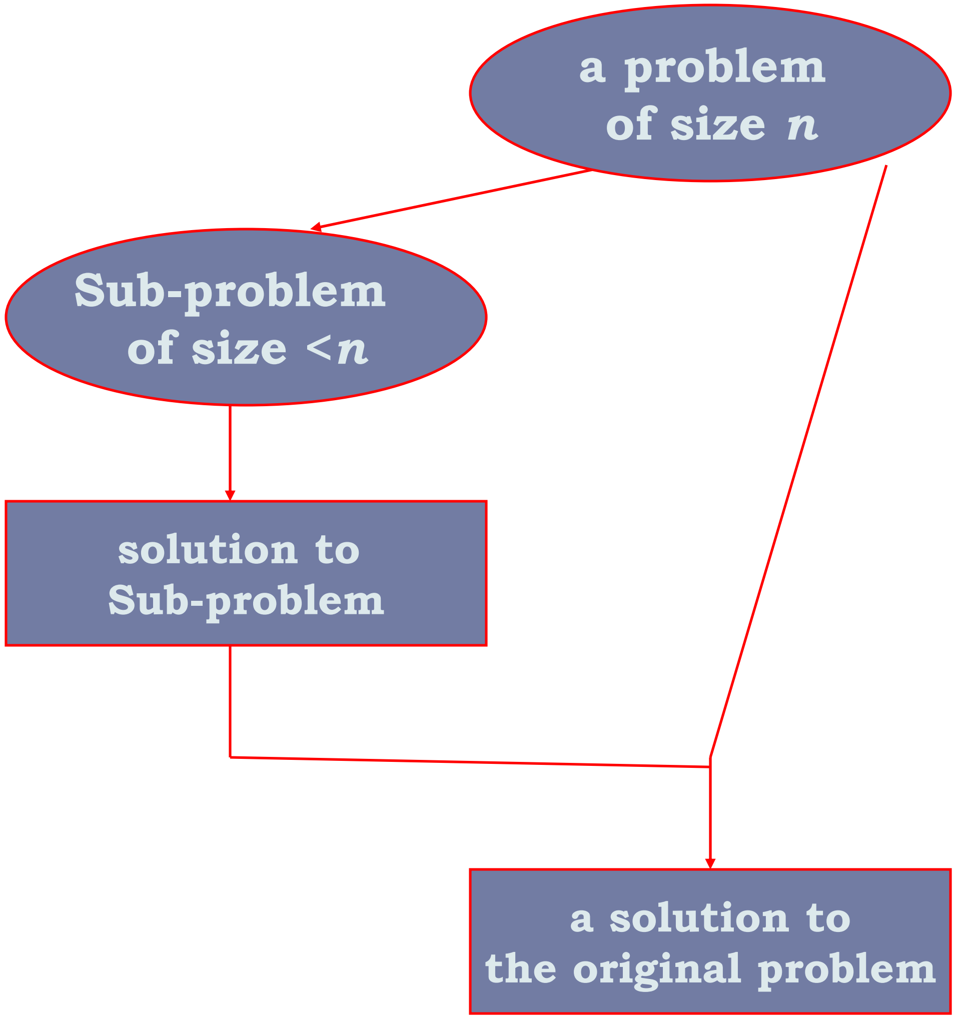
Decrease-and-Conquer

Decrease-and-Conquer Technique



3 Types

- **Decrease by a constant (usually by 1):**
  - insertion sort
  - graph traversal algorithms (DFS and BFS)
  - topological sorting
  - algorithms for generating permutations, subsets

- **Decrease by a constant factor (usually by half):**
  - binary search and bisection method
  - exponentiation by squaring
  - multiplication à la russe

- **Variable-size decrease:**
  - Euclid's algorithm
  - selection by partition
  - Nim-like games

Insertion Sort

- To sort array $A[0..n-1]$, sort $A[0..n-2]$ recursively and then insert $A[n-1]$ in its proper place among the sorted $A[0..n-2]$

