Coding Bootcamp

(Session 2)

Udayan Joshi, Karan Taneja

WHAT IS DSA!?

- An algorithm is an unambiguous specification of how to solve a class of problems
- Involves calculation, data processing and automated reasoning tasks
- A data structure is a data organization and storage format that enables efficient access and modification

WHAT IS COMPUTATIONAL COMPLEXITY

- Time Complexity: Running time of the program as a function of the size of input
- Space Complexity: Amount of computer memory required during the program execution, as a function of the input size
- There are other kinds of resources such as power, number of random bits used etc.

More on Complexity Theory

- Complexity Theory is the study of resources necessary and sufficient for performing different computational tasks
- The goal is thus to determine the intrinsic complexity of any well defined computational task.
- The runtime behaviour of an algorithm, also called its growth rate, can be measured as a function T of its input size n. The function T(n) is equal to the maximum number of basic operations that an algorithm performs on inputs of length n.

- At times this function T(n) is highly dependent on the low-level details.
- Asymptotic complexity provides an abstraction to these low-level details of the underlying machine architecture and involved constants.
- However, it is difficult to determine the complexity for individual tasks. Complexity theorists thus try to determine the relation between different computational tasks.
- a **complexity class** is a set of problems of related resource-based **complexity(**We will not be studying this**).**

Big oh notation

- A convenient way of describing the growth rate of a function and hence the time complexity of an algorithm.
- Gives asymptotic upper bound of the functions

DEFINITION : T(n) is O(f(n)) if the limit of T(n)/f(n), is a constant as n goes to infinity.

O(n) notation thus focuses on the largest term and ignores constants.

- Let $f(n) = n^2 + n + 5$. Then
  - $f(n)$ is $O(n^2)$
  - $f(n)$ is $O(n^3)$
  - $f(n)$ is not $O(n)$
- Let $f(n) = 3^n$
  - $f(n)$ is $O(4^n)$
  - $f(n)$ is not $O(2^n)$
- If $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$, then
  - $f_1(n) + f_2(n)$ is $O(\max(g_1(n), g_2(n)))$

COMMON USAGE

- O(g(n)) is a class of functions
- O(1) is commonly referred to as constant time, O(n) indicates linear time; $O(n^k)$ (k fixed and finite) refers to polynomial time; O(logn) is called logarithmic time; $O(2^n)$ is referred to as exponential time

Growth Rate	Name	Notes
O(1)	constant	Best, independent of input size
O(log log n)		very fast
O(log n)	logarithmic	often for tree-based data structures
O($\log^k n$)	polylogarithmic	
O(n^p), $0 < p < 1$	E.g. O(n1 / 2) = O($\sqrt{n}$)	Still sub-linear
O(n)	linear	Have to look at all data
O(n log n)		Time to sort
O(n^2)	quadratic	Ok if n is small enough;
O(n^k)	polynomial	Tractable
O(2^n), O(n!)	exponential, factorial	bad

Practical Complexities

10^9 instructions/second

n	n	$n\log n$	n^2	n^3
1000	1mic	10mic	1milli	1sec
10000	10mic	130mic	100milli	17min
10^6	1milli	20milli	17min	32years

Other Asymptotic Notations

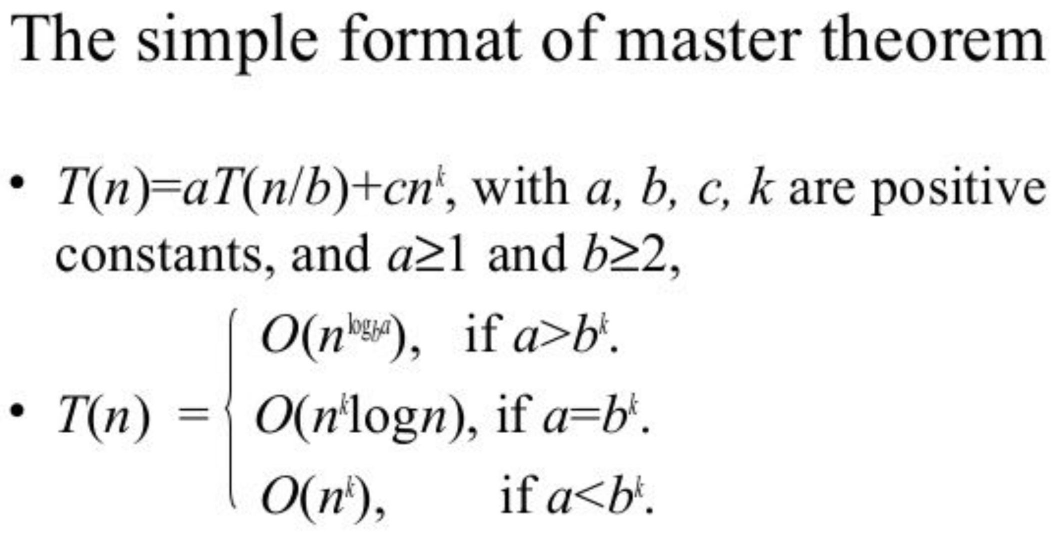
- Small-oh notation: strict asymptotic upper bound
- Big-Omega notation: asymptotic lower bound
- small-Omega notation: strict asymptotic lower bound
- Big-Theta notation: combined upper and lower bound

Is small-Theta notation possible?

