

# 数理モデル

## 第03回

### マッチングとDAアルゴリズム

#### 改訂履歴

日付	担当者	内容
2025/04/09	武田 守	初版

#### 目次

- (1) はじめに
- (2) 基本概念
  - (2.1) 選好とは
  - (2.2) マッチングの安定性
- (3) DAアルゴリズム
  - (3.1) DAアルゴリズムとその性質
- (4) パレート効率性
  - (4.1) パレート効率性とは
  - (4.2) マッチングのパレート効率性
- (5) 応用例と実装
  - (5.1) 機能仕様
  - (5.2) 入出力仕様
  - (5.3) 概略データ構造とアルゴリズム
  - (5.4) 実装
  - (5.5) 実行結果

## (1) はじめに

デイヴィッド・ゲール(David Gale)と ロイド・シャープレー(Lloyd Stowell Shapley)が  
「複数男性と複数女性の間で、各個人の選好順序を元に結婚相手を選択する」という  
「安定結婚問題 (アンティケッコンメンダイ、stable marriage problem)」を、1962年に提示しました。

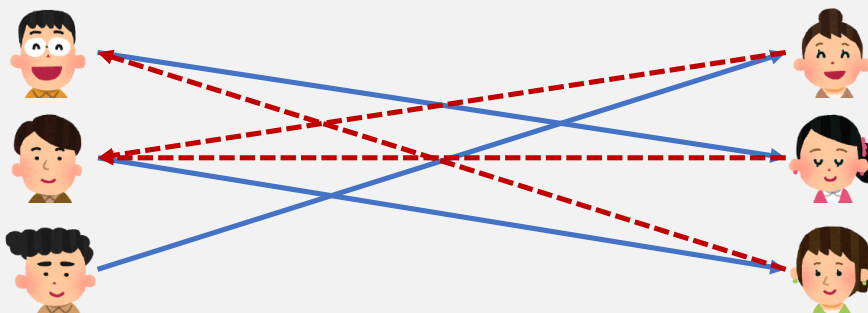
ここでは、安定結婚問題の解法として、兩人によって考案された  
「GSアルゴリズム (ジ-エスアルゴリズム、Gale-Shapley Algorithm)」または  
「DAアルゴリズム (ディ-エアルゴリズム、Deferred Acceptance Algorithm)」という  
マッチングのアルゴリズムについて解説します。

その上で、職員の配属先のマッチングアルゴリズムを考えてみます。

(1 番目のお気に入り)

(複数男性)

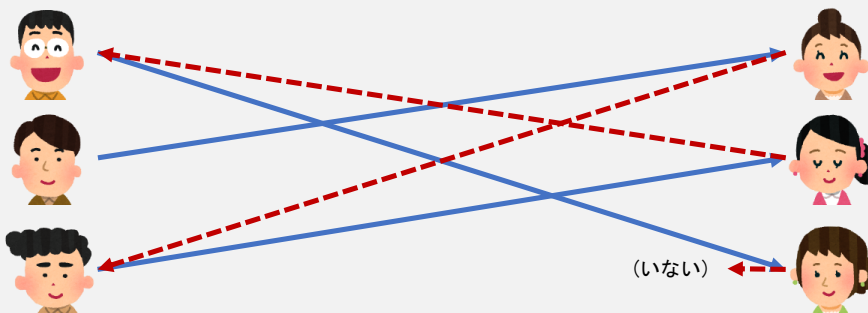
(複数女性)



(2 番目のお気に入り)

(複数男性)

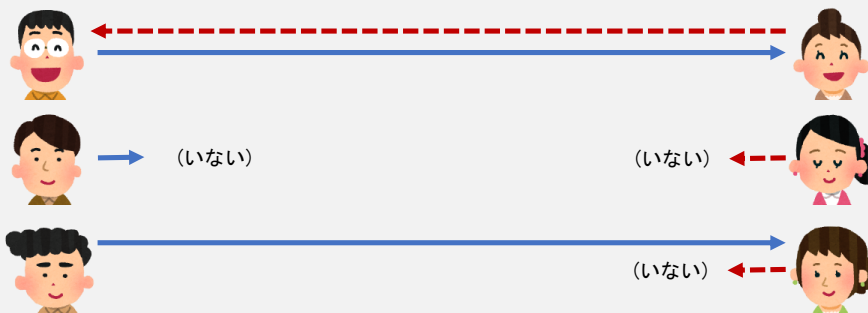
(複数女性)



(3 番目のお気に入り)

(複数男性)

(複数女性)



## (2) 基本概念

### (2.1) 選好とは

#### 選好

意思決定において、意思決定者 $i$ が選択対象に対して持つ好みの順序を「選好 (せんこう、Preference)」と呼び、記号  $>_i$  で表します。

具体的には、意思決定者 $i$ が選択対象 $B$ より $A$ を好む場合、以下のように表します。

$$A >_i B \quad (i \in \{\text{意思決定者}\}, A, B \in \{\text{選択対象}\})$$

(例) 個人  $i$  はある日のおやつで、ミカンよりリンゴを選びます

$$\text{リンゴ} >_i \text{ミカン}$$

#### 選好の推移性

選好は「推移性 (スイテイ、Transitivity)」を満たします。即ち

$$A >_i B \text{ かつ } B >_i C \text{ ならば } A >_i C \quad (\text{式2.1-1})$$

$$(i \in \{\text{意思決定者}\}, A, B, C \in \{\text{選択対象}\})$$

これを纏めて、以下のように表します：

$$>_i : A B C \quad (\text{「} A >_i B \text{ かつ } B >_i C \text{」を意味する})$$

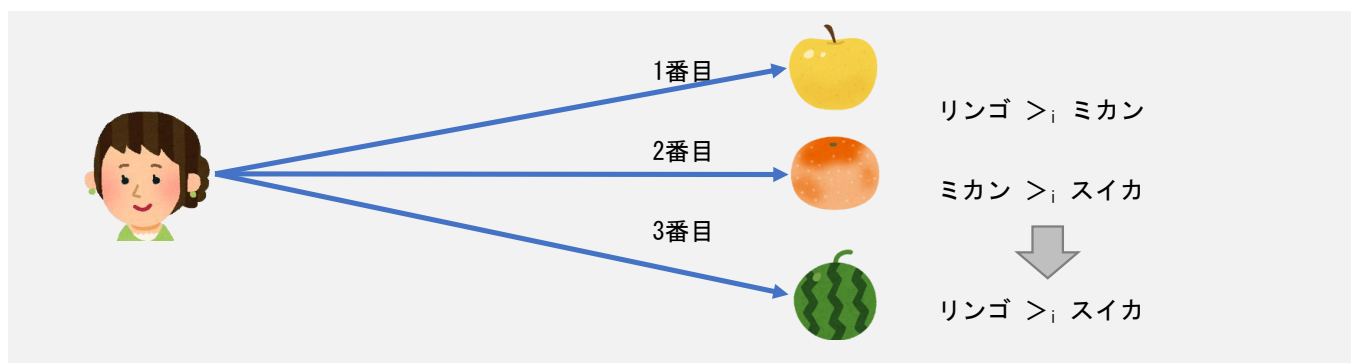
(例) 個人  $i$  はある日のおやつで、ミカンよりリンゴを選び、かつスイカよりミカンを選ぶならば

個人  $i$  はその日のおやつで、スイカよりリンゴを選びます。

$$(\text{リンゴ} >_i \text{ミカン} \text{ かつ } \text{ミカン} >_i \text{スイカ}) \text{ ならば } (\text{リンゴ} >_i \text{スイカ})$$

これは別表記では、以下ようになります：

$$>_i : \text{リンゴ、ミカン、スイカ}$$



## (2.2) マッチングの安定性

### マッチング

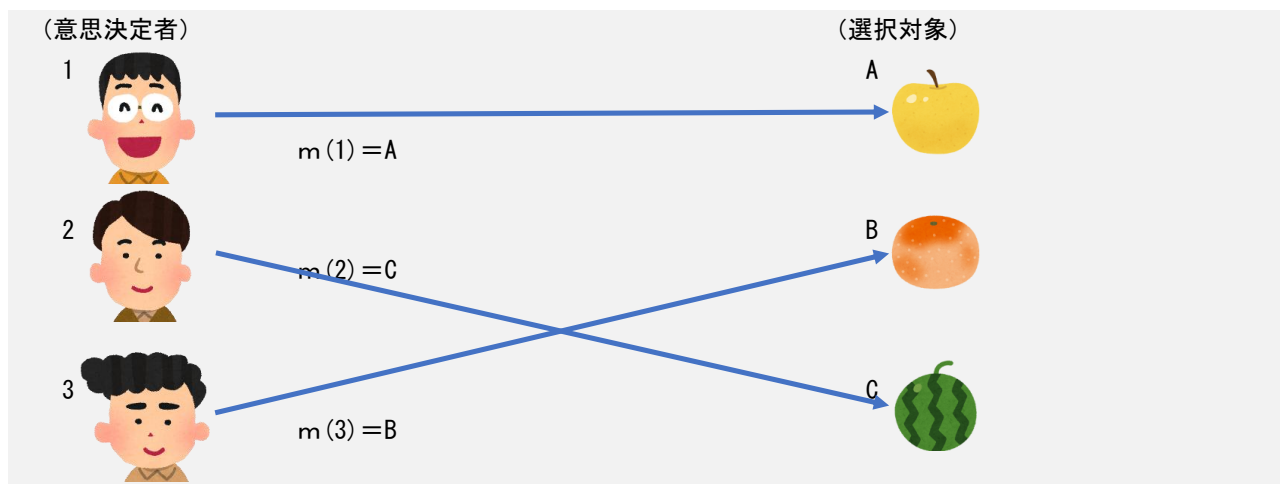
意思決定者が複数いて、一人の意思決定者に対して選択対象が1つだけ決定される場合、意思決定者と選択対象の対応を定義する関数を「マッチング (マッチング、matching)」関数と言います。マッチングでは、選好順序が最適に反映された関数をどのように作成するかが課題になります。

(例) 意思決定者 1, 2, 3 が選択対象A, B, Cを次の組合せ：

(1, A)、(2, C)、(3, B)

で選択した時、マッチング関数  $m$  を以下のように表します：

$$m = (m(1), m(2), m(3)) = (A, C, B)$$



## ブロッキングペア

着目するマッチング関数 $m$ について、

あるマッチングペア  $(i, j)$  ( $i \in \{\text{立場 I}\}$ 、 $j \in \{\text{立場 J}\}$ ) があり、

$j >_i m(i)$  かつ  $i >_j m(j)$

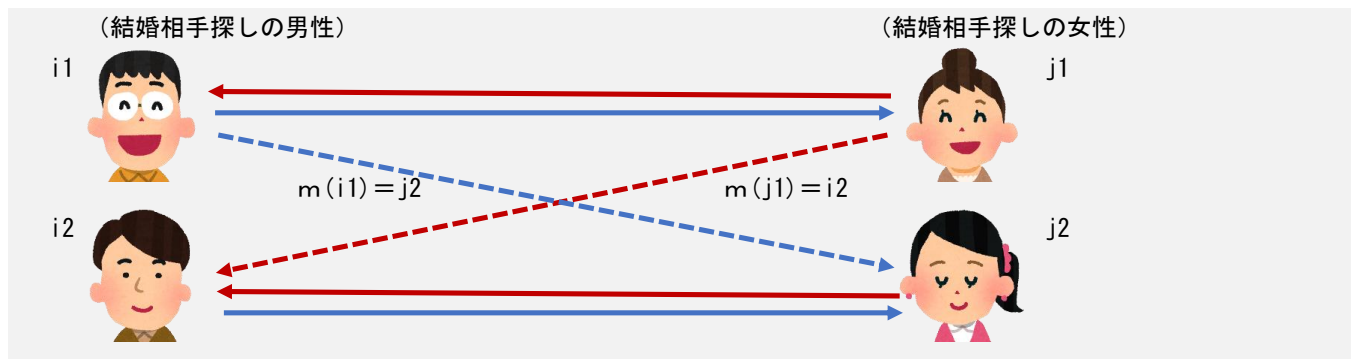
(関数 $m$ が示す  $i$  の相手が、 $i$  が好きな  $j$  よりも選好順序が低く、かつ、

関数 $m$ が示す  $j$  の相手が、 $j$  が好きな  $i$  よりも選好順序が低い)

を満たす (関数 $m$ が示すマッチングペアより、ペアの双方に適している別のマッチングペアがある)

場合、このマッチングペアを「ブロッキングペア (blocking pair)」と言います。

(例)



上記例では、マッチング関数 $m$ が示す相手は

$m(i1) = j2$ 、 $m(j1) = i2$

で、マッチングペアは  $(i1, j2)$ 、 $(i2, j1)$  です。

一方、各自の選好順序では

$j1 >_{i1} j2$ 、 $j2 >_{i2} j1$ 、 $i1 >_{j1} i2$ 、 $i2 >_{j2} i1$

で、マッチングペア  $(i1, j1)$ 、 $(i2, j2)$  が選好順序を反映しています。

上記マッチング関数 $m$ は

$j1 >_{i1} m(i1) = j2$ 、 $i1 >_{j1} m(j1) = i2$

となっており、

- ・  $i1$  にとって、 $(i1, j1)$  のペアの方が関数 $m$ の  $(i1, j2)$  より選好順序を反映している。
- ・  $j1$  にとって、 $(i1, j1)$  のペアの方が関数 $m$ の  $(i2, j1)$  より選好順序を反映している。

これより、関数 $m$ をペア  $(i1, j1)$  がブロックしている、と判断できます。

- ・ お互いに「現在のマッチングの相手を捨てて、新たにペアを作った方が良い」というペアが存在しない時、マッチングは安定的である、と評価されます。

## マッチングの安定性

着目するマッチング関数 $m$ について、

ブロッキングペアが存在しない時、マッチングは「安定 (アンティ, stable)」であると言います。

ブロッキングペアが存在する時、マッチングは「不安定 (アンティ, unstable)」であると言い、

ペアの組み直しでマッチングの最適化が必要になります。

## 【出典・参考】

みずほ証券>ファイナンス用語集>選好

[https://glossary.mizuho-](https://glossary.mizuho-sc.com/faq/show/1866?back=front%2Fcategory%3Ashow&category_id=87&page=1&site_domain=default&sort=sort_adjust_value&sort_order=desc)

[sc.com/faq/show/1866?back=front%2Fcategory%3Ashow&category\\_id=87&page=1&site\\_domain=default&sort=sort\\_adjust\\_value&sort\\_order=desc](https://glossary.mizuho-sc.com/faq/show/1866?back=front%2Fcategory%3Ashow&category_id=87&page=1&site_domain=default&sort=sort_adjust_value&sort_order=desc)

選好⇒「その問題 数理モデルが解決します」 「第6章アルバイトの配属方法」

ベレ出版 浜田宏著 2019年

安定結婚問題⇒

<https://ja.wikipedia.org/wiki/%E5%AE%89%E5%AE%9A%E7%B5%90%E5%A9%9A%E5%95%8F%E9%A1%8C>

### (3) D A アルゴリズム

#### (3.1) D A アルゴリズムとその性質

マッチングする結婚相手を探すアルゴリズムがD A アルゴリズムで、以下のようなものです。

##### D A アルゴリズム

###### 【前提】

- ・ 男性と女性がそれぞれ複数人いる。
- ・ 各男性はマッチしたい女性の希望順位を、  
各女性はマッチしたい男性の希望順位を選好順序表として提出する。
- ・ 選好順序表に記載しない相手とのマッチングは行わない。
- ・ 提出された選好順序表に基づいて男性と女性を以下のアルゴリズムでマッチさせる。

###### 【アルゴリズム】

- ・ 男性側の希望順位を元にしたアルゴリズムは以下になる。

(※女性側の希望順位を元にしたアルゴリズムも同様であるが、  
一般的に男性側の希望順位を元にした結果とは異なる。)

(ステップ1) 各男性は、自身の最も希望順位の高い女性に申し込む。

各女性は、申し込んできた男性の中に自身の選好順序表に書いた人がいれば、  
そのうち最も希望順位の高い男性一人を保留し、それ以外の男性を断る。

(ステップ2) 全ての男性が誰かに保留されるか、申し込める女性がいらない状態になるまで、  
以下を繰り返す。

(2.1) 保留されていない男性は、自身の選好順序表に書いていてまだ申し込んでいない女性のうち、  
最も希望順位の高い女性に申し込む。

各女性は、前ステップで保留した男性と新たに申し込んできた男性のうち、  
自身の選好順序表に書いた人がいればその中で最も希望順位の高い男性一人を保留し、  
それ以外の男性を断る。

(2.2) 繰り返し終了時点において、誰かに保留されている男性は、  
その男性を保留している女性とマッチさせる。

(ステップ3) 誰にも保留されていない男性と、誰も保留していない女性は、  
誰ともマッチしなかったという結果とする。







D A アルゴリズムには、次のような性質があります。

##### D A アルゴリズムの性質

- ・ D A アルゴリズムは、必ず安定的なマッチングを実現する。

(例) D A アルゴリズムの適用例

選好順序表

参加者 希望順位	男性			女性		
	A	B	C	X	Y	Z
						
第1希望	X	X	Y	C	A	A
第2希望	Z	Y	Z	A	B	B
第3希望	Y	Z	X	B	C	(記載無)

(1) 男性側の希望順位を元にしたD A アルゴリズムの適用

(ステップ1) 第1希望の結果

参加者 希望順位	男性			女性		
	A	B	C	X	Y	Z
第1希望	<b>X</b>	<del>X</del>	<b>Y</b>	C	A	A
第2希望	Z	Y	Z	<b>A</b>	B	B
第3希望	Y	Z	X	<del>B</del>	<b>C</b>	(記載無)

<b>太枠</b>	: 着目
<b>太字</b>	: 保留
<del>斜線</del>	: 断り

(ステップ2) 保留されていない男性の希望確認 (男性B)

参加者 希望順位	男性			女性		
	A	B	C	X	Y	Z
第1希望	<b>X</b>	<del>X</del>	<del>Y</del>	C	A	A
第2希望	Z	<b>Y</b>	Z	<b>A</b>	<b>B</b>	B
第3希望	Y	Z	X	<del>B</del>	<del>C</del>	(記載無)

