Algorithm Design and Analysis (ECS 122A) Study Guide

Davis Computer Science Club
Tutoring Committee

Spring Quarter 2015

Contents

Chapter 1

Asymptotic Notation

1.1 O-Notation (Big O)

Notation

$$f(n) \in O(g(n))$$

Formal Definition

For a given function $g(n)$, $O(g(n))$ is the set of functions for which there exists positive constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

$$O(g(n)) = \{f(n) : \exists\ c, n_0 \text{ s.t. } 0 \leq f(n) \leq c \cdot g(n)\ \forall\ n \geq n_0\}$$

Informal Definition

The function $g(n)$ is an asymptotic upper bound for the function $f(n)$ if there exists constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for $n \geq n_0$.

Another way to perceive Big O notation is that for $f(n) \in O(g(n))$, the function f's asymptotic[1] growth is no faster than that of function g's.

Limit Definition

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

1.1.1 Example

Prove that asymptotic upper bound of $f(n) = 2n + 10$ is $g(n) = n^2$.

$$\begin{aligned} 0 \leq f(n) &\leq c \cdot g(n) \text{ for } n \geq n_0 \\ 0 \leq 2n + 10 &\leq c \cdot n^2 \text{ for } n \geq n_0 \end{aligned}$$

Arbitrarily choose c and n_0 values. Simplest is to turn one of the variables into the value 1 and solve. For this example, we will assign the value 1 to n_0.

$$\begin{aligned} 0 \leq 2n + 10 &\leq c \cdot n^2 \text{ for } n \geq 1 \\ 2(1) + 10 &\leq c \cdot (1)^2 \\ 12 &\leq c \end{aligned}$$

By picking $n_0 = 1$ and $c = 12$, the inequality of $2n + 10 \leq 12n^2$ will hold true for all $n \geq 1$. Since there exists a constant c and n_0 that fulfill this inequality, we have proven that $f(n) = 2n + 10 = O(n^2)$.

[1] Asymptotic: As given variable approaches infinity.

1.2 o-Notation (Little O)

Notation

$$f(n) \in o(g(n))$$

Formal Definition

For a given function $g(n)$, $o(g(n))$ is the set of functions for which every positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

$$o(g(n)) = \{f(n) : \exists\ n_0 \text{ s.t. } 0 \leq f(n) \leq c \cdot g(n)\ \forall\ n \geq n_0, c \geq 0\}$$

Informal Definition

The function g(n) is an upper bound that is not asymptotically tight. For all positive constant values of c, there must exists a constant n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. The value of n_0 may not depend on n, but may depend on c.

Another way to perceive Little O notation is that for $f(n) \in o(g(n))$, the function f's asymptotic growth is strictly less than that of the function g's. In this sense, Little O can be seen as a "stronger" bound in comparison to Big O. By proving that a function is an element of Little O, it also proves that the function is an element of Big O.

Limit Definition

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

1.2.1 Example

Prove that $f(n) = 2n$ has an upper bound $o(n^2)$.

$$\begin{aligned} 0 \leq c \cdot g(n) &\leq f(n) \text{ for } n \geq n_0 \\ 0 \leq c \cdot 2n &\leq n^2 \text{ for } n \geq n_0 \\ 2c &\leq n \text{ for } n \geq n_0 \\ 2c &\leq n_0 \end{aligned}$$

For Little O to hold true, the inequality needs to hold true for all $c > 0$ and for all $n > n_0$. From simplifying the inequality, we assert that the inequality will hold true as long as the value of n_0 is twice the value of c. Given that they are both constants, then there exists a constant value of n_0 for all positive constant c that fulfill this inequality.

Another method to solve this problem is to use the limit definition.

$$\begin{aligned} &\lim_{n\to\infty} \frac{2n}{n^2} \\ &\lim_{n\to\infty} \frac{2}{n} = 0 \end{aligned}$$

1.2.2 Example

Prove that $f(n) = 2n^2$ does not have the upper bound $o(n^2)$.

$$\begin{aligned}
0 \leq c \cdot g(n) &\leq f(n) \text{ for } n \geq n_0 \\
0 \leq c \cdot 2n^2 &\leq n^2 \text{ for } n \geq n_0 \\
2c &\leq 1 \text{ for } n \geq n_0
\end{aligned}$$

For a function to have the Little O bound, the inequality must hold true for all positive c. However, simplification of the inequality asserts that the inequality will only hold true for all $c < \frac{1}{2}$. Therefore, $f(n) = 2n^2$ does not have the upper bound $o(n^2)$.

