ACM ICPC 2017 国内予選問題 解説

平石 裕実

（京都産業大学 コンピュータ理工学部）

プログラムの大きさと必要な知識等一覧

問題	ソース	行数	実行時間(秒)	解法や知識・テクニック等
A. 太郎君の買物	pa.c	23	0.210(0.175)	2重ループ、組合せ、最大値
B. ほとんど同じプログラム	pb0.c	51	0.139(0.147)	文字列処理ライブラリ
	pb1.c	46	0.138(0.151)	文字列と文字型配列
C. 池のある庭園	pc.c	50	0.153(0.174)	2次元配列、多重ループ、最大値
D. 弁当作り	pd.c	79	8.938(4.487)	動的計画法、深さ優先探索、parity演算
E. 論理式圧縮機	pe.c	138	2.915(2.690)	BNF記法、論理関数、論理式の評価、構文図
F. リボンたたみ	pf.c	25	0.137(0.144)	空間的思考力、2進数の表現
G. 迷宮を一周	pg0.c	78	2.594(1.655)	2次元配列、深さ優先探索
	pg1.c	111	0.151(0.192)	2次元配列、深さ優先探索、グラフの関節点
	pg2.c	101	0.144(0.215)	2次元配列、右手法による迷路探索
H.等積変形	ph.c	166	0.168(0.203)	幾何、直線の平行判定、3直線が一点で交わる条件

- 行数は、コメントや空行やデバッグ用コードを除いた行数
- 実行時間(秒): コンパイル、サンプルデータ、ジャッジデータに対する判定時間を含む（括弧内は-O2でコンパイル）
  Core i7-5650U @ 2.20GHz, 8GBメモリ, MacOS 10.13.6, Apple LLVM version 8.0.0 (clang-800.0.38)

問題A 太郎君の買物

- 異なる品物の全てのペアの合計金額を求め、予算内の最大値を求める。

```
int n;                          // 品物の個数
int price[1000];                // 品物の価格
for(i=0; i<n-1; i++){
    for(j=i+1; j<n; j++){
        合計 = price[i]+price[j];     // 品物iとjの合計金額
        // 予算内で最大値を超えていたら最大値を更新
        if(合計 <= 予算 && 合計 > 最大値){
            最大値 = 合計;
        }
    }
}
```

問題B ほとんど同じプログラム

- 「“」により、非文字列定数、文字列定数が交互に切り替わる
- 方法1(pb0.c): 文字列処理ライブラリの関数を用いて実装

```
char *str1, *str2;                  // 各プログラムの現在チェック中の位置
char *dq1, *dq2;                    // “ の位置へのポインタ
int ndcc = 0;                       // 非文字列定数の不一致カウント
// 非文字列定数処理
dq1 = strstr(str1, “¥””); dq2 = strstr(str2, “¥””);      // dq1, dq2 は「“」へのポインタ（無ければNULL)
dq1とdq2が両方ともNULLの場合(この非文字列定数の後に文字列定数は無い)
    文字列str1とstr2が等しければndccの値で判断し、等しくなければダメ(DIFFERENT)
dq1とdq2が両方ともNULLでない場合（次に文字列定数が続く）
    dq1とdq2が指す位置の「”」をNULLに置き換える(非文字列定数の切出し)
    文字列str1とstr2が等しくなければダメ(DIFFERENT)
                等しければ、str1とstr2を各々dq1とdq2の次の位置とし文字列定数処理へ
dq1とdq2が異なる場合はダメ(DIFFERENT)（片方のみ次に文字列定数が来る）
// 文字列定数処理
dq1 = strstr(str1, “¥””); dq2 = strstr(str2, “¥””);      // dq1, dq2 は「“」へのポインタ
                                                        // 文字列定数なので「”」は必ずある
dq1, dq2が指す位置の「”」をNULLに置き換える(文字列定数の切出し)
文字列str1とstr2が等しくなければndccを1増やしndccが2になったらダメ(DIFFERENT)
str1とstr2を各々dq1とdq2の次の位置とし非文字列定数処理へ
```

問題B ほとんど同じプログラム（続）

- 方法2(pb1.c)：文字列処理ライブラリの関数を用いずに実装

```
char *str1, *str2;                  // 各プログラムの現在チェック中の位置
int ndcc = 0;                       // 非文字列定数の不一致カウント
// 非文字列定数
c1 = *str1++; c2 = *str2++;         // c1, c2 : 先頭の文字
c1 != c2   ⇒   ダメ(DIFFERENT)
c1 == NULL && c2 == NULL      ⇒   判定終了(ndcc==1 → CLOSE, ndcc==0 → IDENTICAL)
c1とc2がともに「“」                  ⇒   文字列定数処理へ
それ以外(普通の文字の一致)：   上記（非文字列定数処理）を繰り返す
// 文字列定数
c1 = *str1++; c2 = *str2++;         // c1, c2：先頭の文字
c1とc2がともに「”」                  ⇒   非文字列定数処理へ
c1 != c2   ⇒   ndcc++;
               ndcc >= 2   ⇒   ダメ(DIFFERENT)
               *str1, *str2 が「"」になるまでstr1, str2を進め文字列定数処理へ
それ以外(普通の文字の一致)：   上記（文字列定数処理）を繰り返す
```

問題C 池のある庭園

- 池の幅(pw)と奥行き(pd)は３からマップの幅(w)と奥行き(d)まで
- 池の左上の位置(x,y):　$0 \leqq x \leqq w - pw$,　$0 \leqq y \leqq d - pd$
- これらの組合せ（４重ループ）に対して池の容量の最大値を求めれば良い
- 池の容量の計算
  - 池の境界（上下左右の境界）の高さの最小値を求める（各境界に対して１重ループ）
  - 池の内部に対し（２重ループ）境界の最小値以上の高さ　⇒　容量＝０
  - そうでなければ、境界の最小値との差を積算していく

