電気通信大学 試作問題 第3問 数字列の並び替え

情報処理学会・学会誌「情報処理」
2024年3月6日 17:32

佐藤　喬（東京都立産業技術高等専門学校）

今回は，2023年11月26日（日）に電気通信大学で実施された個別学力検査「情報」の受験体験会で用いられた試作問題[1]の第3問について解説します．この問題は，6桁のある数字列を並び替えて，辞書式順で次にくる数字列を求める手順について考える内容です．

目次

問1の解説
問2の解説
問3の解説
問題の難易度と総括
付録：Pythonによるプログラミング

問1の解説

それでは，問1から見ていきましょう．**図A**は，6桁の数字列の定義と大小関係について，例を用いた説明です．

問 1 次の文章を読み，問い (1)〜(4) に答えよ．

1, 2, 3, 4, 5, 6 の数字をそれぞれ 1 回ずつ用いて，数字 6 桁の並びを作る．このようにして作った数字の並びを [123456] や [425316] のように書き，以後は「並び」と呼ぶことにする．同じ数字を 2 回以上使った [112345] などは考えない．並びの大小関係は，並びをそのまま 6 桁の整数とみなした場合の大小関係とする．例えば，[312465] と [321564] では，312465 < 321564 なので，[312465] の方が小さいと考える．並びの中で，最も小さいものから順に 4 番目までは，以下のようになる．

最も小さい並び	[123456]
2 番目に小さい並び	[123465]
3 番目に小さい並び	[123546]
4 番目に小さい並び	[123564]

(1) 最も大きい並びを答えよ．

(2) 4 番目に小さい並び [123564] に続く，5 番目から 7 番目に小さい並びを答えよ．

図A　数字列の定義と大小関係

問い（1）と（2）は，手を動かせば解ける内容と思われます．（1）の正解は，「最も大きい並び」ですから，大きい数字より並べて [654321] です．（2）の正解は，5番目が [123645]，6番目が [123654]，7番目が [124356] となります．

実際に手を動かすと，何らかの機械的な手順がありそうに感じます．その具体的な手順が，問題文の続きに示されています．**図B**「次の並びを求める手順」の説明と**図C**の「次の並びを求める手順」の

実例です．文章が長めですがすべて有益な内容であるため，丁寧に読み進め，自分で数字列を書き，手順を追ってみると理解が進むと思います．

ある並び S があったとき，小さい順で考えて S の次にくる並びを「S の次の並び」のように呼ぶことにする．例えば，最も小さい並び [123456] の次の並びは，2 番目に小さい並び [123465] である．今，ある並びに対して，その次の並びを見つける手順について考える．

まず，ある並びと，その次の並びの間で，変化しない数字について考える．上に例示した並びを見ると，並びの左側が変化していないことに気付く．例えば，最も小さい並びと 2 番目に小さい並びでは，左側の 1234 の部分が変化していない．また，2 番目に小さい並びと 3 番目に小さい並びでは，左側の 123 の部分が変化していない．ここで，左から何桁目までが変化しないかは並びによって様々である．実は，変化しない数字については，次のような法則がある．

並びに含まれる数字を右から順に調べたとき，初めて右隣の数字よりも数が小さくなる桁があったとすると，その桁よりも左にある数字は変化しない．

図B 「次の並びを求める手順」の説明

並び [342651] を例として，この法則をあてはめてみる．[342651] の各桁を右から調べると，5 と 6 の各桁は右隣よりも大きいが，2 は右隣よりも小さい．よって，2 よりも左にある 34 の部分は変化しない．以後，「並びに含まれる数字を右から順に調べたとき，初めて右隣の数字よりも小さくなる桁」のことを「変化する部分の左端」と呼ぶことにする．[342651] の変化する部分の左端の数字は 2 である（図 1 (a) 参照）．

次に，変化する部分の左端の数字が，どの数字に変化するかを考える．[342651] の例では，左にある 34 の部分は変化しないので，651 のどれかの数字に変化する．ここで，もし変化する部分の左端が 1 に変化すると，元の [342651] よりも小さくなってしまう．そのように考えると，651 の中で 2 よりも大きい数のうち最小の数，すなわち 5 に変化するとわかる（図 1 (b) 参照）．

