§2 DLX2-CUTOFF INTRO 1

(Downloaded from https://cs.stanford.edu/~knuth/programs.html and typeset on September 17, 2017)

2.\* This version of the program keeps removing rows at the bottom, thereby finding all solutions that have minimax row number. (And it usually also finds a few more, before it has found the best cutoff point.)

We assume that the columns contain rows in their original order; the *up* and *down* links actually point upwards and downwards. Therefore we disable the "randomizing" feature by which columns could be linked randomly. (Randomization still does apply, however, when choosing a column for branching.)

After this program finds all the desired solutions, it normally prints their total number on *stderr*, together with statistics about how many nodes were in the search tree, and how many "updates" and "cleansings" were made. The running time in "mems" is also reported, together with the approximate number of bytes needed for data storage. (An "update" is the removal of a row from its column. A "cleansing" is the removal of a satisfied color constraint from its row. One "mem" essentially means a memory access to a 64-bit word. The reported totals don't include the time or space needed to parse the input or to format the output.)

```
Here is the overall structure:
```

```
#define o mems++ /* count one mem */
#define oo mems+=2 /* count two mems */
#define ooo mems+=3 /* count three mems */
                      /* used for percent signs in format strings */
#define O "%"
#define mod %
                      /* used for percent signs denoting remainder in C */
                                /* at most this many rows in a solution */
#define max_level 500
#define max\_cols 100000 /* at most this many columns */
#define max_nodes 10000000 /* at most this many nonzero elements in the matrix */
#define bufsize (9*max\_cols + 3) /* a buffer big enough to hold all column names */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "gb_flip.h"
  typedef unsigned int uint; /* a convenient abbreviation */
  typedef unsigned long long ullng; /* ditto */
  \langle \text{Type definitions 5} \rangle;
   \langle \text{Global variables } 3^* \rangle;
  \langle \text{Subroutines 9} \rangle;
  main(\mathbf{int} \ argc, \mathbf{char} * argv[])
     register int cc, i, j, k, p, pp, q, r, t, cur\_node, best\_col;
     \langle \text{Process the command line 4} \rangle;
     \langle \text{Input the column names } 13 \rangle;
     \langle \text{Input the rows 16} \rangle;
     if (vbose & show_basics) (Report the successful completion of the input phase 20);
     if (vbose \& show\_tots) (Report the column totals 21);
     imems = mems, mems = 0;
     \langle Solve the problem 22\rangle;
  done: if (sanity_checking) sanity();
     if (vbose \& show\_tots) \land Report the column totals 21 >;
     if (vbose & show_profile) \langle Print the profile 35 \rangle;
     if (vbose & show_basics) {
       fprintf(stderr, "Altogether_"O"llu_solution"O"s,_"O"llu+"O"llu-"O"llu-"O"llu_mems,", count,
            count \equiv 1? "": "s", imems, mems, lmems);
       bytes = last\_col * sizeof(column) + last\_node * sizeof(node) + maxl * sizeof(int);
       fprintf(stderr, "u"O"llu_updates, u"O"llu_cleansings, ", updates, cleansings);
```

2 INTRO DLX2-CUTOFF §2

```
fprintf (stderr, \verb"\|| O \verb"|| llu \verb||| bytes, \verb||| O \verb"|| llu \verb||| nodes. \verb||n"|, bytes, nodes); \\ \}
```

**3.\*** You can control the amount of output, as well as certain properties of the algorithm, by specifying options on the command line:

- 'v(integer)' enables or disables various kinds of verbose output on stderr, given by binary codes such as show\_choices;
- 'm (integer)' causes every mth solution to be output (the default is m0, which merely counts them);
- 's(integer)' causes the algorithm to make random choices in key places (thus providing some variety, although the solutions are by no means uniformly random), and it also defines the seed for any random numbers that are used:
- 'd(integer)' to sets *delta*, which causes periodic state reports on *stderr* after the algorithm has performed approximately *delta* mems since the previous report;
- 'copositive integer' limits the levels on which choices are shown during verbose tracing;
- 'C' positive integer' limits the levels on which choices are shown in the periodic state reports;
- '1 (nonnegative integer)' gives a *lower* limit, relative to the maximum level so far achieved, to the levels on which choices are shown during verbose tracing;
- 't' positive integer' causes the program to stop after this many solutions have been found;
- 'T(integer)' sets timeout (which causes abrupt termination if mems > timeout at the beginning of a level).

```
#define show_basics 1
                            /* vbose code for basic stats; this is the default */
#define show_choices 2
                             /* vbose code for backtrack logging */
                            /* vbose code for further commentary */
#define show_details 4
                            /* vbose code to report improvements in the cutoff point */
#define show_cutoffs 8
#define show_profile 128
                              /* vbose code to show the search tree profile */
#define show_full_state 256
                                 /* vbose code for complete state reports */
#define show_tots 512
                            /* vbose code for reporting column totals at start and end */
                                  /* vbose code for reporting rows without primaries */
#define show_warnings 1024
\langle Global variables 3*\rangle \equiv
                          /* seed for the random words of gb\_rand */
  int random\_seed = 0;
                     /* has 's' been specified? */
  int randomizing;
  int vbose = show\_basics + show\_warnings; /* level of verbosity */
                 /* solution k is output if k is a multiple of spacing */
  int spacing;
                                     /* above this level, show_choices is ignored */
  int show\_choices\_max = 1000000;
                                     /* below level maxl - show_choices_gap, show_details is ignored */
  int show\_choices\_gap = 1000000;
                                    /* above this level, state reports stop */
  int show\_levels\_max = 1000000;
  int maxl = 0;
                   /* maximum level actually reached */
  char buf [bufsize];
                       /* input buffer */
                  /* solutions found so far */
  ullng count;
  ullng rows:
                 /* rows seen so far */
  ullng imems, mems, lmems;
                                  /* mem counts */
  ullng updates;
                    /* update counts */
                      /* cleansing counts */
  ullng cleansings;
  ullng bytes;
                 /* memory used by main data structures */
                 /* total number of branch nodes initiated */
  ullng nodes;
  ullng thresh = 0; /* report when mems exceeds this, if delta \neq 0 */
  ullng delta = 0;
                     /* report every delta or so mems */
                                           /* stop after finding this many solutions */
  /* give up after this many mems */
  See also sections 7^* and 23.
This code is used in section 2*.
```

§7 DLX2-CUTOFF DATA STRUCTURES 3

```
7.* \langle \text{Global variables } 3^* \rangle + \equiv
  node nd[max_nodes]; /* the master list of nodes */
  int last_node; /* the first node in nd that's not yet used */
  column cl[max\_cols + 2]; /* the master list of columns */
  int second = max\_cols;
                               /* boundary between primary and secondary columns */
                   /* the first column in cl that's not yet used */
  int last_col;
  int cutoff = max\_nodes;
                                /* nodes after this point have essentially disappeared */
11.* Speaking of debugging, here's a routine to check if redundant parts of our data structure have gone
#define sanity_checking 0
                                   /* set this to 1 if you suspect a bug */
\langle \text{Subroutines } 9 \rangle + \equiv
  void sanity(void)
    register int k, p, q, pp, qq, t;
    for (q = root, p = cl[q].next; ; q = p, p = cl[p].next) {
      if (cl[p].prev \neq q) fprintf(stderr, "Bad_prev_field_at_col_"O".8s!\n", cl[p].name);
      if (p \equiv root) break;
       \langle \text{Check column } p \mid 12 \rangle;
  }
```

4 The dancing DLX2-cutoff  $\S26$ 

**26\*** A subtle point should be noted: As we uncover column c, and run across a row ' $c \times ...$ ' that should be restored to column x, the original successors ' $x \times a \times ...$ ', ' $x \times b \times ...$ ', etc., of that row in column x may now be cut off. In such a case we can be sure that those successor rows have disappeared from column x, and they have not been restored.

