令 和 2 年 度

名古屋大学大学院情報学研究科
情報システム学専攻
入学試験問題（専門）

令和元年８月７日

注意事項

１．試験開始の合図があるまでは，この問題冊子を開いてはならない．

２．試験終了まで退出できない．

３．外国人留学生の志願者は，日本語と日本語以外の１言語間の辞書１冊に限り使用してよい．電子辞書の持ち込みは認めない．

４．日本語または英語で解答すること．

５．問題冊子，解答用紙３枚，草稿用紙３枚が配布されていることを確認すること．

６．問題は（1）確率・統計，（2）プログラミング，（3）計算機理論，（4）ハードウェア，（5）ソフトウェア，の５科目がある．このうち３科目を選択して解答すること．なお，選択した科目名を解答用紙の指定欄に記入すること．

７．試験開始の合図の後，解答用紙のホチキスをはずすこと．

８．全ての解答用紙の所定の欄に受験番号を必ず記入すること．解答用紙に受験者の氏名を記入してはならない．

９．解答用紙に書ききれない場合は，裏面を使用してもよい．ただし，裏面を使用した場合は，その旨を解答用紙表面右下に明記すること．

１０．解答用紙は試験終了後に３枚とも提出すること．

１１．問題冊子，草稿用紙は試験終了後に持ち帰ること．

確率・統計

解の導出過程も書くこと．

[1] 表 (head) が出る確率と裏 (tail) が出る確率がともに 1/2 であるコイン (coin) 1 枚を連続して投げ，『裏表表』，『表表表』のような事前に決めた系列 (sequence) が出現したら終了するものとする．このとき，以下の問いに答えよ．

(1)『裏表表』という系列に決めた場合に 3 回で終了する確率，すなわち，1 回目が裏，2 回目が表，3 回目が表となる確率を求めよ．
(2)『裏表表』という系列に決めた場合にちょうど 4 回で終了する確率を求めよ．
(3)『表表表』という系列に決めた場合にちょうど 4 回で終了する確率を求めよ．
(4) 初めて表が出るまでにコインを投げる回数の期待値を $E(\text{表})$ と書くとき $E(\text{表})$ を求めよ．
(5) 初めて 2 回続けて表が出るまでにコインを投げる回数の期待値を $E(\text{表表})$ と書くとき $E(\text{表表})$ を求めよ．なお，初めて表が出た直後に表が出た場合はその時点で 2 回続けて表が出たことになり，裏が出た場合はその後 2 回続けて表が出るまでにコインを投げる回数の期待値が $E(\text{表表})$ と等しくなることから以下の式が成立する．

$$E(\text{表表}) = \frac{E(\text{表}) + 1}{2} + \frac{E(\text{表}) + 1 + E(\text{表表})}{2}$$

(6)『表表表』という系列に決めた場合のコインを投げる回数の期待値を求めよ．

[2] 確率変数 X, Y の同時確率密度関数 $f_{X,Y}(x, y)$ が次式で与えられている．ただし，a は定数である．このとき，以下の問いに答えよ．

$$f_{X,Y}(x, y) = \begin{cases} a(x^2 - y^2)e^{-x} & (0 \leq x,\ -x \leq y \leq x) \\ 0 & (\text{otherwise}) \end{cases}$$

(1) $f_{X,Y}(x, y)$ が最大となる x，y の値を求めよ．
(2) 周辺密度関数 $f_X(x)$ を定数 a を用いて表せ．
(3) 定数 a の値を求めよ．
(4) 確率変数 X の期待値 μ_x を求めよ．

Translation of technical terms

- 確率: probability
- 期待値: expectation
- 確率変数: random variable
- 同時確率密度関数: joint probability density function
- 定数: constant
- 周辺密度関数: marginal probability density function

