令和７年度

千葉大学先進科学プログラム入学者選考課題

課題論述 （数理情報学）

（12:00－15:00）

注意事項

1. この冊子は，監督者から解答を始めるよう合図があるまで開いてはいけません。
2. 問題冊子に印刷または製本の不具合がある場合は，手を上げて申し出てください。
3. 問題すべてに解答してください。
4. 解答用紙は，課題ごとに解答用紙を分けて使用してください。解答用紙は何枚使用してもかまいません。すべての解答用紙に受験番号を必ず記入してください。
5. 検査室に用意してある資料は自由に使用してかまいません。ただし，諸君が持参した教科書，参考書，ノート，パソコンなどの使用は禁止します。
6. 携帯電話やスマートフォン等の電子機器はすべて電源を切り，カバンにしまってください。
7. その他，監督者の指示に従ってください。

問題

1

以下の C 言語のプログラムに関する問いに答えなさい。

問1　データ圧縮法にシャノン・ファノ符号と呼ばれるものがある。この符号では各文字の出現割合を求めて，それに基づいて各文字に2進数の符号語を割り当てる。符号化（圧縮）の際は入力された文字をそれに割り当てられた符号語に変換する。出現割合の高い文字ほど短い符号語を割り当てることにより，符号化後の長さの総和がなるべく短くなるようにして圧縮効率を高める。

表 1-1 文字と出現割合の例

文字	割合
e	0.25
a	0.2
t	0.18
h	0.15
d	0.12
f	0.1

シャノン・ファノ符号では文字の出現割合の高い順に上から下へ並べた表を作成して，その表をそれぞれの含まれる文字の出現割合の和がほぼ等しくなるように上下2分割する。上半分の表に属す文字に対する符号語の末尾に0を付加し，下半分に属す文字に対する符号語の末尾に1を付加する。分割した表に対して同様に分割を繰り返す。次頁のコード 1-1 は上の表 1-1 のように各文字の出現割合が与えられたときのシャノン・ファノ符号における文字への符号語の割り当てを決めるプログラムである。ただし，出題の都合でアルゴリズムを一部改変してある。

以下の問いに答えなさい。

(1)上の説明文中の下線部はあいまいな記述になっている。コード1－1の divide 関数ではどのように判断しているか具体的に説明しなさい。

(2)最初に divide 関数が呼ばれたときには上の表 1-1 全体を 2 分割する。分割の位置を以下の選択肢から選んで記号で答えなさい。

① e と a の間
② a と t の間
③ t と h の間
④ h と d の間
⑤ d と f の間

(3) コード1－1中の空欄 ア ～ エ に当てはまる引数を答えなさい。

(4) コード1－1を実行したときに標準出力に出力される内容を答えなさい。

(5) コード1－1を実行したときに divide 関数が呼ばれる回数を答えなさい。

コード1－1

```
#include <stdio.h>

#define N 6
#define MAX_CODELENGTH 10
double prob[N]= {0.25,0.2,0.18,0.15,0.12,0.1};
int length[N];
char symbol[N]= {'e','a','t','h','d','f'};
char codeword[N][MAX_CODELENGTH];

void divide(int s, int t)
{
	double sum=0;
	double s1=0;
	int i,j;
	if(s==t)
		return;
	for(i=s;i<=t;i++)
		sum+=prob[i];
	for(i=s;i<=t;i++)
	{
		s1+=prob[i];
		if(s1+prob[i+1]>sum/2)
			break;
	}
	for(j=s;j<=t;j++)
	{
		if(j<=i)
			codeword[j][length[j]]='0';
		else
			codeword[j][length[j]]='1';
		length[j]++;
	}
	divide( ア , イ );
	divide( ウ , エ );
}
/* 次ページに続く */
```

```
int main(void)
{
	int i,j;
	for(i=0;i<N;i++)
	{
		length[i]=0;
		for(j=0;j<MAX_CODELENGTH;j++)
			codeword[i][j]=0;
	}
	divide(0,N-1);
	for(i=0;i<N;i++)
		printf("%c: %1.2f, %d, %s\n",symbol[i],prob[i],
			length[i],codeword[i]);
	return 0;
}
```