The reason is that each of those rows must have a primary column; and every primary column was covered before we changed the cutoff. The rows were therefore not restored to column x when we uncovered those primary columns.

```
\langle \text{Subroutines } 9 \rangle + \equiv
  void uncover(int c)
    register int cc, l, r, rr, nn, uu, dd, t;
    {\bf for}\ (o,t=0,rr=nd[c].up;\ rr\geq cutoff;\ o,rr=nd[rr].up)\ t+\!\!+;
                /* t rows that we covered have been cut off */
    if (t) {
       oo, nd[c].len = t;
       if (c \geq second) lmems += 2;
       oo, nd[c].up = rr, nd[rr].down = c;
    for (; rr \geq last\_col; o, rr = nd[rr].up)
       for (nn = rr + 1; nn \neq rr;) {
         if (o, nd[nn].color \ge 0) {
            o, uu = nd[nn].up, dd = nd[nn].down;
            cc = nd[nn].col;
           if (cc \leq 0) {
              nn = uu;
              continue;
            if (dd \ge cutoff) o, nd[nn].down = dd = cc;
                                                               /* see the "subtle point" above */
            oo, nd[uu].down = nd[dd].up = nn;
            o, t = nd[cc].len + 1;
            o, nd[cc].len = t;
           if (cc \geq second) lmems += 2;
         nn++;
    o, l = cl[c].prev, r = cl[c].next;
     oo, cl[l].next = cl[r].prev = c;
```

§30 DLX2-CUTOFF THE DANCING

```
30* Just as purify is analogous to cover, the inverse process is analogous to uncover.
\langle Subroutines 9\rangle + \equiv
  void unpurify(int p)
    register int cc, rr, nn, uu, dd, t, x;
    o, cc = nd[p].col, x = nd[p].color; /* there's no need to clear nd[cc].color */
    for (o, t = 0, rr = nd[cc].up; rr \ge cutoff; o, rr = nd[rr].up) t++;
               /* t rows that we covered have been cut off */
      oo, nd[cc].len = t;
      lmems += 2;
      oo, nd[cc].up = rr, nd[rr].down = cc;
    for (; rr \geq last\_col; o, rr = nd[rr].up) {
      if (o, nd[rr].color < 0) o, nd[rr].color = x;
      else if (rr \neq p) {
         for (nn = rr - 1; nn \neq rr;) {
           o, uu = nd[nn].up, dd = nd[nn].down;
           o, cc = nd[nn].col;
           if (cc \leq 0) {
             nn = dd; continue;
           if (nd[nn].color \ge 0) {
             if (dd \ge cutoff) o, nd[nn].down = dd = cc;
                                                              /* see the "subtle point" above */
             oo, nd[uu].down = nd[dd].up = nn;
             o, t = nd[cc].len + 1;
             o, nd[cc].len = t;
             if (cc \geq second) lmems += 2;
} }
           nn --;
```

6 The dancing Dlx2-cutoff  $\S32$ 

```
32* \langle Record solution and goto recover 32* \rangle \equiv
  {
    count ++;
    for (k = 0, pp = 0; k \le level; k++)
      if (choice[k] > pp) pp = choice[k];
    for (pp ++; o, nd[pp].col > 0; pp ++);
                                                 /* move to end of largest chosen row */
    if (pp \neq cutoff) {
       cutoff = pp;
      if (vbose & show_cutoffs) {
         fprintf(stderr, \verb""-new" cutoff" after \verb"-row" O"d: \verb"-n" , -nd[pp].col");
         prow(nd[pp].up);
      for (k = 0; k \le level; k++) {
         o, cc = nd[choice[k]].col;
                                     /* cc will stay covered until we backtrack */
         for (o, t = 0, pp = nd[cc].up; pp \ge cutoff; o, pp = nd[pp].up) t++;
                    /* need to prune unneeded options from column cc */
         if (t) {
            oo, nd[pp].down = cc, nd[cc].up = pp;
            oo, nd[cc].len = t;
      }
    if (spacing \land (count \bmod spacing \equiv 0)) {
       printf(""O"lld: \n", count);
      for (k = 0; k \le level; k++) print_row(choice[k], stdout);
      fflush(stdout);
    if (count \ge maxcount) goto done;
    goto recover;
```

This code is used in section 22.

