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 the total number of solutions.

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];
  typedef struct {
    long long c2, c1;    /∗ upper and lower halves ∗/
  } dlong;
  dlong dlong0 = {0, 0};
  dlong dlong1 = {0, 1};
  dlong count[memsize];
  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 4*⟩;
}
```

2* First, two-longlong fixed point addition on nonnegative integers. Here and below I assume that the sums will not exceed 10^{36}.

```
#define ten_to_18th 1000000000000000000

⟨ Subroutines 2* ⟩ ≡
  dlong dadd(dlong x, dlong y)
  {
    dlong z;

    z.c1 = x.c1 + y.c1;
    if (z.c1 > ten_to_18th) z.c1 −= ten_to_18th, z.c2 = x.c2 + y.c2 + 1;
    else z.c2 = x.c2 + y.c2;
    if (z.c2 > ten_to_18th) {
      fprintf(stderr, "Possible␣integer␣overflow!\n");
      printf("Possible␣integer␣overflow!\n");
      exit(−666);
    }
    return z;
  }
  void print_dlong(FILE *f, dlong x)
  {
    if (x.c2) fprintf(f, "%lld%018lld", x.c2, x.c1);
    else printf("%lld", x.c1);
  }
```

See also section 3*.

This code is used in section 1*.

3* Next, a recursive subroutine.

```
⟨ Subroutines 2* ⟩ +≡
  void countsols(int p)
  {
    register int q;
    dlong c = dlong0;

    q = mem[0][p].lo;
    if (q) {
      if (mem[0][q].mark ≡ 0) countsols(q);
      c = count[q];
    }
    q = mem[0][p].hi;
    if (q) {
      if (mem[0][q].mark ≡ 0) countsols(q);
      c = dadd(c, count[q]);
    }
    mem[0][p].mark = 1;
    count[p] = c;
  }
```

4* ⟨ Do our thing 4* ⟩ ≡

```
  count[1] = dlong1, mem[0][1].mark = 1, mem[0][1].v = varsize;
  countsols(root[0]);
  print_dlong(stdout, count[root[0]]);
  printf("␣solutions.\n");
```

This code is used in section 1*.

5* Index.

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

ZDDREAD-COUNT