P THINK FAST S

Competitive Programming

From Problem 2 Solution in O(1)

Graph Theory

2 Satisfiability (2SAT)

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Needed Background

- Connectivity or SCC - *Not Covered*.
- Truth Value
- Truth Table
- Logical **Implication**
- Implication Graph
- Conjunctive normal form

Connectivity / SCC

- We will discuss 2 solutions
  - Slow: Floyd / DFS
  - Fast: SCC
- Both topics are not covered here
- Revise their videos from my channel

Truth value

- In classical logic the truth values are:
- true (1 or T) and
- untrue or false (0 or $\perp$)

- That is … boolean variables :)
  - bool bVisited = true;
  - bVisited = false;

Truth Table

- Evaluating **all possible** values the logic function can take
  - Functions: NOT $\neg$, OR $\vee$, AND $\wedge$
- 1 boolean values have 2 combinations
- 2 boolean values have 4 combinations
- 3 boolean values have 8 combinations
- We will focus on the basic 3 + implication

Truth Table

Logical Negation

p	$\neg p$
T	F
F	T

Logical Conjunction

p	q	$p \land q$
T	T	T
T	F	F
F	T	F
F	F	F

Logical Disjunction

p	q	$p \lor q$
T	T	T
T	F	T
F	T	T
F	F	F

Logical Implication

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

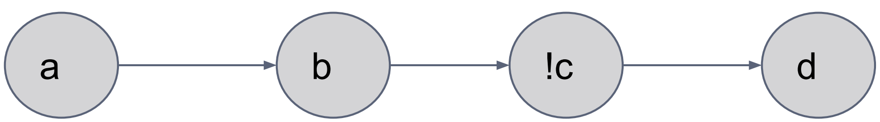
Src: https://en.wikipedia.org/wiki/Truth_table

Logical Implication

- **if p then q** (symbolized as $p \rightarrow q$)
- p is a premise and q is **conclusion**
  - if p is true, q must be true
  - if p is false, q can be whatever
  - That is only 1 case is false: p = true and q = false
  - **P = The sky is overcast.   Q = The sun is not visible.**
  - if sky is overcast → sun is not visible
  - if sky is NOT overcast →sun may or may not be visible (e.g. some eclipse occurs covering sun)
  - Logic fails: if sky is overcast but we see the sun!!!!
- (if p then q) **equivalent to** (if !q then !p)

Implication Graph

- Assume: If a then b. if b then !c. if !c then d
  - Then ⇒ if **a then d**
  - Then ⇒ if a = 1, then b = 1, c = 0, d = 1
- Implication graph
  - Each variable will have **2 nodes**: x and !x
  - Each implication is edge (b →!c) ⇒ Edge (b, !c)



- Then
  - Any path in such graph represents a **new** implications
  - X and !X shouldn't be on a cycle (**both** can't be true)

Implication Graph

- if a then b?
  - If b is true $\Rightarrow$ a must be true
  - If b is false $\Rightarrow$ a must be false
  - equal to: if !b then !a

Logical Implication

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

- if a then b + if b then c?
  - if c = true $\Rightarrow$ a = b = true
  - if c = false $\Rightarrow$ a = b = false
  - if b = true $\Rightarrow$ a = c = true
  - if b = false $\Rightarrow$ a = false, but c = ?
  - There is path: a=> b => c and path !c => !b => !a

Implication Graph

- Set of variables on cycle = ALL same value
  - if true, keep go forward or backward, all assigned true
  - if false, keep go backward, all assigned false
- If we have cycle with value X
  - There must be other cycle with complement node
  - Cycle edges will be switched edges
  - All this cycle is !X
- E.g. Cycle 1 = true: (a, !b), (!b, c), (c, a)
- Then cycle 2 = false: (!a, !c), (!c, b), (b, !a)
- Hint: If have some cycles = compress to a node

Conjunctive normal form

- CNF = a conjunction (and) of clauses, where a clause is a disjunction (or)

$$\neg A \wedge (B \vee C)$$
$$(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$$
$$A \vee B$$
$$A \wedge B$$

2-Satisfiability

- Given CNF with each clause of 2 terms only
- Is it possible to assign variables so than CNF is true?
- $(x1 \vee x2) \wedge (x2^{\overline{\ }} \vee x3^{\overline{\ }}) \wedge (x1^{\overline{\ }} \vee x3)$
  - x1 = 1, x2 = 0, x3 = 1
  - $(1 \vee 0) \wedge (1 \vee 0) \wedge (0 \vee 1) = \text{true}$
- We can do that using 2^n code..Better?

