BIỂU DIỄN ĐỒ THỊ

Phạm Nguyên Khang
Võ Trí Thức

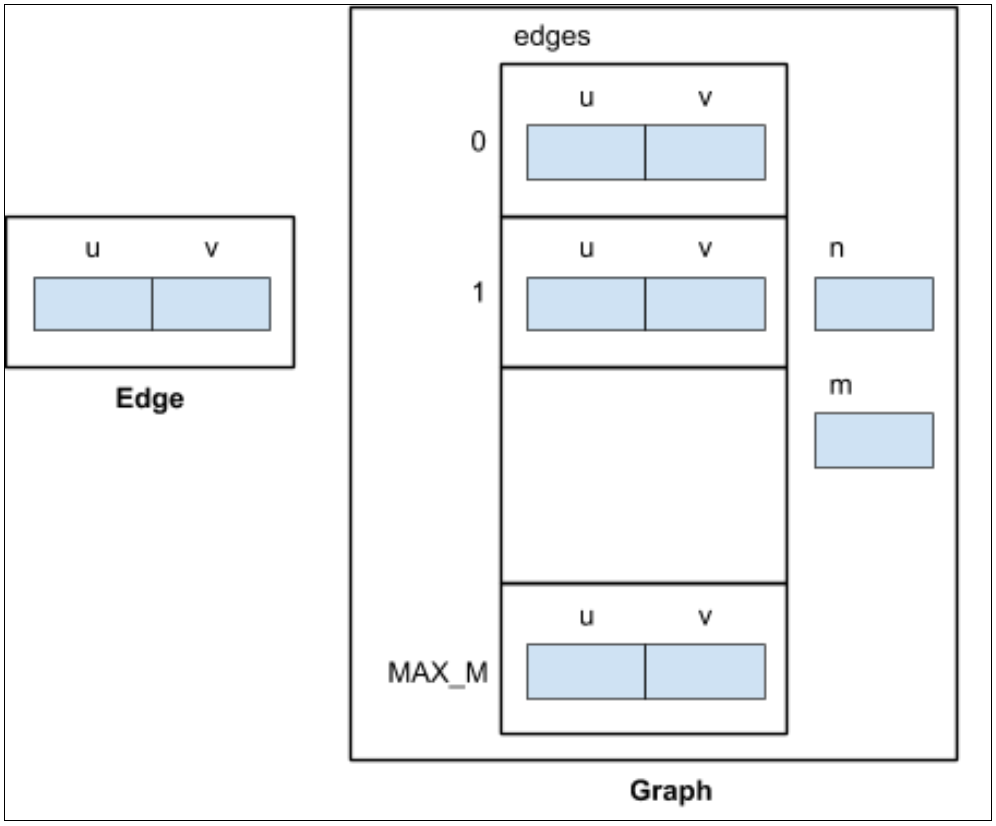
Quy ước chung

- Đánh số các đỉnh từ 1, 2, …, n
- Đỉnh: chỉ cần số lượng đỉnh: **n**
- Cung:
    - Số lượng cung: **m**
    - Chi tiết cung: **tuỳ phương pháp biểu diễn**

Biểu diễn bằng danh sách cung

- **Ý tưởng:**
    - Mỗi cung lưu 2 đỉnh đầu mút (endpoint) của nó
    - Lưu tất cả các cung của đồ thị vào một danh sách (*danh sách đặc* hoặc *danh sách liên kết*)

