情　　報

Ⅰ

〔1〕 次の（ア）～（コ）の項目について正しく説明しているものを指定された選択肢のうちからすべて選び，解答欄にマークせよ。

（ア）コンピュータを構成する基本装置について

（ア）の選択肢

⓪　中央処理装置（CPU）は，演算装置と制御装置から構成される。

①　主記憶装置（メインメモリ）は，プログラムやデータを一時的に記憶する装置である。

②　補助記憶装置は，主記憶装置より高速かつ大容量である。

③　主記憶装置に記憶されたプログラムは，補助記憶装置に取り出された後に中央処理装置で実行される。

④　コンピュータは，入力・演算・記憶・出力のいずれか一つの装置だけでも動作する。

（イ）オペレーティングシステム（OS）について

（イ）の選択肢

⓪　OS は，アプリケーションを土台として動くソフトウェアである。

①　OS は，主記憶装置や補助記憶装置を管理している。

②　OS は，ユーザのプログラムを実行したり制御したりする。

③　OS は，ユーザの操作を制限するため，利便性の観点から削除した方がよい。

④　OS は，アプリケーションとハードウェアの仲立ちをする。

（ウ）インターネットにおける通信の階層構造について

（ウ）の選択肢

⓪　トランスポート層では，IP アドレスの割当てをしたり，インターネットプロトコル（IP）を実現したりする。

①　トランスポート層では，信頼性の高い通信のための送受信制御を行う。

②　ネットワークインターフェース層（またはネットワークアクセス層）では，アプリケーションの用途に応じて通信内容を決定する。

③　アプリケーション層では，Web やメールのサービスに関わる通信処理を行う。

④　インターネット層では，IP アドレスに基づいて通信する。

（エ）TCP/IP におけるルーティングについて

（エ）の選択肢

⓪　ルーティングとは，データを適切な経路で目的地に届けるために通信経路を選択する仕組みである。

①　通信相手のユーザ名を指定することで，通信経路を指定できる。

②　ルーティングは，情報を収集し，インターネット利用料金を計算するための仕組みを持っている。

③　ルーティングは，通信経路の途中で起きた問題により受信できなかったデータを再送する仕組みを持っている。

④　通信経路は，ネットワークの障害などに応じて動的に変化することがある。

（オ）情報の表現について

（オ）の選択肢

⓪　データには，離散的な量として扱うアナログデータと連続的な量として扱うデジタルデータがある。

①　音や画像をデジタル化する際には，標本化，量子化，符号化が行われる。

②　アナログデータをデジタル化する際，標本化と量子化においてデータの劣化が発生する。

③　可逆圧縮の方式によっては，圧縮前のデータに完全に戻すことができないことがある。

④　デジタルデータは，情報の種類によらず，さまざまな媒体に記録できる。

（カ）情報デザインについて

（カ）の選択肢

⓪　近年の情報システムでは，マウスやタッチパネルを用いることで直感的に操作できるキャラクタユーザインタフェース（CUI）が主流である。

①　情報システムを設計する際には，機器やサービスの使いやすさや使い勝手を担保するセキュリティに考慮する必要がある。

②　Web サイトの構築において，文字の大きさや色を変更する機能を提供することで，アクセシビリティを向上させることができる。

③　国籍，性別，年齢，身長，障がいの有無などによらず，どのような人でも簡単にシステムを利用できるようにすることをユニバーサルデザインという。

④　ピクトグラムを作成する際には，ユニバーサルデザインとは切り離して，その対象をできる限り具体的に表現するのがよい。

（キ）情報セキュリティについて

（キ）の選択肢

⓪　機密性とは，情報が許可された人にしか見られないように保護することである。

①　完全性とは，情報が正しく保たれ，勝手に変更されないよう，暗号化することである。

②　可用性とは，誰もが自由に情報へアクセスできる状態にすることである。

③　システムのバックアップは，可用性の確保に役立つ対策の一つである。

④　情報セキュリティでは，技術的対策に加え，人的・物理的な対策も必要である。

（ク）認証について

（ク）の選択肢

⓪　認証は，コンピュータにアクセスしている人が事前に登録された利用者であるかどうかを確認することである。

①　認証が成功すると，ただちにシステムのすべての機能やデータに無制限にアクセスできる。

②　生体認証（バイオメトリクス認証）は，パスワードを覚える必要がなく，紛失する心配も少ないのが利点である。

③　パスワードは，利用者を識別するために必要なものであるため，秘密情報として扱うべきである。

④　生年月日は，個人情報保護法により適正に管理されているので，パスワードとして使用するのに適している。

（ケ）マルウェア（コンピュータウイルス）について

（ケ）の選択肢

⓪　ウイルス対策ソフトを入れておけば，マルウェアに感染することはない。

①　マルウェアによって，個人情報やパスワードが盗まれる可能性がある。

②　マルウェアが原因で，コンピュータの動作が遅くなることや，停止することがある。

③　マルウェアによって，利用者が知らないうちに他人への攻撃に加担してしまうことがある。

④　コンピュータをネットワークへ接続しなければ，マルウェアに感染することはない。

（コ）ファイアウォールについて

（コ）の選択肢

⓪　ファイアウォールは，内部のコンピュータ同士の通信に加え，外部のコンピュータ同士の通信についても，不要な通信を制限できる。

①　ファイアウォールは，外部から内部への通信だけでなく，内部から外部への通信についても不要な通信を制限できる。

②　ファイアウォールを導入する目的の一つに，不正なアクセスを遮断することによる可用性の向上がある。

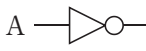
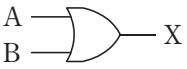
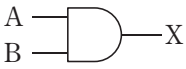
③　外部からの通信をすべて遮断することは，最適なセキュリティの実現と，インターネットの利用を両立するのに重要である。

④　ファイアウォールは，ネットワーク上を流れるデータの内から悪性部分を取り除く。

〔2〕 次の文章中の空欄 サ ～ チ に入れるのに最も適当なものを，指定された選択肢のうちから一つずつ選び，解答欄にマークせよ。同じものを繰り返し選んでもよい。

基本的な論理回路として，否定回路（NOT 回路），論理和回路（OR 回路），論理積回路（AND 回路）の３つがあげられる。これらの図記号と真理値表は表１で示される。真理値表とは，入力と出力の関係を示した表である。

表１　図記号と真理値表

<table>
<tr><th>回路名</th><th>否定回路</th><th>論理和回路</th><th>論理積回路</th></tr>
<tr><td>図記号</td><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td>真理値表</td><td>
<table>
<tr><th>入力</th><th>出力</th></tr>
<tr><th>A</th><th>X</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td><td>
<table>
<tr><th colspan="2">入力</th><th>出力</th></tr>
<tr><th>A</th><th>B</th><th>X</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td><td>
<table>
<tr><th colspan="2">入力</th><th>出力</th></tr>
<tr><th>A</th><th>B</th><th>X</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
</table>

計算機の中央処理装置（CPU）などで演算を担う論理回路には，演算そのものを行うためのものと，制御信号に応じて演算対象のデータの流れを切り替えるためのものがある。ここでは，データの流れを切り替えるための論理回路について考えてみよう。

論理積回路について，入力Ｂをデータの入力，入力Ａを入力Ｂが出力Ｘへどのように伝わるかを切り替えるための制御信号とする使い方を考えてみることにする。論理積回路の真理値表で入力Ａが０である場合だけに注目すると，入力Ｂと出力Ｘの関係は サ といえる。一方，入力Ａが１である場合だけに注目すると，入力Ｂと出力Ｘの関係は シ といえる。

