

専門(記述式)試験問題

注 意 事 項

1. 問題は **6 題** あります。このうち **任意の 2 題** を選んで解答してください。なお、この問題集の **裏表紙** に科目別構成の詳細が記載されていますので、解答開始前によく読んでおいてください。
2. 解答時間は **3 時間** です。
3. 答案用紙の記入について
 - (ア) 答案は濃くはっきり書き、書き損じた場合は、解答の内容がはっきり分かるように訂正してください。
 - (イ) 問題 **1 題に 1 枚** (両面) を使用してください。
 - (ウ) 表側の各欄にそれぞれ必要事項を記入してください。
問題番号欄には、解答した問題の番号をそれぞれ記入してください。
 - (エ) 試験の公正を害するおそれがありますので、答案用紙の氏名欄以外に氏名その他解答と関係のない事項を記載しないでください。
4. 下書き用紙はこの問題集の **中央部** にとじ込んであります。**試験官の指示に従って、試験開始後に問題集から下書き用紙だけを慎重に引きはがして** 使用してください。なお、誤って問題集を破損しても、問題集の交換はできませんので注意してください。
5. この問題集で単位の明示されていない量については、全て国際単位系 (SI) を用いることとします。
6. この問題集は、本試験種目終了後に持ち帰りができます。
7. 本試験種目の途中で退室する場合は、退室時の問題集及び下書き用紙の持ち帰りはできませんが、希望する方には後ほど渡します。別途試験官の指示に従ってください。なお、試験時間中に、この問題集から **下書き用紙以外** を切り取ったり、問題を転記したりしないでください。
8. 下欄に受験番号等を記入してください。

第 1 次試験地	試験の区分	受験番号	氏 名
	デジタル		

指示があるまで中を開いてはいけません。

【No. 1】 閉路をもたないグラフを森と呼び、連結な森を木と呼ぶ。頂点集合を V 、辺集合を E とするグラフ $G = (V, E)$ に対して、頂点集合が V であるような G の部分グラフを、 G の全域部分グラフと呼ぶ。特に、辺数が k ($0 \leq k \leq |V| - 1$) であるような全域部分グラフで森であるようなものを k -全域森、木であるようなものを全域木と呼ぶ。また、 G の連結な部分グラフで、その頂点集合が極大であるようなものを G の連結成分と呼ぶ。グラフ G の連結成分の個数を $c(G)$ で表すものとする。

なお、本問で取り扱うグラフは全て単純な無向グラフとし、頂点数 n 、辺数 $n - 1$ の森は木であるという事実を用いてよいものとする。以下の設問に答えよ。

(1) 以下の問いに答えよ。

- (a) 頂点数 6、辺数 4 の森を一つ図示せよ。
- (b) 頂点数 5 の木で、最大次数が最も小さいもの及び最も大きいものをそれぞれ図示せよ。
- (c) 頂点数 90、辺数 45 のグラフのうち、連結成分の個数が最大となるようなものの一つを G とする。 G の連結成分数を求めよ。

(2) 頂点数 n の $(n - k)$ -全域森 G に対して、 $c(G) = k$ が成り立つことを、 k についての帰納法により証明せよ。なお、証明は 10 行以内で解答すること。

(3) $G = (V, E)$ を無向グラフとして、 $w: E \rightarrow \mathbb{N}$ を G の辺に対する重み関数とする。 G の辺数 k の全域森のうち、そこに含まれる辺の重みの総和が最小となるようなものを G の最小重み k -全域森と呼ぶ。また、特に最小重み $(|V| - 1)$ -全域森は最小重み全域木と呼ぶ。 (G, w) に対して最小重み全域木を計算する方法として、クラスカル法がよく知られている。次にクラスカル法の概要を示す。以下の問いに答えよ。

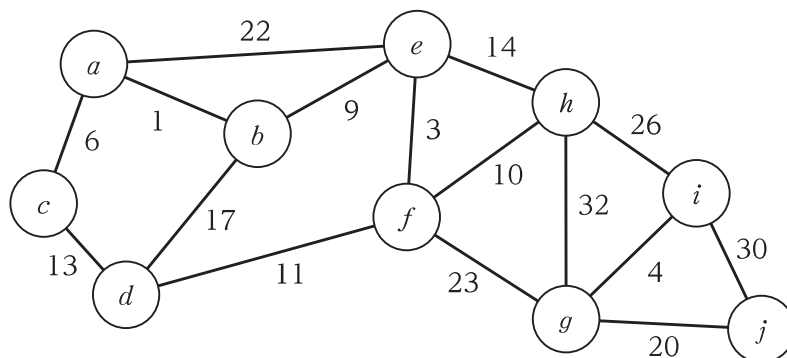
クラスカル法

```

1:  $e_1, e_2, \dots, e_{|E|}$  を  $E$  中の辺を重みの昇順で整列した列とする
2:  $V_T = V; E_T = \{\}$  とする
3: for  $i = 1$  to  $|E|$  do
4:   if  $e_i$  を  $E_T$  に加えてもグラフ  $(V_T, E_T)$  に閉路が生じない then
5:      $E_T \leftarrow E_T \cup \{e_i\}$ 
6:   end if
7: end for
8:  $(V_T, E_T)$  を出力する

```

- (a) 次の図において与えられる重み付きグラフを入力 G として、クラスカル法を実行することを考える。3～7行目の for ループについて、10 周目のループ開始時における (V_T, E_T) を図示せよ。また、 (V_T, E_T) が最小重み全域木となるのは何周目のループ終了時か示せ。



- (b) 次の文章はクラスカル法の正当性証明の概略である。文中の下線部(ア)～(エ)の主張が成立する理由をそれぞれ5行以内で説明せよ。ただし、(2)で証明する命題が成立することを用いてよいものとする。

アルゴリズムが全域木を返すことは明らかであるので、返される辺の重みの総和が最小全域木の重みと一致することを示せばよい。アルゴリズムの5行目において、 k 本目の辺が追加された直後の (V_T, E_T) の内容を $T_k = (V, F_k)$ で表すとする。証明は、任意の $k (0 \leq k \leq |V| - 1)$ について、 (V, F_k) が G の最小重み k -全域森となることを、 k についての帰納法により示す。(ア)この事実が示されれば、アルゴリズムが出力する結果は明らかに最小重み全域木となる。

$k = 0$ のときは自明に成立するので、ある k について (V, F_k) が最小重み k -全域森であると仮定して、 (V, F_{k+1}) が最小重み $(k + 1)$ -全域森となることを示す。 $T^* = (V, F_{k+1}^*)$ を G における任意の最小重み $(k + 1)$ -全域森とし、 F_{k+1} には含まれるが F_k には含まれない辺を e とする。もし、 T_{k+1} が最小重み $(k + 1)$ -全域森でないと仮定すると次の式が成立する。

$$\sum_{f \in F_{k+1}^*} w(f) < w(e) + \sum_{f \in F_k} w(f)$$

このとき、帰納法の仮定に着目すると、(イ)任意の $e' \in F_{k+1}^*$ について、 $w(e') < w(e)$ が成立する。また、(ウ) $c(T^*) < c(T_k)$ が成り立つので、 $(エ)F_{k+1}^*$ の中に含まれる辺で、 F_k へと追加しても閉路が生じないような辺 $e'' \in F_{k+1}^*$ が必ず存在する。しかし、このときクラスカルのアルゴリズムは、 e の代わりに e'' を選ばなければならず、アルゴリズムの動作に矛盾するため、命題が示される。

- (c) 重み付き無向グラフ G の全域木のうち、そこに含まれる辺の重みの最大値が最小となるようなものを、 G の最小最大重み全域木と呼ぶ。クラスカル法を参考にして、最小最大重み全域木を計算する多項式時間アルゴリズム(最悪時実行時間が $|V|$ 及び $|E|$ の多項式で抑えられるアルゴリズム)を構成せよ。解答は10行程度で説明し、構成したアルゴリズムが正しく動作する理由も含めること。

【No. 2】 以下の I、II、III の設問に答えよ。

なお、問題中の RISC-V アセンブリコード内で使用されているレジスタはニーモニック表記となっており、レジスタ名との対応は次の表のとおりである。

ニーモニック (ABI 名)	レジスタ名
ra	x1
sp	x2
t0	x5
a0-7	x10-17

また、一部の命令は疑似命令であり、実際に実行される命令との対応は次の表のとおりである。
ただし、x0 は hard-wired zero である。

疑似命令	実際の命令
mv a0, a1	addi x10, x11, 0
ret	jalr x0, x1, 0
li t0, 2	addi x5, x0, 2
call offset	jal ra, offset

I. コード①のような C 言語コードをコンパイルした結果、コード②のような 32 ビット RISC-V アセンブリコードが得られたとする。以下の問いに答えよ。

コード①

```
1  int bar(int a, int b) {  
2      return a + b;  
3  }  
4  int foo(int a, int b, int c) {  
5      int result = a + bar(b, c);  
6      return result;  
7  }
```