最後に，変化する部分の左端よりも右側の並びについては，残った数字を小さい順に並べればよい．[342651] の例では，すでに左から 3 桁が 345 であることはわかっているので，残った数字を小さい順に並べることにより，[342651] の次の並びは [345126] であるとわかる（図 1 (c) 参照）．

(3) 並び [436521] の変化する部分の左端の数字を答えよ．

(4) 並び [436521] の次の並びを答えよ．

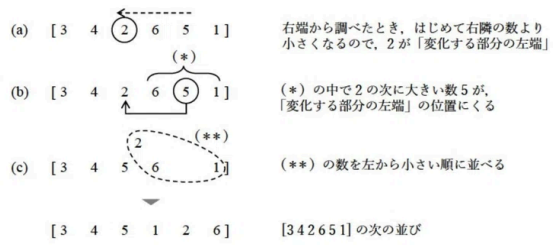
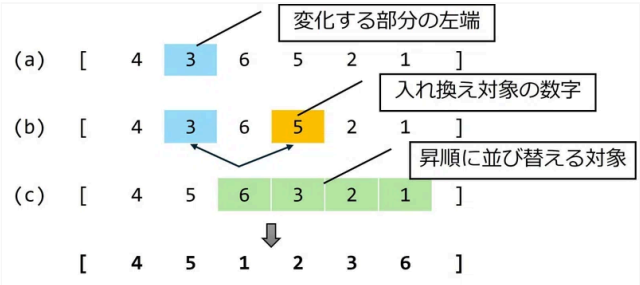


図 1 次の並びを求める手順

図C 「次の並びを求める手順」の実例

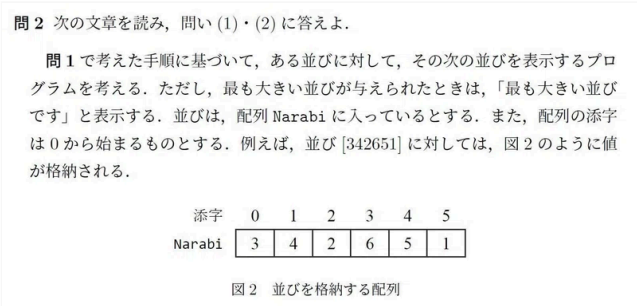
それでは，問い（3）と（4）を解いてみましょう．「次の並びを求める手順」に従って，[436521] の次の並びを求めたものが**図D**となります．よって，（3）の正解は3，（4）の正解は [451236] です．



図D　[436521]の次の並びを求める手順

問2の解説

問2は，問1で扱った「次の並びを求める手順」をプログラムで表現する内容になります．まず**図E**は，数字列のデータ構造の説明です．数字列は Narabi という配列に格納されます．多くのプログラミング言語と同じく，配列の添字の始まりは0からです．添字と数字列の桁との対応を確認しておきましょう．



問 2　次の文章を読み，問い (1)・(2) に答えよ．

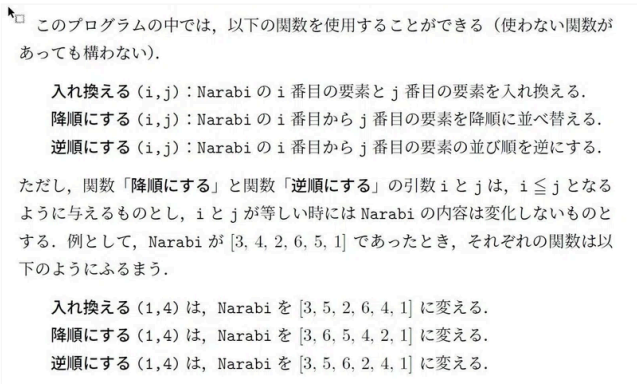
問 1 で考えた手順に基づいて，ある並びに対して，その次の並びを表示するプログラムを考える．ただし，最も大きい並びが与えられたときは，「最も大きい並びです」と表示する．並びは，配列 Narabi に入っているとする．また，配列の添字は 0 から始まるものとする．例えば，並び [342651] に対しては，図 2 のように値が格納される．

添字	0	1	2	3	4	5
Narabi	3	4	2	6	5	1

図 2　並びを格納する配列

図E　データ構造の説明

続いて**図F**は，プログラム中で使用できる関数の説明です．実際に使用した場合に，どのように数字列（Narabi）が変化するかの例示があります．自分の理解と合っているか確認しましょう．なお，世の中では「降順」のことを「逆順」とも言う場合がありますが，この問題における関数「逆順にする」は，単に「並び順を逆にする」働きをします．



