筑波大学　情報学群　情報科学類・情報メディア創成学類

令和５年度　学群編入学試験

学力試験問題（専門科目）

[注意事項]

1. 試験開始の合図があるまで、問題の中を見てはいけません。
2. 解答用紙と下書き用紙の定められた欄に、氏名、受験番号を記入すること。
3. この問題冊子は全部で 11 ページ（表紙、白紙を除く）です。
4. 問題１から問題４（数学、情報基礎）の計４問をすべて答えなさい。
5. 解答用紙は、４問に対して、各問１枚の合計４枚を用いること。
6. 解答用紙上部の［　　］欄に解答する問題番号を記入すること。
7. 解答用紙の裏面を使用する場合には、その旨を解答用紙の表面に示してください。

問題1　数学1

実数 a，b が $a > b > 0$ を満たすものとする．$a_0 = a$，$b_0 = b$ とし，2つの漸化式

$$a_{n+1} = \frac{a_n + b_n}{2}, \quad b_{n+1} = \sqrt{a_n b_n}, \quad n = 0, 1, 2, \ldots$$

によって数列 $\{a_n\}$ と $\{b_n\}$ を定義する．

(1) 次の不等式がすべての n で成り立つことを示しなさい．

$$a_n > a_{n+1} > b_{n+1} > b_n.$$

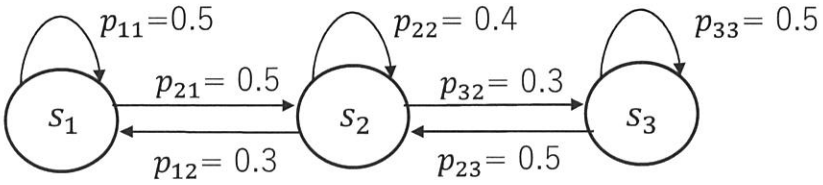
(2) (1) から $\{a_n\}$，$\{b_n\}$ はともに有界な単調数列であり，実数の連続性から $\{a_n\}$ には下限 α が，$\{b_n\}$ には上限 β が存在する．以下の設問に答えなさい．

(2-1) $a_n \to \alpha\ (n \to \infty)$，$b_n \to \beta\ (n \to \infty)$ となることを示しなさい．

(2-2) $\alpha = \beta$ が成り立つことを示しなさい．

問題2　数学2

あるシステムは3つの状態 s_1, s_2, s_3 のいずれかをとり，これら以外の状態になることはない．そして，一定時間毎に，状態 s_j から状態 s_i に以下の図の矢印で示す確率 p_{ij} で遷移するものとする $(i, j = 1, 2, 3)$．状態 s_j から状態 s_i への矢印がない場合は，$p_{ij} = 0$ とする．



このとき，行列 $A = (p_{ij})$ に関する以下の設問 (1)–(5) に答えなさい．

(1) システムの初期状態から k 回 $(k \geq 0)$ 遷移後に状態 s_1, s_2, s_3 にいる確率を q_{1k}, q_{2k}, q_{3k}，$\boldsymbol{q}_k = \begin{pmatrix} q_{1k} \\ q_{2k} \\ q_{3k} \end{pmatrix}$ とするとき，$\boldsymbol{q}_{k+1}$ を A と $\boldsymbol{q}_k$ を使って表しなさい．

(2) 行列 A の逆行列 A^{-1} を求めなさい．

(3) 確率 q_{1k}, q_{2k}, q_{3k} がそれぞれ $0.35, 0.45, 0.2$ であるとき，$\boldsymbol{q}_{k-1}$ を求めなさい．

(4) 状態 s_1, s_2, s_3 にいる確率がそれぞれ q_1, q_2, q_3 であったとき，その後に遷移しても，それぞれの状態にいる確率は変わらなかったとする．このときの q_1, q_2, q_3 を求めなさい．

(5) A の固有値すべてとその各々に対応する，最初の成分が1である固有ベクトル $\begin{pmatrix} 1 \\ a \\ b \end{pmatrix}$ を求めなさい．