コード②

```
1  bar:
2      add  ㊦, a0, a1  # 計算結果を戻り値用のレジスタへ
3      ret
4
5  foo:
6      addi  sp, sp, -16
7      sd    ㊧, 8(sp)  # 呼び出し元に戻るために必要な値を保存
8      sd    ㊨, 0(sp)  # 後で使うため、上書きされる前に値を保存
9
10     mv    a0, a1
11     mv    a1, a2
12     call  bar                # bar 関数を呼び出す
13
14     ld     t0, 0(sp)         # 8 行目で保存した値を t0 に復元
15     ld     ra, 8(sp)        # 7 行目で保存した値を ra に復元
16     addi  sp, sp, 16
17
18     add   a0, t0, a0         # 最終結果を計算し、戻り値用のレジスタへ
19     ret
```

(1) ㉗、㉘、㉙に当てはまる最も妥当なレジスタ名をそれぞれ示せ。

(2) foo 関数ではスタックを利用してレジスタの値を保存する必要があるが、bar 関数ではその必要がない。その理由を次の語句を全て用いて 3 行程度で説明せよ。ただし、解答中の用いた語句に下線を引くこと。

[語句：call 命令、ra レジスタ]

(3) RISC-V の標準的な呼び出し規約では、関数の引数は a0~a7 の八つのレジスタを使って渡される。では、次のような C 言語で記述された 9 個目の引数をもつ関数を呼び出す場合、呼び出し側(caller)と、呼び出される sum9 関数側(callee)は、9 番目の引数 i をどのようにやり取りするかを、次の語句を全て用いて 3 行程度で説明せよ。ただし、解答中の用いた語句に下線を引くこと。

[語句：スタック、スタックポインタ(sp)]

```
1  int sum9(int a, int b, int c, int d, int e, int f, int g, int h, int i) {  
2      return a + b + c + d + e + f + g + h + i;  
3  }
```

II. コード③に示した C 言語コードをコンパイルした結果、コード④のような 32 ビット RISC-V アセンブリコードが得られたとする。以下の設問に答えよ。

コード③

```
1  int fib(int n) {  
2      if (n < 2) return n;  
3      return fib(n - 1) + fib(n - 2);  
4  }
```

コード④

```

1  fib:
2      li    t0, 2
3      bge   a0, t0, recursive_step
4      ret
5
6  recursive_step:
7      addi   sp, sp, -24
8      sd     ㉞, 16(sp) # 戻りアドレスを保存
9      sd     ㉟, 8(sp)  # 元の引数 n を保存
10
11     addi   a0, a0, -1
12     call   fib          # fib (n-1) を呼び出す
13
14     sd     ㊱, 0(sp)    # fib (n-1) の計算結果を一時的に保存
15
16     ld     a0, 8(sp)    # 9 行目で保存した元の引数 n を復元
17     addi   a0, a0, -2
18     call   fib          # fib (n-2) を呼び出す
19
20     ld     t0, 0(sp)    # 14 行目で保存した fib (n-1) の結果を復元
21     add    a0, a0, t0
22
23     ld     ra, 16(sp)   # 8 行目で保存した戻りアドレスを復元
24     addi   sp, sp, 24
25     ret

```

(1) ㉞、㉟、㊱に当てはまる最も妥当なレジスタ名をそれぞれ示せ。

(2) fib(3) 関数を呼び出した際、すなわち引数を 3 として fib 関数を呼び出した際にスタックが最も深くなった時点でのスタックの内容を図で示し、スタックに格納されているそれぞれの値が何を意味するのか説明せよ。

- (3) fib 関数と以下の fac 関数は、どちらも再帰を用いるため、関数の処理中に自身の呼び出し元に戻るための戻りアドレス(ra)と、計算の途中で必要になる元の引数(n)をスタックに保存するという共通点がある。しかし、fib 関数では、それらに加えてもう一つ値をスタックに一時保存する必要があるが、その値とは何か示せ。また、fac 関数ではその必要がなく、fib 関数ではその必要がある理由を、次の C コードの return 文に注目し、戻り値を計算するために必要な関数自身の呼び出し回数という点における構造の違いの観点から 5 行程度で説明せよ。

```
1  int fac(int n) {  
2      if (n == 0) {  
3          return 1;  
4      } else {  
5          return n * fac(n - 1);  
6      }  
7  }
```

Ⅲ. 次の 5 ステージから成る、一般的な RISC-V の命令パイプラインを考える。

IF：命令フェッチ (Instruction Fetch)

ID：命令デコード、レジスタ読み出し (Instruction Decode)

EX：演算実行 (Execute)

MEM：メモリアクセス (Memory Access)

WB：レジスタへの書き込み (Write Back)

このパイプラインには、次の特徴があるとする。

- ・フォワーディング(バイパス)機能が有効であり、多くのデータハザードはストールなしで解決される。
- ・フォワーディングをもってしても解決できないハザードでは、パイプラインがストールする。
- ・分岐命令の分岐先アドレスは ID ステージで確定し、分岐するかどうかの判断は EX ステージで完了する。

II のコード④について、以下の問いに答えよ。

- (1) 20 行目の ld 命令と 21 行目の add 命令の間で、ある実装では、データハザードによるストールが 1 サイクル発生したとする。前提条件で述べたフォワーディング機能が有効であるにもかかわらず、ストールが発生してしまう理由を「フォワーディング」という語句とパイプラインステージ名を用いて 6 行程度で説明せよ。
ただし、メモリアクセスはキャッシュにヒットしたときを考え、十分高速(又はノーウェイト)にアクセスできるものとする。
- (2) 23 行目の ld 命令と 25 行目の ret 命令の間でも、ストールが発生する。25 行目の ret 命令は jalr 命令の疑似命令であり、ra レジスタの値を次のプログラムカウンタとして利用する。これがどのようなハザード(データハザード、制御ハザードなど)に分類され、なぜストールを引き起こすのか 6 行程度で説明せよ。
- (3) 先に示した前提条件のパイプラインにおいて、3 行目の bge 命令で分岐が成立した場合(すなわちラベル recursive_step へジャンプした場合)、一般に「制御ハザード」によるペナルティ(ストールの発生や命令の破棄)が生じる。この理由を、パイプラインの IF ステージと EX ステージの動作と関連付けて 6 行程度で説明せよ。

【No. 3】 任意の論理回路を 1 種類の 2 入力論理ゲート (NAND ゲート) のみで表現することを考える。本問において、H は高電位、L は低電位 (接地電位) を表す。信号がゲート及び回路を通過することによる信号の遅延は無視できるものとする。また、論理式においては、 a と b を入力とする NAND ゲートを (a, b) と表現する。なお、解答においてもこの表現を用いること。

ゲート記号は、NAND ゲートを図 I (a)、XOR ゲートを図 I (b)、NOT ゲートを図 I (c) のように描くものとする。以下の設問に答えよ。

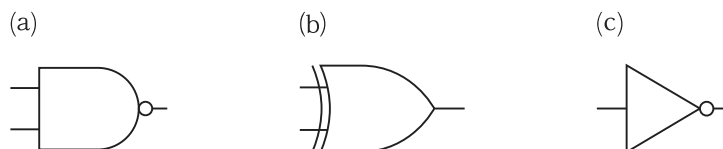


図 I

(1) 以下の問いに答えよ。

- (a) NOT ゲート、AND ゲート、OR ゲートのそれぞれと等価な回路を、NAND ゲートのみを用いて構成する場合に必要な NAND ゲートの数を示せ。
- (b) a , b を入力、 z を出力とする、図 II のような回路の働きを表した論理式を、積和形の最簡形で示せ。

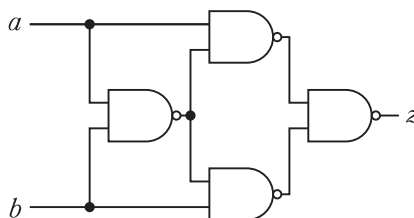


図 II

(2) 以下の問いに答えよ。

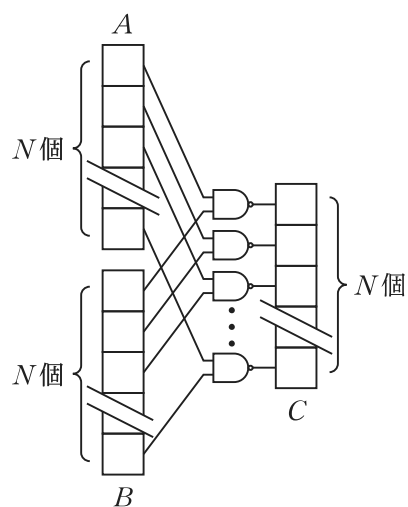
- (a) A_1 から A_N までの N 個の信号源が並んだセット A と、 B_1 から B_N までの N 個の信号源が並んだセット B があるとする。全ての $i (i = 1, 2, \dots, N)$ において、 A_i と B_i が一致したときのみ H を出力する等値比較器を NAND ゲートのみを使って実装するにはどのように構成すればよいか、その実現方法を示せ。ただし、 N は偶数とする。また、信号源は L, H の 2 状態のみをとるものとする。
- (b) (a) で構成した実装において使用する NAND ゲートの数を、 N を用いて示せ。
ただし、導出過程も示すこと。

