Minimum Steiner Tree

Sprinternship 2025

Lambert Ike, Ishita Kumari, Viswa Kotra, Arohan Shrestha, Julia High

cadence

Introductions

Julia High
Human Biology

Arohan Shrestha
Electrical Engineering

Ishita Kumari
Computer Science

Lambert Ike
Mechanical Engineering

Viswaretas Kotra
Computer Science

Overview - Project Goal

- Over the past 2 weeks, we worked to implement a steiner tree that connects all the given input circuit nodes and build further optimizations on it.

- Our objective was to minimize the total wirelength of the tree that connects all the nodes return a list of points, including steiner points and list of edges that create the tree with minimum length.

- After our success with that implementation, we were given a special objective where we were tasked with handling blockages, which required obstacle avoidance.

Overview - Steiner Tree

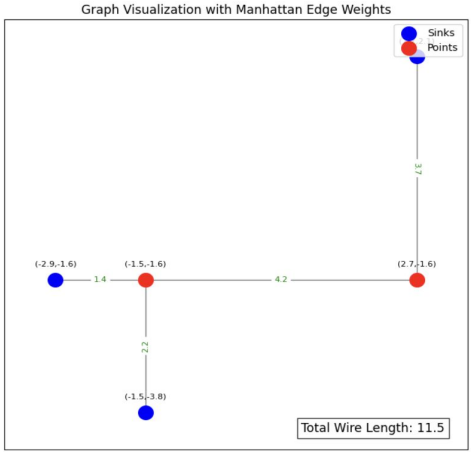
- **Steiner Tree:** A tree that connects input nodes (sinks) with the shortest total wirelength by adding Steiner points to optimize the path.
- **Sinks:** Input nodes that must be connected in the tree.
- **Hanan Grid:** A grid formed by vertical and horizontal lines through each sink, used to identify optimal Steiner point placements.
- **Manhattan Distance:** The distance between two points measured along horizontal and vertical paths (L1 norm).
- **Steiner Points:** Additional points added to reduce the overall wirelength.
- **Objective:** Minimize the total length of edges while connecting all sinks.

Blockage Addition

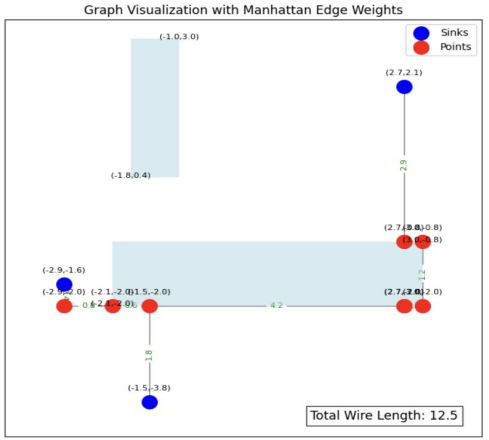
Task: Find a minimum rectilinear steiner tree which avoids rectangular blockages.

Application:

In IC design, chips have many obstacles such as pre-routed wires, power networks, etc. Need to find optimal path around these blockages to minimize wire length saving delay times and cost of chip.



Original Task



Blockage Addition

Applications of this algorithm in different fields

- Bioinformatics
  - The usage of C++, Python, and other software programs allows for the sequencing and annotation of the actual genome to help further observe mutations that occur.
  - There are biological databases that are stored within these programs of the human DNA and annotations of the amino acid sequences.
- Model systems
  - Implementation of code creates the ability to be able to practice whole-cell remodeling.
  - There is also the usage of coming up with new drug discoveries due to retrieving the human genome sequence stored in libraries of softwares
- Telecommunications (ex. data streaming path sending to multiple nodes)
- Transportation infrastructure planning (ex. road length and routes to locations)

