平成 17 年度

東京大学大学院情報理工学系研究科

コンピュータ科学専攻

入学試験問題

専門科目 I

解答

東京大学理学部情報科学科 2009

2012 年 8 月 7 日

問題 1

(1)

2 回.

(2)

一言

当たり前なのと自分が証明苦手なのもあって，微妙だけど見てなんかあれば修正してね

解答

最小の操作を目指すために，問題の操作をするときに選んだおはじきは，2 回選ばれる事は無いとする．

n 個のおはじきの列において k 回とってきて操作するというのは，k 個のおはじきを列から一気に取ってきてそれを残りの $n-k$ 個のおはじきに一つずつ挿入しても同じである．(A) (これは自明だと思う)

以上の事に触れておいて，「A の並び方から B の並び方に置き換える事が k 回以下の操作で可能であること」 $\cdots$ (i) と，「$S_A \cap S_B$ が空集合でないこと」 $\cdots$ (ii) が同値である事を証明する．

(i) $\Rightarrow$ (ii)

k 回以下である $k' \leq k$ 回で可能であるとすると，$k-k'$ 回おはじきの列が変わらない操作を繰り返して k 回の移動可能な操作列を作れる．

(A) より，k 回操作して A から A'' に並び替えられる操作列が存在するということは，n 個のおはじき列 A から k 個のおはじきをとった $n-k$ 個のおはじき列 A' をつくり，それにその取り除いた k 個のおはじきを任意に挿入して得られる n 個のおはじき列が A'' になるような A' が存在する．

A からある k 回の操作列で B に並べ替える事が出来るので，A から k 個のおはじきをとった $n-k$ 個のおはじき列 A' が存在して，取り除いた k 個のおはじきを任意に場所を選びながら挿入することによって，B にすることが出来る．

したがって，B からうまく k 個のおはじきを選んで取り除くことで，A' と同じ列 B' を作る事が出来る．これらはそれぞれ S_A, S_B の要素であり，$A' \in S_A \cap S_B$ である．

(ii) $\Rightarrow$ (i)

空集合ではないので $E \in S_A \cap S_B$ なる E を一つとってくる．この E は，A からある k 個を取り除いた列と同じになり，B からある k 個を取り除いた列とも同じになる．ここで，A と B の白おはじきと黒おはじきの数が同じなので，A から取り除いたおはじきの白黒の数と，B から取り除いたおはじきの白黒の数は同じ．

この取り除き方で A から取り除いて，それを E にし，B から取り除いた方法を用いて挿入することによって B に出来る．

(3)

一言

どういうより高速な方法があるのか知らないけども，適当に A と B の LCS をとって，その長さが L とすると $n - L \leq k$ なら出来るよねって方針で行きます．

解答

A と B の共通部分列は，上記の $S_A \cap S_B$ に含まれる．このとき，共通部分列の長さを l とすると，取り除かれる個数である $n - l$ 回の操作で並び替えることが出来る．なので，A と B の最長共通部分列の長さが L とすると $n - L$ が最短の操作回数である．

これを利用して，最長共通部分列を求めた後に，$n - L \leq k$ であるかどうかを判定する．A, B のおはじきの並びを $a_1 a_2 \cdots a_n, b_1 b_2 \cdots b_n$ とする．計算量は $O(n^2)$.

$LCS := (n+1) \times (n+1)$ 行列
for $i = 0$ **to** n **do**
  $LCS(i, 0) \leftarrow 0$
  $LCS(0, i) \leftarrow 0$
end for
for $i = 1$ **to** n **do**
  for $j = 1$ **to** n **do**
    if $a_i = b_j$ **then**
      $LCS(i, j) \leftarrow \max(LCS(i-1, j), LCS(i, j-1), LCS(i-1, j-1) + 1)$
    else
      $LCS(i, j) \leftarrow \max(LCS(i-1, j), LCS(i, j-1), LCS(i-1, j-1))$
    end if
  end for
end for
if $n - LCS(n, n) \leq k$ **then**
  return true
else
  return false
end if

