Decrease and Conquer

1. Reduce problem instance to smaller instance of the same problem and extend solution.
2. Solve smaller instance.
3. 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
  - 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

Graph Traversal

- Many problems require processing all graph vertices in systematic fashion
- Graph traversal algorithms:
  - Depth-first search
  - Breadth-first search

Depth-first search

- Explore graph always moving away from last visited vertex
- Similar to preorder tree traversals
- Pseudocode for Depth-first search of graph G=(V,E):
  ```pseudo
def DFS(G):
  count := 0
  mark each vertex with 0 (unvisited)
  for each vertex v:
    if v is marked with 0:
      dfs(v)
      count := count + 1
      mark v with count
      for each vertex w adjacent to v:
        if w is marked with 0:
          dfs(w)
  ```

Example – undirected graph

- Depth-first traversal:
## Types of edges

- **Tree edges**: edges comprising forest
- **Back edges**: edges to ancestor nodes
- **Forward edges**: edges to descendants (digraphs only)
- **Cross edges**: none of the above

## Example – directed graph

- Depth-first traversal:

## Depth-first search: Notes

- **DFS can be implemented with graphs represented as:**
  - Adjacency matrices: $O(V^2)$
  - Adjacency linked lists: $O(V + E)$
- **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)

## 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

## Breadth-first search algorithm

```
BFS(G)
    count := 0
    mark each vertex with 0
    for each vertex v ∈ V do
        bfs(v)
```

```python
bfs(v)
    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
```

## Example – undirected graph

- Breadth-first traversal:
Example – directed graph

Breadth-first: Notes

BFS has same efficiency as DFS and can be implemented with graphs represented as:
- Adjacency matrices: $O(V^2)$
- Adjacency linked lists: $O(V+E)$

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

Directed acyclic graph (dag)

A directed graph with no cycles

Arise in modeling many problems, eg:
- prerequisite structure
- food chains

Implies partial ordering on the domain

Topological sorting

Problem: find a total order consistent with a partial order

Example:
Order them so that they don’t have to wait for any of their food (i.e., from lower to higher, consistent with food chain)

NB: problem is solvable iff graph is dag

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, i.e., a vertex that has no incoming edges

Both $O(V+E)$ using adjacency linked lists

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

What about repeated keys?
Searching and insertion in binary search trees

- Searching - straightforward
- Insertion - search for key, insert at leaf where search terminated

All operations: worst case # key comparisons = $h + 1$

$l \log n \leq h \leq n - 1$ with average (random files) $1.44 \log n$

Thus all operations have:
- worst case: $\Theta(n)$
- average case: $\Theta(lg n)$

Bonus: inorder traversal produces sorted list (treesort)