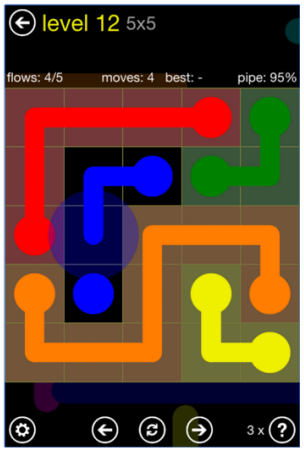
Flow game solver , report　工科海洋 莊博翰

目標: 寫出一個可以解 flow game 的 solver



概述

基本上 flow game 是一個 np complete problem ，用暴力法可能只能解出 5*5 或 6*6，而我希望的結果是可以解 15*15，也就是 flow game 題庫中最難的題目。為此我希望可以使用 sat solver 來幫助我達成這個目標。然而想要使用 sat solver ，我必須想辦法將問題轉換成一個 Boolean CNF expression，以下我將一步一步描述我的解法，與曾經遇到的問題

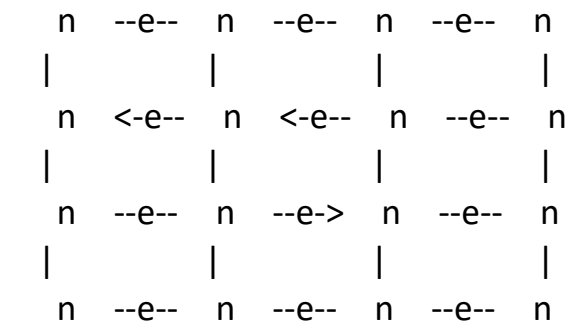
第一步，**Variable representation**

首先我要先釐清如何去表示一個 board(也就是一個版面狀況)
我決定將其拆為 node 和 edge ，因此
假設現在是 4*4board 有 16 個 node ，24 個 edge

我們可以用 C1，C2，C3…..C16 來表示 node 的顏色
(Cn 是(cn1,cn2,cn3…cnk) k 個 Boolean variable 組成的， 2^k > 總 color 數
可視為一個 k bit unsign int)

並用 D1,D2,D3….D16 表示 node 與該顏色起點的距離
(Dn 是 4 bit number 2^4=16 ,最長的聯通距離)

並用 e1, e2, e3…$e_{4*3*2=24}$ 來表示 edge 有無接通，方向
(en ➔ (enC , enD) : (連通，方向:(右，上 :true，左，下 false))
enC , enD 是 Boolean variable)



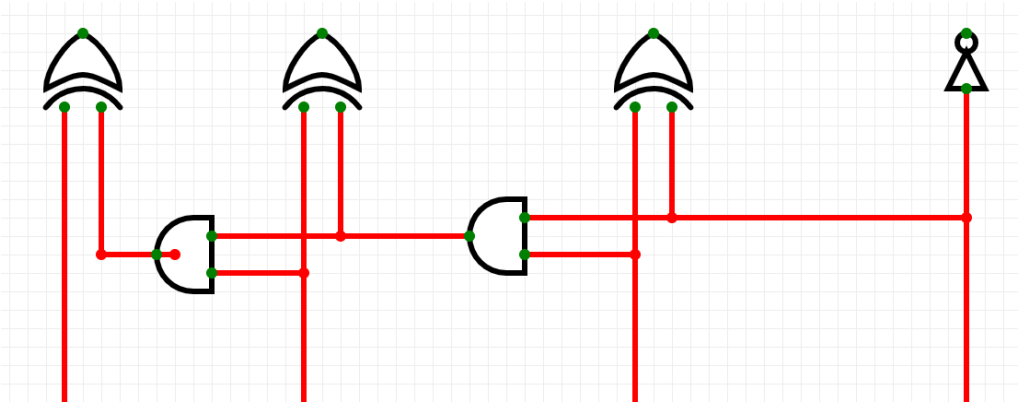
board 可以被拆解成 node 跟 edge (有方向)
(--e-> or <-e-- or --e-- : disconnect)

