CS31005 Algorithms – II

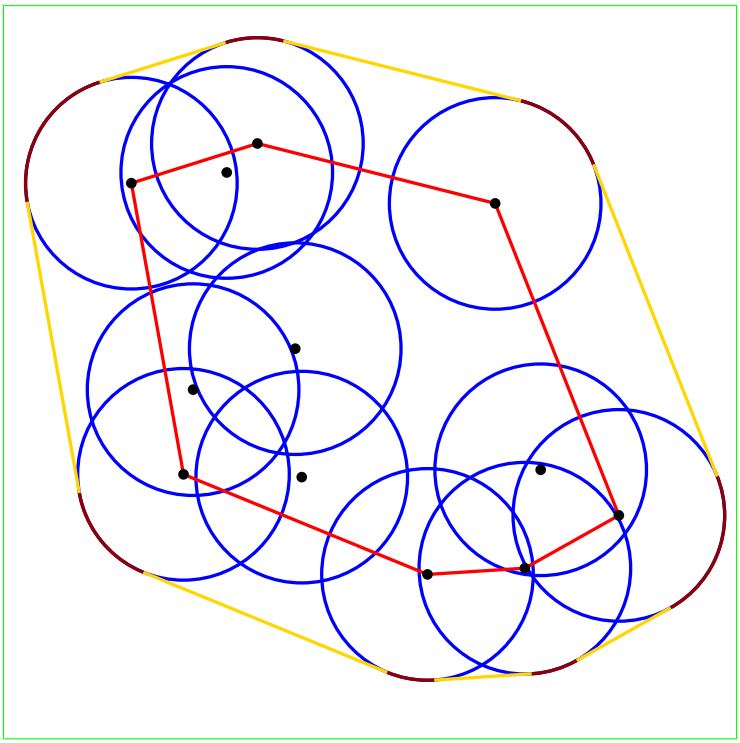
Second Programming Assignment

Last date of submission: 10–November–2020

Convex hull of circles

The mayor of Foonagar identifies n Barona hotspots. Around each hotspot, there is a circle of fixed radius such that the region enclosed by the circle is a high-risk zone. The mayor wants to find the smallest convex region containing all these high-risk zones, and seal this convex region as the containment zone. The mayor is very poor in programming, so your help is solicited to identify the containment zone in Foonagar.

To be precise, let $S = \{P_1, P_2, \ldots, P_n\}$ be the set of points (hotspots) in the two-dimensional plane. Around each P_i, consider a circle C_i (high-risk zone) of a given radius r (the same for all i). Assume that the map of Foonagar is a 1×1 square, and that each C_i is fully contained in the square. Each point P_i is specified by its coordinates x_i, y_i which are floating-point values in the range $[r, 1-r]$, where $r \in (0,1)$ is again a floating-point value. The following figure demonstrates that the boundary of the containment zone consists of circular arcs and line segments. The figure also demonstrates that the circles involved in the boundary of the containment zone can be identified by the convex hull of their centers. With this understanding, solve the following parts. Your overall algorithm should run in $O(n \log n)$ time.



Part 1: The data types

Represent a *point* by a structure of two floating-point (`double`) values x, y. A *circle* is represented by its center (a point) along with its radius. In the current context, all circles have the same radius r, so r can be stored externally, and only the center suffices to represent a circle. A *convex hull* is represented by an array of points (the vertices) listed in clockwise order. We will actually maintain the upper and the lower hulls separately. The *upper hull* is an array of points from the leftmost point to the rightmost point, whereas the *lower hull* is an array of points from the rightmost point to the leftmost point. Assume that the points in S are in general position. Represent a *line segment* by its two endpoints. Finally, an *arc* is represented by the center of the circle along with the beginning and the ending angles in the clockwise direction.

Part 2: Sort the given points

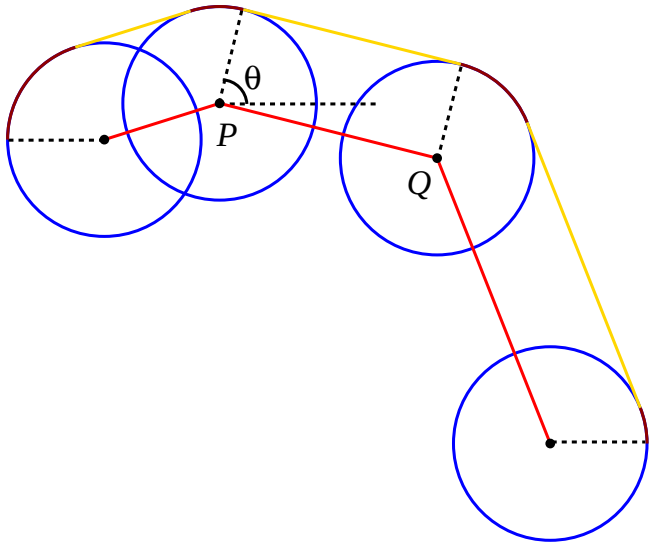
Implement an $O(n \log n)$-time function (like merge sort) to sort the points in S in the increasing values of their x-coordinates.

