問３　クラスタ分析に用いる k-means 法に関する次の記述を読んで，設問 1～3 に答えよ。

k-means 法によるクラスタ分析は，異なる性質のものが混ざり合った母集団から互いに似た性質をもつものを集め，クラスタと呼ばれる互いに素な部分集合に分類する手法である。新聞記事のビッグデータ検索，店舗の品ぞろえの分類，教師なし機械学習などで利用されている。ここでは，2 次元データを扱うこととする。

〔分類方法と例〕

N 個の点を K 個（N 未満）のクラスタに分類する方法を(1)～(5)に示す。

(1)　N 個の点（1 から N までの番号が付いている）からランダムに K 個の点を選び（以下，初期設定という），それらの点をコアと呼ぶ。コアには 1 から K までのコア番号を付ける。なお，K 個のコアの座標は全て異なっていなければならない。

(2)　N 個の点と K 個のコアとの距離をそれぞれ計算し，各点から見て，距離が最も短いコア（複数存在する場合は，番号が最も小さいコア）を選ぶ。選ばれたコアのコア番号を，各点が所属する 1 回目のクラスタ番号（1 から K）とする。ここで，二つの点を XY 座標を用いて P=(a，b)と Q=(c，d)とした場合，P と Q の距離を $\sqrt{(a-c)^2+(b-d)^2}$ で計算する。

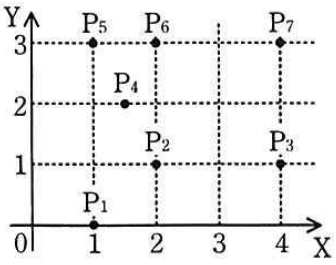
(3)　K 個のクラスタのそれぞれについて，クラスタに含まれる全ての点を使って重心を求める。ここで，重心の X 座標をクラスタに含まれる点の X 座標の平均，Y 座標をクラスタに含まれる点の Y 座標の平均と定義する。ここで求めた重心の番号はクラスタの番号と同じとする。

(4)　N 個の点と各クラスタの重心（G_1，…，G_K）との距離をそれぞれ計算し，各点から見て距離が最も短い重心（複数存在する場合は，番号が最も小さい重心）を選ぶ。選ばれた重心の番号を，各点が所属する次のクラスタ番号とする。

(5)　重心の座標が変わらなくなるまで，(3)と(4)を繰り返す。

次の座標で与えられる 7 個の点を，この分類方法に従い，二つのクラスタに分類する例を図 1 に示す。

$P_1=(1, 0)$　$P_2=(2, 1)$　$P_3=(4, 1)$　$P_4=(1.5, 2)$　$P_5=(1, 3)$　$P_6=(2, 3)$　$P_7=(4, 3)$



(1) 初期設定はコア 1＝P_2，コア 2＝P_5とする。
(2) 各点とコアの距離から表 1 のように所属クラスタを求める。
(3) クラスタの重心を計算し，クラスタ 1 の重心 G_1は（2.5，1.4），クラスタ 2 の重心 G_2は［　ア　］となる。
(4) 各点と重心の距離から表 2 のように所属クラスタを見直す。
(5) (3)と(4)を繰り返し，重心の座標が変わらなくなると①分類が完了する。

図 1　7 個の点の分類

表 1　コアとの距離と所属クラスタ

点	コア 1 との距離	コア 2 との距離	所属クラスタ番号
P_1	$\sqrt{2}$	3	1
P_2	0	$\sqrt{5}$	1
P_3	2	$\sqrt{13}$	1
P_4	$\sqrt{5}/2$	$\sqrt{5}/2$	1
P_5	$\sqrt{5}$	0	2
P_6	2	1	2
P_7	$2\sqrt{2}$	3	1