同様に，表２の真理値表で表される論理回路について，入力Ｂと入力Ｃをデータの入力，入力Ａを入力Ｂと入力Ｃが出力Ｙへどのように伝わるかを切り替えるための制御信号とする使い方を考えてみることにする。表２の真理値表で入力Ａが０である場合だけに注目すると，入力Ｂおよび入力Ｃと出力Ｙの関係は［ス］といえる。一方，入力Ａが１である場合だけに注目すると，入力Ｂおよび入力Ｃと出力Ｙの関係は［セ］といえる。

表２　真理値表

入力			出力
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

表１の図記号を用いて表２の真理値表を実現したものが，図１に示す論理回路である。ただし，図１のＰは表２の真理値表の［ソ］，Ｑは表２の真理値表の［タ］，Ｒは表２の真理値表の［チ］に対応する。

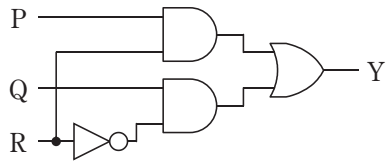


図１　論理回路

サ シ の選択肢

⓪　出力 X の値は入力 B の値に関わらず，常に 0

①　出力 X の値は入力 B の値に関わらず，常に 1

②　出力 X の値は常に入力 B の値と同じ

③　出力 X の値は常に入力 B の値と異なる

ス セ の選択肢

⓪　出力 Y の値は入力 B や入力 C の値に関わらず，常に 0

①　出力 Y の値は入力 B や入力 C の値に関わらず，常に 1

②　出力 Y の値は入力 C の値に関わらず，常に入力 B の値と同じ

③　出力 Y の値は入力 C の値に関わらず，常に入力 B の値と異なる

④　出力 Y の値は入力 B の値に関わらず，常に入力 C の値と同じ

⑤　出力 Y の値は入力 B の値に関わらず，常に入力 C の値と異なる

ソ タ チ の選択肢

⓪　入力 A

①　入力 B

②　入力 C

〔3〕 次の文章中の空欄 ツ に入れるのに適当なものを，指定された選択肢のうちからすべて選び，解答欄にマークせよ。また，空欄 テ と ト に当てはまる適当な数値を，解答欄に記入せよ。

P さんは会社で二次元コードを設計している。会社で現在用いている二次元コードを図 2 に示す。このコードの条件は，以下(a)～(e)のとおりである。

(a) コードは 1 辺16 cm の正方形をしている。

(b) コードには左下を原点として，X 軸方向に16 cm，Y 軸方向に 2 cm の黒色部分がある。また，原点から X 軸方向に 2 cm，Y 軸方向に16 cm の黒色部分がある。

(c) コードには，1 辺10 cm の正方形をしたデータ部があり，データ部の周囲 2 cm は白色である。

(d) データ部には，1 辺 2 cm で等間隔に区切られた25個のセルがある。

(e) データ部において，それぞれのセルは白色または黒色で，データを 2 値で表現する。

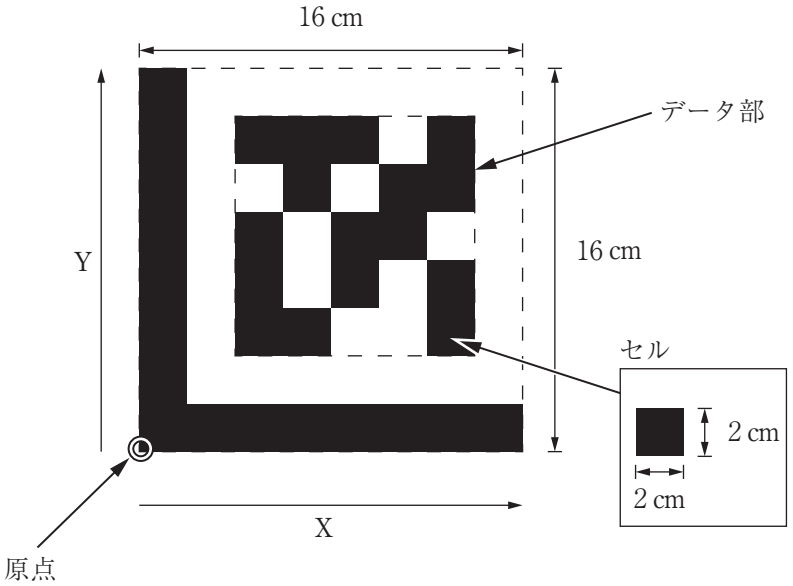


図 2　二次元コード

Pさんは，(a)(b)(c)の条件を保ったまま，二次元コードで表現できるデータ量を増やすよう改善する仕事を会社から依頼された。データ量を増やすためにできることは［ツ］ことである。［ツ］は該当するものをすべて選ぶこと。また，読み取りの技術的制約は考慮しなくてよい。

Pさんは，二次元コードのデータ量を増やすためのいくつかのアイディアを持って上司に相談したところ，(a)(b)(c)の条件だけでなく，(d)の条件も保ったまま，(e)の条件のみを変更せよ，との指示を受けた。

そこで，データを複数の色で表現することでデータ量を増やすアイディアに注目した。白・黒の２値の場合は，１つのセルで表現できるデータ量は１ビットになり，表現できるデータ量は二次元コード全体で２の25乗となる。また，白・黒・赤・青・緑・紫・黄・茶の８値の場合は，１つのセルで表現できるデータ量は［テ］ビットとなり，表現できるデータ量は二次元コード全体で［ト］の25乗となる。

［ツ］の選択肢

⓪　データ部のセルの数を25個から64個に増やす

①　データ部に誤り訂正符号を新たに導入する

②　それぞれのセルにおけるデータの表現を白・黒から白・黒・灰色に変更する

③　セルの面積を大きくする

④　それぞれのセルにおけるデータの表現を白・黒から○（マル）と×（バツ）に変更する

〔4〕 次の文章中の空欄 ナ ～ ネ に入れるのに最も適当なものを，指定された選択肢のうちから一つずつ選び，解答欄にマークせよ。同じものを繰り返し選んでもよい。また，空欄 ノ ～ ヒ に入れるのに適当なものを，指定された選択肢のうちからすべて選び，解答欄にマークせよ。

「人生を豊かに暮らすためには，お金と自由時間がともに大切である」という考え方がある。それを調べるために，学生40名と社会人40名にアンケート調査を行った。調査したデータは，表計算ソフトウェアを使って，表３のようにまとめられている。

表３　アンケート調査データの一部（データシート）

	A	B	C	D
1	学生・社会人の別	ポケットマネー	余暇時間	満足度
2	1	49,800	4.5	3
3	1	69,300	9.3	8
4	2	124,800	2.8	5
5	2	93,600	4.7	7

調査した項目は以下のものである。

- 学生・社会人の別：学生を１，社会人を２で表す。
- ポケットマネー：１か月あたりに自由に使えるお金，お小遣い。100円未満は四捨五入する。
- 余暇時間：食事や睡眠，仕事や学業や家事などを除いて，１日あたりに自由に使える時間。１時間30分を1.5のように10進法で表す。小数第２位を四捨五入する。
- 満足度：仕事や学業や趣味や生活に対する満足度。１～10点の10点満点で各自が答えた評価点であり，正の整数で表す。数が大きいほど高い満足度を示す。

