

Python implementation of the Jarvis march algorithm for finding the convex hull of a nonconvex polygon:

- (d) Python implementation of the Jarvis march algorithm for finding the convex hull of a nonconvex polygon

1- Using list data structure

### Code

```
def convex_hull(polygon):
    # Find the leftmost vertex of the polygon
    leftmost = min(polygon, key=lambda v: v[0])

    # Create a list to store the convex hull vertices
    hull = [leftmost]

    # Start at the leftmost vertex and iterate clockwise
    current = leftmost
    while True:
        next_vertex = None
        for vertex in polygon:
            if vertex == current:
                continue
            elif next_vertex is None:
                next_vertex = vertex
            else:
                # Compare the angle between the current vertex and the
                next candidate vertex
                # with the angle between the current vertex and the
                next_vertex
                cross_product = (vertex[0] - current[0]) * (next_vertex[1]
- current[1]) - (vertex[1] - current[1]) * (next_vertex[0] - current[0])
                if cross_product > 0:
                    next_vertex = vertex
                elif cross_product == 0:
                    # If the cross product is 0, choose the vertex that is
                    farther away
                    dist1 = (vertex[0] - current[0]) ** 2 + (vertex[1] -
current[1]) ** 2
                    dist2 = (next_vertex[0] - current[0]) ** 2 +
(next_vertex[1] - current[1]) ** 2
```

```

        if dist1 > dist2:
            next_vertex = vertex

    # Add the next vertex to the convex hull
    hull.append(next_vertex)

    # Update the current vertex
    current = next_vertex

    # If we have completed a loop and returned to the leftmost vertex,
    exit the loop
    if current == leftmost:
        break

    return hull

def read_txt_file(file_path):
    # Read the contents of the file into a list of strings
    with open(file_path, 'r') as file:
        lines = file.readlines()

    # Create an empty list to store the data
    data = []

    # Iterate through the lines and split them into columns
    for line in lines:
        columns = line.strip().split()
        del(columns[2])
        columns[0] = int(columns[0])
        columns[1] = int(columns[1])
        data.append(columns)
    return data

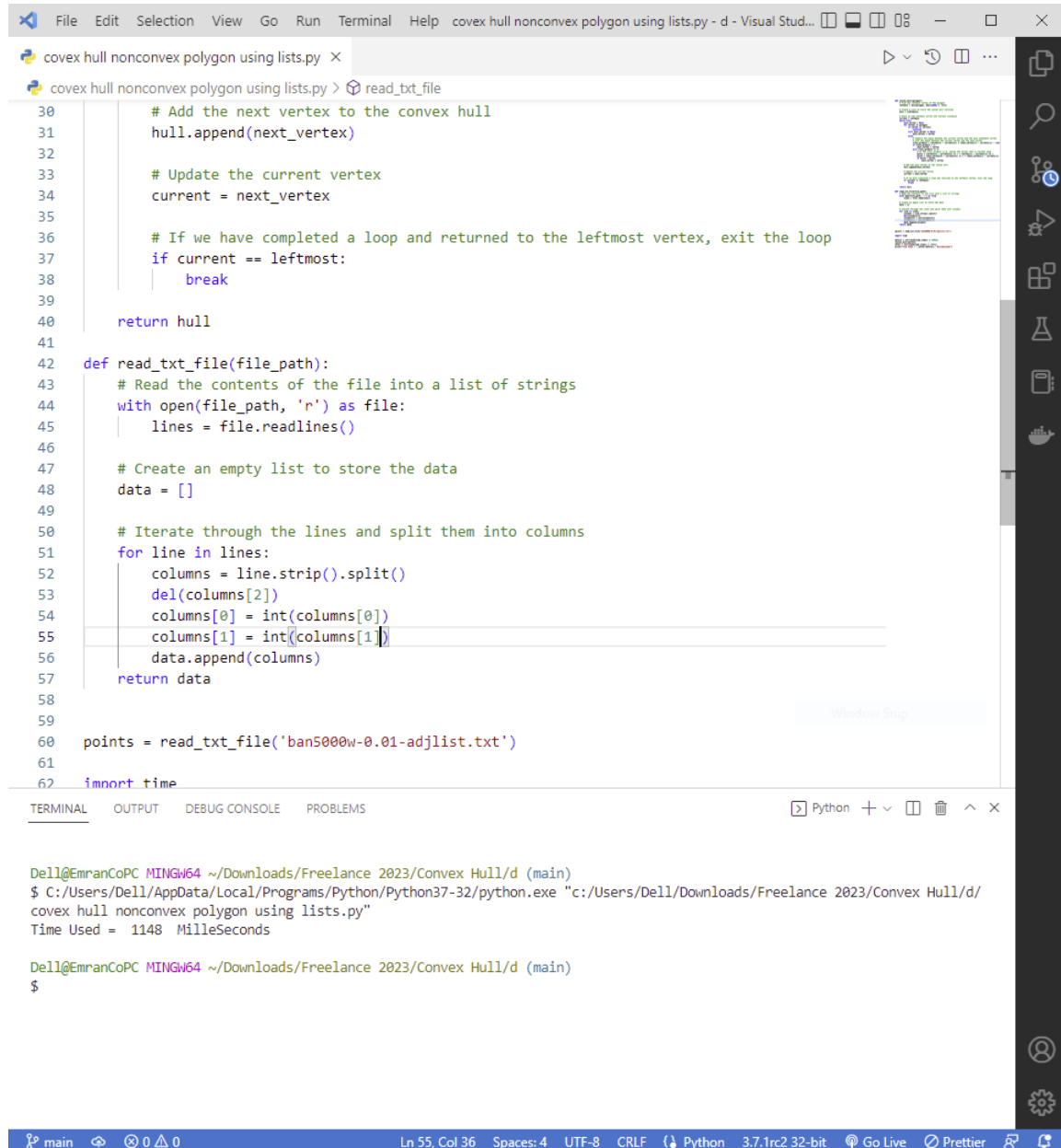
points = read_txt_file('ban5000w-0.01-adjlist.txt')

import time

before = int(round(time.time() * 1000))
convex_hull(points)
after = int(round(time.time() * 1000))
print("Time Used = ",(after-before)," MilleSeconds")