表 2　重心との距離と次の所属クラスタ

点	重心 G_1 との距離	重心 G_2 との距離	次の所属クラスタ番号
P_1	2.05	3.04	1
P_2	0.64	2.06	1
P_3	1.55	3.20	1
P_4	1.16	1.00	2
P_5	2.19	0.50	2
P_6	1.67	0.50	2
P_7	2.19	2.50	1

注記　距離は小数第 3 位以下切捨て

〔クラスタ分析のプログラム〕

この手法のプログラムを図 2 に，プログラムで使う主な変数，関数及び配列を表 3 に示す。ここで，配列の添字は全て 1 から始まり，要素の初期値は全て 0 とする。

表 3　クラスタ分析のプログラムで使う主な変数，関数及び配列

名称	種類	内容
core[t]	配列	コア番号が t である点 P_mの番号 m を格納する。
initial(K)	関数	初期設定として次の処理を行う。N 個の点 {P_1, P_2, …, P_N} からランダムに異なる K 個を抽出し，その番号を順に配列 core に格納する。
data_dist(s, t)	関数	点 P_sとコア番号が t である点の距離を返す。
grav_dist(s, t)	関数	点 P_sと重心 G_tの距離を返す。
data_length[t]	配列	ある点から見た，コア番号が t である点との距離を格納する。
grav_length[t]	配列	ある点から見た，重心 G_tとの距離を格納する。
min_index( ) 引数は data_length 又は grav_length	関数	配列の中で，最小値が格納されている添字 s を返す。最小値が二つ以上ある場合は，最も小さい添字を返す。 例　右の配列に対する戻り値は2 添字　1 2 3 4 5 配列　5 1 4 1 3
cluster[s]	配列	点 P_sが所属するクラスタ番号を格納する。
coordinate_x[s]	配列	重心 G_sの X 座標を格納する。

表３　クラスタ分析のプログラムで使う主な変数，関数及び配列（続き）

coordinate_y[s]	配列	重心 G_s の Y 座標を格納する。
gravity_x(s)	関数	クラスタ s の重心 G_s を求め，その X 座標を返す。
gravity_y(s)	関数	クラスタ s の重心 G_s を求め，その Y 座標を返す。
flag	変数	フラグ（値は 0 又は 1）

```
function clustering (N, K)                    //N はデータ数，K はクラスタ数
    MaxCount ← 100                            //無限ループを防ぐため最大見直し回数を設定
    initial(K)                                //初期設定
    for( s を 1 から N まで 1 ずつ増やす )     //クラスタを決定（1 回目）
        for( t を 1 から K まで 1 ずつ増やす )
            data_length[t] ← data_dist(s, core[t])
        endfor
        cluster[s] ← min_index(data_length)
    endfor
    for( p を 1 から MaxCount まで 1 ずつ増やす )     //クラスタを見直す
        if( p が 1 と等しい )
            for( [   イ   ] )                 //1 回目の重心の計算
                coordinate_x[t] ← gravity_x(t)
                coordinate_y[t] ← gravity_y(t)
            endfor
        else
            [   ウ   ]
            for( [   イ   ] )                 //2 回目以降の重心の計算
                if( gravity_x(t)と coordinate_x[t]が異なる     //重心を修正する
                    又は gravity_y(t)と coordinate_y[t]が異なる )
                      coordinate_x[t] ← gravity_x(t)
                      coordinate_y[t] ← gravity_y(t)
                      flag ← 1
                endif
            endfor
            if( [   エ   ] )                  //終了して抜ける
                return
            endif
        endif
        for( s を 1 から N まで 1 ずつ増やす )       //近い重心を見つける
            for( t を 1 から K まで 1 ずつ増やす )
                grav_length[t] ← grav_dist(s,t)
            endfor
            [          オ          ]
        endfor
    endfor
endfunction
```

