Decrease and Conquer

Algorithm

2014 Fall Semester

SUNG KYUN KWAN UNIVERSITY

BASIC CONCEPTS

Decrease and Conquer

1. Reduce a problem instance to a smaller instance of the same problem and extend solution

2. Solve the smaller instance

3. Extend solution of smaller instance to obtain solution to original problem

- Also referred to as *inductive*, *incremental* approach or *chip and conquer*

Examples of Decrease & Conquer

- *Decrease by one:*
  - Insertion sort
  - Graph search algorithms:
    - DFS
    - BFS
    - Topological sorting
  - Algorithms for generating permutations, subsets

- *Decrease by a constant factor*
  - Binary search
  - Fake-coin problems
  - multiplication à la russe
  - Josephus problem

- *Variable-size decrease*
  - Euclid's algorithm
  - Selection by partition

What's the difference?

Consider the problem of exponentiation:

Compute a^n

- Brute Force:
- Divide and conquer:
- Decrease by one:
- Decrease by constant factor:

SUNG KYUN KWAN UNIVERSITY

What's the difference?

Consider the problem of exponentiation:

Compute a^n

- Brute Force: $a^n = a*a*a*a*...*a$
- Divide and conquer: $a^n = a^{n/2} * a^{n/2}$
- Decrease by one: $a^n = a^{n-1} * a$
- Decrease by constant factor: $a^n = (a^{n/2})^2$

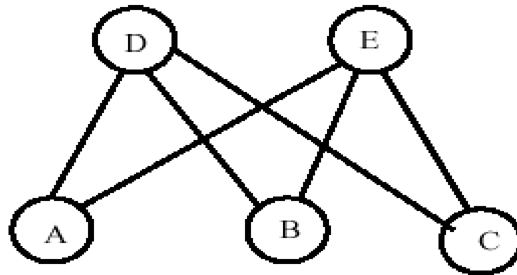
GRAPH

Graph Traversal

- Many problems require processing all graph vertices in systematic fashion
- *Graph traversal algorithms:*
  - Depth-first search
  - Breadth-first search
- First, some definitions!

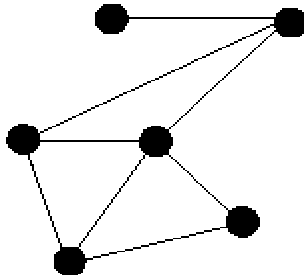
Graphs

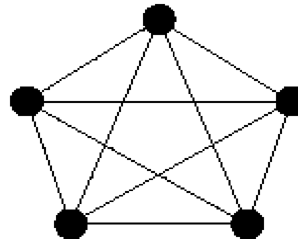
- A (simple) graph $G = (V, E)$ consists of
  - V, a nonempty set of *vertices*
  - E, a set of *unordered* pairs of distinct vertices called *edges*.
- Examples:

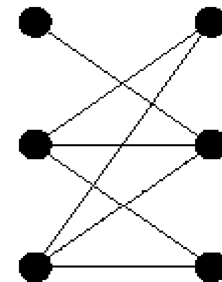


V={A,B,C,D,E}

E={ (A,D),(A,E),(B,D),
(B,E),(C,D),(C,E)}

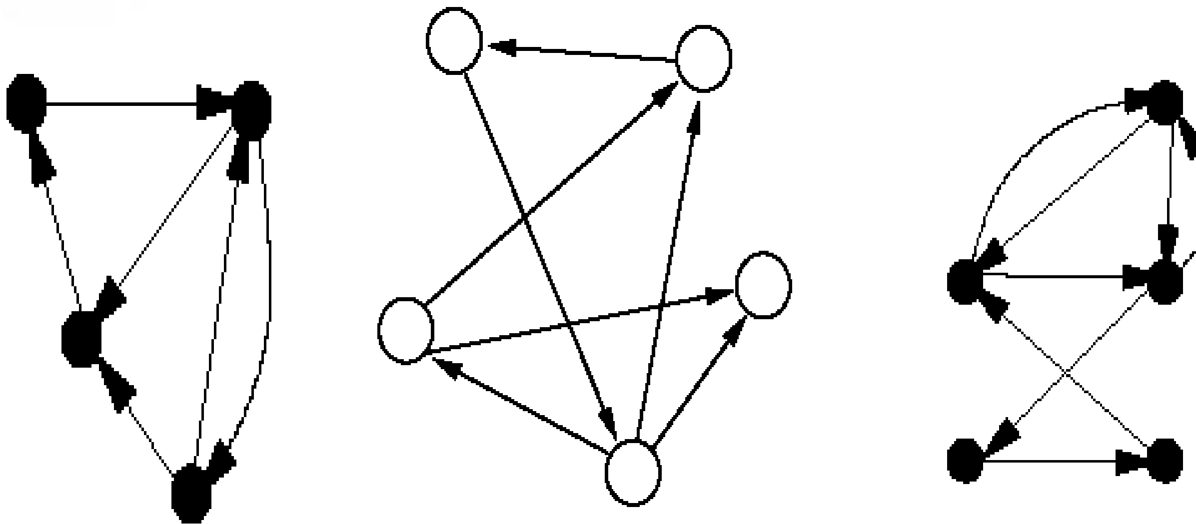






Directed Graphs

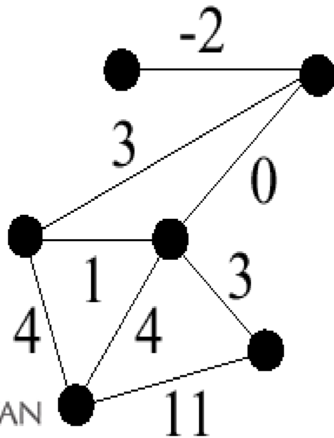
- A directed graph (digraph) $G = (V, E)$ consists of
  - V, a nonempty set of *vertices*
  - E, a set of *ordered* pairs of distinct vertices called *edges*.
- Examples:

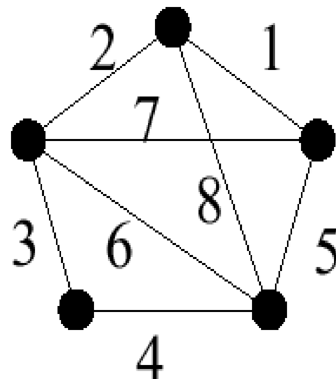


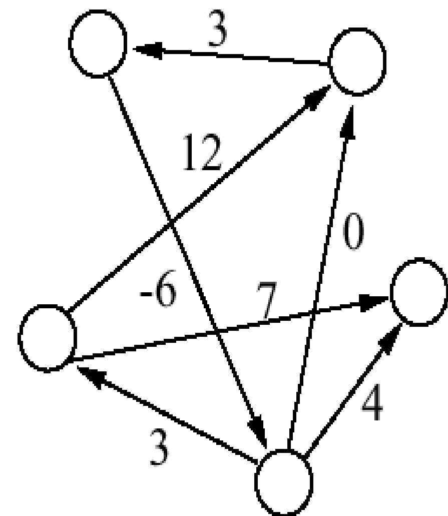
SUNG KYUN KWAN UNIVERSITY

Weighted Graph

- A weighted graph is a triple $G = (V, E, W)$
  - where (V, E) is a graph (or a digraph) and
  - W is a function from E into R, the reals (integer or rationals).
  - For an edge e, $W(e)$ is called the weight of e.
- A weighted digraph is often called a network.
- Examples





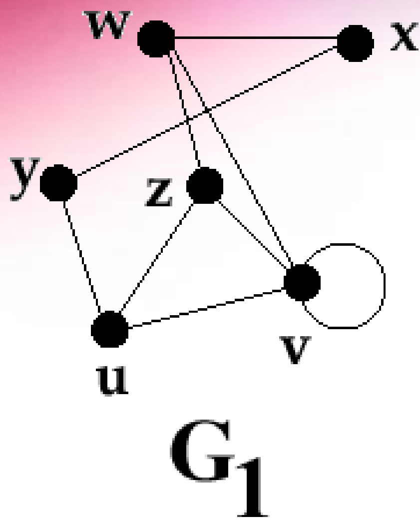


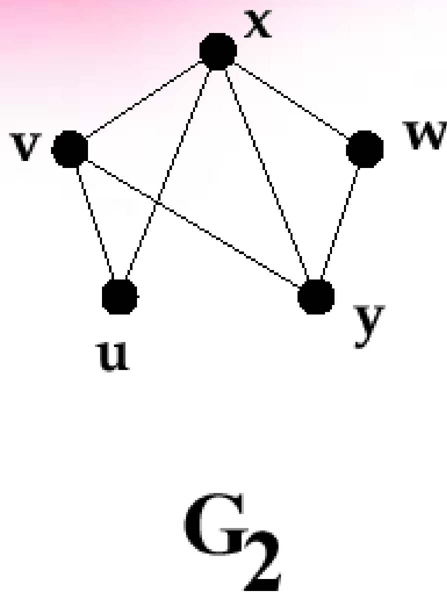
SUNG KYUN KWAN UNIVERSITY

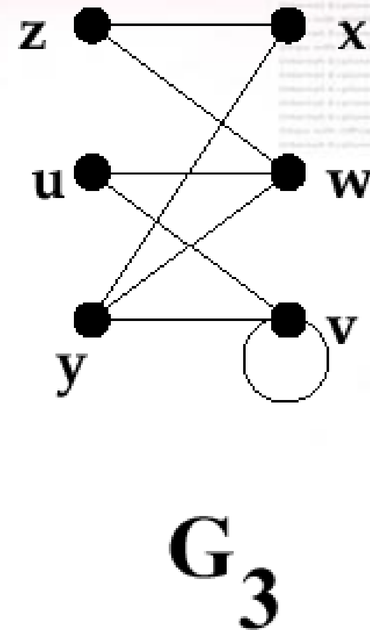
Graph Terminology

- Let u and v be vertices, and let $e = (u,v)$ be an edge in an undirected graph G.
  - The vertices u and v are *adjacent*.
  - The edge e is *incident* with both vertices u and v.
  - The edge e *connects* u and v.
  - The vertices u and v are the *endpoints* of edge e.
  - The *degree* of a vertex, denoted $deg(v)$, in an undirected graph is the number of edges incident with it (where self-loops are counted twice).

Examples

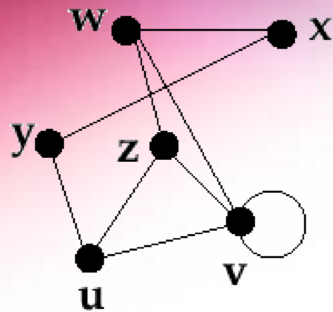


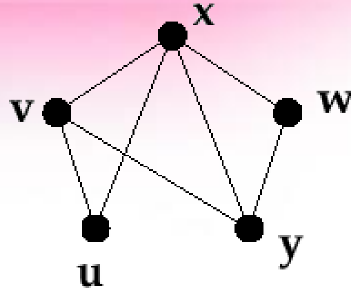


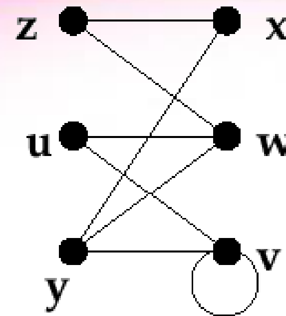


SUNG KYUN KWAN UNIVERSITY

Vertex Degree Examples



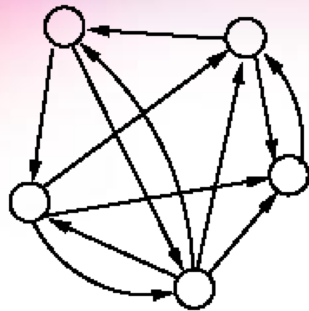


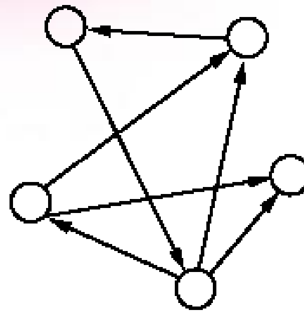


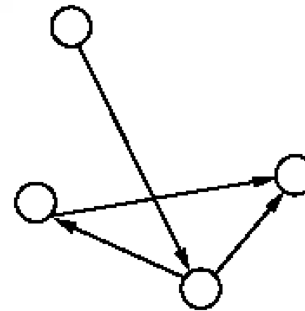
G_1	G_2	G_3
deg(u)=3	deg(u)=2	deg(u)=2
deg(v)=5	deg(v)=3	deg(v)=4
deg(w)=3	deg(w)=2	deg(w)=3
deg(x)=2	deg(x)=4	deg(x)=2
deg(y)=2	deg(y)=3	deg(y)=3
deg(z)=3		deg(z)=2

More Graph Terminology

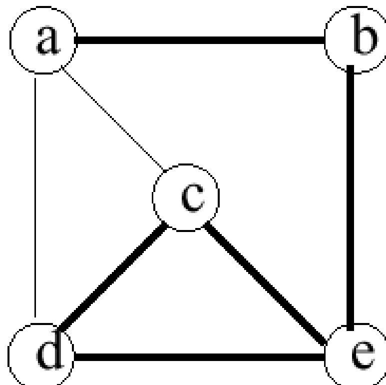
- A subgraph of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$.

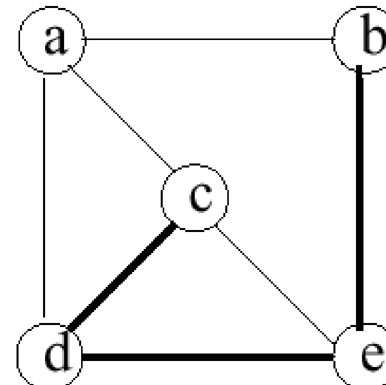






- A path is a sequence of vertices $v_1, v_2, v_3, \cdots, v_k$ such that consecutive vertices v_i and v_{i+1} are adjacent.



