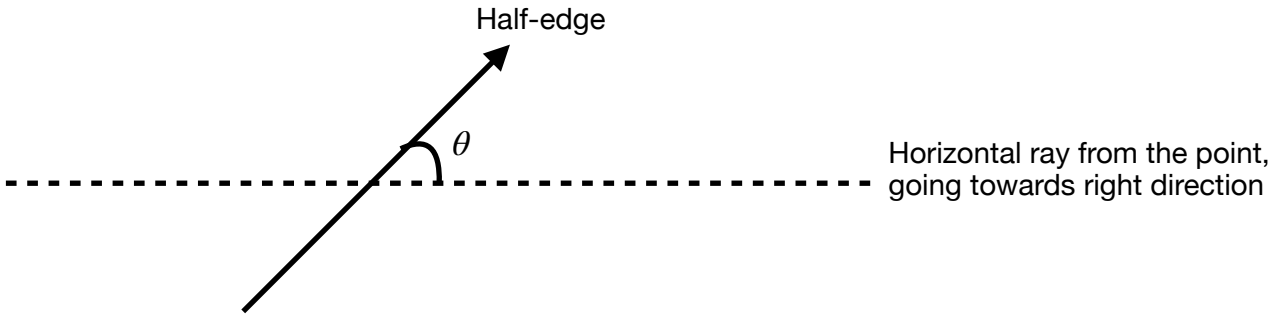
Assumptions and Conventions:

1. The halfedges are printed in **cyclic order for each face**. The faces are in order of creation(oldest first).
2. While splitting, the new face and the older face(that was split) are both assigned the split edge as their incident halfedges.
3. This results in the cyclic order starting from the splitedge of the last split task which involved that face.
4. The programme reads split tasks from the variable split_file. The faces are printed in **order of creation (oldest first)**.
5. **"input.txt"**, **"output.txt"** and **"split.txt"** have been included in the zip file for convenience. Though the names of all 3 files have to be passed as arguments.
6. This programme works for any **n-sided closed polygon**.
7. Output after every split is printed in the output file.
8. The faces of different points are printed at the end.
9. While reading the split_file, the programme identifies a split_task through **"Split"** otherwise it is taken as an **ID_task**.
   Split x y
   ID:x y z
10. The indices to vertices, edges and half-edges are automatically assigned using **static counters**.
11. Instructions to run:
    i. make assignment3
    ii. ./assignment3 input_file split_file output_file

Algorithms:

The algorithm used to find bounding faces is described below:



We will be drawing a horizontal ray pointed towards right and sourced at the given point. This is the only ray that is being referred to in the whole programme.

Classify the half-edges into two types.

Type-1: $-\pi < \theta < 0$

Type-2: $0 < \theta < \pi$

Both types of halfedges should also intersect the horizontal ray pointed towards right and sourced at the given point. Otherwise they will be ignored.

We need not worry about $\theta = 0$ or $\theta = \pi$ as will be clear.

Claim: *The incident face of the first Type-1 half-edge that the ray intersects is the bounding face of the point.*

Proof:

This is an accurate assumption as any ray which starts from within a face will intersect a half-edge of that face before any other. This follows from the fact that the faces are closed. The intersecting half-edge will be of Type-1 if the ray is horizontal and pointed to the right because the half-edges are cyclic in a face by convention.

A borderline case would be when the ray passes through the **common vertex** of two half-edges. In this case I consider only the half-edge which has the vertex as a starting point(**lies below** the ray). This works because we can imagine the ray to be slightly tilted downwards and thus intersecting the half-edge but not at the vertex.

Thus, in my algorithm, we find out all the Type-1 half-edges and the distance of their intersection points from the source point. The nearest intersection point will give the closest half-edge which further gives the bounding face. The sorting has been done using a maximum priority queue with negative data(making it a minimum priority queue). At the end, the top of the priority queue gives us the desired face.

We need not worry about $\theta = 0$ or $\theta = \pi$ because in these cases there will always be a type-1 half-edge among the next half-edges of this half-edge. This follows from the fact that the face has to closed.

Thus, we are able to avoid checking all the faces of the DCEL which could have been time consuming with more faces involved.

Exceptions:

The only exceptions to this claim are when there is a face inside a face, or the point has no face. Both these cases won't be in the test-cases as discussed with the TAs. Still, I have implemented a **ray-casting** algorithm which **conducts a final check** on the face that is given by the priority queue. If the face fails the check then the next closest face is checked and so on until a face passes. This takes care of both the exceptions.

Ray Casting

The ray-casting algorithm states that any ray from a point will intersect an external face even number of times and the bounding face odd number of times. This can be simplified in our case to state that if the face is a bounded face then the number of type-1 half-edges will be more than number of type-2 half-edges unlike an external face where they will be in equal amounts. Using this the probable bounding face that is given by the priority queue is checked.