Tiles

Part of the following description of the algorithm is taken from [1]. The algorithm uses a vertical sweep line ℓ that sweeps over the polygon P from left to right. The intuition is that the algorithm keeps track of how the tiling looks in the region of P to the left of ℓ if a tiling exists. As ℓ sweeps over P, we keep track of how the tiling pattern changes under ℓ. Each vertical edge of P that ℓ sweeps over causes changes to the tiling, and we must update our data structures accordingly. Note that if P is tileable, then the tiling is unique.

Consider the situation where the sweep line is some vertical line ℓ with integer x-coordinate $x(\ell)$. The algorithm stores a set $\mathcal{I}$ of pairwise interior-disjoint closed intervals $\mathcal{I} = I_1, \ldots, I_m \subset \mathbb{R}$, ordered upwards from below. Each interval I_i has endpoints at integers and represents the segment $I_i' := \{x(\ell)\} \times I_i$ on ℓ. In the simple case that no vertical edge of P has x-coordinate $x(\ell)$ (so that no change to the set $P \cap \ell$ happens at this point), the intervals $\mathcal{I}$ together represent the part of ℓ in P, i.e., we have $P \cap \ell = \bigcup_{i \in [m]} I_i'$. If one or more vertical edges of P have x-coordinate $x(\ell)$, then $P \cap \ell$ changes at this point and the intervals $\mathcal{I}$ must be updated accordingly.

For each interval I_i we store a *parity* $p(I_i) \in \{0, 1\}$, which encodes how the tiling must be under ℓ. If the parity of an interval is even, it means that the vertical edges of the tiles have even x-coordinates, and otherwise they have odd x-coordinates.

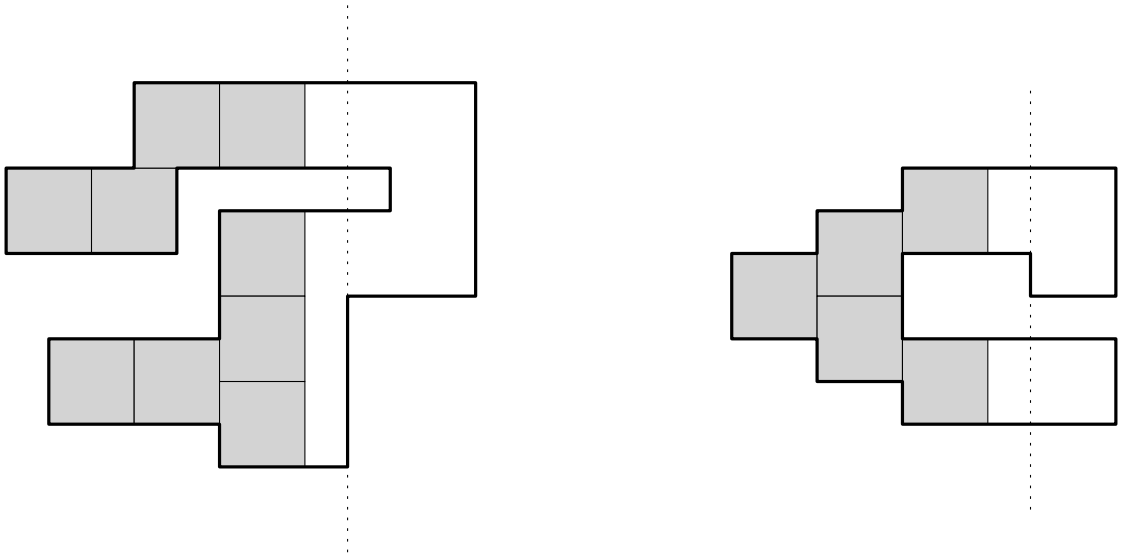

Figure 1: Two polygons that cannot be completely tiled. Left: When the sweep line reaches this point, the algorithm returns at line 13, since the parity of the edge contained in the sweep line does not agree with the parity of the two intervals in $\mathcal{I}$. Right: When the sweep line reaches this point, the algorithm returns at line 20, since an interval in $\mathcal{I}$ has odd length.

We maintain a variable z which is maximum such that $z \leq x(\ell)$ and such that all of P to the left of the vertical line through $(z, 0)$ can be tiled. Whenever the sweep line ℓ has jumped to a new position $x(\ell)$, we update z if all intervals in $\mathcal{I}$ have the same parity. If they all have the same parity as $x(\ell)$, then we update $z := x(\ell)$, otherwise we set $z := x(\ell) - 1$. When the sweep line reaches a point where we cannot proceed with the tiling, or we conclude that all of P can be tiled, we return z.

We maintain that any pair of neighboring intervals of $\mathcal{I}$ that share an endpoint have different

parity. Therefore, if we get such neighbours of the same parity, we merge them into one interval.