(3) 入力 a , b をトリガー信号 s に基づいて選択する 2 : 1 マルチプレクサの回路を描け。

ただし、 s が H のときに a が選択され、NAND ゲートのみを用いることとする。

(6) N 個の NAND ゲートが並んだ NAND ゲートアレイ (以下では NANDs と表す。)を考える。

入力 A , B 及び出力 C についての結線は図VIのようになる。これを図VIIのように表す。



図VI



図VII

図VIIIにおいて、

A を「LLLL HHLH HHHH LHLH LLHH HLLH LLLH HLLH」、

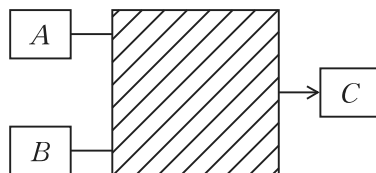
B を「HLLH LLHL LLHH HLLH HLHH LLHL LHHL LHHH」

としたとき、NANDs を組み合わせて、

C を「LLLL HHLH HHLH LHLH LLLL HLLH LLLH HLLL」

とすることを考える。このとき、図VIIIの斜線部に入る回路の機能を表す論理式を示せ。

ただし、 A と B を入力とする NANDs は $(A \circ B)$ と表してよいものとする。



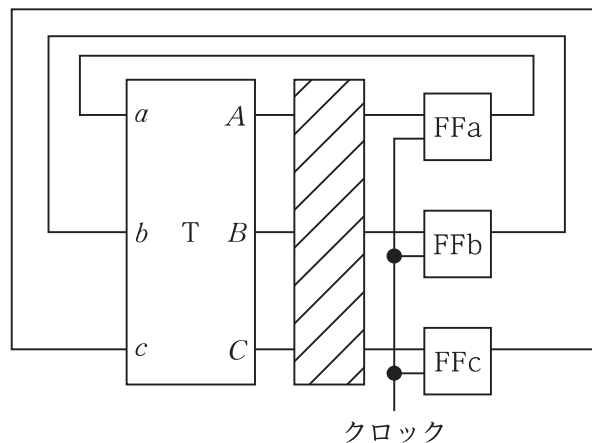
図VIII

(7) NAND ゲートのように、そのみを組み合わせることで任意の真理値表を実現できるゲートを機能的に完全なゲートという。また、NOT ゲートのように可能な全ての入力と可能な全ての出力の間に一対一対応が成り立つゲートを可逆なゲートと呼ぶ。

三つの入力 a, b, c と出力 A, B, C をもち、 A には a を、 B には b を出力し、 C には a と b が共に H のときには $\bar{c}$ 、そうでないときには c を出力するようなゲートをトフォリゲートと呼ぶ。以下の問いに答えよ。

- トフォリゲートの真理値表を書き、トフォリゲートが可逆であることを示せ。
- トフォリゲートが機能的に完全なゲートであることを示せ。ただし、定数の入力は可能であるとする。
- 図IXのように、トフォリゲート T と三つの D-フリップフロップ FFa, FFb, FFc を組み合わせた回路を考える。図IXの斜線部分では T の出力 (A, B, C) を三つの D-フリップフロップの入力に一対一で接続する。ある時刻に入力 (a, b, c) が (H, H, L) であったとする。このとき、再び入力 (a, b, c) が (H, H, L) となるまでのクロック数が最大となるような、斜線部分の接続を一つ挙げよ。ただし、解答は以下の例に示す書式で記述すること。

例： $A \rightarrow \text{FFa}, B \rightarrow \text{FFb}, C \rightarrow \text{FFc}$



図IX

(8) 2 入力 2 出力の回路について以下の問いに答えよ。ただし、入力を (a, b) 、出力を (A, B) とする。

(a) XOR ゲート一つを使った 2 入力 2 出力の可逆な回路と、NOT ゲート一つを使った 2 入力 2 出力の可逆な回路を構成して図示せよ。

(b) (a)で構成した回路を単位ブロックとし、次の真理値表をもつ 2 入力 2 出力の可逆な回路を構成して図示せよ。

入力 a	入力 b	出力 A	出力 B
0	0	0	1
0	1	0	0
1	0	1	0
1	1	1	1

(c) 2 入力 2 出力の次の真理値表を考える。ただし、 X_0, X_1, X_2, X_3 はそれぞれ任意の真理値である。この真理値表の機能を実現する回路は、(a)で構成した回路を直列に組み合わせるだけでは構成できない。その理由を説明せよ。

入力 a	入力 b	出力 A	出力 B
0	0	1	X_0
0	1	1	X_1
1	0	1	X_2
1	1	0	X_3

【No. 4】 計算機による 1 次連立方程式の解法について、以下の設問に答えよ。

計算機で、 $x_0, x_1, \dots, x_{n-1}$ についての n 元連立 1 次方程式を解くことを考える。すなわち、

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix} \text{において、} \mathbf{Ax} = \mathbf{b} \text{の} \mathbf{x} \text{を求め}$$

ることを考える。ここで、 $a_{i,j} (i = 0, 1, \dots, n-1, j = 0, 1, \dots, n-1)$ 及び $b_i (i = 0, 1, \dots, n-1)$ は実数である。本問においては、行列は最も上の行を 0 行、最も左の列を 0 列とする。

本問では、プログラムは C 言語(ANSI C)で記述するものとし、`<math.h>` が読み込まれているものとする。

- (1) 計算機で連立方程式を解く方法として、ガウス・ジョルダン法がある。この基本は、拡大係数行列

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} & b_0 \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} & b_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} & b_{n-1} \end{pmatrix}$$

に対して行基本変形を施し、 $a_{i,j} = \delta_{ij}$ (δ_{ij} はクロネッカーのデルタ) とすることで、 $b_0, b_1, \dots, b_{n-1}$

に解を得るというものである。例えば、 $\begin{pmatrix} -1 & 0 & -1 & 2 \\ 1 & 1 & 3 & -2 \\ 3 & 2 & 1 & 0 \\ 2 & -1 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -2 \\ -2 \\ 4 \\ 1 \end{pmatrix}$ は次のように

解かれる。

$$\begin{pmatrix} -1 & 0 & -1 & 2 & -2 \\ 1 & 1 & 3 & -2 & -2 \\ 3 & 2 & 1 & 0 & 4 \\ 2 & -1 & 0 & 3 & 1 \end{pmatrix} \xrightarrow{\substack{0 \text{ 行と 1 行を入れ替え} \\ 2 \text{ 行} \leftarrow 2 \text{ 行} - 0 \text{ 行} \times 3 \\ 3 \text{ 行} \leftarrow 3 \text{ 行} - 0 \text{ 行} \times 2}} \begin{pmatrix} 1 & 1 & 3 & -2 & -2 \\ -1 & 0 & -1 & 2 & -2 \\ 3 & 2 & 1 & 0 & 4 \\ 2 & -1 & 0 & 3 & 1 \end{pmatrix} \xrightarrow{\substack{1 \text{ 行} \leftarrow 1 \text{ 行} + 0 \text{ 行} \\ 2 \text{ 行} \leftarrow 2 \text{ 行} - 0 \text{ 行} \times 3 \\ 3 \text{ 行} \leftarrow 3 \text{ 行} - 0 \text{ 行} \times 2}} \begin{pmatrix} 1 & 1 & 3 & -2 & -2 \\ 0 & 1 & 2 & 0 & -4 \\ 0 & -1 & -8 & 6 & 10 \\ 0 & -3 & -6 & 7 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 3 & -2 & -2 \\ 0 & 1 & 2 & 0 & -4 \\ 0 & -1 & -8 & 6 & 10 \\ 0 & -3 & -6 & 7 & 5 \end{pmatrix} \xrightarrow{\substack{0 \text{ 行} \leftarrow 0 \text{ 行} - 1 \text{ 行} \\ 2 \text{ 行} \leftarrow 2 \text{ 行} + 1 \text{ 行} \\ 3 \text{ 行} \leftarrow 3 \text{ 行} + 1 \text{ 行} \times 3}} \begin{pmatrix} 1 & 0 & 1 & -2 & 2 \\ 0 & 1 & 2 & 0 & -4 \\ 0 & 0 & -6 & 6 & 6 \\ 0 & 0 & 0 & 7 & -7 \end{pmatrix} \xrightarrow{2 \text{ 行} \leftarrow 2 \text{ 行} \div (-6)} \begin{pmatrix} 1 & 0 & 1 & -2 & 2 \\ 0 & 1 & 2 & 0 & -4 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 7 & -7 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 1 & -2 & 2 \\ 0 & 1 & 2 & 0 & -4 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 7 & -7 \end{pmatrix} \xrightarrow{\substack{0 \text{ 行} \leftarrow 0 \text{ 行} - 2 \text{ 行} \\ 1 \text{ 行} \leftarrow 1 \text{ 行} - 2 \text{ 行} \times 2}} \begin{pmatrix} 1 & 0 & 0 & -1 & 3 \\ 0 & 1 & 0 & 2 & -2 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 7 & -7 \end{pmatrix} \xrightarrow{3 \text{ 行} \leftarrow 3 \text{ 行} \div 7} \begin{pmatrix} 1 & 0 & 0 & -1 & 3 \\ 0 & 1 & 0 & 2 & -2 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 7 & -7 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 3 \\ 0 & 1 & 0 & 2 & -2 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix} \xrightarrow{\begin{array}{l} 0 \text{ 行} \leftarrow 0 \text{ 行} + 3 \text{ 行} \\ 1 \text{ 行} \leftarrow 1 \text{ 行} - 3 \text{ 行} \times 2 \\ 2 \text{ 行} \leftarrow 2 \text{ 行} + 3 \text{ 行} \end{array}} \begin{pmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -2 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