図２　クラスタ分析の関数 clustering のプログラム

〔初期設定の改良〕

このアルゴリズムの最終結果は初期設定に依存し，そこでのコア間の距離が短いと適切な分類結果を得にくい。そこで，関数 initial において一つ目のコアはランダムに選び，それ以降はコア間の距離が長くなる点が選ばれやすくなるアルゴリズムを検討した。検討したアルゴリズムでは，t 番目までのコアが決まった後，t+1 番目のコアを残った点から選ぶときに，それまでに決まったコアから離れた点を，より高い確率で選ぶようにする。具体的には，それまでに決まったコア（コア 1～コア t）と，残った N−t 個の点から選んだ点 P_s との距離の和を T_s とする。N−t 個の全ての点について T_s を求め，T_s が［　カ　］ほど高い確率で点 P_s が選ばれるようにする。このとき，点 P_s が t+1 番目のコアとして選ばれる確率として［　キ　］を適用する。

設問１　〔分類方法と例〕について，(1)，(2)に答えよ。

(1)　図１中の［　ア　］に入れる座標を答えよ。

(2)　図 1 中の下線①のように分類が完了したときに，P_1 と同じクラスタに入る点を全て答えよ。

設問２　図２中の［　イ　］～［　オ　］に入れる適切な字句を答えよ。

設問３　〔初期設定の改良〕について，(1)，(2)に答えよ。

(1)　本文中の［　カ　］に入れる適切な字句を解答群の中から選び，記号で答えよ。

解答群

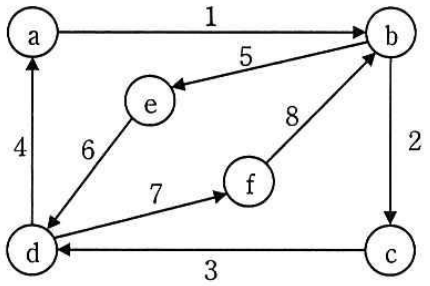
ア　大きい　　イ　小さい

(2)　本文中の［　キ　］に入れる適切な式を T_s と Sum を使って答えよ。

ここで，Sum は N−t 個の全ての T_s の和とする。

問３　一筆書きに関する次の記述を読んで，設問１～４に答えよ。

グラフは，有限個の点の集合と，その中の２点を結ぶ辺の集合から成る数理モデルである。グラフの点と点の間をつなぐ辺の列のことを経路という。本問では，任意の２点間で，辺をたどることで互いに行き来することができる経路が存在する（以下，強連結という）有向グラフを扱う。強連結な有向グラフの例を図１に示す。辺は始点と終点の組で定義する。各辺には１から始まる番号が付けられている。



辺1＝(点a，点b)　辺2＝(点b，点c)
辺3＝(点c，点d)　辺4＝(点d，点a)
辺5＝(点b，点e)　辺6＝(点e，点d)
辺7＝(点d，点f)　辺8＝(点f，点b)

凡例　辺＝(辺の始点，辺の終点)

図１　強連結な有向グラフの例

〔一筆書き〕

本問では，グラフの全ての辺を１回だけ通り，出発点から出て出発点に戻る閉じた経路をもつグラフを，一筆書きができるグラフとする。

〔一筆書きの経路の求め方〕

一筆書きの経路を求めるためには，出発点から辺の向きに従って辺を順番にたどり，出発点に戻る経路を見つける探索を行う。たどった経路（以下，探索済の経路という）について，グラフ全体で通過していない辺（以下，未探索の辺という）がない場合は，この経路が一筆書きの経路となる。未探索の辺が残っている場合は，探索済の経路を，未探索の辺が接続する点まで遡り，その点を出発点として，同じ点に戻る経路を見つけて，遡る前までの経路に連結することを繰り返す。

各点を始点とする辺を接続辺という。グラフの各点に対して接続辺の集合が決まり，辺の番号が一番小さい接続辺を最初の接続辺という。同じ始点をもつ接続辺の集合で，辺の番号を小さいものから順番に並べたときに，辺の番号が次に大きい接続辺を次の接続辺ということにする。

