In[1]. 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 $B_{ij}$ 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 a third parameter) and a color number (specified fourth). The output clauses will limit the number of vertices of that color.

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

int n, r, kk;
int count[nmax+nmax];

int main(int argc, char *argv[])
{
    register int i, j, k, jl, jr, t, tl, tr;
    Graph *g;
    (Process the command line 2*);
    if (r <= 0) (Handle the trivial case directly 6)
    else {
        (Build the complete binary tree with $n$ leaves 3);
        for (i = n-2; i--; i--) (Generate the clauses for node $i$ 4*);
        (Generate the clauses at the root 5);
    }
}
```

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 \(\text{count}\) array.

(Build the complete binary tree with \(n\) leaves 3) \(\equiv\)

\[
\text{for } (k = n + n - 2; k \geq n - 1; k--) \text{ count}\[k\] = 1;
\text{for } ( ; k \geq 0; k--) \text{ count}\[k\] = \text{count}\[k + k + 1\] + \text{count}\[k + k + 2\];
\]

\(\text{if } (\text{count}\[0\] \neq n) \text{ printf}(\text{stderr}, \text{"I'm totally confused.\n"});\)

This code is used in section 1*.

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.

\#define xbar \(k\) \(\text{printf}(\text{"~%s.%d", (g-vertices + (k) - n + 1)-name, kk)\}

(Generate the clauses for node \(i\) 4*) \(\equiv\)

\[
\text{if } (t > r + 1) t = r + 1;
\text{if } (t > r) tl = r;
\text{if } (tl > r) tr = r;
\text{for } (j_1 = 0; j_1 \leq tl; j_1++)
\text{for } (j_2 = 0; j_2 \leq tr; j_2++)
\text{if } ((j_1 + j_2 \leq t) \land (j_1 + j_2) > 0) \{\}
\text{if } (j_1) \{\}
\text{if } (i + i + 1 \geq n - 1) \text{xbar}(i + i + 1);
\text{else } \text{printf}(\text{"B%d.%d", i + i + 2, j_1});\}
\text{if } (j_2) \{\text{printf}(\text{"\n"});\}
\text{if } (i + i + 2 \geq n - 1) \text{xbar}(i + i + 2);
\text{else } \text{printf}(\text{"B%d.%d", i + i + 3, j_2});\}
\text{if } (j_1 + j_2 \leq r) \text{printf}(\text{"B%d.%d\n", i + 1, j_1 + j_2});
\text{else } \text{printf}(\text{"\n"});\}\}
\]

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 $B_{1 \cdot r + 1}$ is false.

\[(\text{Generate the clauses at the root 5}) \equiv \]
\[tl = \text{count}[1], \, tr = \text{count}[2]; \]
\[\text{if } (tl > r) \, tl = r; \]
\[\text{for } (jl = 1; \, jl \leq tl; \, jl++) \{ \]
\[jr = r + 1 - jl; \]
\[\text{if } (jr \leq tr) \{ \]
\[\text{if } (1 \geq n - 1) \, xbar(1); \]
\[\text{else } \text{printf}("\neg B_2.\%d", \, jl); \]
\[\text{printf}("\&"); \]
\[\text{if } (2 \geq n - 1) \, xbar(2); \]
\[\text{else } \text{printf}("\neg B_3.\%d", \, jr); \]
\[\text{printf}(\"\n\"); \]
\[\}\]
\[\}\]
This code is used in section 1*.

6. \(\text{\langle Handle the trivial case directly 6\rangle } \equiv \)
\[\{ \]
\[\text{for } (i = 1; \, i \leq n; \, i++) \{ \]
\[xbar(n - 2 + i); \]
\[\text{printf}(\"\n\"); \]
\[\}\]
\[\}\]
This code is used in section 1*.
7* Index.
The following sections were changed by the change file: 1, 2, 4, 7.

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* 4*
main: 1*
n: 1*
name: 4*
nmax: 1* 2*
printf: 2* 4* 5, 6.
r: 1*
restore_graph: 2*
sscanf: 2*
stderr: 2* 3.
t: 1*
tl: 1* 4* 5.
tr: 1* 4* 5.
vertices: 4*
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*.
(Process the command line 2*) Used in section 1*.