平成 26 年度過去問解答

1　アルゴリズムとプログラミング

1.1　回答

(1)

43 行目での出力は “5 6 10 ’
46 行目での出力は “2 5 5 7 13 18 20 ”

(2)

front = 3
rear = 5

(3)

B は常に昇順にソートされているので,
二分探索を用いることで計算時間 O(logn) で挿入位置を特定できる.

(4)

insert = O(n)
delete = O(1)
insert について，先の問題で 2 分探索法について尋ねられているが,
24 行目の処理で配列内に順番にアクセスしている.

(5)

rear が変化する要因は insert の実行時 1 増加するのみなので,
計 20 回 insert を実行すると配列 A の末端に来てしまい,
21 回目以降の insert は 9 行目の if 文に引っかかる.

(6)

front や rear が配列の末端に到達したら先頭に移動するようにしてループ構造にする.
または insert の実行時にデータ列を後でなく前に詰めるようにする.

1.2　解説

1.2.1　データ配列

本プログラムは固定長配列 A に対して，使用中領域を示す整数 front と rear を用いて擬似的な可変長のデータ列を実現している．また，データ列は常に昇順にソートされている.

1.2.2 insert

データ列が昇順にソートされているため，二分探索法を用いてデータの挿入位置を決める．挿入位置以降のデータは配列の後ろに移動する．“12346789_” に 5 を挿入する場合は “1234_6789” にずらした後，“123456789” となる．挿入後，rear を 1 増やす．

1.2.3 delete

データ配列の先頭の値を返り値として取り出し，front を 1 増やす．

1.2.4 (1)(2)

42-46 行の終了地点でのデータ列および front，rear の状態は以下のようになる．

表 1: 42-46 行目の実行結果

行	データ列	front	rear
42	5,6,10,13,20	0	5
43	13,20	3	5
45	2,5,5,7,13,18,20	3	10
46	ϕ	10	10

1.2.5 (3)

データ列はソートされているため二分探索を使用する．二分探索のアルゴリズムは次の通り．

1. 探索の対象となるデータ列の中央の要素を調べる．
2. 中央の要素が目的値 (ここでは挿入値) と同じならそこが挿入位置となる．
3. 中央の要素が目的値より大きいなら中央より前を，小さいなら中央より後ろを調べる．
4. 探索範囲が無くなったらそこを挿入位置にする．

このアルゴリズムのオーダーは O(logn) となる．

1.2.6 (4)

ひっかけ問題注意先の問題では二分探索について尋ねられているが，24 行目の処理でデータ列に順番にアクセスしているため，insert のオーダーは O(n) となる．delete はデータ列の先頭にのみアクセスするため O(1).

1.2.7 (6)

問題の解決法としては front や rear が配列 A の末端に到達したら先頭に移動するようにしてループ構造にすることと，データ列の操作時に値を前に詰めるようにすることが考えられる．ただし，**delete 実行時にデータ列を詰めるように変更するとオーダーが O(n) に変わる**ため，後者の方法は insert の実行時に行う必要がある．

2 計算機システムとシステムプログラム

2.1 回答

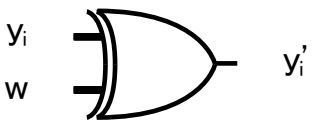
(1-1)

$$
\begin{aligned}
i &= 01011010 &&= 90 \\
j &= 11011000 &&= -40 \\
k &= 00110010 &&= 50 \\
m &= 10000010 &&= -126 \\
n &= 01111100 &&= 126 \\
d &= 00011001 &&= 0.390625 \\
e &= 00110010 &&= 1.125
\end{aligned}
$$

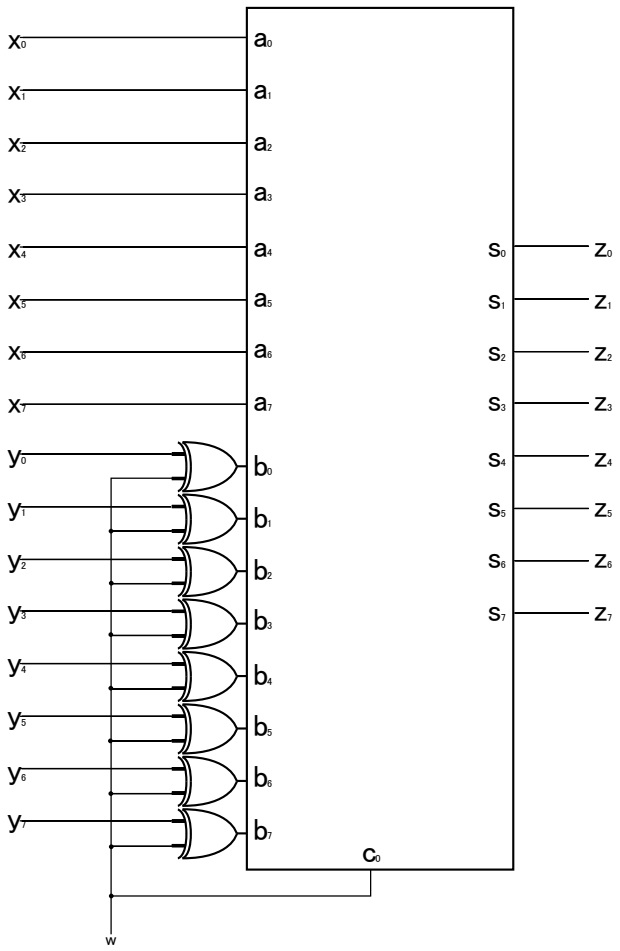
(1-2)

