

Depth-First Search and Breadth-First Search (DFS & BFS)

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

Outline

- 1 Introduction
- 2 Depth First Search (DFS)
- 3 Breadth-First Search (BFS)

Outline

1 Introduction

2 Depth First Search (DFS)

3 Breadth-First Search (BFS)

Elementary Graph Operations

Reachability

- **Given:** an undirected graph $G = (V, E)$, and a vertex $v \in V(G)$
- **Goal:** visit all vertices in G that are reachable from v .



Elementary Graph Operations

Reachability

- **Given:** an undirected graph $G = (V, E)$, and a vertex $v \in V(G)$
- **Goal:** visit all vertices in G that are reachable from v .
- The two ways of completing this task:
 - Depth-First Search (DFS)
 - Similar to the **preorder tree traversal**.
 - Breadth-Frist Search (BFS)
 - Similar to the **level-order tree traversal**.

Elementary Graph Operations

Reachability

- **Given:** an undirected graph $G = (V, E)$, and a vertex $v \in V(G)$
- **Goal:** visit all vertices in G that are reachable from v .
- The two ways of completing this task:
 - Depth-First Search (DFS)
 - Similar to the **preorder tree traversal**.
 - Breadth-Frist Search (BFS)
 - Similar to the **level-order tree traversal**.
- In the following discussion, we shall assume that the **linked adjacency list** representation for graphs is used.

Outline

1 Introduction

2 Depth First Search (DFS)

3 Breadth-First Search (BFS)

Depth First Search (DFS) (1/2)

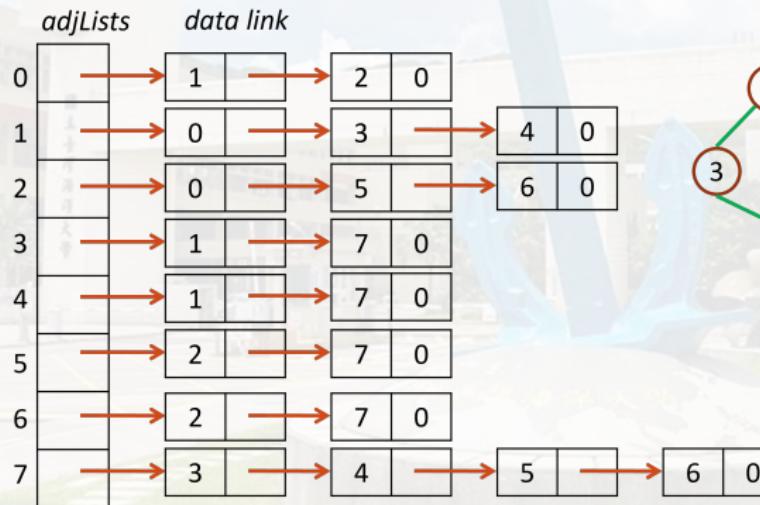
- We begin the search by visiting the start vertex, v .
- Next, we select an **unvisited** vertex, w , from v 's adjacency lists and carry out a DFS on w .
- We preserve our current position in v 's adjacency list by **placing it on a stack**.
- Eventually our search reaches a vertex, say u , that has no unvisited vertices on its adjacency list.

Depth First Search (DFS) (2/2)

- At this point, we remove a vertex from **stack** and continue processing its adjacency list.
- Previously visited vertices are **discarded**; unvisited vertices are visited and placed on the stack.
- The search terminates when the stack is empty.

DFS Example

- Using a stack and recursion.
- It resembles the preoder tree traversal.



- The DFS order: $v_0 \rightarrow v_1 \rightarrow v_3 \rightarrow v_7 \rightarrow v_4 \rightarrow v_5 \rightarrow v_2 \rightarrow v_6$.

The Pseudocode of DFS

```
DFS(G, u) {
    u.visited = True
    for each v in G.Adj[u]
        if v.visited == False
            DFS(G, v)
}

driving main () {
    for each u in G
        u.visited = false
    for each u in G
        DFS(G, u)
}
```

- Demo code in C++.

DFS in C

```
#define FALSE 0
#define TRUE 1
short int visited[MAX_VERTICES];
/* initializing to be FALSE for all */

void DFS(int v)  {
/* DFS beginning at vertex v */
    nodePointer w;
    visited[v] = true;
    printf("%5d",v);
    for(w = graph[v]; w; w = w->link)
        if (!visited[w->vertex])
            DFS(w->vertex);
}
```

Analysis of DFS

- For $G = (V, E)$ represented by an **adjacency list**, vertices adjacent to v can be determined in $|N(v)|$, where $N(v)$ denotes the set of vertices adjacent to v in G .

Analysis of DFS

- For $G = (V, E)$ represented by an **adjacency list**, vertices adjacent to v can be determined in $|N(v)|$, where $N(v)$ denotes the set of vertices adjacent to v in G .
 - Follow the chain of links, $O(1)$ for deriving each neighbor of v .
- DFS examines each node in the adjacency lists at most **once**, the time cost for the search is $O(e)$, where $e = |E|$.