問2　次頁のコード 1-2 は LZ77 と呼ばれるデータ圧縮法のプログラムである。LZ77 ではこれから符号化しようとしている文字列（符号化部）の直前の一定数の文字列（参照部）を用いて符号化する。参照部には先頭から0ではじまるラベルを付けて位置を表す。符号化部の先頭から最も長く一致する文字列を参照部から探し，参照部で一致した文字列の位置（ラベル），長さ，および符号化部で一致した文字列の次の1文字の3つを中間符号語として出力している。

図1－1の例では符号化部の先頭の 3 文字が参照部のラベル 2 からの 3 文字と一致しており，符号化部の次の文字が a なので，出力する中間符号語は（2，3，a）となる。これで abca の 4 文字を符号化したことになるので，参照部と符号化部を 4 文字左にシフトさせて次の 4 文字を読み込んでいる。

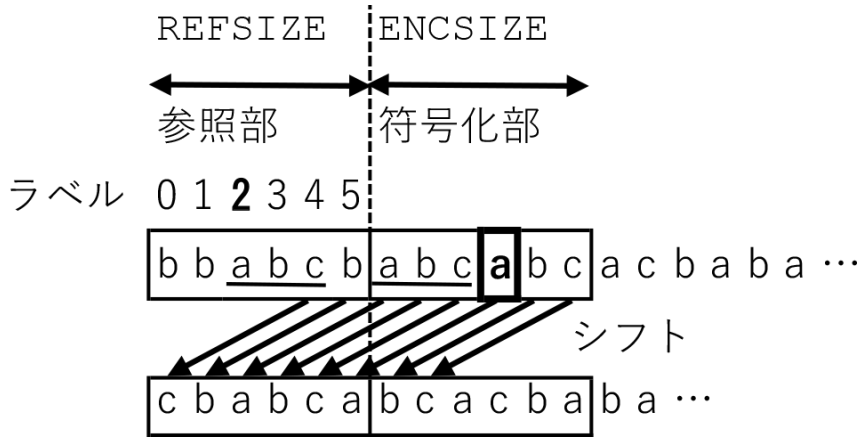

図1－1 LZ77 符号による符号化の例

以下の問いに答えなさい。

（1）図1－1の例でシフト後の符号化部 bcacba を符号化したときの中間符号語を求めなさい。
（2）コード1－2は最も長く一致する文字列が参照部に複数ある場合はどの文字列を選んでいるか具体的に説明しなさい。
（3）プログラムを開始した直後には参照部に該当する文字列はない。コード1－2 ではそのときの参照部をどのように定めているか答えなさい。
（4）コード 1－2 では参照部に一致する文字列が全くなかったときの中間符号語では，一致した文字列の位置（ラベル）と長さをどのような値で出力しているか答えなさい。

コード 1-2

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

#define REFSIZE 6
#define ENCSIZE 6

char buffer[REFSIZE+ENCSIZE];
char source[]="bbabcbabcabcacbaba";
int sourcelength, sourcepoint=0;

char readsource(void)
{
	char s;
	if(sourcepoint >= sourcelength)
		s=0;
	else
		s=source[sourcepoint];
	sourcepoint++;
	return(s);
}

int findmatch(int p)
{
	int i;
	for(i=0;i<ENCSIZE;i++)
		if(buffer[p+i] != buffer[REFSIZE+i])
			break;
	return(i);
}
void printcodeword(int point, int length, char ch)
{
	printf("(%d, %d, %c)\n",point, length, ch);
}
/* 次ページに続く */
```

