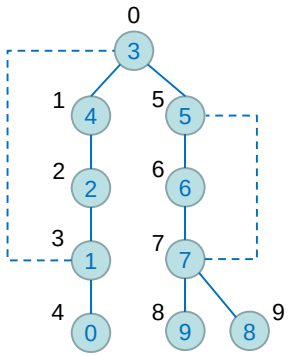
dfn and low (2/2)



vertex	dfn	low	child	lowChild	low:dfn
0	4	4			
1	3	0			
2	2	0			
3	0	0			
4	1	0			
5	5	5			
6	6	5			
7	7	5			
8	9	9			
9	8	8			

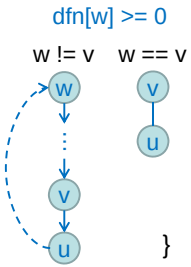
Global Declarations and Initialization

```
#define MIN2(x,y) ((x) < (y) ? (x) : (y))
short int dfn[MAX_VERTICES];
short int low[MAX_VERTICES];
int num;

void init(void)
{
    int i;
    for (i = 0; i < n; i++) {
        visited[i] = FALSE;
        dfn[i] = low[i] = -1;
    }
    num = 0;
}
```


Determining dfn and low

```
void dfnlow(int u, int v)                    Initial call: dfnlow(x, –1)
{ /* compute dfn and low while performing a dfs search
     beginning at vertex u, v is the parent of u (if any) */
     nodePointer ptr;
     int w;
     dfn [u] = low [u] = num++;              low[u] = min{dfn(u), …, …}
     for (ptr = graph[u]; ptr; ptr = ptr->link) {
          w = ptr->vertex;
          if (dfn[w] < 0) { /* w is an unvisited vertex */
               dfnlow(w,u);
               low[u] = MIN2(low[u], low[w]);
          }    low[u] = min{…, min{low(w) | w is a child of u}, …}
          else if (w != v)
               low[u] = MIN2(low[u], dfn[w]);
     }    low[u] = min{…, …, min{dfn(w) | (u,w) is a back edge}}
}
```



Biconnected Components of a Graph

```
void bicon(int u, int v)
{ /* compute dfn and low, and output the edges of G by their
     biconnected components, v is the parent (if any) of u in the
     resulting spanning tree. It is assumed that all entries of dfn[]
     have been initialized to -1, num is initially to 0, and the stack is
     initially empty */
     nodePointer ptr;
     int w, x, y;
     dfn[u] = low[u] = num++;                  low[u] = min{dfn(u), …, …}
     for (ptr = graph[u]; ptr; ptr = ptr->link) {
          w = ptr->vertex;                     (1) dfn[w] == –1: the first time
          if (v != w && dfn[w] < dfn[u])       (2) dfn[w] != –1: by back edge
               push(u,w); /* add edge to stack */
```

Biconnected Components of a Graph

```
    if (dfn[w] < 0) { /* w has not been visited */
        bicon(w, u);      low[u] = min{…, min{low(w) | w is a child of
        low[u] = MIN2(low[u], low[w]);                          u}, …}
        if (low[w] >= dfn[u]) {      articulation point
            printf("New biconnected component: ");
            do { /* delete edge from stack */
                pop(&x, &y);
                printf(" <%d, %d>", x, y);
            } while (!((x == u) && (y == w)));
            printf("\n");
        }
    } else if (w != v) low[u] = MIN2(low[u],dfn[w]);
  }             low[u] = min{…, …, min{dfn(w) | (u,w) is a back edge}}
}
```


Outline

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

Activity-on-Vertex (AOV) Networks (1/2)

- **Definition:** A directed graph G in which the vertices represent tasks or activities and the edges represent precedence relations between tasks is an activity-on-vertex (AOV) network.
- **Definition:** Vertex i in an AOV network G is a predecessor of vertex j iff there is a directed path from i to j. i is an immediate predecessor of j iff $<i, j>$ is an edge in G. If i is a predecessor of j, then j is a successor of i. If i is a immediate predecessor of j, then j is a immediate successor of i.

Activity-on-Vertex (AOV) Networks (2/2)

- **Definition:** A relation $\cdot$ is transitive iff it is the case that for all triples i, j, k, $i \cdot j$ and $j \cdot k \Rightarrow i \cdot k$. A relation $\cdot$ is irreflexive on a set S if for no element x in S is the case that $x \cdot x$. A precedence relation that is both transitive and irreflexive is a partial order.
- **Definition:** A topological order is a linear ordering of the vertices of a graph such that, for any two vertices i and j, if i is a predecessor of j in the network, then i precedes j in the linear ordering.