プログラミング

プログラムAは，ハッシュ表を用いて正整数の集合を操作するためのC言語プログラムである．ハッシュ値ごとの正整数のリスト構造を，3つの配列element, next，hfirstを用いて実現している．element配列は，正整数の集合を格納するための配列である．next配列は，リストの次の要素が格納されているelement配列の添え字を格納するための配列である．hfirst配列はハッシュ表のための配列であり，hfirst[h]はハッシュ値hを持つ正整数のリストの先頭の要素が格納されているelement配列の添え字を表す．ここで，hfirst[h]が-1のときは，ハッシュ値hを持つ正整数のリストが空であることを表す．

hashfunc関数は，ハッシュ関数を表す．search関数，insert関数，delete関数は，それぞれelement配列に対する正整数の探索，挿入，削除を行う関数である．initarrays関数は，hfirst配列，element配列，next配列をそれぞれ初期化する関数である．outputarrays関数は，hfirst配列，element配列，next配列をそれぞれ標準出力に出力する関数である．

プログラムAについて，以下の問いに答えよ．

(1) プログラムAを実行した際に，74行目のoutputarrays関数の呼び出しにより標準出力に出力される文字列を書け．

(2) プログラムAの65行目の`int data[] = {1,2,3,5,7};`を`int data[] = {1,2,18,19,20};`に置き換えて実行した際に，74行目のoutputarrays関数の呼び出しにより標準出力に出力される文字列を書け．

(3) data配列に格納される正整数の集合がどのような特徴を持つときに，search関数の実行時間が集合のサイズに対して長くなるか答えよ．

(4) （ア）と（イ）を適切な式で埋めて，delete関数を完成させよ．

(5) プログラムAを実行した際に，78行目のoutputarrays関数の呼び出しにより標準出力に出力される文字列を書け．ただし，プログラムAは(2)の置き換えを行っていないものとする．

Translation of technical terms:

プログラム	program	関数	function
ハッシュ表	hash table	ハッシュ関数	hash function
正整数	positive integer	探索	search
集合	set	挿入	insertion
C言語	C programming language	削除	deletion
ハッシュ値	hash value	初期化	initialization
リスト	list	標準出力	standard output
構造	structure	出力	output
配列	array	実行	execution
格納する	store	呼び出し	call
要素	element	文字列	character string
添え字	index	実行時間	execution time
空	empty	式	expression

プログラムA

```
1  #include <stdio.h>
2  #define MAXSIZE         1000
3  #define P                 17
4  #define SENTINEL          -1
5  #define NOTFOUND          -2
6  int hfirst[P];
7  int element[MAXSIZE];
8  int next[MAXSIZE];
9  int avail=-1;
10 int maxnode=0;
11 int hashfunc(int data){
12    return data%P;
13 }
14 int search(int h, int data){
15    int pred=-1;
16    if (hfirst[h]==-1) return NOTFOUND;
17    if (element[hfirst[h]]==data) return pred;
18    pred=hfirst[h];
19    while (next[pred]!=SENTINEL){
20       if (element[next[pred]]==data) return pred;
21       pred=next[pred];
22    }
23    return NOTFOUND;
24 }
25 void insert(int h, int data){
26    int u;
27    if (avail!=-1) {
28       u=avail;
29       avail=next[avail];
30    } else {
31       u=maxnode;
32       maxnode=maxnode+1;
33    }
34    element[u]=data;
35    next[u]=hfirst[h];
36    hfirst[h]=u;
37 }
38 void delete(int h, int pred){
39    int u;
40    if (pred!=-1){
41       u=next[pred];
42       next[pred]=[ (ア) ];
43    } else {
44       u=hfirst[h];
```

```
45          hfirst[h]= (イ) ;
46      }
47      next[u]=avail; avail=u;
48  }
49  void initarrays(){
50      int i;
51      for (i=0; i<P; i++) {hfirst[i]=-1;}
52      for (i=0; i<MAXSIZE; i++) {element[i]=0;}
53      for (i=0; i<MAXSIZE; i++) {next[i]=SENTINEL;}
54  }
55  void outputarrays(int maxnode){
56      int i;
57      for (i=0; i<P; i++) {printf("%d,",hfirst[i]);}
58      printf("\n");
59      for (i=0; i<maxnode; i++) {printf("%d,",element[i]);}
60      printf("\n");
61      for (i=0; i<maxnode; i++) {printf("%d,",next[i]);}
62      printf("\n");
63  }
64  int main(){
65      int data[] = {1,2,3,5,7};
66      int h;
67      int i;
68      int pred;
69      initarrays();
70      for (i=0; i<5; i++){
71          h=hashfunc(data[i]);
72          if (search(h,data[i])==NOTFOUND) insert(h,data[i]);
73      }
74      outputarrays(maxnode);
75      h=hashfunc(1);
76      pred=search(h,1);
77      if (pred!=NOTFOUND) delete(h,pred);
78      outputarrays(maxnode);
79      return 0;
80  }
```