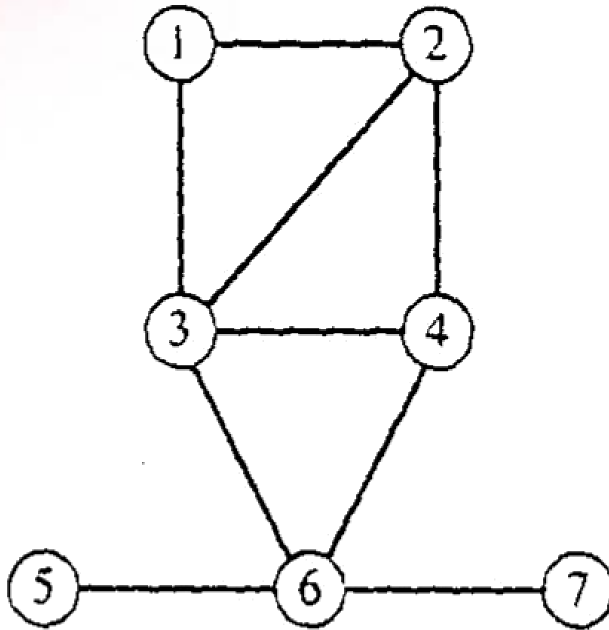


(a simple path)

SUNG KYUN KWAN UNIVERSITY

Graph Representations using Data Structures

- Adjacency Matrix Representation
  - Let G = (V, E), n = |V|, m = |E|, V = {$v1$, $v2$, …, vn)
  - G can be represented by an $n \times n$ matrix C

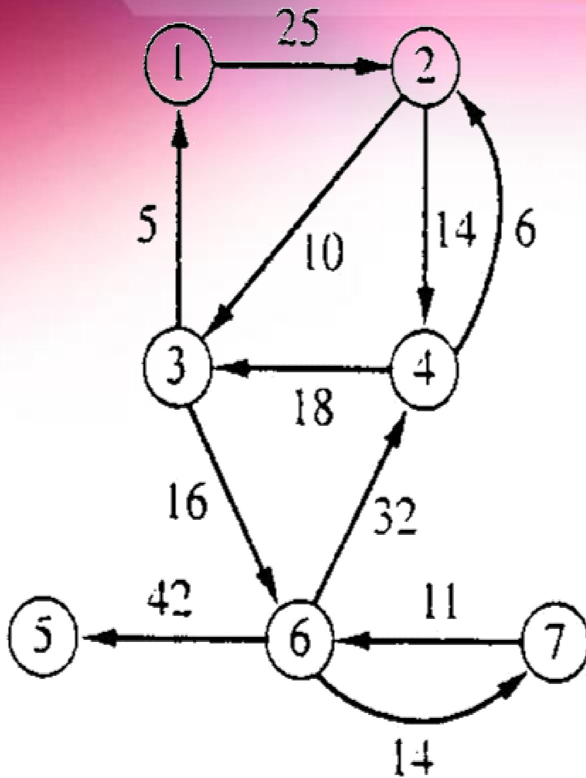


(a) An undirected graph

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

(b) Its adjacency matrix

Adjacency Matrix for weight digraph



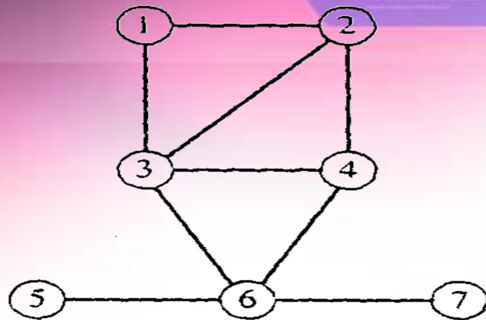
(a) A weighted digraph

$$
\begin{pmatrix}
0 & 25.0 & \infty & \infty & \infty & \infty & \infty \\
\infty & 0 & 10.0 & 14.0 & \infty & \infty & \infty \\
5.0 & \infty & 0 & \infty & \infty & 16.0 & \infty \\
\infty & 6.0 & 18.0 & 0 & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & 0 & \infty & \infty \\
\infty & \infty & \infty & 32.0 & 42.0 & 0 & 14.0 \\
\infty & \infty & \infty & \infty & \infty & 11.0 & 0
\end{pmatrix}
$$

(b) Its adjacency matrix

SUNG KYUN KWAN UNIVERSITY

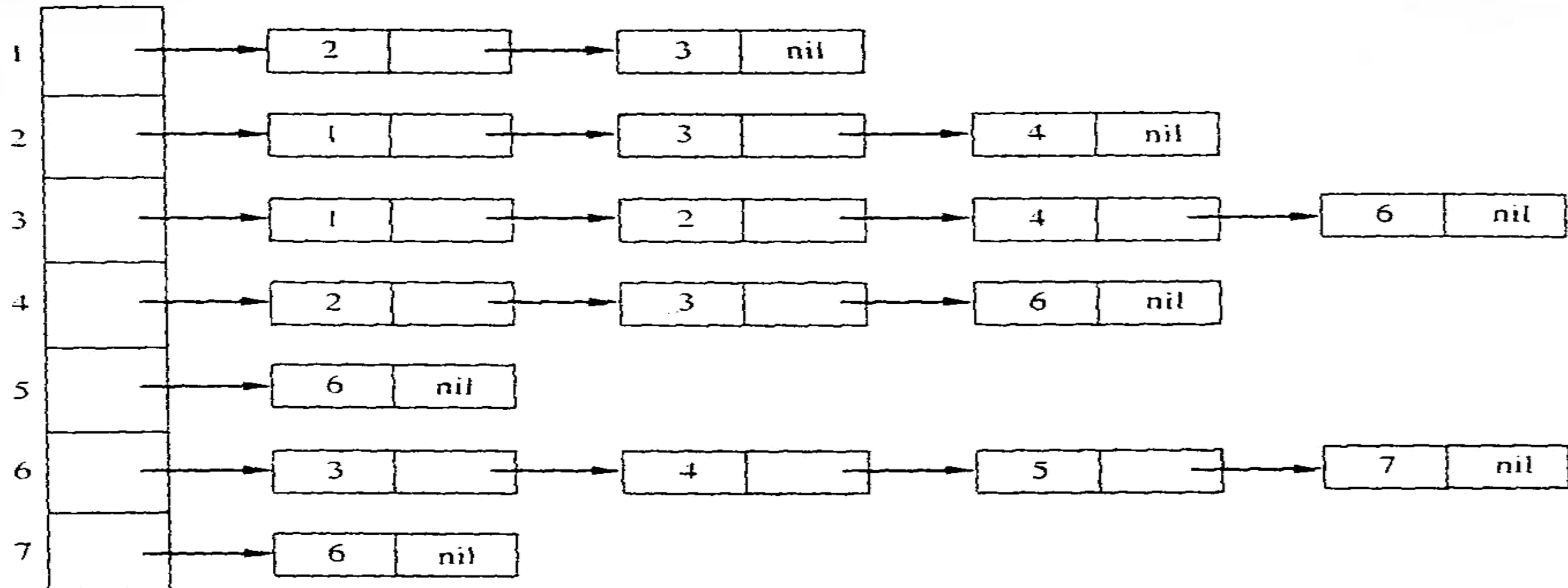
Adjacency List Representation



(a) An undirected graph

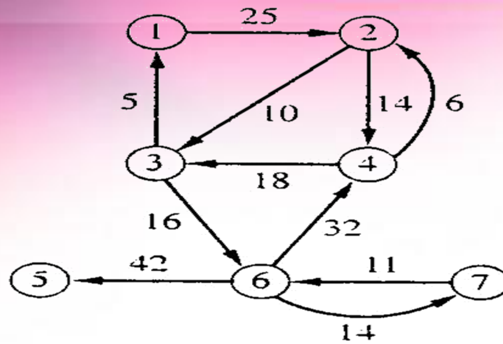
$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(b) Its adjacency matrix



SUNG KYUN KWAN UNIVERSITY

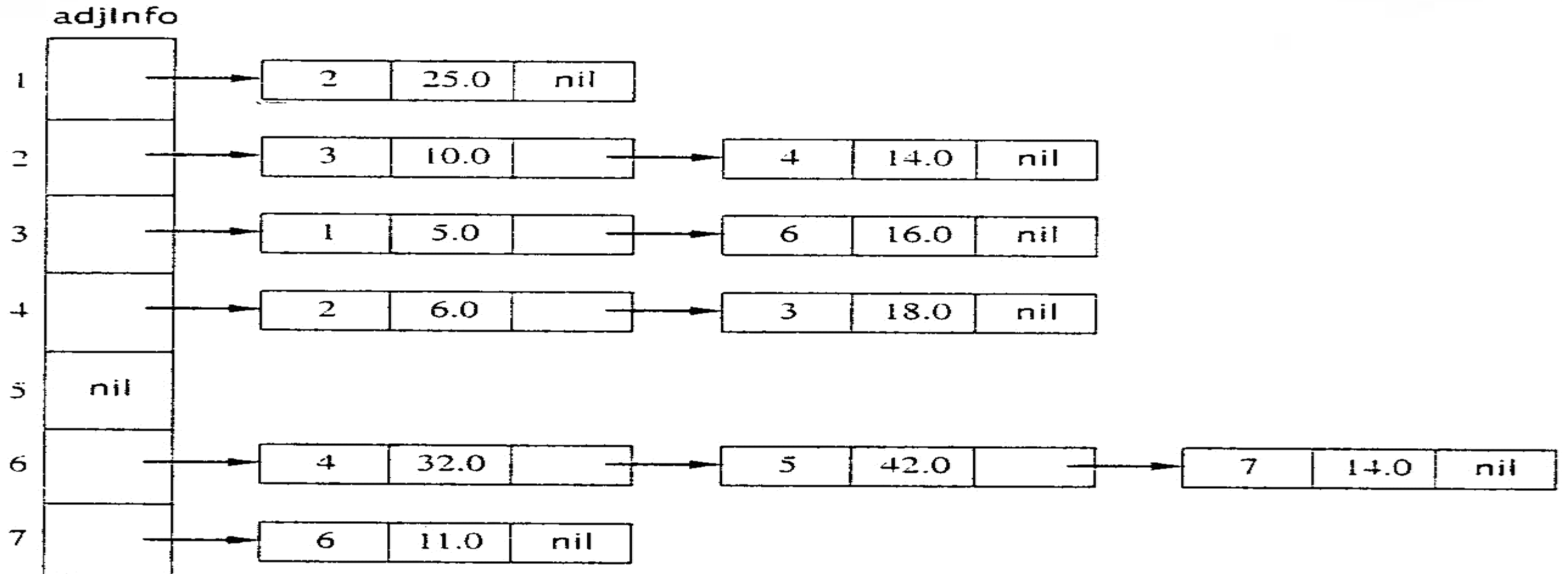
Weighted Digraph Adjacency List Representation



(a) A weighted digraph

$$\begin{pmatrix} 0 & 25.0 & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 10.0 & 14.0 & \infty & \infty & \infty \\ 5.0 & \infty & 0 & \infty & \infty & 16.0 & \infty \\ \infty & 6.0 & 18.0 & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 32.0 & 42.0 & 0 & 14.0 \\ \infty & \infty & \infty & \infty & \infty & 11.0 & 0 \end{pmatrix}$$

(b) Its adjacency matrix



SUNG KYUN KWAN UNIVERSITY

More Definitions

- Subgraph
- Symmetric digraph
- Complete graph
- Adjacency relation
- Path, simple path, reachable
- Connected, Strongly Connected
- Cycle, simple cycle
- Acyclic
- Undirected forest
- Free tree, undirected tree
- Rooted tree
- Connected component

Traversing Graphs

- Most algorithms for solving problems on a graph examine or process each vertex and each edge.
- Depth-First Search (DFS) and Breadth-First Search (BFS)
  - Two elementary traversal strategies that provide an efficient way to "visit" each vertex and edge exactly once.
  - Both work on directed or undirected graphs.
  - Many advanced graph algorithms are based on the concepts of DFS or BFS.
  - The difference between the two algorithms is in the order in which each "visits" vertices.