情報を数値で表すときの性質の違いを尺度と呼ぶ。このアンケート調査では，学生・社会人の別は ナ である。ポケットマネーを表す数値は ニ で，余暇時間は ヌ で，満足度は ネ である。

このデータをもとに，項目間の関係を調べることにした。データの加工には，表計算ソフトウェアを活用した。大学生40名について，図3(a)にはポケットマネーと満足度の関係が，図3(b)には余暇時間と満足度の関係が示されている。折れ線は，階級ごとの満足度の平均値を表している。図3(a)(b)より得られる情報から正しいものをすべて選ぶと，［ ノ ］となる。

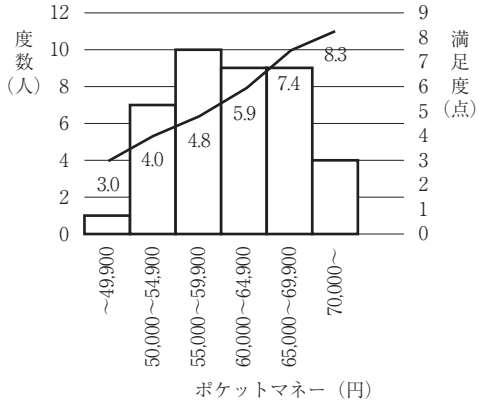


図3(a)　ポケットマネーと満足度

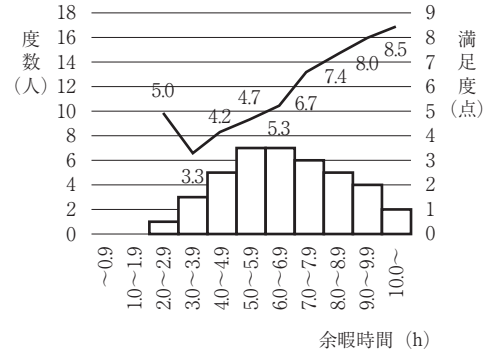


図3(b)　余暇時間と満足度

同じように，社会人40名について，図4(a)にはポケットマネーと満足度の関係が，図4(b)には余暇時間と満足度の関係が示されている。折れ線は，階級ごとの満足度の平均値を表している。図4(a)(b)より得られる情報から正しいものをすべて選ぶと，［ ハ ］となる。

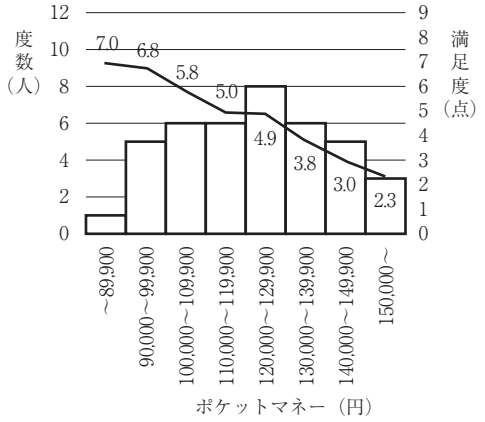


図4(a)　ポケットマネーと満足度

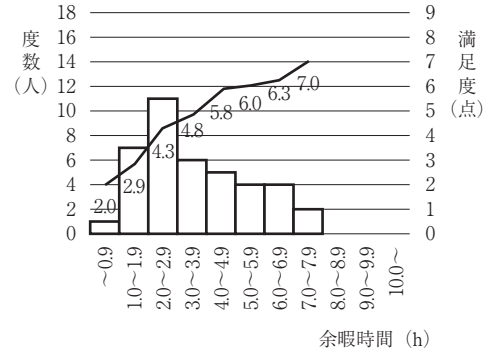


図4（b）　余暇時間と満足度

次に，箱ひげ図を描いて，学生データと社会人データの分布を調べてみた。図5(a)にはポケットマネーについて，図5(b)には余暇時間について，それぞれ学生と社会人が比較されている。図の×は平均値を表す。図5(a)(b)より得られる情報から正しいものをすべて選ぶと，［ ヒ ］となる。

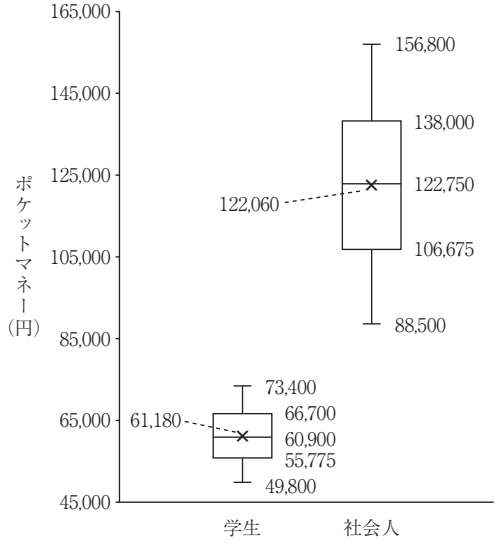


図5(a)　ポケットマネーの比較

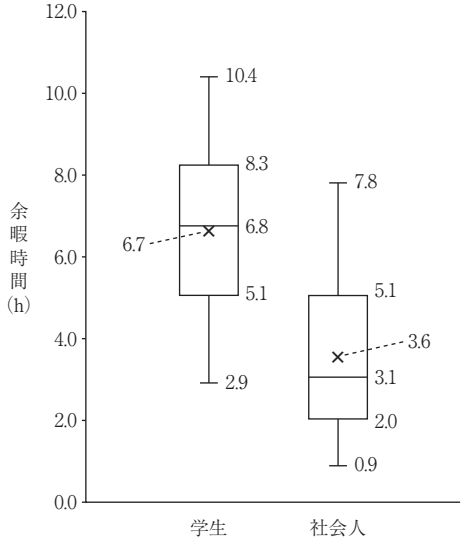


図5(b)　余暇時間の比較

ナ ニ ヌ ネ の選択肢

⓪　名義尺度　　①　順序尺度　　②　間隔尺度　　③　比例尺度

ノ の選択肢

⓪　学業に使う時間が増えれば，満足度の平均値が高くなる

①　ポケットマネーについて，度数が最も大きい階級と，最も小さい階級の満足度の平均値の差は2.0以上ある

②　ポケットマネーが多い階級ほど，満足度の平均値は高くなる

③　ポケットマネーが多いほど，余暇時間は長くなる

④　余暇時間が増えると，満足度が必ず上がる

ハ の選択肢

⓪　仕事に使う時間が増えれば，満足度の平均値が高くなる

①　ポケットマネーが増えたとしても，満足度の平均値は上がらない

②　余暇時間が長い方から４分の１の社会人は，全員の満足度が6.0より大きい

③　余暇時間が多い階級ほど，満足度の平均値は高くなる

④　社会人にとって，ポケットマネーも余暇時間も満足度とは関係がない

ヒ の選択肢

⓪　すべての学生のポケットマネーは，どの社会人のポケットマネーよりも少ない

①　学生の場合，ポケットマネーの中央値と平均値は同じである

②　すべての社会人のポケットマネーは10万円を超える

③　学生の余暇時間は，社会人の余暇時間より，平均で見て２時間以上長い

④　社会人の余暇時間の平均値よりも余暇時間の長い学生は30人以上いる

Ⅱ

次の文章中の空欄 ア カ ～ コ シ ス に入れるのに適当なものを，指定された選択肢のうちからすべて選び，解答欄にマークせよ。空欄 イ に入れるのに最も適当なものを，指定された選択肢のうちから一つ選び，解答欄にマークせよ。空欄 ウ ～ オ サ セ ソ に当てはまる適当な数値または式を，解答欄に記入せよ。