計算機理論

[1] 記号列 w の長さを $||w||$ で表す．また w における記号 a の出現回数を $||w||_a$ で表す．例えば，$||aaba|| = 4$, $||aaba||_a = 3$, $||aaba||_b = 1$ である．以下では，記号の全体集合を $T = \{a, b\}$ とする．記号列から有理数への関数 c を，任意の記号列 $w \in T^*$ に対して，

$$c(w) = (\frac{1}{2})^{||w||_a} \cdot (\frac{1}{3})^{||w||_b}$$

と定義し，この関数 c の定義域を拡張して，任意の言語 $L \subseteq T^*$ に対して，

$$c(L) = \sum_{w \in L} c(w)$$

と定義する．例えば，$c(ab) = \frac{1}{6}$, $c(\{\varepsilon, a, aa\}) = 1 + \frac{1}{2} + \frac{1}{4} = \frac{7}{4}$ である．

(1) 長さが 2 以上であり，右から 2 番目の記号が a であるような記号列すべてからなる言語 L_1 を受理する状態数 4 の決定性有限オートマトンの状態遷移図を示せ．初期状態，受理状態を明記すること．

(2) 正規表現 a^* が表す言語 L_2 に対して $c(L_2)$ は収束する．その値を求めよ．

(3) $c(R) = c(\{a^n b^n \mid n \geq 0\})$ を満たす正規言語 $R \subseteq T^*$ を表す正規表現を 1 つ書け．

(4) 非負整数 n が与えられたとする．$c(T^n)$ を，n を用いた式で表せ．二項定理を利用してもよい．

(5) 小問 (1) の言語 L_1 に対して，$c(L_1)$ を求めよ．小問 (4) の結果を利用してもよい．

Translation of technical terms

記号列	string	初期状態	initial state
出現回数	number of occurrences	受理状態	accepting state
関数	function	正規表現	regular expression
定義域	domain	収束する	converge
言語	language	正規言語	regular language
決定性有限オートマトン	deterministic finite automaton	非負整数	nonnegative integer
状態遷移図	state transition diagram	二項定理	binomial theorem

[2] m, n を正の整数として，次の命題論理式 A_n^m, B_n^m, C_n^m を考える．

$$A_n^m = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{n} p_{i,j}$$

$$B_n^m = \bigwedge_{i=1}^{m} \bigwedge_{1 \le j < k \le n} (\neg p_{i,j} \vee \neg p_{i,k})$$

$$C_n^m = \bigwedge_{j=1}^{n} \bigwedge_{1 \le i < \ell \le m} (p_{i,j} \supset \neg p_{\ell,j})$$

ここで，各 $p_{i,j}$ $(1 \le i \le m, 1 \le j \le n)$ は命題変数を表す．論理結合子の結合の強さは，強いものから順に否定 $(\neg)$，論理積 $(\wedge)$，論理和 $(\vee)$，含意 $(\supset)$ とする．$\bigwedge_{i=1}^{n} A_i$ は $A_1 \wedge \cdots \wedge A_n$ の略記である．同じように，$\bigvee_{i=1}^{n} A_i$ は $A_1 \vee \cdots \vee A_n$ の略記である．命題変数またはその否定をリテラルという．リテラルを論理和で結合した論理式を節という．以下の問いに答えよ．

