EX.3 BIỂU DIỄN ĐỒ THỊ

Thứ ngày / /

Biểu diễn bằng danh sách cung

Edge

u	v

Graph

n	edges[]	
m	0	u | v
	1	u | v
	:	
	MAX	u | v

1.

n = 9

m = 9

1	3
1	5
1	6
2	5
2	6
3	4
3	5
5	6
7	8

2. Graph

n = 9 | m = 9											
Edges											
u	1	1	1	2	2	3	3	5	7	...	
v	3	5	6	5	6	4	5	6	8	...	
	0	1	2	3	4	5	6	7	8	9	

Thứ ngày / /

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

3.
```
void init_graph(Graph* pG, int n) {
    pG->n = n;
    pG->m = 0;
}
```

4.
```
void add_edge(Graph* pG, int x, int y) {
    pG->edges[pG->m].u = x;
    pG->edges[pG->m].v = y;
    pG->m++;
}
```

Thứ ngày / /

5.
```
int degree(Graph* pG, int x) {
    int deg = 0;
    for(int e = 0; e <= pG->m; e++)
        if(pG->edges[e].u == x || pG->edges[e].v == x)
            deg++;
    return deg; }
```

6.
```
int adjacent(Graph* pG, int x, int y) {
    for(int e=1; e <= pG->m; e++) {
        if((pG->edges[e-1].u==x && pG->edges[e-1].v==y)
            || (pG->edges[e-1].u==y && pG->edges[e-1].v==x)
            return 1;
    return 0; }
```

7.
```
void neighbors(Graph G, int x) {
    for(int u=1; u <= G.n; u++) {
        if(adjacent(&G, x, u))
            printf("%d ", u);
    printf("\n"); }
```

Thứ ngày / /

Biểu diễn bằng ma trận kề

Graph: n [ ] m [ ]

A[][]

	1	2	3 ..	n
1				
2				
⋮				
n				

8.

A	1	2	3	4	5	6	7	8	9
1	0	1	1	1	0	0	0	0	0
2	1	0	1	0	0	1	0	0	0
3	1	1	0	0	1	0	1	0	0
4	1	0	0	0	0	0	0	0	0
5	0	0	1	0	0	1	1	0	0
6	0	1	0	0	1	0	0	0	1
7	0	0	1	0	1	0	0	1	0
8	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	1	0	0	0

9.

A	1	2	3	4	5	6	7	8	9
1	0	0	1	0	1	1	0	0	0
2	0	0	0	0	1	1	0	0	0
3	1	0	0	1	1	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	1	1	1	0	0	1	0	0	0
6	1	1	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0

n [9]

m [9]

Graph

Thứ ngày / /

```
#define MAX 500
typedef struct {
    int A[MAX][MAX];
    int n, m;
} Graph;
```

10.
```
void init_graph (Graph *pG, int n) {
    pG->n = n;
    pG->m = 0;
    for (int i = 1; i <= pG->n; i++)
        for (int j = 1; j <= pG->m; j++)
            pG->A[i][j] = 0; }
```

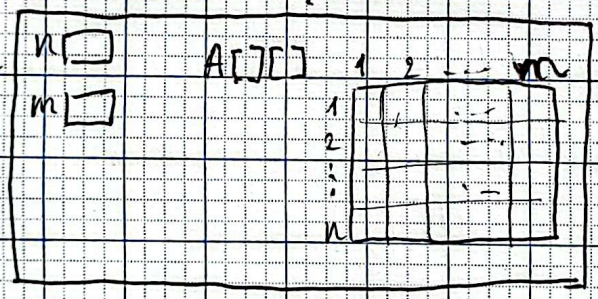
11.
```
void add_edge (Graph *G, int x, int y) {
    G->n = n;        G->A[x][y] = 1;
    G->m = 0;        G->A[y][x] = 1;
    G->m ++; }
```

12.
```
int degree (Graph G, int x) {
    int deg = 0;
    for (int i = 1; i <= G.n; i++) {
        if (G.A[i][x] == 1) deg++;
    return deg; }
```

13.
```
int adjacent (Graph G, int x, int y) {
    return G.A[x][y] == 1; }
```

14.
```
void neighbors (Graph G; int x) {
    for (int u = 1; u <= G.n; u++)
        if ( G.A[u][x] ) printf ("%d", u); }
```

Biểu diễn bằng ma trận liên thuộc (ma trận đỉnh - cung)



15.

	1	2	3	4	5	6	7	8
1	1	1	1		1			
2	1							
3		1	1	1		1		
4					1	1		
5							1	1
6				1			1	1
7								
8								

16.