図１のグラフの各点の接続辺の集合を表１に示す。図１において，点ｂの最初の接

続辺は辺２である。辺２の次の接続辺は辺５となる。辺５の次の接続辺はない。

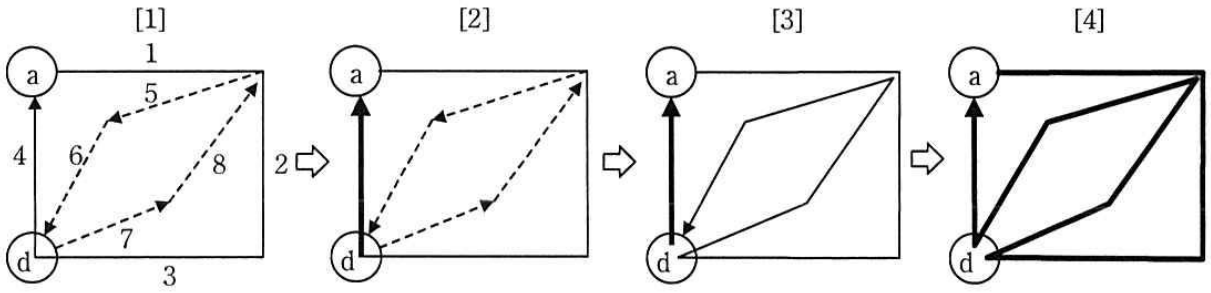
表１　図１のグラフの各点の接続辺の集合

点	接続辺の集合
点ａ	辺１
点ｂ	辺２，辺５
点ｃ	辺３
点ｄ	辺４，辺７
点ｅ	辺６
点ｆ	辺８

一筆書きの経路の探索において，一つの点に複数の接続辺がある場合には，最初の接続辺から順にたどることにする。

図１のグラフで点ａを出発点とした一筆書きの経路の求め方を図２に示す。

経路を構成する辺とその順番が，これ以上変わらない場合，確定済の経路という。



凡例　——：探索済　　----：未探索　　━━：確定済

注記１　点や辺の番号を一部省略している。

注記２　矢印は経路の向きを示す。

図２　図１のグラフで点ａを出発点とした一筆書きの経路の求め方

図２を参考にした一筆書きの経路を求める手順を次に示す。

〔一筆書きの経路を求める手順〕

点ａから探索する場合は，点ａの最初の接続辺である辺１から始め，辺１の終点ｂの最初の接続辺である辺２をたどり，同様に辺３，辺４をたどる。辺４の終点ａからたどれる未探索の辺は存在しないので，これ以上探索が進められない（図２[1]）。

しかし，未探索の辺５，辺６，辺７，辺８が残っているので，未探索の辺が接続する点まで遡る。

終点 a から辺 4 を遡ると，辺 4 の始点 d で未探索の辺 7 が接続している。遡った経路は途中で未探索の辺が存在しないので，これ以上，辺の順番が変わらず，辺 4 は，一筆書きの経路の一部として確定済の経路となる（図 2 [2]）。

点 d から同様に辺 7→辺 8→辺 5→辺 6 と探索できるので，辺 3 までの経路と連結した新しい探索済の経路ができる（図 2 [3]）。

辺 6 の終点 d からは，辺 6→辺 5→辺 8→辺 7→辺 3→辺 2→辺 1 と出発点の点 a まで遡り，これ以上，未探索の辺がないことが分かるので，全ての辺が確定済の経路になる（図 2 [4]）。

一筆書きの経路は，次の(1)～(4)の手順で求められる。

(1) 一筆書きの経路の出発点を決める。

(2) 出発点から，未探索の辺が存在する限り，その辺をたどり，たどった経路を探索済の経路に追加する。

(3) 探索済の経路を未探索の辺が接続する点又は一筆書きの経路の出発点まで遡る。遡った経路は，探索済の経路から確定済の経路にする。未探索の辺が接続する点がある場合は，それを新たな出発点として，(2)に戻って新たな経路を見つける。