よって、 $x_0 = 2$, $x_1 = 0$, $x_2 = -2$, $x_3 = -1$ である。

このアルゴリズムにおいては、 $a_{i,i}$ を 1 にするために i 行を $a_{i,i}$ で割る (この $a_{i,i}$ をピボットという)。このとき、 $a_{i,i}$ が 0 である場合には行の入替えを必要とする。厳密値で計算できるときには必要ないが、計算機による数値計算の場合では、 $a_{i,i}$ の絶対値が小さいと計算誤差が大きくなるおそれがあるため、 $a_{i,i}$ の絶対値が最大になるように行を入れ替えるとよいことが知られている (ピボット選択)。このピボット選択は、着目している行と、それより下の行とを交換して実行することが多い。

記号定数 N について、 N 元 1 次連立方程式をガウス・ジョルダン法で解く `gauss_jordan` 関数を次のように作成した。戻り値は成功がゼロ、失敗 (係数行列が正則でなかったときなど) が非ゼロである。係数行列は `a[][ ]`、右辺の定数項は `b[]` で与えられる。処理が正常に完了すると、`b[]` が解に置き換えられる。ここで、本プログラムでは、 i 行目を処理する際に $a_{i,i}$ の絶対値が最大になるようなピボット選択を実装している。これについて、以下の問いに答えよ。

```
int gauss_jordan(double a[N][N], double b[N]) {
    int crow, i, j, pivrow;
    double maxpiv, invpiv, tmp;

    for (crow = 0; crow < N; crow++) {
        maxpiv = 0.0;
        for (i = crow; i < N; i++) {
            if (fabs(a[i][crow]) > maxpiv) {
                maxpiv = ㉞ ;
                pivrow = i;
            }
        }
        if ( ㉟ ) return !0; /* 係数行列が正則でない */
        if (crow != pivrow) {
            for (i = 0; i < N; i++) {
                ㊦
            }
        }
    }
}
```

```

    ㊦
}
invpiv = 1.0 / a[crow][crow];
for (i = 0; i < N; i++) a[crow][i] = ㊧ ;
b[crow] = ㊨ ;
for (i = 0; i < N; i++) {
    if (i == crow) continue;
    tmp = a[i][crow];
    for (j = 0; j < N; j++) a[i][j] = ㊩ ;
    b[i] = ㊪ ;
}
}
return 0;
}

```

(a) 問題文の例に倣い、連立方程式 $\begin{pmatrix} 1 & -1 & 2 & -3 \\ 2 & 3 & -1 & -1 \\ -2 & 2 & 1 & 1 \\ 0 & 2 & 2 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \\ 5 \\ 4 \end{pmatrix}$ をガウス・ジョルダン法で解け。

(b) プログラムの空所㊦～㊪に適切な記述を補充せよ。

ただし、㊦、㊧、㊨、㊩、㊪及び㊫には式が、㊬、㊭には複数の式文が補充される。

(2) 計算機による 1 次連立方程式の解法に LU 分解を使ったものが知られている。これは、連立方程式 $A\mathbf{x} = \mathbf{b}$ において、同じ A に対して $\mathbf{b}$ を変えて複数回方程式を解く際に有効である。

LU 分解では、 A を下三角行列 L と上三角行列 U に分解する。例えば 4 元連立方程式の場合、

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \text{ は } L = \begin{pmatrix} l_{0,0} & 0 & 0 & 0 \\ l_{1,0} & l_{1,1} & 0 & 0 \\ l_{2,0} & l_{2,1} & l_{2,2} & 0 \\ l_{3,0} & l_{3,1} & l_{3,2} & l_{3,3} \end{pmatrix} \text{ と } U = \begin{pmatrix} u_{0,0} & u_{0,1} & u_{0,2} & u_{0,3} \\ 0 & u_{1,1} & u_{1,2} & u_{1,3} \\ 0 & 0 & u_{2,2} & u_{2,3} \\ 0 & 0 & 0 & u_{3,3} \end{pmatrix} \text{ を}$$

用いて $A = LU$ と分解される。

一般の n 元連立方程式の場合にも LU 分解ができれば、 $A\mathbf{x} = (LU)\mathbf{x} = L(U\mathbf{x}) = \mathbf{b}$ であるか

ら、 $U\mathbf{x} = \mathbf{y}$ とおくと、まず $L\mathbf{y} = \mathbf{b}$ なる $\mathbf{y} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$ を求め、これから $U\mathbf{x} = \mathbf{y}$ を用いて $\mathbf{x}$ を求められる。

これは連立方程式を 2 回解いているが、下三角又は上三角行列が係数行列である連立方程式

は、一般のそれより解くのが簡単であることが知られているため、一度 LU 分解をすれば、以降は効率的に連立方程式を解ける。

行列を LU 分解する方法の一つは、 A を行の入れ替えを除いた行基本変形で上三角行列に変

形することである。例えば、 $\begin{pmatrix} -1 & 0 & -1 & 2 \\ 1 & 1 & 3 & -2 \\ 3 & 2 & 1 & 0 \\ 2 & -1 & 0 & 3 \end{pmatrix}$ は次のように上三角行列に変形できる。

$$\begin{pmatrix} -1 & 0 & -1 & 2 \\ 1 & 1 & 3 & -2 \\ 3 & 2 & 1 & 0 \\ 2 & -1 & 0 & 3 \end{pmatrix} \xrightarrow{\substack{1 \text{ 行} \leftarrow 1 \text{ 行} + 0 \text{ 行} \\ 2 \text{ 行} \leftarrow 2 \text{ 行} + 0 \text{ 行} \times 3 \\ 3 \text{ 行} \leftarrow 3 \text{ 行} + 0 \text{ 行} \times 2}} \begin{pmatrix} -1 & 0 & -1 & 2 \\ 0 & 1 & 2 & 0 \\ 0 & 2 & -2 & 6 \\ 0 & -1 & -2 & 7 \end{pmatrix} \xrightarrow{\substack{2 \text{ 行} \leftarrow 2 \text{ 行} - 1 \text{ 行} \times 2 \\ 3 \text{ 行} \leftarrow 3 \text{ 行} + 1 \text{ 行}}} \begin{pmatrix} -1 & 0 & -1 & 2 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & -6 & 6 \\ 0 & 0 & 0 & 7 \end{pmatrix}$$

この得られた行列が U となる。ここで、行基本変形は行列で表せる。例えば、4 次の正方行

列において、「第 2 行に第 0 行の 3 倍を足し込む」行列は $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ と表せる。一般には、

n 次の正方行列において p 行に q 行の m 倍を足し込む行列 T の (i, j) 成分 $t_{i,j}$ は次のように表せる。

$$t_{i,j} = \begin{cases} m & (i = p, j = q \text{ のとき}) \\ \delta_{ij} & (\text{それ以外}) \end{cases}$$

これを目的の行列に左からかけることで変形が行われる。したがって、この方法で上三角行列 U をつくることは、それまでに要した行基本変形を行う行列を $L_1, L_2, \dots, L_k$ としたとき、 $L_k L_{k-1} \cdots L_2 L_1 A = U$ と表せる。これは $A = (L_k L_{k-1} \cdots L_2 L_1)^{-1} U$ と変形できるため、 $(L_k L_{k-1} \cdots L_2 L_1)^{-1} = L_1^{-1} L_2^{-1} \cdots L_k^{-1} = L$ と表せる。ここで、下三角行列どうしの積が下三角行列であることと、下三角行列の逆行列は下三角行列になることが知られているため、 $L_1, L_2, \dots, L_k$ が下三角行列であれば、 L は求める下三角行列であり、かくして LU 分解ができる。