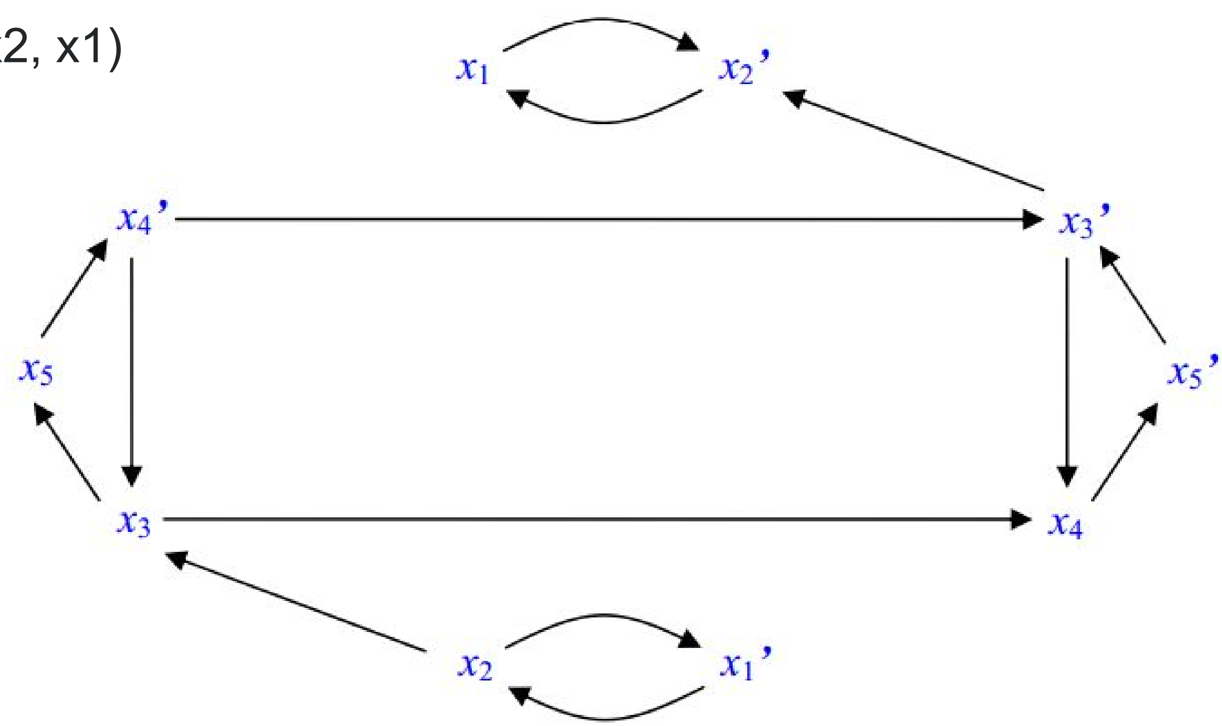
2-Satisfiability

- Thinking: Is it possible to express CNF Clause as implication?
- We need to force each Clause $(a \vee b) = true$?
- What does it imply: $(a \vee b) = true$?
  - **if a = 0, then b must = 1 .. hence (0 or 1) = 1**
  - **if b = 0, then a must = 1 .. hence (1 or 0) = 1**
  - Note: if a = 1 or b = 1 $\Rightarrow$ We can't imply something
- Create **implication graph**
  - clause $\Rightarrow$ 2 edges
  - (a v b) $\Rightarrow$ edge(!a, b) and edge(!b, a)