(4) 全ての辺が確定済の経路になった時点で探索が完了して，その確定済の経路が一筆書きの経路になる。

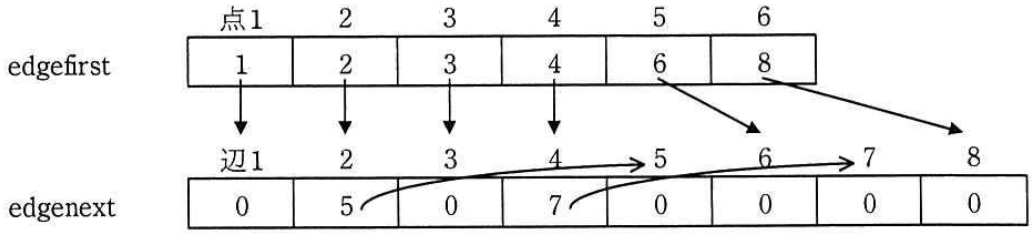
〔一筆書きの経路を求めるプログラム〕

一筆書きの経路を求める関数 directedE のプログラムを作成した。

実装に当たって，各点を点 n（n は 1～N）と記す。例えば，図 1 のグラフでは，点 a は点 1，点 b は点 2 と記す。

グラフの探索のために，あらかじめ，グラフの点に対する最初の接続辺の配列 edgefirst 及び接続辺に対する次の接続辺の配列 edgenext を用意しておく。edgenext において，次の接続辺がない場合は，要素に 0 を格納する。

図 1 のグラフの場合の配列 edgefirst，edgenext を図 3 に示す。



注記　edgefirst にはグラフの点に対する最初の接続辺の番号を格納している。
edgenext には接続辺の次の接続辺の番号を格納している。

図3　図1のグラフの場合の配列 edgefirst，edgenext

edgefirst によって点2の最初の接続辺が辺2であることが分かり，点2から最初にたどる接続辺は辺 2 となる。edgenext によって，辺 2 の次の接続辺が辺 5 であることが分かるので，点2から次にたどる接続辺は辺5となる。辺5の次の接続辺はないので，点2からたどる接続辺はこれ以上ないことが分かる。

プログラム中で使用する定数と配列を表 2 に，作成した関数 directedE のプログラムを図4に示す。

全ての配列の添字は1から始まる。

表2　使用する定数と配列

名称	種類	内容
N	定数	グラフの点の個数
M	定数	グラフの辺の個数
start[m]	配列	start[m] には，辺 m の始点の番号が格納されている。
end[m]	配列	end[m] には，辺 m の終点の番号が格納されている。
edgefirst[n]	配列	edgefirst[n] には，点 n の最初の接続辺の番号が格納されている。
edgenext[m]	配列	edgenext[m] には，辺 m の次の接続辺の番号が格納されている。次の接続辺がない場合は 0 が格納されている。
current[n]	配列	current[n] には，点 n を始点とする未探索の辺の中で最小の番号を格納する。点 n を始点とする未探索の辺がない場合は 0 を格納する。
searched[m]	配列	一筆書きの経路を構成する探索済の辺の番号を順番に格納する。（探索済の経路）
path[m]	配列	一筆書きの経路を構成する確定済の辺の番号を順番に格納する。（確定済の経路）

```
function directedE()
    for ( i を 1 から N まで 1 ずつ増やす ) // 各点での未探索の辺の番号を初期化
        current[i] ← edgefirst[i]
    endfor
    top ← 1                              // 探索済の経路の辺の格納位置を初期化
    last ← M                             // 確定済の経路の辺の格納位置を初期化
    x ← 1                                // 出発点は点 1
    while ( ①last が 1 以上 )
        if ( current[x] が [ ア ] でない )
            temp ← current[x]            // 点 x からたどる接続辺は temp
            searched[top] ← temp         // 接続辺 temp を探索済の経路に登録
            current[x] ← [ イ ]          // 点 x から次にたどる未探索の辺を格納
            x ← end[temp]                // 接続辺 temp の終点を点 x にする
            top ← top + 1
        else
            top ← [ ウ ]                 // 探索済の辺を遡る
            temp ← searched[top]         // 遡った辺は temp
            path[last] ← temp            // 辺 temp を確定済にする
            x ← [ エ ]
            last ← last − 1
        endif
    endwhile
endfunction
```