```

## Output



```
File Edit Selection View Go Run Terminal Help covex hull nonconvex polygon using lists.py - d - Visual Stud...
covex hull nonconvex polygon using lists.py > read_txt_file
30     # Add the next vertex to the convex hull
31     hull.append(next_vertex)
32
33     # Update the current vertex
34     current = next_vertex
35
36     # If we have completed a loop and returned to the leftmost vertex, exit the loop
37     if current == leftmost:
38         break
39
40     return hull
41
42 def read_txt_file(file_path):
43     # Read the contents of the file into a list of strings
44     with open(file_path, 'r') as file:
45         lines = file.readlines()
46
47     # Create an empty list to store the data
48     data = []
49
50     # Iterate through the lines and split them into columns
51     for line in lines:
52         columns = line.strip().split()
53         del(columns[2])
54         columns[0] = int(columns[0])
55         columns[1] = int(columns[1])
56         data.append(columns)
57     return data
58
59
60 points = read_txt_file('ban5000w-0.01-adjlist.txt')
61
62 import time

TERMINAL OUTPUT DEBUG CONSOLE PROBLEMS
Python + - [ ] [x] [^] [x]

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ C:/Users/Dell/AppData/Local/Programs/Python/Python37-32/python.exe "c:/Users/Dell/Downloads/Freelance 2023/Convex Hull/d/
covex hull nonconvex polygon using lists.py"
Time Used = 1148 MilleSeconds

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$
```

## 2- Using queue data structure

### Code

```
import math
```

```
def convex_hull(polygon):
```

```

# Find the leftmost vertex of the polygon
leftmost = min(polygon, key=lambda v: v[0])

# Create a list to store the convex hull vertices
hull = [leftmost]

# Start at the leftmost vertex and iterate clockwise
current = leftmost
while True:
    next_vertex = None
    for vertex in polygon:
        if vertex == current:
            continue
        elif next_vertex is None:
            next_vertex = vertex
        else:
            # Compare the angle between the current vertex and the
            next candidate vertex
            # with the angle between the current vertex and the
            next_vertex
            cross_product = (vertex[0] - current[0]) * (next_vertex[1]
- current[1]) - (vertex[1] - current[1]) * (next_vertex[0] - current[0])
            if cross_product > 0:
                next_vertex = vertex
            elif cross_product == 0:
                # If the cross product is 0, choose the vertex that is
            farther away
                dist1 = math.sqrt((vertex[0] - current[0]) ** 2 +
(vertex[1] - current[1]) ** 2)
                dist2 = math.sqrt((next_vertex[0] - current[0]) ** 2 +
(next_vertex[1] - current[1]) ** 2)
                if dist1 > dist2:
                    next_vertex = vertex

    # Add the next vertex to the convex hull
    hull.append(next_vertex)

    # Update the current vertex
    current = next_vertex

    # If we have completed a loop and returned to the leftmost vertex,
    exit the loop
    if current == leftmost:
        break

