Program Promenade

連載

論理回路のシミュレーション

田中 哲朗（東京大学情報基盤センター）
ktanaka@tanaka.ecc.u-tokyo.ac.jp

■問題の定義

今回取り上げる問題は2001年度のGreater New York地区予選の問題E「Follow My Logic」である．問題文は，以下から入手できる．
http://icpc.baylor.edu/past/icpc2002/regionals/GNY01/problems.html

7月号でも取り上げたようにGreater New York地区予選ではパターンにはまらない問題が出題されることが多いが，今回の問題もかなりユニークである．

今回の問題はアスキーアートで書かれた多入力1出力の組合せ論理回路にさまざまな入力を与えたときの出力をシミュレーションで求めるというものである．論理回路は以下のように文字要素のみを用いて表現される．

入力　`'A'`から`'Z'`までの26個の文字で表される．同じ文字が何度も使われることがある．

出力　`'?'`で表される．1つの問題中に1つ必ず存在する．

配線　`'−','|','+'`を使って表す．

ANDゲート　2入力のみ扱う．以下のように表す．

```
-:\
 : )-
-:/
```

ORゲート　2入力のみ扱う．以下のように表す．

```
-:\
 : >-
-:/
```

NOT　単独では用いず，ゲート入力の左側あるいはゲート出力の右側の配線（`'-'`）を`'o'`に置き換えて表す．以下に例を示す．

```
-o:\
  : >o-
--:/
```

以下の条件が加わっているので，回路シミュレーションとしてはかなり簡単な問題になっている．

1. 回路図のサイズは最大100×100文字までに制限される．
2. 斜めの配線は扱わないので，接続に関しては4近傍だけを扱えばよい．
3. 配線には分岐がない．配線は必ず，1つの出力と1つの入力をつなげている．
4. 配線同士が平面上で交叉することはない．
5. ゲートは必ず入力が左，出力が右という使われ方をする．左右反転，あるいは回転して使うことはない．また，拡大縮小して使うことはない．
6. NOTは必ず入力が左，出力が右という使われ方をする．左右反転，あるいは回転して使うことはない．

コンテストでは，仕様を満たす範囲でなるべく簡単なプログラムを作るのが，重要になる．

問題の入力形式を示す．まず，

```
      +------------------+
A-:\  |  +------:\        |
  : )-+  |       : >--?   |
B-:/     |  C--o:/        |
         +----------------+
```

のように回路図を表す行が続く．回路図は`'*'`で始まる（かつ`'*'`だけの）行で終わる．次からは

```
00000000000000000000000000
10000000000000000000000000
```

```
01000000000000000000000000
11000000000000000000000000
00100000000000000000000000
01100000000000000000000000
11100000000000000000000000
```

のように，'A' から 'Z' までの入力に対応した 1 行 26 文字の '1' または '0' からなる行がいくつか続く．'*' で始まる (かつ '*' だけの) 行で終わりとなる．それぞれの入力 1 行に対応する真偽値を '0' か '1' からなる 1 行で出力する．

■方針

今回の問題は特に C++ 向きの問題というわけではなく，むしろ Java 言語向きの問題だが，C++ で解答プログラムを作成してみることにする．

まずは，回路図を解析 (parse) して内部で扱いやすい形式にすることにする．メモリの節約を重視する古き良き時代のプログラミングでは 1 行ずつ読み込んでは解析することが推奨されたが，現代のプログラミングコンテストでは，最大 1 行 100 文字，100 行程度の回路図は当然，全部を読み込んでから解析することにする．