2-Satisfiability

$(x1 \lor x2) \land (\overline{x2} \lor x3) \land (\overline{x1} \lor \overline{x2}) \land (x3 \lor x4) \land (\overline{x3} \lor x5) \land (\overline{x4} \lor \overline{x5}) \land (\overline{x3} \lor x4)$

x1 ∨ x2 ⇒ E(!x1, x2), E(!x2, x1)
7 clauses = 14 edges
5 vars = 10 nodes



Src: http://cs.stackexchange.com/questions/35425/time-complexity-of-solving-a-set-of-2-sat-instances

2-Satisfiability

- Assume edges (a, b) (b, c) (c, d)
  - Recall: set b = true ⇒ a = c = d = true
  - Recall: set b = false ⇒ a = false .. no constraint on c, d
  - Recall: we have 2 paths: (a ⇒ d) and (!d ⇒ !a)
- What if we have path from x to !x?
  - Let x = true, then !x = true (e.g. x = false) ⇒ **contradict**
  - Let x = false, then no implication constraints on any forward variables. Hence, !x = true ⇒ ok
  - Summary, must: x = false
- What if we have path from !x to x?
  - Similarly, must x = true

2-Satisfiability

- What if we have path from x to !x AND from !x to x (e.g. both on cycle)?
  - We can NOT find correct assignment
- Summary
  - If any 2 variables x and !x on cycle => No solution
  - Otherwise, there is a solution.
  - We can do DFS to check cycles or use SCC
- Note: if path from X to both Y and !Y
  - Then X => !Y => !X is a path too and X = false

Code: Doubling the nodes

- Assume we have N = 5, and need 2N graph
- One mapping:
  - $0 \Rightarrow (0, 1), 1 \Rightarrow (2, 3), 2 \Rightarrow (4, 5)$....
  - Can be coded with xor (x and x^1)
  - Or check if even return odd, if odd return even
- Other mapping
  - $0 \Rightarrow (0, N), 1 \Rightarrow (1, N+1), 2 \Rightarrow (2, N+2)$....
- Write a simple function to NOT(x)
  - Takes i and return its !i to ease code

Slow Approach: build-check

- Create graph of 2N nodes for the N variables
- For each clause (a v b):
  - Add 2 edges (!a, b), (!b, a)
- Compute Transitive Closure
  - Reason - 1: Know if variables on cycle or not.
  - Reason - 2: Know variables forced to be zero
  - Recall: Path x to !x $\Rightarrow$ x = false
  - Recall: Path !x to x $\Rightarrow$ x = true
  - Otherwise...x NOT forced to value
  -
  - For each variable, check if x and !x NOT on **cycle**

Slow Approach: build

```
// Switch between even odd: (0, 1), (2, 3)..
#define NOT(x)    (1^(x))

const int MAX = 100;
int adjMat[MAX][MAX], assigned_val[MAX];
int n, m;

void add_or(int a, int b)
{
    adjMat[NOT(a)][b] = 1;
    adjMat[NOT(b)][a] = 1;
}

void build()
{
    int m;

    lp(i, m)
    {
        int a, b;
        cin>>a>>b;

        add_or(a, b);
    }

    lp(k, n) lp(i, n) lp(j, n)  // transitive closure
        adjMat[i][j] |= adjMat[i][k] & adjMat[k][j];
}
```

Slow Approach: check

```
// -1 (can't assign), 0 (false), 1 (true), 2 (assign later)
int get_value(int i)
{
    int is_off = adjMat[i][NOT(i)], is_on = adjMat[NOT(i)][i];

    if(is_off && is_on) return -1;
    if(is_off) return 0;
    if(is_on) return 1;
    return 2;
}

bool is_solvable()
{
    for(int i = 0; i < n; i+=2)
        if (get_value(i) == -1)
            return false;
    return true;
}

int main()
{
    ...

    if (!is_solvable())
    {
        cout<<"no solution\n";
        continue;
    }
```