問題３　情報１

データの複雑さはデータエントロピーとよばれる。データエントロピーEは入力データに出現する単位データ（シンボルとよぶ）S_iの出現確率をP_iとして以下の式で求めることができる。

$$E = -\sum_i P_i \log_2 P_i$$

データエントロピーは情報の組合せ個数を表すための最少の平均ビット数を表している。この数値は整数とは限らないため、プログラム上では整数にしたビット数で実装する。

ここで、出現確率の偏りを利用して、データを圧縮・解凍するアルゴリズムを考える。データを圧縮する処理は符号化、解凍する処理は復号化ともよばれる。符号化において以下のようなアルゴリズムを考える。

（ステップ 1）入力データのシンボルごとの出現回数を求め、そのリストをつくる。

（ステップ 2）以下の手順で二分木をつくる。

(2-1) 二分木のルートノードを作成する。現在のノードをこのルートノードとする。

(2-2) 出現回数のリストにシンボルがある場合、現在のノードの左右の子ノードを作成する。出現回数のリストにシンボルがない場合、このステップを終了する。

(2-3) 出現回数のリストから出現回数の最も多いシンボルを取り出し、現在のノードの右の子ノードにする。

(2-4) 現在のノードを左の子ノードにし、(2-2) からを繰り返す。

（ステップ 3）（ステップ 2）の二分木のルートノードから検索し、以下の手順で入力データに出現するシンボルの符号を出力する。

(3-1) 入力データがなければ終了する。入力データがあれば入力データを取り出す。

(3-2) 現在のノードをルートノードとする。

(3-3) 現在のノードの右のノードにセットされたシンボルと入力のシンボルを比較し、合致しない場合、0 を出力して、左の子ノードに移動し、(3-3) を繰り返す。合致した場合は 1 を出力して

（3-1）から繰り返す。

（ステップ 4）二分木のデータ構造を出力する。

データの復号化は、符号化の際に出力された二分木のデータ構造を用いて、以下の手順で元のデータに戻すことができる。

（ステップ 1）現在のノードをルートノードとする。

（ステップ 2）入力データがなければ終了する。入力データがあれば、入力データから 1 ビット読み出し、1 の場合、現在のノードの右の子ノードにセットされたシンボルを出力し、（ステップ 1）から繰り返す。0 の場合、現在のノードを左の子ノードにし、（ステップ 2）を繰り返す。

上述の符号化・復号化のアルゴリズムの考え方をもとに、リスト 1 のように C 言語のプログラムを実装した。ただし、encode 関数と decode 関数は、main 関数に記述されているように呼び出され、decode 関数での sym_sort は encode 関数で作られたものが再利用されるとする。

以下の問題に答えなさい。ただし、$\log_2 7 = 2.80$とし、シンボルはアルファベット 1 文字であり、圧縮前は 8 ビットで表されるとする。

（1）以下に示す入力データについて、データエントロピーEを四捨五入して有効数字 2 桁で求めなさい。

ABACBBB

（2）（1）の入力データを上述のアルゴリズムで圧縮する場合の二分木を示しなさい。

（3）（1）の入力データを上述のアルゴリズムで圧縮した場合のビット列を求めなさい。

（4）（3）で圧縮されたビット列について、シンボルあたりの平均ビット数を四捨五入して有効数字 2 桁で求めなさい。

（5）上述のアルゴリズムにより入力シンボル列のビット数を削減できる理由を答えなさい。

(6) 以下のシンボル列を圧縮した場合、リスト 1 の(X)の時点での sym_sort の要素をインデクスの小さい順にすべて答えなさい。

ABACAABBCDEABBCCADEDCDCC

(7) リスト 1 の(Y)に示す比較操作は、n個のシンボルからなる入力データを符号化する場合、(X)に到達するまでに、最悪で何回行われるか答えなさい。

(8) 上述のアルゴリズムの二分木を作らずに、sym_sort の要素の並びを利用することで、二分木を用いる場合と同じように符号化・復号化できる。リスト 1 中の①と②のコードブロックを完成させなさい。

