FOUNDATIONS OF ALGORITHMS

1- Algorithms: Efficiency, Analysis,...

Time Complexity Analysis: In general, a time complexity analysis of an algorithm is the determination of how many times the basic operation is done for each value of the input size.

- Worst-Case Time Complexity Analysis
- Average-Case Time Complexity Analysis
- Best-Case Time Complexity Analysis

Order Definitions:

- **big O (asymptotic upper bound):** For a given complexity function $f(n)$, $O(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constant c and some nonnegative integer N such that for all $n \geq N$
  $g(n) \leq c * f(n)$.
- **Ω (an asymptotic lower bound):** For a given complexity function $f(n)$, $\Omega(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constant c and some nonnegative integer N such that, for all $n \geq N$
  $g(n) \geq c * f(n)$.
- **Θ:** For a given complexity function $f(n)$, $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$. This means that $\Theta(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constants c and d and some nonnegative integer N such that, for all $n \geq N$,
  $c * f(n) \leq g(n) \leq d * f(n)$.
- **small o:** For a given complexity function $f(n)$, $o(f(n))$ is the set of all complexity functions g(n) satisfying the following: For every positive real constant c there exists a nonnegative integer N such that, for all $n \geq N$, $g(n) \leq c * f(n)$.
- The most common complexity categories:
  $$O(\log n) < O(n) < O(n \log n) < O(n^i) < O(i^n)$$

2-Divide-and-Conquer

Binary Search:

- Binary Search locates a key x in a sorted array.
- The steps of Binary Search:
  - **Divide:** the array into two subarrays.
  - **Conquer:** (solve) the subarray by determining whether x is in that subarray.
  - **Obtain:** the solution to the array from the solution to the subarray.
- Binary Search does not have an every-case time complexity.
- Worst-Case Time Complexity
  $$W(n) = \lfloor \log n \rfloor + 1 \in \Theta(\log n)$$

Mergesort:

- By repeatedly combining two sorted arrays into one
- The steps of Mergesort:
  - **Divide:** the array into two subarrays.
  - **Conquer:** (solve) each subarray by sorting it.
  - **Combine:** the solutions to the subarrays by merging them into a single sorted array.
- Worst-Case Time Complexity
  $$W(n) \in \Theta(n \log n)$$
- Mergesort is not an in-place sort (An **in-place sort** is a sorting algorithm that does not use any extra space beyond that needed to store the input).
- Mergesort-2 is the in-place version of Mergesort.

Quicksort

- Quicksort is similar to Mergesort
- The array is partitioned by placing all items smaller than some pivot item before that item and all items larger than or equal to the pivot item after it.
- Quicksort does not have an every-case complexity.
- Worst-Case Time Complexity
  $$W(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$$
- Average-Case Time Complexity
  $$A(n) \approx (n+1)2\text{In } n = (n+1)2(\ln 2)(\lg n) \in \Theta(n \lg n)$$

Strassen's Matrix Multiplication

- Complexity of standard matrix multiplication:
  Multiplications: $T(n) = n^3$ Additions: $T(n) = n^3 - n^2$
- Every-Case Time Complexity:
  Multiplications: $T(n) = 7T(\frac{n}{2}) = n^{\lg 7} \in \Theta(n^{2.81})$
  Additions: $T(n) = 7T(\frac{n}{2}) + 18T(\frac{n}{2})^2 = 6n^{\lg 7} - 6n^2 \in \Theta(n^{2.81})$

Arithmetic With Large Integers

- Split an n-digit integer into two integers of approximately $n/2$ digits.
  $$\underbrace{u}_{n \text{ digits}} = \underbrace{x}_{\lceil n/2 \rceil \text{digits}} \times 10^m + \underbrace{y}_{\lfloor n/2 \rfloor \text{digits}} \quad ; \quad m = \lfloor n/2 \rfloor$$
- Worst-Case Time Complexity
  (1): $W(n) = 4W(\frac{n}{2}) + cn \in \Theta(n^{\lg 4}) = \Theta(n^2)$
  (2): $W(n) \in \Theta(N^{\log_2^3}) \approx \Theta(N^{1.58})$

Determining Thresholds

- Determines for what values of n it is at least as fast to call an alternative algorithm as it is to divide the instance further.
- To determine a threshold, we must consider the computer on which the algorithm is implemented.

3-Dynamic Programming

The steps of Dynamic Programming

- *Establish* a recursive property that gives the solution to an instance of the problem.
- Solve an instance of the problem in a *bottom-up* fashion by solving smaller instances first.