この方法で、記号定数 N について、 N 次の正方行列を LU 分解する `ludecomp1` 関数を次のように作成した。行列 A (配列 `a[][]`) と B (配列 `b[][]`) の積 AB を C (配列 `c[][]`) に計算する補助関数 `matmul` も作成している。`a[][]` に LU 分解したい行列を与えると、`l[][]`, `u[][]` が LU 分解された結果の行列になる。本問では、簡単のため、`ludecomp1` 関数で LU 分解できない行列は考えないものとする (LU 分解できない行列が与えられた場合非ゼロを返す)。これについて、以下の問いに答えよ。

```

void matmul(double c[N][N], const double a[N][N], const double b[N][N]) {
    int i, j, k;

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            c[i][j] = 0.0;
            for (k = 0; k < N; k++) c[i][j] += ⊗ ;
        }
    }
    return;
}

```

```

int ludecomp1(const double a[N][N], double l[N][N], double u[N][N]) {
    int i, j, m, n;
    double c, tmp1[N][N], tmp2[N][N], tmp3[N][N], tmp4[N][N];

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            l[i][j] = i == j ? 1.0 : 0.0; u[i][j] = a[i][j];
            tmp1[i][j] = tmp2[i][j] = i == j ? 1.0 : 0.0;
        }
    }

    for (i = 0; i < N - 1; i++) {
        if (u[i][i] == 0.0) return !0; /* この関数ではLU分解できない */
        for (j = i + 1; j < N; j++) {
            c = -u[j][i] / u[i][i];
            tmp1[ ⊖ ][ ⊕ ] = c;
            tmp2[ ⊖ ][ ⊗ ] = ⊕ ;
            matmul(tmp3, l, tmp2); matmul(tmp4, tmp1, u);
            for (m = 0; m < N; m++) {
                for (n = 0; n < N; n++) {
                    l[m][n] = tmp3[m][n]; u[m][n] = tmp4[m][n];
                }
            }
        }
        tmp1[ ⊖ ][ ⊕ ] = 0.0;
    }
}

```

```

    tmp2[ ⑤ ][ ⑧ ] = 0.0;
}
}
return 0;
}

```

(a) 問題文の例に倣い、行列 $\begin{pmatrix} 2 & 1 & -1 & 0 \\ -4 & -3 & 4 & 0 \\ 0 & -2 & 5 & -2 \\ -2 & 0 & 1 & -5 \end{pmatrix}$ を上三角行列に変形せよ。

(b) 本問の方法で LU 分解をするためには、「 p 行に q 行の m 倍を足し込む行列 T 」の逆行列 T^{-1} が必要である。 T^{-1} の成分を問題文の行列 T の成分表示に倣って示せ。

(c) プログラムの空所⑦～⑨に適切な式を補充せよ。

(3) LU 分解において、前問(2)の方法よりも優れたドゥーリトル法が知られている。例えば、3 次の正方行列が次のように LU 分解できたとする。

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} = \begin{pmatrix} l_{0,0} & 0 & 0 \\ l_{1,0} & l_{1,1} & 0 \\ l_{2,0} & l_{2,1} & l_{2,2} \end{pmatrix} \begin{pmatrix} u_{0,0} & u_{0,1} & u_{0,2} \\ 0 & u_{1,1} & u_{1,2} \\ 0 & 0 & u_{2,2} \end{pmatrix}$$

すると、

- ・ $i < j$ のとき、 $a_{i,j} = l_{i,0}u_{0,j} + l_{i,1}u_{1,j} + \cdots + l_{i,i}u_{i,j}$
- ・ $i = j$ のとき、 $a_{i,j} = l_{i,0}u_{0,j} + l_{i,1}u_{1,j} + \cdots + l_{i,i}u_{i,j}$
- ・ $i > j$ のとき、 $a_{i,j} = l_{i,0}u_{0,j} + l_{i,1}u_{1,j} + \cdots + l_{i,j}u_{j,j}$

である。これは、未知数が 12 個、式が 9 個の方程式であり (一般の n 次の正方行列の場合は、未知数が $n^2 + n$ 個、式の数 n^2 個)、不定な連立方程式である。そこで、ドゥーリトル法では、 $l_{i,i}$ を 1 と定めてしまう。これで、未知数と式の個数が一致する。すると、方程式は、

$$\begin{cases} a_{0,0} = u_{0,0} \\ a_{0,1} = u_{0,1} \\ a_{0,2} = u_{0,2} \\ a_{1,0} = l_{1,0}u_{0,0} \\ a_{1,1} = l_{1,0}u_{0,1} + u_{1,1} \\ a_{1,2} = l_{1,0}u_{0,2} + u_{1,2} \\ a_{2,0} = l_{2,0}u_{0,0} \\ a_{2,1} = l_{2,0}u_{0,1} + l_{2,1}u_{1,1} \\ a_{2,2} = l_{2,0}u_{0,2} + l_{2,1}u_{1,2} + u_{2,2} \end{cases}$$

となる。この方程式は、最初の三つの式で $u_{0,0}$, $u_{0,1}$, $u_{0,2}$ が求められ、これを使って四つ目の式で $l_{1,0}$ が求められ、これらを用いて五つ目の式で $u_{1,1}$ が求められ、……、のように、上から順に未知数を決定できる。より一般には、 n 次の正方行列について、 $i = 0, 1, 2, \dots, n-1$ につい

て、次の二つを順に行い、 $l_{i,j}$ と $u_{i,j}$ を定める。

$$\cdot j = 0, 1, \dots, i-1 \text{ について、 } l_{i,j} = \frac{1}{u_{j,j}} \left(a_{i,j} - \sum_{k=0}^{j-1} l_{i,k} u_{k,j} \right)$$

$$\cdot j = i, i+1, \dots, n-1 \text{ について、 } u_{i,j} = a_{i,j} - \sum_{k=0}^{i-1} l_{i,k} u_{k,j}$$

例えば、
$$\begin{pmatrix} -1 & 0 & -1 & 2 \\ 1 & 1 & 3 & -2 \\ 3 & 2 & 1 & 0 \\ 2 & -1 & 0 & 3 \end{pmatrix}$$
 は次のように LU 分解される。

- $i = 0$ のとき、 $u_{0,0} = a_{0,0} = -1$, $u_{0,1} = a_{0,1} = 0$, $u_{0,2} = a_{0,2} = -1$, $u_{0,3} = a_{0,3} = 2$ 。
- $i = 1$ のとき、 $l_{1,0} = \frac{1}{u_{0,0}} \times a_{1,0} = \frac{1}{-1} \times 1 = -1$, $u_{1,1} = a_{1,1} - l_{1,0}u_{0,1} = 1 - (-1) \times 0 = 1$,
 $u_{1,2} = a_{1,2} - l_{1,0}u_{0,2} = 3 - (-1) \times (-1) = 2$, $u_{1,3} = a_{1,3} - l_{1,0}u_{0,3} = -2 - (-1) \times 2 = 0$ 。
- $i = 2$ のとき、 $l_{2,0} = \frac{1}{u_{0,0}} \times a_{2,0} = \frac{1}{-1} \times 3 = -3$, $l_{2,1} = \frac{1}{u_{1,1}} \times (a_{2,1} - l_{2,0}u_{0,1}) = \frac{1}{1} \times \{2 - (-3) \times 0\} = 2$, $u_{2,2} = a_{2,2} - (l_{2,0}u_{0,2} + l_{2,1}u_{1,2}) = 1 - \{(-3) \times (-1) + 2 \times 2\} = -6$, $u_{2,3} = a_{2,3} - (l_{2,0}u_{0,3} + l_{2,1}u_{1,3}) = 0 - \{(-3) \times 2 + 2 \times 0\} = 6$ 。
- $i = 3$ のとき、 $l_{3,0} = \frac{1}{u_{0,0}} \times a_{3,0} = \frac{1}{-1} \times 2 = -2$, $l_{3,1} = \frac{1}{u_{1,1}} \times (a_{3,1} - l_{3,0}u_{0,1}) = \frac{1}{1} \times \{-1 - (-2) \times 0\} = -1$, $l_{3,2} = \frac{1}{u_{2,2}} \times \{a_{3,2} - (l_{3,0}u_{0,2} + l_{3,1}u_{1,2})\} = \frac{1}{-6} \times [0 - \{(-2) \times (-1) + (-1) \times 2\}] = 0$, $u_{3,3} = a_{3,3} - (l_{3,0}u_{0,3} + l_{3,1}u_{1,3} + l_{3,2}u_{2,3}) = 3 - \{(-2) \times 2 + (-1) \times 0 + 0 \times 6\} = 7$ 。

よって、
$$\begin{pmatrix} -1 & 0 & -1 & 2 \\ 1 & 1 & 3 & -2 \\ 3 & 2 & 1 & 0 \\ 2 & -1 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -3 & 2 & 1 & 0 \\ -2 & -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 & 0 & -1 & 2 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & -6 & 6 \\ 0 & 0 & 0 & 7 \end{pmatrix}。$$

