Announcement

HW #3 has been posted on the website and Blackboard.

Due: Thursday, Feb. 24 before class starts

Chapter 4: Decrease and Conquer

Reduce problem instance to smaller instance of the same problem

Solve smaller instance

Extend solution of smaller instance to obtain solution to original problem

Also referred to as *inductive* or *incremental* approach

Examples of Decrease and Conquer

Decrease by one:

- Insertion sort
- Graph search algorithms:
  - DFS
  - BFS
  - Topological sorting

Decrease by a constant factor

- Binary search

Variable-size decrease

- Euclid’s algorithm for computing gcd
- Selection by partition

Decrease-by-one -- Insertion Sort

This is a typical decrease-by-one technique

Assume *A*[0..*i*-1] has been sorted, how to achieve the sorted *A*[0..*i*]?

Solution: insert the last element *A*[*i*] to the correct position

Comparison direction

$$A[0] \leq \ldots \leq A[j] < A[j+1] \leq \ldots \leq A[i-1] \mid A[i] \; \ldots \; A[n-1]$$

smaller than or equal to $A[i]$ greater than $A[i]$

Insertion Sort

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
```

Input size? Basic operations?

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

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \frac{n(n-1)}{2} \in \Theta(n^2)$$

$$C_{average}(n) \approx \frac{n^2}{4} \in \Theta(n^2)$$

Insertion Sort Examples