問題 3

(1)

(2)

(3)

解答用紙に図があるらしいのですが、解答用紙がないので解けません。

(4)

評価すべき局面のことを探索木の葉と考え、葉の部分が評価されることで枝の部分の表が更新されていくと考える。

このとき分岐 b、深さ d の探索木において、全体の探索にかかる局面の数が E_d であるとき、最初の分岐の探索にかかる局面の数が E_{d-1} かかり、その他の分岐は 1 つ葉を確かめればそれで足りるので、

$E_d = E_{d-1} + (b-1)$

(5)

これを解いて、$E_d = (d-1)(b-1)+1$

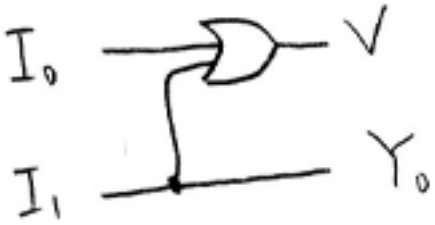
問題 4

(1),(2) の実験結果を用いて (3) を解く問題なんでしょうか。
ですが正直問題の意図があまりよくわかりません。
(2) とか正直 (3) の方法を用いないと入力が 64bit の回路とか書く気が起こりませんので、まず (3) の方法を紹介してそれを用いて (1)(2) の回路を書いていこうと思います。
もし間違いなどありましたらどんどん指摘して下しあ。

