Unit 2.3 Sets and Graphs

Algorithms

EE/NTHU

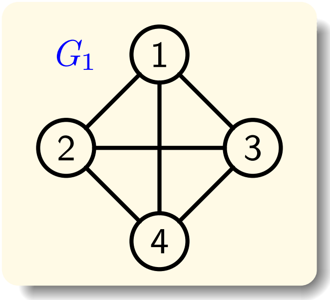
Mar. 14, 2019

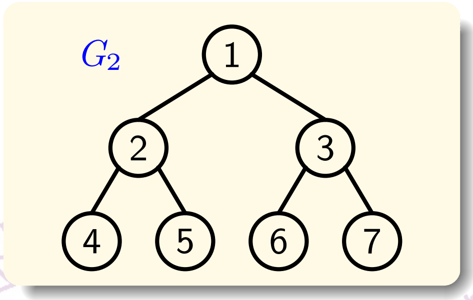
Graphs

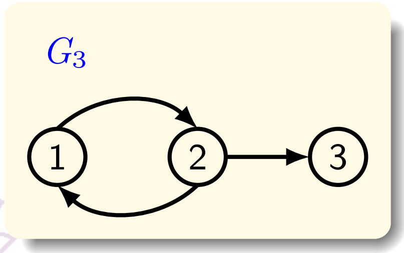
- A graph, G, consists of two sets V and E.
  - The set V is a finite, nonempty set of vertices.
  - The set E is a set of pairs of vertices; these pairs are called edges.
  - They are also denoted by $V(G)$ and $E(G)$.
  - And the graph is also denoted by $G(V, E)$.
- In an undirected graph the pair of vertices representing any edge is unordered.
  - Thus, the pairs (u, v) and (v, u) represent the same edge.
- In a directed graph each edge is represented by a direct pair $\langle u, v\rangle$; u is the tail and v is the head.
  - And, $\langle u, v\rangle$ and $\langle v, u\rangle$ represent two different edges.
  - In a direct graph the edges are drawn as arrows and they are drawn from tail to head.

Graph Examples

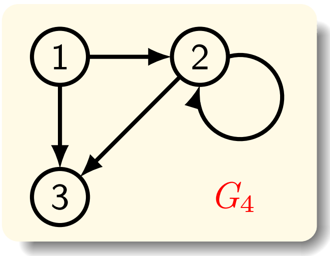
- Examples

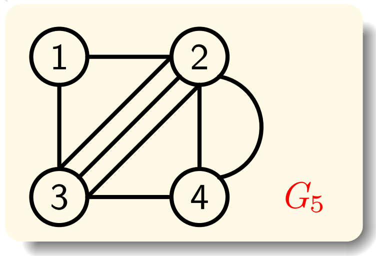






- Note that G_1 and G_2 are undirected graphs; G_3 is a directed graph.
- G_1 is a complete graph; G_2 is a tree.
- The following graphs are not studied in our classes:
  - Graphs with self-edges, which have edges connecting the same vertex.
  - Multi-graph, which has multiple edges between the same two vertices.





Adjacency

- The number of distinct unordered pairs (u, v) with $u \neq v$ in a graph with n vertices is $n(n-1)/2$.
  - This is the maximum number of edges in any n-vertex, undirected graph.
  - An n-vertex, undirected graph with exactly $n(n-1)/2$ edges is said to be complete.
  - In case of a direct graph with n vertices, the maximum number of edges $n(n-1)$.
- If (u, v) is an edge in $E(G)$, the we say vertices u and v are adjacent and edge (u, v) is incident on vertices u and v.
- If $\langle u, v \rangle$ is a directed edge, the vertex u is adjacent to v and v is adjacent from u.
  - The edge $\langle u, v \rangle$ is incident to u and v.
- A subgraph of $G = (V, E)$ is a graph $G' = (V', E')$ such that $V'(G') \subseteq V(G)$ and $E'(G') \subseteq E(G)$.