Approaches to tackle the problem

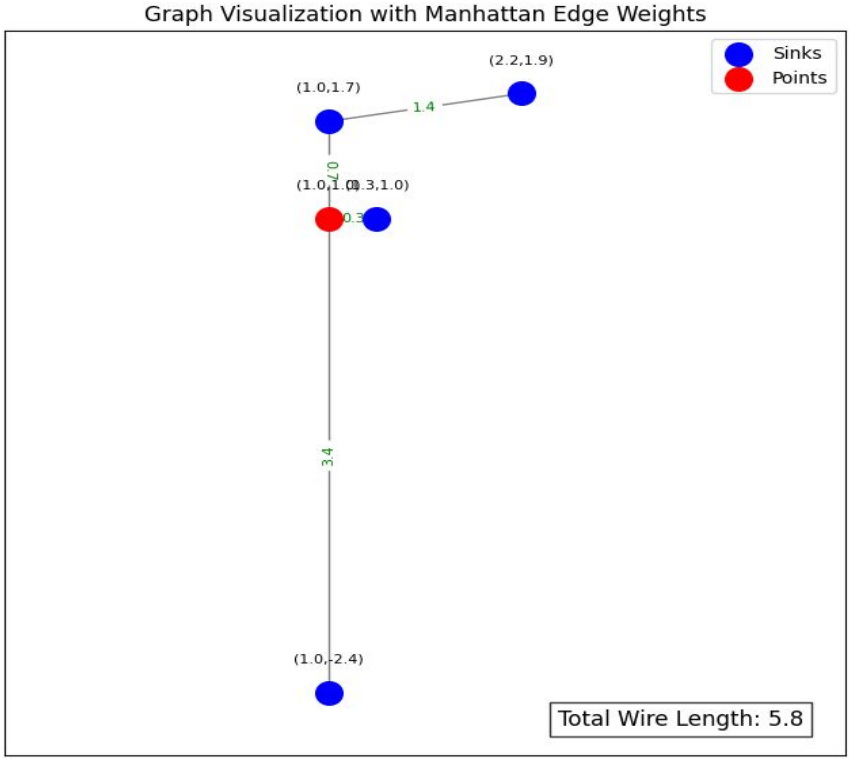
- Iterative steiner point addition + post processing (1st Approach)
- Dynamic Programing Approach
- Obstacle-Avoiding Rectilinear Steiner Minimal Tree (OARSMT)

Lin, C.-W., Chen, S.-Y., Li, C.-F., & Chang, Y.-W. (2008, April). Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs. https://cc.ee.ntu.edu.tw/~ywchang/Papers/tcad08-Stree.pdf

1st Approach

AddSteiner Function

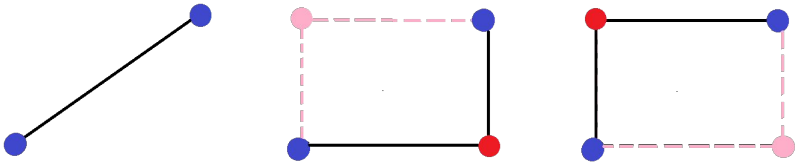
- Computes the initial MST
- Generate the Hanan grid based on the sinks points.
- Initialize a variable for the current tree cost.
- Iteratively add steiner points until no significant improvement