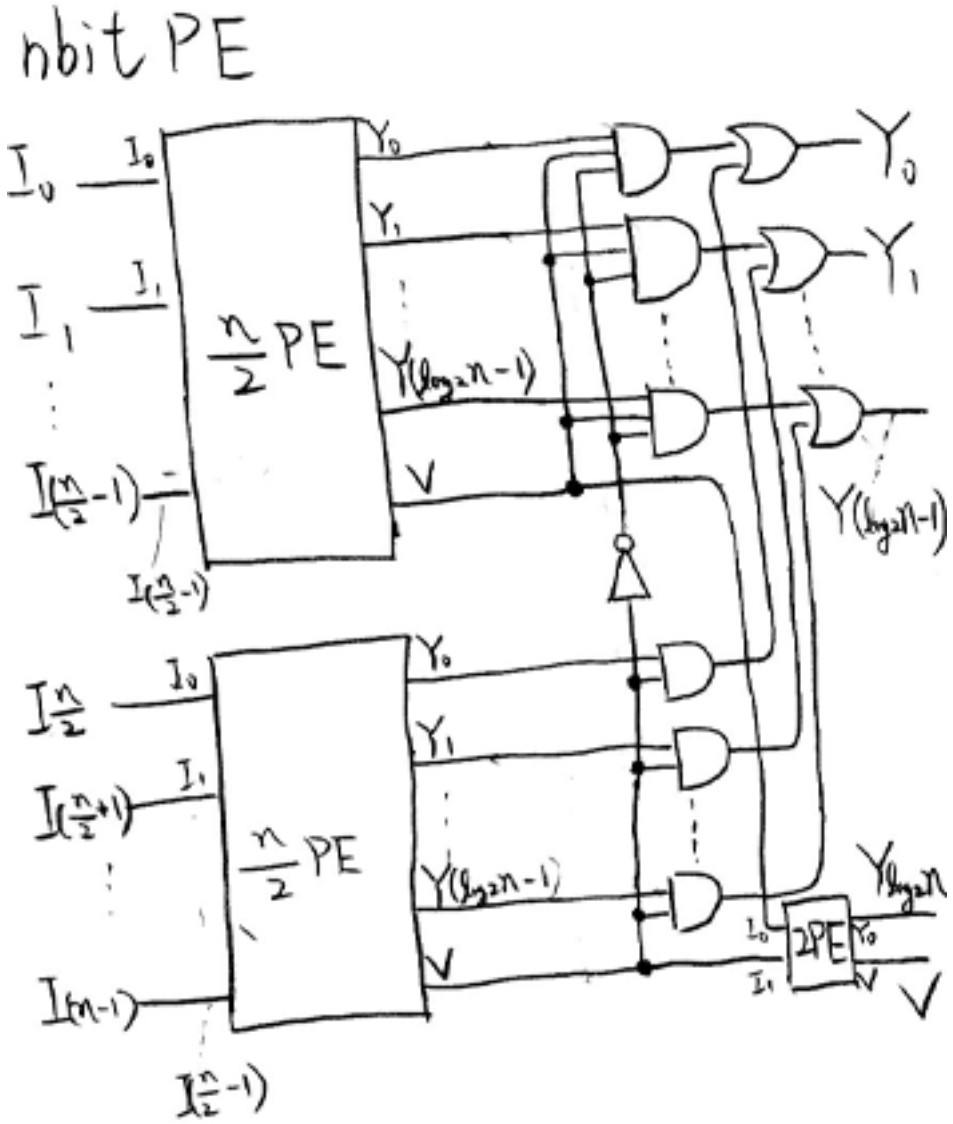
以下、プライオリティエンコーダを PE と表記します。

(3)

まず、2bitPE は以下の様に設計できる。



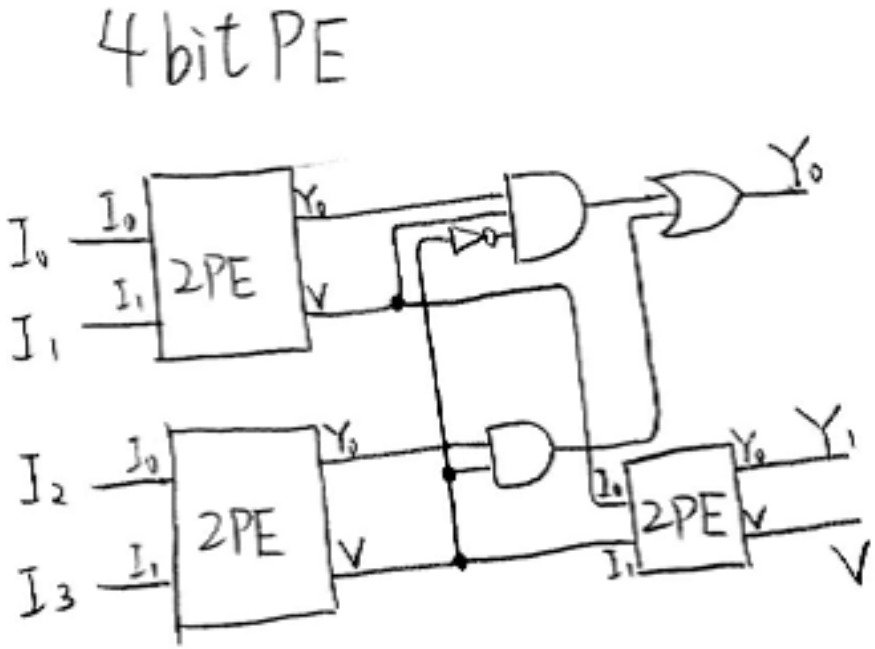
これを用いると、2 の累乗数である n に対して、nbitPE は (n/2)bitPE を用いて以下のように設計できる。



以上より、帰納的に 2 の累乗に対する nbitPE が構成できる。
任意の n に対する PE を構成するには、上記の方法で構成した 2 の累乗数の PE に対して、足りない入力を 0 で補完して入力する回路に変更してやればいい。

(1)