- **Sơ đồ tổ chức dữ liệu (dùng mảng để cài danh sách):**

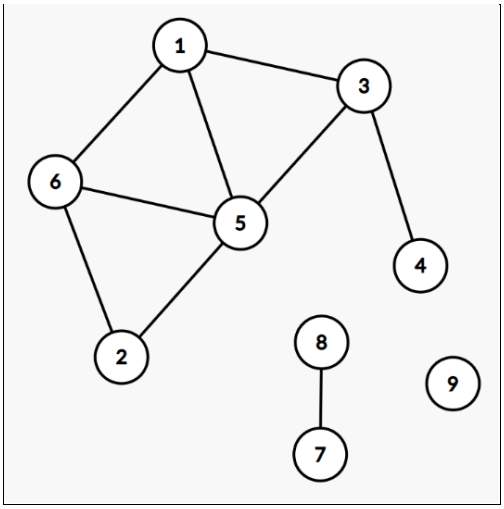


- **Cài đặt:**

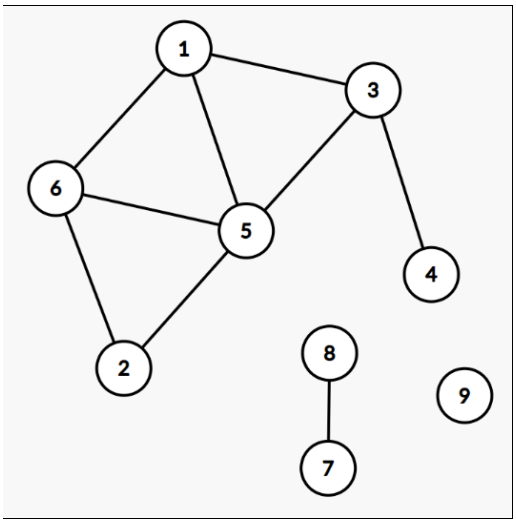
```
#define MAX_M 500
typedef struct {
        int u, v;
} Edge;
typedef struct {
        Edge edges[MAX_M];
        int n, m;
} Graph;
```

Bài tập

1. Liệt kê các cung của đồ thị sau:



2. Hãy vẽ sơ đồ tổ chức dữ liệu cho đồ thị sau sử dụng phương pháp danh sách cung.



Bài tập lập trình

Với khai báo đồ thị như bên trên (sử dụng danh sách cung), *giả sử đồ thị vô hướng* hãy viết các hàm sau:

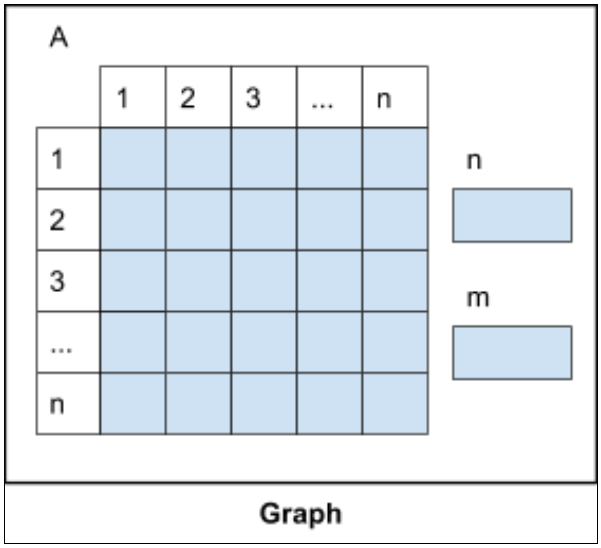
3. **void init_graph(Graph* pG, int n):** khởi tạo đồ thị có n đỉnh và 0 cung.
   Gợi ý: gán pG->n = n và gán pG->m = 0.
4. **void add_edge(Graph* pG, int x, int y)**: thêm cung (x, y) vào đồ thị.
   Gợi ý: thêm phần tử (x, y) vào danh sách edges, tăng số cung m lên 1.
5. **int degree(Graph* pG, int x)**: tính và trả về bậc của đỉnh x.
   Gợi ý: duyệt qua danh sách cung edges và đếm xem có bao nhiêu phần tử có dạng (x, -) hoặc (-, x).
6. **int adjacent(Graph* pG, int x, int y):** trả về 1 nếu x kề với y (pG là đồ thị vô hướng), ngược lại trả về 0.
   Gợi ý: duyệt qua danh sách cung edges kiểm tra xem có phần tử nào có dạng (x, y) hoặc (y, x) không.
7. **void neighbours (Graph* int x)**: in các đỉnh kề của x ra màn hình, mỗi đỉnh cách nhau 1 khoảng trắng.
   Gợi ý: Cho y chạy từ 1 đến n, kiểm tra nếu y kề với x thì in y ra màn hình.

