BÁO CÁO THỰC HÀNH TUẦN 8

Full name: Đặng Bùi Thế Bảo.

Student ID: 24120261.

1. convertMatrixToList

- **Mục đích:** Đọc file input dưới dạng ma trận kề và chuyển đổi thành danh sách kề.
- **Các bước:**
    1. Mở file, đọc số đỉnh n.
    2. Khởi tạo vector matrix với kích thước n.
    3. Đọc từng dòng dữ liệu, xử lý từng ký tự để chuyển đổi thành danh sách các đỉnh kề.
    4. Trả về danh sách kề dưới dạng vector<vector<int>>.

---

2. convertListToMatrix

- **Mục đích:** Đọc file input dưới dạng danh sách kề, chuyển đổi thành ma trận kề.
- **Các bước:**
    1. Mở file, đọc số đỉnh n.
    2. Khởi tạo ma trận n x n với giá trị 0.
    3. Đọc từng dòng danh sách kề, đánh dấu vị trí 1 trong ma trận theo danh sách.
    4. Trả về ma trận kề.

---

3. isDirected

- **Mục đích:** Kiểm tra đồ thị có hướng hay vô hướng.
- **Thuật toán:**
    - Duyệt qua ma trận, nếu tồn tại một cạnh adj[i][j] = 1 nhưng adj[j][i] != 1 thì đồ thị có hướng.
- **Các bước:**

1. Duyệt qua tất cả cặp đỉnh (i, j).
2. Kiểm tra điều kiện trên.
3. Trả về true nếu có hướng, false nếu vô hướng.

---

4. countVertices

- **Mục đích:** Đếm số đỉnh trong đồ thị.
- **Cách làm:** Trả về kích thước ma trận adjMatrix.size().

---

5. coutEdges

- **Mục đích:** Đếm số cạnh trong đồ thị.
- **Thuật toán:**
  - Đếm tổng số 1 trong ma trận kề.
  - Nếu vô hướng, số cạnh = tổng chia 2 vì mỗi cạnh được tính hai lần.
  - Nếu có hướng, trả về tổng số.

---

6. isCompleteGraph

- **Mục đích:** Kiểm tra đồ thị đầy đủ (Complete Graph).
- **Thuật toán:**
  - Đồ thị đầy đủ có cạnh giữa mọi cặp đỉnh khác nhau.
  - Đường chéo chính phải bằng 0.
- **Các bước:**

1. Duyệt ma trận, kiểm tra:
   - adj[i][i] == 0 với mọi i.
   - adj[i][j] == 1 với mọi cặp i != j.
2. Nếu thỏa mãn, trả về true.

---

7. isBipartite

- **Mục đích:** Kiểm tra đồ thị có phải đồ thị hai phân (bipartite).
- **Thuật toán:** BFS + tô màu 2 màu (0 và 1).

- **Các bước:**
    1. Kiểm tra đồ thị vô hướng.
    2. Dùng queue BFS bắt đầu từ đỉnh chưa được tô màu.
    3. Tô màu đỉnh theo quy tắc khác màu với đỉnh kề.
    4. Nếu phát hiện đỉnh kề cùng màu, trả về false.
    5. Nếu duyệt hết không vi phạm, trả về true.

---

8. isCompleteBipartite

- **Mục đích:** Kiểm tra đồ thị có phải là đồ thị hai phân đầy đủ (complete bipartite).
- **Thuật toán:**
    - Trước hết kiểm tra bipartite.
    - Sau đó kiểm tra mọi đỉnh thuộc tập phân 1 kết nối với mọi đỉnh tập phân 2.
    - Kiểm tra không có cạnh trong cùng một tập phân.
- **Các bước:**

1. Tô màu bipartite như isBipartite.
2. Tạo 2 vector chứa các đỉnh thuộc 2 phần.
3. Kiểm tra tính đầy đủ giữa 2 tập: adj[part1[i]][part2[j]] == 1.
4. Kiểm tra không có cạnh trong cùng phần.

---

9. convertToUndirected

- **Mục đích:** Chuyển ma trận kề có hướng sang ma trận kề vô hướng.
- **Thuật toán:** Đối với mỗi cạnh (i, j), nếu có một chiều, thêm chiều ngược lại.
- **Các bước:**
    1. Duyệt tất cả cặp (i, j).
    2. Nếu adj[i][j] == 1 hoặc adj[j][i] == 1, đánh dấu result[i][j] = 1 và result[j][i] = 1.

---

10. getComplementGraph

- **Mục đích:** Tạo đồ thị bù (complement graph).
- **Thuật toán:** Cạnh trong đồ thị bù tồn tại nếu và chỉ nếu không tồn tại trong đồ thị gốc, trừ đường chéo chính.
- **Các bước:**
    1. Kiểm tra đồ thị vô hướng.
    2. Duyệt tất cả cặp (i, j), với i != j.
    3. Nếu adj[i][j] == 0, đặt result[i][j] = 1 và result[j][i] = 1.

---

11. findEulercycle

- **Mục đích:** Tìm chu trình Euler (nếu có).
- **Thuật toán:** Hierholzer's Algorithm.
- **Các bước:**
    1. Sao chép ma trận kề để không làm thay đổi đồ thị gốc.
    2. Bắt đầu từ đỉnh có cạnh, dùng stack để duyệt.
    3. Di chuyển qua các cạnh chưa đi qua, xóa cạnh khi đi qua.
    4. Nếu không còn cạnh đi tiếp, đẩy đỉnh vào chu trình.
    5. Trả về chu trình.

---

12. dfsSpanningTree

- **Mục đích:** Tạo cây khung DFS từ đỉnh start.
- **Thuật toán:** DFS dùng stack.
- **Các bước:**
    1. Khởi tạo visited, stack.
    2. Duyệt theo DFS, đánh dấu visited.
    3. Thêm cạnh vào cây khung.
    4. Trả về cây khung.

---

13. bfsSpanningTree

- **Mục đích:** Tạo cây khung BFS từ đỉnh start.

- **Thuật toán:** BFS dùng queue.
- **Các bước:** Tương tự dfsSpanningTree nhưng dùng queue.

---

14. isConnected

- **Mục đích:** Kiểm tra hai đỉnh có kề nhau hay không.
- **Cách làm:** Kiểm tra adj[u][v] == 1 hoặc adj[v][u] == 1.

---

15. Dijkstra

- **Mục đích:** Tìm đường đi ngắn nhất trong đồ thị có trọng số dương (ở đây trọng số mặc định là 1).
- **Thuật toán:** Thuật toán Dijkstra dùng priority queue.
- **Các bước:**
    1. Khởi tạo khoảng cách Weight cho tất cả đỉnh là vô cực, trừ đỉnh start.
    2. Dùng priority queue lưu các đỉnh theo khoảng cách hiện tại nhỏ nhất.
    3. Cập nhật khoảng cách cho đỉnh kề.
    4. Lặp đến khi xử lý hết.
    5. Thu lại đường đi ngắn nhất từ end về start.

---

16. BellmanFord

- **Mục đích:** Tìm đường đi ngắn nhất cho đồ thị có thể có cạnh trọng số âm.
- **Thuật toán:** Thuật toán Bellman-Ford.
- **Các bước:**
    1. Khởi tạo khoảng cách.
    2. Lặp n-1 lần, cập nhật khoảng cách cho các cạnh.
    3. Nếu sau n-1 lần còn cập nhật được, có chu trình âm.
    4. Thu lại đường đi tương tự Dijkstra.