このプログラムの中では，以下の関数を使用することができる（使わない関数があっても構わない）．

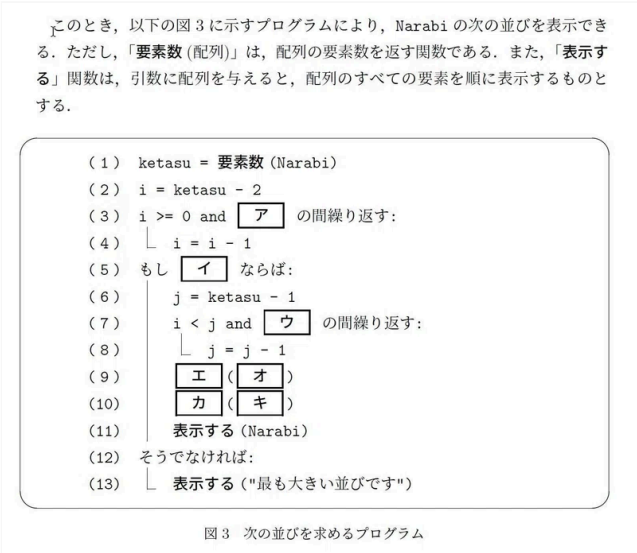
入れ換える (i,j)：Narabi の i 番目の要素と j 番目の要素を入れ換える．
降順にする (i,j)：Narabi の i 番目から j 番目の要素を降順に並べ替える．
逆順にする (i,j)：Narabi の i 番目から j 番目の要素の並び順を逆にする．

ただし，関数「**降順にする**」と関数「**逆順にする**」の引数 i と j は，i ≦ j となるように与えるものとし，i と j が等しい時には Narabi の内容は変化しないものとする．例として，Narabi が [3, 4, 2, 6, 5, 1] であったとき，それぞれの関数は以下のようにふるまう．

入れ換える (1,4) は，Narabi を [3, 5, 2, 6, 4, 1] に変える．
降順にする (1,4) は，Narabi を [3, 6, 5, 4, 2, 1] に変える．
逆順にする (1,4) は，Narabi を [3, 5, 6, 2, 4, 1] に変える．

図F　関数の説明

図Gは，次の並びを求めるプログラムの本体です．共通テスト用プログラム表記 [2) 3)] に準拠しており，「逐次」「繰り返し」「条件分岐」の制御構造を理解できていれば読むことができるでしょう．空欄（ア）～（キ）にどのようなコードを当てはめればよいか考えていきましょう．



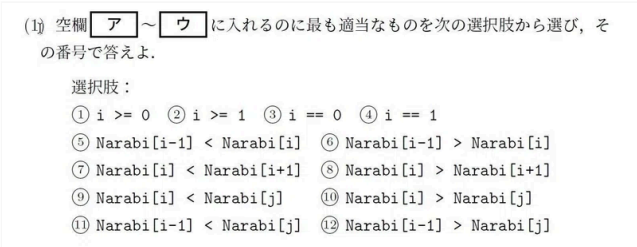
このとき，以下の図 3 に示すプログラムにより，Narabi の次の並びを表示できる．ただし，「**要素数** (配列)」は，配列の要素数を返す関数である．また，「**表示する**」関数は，引数に配列を与えると，配列のすべての要素を順に表示するものとする．

```
(1)  ketasu = 要素数 (Narabi)
(2)  i = ketasu - 2
(3)  i >= 0 and [ア] の間繰り返す:
(4)      i = i - 1
(5)  もし [イ] ならば:
(6)      j = ketasu - 1
(7)      i < j and [ウ] の間繰り返す:
(8)          j = j - 1
(9)      [エ] ([オ])
(10)     [カ] ([キ])
(11)     表示する (Narabi)
(12) そうでなければ:
(13)     表示する ("最も大きい並びです")
```

図 3　次の並びを求めるプログラム

図G　プログラム本体

図Hは，空欄（ア）～（ウ）に当てはまる内容として最も適当なものを選択肢から選ぶものです．空欄（ア）は，「変化する部分の左端」を見つける部分に該当します．



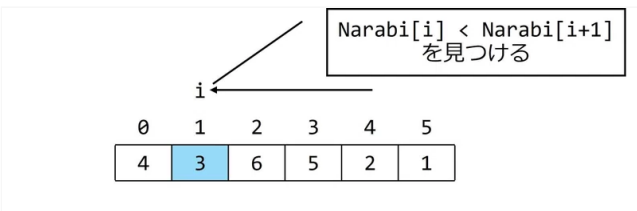
(1) 空欄 [ア] ～ [ウ] に入れるのに最も適当なものを次の選択肢から選び，その番号で答えよ．

