1. **Intro.** This little program outputs clauses that are satisfiable if and only if the graph \( g \) can be \( c \)-colored, given \( g \) and \( c \). It differs from SAT-COLOR because it uses binary labels instead of unary labels to specify the colors.

(It generalizes SAT-PIGEONS-LOG, which is the case where \( g = K_m \) and \( c = n \).)

Suppose the graph has \( m \) edges and \( n \) vertices, and let \( t = \lceil \log c \rceil \). Then there are \( nt \) variables \( v.k \), meaning that vertex \( v \) gets color \((v.1 \ v.2 \ldots v.k)\). There also are \( mt \) auxiliary variables \( u^v.k \), meaning \( u.k \oplus v.k \) when \( u \longrightarrow v \).

There are \( m \) clauses of size \( t \) to ensure that adjacent vertices don’t share a color, plus \( 4mt \) clauses to define the auxiliary variables. And finally, there are \( O(nt) \) additional clauses of size \( t \), to rule out cases where a vertex is assigned to colors \( s \) in the range \( n \leq s < 2^t \).

```c
#include <stdio.h>
#include <stdlib.h>
#include "gb_graph.h"
#include "gb_save.h"

int c;

main(int argc, char *argv[]) {
    register int i, k, t;
    register Arc *a;
    register Graph *g;
    register Vertex *u, *v;
    // (Process the command line 2):
    for (t = 0; c > (1 << t); t++) {
        // (Generate negative clauses to rule out bad colors 3):
        for (v = g→vertices; v < g→vertices + g→n; v++)
            for (a = v→arcs; a; a = a→next) {
                u = a→tip;
                if (u < v) // (Generate clauses to keep u and v from having the same color 4):
                    ...
            }
    }
}
```

2. (Process the command line 2) \( \equiv \)

if \( \argc \neq 3 \lor sscanf(\argv[2], "%d", &c) \neq 1 \) {
    fprintf(stderr, "Usage: %s foo.gb, \n" header, \n"");
    exit(-1);
}

\( g = restore_graph(\argv[1]) \);
if \( (\neg g) \) {
    fprintf(stderr, "I couldn’t reconstruct graph!\n", argv[1]);
    exit(-2);
}
if \( (c \leq 0) \) {
    fprintf(stderr, "c must be positive!\n" header, \n"");
    exit(-3);
}
printf("-s sat-color-log %d\n", argv[1], c);

This code is used in section 1.
3. \{ Generate negative clauses to rule out bad colors \} =
   \textbf{for} (i = 0; i < t; i++)
   \textbf{if} (((c - 1) & (1 \ll i)) \equiv 0) \{
   \textbf{for} (v = g\text{-}vertices; v < g\text{-}vertices + g\text{-}n; v++) \{
      \text{printf}("\neg%s.^{}%d", v\text{-}name, t - i);
      \textbf{for} (k = i + 1; k < t; k++)
      \text{printf}("\neg%s.^{}%d, v\text{-}name, t - k);
      \text{printf}("\n");
   \}
   \}
This code is used in section 1.

4. \{ Generate clauses to keep \( u \) and \( v \) from having the same color \} =
   \textbf{for} (k = 1; k \leq t; k++) \{
   \text{printf}("\neg%s^{}%s^{}%d", u\text{-}name, v\text{-}name, k);
   \text{printf}("\neg%s^{}%s^{}%d", u\text{-}name, v\text{-}name, k);
   \text{printf}("\neg%s^{}%s^{}%d", u\text{-}name, v\text{-}name, k);
   \text{printf}("\neg%s^{}%s^{}%d", u\text{-}name, v\text{-}name, k);
   \}
This code is used in section 1.
5. Index.

a: 1.
Arc: 1.
arcs: 1.
argc: 1, 2.
argv: 1, 2.
c: 1.
exit: 2.
fprintf: 2.
g: 1.
Graph: 1.
i: 1.
k: 1.
main: 1.
name: 3, 4.
next: 1.
printf: 2, 3, 4.
restore_graph: 2.
sscanf: 2.
stderr: 2.
t: 1.
tip: 1.
u: 1.
v: 1.
Vertex: 1.
vertices: 1, 3.
Generate clauses to keep $u$ and $v$ from having the same color \( 4 \)  Used in section 1.

Generate negative clauses to rule out bad colors \( 3 \)  Used in section 1.

Process the command line \( 2 \)  Used in section 1.