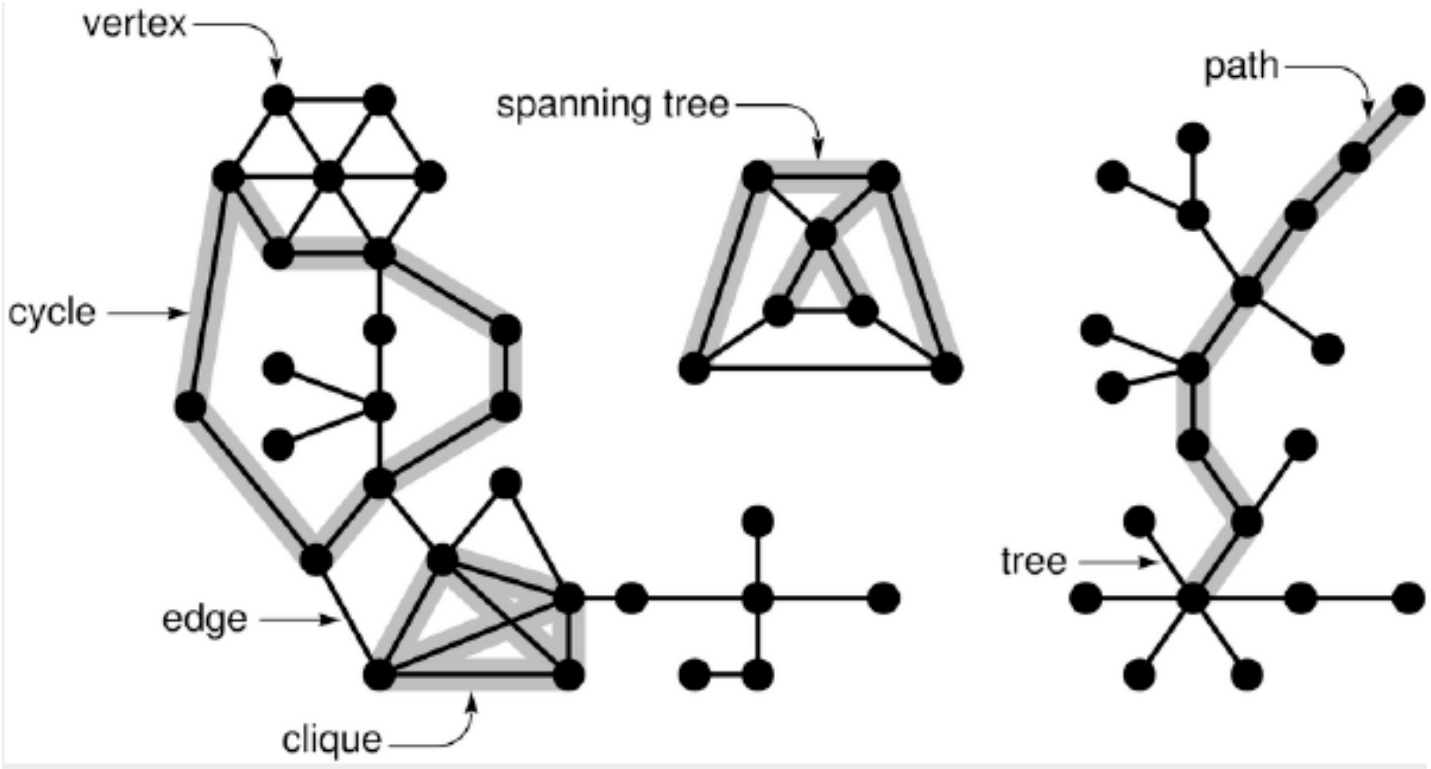
Đồ thị vô hướng

Đồ thị vô hướng

- Đồ thị G=(V, E) trong đó V là tập các đỉnh (vertice), E là tập các cạnh/cung (edge)
- Tại sao nghiên cứu về đồ thị?
  - Hấp dẫn và được sử dụng rộng rãi
  - Vấn đề thách thức trong khoa học máy tính & toán rời rạc
  - Hàng trăm giải thuật về đồ thị
  - Hàng ngàn ứng dụng thực tế: truyền thông, vi mạch, giao thông, hệ thống phần mềm, lập lịch, internet, games, mạng xã hội, mạng nhân tạo