※ 池の容量の計算を関数にしなければ６重ループになる

（データセットを順に処理する部分も含めると7重ループ）

問題D 弁当作り

- レシピ数(n)×材料の種類数(m) ≦ 500 なので、min(n, m) ≦ 22
- 材料が残るかどうかを問題にしており、材料は 2 つ入りなので材料の使用数のparity(奇数か偶数)を考えれば良い
- m≦nの時(m≦22)：材料使用数のparity vector pv(m bit, vector数は2^m, 0≦pv<2^m)に対してレシピを一つずつ加えながら作成可能なレシピ数の最大値を**DP**で求める
  - num[pv]: ここまでのレシピ内でpvで実現できるレシピ数の最大値
    そのような pv を実現出来ないときは -1 としておく
  - num[ ]の初期値： num[0] = 0; // どのレシピも作らない時、使用材料は無いのでpv=0
    num[pv] = -1 for pv ≠ 0 // どのレシピも作っていないので pv ≠ 0 とはならない
  - parity: 今回新たに追加するレシピの使用材料ベクトル
  - next_num[pv] := if(num[pv⊕parity] >= 0) MAX(num[pv], num[pv⊕parity]+1);
    else num[pv];

    num[pv⊕parity] >= 0 の場合、今回追加するレシピを作成するなら使用材料のparity vectorはpv⊕parity⊕parity=pvとなり、作成可能なレシピ数の最大値はnum[pv⊕parity]+1となる。作成しない場合、num[pv]は変化しないので、両者の大きい方をとる。
    num[pv⊕parity] = -1 の時は、今回追加するレシピを作成してもpvにすることは出来ないので、num[pv]は変化しない。

問題D 弁当作り（続）（DPの例）

- num[ ]の初期値：　num[0] = 0;　// どのレシピも作らない時、使用材料は無いのでpv=0
  num[pv] = -1 for pv ≠ 0　// どのレシピも作っていないので pv ≠ 0 とはならない
- next_num[pv] := if(num[pv⊕parity] >= 0)　MAX(num[pv], num[pv⊕parity]+1);
  else　num[pv];

レシピ	材料
レシピ1	0110
レシピ2	1011
レシピ3	1111
レシピ4	1101

- -1からの矢印は省略
- 矢印の色はデータの出所
- 太線：Max勝負に勝った
- 破線：Max勝負に負けた

pv	初期値	レシピ1(0110)	レシピ2(1011)	レシピ3(1111)	レシピ4(1101)
0000	0	0	0	0	3
0001	-1	-1	-1	-1	-1
0010	-1	-1	-1	3	3
0011	-1	-1	-1	-1	-1
0100	-1	-1	-1	2	3
0101	-1	-1	-1	-1	-1
0110	-1	1	1	1	2
0111	-1	-1	-1	-1	-1
1000	-1	-1	-1	-1	-1
1001	-1	-1	-1	2	3
1010	-1	-1	-1	-1	-1
1011	-1	-1	1	1	2
1100	-1	-1	-1	-1	-1
1101	-1	-1	2	2	2
1110	-1	-1	-1	-1	-1
1111	-1	-1	-1	1	4

⟸ 答えは3

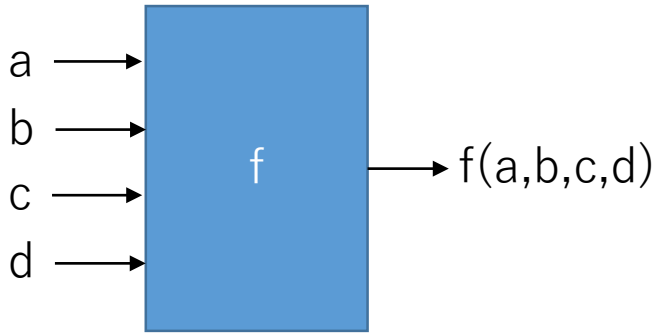
問題D 弁当作り（続）（DFS）

- n<mの時(n≦22): レシピの全ての組合せ(2ⁿ通り)を**DFS**で生成しながら
  各組合せに対する材料使用数のparity vectorを求める。
  parity vectorが0（材料は残らない）のものに対してレシピ数が最大のものを
  求めれば良い。

```
// level=レシピ番号, count=作成レシピ数, cs=材料使用数のparity vector(配列)
dfs(int level, int count, char cs[ ]){
    char ns[m];                    // level番目のレシピを採用した場合の使用材料 parity vector
    if(level==n){                  // 全レシピの組合せが決定
        parity vector cs が 0 で count が最大値を超えていれば最大値を更新;
    } else {
        残りのレシピ(n-level個)を全て採用しても最大値を更新出来ないなら return;          // 枝刈り
        // レシピ数最大のものを求めたいので、このレシピを採用する場合を先に探索
        ns = cs ⊕ level番目のレシピの材料のparity vector;    // このレシピを採用する場合
        dfs(level+1, count+1, ns);
        dfs(level+1, count, cs);           // このレシピを採用しない場合
    }
    return;
}
```

問題E 論理式圧縮機

- 4変数論理関数は $2^{2^4} = 2^{16} = 65{,}536$個しかない。
- 入力4変数の値の組合せ16通りに対して関数値0,1を決める
  ⇒　16bit（真理値ベクトル）で関数を表現:
  aを最上位bit, dを最下位bitとすると
  - 定数 0: 0x0000　　= 0000 0000 0000 0000
  - 定数 1: 0xFFFF　　= 1111 1111 1111 1111
  - 変数 a: 0xFF00　　= 1111 1111 0000 0000
  - 変数 b: 0xF0F0　　= 1111 0000 1111 0000
  - 変数 c: 0xCCCC　　= 1100 1100 1100 1100
  - 変数 d: 0xAAAA　　= 1010 1010 1010 1010