この方法で、記号定数Nについて、N次の正方行列をドゥーリトル法でLU分解する ludecomp2 関数を次のように作成した。a[][] にLU分解したい行列を与えると、l[][], u[][] がLU分解された結果の行列になる。本問でも、簡単のため、ludecomp2 関数でLU分解できない行列は考えないものとする(LU分解できない行列が与えられた場合非ゼロを返す。)。これについて、以下の問いに答えよ。

```
int ludecomp2(const double a[N][N], double l[N][N], double u[N][N]) {
    int i, j, k;
    double s;

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            l[i][j] = i == j ? 1.0 : 0.0; u[i][j] = 0.0;
```

```

    }
}
for (i = 0; i < N; i++) {
    for (j = 0; j < i; j++) {
        s = 0.0;
        for (k = 0; k < j; k++) s += ㉚ ;
        if (u[j][j] == 0.0) return !0; /* この関数ではLU分解できない */
        l[i][j] = ㉛ ;
    }
    for (j = i; j < N; j++) {
        s = 0.0;
        for (k = 0; k < i; k++) s += ㉜ ;
        u[i][j] = ㉝ ;
    }
}
return 0;
}

```

- (a) 問題文の例に倣い、行列 $\begin{pmatrix} 2 & 1 & -1 & 0 \\ -4 & -3 & 4 & 0 \\ 0 & -2 & 5 & -2 \\ -2 & 0 & 1 & -5 \end{pmatrix}$ をドゥーリトル法で LU 分解せよ。
- (b) プログラムの空所㉚～㉝に適切な式を補充せよ。

【No. 5】 以下の設問に答えよ。

カメラは、三次元空間の光を二次元のセンサーに射影し、その像を取得する装置である。三次元空間中の物体から放射された光はレンズを通過して二次元センサー上に結像し、センサー上で格子状に配置された受光素子により光強度として検出される。検出された信号はデジタル化され、画素の集合である画像として得られる。

本問では、カメラモデルとして最も簡略なピンホールカメラモデルを考える。図 I のように、ピンホールカメラでは、レンズの役割をするのはピンホールであり、三次元直交座標系の点 $X = (X, Y, Z)$ から発せられた光のうち、直線状にピンホール O を通ったものがセンサー上の点 $x = (x, y)$ に当たり、センサー上に結ばれた像は画像として取り出される。

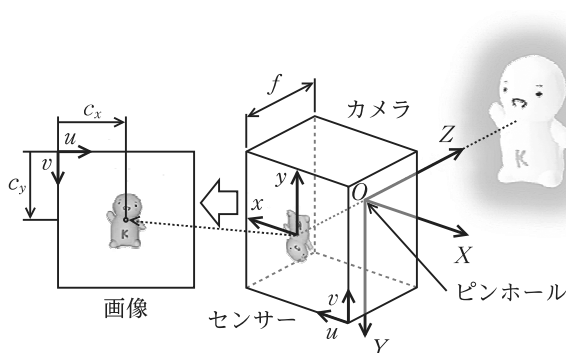


図 I

(1) 図 II のように、二次元空間 (X, Z) にあるピンホールカメラを考える。ピンホール O を原点として、センサーは $Z = -f$ にあり、 $X = 0$ (ゼロ) を光軸とする。センサー上で光軸が通る点を画像中心とし、センサーの原点としたとき、点 $X' = (X, Z)$ は一次元のセンサー上の点座標値 x にある点に射影される。ただし、像は逆向きに投影されるので、 x の座標軸は X 軸と反対の向きに取る。以下の問いに答えよ。

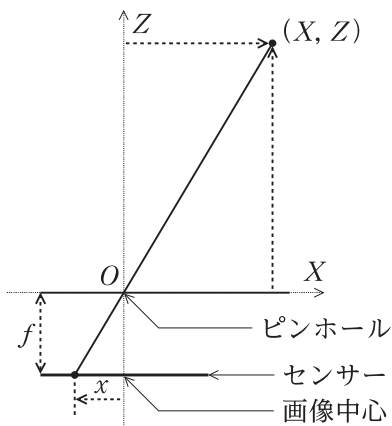
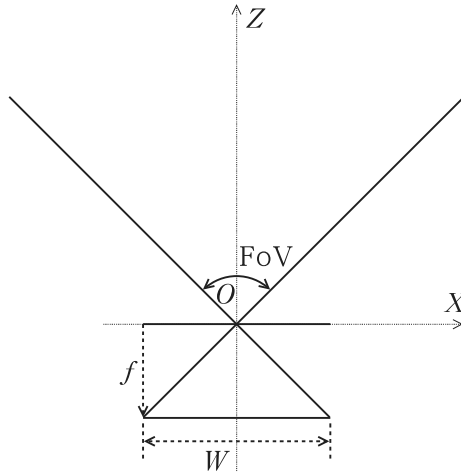


図 II

- (a) このとき、 x を f , X , Z を用いて示せ。ただし、 f , X , Z は同じ単位で表されているとする。
- (b) ここでは一次元のセンサーを考えているので、一次元の画像が取り出される。図Ⅲのように、画像に写る範囲を角度で表したものを FoV(視野角)とする。センサーの受光部の大きさを W とするとき、FoV を求める数式を f , W を用いて示せ。

ただし、画像中心はセンサーの中心と一致しているとする。



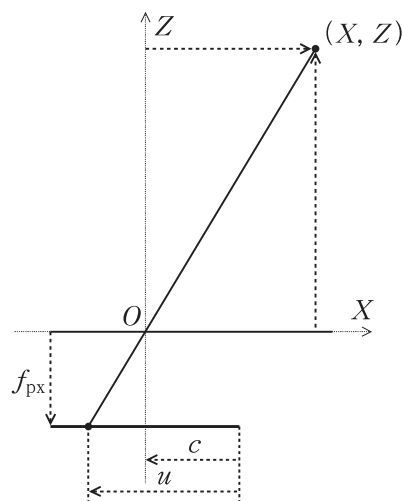
図Ⅲ

- (c) 1画素の大きさが1であるような長さの単位を px(ピクセル)とする。画素の大きさが $10\text{ }\mu\text{m}$ 、 $W = 2\text{ cm}$ 、 $\text{FoV} = 90^\circ$ とするとき、px を単位として焦点距離 f_{px} を求めよ。

ただし、画素は等間隔に配列されており、画素の大きさは隣接する画素中心間の距離とする。

- (d) 図Ⅳのように、画像座標 u の軸を、画像の端を原点とし、 X 軸と逆向きにとる。光軸がセンサーと交わる点の画像座標値を c とする。このとき、点 $X' = (X, Z)$ が射影される点の画像上の座標値 u を f_{px} , X , Z , c を用いて表せ。

ただし、 f_{px} , c , u は px を単位として表されているとする。また、 X と Z の座標値は同じ単位で表されているとする。



図Ⅳ

(e) この変換を、行列を用いて表す。次の式の㉞、㉟、㊱に当てはまるものを示せ。

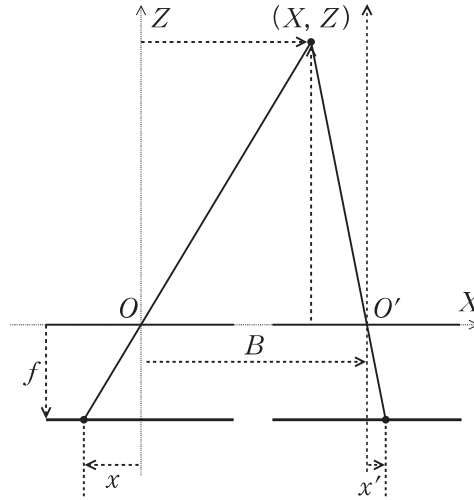
ただし、同じ画素に射影される点の座標値 (X, Z) は一意に定まらないため、その不定性をスケール s で表している。

$$s \begin{bmatrix} u \\ 1 \end{bmatrix} = \begin{bmatrix} \boxed{\text{㉞}} \\ \boxed{\text{㉟}} \end{bmatrix} \begin{bmatrix} \boxed{\text{㊱}} \\ 1 \end{bmatrix} \begin{bmatrix} X \\ Z \end{bmatrix}$$

(f) 図 I のように、 X 軸、 Z 軸と直交する方向に Y 軸を加えた三次元空間を考え、センサーはその軸と逆向きの y 軸を加え二次元とし、画素は正方形で格子状に等間隔に配列し、三次元空間中の点 $X = (X, Y, Z)$ が画像上の座標値 $u = (u, v)$ に射影されたとする。画像座標 (u, v) の原点は画像の左上に取る。このとき、 (X, Y, Z) と (u, v) はどのような関係になるか、(e) のように行列を用いて示せ。

ただし、 f_{px} は X 軸と Y 軸で共通であり、画像中心は (c_x, c_y) とする。

- (2) 次に、図Vのようにピンホールカメラが二つ平行に並んでいる場合を考える。二次元の場合、二つのセンサーが同一直線上にあり、二つのピンホール O , O' は X 軸方向にベースライン B だけ離れているとする。以下の問いに答えよ。

