

COMP20007 Design of Algorithms



Master Theorem

Lars Kulik

Lecture 10

Semester 1, 2023

Divide and Conquer

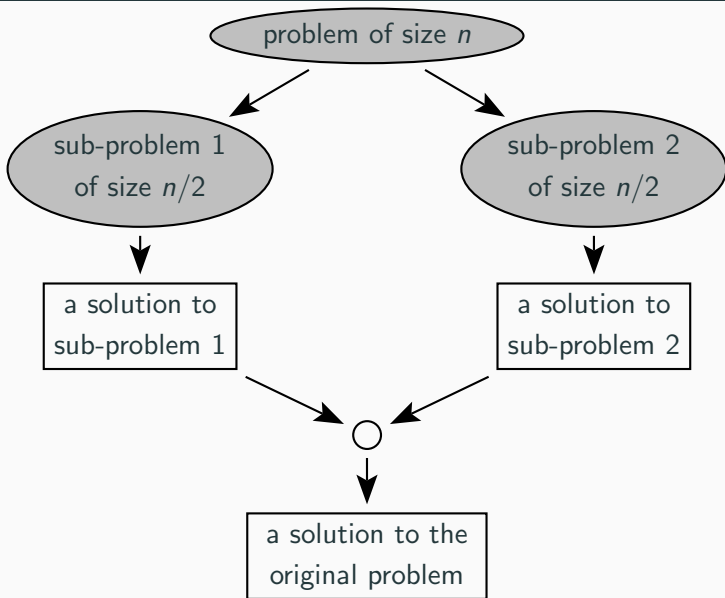
We earlier studied recursion as a powerful problem solving technique.

The divide-and-conquer strategy tries to make the most of this:

1. Divide the given problem instance into smaller instances.
2. Solve the smaller instances recursively.
3. Combine the smaller solutions to solve the original instance.

This works best when the smaller instances can be made to be of equal size.

Split-Solve-and-Join Approach



Divide-and-Conquer Algorithms

You have seen:

- Tree traversal

- Closest pair $O(n^2) \rightarrow O(n \log n)$


You will learn later:

- Mergesort
- Quicksort

Divide-and-Conquer Recurrences

What is the time required to solve a problem of size n by divide-and-conquer?

For the general case, assume we split the problem into b instances (each of size n/b), of which a need to be solved:


$$T(n) = aT(n/b) + f(n)$$

where $f(n)$ expresses the time spent on dividing a problem into b sub-problems and combining the a results.

(A very common case is $T(n) = 2T(n/2) + n$.)

How do we find closed forms for these recurrences?

The Master Theorem

(A proof is in Levitin's Appendix B.)

For integer constants $a \geq 1$ and $b > 1$, and function f with $f(n) \in \Theta(n^d)$, $d \geq 0$, the recurrence

$$T(n) = aT(n/b) + f(n) \quad f(n) \in \Theta(n^d)$$

(with $T(1) = c$) *doesn't matter.* has solutions, and

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Note that we also allow a to be greater than b .

Master Theorem: Example 1

$$T(n) = aT(n/b) + f(n) \quad f(n) \in \Theta(n^d)$$

$$T(n) = 2T(n/2) + n$$

$$a = 2, b = 2, d = 1$$

$$a > b^d, b = 2, d = 1$$

$$a = b^d \Rightarrow T(n) = \Theta(n \log n)$$



$$|-----| \quad 1 \times n$$

$$|-----| |-----| \quad 2 \times n/2$$

$$|-----| |-----| |-----| |-----| \quad 4 \times n/4$$

$$|-----| |-----| |-----| |-----| |-----| |-----| |-----| |-----| \quad \vdots$$

$$|-----| |-----| |-----| |-----| |-----| |-----| |-----| |-----| |-----| |-----| |-----| |-----| |-----| |-----| \quad (\log_2 n \text{ times})$$

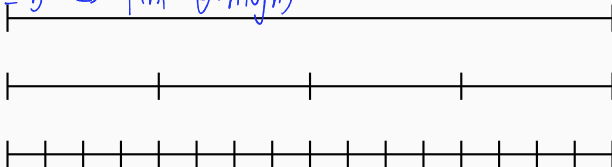
Master Theorem: Example 2

$$T(n) = 4T(n/4) + n$$

$$a = 4, b = 4, d = 1$$

$$a = 4, b = 4, d = 1$$

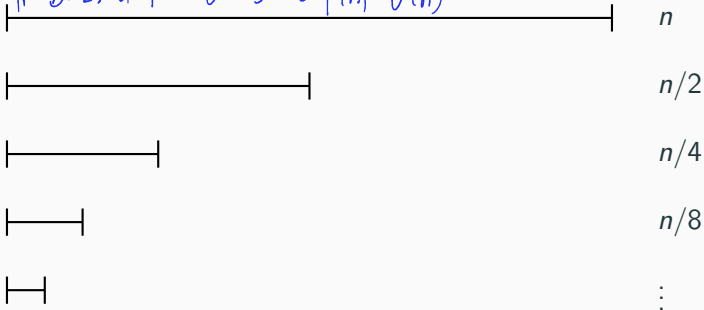
$$a = b^d \Rightarrow T(n) = \Theta(n \log n)$$



Master Theorem: Example 3

$$T(n) = T(n/2) + n \quad a = 1, b = 2, d = 1$$

$$a=1, b=2, d=1 \Rightarrow a < b^d \Rightarrow T(n) = \Theta(n)$$



Master Theorem: Example 4

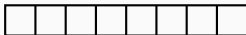
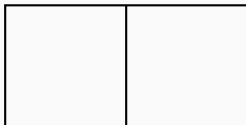
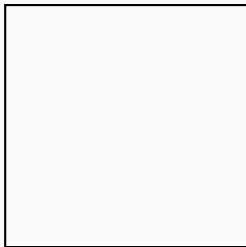
$$a=2, b=2, d=2$$

$$T(n) = 2T(n/2) + n^2$$

$$a < b^d \Leftrightarrow 2 < 4 \Rightarrow T(n) = \Theta(n^2)$$

$$a = 2, b = 2, d = 2$$

Here $a < b^d$ and we simply get n^d .



COMP20007 Design of Algorithms

String Search

Daniel Beck

Lecture 11

Semester 1, 2023

Housekeeping

If you are aware of the content in previous years, bear in mind we have changed the order this year:

- Today: string matching (to accommodate ANZAC day).
- Week 7: Dynamic Programming (based on student feedback).

Additional videos for each week (from 2020) will be uploaded to Ed. They are **optional and non examinable**, but students liked to use them for revision.

Recap

- Algorithm Analysis
- Design Techniques:
 - Brute-Force
 - Divide-and-Conquer
 - Greedy
- Examples of Algorithms:
 - Selection Sort, Closest Pair, etc.
 - Graph Traversal, Prim, Dijkstra, etc.

Moving Forward

- More Algorithms and Data Structures
- More Design Techniques
 - In some cases, they can result in more efficient algorithms in the general case (better complexity even in the worst case).
 - In other cases, they do not change the complexity in the worst case, but can result in better algorithms in practice.

String Search - Recap

- Goal: given text of size n , find a string (pattern) of size m .
- Brute force algorithm: $O(mn)$.

String Search - Brute Force

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
B A R B E R

Input Enhancement

- Longer shifts can be made if we know statistics about the pattern.
- The Horspool's algorithm use this idea to make string search faster.
- Key idea: scan the text from left to right but scan the pattern from right to left.

Longer shifts - Case 1

The last character is not in the pattern.

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P

B A R B E R

Shift the whole pattern.

Longer shifts - Case 2

The last character does not match but it is in the pattern.

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P

B A R B E R

Shift the pattern until the last occurrence of the character.

Longer shifts - Case 3

The last character matches but one of the $m - 1$ characters does not match and the last character is unique.

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
S E E S A W

Shift the whole pattern.

Longer shifts - Case 4

The last character matches but one of the $m - 1$ characters does not match and the last character is not unique.

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
R E O R D E R

Shift the pattern until the second-to-last occurrence of the character.

Your Turn

- Case 1: the last character is not in the pattern. Shift the whole pattern.
- Case 2: the last character does not match but it is in the pattern. Shift the pattern until the last occurrence of the character.
- Case 3: the last character matches but one of the $m - 1$ characters does not match and the last character is unique. Shift the whole pattern.
- Case 4: the last character matches but one of the $m - 1$ characters does not match and the last character is not unique. Shift the pattern until the second-to-last occurrence of the character.

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P

B A R B E R

Your Turn

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
B A R B E R

Simplifying Cases

- Case 1: the last character is not in the pattern. Shift the whole pattern.
- Case 3: the last character matches but one of the $m - 1$ characters does not match and the last character is unique. Shift the whole pattern.

Case “1-3”: there is a character mismatch and the last character in the text is **not** one of the $m - 1$ characters in the pattern. Shift the whole pattern.

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P

B A R B E R

S E E S A W

Simplifying Cases

- Case 2: the last character does not match but it is in the pattern. Shift the pattern until the last occurrence of the character.
- Case 4: the last character matches but one of the $m - 1$ characters does not match and the last character is not unique. Shift the pattern until the second-to-last occurrence of the character.

Case “2-4”: there is a character mismatch and the last character in the text is one of the $m - 1$ characters in the pattern. Shift the pattern until the last occurrence of the character in the $m - 1$ characters in the pattern.

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P

B A R B E R

R E O R D E R

Horspool - Preprocessing

- The number of allowed shifts depends only on the character type.
- Horspool builds a dictionary with all characters in the alphabet and their corresponding allowed skips for a pattern.
 - Dictionary: a set of $\langle \text{key}, \text{value} \rangle$ pairs. We will cover implementations of dictionaries later in the subject.

B A R B E R

character c | A B C D E F ... R ... Z _

shift $t(c)$ | 4 2 6 6 1 6 6 3 6 6 6

Whole $\rightarrow b$

Horspool - FindShifts



function FINDSHIFTS($P[0..m-1]$)

▷ m is the size of the pattern

for $i \leftarrow 0$ to $a-1$ **do**

▷ a is the size of the alphabet

$Shift[i] \leftarrow m$

▷ initialise the table

for $j \leftarrow 0$ to $m-2$ **do**

$Shift[P[j]] \leftarrow m - (j+1)$

return $Shift$

Horspool - Algorithm

function HORSPOOL($P[0..m-1]$, $T[0..n-1]$)

▷ n is the size of the string

$Shift \leftarrow \text{FINDSHIFTS}(P)$

$i \leftarrow m - 1$

while $i \leq n - 1$ **do**

$k \leftarrow 0$

while $k \leq m - 1$ **and** $P[m - 1 - k] = T[i - k]$ **do**

$k \leftarrow k + 1$

if $k = m$ **then**

return $i - m + 1$

else

$i \leftarrow i + Shift[T[i]]$

▷ We have a match

▷ Start of the match

▷ Slide the pattern along

return -1

Horspool - Properties

$O(mn)$ —: $(m) \rightarrow \text{constant}$
 \downarrow
 $O(n)$

- Worst-case still $O(mn)$.
- For random strings, it's linear and faster in practice compared to the brute force version.

Other String Search algorithms

- Boyer-Moore: extends Horspool to allow shifts based on suffixes.
- Knuth-Morris-Pratt: also preprocess the pattern but builds a finite-state automaton.
- Rabin-Karp: uses hash functions to filter negative matches.
 - We will cover hashing in Week 11.

Summary and Practical Uses

- String search algorithms can be sped up by input enhancement.
- Allocates extra memory to preprocess the pattern.
- Horspool uses a dictionary of shifts.
- The Boyer-Moore extension is one of the most used string searching algorithms, for instance in the “grep” Linux command line tool.

In 2 weeks: Dynamic Programming. Enjoy your Easter break!
=)