Sort the numbers in a non-decreasing order

(a) 12 1 13 6 22 4

(b) 1 4 6 12 13 22

(c) 22 13 12 6 4 1

How many comparisons you need?

Insertion Sort Examples

12	1	13	6	22	4
1	12	13	6	22	4
1	12	13	6	22	4
1	6	12	13	22	4
1	6	12	13	22	4
1	4	6	12	13	22

1 comparison
1 comparison
3 comparisons
1 comparison

5 comparisons

1+1+3+1+5=11 comparisons

Graph Traversal

Many problems require processing all graph vertices in systematic fashion

<u>Graph traversal algorithms:</u>

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

They can be treated as decrease-by-one strategy.

Depth-First Search (DFS)

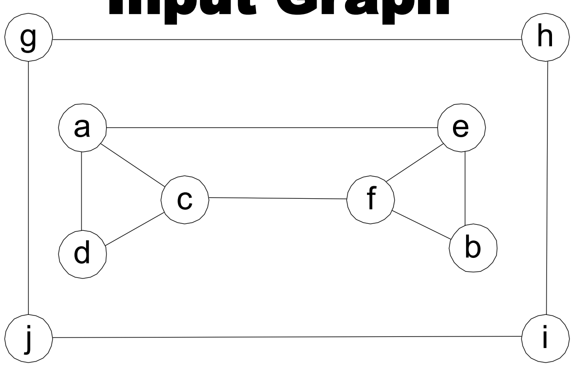
- **Visits graph's vertices by always moving away from last visited vertex to an unvisited one, backtracks if no adjacent unvisited vertex is available.**

- **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

- **Construct a DFS forest**

Example – Undirected Graph

Input Graph

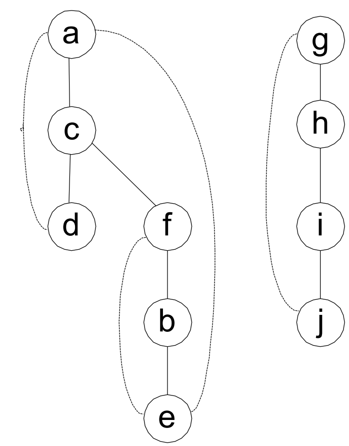


Stack push/pop

	$e_{6,2}$	
	$b_{5,3}$	$j_{10,7}$
$d_{3,1}$	$f_{4,4}$	$i_{9,8}$
$c_{2,5}$		$h_{8,9}$
$a_{1,6}$		$g_{7,10}$

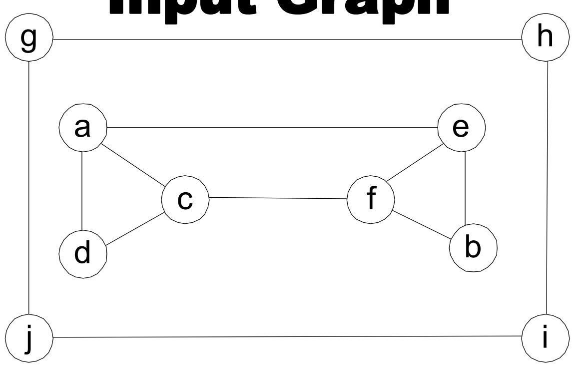
DFS forest

(Tree/Back edge)



Example – Undirected Graph

Input Graph



Adjacency matrix

	a	b	c	d	e	f	g	h	i	j
a	0	0	1	1	1	0	0	0	0	0
b	0	0	0	0	1	1	0	0	0	0
c	1	0	0	1	0	1	0	0	0	0
d	1	0	1	0	0	0	0	0	0	0
e	1	1	0	0	0	1	0	0	0	0
f	0	1	1	0	1	0	0	0	0	0
g	0	0	0	0	0	0	0	1	0	1