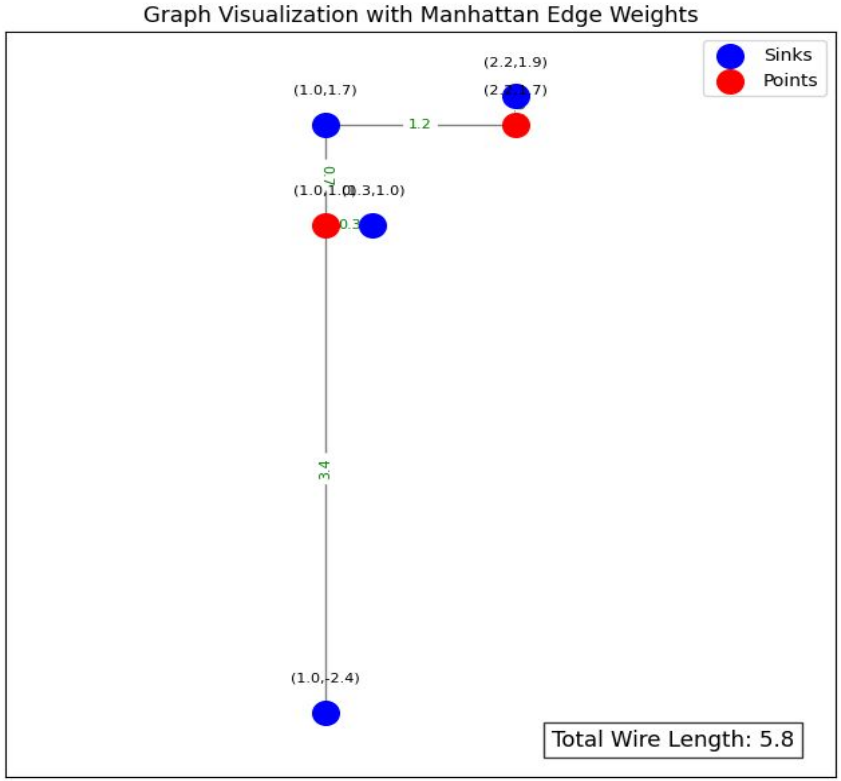
1st Approach

Post-process function

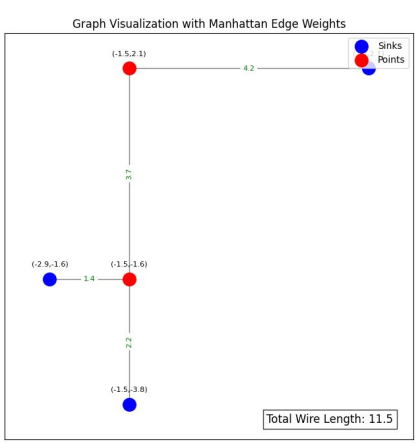
- An intermediate steiner point is added for all diagonal edges.
- Possible “L” points

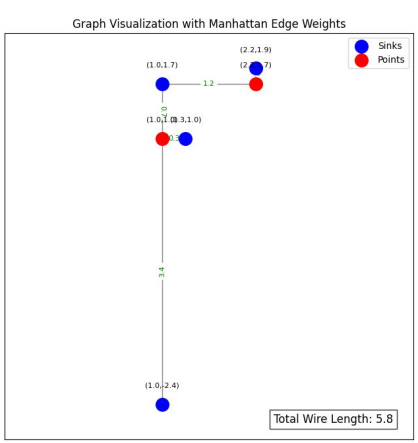


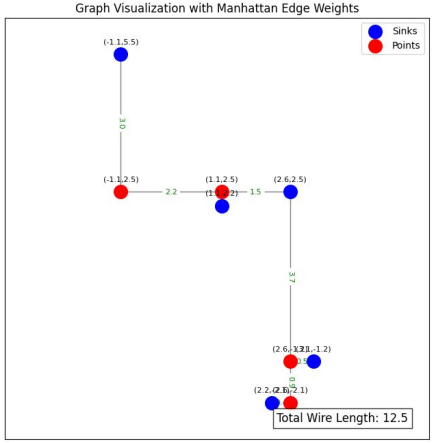
- Any of the intermediate points can be chosen since pathLength is the same.

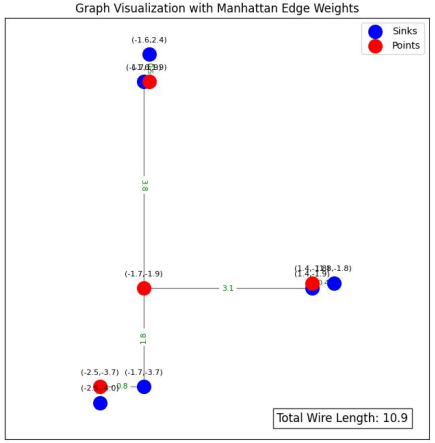


Run Times









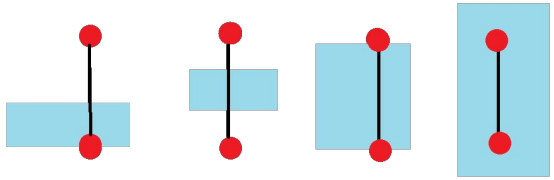
File	r31.in	r42.in	r53.in	r62.in	100.in	1000.in
Time	.038 s	.049 s	.112 s	.096 s	676.63 s	DNF