選択肢：

① i >= 0　② i >= 1　③ i == 0　④ i == 1
⑤ Narabi[i-1] < Narabi[i]　⑥ Narabi[i-1] > Narabi[i]
⑦ Narabi[i] < Narabi[i+1]　⑧ Narabi[i] > Narabi[i+1]
⑨ Narabi[i] < Narabi[j]　⑩ Narabi[i] > Narabi[j]
⑪ Narabi[i-1] < Narabi[j]　⑫ Narabi[i-1] > Narabi[j]

図H　条件式の選択

図Iが実現したいことです．(3)，(4) 行目の繰り返し処理が終わった段階で，i の値が「変化する部分の左端」を示すことを目指します．つまり，Narabi[i] >= Narabi[i+1] の間は，i = i - 1 を繰り返せばよいわけです．ただし，選択肢には該当するものがないことと，同じ数字は含まれないことから空欄（ア）は，⑧ Narabi[i] > Narabi[i+1] が正解となります．

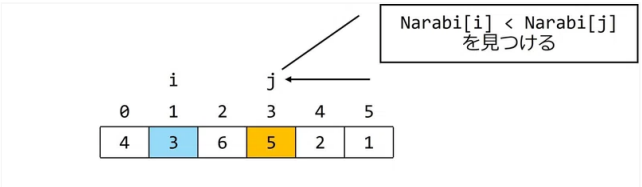


図I 「変化する部分の左端」を見つける

空欄（イ）は，成立しないと「最も大きい並びです」と表示することになります．つまり，「変化する部分の左端」が見つかった場合の条件ですから，空欄（イ）は，① i >= 0 が正解となります．

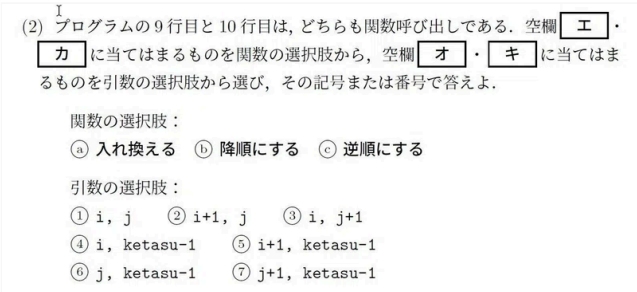
空欄（ウ）は，「変化する部分の左端」と入れ換える数字を見つける内容です．「変化する部分の左端」の右側の数字列から，「変化する部分の左端」の次に大きい数字を探します．

ここで重要なことは，対象の数字列は，右から小さい順に並んでいるということです（(3) 行目の空欄（ア）の条件を再確認しましょう）．このことを利用して，**図J**に（7），（8）行目で入れ換え対象を見つける様子を示します．Narabi[i] < Narabi[j] とするには，Narabi[i] >= Narabi[j] の間，j = j - 1 を繰り返せばよいわけです．こちらも，同じ数字はないという前提から，空欄（ウ）の正解は⑩ Narabi[i] > Narabi[j] となります．



図J　入れ換え対象を見つける

続いて，**図K**の関数呼び出しを考えていきましょう．実現したいことは，Narabi[i] と Narabi[j] を入れ換えて，Narabi[i+1] 〜 Narabi[ketasu-1] を昇順にすることです．


(2) プログラムの 9 行目と 10 行目は，どちらも関数呼び出しである．空欄 エ ・ カ に当てはまるものを関数の選択肢から，空欄 オ ・ キ に当てはまるものを引数の選択肢から選び，その記号または番号で答えよ．