Biểu diễn bằng ma trận kề (ma trận đỉnh - đỉnh)

- **Ý tưởng:**
  - Lưu lại sự kề nhau của các cặp đỉnh (quan hệ giữa đỉnh và đỉnh)
  - Dùng một ma trận 0/1 để lưu sự kề nhau của các cặp đỉnh (u, v)
  - Nếu **đỉnh v là đỉnh kề của đỉnh u** thì phần tử ở **hàng u, cột v sẽ có giá trị 1** ngược lại có giá trị 0.

- **Chú ý:**
  - Đồ thị vô hướng sẽ có ma trận kề đối xứng
  - Không lưu trữ được đa cung (phải dùng phương pháp ma trận kề mở rộng)

- **Sơ đồ tổ chức dữ liệu (dùng mảng 2 chiều để lưu ma trận, bỏ qua cột 0 và hàng 0):**

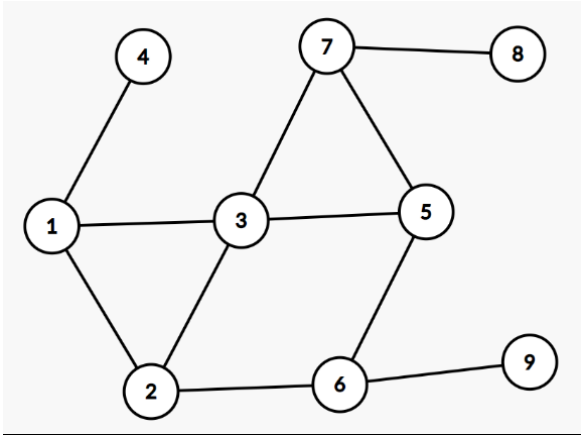


- **Cài đặt:**
  - Để đơn giản và tương thích với quy ước đánh số từ 1, 2, … ta không sử dụng hàng 0 và cột 0 của ma trận
  - Phần tử A[u][v] lưu sự kề nhau của 2 đỉnh u và v.

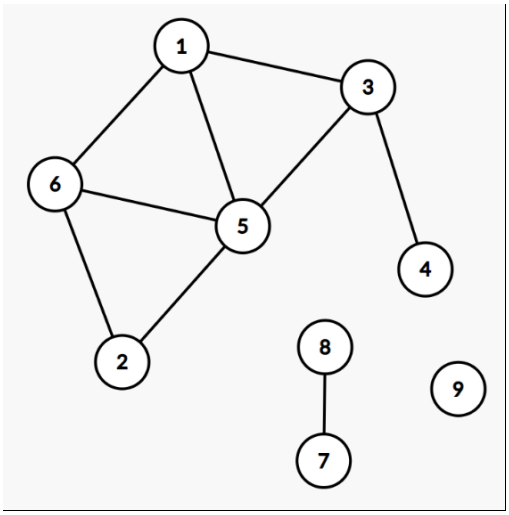
```
1  #define MAX_N 100
2  typedef struct {
3          int A[MAX_N][MAX_N];
4          int n, m;
5  } Graph;
6
```

Bài tập

8. Cho biết ma trận kề của đồ thị sau:



9. Hãy vẽ sơ đồ tổ chức dữ liệu cho đồ thị sau sử dụng phương pháp ma trận kề.



Bài tập lập trình

Với khai báo đồ thị như bên trên (sử dụng ma trận kề), *giả sử đồ thị vô hướng* hãy viết các hàm sau:

10. **void init_graph(Graph* pG, int n):** khởi tạo đồ thị có n đỉnh và 0 cung. Gợi ý: điền các số 0 vào ma trận kề A.
11. **void add_edge(Graph* pG, int x, int y)**: thêm cung (x, y) vào đồ thị. Gợi ý: cho x và y kề nhau (gán A[x][y] = 1 và A[y][x] = 1).
12. **int degree(Graph* pG, int x)**: tính và trả về bậc của đỉnh x. Gợi ý: đếm số lượng số 1 (hoặc tỉnh tổng các phần tử) trên hàng x của ma trận A.
13. **int adjacent(Graph* pG, int x, int y):** trả về 1 nếu x kề với y (pG là đồ thị vô hướng), ngược lại trả về 0. Gợi ý: kiểm tra phần tử A[x][y].
14. **void neighbours (Graph* pG, int x)**: in các đỉnh kề của x ra màn hình, mỗi đỉnh cách nhau 1 khoảng trắng. Gợi ý: Cho y chạy từ 1 đến n, kiểm tra nếu y kề với x thì in y ra màn hình.