```
int main(void)
{
	int i,ml;
	int maxmatch;
	int maxmatchpoint;
	sourcelength=strlen(source);
	for(i=0;i<REFSIZE+ENCSIZE;i++)
		if(i<REFSIZE)
			buffer[i]=0;
		else
			buffer[i]=readsource();
	do
	{
		maxmatch=0;
		maxmatchpoint=-1;
		for(i=0;i<REFSIZE;i++)
		{
			ml=findmatch(i);
			if(maxmatch<ml)
			{
				maxmatch=ml;
				maxmatchpoint=i;
			}
		}
		printcodeword(maxmatchpoint, maxmatch,
			buffer[REFSIZE+maxmatch]);
		for(i=0;i<REFSIZE+ENCSIZE-maxmatch-1;i++)
			buffer[i]=buffer[i+maxmatch+1];
		for(i=0;i<maxmatch+1;i++)
			buffer[REFSIZE+ENCSIZE-maxmatch-1+i]
				=readsource();
	} while(buffer[REFSIZE]>0);
	return 0;
}
```

問3　問2のプログラムでは中間符号語の出力の後に参照部と符号化部をシフトさせるために1文字ずつコピーしていく必要がある。参照部と符号化部のサイズが大きくなるとこのコピーにかかる時間が大きくなる。そこで，コピー回数を削減したシフト動作ができるよう改良したプログラムをコード 1-3 に示す。プログラムは一部省略してあり，改良した部分を太字で示している。また，省略した部分にも修正された部分がある。以下の問いに答えなさい。

（1）コード 1-3 ではどのようにしてコピー回数を削減してシフト動作を実現しているか具体的に答えなさい。（単に手法の名前を解答するだけでは不十分である。）

（2）省略されている部分にある findmatch 関数はどのように修正すればよいか。修正した findmatch を定義するプログラムを書きなさい。

コード 1-3

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

#define REFSIZE 6
#define ENCSIZE 6
int offset=0;

char buffer[REFSIZE+ENCSIZE];
char source[]="bbabcbabcabcacbaba";

int sourcelength, sourcepoint;

(省略)

int main(void)
{
(省略)
       do
       {
              (省略)
              printcodeword(maxmatchpoint, maxmatch,
                     buffer[(REFSIZE+maxmatch+offset)%
                            (REFSIZE+ENCSIZE)]);
              offset = (offset+maxmatch+1) %  (REFSIZE+ENCSIZE);
              for(i=0;i<maxmatch+1;i++)
                     buffer[(REFSIZE+ENCSIZE-maxmatch-1+i+offset)
                            %(REFSIZE+ENCSIZE)]=readsource();
       } while(buffer[(REFSIZE+offset)% (REFSIZE+ENCSIZE)]>0);
       return 0;
}
```

2

以下の C 言語のプログラムに関する問いに答えなさい。

次の3元 1 次連立方程式

$$\begin{cases} 12x_1 + 4x_2 - 2x_3 = 0 \\ 6x_1 - 2x_2 - 3x_3 = -4 \\ -6x_1 + 2x_2 + 13x_3 = -16 \end{cases}$$