(ステップ2) 保留されていない男性の希望確認 (男性C)

参加者 希望順位	男性			女性		
	A	B	C	X	Y	Z
第1希望	<b>X</b>	<del>X</del>	<del>Y</del>	C	A	A
第2希望	Z	<b>Y</b>	<del>Z</del>	<b>A</b>	<b>B</b>	B
第3希望	Y	Z	X	<del>B</del>	<del>C</del>	(記載無)

(ステップ2) 保留されていない男性の希望確認 (男性C)

参加者 希望順位	男性			女性		
	A	B	C	X	Y	Z
第1希望	<del>X</del>	<del>X</del>	<del>Y</del>	<b>C</b>	A	A
第2希望	Z	<b>Y</b>	<del>Z</del>	<del>A</del>	<b>B</b>	B
第3希望	Y	Z	<b>X</b>	<del>B</del>	<del>C</del>	(記載無)

(ステップ2) 保留されていない男性の希望確認 (男性A)

参加者 希望順位	男性			女性		
	A	B	C	X	Y	Z
第1希望	<del>X</del>	<del>X</del>	<del>Y</del>	<b>C</b>	A	<b>A</b>
第2希望	<b>Z</b>	<b>Y</b>	<del>Z</del>	<del>A</del>	<b>B</b>	B
第3希望	Y	Z	<b>X</b>	<del>B</del>	<del>C</del>	(記載無)

全ての男性が女性の誰かに保留されたので、処理終了とします。

マッチング結果は以下のようになりました：

(A, Z) : Aは第2希望、Zは第1希望



(B, Y) : Bは第2希望、Yは第2希望



(C, X) : Cは第3希望、Xは第1希望



(2) 女性側の希望順位を元にしたD Aアルゴリズムの適用

(ステップ1) 第1希望の結果

参加者	男性			女性		
	A	B	C	X	Y	Z
第1希望	X	X	Y	<b>C</b>	<del>A</del>	<b>A</b>
第2希望	<b>Z</b>	Y	Z	A	B	B
第3希望	<del>Y</del>	Z	<b>X</b>	B	C	(記載無)

太枠	: 着目
太字	: 保留
斜線	: 断り

(ステップ2) 保留されていない女性の希望確認 (女性Y)

参加者	男性			女性		
	A	B	C	X	Y	Z
第1希望	X	X	Y	<b>C</b>	<del>A</del>	<b>A</b>
第2希望	<b>Z</b>	<b>Y</b>	Z	A	<b>B</b>	B
第3希望	<del>Y</del>	Z	<b>X</b>	B	C	(記載無)

全ての女性が男性の誰かに保留されたので、処理終了とします。

マッチング結果は以下のようになりました：

(A, Z) : Aは第2希望、Zは第1希望



(B, Y) : Bは第2希望、Yは第2希望



(C, X) : Cは第3希望、Xは第1希望



この例では、男性側の希望順位を元にしたD Aアルゴリズムの適用結果と、女性側の希望順位を元にしたD Aアルゴリズムの適用結果とは、一致しています。また、ブロッキングペアは存在せず、マッチングは「安定」であることがわかります。

【出典・参考】

マッチングアルゴリズムの解説

<https://miiitomi.github.io/p/matching/>

D Aアルゴリズム⇒「その問題 数理モデルが解決します」「第6章アルバイトの配属方法」

ベレ出版 浜田宏著 2019年

イラスト⇒いらすとや

<https://www.irasutoya.com/>



## (4) パレート効率性

### (4.1) パレート効率性とは

マッチングの望ましさを評価する基準として、「パレート効率性」があります。

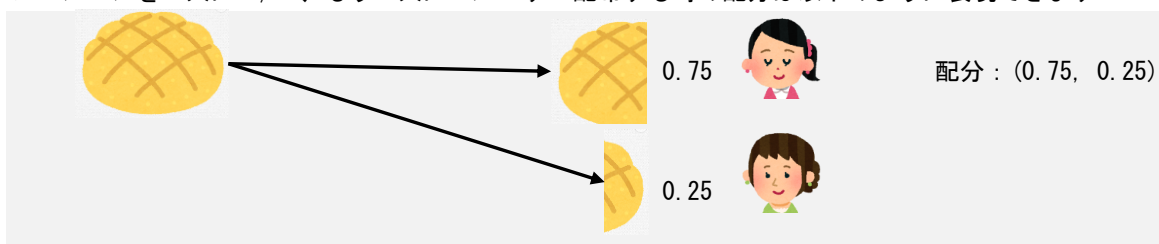
「パレート効率性 (パレトコリツキ, Pareto efficiency)」とは、イタリアの経済学者ヴィルフレド・パレート (Vilfredo Federico Damaso Pareto, 1848 - 1923) が提唱した概念で、資源配分が無駄なく行われている状態を指します。

以下に関連する概念を紹介します：

#### 配分

ある資源を部分に分けて配布する時、全ての配布先の要素  $i$  ( $i \in$  配布先の要素の集合  $I = \{1, \dots, I\}$ ) の各々の取り分の並びを「配分 (ハイブン, Allocation)」と言います。

(例) メロンパンを一人に  $3/4$ 、もう一人に  $1/4$  ずつ配布する時の配分は以下のように表現できます：



#### パレート支配

ある資源の配布で2つの配分  $x$ 、 $y$  があり、

配布先の要素  $i$  ( $i \in$  配布先の要素の集合  $I = \{1, \dots, I\}$ ) に対して

$$y_i \geq x_i \quad \text{for } \forall i \in I = \{1, \dots, I\} \quad (\text{全ての要素について})$$

$$y_i > x_i \quad \text{for } \exists i \in I = \{1, \dots, I\} \quad (\text{ある要素について})$$

$$(x_i \in \text{配分 } x, y_i \in \text{配分 } y, i \in \text{配布先の要素の集合 } I = \{1, \dots, I\})$$

となる時、

配分  $y$  が 配分  $x$  を「(広義の)パレート支配 (パレトシハイ, Weak Pareto domination)」すると言います。

#### パレート効率性

配分  $x$  が「パレート効率的 (パレトコリツキ, Pareto efficient)」であるとは、

配分  $x$  をパレート支配するような別の配分  $y$  が存在しないことです。

ある資源の配分  $x$  があり、

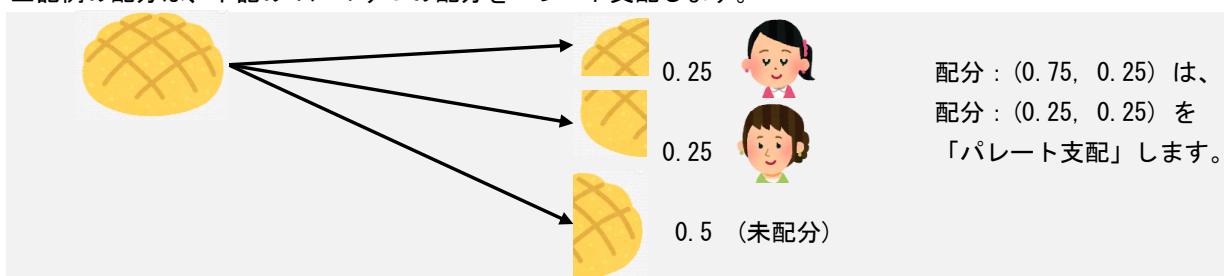
全ての配分の要素  $i$  ( $i \in I = \{1, \dots, I\}$ ) に対して

$$y_i > x_i \quad (x_i \in \text{配分 } x, y_i \in \text{配分 } y, i \in \text{配布先の要素の集合 } I = \{1, \dots, I\})$$

となるような配分  $y$  が存在しない。

※なお、「パレート効率的」であることに、公平さの配慮はありません。

(例) 上記例の配分は、下記の  $1/4$  ずつの配分をパレート支配します。



前者の配分は未配分を出しておらず、「パレート効率的」です。

#### (4.2) マッチングのパレート効率性

マッチングの望ましさを評価する基準として、「パレート効率性」があります。

関数  $m$  が、全ての意思決定者  $i$  にとって、

他のどんな関数  $m'$  よりも、意思決定者  $i$  の選好順序を最もよく反映したマッチング関数であることを  
マッチング関数  $m$  が「パレート効率的」である、と言います。

##### マッチングのパレート効率性

あるマッチング  $m$  が「パレート効率的」であるとは、  
次の条件を満たす別のマッチング  $m'$  が存在しないことである。

任意の意思決定者  $i$  に対して

$$m'(i) >_i m(i) \quad (i \in \{\text{意思決定者}\})$$

##### 安定的なマッチングとパレート効率性

安定的なマッチングは、必ず「パレート効率的」となります。

従って、DAアルゴリズムは、パレート効率的となります。

##### 【出典・参考】

パレート効率的⇒ Google AIによる概要

パレート効率的⇒ Wikipedia

<https://ja.wikipedia.org/wiki/%E3%83%91%E3%83%AC%E3%83%BC%E3%83%88%E5%8A%B9%E7%8E%87%E6%80%A7#:~:text=5%20%E9%96%A2%E9%80%A3%E9%A0%85%E7%9B%AE-,%E5%AE%9A%E7%BE%A9%E3%81%A8%E6%84%8F%E5%91%B3,%E3%81%A7%E3%81%82%E3%82%8B%E3%81%A8%E8%A1%A8%E7%8F%BE%E3%81%99%E3%82%8B%E3%80%82>

パレート支配⇒ WIIS 純粋交換経済におけるパレート効率的な配分

<https://wiis.info/economics/microeconomics/market-equilibrium-theory/efficient-allocation-in-pee/>

<https://www.irasutoya.com/>

パレート効率性⇒「その問題 数理モデルが解決します」「第6章アルバイトの配属方法」

ベレ出版 浜田宏著 2019年

配分⇒広島大学>1.2 経済学の方法

<https://home.hiroshima-u.ac.jp/okochi/Bmicro/Nmic01.pdf>

イラスト⇒いらすとや

(5) 応用例と実装

以下では、配属先の「配属計画表」、職員の「配属先希望表」と、配属先の配員基準ルールを元に、職員の一週間分の配属先への配置アルゴリズムを、D A アルゴリズムに準じたもので考案します。

(5.1) 機能仕様

- ・ 配属先は曜日ごとに「1早番・2遅番・3夜勤」の3つがあり、各曜日の配属先の勤務日数を「勤務日数」、その最少定員を「配属先最少定員」、配属先毎に職員に必要とされる資格を「必要資格順」、その最少定員を「必要資格最少定員」として記載した「配属計画表」を提出する。

・ 配属計画表（例）

曜日	配属先	勤務日数	配属先最少定員	必要資格順	必要資格最少定員
日	1早番	1	2	資格1又は資格2	1
				無資格	0
	2遅番	1	2	資格1又は資格2	1
				無資格	0
	3夜勤	2	2	資格1又は資格2	1
				無資格	0
月	1早番	1	2	資格1又は資格2	1
				無資格	0
	2遅番	1	2	資格1又は資格2	1
				無資格	0
	3夜勤	2	2	資格1又は資格2	1
				無資格	0

- ・ 各職員は「所有資格」と「配属先希望順」と一週間の「曜日毎希望表」と「週の勤務日数」を記載した「配属先希望表」を提出する。

- ・ 「所有資格」には、所有資格・性格・こだわりなどを記載する。  
それが無い場合「無資格」とする。
- ・ 「曜日毎希望表」には以下の内容で記載する。  
－：勤務可能日                      ×：勤務不可日                      記載なし：勤務可能日

・ 配属先希望表（例）

職員	所有資格	配属先希望順			週の勤務日数	曜日毎希望表						
		1	2	3		日	月	火	水	木	金	土
A	資格1	2遅番	1早番	3夜勤	4							
B	資格2 資格3	1早番	3夜勤	(なし)	4		×					×
C	無資格	2遅番	3夜勤	1早番	5			×				

(凡例)「曜日毎希望表」  
X：勤務不可日  
－：勤務可能日

- ・各職員について、決定済の勤務する曜日と配属先がある場合、一週間の「曜日毎配属表」に記載する。

・曜日毎配属表（例）

職員	曜日毎配属表						
	日	月	火	水	木	金	土
A	1早番	2遅番	3夜勤	/	-	-	-
B	-	-	-	-	-	-	-
C	-	-	-	-	-	-	-

（凡例）「曜日毎配属表」

- : 未定
- 1 : 早番
- 2 : 遅番
- 3 : 夜勤
- / : 夜勤明け

- ・提出された諸表に基づいた職員の配属先を、以下の「配員基準」に沿って決定し、「配属予定表」と「職員毎勤務予定表」を作成する。

配員基準

- ・配属先希望表の「配属先希望順」に記載しない配属先への配員は行わない。
- ・夜勤は1回あたり2日分の勤務とする。
- ・各職員の週の勤務日数を確保すること。確保できない場合、その旨を出力する。
- ・配属表の日々の配置で、最少定員は確保すること。確保できない場合、その旨を出力する。
- ・配属表の日々の配置で、最少定員超過は最小限に抑え、日々のばらつきを抑えること。
- ・職員の一週間の「曜日毎配属表」に記載がある場合、配員は「曜日毎配属表」に従う。
- ・職員の「曜日毎希望表」で「勤務不可日」で申請し、かつ「曜日毎配属表」に配属が記載済みでない場合、「曜日毎配属表」で「休み(申請休)」と扱う。

- ・各職員の「配属先希望順」（1, 2, …）に沿った配員とするために、各職員の「希望達成率」を以下のように定義し、配属日毎に「希望達成率」の低い職員を優先して配属を行う。

$$\text{「希望達成率」} = (1 / \text{「週の勤務日数」}) \times \sum_{\text{配属日}} \text{「配属先希望達成率」} \quad (\text{式5-1})$$

$$\text{「配属先希望達成率」} = ( \text{「配属先希望順」の登録数} - \text{配属先希望順} + 1 ) / \text{「配属先希望順」の登録数} \quad (\text{式5-2})$$

（例1）

職員Aについて、「週の勤務日数」=4、「配属先希望順」=（2遅番、1早番、3夜勤）の時で、配属済み日数が2で、各々の配属先が（1早番、3夜勤）の場合、

$$\text{「希望達成率」} = (1/4) \times ( (3-2+1)/3 + (3-3+1)/3 ) = 0.25 \quad (25\%)$$

（例2）

職員Bについて、「週の勤務日数」=2、「配属先希望順」=（2遅番、1早番）の時で、配属済み日数が2で、各々の配属先が（2遅番、2遅番）の場合、

$$\text{「希望達成率」} = (1/2) \times ( (2-1+1)/2 + (2-1+1)/2 ) = 1.00 \quad (100\%)$$

- ・各（曜日、配属先、必要資格）の最少定員を満たす配員が出来るように、各配属先の「配属計画表」と、各職員の「曜日毎希望表」、「所有資格」、「配属先希望順」に基づいて、（曜日、配属先、必要資格）毎の「必要資格毎配属希望率」を以下のように定義し、「必要資格毎配属希望率」の低い（曜日、配属先、必要資格）を優先して配員する。

「必要資格毎配属希望率」＝

$$\sum_{\text{職員}} \text{「当該曜日の勤務可否」} \times \text{「必要資格有無」} \times \text{「配属先希望達成率」} \div \sum_{\text{職員}} 1$$

$$\text{「当該曜日の勤務可否」} = \begin{cases} 0.0 & \text{：職員が当該曜日が勤務不可日の場合} \\ 1.0 & \text{：職員が当該曜日が勤務可能日の場合} \end{cases}$$

$$\text{「必要資格有無」} = \begin{cases} 0.0 & \text{：職員が配属先の必要資格を持っていない場合} \\ 1.0 & \text{：職員が配属先の必要資格を持っている場合} \end{cases}$$

$$\text{職員総数} = \sum_{\text{職員}} 1$$

(式5-3)

- ・（例）

以下の配属計画表と、全職員が3人の「配属先希望表」の場合について例示します。

(配属計画表)

曜日	配属先	勤務日数	配属先最少定員	必要資格順	必要資格最少定員
日	1早番	1	2	資格1又は資格2	1
				無資格	0
	2遅番	1	2	資格1又は資格2	1
				無資格	0
	3夜勤	2	2	資格1又は資格2	1
				無資格	0

曜日	配属先	勤務日数	配属先最少定員	必要資格順	必要資格最少定員
月	1早番	1	2	資格1又は資格2	1
				無資格	0
	2遅番	1	2	資格1又は資格2	1
				無資格	0
	3夜勤	2	2	資格1又は資格2	1
				無資格	0

(配属先希望表)

職員	所有資格	配属先希望順			週の勤務日数	曜日毎希望表						
		1	2	3		日	月	火	水	木	金	土
A	資格1	2遅番	1早番	3夜勤	4							
B	資格2 資格3	1早番	3夜勤	(なし)	4		×					×
C	無資格	2遅番	3夜勤	1早番	5			×				

(凡例) × : 勤務不可日  
記載なし : 勤務可能日

(凡例) 「配属先」

- 1 : 早番
- 2 : 遅番
- 3 : 夜勤

配属先「1早番」について、「配属先希望達成率」は、

$$\text{職員A} = (3-2+1)/3 = 0.66$$

$$\text{職員B} = (2-1+1)/2 = 1.00$$

$$\text{職員C} = (3-3+1)/3 = 0.33$$

これより、日曜日の「1早番」で、必要資格＝「資格1又は資格2」の「必要資格毎配属希望率」は、

$$\{(1.0 \times 1.0 \times 0.66) + (1.0 \times 1.0 \times 1.00) + (1.0 \times 0.0 \times 0.33)\} \div 3 = 0.55$$

同様に、月曜日の「1早番」で、必要資格＝「資格1又は資格2」の「必要資格毎配属希望率」は、

$$\{(1.0 \times 1.0 \times 0.66) + (0.0 \times 1.0 \times 1.00) + (1.0 \times 0.0 \times 0.33)\} \div 3 = 0.22$$

- ・各（曜日、配属先）の最少定員を満たす配員が出来るように、各配属先の「配属計画表」と、各職員の「曜日毎希望表」、「配属先希望順」に基づいて、（曜日、配属先）毎の「配属先毎配属希望率」を以下のように定義し、「配属先毎配属希望率」の低い（曜日、配属先）への配員を優先する。