DECREASE BY ONE

Depth-first search

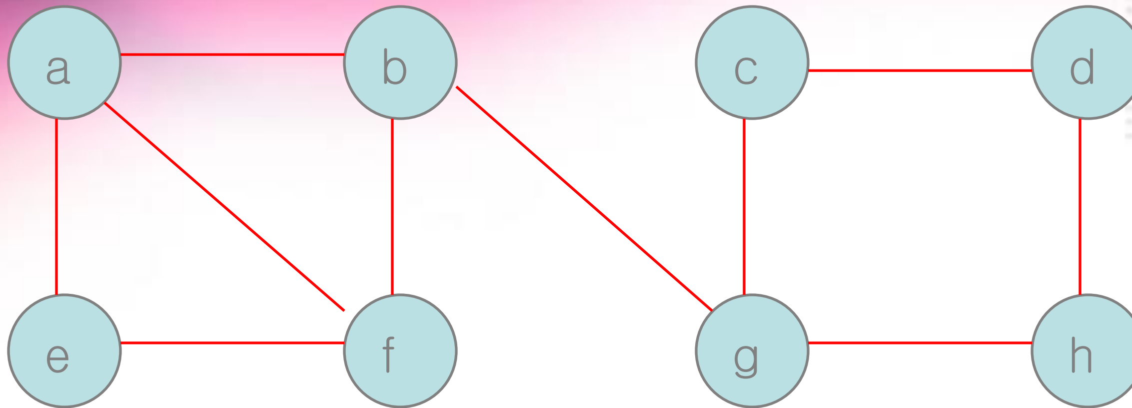
- Explore graph always moving away from last visited vertex
- Pseudocode for Depth-first-search of graph G=(V,E)

```
DFS(G)
count :=0
mark each vertex with 0 (unvisited)
for each vertex v∈ V do
    if v is marked with 0
        dfs(v)
```

```
dfs(v)
count := count + 1
mark v with count
for each vertex w adjacent to v do
    if w is marked with 0
        dfs(w)
```

SUNG KYUN KWAN UNIVERSITY

Example – undirected graph



- Depth-first traversal:

<u>dfs(*v*)</u>

count := count + 1

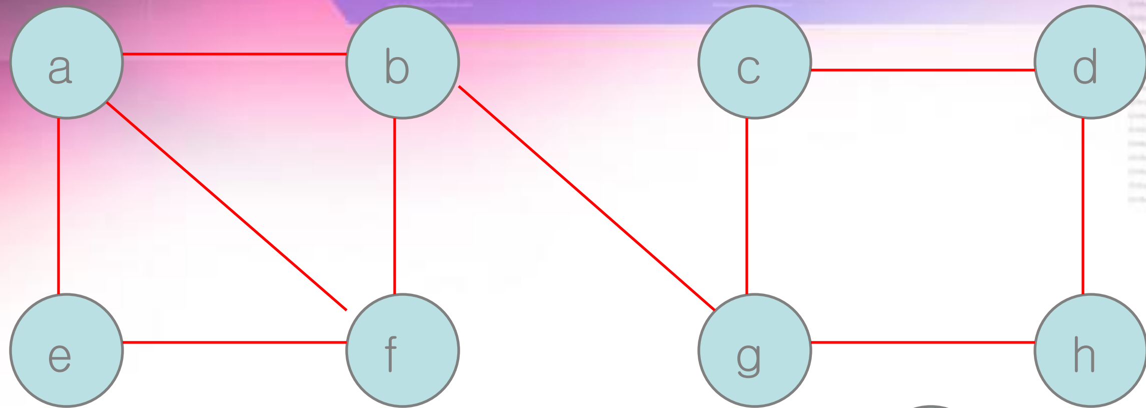
mark *v* with count

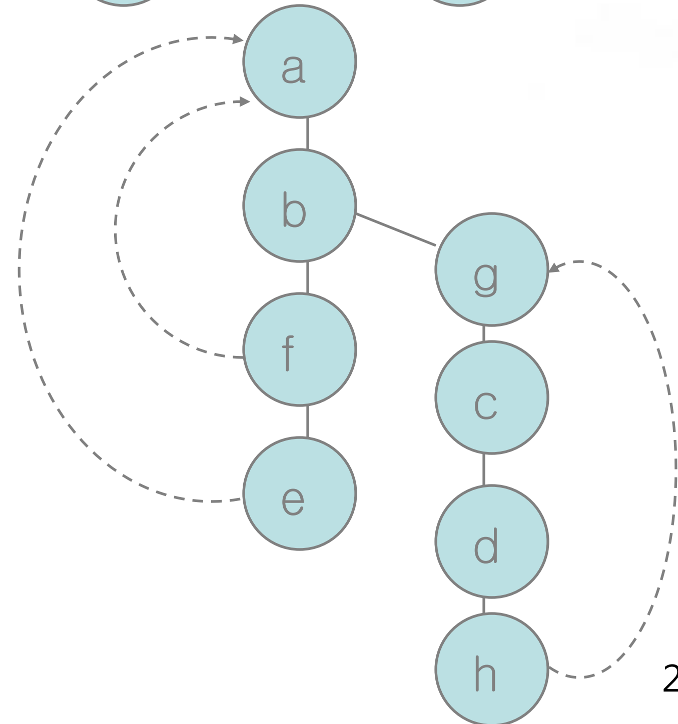
for each vertex *w* adjacent
    to *v* do

    if *w* is marked with 0

        dfs(*w*)

DFS stack and forest





Types of edges

- *Tree edges:* edges comprising forest
- *Back edges:* edges to ancestor nodes

Question

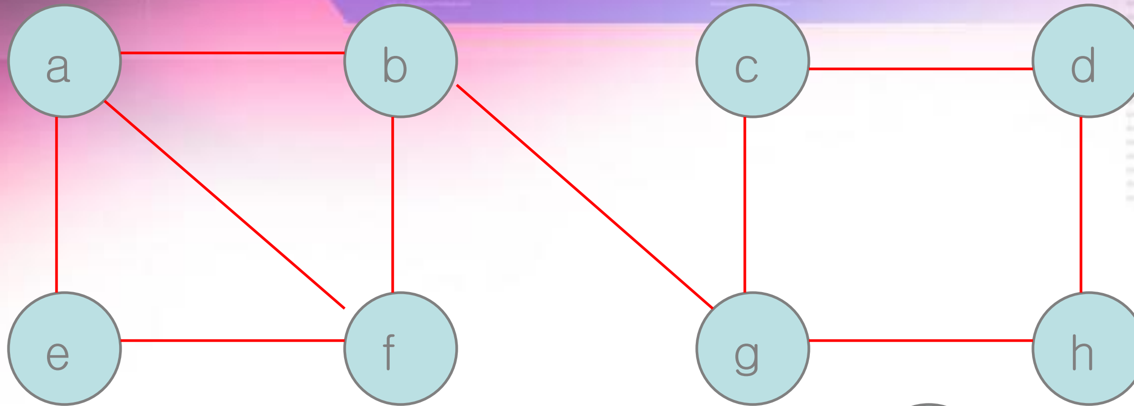
- How to rewrite the procedure dfs(v), using a stack to eliminate recursion

```
dfs(v)
count := count + 1
mark v with count
for each vertex w adjacent to v do
    if w is marked with 0
        dfs(w)
```

Non-recursive version of DFS algorithm

```
Algorithm dfs(v)
s.createStack();
s.push(v);
count := count + 1
mark v with count
while (!s.isEmpty()) {
        let x be the node on the top of the stack s;
        if (no unvisited nodes are adjacent to x)
                s.pop(); // backtrack
        else {
                select an unvisited node u adjacent to x;
                s.push(u);
                count := count + 1
                mark u with count
        }
}
```

DFS stack and forest

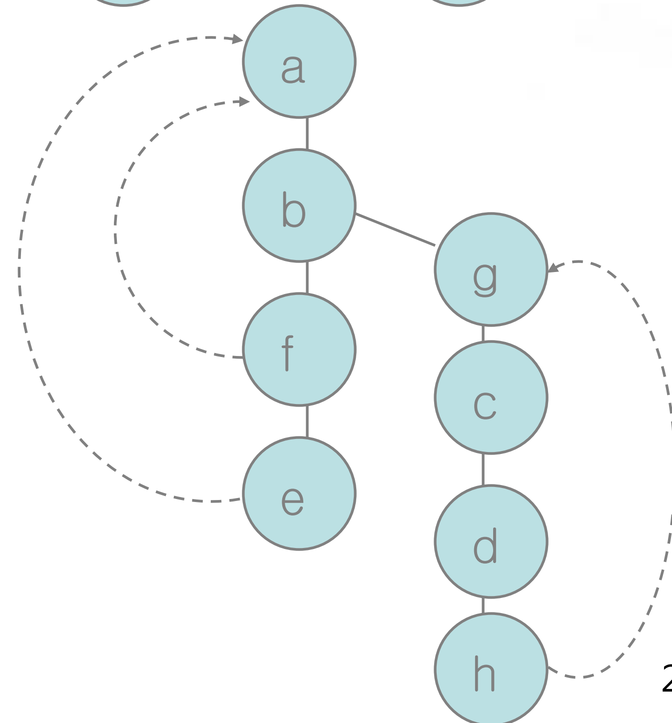


$a_{1,8}$ $b_{2,7}$ $f_{3,2}$ $e_{4,1}$

$g_{5,6}$ $c_{6,5}$ $d_{7,4}$ $h_{8,3}$

preorder: a b f e g c d h

postorder: e f h d c g b a



SUNG KYUN KWAN UNIVERSITY

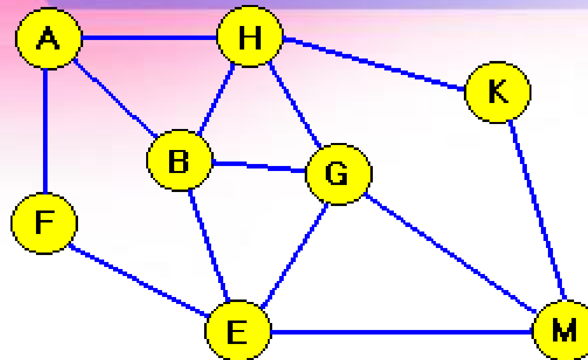
Depth-first search: Notes

- Yields two distinct ordering of vertices:
  - preorder: as vertices are first encountered (pushed onto stack)
  - postorder: as vertices become dead-ends (popped off stack)

- Applications:
  - checking connectivity, finding connected components
  - checking acyclicity
  - searching state-space of problems for solution (AI)

In-class Exercise

- Consider the graph

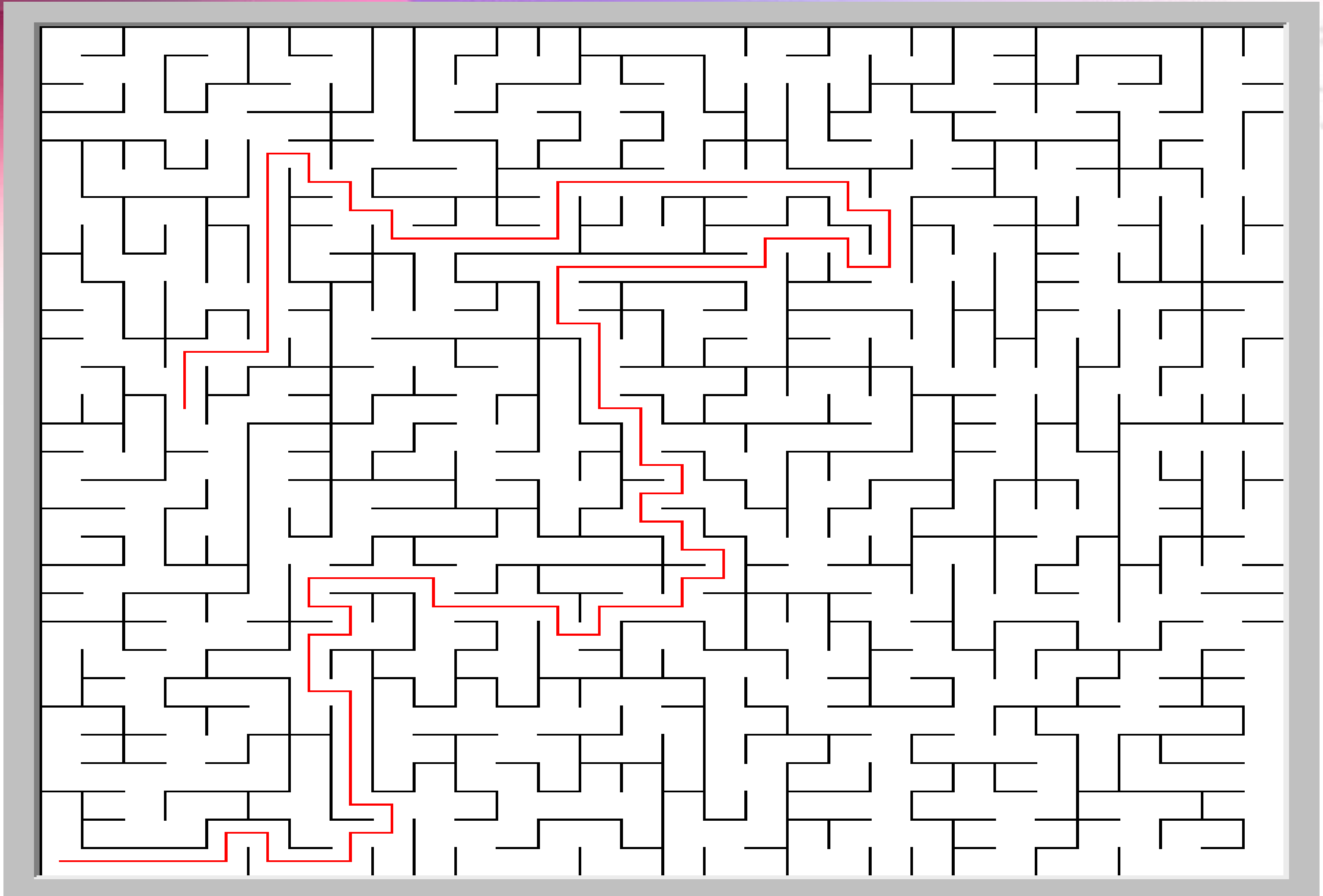


a. Write down the adjacency matrix and adjacency lists specifying this graph. (Assume that the matrix rows and columns and vertices in the adjacency lists follow in the alphabetical order of the vertex labels.)

b. Starting at vertex A and resolving ties by the vertex alphabetical order, traverse the graph by depth-first search and construct the corresponding depth-first search tree. Give the order in which the vertices were reached for the first time (pushed onto the traversal stack) and the order in which the vertices became dead ends (popped off the stack).

SUNG KYUN KWAN UNIVERSITY

Maze traversal



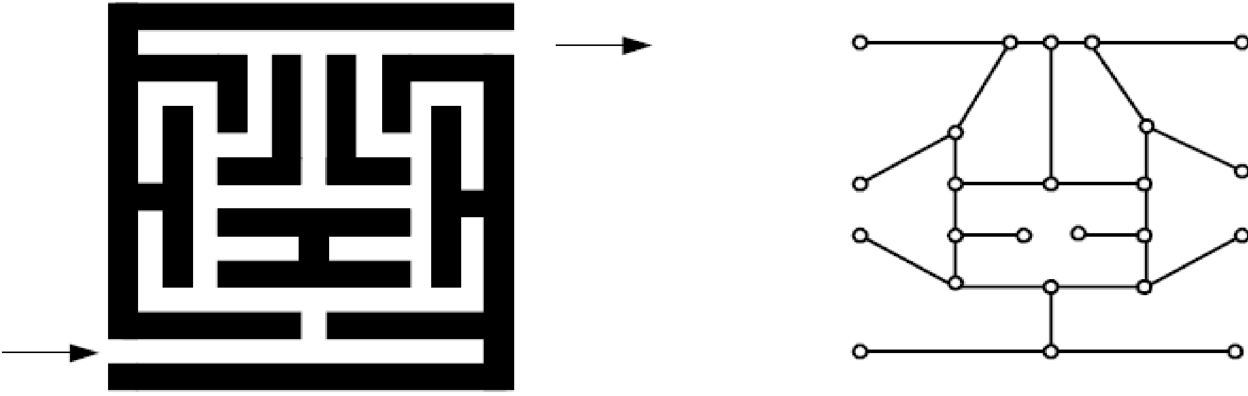
SUNG KYUN KWAN UNIVERSITY

Maze and graph representing it (I)

One can model a maze by having a vertex for a starting point, a finishing point, dead ends, and all the points in the maze where more than one path can be taken and then connecting the vertices according to the paths.

a. Construct such a graph for the following maze.