を解くプログラムを C 言語で作成した。以下にそのコードを示す。

```
(01)  #include <stdio.h>
(02)  #include <stdlib.h>
(03)  #include <math.h>
(04)  #define N 3 // 連立方程式の数

(05)  int main() {
(06)      void forward ();
(07)      void back();
(08)      void printmatrix();
(09)      double matrix[N][N+1] = {
(10)          {12, 4, -2, 0},
(11)          {6, -2, -3, -4},
(12)          {-6, 2, 13, -16}
(13)      };
(14)      printf("問題:\n");
(15)      printmatrix(matrix);
(16)      forward(matrix, 0);
(17)      back(matrix, N - 1);
(18)      printf("解:\n");
(19)      for (int i = 0; i < N; i++) {
(20)          printf("x%d = %.2f\n", i + 1, matrix[i][N]);
(21)      }
(22)      return 0;
(23)  }
(24)  /* 次ページに続く */
```

```
(25)  void forward(double matrix[N][N+1], int step) {
(26)      int i, j;
(27)      int row = step;
(28)      double factor, tmp;
(29)      void printmatrix();
(30)      if (step >= N-1) return;

(31)      /*** ここからコメントアウト部***
(32)      for (i = step + 1; i < N; i++) {
(33)          if (fabs(matrix[i][step]) > fabs(matrix[row][step])) {
(34)              row = i;
(35)          }  //fabs(x)は x の絶対値
(36)      }
(37)      for (j = step; j < N + 1; j++) {
(38)          tmp = matrix[row][j];
(39)          matrix[row][j] = matrix[step][j];
(40)          matrix[step][j] = tmp;
(41)      }
(42)       *** ここまでコメントアウト部***/

(43)      for (i = step + 1; i < N; i++) {
(44)          factor = matrix[i][step] / matrix[step][step];
(45)          for (j = step; j < N + 1; j++) {
(46)              matrix[i][j] -= factor * matrix[step][j];
(47)          }
(48)      }
(49)      printmatrix(matrix);
(50)      forward(matrix, step + 1);
(51)  }
(52)        /* 次ページに続く */
```

```
(53)  void back(double matrix[N][N+1], int step) {
(54)      double factor;
(55)      int i, j;
(56)      void printmatrix();
(57)      if (step < 0) return;
(58)      factor = matrix[step][step];
(59)      for (j = step; j < N + 1; j++) {
(60)          matrix[step][j] /= factor;
(61)      }
(62)      printmatrix(matrix);
(63)      for (i = 0; i < step; i++) {
(64)          factor = matrix[i][step];
(65)          for (j = step; j < N + 1; j++) {
(66)              matrix[i][j] -= factor * matrix[step][j];
(67)          }
(68)      }
(69)      printmatrix(matrix);
(70)      back(matrix, step - 1);
(71)  }

(72)  void printmatrix(double matrix[N][N+1]) {
(73)      int i, j;
(74)      for (i = 0; i < N; i++) {
(75)          for (j = 0; j < N + 1; j++) {
(76)              printf("%.2f ", matrix[i][j]);
(77)          }
(78)          printf("\n");
(79)      }
(80)      printf("\n");
(81)  }
```

(1) 本プログラムを実行したときの標準出力を示しなさい。

(2) back 関数はその中に back 関数を用いる再帰表現で書かれている。この関数を，再帰表現を用いずに書き換えなさい。なお，解答用紙には，もとのプログラムに対してコードの行番号を参照しながら，「○○行を削除」，「○○行の後ろに次の文を追加」のように「どこに」「どのような」文を削除/追加するかが分かれば良い。

次に上記コードにおいて forward 関数の中のコメントアウトを外す。このとき，プログラムは一般化されており，main 関数内(10)～(12)行目を書き換えれば多くの異なる3元1次連立方程式を解くことができる。

(3) forward 関数においてコメントアウトされている(31)～(42)行目はコード内の方程式を解くためには不要だが，与える方程式によってはこの操作がないと解が求まらない。コメントアウト部分はどのような場合に必要となるか，そしてコメントアウト部は何をしているかを答えなさい。

(4) 与える方程式によっては解が導出できない場合がある。このような場合に「解なし」と標準出力し，プログラムを強制終了するよう，コードを修正しなさい。なお，解答用紙には，もとのプログラムに対してコードの行番号を参照しながら「どこに」「どのような」文を削除/追加するかが分かれば良い。

3

以下の Python のプログラムに関する問いに答えなさい。

整数を要素とするリストの検索を行うプログラムを考える。具体的には，整数を要素にもつリストを値とする変数 sequence と変数 pattern が与えられたとき，sequence 中で pattern に一致するすべての箇所を検索するプログラムとする。

引数 sequence のリストから引数 pattern のリストに一致するすべての箇所のインデックスを探し，一致した箇所の先頭の要素のインデックスのリストを返却値とする関数 find_simply(sequence, pattern)をコード 3-1 に示す。

