**Báo cáo môn: Phân tích thiết kế & Giải thuật.**

**Đề bài:** Với bài toán trên đồ thị và thuật toán Topological Ordering hãy trình bày các nội dung sau:

1. Nêu bài toán;

2. Mô tả chi tiết thuật toán;

3. Đánh giá độ phức tạp thuật toán;

4. Tự xác định 2 bộ dữ liệu (với số phần tử N&gt;=5), với mỗi bộ dữ liệu hãy thực hiện

từng bước thuật toán đã mô tả ở mục 2 và ghi ra kết quả mỗi bước;

5. Viết chương trình trên C/C++

6. Viết báo cáo trình bày các nội dung từ 1-4 và kết quả thực hiện chương trình (5).

*1. Nêu bài toán:*

Trên đồ thị có hướng không có chu trình G(V,E) một thứ tự Tôpô của G là một danh sách của tất cả các đỉnh trên đồ thị G, sao cho với bất cứ hai đỉnh u,v trên G mà có cạnh nối từ u đến v thì u phải đứng trước v trong danh sách.

Ví dụ như: Đối với một công việc cần làm trên công trường thì bài toán sắp xếp topo sẽ là: sắp xếp công việc có liên quan với nhau sao cho mọi công việc được thực hiện thuận lợi theo tiến trình của thời gian cụ thể là việc nào cần làm trước để chuẩn bị cho việc khác thì phải thực hiện trước, chẳng hạn phải xẻ gỗ rồi mới đóng cốt pha,rồi mới trộn bê tông,...

*2.Mô tả chi tiết thuật toán:*

Ý tưởng: Sau khi duyệt DFS trên đồ thị, ta sẽ có một mảng kết quả lưu các đỉnh theo thứ tự thăm xong trước tới sau, mảng này chính là trật tự topo.

Xem xét:

* Tìm kiếm theo chiều sâu trên đồ thị (DFS) có thể dùng giải thuật này để tạo một danh sách tuyến tính các đỉnh của một đồ thị. Có ba cách hiện thực phương pháp này, một trong số đó là duyệt đảo hậu thứ tự (reverse postordering). Với các đồ thị có hướng không có chu trình, cách duyệt đảo hậu thứ tự cho ra một trật tự topo của đồ thị đó.
* Với phương pháp sắp xếp topo bằng DFS, nếu đồ thị không phải là DAG thì ta vẫn có một thứ tự, nhưng ta lại có thế dùng DFS để kiểm tra luôn xem đồ thị có là DAG hay không.
* Chứng minh thứ tự nhận được thỏa mản là một trật tự topo: Giả sử tồn tại một đỉnh u đứng trước một đỉnh v trong mảng kết quả mà có cung đi từ v tới u. Khi đó do quá trình DFS u được thăm trước v, nên khi thăm tới v, ta xét các đỉnh kề với đỉnh v, trong đó u được đánh dấu đã thăm -> đồ thị không phải một DAG -> trái với giả thuyết.

algorithm dfs(u, cur\_time):

u.start\_time = cur\_time

cur\_time += 1

u.status = "in\_progress" (vàng)

for v in u.neighbors:

if v.status is "unvisited":

cur\_time = dfs(v, cur\_time)

cur\_time += 1

u.end\_time = cur\_time

u.status = "done" (xám)

return cur\_time

reversed\_topological\_list = []

algorithm dfs(u, cur\_time):

u.start\_time = cur\_time

cur\_time += 1

u.status = "in\_progress" (vàng)

for v in u.neighbors:

if v.status is "unvisited":

cur\_time = dfs(v, cur\_time)

cur\_time += 1 u.

end\_time = cur\_time

u.status = "done" (xám)

reversed\_topological\_list.append(u)

return cur\_time

Algorithm DFS(v)

// Tìm kiếm theo độ sâu xuất phát từ v.

Input: Đỉnh v chưa được thăm

for(mỗi đỉnh u kề v)

if( u chưa được thăm)

Đánh dấu u đã được thăm;

DFS(u);

Algorithm DFS Traversal(G)

// Đi qua đồ thị G=(V, E) theo độ sâu

for (mỗi v ∈V) Đánh dấu v chưa được thăm;

for (mỗi v ∈V)

if (v chưa được thăm)

Thăm v và đánh dấu v đã được thăm;

DFS(v);

•Thực hiện DFS Traversal trên đồ thị G, thêm lệnh L.append(v) vào cuối hàm DFS(v)

•Đảo ngượcL

*3. Đánh giá độ phức tạp thuật toán*

O(|V|+|E|)

4.

Minh họa:

DFS(a)

DFS(c)

DFS(e)

L = (e)

L = (e, c)

DFS(d)

DFS(f)

L = (e, c, f)

L = (e, c, f, d)

L = (e, c, f, d, a)

DFS(b)

L = (e, c, f, d, a, b)

L = (b, a, d, f, c, e)