F さんは，自分が作った作品をある山の展望台で土産物として販売してもらい，その収益を慈善団体に寄付する計画を立てた。この計画の賛同者が，作品を運搬するドローンを手配してくれることになった。作品の運搬にあたっては，手配されたドローンに積載可能な重量を守る必要がある。そこで F さんは，ドローンに積載することができる重量以内でなるべく収益の総和（総収益の見込み）が大きくなるよう，作品を選んで積載することにした。以下，同じ作品名の作品は 1 個しかないものとする。

〔1〕 表 1 に示す A～E の作品があるとき，重量700 g まで積載できるドローンを 1 回だけ使用して作品を運ぶことを考える。はじめに F さんは「収益の大きいものから順に判定する」というアルゴリズム X を考えた。5 つの作品を収益の大きい順に並べると，A，B，C，D，E の順になる。以下，この順で 1 個ずつドローンに積載できるかどうかを判定する。

表 1　収益と重量（1）

作品名	収益（円）	重量（g）
A	500	500
B	440	400
C	360	300
D	300	200
E	200	100

(1) 最初に，最も収益が大きい A について判定する。A は500 g なので積載可能である。そこで，まず A をドローンに積載する。

(2) 次に収益が大きい B について判定する。積載決定済みの A の重量に B の重量を合計すると900 g となり，B は積載できない。

(3) 次に収益が大きい C について判定する。積載決定済みの A の重量に C の

重量を合計すると800 g となり，C は積載できない。

(4) 次に収益が大きいDについて判定する。積載決定済みのAの重量にDの重量を合計しても700 g なので，D は積載可能である。そこでAに加えDをドローンに積載する。

(5) 最後に最も収益が小さいEについて判定する。積載決定済みのAとDの重量にEの重量を合計すると800 g となり，E は積載できない。

以上5回の判定からAとDを積載することが決定された。この場合，総重量は700 g，総収益の見込みは800円となる。

次に，F さんは「重量の軽いものから順に判定する」というアルゴリズムYを考えた。まず，最も軽いEは100 g なので積載可能である。そこで，これをドローンに積載する。以下，順に積載可否を判定し，積載可能な場合は積載する。

この結果，［ア］が積載され，総重量は［イ］g，総収益の見込みは860円となる。［ア］は積載される作品の選択肢をすべて選ぶこと。

アルゴリズムXとアルゴリズムYの結果を比べると，アルゴリズムYの方が総収益の見込みが大きいので，良い結果が得られたことになる。

ここで，F さんはいずれのアルゴリズムも短時間で判定が終了したと感じたが，それは判定の回数がわずか5回だったからであろう。作品が n 個あった場合，いずれのアルゴリズムでも判定の回数は［ウ］回で済む。

ところで，5個の作品の中から0個以上5個以下の作品を選ぶ組合せは全部で［エ］通りある。そこで，F さんは念のため，そのすべてについて総重量と総収益の見込みを求め，総重量700 g 以内で最も総収益の見込みが大きい組合せを探した。その結果，BとDとEを選べば総重量700 g で総収益の見込みを940円にできることがわかった。この方法を作品が n 個あった場合に適用すると，0個以上 n 個以下の作品を選ぶ組合せは全部で［オ］通りある。n が大きいときは何らかの工夫をしないといけないとF さんは感じた。

〔2〕 表2に示すJ～Nの作品があるとき，〔1〕と同様に，重量1000 gまで積載できるドローンを1回だけ使用して作品を運ぶことを考える。

表2　収益と重量（2）

作品名	収益（円）	重量（g）
J	350	700
K	320	500
L	180	300
M	80	200
N	45	100

アルゴリズムXを適用した場合，［カ］が積載される。［カ］は積載される作品の選択肢をすべて選ぶこと。

アルゴリズムYを適用した場合，［キ］が積載される。［キ］は積載される作品の選択肢をすべて選ぶこと。

いま，作品の収益を重量で割った値を収益重量比と呼ぶことにしよう。「収益重量比の大きい作品から順に判定する」というアルゴリズムZを考えた場合，［ク］が積載される。［ク］は積載される作品の選択肢をすべて選ぶこと。

これらの結果を見て，アルゴリズムZはアルゴリズムXやアルゴリズムYに比べて良さそうであるとFさんは感じた。

〔3〕 表3に示すP～Vの作品があるとき，重量1000 g まで積載できるドローン10号と重量500 g まで積載できるドローン5号をそれぞれ1回だけ使用して作品を運ぶことを考える。

表3　収益と重量（3）

作品名	収益（円）	重量（g）
P	585	900
Q	420	600
R	280	700
S	200	400
T	180	300
U	110	200
V	45	100

まず，作品P～Vすべてを対象にアルゴリズムZを適用してドローン10号に作品を積載すると，［ケ］が積載され，次に，残った作品を対象にアルゴリズムZを適用してドローン5号に作品を積載すると，［コ］が積載される。［ケ］と［コ］はそれぞれ積載される作品の選択肢をすべて選ぶこと。この結果，2機のドローンに積載された作品の総収益の見込みは［サ］円である。

一方，作品P～Vすべてを対象にアルゴリズムZを適用してドローン5号に作品を積載すると，［シ］が積載され，次に，残った作品を対象にアルゴリズムZを適用してドローン10号に作品を積載すると，［ス］が積載される。［シ］と［ス］はそれぞれ積載される作品の選択肢をすべて選ぶこと。この結果，2機のドローンに積載された作品の総収益の見込みは［セ］円である。

以上より，2つ以上のドローンがあるときにアルゴリズムZを各ドローンに順に適用する場合，適用するドローンの順番を変えると総収益の見込みが大きく変わってしまうことがわかる。総収益の見込みを最大にするためには，やはりドローンに積載する作品の可能な組合せすべてについて，総重量と総収益の見込みを求めるのが確実であろうとFさんは考えた。ただし，作品が n 個，ドローンが m 機あった場合，作品をドローンに積載する組合せの数は［ソ］通りあるので，すべてを調べるのには多大な手間がかかりそうである。

ア の選択肢

⓪ A　① B　② C　③ D　④ E

イ の選択肢

⓪ 100　① 200　② 300　③ 400　④ 500　⑤ 600　⑥ 700

カ の選択肢

⓪ J　① K　② L　③ M　④ N

キ の選択肢

⓪ J　① K　② L　③ M　④ N

ク の選択肢

⓪ J　① K　② L　③ M　④ N

ケ の選択肢

⓪ P　① Q　② R　③ S　④ T　⑤ U　⑥ V

コ の選択肢

⓪ P　① Q　② R　③ S　④ T　⑤ U　⑥ V

シ の選択肢

⓪ P　① Q　② R　③ S　④ T　⑤ U　⑥ V

ス の選択肢

⓪ P　① Q　② R　③ S　④ T　⑤ U　⑥ V

Ⅲ

次の文章中の空欄 ア ～ コ シ に入れるのに最も適当なものを，指定された選択肢のうちから一つずつ選び，解答欄にマークせよ。 ア ～ ケ については，同じものを繰り返し選んでもよい。また，空欄 サ ス に入れるのに適当なものを，指定された選択肢のうちからすべて選び，解答欄にマークせよ。