※次ページから「リスト 1」があることに注意すること。

リスト 1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct frequency_list_type {
 char symbol;
 int frequency;
 struct frequency_list_type *next;
};

struct frequency_list_type *frequency_list;

char *sym_sort;
int num_symbols;
char comp_code[1000];

void encode (char str[]){
 int str_len = strlen(str);
 frequency_list = NULL;
 struct frequency_list_type *curr, *prev;
 num_symbols = 0;

 int i,j;
 for(i=0;i<str_len;i++){
  if(frequency_list == NULL){
   frequency_list =
       (struct frequency_list_type *)malloc(sizeof(struct frequency_list_type));
   num_symbols ++;
   frequency_list->symbol = str[i];
   frequency_list->frequency = 1;
   frequency_list->next = NULL;
  }
  else {
   curr = frequency_list;
   while(1){
    if(curr->symbol == str[i]){
     curr->frequency ++;
     break;
    }
    if(curr->next != NULL) curr = curr->next;
    else{
     curr->next =
             (struct frequency_list_type *)malloc(sizeof(struct frequency_list_type));
     num_symbols ++;
     curr = curr->next;
     curr->symbol = str[i];
     curr->frequency = 1;
     curr->next = NULL;
     break;
    }
   }
  }
 }
 sym_sort = (char *)malloc(sizeof(char)*num_symbols);
 char max_sym;
 int max_freq;
 struct frequency_list_type *max_curr, *max_prev;
```

```
 i = 0;
 while(frequency_list != NULL){
  curr = frequency_list;
  prev = frequency_list;
  max_freq = curr->frequency;
  max_sym = curr->symbol;
  max_curr = curr;
  max_prev = prev;
  while(curr->next != NULL){
   prev = curr;
   curr = curr->next;
   if(curr->frequency > max_freq){    ・・・・・・(Y)
    max_freq = curr->frequency;
    max_sym = curr->symbol;
    max_curr = curr;
    max_prev = prev;
   }
  }
  sym_sort[i] = max_curr->symbol;
  i++;
  if(max_curr == frequency_list) frequency_list = max_curr->next;
  else max_prev->next = max_curr->next;
  free(max_curr);
 }
・・・・・・(X)
 int comp_code_count = 0;
 for(i=0;i<str_len;i++){
  for(j=0;j<num_symbols;j++){
   if(str[i] == sym_sort[j]){
    putchar('1');
    comp_code[comp_code_count] = '1';
    comp_code_count ++;
    break;
   }
   else{
    putchar('0');
    comp_code[comp_code_count] = '0';
    comp_code_count ++;
   }
  }
 }
 comp_code[comp_code_count] = '\0';
}

void decode(char str[]){
 int str_len = strlen(str);
 int i,j;
 j = 0;
 for(i=0;i<str_len;i++){
  if(str[i] == '0')[        ①        ]
  else{
   putchar(sym_sort[j]);
   [        ②        ]
  }
 }
}

int main(){
 char str[] = "ABACAABBCDEABBCCADEDCDCC";
 encode(str);
 decode(comp_code);
 printf("%s\n",str);
 return 1;
}
```

問題4　情報2

オイラーの分割恒等式の正しさを部分的に確認するプログラムをC言語で作成することを考える。オイラーの分割恒等式とは、整数分割 (integer partition) に関するある性質のことである。整数分割とは、ある正の整数nを、いくつかの正の整数（**和因子**と呼ぶ）の和の形に分割することである。たとえば、整数3は「3」「2+1」「1+1+1」に分割が可能である。このとき、和因子の順番を入れ替えれば同じになるような分割（たとえば「1+2」と「2+1」）は同じ分割とみなす。そのため、3の分割は全部で3種類となる。

オイラーの分割恒等式は、正の整数nの分割のうち、「すべての和因子が奇数 (odd) である」ような分割（「3」や「1+1+1」）の数と、「すべての和因子が相異なる (distinct)」ような分割（「3」や「2+1」）の数は、常に等しくなるという性質を示している。

今回、「すべての和因子が奇数である」分割は、「すべての和因子が相異なる」分割に必ず変換可能であることを確認するプログラムのための関数をいくつか作成する。

以下のリストに示すプログラムに関する以下の問いに答えなさい。なお、各問題の解答に際しては、各関数の引数となっている配列aに関する以下の仕様に留意すること。また、空欄中では、新たな変数を宣言してはならない。

- 配列aは、一つの整数分割を表している。配列に格納されている正整数は、整数分割の各和因子を表す。
- 整数nの分割を表現する配列は、少なくとも$n+1$の長さを持っていなければならない。
- 先頭の和因子はゼロ番目の配列要素（a[0]）から始まっていなければならない。
- 最後尾の和因子の格納された配列要素の次には0が格納されていなければならない。0より後ろの配列要素は無視される。
- すべての和因子は降順に整列されていなければならない。

例：{2, 1, 0, 0}という配列は、「2+1」という分割を表現している。

(1)　関数next_partitionは、ある整数に対する分割を網羅的に生成するための関数である。next_partitionは、ある分割を配列aとして与えて呼び出すと、次の分割をaに入れて返す。配列{n, 0, ···, 0}をaの初期状態として与え、この関数が0を返すまで繰り返し呼び出すと、nに対するすべての分割を得ることができる。

空欄(A)は、与えられた分割から次の分割を求めるための前準備をしている部分である。空欄(A)の実行が終わった時点で以下の条件がすべて満たされている状態になるように、空欄(A)を埋めて関数を完成させなさい。なお、空欄(A)の中では、以下の条件で指定されていない変数や配列を書き換えてはならない。

- r には「与えられた分割の末尾に連続して存在する 1 の和因子の個数」が入っていなければならない。
- k は、1 以外で最も末尾側にある和因子を指していなければならない。言い換えると、「a[k] が、1 以外の和因子の中で、最も末尾側に位置する和因子である」という状態になっていなければならない。

(2) 関数 next_partition に以下の配列を a として与えて実行した場合、関数の実行が終了する時点で a に格納されている内容を解答しなさい。また、そのとき、X というコメントの付いている文が何回実行されるかを解答しなさい。