Paths and Cycles

- A path from vertex u to vertex v in a graph $G = (V, E)$ is a sequence of vertices $u, i_1, i_2, \cdots, i_k, v$, such that (u, i_1), (i_1, i_2), $\cdots$, (i_k, v) are edges in $E(G)$.
  - If $G = (V, E)$ is directed, then the path consists of the edges $\langle u, i_1 \rangle$, $\langle i_1, i_2 \rangle$, $\cdots$, $\langle i_k, v \rangle$ in $E(G)$.
- The length of a path is the number of edges on it.
- A simple path is a path in which all vertices except possibly the first and the last are distinct.
- A cycle is a simple path in which the first and the last vertices are the same.
  - A directed cycle is a cycle in a directed graph.
- In an undirected graph $G = (V, E)$, two vertices u and v are said to be connected if and only if there is a path in G from u to v.
- An undirected graph is said to be connected if and only if for every pair of distinct vertices u and v in $V(G)$, there is a path from u to v.
- A connected component or simply a component H of an undirected graph is a maximal connected subgraph.
  - By maximal we mean that G contains no other subgraph that is both connected and properly contains H.

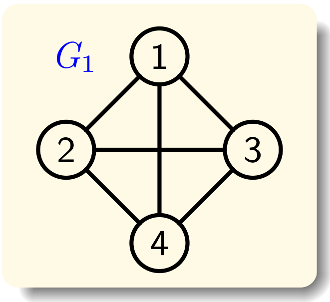
Graph Degrees

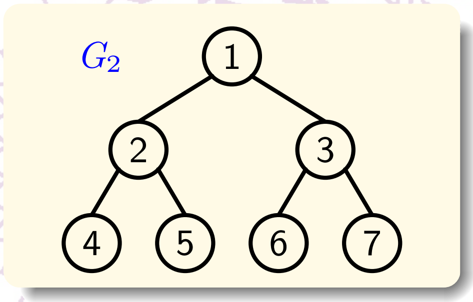
- A tree is a connected acyclic (contain no cycles) graph.
- A directed graph $G = (V, E)$ is said to be strongly connected if and only if for every pair of distinct vertices u and v in $V(G)$, there is a directed path from u to v and also from v to u.
- A strongly connected component is a maximal subgraph that is strongly connected.
- The degree of a vertex is the number of edges incident to that vertex.
- If $G = (V, E)$ is a directed graph, we define the in-degree of a vertex v to be the number of edges for which v is the head.
  - The out-degree is defined to be the number of edges for which v is the tail.
- If d_i is the degree of vertex i in a graph $G = (V, E)$ with n vertices, then the number of edge is

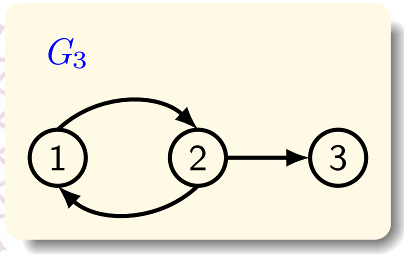
$$e = \left(\sum_{i=1}^{n} d_i \right) / 2. \tag{2.3.1}$$

Graph Representation – Adjacency Matrix

- Let $G = (V, E)$ be a graph with n vertices, $n \geq 1$. The adjacency matrix A is a two-dimensional $n \times n$ matrix with the property that $A[i,j] = 1$ if and only if the edge (i,j) ($\langle i,j \rangle$ for a directed graph) is in $E(G)$. $A[i,j] = 0$ if there is no such edge in $E(G)$.







