1) Summarize your answers for today's exercise.

```
// TODO: 1. Set the arguments in order to start DFS.
dfs_visit(g, vertex_states, g->vertex_starting_search);

// TODO: 2. Change the current vertex state.
vertex_states[current_vertex] = VISITED;

// TODO: 3. If the i-th vertex is not visited yet and the current vertex is connected to it, visit the i-th vertex recursively.
if (vertex_states[i] == UNVISITED && g->adjacent_matrix[current_vertex][i] == CONNECTED) {
   dfs_visit(g, vertex_states, i);
}

// TODO: 4. Set the argument to start BFS.
// Set start vertex to begin searching.
s_queue_enqueue(g->vertex_starting_search);

// TODO: 5. Set the state of the starting vertex to begin search.
// The starting vertex is in the queue and it state should be visited.
vertex_states[g->vertex_starting_search] = VISITED;

// TODO: 6. If the i-th vertex is not visited yet and the current vertex is connected to it, append it to the queue.
if (vertex_states[i] == UNVISITED && g->adjacent_matrix[current_vertex][i] == CONNECTED) {
   vertex_states[i] = VISITED;
   s_queue_enqueue(i);
}
```

2) Explain the meaning of the results obtained by running the completed program.
Start vertex の値と DFS、BFS で得られた結果が表示される。

3) Describe the breadth-first search algorithm using the same style as shown in the right figure in slide 25.

```
// Initialize all vertices by reseting their states to UNVISITED.
   VertexState vertex_states[MAX_VERTEX_SIZE];
   for (size_t i = 0; i < g->vertex_count; i++) {
      vertex_states[i] = UNVISITED;
   }

   // Initialize the queue used for tracking the search process.
   s_queue_init(NULL);

   // TODO: 4. Set the argument to start BFS.
   // Set start vertex to begin searching.
   s_queue_enqueue(g->vertex_starting_search);
```

```
    // TODO: 5. Set the state of the starting vertex to begin search.
    // The starting vertex is in the queue and it state should be visited.
    vertex_states[g->vertex_starting_search] = VISITED;

    while (s_queue_is_empty() == false) {
        size_t current_vertex = s_queue_dequeue();
        printf("%zu ", current_vertex);

        // Loop for child vertices.
        for (size_t i = 0; i < g->vertex_count; i++) {
            // TODO: 6. If the i-th vertex is not visited yet and the current vertex is connected to it, append it to the queue.
            if (vertex_states[i] == UNVISITED && g->adjacent_matrix[current_vertex][i] == CONNECTED) {
                vertex_states[i] = VISITED;
                s_queue_enqueue(i);
            }
        }
    }
```