図 4　関数 directedE のプログラム

設問 1　図 4 中の［　ア　］～［　エ　］に入れる適切な字句を答えよ。

設問 2　図 1 のグラフで関数 directedE を動作させたとき，while 文中の if 文は，何回実行されるか，数値で答えよ。

設問 3　一筆書きができない強連結な有向グラフで関数 directedE を動作させたとき，探索はどのようになるかを，解答群の中から選び，記号で答えよ。

解答群

ア　探索が完了するが，配列 path に格納された経路は一筆書きの経路にならない。

イ　探索が完了せずに終了して，配列 path に格納された経路は一筆書きの経路にならない。

ウ　探索が無限ループに陥り，探索が終了しない。

設問 4　図 4 のプログラムは，配列 searched を配列 path に置き換えることで，使用する領域を減らすことができる。このとき，無駄な繰返しが発生しないように，下線①の繰返し条件を，変数 top と last を用いて変更せよ。

問3　パズルの解答を求めるプログラムに関する次の記述を読んで，設問 1～3 に答えよ。

太線で 3×3 の枠に区切られた 9×9 のマスから成る正方形の盤面に，1～9 の数字を入れるパズルの解答を求めるプログラムを考える。このパズルは，図 1 に示すように幾つかのマスに数字が入れられている状態から，数字の入っていない各マスに，1～9 のうちのどれか一つの数字を入れていく。このとき，盤面の横 1 行，縦 1 列，及び太線で囲まれた 3×3 の枠内の全てにおいて，1～9 の数字が一つずつ入ることが，このパズルのルールである。パズルの問題例を図 1 に，図 1 の解答を図 2 に示す。

2		1		9		7		
	4		2			3		
5					8		2	9
	9		6	7		2		
6			3		5			4
		7		4	9		1	
7	6		9					3
		9			6		4	
		4		1		6		

図 1　問題例

2	8	1	4	9	3	7	6	5
9	4	6	2	5	7	3	8	1
5	7	3	1	6	8	4	2	9
4	9	5	6	7	1	2	3	8
6	1	8	3	2	5	9	7	4
3	2	7	8	4	9	5	1	6
7	6	2	9	8	4	1	5	3
1	5	9	7	3	6	8	4	2
8	3	4	5	1	2	6	9	7

図 2　図 1 の解答

このパズルを解くための方針を次に示す。

方針：数字が入っていない空白のマスに，1～9 の数字を入れて，パズルのルールにのっとって全部のマスを埋めることができる解答を探索する。

この方針に沿ってパズルを解く手順を考える。

〔パズルを解く手順〕

(1)　盤面の左上端から探索を開始する。マスは左端から順に右方向に探索し，右端に達したら一行下がり，左端から順に探索する。

(2)　空白のマスを見つける。

(3)　(2)で見つけた空白のマスに，1～9 の数字を順番に入れる。

(4)　数字を入れたときに，その状態がパズルのルールにのっとっているかどうかをチェックする。

(4-1) ルールにのっとっている場合は，(2)に進んで次の空白のマスを見つける。

(4-2) ルールにのっとっていない場合は，(3)に戻って次の数字を入れる。このとき，入れる数字がない場合には，マスを空白に戻して一つ前に数字を入れたマスに戻り，(3)から再開する。

(5) 最後のマスまで数字が入り，空白のマスがなくなったら，それが解答となる。

〔盤面の表現〕

この手順をプログラムに実装するために，9×9 の盤面を次のデータ構造で表現することにした。

・9×9 の盤面を 81 個の要素をもつ 1 次元配列 board で表現する。添字は 0 から始まる。各要素にはマスに入れられた数字が格納され，空白の場合は 0 を格納する。