関数の選択肢：

ⓐ 入れ換える　ⓑ 降順にする　ⓒ 逆順にする

引数の選択肢：

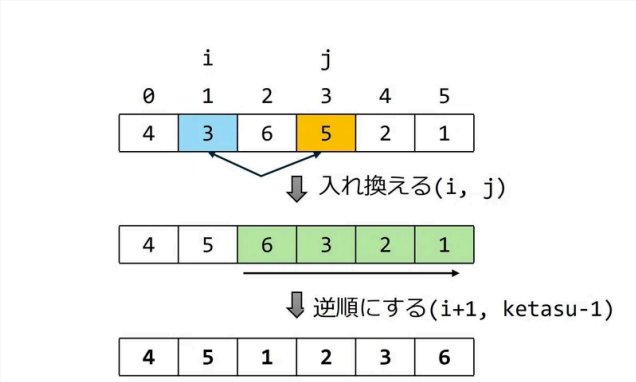
① i, j　② i+1, j　③ i, j+1
④ i, ketasu-1　⑤ i+1, ketasu-1
⑥ j, ketasu-1　⑦ j+1, ketasu-1

図K　関数の選択

しかし，関数「入れ換える」はあるのですが，関数「昇順にする」がありません．ここでも，先程出てきた「並び替え対象の数字列は，右から小さい順に並んでいる」ということが重要になります．並び替え対象の数字列が，「右から小さい順に並んでいる」つまり「すでに降順に並んでいる」ので，これを「逆順にする」と「昇順にする」が実現できるのです．

図Lにその様子を示します．まず Narabi[i] と Narabi[j] を「入れ換える」と，入れ換えたあとも並び替え対象の数字列は必ず降順になっています．この並び替え対象の数字列（Narabi[i+1] から Narabi[ketasu-1] まで）を「逆順にする」と昇順になります．したがって，正解は，空欄（エ）が ⓐ

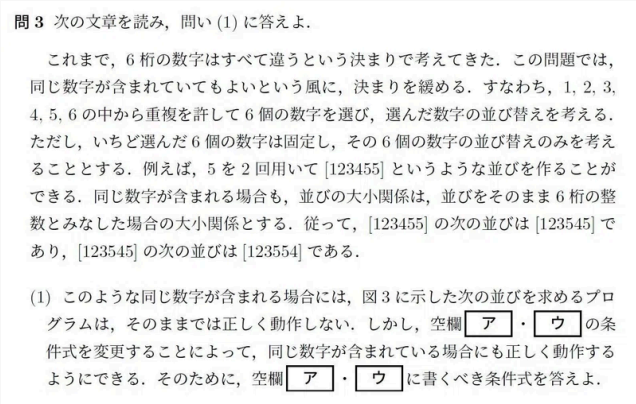
入れ換える，空欄（オ）が ① (i, j)，空欄（カ）が ⓒ 逆順にする，空欄（キ）が ⑤ (i+1, ketasu-1) です．



図L　関数呼び出しによるNarabiの変化

問3の解説

問3 は，使用する数値に重なりを認める拡張をした場合のプログラム修正の問題です（**図M**）．修正個所は，空欄（ア）と（ウ）です．両方とも問2では，「同じ数字は含まれない」ことから等号条件を外した個所になります．よって，等号条件を戻せばよいでしょう．空欄（ア）は Narabi[i] >= Narabi[i+1]，空欄（ウ）は Narabi[i] >= Narabi[j] に変更することが正解です．



問 3 次の文章を読み，問い (1) に答えよ．

これまで，6 桁の数字はすべて違うという決まりで考えてきた．この問題では，同じ数字が含まれていてもよいという風に，決まりを緩める．すなわち，1, 2, 3, 4, 5, 6 の中から重複を許して 6 個の数字を選び，選んだ数字の並び替えを考える．ただし，いちど選んだ 6 個の数字は固定し，その 6 個の数字の並び替えのみを考えることとする．例えば，5 を 2 回用いて [123455] というような並びを作ることができる．同じ数字が含まれる場合も，並びの大小関係は，並びをそのまま 6 桁の整数とみなした場合の大小関係とする．従って，[123455] の次の並びは [123545] であり，[123545] の次の並びは [123554] である．

(1) このような同じ数字が含まれる場合には，図 3 に示した次の並びを求めるプログラムは，そのままでは正しく動作しない．しかし，空欄 ア ・ ウ の条件式を変更することによって，同じ数字が含まれている場合にも正しく動作するようにできる．そのために，空欄 ア ・ ウ に書くべき条件式を答えよ．

図M　同じ数字が含まれる場合

問題の難易度と総括

手を動かせば分かるやさしい問題から，内容を正しく理解していないと解答できない問題まで揃った内容であることと，特定のアルゴリズムやプログラミング言語の前提知識が不要であることから，受験生の「プログラムと解決能力」の力を公平に分別できる構成になっていると思います．