「配属先毎配属希望率」＝

$$\sum_{\text{職員}} \text{「当該曜日の勤務可否」} \times \text{「配属先希望達成率」} / \sum_{\text{職員}} 1$$

$$\text{「当該曜日の勤務可否」} = \begin{cases} 0.0 & \text{：職員が当該曜日が勤務不可日の場合} \\ 1.0 & \text{：職員が当該曜日が勤務可能日の場合} \end{cases}$$

$$\text{職員総数} = \sum_{\text{職員}} 1$$

(式5-3)

- ・（例）

以下の配属計画表と、全職員が3人の「配属先希望表」の場合について例示します。

(配属計画表)

曜日	配属先	勤務日数	配属先最少定員	必要資格順	必要資格最少定員
日	1早番	1	2	資格1又は資格2	1
				無資格	0
	2遅番	1	2	資格1又は資格2	1
				無資格	0
	3夜勤	2	2	資格1又は資格2	1
				無資格	0

曜日	配属先	勤務日数	配属先最少定員	必要資格順	必要資格最少定員
月	1早番	1	2	資格1又は資格2	1
				無資格	0
	2遅番	1	2	資格1又は資格2	1
				無資格	0
	3夜勤	2	2	資格1又は資格2	1
				無資格	0

(配属先希望表)

職員	所有資格	配属先希望順			週の勤務日数	曜日毎希望表						
		1	2	3		日	月	火	水	木	金	土
A	資格1	2遅番	1早番	3夜勤	4							
B	資格2 資格3	1早番	3夜勤	(なし)	4		×					×
C	無資格	2遅番	3夜勤	1早番	5			×				

(凡例) × : 勤務不可日  
記載なし : 勤務可能日

配属先「1早番」について、「配属先希望達成率」は、

$$\text{職員A} = (3-2+1)/3 = 0.66$$

$$\text{職員B} = (2-1+1)/2 = 1.00$$

$$\text{職員C} = (3-3+1)/3 = 0.33$$

これより、日曜日の「1早番」の「配属先毎配属希望率」は、

$$\{(1.0 \times 0.66) + (1.0 \times 1.00) + (1.0 \times 0.33)\} / 3 = 0.67$$

同様に、月曜日の「1早番」の「配属先毎配属希望率」は、

$$\{(1.0 \times 0.66) + (0.0 \times 1.00) + (1.0 \times 0.33)\} / 3 = 0.33$$

(5.2) 入出力仕様

1 入力する諸表

・ 配属先は勤務するにあたり必要な資格を「配属計画表」として用意する。

・ 配属計画表（例）

曜日	配属先	勤務 日数	配属先 最少定 員	必要資格順	必要資 格最少 定員
日	1	1	2	資格1又は資格2	1
				無資格	0
	2	1	2	資格1又は資格2	1
				無資格	0
	3	2	2	資格1又は資格2	1
				無資格	0
月	1	1	2	資格1又は資格2	1
				無資格	0
	2	1	2	資格1又は資格2	1
				無資格	0
	3	2	2	資格1又は資格2	1
				無資格	0
火	1	1	2	資格1又は資格2	1
				無資格	0
	2	1	2	資格1又は資格2	1
				無資格	0
	3	2	2	資格1又は資格2	1
				無資格	0
水	1	1	2	資格1又は資格2	1
				無資格	0
	2	1	2	資格1又は資格2	1
				無資格	0
	3	2	2	資格1又は資格2	1
				無資格	0
木	1	1	2	資格1又は資格2	1
				無資格	0
	2	1	2	資格1又は資格2	1
				無資格	0
	3	2	2	資格1又は資格2	1
				無資格	0
金	1	1	2	資格1又は資格2	1
				無資格	0
	2	1	2	資格1又は資格2	1
				無資格	0
	3	2	2	資格1又は資格2	1
				無資格	0
土	1	1	2	資格1又は資格2	1
				無資格	0
	2	1	2	資格1又は資格2	1
				無資格	0
	3	2	2	資格1又は資格2	1
				無資格	0

(凡例)「配属先」

1 : 早番

2 : 遅番

3 : 夜勤

・各職員は自分の所有資格と、一週間あたりの勤務日数と、勤務したい曜日を「配属先希望表」として用意する。

・配属先希望表（例）

職員	所有資格	配属先希望順			週の勤務日数	曜日毎希望表						
		1	2	3		日	月	火	水	木	金	土
A	資格1	2	1	3	4	-	-	-	-	-	-	-
B	資格2 資格3	1	3	2	4	-	X	-	-	-	-	X
C	無資格	2	3	1	5	X	-	-	-	-	-	-
D	資格2	2	1		3	X	X	-	-	X	-	-
E	無資格	2	1	3	4	-	-	-	-	-	-	-
F	資格1	3	1	2	5	-	-	-	-	-	-	X
G	資格1	2	1	3	3	-	-	-	-	X	X	-
H	資格2 資格3	1	3	2	4	-	-	-	-	-	-	-
I	無資格	2	3	1	5	X	-	-	-	-	-	-
J	資格2	2	1		4	-	-	-	-	-	-	X
K	無資格	2	1	3	4	X	-	-	-	X	X	-
L	資格1	3	1	2	5	X	-	-	-	-	-	-
M	資格2	2	1	3	4	-	-	-	-	-	-	-
N	無資格	2	1	3	4	-	-	-	-	-	-	-
O	無資格	1	2		4	X	-	-	-	-	-	-

・決定済の勤務する曜日と配属先がある場合、「曜日毎配属表」として用意する。

職員	曜日毎配属表						
	日	月	火	水	木	金	土
A	1	2	3	/	-	-	-
B	-	-	-	-	-	-	-
C	-	-	-	-	-	-	-
D	-	-	-	-	-	-	-
E	-	-	-	-	-	-	-
F	-	-	-	1	2	3	/
G	-	-	-	-	-	-	-
H	-	-	-	-	-	-	-
I	-	-	-	-	-	-	-
J	-	-	-	-	-	-	-
K	-	-	-	-	-	-	-
L	-	-	-	-	-	-	-
M	/	-	-	-	1	2	3
N	3	/	2	1	-	-	-
O	-	-	-	-	-	-	-

（凡例）「配属先希望順」

1：早番

2：遅番

3：夜勤

空白：希望なし

（凡例）「曜日毎希望表」

X：勤務不可日

-：勤務可能日

（凡例）「曜日毎配属表」

-：未定

1：早番

2：遅番

3：夜勤

/：夜勤明け



## 2 出力する諸表

・本アルゴリズムで出力するのは、「配属予定表」と「職員毎勤務予定表」である。

・配属予定表（出力例）

曜日	配属先	勤務 日数	配属先 最少定員	必要資格順	必要資 格最少 定員	配属職員		配属先毎配属 希望率	定員充足率	必要資格毎 配属希望率	資格所有者定 員充足率
						1人目	2人目				
日	1	1	2	資格1又は資格2	1	B		0.43	1.00	0.34	1.00
				無資格	0	N				0.09	---
	2	1	2	資格1又は資格2	1	G	J	0.47	1.00	0.33	2.00
				無資格	0					0.13	---
	3	2	2	資格1又は資格2	1	F		0.37	1.00	0.32	1.00
				無資格	0	E				0.04	---
月	1	1	2	資格1又は資格2	1	H	J	0.57	1.50	0.32	2.00
				無資格	0	O				0.24	---
	2	1	2	資格1又は資格2	1	M		0.70	1.50	0.33	1.00
				無資格	0	I	K			0.37	---
	3	2	2	資格1又は資格2	1	L		0.60	1.00	0.34	1.00
				無資格	0	C				0.26	---
火	1	1	2	資格1又は資格2	1	H	F	0.67	1.50	0.42	2.00
				無資格	0	K				0.24	---
	2	1	2	資格1又は資格2	1	J	D	0.79	1.00	0.42	2.00
				無資格	0					0.37	---
	3	2	2	資格1又は資格2	1	G		0.74	1.00	0.49	1.00
				無資格	0	I				0.26	---
水	1	1	2	資格1又は資格2	1	B	D	0.67	1.50	0.42	2.00
				無資格	0	K				0.24	---
	2	1	2	資格1又は資格2	1	M	L	0.79	1.50	0.42	2.00
				無資格	0	E				0.37	---
	3	2	2	資格1又は資格2	1	A		0.74	1.00	0.49	1.00
				無資格	0	N				0.26	---
木	1	1	2	資格1又は資格2	1	B		0.54	1.00	0.34	1.00
				無資格	0	O				0.20	---
	2	1	2	資格1又は資格2	1	J		0.59	1.00	0.29	1.00
				無資格	0	C				0.30	---
	3	2	2	資格1又は資格2	1	F		0.60	1.00	0.37	1.00
				無資格	0	I				0.23	---
金	1	1	2	資格1又は資格2	1	B		0.58	1.00	0.38	1.00
				無資格	0	O				0.20	---
	2	1	2	資格1又は資格2	1	A		0.66	1.00	0.36	1.00
				無資格	0	E				0.30	---
	3	2	2	資格1又は資格2	1	L		0.70	1.00	0.47	1.00
				無資格	0	C				0.23	---
土	1	1	2	資格1又は資格2	1	A		0.52	1.50	0.28	1.00
				無資格	0	O	K			0.24	---
	2	1	2	資格1又は資格2	1	D		0.68	1.00	0.31	1.00
				無資格	0	N				0.37	---
	3	2	2	資格1又は資格2	1	H	M	0.53	1.00	0.28	2.00
				無資格	0					0.26	---

入力の「配属先の必要資格順序表」と同じ

・ 職員毎勤務予定表（出力例）

職員	所有資格	配属先希望順			週の勤務日数	曜日毎配属表							勤務日充足率	希望達成率
		1	2	3		日	月	火	水	木	金	土		
A	資格1	2	1	3	4	-	-	-	3	/	2	1	1.00	0.58
B	資格2 資格3	1	3	2	4	1	x	-	1	1	1	x	1.00	1.00
C	無資格	2	3	1	5	x	3	/	-	2	3	/	1.00	0.73
D	資格2	2	1		3	x	x	2	1	x	-	2	1.00	0.83
E	無資格	2	1	3	4	3	/	-	2	-	2	-	1.00	0.67
F	資格1	3	1	2	5	3	/	1	-	3	/	x	1.00	0.93
G	資格1	2	1	3	3	2	-	3	/	x	x	-	1.00	0.56
H	資格2 資格3	1	3	2	4	/	1	1	-	-	-	3	1.00	0.83
I	無資格	2	3	1	5	x	2	3	/	3	/	-	1.00	0.73
J	資格2	2	1		4	2	1	2	-	2	-	x	1.00	0.88
K	無資格	2	1	3	4	x	2	1	1	x	x	1	1.00	0.75
L	資格1	3	1	2	5	x	3	/	2	-	3	/	1.00	0.87
M	資格2	2	1	3	4	/	2	-	2	-	-	3	1.00	0.67
N	無資格	2	1	3	4	1	-	-	3	/	-	2	1.00	0.58
O	無資格	1	2		4	x	1	-	-	1	1	1	1.00	1.00
(総合計)						15	15	15	15	15	15	15	1.00	0.77

入力の「職員の配属先希望表」と同じ

（凡例）「配属先希望順」

- 1 : 早番
- 2 : 遅番
- 3 : 夜勤
- 空白 : 希望なし

（凡例）「曜日毎希望表」

- X : 勤務不可日
- : 勤務可能日

（凡例）「曜日毎配属表」

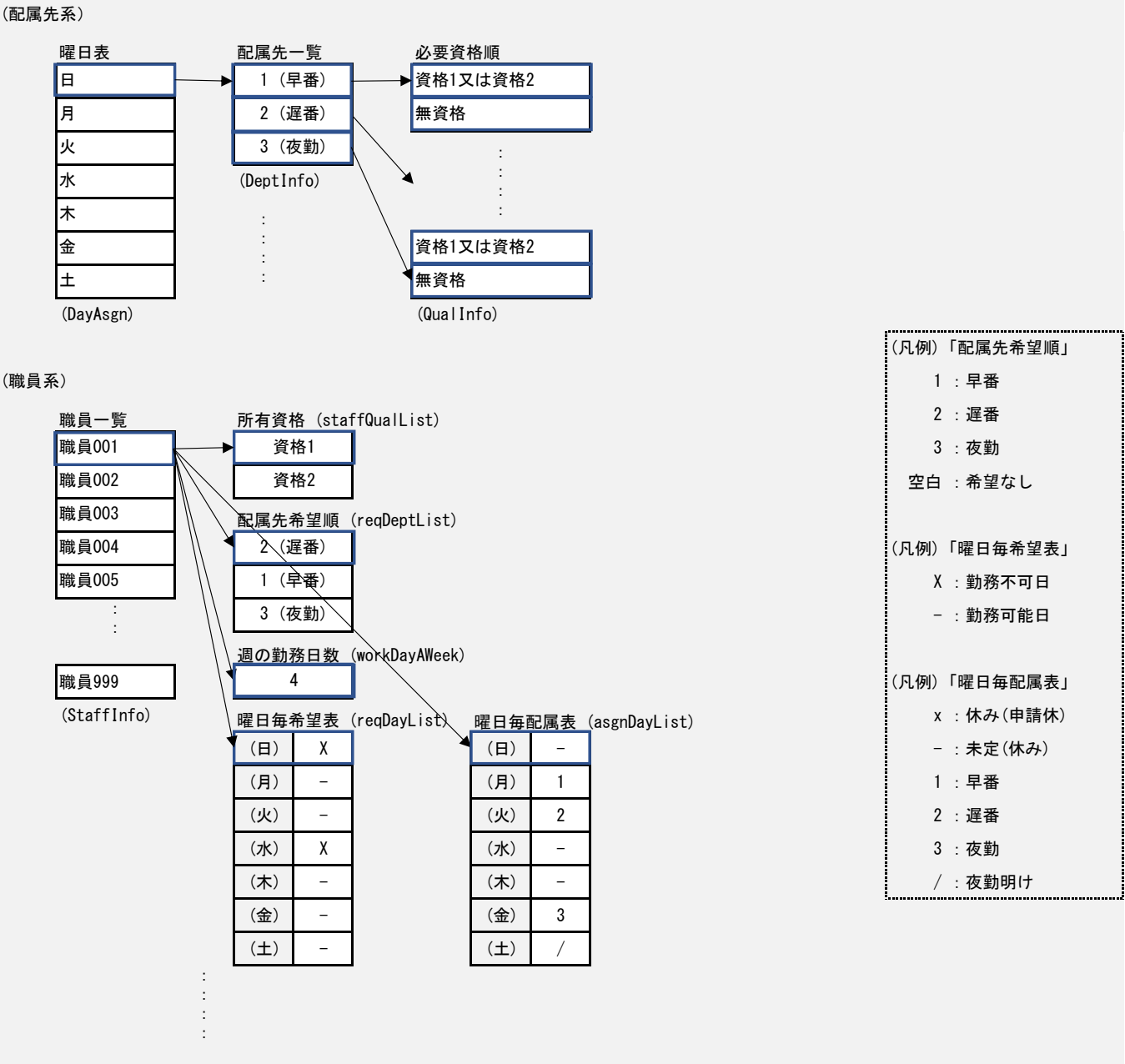
- x : 休み(申請休)
- : 未定(休み)
- 1 : 早番
- 2 : 遅番
- 3 : 夜勤
- / : 夜勤明け

(5.3) 概略データ構造とアルゴリズム

1 概略データ構造

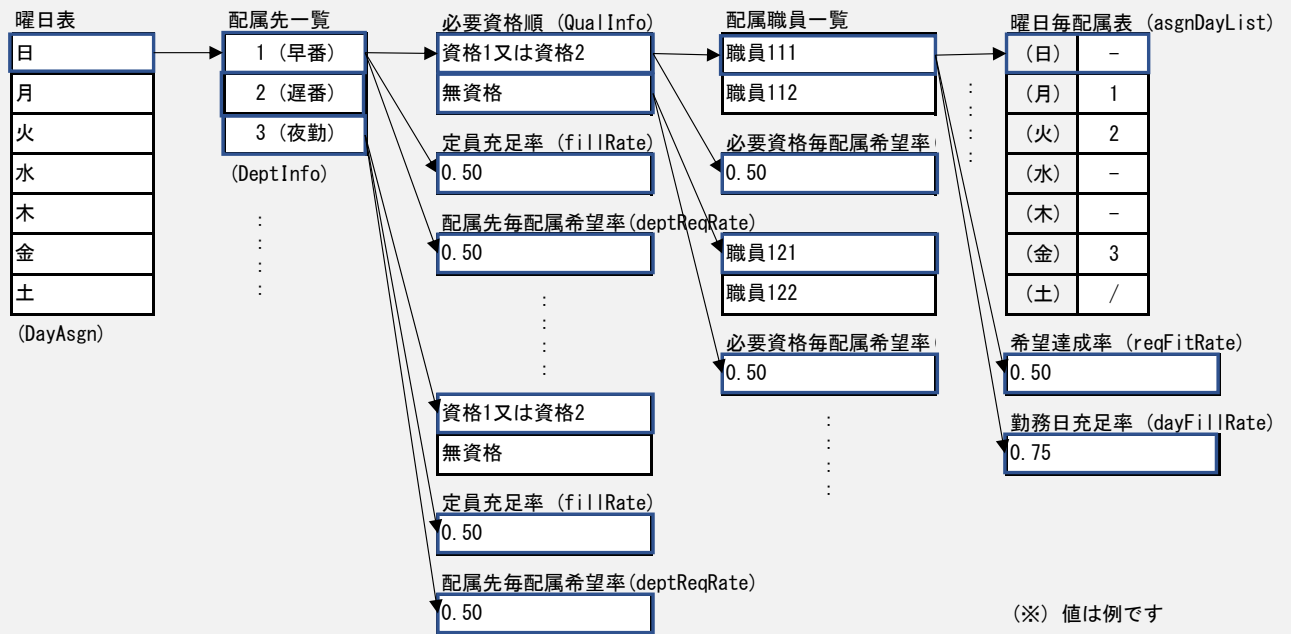
・本アルゴリズムの実装時の概略データ構造を以下に示します。

・入力系のデータ構造



・出力系の概略データ構造

(配属一覧系)