$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Graph Representation – Adjacency Matrix, II

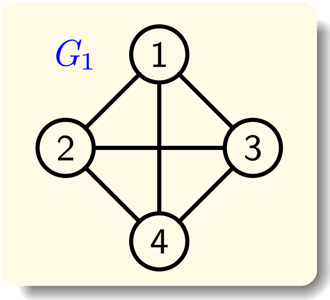
- The adjacency matrix for a undirected graph is symmetric.
  - This is due to that if the edge (i,j) is in $E(G)$ then the edge (j,i) is also in $E(G)$.
- The adjacency matrix for an directed graph may not be symmetric.
- The space needed to represent for a adjacency matrix is n^2 bits.
  - The undirected graph needs only half of this space.
- For an undirected graph the degree of any vertex i is its row sum:

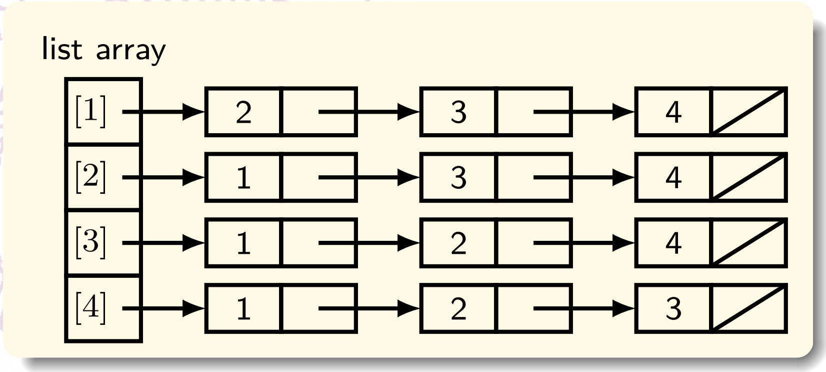
$$\sum_{j=1}^{n} A[i][j]. \tag{2.3.2}$$

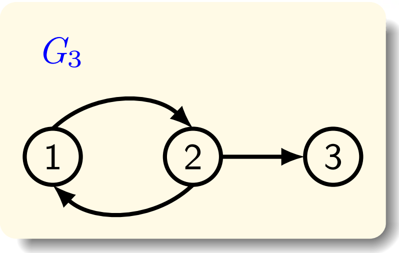
- For a directed graph the row sum if the out-degree and the column sum is the in-degree.
- The adjacency matrix approach to represent the graph is not the most efficient way in both space and execution time.
  - It does not take advantage of the sparsity of the graph.
  - For example, the time complexity to find the number of edges of a graph, with n vertices, represented by a adjacency matrix is $\mathcal{O}(n^2)$.

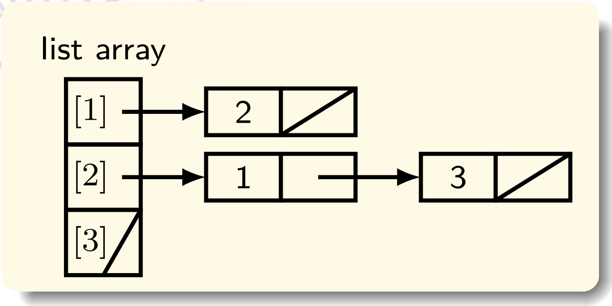
Graph Representation – Adjacency Lists

- A graph, $G = (V, E)$, of n vertices, can also be represented by n linked lists.
  - Each vertex has a linked list to represent the adjacent vertices.
- Examples

