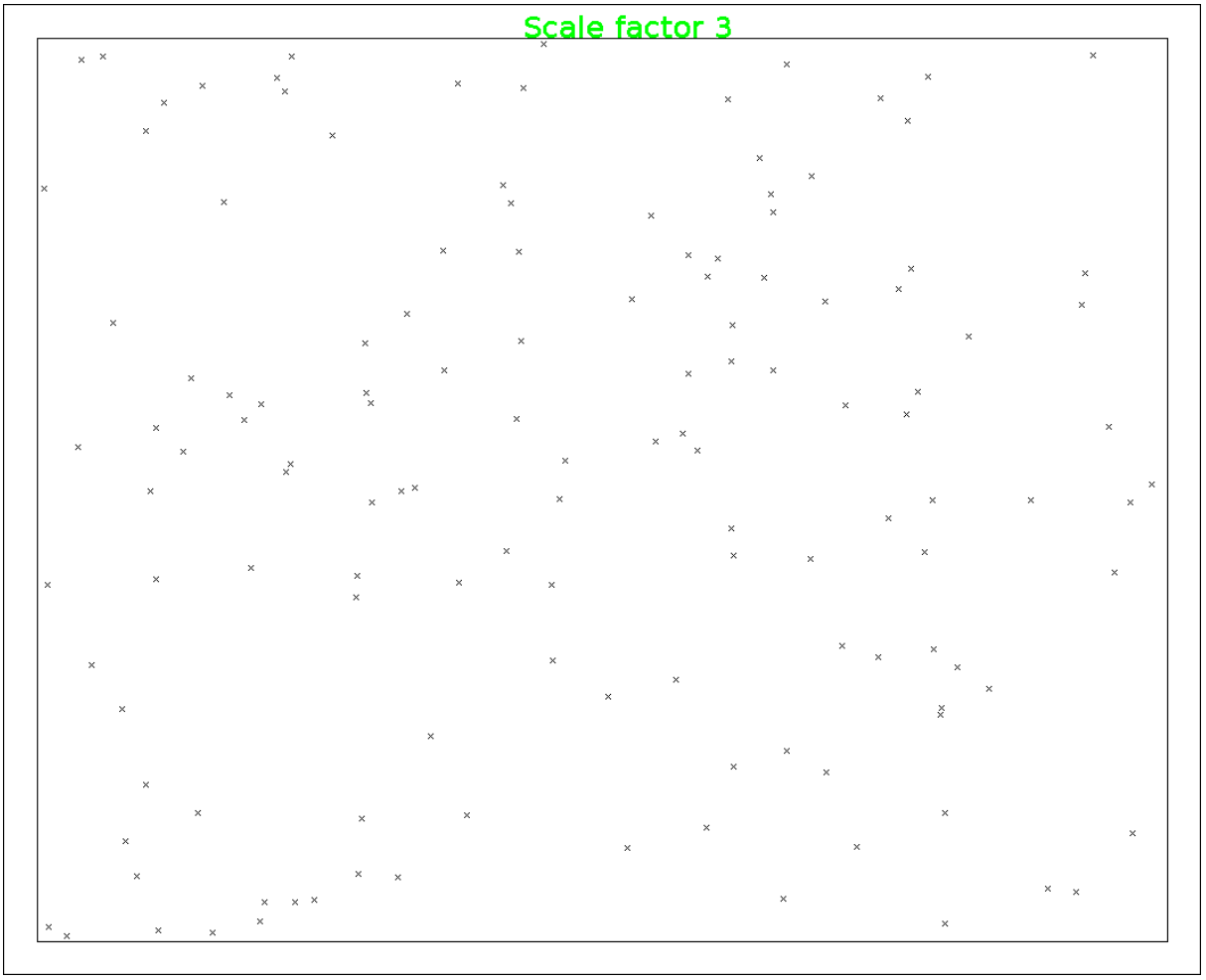
OPTISCHEMA - CODING 400

“R-boy realises the 1964 Mars Rover was working perfectly, but had a performance issue – still present in today's system – which prevented it from completing its mission. Help him optimise the Rover's system to fix this issue.”

This challenge ask us to solve some *travelling salesman problem*. In particular we have to solve 512 TSP consecutively and within 4 seconds per TSP.

The problem in encoded in an image like this one:



The word “Scale factor X” tell which is the scale of the map.

The markers indicate the points of interest for the TSP, as written in the rules we can have different types of POI (X, 0, +), we have to consider each type of marker as a different TSP problem and do the sum for the solution.

