DSE 2256 DESIGN & ANALYSIS OF ALGORITHMS

Lecture 14 & 15

Decrease-and-Conquer:

Insertion Sort
Depth First Search
Breadth First Search

Courtesy : www.alamy.com

Instructors:

Dr. Savitha G,
Assistant Professor, DSCA, MIT, Manipal

Dr. Abhilash K. Pai,
Assistant Professor, DSCA, MIT, Manipal

Recap of L12 & L13

- Exhaustive search
  - Travelling Salesman Problem
  - Knapsack Problem
  - Assignment Problem

Decrease-and-Conquer

1. Reduce problem instance to smaller instance of the same problem

2. Solve smaller instance.

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

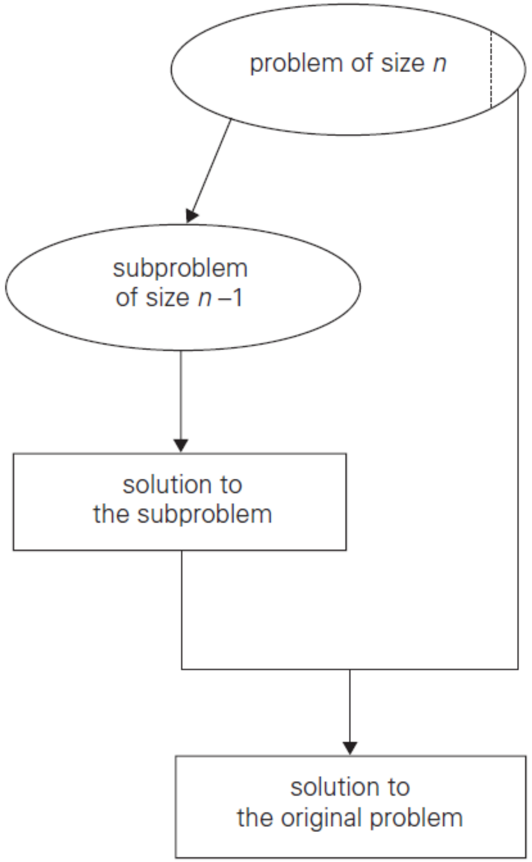
Decrease-and-Conquer: Types

Types of Decrease-and-conquer techniques:

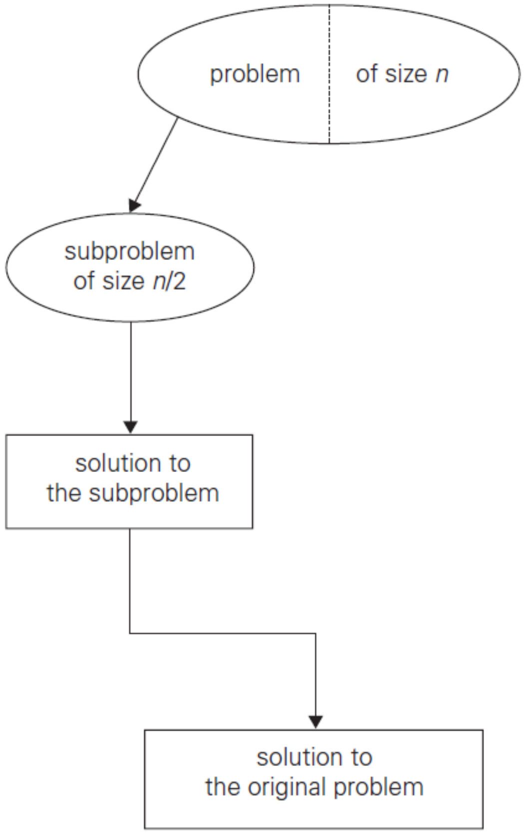
1. Decrease by a constant (usually by 1):
   - ✓ Insertion sort
   - ✓ Topological sorting