Part 3: Compute the convex hull of the centers

Write a function `CH(S,n,flag,H)` that takes S (an array storing the centers) and its size n as arguments, and computes the upper or the lower hull of S. Implement Graham's scan. You need to pass an additional flag to indicate which hull you would like to compute. The function should populate the array H of points (the vertices on the upper/lower hull) following the convention mentioned in Part 1. The function should return the size (number of edges) on the hull.

Part 4: Compute the segments and the arcs of the boundary

The following figure illustrates the construction of the upper section of the boundary of the containment zone. This construction uses $\mathrm{UH}(S)$. The lower section uses $\mathrm{LH}(S)$, and can be constructed analogously.



For the leftmost circle, the arc begins at the angle π, whereas for the rightmost circle, the arc ends at the angle 0. The total arc on each of these two circles is broken down into two subarcs, one coming from $\mathrm{UH}(S)$, and the other coming from $\mathrm{LH}(S)$. The subarcs can later be joined together, but you do not have to do that.

In the upper section, keep all the angles non-negative (in the range $[0,\pi]$). This would actually happen by default. In the lower section, keep all the angles non-positive (in the range $[-\pi,0]$). Subtract 2π from positive angles to ensure this.

Write a function `contzone(UH,u,LH,l,r,T,A)` to implement this part. Here u is the size of the upper hull UH, l is the size of the lower hull LH, and r is the radius of each circle. The function should populate two arrays: T (the array of segments on the boundary) and A (the array of arcs on the boundary). Write another function `printcontzone(u,l,T,A)` to print the boundary in the format specified in the sample output.

The main function

- The user supplies n (the number of points) and r (the radius of each circle).
- The user specifies the n points $(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)$ in that order.
- Sort the points, and print the sorted list.
- Compute the upper and the lower hulls, and print the vertices on them.
- Compute the upper and the lower sections of the boundary of the containment zone, and print the arcs and the segments in the clockwise order.

Submit a single C source file. Do not use global/static variables.

Sample I/O

The following transcript corresponds to the example given earlier.

```
12
0.144337567297406
0.174491577862991 0.757806550598613
0.670734935286797 0.730248320722137
0.732804883612695 0.366851389113279
0.304578738428922 0.772376182848763
0.258718805042430 0.476162853872479
0.839437898173667 0.304665513012868
0.406902698058124 0.356899275610642
0.346442048599218 0.811870507342681
0.711229805234461 0.232718535341657
0.245613896402351 0.360662144776742
0.578360871681180 0.224183475703086
0.398011469467548 0.532126907972678

+++ Circles after sorting
    0.174491577862991 0.757806550598613
    0.245613896402351 0.360662144776742
    0.258718805042430 0.476162853872479
    0.304578738428922 0.772376182848763
    0.346442048599218 0.811870507342681
    0.398011469467548 0.532126907972678
    0.406902698058124 0.356899275610642
    0.578360871681180 0.224183475703086
    0.670734935286797 0.730248320722137
    0.711229805234461 0.232718535341657
    0.732804883612695 0.366851389113279
    0.839437898173667 0.304665513012868

+++ Upper hull
    0.174491577862991 0.757806550598613
    0.346442048599218 0.811870507342681
    0.670734935286797 0.730248320722137
    0.839437898173667 0.304665513012868

+++ Lower hull
    0.839437898173667 0.304665513012868
    0.711229805234461 0.232718535341657
    0.578360871681180 0.224183475703086
    0.245613896402351 0.360662144776742
    0.174491577862991 0.757806550598613

+++ The containment zone
--- Upper section
    Arc     : (0.174491577862991,0.757806550598613) From 3.141592653589793 to 1.875425660245566
    Tangent : From (0.131199026322504,0.895498555865614) to (0.303149497058732,0.949562512609681)
    Arc     : (0.346442048599218,0.811870507342681) From 1.875425660245566 to 1.324225070167672
    Tangent : From (0.381672013906631,0.951842586264771) to (0.705964900594210,0.870220399644227)
    Arc     : (0.670734935286797,0.730248320722137) From 1.324225070167672 to 0.377403016683020
    Tangent : From (0.804914714121416,0.783437796045983) to (0.973617677008286,0.357854988336714)
    Arc     : (0.839437898173667,0.304665513012868) From 0.377403016683020 to 0.000000000000000
--- Lower section
    Arc     : (0.839437898173667,0.304665513012868) From 0.000000000000000 to -1.059415144264078
    Tangent : From (0.910074134447689,0.178793135350516) to (0.781866041508483,0.106846157679305)
    Arc     : (0.711229805234461,0.232718535341657) From -1.059415144264078 to -1.506647779515518
    Tangent : From (0.720482501583259,0.088677843370220) to (0.587613568029979,0.080142783731650)
    Arc     : (0.578360871681180,0.224183475703086) From -1.506647779515518 to -1.960028388757219
    Tangent : From (0.523587935953457,0.090642250438751) to (0.190840960674627,0.227120919512406)
    Arc     : (0.245613896402351,0.360662144776742) From -1.960028388757219 to -2.964386842047912
    Tangent : From (0.103536641865204,0.335218342456331) to (0.032414323325844,0.732362748278203)
    Arc     : (0.174491577862991,0.757806550598613) From -2.964386842047912 to -3.141592653589793
```