h	0	0	0	0	0	1	0	0	1	0
i	0	0	0	0	0	0	0	1	0	1
j	0	0	0	0	0	0	1	0	1	0

Adjacency linked list

$$a \to c \to d \to e$$

$$b \to e \to f$$

$$c \to a \to d \to f$$

$$d \to a \to c$$

$$e \to a \to b \to f$$

$$f \to b \to c \to e$$

$$g \to h \to j$$

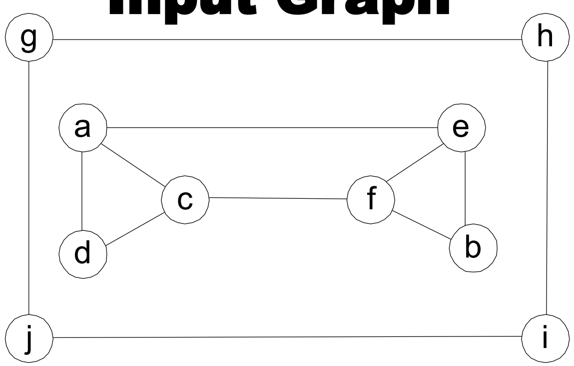
$$h \to g \to i$$

$$i \to h \to j$$

$$j \to g \to h$$

Example – Undirected Graph

Input Graph

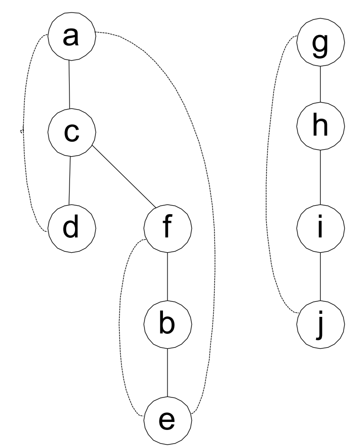


Stack push/pop

	$e_{6,2}$	
	$b_{5,3}$	$j_{10,7}$
$d_{3,1}$	$f_{4,4}$	$i_{9,8}$
$c_{2,5}$		$h_{8,9}$
$a_{1,6}$		$g_{7,10}$

DFS forest

(Tree/Back edge)



Depth-first search (DFS)

Pseudo code for Depth-first-search of graph *G*=(*V*,*E*)

ALGORITHM $DFS(G)$

// Input : Graph G $= \langle V, E \rangle$

// Output : Graph G with its vertices marked with consecutive

// integers in the order they've been first visited by DFS traversal

mark each vertex in V with 0 // label as unvisited

$count \leftarrow 0$

for each vertex v in V do

    if v is marked with 0

        $dfs(v)$

///

Sub-algorithm

$dfs(v)$

$count \leftarrow count + 1$; mark v with $count$

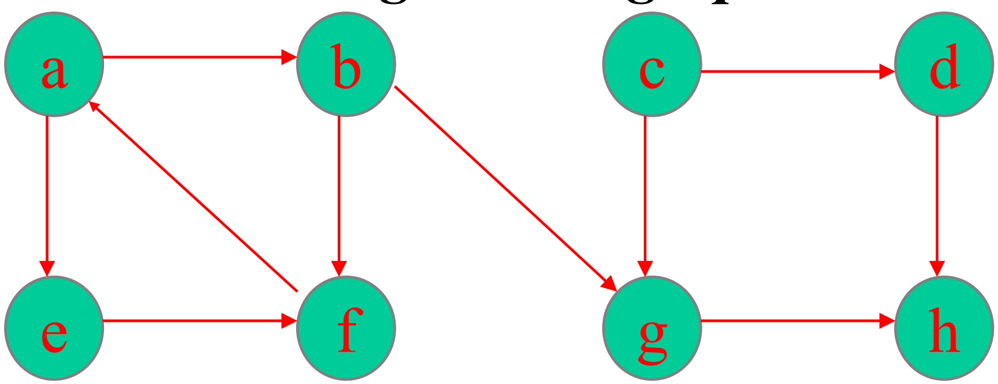
for each vertex w in V ajacent to v do

    if w is marked with 0

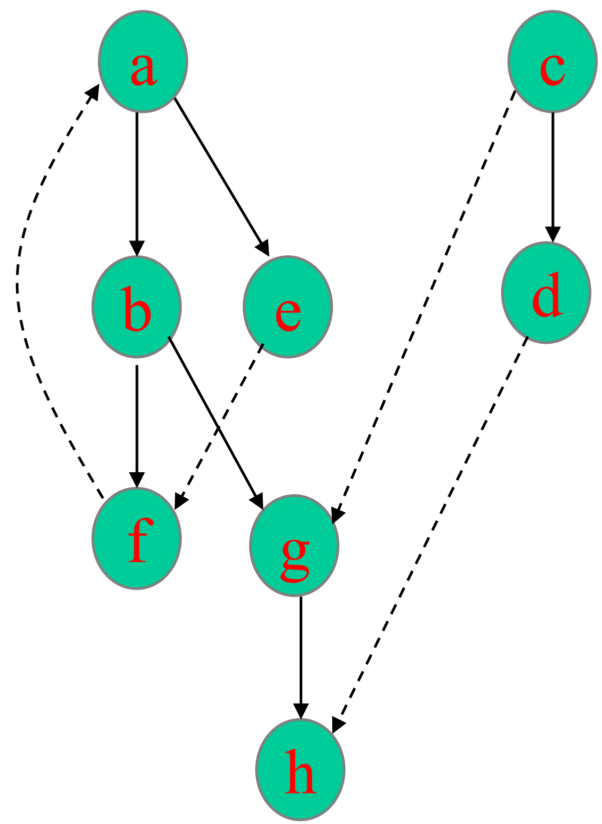
        $dfs(w)$

Example – Directed Graph (Digraph)

Original Diagraph



DFS forest



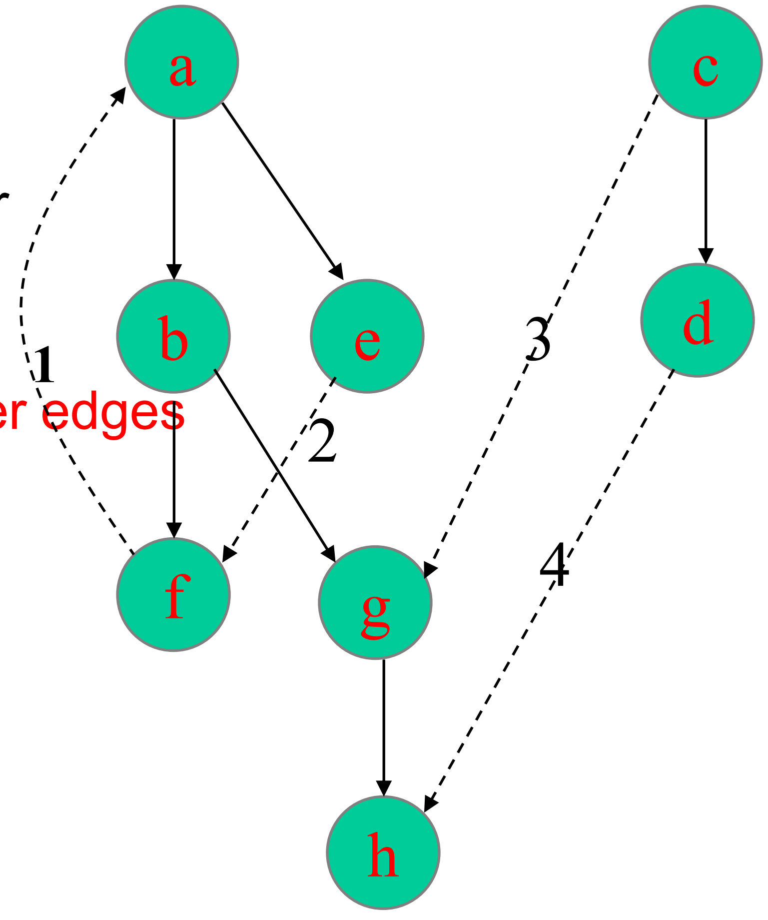