準備のために，元の回路図を表現するデータ型 Map を定義して，それを扱うための関数も用意しておく．

```
// 行(string)のvectorで回路図を表現
typedef vector<string> Map;
// mapの(x,y)にある文字を返す
// 範囲外の場合は ' 'を返す
char mapAt(Map const& map,int x, int y){
  if(x<0 || y<0 || map.size()<=y ||
     map[y].size()<=x)
    return ' ';
  return map[y][x];
}
// mapの(x,y)を文字cに置き換える.
char setAt(Map & map,int x, int y,char c){
  // 範囲外の置き換えは許さない
  assert(0<=x && 0<=y &&
         y<map.size() && x< map[y].size());
  map[y][x]=c;
}
// mapの(x,y)を空白(' ')に置き換える
char clearAt(Map & map,int x, int y){
  setAt(map,x,y,' ');
}
// mapの(x,y)に文字cが入っていることを確かめた
// 上で，空白(' ')に置き換える
char clearCheckAt(Map & map,int x,
                  int y,char c){
  assert(mapAt(map,x,y)==c);
  clearAt(map,x,y,' ');
}
```

clearCheckAt のように assert を通すためだけの関数を用意しておくのも動くプログラムを素早く作るためには大切である．

回路図を解析して，この問題の A ～ Z の 26 個の bool 値の入力 (ここでは，入力行の '0' または '1' が 26 個並んだ行をそのまま与える) 1 組に対して 1 つの bool 値を返す関数に対応するオブジェクトに変換する．このオブジェクトに対応する抽象クラスとして，まず，LogicFunction を定義する．

```
class LogicFunction{
public:
  virtual bool
  getValForVars(string const& vars)=0;
  virtual ~LogicFunction(){}
};
```

LogicFunction の子クラスとして，Var, And, Or, Not 等のクラスを定義する．以下に例として And の定義を示す．

```
class And : public LogicFunction{
  LogicFunction *in1,*in2;
public:
  And(LogicFunction *in1,
      LogicFunction *in2)
      :in1(in1),in2(in2){}
  virtual bool
  getValForVars(string const& vars){
    return in1->getValForVars(vars) &&
           in2->getValForVars(vars);
  }
  virtual ~And(){
    delete in1;
    delete in2;
  }
};
```

きちんと destructor を再帰的に呼ぶ構造にしないと，再利用できないメモリ領域が生じてしまう．

この問題では，配線の分岐がないという条件から，ポインタが一度しか参照されないという保証があるので，上のコードのように destructor 中で内部のポインタを無条件で delete してもよいが，配線の分岐があるような回路に適用するには，参照カウント等に対応したスマートポインタを使う必要がある．これは，boost には入っているが，標準 C++ の機能だけで実現

するには，自分で定義する必要がある．

もっとも，コンテストの場合は，メモリの再利用を考えなくてもメモリ不足にならないような問題設定が選ばれることが多いので，destructor を書くのをさぼる手もある．

ここまで準備すれば，後は回路図から Logic Function へ変換する解析部分さえ定義すればよいことが分かる．

■回路図から関数への変換

関数 parse を書くにあたって，行儀のよいプログラムでは元の入力テキストを読み込んだデータは壊さずに，解析した結果だけを作成するのが普通である．しかし，ここでは，解析中の状態を記憶するための別領域を用意するのが面倒なので，解析の済んだ部分を消しつつ解析を進めていくことにする．

この問題では，'?' が 1 回だけ現れ，そこから全回路図をたどるというトップダウン解析が可能なので，その方針に従って書き進める．

まずは，'?' を探すプログラムを書く．座標は STL の pair で表す．

```
pair<int,int> findQuestion(Map const& map){
  int y=0;
  for(Map::const_iterator it=map.begin();
      it!=map.end();++it,++y){
    int x=(*it).find('?');
    if(x>=0) return make_pair(x,y);
  }
  return make_pair(-1,-1);
}
```