入力(a,b,c,d)=(0,0,0,0)～(1,1,1,1)の16通りに対する関数値0,1を決めると関数がユニークに定まる
この16bitを関数のIDとする

※ この表現を関数のID（$0 \sim 2^{16} - 1 = 65{,}535$）とする。16bitなので unsigned short に格納。

- 論理演算は、上記の表現形式（関数のID）に対して、対応するbit演算を行えば良い。
  - 関数のIDをint型(32bit)変数に入れて(上位2バイトは0)演算を行う場合、論理否定演算は
    bit否定演算（~）適用後下位16bitでマスクする必要がある。

問題E 論理式圧縮機（続）

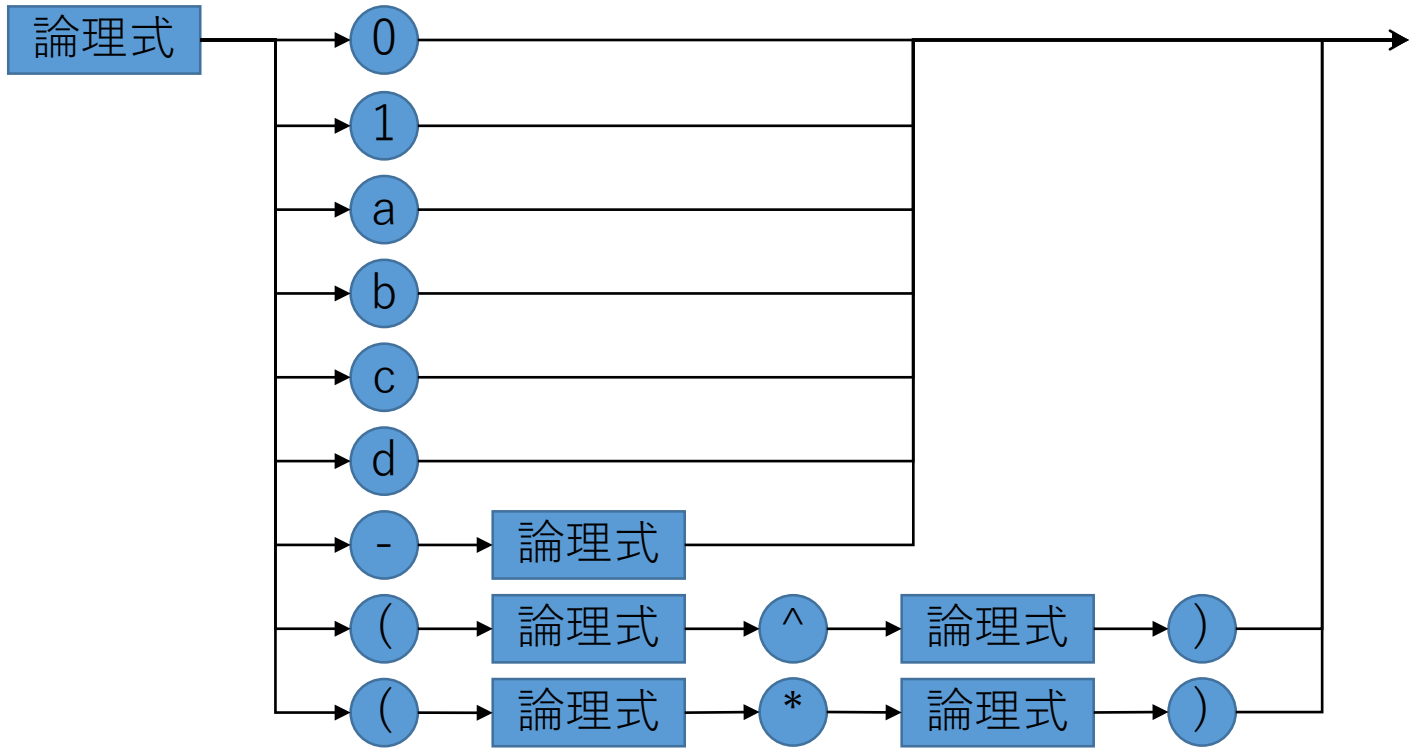
- 事前に長さ16以下の論理式を全て生成し、論理関数ごとに最小の論理式の長さを求めておく。
  - ➢ 論理式の生成
    - ◆ LF_k：論理演算子をk個含む論理式fの集合。
      但し、fと等価で f 以下の長さの論理式が既登録の場合は除く。
    - ◆ $LF_0 = \{0, 1, a, b, c, d\}$ を登録（定数と変数。論理演算子の個数は0）
    - ◆ LF_{k+1}の作成：次の条件を満たす論理式で構成
      - ■ $f \in LF_k, g \in LF_0 \cup \cdots \cup LF_k$ で “-f” または “(f^g)” または “(f*g)”　　// 論理演算子の個数はk+1
      - ■ 式の長さは16以下
      - ■ この式と等価で、この式の長さ以下の長さの論理式が登録されていない
    - ◆ 上記をk=0から順にkを増やしながら繰り返し、$LF_{k'}$ が空集合になれば終了。

    ※ 条件を満たす論理式は論理式テーブルに前から順に追加していく。
    LF_kとLF_{k+1}の開始位置を記憶することで f の選択が容易になる。

    ※ 論理式の論理関数表への登録：
    - 論理式が表す論理関数のIDを求める
    - 論理関数表のIDの位置が未登録　→　この論理式の長さと、論理式表での位置を登録
      登録されている長さの方が大きい　→　登録されている式を論理式表で無効にし、この式の長さと論理式表での位置を登録

