

(Downloaded from <https://cs.stanford.edu/~knuth/programs.html> and typeset on May 28, 2023)

1. Intro. Analyzing the graph created by TIC-TAC-TOE1.

I'm experimenting with a scoring system that includes “difficulty” as part of the desirability of a position. If you have to think harder to win, the position isn't quite as good as one in which no-brainer moves will take you home. Thus, one tries to move to positions that require the opponent to keep alert.

Furthermore, I consider it more desirable to win with lots of marks on the board than with fewer (because you have kindly refrained from humiliating your opponent).

```
#define rank z.I      /* number of moves made */
#define link y.V      /* next vertex of same rank */
#define head x.V      /* first vertex of given rank */
#define winner w.I    /* is this a winning position? */
#define bitcode v.I   /* binary representation of this position */
#define follower u.V  /* an optimal move goes here */
#define score w.I     /* the primary value of this position */
#define nobrainer u.I /* immediate or threatened win */

#include "gb_graph.h"
#include "gb_save.h"
Vertex * pos[1 << 18];
unsigned int diff[1 << 18]; /* the difficulty factor */
int count[9], win[9];

main()
{
    register j, k, s, t, tt, auts;
    register unsigned int d;
    Vertex * u, * v;
    Arc * a;
    Graph * g = restore_graph("/tmp/tictactoe.gb");
    < Fill in the nobrainer fields 2 >;
    for (k = 9; k ≥ 0; k--)
        for (v = (g->vertices + k)-head; v; v = v->link) {
            pos[v->bitcode] = v;
            < Compute the score and difficulty of v 3 >;
        }
    for (k = 9; k ≥ 0; k--)
        for (v = (g->vertices + k)-head; v; v = v->link) {
            < Determine the equivalence class of v 4 >;
        }
    for (k = 0; k ≤ 9; k++) printf("(%d,%d)␣classes␣after␣%d␣moves\n", count[k], win[k], k);
}
```

2. The *nobrainer* field is set to +1 if there's a one-move win, and to -1 if the opponent has a one-move win that should be blocked.

⟨ Fill in the *nobrainer* fields 2 ⟩ ≡

```

for ( $k = 9$ ;  $k \geq 0$ ;  $k--$ )
  for ( $v = (g\text{-vertices} + k)\text{-head}$ ;  $v$ ;  $v = v\text{-link}$ ) {
    for ( $a = v\text{-arcs}$ ;  $a$ ;  $a = a\text{-next}$ ) {
       $u = a\text{-tip}$ ;
      if ( $u\text{-winner}$ ) {
         $v\text{-nobrainer} = 1$ ;
        break;
      }
      if ( $u\text{-nobrainer} > 0$ )  $v\text{-nobrainer} = -1$ ; /* may be set +1 later */
    }
  }

```

This code is used in section 1.

3. The *score* field takes the place of what used to be called *winner*. Scores are computed from the standpoint of the X player: A score of +6 means that X can force a win at move six; a score of -9 means that O can win (or has won) at move nine. A score of zero means that a draw is the best possible outcome.

Positions with equal score are ranked secondarily by their *diff*, which is a sequence of 8 hexadecimal digits. The most significant digit is the number of nonoptimal moves facing the player at this position. The next digit is the *complement* of the number of nonoptimal moves facing the opponent, after the player has made an optimal move. And so on; even-numbered digits are complemented with respect to f.

⟨ Compute the *score* and difficulty of v 3 ⟩ ≡

```

if ( $v\text{-winner}$ )  $v\text{-score} = -v\text{-rank}$ ,  $\text{diff}[v\text{-bitcode}] = \#0f0f0f0f$ ;
else if ( $v\text{-rank} \equiv 9$ )  $v\text{-score} = 0$ ,  $\text{diff}[v\text{-bitcode}] = \#0f0f0f0f$ ;
else {
  for ( $s = 99$ ,  $a = v\text{-arcs}$ ;  $a$ ;  $a = a\text{-next}$ ) {
     $u = a\text{-tip}$ ;
    if ( $s > u\text{-score} \vee (s \equiv u\text{-score} \wedge d < \text{diff}[u\text{-bitcode}])$ )  $s = u\text{-score}$ ,  $d = \text{diff}[u\text{-bitcode}]$ ;
  }
   $t = v\text{-nobrainer}$ ; /* the nobrainer field will become follower now */
  for ( $j = 0$ ,  $a = v\text{-arcs}$ ;  $a$ ;  $a = a\text{-next}$ ) {
     $u = a\text{-tip}$ ;
    if ( $s \neq u\text{-score} \vee d \neq \text{diff}[u\text{-bitcode}]$ )  $j++$ ;
    else  $v\text{-follower} = u$ ;
  }
  if ( $t < 0 \vee s \equiv -k - 1$ )  $j = 0$ ;
   $v\text{-score} = -s$ ,  $\text{diff}[v\text{-bitcode}] = (j \ll 28) + ((\#ffffff - d) \gg 4)$ ;
}