2 の補数系表現を用いると，負数の表現が容易になるため減算の処理を加算回路を用いて行えるようになる．また正数と負数の扱いを区別する必要も無くなる．

(1-3)



(1-4)



(2-1)

a：実行可能
b：実行
c：相互実行

(2-2)

X がテストを実行した後，セットを実行する前に Y がテストを実行した場合，Z が実行可能状態だと認識してしまうため，X と Y が Z に対して相互実行をしてしまう．このため，テストとセットは不可分である必要がある．

2.2 解説

2.2.1 2 の補数

2 の補数とはビット列における負数の表現法であり，ある数値に対して各ビットを反転させて 1 を加えることで得ることができる．例として，8 ビット表現における 50 = “00110010″ の 2 の補

数は $-50 =$ “11001110″ となる．これらの値を加算すると $0 =$ “(1)00000000″ となる．このとき，最上位ビットからの桁上がりは無視される．

2.2.2 浮動小数点数

浮動小数点数とはビット列における小数の表現法であり，$(-1)^s \times (1+m) \times 2^{e-a}$ の形式で表される．実際のビット列では符号 s，指数部 e，仮数部 m の形で得られる．仮数部について，最上位ビットは常に 1 であるとして省略され，以降 $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \cdots$ と続く．また，a の値は任意の整数であり，今回は 3 となる．今回の問題ではビット列は次のように対応する．表 2 での値は $(-1)^0 \times (1+\frac{1}{2}+\frac{1}{4}) \times 2^{3-3} = 1.75$

表 2: 8 ビット浮動小数点数

s	e			m			
0	1	0	1	1	1	0	0

となる．またビット数は有限のため，ビット列で表現する桁以降の値は代入および計算時に切り捨てられる．

2.2.3 (1-1)

i，j は代入された整数値をそのままビット列に変換すればよい．k は計算結果の 50 をビット列に変換する．k，m の計算結果は 130，-130 だが，オーバーフローが起こるために実際の計算結果は-126，126 となる．
d について，正数なので符号は 0．0.4 からビット表現で切り出せる最大値は $0.25 = \frac{1}{4}$ のため指数部は 1．仮数部について，0.4 を超えない最大の表現は $\frac{1}{4} + \frac{1}{8} + \frac{1}{64} = 0.390625$ となる．e について，0.8 のビット列は “00101001″ $= 0.7825$ となる．ここで 2 つの値を加算する時に最下位ビットが切り捨てられるため，計算結果は $e =$ “00110010″ $= 1.125$ となる．

2.2.4 (1-3)

1 の補数とは各ビットを反転させた表現である．xor 回路は $f = g\overline{h} + \overline{g}h$ で示されるため，$h = 1$ の下では $f = g$，つまり not 回路として使用できる．このため，g に y_i を，h に w を与えてやれば出力は y_i' となる．

2.2.5 (1-4)

(1-3) で用いた回路によって 1 の補数が得られる．2 の補数を得るためには 1 の補数に 1 を加算すればよい．このため，桁上げ入力 c_0 に w を与えてやればよい．結果の回路図は回答のようになる．

3 離散構造

3.1 回答

(1-1)

R_2と R_5 が該当.
R_2 について，$S = (x, x)|x \in V$ を含むため反射律を満たす.
対称律は定義より自明である.
$(x, y), (y, z) \in R_2$ の時，y が交差点となり，x と z を含む有向閉路が存在するため $(x, z) \in R_2$
推移律も満たす.
R_5 について，$S = (x, x)|x \in V$ を含むため反射律を満たす.
対称律は定義より自明である.
$(w, x), (x, y) \in R_5$ の時，w,x,y からある z への有向片が存在するため $(w, y) \in R_5$ となり推移律も満たす.