配線をトレースして，スペースに変換しつつ，たどり着いた反対側の座標を返す関数 parseLine を定義する．

```
// 配線の末端にくる文字かどうかの判定
bool isPartEnd(char c){
  return c=='o' || c=='>' || c==')' ||
         c==':' || isupper(c);
}

// 横方向の配線と接続を持つ文字かどうかの判定
bool hasHorConnection(char c){
  return c=='-' || c=='+' ||
         c=='?' || isPartEnd(c) ;
}

// 縦方向の配線と接続を持つ文字かどうかの判定
bool hasVertConnection(char c){
  return c=='|' || c=='+' ||
         c=='?' || isPartEnd(c) ;
}

// mapを空白に置き換えながら配線をトレースする
// 配線以外の要素にたどり着いたら，その座標を返す
pair<int,int> parseLine(Map & map,int x,
                        int y){
  char c=mapAt(map,x,y);
  if(isPartEnd(c))
    return make_pair(x,y);
  clearAt(map,x,y);
  //横方向の接続を持つ文字
  if(hasHorConnection(c)){
    // 左隣が横方向の接続を持つ
    if(hasHorConnection(mapAt(map,x-1,y)))
      return parseLine(map,x-1,y);
    // 右隣が横方向の接続を持つ
    if(hasHorConnection(mapAt(map,x+1,y)))
      return parseLine(map,x+1,y);
  }
  //縦方向の接続を持つ文字
  if(hasVertConnection(c)){
    // 上が縦方向の接続を持つ
    if(hasVertConnection(mapAt(map,x,y-1)))
      return parseLine(map,x,y-1);
    // 下が縦方向の接続を持つ
    if(hasVertConnection(mapAt(map,x,y+1)))
      return parseLine(map,x,y+1);
  }
  //ここにくるのは間違った入力の場合
  assert(0);
}
```

配線に沿って 1 文字進むたびに再帰呼び出しをしている．もちろん，一度行ったら戻る必要のない再帰呼び出しなので，ループで書き直すことは可能だが，再帰で書いた方がずっとシンプルに書けるので，プログラミングコンテストの際にはこのように書いた方がよい．

たどり着いた場所にある部品に応じて，Logic Function の子クラスのインスタンスを作成していく．

```
// 配線をたどって部品にたどり着いたら
// そのインスタンスを作成する
LogicFunction* parseFromLine(Map & map,
        int startX, int startY){
  pair<int,int> parsed=
    parseLine(map,startX,startY);
  int x=parsed.first, y=parsed.second;
  char c=mapAt(map,x,y);
  clearAt(map,x,y);
  switch(c){
  // Andの出力部分の文字
```

```
  case ')':{
    // 壊しながら形を確かめる
    clearCheckAt(map,x-1,y-1,'\\');
    clearCheckAt(map,x-1,y+1,'/');
    clearCheckAt(map,x-2,y-1,':');
    clearCheckAt(map,x-2,y,':');
    clearCheckAt(map,x-2,y+1,':');
    // 入力1以下から配線を辿る
    LogicFunction *in1=
      parseFromLine(map,x-3,y-1);
    // 入力2以下から配線を辿る
    LogicFunction *in2=
      parseFromLine(map,x-3,y+1);
    return new And(in1,in2);
  }
  // Orの出力部分の文字
  case '>':{
    clearCheckAt(map,x-1,y-1,'\\');
    clearCheckAt(map,x-1,y+1,'/');
    clearCheckAt(map,x-2,y-1,':');
    clearCheckAt(map,x-2,y,':');
    clearCheckAt(map,x-2,y+1,':');
    LogicFunction *in1=
      parseFromLine(map,x-3,y-1);
    LogicFunction *in2=
      parseFromLine(map,x-3,y+1);
    return new Or(in1,in2);
  }
  // Notの扱い
  case 'o':{
    // 左から接続しているはず
    LogicFunction *in=
      parseFromLine(map,x-1,y);
    return new Not(in);
  }
  default:
    // 入力変数
    if(isupper(c)){
      return new Var(c);
    }
    assert(0);
  }
}
```

これだけ準備をすると，全体を解析する関数は容易に書ける．

```
// 全体のparse
LogicFunction* parse(Map & map){
  pair<int,int> qPos=findQuestion(map);
  int x=qPos.first, y=qPos.second;
  assert(x!= -1  && y!= -1);
  return parseFromLine(map,x,y);
}
```

main関数は，

```
  LogicFunction *sp=parse(map);
```

で作成した LogicFunction を

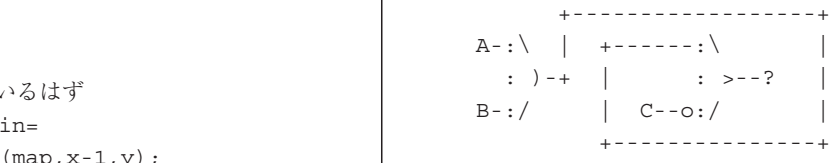
```
  for(;;){
    string line;
    getline(is,line);
    if(line[0]=='*') break;
    else if(line=="") return 0;
    std::cout << sp->getValForVars(line)
              <<std::endl;
  }
```