FIBONACCI PROBLEM

- Leonard Fibonacci, the mathematician studied a pair of fictional and slightly unbelievable baby rabbits, a baby boy rabbit and a baby girl rabbit

Master's theorem



The simple format of master theorem

- $T(n)=aT(n/b)+cn^k$, with a, b, c, k are positive constants, and $a \geq 1$ and $b \geq 2$,
- $$T(n) = \begin{cases} O(n^{\log_b a}), & \text{if } a>b^k. \\ O(n^k \log n), & \text{if } a=b^k. \\ O(n^k), & \text{if } a<b^k. \end{cases}$$

Dynamic Programming

Idea of Dynamic Programming: Dynamic Programming refers to simplifying a complicated problem by *breaking it down into simpler sub-problems in a recursive manner*. The two conditions that the problem must satisfy in order to apply dynamic programming are as follows:

Optimal Substructure Property: A given problems has Optimal Substructure Property if *optimal solution* of the given problem can be obtained by using optimal solutions of its subproblems.

Overlapping Subproblems Property: Simply put, solutions of same subproblems are *needed again and again*.

Dynamic Programming

Given an array of integers, find the subset of non-adjacent elements with the maximum sum. Calculate the sum of that subset.

Example:

```
arr = [-2, 1, 3, -4, 5]

[3, 5] have maximum sum of 8.  Answer is 8.
```

Dynamic Programming

Given an array of integers, find the subset of non-adjacent elements with the maximum sum. Calculate the sum of that subset.

Solution:

```
dp[0] = max(0, arr[0]) , dp[1] = max(dp[0], arr[1])

dp[i] = max(dp[i-1], dp[i-2], dp[i-2]+arr[i])
```

Greedy Algorithms

Idea of Greedy Algorithms

Naively, An algorithm is called greedy if it makes locally optimal choices in hopes of finding a global optimum. Clearly not all greedy algorithms give the correct solution. If you are using a greedy algorithm, then you also need to prove its correctness.

Proofs by Exchange Argument

Assume that there exists another better solution. Show that it is not possible to have such a solution by contradiction.

Greedy Algorithms

A group of friends want to buy a bouquet of flowers. The florist wants to maximize his number of new customers and the money he makes. To do this, he decides he'll multiply the price of each flower by the number of that customer's previously purchased flowers plus 1. Given the size of the group of friends, the number of flowers they want to purchase and the original prices of the flowers, determine the minimum cost to purchase all of the flowers.

Example:

```
2 friends want to buy 3 types of flowers. Prices are 2, 5 and 6.

First friend: (0+1) x 5  +  (1+1) x 2, Second friend: (0+1) x 6. Answer is 15.
```

Greedy Algorithms

A group of friends want to buy a bouquet of flowers. The florist wants to maximize his number of new customers and the money he makes. To do this, he decides he'll multiply the price of each flower by the number of that customer's previously purchased flowers plus 1. Given the size of the group of friends, the number of flowers they want to purchase and the original prices of the flowers, determine the minimum cost to purchase all of the flowers.

Solution:

```
Sort the prices from largest to smallest. Divide the first ‘k’ (k = number of
friends) most expensive flowers among them, i.e. one to each. Divide next k
among them and so on.
```

Graphs

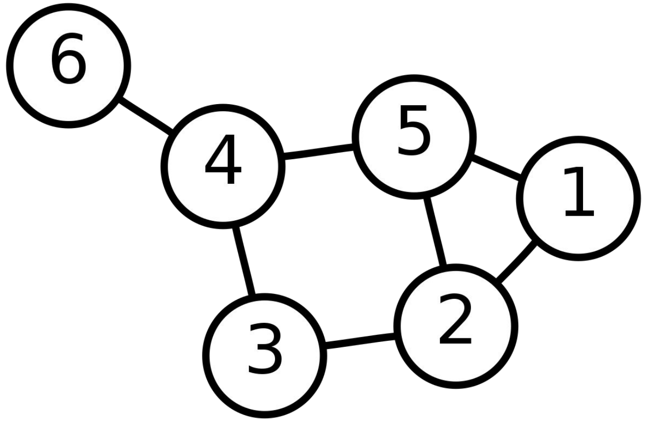
Edges

Nodes/Vertices

Edge Weights

Directed versus Undirected Graph

Adjacency List and Adjacency Matrix



Graphs

Connected Graph

Complete Graph

Bipartite Graph

Cyclic Graph

Trees

Graphs

BFS

DFS

Graphs - Some Homework!

Prim’s Minimum Spanning Tree (MST) Algorithm

Kruskal’s MST Algorithm

Dijkstra’s Shortest Path Algorithm

Bellman Ford Shortest Path Algorithm

Graphs

Example Problem: Link

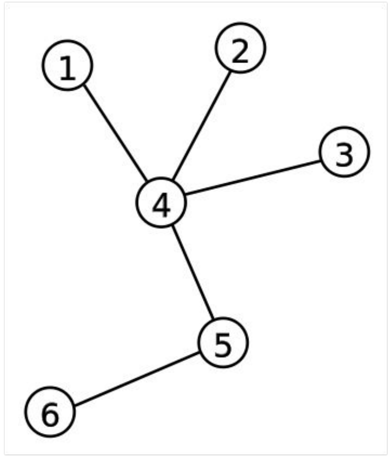
Let’s solve it!

Trees

Forest

Connected Graphs:

(number of edges = number of nodes - 1)



Binary Tree

Binary tree

Example of Huffman Code

Resources

HackerRank: https://www.hackerrank.com/

See resources from previous session for more information.