Problem set 42 - Intersection over Union (IoU) for object detection

For two given sets of boxes, your implementation should return IoUs for each pair of boxes (one box from each set).

Problem 42.1 - Optimize the worst case performance

The functionality should be implemented using a function with the following signature:

```
def iou(boxes_1: np.ndarray, boxes_2: np.ndarray) -> np.ndarray:
    ...
```

Expect the boxes to be of shape `(n, 4)` where `n` stands for the number of boxes in the given array, and `4` stands for `(y1, x1, y2, x2)` (`(y1, x1)` forms a topmost corner of the box, and `(x2, y2)` forms bottommost corner).

You can safely assume well defined boxes (`y1 < y2` and `x1 < x2`).

Return a matrix of shape `(n, m)` where `n` and `m` stand for the number of boxes in `boxes_1` and `boxes_2` respectively. Each element of the returned matrix (`(i,j)`) should represent an IoU value for one of the pairs of boxes (`boxes_1[i]` and `boxes_2[j]` where `0 <= i < n` and `0 <= j < m`).

Both inputs and outputs are of type `float32` .

Given `n,m<2000` your implementation should perform in under a second in Google Colab in a worst case scenario

Problem 42.2 - Matrix sparsity

Output matrix defined in 42.1 can be pretty sparse for some subspace of inputs.

Variables `H` and `W` are introduced such that `0 < y1 < y2 < H` and `0 < x1 < x2 < W` holds for each box.

Assume both sets of boxes come from the same distribution.

Using `(x1,y1,h,w)` notation of the box, where `h := y2 - y1` and `w := x2 - x1` , assume `h` and `w` follow log-normal distribution.

Assume `x1` and `y1` follow uniform distribution along their domain.

Express sparsity (or density) of the output matrix in terms of `n`, `m` and the expectation of the terms `h / H` and `w / W`.

Comment the case where `h << H` and `w << W`.

Problem 42.3 - Optimize the average case performance

Assuming that expectation of the terms `h / H` and `w / W` is less than `1e-2`, with the maximal `n` and `m` from the Problem 42.1, reimplement the algorithm to optimize the average case performance. Feel free to use other output format as well.

General requirements

- use `>=Python3.6`
- only `numpy` can be installed as a dep
- type your code where suitable (`typing` package)
- implement a solution as a module (a single file)
- for coding problems send us the solution using gist or similar public snippet sharing service
- explicit control flow and loop constructs (`for`, `while`, `if`, ...) should be avoided
- code has to be both clean and performant
- for theoretic problems write your solution using LaTeX and send us the PDF