Swift's blog

2-SAT Tutorial

By **Swift**, 2 years ago,

Hi codeforces community.

I thought there is no good *2-SAT* tutorial in the internet, so I decided to write one.

2-SAT is a special case of boolean satisfiability.

Good question! Boolean satisfiability or just *SAT* determines whether we can give values (TRUE or FALSE only) to each boolean variable in such a way that the value of the formula become TRUE or not. If we can do so, we call formula *satisfiable*, otherwise we call it *unsatisfiable*. Look at the example below:

$f = A \ \wedge \ \neg B$, is *satisfiable*, cause A = TRUE and B = FALSE makes it TRUE.

but $g = A \ \wedge \ \neg A$, is *unsatisfiable*, look at this table:

A	$\neg A$	$A \ \wedge \ \neg A$
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE

As you can see g is *unsatisfiable* cause whatever values of its boolean variables are, g is FALSE.

Note: $\neg$ in $\neg X$ is boolean not operation. $\wedge$ in $X \ \wedge \ Y$ is boolean and operation and finally $\vee$ in $X \ \vee \ Y$ is boolean or operation.

→ Pay attention

Before contest
VK Cup 2017 - Qualification Round 1
6 days

Like 82 people like this. Be the first of your friends.

→ shivamgarg210

Rating: **1440**
Contribution: **0**

- Settings
- Blog
- Teams
- Submissions
- Talks
- Contests

shivamgarg210

→ Top rated

#	User	Rating
1	tourist	3534
2	Petr	3400
3	moejy0viiiiiv	3202
4	W4yneb0t	3183
5	ainta	3174
6	bmerry	3089
7	mnbvmar	3059
8	Merkurev	3055
9	V--o_o--V	3031
10	izrak	2996

Countries | Cities | Organizations View all →

→ Top contributors

#	User	Contrib.
1	rng_58	170
2	Errichto	168
3	Petr	163
4	Swistakk	152
5	Zlobober	150
6	csacademy	149
7	GlebsHP	148
8	zscoder	147
8	Um_nik	147
10	Xellos	136

SAT is a `NP-Complete` problem, though we can solve *1-SAT* and *2-SAT* problems in a polynomial time.

1-SAT

Note: This doesn't really exist, I define it cause it help understanding *2-SAT*.

Consider $f = x_1 \wedge x_2 \wedge \ldots \wedge x_n$.

Problem: Is f *satisfiable*?

Solution: Well *1-SAT* is an easy problem, if there aren't both of x_i and $\neg x_i$ in f, then f is *satisfiable*, otherwise it's not.

2-SAT

Consider $f = (x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \ldots \wedge (x_n \vee y_n)$.

Problem: Is f *satisfiable*?

But how to solve this problem? $x_i \vee y_i$ and $\neg x_i \Rightarrow y_i$ and $\neg y_i \Rightarrow x_i$ are all equivalent. So we convert each of $(x_i \vee y_i)$ s into those two statements.

Now consider a graph with $2n$ vertices; For each of $(x_i \vee y_i)$ s we add two directed edges

1. From $\neg x_i$ to y_i
2. From $\neg y_i$ to x_i

f is not *satisfiable* if both $\neg x_i$ and x_i are in the same SCC (Strongly Connected Component) (Why?) Checking this can be done with a simple Kosaraju's Algorithm.

Assume that f is *satisfiable*. Now we want to give values to each variable in order to satisfy f. It can be done with a topological sort of vertices of the graph we made. If $\neg x_i$ is after x_i in topological sort, x_i should be `FALSE`. It should be `TRUE` otherwise.

Some problems:

- SPOJ — BUGLIFE
- SPOJ — TORNJEVI
- UVa — Manhattan
- UVa — Wedding
- CF — The Road to Berland is Paved With Good Intentions
- CF — Ring Road 2
- CF — TROY Query
- CEOI — Birthday party — Solution

Pseudo Code

```
func dfsFirst(vertex v):
    marked[v] = true
    for each vertex u adjacent to v do:
        if not marked[u]:
            dfsFirst(u)
    stack.push(v)

func dfsSecond(vertex v):
    marked[v] = true
    for each vertex u adjacent to v do:
```

```
        if not marked[u]:
            dfsSecond(u)
    component[v] = counter

for i = 1 to n do:
    addEdge(not x[i], y[i])
    addEdge(not y[i], x[i])
for i = 1 to n do:
    if not marked[x[i]]:
        dfsFirst(x[i])
    if not marked[y[i]]:
        dfsFirst(y[i])
    if not marked[not x[i]]:
        dfsFirst(not x[i])
    if not marked[not y[i]]:
        dfsFirst(not y[i])

set all marked values false
counter = 0
flip directions of edges // change v -> u to u -> v

while stack is not empty do:
    v = stack.pop
    if not marked[v]
        counter = counter + 1
        dfsSecond(v)

for i = 1 to n do:
    if component[x[i]] == component[not x[i]]:
        it is unsatisfiable
        exit
    if component[y[i]] == component[not y[i]]:
        it is unsatisfiable
        exit

it is satisfiable
exit
```

