CS483-05 Analysis of Recursive Algorithms and Brute Force

Instructor: Fei Li

Room 443 ST II

Office hours: Tue. & Thur. 4:30pm - 5:30pm or by appointments

`lifei@cs.gmu.edu` with subject: CS483

`http://www.cs.gmu.edu/~ lifei/teaching/cs483_fall07/`

This lecture note is based on Introduction to The Design and Analysis of Algorithms by Anany Levitin.

Example 4: Counting Binary Bits

- Input: A positive decimal integer n.
- Output: The number of binary digits in n's binary representation.

Algorithm 0.1: COUNTBINARYBITS(n)

$count = 1$

while $n > 1$

do $\begin{cases} count = count + 1 \\ n = \lfloor n/2 \rfloor \end{cases}$

return $(count)$

Outline

- Analysis of Recursive Algorithms
- Brute Force
  - Ideas
  - Examples: Selection Sort & Bubble Sort
  - Examples: String Matching
  - Examples: Exhaustive Search

Analysis of Recursive Algorithms

- The iteration method
  - Expand (iterate) the recurrence and express it as a summation of terms depending only on n and the initial conditions.
- The substitution method
- Master Theorem

  (To be introduced in Chapter 4.)

Iteration Method: Examples

- $n!$

$$T(n) = T(n-1) + 1$$

- Tower of Hanoi

$$T(n) = 2T(n-1) + 1$$

Iteration: Example

- $n!$ $(T(n) = T(n-1) + 1)$

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ &= (T(n-2) + 1) + 1 \\ &= T(n-2) + 2 \\ \cdots & \quad \cdots \\ &= T(n-i) + i \\ \cdots & \quad \cdots \\ &= T(0) + n = n \end{aligned}$$

- Tower of Hanoi $(T(n) = 2T(n-1) + 1)$???

Tower of Hanoi $(T(n) = 2T(n-1) + 1)$

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2T(n-2) + 2 + 1 \\ \cdots & \quad \cdots \\ &= 2^iT(n-i) + 2^{i-1} + \cdots + 1 \\ \cdots & \quad \cdots \\ &= 2^{n-1}T(1) + 2^{n-1} + 2^{n-1} + \cdots + 1 \\ &= 2^{n-1}T(1) + \sum_{i=0}^{n-2} 2^i \\ &= 2^{n-1} + 2^{n-1} - 1 \\ &= 2^n - 1 \end{aligned}$$

Analysis of Recursive Algorithms

➢ The iteration method

- Expand (iterate) the recurrence and express it as a summation of terms depending only on n and the initial conditions.

➢ The substitution method

1. Guess the form of the solution
2. Use mathematical induction to find the constants

➢ Master Theorem

Substitution Method: Example 1

- Count number of bits ($T(n) = T(\lfloor n/2 \rfloor) + 1$)

Substitution Method: Example 1

- Count number of bits ($T(n) = T(\lfloor n/2 \rfloor) + 1$)
  - Guess $T(n) \leq \log n$.

$$
\begin{aligned}
T(n) &= T(\lfloor n/2 \rfloor) + 1 \\
&\leq \log(\lfloor n/2 \rfloor) + 1 \\
&\leq \log(n/2) + 1 \\
&\leq (\log n - \log 2) + 1 \\
&\leq \log n - 1 + 1 \\
&= \log n
\end{aligned}
$$

Substitution Method: Example 2

- Tower of Hanoi ($T(n) = 2T(n-1) + 1$)

Substitution Method: Example 2

- Tower of Hanoi ($T(n) = 2T(n-1) + 1$)
  - Guess $T(n) \leq 2^n$.

$$
\begin{aligned}
T(n) &= 2T(n-1) + 1 \\
&\leq 2 \cdot 2^{n-1} + 1 \\
&\leq 2^n + 1, \quad \text{wrong!}
\end{aligned}
$$

Substitution Method: Extension F_n

- Tower of Hanoi ($T(n) = 2T(n-1) + 1$)
  - Guess $T(n) \leq 2^n$.

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &\leq 2 \cdot 2^{n-1} + 1 \\ &\leq 2^n + 1, \quad \text{wrong!} \end{aligned}$$

  - Guess $T(n) \leq 2^n - 1$.

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &\leq 2(2^{n-1} - 1) + 1 \\ &= 2^n - 2 + 1 \\ &= 2^n - 1, \quad \text{correct!} \end{aligned}$$

Substitution Method: Extension F_n

- Fibonacci Numbers ($F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$)

Substitution Method: Extension F_n

Fibonacci Numbers ($F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$)

- $F_{n-2} < F_{n-1} < F_n, \forall n \geq 1$

Substitution Method: Extension F_n

Fibonacci Numbers $(F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2})$

- $F_{n-2} < F_{n-1} < F_n, \forall n \geq 1$
- Assume $2^{n-1} < F_n < 2^n$
- Guess $F_n = c \cdot \phi^n$, $1 < \phi < 2$.

