1. Intro. Given binary strings $s_1, \ldots, s_m$ of length n, and thresholds $r_1, \ldots, r_m$, this program generates clauses to find $x_1 \ldots x_n$ whose Hamming distance from s_j is at most r_j for each j.

String s_j appears on the jth line of *stdin*, as a sequence of `0`s and `1`s, followed by a space and the value of r_j.

```
#define maxn 10000      /* n shouldn't exceed this */
#define bufsize 10020   /* lines of stdin shouldn't be longer than this */
#include <stdio.h>
#include <stdlib.h>
  int r;      /* the current r_j */
  char buf[bufsize];
  int count[maxn + maxn];
  main()
  {
    register int i, j, jl, jr, k, m, n, t, tl, tr;
    n = −1;
    printf("~␣sat-closest-string\n");
    for (j = 1; ; j++) {
    getbuf: if (fgets(buf, bufsize, stdin) ≡ Λ) break;
      if (buf[0] ≡ '!') goto getbuf;      /* allow comments */
      ⟨ Generate clauses for the string in buf 2 ⟩;
    }
  }
```

2. ⟨ Generate clauses for the string in *buf* 2 ⟩ ≡

```
  ⟨ Parse the string in buf and find r 3 ⟩;
  ⟨ Generate cardinality clauses 5 ⟩;
```

This code is used in section 1.

3. ⟨ Parse the string in *buf* and find *r* 3 ⟩ ≡

```
for (i = 0; i < bufsize; i++)
  if (buf[i] ≠ '0' ∧ buf[i] ≠ '1') break;
if (i ≡ bufsize) {
  fprintf(stderr, "Input␣string␣%s␣didn't␣fit␣in␣the␣buffer!\n", buf);
  exit(−1);
}
if (i ≡ 0) {
  fprintf(stderr, "Null␣input␣string!\n");
  exit(−2);
}
if (buf[i] ≠ '␣') {
  buf[i] = '\0';
  fprintf(stderr, "Input␣string␣%s␣not␣followed␣by␣blank␣space!\n", buf);
  exit(−3);
}
buf[i] = '\0';
if (n < 0) {
  n = i;
  ⟨ Build the complete binary tree with n leaves 4 ⟩;
}
else if (n ≠ i) {
  fprintf(stderr, "Input␣string␣%s␣has␣length␣%d,␣not␣%d!\n", buf, i, n);
  exit(−4);
}
if (sscanf(buf + i + 1, "%d", &r) ≠ 1) {
  fprintf(stderr, "Input␣string␣%s␣not␣followed␣by␣a␣distance␣threshold!\n", buf);
  exit(−5);
}
if (r ≤ 0 ∨ r ≥ n) {
  fprintf(stderr, "The␣distance␣threshold␣for␣%s␣should␣be␣between␣1␣and␣%d!\n", buf, n − 1);
  exit(−6);
}
printf("~␣s%d=%s,␣r%d=%d\n", j, buf, j, r);
```

This code is used in section 2.

4. I'm using (again) the method of Bailleux and Boufkhad, explained in SAT-THRESHOLD-BB. It implicitly builds a tree with $2n-1$ nodes, with 0 as the root; the leaves start at node $n-1$. 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 *n* leaves 4 ⟩ ≡

```
for (k = n + n − 2; k ≥ n − 1; k−−) count[k] = 1;
for ( ; k ≥ 0; k−−) count[k] = count[k + k + 1] + count[k + k + 2];
if (count[0] ≠ n) {
  fprintf(stderr, "I'm␣totally␣confused.\n");
  exit(−666);
}
```

This code is used in section 3.

5. ⟨ Generate cardinality clauses 5 ⟩ ≡

```
for (i = n − 2; i; i−−) ⟨ Generate the clauses for node i 6 ⟩;
⟨ Generate the clauses at the root 7 ⟩;
```

This code is used in section 2.

6. 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 xbar(k)
        if (buf[(k) − n + 1] ≡ '0') printf("~x%d", (k) − n + 2);
        else printf("x%d", (k) − n + 2)
```

```
⟨Generate the clauses for node i 6⟩ ≡
  {
    t = count[i], tl = count[i + i + 1], tr = count[i + i + 2];
    if (t > r + 1) t = r + 1;
    if (tl > r) tl = r;
    if (tr > r) tr = r;
    for (jl = 0; jl ≤ tl; jl++)
      for (jr = 0; jr ≤ tr; jr++)
        if ((jl + jr ≤ t) ∧ (jl + jr) > 0) {
          if (jl) {
            if (i + i + 1 ≥ n − 1) xbar(i + i + 1);
            else printf("~%dB%d.%d", j, i + i + 2, jl);
          }
          if (jr) {
            printf("␣");
            if (i + i + 2 ≥ n − 1) xbar(i + i + 2);
            else printf("~%dB%d.%d", j, i + i + 3, jr);
          }
          if (jl + jr ≤ r) printf("␣%dB%d.%d\n", j, i + 1, jl + jr);
          else printf("\n");
        }
  }
```

This code is used in section 5.

7. Finally, we assert that at most r of the (xs)'s are true, by implicitly asserting that the (nonexistent) variable j`B1.`r+1 is false.

```
⟨Generate the clauses at the root 7⟩ ≡
  tl = count[1], tr = count[2];
  if (tl > r) tl = r;
  for (jl = 1; jl ≤ tl; jl++) {
    jr = r + 1 − jl;
    if (jr ≤ tr) {
      if (1 ≥ n − 1) xbar(1);
      else printf("~%dB2.%d", j, jl);
      printf("␣");
      if (2 ≥ n − 1) xbar(2);
      else printf("~%dB3.%d", j, jr);
      printf("\n");
    }
  }
```

This code is used in section 5.

8. Index.

SAT-CLOSEST-STRING