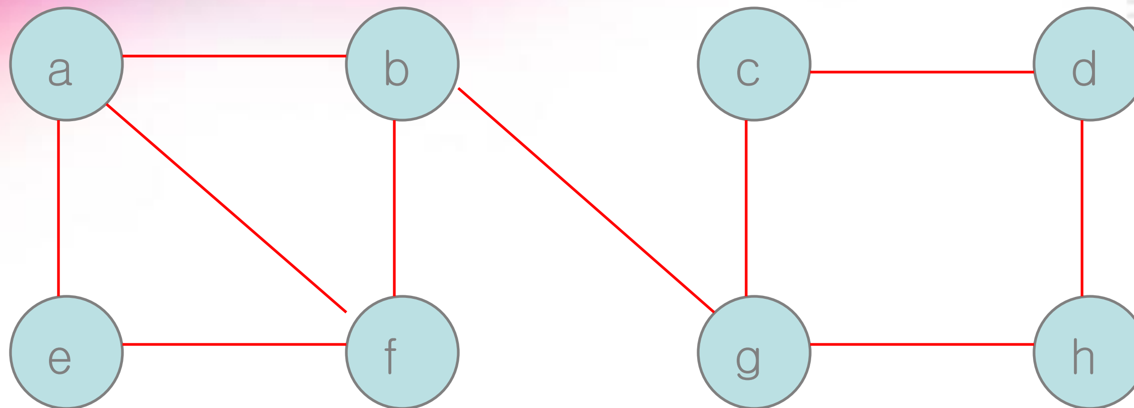
Maze and graph representing it (II)



Breadth-first search

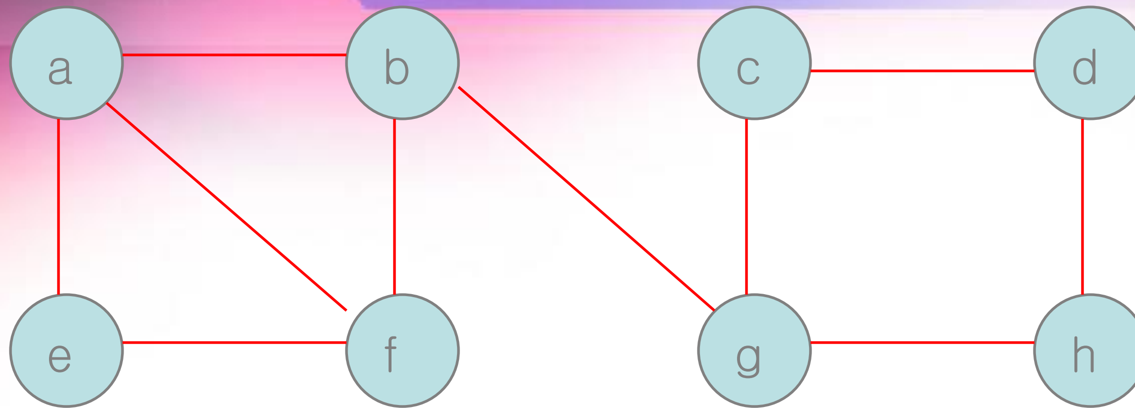
- Explore graph moving across to all the neighbors of last visited vertex
- Similar to level-by-level tree traversals
- Instead of a stack, breadth-first uses queue
- Applications: same as DFS, but can also find paths from a vertex to all other vertices with the smallest number of edges

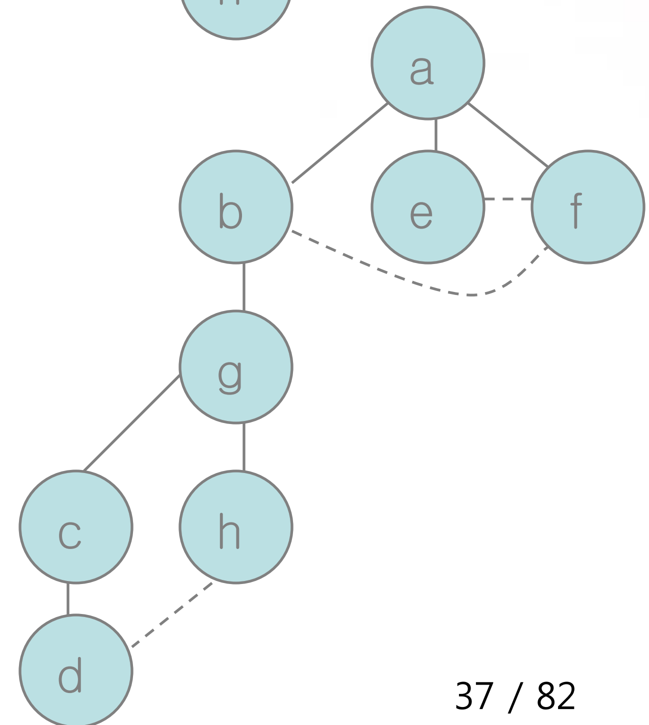
Example – undirected graph



- Breadth-first traversal:

BFS queue and forest





- a b e f g c h d

SUNG KYUN KWAN UNIVERSITY

Breadth-first search algorithm

<u>BFS(G)</u>

```
count :=0
mark each vertex with 0
    for each vertex v∈ V do
        if v is marked with 0
            bfs(v)
```

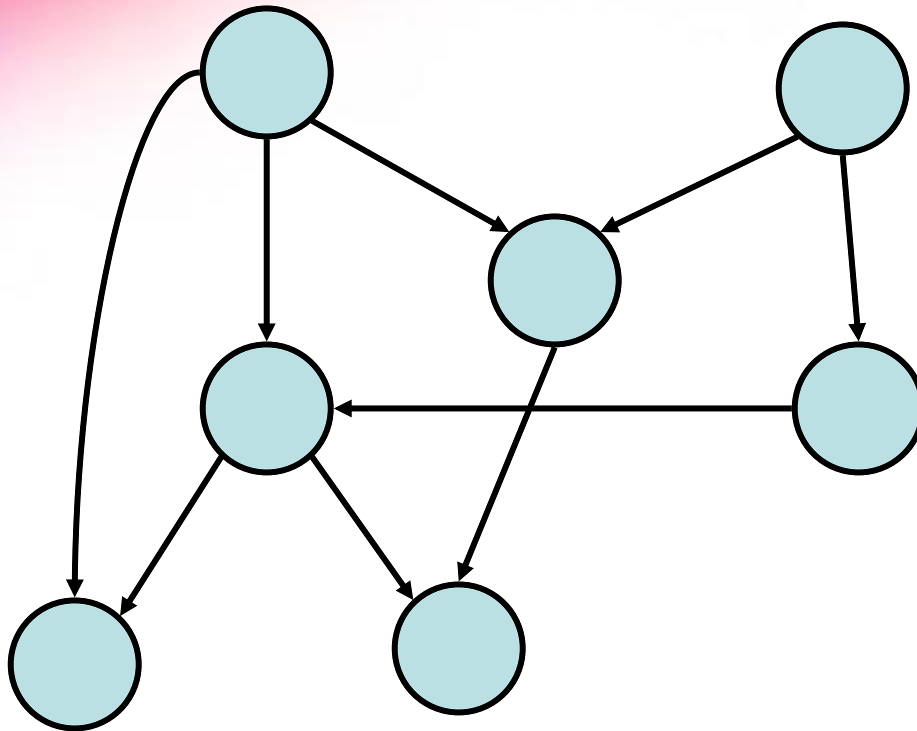
<u>bfs(*v*)</u>

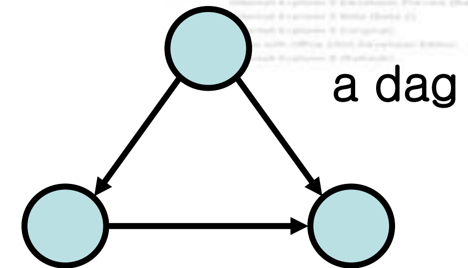
```
count := count + 1
mark v with count
initialize queue with v
while queue is not empty do
    a := front of queue
    for each vertex w adjacent to a do
        if w is marked with 0
            count := count + 1
            mark w with count
            add w to the end of the queue
    remove a from the front of the queue
```

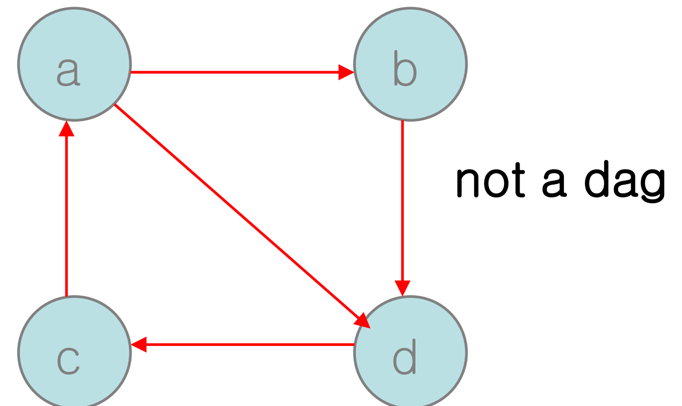
SUNG KYUN KWAN UNIVERSITY

Directed acyclic graphs

- A *directed acyclic graph* or *DAG* is a directed graph with no directed cycles:







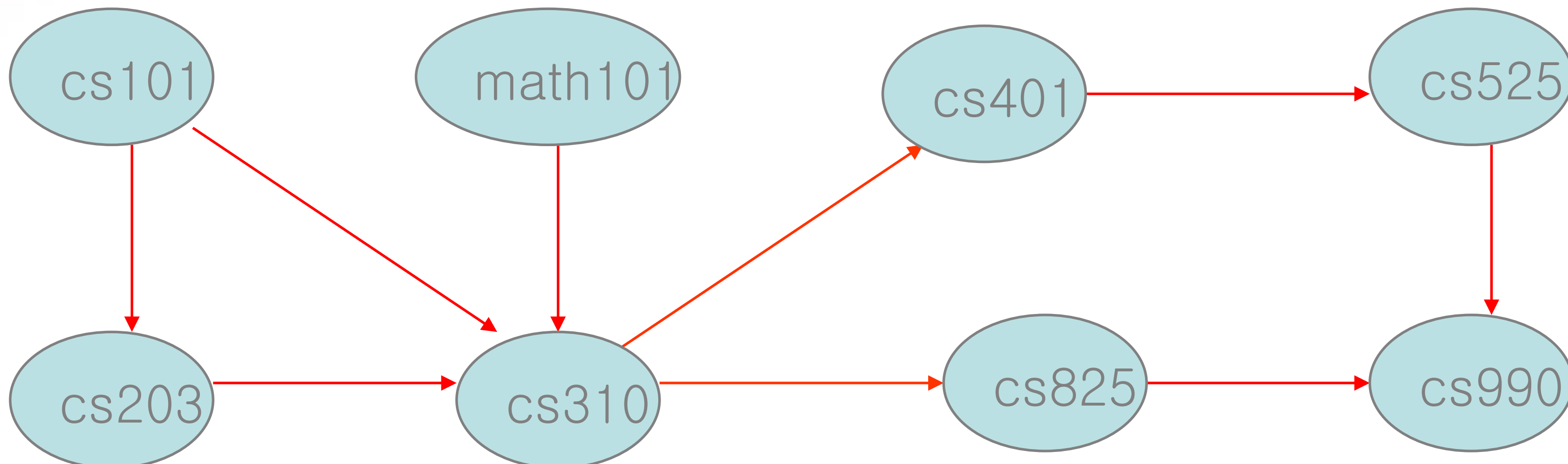
SUNG KYUN KWAN UNIVERSITY

Directed acyclic graphs (DAG)

- Arise in many modeling problems, e.g.:
  - course prerequisite structure
  - food chains
- Imply partial ordering on the domain

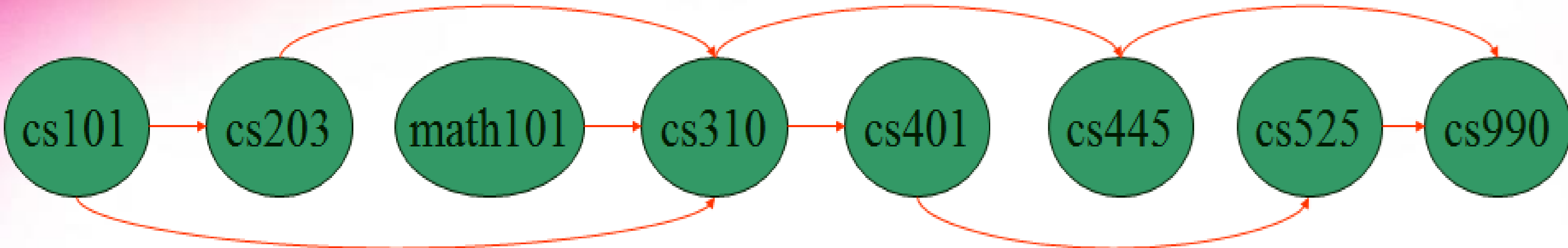
Topological Sort

- *Topological sort* of a DAG:
  - Linear ordering of all vertices in graph G such that for every edge $(u, v) \in G$, the vertex, u, where the edge starts is listed before the vertex, v, where the edge ends.



