Design and Analysis of Algorithms

L16: Decrease and Conquer

Dr. Ram P Rustagi
Sem IV (2019-H1)
Dept of CSE, KSIT/KSSEM
rprustagi@ksit.edu.in

Resources

- Text book 1: Sec 5.1-5.3 - Levitin

Divide and Conquer

- Advantages
  - Solution becomes easier as problem is divided into smaller size
  - Efficient compared to brute force approach
    - Binary search
    - Large number multiplication
    - Matrix Multiplication
  - Smaller problems can be solved in parallel
    - Can improve algorithm running time
  - Can make effficient use of Caches
    - Small problem can be solved in cache itself

Divide and Conquer

- Dis-advantages
  - Makes of recursion heavily, thus computation may slow down a bit
  - Usage of stacks (by recursion) requires more memory
  - Implementation of recursion requires clarity of thought. At times, simple iteration is good enough
    - e.g. print all N-digit decimal numbers
  - Even a minuscle error in recursion termination condition may result in infinite loop (invocation)
    - Program will run out of memory (stack)
  - Can not solve a problem where recursion depth is more than system allows.
  - When subproblems may repeat (e.g. same sub matrix)
    - Then it may do duplication of computation.

Decrease and Conquer

- Reduce the problem instance to a smaller instance problem of the same type
- Solve the smaller instance problem
- Use the solution of smaller instance problem to solve the original bigger instance problem
- Implementation choices
  – Top down (use recursion) or bottom up
  – Incremental approach /inductive solution

Types of Decrease and Conquer

- ***A:***Decrease by a constance value `c` (`n → n-c`)
  - Usually decrease is by 1
  - Examples
    - Insertion sort
    - Graph traversal (DFS, BFS)
    - Topological sort
    - Generating permutations, subsets
- ***B***:Decrease by a constance factor `c` (`n → n/c`)
  - Usually decreases by half i.e. divide in equal half (`c=2`)
  - Examples:
    - Binary search
    - Exponentiation by squaring
    - Multiplication of numbers (a la russe algo)

A La Russe Multiplication Algo

- Write multiplicand and multiplier in two columns
- Repeat the following until left column has value 1
  – Divide value in left column by 2 (ignore fractions)
  – Multiply value in right column 2
  – Cross out the rows where left column value is even
- Sum all the values in right column (answer)

39	51
19	102
9	204
~~4~~	~~408~~
~~2~~	~~816~~
1	1632
	1989

Types of Decrease and Conquer

- *C:* Decrease by a variable size c_i ($n \rightarrow n - c_i$) at i^{th} step
  - The size decrease varies on each iteration
    - Depends upon input problem instance
  - Examples
    - Euclid's algorithm (greatest common divisor)
      - $gcd(m, n) \rightarrow gcd(n, m_{mod\ n})$
      - Selection by partition
      - Nim-like games (2 player)
        - » A pile of n discs
        - » Each player picks min 1, max m discs
        - » The person who picks last is winner.
        - » Soln: when $n=k(m+1)$, 1st person to pick loses

Differences with Divide and Conquer

- Divide and Conquer
  - Given problem instance divided into smaller instances
  - All smaller instances are solved (conquered)
  - Solutions of smaller instances are merged
  - Recursion : $T(n)=aT(n/b)+f(n)$
- Decrease and Conquer
  - Given problem instance reduced to single smaller instance.
  - Only one smaller instance problem is to be solved
  - Use smaller instancre problem to solve bigger instance problem
  - Recursion $T(n)=T(m)+f(n)$,where $m<n$

Differences with Other Approaches

- Problem instance: compute x^n
- Decrease and Conquer approach

  $T(n) = T(n-1)+1 = n-1$
- Brute force approach
  - Multiply x by itself $n-1$ times

  $T(n) = n-1$
- Divide and Conquer approach
  - Multiply $x^{n/2}$ by $x^{n/2}$

  $T(n) = 2T(n/2)+1 = n-1$
- Decrease by a constance factor
  - Multiply k times $x^{n/k}$ by itself