図1に示す電卓を利用して，加減算を行う場面を考える。「1+2」の計算を行うときの手順は以下のようになる。

⑴ 「1」のキーを押す。

⑵ 「+」のキーを押す。

⑶ 「2」のキーを押す。

⑷ 「=」のキーを押す。

なお，電卓の使用において，計算の前には「AC」（クリア）キーを押すが，ここでは手順に含めないこととする。

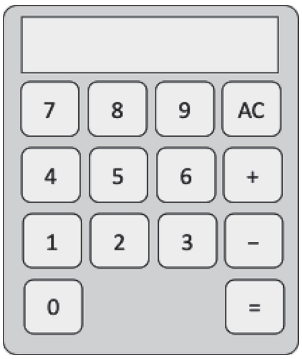


図1　電卓

ここでは，「x」のキーを押すことを，「@x」と表記する。たとえば，「1」のキーを押す操作を「@1」，「+」のキーを押す操作を「@+」と表す。この表記を用いて，上記の手順を図2のように書くことにする。これを状態遷移図という。

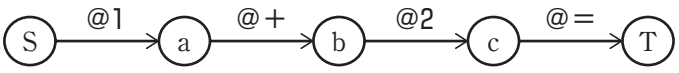


図2　「1+2」の計算を行う際の手順を表す状態遷移図

5個の丸（○）はキーを押した前か後の電卓の状態を表している。矢印は，キーを押す操作を表す。上記の図では，Ⓢ，ⓐ，ⓑ，ⓒ，Ⓣの5つの状態がある。Ⓢは開始を表す状態，Ⓣは終了を表す状態である。Ⓢ，ⓐ，ⓑ，ⓒはそれぞれ「1」，「+」，「2」，「=」のキーを押す前の状態を表している。また，ⓐ，ⓑ，ⓒ，Ⓣはそれぞれ「1」，「+」，「2」，「=」のキーを押した後の状態を表しているともいえる。

〔1〕 上記で説明した状態遷移図の書き方に従うと，「8−5+2」を計算する際の手順は，図3のように書くことができる。

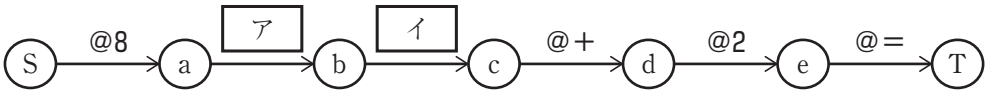


図3 「8−5+2」の計算を行う際の手順を表す状態遷移図

図1の電卓の使用において，加減算を行う対象の数値は1～9の整数とする。ここでは，1～9のうちどれか一つの数字のキーを押すことを「@1^9」と表す。この表記を用いると，1～9の整数の加減算を行う際の手順は図4のように書くことができる。

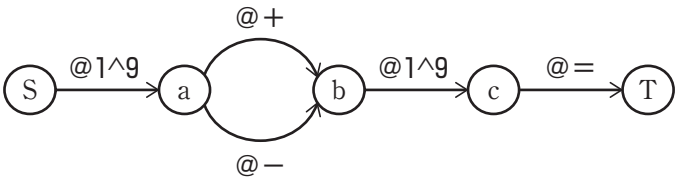


図4 1～9の整数の加減算を行う際の手順を表す状態遷移図

図4の手順には分岐と合流がある。加算「1+2」の計算の場合は，「@＋」の矢印に対応するようにキーを押したことになる。一方で，減算「8−5」の計算の場合は，「@−」の矢印に対応するようにキーを押したことになる。

残念ながら，図4に示す手順では1桁の整数の加減算の手順しか表現していない。そこで，1桁（0を除く1～9）あるいは2桁の整数（10～99）の加減算の手順を表現した状態遷移図を書くことにする。2桁の整数は，［ウ］の数字のキーを1回押した後に，［エ］の数字のキーを1回押せばよい。よって，1桁の整数あるいは2桁の整数の加減算を行う際の手順は図5のように書くことができる。

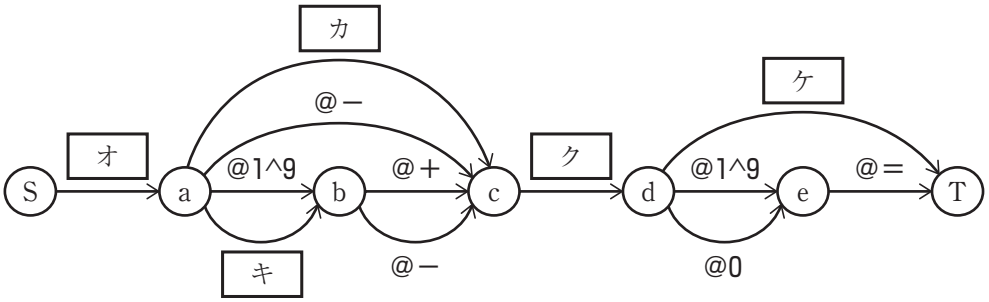


図５　１桁あるいは２桁の整数の加減算を行う際の手順を表す状態遷移図

〔２〕〔１〕の状態遷移図において，「@1」や「@＋」の操作を「１」や「＋」のキーを押すことではなく，「１」や「＋」を並べることと考える。これにより，状態遷移図から計算式を作成できる。計算式の作成方法は，以下のようになる。

⑴　状態Ⓢから状態Ⓣまで矢印をたどり，矢印についている文字を順番に並べる。

⑵　「@」記号をすべて削除する。

たとえば，図６において，状態Ⓢから状態Ⓣまで矢印をたどることで並べられた文字の列は「@3@＋@7@＝」となる。「@」を削除すると，残る文字の列は「3＋7＝」となる。つまり，図６の状態遷移図から「3＋7＝」という計算式を作成できる。

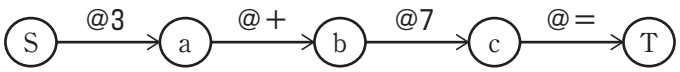


図６　計算式「3＋7＝」を作成する状態遷移図

「@1^9」を@1～@9のうち，どれか一つとみなすことにすると，図７の状態遷移図からは，複数の計算式「3＋j＝」（j は１～９の整数）が作成できる。一方で，「3＋0＝」の計算式を作ろうとすると，状態Ⓢから状態ⓑまで矢印をたどることで並べられた文字の列は「［コ］」となるが，その次の文字を並べることができない。よって，図７の状態遷移図からは，「3＋0＝」の計算式

を作成できない。

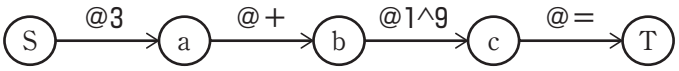


図７　計算式「3＋j＝」（j は１～９の整数）を作成する状態遷移図

状態遷移図において，矢印は自由に状態を結んでよい。図８に示す状態遷移図において，状態ⓒから状態ⓑに戻る矢印を通らない場合には計算式「3＋7＝」が，戻る矢印を１回通る場合には計算式「3＋7＋7＝」が作成される。