(1) $A_5^1 \wedge B_5^1$ を真にする命題変数への真理値割り当てをすべて求めよ．

(2) $A_n^m \wedge B_n^m$ に含まれる節の個数を，m と n を使って表せ．

(3) $A_2^3 \wedge C_2^3$ を同値な連言標準形に変換せよ．

(4) 導出原理を用いて，$A_2^3 \wedge C_2^3$ が充足不能であることを示せ．

(5) $p_{i,j}$ $(1 \le i \le m, 1 \le j \le n)$ が真のとき，整数変数 x_i が整数値 j を取ると考える．このとき，A_n^m は各整数変数 x_i $(1 \le i \le m)$ が集合 $\{1, 2, \ldots, n\}$ 中の値を少なくとも一つ取ることを表す制約である．以下の問いに答えよ．

(a) B_n^m が表す制約を説明せよ．

(b) C_n^m が表す制約を説明せよ．

Translation of technical terms

命題論理式	propositional formula	真理値割り当て	truth assignment
命題変数	propositional variable	同値	equivalent
論理結合子	logical connective	連言標準形	conjunctive normal form
否定	negation	導出原理	resolution principle
論理積	conjunction	充足不能	unsatisfiable
論理和	disjunction	整数変数	integer variable
含意	implication	整数値	integer value
リテラル	literal	集合	set
節	clause	制約	constraint
真	true		

ハードウェア

[1] プロセッサの汎用(はんよう)レジスタ数は16個や32個が一般的である。汎用レジスタの数を多くした場合のデメリットを3種類，それぞれ50文字（英語の場合，20語）以下で述べよ。

[2] 小数点以下2ビットで全6ビットの2進数を用いて，$(7.5)_{10} + (-1.75)_{10}$ の加算(かさん)を行い，結果を2進数で示せ。なお，負数は2の補数(ほすう)を用いて表すものとする。2の補数を用いて計算を行ったことが分かるように計算の過程も明示すること。

[3] 1ビットの入力Xと，1ビットの出力Zを有する同期式順序回路(どうきしきじゅんじょかいろ)を考える。この回路は，現在のクロック・サイクルを含む直近の4クロック・サイクルにおいて，Xに2回以上1が入力された場合にはZに1を出力し，それ以外の場合にはZに0を出力する。なお，回路の動作開始時には，過去のクロック・サイクルにおいてXには0が入力されていたとして動作する。この回路の動作例を下表に示す。このとき以下の問いに答えよ。

クロック・サイクル	1	2	3	4	5	6	7	8	9	10
入力 X	1	1	0	0	0	1	0	1	0	0
出力 Z	0	1	1	1	0	0	0	1	1	0

(1) この回路の状態遷移(じょうたいせんい)と出力を，右表の書式で示せ。なお，右表のように，現在のクロック・サイクルを含まない直近の3クロック・サイクルの入力Xの値を現状態の符号(げんじょうたいのふごう)（左ビットが古い）として用いよ。

(2) この回路の状態数(じょうたいすう)を最小化し，状態遷移と出力を，右表の書式で示せ。状態数を最小化した手順も明示すること。

現状態 (符号)	入力 X=0		入力 X=1	
	次状態	出力	次状態	出力
S_0 (000)				
S_1 (001)				
S_2 (010)				
S_3 (011)				
S_4 (100)				
S_5 (101)				
S_6 (110)				
S_7 (111)				

(3) この回路を必要最小個数のDフリップフロップを用いて実現する。各Dフリップフロップの入力を与える論理関数(ろんりかんすう)の最小積和形表現(さいしょうせきわけいひょうげん)を求めよ。状態割り当て(じょうたいわりあて)も明示すること。

