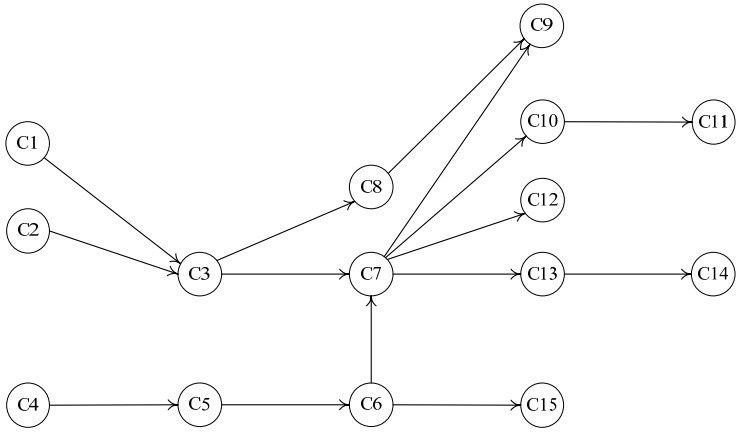
Topological Ordering



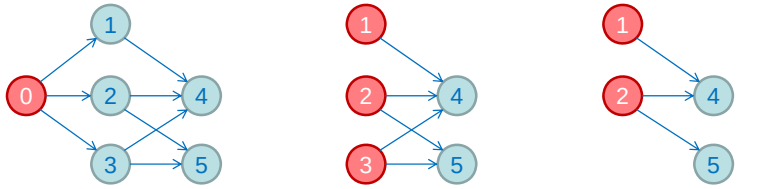
C1, C2, C4, C5, C3, C6, C8, C7, C10, C13, C12, C14, C15, C11, C9

C4, C5, C2, C1, C6, C3, C8, C15, C7, C9, C10, C11, C12, C13, C14

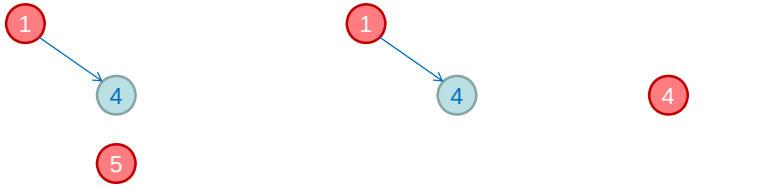
Topological Sort

```
Input the AOV network. Let n be the number of vertices.
for (i = 0; i < n; i++) /* output the vertices */
{
    if (every vertex has a predecessor) return;
        /* network has a cycle and is infeasible */
    pick a vertex v that has no predecessors;
    output v;
    delete v and all edges leading out of v;
}
```


Action of Topological Sort



(a) Initial (b) Vertex 0 deleted (c) Vertex 3 deleted



(d) Vertex 2 deleted (e) Vertex 5 deleted (f) Vertex 1 deleted

Topological order generated: 0, 3, 2, 5, 1, 4

Issues in Data Structure Consideration

- Decide whether a vertex has any predecessors.

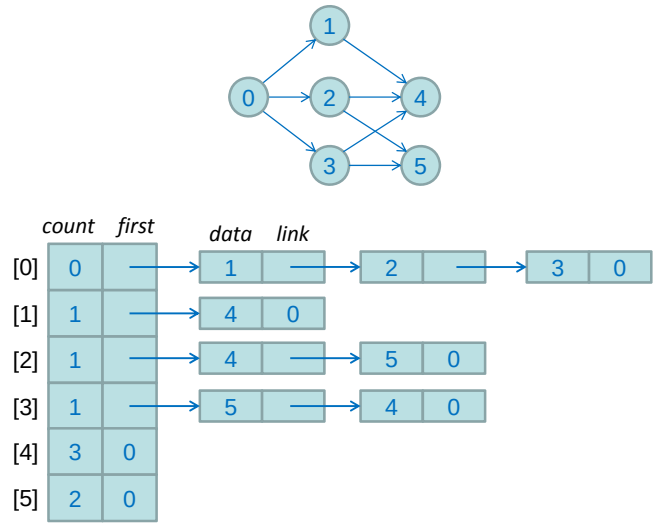
➡ Each vertex has a count.

- Delete a vertex together with all its incident edges.

➡ Adjacency list

```
typedef struct node *nodePointer;
typedef struct {
        int vertex;
        nodePointer link;
        } node;
typedef struct {
        int count;
        nodePointer link;
        } hdnodes;
hdnodes graph[MAX_VERTICES];
```

Adjacency List Representation



Topological Sort

```
void topSort(hdnodes graph[], int n)
{
    int i,j,k,top;
    nodePointer ptr;
    /* create a stack of vertices with no predecessors */
    top = -1;
    for (i = 0; i < n; i++)
        if (!graph[i].count) {
            graph[i].count = top;
            top = i;
        }
    for (i = 0; i < n; i++)
        if (top == -1) {
            fprintf(stderr,
                "\nNetwork has a cycle. Sort terminated. \n");
            exit(EXIT_FAILURE);
```

$O(n)$

No predecessors, stack is linked through count field.

Topological Sort

```
    } else {
        j = top; /* unstack a vertex */
        top = graph[top].count;
        printf("v%d, ",j);
        for (ptr = graph[j].link; ptr; ptr = ptr->link) {
            /* decrease the count of the succesor vertices of j */
            k = ptr->vertex;
            graph[k].count--;
            if (!graph[k].count) {
                /* add vertex k to the stack */
                graph[k].count = top;
                top = k;
            }
        }
    }
}
```

$O((\sum_{i=0}^{n-1} d_i) + n) = O(e + n)$

back1 back2

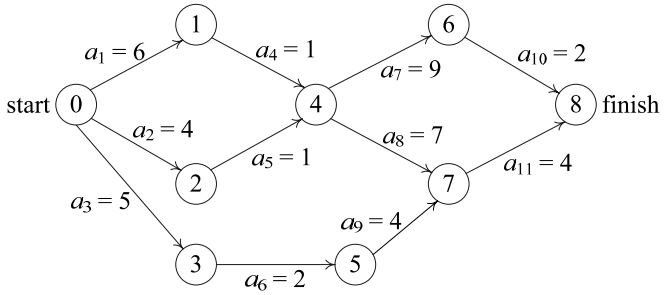
Outline

- Biconnected Components
- Activity-on-Vertex (AOV) Networks
- ***Activity-on-Edge (AOE) Networks***

Activity-on-Edge (AOE) Networks

- Directed edge: tasks or activities to be performed
- Vertex: events which signal the completion of certain activities
- Number: time required to perform the activity

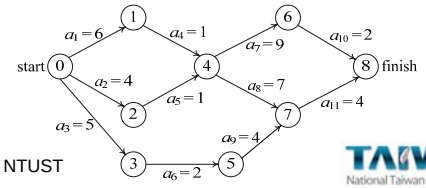
An AOE Network



Event	Interpretation
0	Start of project
1	Completion of activity a_1
4	Completion of activities a_4 and a_5
7	Completion of activities a_8 and a_9
8	Completion of project

Application of AOE Network

- Performance Evaluation
  - Minimum amount of time
  - Activity whose duration time should be shortened
  - …
- Critical Path
  - A path that has the longest length
  - Minimum time required to complete the project
  - v_0, v_1, v_4, v_7, v_8 or v_0, v_1, v_4, v_6, v_8 (18)



Other Factors (1/3)

- Earliest time that v_i can occur
  - The length of the longest path from v_0 to v_i
  - The earliest start time for all activities leaving v_i
  - $e(7) = e(8) = 7$