1. Intro. Johan de Ruiter presented a beautiful puzzle on 14 March 2018, based on the first 32 digits of π.

It's a special case of the following self-referential problem: Given a directed graph, find all vertex labelings such that each vertex is labeled with the number of distinct labels on its successors.

In Johan's puzzle, some of the labels are given, and we're supposed to find the others. He also presented the digraph in terms of a 10×10 array, with each cell pointing either north, south, east, or west; its successors are the cells in that direction.

I've written this program so that it could be applied to fairly arbitrary digraphs, if I decide to make it more general.

At one time I thought that the output of this program should be input to DLX2-SHARP. But I've now seen cases where that is definitely *not* a good idea.

[Hacked from BACK-PI-DAY.]

```
#define N (0 ≪ 4)
#define S (1 ≪ 4)
#define E (2 ≪ 4)
#define W (3 ≪ 4)
#define size 10
#define verts (size ∗ size)    /∗ vertices in the digraph ∗/
#define maxd (size − 1)    /∗ maximum out-degree in the digraph; must be less than 16 ∗/

#include <stdio.h>
#include <stdlib.h>
  char johan[size][size] = {
      {S + 3, W + 1, E + 4, W + 0, S + 1, W + 0, S + 5, S + 0, S + 9, S + 0},
      {E + 0, S + 0, W + 2, S + 6, S + 0, E + 0, S + 0, W + 0, E + 0, S + 5},
      {E + 0, S + 0, E + 0, E + 0, S + 0, S + 0, E + 3, S + 5, W + 8, W + 9},
      {E + 0, E + 0, S + 0, N + 0, S + 0, E + 0, W + 0, S + 0, W + 7, W + 0},
      {E + 9, E + 0, S + 3, S + 0, S + 0, S + 0, W + 0, W + 0, S + 0, W + 0},
      {E + 0, E + 0, E + 0, W + 0, S + 0, E + 0, S + 0, E + 2, S + 0, S + 3},
      {E + 0, E + 8, S + 0, N + 0, S + 0, S + 0, N + 0, W + 0, N + 0, W + 0},
      {N + 4, E + 6, S + 2, N + 6, S + 0, E + 0, S + 0, W + 0, S + 0, N + 0},
      {N + 4, E + 0, E + 0, E + 0, S + 0, W + 0, W + 3, W + 3, W + 0, N + 0},
      {E + 0, E + 8, N + 0, W + 3, N + 0, N + 2, W + 0, W + 7, N + 9, N + 5}};
  int deg[verts], arcs[verts], scra[verts], known[verts], forced[verts];
  char name[verts][8];    /∗ each vertex name is assumed to be at most six characters ∗/
  int tip[2 ∗ verts ∗ verts], next[2 ∗ verts ∗ verts];
  int arcptr = 0;    /∗ this many entries of tip and next are in use ∗/

  main()
  {
    register int a, aa, b, d, i, j, k, u, uu, v, w, ww;

    printf("|␣pi-day-dlx\n");
    ⟨Set up the graph 2⟩;
    ⟨Print the item-name line 3⟩;
    ⟨Print the options 4⟩;
  }
```

2. The arcs from vertex v begin at *arcs*[v], as in the Stanford GraphBase. The reverse arcs that run *to* vertex v begin at *scra*[v].

```
#define inx(i, j) (size ∗ (i) + (j))
#define newarc(ii, jj) next[++arcptr] = arcs[inx(i, j)], tip[arcptr] = inx(ii, jj), arcs[inx(i, j)] = arcptr,
            next[++arcptr] = scra[inx(ii, jj)], tip[arcptr] = inx(i, j), scra[inx(ii, jj)] = arcptr, d++
```

⟨ Set up the graph 2 ⟩ ≡

```
  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++) {
      v = inx(i, j);
      sprintf(name[v], "%d%d", i, j);
      known[v] = johan[i][j] & #f;
      if (¬known[v]) known[v] = −1;
      d = 0;
      switch (johan[i][j] ≫ 4) {
      case N ≫ 4:
        for (k = i − 1; k ≥ 0; k−−) newarc(k, j); break;
      case S ≫ 4:
        for (k = i + 1; k ≤ maxd; k++) newarc(k, j); break;
      case E ≫ 4:
        for (k = j + 1; k ≤ maxd; k++) newarc(i, k); break;
      case W ≫ 4:
        for (k = j − 1; k ≥ 0; k−−) newarc(i, k); break;
      }
      if (d > maxd) {
        fprintf(stderr, "The␣outdegree␣of␣%s␣should␣be␣at␣most␣%d,␣not␣%d!\n", name[v], maxd, d);
        exit(−1);
      }
      deg[v] = d;
      if (d ≤ 1) known[v] = d;
    }
```

This code is used in section 1.

3. Based on ideas by Ricardo Bittencourt, I'm using primary items `#`v for each vertex v, as well as `#+`v for each unknown vertex v, to govern decisions. There are secondary items v whose color is the label on v. Also secondary items `h`vd and `i`vd, whose values/colors respectively represent the predicates [d is seen by v], [d is the label of v].

Nonsharp primary items `E`vd are also introduced, to enforce the relation between `h` and `i` values/colors. These items are omitted, as well as the secondary items `h`vd, when d is known to be visible from v.

The degree of a vertex is adjusted by subtracting 1 whenever v is known to see a label that matches one already seen. The adjusted degree is therefore the largest possible label that v could conceivably have.

Items `i`vd are omitted when the label of v is known in advance, or when d exceeds the adjusted degree of v.