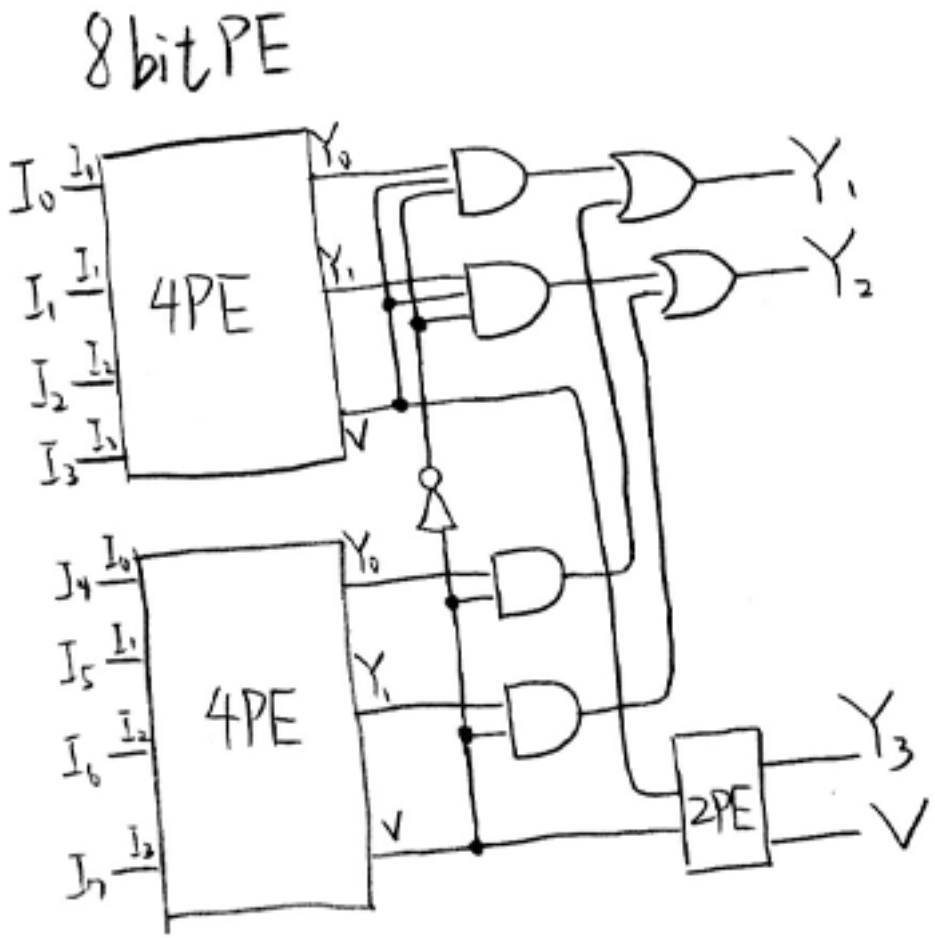
以下のように構成できる。



(2)

(3) の方法を繰り返し用いて構成する。何度も書くのはあまりにも苦痛なので、(2) でつくった 4bitPE と 2bitPE から 8bitPE を構成するとこまでやります。

まず 8bitPE は以下のように構成できる。



次に 16bitPE、32bitPE、64bitPE と構成していけばいい。

専門科目 II

問題 2

(1)

O'Caml で書きました。

```
1  type exp = Var of string | Op of string * exp * exp ;;
2
3  let num = ref 0;;
4
5  let newvar () =
6    num := !num + 1; "tmp." ^ (string_of_int !num);;
7
8  (* val oflist :
9     exp -> (op * operand1 * operand2 * result) list * result  *)
10 let rec oflist = function
11   | Var s -> ([], s)
12   | Op (op, e1, e2) ->
13       let (l1,v1) = oflist e1 in
14       let (l2,v2) = oflist e2 in
15       let v3 = newvar() in
16         (l1 @ l2 @ [(op, v1, v2, v3)], v3);;
```

```
# let l = Op("/", Op("+", (Var "i"), Op("*", (Var "i"), (Var "m"))),
              Op("*", (Var "i"), (Var "m")));;
 val l : exp =
  Op ("/", Op ("+", Var "i", Op ("*", Var "i", Var "m")), Op ("*", Var "i", Var "m"))
# oflist l;;
- : (string * string * string * string) list * string =
([("*", "i", "m", "tmp.1"); ("+", "i", "tmp.1", "tmp.2");
  ("*", "i", "m", "tmp.3"); ("/", "tmp.2", "tmp.3", "tmp.4")],
 "tmp.4")
```