問題E 論理式圧縮機（続）

- 論理式の評価（構文図による評価）




```
unsigned short eval(){
    unsigned short val1, val2;
    次のtokenを取ってくる;
    switch(token){
    case '0':    return 0;
    case '1':    return 0xFFFF;
    case 'a':    return 0xFF00;
    case 'b':    return 0xF0F0;
    case 'c':    return 0xCCCC;
    case 'd':    return 0xAAAA;
    case '-':    val1 = eval(); return ~val1;
    case '(':    val1 = eval();
                 次のtokenを取ってくる;
                 if(token == '^'){
                     val2 = eval();
                     次のtokenを取ってくる;    // ')'
                     retrun val1 ^ val2;
                 else {                              // token == '*'
                     val2 = eval();
                     次のtokenを取ってくる;    // ')'
                     return val1 & val2;
                 }
    }
}
```

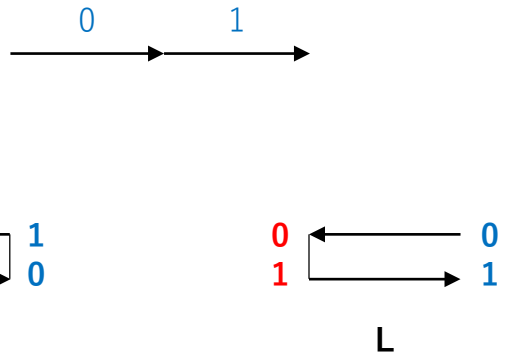
問題F リボンたたみ

※ →はリボンを全て開いたときに左から右の向きを示す

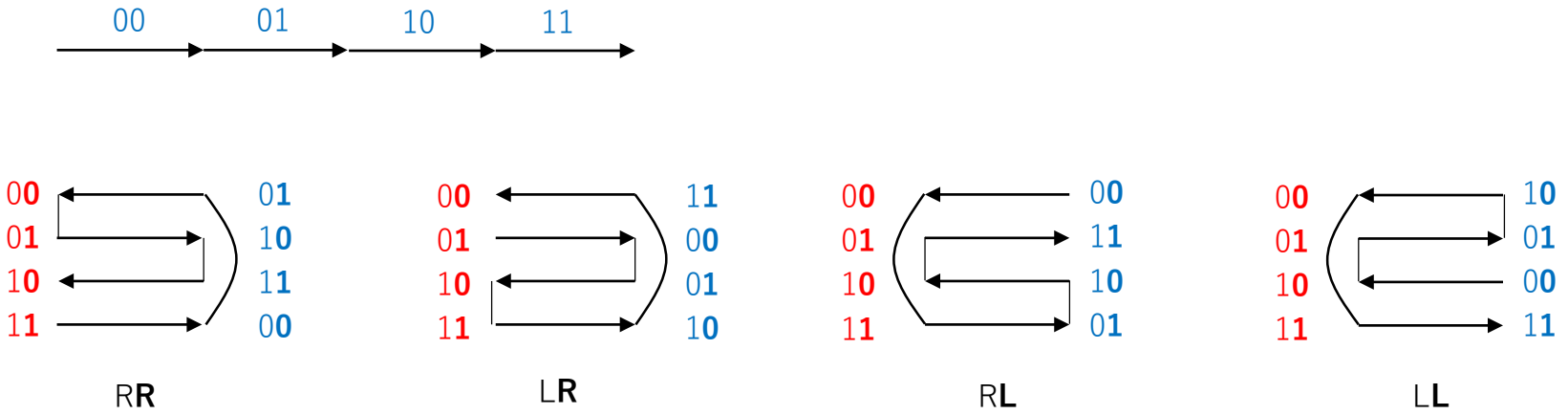
n=1

青：左からの位置（2進数）
（0から数える）

赤：上からの位置（2進数）
（0から数える）



n=2

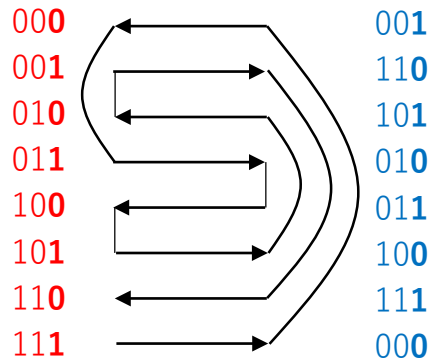


上からの位置と左からの位置の奇偶(最下位bit)が一致　⇔　最後の折りたたみはL
不一致　⇔　最後の折りたたみはR

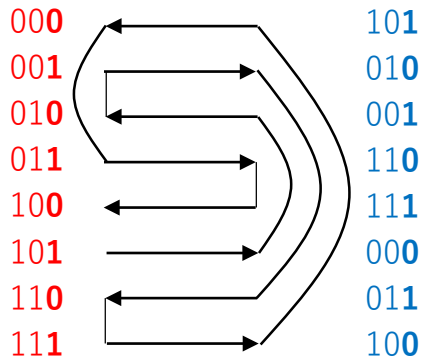
問題F リボンたたみ（続）

n=3

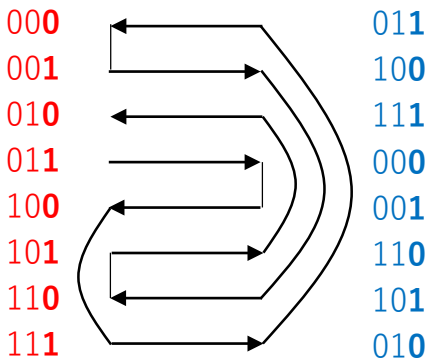
000 → 001 → 010 → 011 → 100 → 101 → 110 → 111