- Usually implemented bottom up (non-recursively)

$$\underbrace{A[0] \le \cdots \le A[j]}_{\text{smaller than or equal to } A[i]} < \underbrace{A[j+1] \le \cdots \le A[i-1]}_{\text{greater than } A[i]} \mid A[i] \cdots A[n-1]$$

ALGORITHM *InsertionSort*$(A[0..n-1])$

```
//Sorts a given array by insertion sort
//Input: An array A[0..n − 1] of n orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```

Analysis of Insertion Sort

- Time efficiency

$$C_{worst}(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

$$C_{avg}(n) \cong \frac{n^2}{4} \in \Theta(n^2)$$

$$C_{best} = n - 1 \in \Theta(n)$$

- Space efficiency:

  in-place

- Stability:

  yes (A sorting algorithm is called ***stable*** if it preserves the relative order of any two equal elements in its input.)
- Best elementary sorting algorithm overall

Graph Traversal

- Many problems require processing all graph vertices

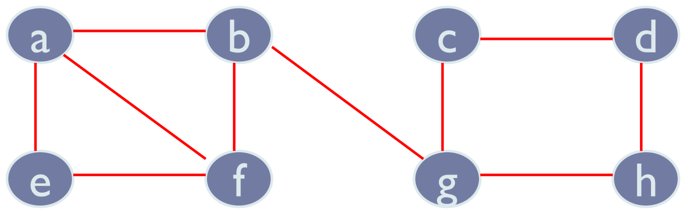
Graph traversal algorithms:

  - Breadth-first search (BFS)
  - Depth-first search (DFS)

Breadth-first search (BFS)

- Visits graph vertices by moving across to all the neighbors of last visited vertex
- BFS uses a queue
- Similar to level-by-level tree traversal
- The result of a BFS is a tree

Example of BFS traversal of undirected graph



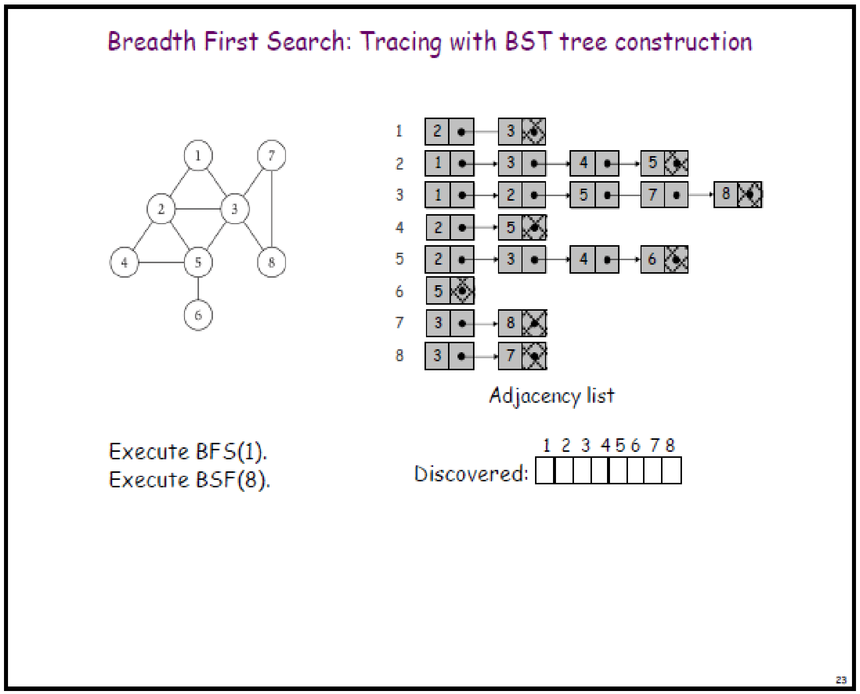
Visited array:
Queue: (one or a separate one for each layer)
Visiting order:

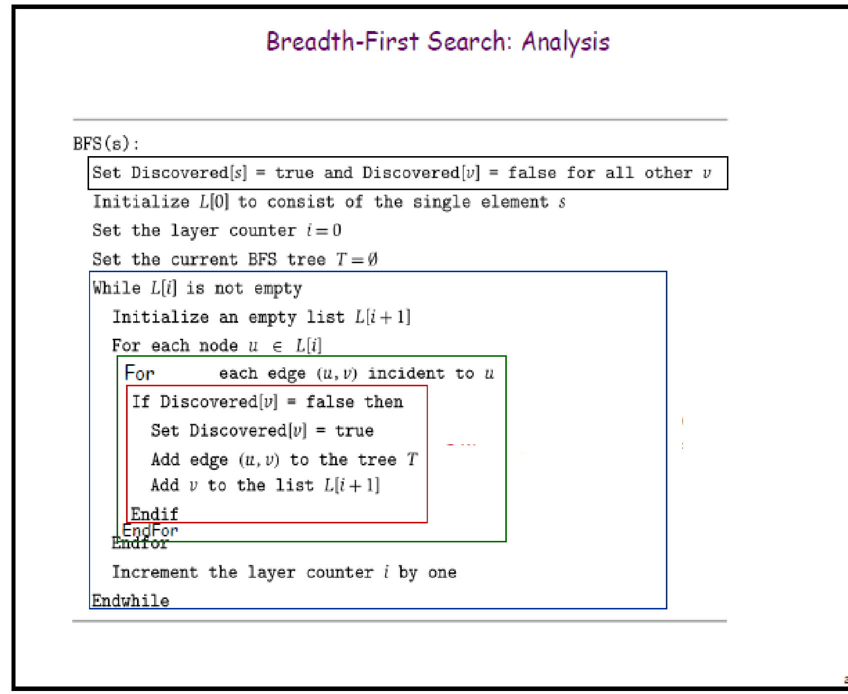
- Add *a* to the queue
- While the queue is not empty
  - pop the head of the queue
  - append it to the Visiting order
  - append all non-visited neighbors to the queue
  - mark these neighbors in the visited array

Breadth-first search (BFS)