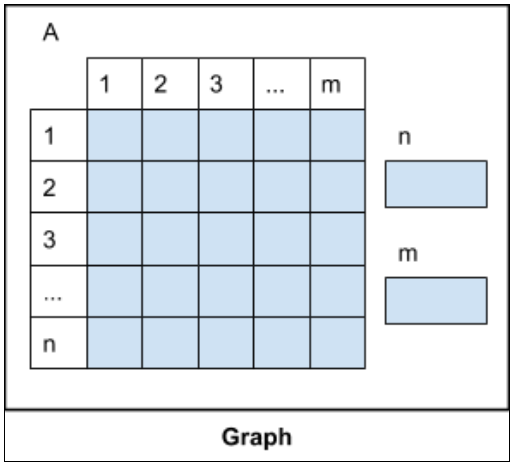
Biểu diễn bằng ma trận liên thuộc

(ma trận đỉnh - cung)

- **Ý tưởng:**
  - Lưu lại sự liên thuộc nhau của các cặp đỉnh-cung (quan hệ giữa đỉnh và cung)
  - Đánh số các cung từ 1 đến m.
  - Dùng một ma trận 0/1 để lưu sự liên thuộc nhau của các cặp đỉnh-cung (u, e), hàng tương ứng với đỉnh, cột tương ứng với cung.
  - Nếu đỉnh **u liên thuộc với cung e** (*u là 1 đầu mút của e*) thì phần tử ở **hàng u, cột e sẽ có giá trị 1** ngược lại có giá trị 0.

- **Chú ý:**
  - Có thể lưu trữ được đa cung.
  - Nếu đồ thị có chứa khuyên cần phải sử dụng giá trị 2 thay cho 1.

- Sơ đồ tổ chức dữ liệu (dùng mảng 2 chiều để lưu ma trận, bỏ qua cột 0 và hàng 0):

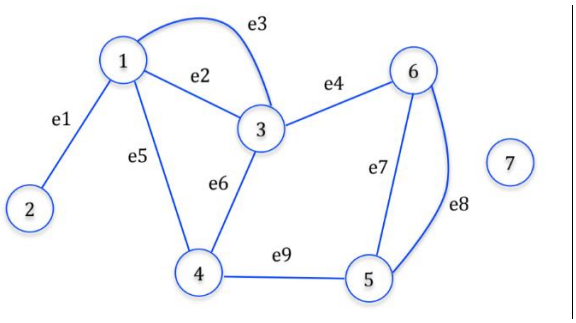


- **Cài đặt:**
  - Để đơn giản và tương thích với quy ước đánh số từ 1, 2, … ta không sử dụng hàng 0 và cột 0 của ma trận.
  - Phần tử A[u][e] lưu sự liên thuộc nhau của 2 đỉnh u và cung e.

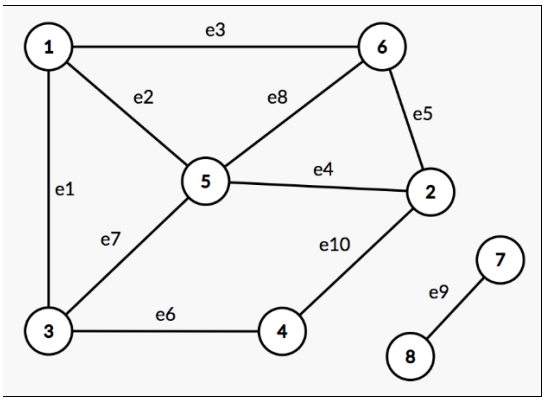
```
1  #define MAX_N 100
2  #define MAX_M 500
3  typedef struct {
4          int A[MAX_N][MAX_M];
5          int n, m;
6  } Graph;
7
```