Thuật ngữ đồ thị

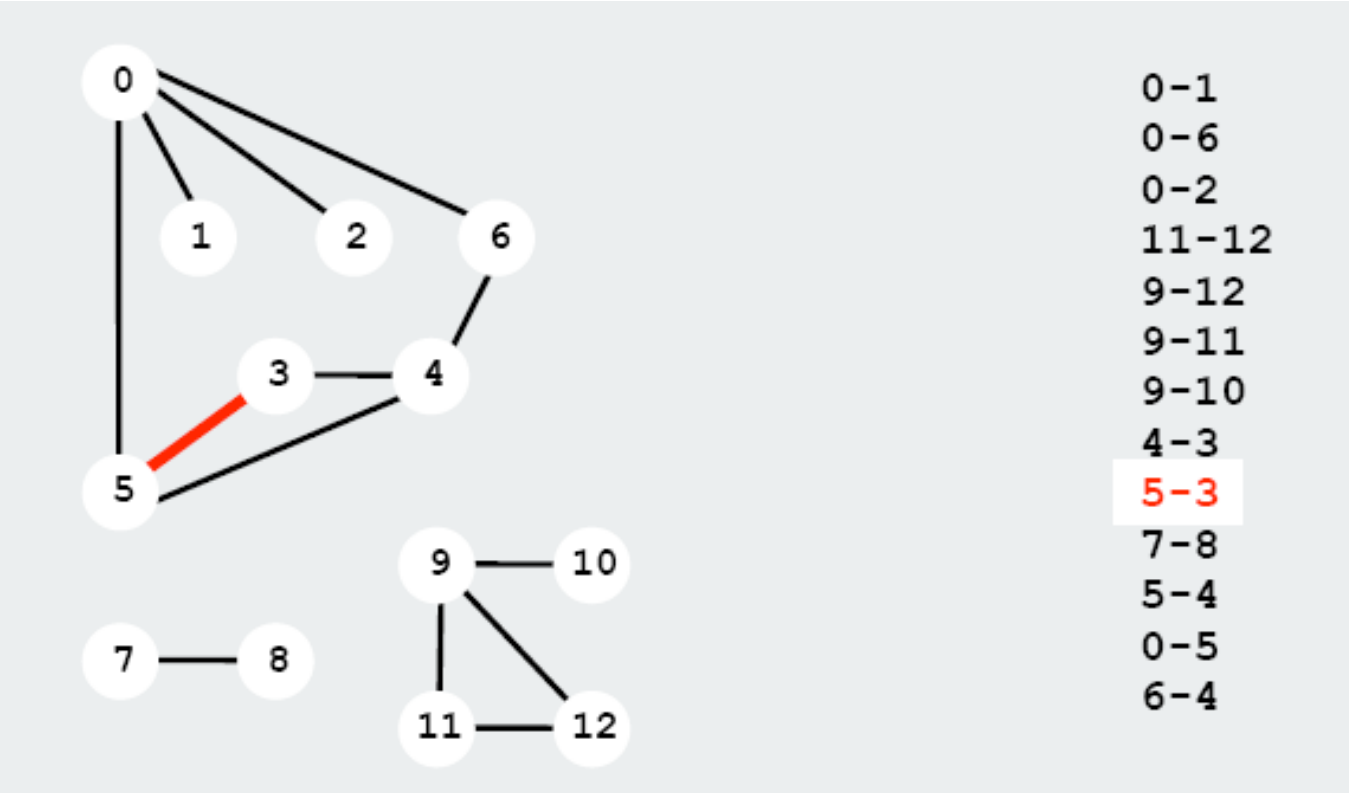


Một vài vấn đề về đồ thị

- Đường đi: Có đường đi giữa s và t?
- Đường đi ngắn nhất: Đâu là đường đi ngắn nhất giữa s và t?
- Chu trình: Tồn tại chu trình trong đồ thị?
- Chu trình Euler: Tồn tại chu trình mà đi qua mỗi cạnh đúng một lần?
- Chu trình Hamilton: Tồn tại chu trình mà đi qua mỗi đỉnh đúng một lần?
- Liên thông: Tồn tại đường đi kết nối tất cả các đỉnh?
- MST: Cách nào tốt nhất để kết nối tất cả các đỉnh?
- Biconnectivity: Tồn tại một đỉnh mà loại bỏ nó sẽ làm đồ thị không liên thông?