(4) 出力Zを表す論理関数の最小積和形表現を求めよ。

[4] パイプライン処理(しょり)プロセッサを考える。パイプラインのステージ数は5段であり，命令(めいれい)は4種類に分類される。各命令のステージ毎の処理内容を以下に示す。なお，表中の「N/A」は何も実行されないステージであることを示す。汎用レジスタはr0からr15の16個持つ。

	IF	ID	EX	MEM	WB
算術論理命令	命令フェッチ	デコード及びレジスタリード	演算	N/A	レジスタライト
ロード命令			アドレス計算	データロード	
ストア命令				データストア	N/A
分岐命令			比較	プログラムカウンタ更新	N/A

上記のパイプライン設計を元に3種類のプロセッサを実装した。各プロセッサのメモリは1サイクルでアクセス可能である。

プロセッサ1は命令(めいれい)メモリとデータメモリが単一のメモリであり，同一のクロック・サイクル中では命令かデータのどちらかのみアクセス可能である。命令とデータのアクセスが同時に発生した場合はデータへのアクセスが優先される。プロセッサ2は命令メモリとデータメモリが独立であり，同一のクロック・サイクルで命令とデータの両方にアクセス可能である。プロセッサ3はプロセッサ2に対してデータハザードを解消するフォワーディングが実装されており，あるステージから別のステージに直接データを渡すことが可能である。

	メモリ	フォワーディング
プロセッサ1	単一	なし
プロセッサ2	命令・データ独立	なし
プロセッサ3	命令・データ独立	あり

これらのプロセッサで以下の命令列（プログラム1）を実行する。なお，#以降は各行の命令を説明したコメントである。

プログラム1

```
1  L1:lw    r0, 0(r5)     # r0にアドレス[r5+0]のデータをロードする
2     lw    r1, 0(r6)     # r1にアドレス[r6+0]のデータをロードする
3     addi  r5, r5, 4     # r5 = r5 + 4
4     sll   r0, r0, 1     # r0 = r0 << 1
5     addi  r6, r6, 4     # r6 = r6 + 4
6     sll   r1, r1, 2     # r1 = r1 << 2
7     add   r2, r0, r1    # r2 = r0 + r1
8     lw    r3, 0(r7)     # r3にアドレス[r7+0]のデータをロードする
9     add   r4, r3, r2    # r4 = r3 + r2
10    bne   r5, r8, L1    # r5とr8が等しくなければL1に分岐
11    sw    r4, 0(r9)     # アドレス[r9+0]にr4のデータをストアする
```

これに関して以下の問いに答えよ。

(1) プロセッサ1でプログラム1の1行目から7行目までを実行した場合のクロック・サイクル毎のパイプライン処理の状況を以下の書式で示し，命令の実行が完了するまでのクロック・サイクル数を答えよ。

命令	1	2	3	4	5	6	...
lw r0, 0(r5)	IF	ID	EX	MEM	WB		
lw r1, 0(r6)		IF	ID	EX	MEM	WB	
...							

(2) プロセッサ2でプログラム1の1行目から7行目までを実行した場合のクロック・サイクル毎のパイプライン処理の状況を (1) と同様の書式で示し，命令の実行が完了するまでのクロック・サイクル数を答えよ。

(3) プロセッサ2でプログラム1の1行目から7行目までを実行した場合にパイプライン・ストールが発生する全ての命令の対とその理由について説明せよ。また，発生するハザードに対してどの様なフォワーディングを行えば低減されるか答えよ。

(4) プロセッサ3でプログラム1の1行目から9行目までを実行した場合のクロック・サイクル毎のパイプライン処理の状況を (1) と同様の書式で示し，命令の実行が完了するまでのクロック・サイクル数を答えよ。

(5) プロセッサ3でプログラム1の10行目以降を実行した場合に発生するハザードについて，名称と発生する理由を述べよ。また，そのハザードを低減する機構の名称を述べよ。