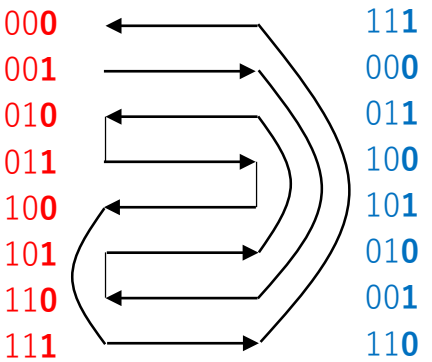
RR**R**



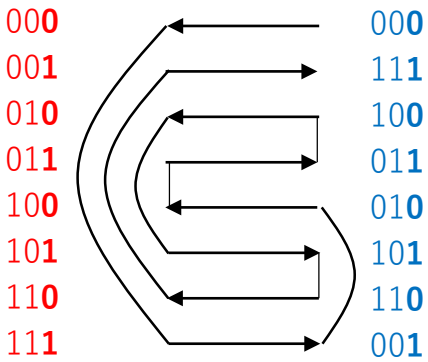
LR**R**



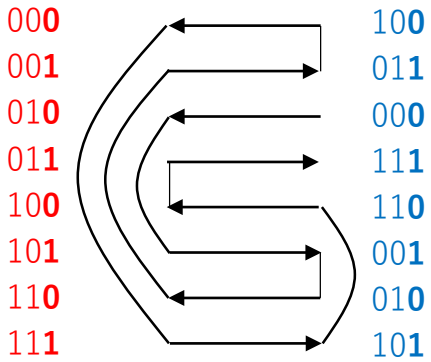
RL**R**



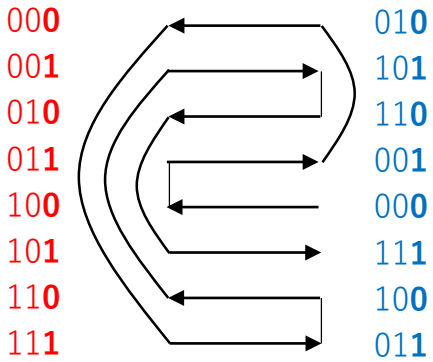
LL**R**



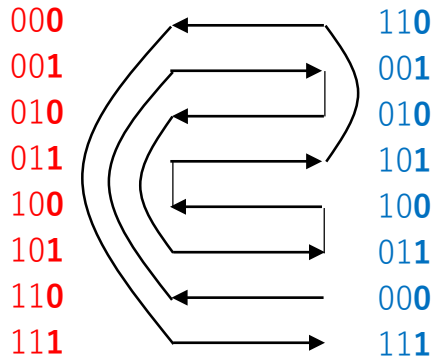
RR**L**



LR**L**



RL**L**



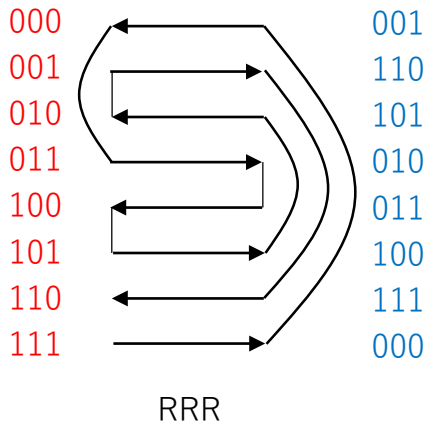
LL**L**

(A)

上からの位置と左からの位置の奇偶(最下位bit)が一致　⇔　最後の折りたたみはL
不一致　⇔　最後の折りたたみはR

問題F リボンたたみ（続）

n=3



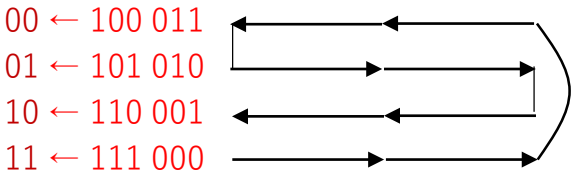
RRR

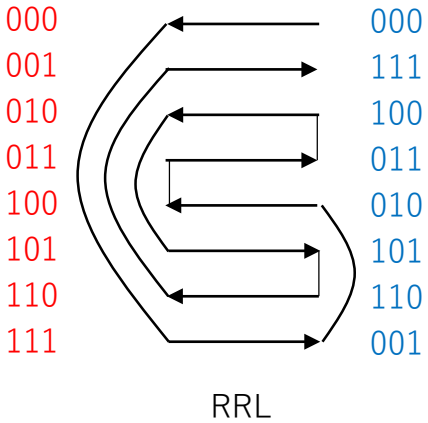
000 → 001 → 010 → 011 → 100 → 101 → 110 → 111

(B) 最後の折りたたみを開くと　⇩　左からの位置は1bit右シフト

00 → 01 → 10 → 11

開く



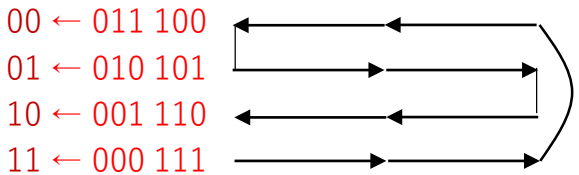


RRL

(C) ⇧
折りたたみを開いた後の上からの位置＝
⇩

- 元の位置が上半分$(0 \sim 2^{n-1}-1)$：　$2^{n-1}-1-$元の位置
- 元の位置が下半分$(2^{n-1} \sim 2^{n}-1)$：　元の位置-2^{n-1}

開く



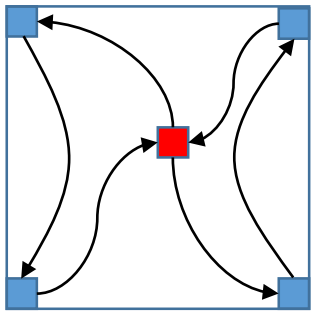
※ LRR, RLR, LLR, LRL, RLL, LLLの場合も同様

(A)(B)(C)を繰り返すことにより最後から最初に向けて折りたたみ方向が求まる

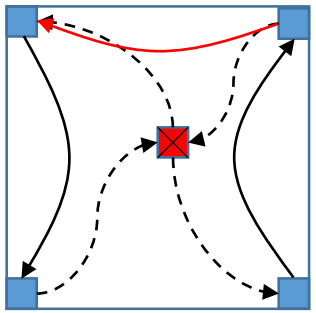
※ 上からの位置や左からの位置は最大$2^{60}-1$なのでlong型を使用する必要有