Biểu diễn đồ thị (1)

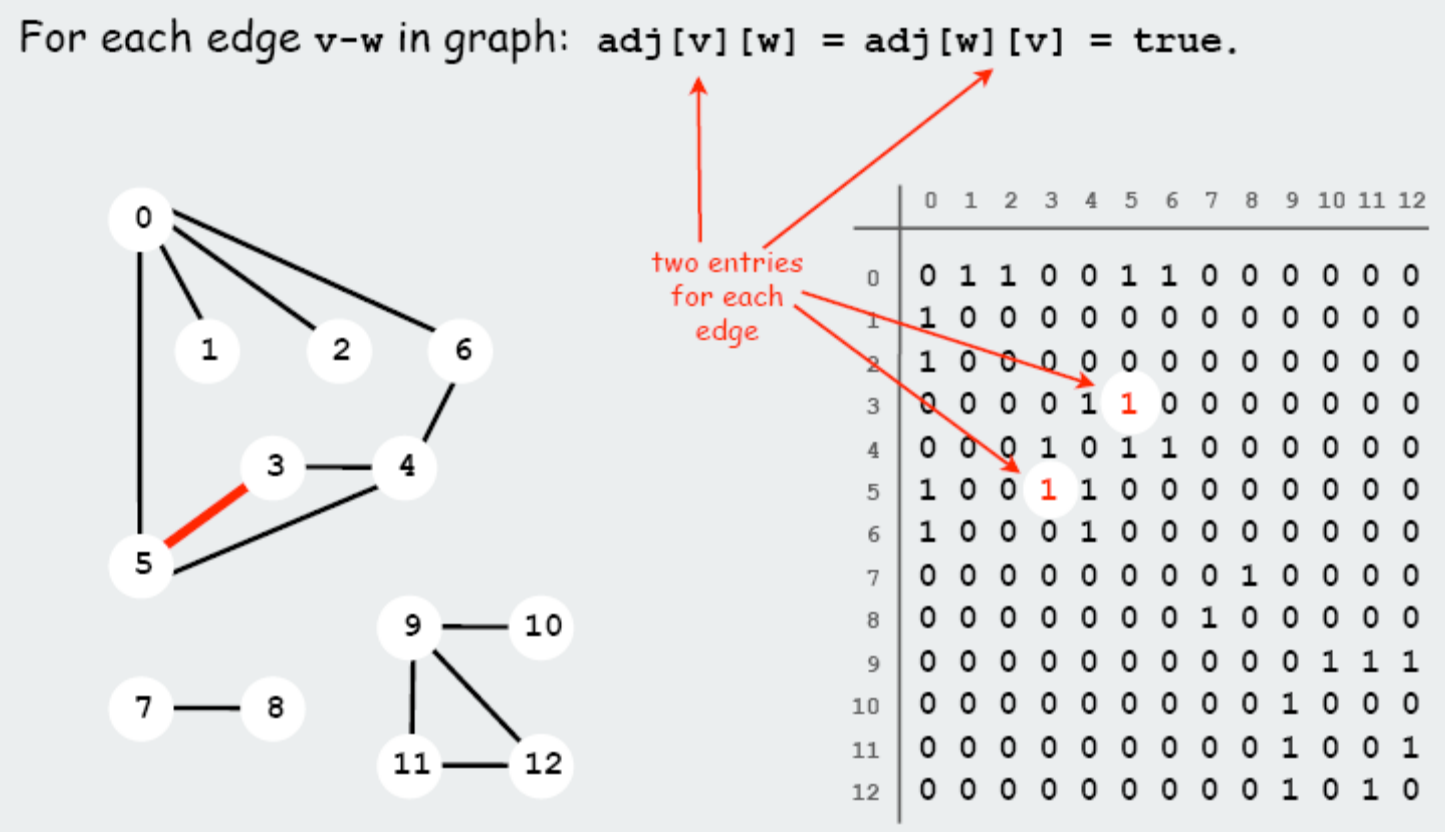
- Lưu trữ danh sách các cạnh



- Không phù hợp cho tìm kiếm

Biểu diễn đồ thị (2)