Translation of technical terms

汎用レジスタ	general purpose register	論理関数	logic function
		最小積和形表現	minimal sum-of-products form
ビット	bit		
加算	addition	状態割り当て	state assignment
2の補数	two's complement	パイプライン処理	pipeline processing
	binary number	命令	instruction
同期式順序回路	synchronous sequential circuit	メモリ	memory
		命令メモリ	instruction memory
クロック・サイクル	clock cycle	データメモリ	data memory
状態遷移	state transition	プログラムカウンタ	program counter
現状態の符号	current state code	データハザード	data hazard
状態数	the number of states	フォワーディング	forwarding

ソフトウェア

[1] 連結無向グラフについて考える．ここで，各頂点の間に辺は高々1本しか存在せず，どの頂点も自分自身との間に辺を持たないとする．

(1) 頂点数 n の無向グラフのうち辺数 m が最大となるグラフについて，m を n の多項式で表せ．

無向グラフ $G=(V,E)$ に対して，関数 $w: V\times V \to \mathbb{N}\cup\{\infty\}$ により辺の長さを与える．ただし，任意の頂点 $v,u\in V$ について，$w(v,u)=w(u,v)$ とし，$(v,u)\notin E$ のときは $w(v,u)=\infty$ とする．

図 1 は無向グラフ $G_1=(\{v_1,v_2,\ldots,v_{14}\},\{(v_1,v_2),(v_1,v_5),\ldots,(v_{13},v_{14})\})$ および関数 w_1 で与えられる辺の長さを図示したものであり，辺 (v_i,v_j) に付記された数はその辺の長さ $w_1(v_i,v_j)$ を表している．

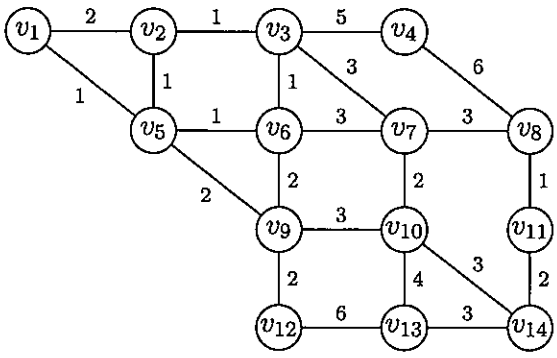

図 1: 無向グラフ G_1 および関数 w_1 で与えられる辺の長さ

無向グラフの 2 頂点間の最短距離とは，その 2 頂点を端点とするグラフ上の路の長さ（すなわち，路に含まれる辺の長さの総和）の最小の値である．例えば，図 1 の頂点 v_1 と v_6 の間の最短距離は 2 である．なお，頂点 v と v 自身の間の最短距離は 0 とする．

(2) 図 1 の G_1 と w_1 について，v_1 と v_{11} の間，v_{11} と v_{12} の間それぞれの最短距離を示せ．

図 2 に示すアルゴリズム 1 はダイクストラ法を形式化したものである．なお，アルゴリズム 1 の手続きに現れる演算子 “$\setminus$” は差集合演算である．

(3) アルゴリズム 1 の 3.(i) で $D(v)$ の値が最小となる頂点 v が複数存在した場合には，添え字が最も小さいものを選択することとする．G_1，w_1，v_{11} を入力としてアルゴリズム 1 を実行したとき，3.(i) において最後に選択される頂点はどれであるか答えよ．

(4) 無向グラフ $G=(V,E)$，G の辺の長さを表す関数 w，頂点 $v_0\in V$ を入力としてアルゴリズム 1 を実行した場合の以下の (a)〜(e) それぞれの処理の漸近的最大時間計算量を G の頂点の数 n，辺の数 m に関する O 記法で示せ．ただし，それぞれの計算量を表す O 記法に n，m の両方が出現するとは限らない．

