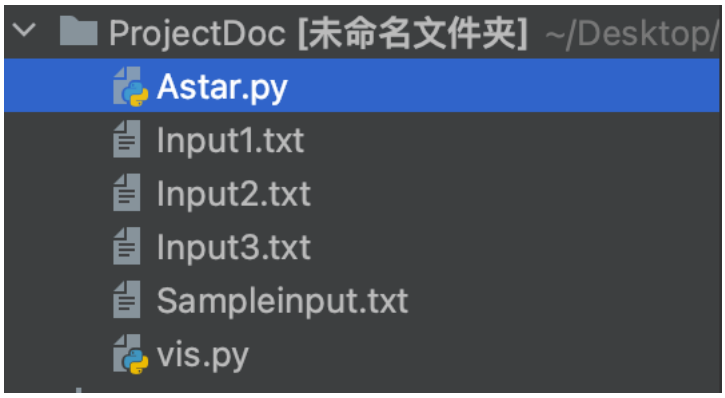
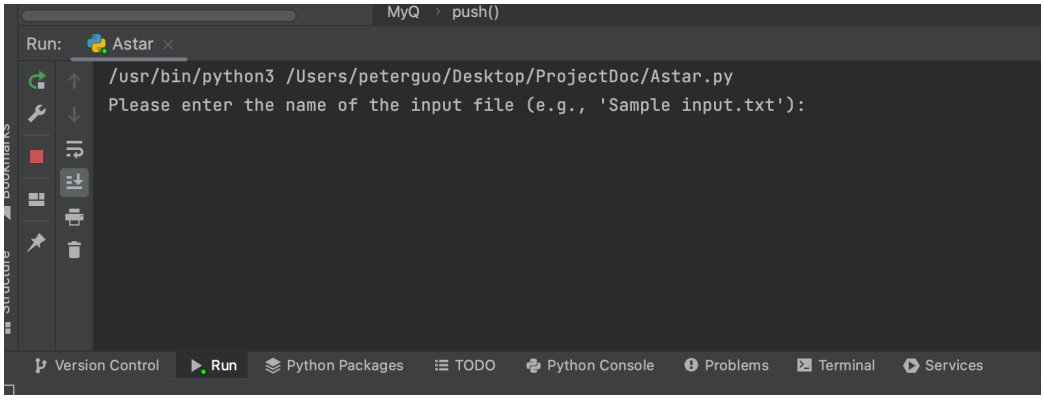
Name: Xinxue Guo
NetID: xg2407

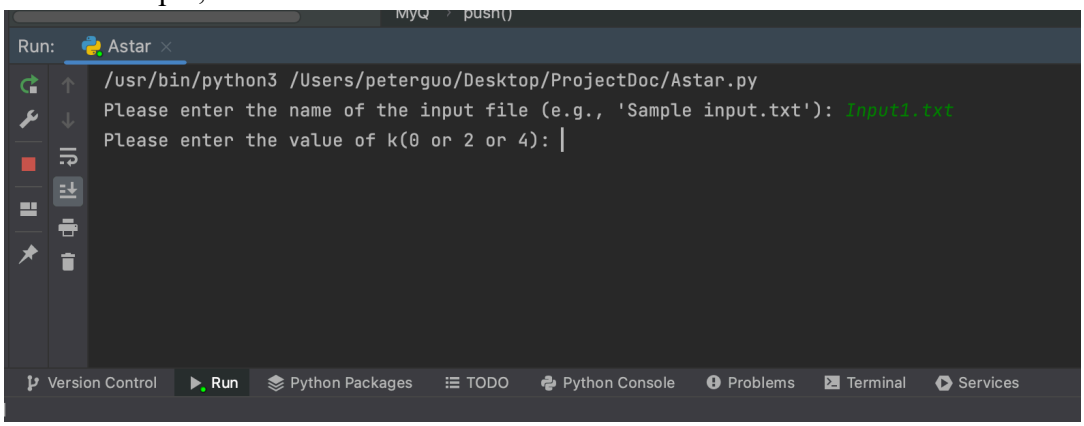
First Step:
Put our python file and input.txt into a same Doc.

