# CODE 1

```
#include <stdio.h>
#define max 100 // Số cung tối đa
// cấu trúc của một cung
typedef struct {
    int x, y;
} Edge;
// cấu trúc của đồ thị G.
typedef struct {
    int n, m;
    Edge edges[max];
} Graph;
// hàm khởi tạo đồ thị
void init_Graph(Graph *G, int n) {
    G->n = n; // gán số đỉnh cho đt
    G->m = 0; // gán số cung cho đt = 0
}
// hàm thêm cung cho đồ thị.
void add_Edge(Graph *G, int x, int y) {
    G->edges[G->m].x = x; // Gán giá trị đỉnh đầu = x.
    G->edges[G->m].y = y; // gán giá trị đỉnh cuối = y.
    G->m++;
}
// kiểm tra đỉnh liền kề
int adTacent(Graph *G, int x, int y) {
    int e;
    for(e = 0; e < G->m; e++) // duyệt qua từng cung.
        // đỉnh đầu = x đỉnh cuối = y or ngược lại.
```

```
if ((G->edge[e].x == x && G->edges[e].y == y)
    || (G->edges[e].y == x && G->edges[e].x == y))
    return 1; // trả về 1
    return 0; // ngược lại trả về 0.
// xác định bậc của một đỉnh.
int degree (Graph *G, int x) {
    int e, deg = 0;
  for (e = 0; e < G->m; e++) { // duyệt từng cung.
      if (G->edges[e].x == x) deg++; // nếu x = đầu
      if (G->edges[e].y == x) deg++; // nếu x = cuối
    }
    return deg;
}

void neighbours (Graph *G, int u) {
  int v;
  for (v = 1; v <= G->n; v++) {
    if (adjacent (G, u, v) != 0)
    printf ("%d ", v);
    printf ("\n");
```

CODE 2.

```
#include <stdio.h>
#define MAX 100
#define MAX_E 100
// Khai báo đồ thị theo dạng Đỉnh - Đỉnh.
typedef struct {
  int A[MAX_E][MAX_E];
  int n, m;
} Graph;
  // Tạo Sơ đồ đỉnh đỉnh.
void init_graph(Graph *G, int n) {
  int i, j;
  G->n = n;
  for (i = 1; i <= G->n; i++) {
    for (j = 1; j <= G->n; j++) {
      G->A[i][j] = 0;
    }
  }
}
// Khởi tạo đồ thị đỉnh cung
void init_graph1(Graph *G, int n, int m) {
  int i, j;
  G->n = n;
  G->m = m;
  for (i = 1; i <= G->n; i++) {
    for (j = 1; j <= G->m; j++) {
      G->A[i][j] = 0;
    }
  }
}
// ... vào đồ thị đỉnh-đỉnh
```

```
// thêm cung vào đồ thị đỉnh - đỉnh
void add_edge (Graph *G, int x, int y) {
    G->A[x][y] = 1;
}

// thêm cung vào đồ thị đỉnh - cung.
void add_edge1 (Graph *G, int x, int y) {
    G->A[x][y] = 1;
}

// kiểm tra có phải láng giềng.
int adjacent (Graph *G, int x, int y) {
    return G->A[x][y] == 1;
}

Tính số bậc của một đỉnh.
int degree (Graph *G, int x) {
    int i;
    int deg = 0;
    for (i = 1; i <= G->n; i++) {
        if (G->A[x][i] == 1) {
            deg++;
        }
    }
    return deg;
}

// in đồ thị đỉnh đỉnh.
void in (Graph G) {
    int i, j;
    for (i = 1; i <= G.n; j++) {
        for (j = 1; j <= G.n; j++) {
            printf ("%d ", G.A[i][j]);
        }
        printf ("\n");
    }
}
```

Code 3.

```
// include <stdio.h>
// define max 100 // Số cung tối đa
// Cách

typedef struct {
    int n, m;      // số đỉnh và cung của đồ thị
    int A[max][max]
// hàm khởi tạo đồ thị.
void init_Graph (Graph *G, int n) {
    G->n = n; // gán số đỉnh cho đồ thị.
    G->m = 0; // gán số cung cho đồ thị = 0
     int u, v;
     for(u=1; u<=n; u++)
     for(v=1; v<=n; v++)
     G->A[u][v] = 0;
}
hàm thêm cung cho đồ thị.
void add_Edge(Graph *G, int x, int y) {
~~G->n = n;~~
G->A[x][y] = 1;
G->A[y][x] = 1;
G->m++;
}
// kiểm tra đỉnh kề
int adjacent (Graph *G, int x, int y) {
  return G->A[x][y] > 0;
}
// xác định bậc của một đỉnh.
int degree (Graph *G, int x) {
   int e, deg = 0;
   for(e=1; e<=G->n; e++) { // duyệt từng cung.
   deg += G->A[x][e];
```

```
    return deg + G -> A[u][u];
}
void neighbours (Graph *G, int u) {
    int v;
    for (v=1; v<= G->n; v++)
    if(G->A[u][v] != 0)
    printf("%d ", v);
    printf("\n");
}
// in đỉnh kề đỉnh.
void in(Graph G){
    int i, j
    for(i=1, i<= n; i++) {
        for(j=1; j<= G.n; j++) {
            printf("%d", G.A[i][j]);
        }
        printf("\n");
    }
}
// in đỉnh - cung.
void in(Graph G){
int i, j;
    for(i=1; i<= G.n; i++) {
        for(j=1, j<= G.n; j++) {
            printf("%d ", G.A[i][j]);
    }
        printf("\n");
    }
}
```