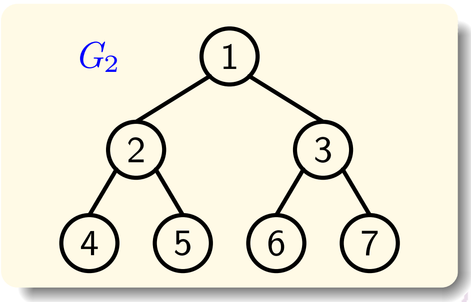


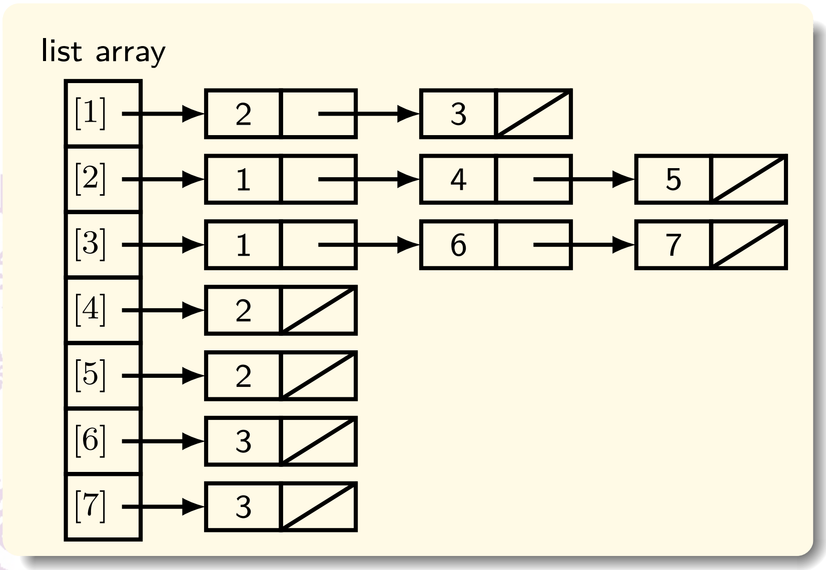






Adjacency Lists – Examples





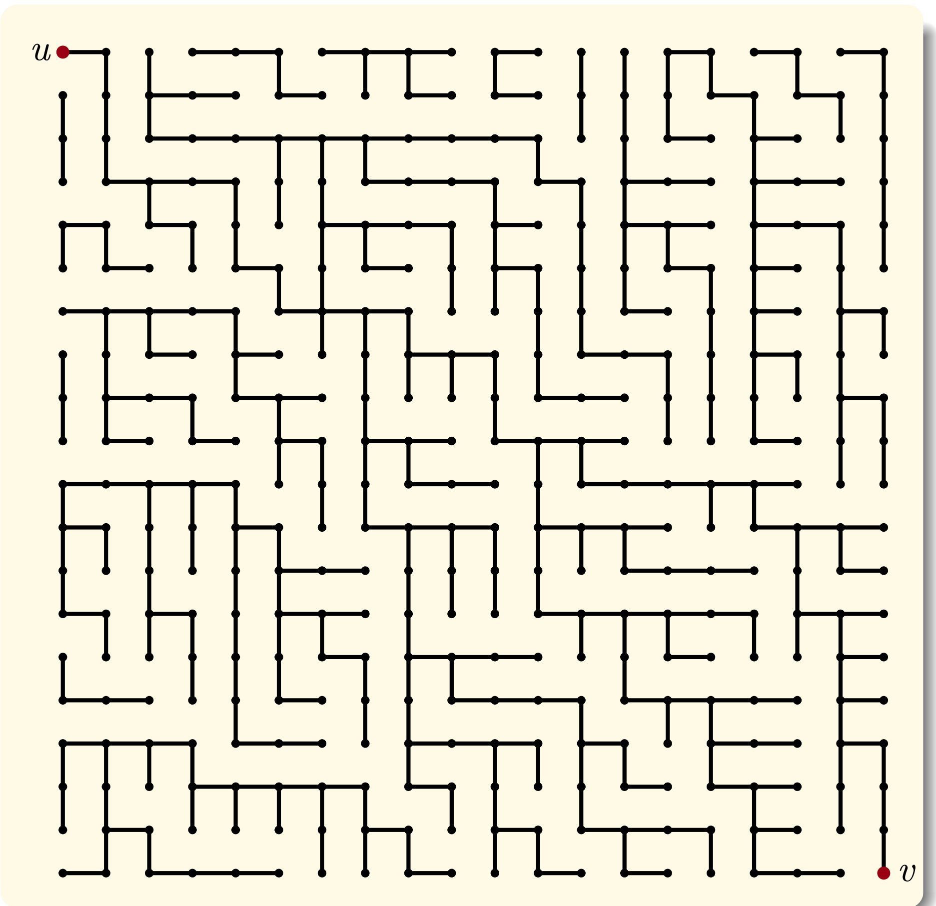
- For an undirected graph with n vertices and e edges, the adjacency list representation requires n head nodes and $2e$ list nodes.
- The degree of any vertex in an undirected graph can be determined by counting the number of nodes in the adjacency list.
  - Hence the total number of edges can be determined in $\mathcal{O}(n + e)$ time.
- For a directed graph, the out-degree of any vertex is again the number of nodes of its adjacency list.
- The in-degree may need to have another inverse adjacency list.