## 2 概略アルゴリズム

- ・本アルゴリズムの実装時の概略を以下に示します。

### 1st) ファイルロード・初期化処理

- ・はじめに、以下のファイルロード・初期化処理を行う。

```
/*=====*/
/* 初期化処理 */
/*=====*/
```

(1) 以下の初期化を行う。

- (1.1) 配属先の「配属計画表」をロードして、配属先情報を初期化する。
- (1.2) 職員の「配属先希望表」をロードして、職員情報を初期化する。
- (1.3) 職員情報で配属先情報を設定する
  - ・職員の「配属先希望順」等で、配属先毎の「配属先毎配属希望率」を設定する。
  - ・職員の「配属先希望順」と「所有資格一覧」等で、配属先の「必要資格」毎の「必要資格毎配属希望率」を設定する。
  - ・職員の「曜日毎配属表」に登録がある場合、配属先の「配属職員一覧」に反映する。

### 2nd) 曜日・配属先・必要資格毎の最少定員の確保

- ・各曜日の各配属先の必要資格毎の定員について、最少定員に充ちるように配員を行う。

```
/*=====*/
/* 「必要資格毎配属希望率」が低い順に、配属先の必要資格を並べ替えた配列を取得*/
/*=====*/
```

(1) (曜日・配属先・必要資格) について「必要資格毎配属希望率」が低い順に並べ替えた配列を取得。

```
/*=====*/
/* 「必要資格毎配属希望率」が低い順に必要資格毎の繰り返し */
/*=====*/
```

(2) 「必要資格毎配属希望率」が低い順に抽出した(曜日・配属先・必要資格) 毎に以下の処理を繰り返す。

- (2.1) 着目する曜日を「曜日W」、配属先を「配属先X」、必要資格を「必要資格Y」とする。
- (2.2) 「曜日W」の「配属先X」の「必要資格Y」についての最少定員割当が済んでいる場合は、次の「必要資格」へ。
- (2.3) 「曜日W」の「配属先X」の「必要資格Y」についての最少定員割当が済んでない場合は、以下の処理を行う。
  - (2.3.1) 「曜日W」の「配属先X」の「必要資格Y」の定員について、最少定員に充ちるか、最少定員を充たすことができないとわかるまで、以下の処理を繰り返す。

```
/*=====*/
/* 「必要資格Y」についての最適職員「職員P」を抽出 */
/*=====*/
```

- (a) 職員の「配属先希望表」をみて、以下の条件を満たす職員「職員P」を探し出す
  - ・「曜日W」に配属済みでない
  - ・「曜日W」が「勤務不可日」でない
  - ・「配属先X」が二日に渡る勤務の場合、
    - ・「曜日W」の次の曜日に配属済みでない
    - ・「曜日W」の次の曜日が「勤務不可日」でない
  - ・「必要資格Y」を「所有資格」が満たしている

- ・仮に配属された場合、（夜勤明けも含めて）「週の勤務日数」の範囲である
- ・上記条件を満たす職員の中で、「配属先X」が「配属先希望順」で最上位である
- ・上記条件を満たす職員候補の中で、「希望達成率」が最も低い職員である

(b) 条件に該当する「職員P」が見つかった時、

(b.1) 「職員P」を「曜日W」の「配属先X」の「必要資格Y」の職員として以下の登録を行う。

- ・「配属先X」の「必要資格Y」の「配属職員一覧」に登録
- ・「配属先X」の「定員充足率」と  
「配属先X」の「必要資格Y」の「資格所有者定員充足率」を更新
- ・「職員P」の「曜日毎配属表」に登録  
(配属先が「夜勤」の場合、「夜勤明け」も登録)
- ・「職員P」の「勤務日充足率」「希望達成率」を更新

(b.2) 最少定員に充ちていなかったら、手順 (a) へ。

(b.4) 最少定員に充ちたら、手順 (2.3.1) を終了し、次の「必要資格」へ。

全ての「必要資格」について処理済みの場合、本手順「2nd)」を終了し、次手順「3rd)」へ。

(c) 条件に該当する「職員P」が見つからない時、

手順 (2.3.1) を終了し、次の「必要資格」へ。

全ての「必要資格」について処理済みの場合、本手順「2nd)」を終了し、次手順「3rd)」へ。

### 3rd) 曜日・配属先毎の最少定員の確保

- ・各曜日の各配属先について、最少定員に充ちるように配員を行う。

```
/*=====*/
/* 配属先について「配属先毎配属希望率」が低い順に並べ替えた配列を取得 */
/*=====*/
```

(1) (曜日・配属先) について「配属先毎配属希望率」が低い順に並べ替えた配列を取得。

```
/*=====*/
/* 「配属先毎配属希望率」が低い順に配属先毎の繰り返し */
/*=====*/
```

(2) 「配属先毎配属希望率」が低い順に抽出した (曜日・配属先) 毎に以下の処理を繰り返す。

(2.1) 着目する曜日を「曜日W」、配属先を「配属先X」とする。

(2.2) 「曜日W」の「配属先X」についての最少定員割当が済んでいる場合は、次の「配属先」へ。

(2.2) 「曜日W」の「配属先X」についての最少定員割当が済んでいない場合は、以下の処理を行う。

(2.3.1) 「曜日W」の「配属先X」の「必要資格Y」の定員について、最少定員に充ちるか、最少定員を充たすことができないとわかるまで、以下の処理を繰り返す。

```
/*=====*/
/* 「曜日W」の「配属先X」についての最適職員「職員P」を抽出 */
/*=====*/
```

(a) 職員の「配属先希望表」をみて、以下の条件を満たす職員「職員P」を探し出す

- ・「曜日W」に配属済みでない
- ・「曜日W」が「勤務不可日」でない
- ・「配属先X」が二日に渡る勤務の場合、
  - ・「曜日W」の次の曜日に配属済みでない
  - ・「曜日W」の次の曜日が「勤務不可日」でない

- ・仮に配属された場合、（夜勤明けも含めて）「週の勤務日数」の範囲である
- ・上記条件を満たす職員の中で、「配属先X」が「配属先希望順」で最上位である
- ・上記条件を満たす職員候補の中で、「希望達成率」が最も低い職員である

(b) 条件に該当する「職員P」が見つかった時、

- (b.1) 「職員P」の所有資格から、配属先の「必要資格Y」を特定する。
- (b.2) 「職員P」を「曜日W」の「配属先X」の「必要資格Y」の職員として以下の登録を行う。
  - ・「配属先X」の「必要資格Y」の「配属職員一覧」に登録
  - ・「配属先X」の「定員充足率」と  
「配属先X」の「必要資格Y」の「資格所有者定員充足率」を更新
  - ・「職員P」の「曜日毎配属表」に登録  
(配属先が「夜勤」の場合、「夜勤明け」も登録)
  - ・「職員P」の「勤務日充足率」「希望達成率」を更新

(b.3) 最少定員に充ちていなかったら、手順 (a) へ。

(b.4) 最少定員に充ちたら、手順 (2.3.1) を終了し、次の「配属先」へ。

全ての「配属先」について処理済みの場合、本手順「3rd)」を終了し、次手順「4th)」へ。

(c) 条件に該当する「職員P」が見つからない時、

手順 (2.3.1) を終了し、次の「配属先」へ。

全ての「配属先」について処理済みの場合、本手順「3rd)」を終了し、次手順「4th)」へ。

#### 4th) 各職員の週の勤務日数調整

- ・各職員の「週の勤務日数」が満ち、かつ、配属先の偏りが少なくなるように以下の調整を行う。

```
/*=====*/
/* 職員毎の繰り返し */
/*=====*/
```

(1) 全ての職員について、以下の処理を繰り返す（着目職員を「職員P」とする）。

- (1.1) 「職員P」の「配属先希望順」に登録が無い場合、何もせず次の職員へ。
- (1.2) 「職員P」の配属済み日数が「週の勤務日数」に達している場合、何もせず次の職員へ。
- (1.3) 「職員P」の配属済み日数が「週の勤務日数」に満たない場合、以下の調整を行う。

```
/*=====*/
/* 配属先について「定員充足率」が低い順に「職員P」の配員を試行 */
/*=====*/
```

- (1.3.1) (曜日、配属先) について「定員充足率」が低い順に並べ替えた配列を取得する。
- (1.3.2) 「定員充足率」が低い順に抽出した (曜日、配属先) 毎に以下の処理を繰り返す。

(1.3.2.1) 曜日を「曜日W」、配属先を「配属先X」とする。

(1.3.2.2) 「曜日W」、「配属先X」の職員として「職員P」の配属処理を以下のように行う。

(a) 職員情報をみて、「職員P」が以下の条件を満たすか？判定する。

- ・「配属先X」が「配属先希望順」リストにある
- ・「曜日W」に配属済みでない
- ・「曜日W」が「勤務不可日」でない
- ・「配属先X」が二日に渡る勤務の場合、
  - ・「曜日W」の次の曜日に配属済みでない
  - ・「曜日W」の次の曜日が「勤務不可日」でない

- ・「必要資格Y」を「所有資格」が満たしている
- ・仮に配属された場合、（夜勤明けも含めて）「週の勤務日数」の範囲である

- (b) 「職員P」が上記条件に該当する時、
- (b.1) 「職員P」の所有資格から、配属先の「必要資格Y」を特定する。
- (b.2) 「職員P」を「曜日W」の「配属先X」の「必要資格Y」の職員として以下の登録を行う。
- ・「配属先X」の「必要資格Y」の「配属職員一覧」に登録
  - ・「配属先X」の「定員充足率」と  
「配属先X」の「必要資格Y」の「資格所有者定員充足率」を更新
  - ・「職員P」の「曜日毎配属表」に登録  
（配属先が「夜勤」の場合、「夜勤明け」も登録）
  - ・「職員P」の「勤務日充足率」「希望達成率」を更新
- (b.3) 「職員P」の配属済み日数が「週の勤務日数」に達した場合、次の「職員」へ。
- (b.4) 「職員P」の配属済み日数が「週の勤務日数」に達していない場合、次の「配属先」へ。  
全ての「配属先」について処理済みの場合、次の「職員」へ。
- (c) 「職員P」が上記条件に該当しない場合、次の「配属先」へ。  
全ての「配属先」について処理済みの場合、次の「職員」へ。

#### 5th) 各職員の「勤務日充足率」と各配属先の「定員充足率」をチェック

- ・各職員の「勤務日充足率」と各配属先の「定員充足率」をチェックし、充足率が不足している場合、その旨を出力する。

```

/*=====*/
/* 全ての職員について「勤務日充足率」をチェック */
/*=====*/
(1) 全ての職員について、以下の処理を繰り返す（着目職員を「職員P」とする）。

(1.1) 「職員P」の「配属先希望順」に登録が無い場合、何もせず次の職員へ。
(1.2) 「職員P」の「週の勤務日数」が0の場合、何もせず次の職員へ。
(1.3) 「職員P」の「勤務日充足率」が 1.0（100%）未満ならば、その旨をメッセージ出力する。

/*=====*/
/* 全ての配属先について「定員充足率」をチェック */
/*=====*/
(2) 全ての曜日について、以下の処理を繰り返す（着目曜日を「曜日W」とする）。

(2.1) 「曜日W」の全ての配属先について、以下の処理を繰り返す（着目配属先を「配属先X」とする）。
(2.1.1) 「配属先X」の「必要資格順」に登録が無い場合、何もせず次の配属先へ。
(2.1.2) 「配属先X」の「配属先最少定員」が0の場合、何もせず次の配属先へ。
(2.1.3) 「配属先X」の「定員充足率」が 1.0（100%）未満ならば、その旨をメッセージ出力する。

```



## (5.4) 実装

以下では、既述の「(5.3) 概略データ構造とアルゴリズム」に基づいて Python で実装したものを掲載します。

本実装では、ファイルインタフェース、GUI のインタフェースはなく、「リスト\_職員の一週間単位の配属\_実装\_04\_入力関数定義と入力値.txt」で、入力値を指定するようにしています。

(リスト\_職員の一週間単位の配属\_実装\_01\_定数定義.txt)

```
*****
# リスト_職員の一週間単位の配属 (part01)
#-----
# 定数定義
*****
import numpy as np
import time

*****
# 定数
*****
# 配属先ID
DEPT_ID_X = "x" # "休み(申請休)"
DEPT_ID_0 = "-" # "休み(申請以外)"
DEPT_ID_1 = "1" # "早番"
DEPT_ID_2 = "2" # "遅番"
DEPT_ID_3 = "3" # "夜勤"
DEPT_ID_4 = "/" # "夜勤明け ("夜勤" の翌日は "夜勤明け")

# 配属先の必要資格ID
QUAL_DEPT_0 = "000" # "無資格"
QUAL_DEPT_1_2 = "012" # "初任者・実務者研修"

# 職員の所有資格ID
QUAL_STAFF_0 = "0" # "無資格"
QUAL_STAFF_1 = "1" # "初任者研修"
QUAL_STAFF_2 = "2" # "実務者研修"
QUAL_STAFF_3 = "3" # "その他"

# 希望勤務状態ID
AVAIL_X = "X" # 勤務不可日
AVAIL_A = "-" # 勤務可能日

# 曜日ID (曜日リスト参照添え字としても使用)
DAY_0 = 0 # 日 "SUN"
DAY_1 = 1 # 月 "MON"
DAY_2 = 2 # 火 "TUE"
DAY_3 = 3 # 水 "WED"
DAY_4 = 4 # 木 "THU"
DAY_5 = 5 # 金 "FRI"
DAY_6 = 6 # 土 "SAT"

*****
# 共通メソッド: ConvDayIdToName
# 曜日IDを名前へ変換する
#-----
# 引数:
#   dayId : 曜日ID
#-----
# 戻り値:
#   dayName : 曜日名
*****
def ConvDayIdToName(dayId):
    if( dayId == DAY_0 ):
        dayName = "日"
    elif( dayId == DAY_1 ):
        dayName = "月"
    elif( dayId == DAY_2 ):
        dayName = "火"
    elif( dayId == DAY_3 ):
        dayName = "水"
    elif( dayId == DAY_4 ):
        dayName = "木"
```

```

elif( dayId == DAY_5 ) :
    dayName = "金"
elif( dayId == DAY_6 ) :
    dayName = "土"
else :
    dayName = ""
return dayName

#*****
# 共通メソッド : ConvStaffQualIdToName
# 職員の所有資格IDを名前へ変換する
#-----
# 引数 :
#   qualId   : 職員の所有資格ID
#-----
# 戻り値 :
#   qualName : 職員の所有資格名
#*****
def ConvStaffQualIdToName(qualId):
    if( qualId == QUAL_STAFF_0 ) :
        qualName = "無資格"
    elif( qualId == QUAL_STAFF_1 ) :
        qualName = "初任者研修"
    elif( qualId == QUAL_STAFF_2 ) :
        qualName = "実務者研修"
    elif( qualId == QUAL_STAFF_3 ) :
        qualName = "その他"
    else :
        qualName = ""
    return qualName

#*****
# 共通メソッド : ConvDeptQualIdToName
# 配属先の必要資格IDを名前へ変換する
#-----
# 引数 :
#   qualId   : 配属先の必要資格ID
#-----
# 戻り値 :
#   qualName : 配属先の必要資格名
#*****
def ConvDeptQualIdToName(qualId):
    if( qualId == QUAL_DEPT_0 ) :
        qualName = "無資格"
    elif( qualId == QUAL_DEPT_1_2 ) :
        qualName = "初任者・実務者研修"
    else :
        qualName = ""
    return qualName

#*****
# 共通メソッド : ConvDeptIdToName
# 配属先IDを配属先配属先名へ変換する
#-----
# 引数 :
#   deptId   : 配属先ID
#-----
# 戻り値 :
#   deptName : 配属先配属先名
#*****
def ConvDeptIdToName(deptId):
    if( deptId == DEPT_ID_X ) :
        deptName = "休み(申請休)"
    elif( deptId == DEPT_ID_0 ) :
        deptName = "休み(申請以外)"
    elif( deptId == DEPT_ID_1 ) :
        deptName = "早番"
    elif( deptId == DEPT_ID_2 ) :
        deptName = "遅番"
    elif( deptId == DEPT_ID_3 ) :
        deptName = "夜勤"
    elif( deptId == DEPT_ID_4 ) :
        deptName = "夜勤明け"
    else :
        deptName = ""
    return deptName

```

```

*****
# リスト_職員の一週間単位の配属 (part02)
#
# クラス定義
#
*****

# クラス      StaffInfo
#
# 職員情報クラス
#
#
*****
class StaffInfo:

    #=====
    # StaffInfo クラス : コンストラクタ
    #
    # 以下のインスタンス変数を定義し、指定値で初期化する :
    #
    # staffId      : (PK) 職員ID
    # staffName    : 職員名称
    # staffQualList : 所有資格一覧 (配列型 [] で QUAL_STAFF_* を保存)
    # deptQualList : 配属先の適合資格一覧 (配列型 [] で QUAL_DEPT_* を保存)
    # workDayAWeek : 週の勤務日数
    # reqDeptList  : 配属先希望順 (配列型 [] で DEPT_ID_* を希望順に保存)
    # reqDayList   : 曜日毎希望表 (配列型 [] で一週間分保存、値 : 希望勤務状態ID (AVAIL_*))
    # asgnDayList  : 曜日毎配属表 (配列型 [] で一週間分保存、値 : 配属先ID (DEPT_ID_*)、初期値無しの場合は None)
    # dayFillRate  : 勤務日充足率 (%)
    # reqFitRate   : 希望達成率 (%)
    #=====

    def __init__(self, staffId, staffName,
                  staffQualList, workDayAWeek, reqDeptList, reqDayList, asgnDayList):
        self.staffId = staffId
        self.staffName = staffName
        self.staffQualList = staffQualList
        self.workDayAWeek = workDayAWeek
        self.reqDeptList = reqDeptList
        self.reqDayList = reqDayList
        if (asgnDayList == None):
            self.asgnDayList = [DEPT_ID_0, DEPT_ID_0, DEPT_ID_0, DEPT_ID_0, DEPT_ID_0, DEPT_ID_0, DEPT_ID_0]
        else:
            self.asgnDayList = asgnDayList
        self.dayFillRate = 0.0
        self.reqFitRate = 0.0

        # 「配属先の適合資格一覧」を「所有資格一覧」から作成
        self.deptQualList = []
        qual1or2 = False
        for aStaffQual in self.staffQualList:
            if (aStaffQual == QUAL_STAFF_1) or (aStaffQual == QUAL_STAFF_2):
                qual1or2 = True
                break

        if (qual1or2 == True):
            self.deptQualList.append( QUAL_DEPT_1_2 )

    #=====
    # StaffInfo クラス : printObj
    # StaffInfo インスタンスの内容を印字する (改行しない)
    #
    # 引数 : なし
    #
    # 戻り値 : なし
    #
    #=====
    def printObj(self):
        # 職員ID、職員名
        print("{0}:{1}:".format(self.staffId, self.staffName), end="")

        # 週当りの勤務日数
        print(" 週当り勤務日数={0} ".format(self.workDayAWeek), end="")

        # reqDayList: 曜日毎希望表 (配列型 [] で一週間分保存、値 : 希望勤務状態ID (AVAIL_*))
        print(" 勤務希望=", end="")
        idx = 0