2sat, **2-sat**

+144 | Swift | 2 years ago | 12

Comments (12)

Write comment?

hellman_

2 years ago, # | **+5**

How can you topsort if there can be cycles? (between different variables, e.g. x -> y -> x). Or you do it on metagraph?

→ Reply

Swift

2 years ago, # ^ | ← Rev. 2 **0**

If x and not x are both in a cycle: unsatisfiable

Else it doesn't matter if x comes first in topsort or y (x and y are in a cycle)

see my solution for 228E

see my solution for 228E
→ Reply

rex321
2 years ago, # ^ | ☆ 0

It works because you integrate the topsort in Kosaraju's algorithm. In general a topsort only works on DAG's. If somebody would implement Kosaraju and Topsort seperatly it might fail, maybe you should mention that in the write-up. Otherwise nice tutorial.
→ Reply

hung06061995
2 years ago, # ^ | ☆ 0

If there is a cycle which you can dectect while using toposort, function f is not satisfied.
→ Reply

Swift
2 years ago, # ^ | ☆ ← Rev. 2 0

You're wrong. Consider this example:

$f = (x\ \ or\ \ y)\ \ and\ \ (not\ \ x\ \ or\ \ not\ \ y)$ is satisfiable with x = `TRUE` and y = `FALSE` but corresponding graph has two cycles.

- (x -> not y -> x)
- (y -> not x -> y)

→ Reply

hung06061995
2 years ago, # ^ | ☆ +3

Oh, sorry. Thanks so much :D
→ Reply

alv-r-
2 years ago, # ^ | ☆ +8

You don't need the toposort to check if it's satisfyable (just check if x and ¬x are in the same component). If you need to find some values that satisfy the formula, all that really matters is the topological sorting of the components (not of the actual nodes).

You just need to set a value to a element there (the first component in topological order) and propagate it (according to the logical implication operation and to the fact that if A=True you must set ¬A=False). If you try to propagate to an element that already has a true or false value (and it doesn't clash with the one you're trying to set) you can simply skip it, cause it means you've already propagated from it, so the cycles are not a problem.

Tarjan's algorithm generate the SCCs already in reverse topological order (Kosaraju's generates them in actual topological order if I'm not mistaken), so you don't need anything else apart from the SCC algorithm.

Also, a nice implementation trick (this is from Competitive Programming 3) is to use variable i as node number 2*i and ¬i as (2*i)+1. This way, you can access each one by xoring the other with one. (ex: n(i)^1 will give you n(¬i), and n(¬i)^1 will give you n(i), where n(x) is the node representing variable x).
→ Reply

2 weeks ago, # ^ | ☆ 0

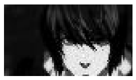
You might need to be careful with forward propagation. For brycesandlund

You might need to be careful with forward propagation. For example, if there's a clause (¬a v ¬a), then the implication a -> ¬a can only be satisfied with a = false.

If you use Kosaraju's algorithm and assign strongly connected components incrementally, that is, the first component found is assigned 1, the second 2, and so on, then you can just compare SCC numbers to determine what is first in topological sort.

Then you assign a = false if SCC[a] < SCC[¬a] and true otherwise.

→ Reply



sat123

2 days ago, # ^ | ☆ 0

this ofcourse will happen very rarely, but if you assign 0 and 1's randomly it would mean hours of debugging if something like this comes up.

→ Reply

mahim007

16 months ago, # | ☆ 0

some one pls provide me some straight forward implementation of 2 sat. **Swift** would you give me the full source code??pls?

→ Reply

mostafa.saad.fci

9 months ago, # | ☆ ← Rev. 2 +3

I have recently created power point for the 2 SAT to explain in competitive programming in my Arabic Channel. Ppt in English.

You may find it useful: https://goo.gl/gBnzKK

→ Reply

abinash

4 months ago, # | ☆ 0

Nice tutorial, Thanks.

→ Reply