Weighted Edges

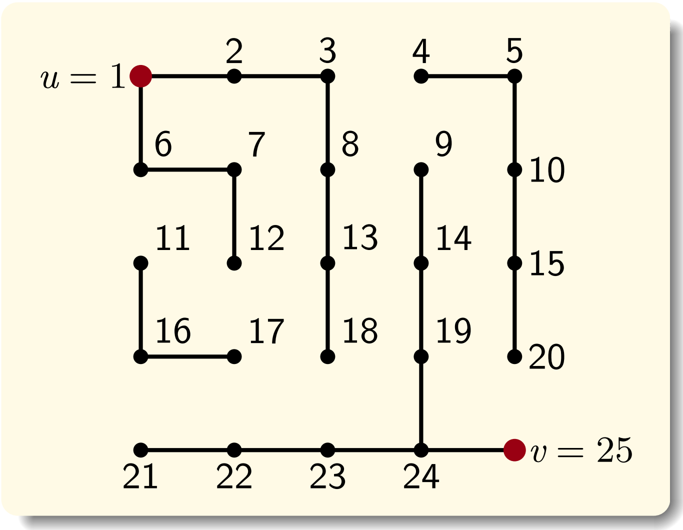
- In many applications, the edges of a graph have weights assigned to them. Thus, the adjacency matrix and adjacency lists need to accommodate these weights information.
- The adjacency matrix can store the weight of edge $\langle i, j\rangle$ to $A[i][j]$ directly.
  - No extra storage is required.
  - Space complexity is $\mathcal{O}(n^2)$.
- For the adjacency lists, each node of the list needs to have an additional field to store the weight.
  - In terms of space complexity, it is still the same as $\Theta(e)$, where e is the number of edges in $G = (V, E)$, Or, in worst-case $\mathcal{O}(n^2)$.

Network Connectivity Problem

- Given a network below, is node u connected to node v?



Network Connectivity Problem, II



- A smaller instance is shown above.
- One solution approach is to form sets of connected nodes, S_i.
- If there is a S_k such that $u, v \in S_k$, then u is connected to v.

Network Connectivity Problem, Algorithm

- A generic algorithm for network connectivity problem is shown below.