[Certain other secondary items could also be deduced from the given partially labeled graph; but they are retained, for simplicity.]

```
⟨ Print the item-name line 3 ⟩ ≡
  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++) {
      v = inx(i, j);
      printf("#%s␣", name[v]);
      if (known[v] < 0) printf("#+%s␣", name[v]);
    }
  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++) {
      v = inx(i, j);
      for (b = 0, a = arcs[v]; a; a = next[a]) {
        w = tip[a];
        if (known[w] ≥ 0) {
          if (b & (1 ≪ known[w])) deg[v]−−;      /∗ duplicate label already seen ∗/
          else b += 1 ≪ known[w];      /∗ v sees a new known label ∗/
        }
      }
      forced[v] = b;
      for (d = 1; d ≤ maxd; d++)
        if ((b & (1 ≪ d)) ≡ 0)      /∗ the label d isn't forced ∗/
          printf("E%s%x␣", name[v], d);
    }
  printf("|");
  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++) {
      v = inx(i, j);
      printf("␣%s", name[v]);
    }
  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
      for (d = 1; d ≤ maxd; d++) {
        v = inx(i, j);
        if ((forced[v] & (1 ≪ d)) ≡ 0) printf("␣h%s%x", name[v], d);
        if (known[v] < 0 ∧ d ≤ deg[v]) printf("␣i%s%x", name[v], d);
      }
  printf("\n");
```

This code is used in section 1.

4. ⟨ Print the options 4 ⟩ ≡

```
⟨ Print the # options 5 ⟩;
⟨ Print the #+ options 6 ⟩;
⟨ Print the E options 7 ⟩;
```

This code is used in section 1.

5. A `#` option states that v has a certain value d and that a certain set of d labels is visible from v.

For simplicity, I compute more combinations than needed, using Gosper's hack, and reject the unsuitable ones. (More precisely, I compute all $\binom{maxd}{d}$ sets of d labels, but reject the cases that don't include every forced label. Furthermore, I reject cases that include the unforced label 0, because 0 is a legal label only when it is forced.)

⟨ Print the # options 5 ⟩ ≡

```
for (i = 0; i < size; i++)
  for (j = 0; j < size; j++) {
    v = inx(i, j);
    for (d = 0; d ≤ deg[v]; d++) {
      if (known[v] ≥ 0 ∧ d ≠ known[v]) continue;
      for (a = (1 ≪ d) − 1; a < 1 ≪ (maxd + 1); ) {
        if ((a & forced[v]) ≡ forced[v] ∧ ((a ⊕ forced[v]) & 1) ≡ 0) {
          printf("#%s␣%s:%x", name[v], name[v], d);
          for (k = 1; k ≤ maxd; k++)
            if ((forced[v] & (1 ≪ k)) ≡ 0) printf("␣h%s%x:%d", name[v], k, a & (1 ≪ k) ? 1 : 0);
          printf("\n");
        }
        if (¬a) break;
        u = a & −a;      /* Gosper's hack (exercise 7.1.3–20) */
        uu = a + u;
        a = uu + (((uu ⊕ a)/u) ≫ 2);
      }
    }
  }
```

This code is used in section 4.

6. A `#+` option states that v has a certain value d and that all of its predecessors can see it.

⟨ Print the #+ options 6 ⟩ ≡

```
for (i = 0; i < size; i++)
  for (j = 0; j < size; j++) {
    v = inx(i, j);
    if (known[v] ≥ 0) continue;
    for (d = 1; d ≤ deg[v]; d++) {
      printf("#+%s␣%s:%x", name[v], name[v], d);
      for (k = 1; k ≤ deg[v]; k++) printf("␣i%s%x:%d", name[v], k, k ≡ d ? 1 : 0);
      for (a = scra[v]; a; a = next[a]) {
        w = tip[a];
        if ((forced[w] & (1 ≪ d)) ≡ 0) printf("␣h%s%x:1", name[w], d);
      }
      printf("\n");
    }
  }
```

This code is used in section 4.

7. The `E`vd options state that `h`vd is 1 if and only if one of v's successors is labeled d. We don't bother to check this condition when it is already known to be true. So we only look at unknown successors of v.

⟨ Print the `E` options 7 ⟩ ≡

```
for (i = 0; i < size; i++)
  for (j = 0; j < size; j++) {
    v = inx(i, j);
    b = forced[v];
    for (d = 1; d ≤ maxd; d++)
      if ((b & (1 ≪ d)) ≡ 0) {      /∗ the label d isn't forced ∗/
        for (a = arcs[v]; a; a = next[a]) {
          w = tip[a];
          if (known[w] ≥ 0) continue;
          if (d > deg[w]) continue;
          printf("E%s%x␣h%s%x:1␣i%s%x:1", name[v], d, name[v], d, name[w], d);
          for (aa = arcs[v]; aa ≠ a; aa = next[aa]) {
            ww = tip[aa];      /∗ ww is an already treated successor of v ∗/
            if (known[ww] ≥ 0) continue;
            if (d ≤ deg[ww]) printf("␣i%s%x:0", name[ww], d);
          }
          printf("\n");
        }
        printf("E%s%x␣h%s%x:0", name[v], d, name[v], d);
        for (a = arcs[v]; a; a = next[a]) {
          w = tip[a];
          if (known[w] ≥ 0) continue;
          if (d ≤ deg[w]) printf("␣i%s%x:0", name[w], d);
        }
        printf("\n");
      }
  }
```

This code is used in section 4.

8. Index.

PI-DAY-DLX