The pseudocode of the algorithm is shown in Algorithm 1. Initially, we sort all vertical edges after their x-coordinates (breaking ties arbitrarily). We then run through the edges in this order. Each edge makes a change to the set $P \cap \ell$, and we need to update the intervals $\mathcal{I}$ accordingly. Figure 1 shows examples where the algorithm returns at lines 13 and 20, respectively.

Consider the event that the sweep line ℓ reaches a vertical edge $e_j = \{x\} \times [y_0, y_1]$. If the interior of P is to the left of e_j, then $P \cap \ell$ shrinks. Each interval $I_i \in \mathcal{I}$ that overlaps $[y_0, y_1]$ must then also shrink, be split into two, or disappear from $\mathcal{I}$. This is handled by the for-loop at line 11. If the parity of one of these intervals I_i does not agree with the parity of e_j, we cannot proceed with the tiling, and hence the algorithm returns z at line 13, since this was the last x coordinate where everything to the left of the sweep line could be tiled.

If on the other hand the interior of P is to the right of e_j, then $P \cap \ell$ expands and a new interval I must be added to $\mathcal{I}$. This is handled by the else-part at line 15. The new interval I may have one or two neighbors in $\mathcal{I}$ with shared endpoints. If one or two such neighbors also have the same parity as I, we merge these intervals into one interval of $\mathcal{I}$.

In line 19, we consider the case that we finished handling all vertical edges at some specific x-coordinate so that the sweep line will move to the right in order to handle the next edge e_{j+1} in the next iteration. If there is an interval I_i of odd length in $\mathcal{I}$, it follows that P cannot be tiled, so the algorithm returns z at line 20. If the entire polygon can be tiled, the algorithm returns z at line 21.

The correctness of the algorithm can be formally proven by the introduction of some loop invariants; see [1] for the details.

Runtime analysis. We sort the vertical edges by their x-coordinates in $O(n \log n)$ time. Since the intervals of $\mathcal{I}$ are pairwise interior-disjoint, we can implement $\mathcal{I}$ as a balanced binary search tree, where each leaf stores an interval I_i.

We keep track of the number of intervals of even and odd parity in $\mathcal{I}$, so that we can evaluate the conditions at lines 5 and 7 in constant time.

We now argue that each vertical edge e_j, with y-coordinates $[y_0, y_1]$, takes only $O(\log n)$ time to handle, since we need to make only $O(1)$ updates to $\mathcal{I}$. If the interior of P is to the left of e_j, then $[y_0, y_1] \subset \bigcup_{i \in [m]} I_i$. It then follows that if $[y_0, y_1]$ overlaps more than one interval I_i, then the algorithm will return at line 13 after considering at most two intervals from $\mathcal{I}$. We therefore do at most $O(1)$ updates to $\mathcal{I}$, so it takes $O(\log n)$ time to handle e_j.

On the other hand, if the interior of P is to the right of e_j, we need to insert a new interval into $\mathcal{I}$ and possibly merge it with one or two neighbors in $\mathcal{I}$, so this also amounts to $O(1)$ changes to $\mathcal{I}$.

At line 19, we need to check the $O(1)$ intervals that were added or changed due to the edge e_j, so this can be done in $O(1)$ time. Hence, the algorithm has runtime $O(n \log n)$.

Algorithm 1:

```
Let e_1, ..., e_k be the vertical edges of P in sorted order.
z := x(e_1)
for j = 1, ..., k do
    if j > 1 and x(e_{j-1}) < x(e_j)
        if all intervals in I_i have the same parity as x(e_j)
            z := x(e_j)
        else if all intervals in I_i have the same parity as x(e_j) - 1
            z := x(e_j) - 1
    Let [y_0, y_1] be the interval of y-coordinates of e_j.
    if the interior of P is to the left of e_j
        for each I_i ∈ 𝓘 that overlaps [y_0, y_1] do
            if I_i and x(e_j) have different parity
                return z
            Remove I_i from 𝓘, let J := I_i \ [y_0, y_1], and if J ≠ ∅, add the interval(s) in J to 𝓘.
    else
        Make a new interval I := [y_0, y_1] with the parity p(I) := x(e_j) mod 2 and add I to 𝓘.
        if I shares an endpoint with one or two intervals in 𝓘 that also have the same parity as I
            Merge those intervals in 𝓘.
    if j < k and x(e_{j+1}) > x(e_j) and some I_i ∈ 𝓘 has odd length
        return z
return z
```

Credits

- Task: Mikkel Abrahamsen (Denmark)
- Solutions and tests: Mikkel Abrahamsen (Denmark), Daumilas Ardickas, Dovydas Vadišius (Lithuania)

References

[1] Anders Aamand, Mikkel Abrahamsen, Thomas D. Ahle, and Peter M. R. Rasmussen. *Tiling with Squares and Packing Dominos in Polynomial Time*. 2021. Preprint, `https://arxiv.org/abs/2011.10983`.