```
BFS(s):
  Set Discovered[s] = true and Discovered[v] = false for all other v
  Initialize L[0] to consist of the single element s
  Set the layer counter i = 0
  Set the current BFS tree T = ∅
  While L[i] is not empty
    Initialize an empty list L[i+1]
    For each node u ∈ L[i]
      For each edge (u,v) incident to u
        If Discovered[v] = false then
          Set Discovered[v] = true
          Add edge (u,v) to the tree T
          Add v to the list L[i+1]
        Endif
      EndFor
    Endfor
    Increment the layer counter i by one
  Endwhile
```

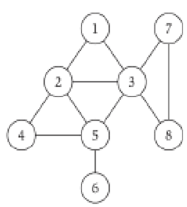
Breadth-first search (BFS)

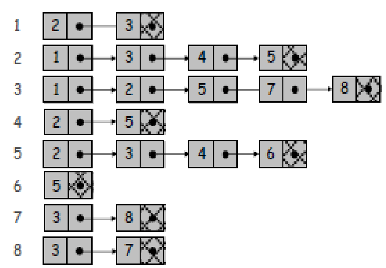




Breadth-first search (BFS)

Breadth First Search: Tracing with BST tree construction





Adjacency list

Execute BFS(1).
Execute BSF(8).

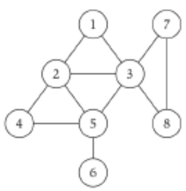
1 2 3 4 5 6 7 8
Discovered:

Breadth-First Search: Analysis

```
BFS(s):
  Set Discovered[s] = true and Discovered[v] = false for all other v        O(n)
  Initialize L[0] to consist of the single element s
  Set the layer counter i = 0
  Set the current BFS tree T = ∅
  While L[i] is not empty
    Initialize an empty list L[i+1]
    For each node u ∈ L[i]
      For     each edge (u,v) incident to u
        If Discovered[v] = false then
          Set Discovered[v] = true
          Add edge (u,v) to the tree T        O(1)    x deg(u) times     O(1)·Σ_{u∈V} deg(u)
          Add v to the list L[i+1]                                     = O(2m) = O(m)
        Endif
      EndFor
    Endfor
    Increment the layer counter i by one
  Endwhile
```

Breadth-first search (BFS)

Breadth First Search: Tracing with BST tree construction



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Adjacency Matrix

Execute BFS(1).
Execute BSF(8).

Discovered:

1	2	3	4	5	6	7	8

23

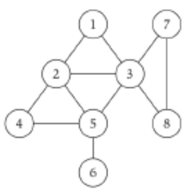
Breadth-First Search: Analysis

