6.2 Elementary Graph Operations

Graph Operations

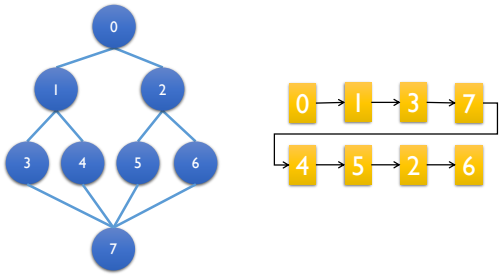
- Graph traversal
  - Depth-first search
  - Breadth-first search
- Connected components
- Spanning trees

6.2.1

Depth-First Search (DFS)

- Starting from a vertex v
  - Visit the vertex v ⇒ DFS(v).
  - For each vertex w adjacent to v, if w is not visited yet, then visit w ⇒ DFS(w).
  - If a vertex u is reached such that all its adjacent vertices have been visited, we go back to the last visited vertex.
- The search terminates when no unvisited vertex can be reached from any of the visited vertices.

Depth-First Search (DFS)



32

Recursive DFS

```
void Graph::DFS(void){
   visited = new bool[n]; // this is a data member of Graph
   fill(visited, visited+n, false);
   DFS(0); // start search at vertex 0
   delete [] visited;
}

void Graph::DFS(const int v){
   // visit all previously unvisited vertices that are adjacent to v
   output(v);
   visited[v]=true;
   for(each vertex w adjacent to v)
     if(!visited[w]) DFS(w);
}
```

33

Non-Recursive DFS

```
void Graph::DFS(int v){
   visited = new bool[n]; // this is a data member of Graph
   fill(visited, visited+n, false);
   Stack<int> s;          // declare and init a stack
   s.Push(v);
   while(!s.IsEmpty()){
      v = s.Top(); s.Pop();
      if(!visited[v]){
         output(v);
         visited[v]=true;
         for(each vertex w adjacent to v)
           if(!visited[w]) s.Push(w);
      }
   }
}
```

34

DFS Complexity

- Adjacency matrix
  - Time to determine all adjacent vertices: $O(n)$
  - At most n vertices are visited:
    $O(n \cdot n) = O(n^2)$
- Adjacency lists
  - There are $n + 2e$ chain nodes
  - Each node in the adjacency list is examined at most once. Time complexity = $O(e)$

35

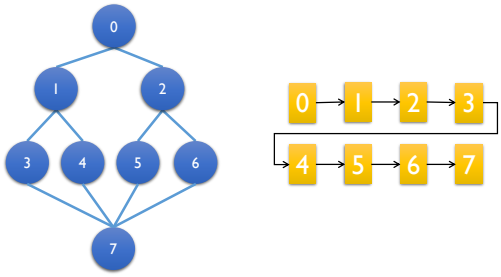
6.2.2

Breadth-First Search (BFS)

- Starting from a vertex v
  - Visit the vertex v .
  - Visit all unvisited vertices adjacent to v.
  - Unvisited vertices adjacent to these newly visited vertices are then visited and so on…

36

Breadth-First Search (BFS)



37

BFS: Implementation

```
void Graph::BFS(int v){
   visited = new bool[n]; // this is a data member of Graph
   fill(visited, visited+n, false);
   Queue<int> q;          // declare and init a queue
   q.Push(v);
   visited[v]=true;
   while(!q.IsEmpty()){
      v = q.Front(); q.Pop();
      output(v);
      for(each vertex w adjacent to v){
         if(!visited[w]){
           q.Push(w);
           visited[w]=true;
         }
      }
   }
   delete [] visited;
}
```

Time complexity is the same as DFS

38

6.2.3

Connected Components

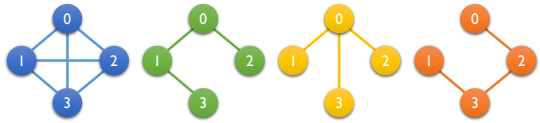
- How to determine whether a graph is connected or not?
  - Call DFS or BFS once and check if there is any unvisited vertices, if Yes, then the graph is not connected.
- How to identify connected components
  - Make a repeated calls to DFS or BFS.
  - Each call will output a connected component.
  - Start next call at an unvisited vertex.

39

6.2.4

Spanning Trees

- Definition: Any tree consists of solely of edges in $E(G)$ and including all vertices of $V(G)$.
- Number of tree edges is $\boldsymbol{n-1}$.
- Add a non-tree edge will create a **cycle.**

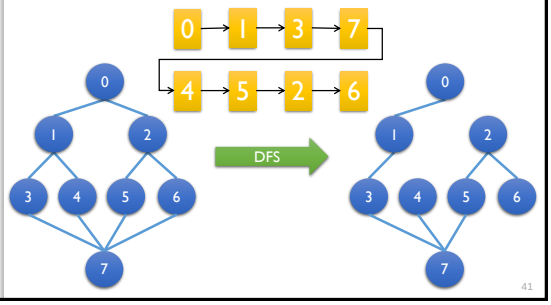


Complete graph Possible spanning trees

40

DFS Spanning Tree

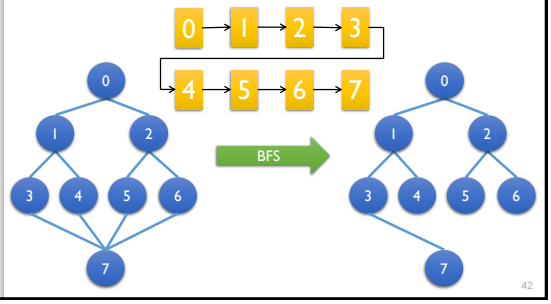
- Tree edges are those edges met during the traversal.



41

BFS Spanning Tree

- Tree edges are those edges met during the traversal.



42