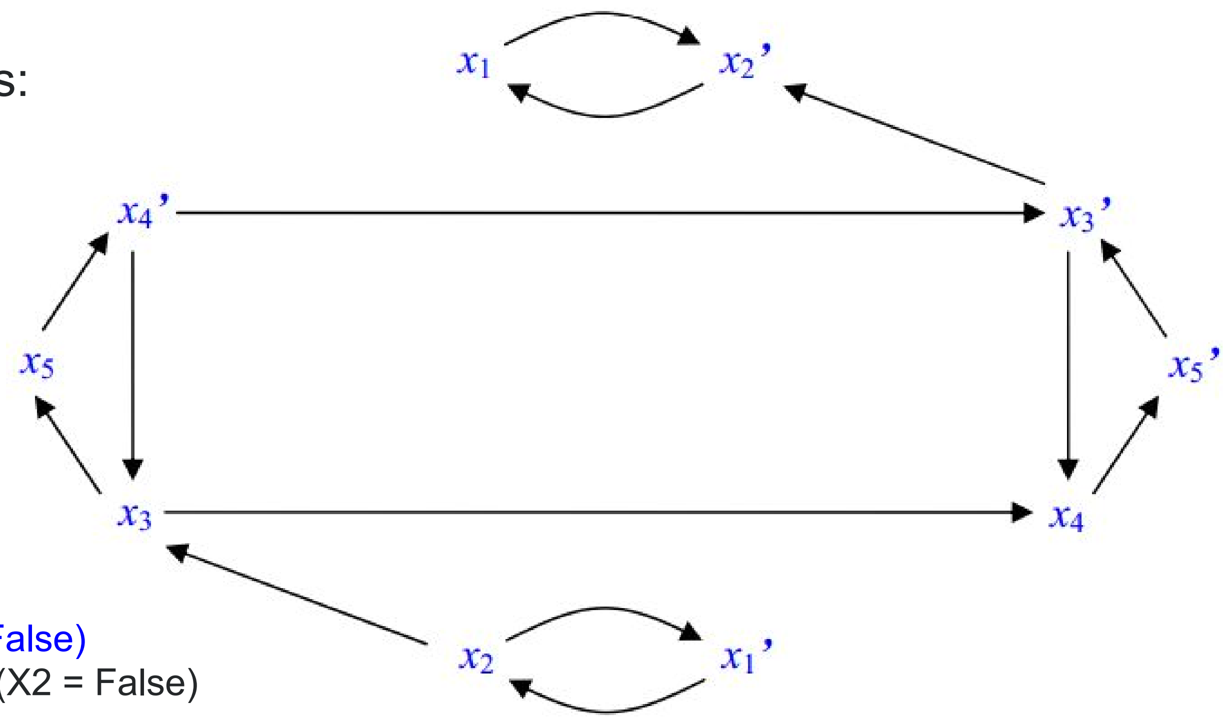
Slow Approach: Assign

- Find unassigned variable that can be true
  - E.g. Not forced to be false (path x to !x)
  - In other words, forced to true or not forced at all
- By implication rules:
  - Any reachable node in the graph must be true too
  - Any node x = true => !x = false
- Keep doing so as long as some variables unassigned

Slow Approach: Assign

Important false variable paths:

- X2 - X2’
- X3 - X3’
- X5 - X5’



- X1? Allowed. X1 = true (X1’ = False)
- Reachable from X1? X2’ = true(X2 = False)
- X1’? Marked
- X2, X2’? Marked
- X3? Must be false. X3’ = true
- X3’? Marked
- X4? Allowed, X4 = true(X4’ = False)
- Reachable from X4? X5’, X3’, X2’, X1 = true (X5, X3, X2, X1’ = false)

Src: http://cs.stackexchange.com/questions/35425/time-complexity-of-solving-a-set-of-2-sat-instances