(1-2)

R_3 が該当.
反射律は定義より自明である.
$(x, y) \in R_3$ とは，“s から y へ到達するには x を通る必要がある” という命題を示し，“x より先に y へ到達することは不可能である” ことを示す．このため $(y, x) \in R_3$ は成り立たず，反対称律を満たす.
$(x, y), (y, z) \in R_3$ の時，z へ到達するために y を通過し，そのために x を通過する必要がある．よって $(x, z) \in R_3$ が成り立つため推移率を満たす.
また，R_3 は比較不可能な頂点が存在するため，全順序関係ではない．したがって半順序関係となる.

(1-3)

R_1, R_3, R_4 が該当する.

(2-1)

$(a) : r(h, x, y) \rightarrow m(h, a, y)$
$(b) : m(h - 1, x, y) \rightarrow m(h, a, y)$
$(c) : m(h, x, y) \rightarrow (m(h - 1, x, y) \vee r(h, x, y))$

(2-2)

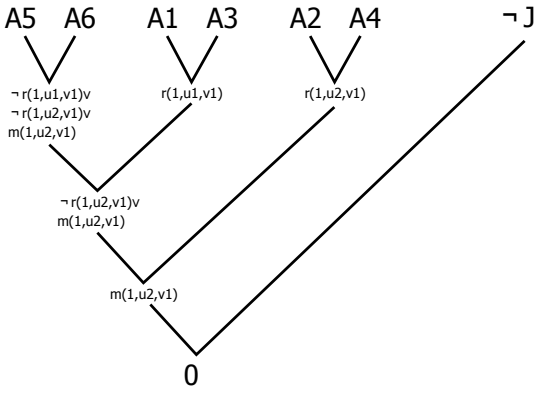
$P = CX1 \wedge CX2 \wedge CX3 \wedge CX4 \wedge CX5 \wedge CY1 \wedge CY2 \wedge CY3 \wedge CY4 \wedge CY5 \wedge$
$CZ1 \wedge CZ2 \wedge \neg H$

(2-3-1)

$(d) : \neg r(1, u_1, v_2)$
$(e) : \neg r(1, u_2, v_2)$
$(f) : \neg r(1, u_1, v_1) \vee m(1, u_1, v_1) \vee m(1, u_2, v_1)$
$(g) : \neg m(1, u_1, v_2) \vee r(1, u_1, v_2)$
$(h) : \neg m(1, u_2, v_2) \vee r(1, u_2, v_2)$

(2-3-2)

$J = m(1, u_2, v_1)$ とする．
$A1 \wedge A2 \wedge A3 \wedge A4 \wedge A4 \wedge A5 \wedge A6 \wedge A7 \wedge A8 \rightarrow J$ を示す．
この式は $\neg(A1 \wedge A2 \wedge A3 \wedge A4 \wedge A4 \wedge A5 \wedge A6 \wedge A7 \wedge A8 \wedge \neg J$ と同値である．
導出原理によって，$(A1 \wedge A2 \wedge A3 \wedge A4 \wedge A4 \wedge A5 \wedge A6 \wedge A7 \wedge A8 \wedge \neg J$ が恒偽であることを示す．導出原理は次の図のようになる．



3.2 解説

3.2.1 ～律について

反射律とは $\forall a \in X, aRa$，つまり全要素が自分自身と関係 R を持つことを指す．
対称律とは $\forall a, b \in X, aRb \rightarrow bRa$，つまり関係 R の前後要素が可換であることを指す．
反対称律とは $\forall a, b \in X, aRb, bRa \rightarrow a = b$，つまり関係 R の前後要素が等しくないならば不可換であることを指す．
推移律とは $\forall a, b, c \in X, aRb, bRc \rightarrow aRc$，つまり関係 R は要素を挟んで成り立つことを指す．

3.2.2 同値関係

同値関係とは反射律，対称律，推移律をすべて満たすことを指す．

3.2.3 順序関係

順序関係とは反射律，反対称律，推移律をすべて満たすことを指す．また，すべての要素について順序の比較が成り立つ場合は全順序関係，そうでない場合は半順序関係となる．例えば，自然数の大小は全順序関係であり，自然数の約数は半順序関係である．

3.2.4 (1-1)

R_1，R_3，R_4 はどれも対称律を満たさない．

3.2.5 (1-2)

R_1，R_2，R_4，R_5 はどれも反対称律を満たさない．

3.2.6 (1-3)

閉路がなくなる場合，R_1と R_4 は可換でなくなるため反対称律を満たすようになり，半順序関係となる．

3.2.7 (2-2)

