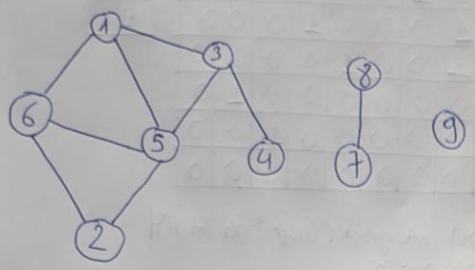
Trần Nguyễn Nhật Huy
B2113333
STT: 56

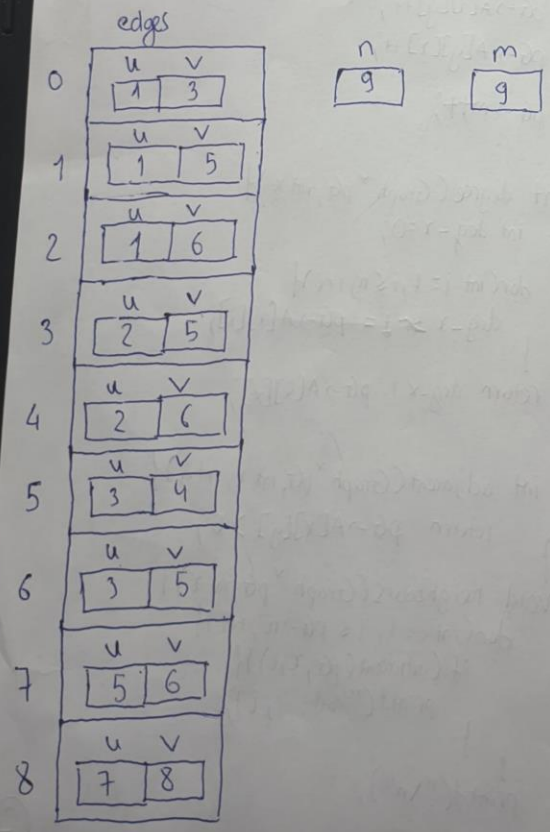
Biểu diễn bằng danh sách cạnh



1. Các cạnh của đồ thị trên là:

(1,3)	(3,4)
(1,5)	(3,5)
(1,6)	(5,6)
(2,5)	(7,8)
(2,6)	

2. Sơ đồ tổ chức dữ liệu:



Bài tập lập trình

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

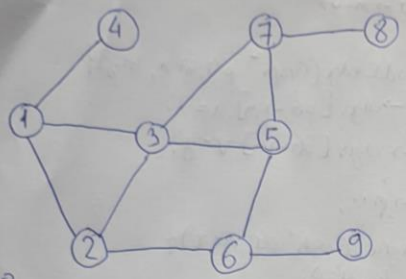
4.
```
void add_edge(Graph* pG, int x, int y){
    pG->edges[pG->m].u = x;
    pG->edges[pG->m].v = y;
    pG->m++;
}
```

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

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

7.
```
void neighbours(Graph *pG, int x){
    for(int i=1; i<=n; ++i){
        if(adjacent(pG, x, i){
            printf("%d ", i);
        }
    }
    printf("\n");
}
```

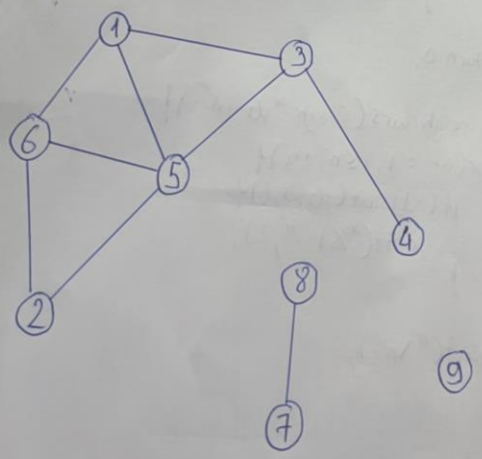
Biểu diễn bằng ma trận kề (ma trận đỉnh-đỉnh)



8. Ma trận kề của đồ thị trên là:

	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	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	1	1	0	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

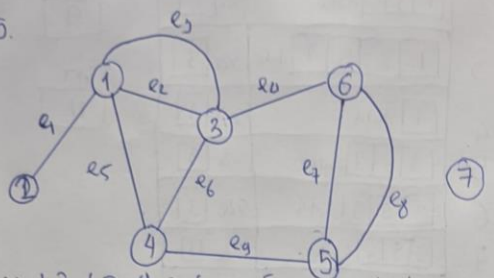
n = 9

m = 9

```
10. void init_graph(Graph *pG, int n){
      pG->n = n;
      pG->m = 0;
      for(int i=1;i<=n;++i){
        for(int j=1;j<=n;++j){
          pG->A[i][j]= 0;
        }
      }
    }
11. void add_edge(Graph *pG, int x, int y){
      pG->A[x][y]++;
      pG->A[y][x]++;
      pG->m++;
    }
12. int degree(Graph *pG, int x){
      int deg_x=0;
      for(int i=1;i<=n;++i){
        deg_x += pG->A[x][i];
      }
      return deg_x + pG->A[x][x];
    }
13. int adjacent(Graph *pG, int x, int y){
      return pG->A[x][y] > 0;
    }
14. void neighbours(Graph *pG, int x){
      for(int i=1; i<= pG->n;++i){
        if (adjacent(pG, x, i)){
          printf("%d ", i);
        }
      }
      printf("\n");
    }
```