1.3 Ω-Notation (Big Omega)

Notation

$$f(n) \in \Omega(g(n))$$

Formal Definition

For a given function $g(n)$, $\Omega(g(n))$ is the set of functions for which there exists positive constants c and n_0 such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0$.

$$\Omega(g(n)) = \{f(n) : \exists\ c, n_0 \text{ s.t. } 0 \leq c \cdot g(n) \leq f(n)\ \forall\ n \geq n_0\}$$

Informal Definition

The function $g(n)$ is an asymptotic lower bound for the function $f(n)$ if there exists constants c and n_0 such that $0 \leq c \cdot g(n) \leq f(n)$ for $n \geq n_0$.

Limit Definition

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0$$

1.4 ω-Notation (Little Omega)

Notation

$$f(n) \in \omega(g(n))$$

Formal Definition

For a given function $g(n)$, $\omega(g(n))$ is the set of functions for which every positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0$.

$$\omega(g(n)) = \{f(n) : \exists\ n_0 \text{ s.t. } 0 \leq c \cdot g(n) \leq f(n)\ \forall\ n \geq n_0, c \geq 0\}$$

Informal Definition

The function g(n) is a lower bound that is not asymptotically tight. For all positive constant values of c, there must exists a constant n_0 such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0$. The value of n_0 may not depend on n, but may depend on c.

Another way to perceive Little ω notation is that for $f(n) \in \omega(g(n))$, the function f's asymptotic growth is strictly greater than that of the function g's. In this sense, Little ω can be seen as a "stronger" bound in comparison to Big Ω. By proving that a function is an element of Little ω, it also proves that the function is an element of Big Ω.

Limit Definition

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

1.5 Θ-notation (Big Theta)

Notation

$$f(n) \in \Theta(g(n))$$

Formal Definition

For a given function $g(n)$, $\Theta(g(n))$ is the set of functions for which there exists positive constants c_1, c_2, and n_0 such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$.

$$\Theta(g(n)) = \{f(n) : \exists\ c_1, c_2, n_0 \text{ s.t. } 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\ \forall\ n \geq n_0\}$$

Informal Definition

The function g(n) is an asymptotic tight bound for the function f(n) if there exists constants c_1, c_2, and n_0 such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for $n \geq n_0$.

Big theta implies that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Limit Definition

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \in \mathbb{R}_{>0}$$

Chapter 2

Recurrence Relations

2.1 Recurrence Relations

A recurrence relation is an equation that recursively defines a sequence of values. After the initial terms are given, each subsequent term is defined as a function of the previous terms.

Fibonacci

Fibonacci is an example of a recurrence relation.

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & n \geq 2 \\ 1, & n = 1 \\ 0, & n = 0 \end{cases}$$

The first two terms are defined while the subsequent terms are a function of the two previous.

2.2 Solving Recurrence Relations

- Substitution Method
- Recursion-Tree Method
- Master Theorem

2.3 Substitution Method

1. Guess the bounds.
2. Apply mathematical induction to prove the bounds.

2.3.1 Example

Find the asymptotic upper bound for the following function:

$$T(n) \begin{cases} 2T(n-1)+1, & n \geq 1 \\ 1, & n = 0 \end{cases}$$

Guess

$$T(n) \in O(2^n)$$

Inductive Basis

$$\begin{aligned} T(0) &= 2^0 \\ &= 1 \end{aligned}$$

Inductive Hypothesis
Assume that $T(n) = 2^n$ holds true for all $n = k$.

