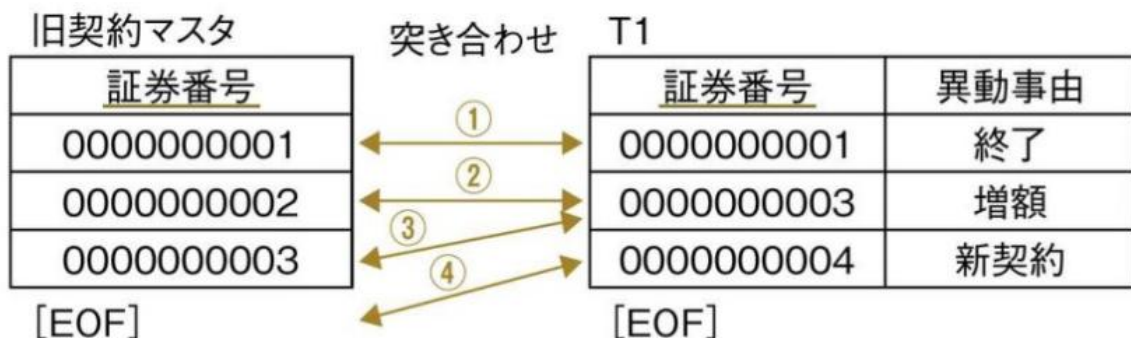
04. マッチング（突き合わせ）のアルゴリズム

ビジネスの基盤となるマスタデータ（商品データ、取引先データなど）と、日々更新されるトランザクションデータ（販売履歴、入金履歴など）を突き合わせ、新しいデータを作成する処理のこと。



◆ レコード間のマッチング（突き合わせ）の反復処理

前処理として、マスタデータとトランザクションデータで同じ識別子を持たせ、両方を昇順で並び替える。



1. マスタデータの一行目と、トランザクションデータの一行目の識別子を突き合わせる。『マスタデータ=トランザクションデータ』の時、そのレコードを新しいデータとして出力する。続けて、マスタデータの一行目と、トランザクションデータの二行目のレコードを突き合わせる。『マスタデータ≠トランザクションデータ』になるまで、トランザクションのN行目のレコードを突き合わせる。
2. マスタデータの二行目と、トランザクションデータの二行目の識別子を突き合わせる。『マスタデータ<トランザクションデータ』になる。

3. マスタデータの三行目と、トランザクションデータの二行目の識別子を突き合わせる。『マスタデータ=トランザクションデータ』の時、そのレコードを新しいデータとして出力する。


05. 誤り検出と訂正のアルゴリズム

◆ Check Digit Check

バーコードやクレジットカードなどの読み取りチェックで使われている誤り検出方法。

1. Check Digitを算出する。
2. 算出されたCheck Digitが正しいかを検証する。

花王Biore 弱酸性化粧水



4 901301 75224 6

CDの算出方法

桁 12 11 10 9 8 7 6 5 4 3 2 1

4	9	0	1	3	0	1	7	5	2	2	4	6
X	X	X	X	X	X	X	X	X	X	X	X	CD
1	3	1	3	1	3	1	3	1	3	1	3	

$4 + 27 + 0 + 3 + 3 + 0 + 1 + 21 + 5 + 6 + 2 + 12 = 84$

$84 \div 10 = 8 \text{ 余り } 4$

$10 - 4 = 6$

◆ Parity Check

◆ CRC : Cyclic Redundancy Check (巡回冗長検査)