```c
#include <stdio.h>
#include <gmp.h>
#include <cairo.h>
#include <pipewire/pipewire.h>
#include <sys/types.h>
#include <sys/stat.h>
typedef float f32;
typedef double f64;
typedef u_int8_t u8;
typedef u_int16_t u16;
typedef u_int32_t u32;
typedef u_int64_t u64;
typedef  int8_t i8;
typedef  int16_t i16;
typedef  int32_t i32;
typedef  int64_t i64;
typedef  size_t usize;
typedef  ssize_t isize;

int radical(int n) {
    if (n < 1) return 0;
    int radn = 1;
    if (n % 2 == 0) {
        radn *= 2;
        while (n % 2 == 0) n /= 2;
    }
    for (int i = 3; i * i <= n; i += 2)
{
      if (n % i == 0) {
      radn *= i;
      while (n % i == 0) n /= i;
    }
  }
  return  n * radn;
}
```

```
#define W 500
#define H 500

static unsigned char img[W * H * 3];

float capsuleSDF(float px, float py,
float ax, float ay, float bx, float by,
float r) {
    float pax = px - ax, pay = py - ay,
bax = bx - ax, bay = by - ay;
    float h = fmaxf(fminf((pax * bax +
pay * bay) / (bax * bax + bay * bay),
1.0f), 0.0f);
    float dx = pax - bax * h, dy = pay
- bay * h;
    return sqrtf(dx * dx + dy * dy) -
r;
}

void alphablend(int x, int y, float
alpha, float r, float g, float b) {
    unsigned char* p = img + (y * W +
x) * 3;
    p[0] = (unsigned char)(p[0] * (1 -
alpha) + r * alpha * 255);
    p[1] = (unsigned char)(p[1] * (1 -
alpha) + g * alpha * 255);
    p[2] = (unsigned char)(p[2] * (1 -
alpha) + b * alpha * 255);
}

void lineSDFAABB(float ax, float ay,
float bx, float by, float r) {
    int x0 = (int)floor(fminf(ax, bx) -
r);
    int x1 = (int) ceil(fmaxf(ax, bx) +
```

```
r);
    int y0 = (int)floor(fminf(ay, by) -
r);
    int y1 = (int) ceil(fmaxf(ay, by) +
r);
    for (int y = y0; y <= y1; y++)
        for (int x = x0; x <= x1; x++)
            alphablend(x, y,
fmaxf(fminf(0.5f - capsuleSDF(x, y, ax,
ay, bx, by, r), 1.0f), 0.0f), 0.0f,
0.0f, 0.0f);
}
/*
int main() {
    memset(img, 255, sizeof(img));

    float cx = W * 0.5f, cy = H * 0.5f;
    for (int j = 0; j < 5; j++) {
        float r1 = fminf(W, H) * (j +
0.5f) * 0.085f;
        float r2 = fminf(W, H) * (j +
1.5f) * 0.085f;
        float t = j * PI / 64.0f, r =
(j + 1) * 0.5f;
        for (int i = 1; i <= 64; i++, t
+= 2.0f * PI / 64.0f) {
            float ct = cosf(t), st =
sinf(t);
            lineSDFAABB(cx + r1 * ct,
cy - r1 * st, cx + r2 * ct, cy - r2 *
st, r);
        }
    }
}*/

typedef struct {
```

```
  uint8_t r;
  uint8_t g;
  uint8_t b;
  uint8_t a;
} pxl;

typedef enum {
  amethyst, blue, caramel, damnson,
ebony,
  forest, green, honey, iron, jade,
  khaki, lime, mellow, navy, orly,
  pink, quagmire, red, sky, turquoise,
  uranium, version, wine, xanthin,
yellow,
  zorange
} ncolor;

static char *colorname[26] = {
  "amethyst", "blue", "caramel",
"damnson", "ebony",
  "forest", "green", "honey", "iron",
"jade",
  "khaki", "lime", "mellow", "navy",
"orly",
  "pink", "quagmire", "red", "sky",
"turquoise",
  "uranium", "version", "wine",
"xanthin", "yellow",
  "zorange"
};
PANIC
static const pxl colors26[26] = {
  {241,163,255}, {0,116,255},
{155,64,0}, {76,0,92}, {26,26,26},
