1* Intro. This program generates clauses that enforce the constraint $x_1 + \cdots + x_n \leq r$, using a method due to Olivier Bailleux and Yacine Boufkhad [Lecture Notes in Computer Science 2833 (2003), 108–122]. It introduces at most $(n - 2)r$ new variables $B_i j$ for $2 \leq i < n$ and $1 \leq j \leq r$, and a number of clauses that I haven’t yet tried to count carefully, but it is at most $O(nr)$. All clauses have length 3 or less.

This version inputs a graph (specified as the third parameter), and the total number of colors (as the fourth). The output clauses will insist that at least $r$ vertices can be colored with more than one color.

```c
#define nmax 10000
#include <stdio.h>
#include <stdlib.h>
#include "gb_graph.h"
#include "gb_save.h"

int n, r, kk; /* the given parameters */
int count[nmax + nmax]; /* the number of leaves below each node */

int main(int argc, char *argv[]) {
  register int i, j, k, jl, jr, t, tl, tr;

  Graph *g;

  if (argc < 4) {
    printf("Usage: %s <graph> <colors>
", argv[0]);
    return 1;
  }

  printf("Loading %s graph...
", argv[1]);
  g = gb_graph_read(argv[1]);

  printf("There are %d nodes.
", gb_graph_node_count(g));

  printf("Building the complete binary tree...
"); // Process the command line 2*

  printf("Generating the clauses for node i 4*
"); // Generate the clauses for node i

  printf("Generating the clauses at the root 5*
"); // Generate the clauses at the root

  gb_graph_free(g);
}
```
INTRO SAT-THRESHOLD-BB-GRAPHS DOUBLE

2* (Process the command line 2*)

if (argc ≠ 5 ∨ sscanf(argv[1], "%d", &n) ≠ 1 ∨ sscanf(argv[2], "%d", &r) ≠ 1 ∨ sscanf(argv[4], "%d", &kk) ≠ 1)
{
  fprintf(stderr, "Usage: %s n r foo.gb k
", argv[0]);
  exit(-1);
}

2* g = restore_graph(argv[3]);

if (~g) {
  fprintf(stderr, "I can't input the graph '%s'!
", argv[3]);
  exit(-2);
}

if (g ≠ n) fprintf(stderr, "Warning: The graph has %ld vertices, not %d!
", g, n);

r = n - r;
/* x1 + ⋯ + xn ≥ r iff x1 + ⋯ + xn ≤ n - r */
if (n > nmax) {
  fprintf(stderr, "Recompile me: I don't allow n>%d
", nmax);
  exit(-2);
}

if (r < 0 ∨ r > n) {
  fprintf(stderr, "Eh? x should be between 0 and n-1!
");
  exit(-2);
}

printf("~ sat-threshold-bb %d %d
", n, r);

This code is used in section 1*.

3. The tree has 2n - 1 nodes, with 0 as the root; the leaves start at node n - 1. Nonleaf node k has left child 2k + 1 and right child 2k + 2. Here we simply fill the count array.

(Build the complete binary tree with n leaves 3) ≡

for (k = n + n - 2; k ≥ n - 1; k--)
  count[k] = 1;
for (k = n; k ≥ 0; k--)
  count[k] = count[k + k + 1] + count[k + k + 2];
if (count[0] ≠ n) fprintf(stderr, "I'm totally confused.
");

This code is used in section 1*.
§4  SAT-THRESHOLD-BB-GRAPHS-DOUBLE

4* If there are \( t \) leaves below node \( i \), we introduce \( k = \min(r, t) \) variables \( B_{i+1,j} \) for \( 1 \leq j \leq k \). This variable is 1 if (but not only if) at least \( j \) of those leaf variables are true. If \( t > r \), we also assert that no \( r + 1 \) of those variables are true.

\[
\text{\#define xbar}(k) \quad \text{printf}("%a\cdot x^n, (g-vertices + (k) - n + 1)-name")
\]