Substitution Method: Extension F_n

Fibonacci Numbers $(F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2})$

- $F_{n-2} < F_{n-1} < F_n, \forall n \geq 1$
- Assume $2^{n-1} < F_n < 2^n$
- Guess $F_n = c \cdot \phi^n$, $1 < \phi < 2$.

$$\begin{aligned} c \cdot \phi^n &= c \cdot \phi^{n-1} + c \cdot \phi^{n-2} \\ \phi^2 &= \phi + 1 \\ \phi &= \frac{1 \pm \sqrt{5}}{2} \end{aligned}$$

General solution: $F_n = c_1 \cdot \phi_1^n + c_2 \cdot \phi_2^n$

$F_1 = 0, F_2 = 1$

General solution: $F_n = c_1 \cdot \phi_1^n + c_2 \cdot \phi_2^n$

$F_1 = 0, F_2 = 1$

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Summary: Algorithm Analysis

- Order of growth of functions
- Analyze algorithms' order of growth (using asymptotic notations).
  - Non-recursive algorithms
  - Recursive algorithms

    a. The iteration method

    b. The substitution method

    c. Master Theorem (to be introduced) ($T(n) = aT(n/b) + f(n)$.)

Outline

- Analysis of Recursive Algorithms
- Brute Force
  - Ideas
  - Examples: Selection Sort & Bubble Sort
  - Examples: String Matching
  - Examples: Exhaustive Search

Brute Force — Ideas

- **Brute force** is a straightforward approach to solve a problem, usually directly based on the problem statement and definitions of the concepts involved.

Outline

- Analysis of Recursive Algorithms
- Brute Force
  - Ideas
  - Examples: Selection Sort & Bubble Sort
  - Examples: String Matching
  - Examples: Exhaustive Search

Selection Sort & Bubble Sort

- Given n orderable items, sort them in non-decreasing order.

Selection Sort

- Given n orderable items, sort them in non-decreasing order.
- Input: An array $A[0, \ldots, n-1]$ of orderable elements.
- Output: An array $A[0, \ldots, n-1]$ sorted in non-decreasing order.

Algorithm 0.2: SELECTIONSORT($A[0, \cdots n-1]$)

for $i = 0$ **to** $n-2$
  do {
    $\min = i$
    for $j = i+1$ **to** $n-1$
      do {
        if $A[j] < A[\min]$
          then $\min = j$
      }
    Swap $A[i]$ and $A[\min]$
  }

Selection Sort

Analysis

- Input size: n.
- Basic operation: $A[j] < A[\min]$
- Running time:

$$
\begin{aligned}
C(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \\
&= \sum_{i=0}^{n-2} [(n-1)-(i-1)+1] = \sum_{i=0}^{n-2} (n-1-i) \\
&= \frac{(n-1)n}{2} \\
&= \Theta(n^2)
\end{aligned}
$$

Bubble Sort

- Given n orderable items, sort them in non-decreasing order.
- Input: An array $A[0, \ldots, n-1]$ of orderable elements.
- Output: An array $A[0, \ldots, n-1]$ sorted in non-decreasing order.

Algorithm 0.3: BUBBLESORT($A[0, \cdots n-1]$)

for $i = 0$ **to** $n-2$
  do {
    for $j = 0$ **to** $n-2-i$
      do {
        if $A[j+1] < A[j]$
          then Swap $A[j]$ and $A[j+1]$
      }
  }

Bubble Sort

Analysis

- Input size: n.
- Basic operation: $A[j+1] < A[j]$
- Running time:

$$
\begin{aligned}
C(n) &= \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 \\
&= \sum_{i=0}^{n-2} [(n-2-i)-0+1] = \sum_{i=0}^{n-2} (n-1-i) \\
&= \frac{(n-1)n}{2} \\
&= \Theta(n^2)
\end{aligned}
$$

Outline

- Analysis of Recursive Algorithms
- Brute Force
  - Ideas
  - Examples: Selection Sort & Bubble Sort
  - Examples: String Matching
  - Examples: Exhaustive Search

String Matching

- Given a string of n characters called the text; and a string of m characters called the pattern, find a substring of the text that matches the pattern.
- Input: An array $T[0,\ldots,n-1]$ of n characters representing a text
  An array $P[0,\ldots,m]$ characters representing a pattern
- Output: The index of the first character in the text that starts a matching substring or -1 if the search is unsuccessful
- Example: Pattern: 001011 Text: 10010101101001100101111010
  Pattern: happy Text: It is never too late to have a happy childhood.