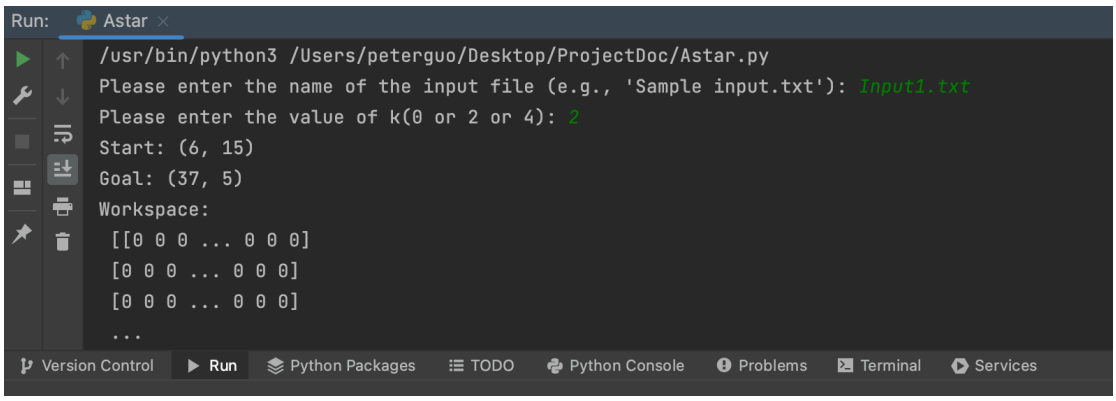


Second Step:
Run the Astar.py. The user can enter the input file name which want to test.
In our example, there will be Input1.txt

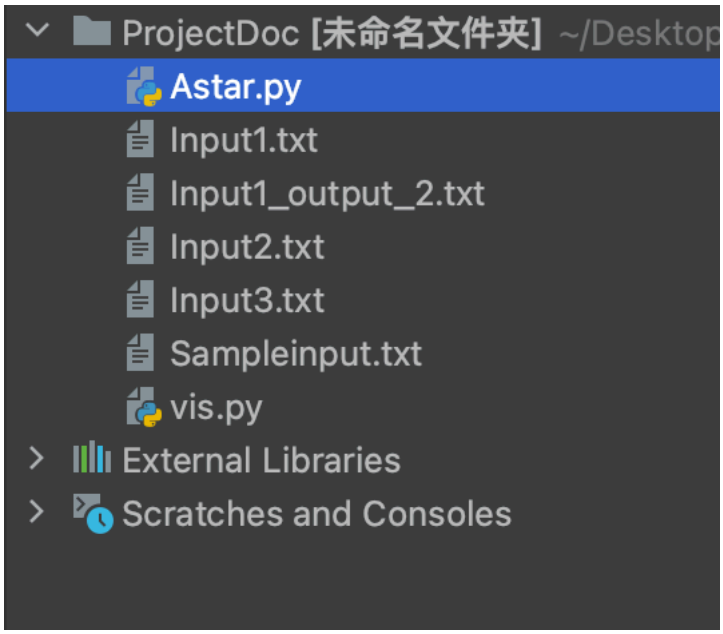


Third Step:
Enter the k value which can be 0 or 2 or 4.
In our example, k value will be 2.





Fourth Step:
The result will be in the same doc with Astar.py.



Here is the Astar.py code:

Astar.py

```python
import math
import numpy as np

# Set up the Queue
class MyQ:
    def __init__(self):
        self.lt = []

    def push(self, t):
        self.lt.append(t)
        self.lt.sort(key=lambda x: (x[1], x[2]), reverse=True)

    def pop(self):
        return self.lt.pop()[0]

    def __len__(self):
        return len(self.lt)
```