(a) 1. の処理．

(b) 3. の繰り返しが終了するまでに行われる 3.(i) のすべての処理．

(c) 3. の繰り返しが終了するまでに行われる 3.(ii) のすべての処理．

アルゴリズム 1

入力 無向グラフ $G = (V, E)$，G の辺の長さを表す関数 w，G の頂点 v_0

出力 頂点 v に対して，v_0 と v の間の最短距離を返す関数 D

手続き 以下を順に実行し，終了後にその時点での D を出力する．

1. $D(v_0)$ を 0 に，各 $v \in V \setminus \{v_0\}$ については $D(v)$ を $w(v_0, v)$ の値にする．
2. U を $V \setminus \{v_0\}$ とする．
3. U が空集合になるまで以下の処理を繰り返す．
   (i) U に含まれる頂点のうち，$D(v)$ が ∞ 以外の値を取る頂点のうちで，その値が最小である頂点 v を選択する．
   (ii) U から v を除去する．
   (iii) $w(v, u) \neq \infty$ である各頂点 $u \in U$ について，「$D(u) = \infty$ である，もしくは $D(v) + w(v, u)$ が $D(u)$ よりも小さい(ア)」ならば，$D(u)$ の値を $D(v) + w(v, u)$ に更新する．

図 2: アルゴリズム 1

(d) 3. の繰り返しが終了するまでに行われる下線部（ア）のすべての処理．

(e) アルゴリズム 1 の全体の処理．

ただし，計算量を考えるにあたり，次にあげる条件 A〜F すべてを仮定する．

A. 各頂点 v_i について $D(v_i)$ の値の参照および更新はそれぞれ $O(1)$ の時間で実行できる．

B. 各頂点 v_i, v_j について，$w(v_i, v_j)$ の参照は $O(1)$ の時間で実行できる．

C. 整数上の比較演算は $O(1)$ の時間で実行できる．

D. 2. の処理は空集合 U に $V \setminus \{v_0\}$ の要素すべてを順に挿入することで実現される．

E. U に要素を挿入する処理，および U が空集合であるかどうかを判定する処理はそれぞれ $O(1)$ で実行できる．

F. U の要素数が k であるとき，U から $D(v)$ の値が最小になる頂点を求める処理，および U から指定した要素を除去する処理はそれぞれ $O(k)$ の時間で実行できる．

Translation of technical terms

連結無向グラフ	connected undirected graph	手続き	procedure
無向グラフ	undirected graph	演算子	operator
頂点	vertex	差集合	set difference
辺	edge	添え字	index
多項式	polynomial	空集合	empty set
最短距離	shortest distance	処理	process
端点	end vertex	漸近的最大時間計算量	asymptotic worst case time complexity
路	path		
アルゴリズム	algorithm	O 記法	big O notation
ダイクストラ法	Dijkstra's algorithm	比較演算	comparison operation

[2] データベースに対して複数のプロセスが読書きするプログラムを考える。以下の (i)～(v) の仕様に基づいて，このプログラムを C 言語で作成するものとする。

(i) データベースからデータを読み出す reader 関数を実行する読出しプロセスと，データベースにデータを書き込む writer 関数を実行する書込みプロセスを用意する。

(ii) データの読出し要求，または書込み要求が発生すると，それぞれの要求を処理する読出しプロセス，または書込みプロセスを実行可能にする。例えば，1 つのデータ書込み要求が完了する前に，他の書込み要求が発生した場合には，2 つの書込みプロセスが実行可能状態になる。

(iii) reader 関数および writer 関数内でデータベースにアクセスしている間は，プロセス間同期の仕組みによって，他のプロセスがデータベースにアクセスできないよう排他制御する。

(iv) プロセス間同期を実現するためにセマフォを使用する。semaphore 型のセマフォ A に対して，以下の操作ができるものとする。

- down(&A)：セマフォ A の値が 0 でなければ，値を 1 だけ減算する。0 の場合には，呼び出したプロセスをセマフォ A に対する待ち状態にする。これらの処理は，アトミックに実行される。
- up(&A)：セマフォ A に対する待ち状態のプロセスがあれば，もっとも早く待ち状態になったプロセスを実行可能状態にする。待ち状態のプロセスがない場合には，セマフォ A の値を 1 だけ加算する。これらの処理は，アトミックに実行される。