(11 mins and 16s)

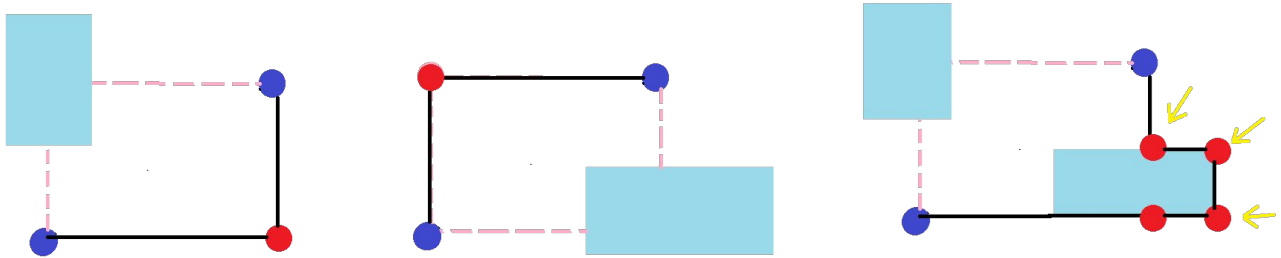
1st Approach

Extension of Post-process to handle blockages

- Adding edge cases to handle what to do when an edge is blocked.
- This way is not feasible for blockage handling since there are too many edge cases, making the approach unscalable.



Ways edges can be blocked



Blockage situations

1st Approach

Pros:

- Straightforward to implement
- Works for smaller test cases

Cons:

- Runtime expensive
- Issues in handling blockages

2nd Approach - Dynamic Programming (DP)

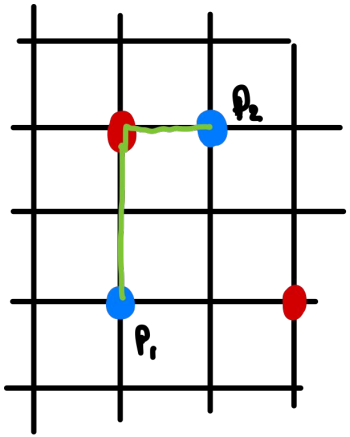
- Decrease run time by strategically adding Steiner points.

DP Table:

- Key: Subset of points (bitmask).
- Value: Minimum MST length + Steiner point ID.

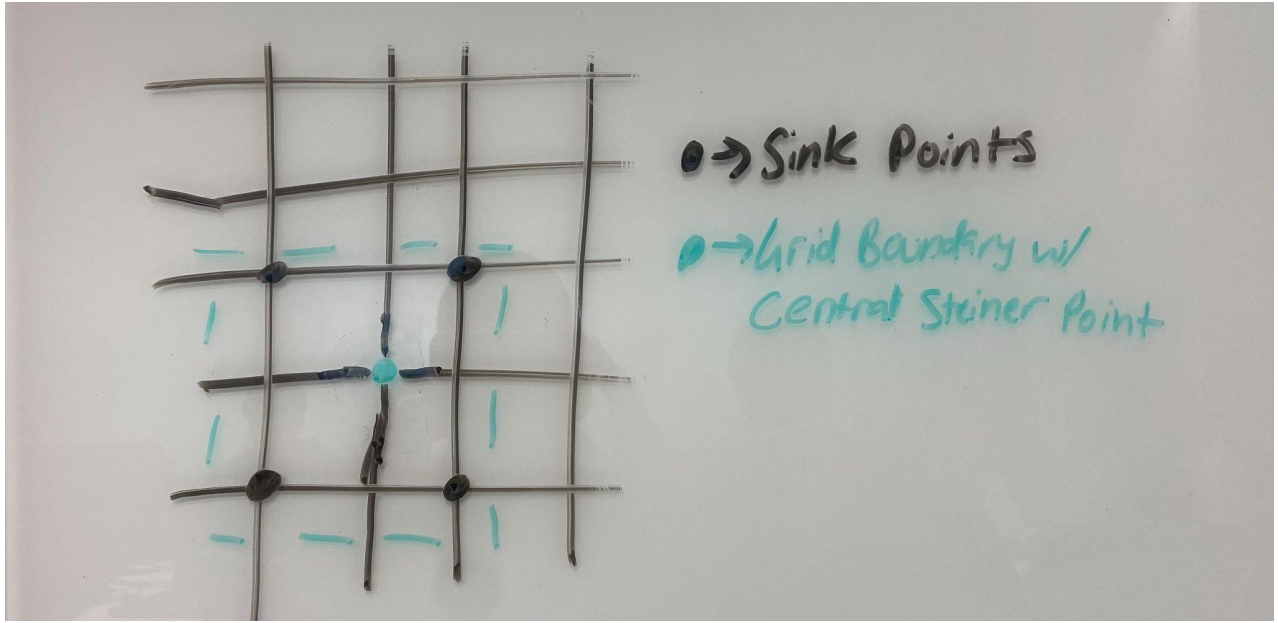
Steps:

1. Initialize Table: Start with input sinks.
2. Base Case: Compute MST with Prim's algorithm if no Steiner points.
3. Recursive Computation: Calculate MST for subsets; use memoization.
4. Final Config: Trace back to find optimal Steiner points.



$DP[P_1, P_2] \simeq$ MST Length + Steiner Point ID

DP Approach Improvement



Grid subsections with centralized steiner point

3rd Approach - Obstacle-Avoiding Rectilinear Steiner Minimal Tree (OARSMT)

Steps for Construction

1. OASG Construction:
   - Combine sinks and obstacle corners.
   - Create edges avoiding obstacles.
2. OAST Construction:
   - Calculate edge weights (Manhattan distance).
   - Use Kruskal's/Prim's algorithm.
3. Transform to OARST:
   - Replace diagonals with horizontal/vertical edges.
   - Add Steiner points for shorter paths.
4. Refine to OARSMT:
   - Eliminate overlapping edges.
   - Ensure no edge intersects obstacles.

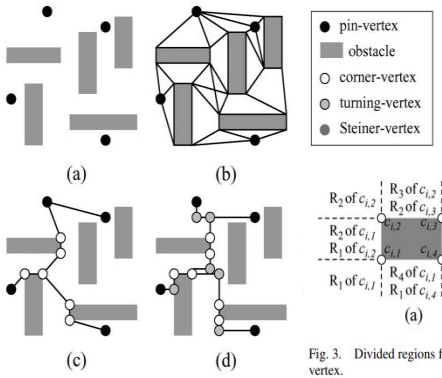


Fig. 2. (b)–(e) Four steps for OARSMT construction

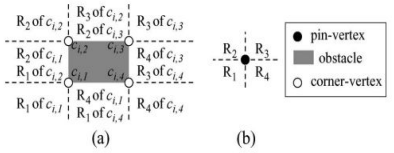


Fig. 3. Divided regions for (a) each corner vertex of an obstacle and (b) a pin vertex.

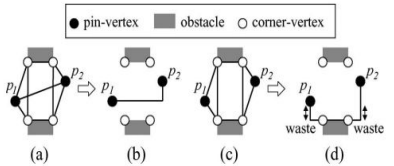
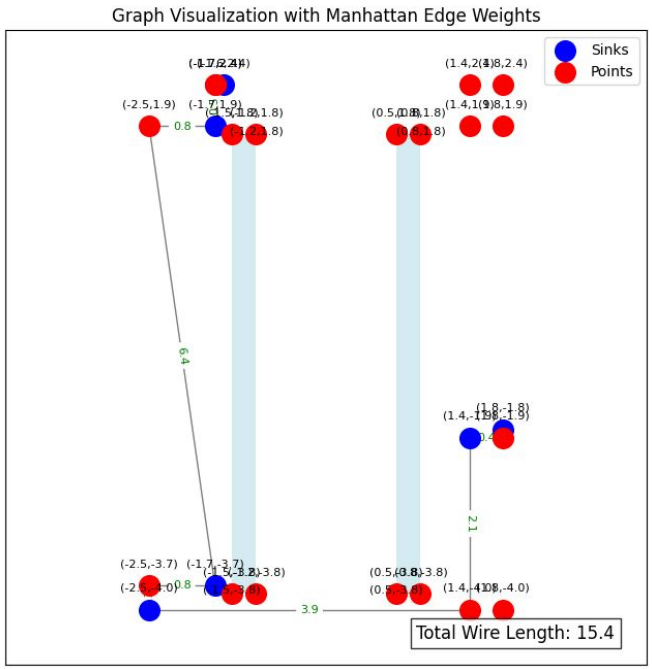


