Đồ thị có hướng

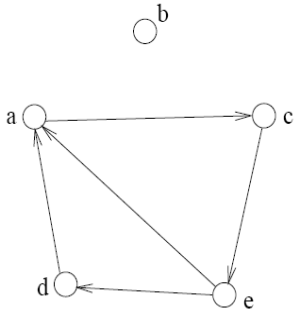
Thuật ngữ

- Đồ thị liên thông
  - Đồ thị liên thông nếu tồn tại đường đi giữa mọi cặp đỉnh phân biệt
- Đồ thị con
  - Đồ thị với tập đỉnh và cạnh là tập con của đỉnh và cạnh của đồ thị gốc
- Thành phần liên thông
  - Một thành phần liên thông là đồ thị con liên thông lớn nhất có thể của đồ thị gốc
- Chu trình
  - Đường đi có đỉnh đầu và đỉnh kết thúc giống nhau
- Cây
  - Đồ thị G là cây nếu nó liên thông và không chứa chu trình
- Đồ thị có hướng
  - Đồ thị chứa các cạnh là có hướng
- Đồ thị có hướng không chứa chu trình

Đồ thị có hướng

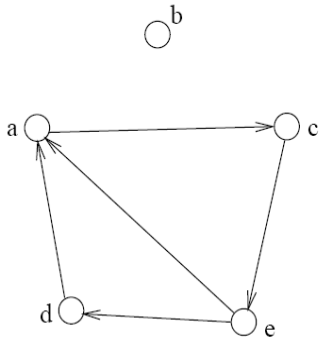
- Đồ thị có hướng có thể được biểu diễn bởi ma trận/danh sách kề tương tự đồ thị vô hướng, ngoại trừ:
  - Mỗi cung (u, v) chỉ tương ứng với một ô/phần tử trong ma trận/danh sách kề

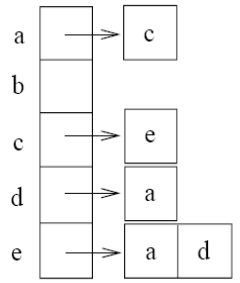
1. *Adjacency Matrix*



	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

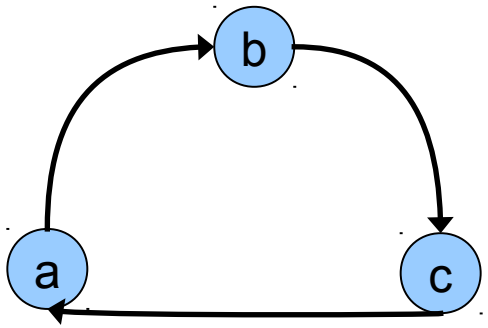
2. *Adjacency List*

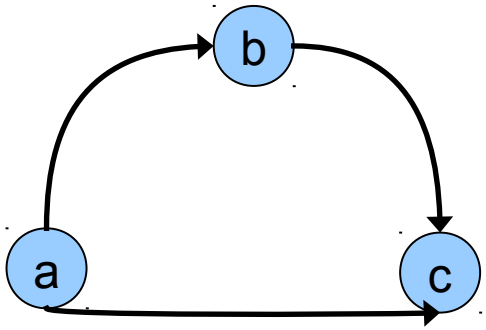




Đường đi/chu trình

- Đồ thị có hướng có thể chứa đường đi/chu trình
  - Đồ thị phía trên chứa đường đi và chu trình
  - Đồ thị phía dưới chứa đường đi nhưng không chứa chu trình





Duyệt đồ thị

- BFS và DFS có thể được sử dụng để duyệt đồ thị có hướng tương tự đồ thị vô hướng

DAG

- DAG là đồ thị có hướng không chứa chu trình
- Để kiểm tra chu trình trong đồ thị có hướng
  - thực thi DFS với đỉnh gốc bất kỳ. Nếu đỉnh gốc có thể được ghé thăm hai lần thì đồ thị chứa chu trình

API đồ thị đầy đủ

- API đồ thị hiện tại chỉ quản lý cạnh. Vì vậy ta không tính toán được số đỉnh. Mỗi đỉnh cũng cần một tên để định danh.
- Viết lại cấu trúc đồ thị để các đỉnh được lưu trữ bằng cây theo cấu trúc sau:

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

API (tiếp)

```
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);
```

Bài tập 1

- Cài đặt API đồ thị có hướng theo đặc tả như trên
- Kiểm thử API vừa tạo với 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");
```

Mã nguồn

- directed_graph.c

Sắp xếp tô-pô

- Có thể sử dụng hướng trong đồ thị có hướng để biểu diễn quan hệ phụ thuộc
  - COMP104 là một điều kiện của COMP171
  - Bữa sáng phải được ăn trước bữa trưa
- Một ứng dụng phổ biến là lập lịch một thứ tự bảo toàn ràng buộc thứ tự hoàn thành theo một giải thuật sắp xếp tô-pô
  - Để mỗi đỉnh biểu diễn một tác vụ cần thực hiện. Các tác vụ phụ thuộc lẫn nhau sao cho một số tác vụ chưa thể thực hiện nếu các tác vụ trước đó chưa hoàn thành
  - Cho một DAG, yêu cầu tạo ra một trật tự tuyến tính của các tác vụ sao cho ràng buộc thứ tự biểu diễn bởi các cạnh được tuân thủ
  - Trật tự tuyến tính không là duy nhất

Giải thuật sắp xếp tô-pô

1. Xây dựng một “bảng bậc-vào” của DAG
2. Cho ra một đỉnh v với bâc-vào không
3. Với đỉnh v, cung (v, w) không còn giá trị vì tác vụ (đỉnh) w không cần đợi v hoàn thành nữa
   - Sau khi cho ra đỉnh v, ta có thể xóa bỏ v và các cung ra của nó. Đồ thị kết quả vẫn là DAG. Vì thế ta có thể lặp lại bước 2 cho tới khi không còn đỉnh nào

Demo

- demo-topological.ppt

Pseudocode

```
Algorithm TSort(G)
Input: a directed acyclic graph G
Output: a topological ordering of vertices
Initialize Q to be an empty queue;
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);
```

Bài tập 2

- Cho một tập mô tả các môn học điều kiện như sau

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

- Sử dụng API đồ thị ở trên để viết một chương trình sinh ra một thứ tự tô-pô của các môn học

Gợi ý

- Tạo ra một hàm mới để sắp xếp tô-pô
  - void topologicalSort(Graph g, int* output, int* n);
- Sử dụng ví dụ sau để kiểm tra chương trình:

```
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]));
```

Solution

- topological_sort.c