TOPOLOGICAL SORTING

```
#include <stdio.h>
#include <stdlib.h>

typedef struct AdjListNode {
    int dest;
    struct AdjListNode* next;
} AdjListNode;

typedef struct AdjList {
    AdjListNode* head;
} AdjList;

typedef struct Graph {
    int V;
    AdjList* array;
} Graph;

AdjListNode* newAdjListNode(int dest) {
    AdjListNode* newNode = (AdjListNode*)malloc(sizeof(AdjListNode));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

Graph* createGraph(int V) {
    Graph* graph = (Graph*)malloc(sizeof(Graph));
    graph->V = V;
    graph->array = (AdjList*)malloc(V * sizeof(AdjList));
```

```
    for (int i = 0; i < V; ++i)
        graph->array[i].head = NULL;
    return graph;
}

void addEdge(Graph* graph, int src, int dest) {
    AdjListNode* newNode = newAdjListNode(dest);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;
    newNode = newAdjListNode(src);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}

void printGraph(Graph* graph) {
    for (int v = 0; v < graph->V; ++v) {
        AdjListNode* pCrawl = graph->array[v].head;
        printf("\nAdjacency list of vertex %d\nhead", v);
        while (pCrawl) {
            printf(" -> %d", pCrawl->dest);
            pCrawl = pCrawl->next;
        }
        printf("\n");
    }
}

void DFSUtil(Graph* graph, int v, int visited[]) {
    visited[v] = 1;
    printf("%d ", v);
    AdjListNode* adjList = graph->array[v].head;
```

```
    while (adjList) {
        int connectedVertex = adjList->dest;
        if (!visited[connectedVertex])
            DFSUtil(graph, connectedVertex, visited);
        adjList = adjList->next;
    }
}

void DFS(Graph* graph, int startVertex) {
    int* visited = (int*)malloc(graph->V * sizeof(int));
    for (int i = 0; i < graph->V; i++)
        visited[i] = 0;
    DFSUtil(graph, startVertex, visited);
    free(visited);
}

int main() {
    int V = 5;
    Graph* graph = createGraph(V);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);

    printf("Graph adjacency list representation:\n");
    printGraph(graph);
```

```
    printf("\nDFS starting from vertex 0:\n");
    DFS(graph, 0);

    return 0;
}
```