Fig. 4. Comparison between our OASG and that of Shen *et al.* (a) Our OASG has the edge (p_1, p_2) and (b) results in an optimal rectilinear connection. (c) The OASG of Shen *et al.* does not contain the edge and (d) results in two wasted segments.

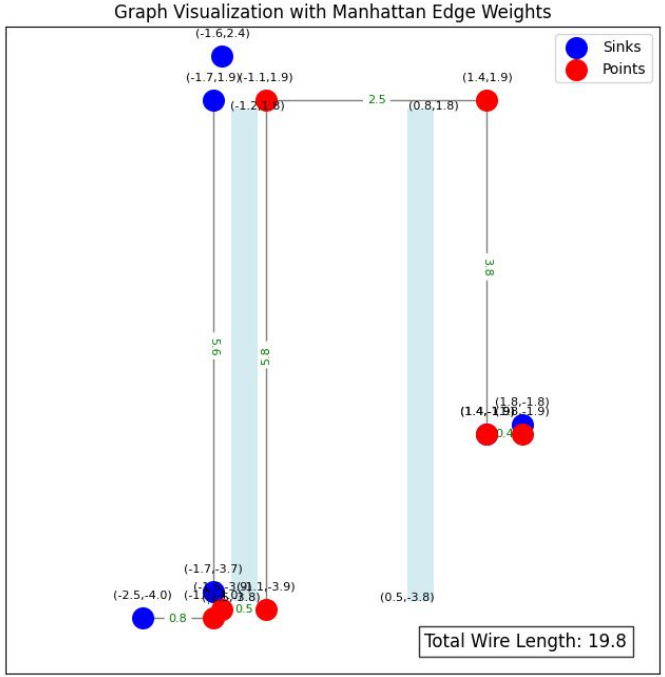
Definition 4: A vertex $f \in P \cup C$ is a neighbor of a vertex $v \in P \cup C$ if no other vertex in $P \cup C$ or obstacle is inside or on the boundary of the bounding box of v and f.

Lin, C.-W., Chen, S.-Y., Li, C.-F., & Chang, Y.-W. (2008, April). Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs. https://cc.ee.ntu.edu.tw/~ywchang/Papers/tcad08-Stree.pdf

OARSMT Implementation Attempts



First Attempt - Only Edge Connections



Second Attempt - Full Tilt implementation

OARSMT Second Attempt Pseudocode

Struct Definitions:

- **Point Struct:**
  - ID, coordinates (x, y), sink flag.
- **Edge Struct:**
  - Endpoints (node1, node2), weight.
- **Block Struct:**
  - ID, corner coordinates (x1, y1) and (x2, y2).

Class: SteinerTree

- **Private Members:**
  - File path, points, edges, blockages, counts of sinks/blocks.
- **Private Methods:**
  - `doLinesIntersect()`: Check line segment intersection.
  - `doesEdgeIntersectBlock()`: Check edge blockage.
  - `isEdgeBlocked()`: Verify edge blockage.
  - `findPathAroundBlockage()`: Create detoured path.
  - `obstacleAvoidingSpanningGraph()`: Construct OASG.
  - `computePrimMST()`: Calculate MST using Prim's.

