Computing Theory

SAT Solvers

SAT Solvers

- SAT is NP-Complete
- In practice SAT-solvers routinely solve instances of thousands of variables

Solving Hard Problems

- One way to solve a hard NP-complete problem is to reduce it to SAT
- Then use a SAT solver (example: MiniSAT)

MiniSAT

- You can install MiniSAT on your machine but you can also use an online MiniSAT here

https://msoos.github.io/cryptominisat_web/

How to use MiniSAT

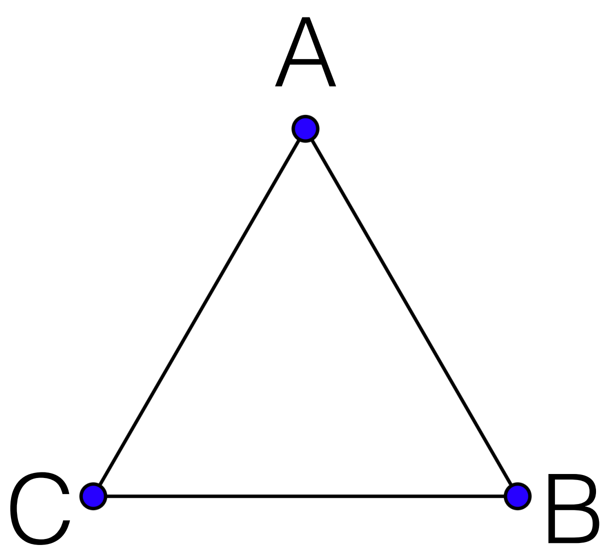
- Consider boolean expression

$$(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$$

- 3 clauses 3 variables
- Format for MiniSAT is

```
p cnf 3 3
1 -2 0
2 -3 0
-1 3 0
```

3-Coloring to SAT

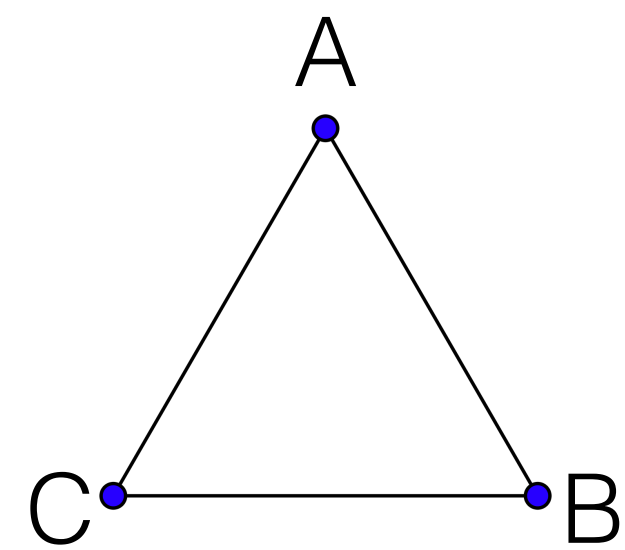


3 nodes -> 12 boolean variables

$A_{red}, A_{blue}, A_{green},$
$B_{red}, B_{blue}, B_{green},$
$C_{red}, C_{blue}, C_{green},$

$A_{blue} = T, A_{red} = F, A_{green} = F$ means
A is colored with color Blue

3-Coloring to SAT



- Node Constraints
  Exactly one color can be assigned to each node

- For node A

$$(A_{red} \vee A_{blue} \vee A_{green}) \wedge \neg(A_{red} \wedge A_{blue}) \wedge \neg(A_{red} \wedge A_{green}) \wedge \neg(A_{green} \wedge A_{blue})$$

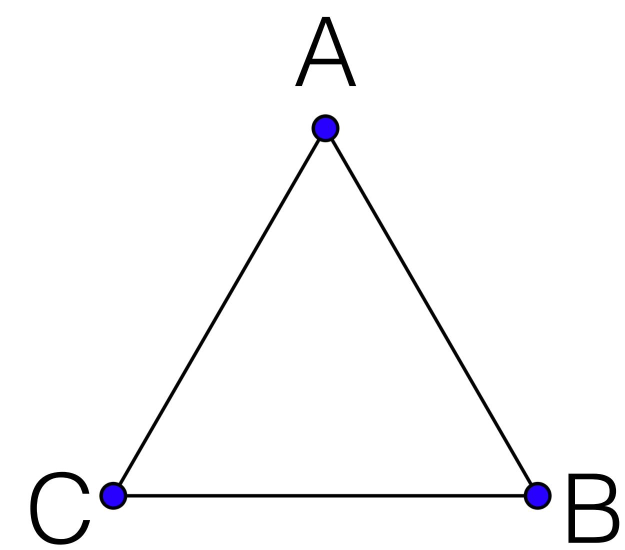
- Converted to CNF

$$(A_{red} \vee A_{blue} \vee A_{green}) \wedge (\neg A_{red} \vee \neg A_{blue}) \wedge (\neg A_{red} \vee \neg A_{green}) \wedge (\neg A_{green} \vee \neg A_{blue})$$