SUNG KYUN KWAN UNIVERSITY

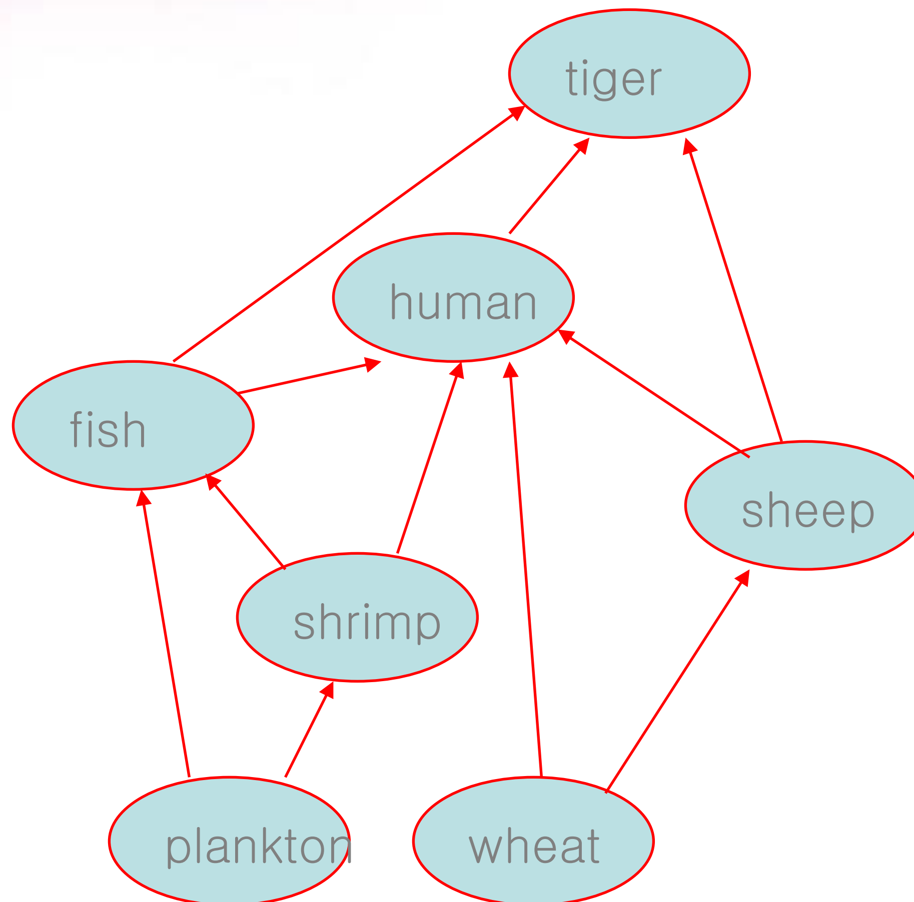
Topological Sort



SUNG KYUN KWAN UNIVERSITY

Topological Sorting Example

- Order the following items in a food chain



SUNG KYUN KWAN UNIVERSITY

Topological sorting Algorithms

1. DFS-based algorithm:
   - DFS traversal noting order vertices are popped off stack
   - Reverse order solves topological sorting
   - Back edges encountered?→ NOT a dag!

2. Source removal algorithm
   - Repeatedly identify and remove a *source* vertex, ie, a vertex that has no incoming edges

Question

- How would you find a source (or determine that such a vertex does not exist) in a digraph
  - represented by adjacency matrix?
  - represented by adjacency linked list?

DECREASE BY A CONSTANT FACTOR

Decrease by a constant factor - Examples

- Fake-coin problem
- Multiplication à la russe (Russian peasant method)
- Josephus problem

Fake-Coin Puzzle (simpler version)

There are *n* identically looking coins one of which is fake. There is a balance scale but there are no weights; the scale can tell whether two sets of coins weigh the same and, if not, which of the two sets is heavier (but not by how much). Design an efficient algorithm for detecting the fake coin. Assume that the fake coin is known to be lighter than the genuine ones.

Decrease by factor 2 algorithm

Decrease by factor 3 algorithm

SUNG KYUN KWAN UNIVERSITY

Russian peasant method

- n*m
- If n is even, n*m = (n/2)*(2m)
- If n is odd, n*m = ((n-1)/2)*(2m)+m
- Repeat the above process until we have the trivial case of 1*x = x

- Only simple operations: halving, doubling, and adding, no need to memorize the table of multiplications

Russian peasant method - Example

- Compute 50 * 65

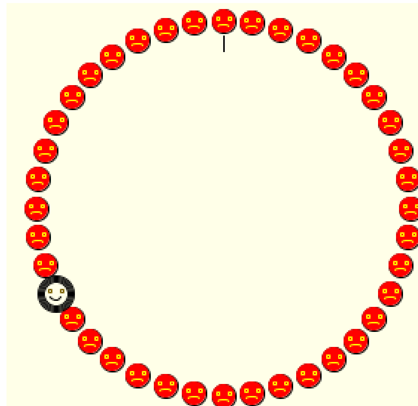
$$n \cdot m = \begin{cases} m & \text{if } n = 1 \\ (n/2) \cdot 2m & \text{if } n \text{ is even} \\ (\lfloor n/2 \rfloor \cdot 2m + m & \text{if } n \text{ is odd} \end{cases}$$

n	m	
50	65	
25	130	
12	260	130
6	520	
3	1, 040	
1	2, 080	1, 040
		2, 080
		3, 250

SUNG KYUN KWAN UNIVERSITY

Josephus problem

- [Josephus Flavius Game]
  - Josephus Flavius was a famous Jewish historian of the first century at the time of the Second Temple destruction. During the Jewish-Roman war he got trapped in a cave with a group of 40 soldiers surrounded by romans. The legend has it that preferring suicide to capture, the Jews decided to form a circle and, proceeding around it, to kill every third remaining person until no one was left. Josephus, not keen to die, quickly found the safe spot in the circle and thus stayed alive.



SUNG KYUN KWAN UNIVERSITY

DECREASE BY A VARIABLE SIZE

Decrease by a variable size - Examples

- Euclid’s algorithm for greatest common divisor
- Selection problem
- Binary search tree

Euclid's algorithm for greatest common divisor

- 50 : 1, 2, 5, 10, 25, 50

  20 : 1, 2, 4, 5, 10, 20
- GCD(x, y) = GCD(x-y, y)

  GCD(x,y) = GCD(y,x)

  Ex) GCD(50,20) = GCD(30,20) = GCD(10,20) = GCD(20,10) = GCD(10,10) = GCD(0,10) = GCD(10,0)
- GCD(X,Y) = GCD(X%Y, Y)

  Ex) GCD(50,20) = GCD(10,20) = GCD (20,10) = GCD(0,10)

Euclid's algorithm for greatest common divisor

- Steps for 2 natural numbers *n* and *m*
  - E1 : r ← m mod n;
  - E2 : if (r = 0) then return;
  - E3 : m ← n; n ← r;

		a	b	Explanations
	gcd(	1071,	1029)	The initial arguments
=	gcd(	1029,	42)	The second argument is 1071 mod 1029
=	gcd(	42,	21)	The second argument is 1029 mod 42
=	gcd(	21,	0)	The second argument is 42 mod 21
=		21		Since b=0, we return a

SUNG KYUN KWAN UNIVERSITY

The selection problem

- Input: A set S of n elements
- Output: The kth smallest element of S
- $K = \left\lceil \frac{n}{2} \right\rceil$ To find the median (the middle value)
- $K = 1$ To find the smallest element
- $K = n$ To find the largest element

The selection problem

- Input: A set S of n elements
- Output: The kth smallest element of S
- The straightforward algorithm:
  - step 1: Sort the n elements
  - step 2: Locate the kth element in the sorted list.

♌ This algorithm is overkill!

Search problem

- Search a key in a sorted list
  - Sequential search
  - Binary search

- How do you search for a name in a telephone book?

  - Using binary search

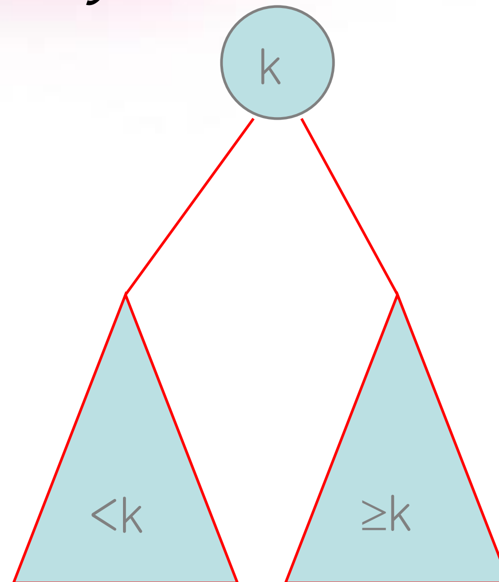
Binary Search Trees (BSTs)

- Binary Search Tree property
  - A binary tree in which the key of an internal node is greater than the keys in its left subtree and less than or equal to the keys in its right subtree.

- An inorder traversal of a binary search tree produces a sorted list of keys.

Variable-size-decrease: Binary search trees

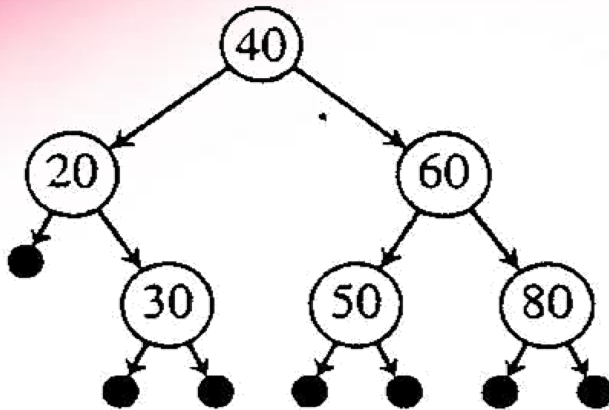
- Keys are arranged in a binary tree with the *binary search tree property*.

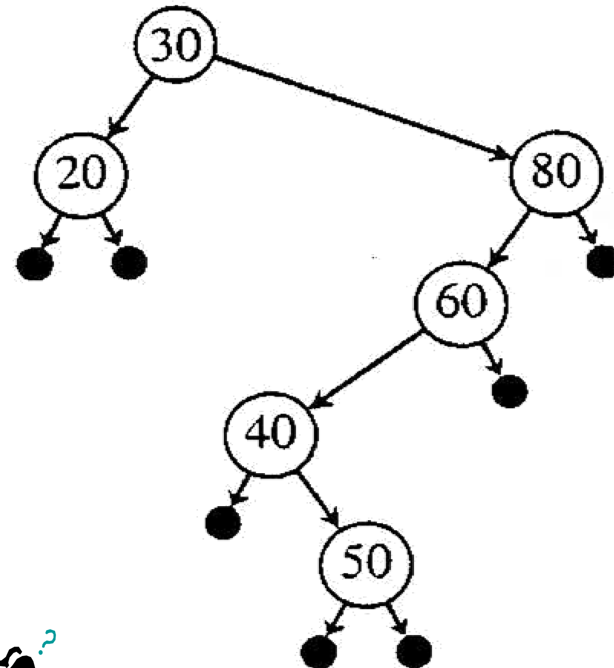


SUNG KYUN KWAN UNIVERSITY

BST Examples

- Binary Search trees with different degrees of balance
- Black dots denote empty trees



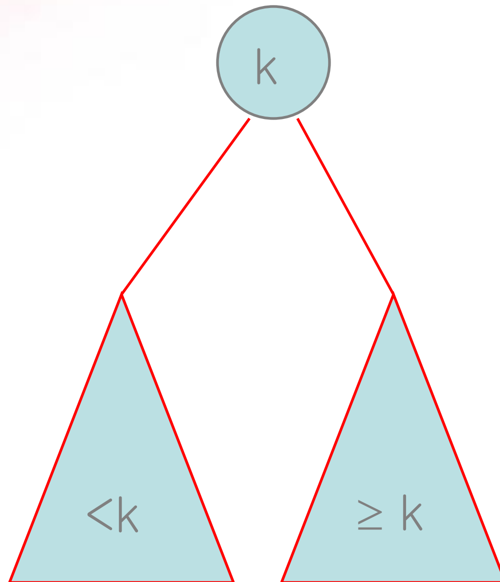


- Size of a search tree

SUNG KYUN KWAN UNIVERSITY

Variable-size-decrease: Binary search trees

- Arrange keys in a binary tree with the *binary search tree property*.

