Đồ thị có hướng

anhtt-fit@mail.hut.edu.vn
Edited by huonglt-fit@mail.hut.edu.vn

Thuật ngữ

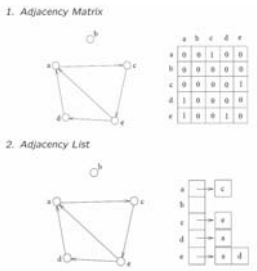
- Đồ thị có hướng
  - 1 đồ thị liên thông nếu và chỉ nếu tồn tại 1 đường đi giữa tất cả các cặp cạnh phân biệt.
- Đồ thị con
  - 1 đồ thị với tập đỉnh và cạnh và tập con của đồ thị gốc
- Các thành phần kết nối
  - 1 thành phần kết nối của 1 đồ thị là đồ thị kết nối tối đa của đồ thị
- Chu trình
  - 1 phần của đồ thị mà bắt đầu và kết thúc tại 1 đỉnh.

Thuật ngữ

- Cây
  - 1 đồ thị G là 1 cây nếu và chỉ nếu nó có kết nối với nhau và không có chu trình.
- Đồ thị có hướng
  - 1 đồ thị mà các cạnh của nó có hướng
- Đồ thị có hướng không chu trình (Directed Acyclic Graph)
  - 1 đồ thị kết nối và không có chu trình (có hướng)
- Indegree và outdegree của 1 đỉnh
  - Indegree = số cạnh nôi đến đỉnh đó
  - Outdegree = số cạnh nối ra khỏi đỉnh đó

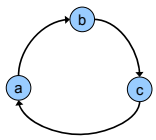
Đồ thị có hướng

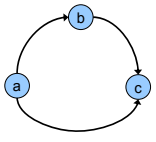
- Đồ thị có hướng có thể biểu diễn như 1 ma trận lân cận hoặc danh sách kết nối, tương tự như đồ thị vô hướng, ngoại trừ:
  - Cạnh (u, v) chỉ tham gia vào 1 đường trong ma trận lân cận hoặc 1 nút trong danh sách kết nối.



Đường dẫn/Chu trình

- 1 đồ thị có hướng có thể chứa các đường và chu trình (đường đi có hướng và chu trình có hướng)
  - Đồ thị trên có đường đi có hướng và chu trình có hướng
  - Đồ thị dưới có đường đi có hướng nhưng không có chu trình có hướng





Duyệt đồ thị

- BFS và DFS có thể dùng để duyệt đồ thị có hướng, giống như đồ thị vô hướng.
- Để kiểm tra tính kết nối của 1 đồ thị
  - Thực hiện BFS hoặc DFS sử dụng 1 đỉnh ngẫu nhiên. Nếu tất cả các đỉnh đã được thăm, đồ thị liên thông. Ngược lại, đồ thị không liên thông

Tìm các thành phần kết nối

- Thực hiện DFS hoặc BFS từ 1 đỉnh
  - Tập các đỉnh đã thăm tạo thành 1 tập kết nối
- Tìm đỉnh khác (i) chưa thăm, thực hiện DFS hoặc BFS với đỉnh đó
  - Có tập kết nối khác
- Lặp lại bước trên đến khi tất cả các đỉnh được thăm
- Thời gian chạy

$$\sum_i O(n_i + m_i) = O(\sum_i n_i + \sum_i m_i) = O(n+m)$$

A complete graph API

- Trên đồ thị API, chỉ quản lý các cạnh. Do đó ta không thể biết có bao nhiêu đỉnh trên đồ thị. Mỗi đỉnh cũng cần có tên để định nghĩa.
- Định nghĩa lại cấu trúc đồ thị để dữ liệu về đỉnh được lưu trong cây như sau:

```
typedef struct {
   JRB edges;
   JRB vertices;
} Graph;
```

Bài 1

- Viết lại cài đặt cho đồ thị có hướng sử dụng API dựa trên cấu trúc dữ liệu mới với các hàm sau:
- Graph createGraph();

```
void addVertex(Graph graph, int id, char* name);
char *getVertex(Graph graph, int id);
void addEdge(Graph graph, int v1, int v2);      them canh v1->v2
void hasEdge(Graph graph, int v1, int v2);      ktra canh v1->v2 ?
int indegree(Graph graph, int v, int* output);  vao tu v
int outdegree(Graph graph, int v, int* output); ra tu v
int getComponents(Graph graph);                 KTRA CON CHU TRINH(DFS)?
void dropGraph(Graph graph);
```

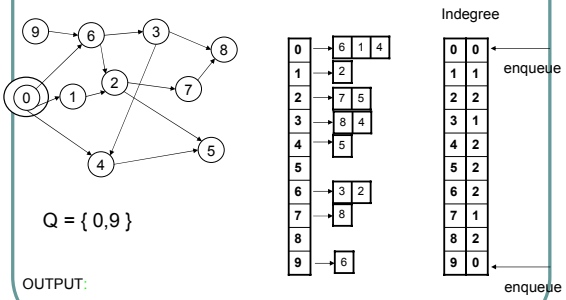
Sắp xếp kiểu topology

- Ta có thể sử dụng hướng trong đồ thị có hướng để biểu diễn 1 quan hệ phụ thuộc
  - COMP104 là điều kiện bắt buộc khi học COMP171
  - Bữa sáng phải ăn trước bữa trưa.
- 1 ứng dụng điển hình là lập lịch thực hiện 1 công việc cần bảo toàn các ràng buộc về trật tự thực hiện các bước, sử dụng thuật toán sắp xếp topology.
  - Coi mỗi nút biểu diễn 1 công việc cần thực hiện. Các công việc có phụ thuộc lẫn nhau. Do đó 1 số việc không thể thực hiện trước khi việc khác hoàn thành.
  - Cho 1 đồ thị không liên thông, đưa ra màn hình 1 trật tự tuyến tính các công việc sao cho các ràng buộc về trình tự đưa ra ở các cung được bảo toàn.
  - Có thể có hơn 1 lịch trình thực hiện

Thuật toán sắp xếp topology

```
Algorithm TSort(G)
Input: đồ thị có định hướng G
Output: trật tự topology của các đỉnh
Khởi tạo hàng đợi Q rỗng
For each vertex v
     do if indegree(v) = 0
           then enqueue(Q,v);
While Q is non-empty
     do v := dequeue(Q);
        output v;
        for each arc(v,w)
            do indegree(w) = indegree(w)-1;
               if indegree(w) = 0
                  then enqueue(w);
```

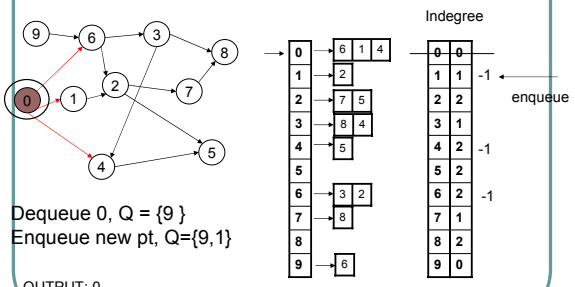
Topological Sort – Example



Q = { 0,9 }

OUTPUT:

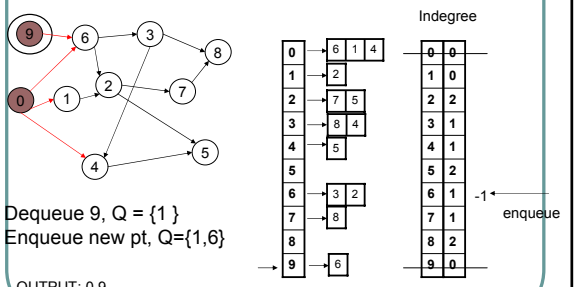
Topological Sort – Example



Dequeue 0, Q = {9 }
Enqueue new pt, Q={9,1}

OUTPUT: 0

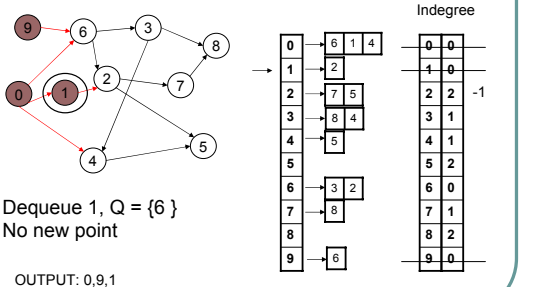
Topological Sort – Example



Dequeue 9, Q = {1 }
Enqueue new pt, Q={1,6}

OUTPUT: 0,9

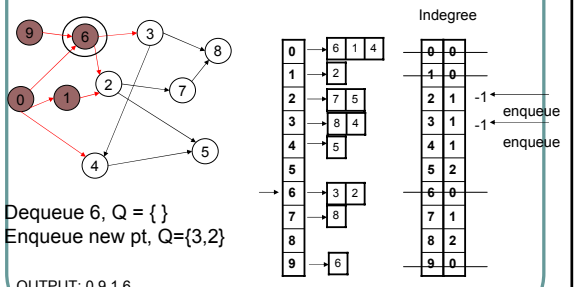
Topological Sort – Example



Dequeue 1, Q = {6 }
No new point

OUTPUT: 0,9,1

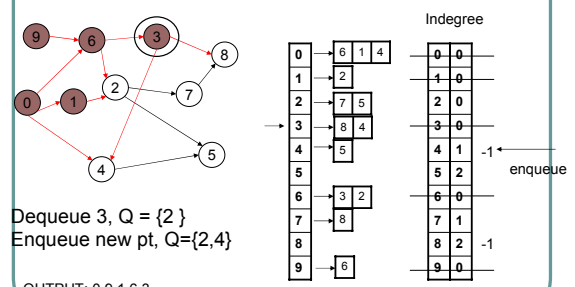
Topological Sort – Example



Dequeue 6, Q = { }
Enqueue new pt, Q={3,2}

OUTPUT: 0,9,1,6

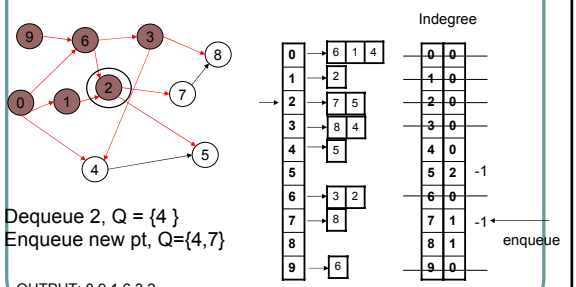
Topological Sort – Example



Dequeue 3, Q = {2 }
Enqueue new pt, Q={2,4}

OUTPUT: 0,9,1,6,3

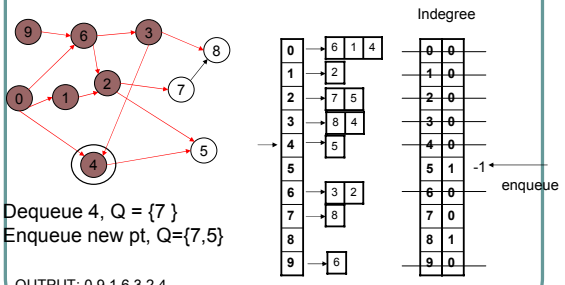
Topological Sort – Example



Dequeue 2, Q = {4 }
Enqueue new pt, Q={4,7}

OUTPUT: 0,9,1,6,3,2

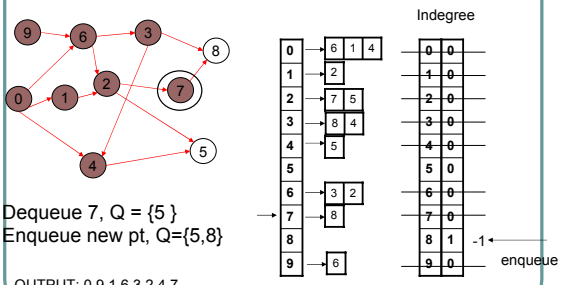
Topological Sort – Example



Dequeue 4, Q = {7 }
Enqueue new pt, Q={7,5}

OUTPUT: 0,9,1,6,3,2,4

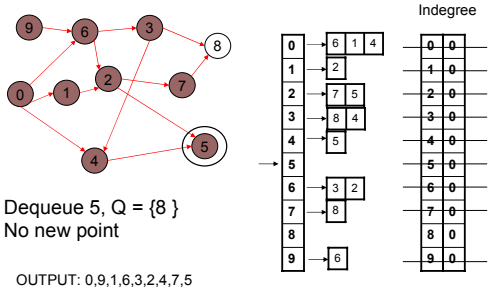
Topological Sort – Example



Dequeue 7, Q = {5 }
Enqueue new pt, Q={5,8}

OUTPUT: 0,9,1,6,3,2,4,7

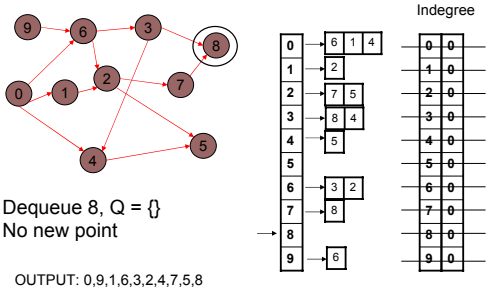
Topological Sort – Example



Dequeue 5, Q = {8 }
No new point

OUTPUT: 0,9,1,6,3,2,4,7,5

Topological Sort – Example



Dequeue 8, Q = {}
No new point

OUTPUT: 0,9,1,6,3,2,4,7,5,8

Bài 2

- Có 1 file mô tả các điều kiện tiên quyết giữa các lớp như sau:

```
CLASS CS140
  PREREQ CS102
CLASS CS160
  PREREQ CS102
CLASS CS302
  PREREQ CS140
CLASS CS311
  PREREQ MATH300
  PREREQ CS302
```

- Sử dụng graph API cuối để viết chương trình sắp trật tự topology của các lớp này