Astar.py

```python
class Astar:
    def __init__(self, w, h, s, g, workspace, k, input_file):
        self.w, self.h = w, h
        self.s, self.g = s, g
        # Flip the workspace upside down so that the input format and search format are consistent
        self.data = workspace
        self.k = k
        self.que = MyQ()
        self.que.push((s, 0, 0))
        self.N = 0  # Number of nodes generated
        self.parent = {}  # Parent node dictionary
        self.cost_g = {}  # g(n) value
        self.cost_f = {}  # f(n) value
        self.direction = {}  # Node direction index
        self.parent[s] = None
        self.cost_g[s] = 0
        self.cost_f[s] = self.heuristic(self.s, self.g)
        self.direction[s] = None  # The initial node has no direction
        self.input_file = input_file[:-4]
```

Astar.py

```
39
40      def find_path(self):
41          directions = [(0, 1), (-1, 1), (-1, 0), (-1, -1),
42                        (0, -1), (1, -1), (1, 0), (1, 1)]
43          while self.que:
44              current = self.que.pop()
45              if current == self.g:  # If reach the target node
46                  break
47              for node in self.neighbors(current):
48                  new_cost_g = self.cost_g[current] + self.cal_cost(current, node)  # g(n)
49                  if node not in self.cost_g or self.cost_g[node] > new_cost_g:
50                      self.cost_g[node] = new_cost_g
51                      h = self.heuristic(node, self.g)  # h(n)
52                      f = new_cost_g + h
53                      self.cost_f[node] = f  # record f(n)
54                      delta = (node[0] - current[0], node[1] - current[1])
55                      direction_index = directions.index(delta)
56                      self.direction[node] = direction_index  # Storage direction index
57                      self.N += 1  # Counting the number of nodes
58                      self.que.push((node, f, h))
59                      self.parent[node] = current  # Record parent node
60          self.draw_path()
61
62      def heuristic(self, node, goal):
63          return math.sqrt((goal[0] - node[0]) ** 2 + (goal[1] - node[1]) ** 2)
64
```

Astar.py

```
65      def draw_path(self):
66          path_nodes = []
67          path_directions = []
68          path_f_values = []
69          g = self.g  # Target Nodes
70          while g is not None:
71              path_nodes.append(g)
72              path_f_values.append(round(self.cost_f.get(g, 0), 2))
73              if self.direction.get(g) is not None:
74                  path_directions.append(self.direction[g])
75              g = self.parent[g]
76          path_nodes.reverse()
77          path_f_values.reverse()
78          path_directions.reverse()
79          d = len(path_nodes) - 1  # Search Depth
80
81          # Draw Path
82          for g in path_nodes:
83              check = (self.s[0], self.s[1])
84              check2 = (self.g[0], self.g[1])
85              if g != check and g != check2:
86                  self.data[g] = 4
87
88          # Create Output.txt
89          fileName = self.input_file+"_output_"+str(self.k)+".txt"
90          with open(fileName, 'w') as output_file:
91              output_file.write(str(d) + '\n')  # Search Depth
92              output_file.write(str(self.N) + '\n')  # The number of nodes generated
93              output_file.write(' '.join(map(str, path_directions)) + '\n')  # Direction Change Collection
94              output_file.write(' '.join(map(str, path_f_values)) + '\n')  # f(n) value
95              for row in self.data:
96                  output_file.write(' '.join(map(str, row)) + '\n')
97
```

Astar.py

```
    def neighbors(self, node):
        l = []
        directions = [(0, 1), (-1, 1), (-1, 0), (-1, -1),
                      (0, -1), (1, -1), (1, 0), (1, 1)]
        for d in directions:
            new_node = (node[0] + d[0], node[1] + d[1])
            # Make sure the new node is within the workspace and is not an obstruction
            if 0 <= new_node[0] < self.w and 0 <= new_node[1] < self.h and self.data[new_node] != 1:
                l.append(new_node)
        return l

    def cal_cost(self, current, next_node):
        directions = [(0, 1), (-1, 1), (-1, 0), (-1, -1),
                      (0, -1), (1, -1), (1, 0), (1, 1)]

        delta = (next_node[0] - current[0], next_node[1] - current[1])
        direction_index = directions.index(delta)
        # Calculate distance cost
        if direction_index % 2 == 0:
            path_cost = 1  # Horizontal and vertical movement
        else:
            path_cost = math.sqrt(2)  # Diagonal movement
        # Calculating the angle cost
        prev_direction = self.direction.get(current)
        angle_cost = self.cal_angle_cost(prev_direction, direction_index)
        return path_cost + angle_cost
```

