

SoulMate Roulette

Tutorial · CC3036 · 2021

Diogo Peralta Cordeiro

From images to a ranking

The task has three distinct stages: interpret the binary images, represent each eye as a set of hull vertices, and rank every pair using the specified distance. There is no need to find a perfect matching. A citizen can appear in many output pairs because the authorities make the final assignments elsewhere.

Keep the statement beside this tutorial. Its image format, normalisation and tie-breaking rules define the result; a plausible geometric alternative may produce different answers.

Read and validate the complete card

There are $N = 1337$ images, each occupying exactly 398 bytes. Thus the input must contain $1337 \times 398 = 532126$ bytes. Read at most one byte beyond this amount so that extra data is detected. A single call to an input stream's ordinary read method need not fill its destination buffer. The Java solution uses `readNBytes` and checks the returned length.

Image i occupies the half-open byte interval $[398i, 398(i + 1))$. Interpret its header fields as little-endian values. Check the signature, file size, 40-byte DIB header, dimensions, colour planes, one-bit depth, absence of compression, pixel offset and black/white palette before accepting it.

The 42 bits of useful pixels in a row occupy six bytes; BMP pads the row to eight bytes. With a 62-byte header and palette, the total is $62 + 42 \times 8 = 398$ bytes. The positive height indicates bottom-up storage. ImageIO handles the pixel decoding in the supplied Java program after the header has been checked.

Do not print pair scores while reading. If the final image is invalid, the required output is still only the failure message. Validate the entire set first.

Construct one convex hull per image

Collect the coordinates of all white pixels. Fewer than three points, or a set in which all points are collinear, makes the whole input invalid.

A convex hull is the smallest convex polygon containing the samples. The supplied code uses Graham scan:

1. Choose the point with the smallest vertical coordinate, breaking ties by horizontal coordinate.
2. Sort the other points by angle around that point, placing nearer points first when the angle is equal.
3. Maintain a stack of candidate vertices. A clockwise turn removes the middle candidate; a collinear turn keeps the extreme endpoint.

The orientation test is the sign of the cross product:

$$(b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x).$$

Positive, negative and zero values distinguish counterclockwise, clockwise and collinear triples. Intermediate points on straight hull edges are excluded. The implementation repeats the first vertex at the end to close the hull; repeating that point does not affect any minimum or maximum distance.

The hull stage is a representation choice required by this exercise. Hausdorff distance itself can also be defined for non-convex point sets.

Evaluate the discrete distance

For each vertex in the first hull, find the closest vertex in the second. The largest of those nearest distances is the directed distance. Repeat in the other direction and take the larger result.

```
function directedSquared(A, B):
    largest = 0
    for each point a in A:
        nearest = infinity
        for each point b in B:
            nearest = min(nearest, squaredDistance(a, b))
        largest = max(largest, nearest)
    return largest

distance = sqrt(max(directedSquared(A, B),
                    directedSquared(B, A)))
```

Squared distances let the inner loop use multiplication and addition. Because square root is increasing on non-negative numbers, taking it after the minimum and maximum comparisons gives the same result.

As a small example, consider the triangles $A = \{(0, 0), (2, 0), (0, 2)\}$ and $B = \{(1, 0), (3, 0), (1, 2)\}$. Every vertex has a nearest vertex at distance 1 in the other triangle, so both directed distances are 1. The score is approximately 98.3164%, printed as 98.32%. Identical vertex sets have distance 0 and score 100.00%.

Do not substitute point-to-edge distance. The statement uses distances between hull vertices, and a nearest position in the middle of an edge is not a candidate.

Produce a deterministic order

Generate each pair once, with $a < b$. There are $M = N(N - 1)/2 = 893116$ pairs. Store the identifiers and the unrounded score:

$$100 \left(1 - \frac{H(A, B)}{\sqrt{42^2 + 42^2}} \right).$$

Sort by descending score, then ascending first identifier, then ascending second identifier. The identifiers are integers: lexical ordering would place 10 before 2. A stable sort alone is insufficient if the input collection has no defined order, as with a hash map.

Round only when formatting the output. Two scores that both display as 98.32% can still have a strict order. Use `Locale.R00T` with `%.2f` so that a machine configured for Portuguese does not print a decimal comma. Buffered output matters for nearly nine hundred thousand lines.

Complexity and practical limits

Let P be the maximum number of white pixels per image and H the maximum number of hull vertices. Reading costs $O(N \times 42^2)$; hull construction costs $O(NP \log P)$. The direct distance calculation costs $O(H^2)$ per pair, and sorting costs $O(M \log M)$.

The total is therefore $O(N \times 42^2 + NP \log P + N^2 H^2 + M \log M)$, with $O(NH + M)$ storage after image decoding, in addition to the input buffer. Use these quantities separately: the distance loop is quadratic in the two hull sizes, not linear merely because the polygons are convex. Java object and string overhead also contributes to the memory budget.

Build and check

Use JDK 17 or newer:

```
javac Solution.java InputGenerator.java
tar -xzf example_io.tar.gz
java -Xmx550m Solution < example_input.raw > actual.txt
cmp example_output.txt actual.txt
python3 test.py
```

The test compares the complete sample output and checks truncated input, extra bytes, a wrong dimension and an empty shape. Additional useful cases include identical hulls, collinear samples, different sets with equal scores and identifiers with different digit counts.

Generate another card with:

```
java InputGenerator bmps
sh from_bmps_to_raw.sh bmps generated.raw
java -Xmx550m Solution < generated.raw > generated-output.txt
```

The generator uses seed 42. The concatenation script orders BMPs by numeric identifier and checks all 1337 files before writing the card. Algorithm attribution remains in `Solution.java`.