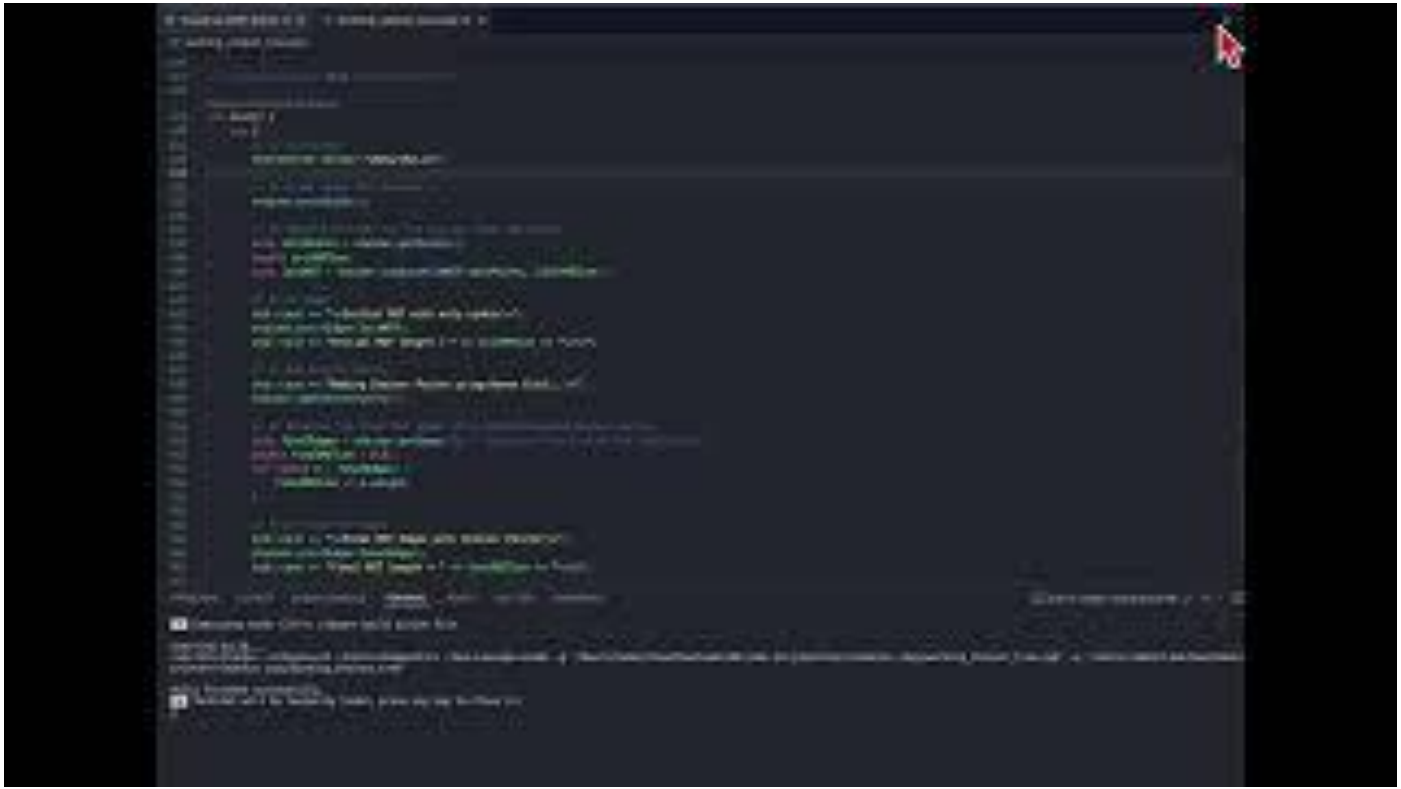
Public Methods:

- **Constructor:**
  - Initialize from input file.
- `readInputFile()`: Parse input.
- `printSinks()`: Display sink info.
- `displayBlockages()`: Show blockages.
- `mhDistance()`: Calculate Manhattan distance.
- `findPointByID()`: Locate point by ID.
- `obstacleAvoidingRectilinearSteinerTree()`: Compute OASG and MST.
- `addSteinerPointsToMST()`: Update MST, add Steiner points.
- `postProcess()`: Align edges.
- `writeOutputToFile()`: Output final tree.

Main Function:

- Create `SteinerTree` object.
- Print sinks and blockages.
- Compute rectilinear Steiner tree:
  - Calculate MST, add Steiner points, align edges.
- Write output.
- Handle exceptions.