After we got the picture from the website, we must elaborate it.

For the extraction of the scale, we used **pytesseract** as OCR:

```
# Find scale
min_rect_x = width
min_rect_y = height
max_rect_x = 0
max_rect_y = 0

# Find label position
for x in range(1, width-1):
    for y in range(1, height-1):
        col = pix[x, y]
        if col == (0, 255, 0, 255):
            if x < min_rect_x:
                min_rect_x = x
            if y < min_rect_y:
                min_rect_y = y
            if x > max_rect_x:
                max_rect_x = x
            if y > max_rect_y:
                max_rect_y = y

            pix[x, y] = (255, 255, 255, 255)

# OCR it
scale = im.crop((min_rect_x-5, min_rect_y-5, max_rect_x+5, max_rect_y+5)).convert('1')
scale = int(pytesseract.image_to_string(scale)[-1])
```

Now we cut the image to the internal rectangle position:

```
# Find rectangle
# Different colors: (0, 0, 0, 0) (0, 0, 0, 255) (0, 255, 0, 255)
min_rect_x = width
min_rect_y = height
max_rect_x = 0
max_rect_y = 0

for x in range(1, width-1):
    for y in range(1, height-1):
        col = pix[x, y]
        if col == (0, 0, 0, 255):
            if x < min_rect_x:
                min_rect_x = x
            if y < min_rect_y:
                min_rect_y = y
            if x > max_rect_x:
                max_rect_x = x
            if y > max_rect_y:
                max_rect_y = y
            pix[x, y] = (255, 255, 255, 255)

internal_original = im.crop((min_rect_x, min_rect_y, max_rect_x+1, max_rect_y+1))
pix_debug = internal_original.load()

internal = internal_original.convert('1')
pix = internal.load()
points = [[], [], []]
iwidth, iheight = internal.size
```

Now we have to find each symbol and it's center point, there is the code for **x**:

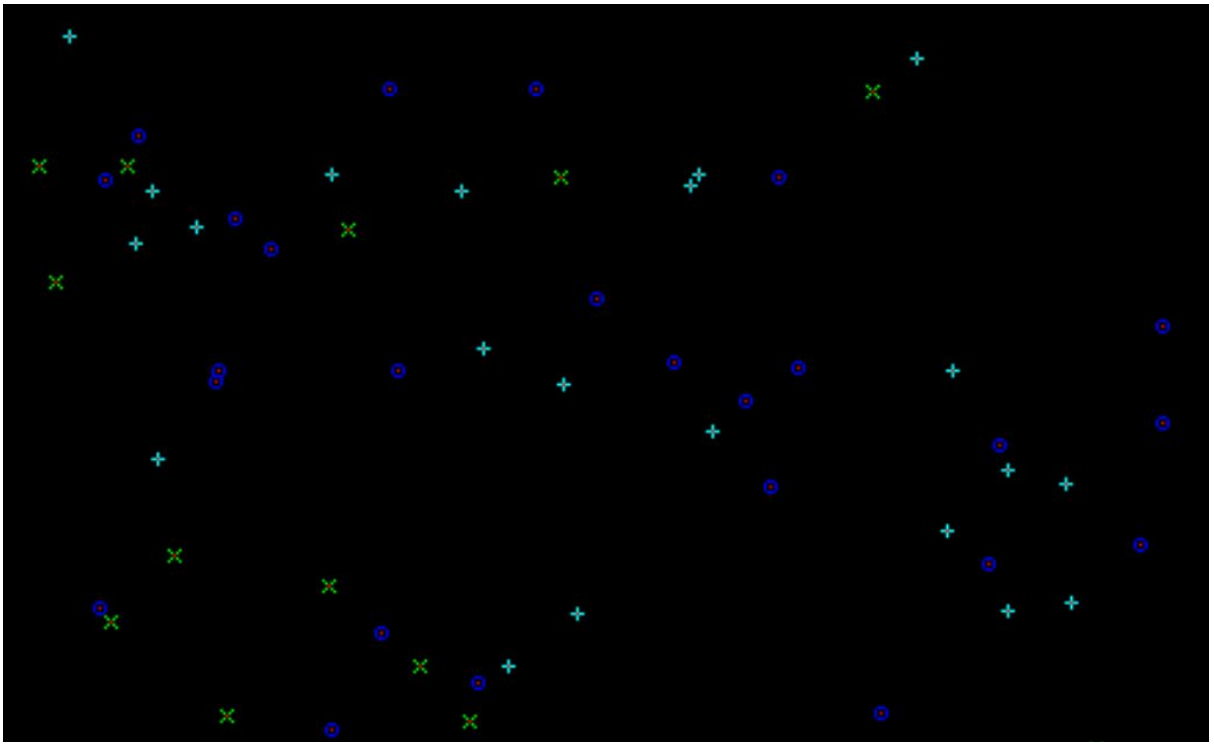
```
# Trova x
points[0] = []
for x in range(0, iwidth):
    for y in range(0, iheight):
        if pix[x, y] == 255:
            p = [(x-2, y-2), (x-1, y-1), (x-2, y+2), (x-1, y+1), (x+2, y+2), (x+1, y+1), (x+2, y-2), (x+1, y-1)]
            all_white = True
            debug_color = [] # debug
            for a in p:
                rx, ry = a
                if rx < 0 or ry < 0 or rx >= iwidth or ry >= iheight:
                    all_white = False
                    break

                if pix[rx, ry] != 255:
                    all_white = False
                    break
                debug_color.append((rx, ry)) # debug

            if all_white:
                # debug
                for deb in debug_color:
                    rx, ry = deb
                    pix_debug[rx, ry] = (0, 255, 0, 255)
                pix_debug[x, y] = (255, 0, 0, 255)
                # end debug

                points[0].append((x, y))
```