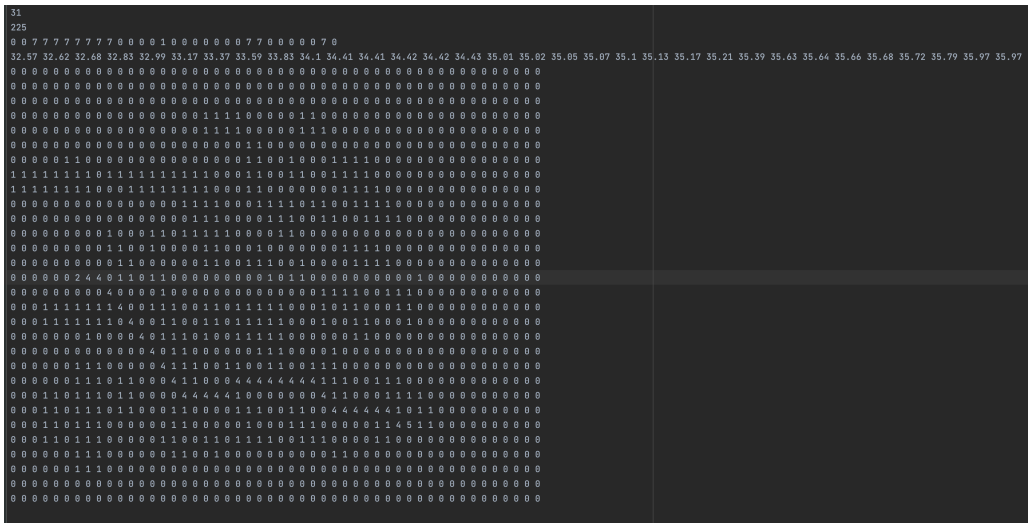
Astar.py

```
    def cal_angle_cost(self, prev_direction, current_direction):
        if prev_direction is None:
            return 0  # There is no angle cost for the initial move
        delta_theta = abs(current_direction - prev_direction)
        if delta_theta > 4:
            delta_theta = 8 - delta_theta
        angle_cost = self.k * (delta_theta * 45) / 180  # Convert direction index to angle calculation
        return angle_cost


def read_input_file(filename):
    with open(filename, 'r') as file:
        # Read the first line and get the starting and ending coordinates
        coordinates = list(map(int, file.readline().split()))
        start = (coordinates[0], coordinates[1])  # Start
        goal = (coordinates[2], coordinates[3])  # End
        # Read the remaining workspace matrix
        workspace = [list(map(int, line.split())) for line in file if line.strip()]
        workspace = np.array(workspace)
    return start, goal, workspace
```