A	1	2	3	4	5	6	7	8	9	10
1	1	1	1	0	0	0	0	0	0	0
2	0	0	0	1	1	0	0	0	0	1
3	1	0	0	0	0	1	1	0	0	0
4	0	0	0	0	0	1	0	0	0	1
5	0	1	0	1	0	0	1	1	0	0
6	0	0	1	0	1	0	0	1	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	0	1	0

n = 8 m = 10

Graph

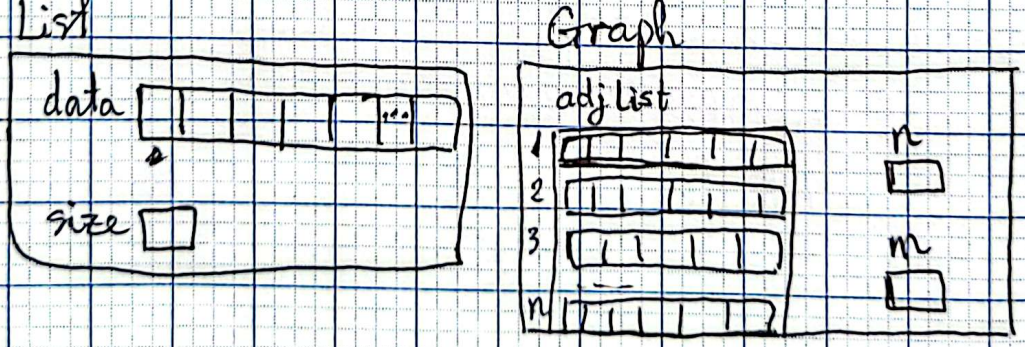
Thứ ngày / /

```
#define MAXM 500
#define MAXN 100
typedef struct {
    int A[MAXN][MAXM];
    int n, m;
} Graph;
17. void init_graph (Graph* G, int n, int m) {
    G->n = n;
    G->m = 0;
    for (int i = 1; i <= G->n; i++)
        for (int j = 1; j <= G->m; j++)
            G->A[i][j] = 0;
}
18. void add_edge (Graph* G, int e, int x, int y) {
    G->A[x][e] = 1;
    G->A[y][e] = 1;
    G->m++;
}
19. int degree (Graph G, int x) {
    int deg = 0;
    for (int e = 1; e <= G.m; e++)
        if (G.A[x][e] == 1) deg++;
    return deg;
}
```

20.
```
int adjacent(Graph G, int x, int y) {
    for(int e = 1; e <= G.m; e++)
        if(G.A[x][e] == 1 && G.A[y][e] == 1)
            return 1;
    return 0; }
```

21.
```
void neighbors(Graph G, int x) {
    for(int u = 1; u <= G.n; u++)
        if(adjacent(G, u, x))
            printf("%d", u);
    printf("\n"); }
```

Biểu diễn bằng danh sách đỉnh kề



Thứ ngày / /

22.

n = 9

m = 9

(1) = {3, 5, 6}

(2) = {5, 6}

(3) = {1, 4, 5}

(4) = {3}

(5) = {1, 2, 3, 6}

(6) = {1, 2, 5}

(7) = {8}

(8) = {7}

(9) = { }

23. Graph

adjList

1	2 5 6 3 4
2	5 4
3	1 4 5
4	3 1
5	1 2 3 6
6	9 2 5
7	8 3 5
8	7
9	6

n = 9

m = 9

Thứ ngày / /

```
#define MAX 200
typedef struct {
    int data[MAX];
    int size;
} List;
typedef struct {
    List adjList[MAX];
    int n, m;
} Graph;
```

24.
```
void init_graph(Graph* G, int n, int m) {
    G->n = n;
    G->m = m;
    for(int u = 1; u <= G->n; u++) {
        make_null(&G->adjList[u]); } }
```

25.
```
void add_edge(Graph* G, int x, int y) {
    push_back(&G->adjList[x], y);
    push_back(&G->adjList[y], x);
    G->m ++; }
```

Thứ ngày / /

26. int degree(Graph * G, int x) {
~~int deg = 0;~~ return G→adjList[x].size; }
~~for (~~

27. int adjacent(Graph * G, int x, int y) {
~~for~~ for(int u = 1; u <= G.n; u++) {
if(element_at(G→adjList[x], u) == y)
return 1;
return 0; }

28. void neighbors(Graph * G, int x) {
for(int u = 1; u <= G→n; u++)
printf("%d ", G→adjList[x].data[u]);
printf("\n"); }