```
BFS(s):
 Set Discovered[s] = true and Discovered[v] = false for all other v
 Initialize L[0] to consist of the single element s
 Set the layer counter i=0
 Set the current BFS tree T=∅
 While L[i] is not empty
   Initialize an empty list L[i+1]
   For each node u ∈ L[i]
    For      each edge (u,v) incident to u
     If Discovered[v] = false then
       Set Discovered[v] = true
       Add edge (u,v) to the tree T
       Add v to the list L[i+1]
     Endif
    EndFor
   Endfor
   Increment the layer counter i by one
 Endwhile
```

24

Breadth-first search (BFS)

Breadth First Search: Tracing with BST tree construction



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Adjacency Matrix

Execute BFS(1).
Execute BSF(8).

	1	2	3	4	5	6	7	8
Discovered:								

23

Breadth-First Search: Analysis

```
BFS(s):
  Set Discovered[s] = true and Discovered[v] = false for all other v      O(n)
  Initialize L[0] to consist of the single element s
  Set the layer counter i = 0
  Set the current BFS tree T = Ø
  While L[i] is not empty
    Initialize an empty list L[i+1]
    For each node u ∈ L[i]
      For each edge (u,v) incident to u
        If Discovered[v] = false then
          Set Discovered[v] = true
          Add edge (u,v) to the tree T          O(1)     x n times     x n times
          Add v to the list L[i+1]
        Endif
      EndFor
    Endfor
    Increment the layer counter i by one
  Endwhile
```

24

Notes on BFS

- BFS has same efficiency as DFS and can be implemented with graphs represented as:
  - adjacency matrices: $\Theta(|V|^2)$
  - adjacency lists: $\Theta(|V|+|E|)$

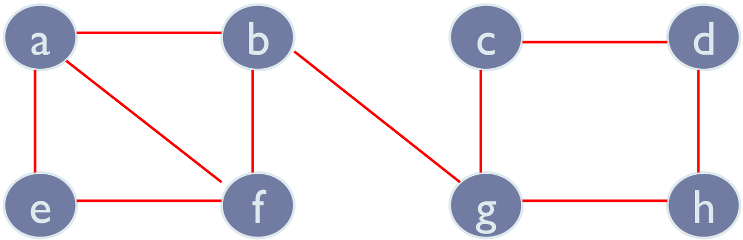
- Yields single ordering of vertices (order added/deleted from queue is the same)

- Applications:
  - find paths from a vertex to all other vertices with the smallest number of edges

Depth-First Search (DFS)

- Visits graph's vertices by always moving away from last visited vertex to unvisited one
- Backtracks if no adjacent unvisited vertex is available.
- Uses a stack
  - a vertex is pushed onto the stack when it's reached for the first time
  - a vertex is popped off the stack when it becomes a dead end, i.e., when there is no adjacent unvisited vertex

Example: DFS traversal of undirected graph



Visited array:
Stack:
Visiting order:
Pop order:

- Add *a* to the stack
- While stack is not empty:
  - Pop the head of the stack
  - Add it to the pop order
  - Add all non-visited neighbors to the visiting order and the stack
  - Mark these neighbors in visited array

Notes on DFS

- DFS can be implemented with graphs represented as:
  - adjacency matrices: $\Theta(V^2)$
  - adjacency lists: $\Theta(|V|+|E|)$
- Yields two distinct ordering of vertices:
  - order in which vertices are first encountered (pushed onto stack)
  - order in which vertices become dead-ends (popped off stack)
- Applications:
  - checking connectivity, finding connected components
  - checking acyclicity
  - finding articulation points and biconnected components
  - searching state-space of problems for solution (AI)