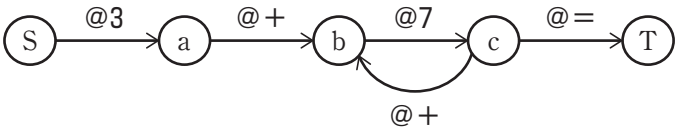


図８　計算式「3＋7＝」「3＋7＋7＝」などを作成する状態遷移図

図９の状態遷移図から作成できる計算式を，選択肢のうちからすべて選ぶと，［サ］となる。

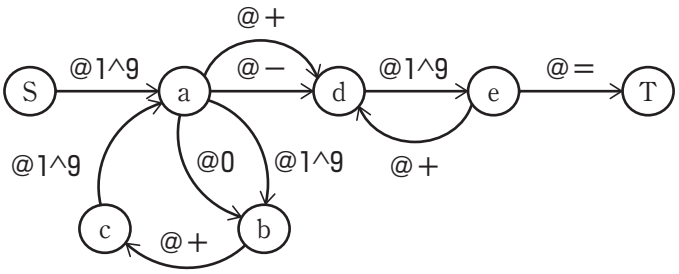


図９　計算式を作成する状態遷移図（１）

〔3〕 計算式を作成する状態遷移図において，「@」のみを並べる操作を持つ矢印を追加する。たとえば，図10の状態遷移図において，状態Ⓢから状態Ⓣまで矢印をたどることで並べられた文字の列は「@3@@＋@7@＝」となる。「@」を削除すると，残る文字の列は「3＋7＝」となる。

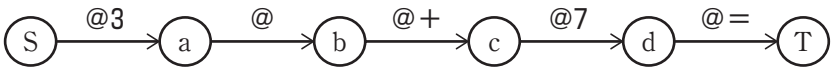


図10　計算式を作成する状態遷移図（2）

このように，計算式「3＋7＝」は，図6の状態遷移図を用いても図10の状態遷移図を用いても作成できる。図11の状態遷移図と同じ計算式を作成できて，かつ同じ計算式しか作成できない状態遷移図は［ シ ］である。

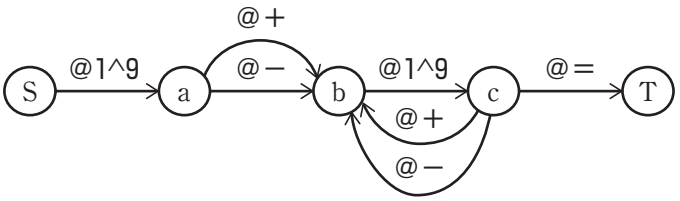


図11　計算式を作成する状態遷移図（3）

また，図12の状態遷移図から作成できる計算式を，選択肢のうちからすべて選ぶと，［ ス ］となる。

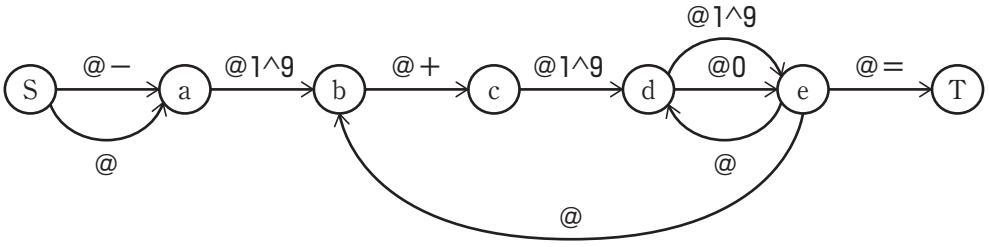


図12　計算式を作成する状態遷移図（4）

ア　イ　の選択肢

⓪　@8　　①　@5　　②　@2　　③　@＋　　④　@－

ウ　エ　の選択肢

⓪　0　　①　1　　②　0～9　　③　1～9　　④　+

⑤　－　　⑥　=

オ　カ　キ　ク　ケ　の選択肢

⓪　@0　　①　@1　　②　@1^9

③　@＋　　④　@－　　⑤　@＝

コ　の選択肢

⓪　@3　　①　@0　　②　@3@＋

③　@0@＋　　④　@3@＋@0　　⑤　@0@＋@3

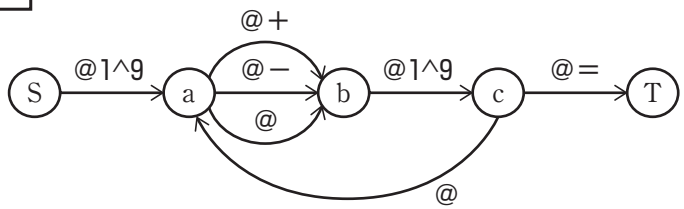
サ　の選択肢

⓪　7－8＋12＝　　①　6＋2＋5＋4＝　　②　3＋5－7＋1＝

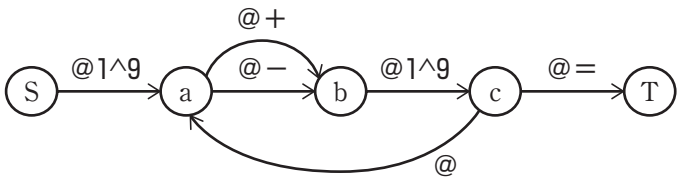
③　40＋98＋5－8＝　　④　22＋65＋48＝　　⑤　61－5＝

シ の選択肢

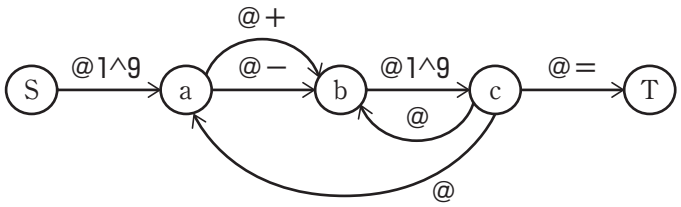
⓪



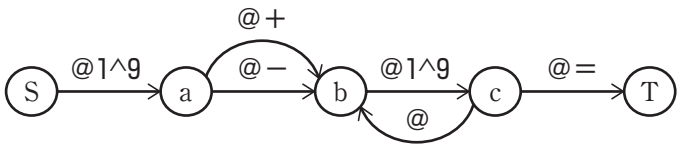
①



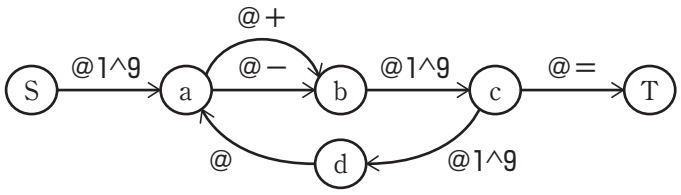
②



③



④



ス の選択肢

⓪ 7－125＝　　① －2＋30＝

② －9＋503＋7＝　　③ －5＋30104＋68＋207＝

Ⅳ

次の文章中の空欄 ア ～ タ に入れるのに最も適当なものを，指定された選択肢のうちから一つずつ選び，解答欄にマークせよ。 イ エ カ ～ サ ス セ については，同じものを繰り返し選んでもよい。巻末の「プログラム表記の例示」を参考にせよ。

K さんは，近所の鉄道会社向けの料金計算アプリを開発することにした。Web ページを調べると，乗車料金は，乗降する駅間の路線の距離に応じて，次のように規定されていた。

- 3 km 以内は150円
- それ以降は，1 km までごとに50円加算

たとえば，3 km を越えて 4 km 以内は200円，4 km を越えて 5 km 以内は250円となる。

図 1 に示すように，鉄道会社は枝分かれのない路線を所有している。10駅あり，各駅には駅名 A～J が付けられ，かつ 0 ～ 9 の駅番号が割り当てられている。また，A から J 方向を下り，J から A 方向を上りと呼ぶ。

