Data Structures:

NoProperty:

1. 代表沒有任何資料的型態。

EdgeProperty<EdgeData>:

1. EdgeData: 代表edge的資料，可以是任何型態。

VertexProperty<VertexData>:

1. VertexData: 代表vertex的資料，可以是任何型態。

Graph<VertexData, EdgeData>:

1. VertexData: 代表vertex的資料，可以是任何型態。
2. EdgeData: 代表edge的資料，可以是任何型態。
3. outEdgesList, inEdgesList: 用來儲存vertex的outEdges和inEdges。
4. vertexPropertiesMap, edgePropertiesMap: 用來儲存vertex, edge的資料。

Coordinates<Data>:

1. Data: 代表vertex的資料，可以是任何型態。
2. x, y: 儲存了vertex的座標。

SEQPairGraph:

1. 繼承自Graph<Coordinates<int> \*, NoProperty>
2. Coordinates<int>\* 用來儲存每個vertex有座標(x, y)和一個int值，代表這個vertex的值。
3. Edge沒有而外的資訊，僅儲存target以及weight。
4. vector<int> seqX, seqY; 是兩個lut，使用coordinate查詢vertex的index。

Macro:

1. IO時查表使用，內有name, width, height。

Scheduler:

1. 定義SA如何運作，包含了一些參數。
2. start, end time的計算。
3. numNodes, k, temperature, coolingRate等SA參數。
4. run, isImproving flags。
5. previousMove, previousIndices 用來保存上一步的資訊，如果reject需要回復。
6. enum Moves, NUM\_MOVES 提供switch case使用。

SimulatedAnnealing:

1. temperature, coolingRate, absoluteTemperature, iterations 為SA必要參數。
2. logFile為ofstream，用來寫log。

Algorithms:

TopologicalSort:

1. Initialize a container to store the topological order
2. Repeat until all vertices are visited(dfs):
   1. Select a vertex has not been visited, mark it as visited
   2. For each outgoing edge from current vertex, if the destination vertex is not visited, recursively visit the vertex
   3. Add the current vertex to the topological order

LongestPath:

1. Initialize the distance array with -inf
2. Repeat until the stopping criterion is met:
   1. Select a vertex by topological order
   2. Calculate the distance of the vertex
   3. Update the distance array if the distance of the vertex is greater than the current distance.

SA:

1. Initialize the current state
2. Repeat until the stopping criterion is met:

Repeat until the stopping criterion is met:

* + 1. Select a candidate state
    2. Calculate the cost of the candidate state
    3. If the cost of the candidate state is less than the cost of the current state, accept the candidate state
    4. If the cost of the candidate state is greater than the cost of the current state, accept the candidate state with probability
    5. if c or d is accepted, update the current state

1. Update the temperature

SEQPairGraph:

1. Initialize vertices with coordinates [(i, i) for i in range(n)]
2. for all pairs of vertices (v1, v2) in the graph check if v1.x < v2.x and v1.y < v2.y. If true, add an edge from v1 to v2
3. There is 3 types of operations:
   1. swapX: swap the x coordinates of two vertices
   2. swapY: swap the y coordinates of two vertices
   3. swapBoth: swap the x and y coordinates of two vertices

input of them is a pair of position. it'll be converted to vertices using seqX, seqY. After the operation, the graph edges, seqX, seqY will be updated.