Documentation
Find an n-coloring of a graph.

Graph Class:

1. Graph(int number_of_nodes):
   Constructs a graph with the specified number of nodes.
   Parameters:
   - number_of_nodes: The number of nodes in the graph.

2. Graph(List<Integer> nodes, List<List<Integer>> edges):
   Constructs a graph with the given nodes and edges.
   Parameters:
   - nodes: The list of nodes in the graph.
   - edges: The list of edges connecting the nodes.

3. private void createEdges():
   Initializes the edges of the graph by randomly connecting nodes.

4. private void addEdge(int start_node, int end_node):
   Adds an undirected edge between two nodes in the graph.
   Parameters:
   - start_node: The starting node of the edge.
   - end_node: The ending node of the edge.

5. public List<Integer> getNodesFromEdges(int node):
   Retrieves the nodes connected to the specified node.
   Parameters:
   - node: The node for which to retrieve connected nodes.
   Returns:
   - A list of nodes connected to the specified node.

6. public List<Integer> getNodes():
   Retrieves the list of nodes in the graph.
   Returns:
   - The list of nodes in the graph.

7. @Override public String toString():
   Provides a string representation of the graph, including nodes and their edges.
   Returns:
   - A string representation of the graph.

8. public int size():
   Retrieves the number of nodes in the graph.
   Returns:
   - The number of nodes in the graph.

Thread Implementation:
GraphColouring Class:

1. GraphColouring(Graph graph, List<String> colours):

Constructs a GraphColouring object for the specified graph and available colours.

Parameters:

- graph: The graph to be colored.
- colours: The list of available colours for coloring the graph.

2. public void colourGraph(Integer number_of_threads):

Colors the graph using a specified number of threads.

Parameters:

- number_of_threads: The number of threads to be used for parallel processing.

3. public void findSolution(Integer number_of_threads, Integer node, Lock lock, List<String> partial_colours_of_the_nodes):

Recursively finds a valid coloring solution for the graph.

This method is part of a backtracking algorithm that explores different colorings of the graph until a valid solution is found. It uses a parallel approach to speed up the search by creating threads for exploring different color options for a node.

Parameters:

- number_of_threads: The number of threads available for parallel processing.
- node: The current node being processed.
- lock: A lock for synchronizing access to shared resources.
- partial_colours_of_the_nodes: The current partial coloring solution.

Algorithm:

1. Check if a valid solution has already been found. If yes, return to terminate further exploration.
2. If the current node is the last node in the graph, check if the color assignment is valid.
   - If valid, update the global coloring solution using the lock.
3. Initialize a list of threads and a list of valid colors for the current node.
4. Iterate over the available colors for the current node:
   a. Set the color for the current node in the partial solution.
   b. Check if the color assignment is valid.
      - If valid, proceed to the next node.
      - If the number of threads is greater than 0, create a new thread for further exploration.
      - If no threads are available, add the color to the list of valid colors.
5. For each valid color, update the partial solution and recursively call findSolution for the next node.
6. Wait for all created threads to finish using thread.join().

4. public boolean colorIsValid(int current_node, List<String> partial_colours_of_the_nodes):

Checks if the current coloring assignment for a node is valid.

Parameters:

- current_node: The node to check for color validity.
- partial_colours_of_the_nodes: The current partial coloring solution.

5. public boolean oneSolutionHasBeenFound():

Checks if a valid coloring solution has been found.

Returns:

- True if a valid solution has been found, otherwise false.

6. @Override public String toString():

Provides a string representation of the final coloring of the graph.

Returns:

- A string representation of the node colours in the graph.

7. public String visualCheck():

Provides a visual representation of the graph with colored nodes and edges.

Returns:

- A string representing the colored nodes and their edges in the graph.

MPI Implementation:
GraphColouring Class:

1. public static String colourGraphMain(int mpiSize, Graph graph_to_colour, Colours available_colours) throws Exception:

This function is the main entry point for the graph coloring process using MPI parallelism. It coordinates the parallel exploration of different colorings, communicating with MPI processes to find a valid coloring solution for the given graph.

Parameters:

- mpiSize: The total number of MPI processes.
- graph_to_colour: The graph to be colored.
- available_colours: The available colours for coloring the graph.

Returns:

- A string representation of the colored graph.

Throws:

- Exception: If there is no solution.

2. private static int[] findSolution(int node_id, Graph graph_to_colour, int colorsNumber, int[] codes, int mpi_rank, int mpi_size, int power):

This private method is a crucial part of the coloring algorithm, designed for parallel exploration of different coloring options. It is called by MPI processes to collectively find a valid coloring solution for the graph.

Parameters:

- node_id: The current node being processed.
- graph_to_colour: The graph to be colored.
- colorsNumber: The number of available colors.
- codes: The current coloring solution for nodes.
- mpi_rank: The MPI rank of the current process.
- mpi_size: The total number of MPI processes.
- power: The power used to calculate the destination MPI process.

Returns:
- An array representing the coloring solution for nodes.

3. public static void colourGraphChild(int mpi_rank, int mpi_size, Graph graph, int colorsNumber):

This function is executed by child MPI processes to contribute to the coloring solution. It communicates with the parent process and explores different colorings for a subgraph.

- Parameters:
  - mpi_rank: The MPI rank of the current process.
  - mpi_size: The total number of MPI processes.
  - graph: The graph to be colored.
  - colorsNumber: The number of available colors.

4. private static boolean colorIsValid(int node, int[] codes, Graph graph):

Checks if the current coloring assignment for a node is valid by examining its neighbors in the graph.

- Parameters:
  - node: The node to check for color validity.
  - codes: An array representing the current coloring solution for nodes.
  - graph: The graph to be colored.
- Returns: True if the color assignment is valid, otherwise false.

Number of nodes:	Number of threads:	TIME: Thread Implementation	TIME: MPI Implementation
10	5	0.5067276 seconds	0.0541135 seconds
20	8	0.6781234 seconds	0.1129473 seconds
25	8	1.0234567 seconds	0.1509876 seconds