Algorithm Design and Analysis (CS60007)
Assignment 3

October 27, 2020

Note

1. All codes should be written in C/C++ that can be compiled using GNU compiler and run in Ubuntu.
2. Don't copy codes from other groups. It is subject to penalty or failing in the extreme case.
3. All group members should participate in coding and with uniform division of tasks. In vivas, we may ask each individual to modify codes so as to change something or the other.
4. How to write an svg file?
   Just download and save `http://cse.iitkgp.ac.in/~pb/test.svg`. Open it in a browser, and also open it a plain text editor. Compare the image with the text—you will understand the basics. It's very simple.

   There are many resources on svg file format available in Internet. The most important one is `http://www.w3schools.com/svg/` where you will find many examples.

   A properly written svg can always be opened and seen in any browser like Mozilla or Chrome, and also in any standard image viewer. So, just check and ensure that the output svg produced by your code is opened and seen properly in these two browsers.
5. Use proper indentation and commenting in your codes so that we can understand your code.
6. Write in the beginning of each code the following:
   i) your names and roll nos.
   ii) purpose of the code / problem description in brief
   iii) compilation and execution instructions in Ubuntu
   iv) any other point that you feel important.
7. You have to submit all source codes and 3 sets of input and output files (small, medium, and large sizes).

1 Convex hull (divide and conquer)

a) Write a program to randomly generate n distinct points such that their coordinates are integers in the range $[20, 800]$ and divisible by 20.
User input: n.

Store the value of n and the coordinates of all n points in a plain text file named `points.txt`.

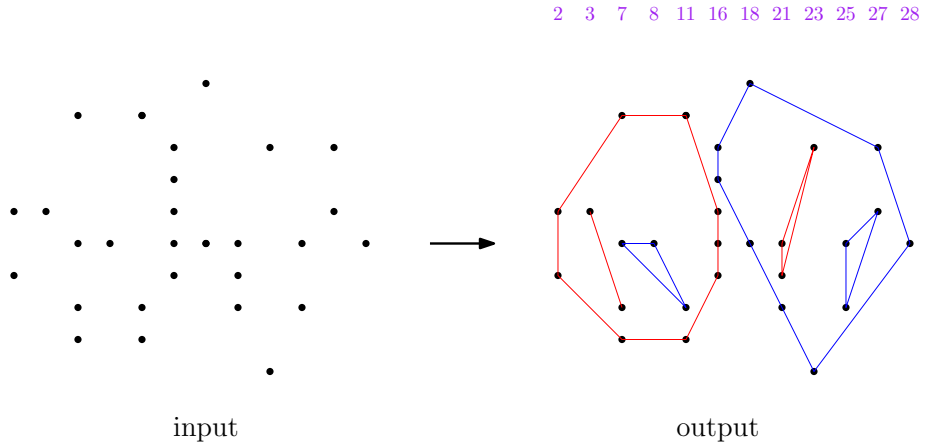
b) Write another program that reads the file `points.txt` and constructs its convex hull using divide and conquer.

Visualize the left convex hull in red and the right one in blue.

For the left convex hull, find the convex hulls of the points lying in its interior, and visualize them as above.
For the right one, do similar.

So, finally, hierarchical convex hulls will be formed.

Save the final result as an svg file named `hull.svg` in which all n points are colored black and hull edges colored red / blue.



input output

2 Convex hull (Graham scan)

a) Write a program to randomly generate n distinct points such that their coordinates are integers in the range $[20, 800]$ and divisible by 20.
User input: n.

Store the value of n and the coordinates of all n points in a plain text file named `points.txt`.

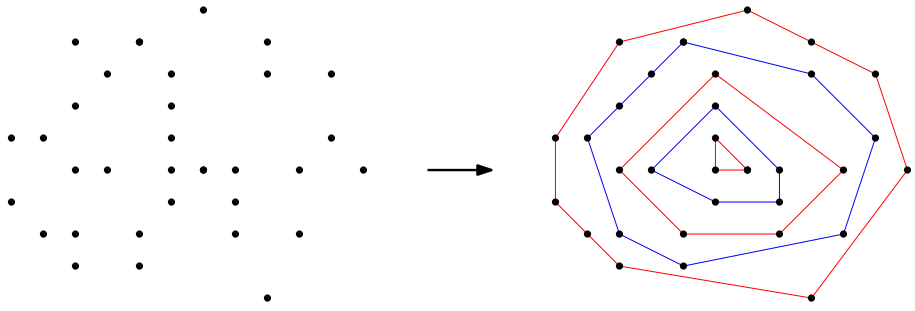
b) Write another program that reads the file `points.txt` and constructs its convex hull using Graham scan.

Visualize the convex hull H_1 in red.

Next, find the convex hull H_2 of the points lying in the interior of H_1, and visualize it in blue.

