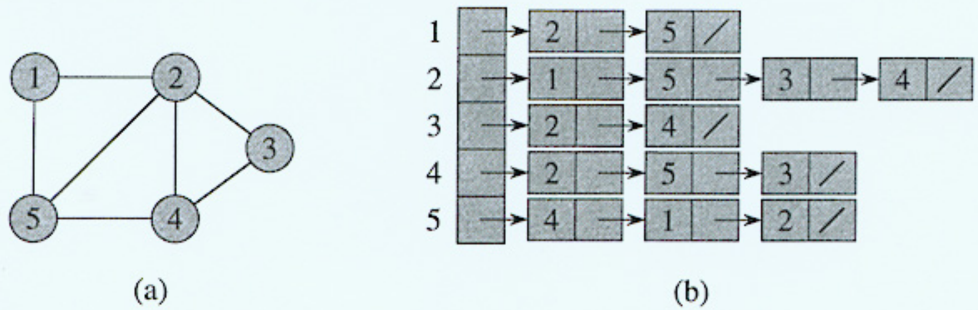
Data Structures

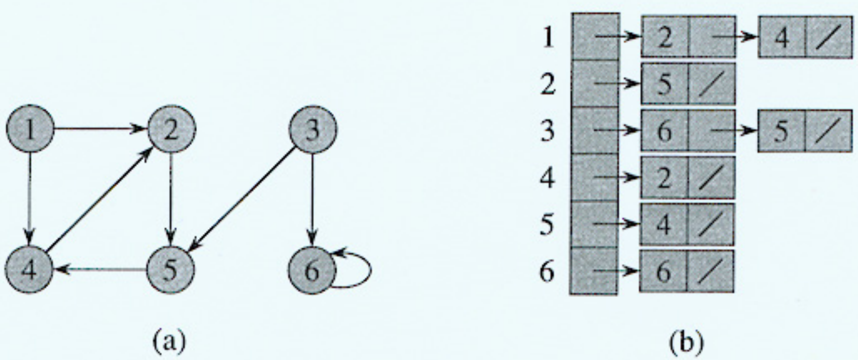
Giri Narasimhan
Office: ECS 254A
Phone: x-3748
giri@cs.fiu.edu



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G having five vertices and seven edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G having six vertices and eight edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Depth-First Search

- Preorder traversal
  - Start at some vertex, v
  - Recursively traverse all vertices adjacent to v
- DFS: Generalization of above for arbitrary graphs
  - Start at some vertex, v
  - Recursively traverse all unvisited vertices adjacent to v
  - We assume that for undirected graphs every edge (v, w) appears twice in adjacency lists: as (v, w) and as (w, v)

DFS Pseudocode

```
void dfs (Vertex v) {
	v.visited = true;
	for each Vertex w adjacent to v
		if ( !w.visited)
			dfs(w);
}
```

DFS Improved Pseudocode

```
void DFS (Vertex s) {
        DFScount = 1;
        s.DFSnum = DFScount;
        dfs(s);
}

void dfs (Vertex v) {
        v.visited = true;
        v.DFSnum = DFScount++;
        processVertex(v);
        for each Vertex w adjacent to v {
                processEdge(v,w);
                if ( !w.visited )
                        dfs(w);
        }
}
```

Connected Components

- Given an undirected graph G(V,E), a **connected component** is a maximal connected subgraph such that there is a path between any pair of vertices.

- How to compute all connected components
  - Perform DFS or BFS from arbitrary vertex v
  - All visited vertices and edges are in the same component
  - If all vertices have not been visited then
    - restart from unvisited vertex
  - Number of connected components = number of starts
  - Directed graphs need a different strategy

Relations

- A **relation R** is defined on a set S if
  - for every pair of elements (a, b), a, b ∈ S, **aRb** is either true or false.
- An equivalence relation is a relation R that satisfies:
  - **(Reflexive)** **aRa**, for all a ∈ S.
  - **(Symmetric)** **aRb** if and only if **bRa**.
  - **(Transitive)** **aRb** and **bRc** implies that **aRc**.
- Examples:
  - The ≤ relationship is
    - reflexive, transitive, not symmetric; not equivalence
  - Electrical connectivity is
    - an equivalence relation – reflexive, symmetric, and transitive
- Related : **aRb**; Equivalence Class : aEb