Implementation on VS Code



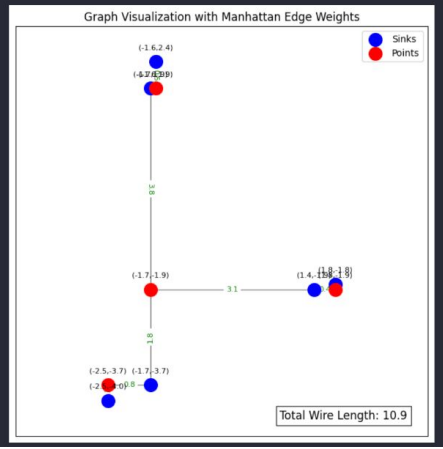
Implementation on VS Code

Solution output

```
# File path to the .out file
file_path = "data/r62.out"

# Read the .out file
sinks, blockages, points, edges = read_out_file(file_path)


# Print the parsed data
print(f"Sinks: {sinks}")
print(f"Blockages: {blockages}")
print(f"Points: {points}")
print(f"Edges: {edges}")

# Plot the graph
plot_graph(sinks, points, edges, blockages=blockages)
```

0.3s Python

```
Sinks: [(6, 1.8, -1.8), (26, -1.6, 2.4), (14, 1.4, -1.9), (15, -1.7, 1.9), (0, -2.5, -4.0), (27, -1.7, -3.7)]
Blockages: []
Points: [(107, -1.7, -1.9), (1000, 1.8, -1.9), (1001, -1.6, 1.9), (1002, -2.5, -3.7)]
Edges: [(6, 1000), (15, 1001), (0, 1002), (14, 1000), (26, 1001), (27, 1002), (27, 107), (14, 107), (15, 107)]
```

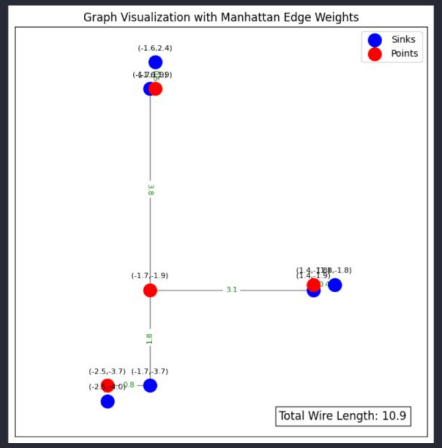


Our output

```
#-File path to the .out file
file_path = "steiner_tree_output.txt"

#-Read the .out file
sinks, blockages, points, edges = read_out_file(file_path)


#-Print the parsed data
print(f"Sinks: {sinks}")
print(f"Blockages: {blockages}")
print(f"Points: {points}")
print(f"Edges: {edges}")

#-Plot the graph
plot_graph(sinks, points, edges, blockages=blockages)
```

2.8s Python

```
Sinks: [(6, 1.8, -1.8), (26, -1.6, 2.4), (14, 1.4, -1.9), (15, -1.7, 1.9), (0, -2.5, -4.0), (27, -1.7, -3.7)]
Blockages: []
Points: [(1006, -1.7, -1.9), (20014, 1.4, -1.8), (20000, -2.5, -3.7), (20026, -1.6, 1.9)]
Edges: [(14, 20014), (20014, 6), (1006, 14), (27, 1006), (0, 20000), (20000, 27), (15, 1006), (26, 20026), (20026, 15)]
```



Key Takeaways

- Learned about Prim's, Kruskal's, Hanan grid, Manhattan distance, dynamic programming, and their implementation in C++ to produce a optimal solution.

- Participated in team meetings, resume workshops, brainstorming sessions, and code reviews to enhance project outcomes.

- Thank you to everyone at Cadence, especially our mentors! We couldn't have done it without your support! :)

References/links

GitHub:

https://github.com/lambert-ike-1232/Sprintern-Cadence

Papers:

https://cc.ee.ntu.edu.tw/~ywchang/Papers/tcad08-Stree.pdf

http://users.ece.northwestern.edu/~haizhou/publications/zhu05tcad.pdf