```
}
```

For the approximation solution we are using a greedy approach. First, we will take the vertex with the highest degree and select this vertex. All ties of vertices with the same degree will be resolved using random picking. We will then send this possible solution to the np-verifier. If the verifier returns false, we select the next highest degree.

Runtime Big O Analysis:
Computing the degree of each vertex involves iterating through all vertices and their neighbors, which takes O(V + E) time in the worst case, where V is the number of vertices and E is the number of edges. The main find_min_cover function calls remove_highest_degree and check_edges repeatedly until all edges are removed. In the worst case, it performs this process once for every vertex. Each call to remove_highest_degree takes O(V + E), and the number of iterations is the number of vertices (V). Therefore, the overall complexity for find_min_cover: O(V * (V + E)) = O(V^2 + V * E).

Pseudocode:

```
find_min_cover (adj_matrix) {
  result = set()

  new_adj, removed_vertex = remove_highest_degree(adj_matrix)

  result.add(removed_vertex)
  if (check_edges(new_adj)) {
    return result
  } else {
        result.add(find_min_cover(new_adj))
        return result
  }
}

remove_highest_degree(adj_matrix) {
        // This will return a tuple, the first element will be the removed vertex
and the second element will be
        // the remaining adj_matrix without the vertex and edges associated with it.
It will utilize randomness to
        // break ties in degree

}

check_edges(adj_matrix) {
        // this will first check if there are any edges left in the adj_matrix
        // True if there are no edges
        // False otherwise
}
```

```
main () {
        // take input
        // construct adj_matrix
        // adj_matrix =
        // {
        //        a : [b, c]
        //        b : [a, c]
        //    c : [a, b, d]
        //    d : [c]
        // }

        // call find_min_cover(adj_matrix)
        // verify with np_verifier(adj_matrix, find_min_cover(adj_matrix))
}
```