- Lưu trữ ma trận liền kề



- Phù hợp cho truy cập ngẫu nhiên tới các cạnh

Cấu trúc dữ liệu đồ thị

- Sử dụng mảng động để biểu diễn đồ thị

```
typedef struct {
    int * matrix;
    int sizemax;
} Graph;
```

- Lưu ý: thay vì lưu trữ ở mảng hai chiều, chúng ta sử dụng mảng một chiều để lưu trữ

A[n][n] => A[n*n]

A[i][j] => A[i*n+j]

- Định nghĩa các hàm API

```
Graph createGraph(int sizemax);
void addEdge(Graph graph, int v1, int v2);
int adjacent(Graph graph, int v1, int v2);
int getAdjacentVertices(Graph graph, int vertex, int* output); // return
    the number of adjacent vertices.
void dropGraph(Graph graph);
```

Ví dụ khi sử dụng API?

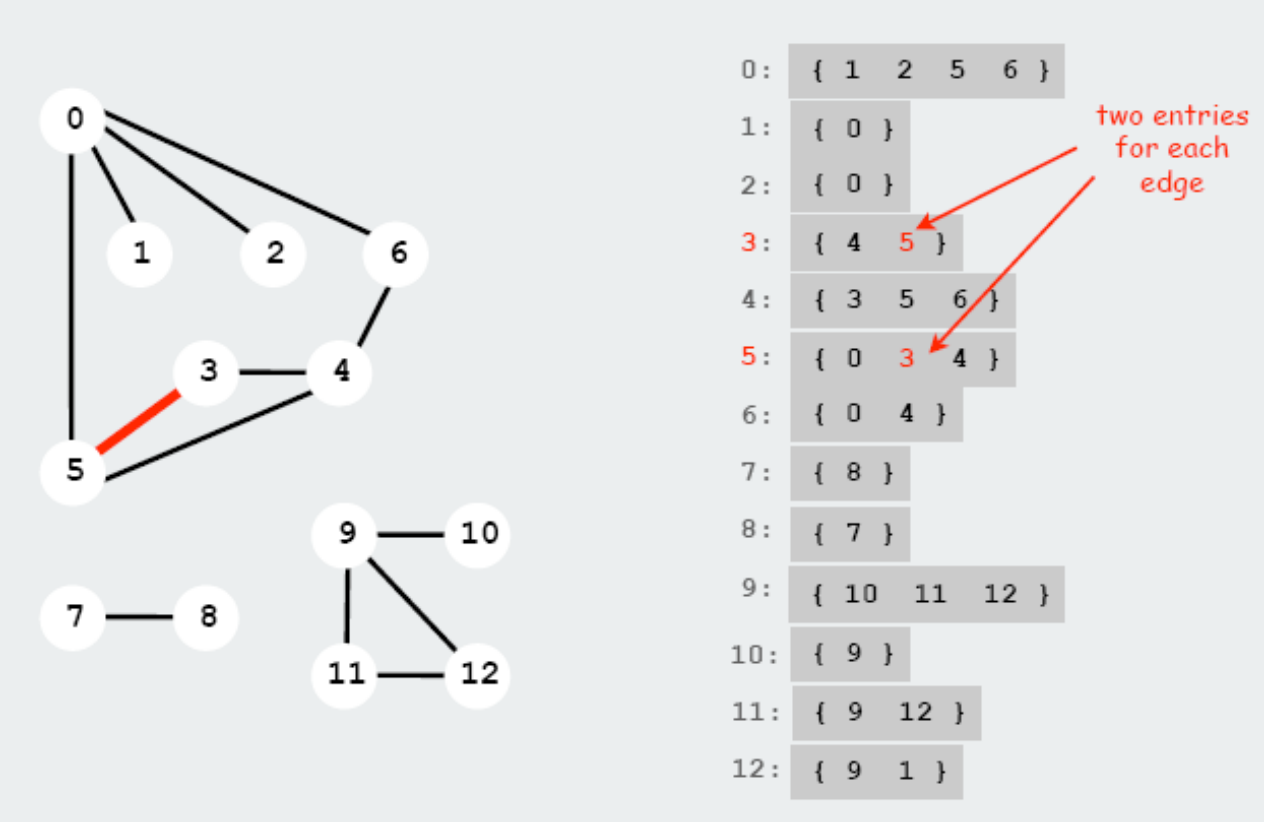
```
int i, n, output[100];
Graph g = createGraph(100);
addEdge(g, 0, 1);
addEdge(g, 0, 2);
addEdge(g, 1, 2);
addEdge(g, 1, 3);
n = getAdjacentVertices (g, 1, output);
if (n==0) printf("No adjacent vertices of node 1\n");
else {
   printf("Adjacent vertices of node 1:");
   for (i=0; i<n; i++) printf("%5d", output[i]);
}
```