式を受け取って 4 つ組の列と結果の変数の組を返す。

受け取った構文解析木が葉 (=変数) ならば空のリストと変数の組を返す。ノード (=二項演算子) ならば左右の子に対して再帰的に呼び出し、4 つ組列と結果の変数の組を受け取って新しい変数を作り、4 つ組の列にこのノードの演算を追加したものと変数の組を返す。

(2)

add r1 r2 r3	$r_3 := r_1 + r_2$
sub r1 r2 r3	$r_3 := r_1 - r_2$
mult r1 r2 r3	$r_3 := r_1 * r_2$
div r1 r2 r3	$r_3 := r_1 / r_2$
load r1 r2 I	$r_2 := M[r_1 + I]$
store r1 r2 I	$M[r_1 + I] := r_2$

各 4 つ組に対し

1. オペランドがレジスタに割り当てられてなかったら load 命令を挿入する
   (a) 空いているレジスタがあればそこへ割り当てる
   (b) 今後参照されない変数にレジスタに割り当てられていれば、そこへ割り当てる

（c）割り当てられるレジスタがなければ、store 命令を挿入して適当な変数を退避する

2. 結果の出力先を割り当てる
   （a）返り値ならば返り値用レジスタに割り当てる
   （b）空いているレジスタがあればそこへ割り当てる
   （c）今後参照されない変数にレジスタに割り当てられていれば、そこへ割り当てる
   （d）割り当てられるレジスタがなければ、store 命令を挿入して適当な変数を退避する
3. 演算命令を挿入する

を繰り返す。

レジスタ割り当てに関してはデータフロー解析やグラフ彩色アルゴリズム、整数計画問題など様々な方法があるが、条件分岐がないためこの程度でよいと思います。

問題 4

(1)

signal、wait は、0 か 1 のどちらかの値をとる 2 値の整数 Semaphore 型変数 S に対して、以下の様な関数として定義できる。ただしどちらの関数も atomic に実行されなければならない。

```
void wait(int S){
 while(S<=0);
 S--;
}

void signal(int S){
 S++;
}
```

(2)

プロセス同期を支援する機械語命令の例として Swap 命令が挙げられる。Swap 命令とは、二つの変数の値を atomic に入れ替える命令である。この命令を用いることで排他的制御 (Mutual Exclusion) を実現することが可能である。同様にプロセス同期を支援する命令の例として Test and Set 命令がある。

(3)

//2 プロセスの場合は count に関する操作だけ critical section とすれば充分である。

```
void put1byte(byte ch) {
 loop:
 if (count == 32) {
  goto loop;
 }

 wait(S);
 count++;
 signal(S);

 buffer[in] = ch;
 in = (in + 1) % 32;
}
```

```
byte get1byte() {
 byte ch;
 loop:
 if (count == 0) {
  goto loop;
 }

 wait(S);
 --count;
 signal(S);

 ch = buffer[out];
 out = (out + 1) % 32;
 return ch;
}
```

(4)

//get1byte 関数が呼ばれ続けると、get2byte 関数が実行されずに飢餓状態に陥ることがあるのでそこを注意する。

```
byte get2byte() {
 short ch_short;

 wait(S_get);

 loop:
 if (count == 0) {
  goto loop;
 }

 wait(S);
 count = count - 2;
 signal(S);

 ch_short = (short)(buffer[out] << 4) + buffer[out+1];
 out = (out + 2) % 32;
```

```
 signal(S_get);

 return ch;
}

byte get1byte() {
 byte ch;

 wait(S_get);

 loop:
 if (count == 0) {
  goto loop;
 }

 wait(S);
 --count;
 signal(S);

 ch = buffer[out];
 out = (out + 1) % 32;
 signal(S_get);
 return ch;
}
```