CSC 8301 Design and Analysis of Algorithms

Lecture 7 Decrease and Conquer Algorithms

Reading: Chapter 5, Sections 5.1-3
Exercises:
- 5.1: 1, 4, 5
- 5.2: 1, 2, 3, 4, 6, 7, 10
- 5.3: 1, 3, 5, 6

Next time: Selection problem, Transform and Conquer
(5.5, 5.6, 6.1, 6.3)

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:
  - BFS
  - DFS
  - Topological order
- Algorithms for generating permutations, subsets

Decrease by a constant factor:
- Binary search
- Fake-coin problems
- Fast-multiplication A x B
- Josephus problem

Examples of iterative decrease:
- Quick sort
- Selection by partition
What's the difference?

Consider the problem of exponentiation: Compute $a^n$

- **Brute Force:**
- **Divide and conquer:**
- **Decrease by one:**
- **Decrease by constant factor:**

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):**

1. `DFS(G)`
2. `count := 0`
3. `mark each vertex with 0 (unvisited)`
4. `for each vertex v ∈ V do`
5.   `if v is marked with 0`
6.     `dfs(v)`
7.     `count := count + 1`
8.     `mark v with count`
9. `for each vertex w adjacent to v do`
10.   `if w is marked with 0`
11.      `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

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

2. Yields two distinct ordering of vertices:
   - preorder: as vertices are first encountered (pushed onto stack)
   - postorder: as vertices become dead-ends (popped off stack)

3. Applications:
   - Checking connectivity, finding connected components
   - Checking acyclicity
   - Searching state-space of problems for solution (AI)

Breadth-first search

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

5. Similar to level-by-level tree traversals

6. Instead of a stack, breadth-first uses queue

7. 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

```c
bfs(G)
    count := 0
    mark each vertex with 0
    for each vertex v ∈ V do
        if v is not marked
            bfs(v)
            count := count + 1
            mark v with count
            initialize queue with v
            while queue is not empty do
                a := front of queue
                count := count + 1
                mark a with count
                add a to the end of the queue
                remove a from the front of the queue
```
Example – undirected graph

\[
\begin{align*}
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{d} \\
\text{e} & \quad \text{f} & \quad \text{g} & \quad \text{h}
\end{align*}
\]

**Breadth-first traversal:**

Example – directed graph

\[
\begin{align*}
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{d} \\
\text{e} & \quad \text{f} & \quad \text{g} & \quad \text{h}
\end{align*}
\]

**Breadth-first traversal:**

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)
Directed acyclic graph (dag)

- A directed graph with no cycles
- Arise in modeling many problems, e.g:
  - prerequisite structure
  - food chains
- Imply partial ordering on the domain

Topological sorting

- Problem: find a total order consistent with a partial order
- Example:
  - fish
  - human
  - shrimp
  - sheep
  - plankton
  - wheat
  - tiger

Order them so that they don't have to wait for any of their food

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 $\Theta(V+E)$ using adjacency linked lists