Bài 1

- Định nghĩa các hàm API đã khai báo
- Viết chương trình kiểm tra các API đó

Biểu diễn ma trận(3)

- Lưu trữ danh sách liền kề



So sánh

- Danh sách liền kề là cách biểu diễn phù hợp khi biểu diễn đồ thị thưa (sparse) với |E| nhỏ hơn rất nhiều so với $|V|^2$
- Ma trận liền kề là cách biểu diễn phù hợp khi đồ thị quá dày (dense) hoặc khi chúng ta cần có khả năng trả lời nhanh có cạnh nối hai đỉnh hay không?

Cài đặt

- Cây đỏ đen có thể được sử dụng để lưu trữ đồ thị, trong đó mỗi nút trên cây là một đỉnh, và giá trị của nó là danh sách các đỉnh liền kề

Bài 2

- Định nghĩa lại các hàm API cho đồ thị sử dụng thư viện libfdr

```
#include "jrb.h"
typedef JRB Graph;

Graph createGraph();
void addEdge(Graph graph, int v1, int v2);
int getAdjacentVertices (Graph graph, int v, int* output);
void dropGraph(Graph graph);
//Có thể bỏ qua hàm adjacent
```

Hướng dẫn (1)

- Để tạo đồ thị

  Gọi hàm make_jrb()
- Để add một cạnh (v1, v2) vào đồ thị g
- Nếu nút v1 chưa được chèn vào cây

```
tree = make_jrb();
jrb_insert_int(g, v1, new_jval_v(tree));
jrb_insert_int(tree, v2, new_jval_i(1));
```

- Nếu nút v1 đã được chèn vào cây

```
node = jrb_find_int(g, v1);
tree = (JRB) jval_v(node->val);
jrb_insert_int(tree, v2, new_jval_i(1));
```

Hướng dẫn (2)

- Để lấy ra các đỉnh liền kề của v trong đồ thị g

```
node = jrb_find_int(g, v);
tree = (JRB) jval_v(node->val);
total = 0;
jrb_traverse(node, tree)
    output[total++] = jval_i(node->key);
```

- Để xóa/giải phóng đồ thị

```
jrb_traverse(node, graph)
    jrb_free_tree( jval_v(node->val) );
```

Bài 3

- Để mô tả về hệ thống đường tàu điện ngầm, người ta lưu trữ dữ liệu trong file văn bản theo dạng

  [STATIONS]
  S1=Name of station 1
  S2=Name of station 2
  …
  [LINES]
  M1=S1 S2 S4 S3 S7
  M2=S3 S5 S6 S8 S9
  …

- Viết chương trình đọc file và thiết lập mạng lưới nhà ga sử dụng các API đã có
- Viết hàm với tham số đầu vào là tên một nhà ga, và trả về tất cả các nhà ga liền kề với nó