This way, hierarchical convex hulls will be formed.

Save the final result as an svg file named `hull.svg` in which all n points are colored black and hulls are colored alternately in red and blue.



input output

3 Euclidean MST by Prim's Algorithm

The task is to find MST by Prim's Algorithm for a completely connected Euclidean graph. (For an Euclidean graph, the weight of each edge is given by the Euclidean distance between its two endpoints.)
User input: integers k, n_1, n_2, n_3.

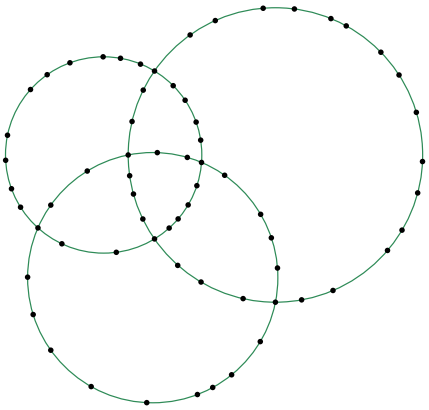
a) Write a program that takes as input the centers and the radii of three circles as follows:
1st circle: center $= (125, 175), r_1 = 100 + \Delta$
2nd circle: center $= (300, 175), r_2 = 150 + \Delta$
3rd circle: center $= (175, 325), r_3 = 125 + \Delta$
where Δ is a random integer in the interval $[-k, k]$, $k =$ user input.

Now, generate points on these circles as follows. They will serve as the vertices of the graph.

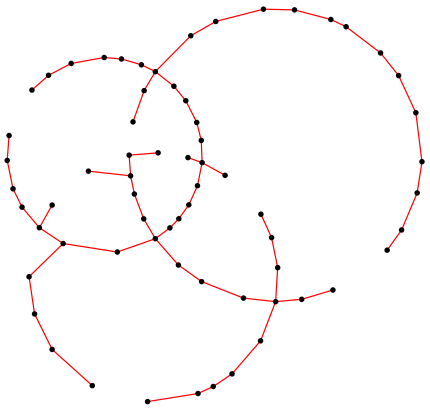
Generate the intersection points of the circles.
In addition, for each circle, generate random points lying on the circle so that two consecutive points are neither "too close" nor "too far".
The number of points n_1, n_2, n_3 on the three circles should be taken as input during execution of your code.

b) Consider the completely connected Euclidean graph defined by the above set of vertices, and compute its MST using Prim's algorithm.

c) Save the input and the output as `in.svg` and `out.svg`, as shown in the figure.



Input: Vertices of the completely connected graph and the circles on which they lie. Edges are not shown for clarity.



Output: MST.

4 Dijkstra's algorithm on Euclidean graph

The task is to find single-source shortest paths for a directed Euclidean graph. (For an Euclidean graph, the weight of each edge is given by the Euclidean distance between its two endpoints.)
User input: integers k, c, r, n_1, n_2.

a) Write a program that draws k circles with center c and radii $r_i = ri$ for $i = 1, 2, \ldots, k$.

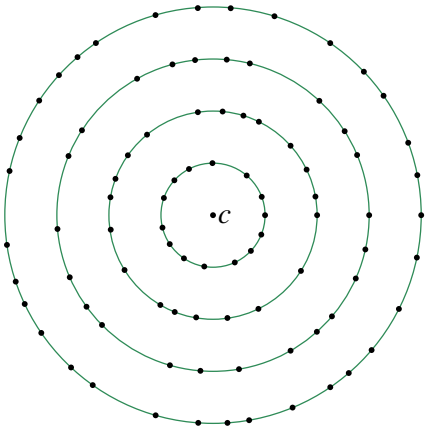
Now, randomly generate points on these circles so that the number of points on the i-th circle lies between $n_1\sqrt{r_i}$ and $n_2\sqrt{r_i}$. These points will serve as the vertices of the graph. Care should be taken so that for each circle, two consecutive points are neither "too close" nor "too far".

b) For $i = 1, 2, \ldots, k$, let P_i denote the set of random points on the i-th circle. Consider the directed Euclidean graph $G(V, E)$, where

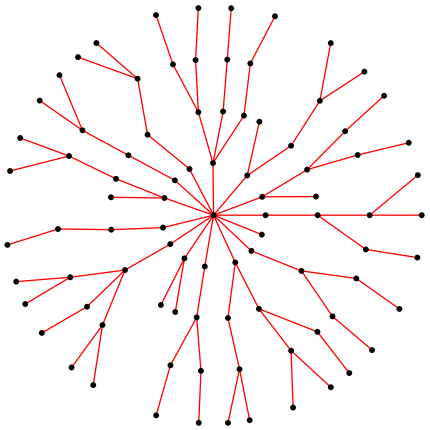
$$V = \{c\} \bigcup_{i=1}^{k} \{P_i\}$$
$$\text{and} \quad E = \{(c, p) : p \in P_1\} \bigcup_{i=1}^{k-1} \{(p, q) : p \in P_i, q \in P_{i+1}\}.$$

Compute shortest paths from c to all vertices.

c) Save the input and the output as `dijk-in.svg` and `dijk-out.svg`, as shown in the figure.