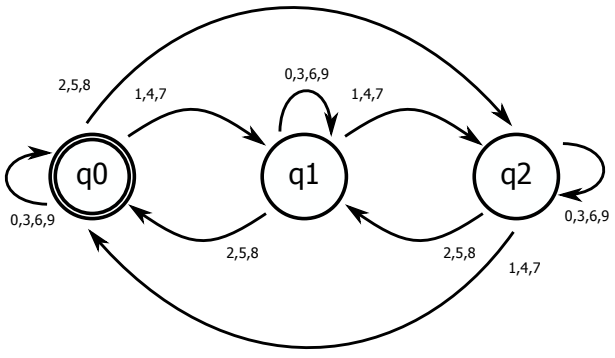
$\forall(x,y) \in ((X \times Y) \backslash M)(\exists(a,b) \in M(x \leq_y a \vee y \leq_x b))$ とは，マッチングでない全ての関係 (x,y) に対して，その x および y から見て上位にある $a \in X$ と $b \in Y$ のマッチングが存在することをさす．言い換えると，“互いにマッチングし直したほうが良い組み合わせが存在しない” ことを指す．条件下で常に H が満たされていることを証明するには，条件下で $\neg H$ が満たせないことを証明すればよい．

4 計算理論

4.1 回答

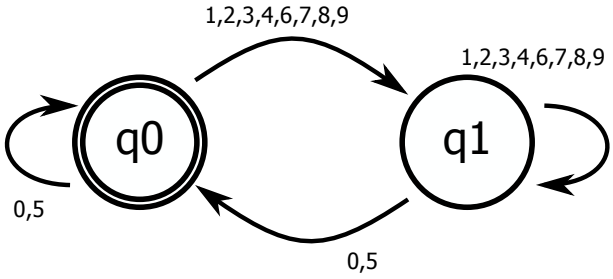
(1-1)

同値類で考えて，3 で割った時の余りについての状態を構築すればよい．具体的には余りが 0 の受理状態，余りが 1 の状態，余りが 2 の状態を設定し，入力値 x が与えられたとき，現在の余りを y とすると，2y+x mod 3 の状態に遷移すればよい．オートマトン B は次の図のようになる．



(1-2)

] オートマトン C は次の図のようになる．



これと先述のオートマトン B の状態の積を持つオートマトン D を作り，B と C を同時に受理する状態を受理状態とすればよい．

(1-3)

(1-2) 同様に B と C の積オートマトンを作り，B か C のいずれかが受理する状態をすべて受理状態をすればよい．

(2-1)

$G_1 = (\{S\}, \{c\}, \{S \to cS, S \to c\}, S)$

(2-2)

$G_2 = (\{S, A, B\}, \{a, b, c\}, \{S \to AB, A \to aAb, A \to ab, B \to cB, B \to c\}, S)$
$G_3 = (\{S, A, B\}, \{a, b, c\}, \{S \to AB, A \to aA, A \to a, B \to bBc, B \to bc\}, S)$
G_2において，$S \to AB$ を 1 回，$A \to aAb$ を $n-1$ 回，$A \to ab$ を 1 回，$B \to cB$ を $m-1$ 回，$B \to c$ を 1 回生成すれば $L_2 = a^n b^n c^m$ となる．
同様に G_3において，$S \to AB$ を 1 回，$A \to aA$ を $m-1$ 回，$A \to a$ を 1 回，$B \to bBc$ を $n-1$ 回，$B \to bc$ を 1 回生成すれば $L_3 = a^3 b^n c^n$ となる．

(2-3)

L_2と L_3 の積演算は $L_4 = \{a^n b^n c^n | n \geq 1\}$ である．L_2と L_3は共に文脈自由言語だが，L_4は文脈自由言語ではない．よって，文脈自由言語の集合は積演算について閉じていない．

4.2 解説

4.2.1 (1-2)

考え方は (1-1) 同様だが，5 で割り切れるかは 1 の位のみで決定できるため，状態数は “5 の倍数” と “そうでない” の 2 つで良い．また，どちらの状態についても，0,5 が入力されたら前者に，それ以外が入力されたら後者に遷移する．
15 で割り切れるということは，3 で割り切れてかつ 5 でも割り切れる状態を示すため，2 つのオートマトン B と C を同時に操作するようなオートマトンを設計すればよい．つまり，オートマトンの積をとる．

4.2.2 (1-3)

(1-2) とは逆に，今度はオートマトンの和をとればよい.

4.2.3 (2)

文法の生成規則においては，$m, n \geq 1$ 等の条件を必ず満たすように注意すること．複雑な文法を設計した場合によく見落とす.