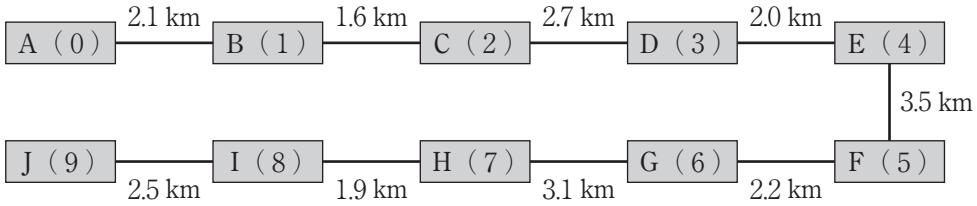


図 1　路線と駅間距離

プログラムは，図 2 のような動作を考えており，乗車駅と降車駅の駅名を入力すると，区間，距離，乗車料金が表示される。ただし，駅名に A～J 以外が指定された場合，あるいは乗車駅と降車駅に同じ駅が指定された場合，図 3 のように「入力が正しくありません」と表示される。

```
乗車駅を入力してください（A～J）：J
降車駅を入力してください（A～J）：C
区間：J駅 → C駅
距離：17.9 km
乗車料金：900 円
```

図２　プログラムの動作例

```
乗車駅を入力してください（A～J）：D
降車駅を入力してください（A～J）：D
入力が正しくありません
```

図３　プログラムの動作例
（乗車駅と降車駅が同一の場合）

プログラムとしては，次の機能を持つ関数があればよいであろうと考えた。

- 駅名を駅番号に変換する関数 **ekimei_henkan()**
- 乗車駅と降車駅の駅番号を与えると，距離を返す関数 **kyori_keisan()**
- 距離を与えると，乗車料金を返す関数 **ryokin_keisan()**
- 上記の関数を使って乗車料金を表示する関数 **ryokin_hyoji()**

〔1〕 K さんは，まず，距離（**kyori**）を指定すると，乗車料金（**ryokin**）を計算し，計算結果を返す関数 **ryokin_keisan()** を作成した。なお，**kyori** と **ryokin** は変数を表す。

作成時の方針や工夫したこと

- **kyori** に格納される値は正である必要がある。これについては，この関数を呼び出す時にチェックをすることにした。そのため，ここでは **kyori** には必ず正の値が与えられると仮定してプログラムを作成した。
- 関数内では，数値を与えると小数部分を切り上げて整数を返す関数「切り上げ**()**」を使った。たとえば，「切り上げ**(1.2)**」では **2** が，「切り上げ**(3.0)**」では **3** が返ってくる。

作成した関数 **ryokin_keisan(kyori)** は以下のようになる。

関数 **ryokin_keisan(kyori)**

```
(01) ryokin = 150
(02) もし [ ア ] ならば：
(03)   ryokin = [ イ ] + ( 切り上げ( [ ウ ] ) * [ エ ] )
(04) ryokin の値を返す
```

〔2〕 次に，K さんは，乗車駅(**from**)と降車駅(**to**)を，0〜9 の駅番号で与えると，距離(**kyori**)を計算し，計算結果を返す関数 **kyori_keisan()** を作成した。なお，**from**，**to**，**kyori** は変数を表す。

作成時の方針や工夫したこと

- 上りと下りのどちらの方向を指定されても正しく動作するようにした。
- **from** と **to** に格納される駅番号は必ず0〜9の値であり，かつ **from** と **to** は異なる値が指定される必要がある。これについては，この関数を呼び出す時にチェックをすることにした。そのため，ここでは，**from** と **to** に格納される値に関する条件が必ず満たされると仮定してプログラムを作成した。
- 駅間の距離の情報には配列を使った。配列 **H_kyori_ekikan** に，AB 間，BC 間，… ，IJ 間の順に，9 個のデータを格納した。配列は **H_kyori_ekikan[0]** から始まり，たとえば **H_kyori_ekikan[0]** には AB 間の距離を，**H_kyori_ekikan[8]** には IJ 間の距離を格納した。

作成した関数 **kyori_keisan(from, to)** は以下のようになる。

関数 **kyori_keisan(from, to)**

```
(01) H_kyori_ekikan =
       [2.1, 1.6, 2.7, 2.0, 3.5, 2.2, 3.1, 1.9, 2.5]
(02) kyori = 0.0
(03) もし［ オ ］ならば：
(04) ｜ i を［ カ ］から［ キ ］まで 1 ずつ増やしながら繰り返す：
(05) ｜ ｜ kyori = kyori + H_kyori_ekikan[i]
(06) そうでなければ：
(07) ｜ i を［ ク ］から［ ケ ］まで 1 ずつ減らしながら繰り返す：
(08) ｜ ｜ kyori = kyori + H_kyori_ekikan[i]
(09) kyori の値を返す
```

〔3〕 その後，Kさんは，駅名(**ekimei**)を与えると，駅番号(**bangou**)に変換し，変換結果を返す関数**ekimei_henkan()**を作成した。なお，**ekimei**と**bangou**は変数を表す。

作成時の方針や工夫したこと

- この関数では，**ekimei**にどのような値が指定されるかわからないと仮定した。駅名にA，B，…，Jのいずれかが指定された場合は，駅名に対応する駅番号（0〜9）を返す。駅名にそれ以外の値（たとえば，X）が指定された場合には，エラーであることを示すために-1を返すことにした。
- 駅名を駅番号に変換するための対応表として，配列**H_ekimei**を使った。配列の要素の値は**H_ekimei[0]**がA，**H_ekimei[1]**がB，…，**H_ekimei[9]**がJとなっており，配列の添え字（配列の番号）が駅番号，その配列の要素の値が駅名となるように対応づけた。

作成した関数**ekimei_henkan(ekimei)**は以下のようになる。

関数**ekimei_henkan(ekimei)**

```
(01) H_ekimei =
     ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]
(02) bangou = -1
(03) i を [ コ ] から [ サ ] まで 1 ずつ増やしながら繰り返す：
(04)  | もし [ シ ] ならば：
(05)  | | bangou = i
(06) bangou の値を返す
```

〔4〕 最後に，K さんは，乗車料金を計算したい乗車駅と降車駅の名前を受け取り，〔1〕～〔3〕で作成した関数を使って乗車料金を計算し，計算結果を表示する関数 **ryokin_hyoji()** を作成した。

作成時の方針や工夫したこと

- 乗車駅と降車駅の名前をキーボードから入力し，それぞれ変数 **ekimei_from** と **ekimei_to** に格納するようにした。
- 駅名が正しく与えられたことを検査して，正しくない場合には，「入力が正しくありません」と表示するようにした。

作成した関数 **ryokin_hyoji()** は以下のようになる。

関数 **ryokin_hyoji()**

```
(01) 表示する("乗車駅を入力してください（A～J）： ")
(02) ekimei_from =【キーボードからの入力】
(03) 表示する("降車駅を入力してください（A～J）： ")
(04) ekimei_to =【キーボードからの入力】
(05) bangou_from = ekimei_henkan(ekimei_from)
(06) bangou_to = ekimei_henkan(ekimei_to)
(07) もし (bangou_from == -1) [ ス ] (bangou_to == -1)
                [ セ ] (bangou_from == bangou_to) ならば：
(08) ｜ 表示する("入力が正しくありません")
(09) そうでなければ：
(10) ｜ kyori = kyori_keisan(bangou_from, bangou_to)
(11) ｜ ryokin = ryokin_keisan(kyori)
(12) ｜ 表示する("区間：", ekimei_from,
                "駅 → ", ekimei_to, "駅")
(13) ｜ 表示する("距離：", kyori, "km")
(14) ⌊ 表示する("乗車料金：", ryokin, "円")
```