```

This code is used in section 1.

4. The tic-tac-toe board has eight automorphisms. So I can save a factor of roughly eight when I'm trying to understand this data.

⟨ Determine the equivalence class of v 4 ⟩ \equiv

```

    t = tt = v-bitcode, auts = 1;
    for (j = 1; j < 4; j++) {
        t = t  $\oplus$  ((t  $\oplus$  (t  $\gg$  2)) & #3f3f)  $\oplus$  ((t  $\oplus$  (t  $\ll$  6)) & #c0c0);    /* rotate 90° */
        if (t < tt) tt = t, auts = 1;
        else if (t  $\equiv$  tt) auts++;
    }
    t = t  $\oplus$  ((t  $\oplus$  (t  $\gg$  2)) & #3300)  $\oplus$  ((t  $\oplus$  (t  $\ll$  2)) & #cc00)  $\oplus$  ((t  $\oplus$  (t  $\gg$  4)) & #3)  $\oplus$  ((t  $\oplus$  (t  $\ll$  4)) & #30);
    /* reflect */
    if (t < tt) tt = t, auts = 1;
    else if (t  $\equiv$  tt) auts++;
    for (j = 1; j < 4; j++) {
        t = t  $\oplus$  ((t  $\oplus$  (t  $\gg$  2)) & #3f3f)  $\oplus$  ((t  $\oplus$  (t  $\ll$  6)) & #c0c0);    /* rotate 90° */
        if (t < tt) tt = t, auts = 1;
        else if (t  $\equiv$  tt) auts++;
    }
    if (tt  $\equiv$  v-bitcode) ⟨ Print the best move from  $v$  5 ⟩;
    if (v-score  $\neq$  pos[tt]-score  $\vee$  diff[v-bitcode]  $\neq$  diff[pos[tt]-bitcode]) printf("I goofed!\n");

```

This code is used in section 1.

5. Whenever v is the leader of an equivalence class, I print out its characteristics.

⟨ Print the best move from v 5 ⟩ \equiv

```

{
    count[v-rank]++;
    printf("%s has score %d(%08x), size %d", v-name, v-score, diff[v-bitcode]  $\oplus$  #0f0f0f0f, 8/auts);
    if (v-follower) {
        printf(", >");
        for (j = 0; j  $\leq$  10; j++)
            printf("%c", v-follower-name[j] < 'A' ? v-follower-name[j] : 'X' + 'O' - v-follower-name[j]);
    } else win[k]++;
    printf("\n");
}

```

This code is used in section 4.

6. Index.

Arc: 1.
arcs: 2, 3.
auts: 1, 4, 5.
bitcode: 1, 3, 4, 5.
count: 1, 5.
d: 1.
diff: 1, 3, 4, 5.
follower: 1, 3, 5.
Graph: 1.
head: 1, 2.
j: 1.
k: 1.
link: 1, 2.
main: 1.
name: 5.
next: 2, 3.
nobrainier: 1, 2, 3.
pos: 1, 4.
printf: 1, 4, 5.
rank: 1, 3, 5.
restore_graph: 1.
s: 1.
score: 1, 3, 4, 5.
t: 1.
tip: 2, 3.
tt: 1, 4.
Vertex: 1.
vertices: 1, 2.
win: 1, 5.
winner: 1, 2, 3.

- ⟨ Compute the *score* and difficulty of v 3 ⟩ Used in section 1.
- ⟨ Determine the equivalence class of v 4 ⟩ Used in section 1.
- ⟨ Fill in the *nobrainer* fields 2 ⟩ Used in section 1.
- ⟨ Print the best move from v 5 ⟩ Used in section 4.

TICTACTOE2

	Section	Page
Intro	1	1
Index	6	4