Inductive Step

$$\begin{aligned}
T(n) &= 2T(n-1)+1 && \text{Base equation} \\
&= 2T((k+1)-1)+1 && \text{Substitute n with } k+1 \\
&= 2T(k)+1 && \text{Simplify parameters to T(n)} \\
&= 2(2^k)+1 && \text{Substitute T(n) with inductive hypothesis} \\
&= 2^{k+1}+1 && \text{Property of exponents} \\
& && \text{Q.E.D}
\end{aligned}$$

2.4 Master Theorem

Used for divide and conquer recurrences that follow the generic form:

$$T(n) = a \cdot T(\frac{n}{b}) + f(n) \text{ where } a \geq 1, b > 1$$

2.4.1 Case 1

Condition

$$f(n) \in O(n^c)$$

$$c < log_b(a)$$

Solution

$$T(n) \in \Theta(n^{log_b(a)})$$

2.4.2 Case 2

Condition

$$f(n) \in \Theta(n^c)$$

$$c = log_b(a)$$

Solution

$$T(n) \in \Theta(n^{log_b(a)} \cdot log_2(n))$$

2.4.3 Case 3

Condition

$$f(n) \in \Omega(n^c)$$

$$c > log_b(a)$$

Regularity Condition

This case must also fulfill the regularity condition.

$$a \cdot f(\frac{n}{b}) \leq k \cdot f(n) \text{ where } k < 1$$

Solution

$$T(n) \in \Theta(f(n))$$

Remark

The idea behind this case is that given the generic form, the function $f(n)$ will grow far quicker than $a \cdot T(\frac{n}{b})$ and will be the primary influence of $T(n)$'s asymptotic behavior.

2.4.4 Example

$$T(n) = 64T(\frac{n}{4}) + 1000n^2$$

Given

$$f(n) = 1000n^2 \in \Theta(n^2)$$

$$a = 64$$

$$b = 4$$

$$c = 2$$

Condition

$$\begin{array}{ccc} c & ? & log_b(a) \\ 2 & ? & log_4(64) \\ 2 & < & 3 \end{array}$$

Condition satisfied for case 1

Solution

$$\therefore T(n) \in \Theta(n^{log_4(64)}) = \Theta(n^3)$$

2.4.5 Example

$$T(n) = 32T(\frac{n}{2}) + 20n^5$$

Given

$$f(n) = 20n^5 \in \Theta(n^5)$$

$$a = 32$$

$$b = 2$$

$$c = 5$$

Condition

$$\begin{array}{ccc} c & ? & log_b(a) \\ 5 & ? & log_2(32) \\ 5 & = & 5 \end{array}$$

Condition satisfied for case 2

Solution

$$\therefore T(n) \in \Theta(n^{log_2(32)} \cdot log_2(n)) = \Theta(n^5 \cdot lg(n))$$

2.4.6 Example

$$T(n) = 7T(\frac{n}{7}) + 19n^{11}$$

Given

$$f(n) = 19n^{11} \in \Theta(n^{11})$$

$$a = 7$$

$$b = 7$$

$$c = 11$$

Condition

$$\begin{array}{rcl} c & ? & log_b(a) \\ 11 & ? & log_7(7) \\ 5 & > & 1 \end{array}$$

Condition partially fulfilled for case 3. Must also check regularity condition.

$$\begin{array}{rcl} a \cdot f(\frac{n}{b}) & \leq & k \cdot f(n) \\ 7 \cdot \left[19(\frac{n}{7})^{11}\right] & \leq & k \cdot 19n^{11} \\ 7 \cdot \frac{n^{11}}{7^{11}} & \leq & k \cdot n^{11} \\ \frac{1}{7^{10}} \cdot n^{11} & \leq & k \cdot n^{11} \end{array}$$

Choosing $k = \frac{1}{7^{10}} < 1$ fulfills the regularity condition.

Solution

$$\therefore T(n) \in \Theta(19n^{11})$$

Chapter 3

Divide and Conquer Paradigm

3.1 Steps

1. **Divide** the problem into a number of independent subproblems.
2. **Conquer** the subproblems by solving them recursively.
3. **Combine** the solutions of the subproblems into the solution of the original problem.

3.2 Case Study: Merge Sort

Steps

1. **Divide** the list of n elements into two sublists with $\frac{n}{2}$ elements each.
2. **Conquer** the sublists by sorting the two sublists recursively using merge sort. When the sublists are of size 1, it becomes sorted.
3. **Combine** the elements of the two sublists by mering them in a sorted sequence.

Recurrence Relation

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + cn, & n \geq 2 \\ c, & n = 1 \end{cases}$$

Complexity

$$T(n) = \Theta(n \cdot lg(n))$$

3.3 Case Study: Fibonacci Sequence

Theorem

Fibonacci Sequence Starting with 0

Sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_0 \\ F_1 \end{bmatrix}$$

Fibonacci Sequence Starting with 1

Sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

Derivation

$$\begin{aligned}
\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-2} \\ F_{n-3} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-3} \\ F_{n-4} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^4 \begin{bmatrix} F_{n-4} \\ F_{n-5} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_0 \\ F_1 \end{bmatrix}
\end{aligned}$$

To verify, let's choose $n = 5$

$$\begin{aligned}
\begin{bmatrix} F_5 \\ F_4 \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^4 \begin{bmatrix} F_0 \\ F_1 \end{bmatrix} \\
&= \begin{bmatrix} 5 & 3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 3 \\ 2 \end{bmatrix}
\end{aligned}$$

The fifth Fibonacci number (assuming that the sequence starts at 0) is 3.

Recurrence Relation

$$T(n) = T(\frac{n}{2}) + O(1)$$

Complexity

$$T(N) \in \Theta(lg(n))$$

3.4 Case Study: Maximum Subarray

Steps

1. Divide the array in half into two subarrays (left subarray and right subarray).
2. Recursively repeat this process until each subarray consists of only one element. At this point, the maximum sum of each subarray is the single element.
3. Calculate the maximum sum for the cross section.
   (a) Start from the mid-point of the subarray.
   (b) Sum up all numbers from the mid-point to the first element. Whenever the sum exceeds its previous value, that value becomes the left sum.
   (c) Sum up all numbers from the mid-point+1 to the last element. Whenever the sum exceeds its previous value, that values becomes the right sum.
   (d) The summation of the left sum and the right sum becomes the maximum sum for the cross section. Note: If all the elements in the subarrays are negative, then the left and right sum will return 0 by default.
4. Compare the maximum sum from the left array, right array, and cross section. The largest of the three get returned.

Recurrence Relation

$$T(n) = 2T(\frac{n}{2}) + \Theta(n)$$

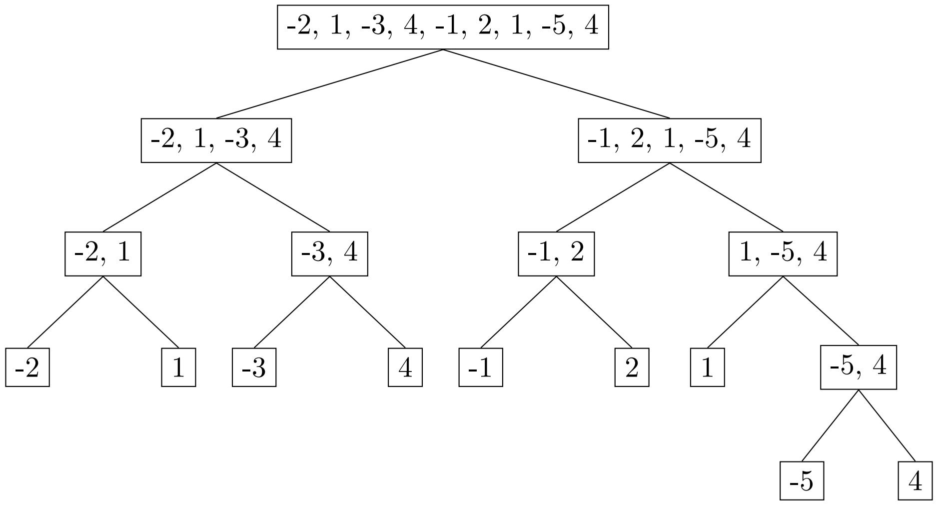
Complexity

$$T(n) \in \Theta(n \cdot lg(n))$$

3.4.1 Example

Find the maximum subarray of the following array: $\{-2, 1, -3, 4, -1, 2, 1, -5, 4\}$

Divide



Combine

Depth	Left Subarray	Right Subarray	Max(Left)	Max(Right)	Max(Cross)	Return
4	$\{-5\}$	$\{4\}$	-5	4	4	4
3	$\{-2\}$	$\{1\}$	-2	1	1	1
	$\{-3\}$	$\{4\}$	-3	4	4	4
	$\{-1\}$	$\{2\}$	-1	2	2	2
	$\{1\}$	$\{-5, 4\}$	1	4	4	4
2	$\{-2, 1\}$	$\{-3, 4\}$	1	4	1	4
	$\{-1, 2\}$	$\{1, -5, 4\}$	2	4	3	4
1	$\{-2, 1, -3, 4\}$	$\{-1, 2, 1, -5, 4\}$	4	4	6	6

The maximum sum is 6 from indices 3 to 6.

Visual Method of Finding the Max of Cross Section

Taking depth = 1 with left subarray = $\{-2, 1, -3, 4\}$ and right subarray = $\{-1, 2, 1, -5, 4\}$.

Cross Section Left Sum

$$\{-2, 1, -3, 4, \underbrace{-1}_{\text{Mid}}, 2, 1, -5, 4\}$$

$$\{-2, 1, -3, 4, \underbrace{-1}_{\text{-1}}, 2, 1, -5, 4\}$$

$$\{-2, 1, -3, \underbrace{4, -1}_{3}, 2, 1, -5, 4\}$$

$$\{-2, 1, \underbrace{-3, 4, -1}_{0}, 2, 1, -5, 4\}$$

$$\{-2, \underbrace{1, -3, 4, -1}_{1}, 2, 1, -5, 4\}$$

$$\{\underbrace{-2, 1, -3, 4, -1}_{\text{-1}}, 2, 1, -5, 4\}$$

Max Left Sum = 3

Cross Section Right Sum

$$\{-2, 1, -3, 4, -1, \underbrace{2}_{\text{Mid + 1}}, 1, -5, 5\}$$

$$\{-2, 1, -3, 4, -1, \underbrace{2}_{2}, 1, -5, 5\}$$

$$\{-2, 1, -3, 4, -1, \underbrace{2, 1}_{3}, -5, 5\}$$

$$\{-2, 1, -3, 4, -1, \underbrace{2, 1, -5}_{\text{-2}}, 4\}$$

$$\{-2, 1, -3, 4, -1, \underbrace{2, 1, -5, 4}_{2}\}$$

Max Right Sum = 3

Max Sum = 3 + 3 = 6

Chapter 4

Greedy Algorithm

4.1 Properties

Greedy Choice

A globally optimal solution can be arrived at by making a locally optimal (greedy) choice.

Optimal Substructure Property

An optimal solution to the problem contains within it optimal solution to the subproblems.

4.2 Case Study: Activity-Selection

Formal Problem Statement

Assume there exists n activities, each with a start time s_i and finish time f_i. Two activities i and j are said to be non-conflicting if $s_i \geq f_j$ or $s_j \geq f_i$. The objective is to find the maximum solution set of non-conflicting activities.

Informal Problem Statement

Given n activities and their respective start (s_i) and finish (f_i) times, find the maximum number of activities that can be performed.

Greedy Choice

Choose the next activity with a start time greater than or equal to the previous activity's finish time and has the next smallest finish time.

Steps

1. Sort the activities according to their finish times.
2. Select the first activity from the sorted list.
3. Repeat this process for the remaining activities with the condition that the start time of subsequent activities are greater than or equal to the preceding activity's finish time.

Pseudocode

```
procedure ActivitySelection(A)
    Sort(A)                                              ▷ Sort by finish times

    Let F be the set of finish times corresponding to the sorted list A
    Let B be the set of start times correspdonding to the sorted list A

    S = { A[1] }
    f = F_0

    for i=2 to n do
        if F_i ≥ f then
            S ∪ { A[i] }
            f = F_i
        end if
    end for
end procedure
```

Complexity

$$O(n \cdot lg(n))$$ [1]

[1] Total Time = O($n \cdot lg(n)) + \Theta(n)$. Sort Time + Greedy Activity Selection. Sort time will dominate.

4.3 Case Study: Huffman Coding

Formal Problem Statement

Let A be defined as the set of alphabets. ($A = \{a_0, a_1, a_2, ..., a_n\}$)
Let W be defined as the set of weights for which $w_i = \text{Weight}(a_i)$. ($W = \{w_0, w_1, w_2, ..., w_n\}$)
Let C be defined as the set of (binary) codewords for which $c_i = \text{CodeWord}(a_i)$.

Assume there exists n alphabets, each with a weight w_i. Find and define the codewords c_i for each respective alphabet a_i such that $\sum_{i=0}^{n} w_i \cdot length(c_i)$ is the smallest possible.

Informal Problem Statement

Given a set of symbols and their weights (probabilities), find a prefix-free binary code with minimum expected codeword length.

Greedy Choice

Choose the two alphabets with the lowest weight.

Steps

1. Pick two letters x and y from the alphabet A with the lowest frequencies or weight w_i.
2. Create a subtree with x and y as leaves. We will define the root as z.
3. The frequency or weight of node z will be define as $w_z = w_x + w_y$.
4. Remove x and y from alphabet.
   $A' = A - \{x, y\}$
5. Insert z into the alphabet.
   $A' = A + \{z\}$
6. Repeat this process until the set of alphabets A consists of only one alphabet.

Complexity

$$O(n \cdot lg(n))$$

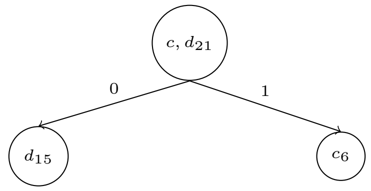
Total = $O(n \cdot lg(n)) + \Theta(n)$. Cost to sort alphabet by weight and cost to iterate through all alphabets.

4.3.1 Example

Let $A = \{a, b, c, d, e\}$ and $W = \{30, 16, 6, 15, 35\}$. Find their corresponding Huffman codes.

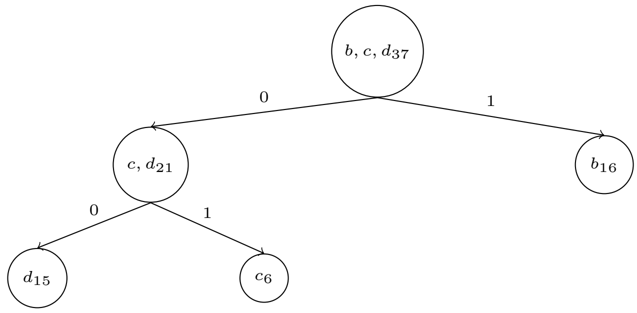
Merge c and d

Alphabet	Weight
e	35
a	30
b	16
d	15
c	6



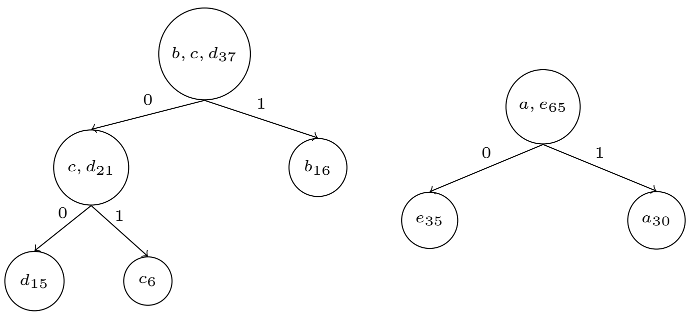
Merge c, d and b

Alphabet	Weight
e	35
a	30
c,d	21
b	16



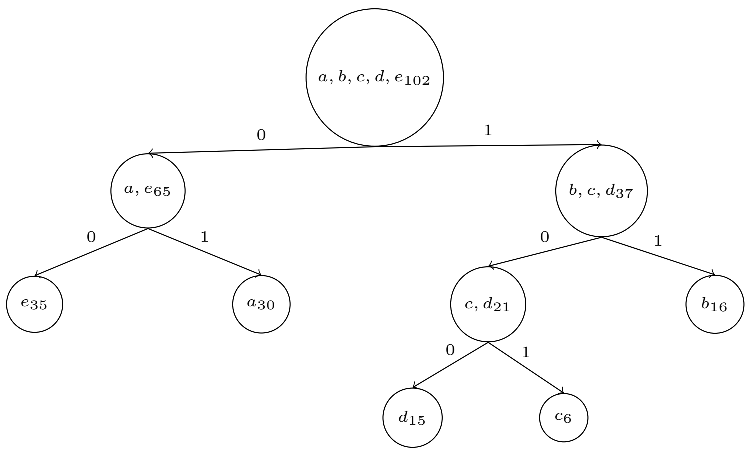
Merge e and a

Alphabet	Weight
b,c,d	37
e	35
a	30



Merge a, b, c, d and e

Alphabet	Weight
a,e	65
b,c,d	37



Solution

Alphabet	Weight	Codeword
e	35	00
a	30	01
b	16	11
d	15	100
e	6	101

Chapter 5

Dynamic Programming

5.1 Sequence

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution in a bottom-up fashion
4. Construct an optimal solution from computed information

5.2 Case Study: Rod Cutting

Problem Statement

Given a rod of length n and a set of prices $P = \{p_1, p_2, ...p_n\}$ such that p_i denotes the price of a piece of rod with length i, find the optimal (maximum) revenue r_i for cutting the rod into pieces whose length sum to n.

Steps

1. Start from rod length = 1.
2. With each subrod length, there will always be one "cut" – splitting the rod into a left half and right half. If the cut is equivalent to the length of the subrod, then it means that the entire length of the subrod was used (Left half will have the full length and the right half will have zero length).
3. Iterate from rod length = 1 to rod length = n.
   On each iteration of length i:
   (a) Assume that the left half has the full length and the right half has length 0.
   (b) Decrement the left half's left by 1 and increase the right half's length by 1.
   (c) Sum the revenue of the left half with the price of the right half.
   (d) Repeat this process until the left half is of length 0.
   (e) The maximum of all these sums become the maximum revenue of length i.

5.2.1 Example

Given a rod of length = 8 and P defined as $\{1, 5, 8, 9, 10, 17, 17, 20\}$, find the maximum revenue.

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

Subrod Length = 1

Length(Left)	Length(Right)	Revenue(Left) + Price(Right)
0	1	$0 + 1 = 1$

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20
Revenue	1							

Subrod Length = 2

Length(Left)	Length(Right)	Revenue(Left) + Price(Right)
1	1	$1 + 1 = 2$
0	2	$0 + 5 = 5$

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20
Revenue	1	5						

Subrod Length = 3

Length(Left)	Length(Right)	Revenue(Left) + Price(Right)
2	1	$5 + 1 = 6$
1	2	$1 + 5 = 6$
0	3	$0 + 8 = 8$

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20
Revenue	1	5	8					

Subrod Length = 4

Length(Left)	Length(Right)	Revenue(Left) + Price(Right)
3	1	8 + 1 = 9
2	2	5 + 5 = 10
1	3	1 + 8 = 9
0	4	0 + 9 = 9

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20
Revenue	1	5	8	10				

Subrod Length = 5

Length(Left)	Length(Right)	Revenue(Left) + Price(Right)
4	1	10 + 1 = 11
3	2	8 + 5 = 13
2	3	5 + 8 = 13
1	4	1 + 9 = 10
0	5	0 + 10 = 10

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20
Revenue	1	5	8	10	13			

Subrod Length = 6

Length(Left)	Length(Right)	Revenue(Left) + Price(Right)
5	1	13 + 1 = 14
4	2	10 + 5 = 15
3	3	8 + 8 = 16
2	4	5 + 9 = 14
1	5	1 + 10 = 11
0	6	0 + 17 = 17

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20
Revenue	1	5	8	10	13	17		

Subrod Length = 7

Length(Left)	Length(Right)	Revenue(Left) + Price(Right)
6	1	17 + 1 = 18
5	2	13 + 5 = 18
4	3	10 + 8 = 18
3	4	8 + 9 = 17
2	5	5 + 10 = 15
1	6	1 + 17 = 18
0	7	0 + 17 = 17

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20
Revenue	1	5	8	10	13	17	18	

Subrod Length = 8

Length(Left)	Length(Right)	Revenue(Left) + Price(Right)
7	1	18 + 1 = 19
6	2	17 + 5 = 22
5	3	13 + 8 = 21
4	4	10 + 9 = 19
3	5	8 + 10 = 18
2	6	5 + 17 = 22
1	7	1 + 17 = 18
0	8	0 + 20 = 20

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20
Revenue	1	5	8	10	13	17	18	22

The maximum revenue for a rod of length 8 is 22.

5.3 Case Study: Matrix Chain Multiplication

5.4 Case Study: Longest Common Subsequence

5.5 Case Study: Knapsack

Chapter 6

Graph Theory

6.1 Case Study: Breadth First Search (BFS)

6.2 Case Study: Depth-First Search (DFS)

Chapter 7

Side Topics

7.1 Proof by Mathematical Induction

Steps

1. Basis (Base Case)
2. Inductive Hypothesis
3. Inductive Step

7.1.1 Example

Prove that the following systems of equations has the solution $T(n) = n \cdot lg(n)$.

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + n, & n = 2^k \text{ for } k > 1 \\ 2, & n = 2 \end{cases}$$

Basis

$$\begin{aligned} T(2) &= (2) \cdot lg(2) \\ &= 2 \cdot 1 \\ &= 2 \end{aligned}$$

Inductive Hypothesis

Assume that $T(n) = n \cdot lg(n)$ holds true for all $n = 2^k$.

Inductive Step

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + n && \text{Base equation} \\ &= 2T(\frac{2^{k+1}}{2}) + 2^{k+1} && \text{Substitute n with } 2^{k+1} \\ &= 2T(2^k) + 2^{k+1} && \text{Simplify parameters to function T(...)} \\ &= 2(2^k \cdot lg(2^k)) + 2^{k+1} && \text{Inductive hypothesis} \\ &= 2^{k+1}\left[lg(2^k) + 1\right] && \text{Distributive property} \\ &= 2^{k+1}\left[lg(2^k) + lg(2)\right] && \text{Logarithmic identity} \\ &= 2^{k+1} \cdot lg(2^k \cdot 2) && \text{Logarithmic identity} \\ &= 2^{k+1} \cdot lg(2^{k+1}) && \text{Exponent property} \\ & && \text{Q.E.D} \end{aligned}$$

7.1.2 Example

Prove that the following systems of equations has the solution $T(n) = 2F(n) - 1$ where $F(n) = F(n-1) + F(n-2)$.

$$T(n) \begin{cases} T(n-1) + T(n-2) + 1, & \text{if } n \geq 2 \\ 0, & \text{if } n = \{0, 1\} \end{cases}$$

Basis

$T(0) = 1$

Inductive Hypothesis

Assume that $T(n) = F(n) - 1$ is true for all $n = k$.

Inductive Step

$$\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 && \text{Base equation} \\
T(k+1) &= T((k+1)-1) + T((k+1)-2) + 1 && \text{Substitute n with k+1} \\
&= T(k) + T(k-1) + 1 && \text{Simplify parameters to function T(...)} \\
&= (2F(k) - 1) + (2F(k-1) - 1) + 1 && \text{Inductive hypothesis} \\
&= 2F(k) + 2F(k-1) - 1 && \text{Simplify equation} \\
&= 2(F(k) + F(k-1)) - 1 && \text{Distributive property} \\
&= 2(F(k+1)) - 1 && \text{Definition of function: } F(k+1) = F(k) + F(k-1) \\
&= 2F(k+1) - 1 && \text{Simplify} \\
& && \text{Q.E.D}
\end{aligned}$$