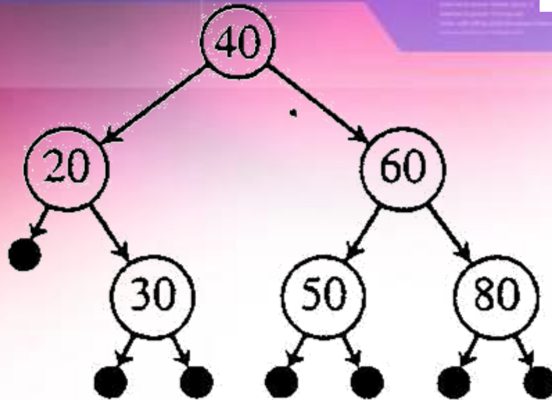


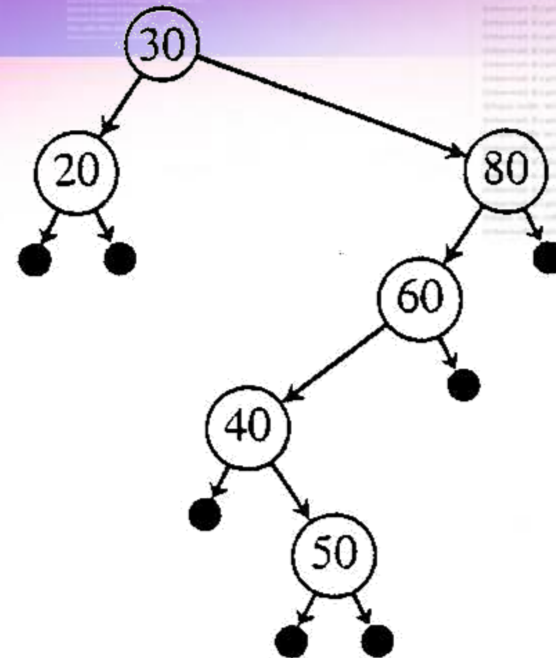
Example 1: 5, 10, 3, 1, 7, 12, 9

Example 2: 4, 5, 7, 2, 1, 3, 6

SUNG KYUN KWAN UNIVERSITY

BST Operations





- Find the min/max element (√)
- Search for an element
- Find the successor/predecessor of an element
- Insert an element
- Delete an element

BST: Min/Max

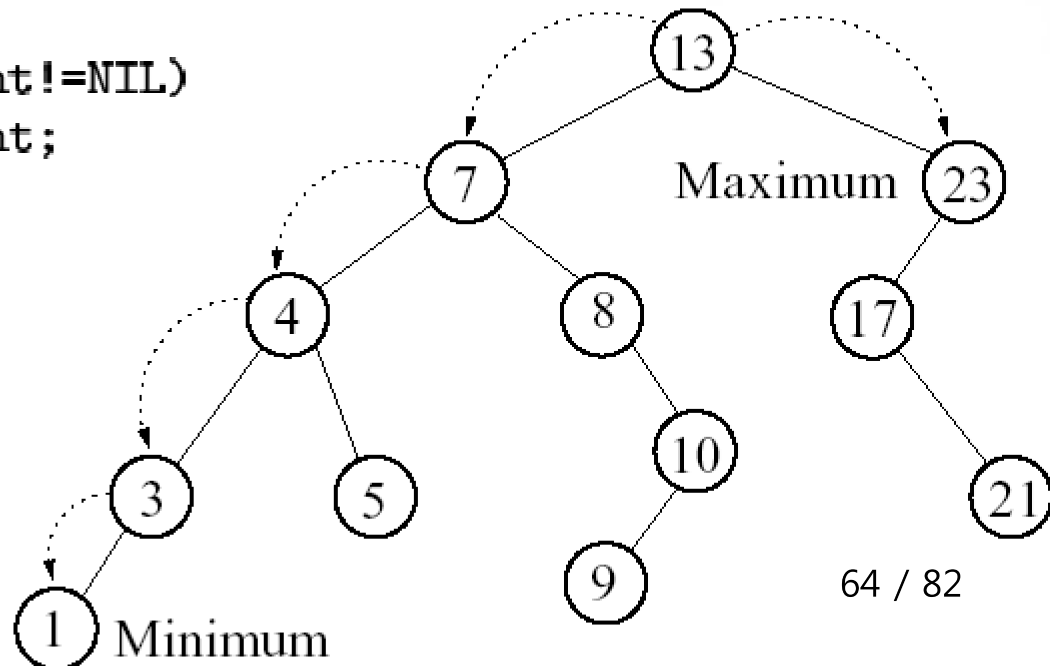
- The minimum element is the left-most node

x is a non-empty BST

```
node Find_Min(x) {
    while(x->left!=NIL)
        x=x->left;
    return x;
    }
```

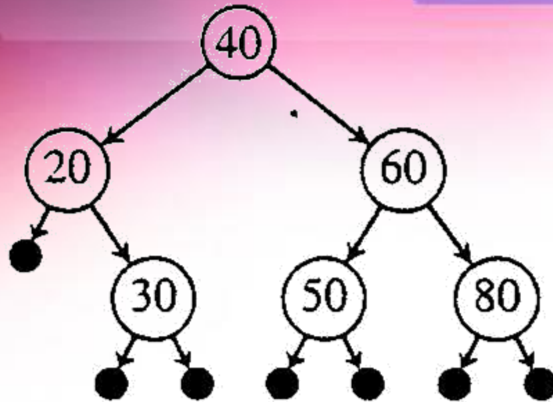
- The maximum element is the right-most node

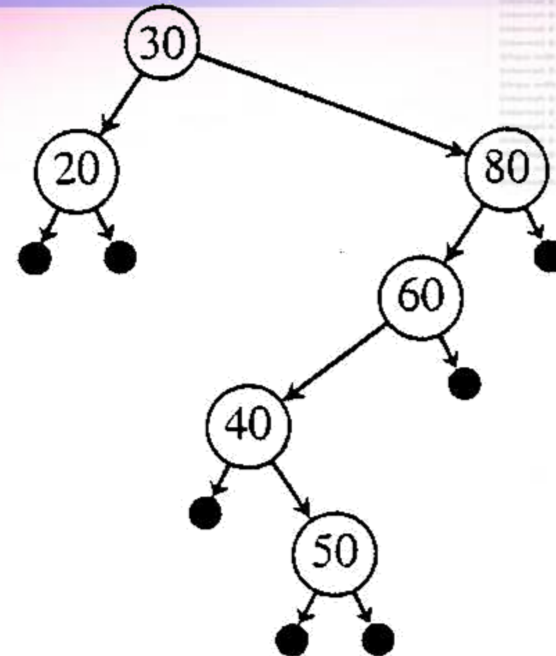
```
node Find_Max(x) {
    while(x->right!=NIL)
        x=x->right;
    return x;
    }
```



SUNG KYUN KWAN UNIVERSITY

BST Operations

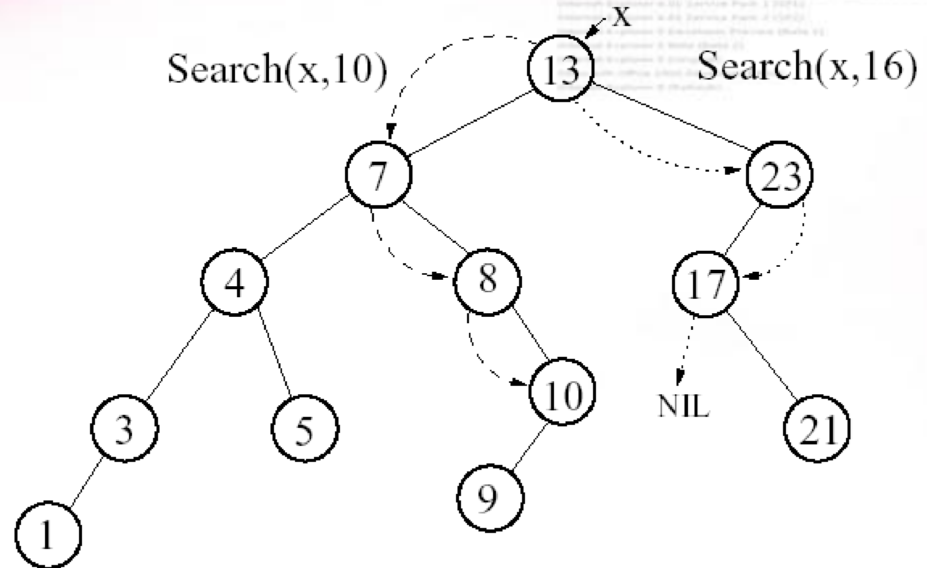




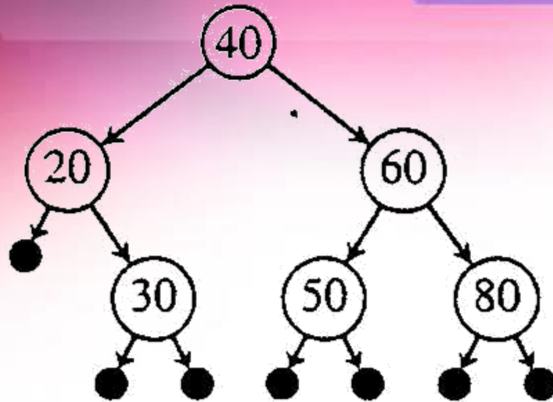
- Find the min/max element
- Search for an element (√)
- Find the successor/predecessor of an element
- Insert an element
- Delete an element

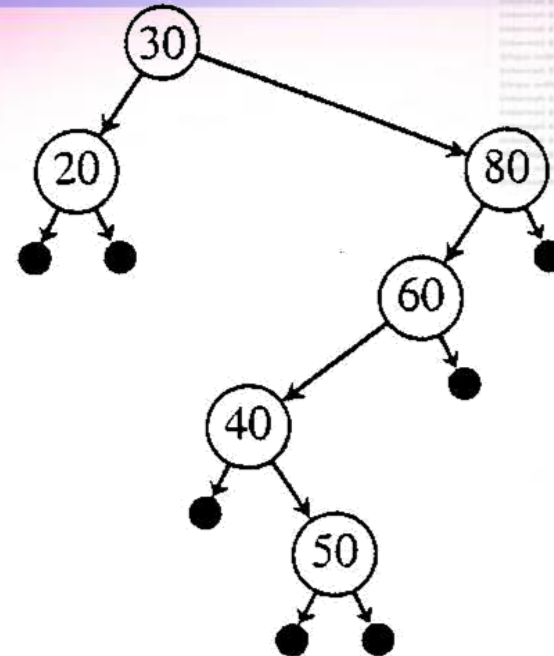
BST Search (Retrieval)

```
Element bstSearch(BinTree bst, Key K)
1.  Element found
2.  if (bst == nil)
3.              found = null;
4.  else
5.      Element root = root(bst);
6.      if (K == root.key)
7.          found = root;
8.      else if (K < root.key)
9.          found = bstSearch (leftSubtree(bst), K);
10.     else
11.         found = bstSearch(rightSubtree(bst), K);
12. return found;
```



BST Operations

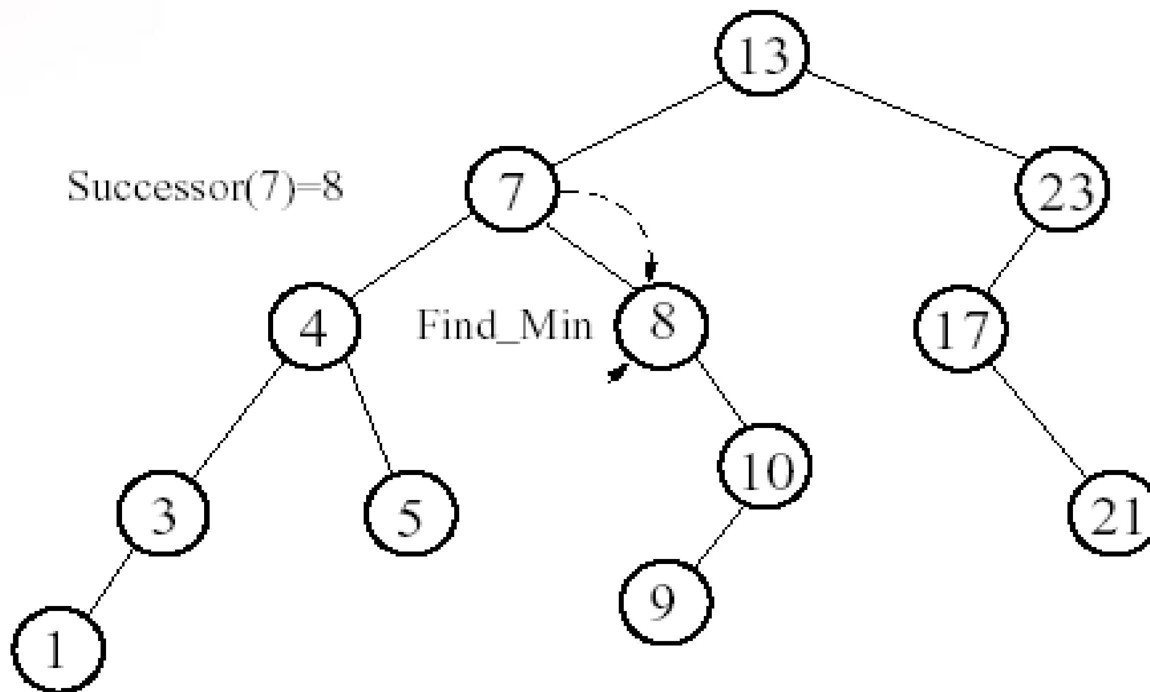




- Find the min/max element
- Search for an element
- Find the successor/predecessor of an element (√)
- Insert an element
- Delete an element