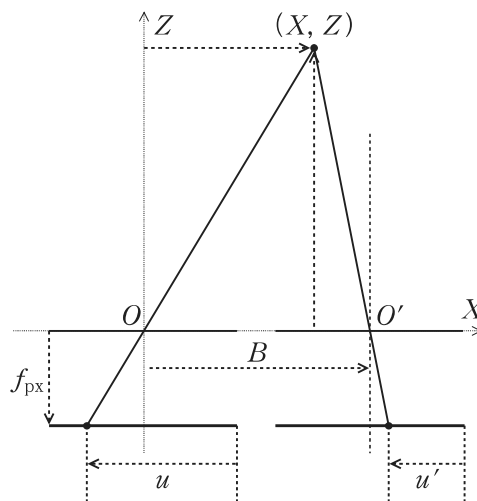


図V

- (a) 点 (X, Z) が右側のセンサー上で座標 x' に射影されたとする。左側のセンサー上に射影される座標を(1)(a)での x とする。左と右のセンサーで同一の点 (X, Z) が射影される座標の差 $D = x - x'$ を視差と呼ぶとき、 D を f , B , Z を用いて示せ。

ただし、 f , B , X , Z は同じ単位で表されているものとし、センサー上の座標 x , x' の軸は X 軸と逆向きにするものとする。

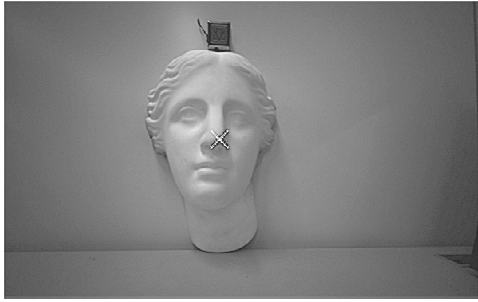
- (b) 画像座標で表した視差 $D_{px} = u - u'$ と f_{px} , B を用いて、図VIに示す奥行き Z を示せ。



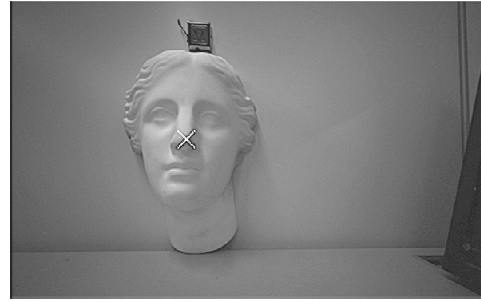
図VI

(c) 図Ⅶは左のカメラで、図Ⅷは右のカメラで取得された画像をそれぞれ示している。三次元空間中で同一の点がそれぞれの画像の「×」印で示す位置で観測されたとする。「×」印のある画素の画像座標 (u, v) が図Ⅶは $(578, 375)$ 、図Ⅷは $(473, 375)$ とすると、この点の奥行き Z を有効数字2桁で求めよ。

ただし、画素の大きさを $3\mu\text{m}$ 、焦点距離 $f = 1.88\text{ mm}$ 、ベースライン $B = 50\text{ mm}$ とする。



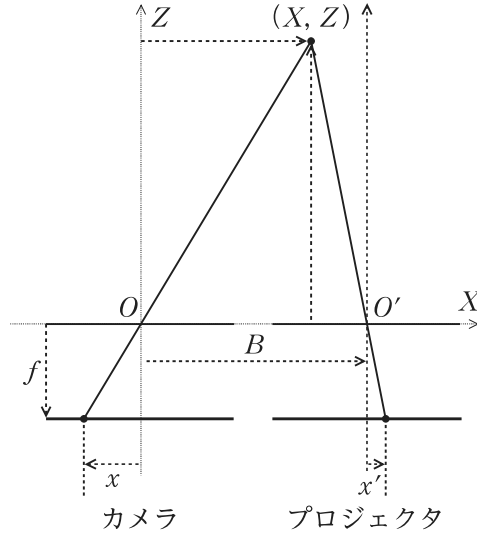
図Ⅶ



図Ⅷ

(3) (2)で見たように、二つのカメラで取得された画像に映っている同一の点が分かれば、その画像上の位置の差(視差)からその点までの奥行きが分かる。ここで必要なのは、画像間の画素の対応関係である。しかし、対象物の表面が均一であるなど、そのような対応関係を見つけるのが困難な場合がある。その場合、片方のカメラの代わりにプロジェクタを用いることで、光がどの方向から投射されたかが分かるようにすれば、奥行き情報の獲得が可能になる。

プロジェクタには様々な方式があるが、ここでは、センサーと同じ位置に、画素ごとに発光が制御できるような発光デバイスが置かれていて、発光デバイスの各画素から発せられた光がピンホールを通ることによって、三次元空間中に強弱のパターン光を投射するような装置を想定する。三次元空間中の物体に当たった光のパターンがカメラで画像として観測される。ここでは、図Ⅸのように、(1)、(2)と同様に二次元に簡略化した場合について考える。



図IX

プロジェクタで図Xに示すような三つのパターンを順次投射する。ここで、図Xでは概念を分かりやすくするため、実際の三次元物体に二次元のパターンを投射した例を示している。それぞれのパターンで、発光デバイスの位置 u' から射出される光の輝度は次の式で表される。

$$\text{パターン 1 : } J_1(u') = \frac{1}{2}(\cos \varphi + 1)$$

$$\text{パターン 2 : } J_2(u') = \frac{1}{2}\left(\cos\left(\varphi + \frac{2\pi}{3}\right) + 1\right)$$

$$\text{パターン 3 : } J_3(u') = \frac{1}{2}\left(\cos\left(\varphi + \frac{4\pi}{3}\right) + 1\right)$$

ただし、 $\varphi = \frac{2\pi u'}{\lambda}$ とし、 λ はパターンの波長とする。また、パターンを切り替えている間、対象物体は静止していて環境光は一定とし、カメラの感度は光の強度と線形の関係にあるとする。

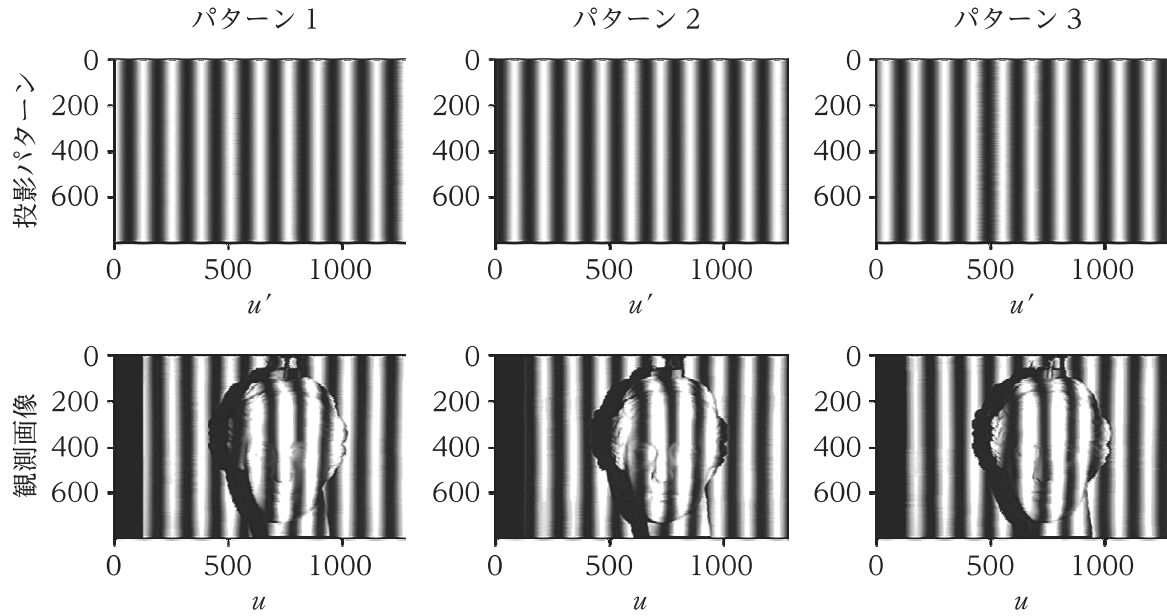


図 X

パターン 1、2、3 を投射したとき、カメラの画素 u で観測された輝度値が、それぞれ $I_1(u)$, $I_2(u)$, $I_3(u)$ であるとする。これらの輝度値に関して、未知数 α , β , φ に関する次の連立方程式を立てる。

$$I_1(u) = \alpha \cos \varphi + \beta$$

$$I_2(u) = \alpha \cos \left(\varphi + \frac{2\pi}{3} \right) + \beta$$

$$I_3(u) = \alpha \cos \left(\varphi + \frac{4\pi}{3} \right) + \beta$$

以下の問いに答えよ。ただし、位相値 $\varphi = \frac{2\pi u'}{\lambda}$ とする。

- (a) β を $I_1(u)$, $I_2(u)$, $I_3(u)$ で表せ。
- (b) φ を $I_1(u)$, $I_2(u)$, $I_3(u)$ で表せ。
- (c) 次の㊶～㊸の要素のうち、項 β で表されている要素はどれか。