§36 DLX2-CUTOFF INDEX 7

## 36\* Index.

The following sections were changed by the change file: 2, 3, 7, 11, 26, 30, 32, 36.

```
advance: 22.
                                                                       len: \underline{5}, 9, 10, 12, 15, 17, 18, 19, 21, 25, 26*
argc: \underline{2}^*, \underline{4}.
                                                                             29, 30, 31, 32, 34.
argv: 2^*, 4.
                                                                       level: 22, 23, 31, 32, 33, 34, 35.
aux: \ \underline{5}, \ 17, \ 18, \ 19.
                                                                       lmems: 2,* 3,* 25, 26,* 29, 30.*
backup: \underline{22}.
                                                                       main: \underline{2}^*
best_col: \underline{2}^*, \underline{22}, \underline{31}.
                                                                       max\_cols: \ \ \underline{2}, \ 7, \ 13, \ 15.
buf: 3^*, 13, 16.
                                                                       max\_level: 2, 22, 23.
bufsize: 2,* 3,* 13, 16.
                                                                       max_nodes: 2* 7* 13, 16, 17, 31.
bytes: 2,* 3.*
                                                                       maxcount: 3* 4, 32*
                                                                       maxl: 2, 3, 22, 31, 33, 35.
c: 10, 25, 26* 34.
cc: 2*, 25, 26*, 27, 28, 29, 30*, 32*
                                                                       mems: 2, 3, 18, 24, 31, 33, 34.
choice: 22, <u>23</u>, 32, 33, 34.
                                                                       mod: \ \ \underline{2}^*, \ 32^*
cl: 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
                                                                       name: 6, 8, 9, 10, 11*12, 13, 14, 16, 17, 31.
     22, 25, 26, 31.
                                                                       nd: 5, \frac{7}{5}, 9, 10, 12, 13, 15, 16, 17, 18, 19, 21, 22,
cleansings: 2, 3, 29.
                                                                            25, 26, 27, 28, 29, 30, 31, 32, 34.
col: 5, 9, 12, 13, 16, 17, 19, 22, 25, 26, 27,
                                                                       next: \underline{6}, 10, 11*, 13, 15, 22, 25, 26*, 31.
                                                                       nn: \ \underline{25}, \ \underline{26}, \ \underline{29}, \ \underline{30}.
     28, 29, 30, 32, 34.
                                                                       node: 2, 5, 7*
col_struct: 6.
color: <u>5,</u> 9, 16, 25, 26, 27, 28, 29, 30.
                                                                       node\_struct: 5.
column: 2, 6, 7, 8.
                                                                       nodes: 2, 3, 22.
count: 2,* 3,* 32,* 33, 34.
                                                                       O: 2*
                                                                       o: 2*
cover: 22, 25, 27, 30*
cur\_node: \ \ \underline{2}, \ 22, \ 27, \ 28.
                                                                       oo: 2*, 15, 19, 22, 25, 26*, 29, 30*, 32*
                                                                       ooo: \underline{2}^*, 18.
cutoff: 7* 26* 30* 32*
                                                                       p: \quad \underline{2}, \underline{9}, \ \underline{10}, \ \underline{11}, \underline{29}, \ \underline{30}, \underline{34}.
d: \underline{34}.
dd: <u>25, 26</u>*, <u>29, 30</u>*.
                                                                       panic: 13, 14, 15, 16, 17.
delta: 3^*, 4, 24.
                                                                       pp: 2* 11* 12, 16, 17, 27, 28, 32*
done: 2^*, 22, 24, 32*
                                                                       prev: 6, 10, 11, 13, 15, 25, 26.
down: 2, 5, 9, 10, 12, 15, 16, 18, 19, 22, 25,
                                                                       print\_col: \underline{10}.
     26* 28, 29, 30* 32* 34.
                                                                       print_progress: 24, 34.
                                                                       print_row: 9, 22, 29, 32* 33.
exit: 4, 13, 22.
f: \underline{34}.
                                                                       print\_state: 24, 33.
                                                                       printf: 32*
fd: \underline{34}.
fflush: 32*
                                                                       profile: 22, 23, 35.
fgets: 13, 16.
                                                                       prow: 9, 10, 32*
forward: \underline{22}.
                                                                       purify: 27, 29, 30*
fprintf: 2* 4, 9, 10, 11* 12, 13, 16, 20, 21, 22,
                                                                       q: 2* 9, 11*
     24, 31, 32* 33, 34, 35.
                                                                       qq: 11^*, 12.
                                                                       r: \ \underline{2}^*, \ \underline{25}, \ \underline{26}^*
gb\_init\_rand: 4.
gb\_rand: 3*
                                                                       random\_seed: 3^*, 4.
gb\_unif\_rand: 18, 31.
                                                                       randomizing: 3^*, 4, 5, 18, 31.
i: \underline{2}^*
                                                                       recover: <u>22</u>, 32*
imems: 2, 3, 3
                                                                       root: 8, 10, 11, 13, 22, 31.
                                                                       rows: 3^*, 16, 20.
isspace: 13, 16.
j: 2*
                                                                       rr: 25, 26, 29, 30.
k: \quad \underline{2}^*, \ \underline{9}, \ \underline{11}^*, \ \underline{34}.
                                                                       sanity: 2* 11* 22.
l: \ \underline{25}, \ \underline{26}, \ \underline{33}, \ \underline{34}.
                                                                       sanity\_checking: 2, 11, 22.
last_col: 2, 7, 9, 10, 13, 14, 15, 16, 17, 20, 21,
                                                                       second: 7, 10, 13, 15, 16, 17, 20, 21, 25, 26, 29, 30.
     25, 26* 29, 30*
                                                                       show\_basics: 2, 3.
last_node: 2,* 7,* 9, 13, 16, 17, 18, 19, 20.
                                                                       show\_choices: 3, 22.
```