Dynamic Equivalence Relation

- Given n entities, we want to dynamically maintain a set of (equivalence) relationships
- We need data structure DisjointSet with 2 operations
  - find(a): returns the equivalence class for a
  - union(a, b): adds a relationship between a and b, if needed

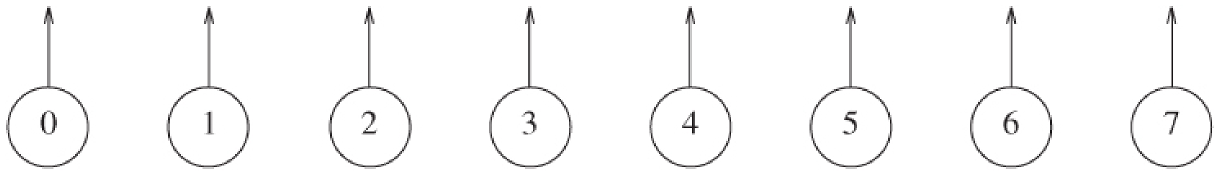


Figure 8.1 Eight elements, initially in different sets

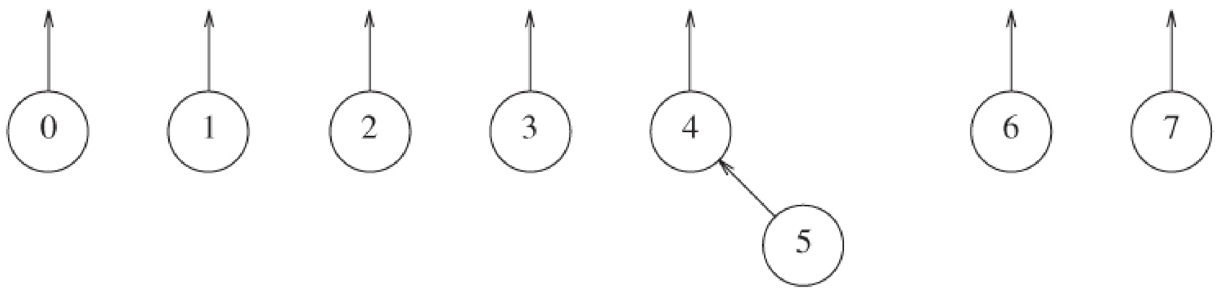


Figure 8.2 After `union(4,5)`

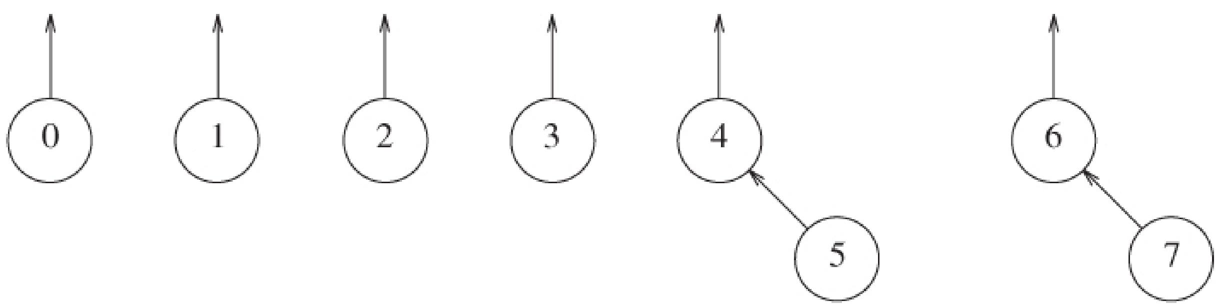
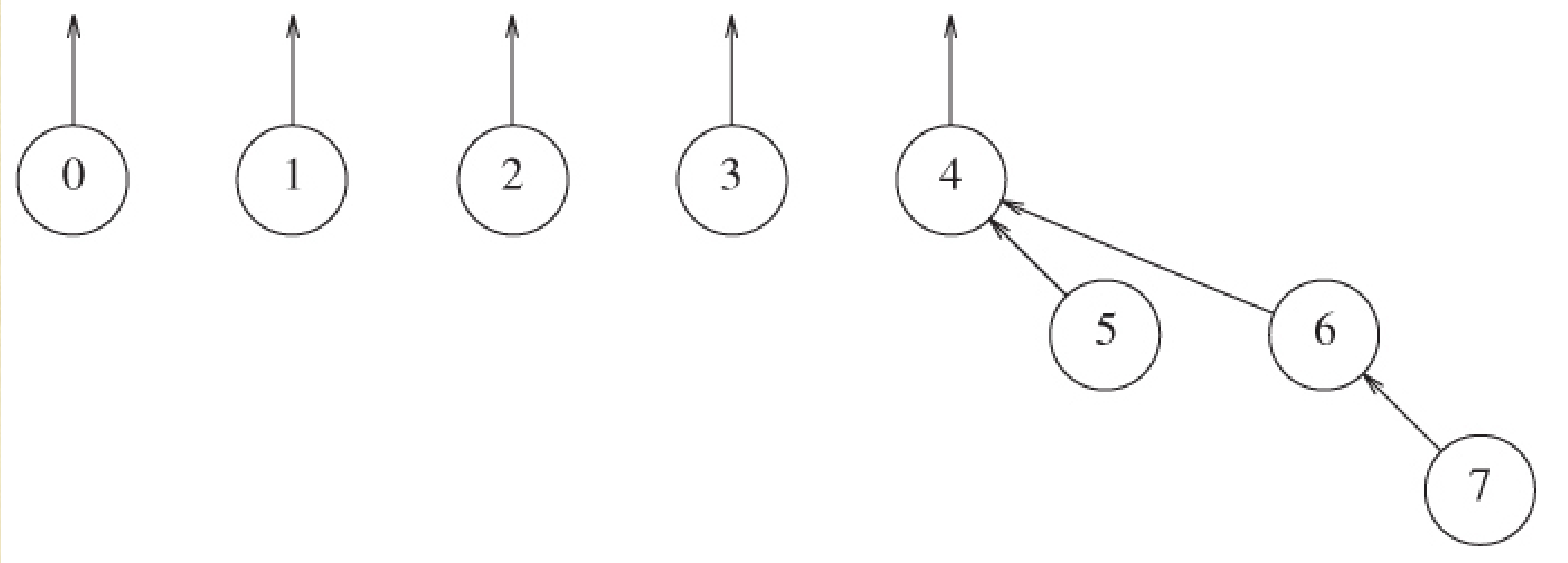


Figure 8.3 After `union(6,7)`

Union(4,6)



Disjoint Sets interface

```
public class DisjSets {
    public DisjSets (int numElements) // Figure 8.7
    public void union (int root1, int root2) // Figs 8.8, 8.14
    public int find (int x) // Figs 8.9, 8.16
}

public int find( int x) { // s[x] is height of tree rooted at x
        if (s[x] < 0) return x;
        else return find(s[x]);
}
public void union(int root1, int root2) {
        s[root2] = root1;
}
```

Improvements

- Height heuristic
  - If 2 disjoint sets are to be unioned, then always make the root of the taller tree to be root of entire tree.

```
public void union (int root1, int root2) {
    if (s[root2] < s[root1]) s[root1] = root2;
    else {
            if (s[root1] == s[root2]) s[root1]--;
            s[root2] = root1;
    }
}
```

- Depth of trees is at most O(log n)
- M operations take O(M log n)

2^{nd} Improvement

- Path Compression
  - Every time a find operation is performed on node x, all vertices along the path form x to its root are connected directly to the root, thus compressing the paths that have been recently visited
  - Time Complexity = O(M log*n) = O(M α(n))