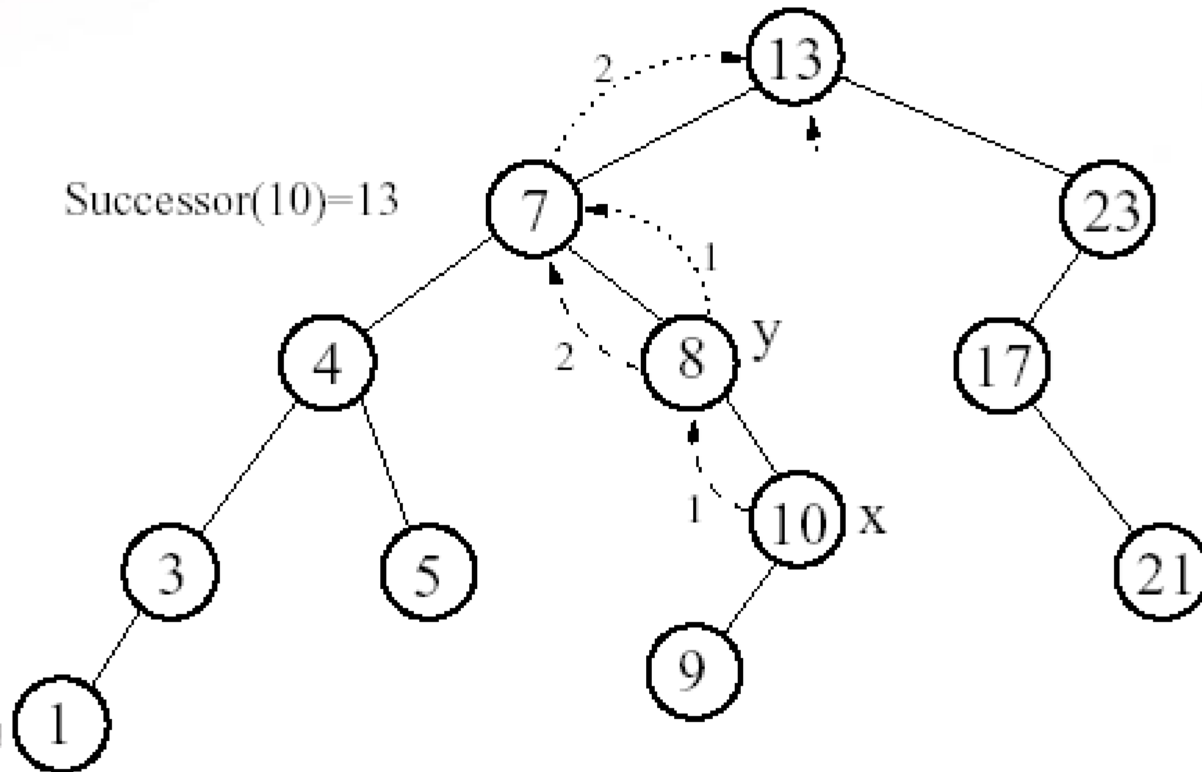
BST: Successor/Predecessor

- Finding the successor of a node *x* (if it exists):
  - If *x* has a nonempty right subtree, then successor(x) is the smallest element in the tree root at rightSubtree(x)



BST: Successor/Predecessor

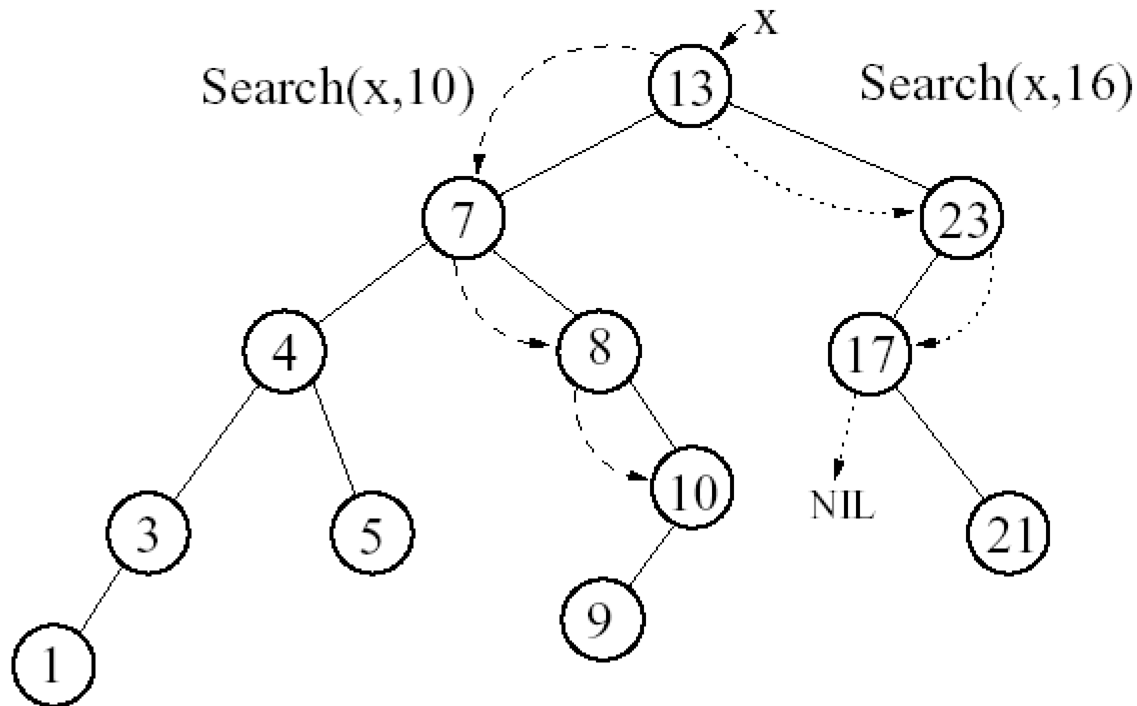
- Finding the successor of a node *x* (if it exists):
  - If rightSubtree(*x*) is empty, then successor(*x*) is the lowest ancestor of *x* whose left child is also an ancestor of *x*.



Why binary search tree?

Array: 1 3 4 5 7 8 9 10 13 17 21 23

BST:

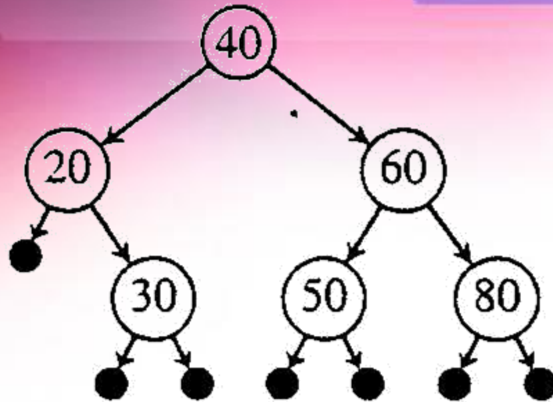


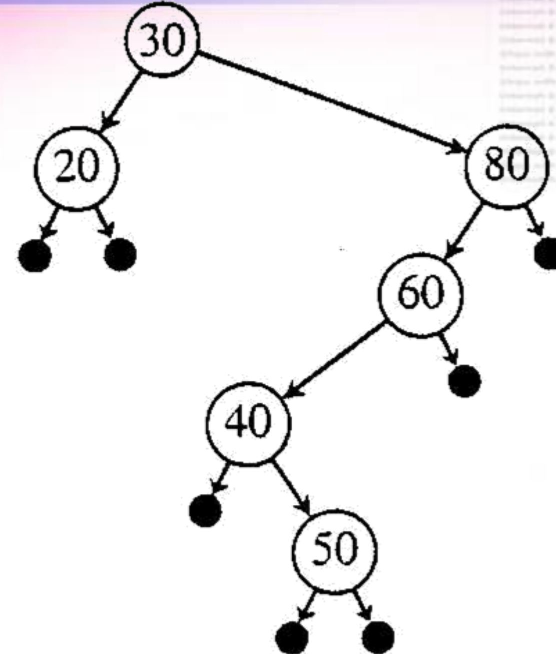
SUNG KYUN KWAN UNIVERSITY

BST: advantage

- The advantage to the binary search tree approach is that it combines the advantage of an array--the ability to do a binary search with the advantage of a linked list--its dynamic size.

BST Operations

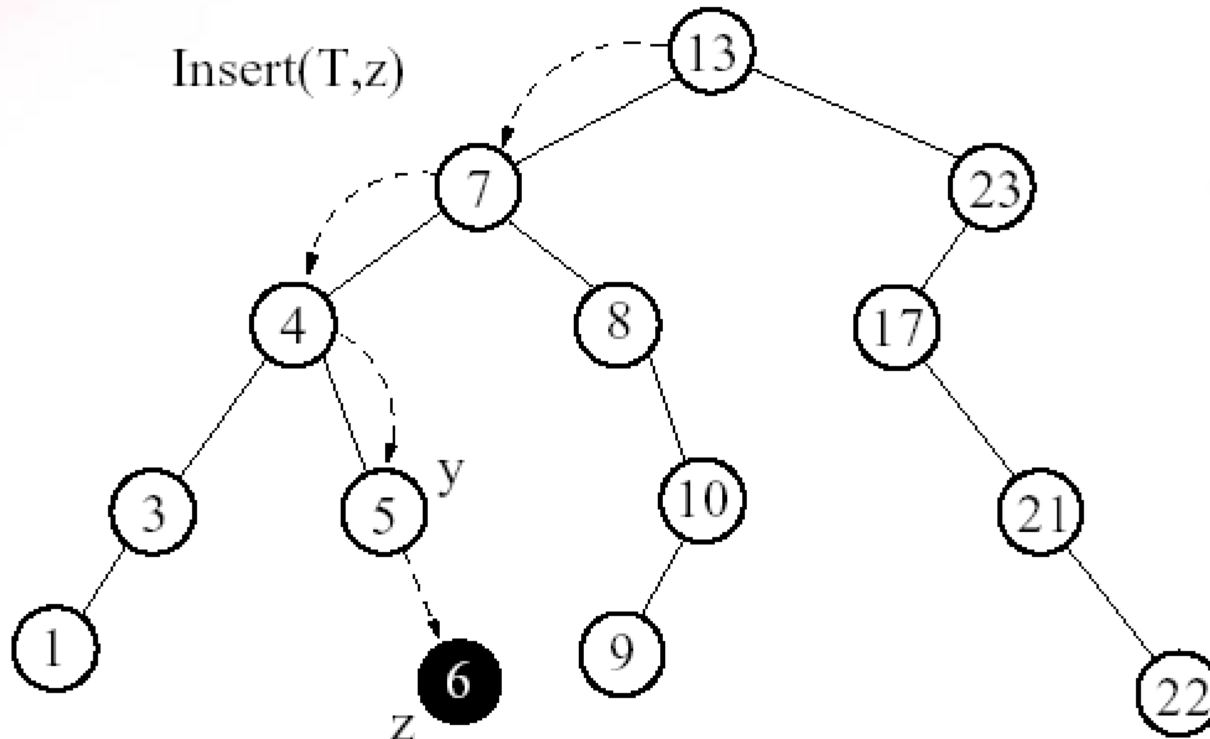




- Find the min/max element
- Search for an element
- Find the successor/predecessor of an element
- Insert an element (√)
- Delete an element

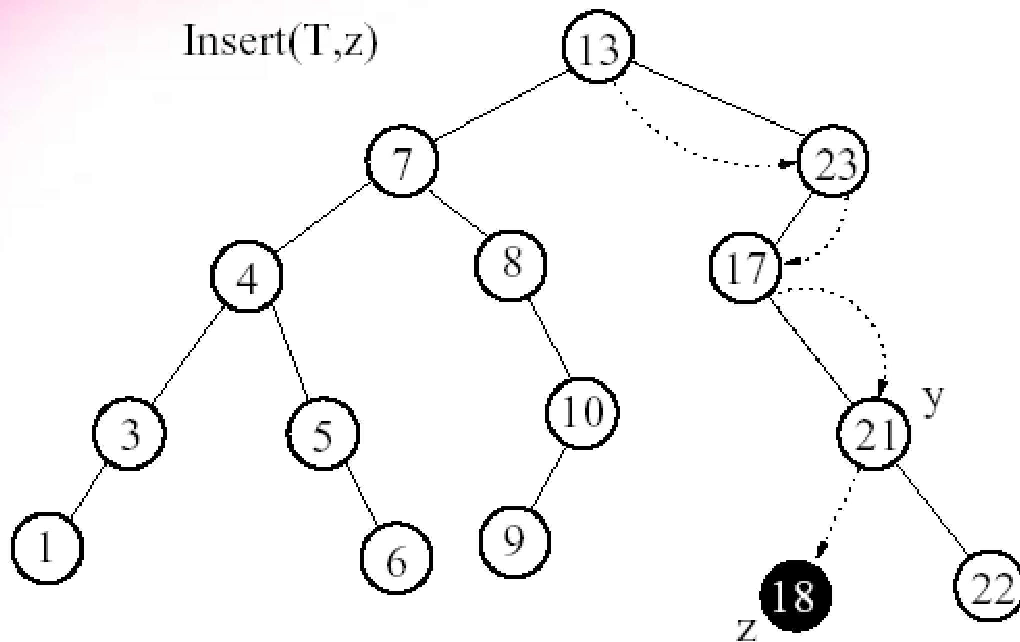
BST: Insertion

- To insert a node into a BST, we search the tree until we find a node whose appropriate subtree is empty, and insert the new node there.