ただし、点 P は対象とする物体上の点とする。

㊶ プロジェクタと点 P の距離 ㊷ 点 P での反射率 ㊸ 点 P とカメラの距離 ㊹ 環境光

㊺ プロジェクタの光の強度

- (d) $I_1(u)$, $I_2(u)$, $I_3(u)$ から φ が分かれば u' が求められ、視差 $D_{\text{px}} = u - u'$ から奥行き値 Z が求められるが、 Z を決定する際にどのような問題があり、それに対してどのような解決方法があるか 10 行程度で説明せよ。

【No. 6】 保存データの可用性や秘匿性を保つために、複数のストレージ・デバイスを用いる様々なシステムに関する以下の I ~ IV の設問に答えよ。

本問では、データ量が C であるデータを保存するために、システムとして合計で容量 C_{all} のストレージやデバイスを必要とするとき、 $\frac{C}{C_{all}}$ をそのシステムの容量効率と呼ぶ。例えば、二つのストレージにデータを保存するミラーリングの容量効率は $\frac{1}{2}$ である。

また、ある一定の条件の下で、個々のストレージやデバイスからデータが読み書きできなくなる確率を単体故障率、同じ条件の下で、複数のストレージやデバイスで構成したシステムを用いて保存したデータが読み書きできなくなる確率をシステム故障率と呼ぶ。同様に、個々のストレージ・デバイス内に保存したデータが何らかの原因で漏えいする確率を単体漏えい率、システムを用いて保存したデータが漏えいする確率をシステム漏えい率と呼ぶ。

I. ハードディスクの故障に対処するための技術の一つに、RAID がある。

RAID-1 は、複数のハードディスクに同一のデータを保存するため、ミラーリングとも呼ばれる。

N 個のハードディスク $H_0, \dots, H_{N-1}$ による RAID-4 においては、保存されるデータがデータブロック $D_0, D_1, D_2, \dots$ から成るとき、 D_k はハードディスク $H_{k \bmod (N-1)}$ へ保存される。また、連続した $N-1$ 個のデータブロック $(D_{(N-1)i}, D_{(N-1)i+1}, \dots, D_{(N-1)i+N-2})$ のパリティがハードディスク H_{N-1} に保存される。一方、 N 個のディスク $H_0, \dots, H_{N-1}$ による RAID-5 においては、連続した $N-1$ 個のデータブロック $(D_{(N-1)i}, D_{(N-1)i+1}, \dots, D_{(N-1)i+N-2})$ とそのパリティが N 個のハードディスクにそれぞれ保存されることは RAID-4 と同じだが、ハードディスク $H_{i \bmod N}$ にパリティを保存し、それ以外のハードディスクに各データブロックを保存する。以下の問いに答えよ。

(1) N 個のハードディスクで構成される RAID-4 の容量効率はいくらか。

(2) 4 個のハードディスクで構成される RAID-4 において、データブロック $D_0, D_7, D_{14}, D_{21}, D_{28}, D_{35}$ を 1 回ずつ上書きした場合、各ハードディスクへの書き込み回数はそれぞれいくらか。

ただし、読み込み回数は数えなくてよい。

(3) 単体故障率が p_0 であるハードディスク N 個で RAID-5 を構成し、故障率はハードディスクへのアクセス回数には依存しないと仮定したとき、システム故障率はいくらか。

(4) RAID-5 は RAID-4 よりも優れていると言われている。その理由を 4 行程度で説明せよ。

II. データの可用性を保つための手段の一つに、データのバックアップがある。元データが保存されているストレージ以外に、 $N - 1$ 個のストレージにバックアップを取るシステムを考える。以下の問いに答えよ。

- (1) N 個のストレージから成るシステム全体の容量効率はいくらか。
- (2) 各ストレージの単体故障率が p_0 であるとき、 N 個のストレージから成るシステムのシステム故障率はいくらか。
ただし、故障率はストレージへのアクセス回数や設置場所等には依存しないとする。
- (3) 各ストレージの単体漏えい率が p'_0 であるとき、 N 個のストレージから成るシステムのシステム漏えい率はいくらか。
ただし、漏えい率はストレージへのアクセス回数や設置場所等には依存しないとする。
- (4) バックアップが必要となる要因には様々なものがある。人為的な要因とそれ以外の要因について、それぞれ一つ具体例を挙げよ。また、挙げた各要因への対策として、バックアップ以外による対策について、それぞれ 5 行程度で説明せよ。
- (5) RAID-1 によるデータの冗長化と、データのバックアップは、どちらもデータを複製して保存することで可用性を確保するものと考えることができる。(4)で解答した各要因に対して、バックアップの代わりに RAID-1 を導入することが適切かどうか、理由とともに示せ。

III. 重要なデータの機密性を向上させるための手段として、暗号化がある。データの暗号文を一つのストレージに保管し、暗号化に用いた鍵を別のストレージに保管するシステムを考える。以下の問いに答えよ。

- (1) データの暗号文が元のデータよりも 128 ビット長くなり、暗号化の鍵が 256 ビットであるとき、容量効率が 99 % を超えるのは、データが何バイト以上のときか求めよ。
- (2) 二つのストレージの単体故障率がどちらも 0.1 % であるとき、このシステムのシステム故障率はいくらか。

IV. ブロックチェーン技術を使った暗号資産においては、デジタル署名によって不正送金を防いでおり、署名に必要な秘密鍵の厳重な保管が必須である。そこで、以下の手順から成るシステムで秘密鍵を保管することを考える。以下の問いに答えよ。

ただし、 q を素数としたとき、秘密鍵 s は集合 $\{0, 1, 2, \dots, q - 1\}$ からランダムに選ばれている値であるとする。

秘密鍵 s の保管の手順

Step 1 : $\{0, 1, 2, \dots, q-1\}$ から乱数 r を選び、 s と r から定まる一次式を $f(x) = s + rx$ とする。

Step 2 : 三つの分散情報 $a_1 = f(1) \bmod q$, $a_2 = f(2) \bmod q$, $a_3 = f(3) \bmod q$ を計算し、三つのデバイス T_1 , T_2 , T_3 に a_1 , a_2 , a_3 を一つずつ保管する。

Step 3 : 暗号資産の取引を行う際に、 T_1 , T_2 , T_3 から a_1 , a_2 , a_3 のうち二つの分散情報を集め、 s を復元して署名に利用する。

- (1) 例として、 $q = 7$, $a_1 = 1$, $a_3 = 5$ であったとする。 s を復元せよ。
- (2) T_1 , T_2 , T_3 に保存した情報が何らかの理由で使用できなくなる単体故障率を、それぞれ p_1 , p_2 , p_3 とする。デバイスの故障によって暗号資産を失う確率はいくらか。
ただし、 T_1 , T_2 , T_3 が故障する確率は互いに独立であるとする。
- (3) T_1 , T_2 , T_3 に保存した情報が何らかの理由で漏えいする単体漏えい率をそれぞれ p'_1 , p'_2 , p'_3 とする。デバイスからの情報漏えいによって暗号資産が不正送金される確率はいくらか。
ただし、 T_1 , T_2 , T_3 の情報漏えいの確率は互いに独立であるとする。
- (4) このシステムを、 N 台のデバイスを利用するシステムに一般化することを考える。ここで、秘密鍵が s であるとき、分散情報は、 t 次多項式 $f(x) = s + r_1x + r_2x^2 + \dots + r_tx^t$ を用いて、 $a_i = f(i) \bmod q$ ($i = 1, \dots, N$) と計算される。以下の問いに答えよ。
ただし、 $r_1, \dots, r_t$ はランダムに選んだ値であるとする。
 - (a) このシステムの容量効率はいくらか。
 - (b) s を復元するために必要な分散情報は何個か、理由と共に示せ。
 - (c) 分散データを保存している各デバイスの単体故障率と単体漏えい率が、それぞれ p_0 , p'_0 で、全てのデバイスで共通であるとする。ここで、 p_0 , p'_0 はいずれも 1 より十分小さいと仮定する。端末の故障によって暗号資産を失う確率を p_0^3 より小さくし、また、端末からの情報漏えいによって暗号資産が不正送金される確率を $(p'_0)^2$ より小さくしたい。このとき、容量効率が最大となるような t と N の値はそれぞれいくらか。

科目別構成の詳細

科	目	出題数	問題番号	ページ
計算機科学		1 題	No. 1	1～3
情報工学(ハードウェア)		2 題	No. 2, 3	4～14
情報工学(ソフトウェア)		2 題	No. 4, 5	15～29
情報技術		1 題	No. 6	30～32

○ 6 題のうちから任意の 2 題を選んで解答してください。