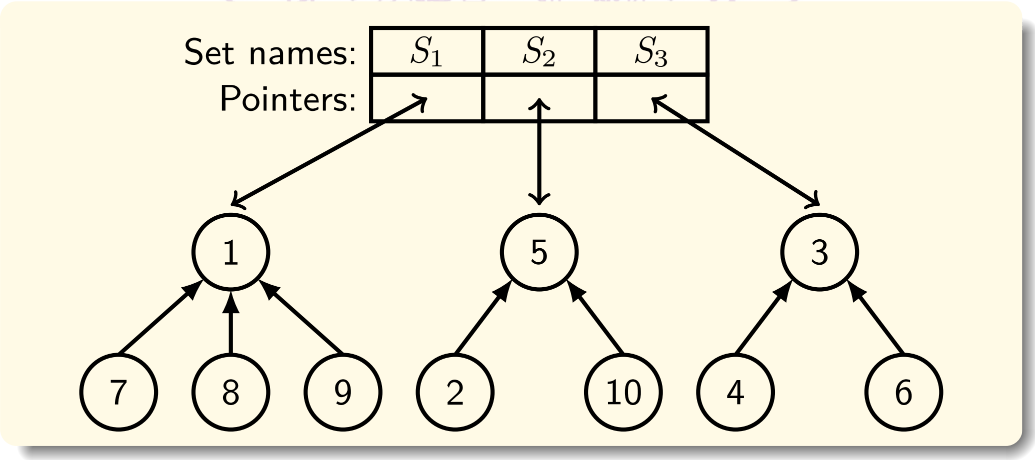
Algorithm 2.3.1. Connectivity – Generic.

```
   // Given G(V, E) and u, v ∈ V, find if u and v are connected.
   // Input: G, u, v; Output: true if connected, false otherwise.
1  Algorithm Connected(G, u, v)
2  {
3      for each v_i ∈ V do S_i := {v_i} ; // One element for each set.
4      for each e = (v_i, v_j) do { // Connected vertices
5          S_i := SetFind(v_i) ; S_j := SetFind(v_j) ;
6          S_i := S_i ∪ S_j ; // Set union.
7      }
8      if SetFind(u) = SetFind(v) then return true ;
9      return false ;
10 }
```

- The time complexity is dominated by the loop on lines 4-7
  - Iterations: $\mathcal{O}(|E|)$.
  - Two SetFind and one SetUnion per iteration.

Disjoint Sets

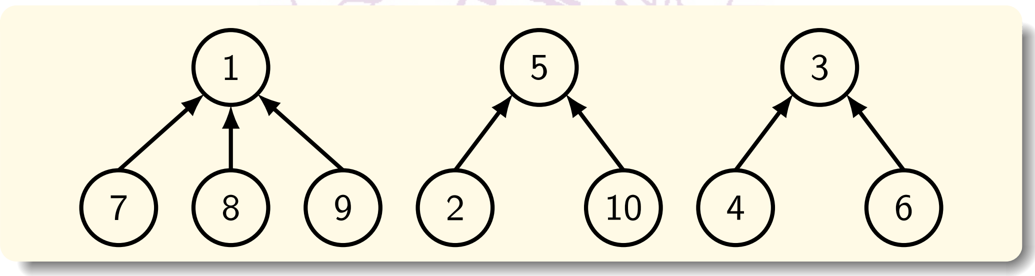
- Disjoint sets
  - Assume the elements are numbered 1, 2, ..., n.
  - Disjoint sets S_i, S_j such that $S_i \cap S_j = \emptyset$, $i \neq j$.
  - Forest can be used to represent disjoint sets
- Operations important to set manipulations
  - `Union`: Merge two disjoint sets into one.
  - `Find`(i): Given an element i find the set that contains i.
- Example: $S_1 = \{1, 7, 8, 9\}$, $S_2 = \{2, 5, 10\}$, $S_3 = \{3, 4, 6\}$.



- Note that for the forest the link is pointing from child to parent.
  - In this way, the set name can be found by following the pointers.

Disjoint Sets – Array Representation

- Simple array can also be used to represent disjoint sets.
- Example: $S_1 = \{1, 7, 8, 9\}$, $S_2 = \{2, 5, 10\}$, $S_3 = \{3, 4, 6\}$.