たとえば，コード 3-2 に示す sequence と pattern を利用して find_simply(sequence, pattern)を呼び出した場合には，sequence 末尾の sequence[6:11] が pattern と一致する。一致した箇所の先頭の要素のインデックスは 6 であり，返却値は[6]である。図 3-1 にコード 3-2 を図示する。

コード 3-1: find_simply

```
def find_simply(sequence : list, pattern : list) -> list:
    indices = []
    sindex = 0 # sequence のインデックス
    pindex = 0 # pattern のインデックス
    slen = len(sequence)    # sequence の長さ
    plen = len(pattern)     # pattern の長さ
    while sindex < slen and pindex < plen:
        if is_equal(sequence[sindex], pattern[pindex]):
            sindex += 1
            pindex += 1
        else:
            sindex = sindex - pindex + 1
            pindex = 0
        if pindex >= plen:
            indices.append(sindex-plen) # 一致したインデックスを保存
            pindex = 0
    return indices
```

コード 3-2: sequence と pattern

```
sequence = [1, 2, 1, 2, 1, 3, 1, 2, 1, 2, 3]
pattern = [1, 2, 1, 2, 3]
```

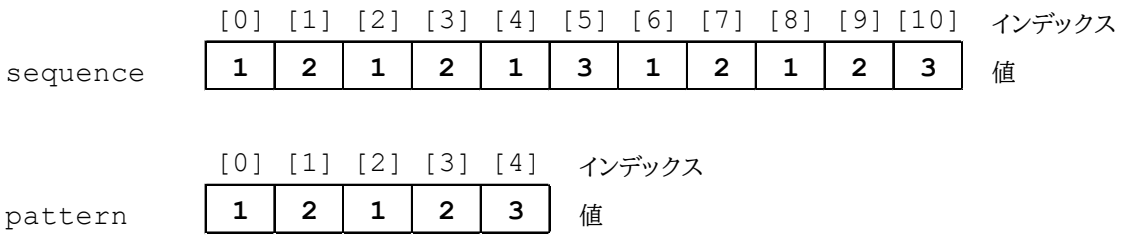

図 3-1: sequence と pattern

コード 3-3 に示す関数 is_equal(a, b)は，整数 a, b が一致する場合には True，一致しない場合には False を返却値とする。

コード 3-3: is_equal

```
def is_equal(a : int, b : int) -> bool:
  return a == b
```

(1) コード 3-2 に示すリスト sequence とリスト pattern を引数にして関数 find_simply(sequence, pattern)を実行した際に，関数 is_equal が呼び出される回数を答えなさい。

(2) リスト sequence の長さをn，リスト pattern の長さをmとした場合に，関数 find_simply(sequence, pattern)を実行した際の関数 is_equal の呼び出し回数の最大値と最小値をnとmのうち必要なものを用いた式で答えなさい。ただし，$1 \leqq m \leqq n$とする。

ここから，リストの検索の際の関数 is_equal の呼び出し回数を減らすことを考える。コード 3-2 の sequence と pattern を引数にして関数 find_simply(sequence, pattern) を実行する場合を例として呼び出し回数を減らす方法を検討する。pattern[0] と pattern[1] が異なり，pattern[0:2]と pattern[2:4]が等しいことを利用すると呼び出し回数が減らせることを見てみよう。

関数 find_simply に従って検索を進めると図 3-2 に示すように，sequence[0:4] の[1, 2, 1, 2]が pattern[0:4] の[1, 2, 1, 2]に一致し，sequence[4]と pattern[4]が不一致となる。