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

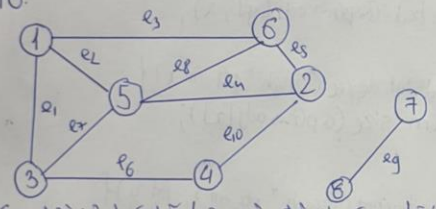
15.



Ma trận liên thuộc tương ứng với đồ thị trên:

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

16.



Sơ đồ tồ chứ dữ liệu của đồ thị sau là:

	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	1
4	0	0	0	0	0	1	0	0	0	0
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

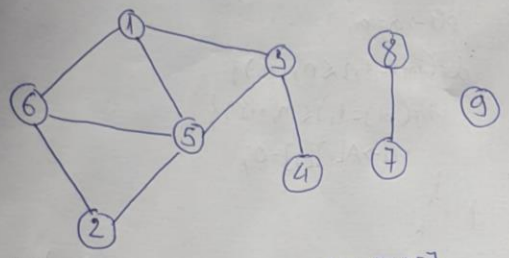
A

n
8

m
10

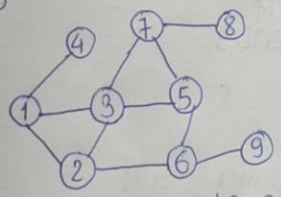
```
17. void init_graph(Graph *pG, int n, int m){
      pG->n = n;
      pG->m = m;
      for(int i=1; i<=n; ++i){
        for(int j=1; j<=m; ++j){
          pG->A[i][j]=0;
        }
      }
    }
18. void add_edge(Graph* pG, int e, int x, int y){
      pG->A[x][e]=1;
      pG->A[y][e]=1;
    }
19. int degree(Graph *pG, int x){
      int deg_x=0;
      for(int i=1; i<=m; ++i){
        deg_x += pG->A[x][i];
      }
      return deg_x;
    }
20. int adjacent(Graph *pG, int x, int y){
      for(int i=1; i<=m; ++i){
        if(pG->A[x][i]==1 && pG->A[y][i]==1){
          return 1;
        }
      }
      return 0;
    }
21. void neighbours(Graph *pG, int x){
      for(int i=1; i<=n; ++i){
        if(adjacent(pG, x, i){
          printf("%d ", i);
        }
      }
      printf("\n");
    }
```

22. Biểu diễn bằng danh sách đỉnh kề:



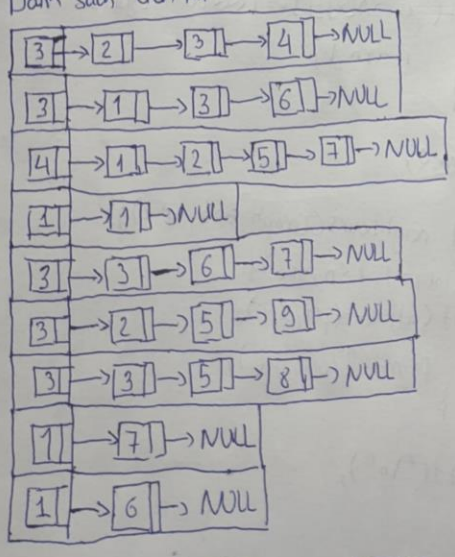
adj[1] = [3,5,6]
adj[2] = [5,6]
adj[3] = [1,4,5]
adj[4] = [3]
adj[5] = [1,2,3,6]
adj[6] = [1,2,5]
adj[7] = [8]
adj[8] = [7]
adj[9] = []

23



Sơ đồ tổ chức dữ liệu của đồ thị trên:

* Danh sách liên kết:



adj | 9 (n) | 11 (m)

* Danh sách đặc


```
24. void init_graph(Graph *pG, int n, int m){
    for(int i=1; i<=n; ++i){
        make_null(&pG->adj[i]);
    }
    pG->n = n;
    pG->m = m;
}
25. void add_edge(Graph *pG, int x, int y){
    push_back(&pG->adj[x], y);
    push_back(&pG->adj[y], x);
}
26. ~~void~~ int degree(Graph *pG, int x){
    return size(&pG->adj[x]);
}
27. int adjacent(Graph *pG, int x, int y){
    for(int i=1; i<=~~size(pG)~~ degree(pG,x); ++i){
        size(
        if(element_at(&pG->adj[x], i) == y){
            return 1;
        }
    }
    return 0;
}
28. void neighbours(Graph *pG, int x){
    for(int i=1; i<=degree(pG,x); ++i){
        printf("%d ", element_at(&pG->adj[x], i));
    }
    printf("\n");
}
```