問題G 迷宮を一周

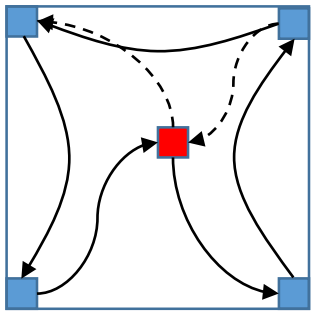
- 与えられたマップデータに対し、周りを入室不可の部屋で囲んでおく
- 4隅の部屋を通る単純閉路（同じ部屋は2度通らない）の有無をチェックすればよい
- 方法１（pg0.c）：
  - 4隅の部屋が非連結なら4隅の部屋を通る閉路は存在しない
  - 4隅以外の部屋を一つ取り除いて（入れない部屋とする）、4隅の部屋が連結しているかをチェック。ある部屋を取り除いたときに、4隅の部屋が非連結になるなら、4隅の部屋を通る単純閉路は存在しない。
  - 4隅の部屋が連結しているかどうかは、例えば左上隅の部屋からDFSで探索し、他の3つの隅の部屋に到達できるかどうかをチェックすれば良い。



(a)



(b)



(c)

- 4隅を通る閉路がどうしても同じ部屋（赤い部屋）を通る（単純閉路が存在しない）とする(a)。このとき赤い部屋を入室不可にすると、4隅の部屋は非連結になる。
(∵ 赤い部屋を取り除いても連結なら、赤い部屋を通らないルートが存在する(b)。このルートを使うと赤い部屋を2度通らない閉路が存在する(c)ことになり矛盾)
- 逆に、単純閉路が存在するとすると、4隅以外の任意の部屋を一つ入室不可にしても、4隅の部屋は連結したままである。

問題G 迷宮を一周（続）

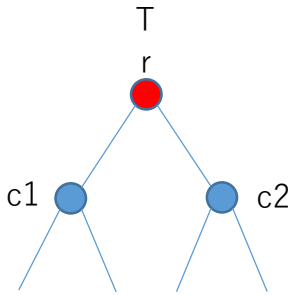
- 方法２（pg1.c）： 
  - 出入口の部屋から移動可能な部屋を深さ優先で探索しDFS木Tを構築する。
  - この時、ある宝物の部屋がDFS木に含まれないなら、出入口から宝物の部屋全てを回る閉路は存在しない。
  - また、Tの構築に際し各節点（部屋）のlowlinkも計算する。
  - これにより、グラフ（節点＝入室可能な部屋、枝＝移動可能（入室可能な部屋が隣接している））の関節点（その点とその点に接続している枝を削除するとグラフが分離する）と分離点（Tにおいて、関節点の子節点で、関節点を取り除くとTの根と分離する節点）を求める。
  - Tにおいて宝物のある各部屋から分離点or根節点に達するまで祖先方向に遡る。
  - この時ひとつでも分離点に達したら、出入口から宝物の部屋全てを回る単純閉路は存在しない（分離点の親の関節点を2回通る必要がある）。

問題G 迷宮を一周（続）

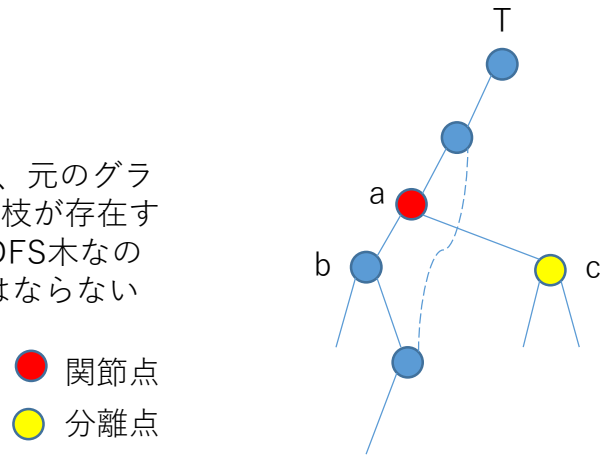
- order(u) = Tにおける節点uの訪問順(inorder)
- parent(u) = Tにおける節点uの親節点
- lowlink(u) = 次の値の最小値
  1. order(u)
  2. 節点uから節点vへTに属さない枝が存在するならorder(v)
  3. Tにおいて節点uの全ての子vに対するlowlink(v)

  ※ lowlinkはTにおける節点uの子孫からTに属さない枝を高々１本通って到達できる最も若い（訪問順が早い）節点を表す
- 関節点と分離点
  1. Tの根：２つ以上の子供がある　⇔　関節点
  2. Tの根以外の節点u： order(parent(u)) ≦ lowlink(u)　⇔　parent(u)は関節点。uを分離点と呼ぶことにする。

     ※ Tにおけるuの子孫からは、もとのグラフにおいてparent(u)の真の祖先への枝は無い　⇔　Tにおけるuの子孫はparent(u)によりTの根から分離される



1. Tの根rが関節点でないとすると、元のグラフでc1の子孫とc2の子孫の間に枝が存在することになるが、その場合TはDFS木なのでc2はc1の子孫となりrの子にはならない



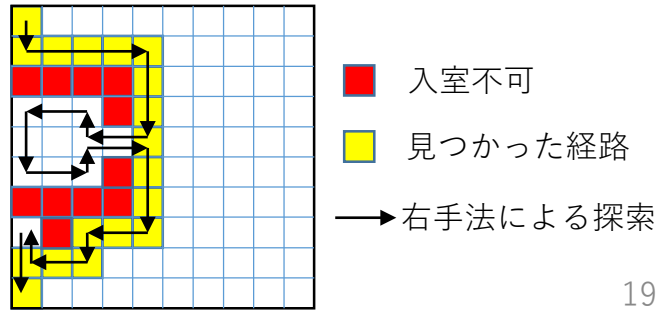
2. order(a) > lowlink(b), order(a)≦ lowlink(c)とすると、元のグラフにおいて、bの子孫からaの真の祖先への枝が少なくとも1本存在し、cの子孫からaの真の祖先への枝は存在しない。また、bの子孫とcの子孫との間に枝は存在しない（もしそのような枝が存在したとすると、TはDFS木なのでcはbの子孫となりaの子とはならない）。したがって、もし宝の部屋がcの子孫だとすると、関節点aにより出入口（根節点）から分離され４隅の部屋を通る単純閉路は存在しない