	$h_{5,2}$		
$f_{3,1}$	$g_{4,3}$		
$b_{2,4}$		$e_{6,5}$	$d_{8,7}$
$a_{1,6}$			$c_{7,8}$

Stack push/pop

DFS Forest and Stack

Four types of edges:

- Tree edge: parent to child (solid lines)
- Back edge: descendant to an ancestor
- Forward edge (directed graph only):
  a link to a nonchild descendant
- Cross edge (directed graph only): other edges



How many back edges? 1

Forward edges? 0

Cross edges? 3

DFS: Notes

DFS can be implemented with graphs represented as:

- Adjacency matrices: $\Theta(|V|^2)$
- Adjacency linked lists: $\Theta(|V|+|E|)$

Why?

Yields two distinct ordering of vertices:

- Preorder traversal: as vertices are first encountered (pushed onto stack)
- Postorder traversal: as vertices become dead-ends (popped off stack)

Applications:

- checking connectivity, finding connected components
- checking acyclicity

Breadth-First Search (BFS)

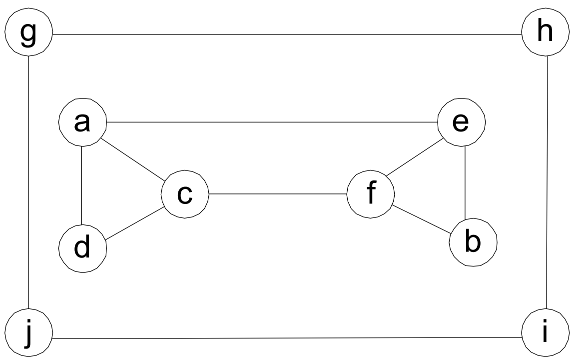
Explore graph moving across to all the neighbors of last visited vertex

Similar to level-by-level tree traversals

Instead of a stack (LIFO), breadth-first uses queue (FIFO)

Applications: same as DFS

BFS Example – undirected graph

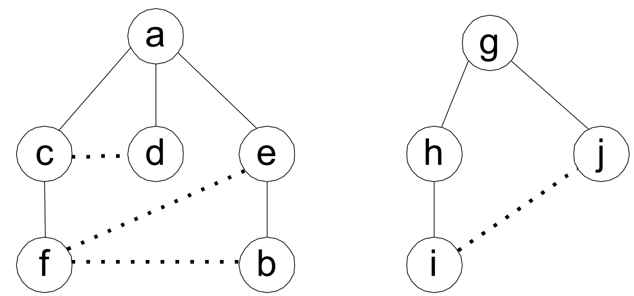


Input Graph

(Adjacency matrix / linked list

a_1	g_7
c_2	h_8
d_3	j_9
e_4	i_{10}
f_5	
b_6	

Queue



BFS forest

(Tree edge / Cross edge)

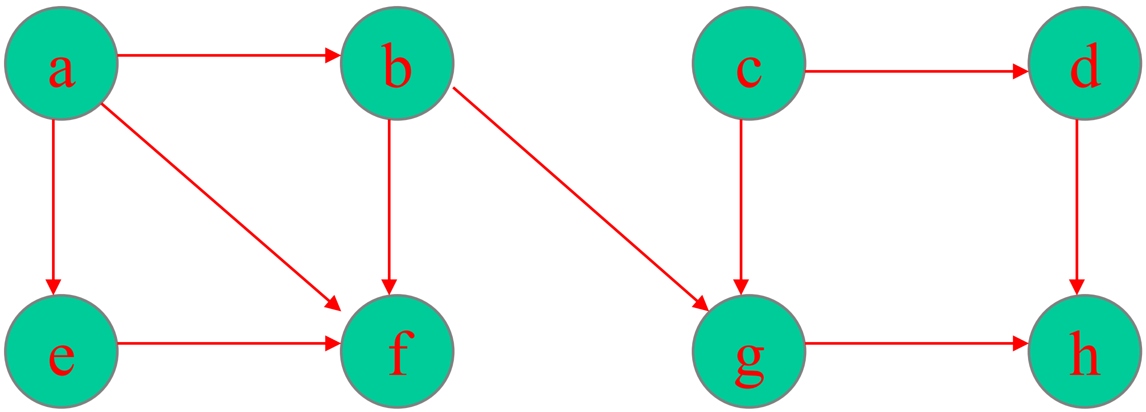
BFS algorithm