2. Decrease by a constant factor (usually by half):
   - ✓ Binary search

3. Variable-size decrease
   - ✓ Euclid's algorithm



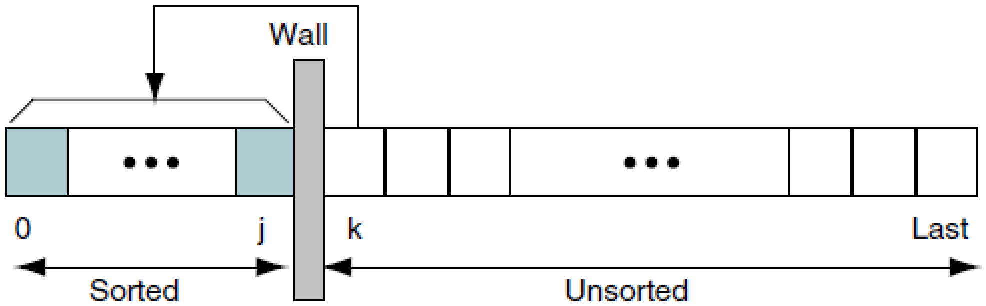
Decrease-by-1



Decrease-by-half

Insertion Sort

1. Given a list, it is divided into two parts: sorted and unsorted.
2. In each pass the first element of the unsorted sublist is transferred to the sorted sublist by inserting it at the appropriate place.
3. If list has n elements, it will take at most n – 1 passes to sort the data.



6 5 3 1 8 7 2 4

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 $i \leftarrow 1$ **to** $n-1$ **do**
    $v \leftarrow A[i]$
    $j \leftarrow i-1$
    while $j \geq 0$ **and** $A[j] > v$ **do**
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j-1$
    $A[j+1] \leftarrow v$

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

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

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

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

Graph Traversal

Many problems require processing all graph vertices (and edges) in systematic fashion

Graph traversal algorithms:

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

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.

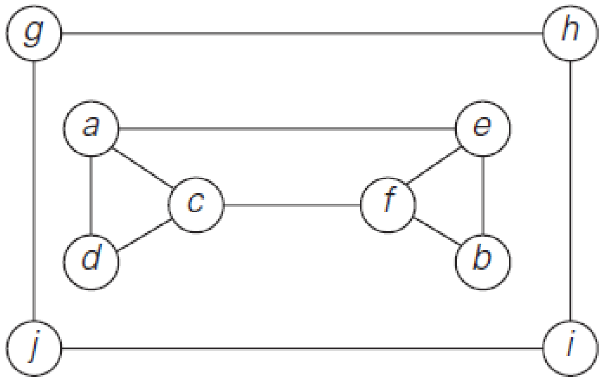
- Recursive or it **uses a stack**
  - A vertex is **pushed** onto the stack when it's **reached for the first time**.
  - A vertex is **popped** off the stack when it becomes a **dead end**, i.e., when there is no adjacent unvisited vertex.

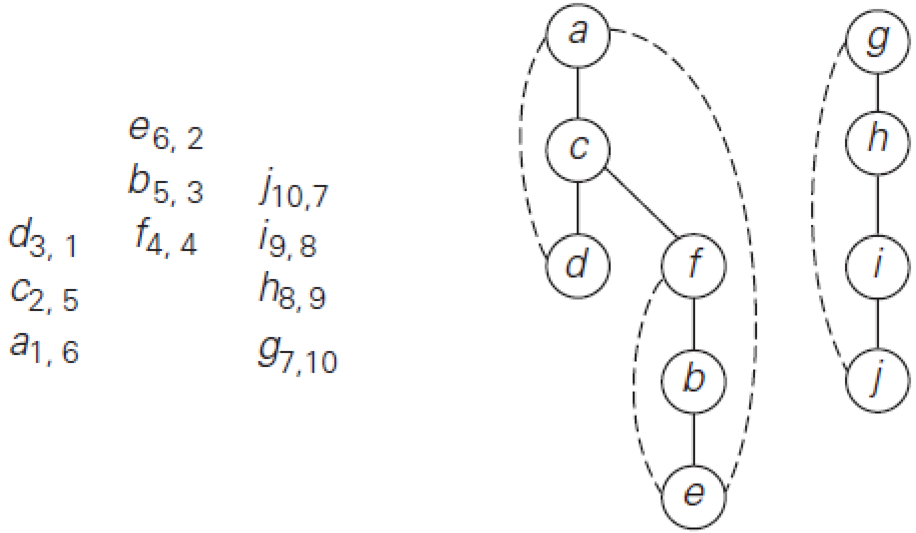
- "Redraws" graph in tree-like fashion (with tree edges and back edges for undirected graph)