(v) reader 関数および writer 関数内のクリティカルセクションは，できる限り短くする。

(1) 上記の (i)～(v) を満たすようプログラム 1 を記述した。(a)～(f) の空欄に適切なセマフォ操作を埋めてプログラムを完成させよ。なお，空欄に何も埋める必要がない場合には，nop と答えよ。

(2) プログラム 1 の読出し処理を効率化するために，(iii) を以下の (iii') に置き換えてプログラム 2 を記述した。(g)～(r) に適切なセマフォ操作を埋めてプログラムを完成させよ。なお，空欄に何も埋める必要がない場合には，nop と答えよ。

(iii') 読出しプロセスが reader 関数内でデータベースにアクセスしている間は，他の読出しプロセスも reader 関数内でデータを読み出せる。書込みプロセスが writer 関数内でデータベースにアクセスしている間は，他のプロセスがデータベースにアクセスできないよう排他制御する。

(3) プログラム 2 には，書込みプロセスにおける，データベースへの書込み処理の応答性に関する問題がある。どのような問題か，具体的な例を挙げて説明せよ。

(4) プログラム 2 が満たす仕様 (i)，(ii)，(iii')，(iv)，(v) を維持したまま，(3) の問題を解決するための対策を述べよ（ただし，プログラムを書く必要はない）。

プログラム 1

```
1  semaphore db = 1; /* データベースへのアクセスを制御するセマフォ */
2
3  void reader(void)
4  {
5      while (TRUE) {  /* 永久に繰り返す */
6          [ (a) ];
7          read_data_base();  /* データベースからデータを読み出す処理 */
8          [ (b) ];
9          use_data();  /* 読み出したデータを使用する処理 */
10         [ (c) ];
11     }
12 }
13
14 void writer(void)
15 {
16     while (TRUE) {  /* 永久に繰り返す */
17         [ (d) ];
18         generate_data();  /* 書き込むデータを生成する処理 */
19         [ (e) ];
20         write_data_base();  /* データベースにデータを書き込む処理 */
21         [ (f) ];
22     }
23 }
```

プログラム 2

```
1  unsigned int rc = 0; /* 読出し中,または読出し待機中のプロセス数 */
2
3  semaphore mutex = 1; /* 変数 rc へのアクセスを制御するセマフォ */
4  semaphore db = 1; /* データベースへのアクセスを制御するセマフォ */
5
6  void reader(void)
7  {
8      while (TRUE) {  /* 永久に繰り返す */
9          [ (g) ];
10         rc = rc + 1;
11         [ (h) ];
12         if (rc == 1) [ (i) ];
13         [ (j) ];
14         read_data_base();  /* データベースからデータを読出す処理 */
15         [ (k) ];
16         rc = rc - 1;
17         [ (l) ];
18         if (rc == 0) [ (m) ];
19         [ (n) ];
20         use_data();  /* 読み出したデータを使用する処理 */
21         [ (o) ];
22     }
23 }
24
25 void writer(void)
26 {
27     while (TRUE) {  /* 永久に繰り返す */
28         [ (p) ];
29         generate_data();  /* 書き込むデータを生成する処理 */
30         [ (q) ];
31         write_data_base();  /* データベースにデータを書き込む処理 */
32         [ (r) ];
33     }
34 }
```

Translation of technical terms

プロセス	process	アクセス	access
読書き	read and write	プロセス間同期	inter-process synchronization
データベース	database	排他制御	mutual exclusion
仕様	specification	セマフォ	semaphore
C 言語	C language	待ち状態	waiting state
関数	function	アトミックに実行	atomic execution
実行可能状態	ready state	クリティカルセクション	critical section
要求	request	応答性	responsiveness