由上述條件可發現，我們會頻繁的使用 n bit number，也就是我們將來可能會不斷使用一些邏輯表達如 number1 = number2(兩顏色,距離相同)，number1=number2+1(距離 1 =距離 2 加一) (注意到這裡的 = 並非 code 中的 assign，而是數學中的 =,相等)，也就是我希望可以達成如下功能
指定 num1 = num2 , num3 = num4 , num2+1 = num3 , num1=0 ,
solve ➔ num1=0, num2=0 ,num3=1, num4=1 ,或
指定 num1=num2+1 , num2=num1+1
solve ➔ 無解
可以發現，這跟 sat solver 有十足的相似感，因此我就寫了一個 class

```
class SATnumber{
    public:
    SATnumber(int nbits,SatSolver&s): solver(s){
        assert(nbits>0);
        for(int i=0;i<nbits;i++){
            numberid.push_back(solver.newVar());
            //cout<<" num : "<<numberid[i];
        }
    }
    // assert this number always be 0 ( unused )
    void assertzero();
    // assert this number always be N
    void assertisN(int n);
    // return a SATnumber that always greater than this number by 1
    SATnumber addbyone();
    // two number is equal  <==> return variable(var in solver)
    int assertequalto(SATnumber);
    // value of this number
    int value();
    string tostring(){
        stringstream s1;
        s1<<" bits "<<numberid.size();
        return s1.str();
    }
    private:
    vector<int> numberid;
    SatSolver& solver;
```

基本上可以看出這個 SATnumber 就是一個用許多 SAT Var 組成的結構
功能不外乎是指定: num =0 , num=N , num+1 , num1= num2
這裡可以注意到， addbyone() 這個 function 回傳的是一個 SATnumber
也就是它傳回去的 SATnumber re = this +1 (always hold)example:
SATnumber re = now.addbyone()
(now.value()==k <==> re.value()==k+1 || unsat)
如果要以電路的方式來敘述這個 function 加入 solver 的 constraint 會是如下

re.numberid[3] re2 re1 re0



now.numberid[3] now2 now1 now0

另一個重要的 function 是 equal() ，也就是指定兩 SATnumber 相等，
也就是(num1bit3==num2bit3 && num1bit2==num2bit2 && num1bit1==num2bit1…
不過為了使這個 constraint 更有彈性，我加了一個 Var re，使 constraint 變成
re <==>(num1bit3==num2bit3 && num1bit2==num2bit2……)
也就是將來可以用 Var re 來表示這兩個 SATnumber 是否相等。
為此我在 sat.h 中加了一些 function 如

```
void addEqualCNF(Var vf, Var va, bool fa, Var vb, bool fb)
void addImpliedCNF(Var vf, Var va, bool fa, Var vb, bool fb)
void addORCNF(Var vf, Var va, bool fa, Var vb, bool fb)
void addNotCNF(Var vf,Var va)
```

還有多個 fanin 的 aig gate

```
int addmultiAigCNF(const vector<int>& vas,const vector<bool>& invs)
```

SATnumber 這個 class 的大致結構就差不多是這樣，有了這個 class 之後，
就像有了一個好用的工具，可以降低接下來 codeing 的複雜程度。

setting constraints

回到我們的主題，Flow game，我發現 flow game 完成的條件可以列為以下三點

1: node 需連接剛好 1 fanin ,1 fanout ，起點需連接 1 fanout
終點需連接 1 fanin (N <--e-- : fanin ， N —e→ :fanout)

2. 一個有接通的 edge 兩端要同色

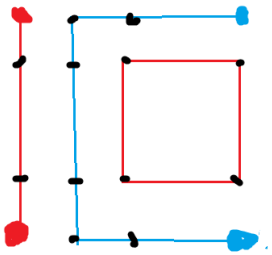
如果單看 constraints 1,2 我們可能會解出有 loop 的結果

3. (容易被忽略的難點) 避免產生 illegal loop

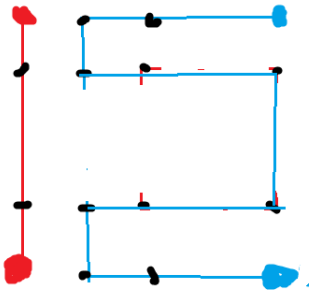
:任兩個接通的 node ，fanin 端的 Di == fanout 端的 Dj + 1

(Node i (Di) <--e--- Node j (Dj))

illegal loop (沒有接到頂點的圈)