(Generate the clauses for node \( i \)’s) \( \equiv \)
\[
\{ \\
\quad t = \text{count}[i], tl = \text{count}[i + i + 1], tr = \text{count}[i + i + 2]; \\
\quad \text{if} (t > r + 1) \ t = r + 1; \\
\quad \text{if} (t > r) \ tl = r; \\
\quad \text{if} (tr > r) \ tr = r; \\
\quad \text{for} (jl = 0; jl \leq tl; jl++) \\
\quad \quad \text{for} (jr = 0; jr \leq tr; jr++) \\
\quad \quad \quad \text{if} ((jl + jr \leq t) \land (jl + jr) > 0) \{ \\
\quad \quad \quad \quad \text{if} (jl) \{ \\
\quad \quad \quad \quad \quad \text{if} (i + i + 1 \geq n - 1) \ xbar(i + i + 1); \\
\quad \quad \quad \quad \text{else} \ \text{printf}("\%B%d.\%d", i + i + 2, jl); \\
\quad \quad \quad \} \\
\quad \quad \quad \text{if} (jr) \{ \\
\quad \quad \quad \quad \text{printf}"\%d"; \\
\quad \quad \quad \quad \text{if} (i + i + 2 \geq n - 1) \ xbar(i + i + 2); \\
\quad \quad \quad \quad \text{else} \ \text{printf}("\%B%d.\%d", i + i + 3, jr); \\
\quad \quad \quad \} \\
\quad \quad \quad \text{if} (jl + jr \leq r) \ \text{printf}("\%B%d\cdot n", i + 1, jl + jr); \\
\quad \quad \quad \text{else} \ \text{printf}("\%n"); \\
\quad \quad \} \\
\} \\
\}
\]

This code is used in section 1*.

5. Finally, we assert that at most \( r \) of the \( x \)'s are true, by implicitly asserting that the (nonexistent) variable \( B_1.r+1 \) is false.

(Generate the clauses at the root \( 5 \)) \( \equiv \)
\[
\{ \\
\quad tl = \text{count}[1], tr = \text{count}[2]; \\
\quad \text{if} (tl > r) \ tl = r; \\
\quad \text{for} (jl = 1; jl \leq tl; jl++) \{ \\
\quad \quad jl = r + 1 - jl; \\
\quad \text{if} (jl \leq tl) \{ \\
\quad \quad \text{if} (1 \geq n - 1) \ xbar(1); \\
\quad \quad \text{else} \ \text{printf}("\%B2.\%d", jl); \\
\quad \quad \text{printf}"\%n"; \\
\quad \quad \} \\
\quad \text{if} (2 \geq n - 1) \ xbar(2); \\
\quad \text{else} \ \text{printf}("\%B3.\%d", jr); \\
\quad \text{printf}"\%n"; \\
\quad \} \\
\}
\]

This code is used in section 1*.
6. \langle \text{Handle the trivial case directly} \rangle \equiv
\{ 
  \text{for} \ (i = 1; \ i \leq n; \ i++) \ \{ 
    \text{zbar}(n - 2 + i); 
    \text{printf}("\n"); 
  \} 
\}

This code is used in section 1*.

7* \langle \text{Output clauses for multicolored vertices} \rangle \equiv 
\{ \text{for} \ (k = 0; \ k < g\cdot n; \ k++) \ \{ 
  \text{for} \ (i = 1; \ i \leq kk; \ i++) \ \{ 
    \text{for} \ (j = 1; \ j \leq kk; \ j++) 
      \text{if} \ (j \neq i) \ \text{printf}("\%s.\%d", (g\cdot \text{vertices} + k)\cdot \text{name}, j); 
    \text{printf}("\~%s.x\n", (g\cdot \text{vertices} + k)\cdot \text{name}); 
  \} 
\} 
\}

This code is used in section 1*.
§8  SAT-THRESHOLD-BB-GRAPHS-DOUBLE

**Index.**
The following sections were changed by the change file: 1, 2, 4, 7, 8.

*argc: 1, 2*
*argv: 1, 2*
*count: 1, 3, 4, 5.*
*exit: 2*
*fprintf: 2, 3.*
*Graph: 1*
*i: 1*
*j: 1*
*jl: 1, 4, 5.*
*jr: 1, 4, 5.*
*k: 1*
*kk: 1, 2, 7*
*main: 1*
*n: 1*
*name: 4, 7*
*nmax: 1, 2*
*printf: 2, 4, 5, 6, 7*
*r: 1*
*restore_graph: 2*
*sscanf: 2*
*stderr: 2, 3.*
*t: 1*
*tl: 1, 4, 5.*
*tr: 1, 4, 5.*
*vertices: 4, 7*
*xbar: 4, 5, 6.*
(Build the complete binary tree with $n$ leaves 3) Used in section 1*.
(Generate the clauses at the root 5) Used in section 1*.
(Generate the clauses for node $i$ 4*) Used in section 1*.
(Handle the trivial case directly 6) Used in section 1*.
(Output clauses for multicolored vertices 7*) Used in section 1*.
(Process the command line 2*) Used in section 1*.