  $T(n) = T(n/k)+k-1 = n-1$

Review of DFS and BFS

- Graph
  - Set of nodes (vertices) connected by edges
  - Max number of edges are `n(n-1)/2`
  - Assumption: no multiple edges b /w any two nodes.
  - Some pair of nodes may not have any edge
- Directed Graph
  - When edges are directed
    - A→B is different than B→A
- Implementation
  - Adjancey (Linked) list
  - Adjacency Matrix
    - Symmetric for undirected graph
    - Asymmetric for directed graph

DFS

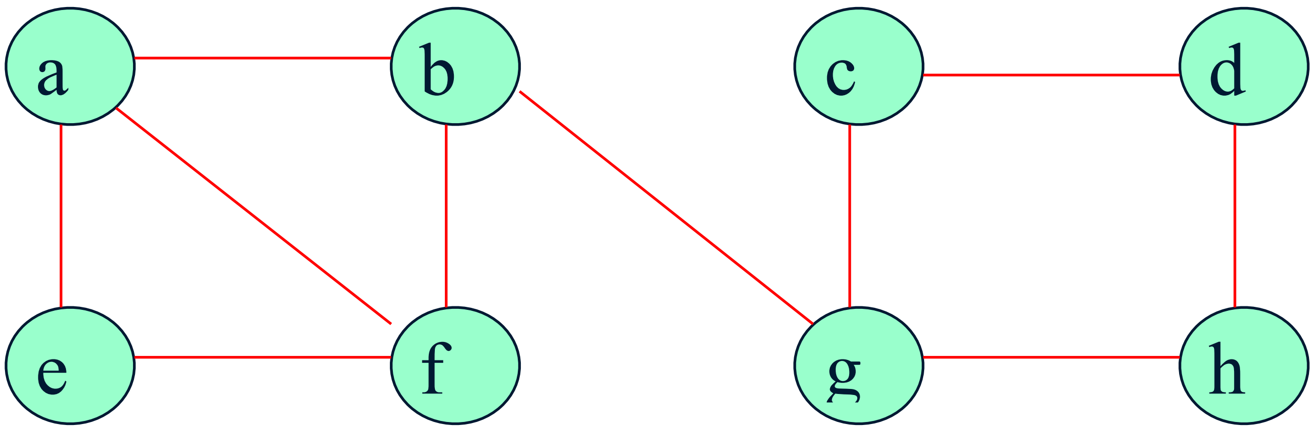
- DFS:
  - Start from a vertex (called root), mark it visited
  - Repeat the following
    - Find an unvisited vertex (not marked) connected by current node under consideration.
      - Mark this node as visited.
    - If there is no unvisited (unmarked) node connected to current node, backtrack.
- DFS Implementation
  - Using recursion
  - Using stack
  -