8 INDEX DLX2-CUTOFF  $\S 36$ 

```
show\_choices\_gap: \underline{3}, 4, 31.
show\_choices\_max: 3^*, 4, 22, 31.
show\_cutoffs: 3^*, 32^*
show\_details: \underline{3}^*, \underline{31}.
show\_full\_state: \underline{3}, 24.
show\_levels\_max: \underline{3}, 4, 33, 34.
show\_profile: 2, 3, 22.
show_tots: 2,* 3.*
show\_warnings: \underline{3}, \underline{4}
spacing: \underline{3}, \underline{4}, \underline{32}.
sscanf: 4.
stderr: 2,* 3,* 4, 9, 10, 11,* 12, 13, 16, 20, 21, 22,
      24, 31, 32, 33, 34, 35.
stdin: 13, 16.
stdout: 32*
stream: \underline{9}.
strlen: 13, 16.
strncmp: 14, 17.
t: 2, 11, 25, 26, 29, 30.
thresh: \underline{3}, \underline{4}, \underline{24}.
timeout: \underline{3}^*, 4, 24.
uint: \underline{2}^*
ullng: \underline{2}^*, 3^*, \underline{23}.
uncover: 22, <u>26</u>,* 28, 30.*
unpurify: 28, 30*
up: 2, 5, 9, 12, 15, 16, 18, 19, 25, 26, 27, 29, 30, 32
updates: 2, 3, 25, 29.
uu: \ \underline{25}, \ \underline{26}, \ \underline{29}, \ \underline{30}.
vbose: 2, 3, 4, 16, 22, 24, 31, 32.
x: \ \underline{29}, \ \underline{30}^*
```

DLX2-CUTOFF NAMES OF THE SECTIONS

```
\langle \text{ Check column } p \text{ 12} \rangle Used in section 11*.
(Check for duplicate column name 14) Used in section 13.
\langle \text{ Cover all other columns of } cur\_node 27 \rangle Used in section 22.
 Create a node for the column named in buf[p] 17 Used in section 16.
(Do special things if enough mems have accumulated 24) Used in section 22.
Global variables 3^*, 7^*, 23 Used in section 2^*.
(Initialize last_col to a new column with an empty list 15) Used in section 13.
\langle \text{Input the column names } 13 \rangle Used in section 2*.
\langle \text{Input the rows } 16 \rangle \text{ Used in section } 2^*.
\langle \text{Insert node } last\_node \text{ into the list for column } k \mid 18 \rangle Used in section 17.
\langle \text{ Print the profile 35} \rangle Used in section 2*.
\langle \text{Process the command line 4} \rangle Used in section 2*.
\langle \text{ Record solution and } \text{ goto } \text{ recover } 32^* \rangle Used in section 22.
\langle \text{Remove } last\_node \text{ from its column } 19 \rangle Used in section 16.
(Report the column totals 21) Used in section 2^*.
 Report the successful completion of the input phase 20 \ Used in section 2*.
\langle Set best_col to the best column for branching 31\rangle Used in section 22.
\langle \text{ Solve the problem } 22 \rangle Used in section 2*.
(Subroutines 9, 10, 11*, 25, 26*, 29, 30*, 33, 34) Used in section 2*.
\langle \text{Type definitions } 5, 6 \rangle Used in section 2*.
\langle Uncover all other columns of cur\_node\ 28\rangle Used in section 22.
```

## DLX2-CUTOFF

	Section	a $Pag\epsilon$
Data structures		5 2
Inputting the matrix	13	3 3
The dancing	2:	2 3
Indox	2	6 7