〔5〕 後日，鉄道会社はA駅とJ駅を結んで環状線とした。それに合わせて，Kさんは，料金計算アプリを更新することにした。なお，駅の数，駅の名称や番号，乗車料金の計算方法はこれまでと変更がない。新しくできる環状線は，図4に示すような路線となり，AJ間の距離は3.8 kmである。

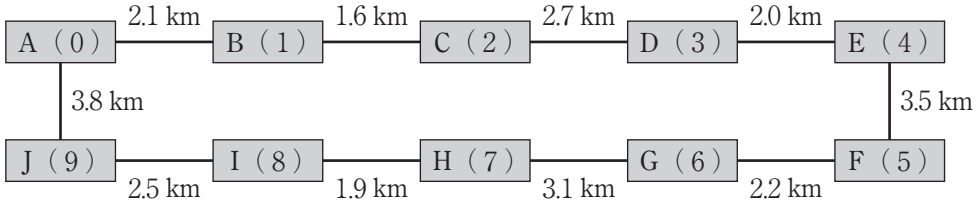


図4　新しい路線と駅間距離

これまで作成したプログラムを再検討した結果，4つの関数のうち，乗車駅と降車駅を駅番号で指定すると距離が得られる関数**kyori_keisan()**について，距離の計算方法を変更しないといけないことがわかった。一方，それ以外の関数は変更不要であることがわかった。

作成時の方針や工夫したこと

- 作成し直す関数**kyori_keisan()**では，**H_kyori_ekikan[9]**にAJ間の距離を格納した。
- まず，**from**から**to**へ時計回り（A，B，C，… の順）で行く場合の距離**kyori_from_to**を計算する。次に，**from**から**to**へ反時計回り（A，J，I，… の順）で行く場合の距離**kyori_to_from**を計算する。ただし，**from**から**to**へ反時計回りで行く場合の距離は，**to**から**from**へ時計回りで行く場合の距離と同じである。通常，図4のような環状線では距離が短い方の経路を選択するので，これら2つの距離のうち，短い方を選択して値を返すことにした。

作成し直した関数 **kyori_keisan(from, to)** は次のようになる。

作成し直した関数 **kyori_keisan(from, to)**

```
(01) H_kyori_ekikan =
     [2.1, 1.6, 2.7, 2.0, 3.5, 2.2, 3.1, 1.9, 2.5, 3.8]
(02) kyori_from_to = 0.0
(03) i = from
(04) i [ ソ ] to の間繰り返す：
(05)  | kyori_from_to = kyori_from_to + H_kyori_ekikan[i]
(06)  └ i = (i + 1) [ タ ] 10
(07) kyori_to_from = 0.0
(08) i = to
(09) i [ ソ ] from の間繰り返す：
(10)  | kyori_to_from = kyori_to_from + H_kyori_ekikan[i]
(11)  └ i = (i + 1) [ タ ] 10
(12) もし kyori_from_to < kyori_to_from ならば：
(13)  | kyori = kyori_from_to
(14) そうでなければ：
(15)  └ kyori = kyori_to_from
(16) kyori の値を返す
```

ア の選択肢

⓪ `kyori > 3.0`　① `kyori == 3.0`　② `kyori == 0.0`

③ `kyori != 3.0`　④ `kyori != 0.0`　⑤ `kyori < 3.0`

イ エ の選択肢

⓪ `ryokin`　① `kyori`　② `0.0`

③ `3.0`　④ `50`　⑤ `200`

ウ の選択肢

⓪ `kyori`　① `kyori + 3.0`　② `kyori - 3.0`

③ `kyori * 50`　④ `ryokin`　⑤ `ryokin + 150`

⑥ `ryokin - 150`　⑦ `ryokin * 50`

オ の選択肢

⓪ `from > to`　① `from < to`

② `from == to`　③ `from != to`

カ キ ク ケ の選択肢

⓪ `from`　① `from + 1`　② `from - 1`

③ `from - to`　④ `to`　⑤ `to + 1`

⑥ `to - 1`　⑦ `to - from`　⑧ `0`

⑨ `9`

コ サ の選択肢

⓪ `bangou`　① `ekimei`　② `-1`　③ `0`

④ `1`　⑤ `8`　⑥ `9`　⑦ `10`

シ の選択肢

⓪ `i == ekimei`
① `i == bangou`
② `bangou == ekimei`
③ `H_ekimei[i] == ekimei`
④ `H_ekimei[i] == bangou`
⑤ `H_ekimei[bangou] == i`
⑥ `H_ekimei[bangou] == ekimei`
⑦ `H_ekimei[ekimei] == i`
⑧ `H_ekimei[ekimei] == bangou`

ス セ の選択肢

⓪ `==`　① `!=`　② `>`　③ `<`　④ `>=`
⑤ `<=`　⑥ `and`　⑦ `or`　⑧ `not`

ソ の選択肢

⓪ `==`　① `!=`　② `>`　③ `<`
④ `>=`　⑤ `<=`

タ の選択肢

⓪ `+`　① `-`　② `*`　③ `/`
④ `÷`　⑤ `%`　⑥ `**`

プログラム表記の例示

1. 変数

- 変数名は英字で始まる英数字と『_』の並び： **kingaku** や **goukei_kingaku**
- 配列を表す変数名は先頭文字を大文字： **H_ryokin[3]**

  ※ 特に説明がない場合，配列の要素を指定する添字は 0 から始まる。

2. 文字列

- 文字列はダブルクォーテーション『**"**』で囲む。
- **"**JOHO**"** や **"**ID_12345**"**

3. 代入文

- **kingaku = 300**
- **goukei_kingaku = tanka * kosu**
- **kamoku = "**JOHO**"**
- **H_ryokin = [100, 50, 200]**
- **moji =**【キーボードからの入力】

4. 算術演算

- 加減乗除の四則演算は，『**+**』，『**-**』，『*****』，『**/**』で表す。
- 整数の除算では，商（整数）を『**÷**』で，余りを『**%**』で表す。
- べき乗は『******』で表す。

5. 比較演算

- 『**==**』（等しい），『**!=**』（等しくない），『**>**』，『**<**』，『**>=**』，『**<=**』

  ※ 文字列の場合，『**==**』と『**!=**』で比較できる。

6. 論理演算

- 『**and**』（論理積），『**or**』（論理和），『**not**』（否定）

※ 4 ～ 6 の演算については，括弧の中を優先する。

7. 関数

- 値を返す関数： **ryokin =** 切り上げ**(ryokin * 0.8)**
- 値を返さない関数： 表示する**(ryokin)**

  表示する**(hinmei, "**の値段は**", nedan, "**です**")**

※「表示する」関数は，カンマ区切りで文字列や数値を連結できる。

- 値を返す関数を定義する場合は，戻り値を指定する： **ryokin** の値を返す

8. 制御文

※『｜』で制御範囲を表し，『└』は制御文の終わりを示す。

- 条件分岐

もし **x < 3** ならば：
｜ **x = x + 1**
└ **y = y + 1**

もし **x == 3** ならば：
｜ **x = x - 1**
そうでなければ：
└ **y = y * 2**

もし **x >= 3** ならば：
｜ **x = x - 1**
そうでなくもし **x < 0** ならば：
｜ **x = x * 2**
そうでなければ：
└ **y = y * 2**

- 繰返し

i を**0**から**9**まで１ずつ増やしながら繰り返す：
└ **goukei = goukei + H_ryokin[i]**

※「減らしながら」もある。

n < 10 の間繰り返す：
｜ **goukei = goukei + n**
└ **n = n + 1**