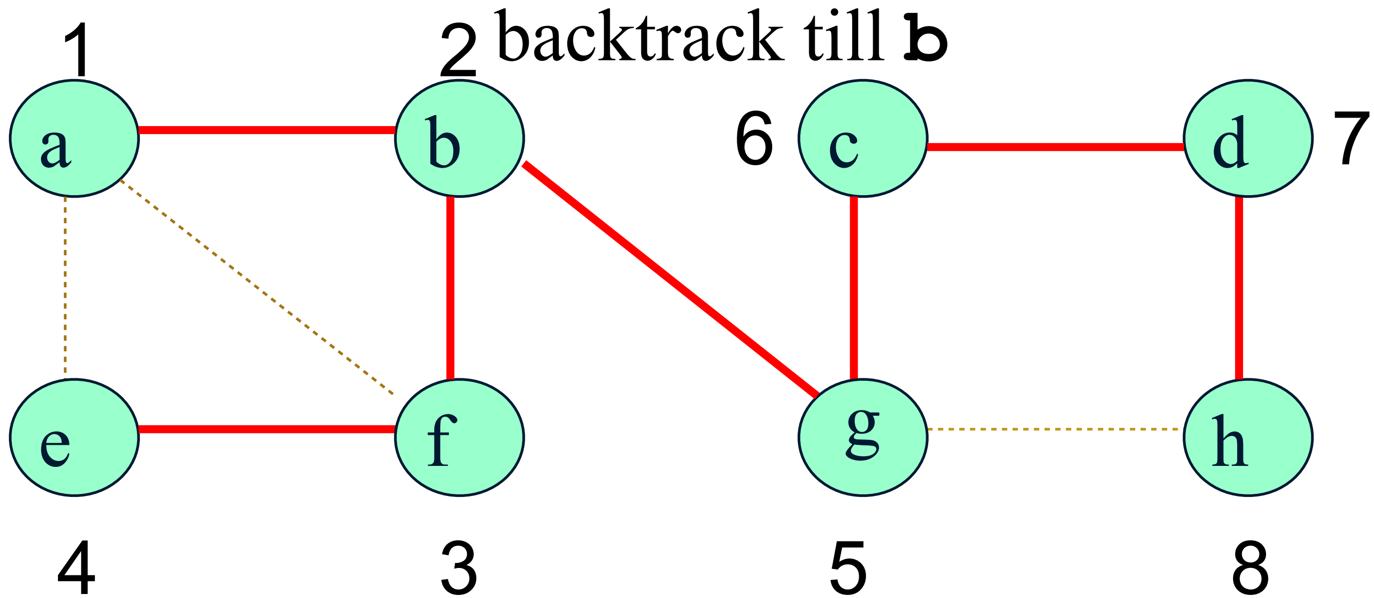
DFS Algo

```
# Input: G=(V, E)
# o/p: nodes V marked in the order these are visited.
# mark of 0 implies unvisited.
proc dfs(v)
   count ← count + 1
   mark(v) ← count
  for each vertex w ∈ V adjacent to v do
     if w is marked with 0, then
        dfs(w)
#end proc dfs(v)

for each vertex v ∈ V do
   mark(v) ← 0
count ← 0
for each vertex v ∈ V do
   if v is marked with 0, then
      dfs(v)
```

DFS Traversal





DFS Traversal: Time Complexity

- DFS implementation by Adjacency Matrix
  $\Theta(|V|^2)$
- DFS implementation by Adjacency Lists
  $\Theta(|V|+|E|)$
- Applications
  - Connected components
  - Checking for connected graph
  - Checking for acyclicity
  - Finding bi-connected components

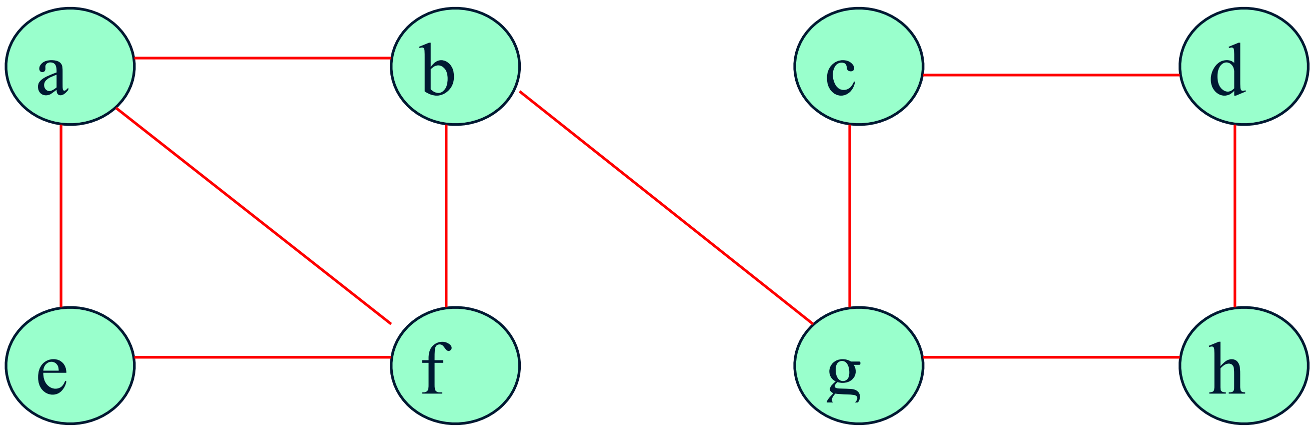
BFS Traversal

- Visits graph vertices by
  - visiting all neighbours of last visted node
- Instead of a stack based implementation
  - Uses queue based implementation