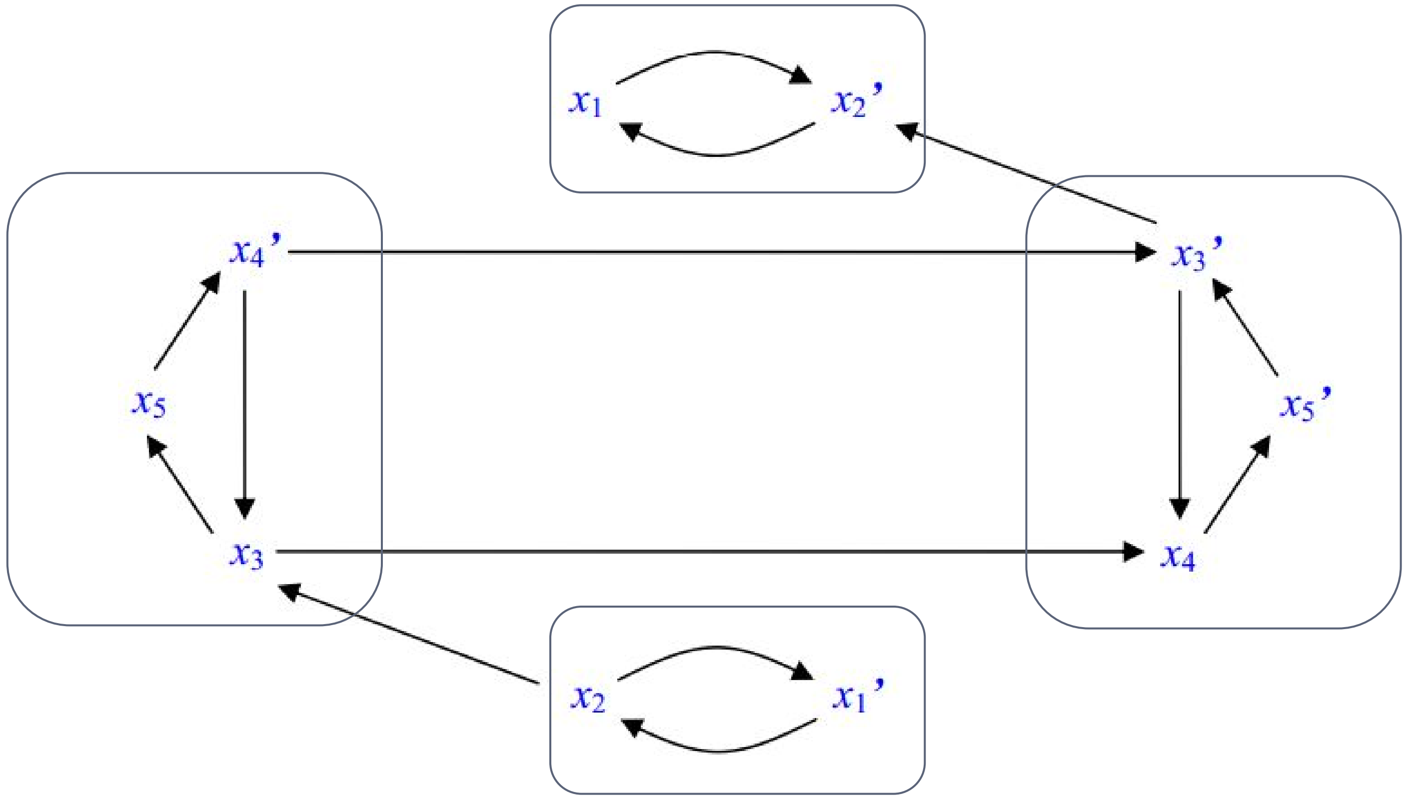
Slow Approach: Assign

```
void assign_on_dfs(int i)
{
    if (assigned_val[i] != -1)
        return;

    assigned_val[i] = 1, assigned_val[NOT(i)] = 0;

    lp(j, n) if(j != i && adjMat[i][j])
        assign_on_dfs(j);
}

void assign_values()
{
    lp(i, n) if (assigned_val[i] == -1)
    {
        if (get_value(i) == 0)
        {
            assigned_val[i] = 0, assigned_val[NOT(i)] = 1;
            continue;
        }
        assign_on_dfs(i);
    }
}
```

Fast Approach

- Build graph
- Find SCC
  - SCC actually can tell us if 2 vars on cycle
  - X and !X if part of same component => cycle => No sol
- Compute Component Graph
  - All nodes on cycle must have same value
  - Complement nodes must be other comp node
  - Recall: if a then b ⇒ if !b then a
- Get reverse topological order for components
- Using tarjan, we get all of this with 1 DFS

Identify SCCs - Component Graph



Checking Satisfiability?

Any component has variable and its negation? E.g. x1 and x1’ ?