- For all nodes:

$$(A_{red} \vee A_{blue} \vee A_{green}) \wedge (\neg A_{red} \vee \neg A_{blue}) \wedge (\neg A_{red} \vee \neg A_{green}) \wedge (\neg A_{green} \vee \neg A_{blue})$$
$$\wedge (B_{red} \vee B_{blue} \vee B_{green}) \wedge (\neg B_{red} \vee \neg B_{blue}) \wedge (\neg B_{red} \vee \neg B_{green}) \wedge (\neg B_{green} \vee \neg B_{blue})$$
$$\wedge (C_{red} \vee C_{blue} \vee C_{green}) \wedge (\neg C_{red} \vee \neg C_{blue}) \wedge (\neg C_{red} \vee \neg C_{green}) \wedge (\neg C_{green} \vee \neg C_{blue})$$

3-Coloring to SAT



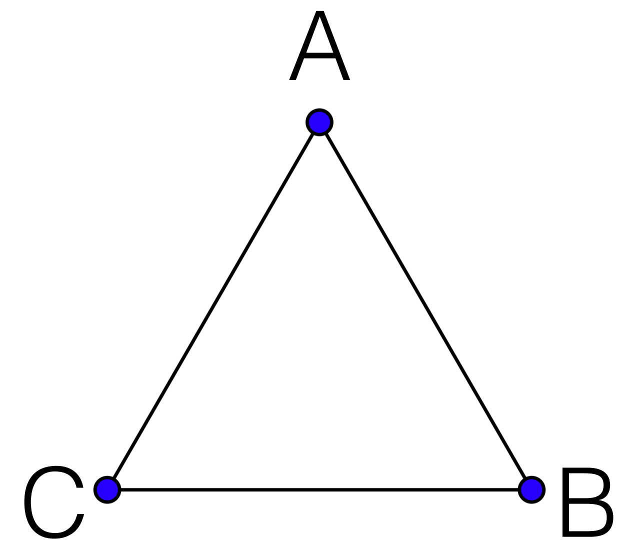
- Edge Constraints:
  Adjacent nodes cannot
  have the same color

- For edge (A,B)
  $$\neg(A_{red} \wedge B_{red}) \wedge \neg(A_{green} \wedge B_{green}) \wedge \neg(A_{blue} \wedge B_{blue})$$

- Converted to CNF
  $$(\neg A_{red} \vee \neg B_{red}) \wedge (\neg A_{green} \vee \neg B_{green}) \wedge (\neg A_{blue} \vee \neg B_{blue})$$

- For all edges:
  $$\neg(A_{red} \wedge B_{red}) \wedge \neg(A_{green} \wedge B_{green}) \wedge \neg(A_{blue} \wedge B_{blue})$$
  $$\wedge \neg(A_{red} \wedge C_{red}) \wedge \neg(A_{green} \wedge C_{green}) \wedge \neg(A_{blue} \wedge C_{blue})$$
  $$\wedge \neg(C_{red} \wedge B_{red}) \wedge \neg(C_{green} \wedge B_{green}) \wedge \neg(C_{blue} \wedge B_{blue})$$

3-Coloring to SAT



- Putting all together

- $(A_{red} \vee A_{blue} \vee A_{green}) \wedge (\neg A_{red} \vee \neg A_{blue}) \wedge (\neg A_{red} \vee \neg A_{green}) \wedge (\neg A_{green} \vee \neg A_{blue})$
  $\wedge (B_{red} \vee B_{blue} \vee B_{green}) \wedge (\neg B_{red} \vee \neg B_{blue}) \wedge (\neg B_{red} \vee \neg B_{green}) \wedge (\neg B_{green} \vee \neg B_{blue})$
  $\wedge (C_{red} \vee C_{blue} \vee C_{green}) \wedge (\neg C_{red} \vee \neg C_{blue}) \wedge (\neg C_{red} \vee \neg C_{green}) \wedge (\neg C_{green} \vee \neg C_{blue})$
  $\neg(A_{red} \wedge B_{red}) \wedge \neg(A_{green} \wedge B_{green}) \wedge \neg(A_{blue} \wedge B_{blue})$
  $\wedge \neg(A_{red} \wedge C_{red}) \wedge \neg(A_{green} \wedge C_{green}) \wedge \neg(A_{blue} \wedge C_{blue})$
  $\wedge \neg(C_{red} \wedge B_{red}) \wedge \neg(C_{green} \wedge B_{green}) \wedge \neg(C_{blue} \wedge B_{blue})$

Goal of the Assignment

- Write a program (python or java or c++) that solves 3-coloring using the MiniSat solver
  - Input: a graph to be 3-colored
  - Output: corresponding boolean formula (as explained in previous video) that can be fed into MiniSAT