String Matching

Algorithm 0.4: STRINGMATCHING($T[0,\cdots n-1], P[0,\ldots,m-1]$)

for $i = 0$ **to** $n - m$
do
  $j = 0$
  while $j < m$ **and** $P[j] = T[i+j]$
    do $j = j + 1$
  if $j = m$
    then return (i)
return (-1)

String Matching

Analysis

- Input size: $n + m$.
- Basic operation: $P[j] = T[i+j]$
- Running time (worst-case):

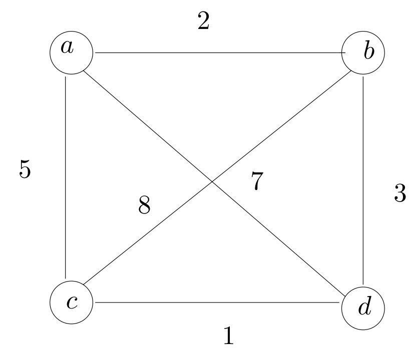
$$C(n+m) = (n-m+1)\cdot m = \Theta(nm)$$

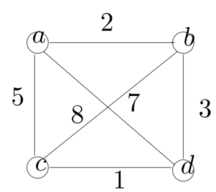
Outline

- Analysis of Recursive Algorithms
- Brute Force
  - Ideas
  - Examples: Selection Sort & Bubble Sort
  - Examples: String Matching
  - Examples: Exhaustive Search

Traveling Salesman Problem

- **TSP**: Find the shortest tour through a given set of n cities that visits each city exactly once before returning to the city where it starts.





Tour	Cost
$a \to b \to c \to d \to a$	$2+3+7+5=17$
$a \to b \to d \to c \to a$	$2+4+7+8=21$
$a \to c \to b \to d \to a$	$8+3+4+5=20$
$a \to c \to d \to b \to a$	$8+7+4+2=21$
$a \to d \to b \to c \to a$	$5+4+3+8=20$
$a \to d \to c \to b \to a$	$5+7+3+2=17$

Traveling Salesman Problem

Analysis

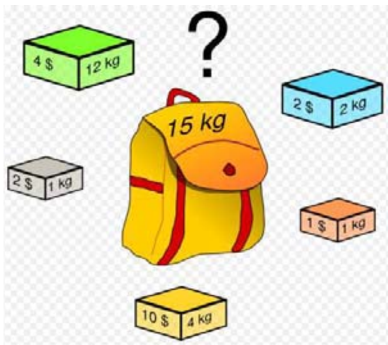
- Input size: $n \cdot (n-1)$.
- Running time:

$$C(n) = (n-1)!/2.$$

Knapsack Problem

- **Knapsack Problem**: Given n objects, each object i has weight w_i and value v_i, and a knapsack of capacity W, find most valuable items that fit into the knapsack

  Items are not splittable



Example: Knapsack capacity $W = 16$

Item	Weight	Value
1	2	$20
2	5	$30
3	10	$50
4	5	$10

Subset	Total weight	Total value
$\{1\}$	2	$20
$\{2\}$	5	$30
$\{3\}$	10	$50
$\{4\}$	5	$10
$\{1,2\}$	7	$50
$\{1,3\}$	12	$70
$\{1,4\}$	7	$30
$\{2,3\}$	15	$80
$\{2,4\}$	10	$40
$\{3,4\}$	15	$60
$\{1,2,3\}$	17	not feasible
$\{1,2,4\}$	12	$60
$\{1,3,4\}$	17	not feasible
$\{2,3,4\}$	20	not feasible
$\{1,2,3,4\}$	22	not feasible

Knapsack Problem

Analysis

- Input size: n (items).
- Running time:

  The number of subsets of an n-element set is 2^n.

$$C(n) = \Omega(2^n).$$

Assignment Problem

- **Assignment Problem**: There are n people to execute n jobs, one person per job. If ith person is assigned the jth job, the cost is $C[i, j], i, j = 1, \ldots, n$. Find the assignment with the minimum total cost.

	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

Assignment Problem

Analysis

- Input size: n.
- Running time:

$$C(n) = n!.$$

Summary for Brute Force

- Strengths
  1. Wide applicability
  2. Simplicity
  3. Yields reasonable algorithms for some important problems (e.g., matrix multiplication, sorting, searching, string matching)
- Weaknesses
  1. Rarely yields efficient algorithms
  2. Some brute-force algorithms are unacceptably slow
  3. Not as constructive as some other design techniques

Summary for Brute Force

- Exhaustive-search algorithms run in a realistic amount of time only on very small instances
- In some cases, there are much better alternatives
  - Shortest paths (greedy)
  - Minimum spanning tree (greedy)
  - Assignment problem (iterative improvement)
- In many cases, exhaustive search or its variation is the only known way to get exact solution

Summary

- Read Chap. 3.
- Next class: Chap. 4 and Master Theorem.