のように，おのおのの入力に対して適用していくだけでよい．これが，おそらく出題者の想定した解答と思われる．

■書き換えによる直接解法

これだけでは面白くないので，回路図はそのままの形で保持しておいて，特定の入力に対する値を求めるたびに，回路図のコピーを作成し，このコピーを値に応じて書き換えていく方法を考える．

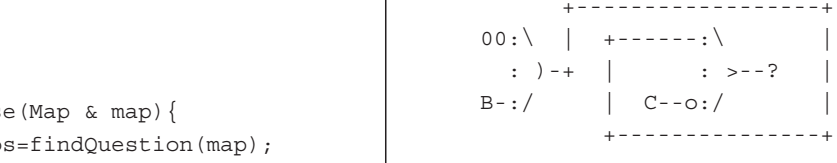
たとえば，



という回路図に，

```
01000000000000000000000000
```

という入力を与えた場合を考える．今度は，'?' ではなく，入力 'A'－'Z' を探していく．最初に 'A' が見つかる．この入力では 'A' の値は 0 なので，配線に沿ってこの値をを伝播していきつつ，回路図も以下のように書き換える．



AND ゲートの入力に至るが，AND は片方が 0 なら必ず 0 になるので，そこから 0 を伝播していく．もう片方の入力は不要になるので，ついでに AND ゲートを壊す．

```
        00000000000000000000
   000    0  0000000:\        0
     00000  0       : >--?   0
   B-       0  C--o:/        0
            00000000000000000
```

OR ゲートは，入力が 0 の場合は，もう一方の入力が出力に直結されると見なしてよいので，そのように書き換える．

```
        00000000000000000000
   000    0  0000000          0
     00000  0       +----?   0
   B-       0  C--o+         0
            00000000000000000
```

ここで，伝播は途切れるので，次の入力を探していき，'B' が見つかる．1 を伝播するがすぐに途切れてしまう．

```
        00000000000000000000
   000    0  0000000          0
     00000  0       +----?   0
   11       0  C--o+         0
            00000000000000000
```

次に，'C' が見つかり，0 を伝播していく．途中で NOT を見つけて反転する．

```
        00000000000000000000
   000    0  0000000          0
     00000  0       111111   0
   11       0  00001          0
            00000000000000000
```