Binomial Coefficient

- Equation:
  $$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad \text{for } 0 \leq k \leq n.$$
- Recursive binomial coefficient:
  $$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0 \text{ or } k = n \end{cases}$$
- The total number of passes $\in \Theta(nk)$

Floyd's Algorithm for Shortest Paths: Finding the shortest paths from each vertex to all other vertices in a weighted digraph

- Create adjacency matrix representation of the graph (W)
- Set $D^{(0)} = W$ and compute $D^{(k)}$ from $D^{(k-1)}$.
- Select All shortest paths from v_i to v_j using only vertices in $[v_1, v_2, \ldots, v_k]$ as intermediate vertices
- Every-Case Time Complexity: $T(n) \in \Theta(n^3)$

Chained Matrix Multiplication

- Optimal order to multiply n matrices
- Every-Case Time Complexity: $T(n) \in \Theta(n^3)$

Optimal Binary Search Trees

- A binary search tree
  - Each node contains one key.
  - The keys in the left subtree of a given node are less than or equal to the key in that node.
  - The keys in the right subtree of a given node are greater than or equal to the key in that node.
- Determine an optimal binary search tree for a set of keys, each with a given probability of being the search key.
- Every-Case Time Complexity: $T(n) \in \Theta(n^3)$

The Traveling Salesperson Problem

- A tour (also called a **Hamiltonian Circuit**) in a directed graph is a path from a vertex to itself that passes through each of the other vertices exactly once.
- An optimal tour in a weighted, directed graph is such a path of minimum length.
- The Traveling Salesperson Problem is to find an optimal tour in a weighted, directed graph when at least one tour exists.
- Every-Case Time and Space Complexity:
  $$T(n) \in \Theta(n^2 2^n), \quad M(n) \in \Theta(n 2^n)$$

4-The Greedy Approach

A greedy algorithm arrives at a solution by making a sequence of choices, each of which simply looks the best at the moment.

Minimum Spanning Trees

- A spanning tree for G is a connected subgraph that contains all the vertices in G and is a tree.
- Prim's Algorithm $T(n) \in \Theta(n^2)$
  - Select an arbitrary vertex.
  - Add nearest vertices
  - ...
- Kruskal's Algorithm $W(m,n) \in \Theta(m \lg m)$and $\in \Theta(n^2 \lg n)$
  where $(n-1) \leq m \leq \frac{n(n-1)}{2}$
  - Sort the edges in E in nondecreasing order
  - ....
  - For a graph whose number of edges m is near the low end of these limits (the graph is very sparse) Kruskal's Algorithm is $\Theta(n \lg n)$
  - For a graph whose number of edges is near the high end (the graph is highly connected), Kruskal's Algorithm is $\Theta(n^2 \lg n)$, which means that Prim's Algorithm should be faster.

Dijkstra's Algorithm

- Determine the shortest paths from Uj to all other vertices in a weighted, directed graph.
- $T(n) = 2(n-1)^2 \in \Theta(n^2)$

Scheduling

- *Minimizing Total Time in the System:* Schedule the customers in such a way as to minimize the total time they spend both waiting and being served (getting treated).
- sort the jobs by service time in nondecreasing order
- ...
- Worst-Case Time Complexity: $W(n) \in \Theta(n \lg n)$
- sort the jobs in nonincreasing order by profit
- ...
- *Scheduling with Deadlines:* Determine the schedule with maximum total profit given that each job has a profit that will be obtained only if the job is scheduled by its deadline.
- Worst-Case Time Complexity : $W(n) \in \Theta(n^2)$

Greedy vs Dynamic Programming ...

5-Backtracking

- Backtracking is a modified depth-first search of a tree.
- We call a node **nonpromising** if when visiting the node we determine that it cannot possibly lead to a solution. Otherwise, we call it **promising**.
- Backtracking consists of doing a depth-first search of a state space tree, checking whether each node is promising, and, if it is nonpromising, backtracking to the node's parent.

The n-Queens Problem : Position n queens on a chessboard so that no two are in the same row, column, or diagonal.

Monte Carlo Estimate: Estimate the efficiency of a backtracking algorithm using a Monte Carlo algorithm.

- In each level let m_i be the number of promising children of the level. Then randomly generate a promising child of the node obtained in the level and go to the nest level.
- This process continues until no promising children are found.

The Sum-of-Subsets Problem: Given n positive integers (weights) and a positive integer W, determine all combinations of the integers that sum to W.

Graph Coloring: finding all ways to color an undirected graph using at most m different colors, so that no two adjacent vertices are the same color.

The Hamiltonian Circuits Problem: Determine all Hamiltonian Circuits in a connected, undirected graph.

The 0-1 Knapsack Problem: Let n items be given, where each item has a weight and a profit. The weights and profits are positive integers. Furthermore, let a positive integer W be given. Determine a set of items with maximum total profit, under the constraint that the sum of their weights cannot exceed W.

References:

[1] Foundations of Algorithms, by Richard E. Neapolitan and Kumarss Naimipour

Made by ma.mehralian using LaTeX