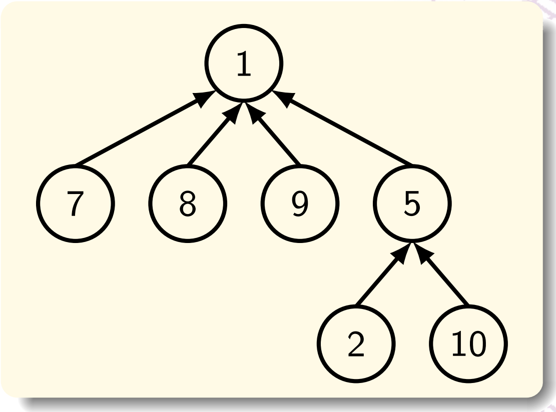


	1	2	3	4	5	6	7	8	9	10
p	-1	5	-1	3	-1	3	1	1	1	5

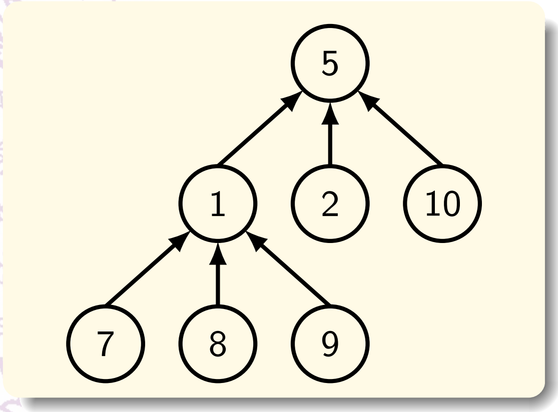
- A single array, p, can represent the disjoint sets.

Disjoints Sets – Union

- Two disjoint sets can be united easily.
- Example



$S_1 \cup S_2$



$S_1 \cup S_2$

- Both scenarios are legal and efficient.
- Union of two sets are done by setting one of the roots to be the parent of another root.

Disjoint Sets – Algorithms

- Using the array p to represent the disjoint sets, then the following algorithms perform the desired operations.

Algorithm 2.3.2. Set Union.

```
// Form union of two sets with roots, i and j.
// Input: roots, i and j ; Output: none.
Algorithm SetUnion(i, j)
{
    p[i] := j;
}
```

Algorithm 2.3.3. Set Find.

```
// Find the set that element i is in.
// Input: element i ; Output: root element of the set.
Algorithm SetFind(i)
{
    while (p[i] ≥ 0) do i := p[i] ;
    return i;
}
```

Disjoint Sets – Weighting Rule

- Algorithm SetFind(i) has the complexity $\mathcal{O}(h)$, where h is the height of the tree the element i is in.
- Algorithm SetUnion(i, j) has the time complexity $\mathcal{O}(1)$.
  - However, each union operation increases the height of the tree by 1.
  - Thus, after some union operations the tree might become skewed and the execution time of SetFind increases.
  - This issue can be alleviated by using the weighting rule.

Definition 2.3.4. Weighting rule for set union.

If the number of nodes in the tree with root i is less than the number of nodes in the tree with root j, then make j the parent of i; otherwise make i the parent of j.

- In order to implement the weighting rule, we need to know the number of elements in each set. This can be done using the root location in the p array. Set it to $-count(i)$, $count(i)$ is the number of elements in set i.
- Example: the disjoint sets can be represented as

	1	2	3	4	5	6	7	8	9	10
p	-4	5	-3	3	-3	3	1	1	1	5

Disjoint Sets – Weighted Set Union

Algorithm 2.3.5. Weighted Set Union.

```
// Form union of two sets with roots, i and j, using the weighting rule.
// Input: roots of two sets i, j ; Output: none.
Algorithm WeightedUnion(i, j)
{
      temp := p[i] + p[j] ; // Note that temp < 0.
      if (p[i] > p[j]) then { // i has fewer elements.
            p[i] := j; p[j] := temp;
      }
      else { // j has fewer elements.
            p[j] := i; p[i] := temp;
      }
}
```

- Using this algorithm, the depth of the union tree can be controlled.

Weighted Set Union – Complexity

Lemma 2.3.6.

Assume that we start with a forest of trees, each having one element. Let T be a tree with m nodes created as a result of a sequence of unions each performed using `WeightedUnion` algorithm. The height of T is no greater than $\lfloor \lg m \rfloor + 1$.