Depth-First Search (DFS)

```
ALGORITHM  DFS(G)
    //Implements a depth-first search traversal of a given graph
    //Input: Graph G = ⟨V, E⟩
    //Output: Graph G with its vertices marked with consecutive integers
    //          in the order they are first encountered by the DFS traversal
    mark each vertex in V with 0 as a mark of being "unvisited"
    count ← 0
    for each vertex v in V do
        if v is marked with 0
            dfs(v)

dfs(v)
//visits recursively all the unvisited vertices connected to vertex v
//by a path and numbers them in the order they are encountered
//via global variable count
count ← count + 1;   mark v with count
for each vertex w in V adjacent to v do
    if w is marked with 0
        dfs(w)
```





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 (if no back edges)
  - Finding articulation points

Breadth-first search (BFS)

- Visits graph vertices by moving across to all the neighbors of the last visited vertex

- Instead of a stack, BFS uses a queue

- Similar to level-by-level tree traversal

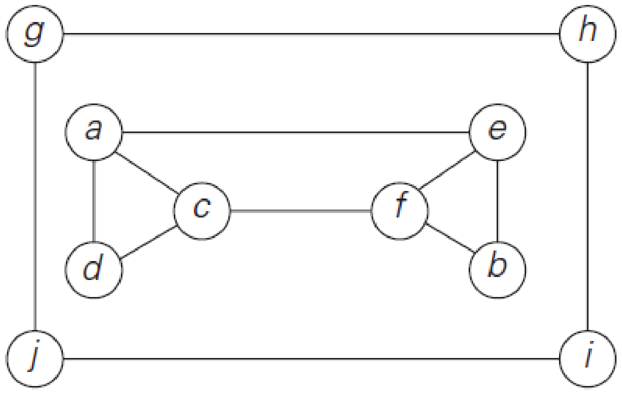
- “Redraws” graph in tree-like fashion (with tree edges and cross edges for undirected graph)

Breadth-first search (BFS)

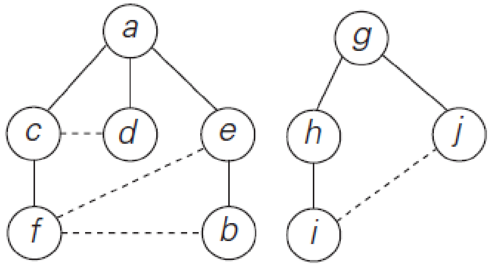
ALGORITHM *BFS(G)*

```
//Implements a breadth-first search traversal of a given graph
//Input: Graph G = ⟨V, E⟩
//Output: Graph G with its vertices marked with consecutive integers
//          in the order they are visited by the BFS traversal
mark each vertex in V with 0 as a mark of being "unvisited"
count ← 0
for each vertex v in V do
    if v is marked with 0
        bfs(v)

bfs(v)
//visits all the unvisited vertices connected to vertex v
//by a path and numbers them in the order they are visited
//via global variable count
count ← count + 1;   mark v with count and initialize a queue with v
while the queue is not empty do
    for each vertex w in V adjacent to the front vertex do
        if w is marked with 0
            count ← count + 1;   mark w with count
            add w to the queue
    remove the front vertex from the queue
```



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



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: same as DFS, but can also find paths from a vertex to all other vertices with the smallest number of edges

Thank you!

Any queries?