問題G 迷宮を一周（続）

- 方法３（pg2.c）：出来るだけ外壁に沿って4隅の部屋を順に回れるかをチェック 
  - 北西角→ 南西角：右手法で迷路探索。到達できたら訪問した部屋は入室不可に
  - 南西角→ 南東角：右手法で迷路探索。到達できたら訪問した部屋は入室不可に
  - 南東角→ 北東角：右手法で迷路探索。到達できたら訪問した部屋は入室不可に
  - 北東角→ 北西角：右手法で迷路探索。到達できたら訪問した部屋は入室不可に

※ 右手法で探索中は同じ部屋を通っても良い（袋小路等から脱出するため）
※ 右手法で経路が見つかれば、同じ部屋を通らない経路が存在する
※ 右手法で探索時に訪問する部屋は、見つかった経路より外壁側にしか存在しない。
（中心側に訪問した部屋があるとすると、右手法で経路が見つかったことに矛盾）
したがって、見つかった経路上の部屋だけでなく、探索時に訪問した部屋も入室不可としても次の探索に影響しない。
※ 右手法で見つかった経路よりも外壁側を通る経路は存在しない。
※ 上記の方法で順に回れる→そのルートが単純閉路
単純閉路が存在→右手法で見つかるルートは単純閉路を構成するルートorそれより外壁側→上記の方法で順に回れる

北西角→南西角への右手法による探索

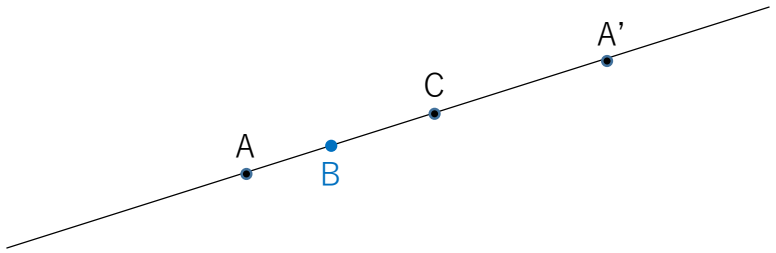


問題H 等積変形

- 三角形T1 = △ABC, 三角形T2 = △A'B'C' とする
  T1の3頂点とABCの対応付は 3! = 6 通り、
  T2の3頂点とA'B'C'の対応付は 3! = 6 通り
  これらの組み合わせ(36通り)に対し、
  A → A', B → B', C → C' の順に移動可能かをチェック
- 3回の移動で可能な場合
  - A → A', B → B', C → C' と移動可能 ⇔ AA' // BC かつ BB' // A'C かつ CC' // A'B'
    (AはBCに平行に移動するので、AA'//BC。BはACに平行に移動するがAはA'に移動しているのでBB'//A'C。CはABに平行に移動するがA,Bは既にA',B'に移動しているのでCC'//A'B'。)
- 平行性のチェック
  - $A = (x_a, y_a), B = (x_b, y_b), C = (x_c, y_c), D = (x_d, y_d),$
    $v_1 = \overrightarrow{AB} = (dx_1, dy_1) = (x_b - x_a, y_b - y_a), v_2 = \overrightarrow{CD} = (dx_2, dy_2) = (x_d - x_c, y_d - y_c)$とする
    $AB \;//\; CD \quad \Leftrightarrow \quad v_1 \;//\; v_2 \quad \Leftrightarrow \quad dy_1 \times dx_2 - dy_2 \times dx_1 = 0$
    ※ 3角形の頂点の座標は整数なので、この問題を解く際の平行性チェックでの座標値も整数

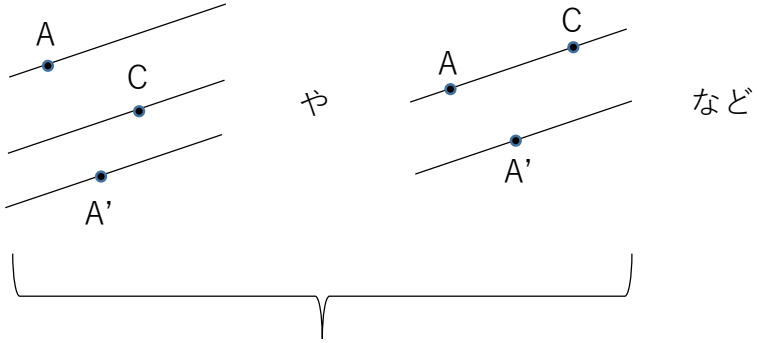
問題H 等積変形（続）

- 4回の移動で可能な場合
  - ➢ A → A’ がDを経由する場合
    - ■ A → D, D → A’, B → B’, C → C’ ：考慮する必要なし（∵ 連続して A → D, D → A’ が可能なら A → A’ も可能）
    - ■ A → D, B → B’, D → A’, C → C’ （Case1）：
      AD // BC　かつ　BB’ // DC　かつ　DA’ // B’C　かつ　CC’ // A’B’　　⇔
      CC’ // A’B’　かつ　Aを通りBCと平行な直線、Cを通りBB’と平行な直線、A’を通りB’Cと平行な直線がある点Dを共有
      右側の条件内の3本の直線がすべて一致することはない(図1)ので、3本の直線が互いに平行ならば共有点Dは存在しない(図2)
      したがって、右側の条件は、3本の直線が平行の場合を除いて、3本の直線が共有点を持つかどうかを判定すれば良い