Bài tập

15. Cho biết ma trận liên thuộc tương ứng của đồ thị sau:



16. Hãy vẽ sơ đồ tổ chức dữ liệu cho đồ thị sau sử dụng phương pháp ma trận liên thuộc.



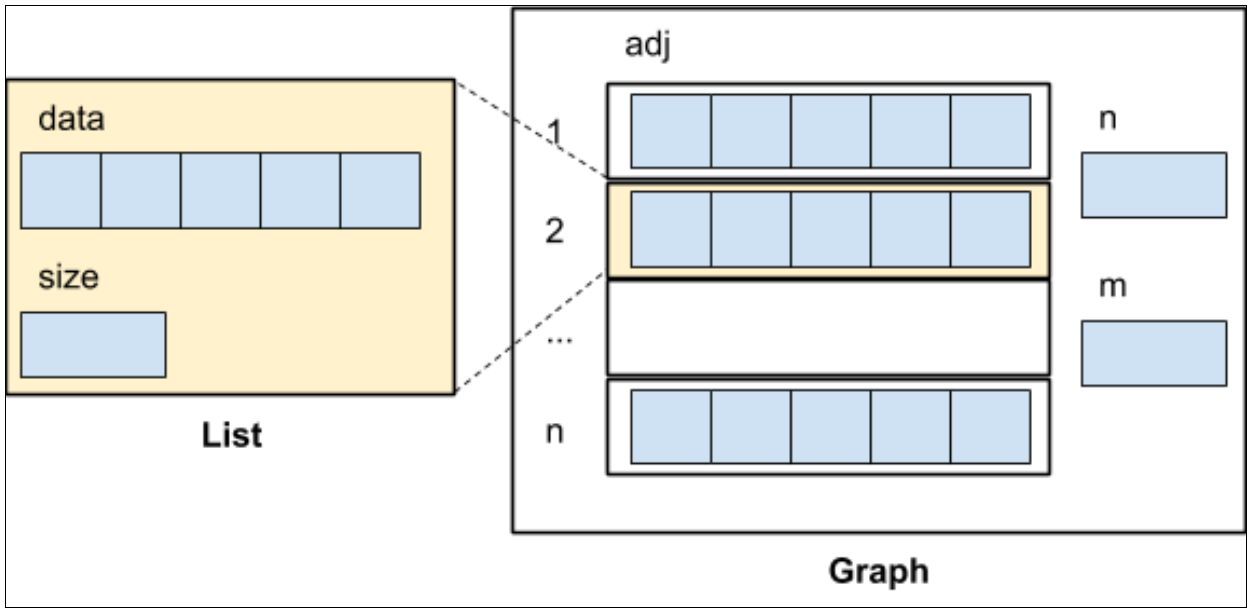
Bài tập lập trình

Với khai báo đồ thị như bên trên (sử dụng ma trận liên thuộc), *giả sử đồ thị vô hướng* hãy viết các hàm sau:

17. **void init_graph(Graph* pG, int n, int m):** khởi tạo đồ thị có n đỉnh và m cung. Gợi ý: điền các số 0 vào ma trận liên thuộc A.
18. **void add_edge(Graph* pG, int e, int x, int y)**: thêm cung e = (x, y) vào đồ thị. Gợi ý: cho x liên thuộc với e và y liên thuộc với e (gán A[x][e] = 1 và A[y][e] = 1).
19. **int degree(Graph* pG, int x)**: tính và trả về bậc của đỉnh x. Gợi ý: đếm số lượng số 1 trên hàng x (hoặc cộng tất cả các số trên hàng x) của ma trận A.
20. **int adjacent(Graph* pG, int x, int y):** trả về 1 nếu x kề với y (pG là đồ thị vô hướng), ngược lại trả về 0. Gợi ý: duyệt qua các cột (tương ứng với cung), kiểm tra xem có cột e nào có 2 số 1 tương ứng ở hàng x và hàng y không (A[x][e] == 1 && A[y][e] == 1).
21. **void neighbours (Graph *pG, int x)**: in các đỉnh kề của x ra màn hình, mỗi đỉnh cách nhau 1 khoảng trắng. Gợi ý: Cho y chạy từ 1 đến n, kiểm tra nếu y kề với x thì in y ra màn hình.