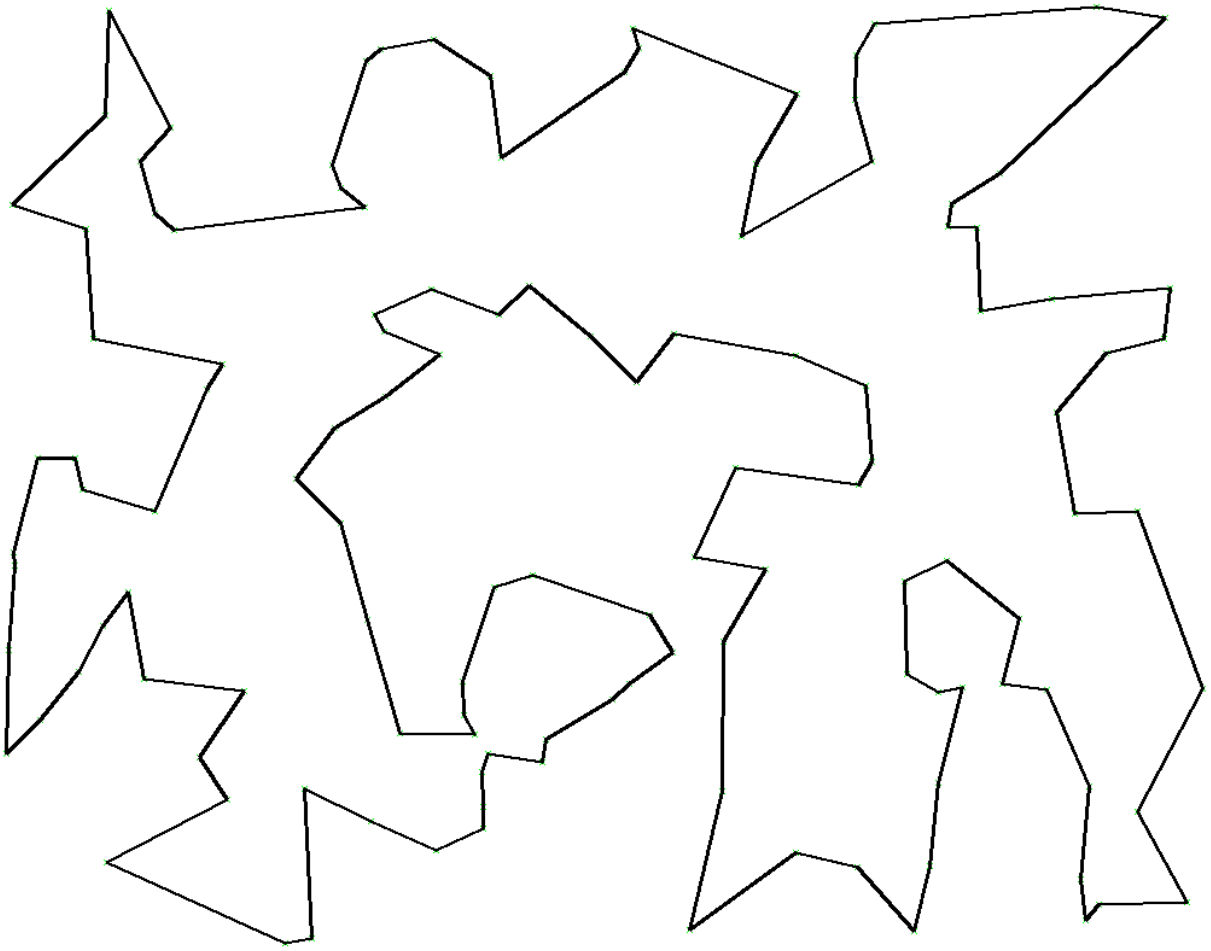
We wrote a similar code for the other symbols, and there is a portion of the debug image:



Once we have written all the utilities to parse the input images we have to solve the TSP problem.

For this task we have used a python wrapper of the Concorde TSP Solver (https://en.wikipedia.org/wiki/Concorde_TSP_Solver) which is considered the fastest TSP existing.

This is an example of TSP for an input:



We run the final script to solve all the 512 TSP and after some minutes we manage to get the flag in the last page:

{FLG:F!n4LlyW3c4Nh4v35oM3Fr3sH41R!}

Final code for the brave:

```
import requests
import pytesseract
import math
from PIL import Image, ImageDraw
import sys
import time

from concorde.tsp import TSPSolver
from concorde.tests.data_utils import get_dataset_path

url = 'http://gamebox2.reply.it:1337/optischema/play'
image_url = 'http://gamebox2.reply.it:1337/optischema/schema.png'
s = requests.Session()

s.get(url)

def get_schema(step):
    r = s.get(image_url)
    with open('img/schema_' + str(step) + '.png', 'wb') as schema:
        schema.write(r.content)

    im = Image.open('img/schema_' + str(step) + '.png')
    pix = im.load()
    width, height = im.size

    # Find scala
    min_rect_x = width
    min_rect_y = height
    max_rect_x = 0
    max_rect_y = 0

    for x in range(1, width-1):
        for y in range(1, height-1):
            col = pix[x, y]
            if col == (0, 255, 0, 255):
                if x < min_rect_x:
                    min_rect_x = x
                if y < min_rect_y:
                    min_rect_y = y
                if x > max_rect_x:
                    max_rect_x = x
                if y > max_rect_y:
                    max_rect_y = y
```

```
                pix[x, y] = (255, 255, 255, 255)

    scale = im.crop((min_rect_x-5, min_rect_y-5, max_rect_x+5,
max_rect_y+5)).convert('1')
    scale = int(pytesseract.image_to_string(scale)[-1])

    # Find rectangle
    # Different colors: (0, 0, 0, 0) (0, 0, 0, 255) (0, 255, 0, 255)
    min_rect_x = width
    min_rect_y = height
    max_rect_x = 0
    max_rect_y = 0

    for x in range(1, width-1):
        for y in range(1, height-1):
            col = pix[x, y]
            if col == (0, 0, 0, 255):
                if x < min_rect_x:
                    min_rect_x = x
                if y < min_rect_y:
                    min_rect_y = y
                if x > max_rect_x:
                    max_rect_x = x
                if y > max_rect_y:
                    max_rect_y = y
                pix[x, y] = (255, 255, 255, 255)

    internal_original = im.crop((min_rect_x, min_rect_y, max_rect_x+1,
max_rect_y+1))
    pix_debug = internal_original.load()

    internal = internal_original.convert('1')
    pix = internal.load()
    points = [[], [], []]
    iwidth, iheight = internal.size

    for x in range(0, iwidth):
        for y in range(0, iheight):
            pix_debug[x, y] = (0, 0, 0, 255)

    # Find x
    points[0] = []
    for x in range(0, iwidth):
        for y in range(0, iheight):
```

```
            if pix[x, y] == 255:
                p = [(x-2, y-2), (x-1, y-1), (x-2, y+2), (x-1, y+1),
(x+2, y+2), (x+1, y+1), (x+2, y-2), (x+1, y-1)]
                all_white = True
                debug_color = [] # debug
                for a in p:
                    rx, ry = a
                    if rx < 0 or ry < 0 or rx >= iwidth or ry >=
iheight:
                        all_white = False
                        break

                    if pix[rx, ry] != 255:
                        all_white = False
                        break
                    debug_color.append((rx, ry)) # debug

                if all_white:
                    # debug
                    for deb in debug_color:
                        rx, ry = deb
                        pix_debug[rx, ry] = (0, 255, 0, 255)
                    pix_debug[x, y] = (255, 0, 0, 255)
                    # end debug

                    points[0].append((x, y))

    # Find o
    points[1] = []
    for x in range(0, iwidth):
        for y in range(0, iheight):