No ⇒ then solution exists

```
void add_or(int a, int b)
{
    adjList[NOT(a)].push_back(b);
    adjList[NOT(b)].push_back(a);
}
```

```
bool is_solvable()
{
    for(int i = 0 ; i < n ; i += 2)
        if( comp[i] == comp[NOT(i)] )
                return false;
    return true;
}
```

Tarjan simple modification

- Instead of explicitly building component graph
- Let's define root component node
  - Any node in the component is ok..e.g. root node
- Recall the root node: dfn# = lowlink #
- E.g. if we have 4 components we can have:
  - (0, 10), (1, 7), (2, 6), (3, 12)...(component, node)
- Recall: Tarjan components are reverse topological order: 0, 1, 2, 3 = correct order

Tarjan simple modification

```
vector<int> cmp_root_node;

void tarjan(int node) {
    ....

    if (lowLink[node] == dfn[node]) {
        comps.push_back(vector<int> ());
        int x = -1;
        while (x != node) {
            x = stk.top(), stk.pop(), inStack[x] = 0;
            comps.back().push_back(x);
            comp[x] = sz(comps) - 1;
        }
        cmp_root_node[ comp[node] ] = node;
    }
}
```

Assigning Values

- Order components: reverse topological order
- If component unassigned
  - Set it to 1
  - Set its dual component to 0
- Dual component of C
  - Let node x be the root node of this component
  - Let !x be its dual node...find its component !C
  - Set !C = 0
- Assign all the nodes of component to component value

Assigning Values

$$\begin{tikzcd}
\boxed{\textbf{C1} = x2,\ x1'} \arrow[r] & \boxed{\textbf{C2} = x3,\ x5,\ x4'} \arrow[r] & \boxed{\textbf{C3} = x3',\ x4,\ x5'} \arrow[r] & \boxed{\textbf{C4} = x1,\ x2'}
\end{tikzcd}$$

Notice: C3 = !C2

Find reverse topological order: C4, C3, C2, C1...Assign By order

C4: unassigned. Set to 1 (e.g. x1 = 1, x2 = 0). Set dual component C1 = 0
C3: unassigned. Set to 1 (e.g. x3 = 0, x4 = 1, x5 = 0). Set C2 = 0
C2: assigned already
C1: assigned already

$$(x1 \lor x2) \land (\overline{x2} \lor x3) \land (\overline{x1} \lor \overline{x2}) \land (x3 \lor x4) \land (\overline{x3} \lor x5) \land (\overline{x4} \lor \overline{x5}) \land (\overline{x3} \lor x4)$$

$$(1 \lor 0) \land (1 \lor 0) \land (0 \lor 1) \land (0 \lor 1) \land (1 \lor 1) \land (0 \lor 1) \land (1 \lor 1) \;=\; 1$$

Assigning Values

```
void assign_values()
{
    vector<int> comp_assigned_value(comps.size(), -1);

    lp(i, comps.size()) {
        if (comp_assigned_value[i] == -1){
            comp_assigned_value[i] = 1;
            int not_ithcomp = comp[ NOT(cmp_root_node[i]) ];
            comp_assigned_value[ not_ithcomp ] = 0;
        }
    }

    lp(i, n)
        assigned_val[i] = comp_assigned_value[ comp[i] ];
}
```

CNF and Implications

- Sometimes we can formulate problem easily as CNF to do 2SAT
- Sometimes, you extract implication statements and model them. Then SCC as mentioned.
- Your turn:
  - How to early force variable x to value?
  - Given (a, b) how to allow only (1, 0) (0, 1) but not (1, 1)

Further readings?

- I escaped correctness for 2 assignment methods
- For proofs
- link 1
- link 2
- link 3

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً

Problems

- SPOJ(BUGLIFE, TORNJEVI), UVA(11294, 10319), LiveArchive(4452), CF(228E, 27D, gym-100570-D), TJU(2506), SGU(307)
- http://www.oi.edu.pl/php/show.php?ac=e181113&module=show&file=zadania/oi8/spokojna
- http://web.ics.upjs.sk/ceoi/documents/tasks/party-tsk.pdf