Input graph ($k = 4$). Vertices are random points on the concentric circles, edges are not shown for clarity.



Output: Shortest paths from c.

5 s-t shortest path

The task if to find a shortest path from a source s to a destination t in an Euclidean graph. (For an Euclidean graph, the weight of each edge is given by the Euclidean distance between its two endpoints.)
User input: integers k, m, n, g.

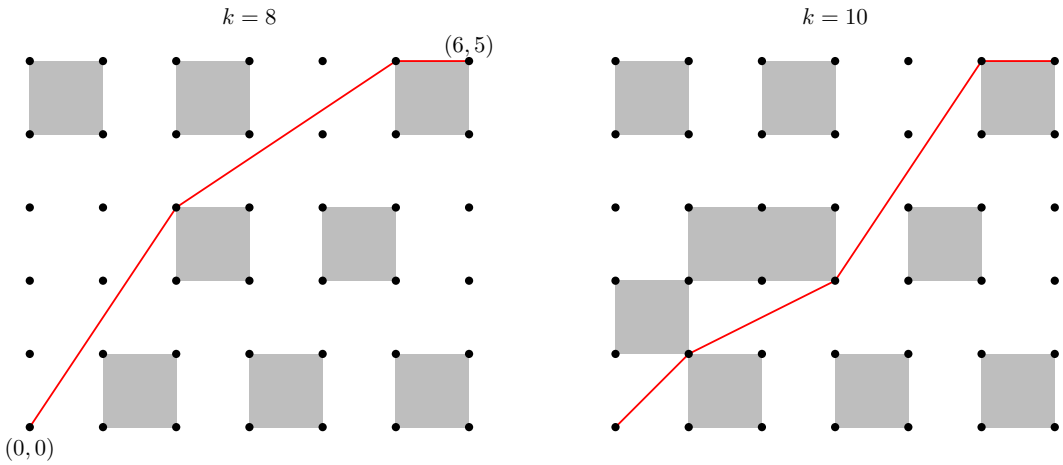
a) Write a program that randomly generates k unit squares in a grid of size $m \times n$ with g as the grid unit.

   The corners of the squares should coincide with the grid points. The squares are basically obstacles.

   The grid points are the vertices of the graph $G(V, E)$. (p, q) is an edge of the graph if and only if the straight line segment $\overline{pq}$ does not intersect the interior of any square.

   Compute a shortest path, if any, from s to t, where s is the bottom-left grid point, and t the top-right one.

b) Save the output as `spath.svg`, as shown in the figure.



$L = 1 + 2\sqrt{13} \approx 8.211$ $L = 1 + \sqrt{2} + \sqrt{5} + \sqrt{13} \approx 8.256$

6 Tiling

Omino means a unit square, whereas domino means two ominoes, i.e., a rectangle of size 1×2 or 2×1.
A polyomino or n-omino is basically an axis-parallel polygon comprising n ominoes. Given an n-omino, the task is to represent it as the smallest collection of dominoes and ominoes.

User input: integers $m, n (n \leq m^2)$.

Use the following steps for coding:

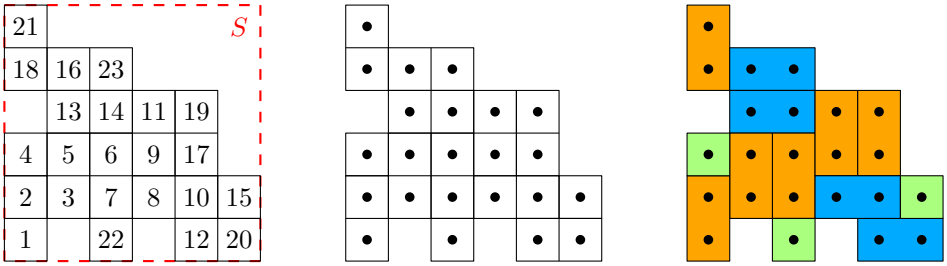
a) Generate a random n-omino in a square S of size $m \times m$ omnioes. The size of an omino may be 15×15 or so, and its center should be marked by a dot.

You can place the first omino at the left-bottom place of S, and then place each new omino at some random place inside S so that the new omino shares an edge with an existing omino.
Iterate until you get n ominoes.
In figure (a), $n = 23$, and the ominoes are numbered in the order as generated.

b) Use maximum matching in a bipartite graph to find the maximum number of dominoes that can cover the n-omino as much as possible.
Color the dominoes and the leftover ominoes as shown in figure (c).