Graphic visualization

You can visualize the output of your program using a graphics utility. If you have done a course on Computer Graphics, you should know how to do that. If not, here is an easy way to generate image files. We use the fig format to view and export the output to a jpg image. **This is not a part of your assignment. Do not submit codes for generating fig files.** You can do it in your personal codes, in case this interests you.

A fig file can be viewed and processed by the XWindows utility **`xfig`**. If you use Microsoft Windows, install **`WinFIG`**. Fig files are text files, and can be easily generated. The file should start with the following lines.

```
#FIG 3.2  Produced by me
Landscape
Center
Metric
A4
100.00
Single
-2
1200 2
```

The fig format treats the top left corner as the origin. The horizontal direction is the x direction, and the vertical direction is the y direction. We are interested in the 1×1 square. Fig files do not support fractional coordinates. You choose a resolution (recommended value is 10000). For a point (x, y) with our convention of origin at the bottom left corner, the coordinates will be $(10000x, 10000(1-y))$ both rounded to the nearest integers (use the math library function **`round`**). Let us call these converted coordinates (pos_x, pos_y).

Let us now see how different objects can be represented in the fig format. First consider a **circle** C with center at (x, y) and radius r. Then $(x+r, y)$ is a point on the circle. You represent the circle as follows.

```
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 pos_x pos_y round(10000*r) pos_x pos_y pos_x+r pos_y
```

You also need to show the center by a filled circle of some small radius ρ. The value $\rho = 0.005$ is recommended. The center can be drawn as follows.

```
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 pos_x pos_y round(10000*rho) pos_x pos_y pos_x+rho pos_y
```

Both segments and the convex hull can be represented by a **poly-line** object. Let $P_1, P_2, P_3, \ldots, P_k$ be a poly-line of k points (for a segment, $k = 2$, whereas for a convex hull, k is one plus the number of vertices, because you have to repeat the first vertex for getting a closed polygon). Let $P_i = (x_i, y_i)$. Denote (pos_{x_i}, pos_{y_i}) by $(posx_i, posy_i)$. A poly-line specification requires two lines. The second line must start with a tab.

```
2 1 0 4 4 7 40 -1 -1 0.000 0 0 -1 0 0 k
        posx_1 posy_1 posx_2 posy_2 ... posx_k posy_k
```

In order to draw the segments in the yellow color, use 31 in place of the second 4 in the first line.

```
2 1 0 4 31 7 40 -1 -1 0.000 0 0 -1 0 0 2
        posx_1 posy_1 posx_2 posy_2
```

You should also show the 1×1 bounding box. The green color has code 2.

Finally, you need to draw an **arc**. Let the center of the circle be (x, y), and the arc extends from the angle θ_s to the angle θ_e in the clockwise direction. The arc requires a third point. You may use the angle $\theta_m = (\theta_s + \theta_e)/2$. Find the three points P_s, P_m, P_e on the circle at the angles $\theta_s, \theta_m, \theta_e$. The arc can be drawn as follows.

```
5 1 0 4 18 7 40 -1 -1 0.000 0 0 0 0 pos_x pos_y posx_s posy_s posx_m posy_m posx_e posy_e
```

If you want to know more about the fig format, you can visit the following sites.