  {0,92,48}, {42,207,72},
{255,205,153}, {126,126,126},
```

```
{149,255,181},
   {143,124,0}, {157,205,0},
{195,0,137}, {50,129,255}, {165,4,255},
   {255,169,187}, {66,102,0}, {255,0,0},
{94,241,243}, {0,153,143},
   {225,255,102}, {16,10,255},
{153,0,0}, {255,255,129},{255,225,0},
   {255,80,0}
};

#define BIG_SZ 4210 * 2976 * 16

typedef struct {
  int nr;
  int nrj;
  int pr[BIG_SZ];
} pxr;

int pxr_count(pxr *pr) {
  return pr->nr - pr->nrj;
}

int pxr_get(pxr *pr, int x, int y, int w) {
  int n = (y * w) + x;
  int r = pr->pr[n];
  eturn r;

r_new(pxr *pr, int x, int y, int w) {
 = (y * w) + x;
 = ++pr->nr;
n] = r;
("New Region: %d @ %dx

_join(pxr *pr, int r, int x,
 = (y * 
 d
pr[n];
 &&
 _r
 0;

 && (
 (p2->g <
)) return 1;
```

```
  if ((p1->r == p2->r) && (p1->g == p2->g) && (p1->b == p2->b)) return
1;
  return 0;
}

int pxrscan(pxr *pr, pxl *px, int w, int h) {
  if ((w < 1) || (h < 1) || !pr || !px) return 1;
  int x;
  int y;
  int r;
  u64 np = w * h;
  printf("Area: %dx%d %lu MAX: %d\n",  w, h, np, BIG_SZ);
  if (np > BIG_SZ) { printf("To fn big many pixels!\n"); return 1; }
  for (y = 0; y < h; y++) {
    for (x = 0; x < w; x++) {
      pxl *upleft = NULL;
      pxl *up = NULL;
      pxl *left = NULL;
      pxl *upright = NULL;
      pxl *cur = &px[(y * w) + x];
      if (x > 0) left = &px[(y * w) + (x - 1)];
      if (y > 0) {
        up = &px[((y - 1) * w) + x];
        if (x > 0) upleft = &px[((y - 1) * w) + (x - 1)];
        if (x < (w - 1)) upright = &px[((y - 1) * w) + (x + 1)];
      }
      if ((upleft) && (pxlcmp(cur, upleft))) {
        r = pxr_get(pr, x - 1, y - 1, w);
        pxr_join(pr, r, x, y, w);
      }
      if ((up) && (pxlcmp(cur, up))) {
        r = pxr_get(pr, x, y - 1, w);
        pxr_join(pr, r, x, y, w);
      }
      if ((left) && (pxlcmp(cur, left))) {
        r = pxr_get(pr, x - 1, y, w);
        pxr_join(pr, r, x, y, w);
      }
      if ((upright) && (pxlcmp(cur, upright))) {
        r = pxr_get(pr, x + 1, y - 1, w);
        pxr_join(pr, r, x, y, w);
      }
      if (!pxr_get(pr, x, y, w)) {
        pxr_new(pr, x, y, w);
      }
    }
  }
  return pxr_count(pr);
}

int pxrprint(pxr *pr, pxl *px, int w, int h, char *filename) {
  if ((w < 1) || (h < 1) || !pr || !px) return 1;
  int x;
  int y;
  u64 r;
  u64 n = pr->nr + pr->nrj;
  char name[4096];
  memset(name, 0, 4096);
  sprintf(name, "%s.nfo", filename);
  mkdir(name, S_IRWXU | S_IRWXG | S_IROTH);
  /*
  for (r = 0; r <= n; r++) {
    cairo_surface_t *s = NULL;
    s = cairo_image_surface_create(CAIRO_FORMAT_ARGB32, w, h);
```

```
    if (cairo_surface_status(s)) return 1;
    cairo_t *c = cairo_create(s);
    cairo_set_source_rgba(c, 1, 1, 1, 0);
    cairo_paint(c);
    cairo_destroy(c);
    cairo_surface_flush(s);
    pxl *npx = (pxl *)cairo_image_surface_get_data(s);
    int rpx = 0;
    for (y = 0; y < h; y++) {
      for (x = 0; x < w; x++) {
        if (r != pxr_get(pr, x, y, w)) continue;
        npx[(y * w) + x] = px[(y * w) + x];
        rpx++;
      }
    }
    if (rpx) {
      cairo_surface_mark_dirty(s);