Biểu diễn bằng danh sách đỉnh kề

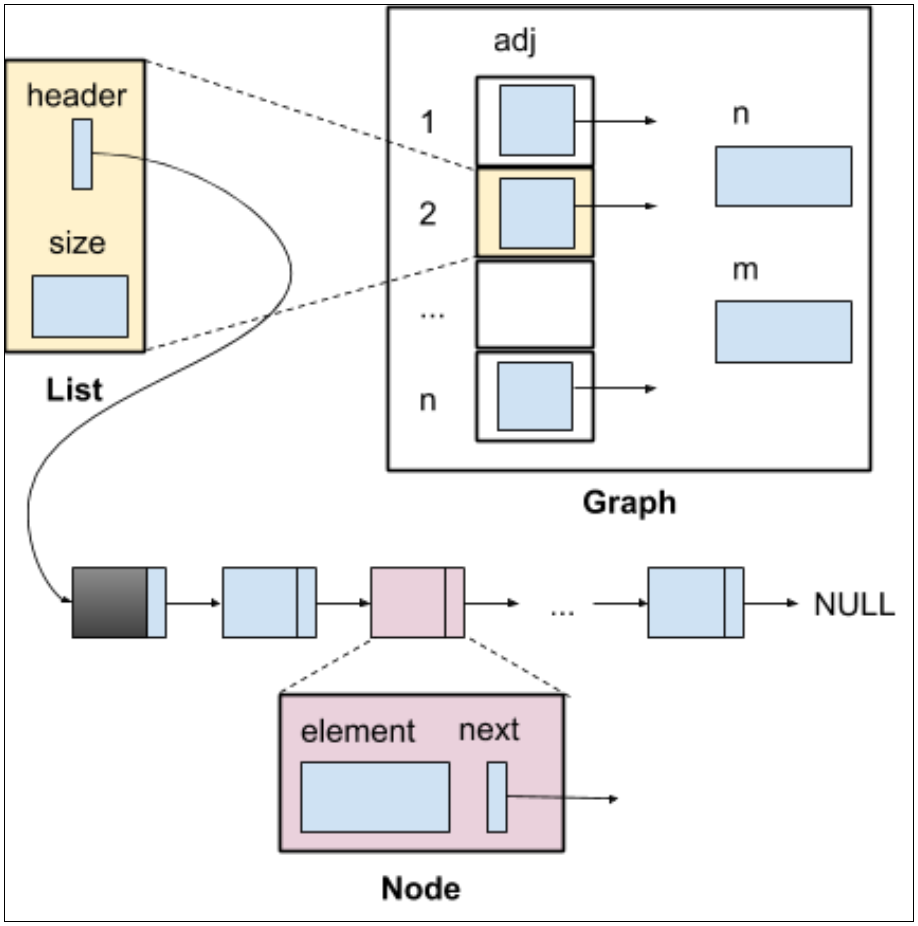
- **Ý tưởng:**
  - Mỗi đỉnh, lưu lại tất cả các đỉnh kề với nó
  - Có n đỉnh, nên cần n danh sách => một mảng chứa n danh sách
- **Chú ý:**
  - Không lưu trữ được đa cung (có thể mở rộng để lưu đa cung).
- Sơ đồ tổ chức dữ liệu (dùng mảng để chứa n danh sách, để đơn giản bỏ qua phần tử thứ 0 của mảng, **sử dụng danh sách đặc**):



- **Cài đặt:**
  - Để đơn giản và tương thích với quy ước đánh số từ 1, 2, … ta không sử dụng hàng 0 của mảng.

```
#define MAX_N 100
#define MAX_M 500
typedef struct {
        int data[MAX_N];
        int size;
} List;
typedef struct {
        List adj[MAX_N]; //mảng các danh sách
        int n, m;
} Graph;
```

- Sơ đồ tổ chức dữ liệu (dùng mảng để chứa n danh sách, để đơn giản bỏ qua phần tử thứ 0 của mảng, **sử dụng danh sách liên kết**):



- Cài đặt:
    - Để đơn giản và tương thích với quy ước đánh số từ 1, 2, … ta không sử dụng hàng 0 của mảng.

