

July 8, 2025 at 11:43

1* Intro. This program reads output from an `o` command in BDD14 into internal arrays, by brute force. Here I compute all the solutions of minimum weight. If there's more than one, I print only the first, but note that more are possible.

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 = mem[r][k].mark = 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 4*⟩;
}

```

2* First, a recursive subroutine to find the lengths of the lightest solutions. It also zeros out all links that don't lead to lightest solutions. (I could negate them instead of clobbering them.)

⟨Subroutines 2*⟩ ≡

```

void prep(int p)
{
    register int q, lomark, himark;
    q = mem[0][p].lo, lomark = varsize + 2;
    if (q) {
        if (mem[0][q].mark ≡ 0) prep(q);
        lomark = mem[0][q].mark;
    }
    q = mem[0][p].hi, himark = varsize + 2;
    if (q) {
        if (mem[0][q].mark ≡ 0) prep(q);
        himark = mem[0][q].mark + 1;
    }
    if (lomark > himark) mem[0][p].lo = 0, mem[0][p].mark = himark;
    else if (lomark < himark) mem[0][p].hi = 0, mem[0][p].mark = lomark;
    else mem[0][p].mark = lomark;
}

```

See also section 3*.

This code is used in section 1*.

3* Next, a recursive subroutine to print out all paths that remain.

⟨Subroutines 2*⟩ +≡

```

void printpaths(int p)
{
    register int q, s = stackptr;
    if (p ≤ 1) {
        if (serial) {
            printf("_and_more...\n");
            exit(0);
        } else {
            serial = 1;
            for (q = 0; q < stackptr; q++) printf("_%d", stack[q]);
            printf("_(length_%d)\n", stackptr);
        }
        return;
    }
    q = mem[0][p].hi;
    if (q) {
        stack[stackptr++] = mem[0][p].v;
        printpaths(q);
    }
    q = mem[0][p].lo;
    if (q) {
        printpaths(q);
    }
    stackptr = s;
}

```

4* ⟨Do our thing 4*⟩ ≡

```

mem[0][1].mark = 1, mem[0][1].v = varsize;
prep(root[0]);
printpaths(root[0]);

```

This code is used in section 1*.

5* Index.

The following sections were changed by the change file: 1, 2, 3, 4, 5.

argc: 1*
argv: 1*
bdds: 1*
buf: 1*
bufsize: 1*
exit: 1*, 3*
fgets: 1*
fopen: 1*
fprintf: 1*
hi: 1*, 2*, 3*
himark: 2*
infile: 1*
i1: 1*
i2: 1*
i3: 1*
i4: 1*
j: 1*
k: 1*
lo: 1*, 2*, 3*
lomark: 2*
main: 1*
malloc: 1*
mark: 1*, 2*, 4*
mem: 1*, 2*, 3*, 4*
memmax: 1*
memsize: 1*
minv: 1*
node: 1*
p: 2*, 3*
prep: 2*, 4*
present: 1*
printf: 1*, 3*
printpaths: 3*, 4*
q: 2*, 3*
r: 1*
root: 1*, 4*
s: 3*
serial: 1*, 3*
sscanf: 1*
stack: 1*, 3*
stackptr: 1*, 3*
stderr: 1*
v: 1*
varsize: 1*, 2*, 4*

⟨Do our thing 4*⟩ Used in section 1*.
⟨Subroutines 2*, 3*⟩ Used in section 1*.

ZDDREAD-MINSOLS-SHORT

	Section	Page
Intro	1	1
Index	5	4