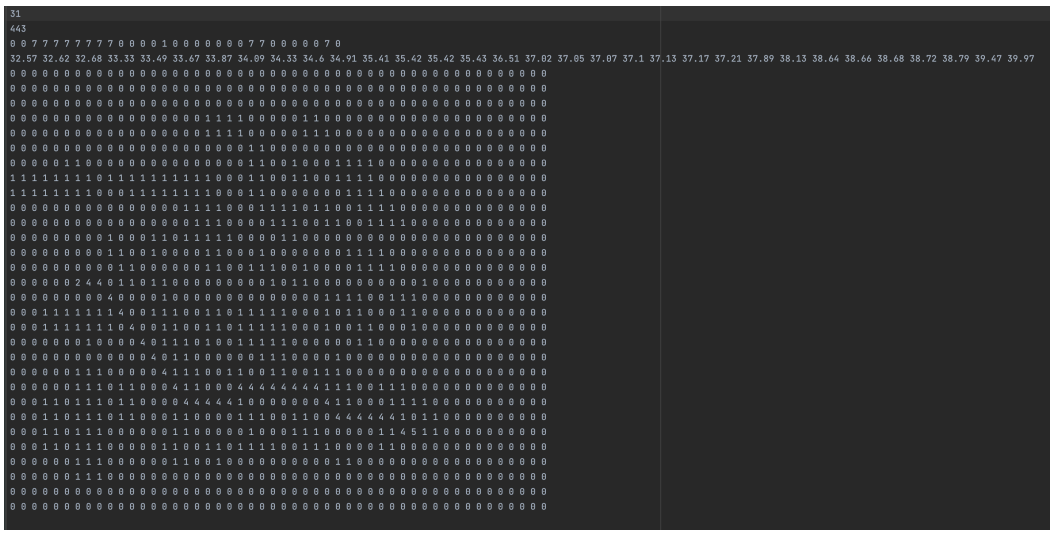
```
if __name__ == "__main__":
    input_file = input("Please enter the name of the input file (e.g., 'Sample input.txt'): ")
    k = int(input("Please enter the value of k(0 or 2 or 4): "))
    start, goal, workspace = read_input_file(input_file)
    print("Start:", start)  # Start
    print("Goal:", goal)  # End
    print("Workspace:\n", workspace)
    newStart = (workspace.shape[0] - start[1] - 1, start[0])  # Convert to numpy coordinate system
    newEnd = (workspace.shape[0] - goal[1] - 1, goal[0])
    print(newStart, newEnd)

    astar_solver = Astar(workspace.shape[0], workspace.shape[1], newStart, newEnd, workspace, k, input_file)
    astar_solver.find_path()
```

Here are the 3 different k value for Input1.txt

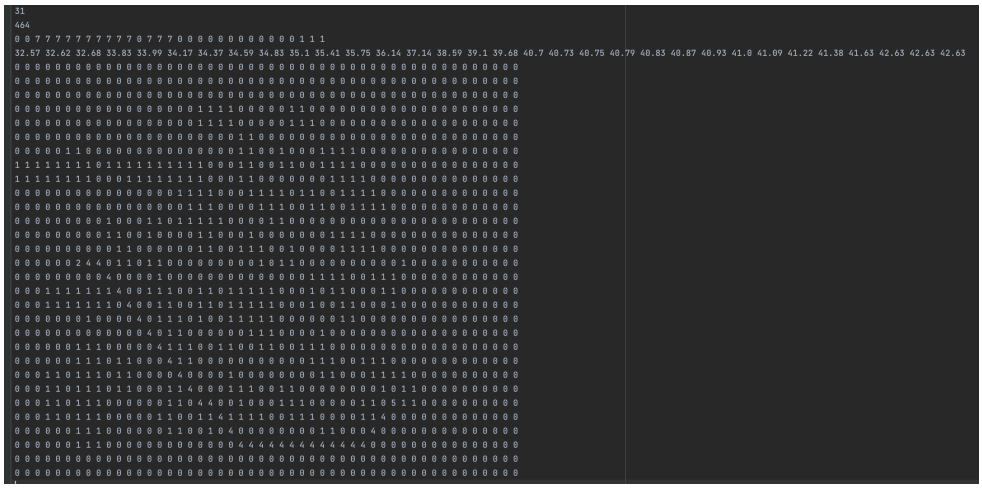
When k = 0:



When k = 2:

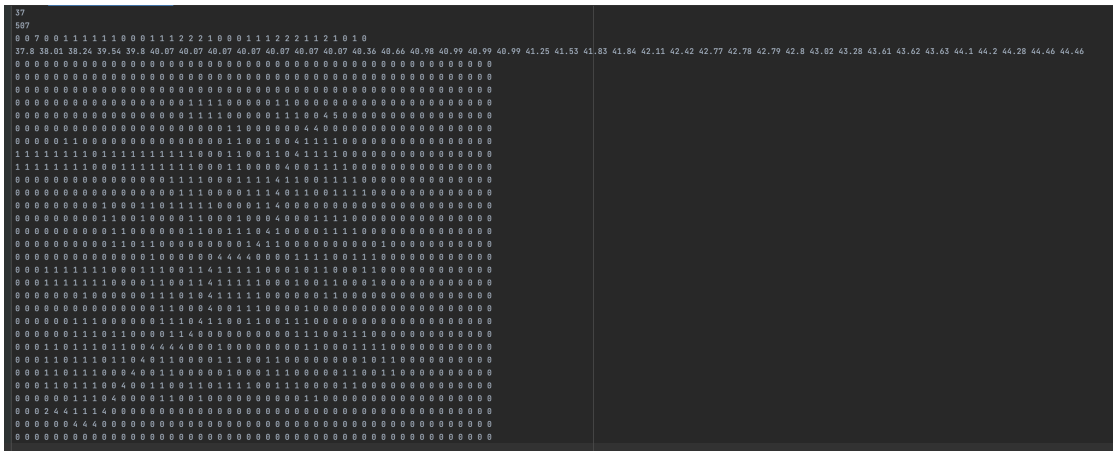


When k = 4:

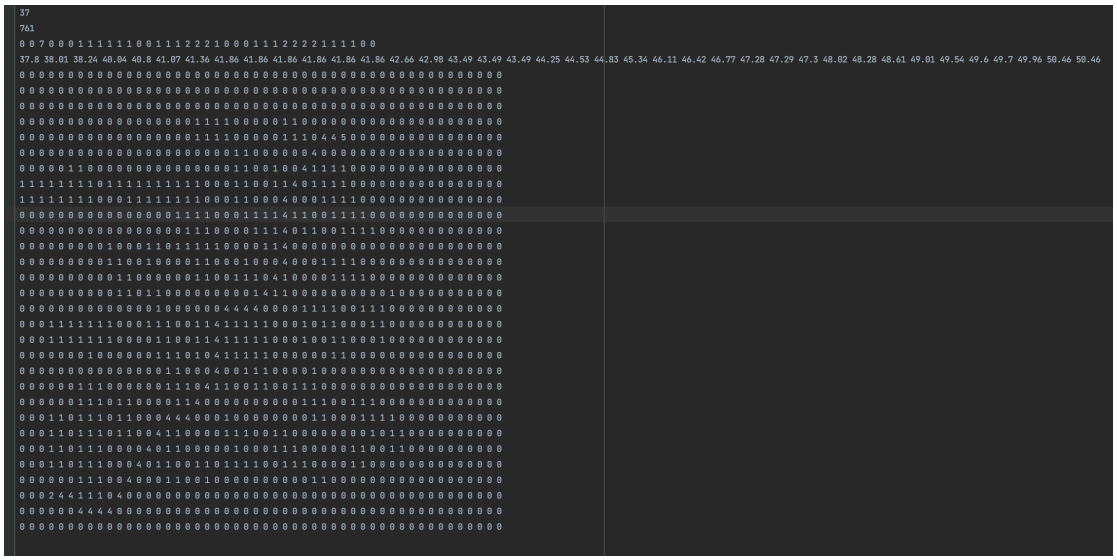


Here are the 3 different k value for Input2.txt

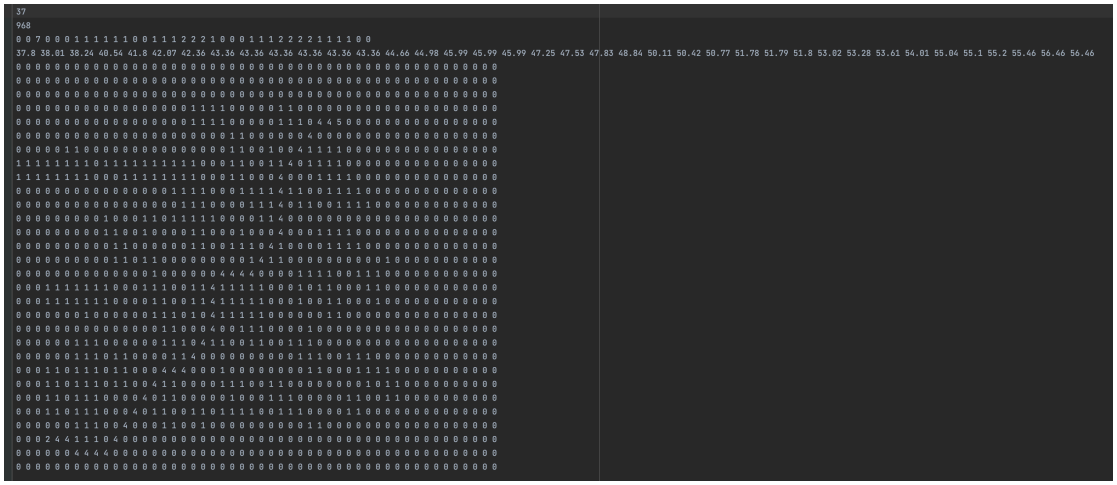
When k = 0:



When k = 2:

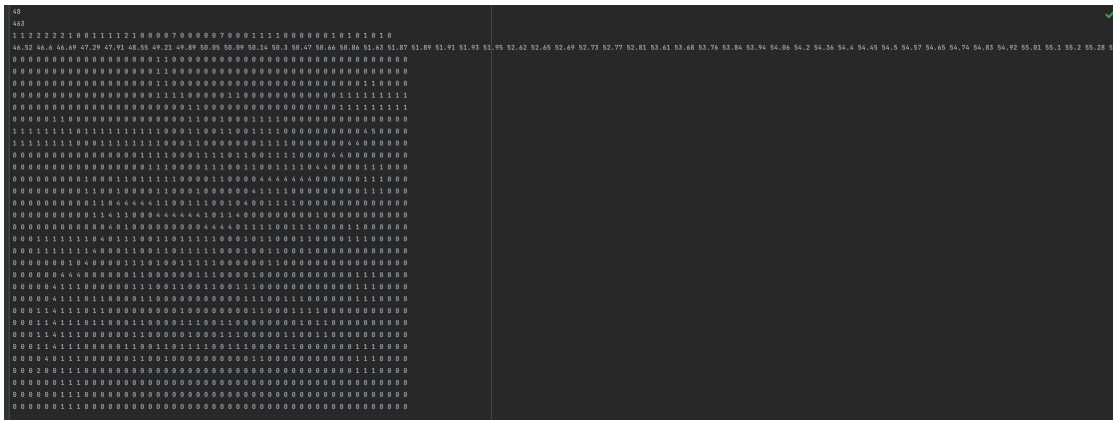


When k = 4:

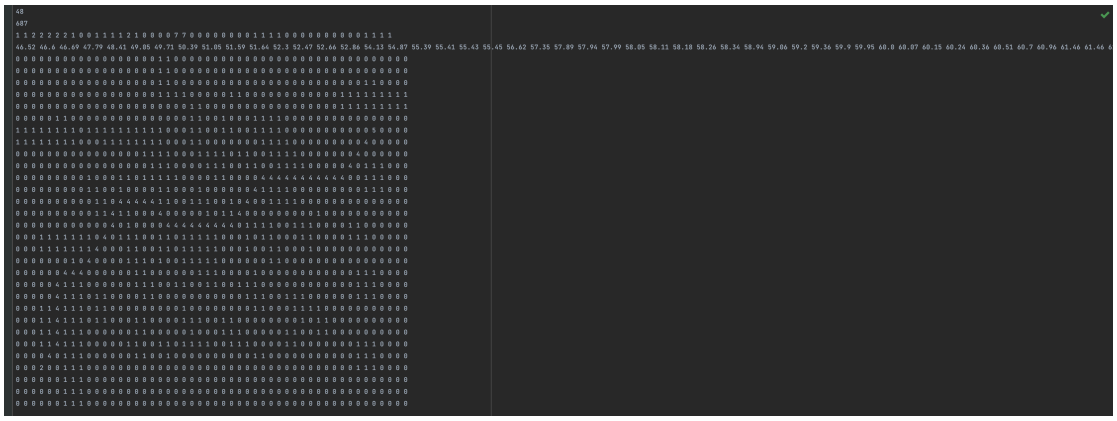


Here are the 3 different k value for Input3.txt

When k = 0:



When k = 2:



When k = 4: