Convex Hull (Scheme+Haskell - 7+7 Points)

June 18, 2021

The convex hull of a set of points is the smallest convex shape that contains the whole set. Fig. 1A shows a set of six points - five of which constitute the *generating points* of their convex hull. In this task you have to find smallest set of generating points of the convex hull of a given set of points.

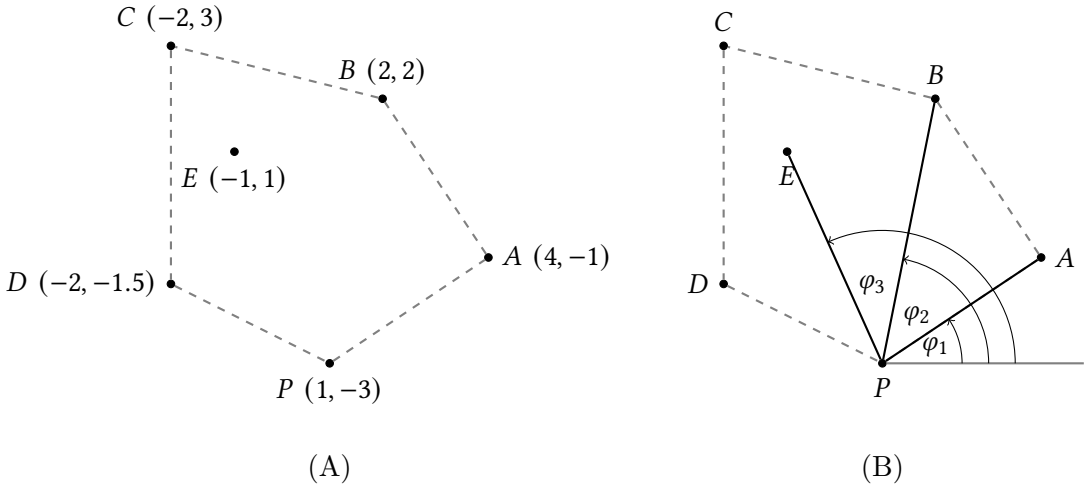

Figure 1: (A) Convex hull of a set of points depicted by the dashed line. The generating points in counter-clockwise order are: P, A, B, C, D. (B) Three of the polar angles φ_i that have to be computed to sort the points.

Algorithm

The imperative pseudocode for finding the generating points is shown below.

```
points = [list of points]
stack  = empty_stack()

p_0 = initial_point(points) # find lowest y-coordinate and rightmost point
sorted_points = sort_by_angle(p_0, points) # sort by polar angle with p0

for p_i in sorted_points:
  # pop last point from the stack if we turn right to reach this point
  while length(stack) > 1
    and (not isLeftTurn(next_to_top(stack), top(stack), p_i)):
      pop stack
  push p_i to stack
end
```

1. In the initial phase of the algorithm you have to find the point with smallest y-coordinate and largest x-coordinate, further called p_0.

2. Now, sort all points based on the *polar angle* they form with p_0. The polar angle is the angle that a vector forms with the x-axis (as shown in Fig. 1B). Note that the polar angle of p_0 with itself is not well defined, so you can just add p_0 as the first element in your sorted list. Further, you can assume that there are no points with the same polar angle in your input set of points.

3. Initialize an empty stack $\mathcal{G}$. While your stack has less than three elements, just push points from your sorted list of points. For each remaining point p_i in the sorted list, check if the two top points from the stack (p_{i-2}, p_{i-1}) and p_i form a *left turn*. If it is a left turn, add p_i to $\mathcal{G}$ and continue to the next point. If it is a right turn, p_{i-1} is *not* part of the hull; remove p_{i-1} from $\mathcal{G}$, and repeat the check for a left turn with the current point p_i and the updated $\mathcal{G}$.

Exemplary iterations of the algorithm are shown in Fig. 2. You can assume that the set of points does not contain the same point multiple times.

