1*   Intro.   This program reads output from an o command in BDD14 into internal arrays, by brute force. Here I compute all the solutions, using ZDD conventions.

The output from an o command in BDD15 can be read in an identical fashion, so zddread.w is identical to bddread.w. However, the interpretations are different and the correct program must be applied to each output.

Note that if a variable is not present in the input to bddread then it is ignored. If there is a variable that is not present in the BDD (because it is allowed to be true or false in all solutions), then you need to add this possibility yourself. For example, you must multiply the BDD solution count by a factor of 2 for every such variable. This is typically not a problem in ZDDs, because a variable not present in a ZDD is forced to be false.

```
#define memsize  40000000    /* this many nodes */
#define varsize  8192     /* this many variables */
#define bdds  1     /* this many BDDs */
#define bufsize  100     /* buffer size; 100 is plenty big */

#include <stdio.h>
#include <stdlib.h>
  typedef struct {
    int v;
    int lo;
    int hi;
    int mark;
  } node;
  int present[varsize];
  node *mem[bdds];
  int stack[varsize];
  int stackptr;
  int serial;
  int root[bdds];
  FILE *infile;
  char buf[bufsize];
  unsigned int i1, i2, i3, i4;
  int memmax;
```

⟨ Subroutines 2* ⟩

```
  int main(int argc, char *argv[])
  {
    register int j, k, r, minv;

    for (r = 0; r < bdds; r++) {
      mem[r] = (node *) malloc(memsize * sizeof(node));
      if (¬mem[r]) {
        printf("Sorry,␣I␣can't␣allocate␣mem[%d]!\n", r);
        exit(−99);
      }
      for (k = 0; k < memsize; k++)  mem[r][k].lo = mem[r][k].hi = 0;
      if (¬(infile = fopen(argv[r + 1], "r"))) {
        printf("Sorry,␣I␣can't␣open␣'%s'␣for␣reading!\n", argv[r + 1]);
        exit(−1);
      }
      for (k = 0, minv = varsize; ; ) {
        if (¬fgets(buf, bufsize, infile))  break;
```

```
      j = sscanf (buf , "%x:␣(~%u?%x:%x)\n", &i1 , &i2 , &i3 , &i4 );
      if (j ≠ 4)  printf ("!␣I␣got␣only␣%d␣inputs␣from␣the␣line␣%s", j, buf );
      else {
        if (i1 > memmax )  memmax = i1 ;
        if (i3 > memmax )  memmax = i3 ;
        if (i4 > memmax )  memmax = i4 ;
        if (i1 ≥ memsize ∨ i2 ≥ varsize ∨ i3 ≥ memsize ∨ i4 ≥ memsize ) {
          printf ("!␣address␣out␣of␣range␣in␣the␣line␣%s", buf );
          exit (−69);
        } else if (mem[r][i1 ].lo ∨ mem[r][i1 ].hi )  printf ("!␣clobbered␣node␣in␣the␣line␣%s", buf );
        else {
          if (i2 < minv )  minv = i2 , root [r] = i1 ;
          k++, mem[r][i1 ].v = i2 , mem[r][i1 ].lo = i3 , mem[r][i1 ].hi = i4 ;
          present [i2 ] = 1;
        }
      }
    }
    fprintf (stderr , "%d␣nodes␣input␣into␣mem%d\n", k, r);
    fprintf (stderr , "(memmax=%d)\n", memmax );
  }
  for (j = k = 0; j < varsize ; j++)
    if (present [j ])  k++;
  fprintf (stderr , "There␣are␣%d␣variables.\n", k);
  ⟨ Do our thing 3* ⟩;
}
```

2* First, a recursive subroutine.

#**define** *var* (p)   *mem*[0][p].v

⟨ Subroutines 2* ⟩ ≡

```
  void printpaths (int p)
  {
    register int q;
    if (p ≤ 1) {
      printf ("%d:", serial );
      for (q = 0; q < stackptr ; q++)  printf ("␣%d", stack [q]);
      printf ("\n");
      serial ++;
      return;
    }
    q = mem[0][p].lo ;
    if (q)  printpaths (q);
    q = mem[0][p].hi ;
    if (q) {
      stack [stackptr ++] = mem[0][p].v ;
      printpaths (q);
      stackptr −−;
    }
  }
```

This code is used in section 1*.

3* ⟨Do our thing 3*⟩ ≡
   $mem[0][0].v = mem[0][1].v = varsize$;
   if $(root[0])$ $printpaths(root[0])$;

This code is used in section 1*.

4* Index.

The following sections were changed by the change file: 1, 2, 3, 4.

⟨ Do our thing 3* ⟩   Used in section 1*.
⟨ Subroutines 2* ⟩   Used in section 1*.

ZDDREAD-SOLS-ALL