```

```

for wd in self.reqDayList:
    if (idx > 0):
        print(", ", end="")
        print("{0}".format(wd), end="")
        idx = idx + 1
print("") ", end=")

# asgnDayList: 曜日毎配属表 (配列型 [] で一週間分保存、値: 配属先ID (DEPT_ID_*))
print(" 勤務予定=( ", end="")
idx = 0
for wd in self.asgnDayList:
    if (idx > 0):
        print(", ", end="")
        print("{0}".format(wd), end="")
        idx = idx + 1
print("") ", end=")

# 勤務日充足率
print(" 勤務日充足率={0:.2f}".format(self.dayFillRate), end="")

# 希望達成率
print(" 希望達成率={0:.2f}".format(self.reqFitRate), end="")

# 配属先希望順
print(" 配属先希望順=( ", end="")
idx = 0
for ad in self.reqDeptList:
    if (idx > 0):
        print(", ", end="")
        print("{0}".format(ad), end="")
        idx = idx + 1
print("") ", end=")

# 所有資格一覧
print(" 所有資格=( ", end="")
if ( len(self.staffQualList) == 0 ):
    print("なし", end="")
else:
    idx = 0
    for ql in self.staffQualList:
        if (idx > 0):
            print(", ", end="")
            print("{0}".format(ConvStaffQualIdToName(ql)), end="")
            idx = idx + 1
print("") ", end=")

# 適合資格一覧
print(" 適合資格=( ", end="")
if ( len(self.deptQualList) == 0 ):
    print("なし", end="")
else:
    idx = 0
    for ql in self.deptQualList:
        if (idx > 0):
            print(", ", end="")
            print("{0}".format(ConvDeptQualIdToName(ql)), end="")
            idx = idx + 1
print("") ", end=")

# 改行
print("")

```

```

#####
# クラス      BestStaffInfo
# -----
# 最適職員情報クラス
# -----
#####
class BestStaffInfo ():
    #=====
    # BestStaffInfo クラス : コンストラクタ
    # -----
    # 以下のインスタンス変数を定義し、指定値で初期化する :
    #
    # staffInfo      : 職員情報インスタンス (StaffInfo)
    # deptIndex      : 配属先希望順位 (配列添え字)

```

```

#=====
def __init__(self, staffInfo, deptIndex ):
    self.staffInfo = staffInfo
    self.deptIndex = deptIndex

#*****
# クラス      QualInfo
#-----
# 配属先の必要資格のクラス
#-----
#-----
#*****
class QualInfo:

#*****
# QualInfo クラス : コンストラクタ
#-----
# 以下のインスタンス変数を定義し、指定値で初期化する :
#
# dayId          : (PK) 曜日ID (DAY_*)
# deptId         : (PK) 配属先ID (DEPT_ID_*)
# qualId         : (PK) 配属先の必要資格ID (QUAL_DEPT_*)
# parentObj      : 所属する配属先オブジェクト (DeptInfo)
# reqCapaMin     : 資格所有者最少定員
# reqCapaRate    : 資格所有者定員充足率
# qualReqRate    : 必要資格毎配属希望率
# staffList      : 配属職員一覧 (StaffInfo の配列 [])
#*****
def __init__(self, dayId, deptId, qualId, reqCapaMin):
    self.dayId = dayId
    self.deptId = deptId
    self.qualId = qualId
    self.parentObj = None
    self.reqCapaMin = reqCapaMin
    self.reqCapaRate = 0.0
    self.qualReqRate = 0.0
    self.staffList = []

#=====
# QualInfo クラス : printObj
# QualInfo インスタンスの内容を印字する (改行しない)
#-----
# 引数 : なし
#-----
# 戻り値 : なし
#=====
def printObj(self):
    print("{0} : 最少定員={1} 名、資格定員充足率=".format(
        ConvDeptQualIdToName(self.qualId), self.reqCapaMin), end="")

    # 定員充足率
    if ( self.reqCapaMin <= 0 ):
        print("----", end="")
    else:
        print("{0:.2f}".format(self.reqCapaRate), end="")

    # 必要資格毎配属希望率
    print("、必要資格毎配属希望率={0:.2f}".format(self.qualReqRate), end="")

    # 配属職員一覧
    if len(self.staffList) > 0:
        print(" 配属職員=( ", end="")
        for s in self.staffList:
            print("{0}:{1} ".format(s.staffId, s.staffName), end="")
        print(")", end="")

#=====
# QualInfo クラス : setParent
# 所属する配属先オブジェクト (DeptInfo) を設定する。
#-----
# 引数 :
# parentObj : 所属する配属先オブジェクト (DeptInfo)
#-----
# 戻り値 : なし
#=====
def setParent(self, parentObj):
    self.parentObj = parentObj

```

```

#*****
# クラス      DeptInfo
#-----
# 配属先情報のクラス
#-----
#
#*****
class DeptInfo:

    #=====
    # DeptInfo クラス : コンストラクタ
    #-----
    # 以下のインスタンス変数を定義し、指定値で初期化する :
    #
    # dayId      : (PK) 曜日ID (DAY_*)
    # deptId     : (PK) 配属先ID (DEPT_ID_*)
    # occupyDays : 配属先の一勤務当たりの勤務日数
    # capaMin    : 配属先最少定員
    # deptReqRate : 配属先毎配属希望率
    # fillRate   : 定員充足率
    # qualInfoList : 必要資格順 (QualInfo インスタンスを優先順に並べた配列 [])
    #=====
    def __init__(self, dayId, deptId, occupyDays, capaMin, qualInfoList):
        self.dayId = dayId
        self.deptId = deptId
        self.occupyDays = occupyDays
        self.capaMin = capaMin
        self.deptReqRate = 0.0
        self.fillRate = 0.0
        self.qualInfoList = qualInfoList
        for aDeptQual in self.qualInfoList :
            aDeptQual.setParent( self )

    #=====
    # DeptInfo クラス : printObj
    # DeptInfo インスタンスの内容を印字する (改行する)
    #-----
    # 引数 : なし
    #-----
    # 戻り値 : なし
    #-----
    def printObj(self):
        print("({0}: {1} {2}): 勤務日数={3} 日, 最少定員={4} 名, 配属先毎配属希望率={5:.2f}, 定員充足率={6:.2f}, ".format(
            ConvDayIdToName(self.dayId), self.deptId, ConvDeptIdToName(self.deptId),
            self.occupyDays, self.capaMin, self.deptReqRate, self.fillRate), end="")

        # 必要資格順
        idx = 0
        for qi in self.qualInfoList:
            if ( idx == 0 ) :
                print("必要資格=( ", end="")
            else:
                print(" / ", end="")
            qi.printObj()
            idx = idx + 1
        print(" ")

#*****
# クラス      DayAssign
#-----
# 曜日ごとの配属情報のクラス
#-----
#
#*****
class DayAssign:

    #=====
    # DayAssign クラス : コンストラクタ
    #-----
    # 以下のインスタンス変数を定義し、指定値で初期化する :
    #
    # dayId      : (PK) 曜日ID (DAY_*)
    # deptInfoList : 当該曜日の配属先情報の配列 (DeptInfo の配列 [])
    #=====
    def __init__(self, dayId, deptInfoList):
        self.dayId = dayId

```

```
self.deptInfoList = deptInfoList

#=====
# DayAssign クラス : printObj
# DayAssign インスタンスの内容を印字する（改行する）
#=====
# 引数：なし
#
# 戻り値：なし
#=====
def printObj(self):
    print("{0}: ".format(ConvDayIdToName(self.dayId)))

# 配属先情報
for aDept in self.deptInfoList:
    print(" ", end=" ")
    aDept.printObj()
```

```

*****
# リスト_職員の一週間単位の配属 (part03)
#
# 共通関数定義
#
*****

# 共通メソッド : GetIndexOnArray
# 配列で指定した値が登録されていれば、その先頭の添え字を返す。
# 登録されていなければ -1 を返す
#
# 引数 :
#   array   : 検査する配列
#   value   : 一致確認する配列の要素値
#
# 戻り値 :
#   retii   : 配列上で値の一致が確認できた先頭要素の添え字 (一致しなければ -1)
*****
def GetIndexOnArray(array, value):
    retii = -1
    refii = -1
    for aElm in array :
        refii = refii + 1
        if ( aElm == value ) :
            retii = refii
            break
    return retii

*****
# メソッド : GetDeptFromDayIDandDeptID
# 引数で指定した曜日ID (DAY_*) と配属先ID (DEPT_ID_*) から
# 対応する「配属先」を取得する。
#
# 引数 :
#   dayAssignList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAssign)
#   dayId         : 曜日ID (DAY_*)
#   deptId        : 配属先ID (DEPT_ID_*)
#
# 戻り値 :
#   aDept         : 指定した曜日ID と配属先ID に対応する「配属先情報」オブジェクト (DeptInfo)
*****
def GetDeptFromDayIDandDeptID( dayAssignList, dayId, deptId ) :

    # 指定曜日の配属情報を取得
    aDayAssign = dayAssignList[dayId]
    for aDept in aDayAssign.deptInfoList :
        if ( aDept.deptId == deptId ) :
            return aDept

    # 該当する配属先が無い場合、None を返す
    return None

*****
# メソッド : PrintDayAssignList
# 「配属予定表」を印字する
#
# 引数 :
#   dayAssignList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAssign)
#
# 戻り値 : なし
*****
def PrintDayAssignList(dayAssignList) :

    print("=====")
    print("= 配属先一覧=")
    print("=====")
    for dayAssign in dayAssignList:
        dayAssign.printObj()

*****
# メソッド : PrintStaffInfoList
# 職員一覧を印字する
#

```



```

# 引数 :
#   staffInfoList : 職員オブジェクト配列 (配列要素は StaffInfo)
# -----
# 戻り値 : なし
#*****
def PrintStaffInfoList(staffInfoList) :

    print("=====")
    print("= 職員一覧")
    print("=====")
    for staff in staffInfoList:
        staff.printObj()

#*****
# メソッド : UpdateStaffRate
#   指定した職員の「勤務日充足率」「希望達成率」を更新する。
# -----
# 引数 :
#   aStaff : 更新対象の職員 (StaffInfo)
# -----
# 戻り値 : なし
#*****
def UpdateStaffRate( aStaff ) :

    reqFitRate = 0.0
    asgnDays = 0
    reqDeptLen = len(aStaff.reqDeptList) # 配属先希望順の登録数
    if ( reqDeptLen > 0 ) :
        for deptId in aStaff.asgnDayList :
            asgnDept = None

            # “休み” 以外で更新
            if ( (deptId != DEPT_ID_0) and (deptId != DEPT_ID_X) ) :
                asgnDays = asgnDays + 1

            # “夜勤明け” は “夜勤” での希望相当
            asgnDept = deptId
            if (deptId == DEPT_ID_4) :
                asgnDept = DEPT_ID_3

            # 配属先希望達成率を累積
            asgnIndex = GetIndexOnArray(aStaff.reqDeptList, asgnDept)
            reqFitRate = reqFitRate + (reqDeptLen - asgnIndex) / reqDeptLen

    # 勤務日充足率・希望達成率を更新
    if ( aStaff.workDayAWeek > 0 ) :
        aStaff.reqFitRate = reqFitRate / aStaff.workDayAWeek
        aStaff.dayFillRate = asgnDays / aStaff.workDayAWeek

#*****
# メソッド : UpdateDeptFillRate
#   指定した配属先の「定員充足率」と
#   必要資格毎の「資格所有者定員充足率」を更新する。
# -----
# 引数 :
#   aDept : 更新対象の配属先 (DeptInfo)
# -----
# 戻り値 : なし
#*****
def UpdateDeptFillRate( aDept ) :

    if( aDept.capaMin > 0 ) :
        asgnNo = 0
        for qi in aDept.qualInfoList:
            asgnNo = asgnNo + len(qi.staffList)

        # 資格所有者定員充足率
        if ( qi.reqCapaMin > 0 ) :
            qi.reqCapaRate = len(qi.staffList) / qi.reqCapaMin

        aDept.fillRate = asgnNo / aDept.capaMin
    else :
        aDept.fillRate = 0.0

#*****

```

```

# メソッド : SetDeptReqRate
#  職員の「配属先希望順」等で、配属先毎の「配属先毎配属希望率」を設定
# -----
# 引数 :
#  dayAsgnList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAsgn)
#  staffInfoList : 「職員情報」オブジェクト配列 (配列要素は StaffInfo)
# -----
# 戻り値 : なし
# *****
def SetDeptReqRate( dayAsgnList, staffInfoList ) :
    # 職員総数
    staffTotalNo = len(staffInfoList)

    # 曜日毎の繰り返し⇒「曜日W」
    for aDayAsgn in dayAsgnList:
        dayId = aDayAsgn.dayId

        # 配属先毎の繰り返し⇒「配属先X」
        for aDept in aDayAsgn.deptInfoList :
            deptId = aDept.deptId

            # 全ての職員について繰り返し
            for aStaff in staffInfoList:

                # 配属先希望順の登録数
                reqDeptLen = len(aStaff.reqDeptList)
                if ( reqDeptLen == 0 ) :
                    continue

                # 職員は勤務不可日？
                if ( aStaff.reqDayList[dayId] == AVAIL_X ) :
                    continue

                # 職員の配属先希望達成率を算出
                asgnIndex = GetIndexOnArray(aStaff.reqDeptList, deptId)
                reqFitRate = (reqDeptLen - asgnIndex) / reqDeptLen

                # 配属先希望達成率を累積して「配属先毎配属希望率」を算出
                aDept.deptReqRate = aDept.deptReqRate + (reqFitRate / staffTotalNo)

# *****
# メソッド : SetQualReqRate
#  職員の「配属先希望順」と「所有資格一覧」等で、
#  配属先の「必要資格」毎の「必要資格毎配属希望率」を設定する。
# -----
# 引数 :
#  dayAsgnList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAsgn)
#  staffInfoList : 「職員情報」オブジェクト配列 (配列要素は StaffInfo)
# -----
# 戻り値 : なし
# *****
def SetQualReqRate( dayAsgnList, staffInfoList ) :
    # 職員総数
    staffTotalNo = len(staffInfoList)

    # 曜日毎の繰り返し⇒「曜日W」
    for aDayAsgn in dayAsgnList:
        dayId = aDayAsgn.dayId

        # 配属先毎の繰り返し⇒「配属先X」
        for aDept in aDayAsgn.deptInfoList :
            deptId = aDept.deptId

            # 必要資格毎の繰り返し⇒「必要資格Y」
            for aDeptQual in aDept.qualInfoList:
                aDeptQual.qualReqRate = 0.0
                aDeptQualId = aDeptQual.qualId

            # 全ての職員について繰り返し
            for aStaff in staffInfoList:

                # 配属先希望順の登録数
                reqDeptLen = len(aStaff.reqDeptList)
                if ( reqDeptLen == 0 ) :
                    continue

                # 職員は勤務不可日？

```

```

        if ( aStaff.reqDayList[dayId] == AVAIL_X ) :
            continue

        # 職員の「所有資格」が配属先の「必要資格Y」を満たしている？
        qualOK = False
        if (aDeptQualId == QUAL_DEPT_1_2) :
            for aStaffDeptQual in aStaff.deptQualList :
                if (aStaffDeptQual == aDeptQualId) :
                    qualOK = True
                    break
        elif (aDeptQualId == QUAL_DEPT_0) :
            if (len(aStaff.deptQualList) == 0) :
                qualOK = True

        if ( qualOK == False ) :
            continue

        # 職員の配属先希望達成率を算出
        asgnIndex = GetIndexOnArray(aStaff.reqDeptList, deptId)
        reqFitRate = (reqDeptLen - asgnIndex) / reqDeptLen

        # 配属先希望達成率を累積して「必要資格毎配属希望率」を算出
        aDeptQual.qualReqRate = aDeptQual.qualReqRate + (reqFitRate / staffTotalNo)

#####
# メソッド : SetStaffAsgnDayListToDeptStaffList
# 職員の「曜日毎配属表」で、配属先の「配属職員一覧」を設定する。
#-----
# 引数 :
#   dayAsgnList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAsgn)
#   staffInfoList : 「職員情報」オブジェクト配列 (配列要素は StaffInfo)
#-----
# 戻り値 :
#   なし
#####
def SetStaffAsgnDayListToDeptStaffList( dayAsgnList, staffInfoList ) :

    # 職員についての繰り返し
    for aStaff in staffInfoList:

        # 「曜日毎配属表」の曜日ごとの繰り返し
        if ( aStaff.asgnDayList != None ) :

            # 曜日毎の繰り返し⇒「曜日W」
            for aDayAsgn in dayAsgnList:
                dayId = aDayAsgn.dayId

                # 「曜日毎配属表」に既登録の配属先指定が無い場合、
                # 「曜日毎希望表」で「勤務不可日」の場合、「休み(申請休)」とする
                if ( aStaff.asgnDayList[dayId] == DEPT_ID_0 ) :
                    if ( aStaff.reqDayList[dayId] == AVAIL_X ) :
                        aStaff.asgnDayList[dayId] = DEPT_ID_X

                # 「曜日毎配属表」に既登録の配属先指定がある場合
                else :
                    # 曜日IDと配属先IDから対応する「配属先」を取得
                    aDept = GetDeptFromDayIDandDeptID( dayAsgnList, dayId, aStaff.asgnDayList[dayId] )
                    if ( aDept != None ) :
                        #Debug
                        #aDept.printObj()
                        #End Debug

                        # 配属先の「配属職員一覧」に登録
                        RegistStaffToDeptStaffList( aStaff, aDept )

                        # 配属先の「定員充足率」と必要資格毎の「資格所有者定員充足率」を更新
                        UpdateDeptFillRate( aDept )

            # 職員の「勤務日充足率」「希望達成率」を更新
            UpdateStaffRate( aStaff )

#####
# メソッド : SetDeptInfoWithStaffInfo
# 職員情報で配属先情報を設定する
#-----