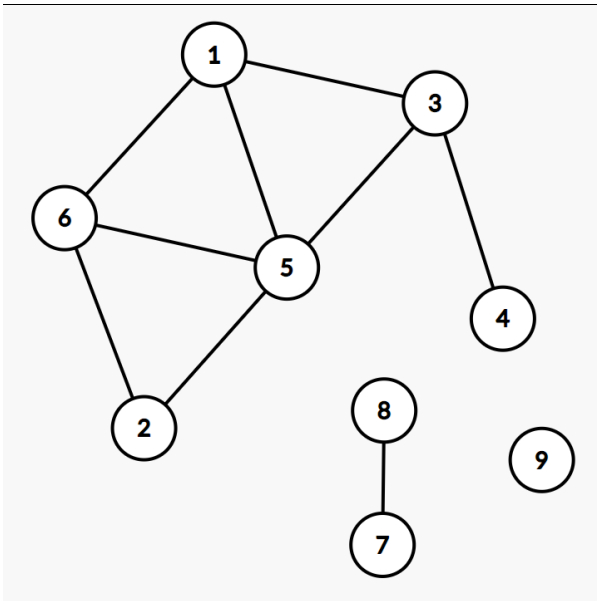
```
1  #define MAX_N 100
2  #define MAX_M 500
3  typedef struct Node* NodePointer;
4  struct Node {
5          int element;
6          NodePointer next;
7  };
8
```

```
9  typedef struct {
10         NodePointer header;
11         int size;
12 } List;
```

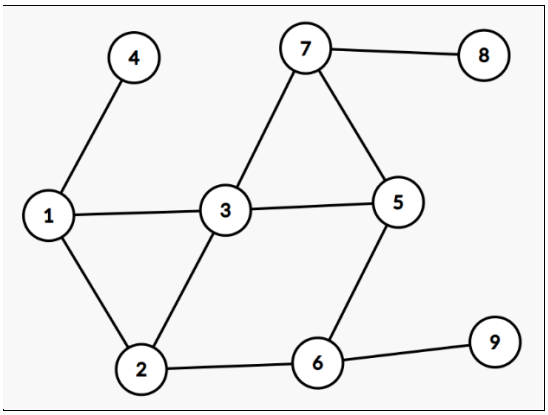
```
14  typedef struct {
15          List adj[MAX_N]; //mảng các danh sách
16          int n, m;
17  } Graph;
```

Bài tập

22. Cho biết danh sách đỉnh kề của các đỉnh của đồ thị sau.



23. Hãy vẽ sơ đồ tổ chức dữ liệu cho đồ thị sau sử dụng phương pháp danh sách đỉnh kề (**sử dụng danh sách đặc và danh sách liên kết**).



Bài tập lập trình

Với khai báo đồ thị như bên trên (phương pháp danh sách đỉnh kề, sử dụng danh sách đặc), *giả sử đồ thị vô hướng* và ta đã có các hàm sau:

void make_null(List* pL): khởi tạo danh sách rỗng. **List** được cài đặt bằng mảng.
void push_back(List* pL, int x): thêm phần tử x vào cuối danh sách pL.
int element_at(List* pL, int p): trả về phần tử tại vị trí p trong danh sách pL, vị trí tính từ 1.
int size(List* pL): trả về số phần tử của danh sách pL.
Hãy viết thêm các hàm sau:

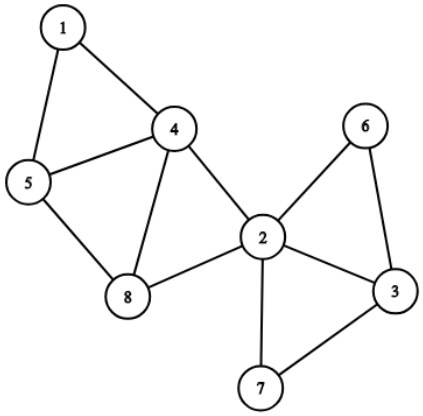
24. **void init_graph(Graph* pG, int n, int m):** khởi tạo đồ thị có n đỉnh và m cung. Gợi ý: khởi tạo n danh sách rỗng.
25. **void add_edge(Graph* pG, int x, int y)**: thêm cung (x, y) vào đồ thị. Gợi ý: thêm y vào danh sách adj[x] và thêm x vào danh sách của adj[y].
26. **int degree(Graph* pG, int x)**: tính và trả về bậc của đỉnh x. Gợi ý: trả về số phần tử của danh sách adj[x].
27. **int adjacent(Graph* pG, int x, int y):** trả về 1 nếu x kề với y (pG là đồ thị vô hướng), ngược lại trả về 0. Gợi ý: tìm y trong danh sách adj[x].
28. **void neighbours (Graph* int x)**: in các đỉnh kề của x ra màn hình, mỗi đỉnh cách nhau 1 khoảng trắng. Gợi ý: in các phần tử trong danh sách adj[x].

