Breadth-First Search

(BFS)

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

Outline

Outline

Breadth First Search (BFS) (1/2)

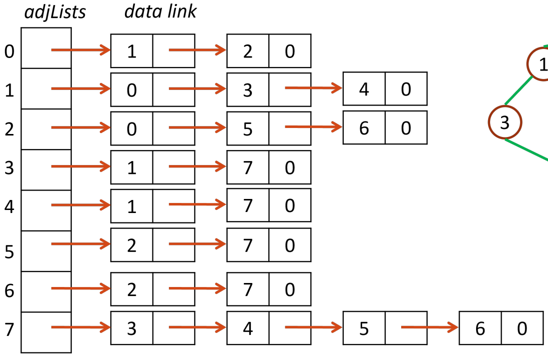
- The algorithm starts at vertex v and marks it as visited.
- Then visiting each of the vertices on v's adjacency list.
- When we have visited all the vertices on v's adjacency list, we visit all the unvisited vertices that are adjacent to the first vertex on v's adjacency list.
- To implement this scheme,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

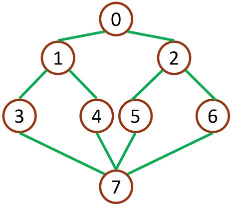
Breadth-First Search (BFS) (2/2)

- 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 and recursion.
  - It resembles the level-order tree traversal.





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

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)
}
```

DFS in C

```c
#define FALSE 0
#define TRUE 1
short int visited[MAX_VERTICES];
/* intializing 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: $O(n^2)$.

Discussions