配列 board による盤面の表現を図 3 に示す。ここで括弧内の数字は配列 board の添字を表す。

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]
[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]
[27]	[28]	[29]	[30]	[31]	[32]	[33]	[34]	[35]
[72]	[73]	[74]	[75]	[76]	[77]	[78]	[79]	[80]

図 3　配列 board による盤面の表現

〔ルールのチェック方法〕

パズルのルールにのっとっているかどうかのチェックでは，数字を入れたマスが含まれる横 1 行の左端のマス，縦 1 列の上端のマス，3×3 の枠内の左上端のマスを特定し，行，列，枠内のマスに既に格納されている数字と，入れた数字がそれぞれ重複していないことを確認する。このチェックを“重複チェック”という。

〔解法のプログラム〕

プログラムで使用する配列，関数，変数及び定数の一部を表 1 に示す。なお，表 1 の配列及び変数は大域変数とする。

表１　プログラムで使用する配列，関数，変数及び定数の一部

名称	種類	内容
board[ ]	配列	盤面の情報を格納する配列。 初期化時には問題に合わせて要素に数字が設定される。
solve(x)	関数	パズルを解くための手順を実行する関数。 盤面を表す board[ ]の添字 x を引数とする。
row_ok(n, x)	関数	横 1 行の重複チェックを行う関数。チェック対象の数字 n，チェック対象のマスを示す添字 x を引数とする。 数字の重複がない場合は true，重複がある場合は false を返す。
column_ok(n, x)	関数	縦 1 列の重複チェックを行う関数。チェック対象の数字 n，チェック対象のマスを示す添字 x を引数とする。 数字の重複がない場合は true，重複がある場合は false を返す。
frame_ok(n, x)	関数	3×3 の枠内の重複チェックを行う関数。チェック対象の数字 n，チェック対象のマスを示す添字 x を引数とする。 数字の重複がない場合は true，重複がある場合は false を返す。
check_ok(n, x)	関数	row_ok，column_ok，frame_ok を呼び出し，全ての重複チェックを実行する関数。チェック対象の数字 n，チェック対象のマスを示す添字 x を引数とする。 全てのチェックで数字の重複がない場合は true，一つ以上のチェックで数字の重複がある場合は false を返す。
div(n, m)	関数	整数 n を整数 m で割った商を求める関数。
mod(n, m)	関数	整数 n を整数 m で割った剰余を求める関数。
print_board( )	関数	board[ ]の内容を 9×9 の形に出力する関数。
row_top	変数	数字を入れようとするマスが含まれる横 1 行の左端のマスを示す添字を格納する変数。
column_top	変数	数字を入れようとするマスが含まれる縦 1 列の上端のマスを示す添字を格納する変数。
frame_top	変数	数字を入れようとするマスが含まれる 3×3 の枠内の左上端のマスを示す添字を格納する変数。
MAX_BOARD	定数	盤面に含まれるマスの数を表す定数で 81。

解法のプログラムのメインプログラムを図 4 に，関数 solve のプログラムを図 5 に，重複チェックを行うプログラムの一部を図 6 に示す。

```
function main()
  board[]を初期化する    //問題を盤面に設定する
  solve(0)               //盤面の左上端のマスを示す添字を引数として関数 solve を呼び出す
endfunction
```

図４　メインプログラム

```
function solve(x)
  if (x が MAX_BOARD−1 より大きい)
    print_board()                          //解答を出力する
    exit()                                 //メインプログラムの処理を終了する
  else
    if ( [ア] )                            //対象のマスが空白でない場合
      solve ( [イ] )                       //次の探索
    else
      for (n を 1 から 9 まで 1 ずつ増やす)  //1〜9 の数字を順にマスに入れる
        if ( [ウ] )
          board[x] ← n
          solve ( [イ] )                   //次の探索
          board[x] ← [エ]                  //再帰から戻った場合のマスの初期化
        endif
      endfor
    endif
  endif
endfunction
```