```

```

# 引数 :
#   dayAssignList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAssign)
#   staffInfoList : 「職員情報」オブジェクト配列 (配列要素は StaffInfo)
# -----
# 戻り値 :
#   なし
# *****
def SetDeptInfoWithStaffInfo( dayAssignList, staffInfoList ) :

    # 職員の「配属先希望順」等で、配属先毎の「配属先毎配属希望率」を設定
    SetDeptReqRate( dayAssignList, staffInfoList )

    # 職員の「配属先希望順」と「所有資格一覧」等で、
    # 配属先の「必要資格」毎の「必要資格毎配属希望率」を設定
    SetQualReqRate( dayAssignList, staffInfoList )

    # 職員の「曜日毎配属表」で、配属先の「配属職員一覧」を設定
    SetStaffAsgnDayListToDeptStaffList( dayAssignList, staffInfoList )

# *****
# メソッド : IsAvailableStaff
#   引数で指定した職員が、
#   引数で指定した曜日「曜日W」の配属先「配属先X」の必要資格「必要資格Y」
#   を持った職員として配属するのに、以下の配属条件を満たすか判定する。
#
#   【配属条件】
#   ・「配属先X」が「配属先希望順」リストにある
#   ・「曜日W」に配属済みでない
#   ・「曜日W」が「勤務不可日」でない
#   ・「配属先X」が二日に渡る勤務の場合、「曜日W」の次の曜日に配属済みでない
#   ・「配属先X」が二日に渡る勤務の場合、「曜日W」の次の曜日が「勤務不可日」でない
#   ・「必要資格Y」を「所有資格」が満たしている
#   ・仮に配属された場合、(夜勤明けも含めて)「週の勤務日数」の範囲である
# -----
# 引数 :
#   aStaff      : 職員オブジェクト (StaffInfo)
#   dayId       : 曜日ID (DAY_*)
#   deptId      : 配属先ID (DEPT_ID_*)
#   occupyDays  : 配属先の一勤務当たりの勤務日数
#   aDeptQualId : 配属先の必要資格ID (QUAL_DEPT_*, 必要資格無関係の場合は None)
# -----
# 戻り値 :
#   isAvalilable = True:配属条件を満たす、False:配属条件を満たさない
# *****
def IsAvailableStaff(aStaff, dayId, deptId, occupyDays, aDeptQualId) :
    isAvalilable = True

    # 「配属先X」が「配属先希望順」リストにある
    if( isAvalilable == True ) :
        deptIndex = GetIndexOnArray(aStaff.reqDeptList, deptId)
        if (deptIndex == -1) :
            isAvalilable = False

    # 「曜日W」に配属済みでない?
    if( isAvalilable == True ) :
        if (aStaff.asgnDayList[dayId] != DEPT_ID_0) :
            isAvalilable = False

    # 「曜日W」が「勤務不可日」でない?
    if( isAvalilable == True ) :
        if (aStaff.reqDayList[dayId] == AVAIL_X) :
            isAvalilable = False

    # 1勤務が2日に渡る配属先?
    if( isAvalilable == True ) :
        if (occupyDays > 1) :

            # 「曜日W」の翌日に配属済みでない?
            if (dayId + 1 < len(aStaff.asgnDayList) ) :
                if ( aStaff.asgnDayList[dayId + 1] != DEPT_ID_0 ) :
                    isAvalilable = False
            else:
                if ( aStaff.asgnDayList[0] != DEPT_ID_0 ) :
                    isAvalilable = False

    # 「曜日W」の翌日が「勤務不可日」でない?

```

```

        if (dayId + 1 < len(aStaff.reqDayList) ):
            if( aStaff.reqDayList[dayId + 1] == AVAIL_X) :
                isAvalilable = False
            else:
                if( aStaff.reqDayList[0] == AVAIL_X) :
                    isAvalilable = False

# 「必要資格 Y」を職員の「適合資格」が満たしている？
if( isAvalilable == True ) :
    qualOK = False
    if (aDeptQualId is None):
        qualOK = True
    elif (aDeptQualId == QUAL_DEPT_1_2) :
        for aStaffDeptQual in aStaff.deptQualList :
            if (aStaffDeptQual == aDeptQualId) :
                qualOK = True
                break
    elif (aDeptQualId == QUAL_DEPT_0):
        if (len(aStaff.deptQualList) == 0):
            qualOK = True

    if (qualOK == False) :
        isAvalilable = False

# 仮に配属された場合、「週の勤務日数」の範囲である
if( isAvalilable == True ) :
    asgnDayTotal = len(aStaff.asgnDayList) ￥
    - aStaff.asgnDayList.count(DEPT_ID_0) ￥
    - aStaff.asgnDayList.count(DEPT_ID_X)
    asgnDayTotal += occupyDays
    if (aStaff.workDayAWeek < asgnDayTotal) :
        isAvalilable = False

return isAvalilable

#*****
# メソッド : ExtractBestStaff
# 引数で指定した曜日「曜日W」の配属先「配属先X」へ必要資格「必要資格Y」
# を持った職員として配属するのに、以下の条件を満たす職員を、最適職員として抽出する。
#
# 【条件】
# ・「曜日W」に配属済みでない
# ・「曜日W」が「勤務不可日」でない
# ・「配属先X」が二日に渡る勤務の場合、「曜日W」の次の曜日に配属済みでない
# ・「配属先X」が二日に渡る勤務の場合、「曜日W」の次の曜日が「勤務不可日」でない
# ・「必要資格Y」を「所有資格」が満たしている
# ・仮に配属された場合、(夜勤明けも含めて)「週の勤務日数」の範囲である
# ・上記条件を満たす職員の中で、「配属先X」が「配属先希望順」で最上位である
# ・上記条件を満たす職員候補の中で、「希望達成率」が最も低い職員である
#
# -----
# 引数 :
# staffInfoList : 職員オブジェクト配列 (配列要素は StaffInfo)
# dayId          : 曜日ID (DAY_*)
# deptId         : 配属先ID (DEPT_ID_*)
# occupyDays     : 配属先の一勤務当たりの勤務日数
# aDeptQualId    : 配属先の必要資格ID (QUAL_DEPT_*, 必要資格無関係の場合は None)
# -----
# 戻り値 :
# bestStaff      : 配属するのに最適な職員 (StaffInfo)
#*****
def ExtractBestStaff(staffInfoList, dayId, deptId, occupyDays, aDeptQualId) :

    # 職員オブジェクト配列から、配属候補を抽出
    candList = []      # 候補職員のリスト
    bStaff = None
    for aStaff in staffInfoList:
        staffId = aStaff.staffId
        staffName = aStaff.staffName
        workDayAWeek = aStaff.workDayAWeek

        # 配属条件を満たす職員か
        if ( IsAvalilableStaff(
            aStaff, dayId, deptId, occupyDays, aDeptQualId ) == True ):

            # 配属条件を満たす職員の中で、「配属先X」が「配属先希望順」で最上位の候補を列举
            deptIndex = GetIndexOnArray(aStaff.reqDeptList, deptId)

```

```

    if (deptIndex != -1) :
        if (bStaff == None) :
            bStaff = BestStaffInfo(aStaff, deptIndex)
            candList.append( aStaff )

        elif (deptIndex == bStaff.deptIndex) :
            candList.append( aStaff )

        elif (deptIndex < bStaff.deptIndex) :
            bStaff = BestStaffInfo(aStaff, deptIndex)
            candList = []
            candList.append( aStaff )

# 配属候補から「希望達成率」が最も低い職員を配属させる
bestStaff = None
reqFitRate = 1.0
for aStaff in candList:
    if (bestStaff == None) or (reqFitRate >= aStaff.reqFitRate):
        reqFitRate = aStaff.reqFitRate
        bestStaff = aStaff

return bestStaff

#*****
# メソッド : ExtractFillOrderDeptList
# 配属先について、
# 「定員充足率」が低い順に並べ替えた配列 (配列要素は DeptInfo) を取得する。
#-----
# 引数 :
#   dayAssignList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAssign)
#-----
# 戻り値 :
#   deptList : 配属先について、定員充足率が低い順に並べ替えた配列 (配列要素は DeptInfo)
#*****
def ExtractFillOrderDeptList( dayAssignList ) :
    deptList = []
    deptListLen = 0

    # 曜日毎の繰り返し⇒「曜日W」
    for aDayAsgn in dayAssignList:
        dayId = aDayAsgn.dayId

        # 「配属先X」の定員充足率でソート
        for aDept in aDayAsgn.deptInfoList:
            if (deptListLen == 0):
                deptList.append( aDept )
                deptListLen = deptListLen + 1
            else:
                if (deptList[ deptListLen - 1 ].fillRate <= aDept.fillRate) :
                    deptList.append( aDept )
                    deptListLen = deptListLen + 1
                elif (deptList[0].fillRate > aDept.fillRate) :
                    deptList.insert(0, aDept)
                    deptListLen = deptListLen + 1
                else:
                    for ii in range( deptListLen - 1 ) :
                        if ( (deptList[ii].fillRate <= aDept.fillRate) and
                            (aDept.fillRate < deptList[ii + 1].fillRate) ) :
                            deptList.insert( ii+1, aDept)
                            deptListLen = deptListLen + 1
                            break

        # Debug Start =====
        #print("(0): {1}): fillRate={2:.2f}, deptListLen={3}, ".
        # format(ConvDayIdToName(dayId), aDept.deptId, aDept.fillRate, deptListLen))
        # Debug End -----

    return deptList

#*****
# メソッド : ExtractReqOrderDeptList
# 配属先について「配属先毎配属希望率」が低い順に並べ替えた配列を取得する。
#-----
# 引数 :
#   dayAssignList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAssign)
#-----

```

```

# 戻り値 :
#   deptList : 配属先について「配属先毎配属希望率」が低い順に並べ替えた配列 (配列要素は DeptInfo)
#*****
def ExtractReqOrderDeptList( dayAssignList ) :
    deptList = []
    deptListLen = 0

    # 曜日毎の繰り返し⇒「曜日W」
    for aDayAsgn in dayAssignList:
        dayId = aDayAsgn.dayId

        # 配属先毎の繰り返し⇒「配属先X」
        for aDept in aDayAsgn.deptInfoList :
            deptId = aDept.deptId

            if (deptListLen == 0):
                deptList.append( aDept )
                deptListLen = deptListLen + 1
            else:
                if (deptList[ deptListLen - 1 ].deptReqRate <= aDept.deptReqRate) :
                    deptList.append( aDept )
                    deptListLen = deptListLen + 1
                elif (deptList[0].deptReqRate > aDept.deptReqRate) :
                    deptList.insert(0, aDept)
                    deptListLen = deptListLen + 1
                else:
                    for ii in range( deptListLen - 1 ) :
                        if ( (deptList[ii].deptReqRate <= aDept.deptReqRate) and
                            (aDept.deptReqRate < deptList[ii+1].deptReqRate) ) :
                            deptList.insert( ii+1, aDept )
                            deptListLen = deptListLen + 1
                    break

            # Debug Start =====
            #print("({0}: {1}): deptReqRate={2:.2f}, deptListLen={3}, ".
            #      format(ConvDayIdToName(dayId), deptId, aDept.deptReqRate, deptListLen))
            # Debug End -----

    return deptList

#*****
# メソッド : ExtractReqOrderDeptQualList
#   配属先の必要資格について「必要資格毎配属希望率」が低い順に並べ替えた配列を取得する。
#-----
# 引数 :
#   dayAssignList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAssign)
#-----
# 戻り値 :
#   qualList : 配属先の必要資格について「必要資格毎配属希望率」が低い順に並べ替えた配列 (配列要素は QualInfo)
#*****
def ExtractReqOrderDeptQualList( dayAssignList ) :
    qualList = []
    qualListLen = 0

    # 曜日毎の繰り返し⇒「曜日W」
    for aDayAsgn in dayAssignList:
        dayId = aDayAsgn.dayId

        # 配属先毎の繰り返し⇒「配属先X」
        for aDept in aDayAsgn.deptInfoList :
            deptId = aDept.deptId

            # 「必要資格Y」の「必要資格毎配属希望率」でソート
            for aDeptQual in aDept.qualInfoList:

                if (qualListLen == 0):
                    qualList.append( aDeptQual )
                    qualListLen = qualListLen + 1
                else:
                    if (qualList[ qualListLen - 1 ].qualReqRate <= aDeptQual.qualReqRate) :
                        qualList.append( aDeptQual )
                        qualListLen = qualListLen + 1
                    elif (qualList[0].qualReqRate > aDeptQual.qualReqRate) :
                        qualList.insert(0, aDeptQual)
                        qualListLen = qualListLen + 1
                    else:
                        for ii in range( qualListLen - 1 ) :

```

```

        if ( (qualList[ii].qualReqRate <= aDeptQual.qualReqRate) and
              (aDeptQual.qualReqRate < qualList[ii+1].qualReqRate) ) :
            qualList.insert( ii+1, aDeptQual )
            qualListLen = qualListLen + 1
            break

    # Debug Start =====
    #print("{0}: {1}: {2}: qualReqRate={3:.2f}, qualListLen={4}, ".
    #      format(ConvDayIdToName(dayId), deptId, aDeptQual.qualId, aDeptQual.qualReqRate, qualListLen))
    # Debug End -----

return qualList

#*****
# メソッド : RegistStaffAsgnDayList
#   指定した「職員」と「配属先」を元に、職員の「曜日毎配属表」に登録
#-----
# 引数 :
#   staffInfo : 登録する「職員情報」オブジェクト (StaffInfo)
#   deptInfo  : 登録先の「配属先情報」オブジェクト (DeptInfo)
#-----
# 戻り値 : なし
#*****
def RegistStaffAsgnDayList( staffInfo, deptInfo ) :

    # 職員の「曜日毎配属表」に登録
    staffInfo.asgnDayList[deptInfo.dayId] = deptInfo.deptId
    if (deptInfo.deptId == DEPT_ID_3) :
        if (deptInfo.dayId + 1 < len(staffInfo.asgnDayList)) :
            staffInfo.asgnDayList[deptInfo.dayId + 1] = DEPT_ID_4
        else:
            staffInfo.asgnDayList[0] = DEPT_ID_4

#*****
# メソッド : RegistStaffToDeptStaffList
#   指定した「職員」について、
#   指定した「配属先」の該当する資格を保持する職員として「配属職員一覧」に登録
#-----
# 引数 :
#   staffInfo : 登録する「職員情報」オブジェクト (StaffInfo)
#   deptInfo  : 登録先の「配属先情報」オブジェクト (DeptInfo)
#-----
# 戻り値 : なし
#*****
def RegistStaffToDeptStaffList( staffInfo, deptInfo ) :

    # 該当する資格を照合する
    aDeptQualId = QUAL_DEPT_0
    for aStaffQual in staffInfo.staffQualList :
        if (aStaffQual == QUAL_STAFF_1) or (aStaffQual == QUAL_STAFF_2) :
            aDeptQualId = QUAL_DEPT_1_2
            break

    # 配属先の「配属職員一覧」に登録
    for aDeptQual in deptInfo.qualInfoList:
        if (aDeptQual.qualId == aDeptQualId) :
            aDeptQual.staffList.append(staffInfo)
            break