情報入試体験会の実施結果概要の報告[4]によると，解答時間は60分，第3問の配点は100点満点のうち40点であり，前半の計算手順の設問の得点率は高かったが，後半のプログラミングの設問の得点率は低かったそうです．限られた試験時間の中で，プログラムの動作を考えることは大変だったのかもしれません．しかし，報告書にも記載されているとおり，前半の計算手順は理解できているとのことなので，後半のプログラミングについても十分に伸びしろがあると思われます．

なお，試験中に問題が解けたかどうかにかかわらず，試験後に自分で図を描いて考えたり，プログラムを書いて動作を確認してみるとよいとでしょう．そのような過程を楽しめる人こそ，「プログラムと解決能力」の領域の力を伸ばすことができる人材だと思います．

付録：Pythonによるプログラミング

実際にPythonでプログラミングしたコードを示します．変更点として，「視覚化する」という関数を追加し，入れ換えと並べ替えの処理前の途中経過も表示するようにしました．「視覚化する」関数では「変化する部分の左端」と「入れ換え対象」を強調表示しています．

```
# 関数「視覚化する」のために rich モジュールの print をインポート
from rich import print

# 問題文中の関数
要素数 = len
表示する = print
def 入れ換える(i, j):
    Narabi[i], Narabi[j] = Narabi[j], Narabi[i]
def 降順にする(i, j):
    Narabi[i:j+1] = sorted(Narabi[i:j+1], reverse=True)
def 逆順にする(i, j):
    Narabi[i:j+1] = reversed(Narabi[i:j+1])

# 「変化しない数字列」を地味に表示,「変化する部分の左端(i)」と「交換対象(j)」の数字を強調表示する関数
def 視覚化する(i, j):
    narabi_str = []
    for k in range(要素数(Narabi)):
        if k < i:
            tag = "[gray70]"
        elif k == i:
            tag = "[underline black on sky_blue1]"
        elif k == j:
            tag = "[underline black on orange1]"
        else:
            tag = "[black]"
        narabi_str.append(tag+str(Narabi[k])+"[/]")
    print("["+", ".join(narabi_str)+"]")

# 数字の並び配列の初期化
```

```
Narabi = [1, 2, 3, 4, 5, 6]

# 処理の本体(最も大きい並びとなるまで表示を繰り返す)
ketasu = 要素数(Narabi)
while True:
    i = ketasu - 2
    while i >= 0 and Narabi[i] >= Narabi[i+1]:
        i = i - 1
    if i >= 0:
        j = ketasu - 1
        while i < j and Narabi[i] >= Narabi[j]:
            j = j - 1
        視覚化する(i, j) # 追加
        入れ換える(i, j)
        逆順にする(i+1, ketasu-1)
        表示する(Narabi)
    else:
        表示する("最も大きい並びです")
        break
```

図Oが実行結果です．出力結果は長いため，開始部と終了部のみ示しています．通常のNarabiの出力に加えて，強調された出力が追加されています．

図O　実行結果

参考文献

1）電気通信大学：【報告】2025年度入試に向けた「情報」入試／CBTに関する体験会を開催，https://www.uec.ac.jp/news/announcement/2023/20231204_5822.html（2024年2月18日閲覧）
2）水野修治：令和7年度大学入学共通テスト『情報I』の実施に向けて～問題作成方針に関する検討の方向性と試作問題～，情報処理，Vol.64，No.2，pp.74-77 (2023)，http://doi.org/10.20729/00223448

3）岩崎英哉：ドント方式による議席配分，情報処理，Vol.64，No.11，pp.e41-e54 (2023)，
http://doi.org/10.20729/00228384
4）小宮常康，渡辺博芳，中山泰一，成見　哲，山路浩夫：電気通信大学における情報入試体験会の実施結果概要の報告，情報処理学会第86回全国大会，2H-06（2024年3月15日），
https://uec.repo.nii.ac.jp/records/2000092

（2024年2月18日受付）
（2024年3月6日note公開）

■**佐藤　喬**（正会員）
2014年，電気通信大学大学院情報システム学研究科博士後期課程修了，博士（工学）．現在，東京都立産業技術高等専門学校准教授．オペレーティングシステムとICTインフラ技術に興味を持つ．本会情報入試委員会委員．シニア会員．