```

```

    return hull

def read_txt_file(file_path):
    # Read the contents of the file into a list of strings
    with open(file_path, 'r') as file:
        lines = file.readlines()

    # Create an empty list to store the data
    data = []

    # Iterate through the lines and split them into columns
    for line in lines:
        columns = line.strip().split()
        del(columns[2])
        columns[0] = int(columns[0])
        columns[1] = int(columns[1])
        data.append(columns)
    return data

points = read_txt_file('./ban5000w-0.01-adjlist.txt')

import time

before = int(round(time.time() * 1000))
convex_hull(points)
after = int(round(time.time() * 1000))
print("Time Used = ",(after-before)," MilleSeconds")

```

## Output

```
File Edit Selection View Go Run Terminal ... covex hull nonconvex polygon using queue.py - d - Visual Stu...
covex hull nonconvex polygon using lists.py covex hull nonconvex polygon using queue.py X
covex hull nonconvex polygon using queue.py > convex_hull
1 import math
2
3 def convex_hull(polygon):
4     # Find the leftmost vertex of the polygon
5     leftmost = min(polygon, key=lambda v: v[0])
6
7     # Create a list to store the convex hull vertices
8     hull = [leftmost]
9
10    # Start at the leftmost vertex and iterate clockwise
11    current = leftmost
12    while True:
13        next_vertex = None
14        for vertex in polygon:
15            if vertex == current:
16                continue
17            elif next_vertex is None:
18                next_vertex = vertex
19            else:
20                # Compare the angle between the current vertex and the next candidate vertex
21                # with the angle between the current vertex and the next_vertex
22                cross_product = (vertex[0] - current[0]) * (next_vertex[1] - current[1]) - (vertex[1] - current[1]) * (next_vertex[0] - current[0])
23                if cross_product > 0:
24                    next_vertex = vertex
25                elif cross_product == 0:
26                    # If the cross product is 0, choose the vertex that is farther away
27                    dist1 = math.sqrt((vertex[0] - current[0]) ** 2 + (vertex[1] - current[1]) ** 2)
28                    dist2 = math.sqrt((next_vertex[0] - current[0]) ** 2 + (next_vertex[1] - current[0]) ** 2)
29                    if dist1 > dist2:
30                        next_vertex = vertex
31
32        # Add the next vertex to the convex hull
33        hull.append(next_vertex)
```

TERMINAL OUTPUT DEBUG CONSOLE PROBLEMS Python + - X

```
Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ C:/Users/Dell/AppData/Local/Programs/Python/Python37-32/python.exe "c:/Users/Dell/Downloads/Freelance 2023/Convex Hull/d/covex hull nonconvex polygon using lists.py"
Time Used = 1148 MilleSeconds

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ C:/Users/Dell/AppData/Local/Programs/Python/Python37-32/python.exe "c:/Users/Dell/Downloads/Freelance 2023/Convex Hull/d/covex hull nonconvex polygon using queue.py"
Time Used = 1047 MilleSeconds

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$
```

main\* 0 0 Ln 12, Col 16 Spaces: 4 UTF-8 CRLF Python 3.7.1rc2 32-bit Go Live Prettier