```
{2, 2, 1, 1, 0, 0, 0}
```

(3) 関数 is_all_odd は、与えられた分割に含まれている和因子がすべて奇数であるかどうかを検査する関数である。この関数は、和因子がすべて奇数である場合は 1 を、そうでない場合は 0 を返す。空欄 (B) を埋めてこの関数を完成させなさい。

(4) 関数 is_all_distinct は、与えられた分割に含まれている和因子がすべて相異なっているかどうかを検査する関数である。この関数は、和因子がすべて相異なっている場合は 1 を、そうでない場合は 0 を返す。以下の方針に従い、空欄 (C) を埋めてこの関数を完成させなさい。

- 配列 check を使って、ある和因子が既に出現したことがあるかどうかを記録する。具体的には、ある和因子が出現したら、その和因子に対応する要素に 1 を格納する。

(5) 関数 odd_to_distinct は、すべての和因子が奇数の分割を、すべての和因子が相異なる分割に変換する関数である。たとえば、「1+1+1+1+1+1」という分割を a を介してこの関数に与えると、関数の終了時には「4+2」という分割（すべての和因子が相異なる）が a に格納された状態になる。そのような変換は、同じ和因子が 2 つ連続している場合に、それらの和因子を加算して一つの和因子に集約するという動作を繰り返すことで実現可能である。空欄 (D) を埋めてこの関数を完成させなさい。

（問題はここまで。プログラムリストは次のページから。）

プログラムリスト

注意： 空欄の大きさは、そこに入るべき文字数や行数を反映したものではない。

```
#include <stdio.h>
#define N 6

int next_partition(int a[]) {
  int k = 0;
  int r = 0;

  while (k < N && a[k+1] != 0) {
    k++;
  }

  ┌──────────────────────────────┐
  │             (A)              │
  └──────────────────────────────┘

  if (k < 0) return 0;

  a[k]--;
  r++;

  while (r > a[k]) {
    a[k+1] = a[k]; /* X */
    r = r - a[k];
    k++;
  }

  a[k+1] = r;
  a[k+2] = 0;

  return 1;
}

int is_all_odd(int a[]) {
  int i = 0;

  while (a[i] != 0) {
    ┌──────────────────────────────┐
    │             (B)              │
    └──────────────────────────────┘
  }

  return 1;
}

int is_all_distinct(int a[]) {
  int i = N;
  int check[N];
```

```
  do {
    i--;
    check[i] = 0;
  } while (i > 0);

  while (a[i] != 0) {
```

(C)

```
  }

  return 1;
}

int odd_to_distinct(int a[]) {
  int i = 0;
  int j;

  do {
```

(D)

```
  } while (a[i] != 0);
}

/* プログラムリストはここまで。以下余白。 */
```