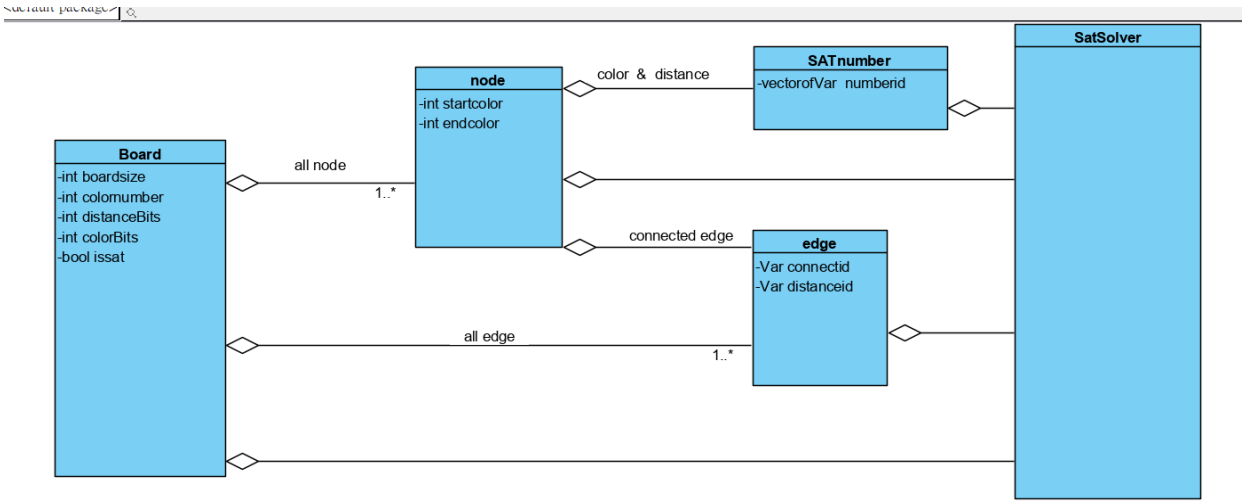
which should be



單看前兩個 constraint 會發現其實已經可以解大部分的圖，然而在空位比較多的問題中容易出現 illegal loop(我是這麼認為)

有了 constraints 後，我們要把它轉成 CNF 才能使用 SAT solver

Code overview



constraints to boolean logic

例: 假設已知 有兩色 紅藍 的起點 s➔start e➔end

C1s (D1) e1 C2(D2) e2 C3(D3) e3 C4e(D4)

e13 e14 e15 e16

C5 e4 C6 e5 C7 e6 C8

e17 e18 e19 e20

C9 e7 C10 e8 C11 e9 C12

e21 e22 e23 e24

C13e e10 C14s e11 C15 e12 C16

by constraint 1 : node 需連接剛好 1 fanin ,1 fanout ，起點需連接 1 fanout 終點 1 fanin。edge 表示法如下

(en ➔ (enC , enD) : (連通，方向:(右，上 :true，左，下 false))

e1 ➔ (e1C: 1 , e1D: 0) 表示 e1 連通，方向向左。

constraint:

起點: (e1C xor e13C) (e1C ➔ e1D)(e13C ➔ e13D') …

接通其一 接通的 edge 是 fanout

終點: (e21C xor e10C) (e21C ➔ e21D')(e10C ➔e10D')….

接通其一 接通的 edge 是 fanin

一般的以 Node 7 為例

: onlyOneHold(e15C e15D, e5C e5D' , e6C e6D ,e19C e19D')

選一 edge 當 fanout

onlyOneHold (e15C e15D' , e5C e5D , e6C e6D' , e19C e19D)….

選一 edge 當 fanin ，同時 fanout 選過的 fanin 選不到，如:

e15C e15D ➔true ，e15C e15D'➔false

實際在 class Node 中

```
    // start connects one fanin, end node  connects one fanout
    void assert1In1OutForStartEnd(const vector<int>& isconnect,const ve
ctor<int>& isfanout,const vector<bool>& inv);
    // except start and end node, each node connects one fanin and one
fanout
    void assert1In1Out(const vector<int>& isconnect,const vector<int>&
pointTohigh);
```