BST: Insertion

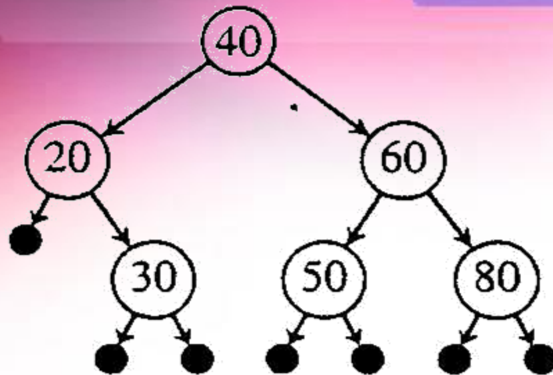
- Sample code for Insert

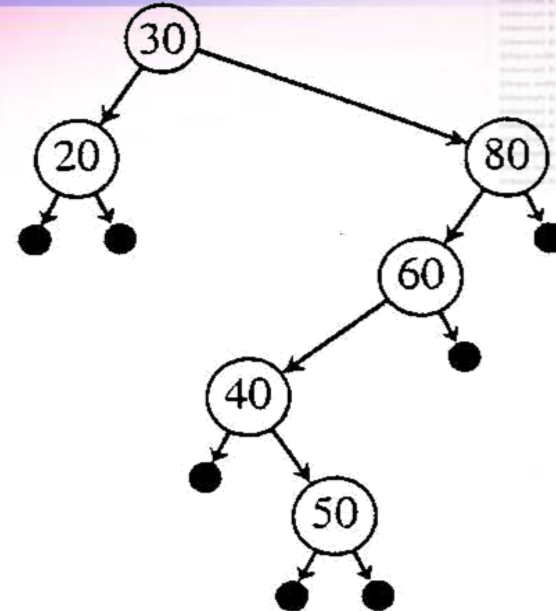



```
Insert(T,z) {
    node y=NIL;
    x=T.root;
    while(x!=NIL) {
        y=x;
        if(z.key<x.key)
          x=x->left;
        else
          x=x->right;
    }
    z->parent=y;
    if(y==NIL)
        T.root=z;
    else
        if(z.key<y.key)
           y->left=z;
        else
           y->right=z;
}
```

SUNG KYUN KWAN UNIVERSITY

BST Operations



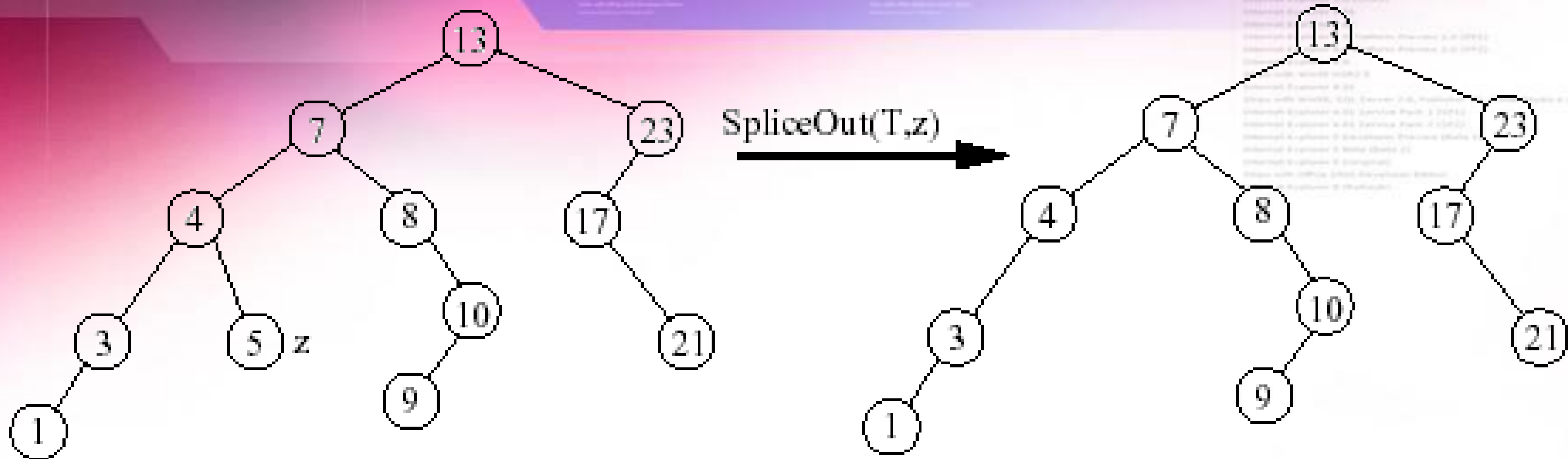


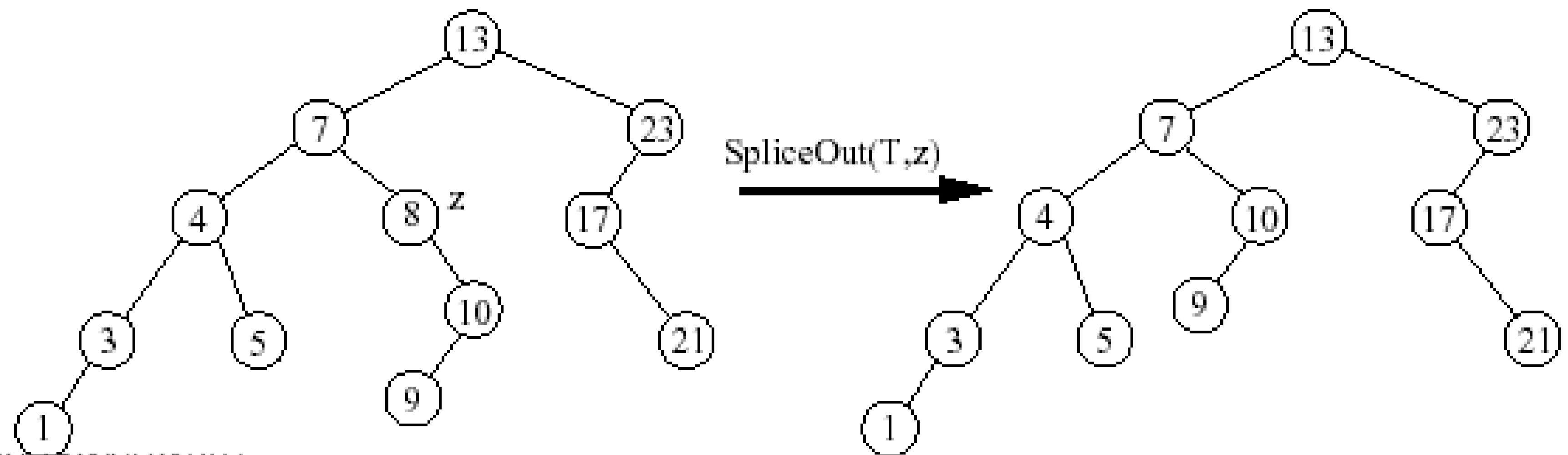
- Find the min/max element
- Search for an element
- Find the successor/predecessor of an element
- Insert an element
- Delete an element (√)

BST: Delete

- Deleting a node *z* is by far the most difficult BST operation.
- There are three cases to consider
  - If *z* has no children, just delete it.
  - If *z* has one child, splice out *z*, That is, link *z*'s parent and child
  - If *z* has two children, splice out *z*'s successor *y*, and replace the contents of *z* with the contents of *y*
- The last case works because if z has two children, then its successor has no left child

BST: splice out examples





BST: splice out algorithm

Only works when a node has at most one child

```
SpliceOut(T,y) {
                        //Two children--can't splice out.
  if(y->left!=NIL && y->right!=NIL)
     return;

  if(y->left!=NIL)              //Locate child of y
     x=y->left;
  else if (y->right!=NIL)
     x=y->right;
  else
     x=NIL;

  if(x!=NIL)              //If y has child, set parent
     x->parent=y->parent;

                 //Set y's parent's child to y's child
  if(y->parent==NIL)
     x=T.root;
  else {
     if(y==y->parent->left)
        y->parent->left=x;
     else
        y->parent->right=x;
     }
}
```

SUNG KYUN KWAN UNIVERSITY

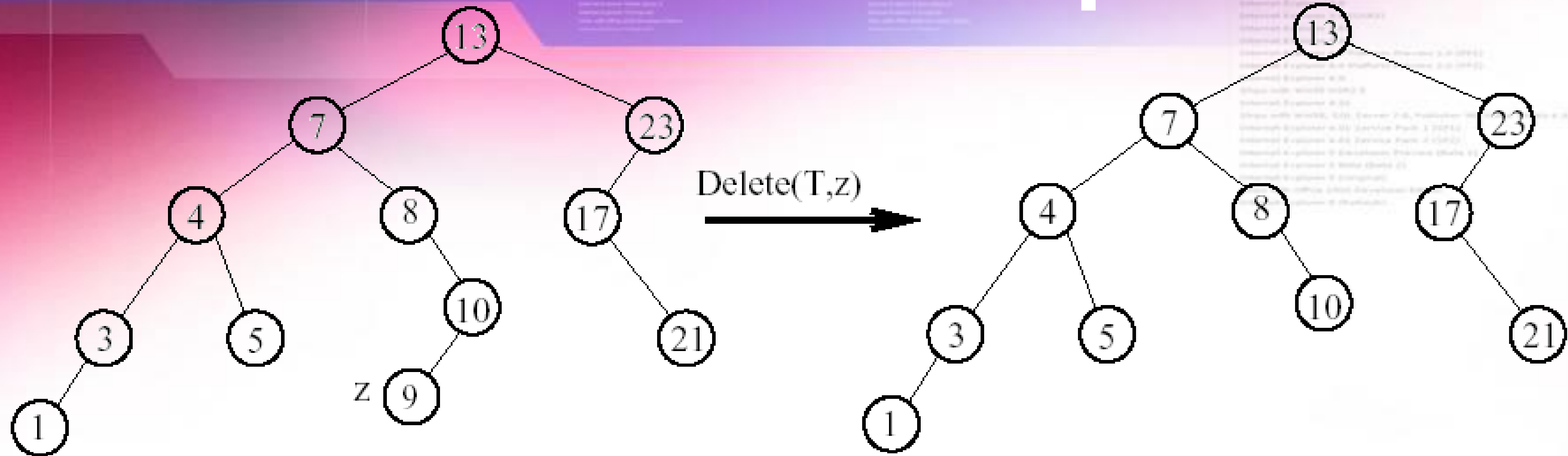
BST: Deletion Algorithm

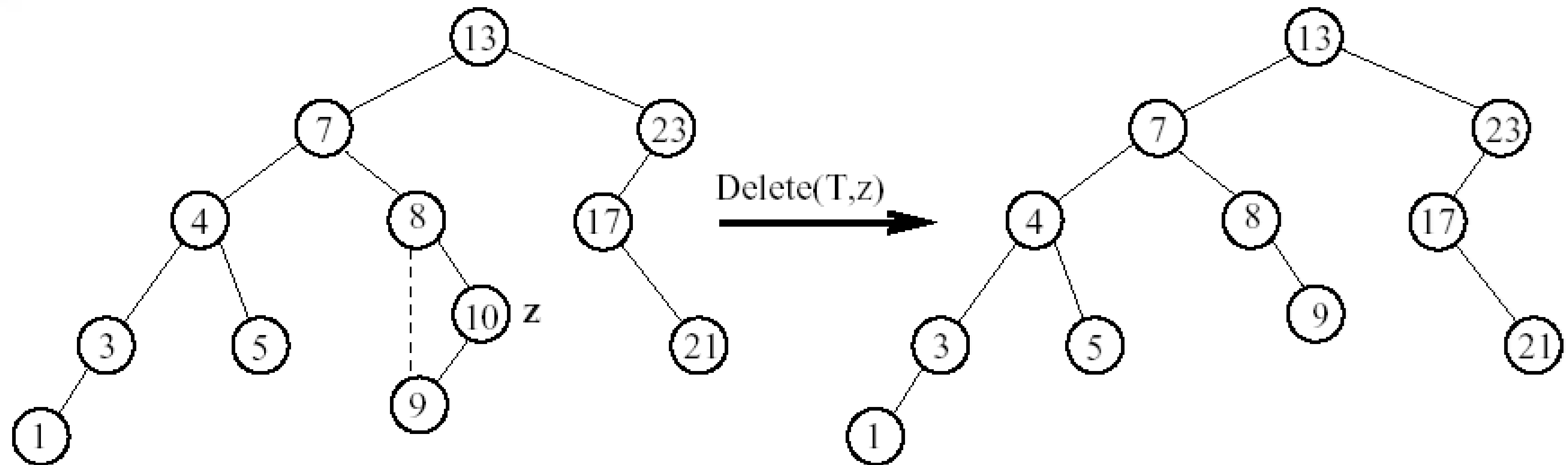
- Delete is now simple!

```
Delete(T,z) {
        if(z->left==NIL || z->right==NIL)
            SpliceOut(T,z);
        else {
            y=Successor(z);
            z->key=y->key;
            SpliceOut(T,y);
            }
    }
```

SUNG KYUN KWAN UNIVERSITY

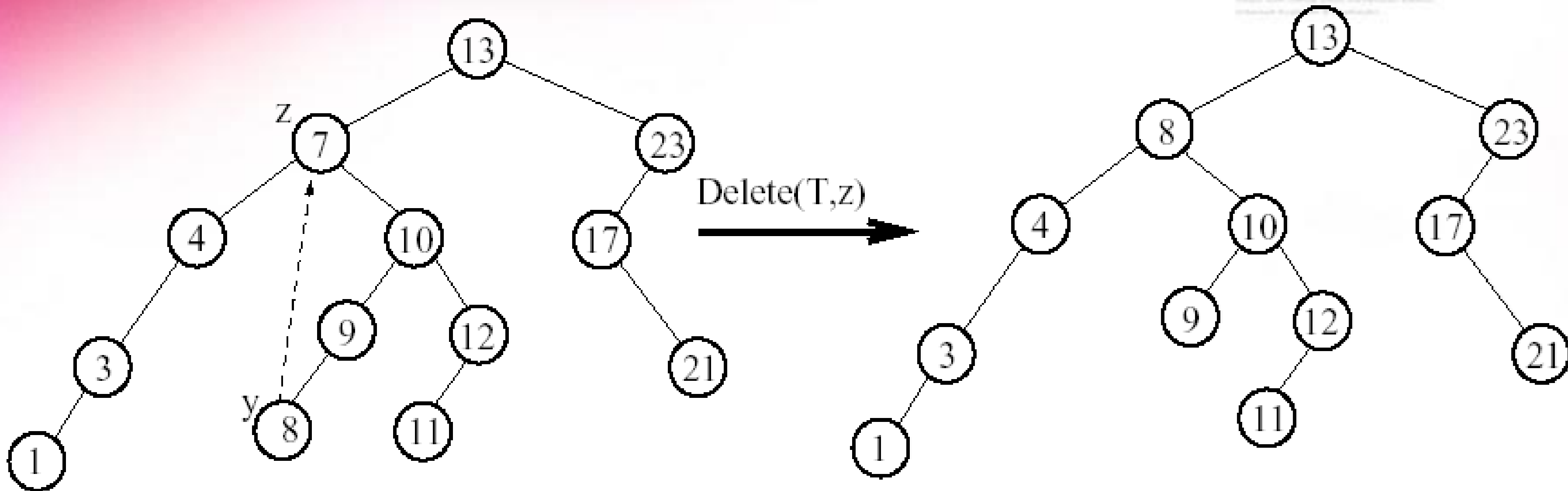
BST: delete examples





SUNG KYUN KWAN UNIVERSITY

BST: one more delete example



Q & A