The minimum cut is the smallest number of edges whose removal disconnects the graph into two parts.

Step-by-Step Process (Matching the Diagram)

1. Take Edge Input
    - The program reads the number of vertices n and edges m.
    - All edges (u, v) are stored in a vector.
2. Initialize Disjoint Sets
    - Each vertex starts as its own parent (separate set)
    - A size array is used for efficient union operations.
3. Random Edge Selection
    - A random edge (u, v) is selected from the edge list
    - This matches the "for each random edge → find (u, v)" step in the picture.
4. Find Super Nodes
    - Using find_set(), the algorithm finds the current super-nodes of u and v
    - These represent contracted components
5. Contract the Edge
    - If u and v belong to different components:
        - They are merged using union_sets()
        - The number of vertices is reduced by 1
    - This corresponds to creating a *super node (n+1)* and connecting edges to it.
6. Ignore Self-Loops
    - If both endpoints belong to the same component, the edge is skipped.
    - This matches the "remove (u–u) edge" logic in the diagram.
7. Repeat Until Two Super Nodes Remain
    - The contraction continues until only *two components* are left.
    - This matches the "break if edge vector size is 1 / vertices = 2" step

8. Count Crossing Edges
    - All remaining edges that connect the two super-nodes are counted
    - This count represents the *cut size*.
9. Repeat Multiple Times
    - Since the algorithm is randomized, it runs *100 times*
    - The minimum cut found across all runs is selected as the final answer.

Final Output

- The program prints the *minimum cut value* found after multiple iterations
- Repetition increases the probability of finding the true minimum cut

Conclusion

This implementation follows the exact idea shown in the picture:

- Random edge selection
- Super-node creation
- Edge contraction
- Removal of self-loops
- Repetition for accuracy