Duyệt đồ thị theo chiều sâu

```
1   /* Khai bao Stack*/
2   #define MAX_ELEMENTS 100
3   typedef struct {
4       int data[MAX_ELEMENTS];
5       int size;
6   } Stack;
7
8   void make_null_stack(Stack* S) {
9       S->size = 0;
10  }
11  void push(Stack* S, int x) {
12      S->data[S->size] = x;
13      S->size++;
14  }
15  int top(Stack* S) {
16      return S->data[S->size - 1];
17  }
18  void pop(Stack* S) {
19      S->size--;
20  }
21  int empty(Stack* S) {
22      return S->size == 0;
23  }
```

```
1   /* Duyet do thi theo chieu sau */
2   void depth_first_search(Graph* G) {
3       Stack frontier;
4       int mark[MAX_VERTEXES];
5       make_null_stack(&frontier);
6   /* Khởi tạo mark, chưa đỉnh nào được xét */
7       int j;
8       for (j = 1; j <= G->n; j++)
9           mark[j] = 0;
10  /* Đưa 1 vào frontier */
11      push(&frontier, 1);
12  /* Vòng lặp chính dùng để duyệt */
13      while (!empty(&frontier)) {
14  /* Lấy phần tử đầu tiên trong frontier ra */
15          int x = top(&frontier); pop(&frontier);
16          if (mark[x]!=0) //Đã được duyệt rồi
17              continue;
18          printf("Duyet %d\n", x);
19          mark[x]=1; //Đánh dấu đã được duyệt rồi
20  /* Lấy các đỉnh kề của nó */
21          List list = neighbors(G, x);
22  /* Xét các đỉnh kề của nó */
23          for (j = 1; j <= list.size; j++) {
24              int y = element_at(&list, j);
25              push(&frontier, y);
26          }
27      }
28  }
29
```

Bài tập: Chạy từng dòng code bằng tay và thực hiện duyệt đồ thị sau đây theo chiều sâu:



Duyệt đồ thị theo chiều rộng

```
1   /* Khai bao Queue */
2   #define MAX_ELEMENTS 100
3   typedef struct {
4       int data[MAX_ELEMENTS];
5       int front, rear;
6   } Queue;
7   void make_null_queue(Queue* Q) {
8       Q->front = 0;
9       Q->rear = -1;
10  }
11  void push(Queue* Q, int x) {
12      Q->rear++;
13      Q->data[Q->rear] = x;
14  }
15  int top(Queue* Q) {
16      return Q->data[Q->front];
17  }
18  void pop(Queue* Q) {
19      Q->front++;
20  }
21  int empty(Queue* Q) {
22      return Q->front > Q->rear;
23  }
```

```
1   /* Duyệt đồ thị theo chiều rộng */
2   void breath_first_search(Graph* G) {
3       Queue frontier;
4       int mark[MAX_VERTEXES];
5       make_null_queue(&frontier);
6   /* Khởi tạo mark, chưa đỉnh nào được xét */
7       int j;
8       for (j = 1; j <= G->n; j++)
9           mark[j] = 0;
10  /* Đưa 1 vào frontier */
11      push(&frontier, 1);
12      mark[1] = 1;
13  /* Vòng lặp chính dùng để duyệt */
14      while (!empty(&frontier)) {
15  /* Lấy phần tử đầu tiên trong frontier ra */
16          int x = top(&frontier); pop(&frontier);
17          printf("Duyet %d\n", x);
18  /* Lấy các đỉnh kề của nó */
19          List list = neighbors(G, x);
20  /* Xét các đỉnh kề của nó */
21          for (j = 1; j <= list.size; j++) {
22              int y = element_at(&list, j);
23              if (mark[y] == 0) {
24                  mark[y] = 1;
25                  push(&frontier, y);
26              }
27          }
28      }
29  }
```

Bài tập: Chạy từng dòng code bằng tay và thực hiện duyệt đồ thị sau đây theo chiều rộng: