Directed graphs

huydq@soict.hust.edu.vn

Thuật ngữ

- Đồ thị liên thông

  1 đồ thị liên thông nếu và chỉ nếu tồn tại đường đi giữa tất cả các cặp đỉnh.

- Đồ thị con

  1 đồ thị với tập đỉnh và cạnh và tập con của đồ thị gốc

- Các thành phần liên thông

  1 thành phần liên thông của đồ thị là 1 đồ thị con liên thông tối đa của đồ thị gốc

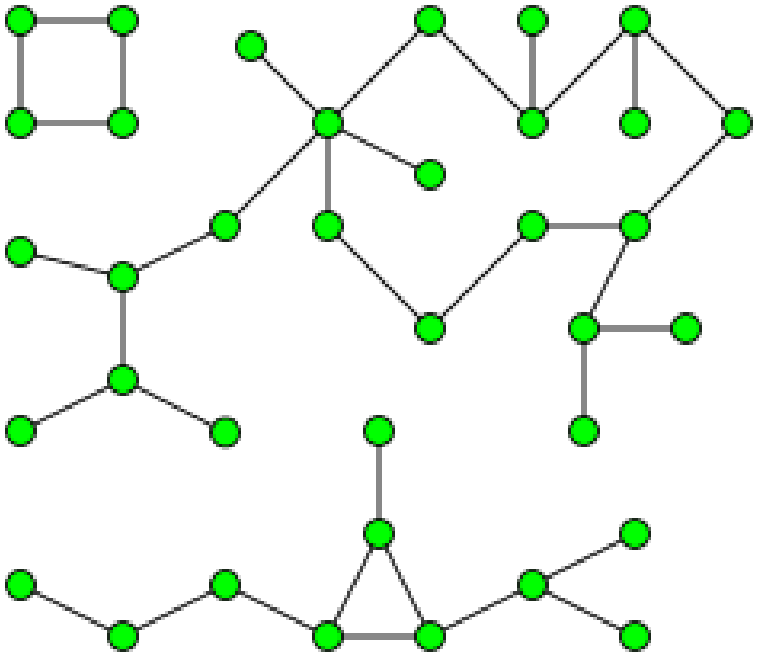
- Chu trình

  1 phần của đồ thị mà bắt đầu và kết thúc tại 1 đỉnh.

Thuật ngữ

- Các thành phần liên thông

  1 thành phần liên thông của đồ thị là 1 đồ thị con liên thông tối đa của đồ thị gốc



đồ thị với ba thành phần liên thông

Thuật ngữ

- Cây

  1 đồ thị G là 1 cây nếu và chỉ nếu nó liên thông 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-DAG)

  1 đồ thị kết nối và không có chu trình (có hướng)

- Indegree và outdegree của 1 đỉnh

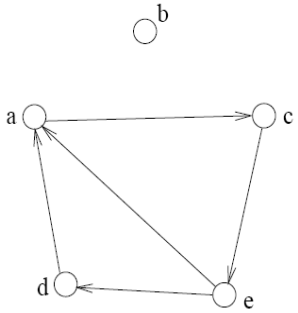
Indegree = số cạnh nối đến đỉnh đó

Outdegree = số cạnh nối ra khỏi đỉnh đó

Đồ thị có hướng

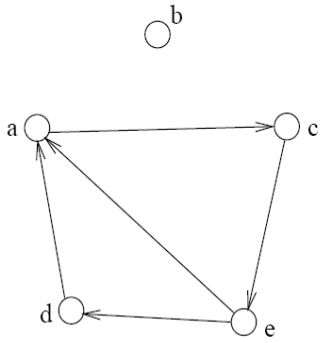
- Đồ thị có hướng có thể biểu diễn như 1 ma trận lân cận hoặc danh sách liên kết, tương tự như đồ thị vô hướng, ngoại trừ:
- Cạnh (u, v) chỉ tham gia vào 1 đường trong ma trận lân cận hoặc 1 nút trong danh sách liên kết.

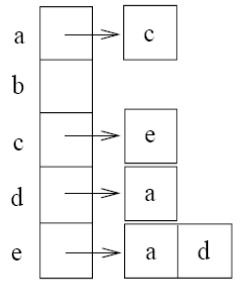
1. Adjacency Matrix



	a	b	c	d	e
a	0	0	1	0	0
b	0	0	0	0	0
c	0	0	0	0	1
d	1	0	0	0	0
e	1	0	0	1	0

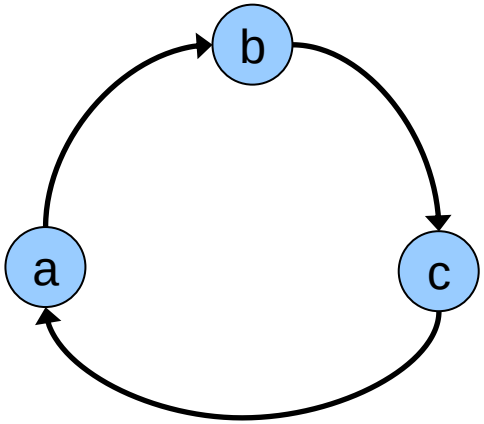
2. Adjacency List



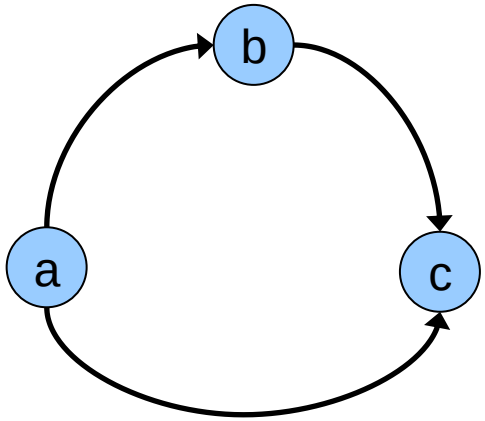


Đường đi/ Chu trình

- 1 đồ thị có hướng có thể chứa các đường đi 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 liên thông 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

API đầy đủ để thao tác với đồ thị

- Các API thao tác với đồ thị hiện tại, 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;
```

API (cont.)

```
Graph createGraph();
void addVertex(Graph graph, int id, char* name);
char *getVertex(Graph graph, int id);
void addEdge(Graph graph, int v1, int v2);
int hasEdge(Graph graph, int v1, int v2);
int indegree(Graph graph, int v, int* output);
int outdegree(Graph graph, int v, int* output);
int DAG(Graph graph);
void dropGraph(Graph graph);
```

Quiz 1

- Cài đặt các hàm API đồ thị dựa trên đặc tả trên.
- Kiểm tra hàm đã viết theo ví dụ sau:

```
Graph g = createGraph();
addVertex(g, 0, "V0");
addVertex(g, 1, "V1");
addVertex(g, 2, "V2");
addVertex(g, 3, "V3");
addEdge(g, 0, 1);
addEdge(g, 1, 2);
addEdge(g, 2, 0);
addEdge(g, 1, 3);
if (DAG(g)) printf("The graph is acycle\n");
else printf("Have cycles in the graph\n");
```

Sắp xếp tô-pô (Topological Sort)

- Ta có thể sử dụng hướng trong đồ thị có hướng để biểu diễn 1 quan hệ phụ thuộc
  - C-Basic là điều kiện bắt buộc khi học C-Advance
  - Bữa sáng phải ăn trước bữa trưa.
- 1 ứng dụng khác của đồ thị có hướng là:
- lập lịch thực hiện 1 công việc mà công việc này yêu cầu cần đảm bảo trật-tự-thực-hiện-các-bước, sử dụng thuật toán sắp xếp tô pô.

Topological Sort (tiếp)

- 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, mục tiêu của chúng ta là 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 tô pô

1. Xây dựng “bảng indegree” của DAG
2. Đưa ra đỉnh v có indegree = 0
3. Với đỉnh v, cung (v, w) không còn hữu ích nữa bởi vì công việc (đỉnh) w không còn cần phải đợi v hoàn thành xong nữa.
   - Do đó, sau khi đưa ra đỉnh v, ta có thể xoá v và các cung xuất phát từ nó. Đồ thị kết quả vẫn là đồ thị có hướng không chu trình.
   - Do đó ta có thể lặp lại bước 2 cho đến khi không còn đỉnh nào

Demo

- demo-topological.ppt

Pseudocode

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);
```

Quiz 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

Hint

- Tạo 1 hàm mới cho sắp xếp topological .
  - void topologicalSort(Graph g, int* output, int* n);
- Dùng lại ví dụ sau để kiểm tra hàm vừa viết.

```
Graph g = createGraph();
addVertex(g, 0, "CS102"); addVertex(g, 1, "CS140");
addVertex(g, 2, "CS160"); addVertex(g, 3, "CS302");
addVertex(g, 4, "CS311"); addVertex(g, 5, "MATH300");
addEdge(g, 0, 1); addEdge(g, 0, 2);
addEdge(g, 1, 3); addEdge(g, 5, 4); addEdge(g, 3, 4);
if (!DAG(g)) {
   printf("Can not make topological sort\n");
   return 1; }
topologicalSort(g, output, &n);
printf("The topological order:\n");
for (i=0; i<n; i++)
    printf("%s\n", getVertex(g, output[i]));
```