```



(リスト\_職員の一週間単位の配属\_実装\_04\_入力関数定義と入力値.txt)

```
*****
# リスト_職員の一週間単位の配属 (part04)・・・ケース1
#
# 入力関数定義と入力値
#
*****

# メソッド：LoadDayAssignList
# 配属先の「配属計画表」をロードして、配属先情報を初期化する
#
# 引数：
#   filePath    : 配属先の「配属計画表」ファイルのパス
#
# 戻り値：
#   dayAssignList : 配属先情報オブジェクト配列 (配列要素は DayAssign)
#
# (※) TODO ファイル (filePath) 読み込みは現在未対応。
#       将来的には EXCEL ファイル読み込みで置換える予定。
#
# 本入力データは以下の場合の例：
#
# 配属先の「配属計画表」
#   ・各曜日に (早番、遅番、夜勤) の配属先がある
#   ◎各配属先の最少定員は2
#   ・各配属先の「初任者研修または実務者研修」の有資格者の最少定員は1
#   ◎各配属先の「無資格」の最少定員は0
#
*****
def LoadDayAssignList(filePath) :

    dayAssignList = [
        DayAssign(DAY_0,
            [DeptInfo(DAY_0, DEPT_ID_1, 1, 2, [QualInfo(DAY_0, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_0, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_0, DEPT_ID_2, 1, 2, [QualInfo(DAY_0, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_0, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_0, DEPT_ID_3, 2, 2, [QualInfo(DAY_0, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_0, DEPT_ID_3, QUAL_DEPT_0, 0)])]),
        DayAssign(DAY_1,
            [DeptInfo(DAY_1, DEPT_ID_1, 1, 2, [QualInfo(DAY_1, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_1, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_1, DEPT_ID_2, 1, 2, [QualInfo(DAY_1, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_1, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_1, DEPT_ID_3, 2, 2, [QualInfo(DAY_1, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_1, DEPT_ID_3, QUAL_DEPT_0, 0)])]),
        DayAssign(DAY_2,
            [DeptInfo(DAY_2, DEPT_ID_1, 1, 2, [QualInfo(DAY_2, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_2, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_2, DEPT_ID_2, 1, 2, [QualInfo(DAY_2, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_2, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_2, DEPT_ID_3, 2, 2, [QualInfo(DAY_2, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_2, DEPT_ID_3, QUAL_DEPT_0, 0)])]),
        DayAssign(DAY_3,
            [DeptInfo(DAY_3, DEPT_ID_1, 1, 2, [QualInfo(DAY_3, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_3, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_3, DEPT_ID_2, 1, 2, [QualInfo(DAY_3, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_3, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_3, DEPT_ID_3, 2, 2, [QualInfo(DAY_3, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_3, DEPT_ID_3, QUAL_DEPT_0, 0)])]),
        DayAssign(DAY_4,
            [DeptInfo(DAY_4, DEPT_ID_1, 1, 2, [QualInfo(DAY_4, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_4, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_4, DEPT_ID_2, 1, 2, [QualInfo(DAY_4, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_4, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_4, DEPT_ID_3, 2, 2, [QualInfo(DAY_4, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_4, DEPT_ID_3, QUAL_DEPT_0, 0)])]),
        DayAssign(DAY_5,
            [DeptInfo(DAY_5, DEPT_ID_1, 1, 2, [QualInfo(DAY_5, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_5, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_5, DEPT_ID_2, 1, 2, [QualInfo(DAY_5, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_5, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_5, DEPT_ID_3, 2, 2, [QualInfo(DAY_5, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_5, DEPT_ID_3, QUAL_DEPT_0, 0)])]),
        DayAssign(DAY_6,
            [DeptInfo(DAY_6, DEPT_ID_1, 1, 2, [QualInfo(DAY_6, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_6, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_6, DEPT_ID_2, 1, 2, [QualInfo(DAY_6, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_6, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_6, DEPT_ID_3, 2, 2, [QualInfo(DAY_6, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_6, DEPT_ID_3, QUAL_DEPT_0, 0)])])
    ]
    return dayAssignList

*****
# メソッド：LoadStaffInfoList
# 職員の「配属先希望表」をロードして、職員情報を初期化する。
#
# 引数：
#   filePath    : 職員の「配属先希望表」ファイルのパス
#
# 戻り値：
#   staffInfoList : 職員オブジェクト配列 (配列要素は StaffInfo)
#
# (※) TODO ファイル (filePath) 読み込みは現在未対応。
#       将来的には EXCEL ファイル読み込みで置換える予定。
#
*****
```

```

# 本入力データは以下の場合の例：
#
# 職員の「配属先希望表」
#   ・所有資格は、様々（無資格、初任者研修、実務者研修、その他）
#   ・週の勤務日数は、様々（3～5）
#   ・配属先希望順は、様々（早番、遅番、夜勤）
#   ・曜日毎希望は、様々（勤務不可日、勤務可能日）
#   ◎曜日毎配属は、指定なし（None）
#
#*****
def LoadStaffInfoList(filePath) :

    staffInfoList = [
        StaffInfo("A", "職員A", [QUAL_STAFF_1],          4, [DEPT_ID_2, DEPT_ID_1, DEPT_ID_3], ["-", "-", "-", "-", "-", "-", "-"], None),
        StaffInfo("B", "職員B", [QUAL_STAFF_1, QUAL_STAFF_2], 4, [DEPT_ID_1, DEPT_ID_3, DEPT_ID_2], ["-", "X", "-", "-", "-", "-", "X"], None),
        StaffInfo("C", "職員C", [], 5, [DEPT_ID_2, DEPT_ID_3, DEPT_ID_1], ["X", "-", "-", "-", "-", "-", "-"], None),
        StaffInfo("D", "職員D", [QUAL_STAFF_2],          3, [DEPT_ID_2, DEPT_ID_1],          ["X", "X", "-", "-", "X", "-", "-"], None),
        StaffInfo("E", "職員E", [], 4, [DEPT_ID_2, DEPT_ID_1, DEPT_ID_3], ["-", "-", "-", "-", "-", "-", "-"], None),
        StaffInfo("F", "職員F", [QUAL_STAFF_1],          5, [DEPT_ID_3, DEPT_ID_1, DEPT_ID_2], ["-", "-", "-", "-", "-", "-", "X"], None),
        StaffInfo("G", "職員G", [QUAL_STAFF_1],          3, [DEPT_ID_2, DEPT_ID_1, DEPT_ID_3], ["-", "-", "-", "-", "X", "X", "-"], None),
        StaffInfo("H", "職員H", [QUAL_STAFF_2, QUAL_STAFF_3], 4, [DEPT_ID_1, DEPT_ID_3, DEPT_ID_2], ["-", "-", "-", "-", "-", "-", "-"], None),
        StaffInfo("I", "職員I", [], 5, [DEPT_ID_2, DEPT_ID_3, DEPT_ID_1], ["X", "-", "-", "-", "-", "-", "-"], None),
        StaffInfo("J", "職員J", [QUAL_STAFF_2],          4, [DEPT_ID_2, DEPT_ID_1],          ["-", "-", "-", "-", "-", "-", "X"], None),
        StaffInfo("K", "職員K", [], 4, [DEPT_ID_2, DEPT_ID_1, DEPT_ID_3], ["X", "-", "-", "-", "X", "X", "-"], None),
        StaffInfo("L", "職員L", [QUAL_STAFF_1],          5, [DEPT_ID_3, DEPT_ID_1, DEPT_ID_2], ["X", "-", "-", "-", "-", "-", "-"], None),
        StaffInfo("M", "職員M", [QUAL_STAFF_2],          4, [DEPT_ID_2, DEPT_ID_1, DEPT_ID_3], ["-", "-", "-", "-", "-", "-", "-"], None),
        StaffInfo("N", "職員N", [], 4, [DEPT_ID_2, DEPT_ID_1, DEPT_ID_3], ["-", "-", "-", "-", "-", "-", "-"], None),
        StaffInfo("O", "職員O", [], 4, [DEPT_ID_1, DEPT_ID_2],          ["X", "-", "-", "-", "-", "-", "-"], None) ]

    return staffInfoList

```

```

*****
# リスト_職員の一週間単位の配属 (part05)
# -----
# 処理関数定義と主制御
# -----
*****

*****
# メソッド : Process1st
# 主制御 (1)
#   ファイルロード・初期化処理。
# -----
#   配属先の「配属計画表」と職員の「配属先希望表」をロードして、
#   各種の初期化を行う。
# -----
# 引数 : なし
# -----
# 戻り値 :
#   dayAssignList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAssign)
#   staffInfoList : 「職員情報」オブジェクト配列 (配列要素は StaffInfo)
*****
def Process1st() :

    # 配属先の「配属計画表」をロードして、配属先情報を初期化する
    dayAssignList = LoadDayAssignList("dummy")

    # 職員の「配属先希望表」をロードして、職員情報を初期化する。
    staffInfoList = LoadStaffInfoList("dummy")

    # 職員情報で配属先情報を設定する
    SetDeptInfoWithStaffInfo( dayAssignList, staffInfoList )

    return dayAssignList, staffInfoList

*****
# メソッド : Process2nd
# 主制御 (2)
#   曜日・配属先・必要資格毎の最少定員の確保
# -----
#   各曜日の各配属先の必要資格毎の定員について、
#   最少定員に充ちるように配員を行う。
# -----
# 引数 :
#   dayAssignList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAssign)
#   staffInfoList : 「職員情報」オブジェクト配列 (配列要素は StaffInfo)
# -----
# 戻り値 : なし
*****
def Process2nd(dayAssignList, staffInfoList) :

    # 配属先の必要資格について「必要資格毎配属希望率」が低い順に並べ替えた配列を取得
    deptQualList = ExtractReqOrderDeptQualList( dayAssignList )

    # 「必要資格毎配属希望率」が低い順に必要資格毎の繰り返し
    for aDeptQual in deptQualList: #⇒「必要資格Y」
        dayId = aDeptQual.dayId      #⇒「曜日W」
        deptId = aDeptQual.deptId    #⇒「配属先X」
        occupyDays = aDeptQual.parentObj.occupyDays

        # Debug Start =====
        # print("{0}: {1} {2}): ".format(ConvDayIdToName(dayId), deptId, ConvDeptIdToName(deptId)), end="")
        # aDeptQual.printObj()
        # print("")
        # Debug End -----

        # 「必要資格Y」についての最少定員割当が済んでいる場合は、次の「必要資格」へ
        asgnCnt = len( aDeptQual.staffList )
        if aDeptQual.reqCapaMin <= asgnCnt :
            continue

        # 「必要資格Y」についての最少定員割当を充たす配属をする
        while (aDeptQual.reqCapaMin > asgnCnt) :

            # 職員の「配属先希望表」をみて、条件を満たす職員「職員P」を探し出し、
            # 「曜日W」の「配属先X」の「必要資格Y」の職員として登録
            bestStaff = ExtractBestStaff(staffInfoList, dayId, deptId, occupyDays, aDeptQual.qualId )

```

```

if (bestStaff != None) :

    # 職員の「曜日毎配属表」に登録
    RegistStaffAsgnDayList( bestStaff, aDeptQual.parentObj )

    # 配属先の「配属職員一覧」に登録
    RegistStaffToDeptStaffList( bestStaff, aDeptQual.parentObj )

    # 職員の「勤務日充足率」「希望達成率」を更新
    UpdateStaffRate( bestStaff )

    # 配属先の「定員充足率」と必要資格毎の「資格所有者定員充足率」を更新
    UpdateDeptFillRate( aDeptQual.parentObj )

    # 「必要資格 Y」についての配員済みの人数
    asgnCnt = asgnCnt + 1

# 条件に該当する「職員 P」が見つからない場合は、次の「必要資格」へ
else:
    break

#*****
# メソッド : Process3rd
# 主制御 (3)
# 曜日・配属先毎の、必要資格とは無関係な最少定員の確保
#-----
# 各曜日の各配属先の定員について、必要資格とは無関係に、
# 最少定員に充てるように配員を行う。
#-----
# 引数 :
# dayAsgnList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAsgn)
# staffInfoList : 「職員情報」オブジェクト配列 (配列要素は StaffInfo)
#-----
# 戻り値 : なし
#*****
def Process3rd(dayAsgnList, staffInfoList) :

    # 配属先について「配属先毎配属希望率」が低い順に並べ替えた配列を取得
    deptList = ExtractReqOrderDeptList( dayAsgnList )

    # 「配属先毎配属希望率」が低い順に配属先毎の繰り返し
    for aDept in deptList: # => 「配属先 X」
        dayId = aDept.dayId # => 「曜日 W」
        deptId = aDept.deptId

        # Debug Start =====
        # aDept.printObj()
        # Debug End -----

        # 「配属先 X」の配員済みの人数
        asgnCnt = 0
        for aDeptQual in aDept.qualInfoList:
            asgnCnt = asgnCnt + len(aDeptQual.staffList)

        # 「配属先 X」の最少定員割当を充たす配属をする
        while (aDept.capaMin > asgnCnt) :

            # 職員の「配属先希望表」をみて、条件を満たす職員「職員 P」を探し出し、
            # 「曜日 W」の「配属先 X」の職員として登録
            bestStaff = ExtractBestStaff(staffInfoList, dayId, deptId, aDept.occupyDays, None)

            if (bestStaff != None) :

                # Debug Start =====
                # bestStaff.printObj()
                # Debug End -----

                # 職員の「曜日毎配属表」に登録
                RegistStaffAsgnDayList( bestStaff, aDept )

                # 配属先の「配属職員一覧」に登録
                RegistStaffToDeptStaffList( bestStaff, aDept )

                # 職員の「勤務日充足率」「希望達成率」を更新
                UpdateStaffRate( bestStaff )

                # 配属先の「定員充足率」と必要資格毎の「資格所有者定員充足率」を更新

```

```

UpdateDeptFillRate( aDept )

# 「配属先X」の配員済みの人数
asgnCnt = asgnCnt + 1

# 条件に該当する「職員P」が見つからない時、調整終了
else:
    break

#*****
# メソッド : Process4th
# 主制御 (4)
# 各職員の週の勤務日数調整
#-----
# 各職員の「週の勤務日数」が満ち、かつ、
# 配属先の偏りが少なくなるように配員を行う。
#-----
# 引数 :
# dayAsgnList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAsgn)
# staffInfoList : 「職員情報」オブジェクト配列 (配列要素は StaffInfo)
#-----
# 戻り値 : なし
#*****
def Process4th( dayAsgnList, staffInfoList ) :

    # 全ての職員について繰り返し
    for aStaff in staffInfoList:

        # Debug Start =====
        # aStaff.printObj()
        # Debug End -----

        # 配属希望先が無ければ、次の職員へ
        if (len(aStaff.reqDeptList) <= 0):
            continue

        # 配属済み日数が「週の勤務日数」に達していたら、次の職員へ
        asgnDayTotal = len(aStaff.asgnDayList) ￥
            - aStaff.asgnDayList.count( DEPT_ID_0 ) ￥
            - aStaff.asgnDayList.count( DEPT_ID_X )
        if (asgnDayTotal >= aStaff.workDayAWeek):
            continue

        # 配属先について「定員充足率」が低い順に並べ替えた配列を取得
        deptList = ExtractFillOrderDeptList( dayAsgnList )

        # 配属先の「定員充足率」が低い順に配属を調整する
        for aDept in deptList:

            # Debug Start =====
            # aDept.printObj()
            # Debug End -----

            dayId = aDept.dayId
            deptId = aDept.deptId
            occupyDays = aDept.occupyDays

            # 配属条件を満たす職員か
            if ( IsAvailableStaff( aStaff, dayId, deptId, occupyDays, None ) == False ):
                continue

            # 職員の「曜日毎配属表」に登録
            RegistStaffAsgnDayList( aStaff, aDept )

            # 配属先の「配属職員一覧」に登録
            RegistStaffToDeptStaffList( aStaff, aDept )

            # 職員の「勤務日充足率」「希望達成率」を更新
            UpdateStaffRate( aStaff )

            # 配属先の「定員充足率」と必要資格毎の「資格所有者定員充足率」を更新
            UpdateDeptFillRate( aDept )

            # 配属済み日数を更新し、「週の勤務日数」に達していたら次の職員へ
            asgnDayTotal += occupyDays
            if (asgnDayTotal > aStaff.workDayAWeek):
                break

```

```

#*****
# メソッド : Process5th
# 主制御 (5)
# 全ての職員について「勤務日充足率」をチェックする。
# 全ての配属先について「定員充足率」をチェックする。
# もし各充足率が 100% 未満の場合、その旨を出力する。
#-----
# 引数 :
#   dayAsgnList : 「曜日ごとの配属情報」オブジェクト配列 (配列要素は DayAsgn)
#   staffInfoList : 「職員情報」オブジェクト配列 (配列要素は StaffInfo)
#-----
# 戻り値 : なし
#*****
def Process5th( dayAsgnList, staffInfoList ) :

    #-----
    # 全ての職員についてチェック
    #-----
    print("=====")
    print("= 全ての職員について「勤務日充足率」をチェック")
    print("=====")
    printLine = 0

    for aStaff in staffInfoList:

        # 「配属先希望順」に登録が無ければ、次の職員へ
        if (len(aStaff.reqDeptList) <= 0) :
            continue

        # 「週の勤務日数」が0ならば、次の職員へ
        if (aStaff.workDayAWeek <= 0) :
            continue

        # 「勤務日充足率」が 100%未満ならば、メッセージ出力
        if (aStaff.dayFillRate < 1.0) :
            printLine = printLine + 1
            if ( printLine == 1 ) :
                print("以下の職員は、希望した勤務日数に達していません:")

            aStaff.printObj()

        # 「勤務日充足率」が全て 100% ならば、メッセージ出力
        if ( printLine <= 0 ) :
            print("全ての職員は、希望した勤務日数の配員になっています。")

    #-----
    # 全ての配属先についてチェック
    #-----
    print("") # 改行
    print("=====")
    print("= 全ての配属先について「定員充足率」をチェック")
    print("=====")
    printLine = 0

    # 曜日毎の繰り返し⇒「曜日W」
    for aDayAsgn in dayAsgnList:
        dayId = aDayAsgn.dayId

        # 配属先毎の繰り返し⇒「配属先X」
        for aDept in aDayAsgn.deptInfoList :
            deptId = aDept.deptId

            # 「必要資格順」に登録が無ければ、次の職員へ
            if (len(aDept.qualInfoList) <= 0) :
                continue

            # 「配属先最少定員」が0以下ならば、次の配属先へ
            if (aDept.capaMin <= 0) :
                continue

            # 「定員充足率」が 100%未満ならば、メッセージ出力
            if (aDept.fillRate < 1.0) :
                printLine = printLine + 1
                if ( printLine == 1 ) :
                    print("以下の配属先は、最少定員に達していません:")

```

```

aDept.printObj()

# 「定員充足率」が全て 100% ならば、メッセージ出力
if ( printLine <= 0 ) :
    print("全ての配属先は、最少定員に達しています。")

#####
#
# 主制御
#
#####

print("*****")
print("* 1st) 主制御 (1) ファイルロード・初期化处理")
print("*****")
dayAssignList, staffInfoList = Process1st()

# 「配属予定表」と「曜日毎配属表」の出力
PrintDayAssignList(dayAssignList)
PrintStaffInfoList(staffInfoList)

print("")
print("*****")
print("* 2nd) 曜日・配属先・必要資格毎の最少定員の確保")
print("*****")
Process2nd(dayAssignList, staffInfoList)

# 「配属予定表」と「曜日毎配属表」の出力
PrintDayAssignList(dayAssignList)
PrintStaffInfoList(staffInfoList)

print("")
print("*****")
print("* 3rd) 曜日・配属先毎の最少定員の確保")
print("*****")
Process3rd(dayAssignList, staffInfoList)

# 「配属予定表」と「曜日毎配属表」の出力
PrintDayAssignList(dayAssignList)
PrintStaffInfoList(staffInfoList)

print("")
print("*****")
print("* 4th) 各職員の週の勤務日数調整")
print("*****")
Process4th(dayAssignList, staffInfoList)

# 「配属予定表」と「曜日毎配属表」の出力
PrintDayAssignList(dayAssignList)
PrintStaffInfoList(staffInfoList)

print("")
print("*****")
print("* 5th) 全ての職員について「勤務日充足率」をチェック")
print("*      全ての配属先について「定員充足率」をチェック")
print("*      もし各充足率が 100% 未満の場合、その旨を出力する。")
print("*****")
Process5th(dayAssignList, staffInfoList)

```

## (5.5) 実行結果

以下では、既述の「(5.4) 実装」で実装したものを実行した結果を掲載します。

本実装では、ファイルインタフェース、G U I のインタフェースはなく、  
「リスト\_職員の一週間単位の配属\_実装\_04\_入力関数定義と入力値.txt」で、入力値を指定するようにしています。