Analysis of DFS

- For $G = (V, E)$ represented by an **adjacency list**, vertices adjacent to v can be determined in $|N(v)|$, where $N(v)$ denotes the set of vertices adjacent to v in G .
 - Follow the chain of links, $O(1)$ for deriving each neighbor of v .
- DFS examines each node in the adjacency lists at most **once**, the time cost for the search is $O(e)$, where $e = |E|$.
- For $G = (V, E)$ represented by an **adjacency matrix**, vertices adjacent to v can be determined in $O(n)$ time, where $n = |V|$.
 - One needs to scan the corresponding row of the adjacency matrix.

Analysis of DFS

- For $G = (V, E)$ represented by an **adjacency list**, vertices adjacent to v can be determined in $|N(v)|$, where $N(v)$ denotes the set of vertices adjacent to v in G .
 - Follow the chain of links, $O(1)$ for deriving each neighbor of v .
- DFS examines each node in the adjacency lists at most **once**, the time cost for the search is $O(e)$, where $e = |E|$.
- For $G = (V, E)$ represented by an **adjacency matrix**, vertices adjacent to v can be determined in $O(n)$ time, where $n = |V|$.
 - One needs to scan the corresponding row of the adjacency matrix.
- Total time: $O(n^2)$.

Outline

1 Introduction

2 Depth First Search (DFS)

3 Breadth-First Search (BFS)



Breadth First Search (BFS) (1/2)

- The algorithm starts at vertex v and marks it as visited.
- Then visiting each of the vertices on v 's adjacency list.
- When we have visited all the vertices on v 's adjacency list, we visit all the unvisited vertices that are adjacent to the first vertex on v 's adjacency list.
- To implement this scheme, as we visit each vertex we place the vertex in a **queue**.

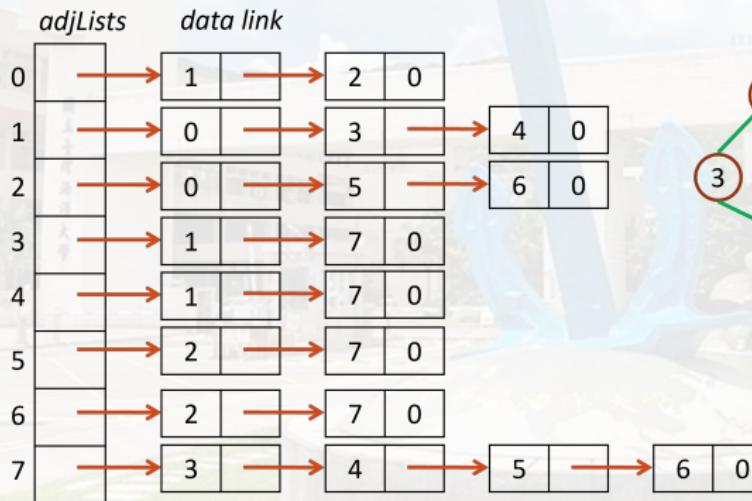
Breadth-First Search (BFS) (2/2)

- When we have exhausted an adjacency list, we remove a vertex from the queue and proceed by examining each of the vertices on its adjacency list.
- Unvisited vertices are visited and placed on the queue; visited are ignored.
- Finish the search when the queue is empty.



BFS Example

- Using a queue.
- It resembles the level-order tree traversal.



- The DFS order: $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7$.

The Pseudocode of BFS

```
BFS(G, u) { // let Q be the queue
    Q.enqueue(u)
    u.visited = true
    while (Q.empty() == false) { // when Q is not empty
        v = dequeue(Q)
        for all w in N(v) {
            if (w.visited == false) {
                Q.enqueue(w)
                w.visited = true
            }
        }
    }
}
driving main () {
    for each u in G
        u.visited = false
    BFS(G, u)
}
```

BFS in C

```
void bfs(int v) {
    nodePointer w;
    front = rear = NULL; /* initialize queue */
    printf("%5d",v);
    visited[v] = true;
    enqueue(v);
    while (front) {
        v = dequeue();
        for (w = graph[v]; w ; w->link)
            if (!visited[w->vertex]) {
                printf("%5d", w->vertex);
                enqueue(w->vertex);
                visited[w->vertex] = true;
            }
    }
}
```

- Demo code in C++.

Analysis of BFS

- Since each vertex is placed on the queue **exactly once**, the while loop is iterated at most n times.



Analysis of BFS

- Since each vertex is placed on the queue **exactly once**, the while loop is iterated at most n times.
 - For the **adjacency list** representation, this loop has a total cost of $d_0 + d_1 + \dots + d_{n-1} = O(e)$, where $d_i = \text{degree}(v_i)$.

Analysis of BFS

- Since each vertex is placed on the queue **exactly once**, the while loop is iterated at most n times.
 - For the **adjacency list** representation, this loop has a total cost of $d_0 + d_1 + \dots + d_{n-1} = O(e)$, where $d_i = \text{degree}(v_i)$.
- For the **adjacency matrix** representation, the while loop takes $O(n)$ time for each vertex visited.
 - Therefore, the total time is $O(n^2)$.

As was true of DFS, all vertices visited, together with all edges incident to them, form a **connected component** of G .

- Demo code in C++.



Discussions