図 5　関数 solve のプログラム

```
function row_ok(n, x)                      //横 1 行の重複チェック
  row_top ← [オ]                           //行の左端のマスを示す添字を求める
  for (i を 0 から 8 まで 1 ずつ増やす)
    if ( [カ] )
      return false
    endif
  endfor
  return true
endfunction

function column_ok(n, x)                   //縦 1 列の重複チェック
  column_top ← [キ]                        //列の上端のマスを示す添字を求める
  for (i を 0 から 8 まで 1 ずつ増やす)
    if ( [ク] )
      return false
    endif
  endfor
  return true
endfunction

function frame_ok(n, x)                    //3×3 の枠内の重複チェック
  frame_top ← x − [ケ] − mod(x, 3)         //枠内の左上端のマスを示す添字を求める
  for (i を 0 から 2 まで 1 ずつ増やす)
    for (j を 0 から 2 まで 1 ずつ増やす)
      if (board[frame_top + 9 * i + j]が n と等しい)
        return false
      endif
    endfor
  endfor
  return true
endfunction
```

図 6　重複チェックを行うプログラムの一部

〔プログラムの改善〕

解法のプログラムは深さ優先探索であり，探索の範囲が広くなるほど，再帰呼出しの回数が指数関数的に増加し，重複チェックの実行回数も増加する。

そこで，重複チェックの実行回数を少なくするために，各マスに入れることができる数字を保持するためのデータ構造 Z を考える。データ構造 Z は盤面のマスの数×9の要素をもち，添字 x は 0 から，添字 n は 1 から始まる 2 次元配列とする。Z[x][n]は，ゲームのルールにのっとって board[x]に数字 n を入れることができる場合は要素に 1 を，できない場合は要素に 0 を格納する。データ構造 Z の初期化処理と更新処理を表 2 のように定義した。

なお，データ構造 Z は大域変数として導入する。

表 2　データ構造 Z の初期化処理と更新処理

処理の名称	処理の内容
初期化処理	初期化時の盤面に対し，個々の空白のマスについて 1～9 の数字を入れた場合の重複チェックを行う。 重複チェックの結果によって，初期化時の盤面の状態で個々の空白のマスに入れることができない数字は，データ構造 Z の該当する数字の要素に 0 を設定する。それ以外の要素には 1 を設定する。
更新処理	空白のマスに数字を入れたとき，そのマスが含まれる横 1 行，縦 1 列，3×3 の枠内の全てのマスを対象に，データ構造 Z の該当する数字の要素を 0 に更新する。

〔パズルを解く手順〕の(1)の前にデータ構造 Z の初期化処理を追加し，〔パズルを解く手順〕の(2)～(5)を次の(2)～(4)のように変更した。

(2)　空白のマスを見つける。

(3)　データ構造 Z を参照し，(2)で見つけた空白のマスに入れることができる数字のリストを取得し，リストの数字を順番に入れる。

(3-1) 入れる数字がある場合，<u>①処理 A</u> を行った後，マスに数字を入れる。その後，データ構造 Z の更新処理を行い，(2)に進んで次の空白のマスを見つける。

(3-2) 入れる数字がない場合，マスを空白に戻し，<u>②処理 B</u> を行った後，一つ前に数字を入れたマスに戻り，戻ったマスで取得したリストの次の数字から再開する。

(4)　最後のマスまで数字が入り，空白のマスがなくなったら，それが解答となる。

設問1　図5中の［ア］～［エ］に入れる適切な字句を答えよ。

設問2　図6中の［オ］～［ケ］に入れる適切な字句を答えよ。

設問3　〔プログラムの改善〕について，下線①の処理A及び下線②の処理Bの内容を，“データ構造Z”という字句を含めて，それぞれ20字以内で述べよ。