(リスト\_職員の一週間単位の配属\_実装\_04\_入力関数定義と入力値.txt)

```
#####
# リスト_職員の一週間単位の配属 (part04)・・・ケース5
#
# 入力関数定義と入力値
#
#####
#####
# メソッド：LoadDayAssignList
# 配属先の「配属計画表」をロードして、配属先情報を初期化する
#
# 引数：
#   filePath    : 配属先の「配属計画表」ファイルのパス
#
# 戻り値：
#   dayAssignList : 配属先情報オブジェクト配列 (配列要素は DayAssign)
#
# (※) TODO ファイル (filePath) 読み込みは現在未対応。
#       将来的には EXCEL ファイル読み込みで置換える予定。
#
# 本入力データは以下の場合の例：
#
# 配属先の「配属計画表」
# ・各曜日に (早番、遅番、夜勤) の配属先がある
#   ◎各配属先の最少定員は2
#   ・各配属先の「初任者研修または実務者研修」の有資格者の最少定員は1
#   ◎各配属先の「無資格」の最少定員は0
#
#####
def LoadDayAssignList(filePath) :

    dayAssignList = [
        DayAssign(DAY_0,
            [DeptInfo(DAY_0, DEPT_ID_1, 1, 2, [QualInfo(DAY_0, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_0, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_0, DEPT_ID_2, 1, 2, [QualInfo(DAY_0, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_0, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_0, DEPT_ID_3, 2, 2, [QualInfo(DAY_0, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_0, DEPT_ID_3, QUAL_DEPT_0, 0)]))],
        DayAssign(DAY_1,
            [DeptInfo(DAY_1, DEPT_ID_1, 1, 2, [QualInfo(DAY_1, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_1, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_1, DEPT_ID_2, 1, 2, [QualInfo(DAY_1, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_1, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_1, DEPT_ID_3, 2, 2, [QualInfo(DAY_1, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_1, DEPT_ID_3, QUAL_DEPT_0, 0)]))],
        DayAssign(DAY_2,
            [DeptInfo(DAY_2, DEPT_ID_1, 1, 2, [QualInfo(DAY_2, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_2, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_2, DEPT_ID_2, 1, 2, [QualInfo(DAY_2, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_2, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_2, DEPT_ID_3, 2, 2, [QualInfo(DAY_2, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_2, DEPT_ID_3, QUAL_DEPT_0, 0)]))],
        DayAssign(DAY_3,
            [DeptInfo(DAY_3, DEPT_ID_1, 1, 2, [QualInfo(DAY_3, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_3, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_3, DEPT_ID_2, 1, 2, [QualInfo(DAY_3, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_3, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_3, DEPT_ID_3, 2, 2, [QualInfo(DAY_3, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_3, DEPT_ID_3, QUAL_DEPT_0, 0)]))],
        DayAssign(DAY_4,
            [DeptInfo(DAY_4, DEPT_ID_1, 1, 2, [QualInfo(DAY_4, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_4, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_4, DEPT_ID_2, 1, 2, [QualInfo(DAY_4, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_4, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_4, DEPT_ID_3, 2, 2, [QualInfo(DAY_4, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_4, DEPT_ID_3, QUAL_DEPT_0, 0)]))],
        DayAssign(DAY_5,
            [DeptInfo(DAY_5, DEPT_ID_1, 1, 2, [QualInfo(DAY_5, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_5, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_5, DEPT_ID_2, 1, 2, [QualInfo(DAY_5, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_5, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_5, DEPT_ID_3, 2, 2, [QualInfo(DAY_5, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_5, DEPT_ID_3, QUAL_DEPT_0, 0)]))],
        DayAssign(DAY_6,
            [DeptInfo(DAY_6, DEPT_ID_1, 1, 2, [QualInfo(DAY_6, DEPT_ID_1, QUAL_DEPT_1_2, 1), QualInfo(DAY_6, DEPT_ID_1, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_6, DEPT_ID_2, 1, 2, [QualInfo(DAY_6, DEPT_ID_2, QUAL_DEPT_1_2, 1), QualInfo(DAY_6, DEPT_ID_2, QUAL_DEPT_0, 0)]),
            DeptInfo(DAY_6, DEPT_ID_3, 2, 2, [QualInfo(DAY_6, DEPT_ID_3, QUAL_DEPT_1_2, 1), QualInfo(DAY_6, DEPT_ID_3, QUAL_DEPT_0, 0)]))],
    ]
    return dayAssignList

#####
# メソッド：LoadStaffInfoList
# 職員の「配属先希望表」をロードして、職員情報を初期化する。
#
# 引数：
#   filePath    : 職員の「配属先希望表」ファイルのパス
#
# 戻り値：
#   staffInfoList : 職員オブジェクト配列 (配列要素は StaffInfo)
#
# (※) TODO ファイル (filePath) 読み込みは現在未対応。
#       将来的には EXCEL ファイル読み込みで置換える予定。
#
```



```

# 本入力データは以下の場合の例：
#
# 職員の「配属先希望表」
# ・所有資格は、様々（無資格、初任者研修、実務者研修、その他）
# ・週の勤務日数は、様々（3～5）
# ・配属先希望順は、様々（早番、遅番、夜勤）
# ・曜日毎希望は、様々（勤務不可日、勤務可能日）
# ◎曜日毎配属は、指定あり（職員A、職員F、職員M、職員N）
#
#*****
def LoadStaffInfoList(filePath)：

    staffInfoList = [
        StaffInfo("A", "職員A", [QUAL_STAFF_1], 4, [DEPT_ID_2,DEPT_ID_1,DEPT_ID_3], ["-", "-", "-", "-", "-", "-"], ["1", "2", "3", "/", "-", "-", "-"] ),
        StaffInfo("B", "職員B", [QUAL_STAFF_1,QUAL_STAFF_2], 4, [DEPT_ID_1,DEPT_ID_3,DEPT_ID_2], ["-", "X", "-", "-", "-", "-"], None ),
        StaffInfo("C", "職員C", [], 5, [DEPT_ID_2,DEPT_ID_3,DEPT_ID_1], ["X", "-", "-", "-", "-", "-"], None ),
        StaffInfo("D", "職員D", [QUAL_STAFF_2], 3, [DEPT_ID_2,DEPT_ID_1], ["X", "X", "-", "-", "-", "-"], None ),
        StaffInfo("E", "職員E", [ ], 4, [DEPT_ID_2,DEPT_ID_1,DEPT_ID_3], ["-", "-", "-", "-", "-", "-"], None ),
        StaffInfo("F", "職員F", [QUAL_STAFF_1], 5, [DEPT_ID_3,DEPT_ID_1,DEPT_ID_2], ["-", "-", "-", "-", "-", "-"], ["-", "-", "-", "1", "2", "3", "/"] ),
        StaffInfo("G", "職員G", [QUAL_STAFF_1], 3, [DEPT_ID_2,DEPT_ID_1,DEPT_ID_3], ["-", "-", "-", "-", "X", "-"], None ),
        StaffInfo("H", "職員H", [QUAL_STAFF_2,QUAL_STAFF_3], 4, [DEPT_ID_1,DEPT_ID_3,DEPT_ID_2], ["-", "-", "-", "-", "-", "-"], None ),
        StaffInfo("I", "職員I", [], 5, [DEPT_ID_2,DEPT_ID_3,DEPT_ID_1], ["X", "-", "-", "-", "-", "-"], None ),
        StaffInfo("J", "職員J", [QUAL_STAFF_2], 4, [DEPT_ID_2,DEPT_ID_1], ["-", "-", "-", "-", "-", "-"], None ),
        StaffInfo("K", "職員K", [], 4, [DEPT_ID_2,DEPT_ID_1,DEPT_ID_3], ["X", "-", "-", "-", "X", "-"], None ),
        StaffInfo("L", "職員L", [QUAL_STAFF_1], 5, [DEPT_ID_3,DEPT_ID_1,DEPT_ID_2], ["X", "-", "-", "-", "-", "-"], None ),
        StaffInfo("M", "職員M", [QUAL_STAFF_2], 4, [DEPT_ID_2,DEPT_ID_1,DEPT_ID_3], ["-", "-", "-", "-", "-", "-"], ["-", "-", "-", "1", "2", "3"] ),
        StaffInfo("N", "職員N", [], 4, [DEPT_ID_2,DEPT_ID_1,DEPT_ID_3], ["-", "-", "-", "-", "-", "-"], ["3", "/", "2", "1", "-", "-", "-"] ),
        StaffInfo("O", "職員O", [], 4, [DEPT_ID_1,DEPT_ID_2], ["X", "-", "-", "-", "-", "-"], None ) ]

    return staffInfoList

```





\*\*\*\*\*  
\* 4th) 各職員の週の勤務日数調整  
\*\*\*\*\*

= 配属先一覧

日:  
(日: 1 早番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.43, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.34 配属職員=((A:職員A) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.09 配属職員=((E:職員E) ) )  
(日: 2 遅番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.47, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=2.00、必要資格毎配属希望率=0.33 配属職員=((J:職員J) (F:職員F) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.13)  
(日: 3 夜勤): 勤務日数=2日, 最少定員=2名, 配属先毎配属希望率=0.37, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.32 配属職員=((H:職員H) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.04 配属職員=((N:職員N) ) )  
月:  
(月: 1 早番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.57, 定員充足率=1.50, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.32 配属職員=((L:職員L) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.24 配属職員=((O:職員O) (C:職員C) ) )  
(月: 2 遅番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.79, 定員充足率=1.50, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=2.00、必要資格毎配属希望率=0.42 配属職員=((A:職員A) (J:職員J) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.37 配属職員=((K:職員K) ) )  
(月: 3 夜勤): 勤務日数=2日, 最少定員=2名, 配属先毎配属希望率=0.60, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.34 配属職員=((G:職員G) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.26 配属職員=((I:職員I) ) )  
火:  
(火: 1 早番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.67, 定員充足率=1.50, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.42 配属職員=((B:職員B) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.24 配属職員=((O:職員O) (C:職員C) ) )  
(火: 2 遅番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.70, 定員充足率=1.50, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=2.00、必要資格毎配属希望率=0.42 配属職員=((D:職員D) (J:職員J) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.37 配属職員=((N:職員N) ) )  
(火: 3 夜勤): 勤務日数=2日, 最少定員=2名, 配属先毎配属希望率=0.74, 定員充足率=1.50, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=2.00、必要資格毎配属希望率=0.49 配属職員=((A:職員A) (L:職員L) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.26 配属職員=((K:職員K) ) )  
水:  
(水: 1 早番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.67, 定員充足率=1.50, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=2.00、必要資格毎配属希望率=0.42 配属職員=((F:職員F) (D:職員D) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.24 配属職員=((N:職員N) ) )  
(水: 2 遅番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.69, 定員充足率=1.50, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.29 配属職員=((J:職員J) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.37 配属職員=((C:職員C) ) )  
(水: 3 夜勤): 勤務日数=2日, 最少定員=2名, 配属先毎配属希望率=0.74, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.49 配属職員=((B:職員B) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.26 配属職員=((E:職員E) ) )  
木:  
(木: 1 早番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.54, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.34 配属職員=((M:職員M) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.20 配属職員=((O:職員O) ) )  
(木: 2 遅番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.59, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.29 配属職員=((F:職員F) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.30 配属職員=((I:職員I) ) )  
(木: 3 夜勤): 勤務日数=2日, 最少定員=2名, 配属先毎配属希望率=0.60, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.37 配属職員=((L:職員L) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.23 配属職員=((C:職員C) ) )  
金:  
(金: 1 早番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.58, 定員充足率=1.50, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=3.00、必要資格毎配属希望率=0.38 配属職員=((B:職員B) (H:職員H) (D:職員D) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.20)  
(金: 2 遅番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.66, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.36 配属職員=((M:職員M) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.30 配属職員=((E:職員E) ) )  
(金: 3 夜勤): 勤務日数=2日, 最少定員=2名, 配属先毎配属希望率=0.70, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.47 配属職員=((F:職員F) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.23 配属職員=((I:職員I) ) )  
土:  
(土: 1 早番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.52, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.28 配属職員=((H:職員H) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.24 配属職員=((O:職員O) ) )  
(土: 2 遅番): 勤務日数=1日, 最少定員=2名, 配属先毎配属希望率=0.68, 定員充足率=1.00, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.31 配属職員=((G:職員G) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.37 配属職員=((K:職員K) ) )  
(土: 3 夜勤): 勤務日数=2日, 最少定員=2名, 配属先毎配属希望率=0.53, 定員充足率=0.50, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.28 配属職員=((M:職員M) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.26)

= 職員一覧

(A:職員A): 適当り勤務日数=(4) 勤務希望=(,,-,-,-,-,-,-) 勤務予定=(1,2,3,/,,-,-,-) 勤務日充足率=1.00 希望達成率=0.58 配属先希望順=(2,1,3) 所有資格=(初任者研修) 適合資格=(初任者・実務者研修)  
(B:職員B): 適当り勤務日数=(4) 勤務希望=(X,-,-,-,-,-,-,X) 勤務予定=(X,-,1,3,/,1,x) 勤務日充足率=1.00 希望達成率=0.83 配属先希望順=(1,3,2) 所有資格=(初任者研修,実務者研修) 適合資格=(初任者・実務者研修)  
(C:職員C): 適当り勤務日数=(5) 勤務希望=(X,-,-,-,-,-,-,-) 勤務予定=(x,1,1,2,3,/,-) 勤務日充足率=1.00 希望達成率=0.60 配属先希望順=(2,3,1) 所有資格=(なし) 適合資格=(なし)  
(D:職員D): 適当り勤務日数=(3) 勤務希望=(X,X,-,-,-,-,-,-) 勤務予定=(x,x,2,1,x,1,-) 勤務日充足率=1.00 希望達成率=0.67 配属先希望順=(2,1) 所有資格=(実務者研修) 適合資格=(初任者・実務者研修)  
(E:職員E): 適当り勤務日数=(4) 勤務希望=(,-,-,-,-,-,-,-) 勤務予定=(1,-,-,3,/,2,-) 勤務日充足率=1.00 希望達成率=0.58 配属先希望順=(2,1,3) 所有資格=(なし) 適合資格=(なし)  
(F:職員F): 適当り勤務日数=(5) 勤務希望=(,-,-,-,-,-,-,X) 勤務予定=(2,-,-,1,2,3,/) 勤務日充足率=1.00 希望達成率=0.67 配属先希望順=(3,1,2) 所有資格=(初任者研修) 適合資格=(初任者・実務者研修)  
(G:職員G): 適当り勤務日数=(3) 勤務希望=(,-,-,-,-,-,X,X,-) 勤務予定=(X,3,/,x,x,x,2) 勤務日充足率=1.00 希望達成率=0.56 配属先希望順=(2,1,3) 所有資格=(初任者研修) 適合資格=(初任者・実務者研修)  
(H:職員H): 適当り勤務日数=(4) 勤務希望=(,-,-,-,-,-,-,-) 勤務予定=(3,/,,-,-,1,1) 勤務日充足率=1.00 希望達成率=0.83 配属先希望順=(1,3,2) 所有資格=(実務者研修,その他) 適合資格=(初任者・実務者研修)  
(I:職員I): 適当り勤務日数=(5) 勤務希望=(X,-,-,-,-,-,-,-) 勤務予定=(x,3,/,-,2,3,/) 勤務日充足率=1.00 希望達成率=0.73 配属先希望順=(2,3,1) 所有資格=(なし) 適合資格=(なし)  
(J:職員J): 適当り勤務日数=(4) 勤務希望=(,-,-,-,-,-,-,X) 勤務予定=(2,2,2,2,/,-,x) 勤務日充足率=1.00 希望達成率=1.00 配属先希望順=(2,1) 所有資格=(実務者研修) 適合資格=(初任者・実務者研修)  
(K:職員K): 適当り勤務日数=(4) 勤務希望=(X,-,-,-,-,-,X,X,-) 勤務予定=(x,2,3,/,x,x,2) 勤務日充足率=1.00 希望達成率=0.67 配属先希望順=(2,1,3) 所有資格=(なし) 適合資格=(なし)  
(L:職員L): 適当り勤務日数=(5) 勤務希望=(X,-,-,-,-,-,-,-) 勤務予定=(x,1,3,/,3,/,-) 勤務日充足率=1.00 希望達成率=0.93 配属先希望順=(3,1,2) 所有資格=(初任者研修) 適合資格=(初任者・実務者研修)  
(M:職員M): 適当り勤務日数=(4) 勤務希望=(,-,-,-,-,-,-,-) 勤務予定=(/,/,,-,-,1,2,3) 勤務日充足率=1.00 希望達成率=0.58 配属先希望順=(2,1,3) 所有資格=(実務者研修) 適合資格=(初任者・実務者研修)  
(N:職員N): 適当り勤務日数=(4) 勤務希望=(,-,-,-,-,-,-,-) 勤務予定=(3,/,2,1,-,-,-) 勤務日充足率=1.00 希望達成率=0.58 配属先希望順=(2,1,3) 所有資格=(なし) 適合資格=(なし)  
(O:職員O): 適当り勤務日数=(4) 勤務希望=(X,-,-,-,-,-,-,-) 勤務予定=(x,1,1,-,1,-,1) 勤務日充足率=1.00 希望達成率=1.00 配属先希望順=(1,2) 所有資格=(なし) 適合資格=(なし)

\*\*\*\*\*  
\* 5th) 全ての職員について「勤務日充足率」をチェック  
\* 全ての配属先について「定員充足率」をチェック  
\* もし各充足率が 100% 未満の場合、その旨を出力する。  
\*\*\*\*\*

= 全ての職員について「勤務日充足率」をチェック

全ての職員は、希望した勤務日数の配員になっています。

= 全ての配属先について「定員充足率」をチェック

以下の配属先は、最少定員に達していません:  
(土: 3 夜勤): 勤務日数=2日, 最少定員=2名, 配属先毎配属希望率=0.53, 定員充足率=0.50, 必要資格=(初任者・実務者研修: 最少定員=1名、資格定員充足率=1.00、必要資格毎配属希望率=0.28 配属職員=((M:職員M) ) / 無資格: 最少定員=0名、資格定員充足率=---、必要資格毎配属希望率=0.26)