            if pix[x, y] == 255: # 3 x 3 x 3
                p = [(x+1, y), (x+2, y), (x+3, y+1), (x+3,y+2),
(x+3,y+3), (x-1, y+1), (x-1, y+2), (x-1, y+3), (x, y+4), (x+1, y+4),
(x+2, y+4)]
                all_white = True
                debug_color = [] # debug
                debug_color.append((x, y)) # debug
                for a in p:
                    rx, ry = a
                    if rx < 0 or ry < 0 or rx >= iwidth or ry >=
iheight:
                        all_white = False
                        break
```

```
                    if pix[rx, ry] != 255:
                        all_white = False
                        break
                    debug_color.append((rx, ry)) # debug

                if all_white:
                    # debug
                    for deb in debug_color:
                        rx, ry = deb
                        pix_debug[rx, ry] = (0, 0, 255, 255)

                    tx = x + 1
                    ty = y + 2
                    pix_debug[tx, ty] = (255, 0, 0, 255)
                    # end debug

                    points[1].append((tx, ty))

    # Find +
    points[2] = []
    for x in range(0, iwidth):
        for y in range(0, iheight):
            if pix[x, y] == 255: # 3 x 3 x 3
                p = [(x, y+1), (x, y+2), (x, y-1), (x, y-2), (x-2, y),
(x-1, y), (x+1, y), (x+2, y)]
                all_white = True
                debug_color = [] # debug
                debug_color.append((x, y)) # debug
                for a in p:
                    rx, ry = a
                    if rx < 0 or ry < 0 or rx >= iwidth or ry >=
iheight:
                        all_white = False
                        break

                    if pix[rx, ry] != 255:
                        all_white = False
                        break
                    debug_color.append((rx, ry)) # debug

                if all_white:
                    # debug
                    for deb in debug_color:
                        rx, ry = deb
                        pix_debug[rx, ry] = (0, 255, 255, 255)
```

```
                    pix_debug[x, y] = (255, 0, 0, 255)
                    # end debug

                    points[2].append((x, y))

    internal_original.save('img/crop_debug_' + str(step) + '.png') #
debug
    return points, scale

def send_answer(ans):
    r = s.post(url, data={'schemaCode': ans})
    return r.text

for t in range(512):

  print("TURN", t)

  inizio = time.time()
  points_t, scala = get_schema(t)
  soluzione = 0

  for points in points_t:
    if len(points) == 0:
      continue

    tsp_file = open("to_solve.tsp", "w")

    tsp_file.writelines(["NAME: to_solve", "\n"])
    tsp_file.writelines(["TYPE: TSP", "\n"])
    tsp_file.writelines(["COMMENT: to_solve", "\n"])
    tsp_file.writelines(["DIMENSION: "+str(len(points)), "\n"])
    tsp_file.writelines(["EDGE_WEIGHT_TYPE: EUC_2D", "\n"])
    tsp_file.writelines(["NODE_COORD_SECTION", "\n"])
    for i in range(len(points)):
        tsp_file.writelines([str(i+1) + " " + str(float(points[i][0])) +
" " + str(float(points[i][1])), "\n"])
    tsp_file.writelines(["EOF", "\n"])

    tsp_file.close()

    solver = TSPSolver.from_tspfile("to_solve.tsp")
    solution = solver.solve()

    im = Image.open("img/crop_debug_"+str(t)+".png")
```

```
    draw = ImageDraw.Draw(im)
    draw.line((100,200, 150,300), fill=128)

    check = 0

    def d(a, b):
        r = (a[0]-b[0])*(a[0]-b[0]) + (a[1]-b[1])*(a[1]-b[1])
        return math.sqrt(r)

    for i in range(len(solution.tour)):
        p0 = points[solution.tour[i]]
        p1 = points[solution.tour[(i+1)%len(points)]]
        check = check + round(d(p0, p1))
        draw.line((p0[0], p0[1], p1[0], p1[1]), fill=(0,0,0), width=3)

    im.save('img/crop_debug_tour_' + str(t) + '.png')
    soluzione = soluzione + round(solution.optimal_value*scala)

fine = time.time()
res = send_answer(str(soluzione))

print(res)
print(fine-inizio)

if "Step 0 of 512" in res:
    break
```