### 3- Using stack data structure

#### Code

```
def convex_hull(points):
    # remove all points in the extremal quadrilateral
    xmin, ymin, xmax, ymax = float('inf'), float('inf'), float('-inf'),
float('-inf')
    for x, y in points:
        if x < xmin:
            xmin = x
        if y < ymin:
            ymin = y
        if x > xmax:
            xmax = x
        if y > ymax:
            ymax = y
    points = [p for p in points if not (p[0] == xmin or p[0] == xmax or
p[1] == ymin or p[1] == ymax)]
    # sort the points by x-coordinate
    points.sort(key=lambda p: (p[0], p[1]))
    # initialize the stack and add the leftmost point to it
    hull = []
    for p in points:
        while len(hull) > 1 and cross(hull[-2], hull[-1], p) <= 0:
            hull.pop()
        hull.append(p)
    # return the convex hull
    return hull

# function for computing the cross product of vectors (p1, p2) and (p1,
p3)
def cross(p1, p2, p3):
    return (p2[0] - p1[0]) * (p3[1] - p1[1]) - (p2[1] - p1[1]) * (p3[0] -
p1[0])

def read_txt_file(file_path):
    # Read the contents of the file into a list of strings
    with open(file_path, 'r') as file:
        lines = file.readlines()

    # Create an empty list to store the data
    data = []
```

```

# Iterate through the lines and split them into columns
for line in lines:
    columns = line.strip().split()
    del(columns[2])
    columns[0] = int(columns[0])
    columns[1] = int(columns[1])
    data.append(columns)
return data

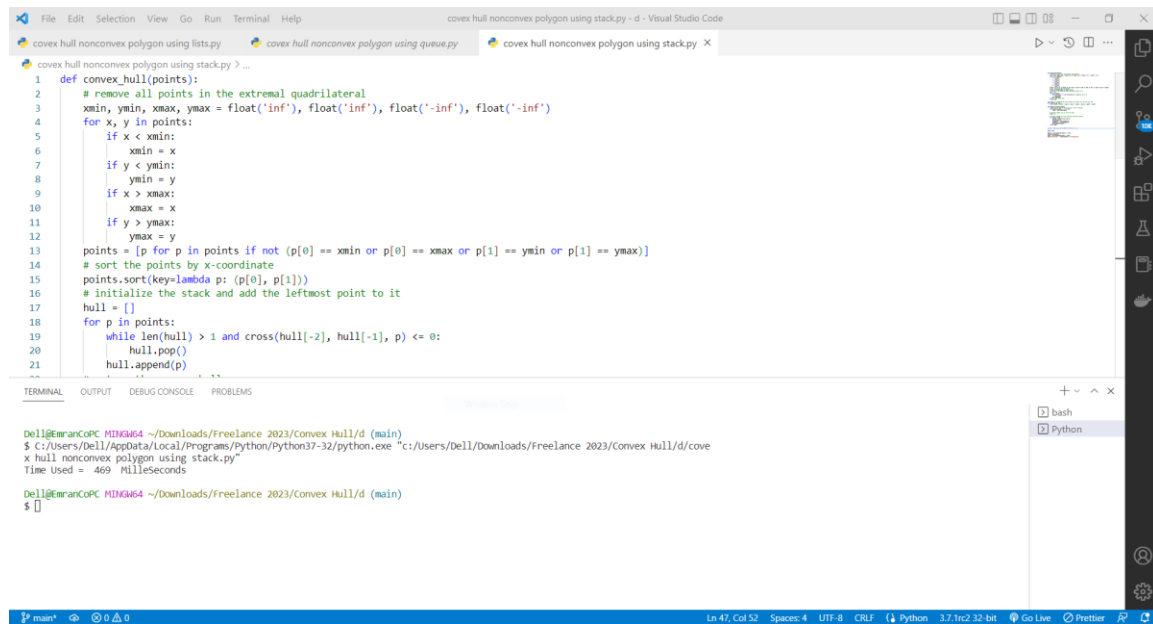
points = read_txt_file('ban5000w-0.01-adjlist.txt')

import time

before = int(round(time.time() * 1000))
convex_hull(points)
after = int(round(time.time() * 1000))
print("Time Used = ",(after-before)," MilleSeconds")

```

## Output



The screenshot shows a Visual Studio Code window with a Python script titled 'convex\_hull\_nonconvex\_polygon\_using\_stack.py'. The script defines a function 'convex\_hull(points)' that calculates the convex hull of a set of points using a stack-based algorithm. It removes points on the extremal quadrilateral, sorts the remaining points by x-coordinate, and then iteratively adds points to the hull while maintaining the convexity. The main part of the script reads a file 'ban5000w-0.01-adjlist.txt', imports the 'time' module, and prints the execution time in milliseconds.