的確也分為起點終點，和其他點考慮。

同時以上用到的 onlyOneHold 也寫在 sat.h 中

```
    // Clause: ( ab'c' +a'bc' + a'b'c )  ==> only one true
      void AlwaysOnlyOneTrue(const vector<int>& its){
```

by constraint 2 一個有接通的 edge 兩端要同色

⇨ (e1C ➔ C1==C2)(e2C ➔ C2==C3) ...(C1==r)(C13==r)(C4==b)(C14==b)

by constraint 3

⇨ 任兩個接通的 node ，fanin 端的 Di == fanout 端的 Dj + 1

⇨ 以 edge 5 node 6,7 為例 :

⇨ (e5C e5D ➔ D6+1==D7) (e5C e5D' ➔ D6==D7 +1)

⇨ 並且起點是 0

⇨ (D1==0)(D14==0)

這裡所提到的同色，距離加一，距離是零，設定顏色等概念

其實都可對應到 SATnumber:

同色:SATnumber.equal()

距離加一:SATnumber.addbyone()

距離是零，設定顏色: SATnumber.assertToN()

```
void Edge::assertSameColor( SATnumber& highcolor,SATnumber& lowcolor){
  assert(!alreadycolor);
  int consttrue=solver.newVar();
  solver.addImpliedCNF(consttrue,connectid,false, highcolor.assertequal
to(lowcolor) ,false);
  solver.assertProperty(consttrue,true);
  alreadycolor=true;
}

void Edge::assertDistanceDiffBy1(SATnumber& highdis,SATnumber& highdisP
1,SATnumber& lowdis,SATnumber& lowdisP1){
  assert(!alreadydis);
  int consttrue=solver.newVar();
  solver.addImpliedCNF(consttrue,connectAndPoint2H,false,lowdisP1.asser
tequalto(highdis),false);
  solver.addImpliedCNF(consttrue,connectAndPoint2L,false,highdisP1.asse
rtequalto(lowdis),false);
  solver.assertProperty(consttrue,true);
  alreadydis=true;
}
```

其 code 也滿簡單的，可以注意到在做

```
assertDistanceDiffBy1()
```

時，要傳入兩側現在的距離與兩側現在的距離加一

到此為止難的地方基本上已經結束了，只差把全部整合起來

Class Board:

基本上也沒做啥大不了的，無非是把檔案讀入，建好所有的 Node edge (分 row column)　還有唯一的 solver　。

所有 data 的本體都在這裡，其他地方存的都是 reference

```
    SatSolver solver;
    vector<Node> nodes;
    vector<Edge> edger;
    vector<Edge> edgec;
```

然後連接所有的 node edge (也就是把 constrains 都建好)

```
    bool connectAllEdgeToNode();
    bool connectAllNodeToedge();
```