今回は，'?' にたどり着き，'?' を 1 で書き換えたので，出力は 1 と分かる．

プログラムは以下のようになる．

```
// 0と1を文字として反転する.
char alt(char boolCode){
  if(boolCode=='1') return '0';
  assert(boolCode=='0');
  return '1';
}

// '?'に着いたかをreturn valueにする.
// '?'に着いた場合はretに真偽値を入れる
// boolCodeが伝播しようとする真偽値
bool propagateSignal(Map& map,int x, int y,
                     char boolCode,
                     bool &ret){
  char c=mapAt(map,x,y);
  setAt(map,x,y,boolCode);
  if(c=='?'){
    ret=boolCode;
    return true;
  }
  else if(isPartEnd(c)){
    // Notでは反転
    if(c=='o'){
      return
        propagateSignal(map,x+1,y,
                        alt(boolCode),
                        ret);
    }
    // AndかOrの入力
    else if(c==':'){
      // 自分が入力1ならyDirに1が
      // 自分が入力2ならyDirに-1を入れる
      int yDir=1;
      if(mapAt(map,x,y-1)==':')
        yDir=-1;
      else{
        assert(mapAt(map,x,y+1)==':');
      }
      // Orの入力だった
      if(mapAt(map,x+2,y+yDir)=='>'){
        //1ならゲートを破壊して1を伝播
        if(boolCode=='1'){
          setAt(map,x,y+2*yDir,' ');
          setAt(map,x+1,y+2*yDir,' ');
          setAt(map,x+1,y,' ');
          setAt(map,x,y+yDir,'1');
          setAt(map,x+1,y+yDir,'1');
          setAt(map,x+2,y+yDir,'1');
          return
            propagateSignal(map,x+3,y+yDir,
                            '1',ret);
        }
        else{
        //0ならゲートを配線に置き換える
          setAt(map,x+1,y+2*yDir,' ');
          setAt(map,x+1,y,' ');
          setAt(map,x,y+2*yDir,'+');
          setAt(map,x,y+yDir,'+');
          setAt(map,x+1,y+yDir,'-');
          setAt(map,x+2,y+yDir,'-');
          return false;
        }
      }
      // Andの入力だった
      else if(mapAt(map,x+2,y+yDir)==')'){
        //0ならゲートを破壊して0を伝播
        if(boolCode=='0'){
          setAt(map,x,y+2*yDir,' ');
          setAt(map,x+1,y+2*yDir,' ');
          setAt(map,x+1,y,' ');
          setAt(map,x,y+yDir,'0');
          setAt(map,x+1,y+yDir,'0');
          setAt(map,x+2,y+yDir,'0');
          return
            propagateSignal(map,x+3,y+yDir,
```

```
                                '0',ret);
        }
        else{
        //1ならゲートを破壊して配線に置き換える
          setAt(map,x+1,y+2*yDir,' ');
          setAt(map,x+1,y,' ');
          setAt(map,x,y+2*yDir,'+');
          setAt(map,x,y+yDir,'+');
          setAt(map,x+1,y+yDir,'-');
          setAt(map,x+2,y+yDir,'-');
          return false;
        }
      }
    }
    return false;
  }
  // 各方向の配線への伝播
  if(hasHorConnection(mapAt(map,x-1,y)) &&
     propagateSignal(map,x-1,y,
                     boolCode,ret))
    return true;
  if(hasHorConnection(mapAt(map,x+1,y)) &&
     propagateSignal(map,x+1,y,
                     boolCode,ret))
    return true;
  if(hasVertConnection(mapAt(map,x,y-1)) &&
     propagateSignal(map,x,y-1,
                     boolCode,ret))
    return true;
  if(hasVertConnection(mapAt(map,x,y+1)) &&
     propagateSignal(map,x,y+1,
                     boolCode,ret))
    return true;
  // ゲートが破壊されて途切れることがある.
  return false;
}
```

入力を順に探して，入力を見つけたら propagate Signal を呼ぶ関数 directValue は以下のように書ける．

```
/*
 * mapはコピーするが
 * lineは参照のみ
 */
bool directValue(Map map,string const&
vars){
  // 各行について
  for(int y=0;y<map.size();y++){
    string line=map[y];
    for(int x=0;x<line.size();x++){
      char c=line[x];
      // 入力を探す
      if(isupper(c)){
        // その入力の真偽値
        char boolCode=
          vars[Var::charToIndex(c)];
        assert(boolCode=='1' ||
               boolCode=='0');
        setAt(map,x,y,boolCode);
        bool ret;
        // '?'までたどり着いたらretに帰る.
        if(propagateSignal(map,x,y,
                           boolCode,ret))
          return ret;
        cerr << map << endl;
      }
    }
  }
  assert(0);
}
```

こちらも呼び出し方は，ほぼ同じである．

```
    for(;;){
      string line;
      getline(is,line);
      if(line[0]=='*') break;
      else if(line=="") return 0;
      std::cout <<directValue(map,line)
                <<std::endl;
    }
```

こちらのプログラムは，1 種類の入力ごとに map のコピーを作成して，それを書き換えていくので，効率は明らかに前のプログラムよりも劣る．

しかし，このような効率の悪いプログラムでも，プログラミングコンテストの条件下では余裕を持って時間制限内に終了する．先ほどのプログラムと比較すると，デバッグの際に途中の状態を print して見やすい点は有利である．

先ほどのプログラムと，このプログラムと，どちらがよいかは一概にはいえないだろう．コンテスト参加者の得手不得手に従った解法を採用するのがよいものと思われる．

（平成 15 年 10 月 20 日受付）