Aを通る直線、Cを通る直線、A’を通る直線が一致 → A,C,A’は同一直線上
この直線はBCと平行なのでBもこの直線上にある → ABCが三角形にならない → 矛盾
∴3本が一致することはない

図1

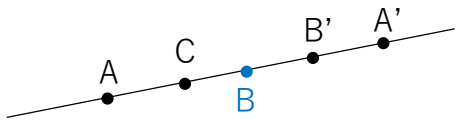


3直線の共有点は無い

図2

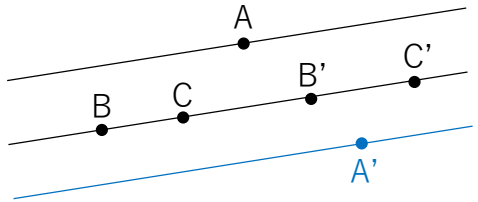
問題H 等積変形（続）

- 4回の移動で可能な場合(三角形の頂点以外の点を一つ経由)
  - ➢ A → A’ がDを経由する場合
    - ■ A → D, B → B’, C → C’, D → A’ （Case2）：
      AD // BC　かつ　BB’ // DC　かつ　CC’ // DB’　かつ　DA’ //B’C’　　⇔
      Aを通りBCと平行な直線、Cを通りBB’と平行な直線、B’を通りCC’と平行な直線、A’を通りB’C’と平行な直線がある点Dを共有
      この4本の直線がすべて一致することはないので、4本の直線が互いに平行ならば共有点Dは存在しない(図3)
      4本の内3本が平行ならば4本とも平行になる(図4)。したがって4本が平行になるケースを除けば、3本が平行になることはない。
      これより、4本が平行になるケースを除外した上で**全ての3本の組み合わせに対して**共有点があるかどうかを判定すれば良い。



Aを通る直線、Cを通る直線、B’を通る直線、A’を通る直線が一致 → A,C,B’,A’は同一直線上
この直線はBCとも平行なのでBもこの直線上にある → ABCは三角形にならない → 矛盾
∴4本が一致することはない。 → この4本が互いに平行なら2本以上の平行直線 → 共有点無

図3



例えばAを通る直線(//BC)、Cを通る直線(//BB’)、B’を通る直線(//CC’)が平行
→　BC//BB’//CC’　→　B,C,B’,C’は同一直線上　→　B’C’とも平行
→　A’を通る直線(//B’C’)も平行　→　結局4本とも平行　→　共有点無

図4

問題H 等積変形（続）

- 4回の移動で可能な場合
  - ➢ B → B’ がDを経由する場合
    - ■ A → A’, B → D, D → B’, C → C’ ：考慮する必要なし（∵連続して B → D, D → B’ が可能なら B → B’ も可能）
    - ■ A → A’, B → D, C → C’, D → B’ （Case3）：
      AA’ // BC　かつ　BD // A’C　かつ　CC’ // A’D　かつ DB’ // A’C’　　⇔
      AA’ // BC かつ　Bを通りA’Cと平行な直線、A’を通りCC’と平行な直線、B’を通りA’C’と平行な直線がある点Dを共有
      右側の条件の3本の直線がすべて一致することはないので、3本の直線が互いに平行ならば共有点Dは存在しない
      したがって、右側の条件は、**3本の直線が平行の場合を除いて、3本の直線が共有点を持つかどうかを判定すれば良い**
      （Case1と同様）
  - ➢ C → C’ がDを経由する場合
    - ■ 考慮する必要なし（∵連続して C → D, D → C’ が可能なら C → C’ も可能）
- 3回、4回で移動出来ないなら、5回以上

問題H 等積変形（続）

- $P=(x_p,y_p), A=(x_a,y_a), B=(x_b,y_b)$ とする。
  点Pを通りABに平行な直線の方程式は $(y-y_p)\times(x_b-x_a)=(x-x_p)\times(y_b-y_a)$
  $d_x=x_b-x_a, d_y=y_b-y_a$ とすると　$d_y\cdot x-d_x\cdot y-x_p\cdot d_y+y_p\cdot d_x=0$
- 3本の直線が一点を共有する条件（平行な直線が2本以下の場合）：

$A=\begin{pmatrix}a_0 & b_0 & c_0\\ a_1 & b_1 & c_1\\ a_2 & b_2 & c_2\end{pmatrix}$とする。3本の直線 $A\cdot\begin{pmatrix}x\\ y\\ 1\end{pmatrix}=\begin{pmatrix}0\\ 0\\ 0\end{pmatrix}$ が点(x', y')を通るとすると

$A\cdot\begin{pmatrix}x'\\ y'\\ 1\end{pmatrix}=\begin{pmatrix}0\\ 0\\ 0\end{pmatrix}$ である。これに Aの逆行列A^{-1}を左からかけると$\begin{pmatrix}x'\\ y'\\ 1\end{pmatrix}=\begin{pmatrix}0\\ 0\\ 0\end{pmatrix}$ となり矛盾。故にAの逆行列は存在しない。すなわち $\det(A)=0$ (Aの行列式の値は0)。
平行な直線が2本以下の場合、これが必要十分条件。（3本とも平行な場合も$\det(A)=0$ ）
※ 平行な直線が2本の場合、それらが同一直線ならば $\det(A)=0$、そうでなければ$\det(A)\neq 0$。

$\det(A)=a_0b_1c_2+b_0c_1a_2+c_0a_1b_2-c_0b_1a_2-b_0a_1c_2-a_0c_1b_2=0$

※ 3角形の頂点の座標は整数なので、ここで出てくる判定は全て整数演算で可能
但し、行列式の計算では int 型の範囲を超える可能性があるので long 型を用いる