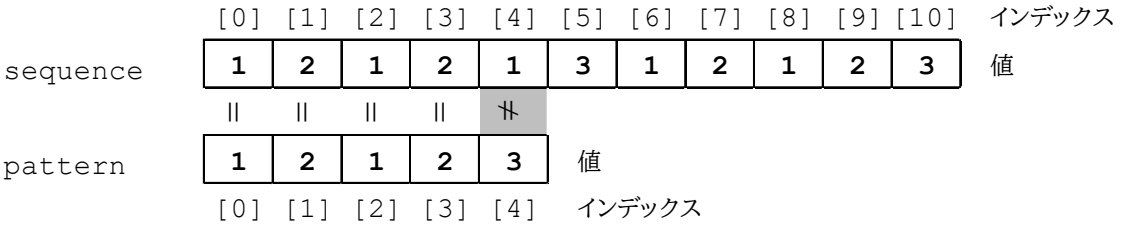

図 3-2: 不一致

関数 find_simply ではこのあとに sequence[1] の 2 と pattern[0]の 1 の比較が行われる。しかし，pattern[1]が pattern[0]と異なっているために，そのインデックスでは一致することはない。このことから，sequence との比較の際に pattern のどこまで一致したかによって，次に一致を調べる pattern の位置を無条件に先頭に戻さないで比較を進めることができる場合があることがわかる。

また，sequence[4]と pattern[4]の比較で不一致となった場合，pattern[0:4] の[1, 2, 1, 2]が sequence[0:4]に一致しており，pattern[0:2]の[1, 2]と pattern[2:4] の[1, 2]が同じである場合には，sequence[2:4]は pattern[0:2]と一致することがわかる。この場合には，図 3-3 のように sequence[4]と pattern[2]から比較を行えばよい。

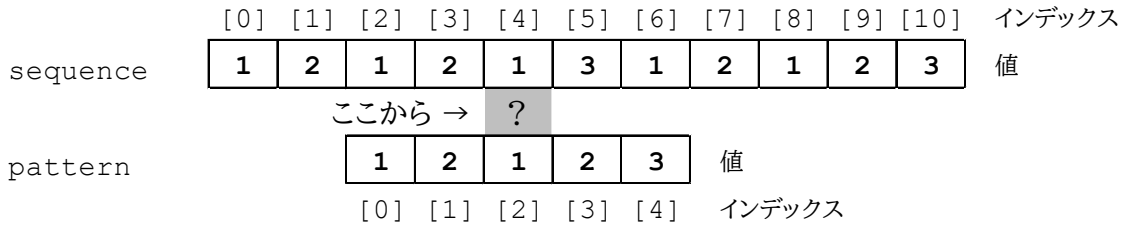

図 3-3: 無駄を省いた比較の再開

そこで，要素の比較を常に順に行うのではなく，pattern において不一致となった要素のインデックスをもとに，比較を次に行うべき sequence と pattern のインデックスを変化させて，比較回数を減少させる方法(Knuth–Morris–Pratt 法)の考え方に従ってコードの改善を行う。

まず，不一致となった要素の pattern におけるインデックス pindex から，次に比較を行うべき pattern のインデックスに変換するためのリストを next とすると，next[pindex]が次に比較を行うべき pattern のインデックスを表す。不一致となった要素の sequence におけるインデックスを sindex とすると，pattern[next[pindex]]と sequence[sindex] から比較することで無駄な比較を省く。ただし，pattern の先頭で不一致の場合には，次に比較すべき pattern のインデックスは 0 であり，sequence のインデックスは sindex+1 とする。

pattern において不一致だったインデックスから次に比較すべきインデックスに変換するためのリストを作成し，返却値とする関数 build_next(pattern)を考える。関数 build_next で作成されるリスト(next とする)は，インデックス 0 を除いて，pattern の接頭辞(prefix)の長さをインデックスとし，真の接尾辞でもある最長の真の接頭辞（longest prefix suffix – 以下 LPS と略す)の長さを要素とする。真の接頭辞(proper prefix)および真の接尾辞(proper suffix)には，空リストやリスト全体は含まれない。ただし，以下では LPS が存在しないことを，空リストで表し，LPS が存在しない場合の next の要素を 0 とする。また，next のインデックス 0 は，先頭で不一致の場合に利用され，前述の通りの処理が必要であることを示すために特別に-1 とする。

