COMP3821

Extension Algorithms and Programming Techniques

Charlie Bradford

March 28, 2018

1 Introduction

2 Algorithm Analysis

2.1 Asymptotic Behaviour

$f(n) = O(g(n))$ $\exists n_0, c \in \mathbb{R}(\forall n > n_0(0 \leq f(n) \leq cg(n)))$ i.e. g(n) is the worst case.

$f(n) = \Omega(g(n))$ $\exists n_0, c \in \mathbb{R}(\forall n > n_0(0 \leq cg(n) \leq f(n)))$ i.e. g(n) is the best case.

$f(n) = \Theta(g(n))$ $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$ i.e. f(n) has the same best and worst case growth.

We also have the notation $f(n) = o(g(n))$. This is used when $g(n)$ is not asymptotically tight. For example $n \neq O(n^2)$ but $n = o(n^2)$.

2.2 The Master Theorem

Let:
$a, b \in \mathbb{Z}, a \geq 1 \wedge b > 1$
$f(n) \geq 0 \wedge f'(n) \geq 0$
$T(n) = aT(\frac{n}{b}) + f(n)$
Then:
$\exists \epsilon \geq 0(f(n) = O(n^{log_b a - \epsilon})) \Rightarrow T(n) = \Theta(n^{log_b a})$
$\exists \epsilon \geq 0(f(n) = \Theta(n^{log_b a})) \Rightarrow T(n) = \Theta(n^{log_b a} log_2 n)$
$\exists \epsilon \geq 0(f(n) = \Omega(n^{log_b a + \epsilon})) \wedge \exists c < 1(af(\frac{n}{b}) \leq cf(n)) \Rightarrow T(n) = \Theta(f(n))$

2.3 Fast Integer Multiplication

If we are multiplying two n-digit numbers, A and B, we first split them into halves

$$A = A_1 10^{\frac{n}{2}} + A_0$$

$$B = B_1 10^{\frac{n}{2}} + B_0$$

Then we can calculate AB:

$AB = A_1 B_1 10^n + (A_0 B_1 + A_1 B_0) 10^{\frac{n}{2}} + A_0 B_0$

Here we have halved the size of each multiplication (reducing complexity by fourfold)

but quadrupling the number of multiplications

$= A_1 B_1 10^n + ((A_1 + A_0)(B_1 + B_0) - A_1 B_1 - A_0 B_0) 10^{\frac{n}{2}} + A_0 B_0$

Now we have reduced the number of multiplications, leading to a lower complexity than $O(n^2)$

Now we have $T(n) = 3T(\frac{n}{2}) + cn$. $f(n) = cn$ as addition is linear. $log_2 3 \sim 1.6$ so $cn = O(n^{log_2 3})$. Therefore the first case of the master theorem applies and $T(n) = \Theta(n^{log_2 3})$.

Theoretically we could keep making the algorithm faster by splitting the number into more bits, but the constant factors start to get so large that the performance is to slow with reasonable n.

2.4 Problems

Put examples.

3 Divide-And-Conquer Method

3.1 Weighing Coins

Problem: we have nine coins and one is lighter than the others. How do we find the lighter by using a scale only twice times. Solution: Weigh three of the coins against three more. Select the lighter three, if they are the same then take the coins that were not eighed. Weigh one coin agaist the other, now you have the lighter coin.

3.2 Multiplying Polynomials

A polynomial of degree n is uniquely determined by the coefficients A_i. As in $A_0 + A_1x + A_2x^2 + ... + A_nx^n$. Thus we can determine the values of the coefficents using only n+1 different valuex of x.

$$P_A(x) \leftrightarrow \{(x_0, P_A(x_0)), (x_1, P_A(x_1), ..., (x_n, P_A(x_n))\}$$

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{2n} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & x_{2n+1} & x_{2n+1}^2 & \dots & x_{2n+1}^{2n} \\ 1 & x_{2n+1}^2 & x_{2n+1}^{2*2} & \dots & x_{2n+1}^{2*2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{2n+1}^{2n} & x_{2n+1}^{2n*2} & \dots & x_{2n+1}^{2n*2n} \end{bmatrix} = \begin{bmatrix} P_a(1) \\ P_a(x_{2n}) \\ P_a(x_{2n}^2) \\ \vdots \\ P_a(x_{2n}^{2n-1}) \end{bmatrix}$$

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{2n} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & x_{2n+1} & x_{2n+1}^2 & \dots & x_{2n+1}^{2n} \\ 1 & x_{2n+1}^2 & x_{2n+1}^{2*2} & \dots & x_{2n+1}^{2*2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{2n+1}^{2n} & x_{2n+1}^{2n*2} & \dots & x_{2n+1}^{2n*2n} \end{bmatrix}^{-1} \begin{bmatrix} P_a(1) \\ P_a(x_{2n}) \\ P_a(x_{2n}^2) \\ \vdots \\ P_a(x_{2n}^{2n-1}) \end{bmatrix}$$

3.3 Fast Fourier Transform

3.4 Discrete Fourier Transform

4 The Greedy Method

A greedy algorithm creates a solution through a series of steps, choosing each movement at each step without considering the whole problem.

4.1 Activity Selection

4.2 Djikstra's Algorithm

Djikstra's algorithm finds the shortest distance between two nodes in a graph. Starting from the origin node s in a graph $G(V, E)$, algorithm maintains a set S of vertices u such that the minimum distance between s and u, $d(u)$, is known. Then for each node $v \in V - S$, we find the shortest path through S to u, where there is a single edge connecting v and u. Then, for all such nodes, we consider $d'(v) = \forall u \in S.e = (u, v), min(d(u) + l_e)$. Then once we have the pair (u, v) for which $d'(v)$ is minimal, we add v to S.

4.3 Machining Problem

Items have to be machined and then polished. One machine does the machining, and a second does the polishing. N items I, for each item I_k you know the machining time M_k and the polishing time P_k. How do you schedule the machining and polishing so that so that the entire process takes as little time as possible?

Answer: In increasing order of P_k.

Explanation: The machining machine can be run constantly so $\sum_{k=1}^{n} M_k$ is constant. So by scheduling items with the lowest P_k first we get the over as quickly as possible to free time for later items.

4.4 Discrete Knapsack Problem

4.5 Huffman Codes

4.6 Set Cover Approximation

5 Dynamic Programming

6 Network Flow Algorithms

7 Linear Programming

8 Intractable Problems and Approximation Algorithms

9 Randomisation

9.1 Random Select

9.2 Linear Time for Order

9.3 Hash Functions

9.4 Skip Lists

- Like a doubly linked list but certain nodes have different heights (up to max $log_2 n$ for n-element list)
- Searching for k:
  1. Start at head
  2. Move to the node pointed to at the highest level
  3. If the value is smaller than k go to 2.
  4. If the value is equal to k return
  5. Move the the node pointed to at the next highest level, go to 3.
  6. Expected time $O(log_2 n)$
- Insertion node with value k:
  - Find the correct location for k
  - The height, i, for k's node is $\frac{1}{2^i}$
  - The node is linked from the bottom up
  - Expected time if $O(log_2 n)$ for searching and $O(1)$ for inserting
- Deletion:
  - As in a doubly linked list
  - Sort out all pointers from the bottom up