The terminal output shows the script being executed from the command line. The output indicates that the script ran successfully and took 469 milliseconds to complete.

```

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ c:\Users/Dell\AppData/Local/Programs/Python/Python37-32/python.exe "c:\Users/Dell/Downloads/Freelance 2023/Convex Hull/d/convex_hull_nonconvex_polygon_using_stack.py"
Time Used = 469 MilleSeconds

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$

```



#### 4- Using priority queue data structure

##### Code

```
import math

def convex_hull(points):
    # remove all points in the extremal quadrilateral
    xmin, ymin, xmax, ymax = float('inf'), float('inf'), float('-inf'),
float('-inf')
    for x, y in points:
        if x < xmin:
            xmin = x
        if y < ymin:
            ymin = y
        if x > xmax:
            xmax = x
        if y > ymax:
            ymax = y
    points = [p for p in points if not (p[0] == xmin or p[0] == xmax or
p[1] == ymin or p[1] == ymax)]
    # sort the points by polar angle with respect to the lowest point
    p0 = min(points, key=lambda p: (p[1], p[0]))
    points.sort(key=lambda p: (angle(p0, p), distance(p0, p)))
    # initialize the stack and add the first three points to it
    hull = []
    for p in points[:3]:
        while len(hull) > 1 and cross(hull[-2], hull[-1], p) <= 0:
            hull.pop()
        hull.append(p)
    # process the remaining points
    for p in points[3:]:
        while len(hull) > 1 and cross(hull[-2], hull[-1], p) <= 0:
            hull.pop()
        hull.append(p)
    # return the convex hull
    return hull

# function for computing the angle between two points
def angle(p1, p2):
    return math.atan2(p2[1] - p1[1], p2[0] - p1[0])

# function for computing the distance between two points
def distance(p1, p2):
```

```

    return math.sqrt((p2[1] - p1[1]) ** 2 + (p2[0] - p1[0]) ** 2)

# function for cross product of two vectors
def cross(p1, p2, p3):
    return (p2[0] - p1[0]) * (p3[1] - p1[1]) - (p2[1] - p1[1]) * (p3[0] -
p1[0])

def read_txt_file(file_path):
    # Read the contents of the file into a list of strings
    with open(file_path, 'r') as file:
        lines = file.readlines()

    # Create an empty list to store the data
    data = []

    # Iterate through the lines and split them into columns
    for line in lines:
        columns = line.strip().split()
        del(columns[2])
        columns[0] = int(columns[0])
        columns[1] = int(columns[1])
        data.append(columns)
    return data

points = read_txt_file('ban5000w-0.01-adjlist.txt')

import time

before = int(round(time.time() * 1000))
convex_hull(points)
after = int(round(time.time() * 1000))
print("Time Used = ",(after-before)," MilleSeconds")

```

## **Output**

The screenshot shows a Visual Studio Code editor with a Python file named 'convex hull nonconvex polygon using priority queue.py'. The code defines a function 'read\_txt\_file' that reads a file, splits each line into columns, and appends the second and third columns to a list. It then calls this function with the file path 'ban5000w-0.01-adjlist.txt' and assigns the result to 'points'. The terminal at the bottom shows the command being executed and the output: 'Time Used = 762 MilleSeconds'.

```
46
47 def read_txt_file(file_path):
48     # Read the contents of the file into a list of strings
49     with open(file_path, 'r') as file:
50         lines = file.readlines()
51
52     # Create an empty list to store the data
53     data = []
54
55     # Iterate through the lines and split them into columns
56     for line in lines:
57         columns = line.strip().split()
58         del(columns[2])
59         columns[0] = int(columns[0])
60         columns[1] = int(columns[1])
61         data.append(columns)
62     return data
63
64
65 points = read_txt_file('ban5000w-0.01-adjlist.txt')
66
```

```
De1l@EmranCPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ C:/Users/Dell/AppData/Local/Programs/Python/Python37-32/python.exe "c:/Users/Dell/Downloads/Freelance 2023/Convex Hull/d/convex hull nonconvex polygon using priority queue.py"
Time Used = 762 MilleSeconds

De1l@EmranCPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$
```

## Compare Algorithm 1 Vs Algorithm 2

Algorithm 1 : Time Used = 1148 Millisecond's

Algorithm 2 : Time Used = 1047 Millisecond's

Algorithm 3 : Time Used = 469 Millisecond's

Algorithm 4 : Time Used = 762 Millisecond's

```
convex hull nonconvex polygon using lists.py"
```

```
Time Used = 1148 MilleSeconds
```

```
convex hull nonconvex polygon using queue.py"
```

```
Time Used = 1047 MilleSeconds
```

```
x hull nonconvex polygon using stack.py"
```

```
Time Used = 469 MilleSeconds
```

```
$ C:/Users/Dell/AppData/Local/Pr
```

```
Time Used = 762 MilleSeconds
```

The Best is Algorithm 3