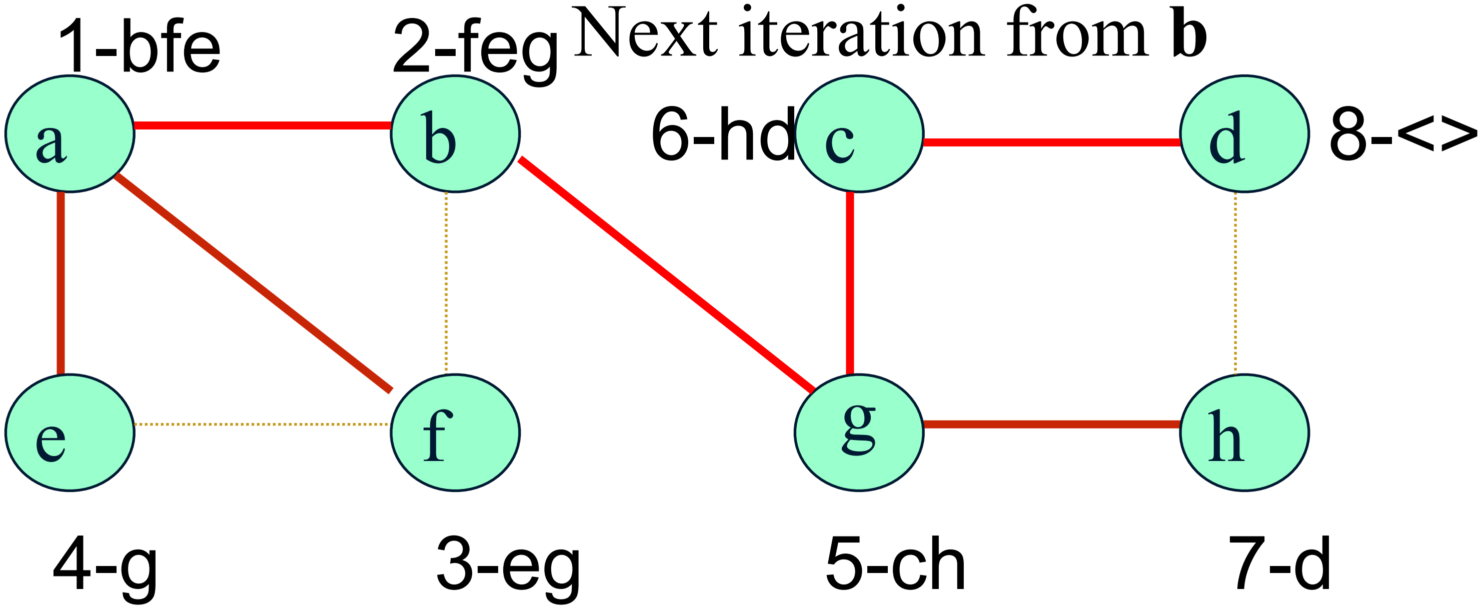
BFS Algo

```
proc BFS(v)
   count←count+1
   mark(v) ← count
   initialize queue with v.
   while queue is not empty do
      for each vertex w ∈ adjacency(v) do
         if w is marked with 0
            count←count+1
            add w to the queue
      remove front vertex (i.e. v) from queue
count←0
for each vertex v∈V do
   mark(v) ← 0
for each vertex v∈V do
   if mark(v) is 0
      BFS(v)
```

BFS Traversal



Next iteration from **b**



BFS Time Complexity

- Same efficiency as DFS
  - Adjacency matrices: $\Theta(|V|^2)$?
  - Adjacency lists: $\Theta(|V|+|E|)$?
- Vertices ordering
  - Single ordering of vertices
- Applications
  - Similar to DFS
  - Finding shortest path from a vertex to another becomes easier

Summary

- Advantages and disadvantages of Divide and Conquer
- Decrease and conquer approach
- DFS traversal
- BFS traversal