      memset(name, 0, 4096);
      sprintf(name, "%s.nfo/%lu.png", filename, r);
      cairo_surface_write_to_png(s, name);
    }
    cairo_surface_destroy(s);
  }
  */
  for (r = 0; r <= n; r++) {
    int rt = -1;
    int rl = -1;
    int rr = 0;
    int rb = 0;
    for (y = 0; y < h; y++) {
      for (x = 0; x < w; x++) {
        if (r != pxr_get(pr, x, y, w)) continue;
        if (rt == -1) { rt = y; rl = x; }
        if (x < rl) rl = x;
        if (x > rr) rr = x;
        if (y > rb) rb = y;
      }
    }
    if ((rl == -1) && (rt == -1)) continue;
    int rw = (rr - rl) + 1;
    int rh = (rb - rt) + 1;
    printf("r: %lu %d,%d %dx%d\n", r, rl, rt, rw, rh);
    cairo_surface_t *s = NULL;
    s = cairo_image_surface_create(CAIRO_FORMAT_ARGB32, rw, rh);
    if (cairo_surface_status(s)) return 1;
    cairo_t *c = cairo_create(s);
    cairo_set_source_rgb(c, 1, 0, 0);
    cairo_paint(c);
    pxl *npx = (pxl *)cairo_image_surface_get_data(s);
    for (y = 0; y < rh; y++) {
      for (x = 0; x < rw; x++) {
        npx[y * rw + x] = px[(y + rt) * w + (x + rl)];
      }
    }
    cairo_surface_mark_dirty(s);
    memset(name, 0, 4096);
    sprintf(name, "%s.nfo/%lu_out.png", filename, r);
    cairo_surface_write_to_png(s, name);
    cairo_surface_destroy(s);
  }
  return 0;
}

int main(int argc, char *argv[]) {
```

```
  mpz_t n;
  mpz_init(n);
  mpz_clear(n);
  pw_init(&argc, &argv);
  printf("pipewire %s/%s ", pw_get_headers_version(),
   pw_get_library_version());
  printf("cairo: %s gmp: %s\n", cairo_version_string(), gmp_version);
  cairo_surface_t *s = NULL;
  if (argc == 2) {
    s = cairo_image_surface_create_from_png(argv[1]);
    if (s) {
      if (cairo_surface_status(s)) return 1;
      pxr *pr;
      pr = malloc(sizeof(*pr));
      memset(pr, 0, sizeof(*pr));
      printf("scanning %s\n", argv[1]);
      pxrscan(pr, (pxl *)cairo_image_surface_get_data(s),
       cairo_image_surface_get_width(s),
       cairo_image_surface_get_height(s));
      pxrprint(pr, (pxl *)cairo_image_surface_get_data(s),
       cairo_image_surface_get_width(s),
       cairo_image_surface_get_height(s),
       argv[1]);
      free(pr);
      cairo_surface_destroy(s);
    }
  } else {
    for (int L = 0; L < 26; L++) { printf("%s %d %d %d\n",
      colorname[L], colors26[L].r, colors26[L].g, colors26[L].b);
    }
    int log[1024];
    memset(log,0,sizeof(log));
    u64 i = 2;
    for (;i < 1000000000;) {
      u64 n = i;
      u64 steps = 0;
      for (; n != 1;) {
        steps++;
        if ((n % 2) == 0) { n = n / 2; } else { n = (n * 3) + 1; }
      }
      log[steps]++;
      if (steps == 26) printf("742    %10lu: %4lu steps #[%d]\n", i,
steps, log[steps]);
/*      if (steps == 742) printf("742 %10lu: %4lu steps #[%d]\n", i,
steps, log[steps]);
      if (steps == 800) printf("800 %10lu: %4lu steps #[%d]\n", i,
steps, log[steps]);
      if (steps == 803) printf("803 %10lu: %4lu steps #[%d]\n", i,
steps, log[steps]);
 */     i++;
    }
    for (i = 0; i < 1000; i++) {
      if (!log[i]) continue;
    //  printf("%10lu: %3d\n", i, log[i]);
    }
  }
  return 1;
}
```