Proof. The first step is true when two sets of one element are united. Assume the Lemma is true for the first $m-1$ operations, consider the last step of the union operations, `WeightedUnion`(k, j). If set j has a elements, then set k has $m-a$ elements. And, $1 \le a \le m/2$. The height of T must be the same as that of k or one more than that of j. In the former case, the height of T is $\le \lfloor \lg(m-a) \rfloor + 1 \le \lfloor \lg m \rfloor + 1$. In the latter case, the height of T is $\le \lfloor \lg a \rfloor + 2 \le \lfloor \lg m/2 \rfloor + 2 \le \lfloor \lg m \rfloor + 1$. □

- Thus, the union set created using Algorithm `WeightedUnion` has no more than $\lfloor \lg m \rfloor + 1$ levels.
- And the time complexity of `Find` algorithm on the resulting set is $\mathcal{O}(\lg m)$.

Disjoint Sets – Collapsing Find

- The height of a set may still be improved using the collapsing rule.

Definition 2.3.7. Collapsing Rule.

If j is an element on the path from i to its root and $p[i] \ne root(i)$, then set $p[j]$ to $root(i)$.

- The `CollapsingFind` algorithm below utilizes this rule.

Algorithm 2.3.8. Collapsing Find.

```
// Find the root of i, and collapsing the elements on the path.
// Input: an element i; Output: root of the set containing i.
Algorithm CollapsingFind(i)
{
    r := i; // Initialized r to i.
    while (p[r] > 0) do r := p[r]; // Find the root.
    while (i ≠ r) do { // Collapse the elements on the path.
        s := p[i]; p[i] := r; i := s;
    }
    return r;
}
```

Ackermann's Function

Definition 2.3.9. Ackermann's function.

The Ackermann's function is defined as

$$\begin{aligned} A(1,j) &= 2^j && \text{for } j \geq 1, \\ A(i,1) &= A(i-1,2) && \text{for } i \geq 2, \\ A(i,j) &= A(i-1, A(i,j-1)) && \text{for } i,j \geq 2. \end{aligned} \tag{2.3.3}$$

Also define

$$\alpha(p,q) = \min\{z \geq 1 | A(z, \lfloor \frac{p}{q} \rfloor) > \lg q\}, p \geq q \geq 1. \tag{2.3.4}$$

$$\begin{array}{llll} A(1,1) = 2 & A(1,2) = 4 & A(1,3) = 8 & A(1,4) = 16 \\ A(2,1) = 4 & A(2,2) = 16 & A(2,3) = 2^{16} & A(2,4) = 2^{65536} \\ A(3,1) = 16 & A(3,2) \gg 2^{65536} & & \\ A(4,1) \gg 2^{65536} & & & \end{array}$$

- A is very fast growing function and α is a very slow growing function.
- Note that $A(3,1) = 16$, $\alpha(p,q) \leq 3$ for $q < 2^{16} = 65,536$ and $p > q$.
- Since $A(4,1)$ is a very large number, $\alpha(p,q) \leq 4$ for all practical purposes.

Tarjan and Van Leeuwen Bound

Lemma 2.3.10. Tarjan and Van Leeuwen bounds.

Assume that we start with a forest of trees, each having one node. Let $T(f,u)$ be the maximum time required to process any intermixed sequence of f finds and u unions. Assume that $u \geq n/2$, then

$$k_1\Big(n + f \cdot \alpha(f+n,n)\Big) \leq T(f,u) \leq k_2\Big(n + f \cdot \alpha(f+n,n)\Big) \tag{2.3.5}$$

for some positive constants k_1 and k_2.

- Proof please see textbook [Cormen], pp. 575-581.

- Thus, manipulating disjoint sets are rather efficient.
- Though algorithms (2.3.2), (2.3.3), (2.3.5), and (2.3.8) assume the disjoint sets are represented using a simple array, they can be implemented if the disjoint sets are represented using linked lists as well.
- The complexities are the same with either data structure.

Summary

- Graphs
  - Definitions.
  - Adjacency matrix.
  - Adjacency lists.
- Network connectivity problem
- Disjoint sets.
  - Set union.
  - Set find.
  - Weighted set union.
  - Collapsing set find.