當然要 solve 一下

```
   bool solve();
```

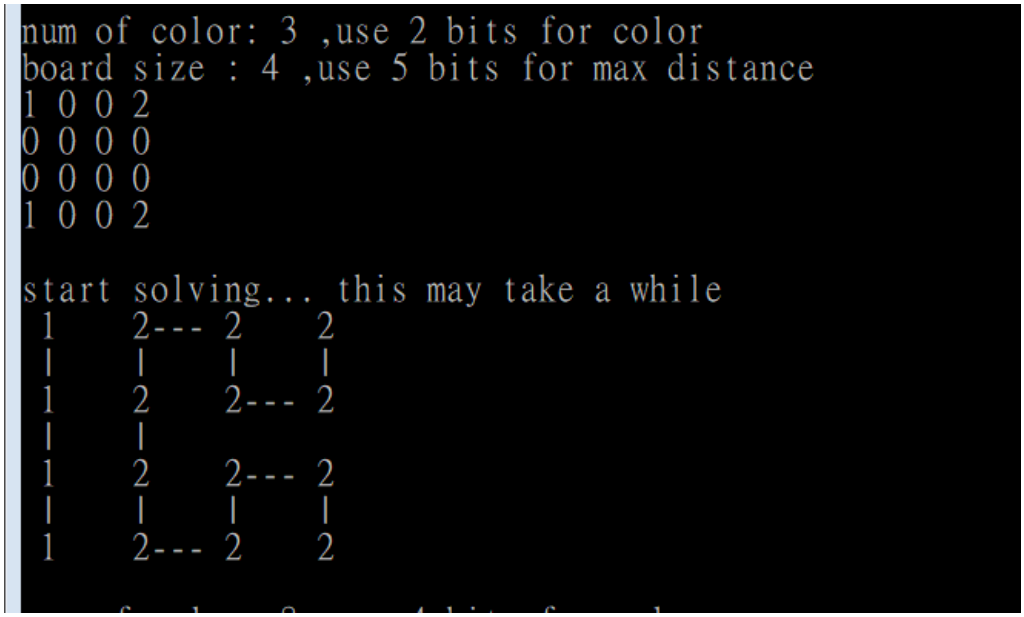
最後就看結果

```
string ans2string();
```

appendix :執行結果

(1)

確實避免了 illegal loop



如果沒有 constraint 3……會怎樣? (board.cpp　68 , 72)

```
 this->connectedNodeRow(i,idh,idl);
// edger[i].assertDistanceDiffBy1(*(nodes[idh].getdi
 edger[i].assertSameColor(*nodes[idh].getcolor(),*nc
```

註解掉 constraint 3　後 出現了 illegal loop

```
num of color: 3 ,use 2 bits for color
board size : 4 ,use 5 bits for max distance
1 0 0 2
0 0 0 0
0 0 0 0
1 0 0 2

start solving... this may take a while
 1    2--- 2--- 2
 |    |
 1    2    3--- 3
 |    |    |    |
 1    2    3--- 3
 |    |
 1    2--- 2--- 2
```

(2) 極限 2min　solving　15*15　board

```
num of color: 14 ,use 4 bits for color
board size : 15 ,use 8 bits for max distance
9 1 7 0 0 0 0 0 7 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 5
0 0 0 0 8 8 0 0 0 5 0 2 4 0 4
0 0 0 0 0 0 0 0 6 6 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 10 0 0 2 0 14 0 0 0 0 0 0 0 0
0 11 0 0 0 0 3 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
9 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 14 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 10 0 0 12 0 0 0 0 0 0 0 0 0 0
0 0 0 0 11 0 0 3 0 0 0 0 0 0 0
13 13 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 12

start solving... this may take a while
 9    1    7--- 7--- 7--- 7--- 7--- 7--- 7    5--- 5    5--- 5    5--- 5
 |    |                                        |    |    |    |    |    |
 9    1    8--- 8--- 8--- 8    1--- 1--- 1    5    5--- 5    5--- 5    5
 |    |    |              |    |         |    |
 9    1    8--- 8--- 8    8    1    1--- 1    5    2--- 2    4--- 4--- 4
 |    |                        |    |              |
 9    1    1--- 1--- 1--- 1--- 1    1    6    6    2    2--- 2--- 2--- 2
 |    |    |                   |    |    |    |    |                   |
 9    1--- 1   14---14---14---14    1    6--- 6    2    2    1--- 1    2
 |              |              |    |              |    |    |    |    |
 9   10---10   14    2--- 2   14    1--- 1--- 1    2--- 2    1    1    2
 |         |    |    |              |                        |    |    |
 9   11   10   14    2--- 2    3--- 3--- 3    1    1--- 1--- 1    1    2
 |    |    |    |    |                   |    |    |              |    |
 9   11   10   14    2--- 2--- 2--- 2    3    1--- 1    1--- 1    1    2
 |    |    |    |                   |    |                   |    |    |
 9   11   10   14   14---14---14    2    3    1    1--- 1    1    1    2
      |    |    |    |         |    |    |    |    |         |    |    |
11---11   10   14   14   14   14    2    3    1    1--- 1    1    1    2
 |         |    |    |    |    |    |    |    |              |    |    |
11   10---10   14---14   14---14    2    3    1    1--- 1    1--- 1    2
 |    |                             |    |    |    |                   |
11   10   11---11   12---12    2--- 2    3    1    1    2--- 2--- 2    2
 |         |    |         |    |         |    |    |    |         |    |
11---11---11   11   11   12    2    3--- 3    1--- 1    2--- 2    2--- 2
               |    |    |    |                                   |
13   13---13   11   11   12    2--- 2--- 2--- 2--- 2--- 2--- 2   12---12
 |         |    |    |    |                                        |    |
13---13---13   11---11   12---12---12---12---12---12---12---12---12   12
```

----------finish----------