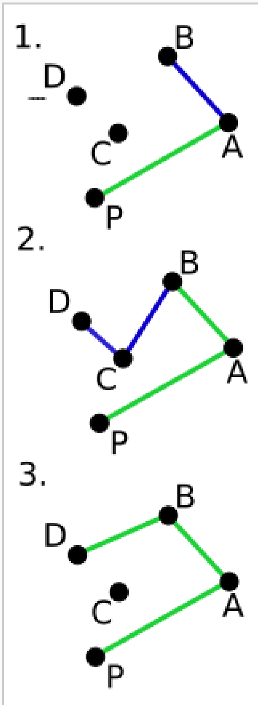
Graham scan

From Wikipedia, the free encyclopedia

Graham's scan is a method of finding the convex hull of a finite set of points in the plane with time complexity O(n log n). It is named after Ronald Graham, who published the original algorithm in 1972.[1] The algorithm finds all vertices of the convex hull ordered along its boundary.

Contents [hide]

Algorithm [edit]



As one can see, PAB and ABC are counterclockwise, but BCD isn't. The algorithm detects this situation and discards previously chosen segments until the turn taken is counterclockwise (ABD in this case.)

The first step in this algorithm is to find the point with the lowest y-coordinate. If the lowest y-coordinate exists in more than one point in the set, the point with the lowest x-coordinate out of the candidates should be chosen. Call this point *P*. This step takes O(n), where n is the number of points in question.

Next, the set of points must be sorted in increasing order of the angle they and the point *P* make with the x-axis. Any general-purpose sorting algorithm is appropriate for this, for example heapsort (which is O(n log n)).

Sorting in order of angle does not require computing the angle. It is possible to use any function of the angle which is monotonic in the interval $[0,\pi)$. The cosine is easily computed using the dot product, or the slope of the line may be used. If numeric precision at a stake, the comparison function used by the sorting algorithm can use the sign of the cross product to determine relative angles.

The algorithm proceeds by considering each of the points in the sorted array in sequence. For each point, it is first determined whether traveling from the two points immediately preceding this point constitutes making a left turn or a right turn. If a right turn, the second-to-last point is not part of the convex hull, and lies 'inside' it. The same determination is then made for the set of the latest point and the two points that immediately precede the point found to have been inside the hull, and is repeated until a "left turn" set is encountered, at which point the algorithm moves on to the next point in the set of points in the sorted array minus any points that were found to be inside the hull; there is no need to consider these points again. (If at any stage the three points are collinear, one may opt either to discard or to report it, since in some applications it is required to find all points on the boundary of the convex hull.)

Again, determining whether three points constitute a "left turn" or a "right turn" does not require computing the actual angle between the two line segments, and can actually be achieved with simple arithmetic only. For three points $P_1=(x_1,y_1)$, $P_2=(x_2,y_2)$ and $P_3=(x_3,y_3)$, simply compute the *z*-coordinate of the cross product of the two vectors $\overrightarrow{P_1P_2}$ and $\overrightarrow{P_1P_3}$, which is given by the expression $(x_2-x_1)(y_3-y_1)-(y_2-y_1)(x_3-x_1)$. If the result is 0, the points are collinear; if it is positive, the three points constitute a "left turn" or counter-clockwise orientation, otherwise a "right turn" or clockwise orientation (for counter-clockwise numbered points).

This process will eventually return to the point at which it started, at which point the algorithm is completed and the stack now contains the points on the convex hull in counterclockwise order.

Time complexity [edit]

Sorting the points has time complexity O(*n* log *n*). While it may seem that the time complexity of the loop is O(n^2), because for each point it goes back to check if any of the previous points make a "right turn", it is actually O(*n*), because each point is considered at most twice in some sense. Each point can appear only once as a point (x_2, y_2) in a "left turn" (because the algorithm advances to the next point (x_3, y_3) after that), and as a point (x_2, y_2) in a "right turn" (because the point (x_2, y_2) is removed). The overall time complexity is therefore O(*n* log *n*), since the time to sort dominates the time to actually compute the convex hull.

Pseudocode [edit]

First, define

```
# Three points are a counter-clockwise turn if ccw > 0, clockwise if
# ccw < 0, and collinear if ccw = 0 because ccw is a determinant that
# gives twice the signed  area of the triangle formed by p1, p2 and p3.
function ccw(p1, p2, p3):
    return (p2.x - p1.x)*(p3.y - p1.y) - (p2.y - p1.y)*(p3.x - p1.x)
```

Then let the result be stored in the array `points`.

```
let N           = number of points
let points[N+1] = the array of points
swap points[1] with the point with the lowest y-coordinate
sort points by polar angle with points[1]

# We want points[0] to be a sentinel point that will stop the loop.
let points[0] = points[N]

# M will denote the number of points on the convex hull.
let M = 1
for i = 2 to N:
    # Find next valid point on convex hull.
    while ccw(points[M-1], points[M], points[i]) <= 0:
          if M > 1:
                  M -= 1
          # All points are collinear
          else if i == N:
                  break
          else
                  i += 1

    # Update M and swap points[i] to the correct place.
    M += 1
    swap points[M] with points[i]
```

This pseudocode is adapted from Sedgewick and Wayne's *Algorithms, 4th edition*.

The check inside the while statement is necessary to avoid the case when all points in the set are collinear.

Notes [edit]

The same basic idea works also if the input is sorted on x-coordinate instead of angle, and the hull is computed in two steps producing the upper and the lower parts of the hull respectively. This modification was devised by A. M. Andrew[2] and is known as Andrew's Monotone Chain Algorithm. It has the same basic properties as Graham's scan.[3]

The stack technique used in Graham's scan is very similar to that for the all nearest smaller values problem, and parallel algorithms for all nearest smaller values may also be used (like Graham's scan) to compute convex hulls of sorted sequences of points efficiently.[4]

References [edit]

1. ^ Graham, R.L. (1972). An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. Information Processing Letters 1, 132-133
2. ^ Andrew, A. M. (1979). "Another efficient algorithm for convex hulls in two dimensions". *Information Processing Letters* **9** (5): 216–219. doi:10.1016/0020-0190(79)90072-3.
3. ^ De Berg, Mark; Cheong, Otfried; Van Kreveld, Marc; Overmars (2008). *Computational Geometry Algorithms and*

Applications. Berlin: Springer. pp. 2–14. doi:10.1007/978-3-540-77974-2. ISBN 978-3-540-77973-5. |first5= missing |last5= in Authors list (help)

4. ^ Berkman, Omer; Schieber, Baruch; Vishkin, Uzi (1993). "Optimal double logarithmic parallel algorithms based on finding all nearest smaller values". *Journal of Algorithms* **14** (3): 344–370. doi:10.1006/jagm.1993.1018..

- Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford (2001) [1990]. "33.3: Finding the convex hull". *Introduction to Algorithms* (2nd ed.). MIT Press and McGraw-Hill. pp. pp. 949–955. ISBN 0-262-03293-7.

Categories: Convex hull algorithms