```c
// Can Da (Ap dung xep hang do thi)
#include <stdio.h>

#define MAX_VERTICES 100

// List
typedef struct
{
    int data[MAX_VERTICES];
    int size;
} List;

void make_null_list(List *L)
{
    L->size = 0;
}

void push_back(List *L, int x)
{
    L->data[L->size] = x;
    ++L->size;
}

int element_at(List *L, int i)
{
    return L->data[i - 1];
}

void copy_list(List *s1, List *s2)
{
    make_null_list(s1);

    int i;
    for (i = 1; i <= s2->size; ++i)
    {
        push_back(s1, element_at(s2, i));
    }
}

// Queue
typedef struct
{
    int data[100];
    int front, rear;
} Queue;

void make_null_queue(Queue *Q)
{
    Q->front = 0;
    Q->rear = -1;
}

void push(Queue *Q, int x)
{
    ++Q->rear;
    Q->data[Q->rear] = x;
}

int top(Queue *Q)
{
    return Q->data[Q->front];
}

void pop(Queue *Q)
{
    ++Q->front;
}

int empty(Queue *Q)
{
    return Q->front > Q->rear;
}

// Graph
typedef struct
{
    int A[MAX_VERTICES][MAX_VERTICES];
    int n;
} Graph;

void init_graph(Graph *G, int n)
{
    G->n = n;

    int i, j;
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= n; ++j)
        {
            G->A[i][j] = 0;
        }
    }
}

void add_edge(Graph *G, int x, int y)
{
    G->A[x][y] = 1;
}

int adjacent(Graph *G, int x, int y)
{
    return G->A[x][y];
}

List neighbors(Graph *G, int x)
{
    int y;
    List list;

    make_null_list(&list);

    for (y = 1; y <= G->n; ++y)
    {
        if (adjacent(G, x, y))
        {
            push_back(&list, y);
        }
    }

    return list;
}

int rank[MAX_VERTICES];

List topo_sort(Graph *G)
{
    int d[MAX_VERTICES];
    int x, u;
```

```
for (u = 1; u <= G->n; ++u)
{
    d[u] = 0;
}

for (x = 1; x <= G->n; ++x)
{
    for (u = 1; u <= G->n; ++u)
    {
        if (adjacent(G, x, u))
        {
            ++d[u];
        }
    }
}

List l;

Queue q;

make_null_list(&l);
make_null_queue(&q);

for (u = 1; u <= G->n; ++u)
{
    if (!d[u])
    {
        push(&q, u);
    }
}

int i;

while (!empty(&q))
{
    int u = top(&q);
    pop(&q);

    push_back(&l, u);

    List list = neighbors(G, u);
```

```
        for (i = 1; i <= list.size; ++i)
        {
            int v = element_at(&list, i);

            if (d[v] >= 0)
            {
                --d[v];
            }

            if (!d[v])
            {
                push(&q, v);
            }
        }
    }

    return l;
}

int main()
{
    Graph G;
    int n, m, u, v, w, e;
    scanf("%d%d", &n, &m);
    init_graph(&G, n);

    for (e = 0; e < m; e++)
    {
        scanf("%d%d", &u, &v);
        add_edge(&G, u, v);
    }

    List l = topo_sort(&G);

    int i;
    for (i = 1; i <= l.size; ++i)
    {
        printf("%d ", element_at(&l, i));
    }

    return 0;
}
```