1*. Intro. Kazimierz Zarankiewicz asked [*Colloquium Mathematicum* **2** (1951), 301] for the smallest N such that every $n \times n$ matrix of zeros and ones contains a 2×2 submatrix of ones. R. K. Guy [in *Theory of Graphs* (Academic Press, 1968), 119–150] considered generalizations of the problem to nonsqaure matrices and submatrices, and tabulated results for small cases. Here I simply generate clauses that are satisfiable if and only if there's an $m \times n$ matrix containing at least r 1s but no such 2×2 submatrix.

This problem is interesting because of its many symmetries: $m!$ ways to permute the rows, times $n!$ ways to permute the columns. (If $m = n$, we can also transpose the matrix.)

I remove many of the symmetries, by requiring that the rows are in lexicographic order (when restricted to the first p columns) and the columns are in lexicographic order (when read top-down and restricted to the first q rows).

Setting $p = n$ and $q = m$ gives the maximum constraints, but smaller values may provide satisfactory symmetry breaking with less total cost.

In this version I require the solution to be equal to its transpose.

```
#define nmax 1000   /* upper bound on m × n */
#include <stdio.h>
#include <stdlib.h>
  int m, n, r, p, q;   /* command-line parameters */
  int count[2 * nmax];   /* used for the cardinality constraints */
  main(int argc, char *argv[])
  {
    register int i, j, ii, jj, k, mn, t, tl, tr, jl, jr;
    ⟨Process the command line 2*⟩;
    ⟨Generate the clauses for the lexicographic row constraints 4⟩;
    ⟨Generate the clauses for the lexicographic column constraints 5⟩;
    ⟨Generate the clauses for the rectangle constraints 3⟩;
    ⟨Generate the clauses for symmetry under reflection 10*⟩;
    ⟨Generate the clauses for the cardinality constraints 6⟩;
  }
```

2* ⟨ Process the command line 2* ⟩ ≡

```
if (argc ≠ 6 ∨ sscanf(argv[1], "%d", &m) ≠ 1 ∨ sscanf(argv[2], "%d", &n) ≠ 1 ∨ sscanf(argv[3], "%d",
      &r) ≠ 1 ∨ sscanf(argv[4], "%d", &p) ≠ 1 ∨ sscanf(argv[5], "%d", &q) ≠ 1) {
  fprintf(stderr, "Usage: %s m n r p q\n", argv[0]);
  exit(−1);
}
mn = m ∗ n;
if (mn > nmax) {
  fprintf(stderr, "Sorry: mn is %d, and I'm set up for at most %d!\n", mn, nmax);
  exit(−2);
}
if (p > n) {
  fprintf(stderr, "Parameter p should be at most n (%d), not %d!\n", n, p);
  exit(−3);
}
if (q > m) {
  fprintf(stderr, "Parameter q should be at most m (%d), not %d!\n", m, q);
  exit(−4);
}
if (m ≠ n) {
  fprintf(stderr, "In this version m must equal n!\n");
  exit(−5);
}
printf("~ sat-zarank-symm %d %d %d %d %d\n", m, n, r, p, q);
```

This code is used in section 1*.

3. ⟨ Generate the clauses for the rectangle constraints 3 ⟩ ≡

```
for (i = 0; i < m; i++)
  for (ii = i + 1; ii < m; ii++)
    for (j = 0; j < n; j++)
      for (jj = j + 1; jj < n; jj++) {
        printf("~%d.%d ~%d.%d ~%d.%d ~%d.%d\n", i, j, ii, j, i, jj, ii, jj);
      }
```

This code is used in section 1*.

4. (See SAT-LEXORDER.) I choose *decreasing* order, because (a) fewer binary matrices with a given number of 1s (assumed less than mn/2) are doubly ordered when we do it this way; and (b) the connected components of the underlying bipartite graph are nicely revealed, as proved by Mader and Mutzbauer in 2001.

⟨ Generate the clauses for the lexicographic row constraints 4 ⟩ ≡

```
for (i = 1; i < m; i++) {
  for (k = 1; k ≤ p; k++) {
    if (k ≠ p) {
      if (k ≠ 1) printf("~R%d.%d", i, k − 1);
      printf(" R%d.%d %d.%d\n", i, k, i − 1, k − 1);
      if (k ≠ 1) printf("~R%d.%d", i, k − 1);
      printf(" R%d.%d ~%d.%d\n", i, k, i, k − 1);
    }
    if (k ≠ 1) printf("~R%d.%d", i, k − 1);
    printf(" %d.%d ~%d.%d\n", i − 1, k − 1, i, k − 1);
  }
}
```

This code is used in section 1*.

5. ⟨ Generate the clauses for the lexicographic column constraints 5 ⟩ ≡

```
for (i = 1; i < n; i++) {
  for (k = 1; k ≤ q; k++) {
    if (k ≠ q) {
      if (k ≠ 1) printf("~C%d.%d", k - 1, i);
      printf("␣C%d.%d␣%d.%d\n", k, i, k - 1, i - 1);
      if (k ≠ 1) printf("~C%d.%d", k - 1, i);
      printf("␣C%d.%d␣~%d.%d\n", k, i, k - 1, i);
    }
    if (k ≠ 1) printf("~C%d.%d", k - 1, i);
    printf("␣%d.%d␣~%d.%d\n", k - 1, i - 1, k - 1, i);
  }
}
```

This code is used in section 1*.

6. Finally come the clauses that require at least r 1s in the matrix. As usual, I copy stuff from SAT-THRESHOLD-BB.

⟨ Generate the clauses for the cardinality constraints 6 ⟩ ≡

```
⟨ Build the complete binary tree with mn leaves 7 ⟩;
r = mn - r;    /* convert to asking for at most mn - r zeroes */
for (i = mn - 2; i; i--) ⟨ Generate the clauses for node i 8 ⟩;
⟨ Generate the clauses at the root 9 ⟩;
```

This code is used in section 1*.

7. The tree has $2mn-1$ nodes, with 0 as the root; the leaves start at node $mn-1$. 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 mn leaves 7 ⟩ ≡

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

This code is used in section 6.

8. If there are t leaves below node i, we introduce $k = \min(r, t)$ variables `B`i+`1.`j for $1 \le j \le k$. 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 $x(k)$ $\mathit{printf}(\texttt{"\%d.\%d"}, ((k) - mn + 1)/n, ((k) - mn + 1) \mathbin{\%} n)$

⟨ Generate the clauses for node i 8 ⟩ ≡
  {
    $t = \mathit{count}[i], \mathit{tl} = \mathit{count}[i + i + 1], \mathit{tr} = \mathit{count}[i + i + 2]$;
    if $(t > r + 1)$ $t = r + 1$;
    if $(\mathit{tl} > r)$ $\mathit{tl} = r$;
    if $(\mathit{tr} > r)$ $\mathit{tr} = r$;
    for $(\mathit{jl} = 0;\ \mathit{jl} \le \mathit{tl};\ \mathit{jl}{+}{+})$
      for $(\mathit{jr} = 0;\ \mathit{jr} \le \mathit{tr};\ \mathit{jr}{+}{+})$
        if $((\mathit{jl} + \mathit{jr} \le t) \land (\mathit{jl} + \mathit{jr}) > 0)$ {
          if (jl) {
            if $(i + i + 1 \ge mn - 1)$ $x(i + i + 1)$;
            else $\mathit{printf}(\texttt{"\~{}B\%d.\%d"}, i + i + 2, \mathit{jl})$;
          }
          if (jr) {
            $\mathit{printf}(\texttt{"␣"})$;
            if $(i + i + 2 \ge mn - 1)$ $x(i + i + 2)$;
            else $\mathit{printf}(\texttt{"\~{}B\%d.\%d"}, i + i + 3, \mathit{jr})$;
          }
          if $(\mathit{jl} + \mathit{jr} \le r)$ $\mathit{printf}(\texttt{"␣B\%d.\%d\textbackslash n"}, i + 1, \mathit{jl} + \mathit{jr})$;
          else $\mathit{printf}(\texttt{"\textbackslash n"})$;
        }
  }

This code is used in section 6.

9. Finally, we assert that at most r of the x's aren't true, by implicitly asserting that the (nonexistent) variable `B1.`r+1 is false.

⟨ Generate the clauses at the root 9 ⟩ ≡
  $\mathit{tl} = \mathit{count}[1], \mathit{tr} = \mathit{count}[2]$;
  if $(\mathit{tl} > r)$ $\mathit{tl} = r$;
  for $(\mathit{jl} = 1;\ \mathit{jl} \le \mathit{tl};\ \mathit{jl}{+}{+})$ {
    $\mathit{jr} = r + 1 - \mathit{jl}$;
    if $(\mathit{jr} \le \mathit{tr})$ {
      if $(1 \ge mn - 1)$ $x(1)$;
      else $\mathit{printf}(\texttt{"\~{}B2.\%d"}, \mathit{jl})$;
      $\mathit{printf}(\texttt{"␣"})$;
      if $(2 \ge mn - 1)$ $x(2)$;
      else $\mathit{printf}(\texttt{"\~{}B3.\%d"}, \mathit{jr})$;
      $\mathit{printf}(\texttt{"\textbackslash n"})$;
    }
  }

This code is used in section 6.

10* ⟨ Generate the clauses for symmetry under reflection 10* ⟩ ≡
  for $(i = 0;\ i < m;\ i{+}{+})$
    for $(j = 0;\ j < n;\ j{+}{+})$
      if $(i \ne j)$ $\mathit{printf}(\texttt{"\%d.\%d␣\~{}\%d.\%d\textbackslash n"}, i, j, j, i)$;

This code is used in section 1*.

11.* Index.

The following sections were changed by the change file: 1, 2, 10, 11.

SAT-ZARANK-SYMM