たとえば，リスト[1, 2, 1, 2, 3]の真の接頭辞は [1]，[1, 2]，[1, 2, 1]，[1, 2, 1, 2]であり，真の接尾辞は，[3]，[2, 3]，[1, 2, 3]，[2, 1, 2, 3]である。これらを図 3-4 に図示する。

より一般的に pattern の長さを plen とすると，リスト pattern の接頭辞は，pattern[0:x] (0 <= x <= plen)であり，真の接頭辞は，pattern[0:y] (0 < y < plen) である。同様に，リスト pattern の接尾辞は，pattern[z:plen] (0 <= z <= plen) であり，真の接尾辞は pattern[w:plen] (0 < w < plen)である。

[1]，[1, 2]，[1, 2, 1]，[1, 2, 1, 2]の LPS はそれぞれ，[]，[]，[1]，[1,2]であり(LPS が存在しないことを空リストで表している)，その長さはそれぞれ 0，0，1，2 である。この関係を図 3-5 に図示する。

以上から build_next([1, 2, 1, 2, 3])によって作成されるリスト next の値は図 3-6 に示すように[-1, 0, 0, 1, 2]となる。

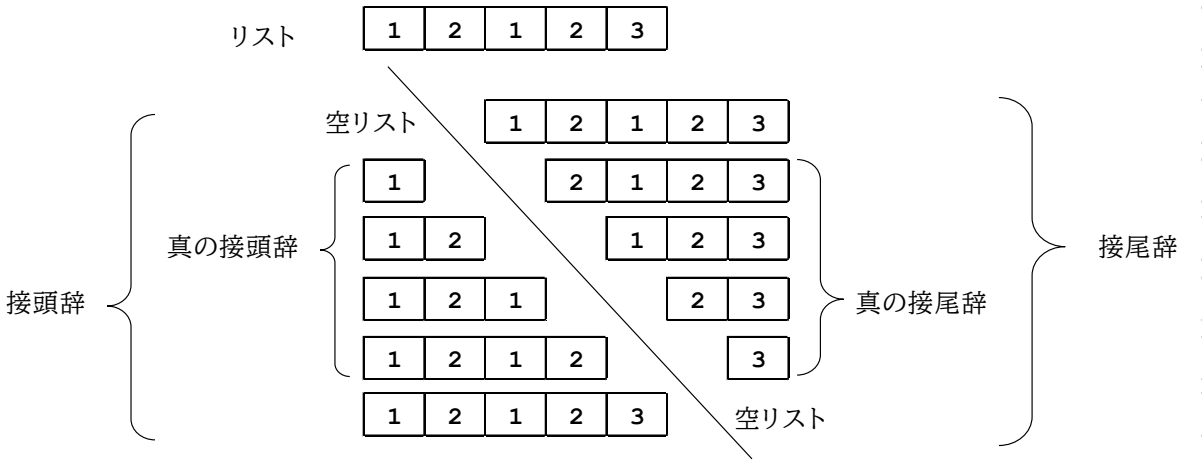

図 3-4: 真の接頭辞と真の接尾辞

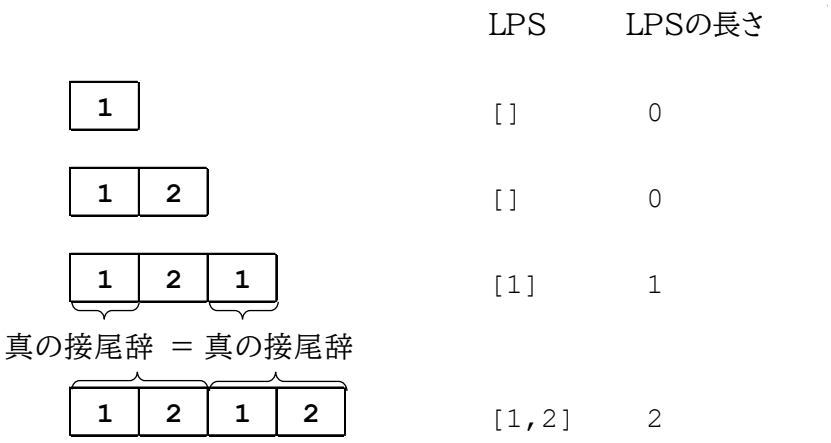

図 3-5: 真の接尾辞でもある最長の真の接頭辞(LPS)の長さ

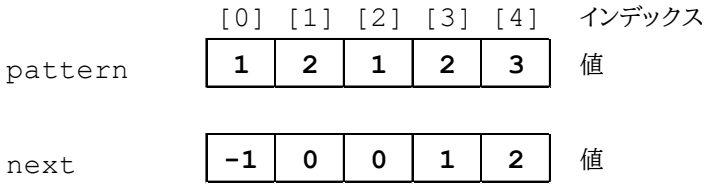

図 3-6: `pattern` と `next`

(3) リストの LPS の長さを求める関数を補助する関数 are_lists_equal を考える。関数 are_lists_equal(list1, list2) は，2 つのリスト list1 と list2 を比較し，対応する要素が一致する同一のリストの場合に True，そうでない場合に False を返却値とする関数である。
関数 are_lists_equal(list1, list2)を定義するプログラムを解答用紙に書きなさい。要素の比較には関数 is_equal を利用しなさい。解答のコードには組込み関数を用いてよい。

(4) 空リストではないリスト list の LPS の長さを求める関数 lps_length(list)を定義したプログラムをコード 3-4 に示す。コード 3-4 のプログラムの空欄 （ア）, （イ）を適切に埋め，それらを解答用紙に書きなさい。解答のコードには組込み関数を用いてよい。

コード 3-4: lps_length

```
def lps_length(list: list) -> int:
    length = len(list)-1
    while length > 0:
        if are_lists_equal(list[ （ア） ],
                           list[ （イ） ]):
            break