```
ALGORITHM BFS(G)
//Input: Graph G =< V, E >
//Output: Graph G with its
//vertices marked with
//consecutive integers in the
//order they've been visited by
//BFS traversal
count ← 0
mark each vertex with 0
for each vertex v in 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**
  for each vertex *w* adjacent to *the front vertex* **do**
    if *w* is marked with 0
      count ←*count* + 1
      mark *w* with *count*
      add *w* to the end of the *queue*
  remove the front vertex from the *queue*

Example – Directed Graph



BFS traversal:

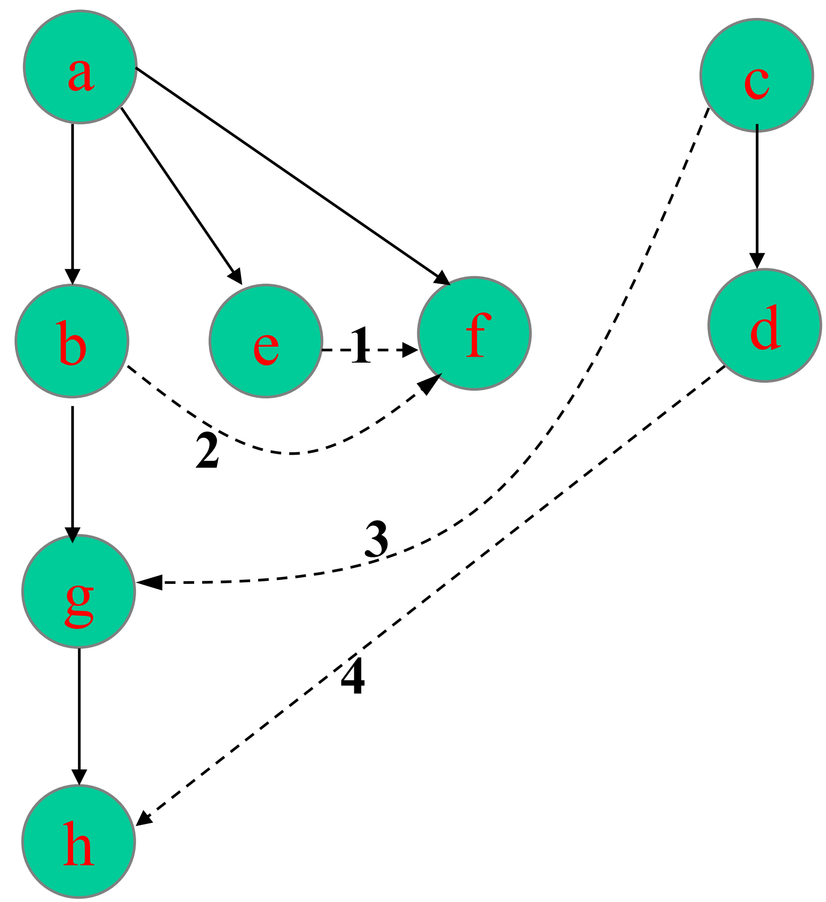
BFS Forest and Queue

a_1
b_2
e_3
f_4
g_5
h_6

c_7
d_8

How many cross edges? 4

Queue



BFS forest

Breadth-first search: Notes

BFS has same efficiency as DFS and can be implemented with graphs represented as:

- Adjacency matrices: $\Theta(|V|^2)$
- Adjacency linked lists: $\Theta(|V|+|E|)$

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

Graph Traversal

- **DFS**
  - Uses a stack
  - Yields two distinct ordering of vertices:
    - Preorder traversal: as vertices are first encountered (pushed onto stack)
    - Postorder traversal: as vertices become dead-ends (popped off stack)
  - Result in a DFS forest

    -- Tree edges, back edges, forward edges, and cross edges

- **BFS**
  - Uses a queue
  - Yields one ordering of vertices
  - Result in a BFS forest with tree edges and cross edges
- **Both DFS and BFS have efficiency**
  - Adjacency matrices: $\Theta(|V|^2)$
  - Adjacency linked lists: $\Theta(|V|+|E|)$