The graph $G(V, E)$ has $V = \{\text{ominoes}\}$ and
$E = \{(u, v) : \text{omnio } u \text{ and omino } v \text{ have a common edge}\}$.
Since this is a subgraph of a graph representing a chequer-board, it is 2-colorable and hence bipartite.

c) Save the input n-omino as `polyomino1.svg`, `polyomino2.svg`, and the output as `tiling.svg`, as shown in figures (a–c).



a b c

7 Maximum line matching

a) Write a program to randomly generate m horizontal straight line segments and n vertical straight line segments with the following constraints:

   i) The endpoints should be random integers in multiples of 20 and should lie inside a rectangle R of width 1000 and height 800.

   ii) The endpoints of each segment should be distinct, i.e., each segment should have positive length.

   iii) Two horizontal segments should not overlap or share an endpoint.

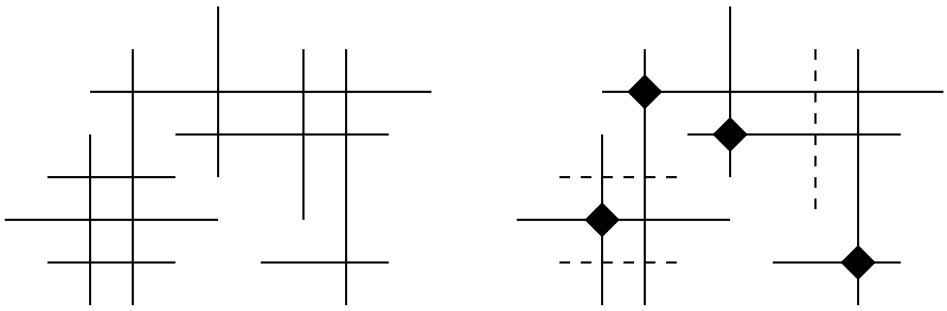
   iv) Two vertical segments should not overlap or share an endpoint.

   v) A horizontal segment and a vertical segment should not share an endpoint.

   Store the values of m, n, and the coordinates of all endpoints in a meaningful order in a txt file named `hvLines.txt`.

   User input: m, n.

b) Write another program that reads the file `hvLines.txt`, computes the intersection among the line segments, find the maximum matching in which each pair comprises a horizontal and a vertical segment. Save the final result as an svg file named `pairs.svg`, with a diamond centered at the intersection point of each pair, and showing the unmatched segments as dashed.



input output

line segments can intersect at interior points only

8 2-color intersection

a) Write a program to randomly generate m horizontal straight line segments and n vertical straight line segments with the following constraints:

   i) The endpoint coordinates should be random integers in multiples of 20 and should lie in $[20, 1000]$.

   ii) The length of each segment should be 80.

   iii) Two horizontal segments should not overlap or share an endpoint.

   iv) Two vertical segments should not overlap or share an endpoint.

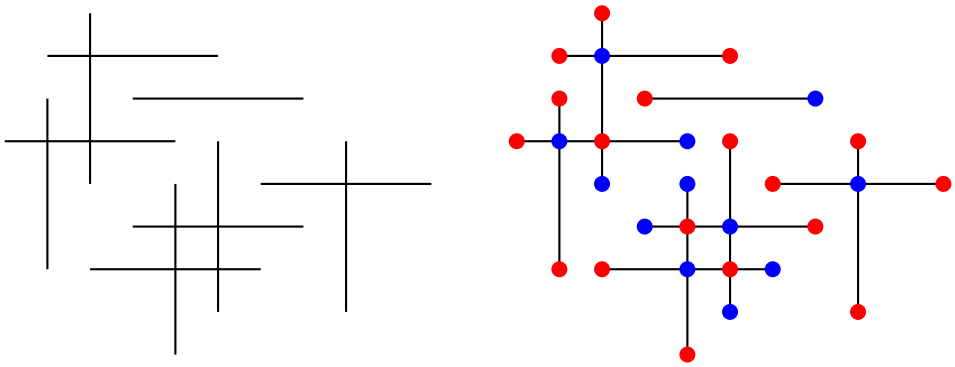
   v) A horizontal segment and a vertical segment should not share an endpoint.

   vi) An intersection point cannot be any endpoint.

   Store the values of m, n, and the coordinates of all endpoints in a meaningful order in a txt file named `hvLines.txt`.

   User input: m, n.

b) Write another program that reads the file `hvLines.txt`, computes the intersection among the line segments, and color the intersection points and endpoints using two colors (red and blue) such that no two consecutive points on the same segment get the same color.
   Save the final result as an svg file named `intersect.svg`.



input output