`http://mcj.sourceforge.net/`

`https://www.xfig.org/fig-format.html`

In order to install **xfig** in Ubuntu, use the following.

```
sudo apt install xfig gsfonts-x11
```

WinFIG can be downloaded from:

```
http://winfig.com/
```

You can convert a fig file interactively using **xfig** or **WinFIG**, or in the command line as:

```
fig2dev -L jpeg -b 20 -m 1.50 -S 4 -q 95 output.fig output.jpg
```

The fig file for the example

```
#FIG 3.2  Produced by AD
Landscape
Center
Metric
A4
100.00
Single
-2
1200 2
2 1 0 2 2 7 60 -1 -1 0.000 0 0 -1 0 0 5
        0 0 0 10000 10000 10000 10000 0 0 0
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 1745 2422 1443 1443 1745 2422 3188 2422
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 1745 2422 50 50 1745 2422 1795 2422
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 2456 6393 1443 1443 2456 6393 3900 6393
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 2456 6393 50 50 2456 6393 2506 6393
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 2587 5238 1443 1443 2587 5238 4031 5238
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 2587 5238 50 50 2587 5238 2637 5238
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 3046 2276 1443 1443 3046 2276 4489 2276
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 3046 2276 50 50 3046 2276 3096 2276
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 3464 1881 1443 1443 3464 1881 4908 1881
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 3464 1881 50 50 3464 1881 3514 1881
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 3980 4679 1443 1443 3980 4679 5423 4679
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 3980 4679 50 50 3980 4679 4030 4679
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 4069 6431 1443 1443 4069 6431 5512 6431
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 4069 6431 50 50 4069 6431 4119 6431
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 5784 7758 1443 1443 5784 7758 7227 7758
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 5784 7758 50 50 5784 7758 5834 7758
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 6707 2698 1443 1443 6707 2698 8151 2698
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 6707 2698 50 50 6707 2698 6757 2698
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 7112 7673 1443 1443 7112 7673 8556 7673
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 7112 7673 50 50 7112 7673 7162 7673
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 7328 6331 1443 1443 7328 6331 8771 6331
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 7328 6331 50 50 7328 6331 7378 6331
1 3 0 4 1 7 50 -1 -1 0.000 1 0.0000 8394 6953 1443 1443 8394 6953 9838 6953
1 3 0 4 0 7 30 -1 0 0.000 1 0.0000 8394 6953 50 50 8394 6953 8444 6953
2 1 0 4 4 7 40 -1 -1 0.000 0 0 -1 0 0 8
        1745 2422 3464 1881 6707 2698 8394 6953 7112 7673 5784 7758 2456 6393 1745 2422
2 1 0 4 31 7 40 -1 -1 0.000 0 0 -1 0 0 2
        1312 1045 3031 504
2 1 0 4 31 7 40 -1 -1 0.000 0 0 -1 0 0 2
        3817 482 7060 1298
2 1 0 4 31 7 40 -1 -1 0.000 0 0 -1 0 0 2
        8049 2166 9736 6421
2 1 0 4 31 7 40 -1 -1 0.000 0 0 -1 0 0 2
        9101 8212 7819 8932
2 1 0 4 31 7 40 -1 -1 0.000 0 0 -1 0 0 2
        7205 9113 5876 9199
2 1 0 4 31 7 40 -1 -1 0.000 0 0 -1 0 0 2
        5236 9094 1908 7729
2 1 0 4 31 7 40 -1 -1 0.000 0 0 -1 0 0 2
        1035 6648 324 2676
5 1 0 4 18 7 40 -1 -1 0.000 0 0 0 0 1745 2422 302 2422 581 1568 1312 1045
5 1 0 4 18 7 40 -1 -1 0.000 0 0 0 0 3464 1881 3031 504 3423 439 3817 482
5 1 0 4 18 7 40 -1 -1 0.000 0 0 0 0 6707 2698 7060 1298 7659 1612 8049 2166
5 1 0 4 18 7 40 -1 -1 0.000 0 0 0 0 8394 6953 9736 6421 9812 6683 9838 6953
5 1 0 4 18 7 40 -1 -1 0.000 0 0 0 0 8394 6953 9838 6953 9640 7683 9101 8212
5 1 0 4 18 7 40 -1 -1 0.000 0 0 0 0 7112 7673 7819 8932 7522 9057 7205 9113
5 1 0 4 18 7 40 -1 -1 0.000 0 0 0 0 5784 7758 5876 9199 5550 9183 5236 9094
5 1 0 4 18 7 40 -1 -1 0.000 0 0 0 0 2456 6393 1908 7729 1333 7300 1035 6648
5 1 0 4 18 7 40 -1 -1 0.000 0 0 0 0 1745 2422 324 2676 307 2550 302 2422
```