        length -= 1
    return length
```

(5) 関数build_next(pattern)を定義するプログラムを解答用紙に書きなさい。その中でLPSの長さを計算する際には関数 lps_length を利用しなさい。解答のコードには組込み関数を用いてよい。

(6) 関数 build_next(pattern)によって生成されるリストを next とし，それを利用して関数 is_equal の呼び出し回数を減らすようにリストの検索を行い，関数 find_simply と同じ結果を返す関数 find_with_next(sequence, pattern, next)をコード 3-5 に示す。コード 3-5 の空欄 （ウ）～（キ）を適切に埋め，それらを解答用紙に書きなさい。

コード 3-5: find_with_next

```
def find_with_next(sequence: list, pattern: list,
                                          next: list) -> list:
    indices = []
    sindex = 0
    pindex = 0
    slen = len(sequence)
    plen = len(pattern)
    while sindex < slen and pindex < plen:
        while (pindex >= 0 and
               not is_equal(sequence[sindex], pattern[pindex])):
            pindex = [   (ウ)   ]
        sindex = [   (エ)   ]
        pindex = [   (オ)   ]
        if pindex >= plen:
            indices.append( [   (カ)   ] )
            pindex = [   (キ)   ]
    return indices
```

(7) コード 3-2 のリスト sequence とリスト pattern を引数にしてコード 3-6 の関数 find_pattern(sequence, pattern)を呼び出した場合を考える。コード 3-6 の関数 find_with_next(sequence, pattern, next)を実行中に関数 is_equal が呼び出される回数を答えなさい。ただし, 関数 build_next の実行中に関数 is_equal が呼び出される回数は含めないこと。

コード 3-6: find_pattern

```
def find_pattern(sequence: list, pattern: list) -> list:
    next = build_next(pattern)
    indices = find_with_next(sequence, pattern, next)
    return indices
```