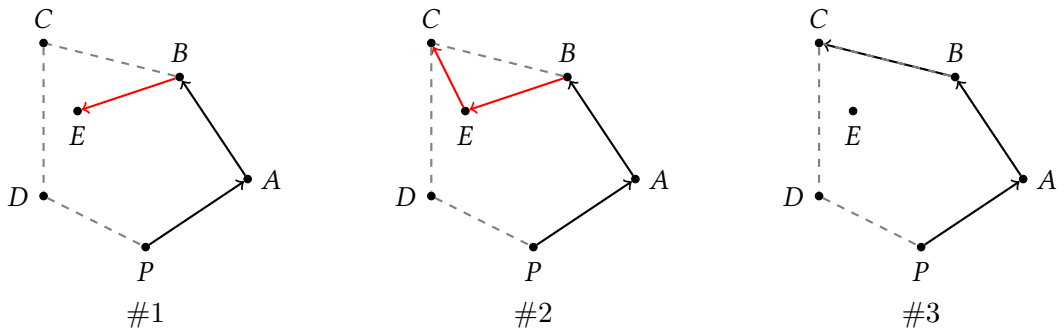

Figure 2: Before #1, the stack of points is $\mathcal{G} = [P, A, B] = [p_{i-3}, p_{i-2}, p_{i-1}]$. In #1 we see that $\overrightarrow{ABE}$ is a left turn so we add $p_i = E$ to $\mathcal{G}$. Now $\mathcal{G} = [P, A, B, E]$. In #2 the algorithm checks whether $\overrightarrow{BEC}$ is a left turn. It is not, so we remove E (corresponding to p_{i-1}) and check again if $\overrightarrow{ABC}$ is a left turn. This is the case, so C is added to $\mathcal{G}$ in #3.

Hint #1 (Left Turn): To check whether three points (x_1, y_1), (x_2, y_2), and (x_3, y_3) represent a left turn you can compute the cross product

$$c = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1). \tag{1}$$

If $c > 0$ then the three points constitute a left turn, otherwise a right turn.

1 Task 3 - Scheme

In Scheme, implement a function `convex-hull` that accepts a list of points and returns the generating points of the convex hull as described above. **The list returned by `convex-hull` must start at the initial point p_0 and contain the generating points in counter-clockwise order.**

```
(define points '((-2 3) (2 2) (-1 1) (-2 -1.5) (4 -1) (1 -3)))

> (convex-hull points)
'((1 -3) (4 -1) (2 2) (-2 3) (-2 -1.5))
```

Your file has to be called `convexhull.rkt` and must provide the function `convex-hull` so it should start like this:

```
#lang racket
(provide convex-hull)

; your code goes here
```

Hint #2 (Polar Angle) In Scheme, you can compute the polar angle of a vector $\vec{v} = (x, y)$ with

```
(atan y x)  ; note the argument order!
```

Computing the polar angle like above will result in $\varphi \in (-\pi, \pi)$. You may want to shift the output by adding 2π to negative angles in order to obtain $\varphi \in (0, 2\pi)$.

Hint #3 (Sorting) To implement custom sorting you can use the function `sort` which takes a list and a comparison function. The comparison function has to accept two elements from the list and return `#t` if the first argument is less than the second and otherwise `#f`. Below is an example that sorts a list of points by the x-coordinate.

```
(define (xcomp a b)
  (if (< (car a) (car b)) #t #f))

> (sort '((2 5) (1 1) (3 0)) xcomp)
'((1 1) (2 5) (3 0))
```

2 Task 4 - Haskell

In Haskell, implement a function `convexHull :: RealFloat a => [(a,a)] -> [(a,a)]` that accepts a list of points and returns the generating points of the convex hull as described above. **The list returned by `convexHull` must start at the initial point p_0 and contain the generating points in counter-clockwise order.**

```
points = [(-2.0, 3.0), ( 2.0, 2.0), (-1.0, 1.0),
          (-2.0,-1.5), ( 4.0,-1.0), ( 1.0,-3.0)]

> convexHull points
[(1.0,-3.0),(4.0,-1.0),(2.0,2.0),(-2.0,3.0),(-2.0,-1.5)]
```

Your file has to be called `ConvexHull.hs` and must export the function `convexHull` so it should start like this:

```
module ConvexHull (convexHull)  where
import Data.List -- for sortBy

-- your code goes here
```

Hint #4 (Polar Angle): In Haskell, you can compute the polar angle of a vector `v = (x,y)` with

```haskell
atan2 y x -- note the argument order!
```

Computing the polar angle like above will result in $\varphi \in (-\pi, \pi)$. You may want to shift the output by adding 2π to negative angles in order to obtain $\varphi \in (0, 2\pi)$.

Hint #5 (Sorting): To implement custom sorting of a list of tuples you can use the function `sortBy :: (a -> a -> Ordering) -> [a] -> [a]` provided by `Data.List`. Below is an example that sorts a list of tuples by the x-coordinate

```haskell
import Data.List

xcomp (x1,y1) (x2,y2)
  | x1 == x2  = EQ
  | x1 < x2   = LT
  | otherwise = GT

> sortBy xcomp [(2,5),(1,1),(3,0)]
[(1,1),(2,5),(3,0)]
```