Algorithm Design and Analysis

วิชาบังคับก่อน: 204251 และ 206281

ผู้สอน: ตอน 1 ผศ. เบญจมาศ ปัญญางาม
ตอน 2 ผศ. ดร. จักริน ชวชาติ

บทที่ 2
ความสำคัญของอัลกอริทึมที่มีประสิทธิภาพ
(The importance of efficient algorithm)

Introduction

- The objective of this chapter is to explain the importance of analysis of algorithms, their notations, relationship and solving as problem as possible.

The importance of efficient algorithm

Cycle of program development

- Defining the problem
  - Problem ? , Scope? , Input? ,Output?
- Designing the solution
  - a functional description of the task (an algorithm)
  - a flowchart (a diagram that represents an algorithm or process)
- Coding and debugging the program
- Testing the program
- Documenting the program
- Maintaining the program

What is an Algorithm?

Let us consider the problem of preparing an omelet.

- Get the frying pan.
- Get the oil.
- Do we have oil?
  - If yes, put it in the pan.
  - If no, do we want to buy an oil?
    - If yes, the go out and buy.
    - If no, we can terminate
- Turn on the stove, etc...

What is an Algorithm?

- What we are doing, for a given problem(preparing an omelet), giving step by step procedure for solving it.
- Formal definition of an algorithm can be given as:

An algorithm is the step-by-step instructions to a given problem

- One important note to remember while writing the algorithm is: we do not have to prove each step of the algorithm.

The importance of efficient algorithm

- Computational problem
  - Sorting, Searching, Partitioning
  - Matrix multiplication, Prime number testing, Random number generation
  - Minimum spanning tree, Graph coloring
- Example : Sorting problem
  - Input: A sequence of n numbers (a1, a2, ..., an)
  - Output: The permutation (reordering) of the input sequence such as á1, á2, .. , án such that á1 ≤ á2 ≤ ... ≤ án
- Instance : 5 9 1 3

5 9 1 3 → Sorting Algorithm → 1 3 5 9

The importance of efficient algorithm

Algorithms in real world applications

- Internet
  - Sorting , Shortest path, Hashing, String Matching
- Games and puzzles
  - Word Matching , Path Finding
- Network/Security
  - Public-key cryptography and digital signatures in Electronic Commerce
  - Fingerprint Recognition

Why analysis of algorithm?

- If we want to go from city “A” to city “B”. There are many ways of doing this: by flight, by bus, by train, and by the cycle. Depending on the availability and convenience we choose the one which suits us.
- Similarly, in computer science there can be multiple algorithms exists for solving the same problem.
- Algorithm analysis helps us determining which of them is efficient in terms of time and space consumed.

The importance of efficient algorithm

- Sorting Algorithm
  - Insertion Sort or Merge Sort
- Desired Properties of algorithm
- "Correctness"
  - Correct : for all possible, it halts with the correct output.
  - Incorrect : not halt at all on some input instances, or halt with other than the desired answer.
- "Efficient Algorithm"
- Analyzing Algorithm to predict resource utilization
  - Memory : Space complexity
  - Running time : Time complexity

Goal of Analysis of Algorithms

- The goal of analysis of algorithms is to compare algorithms (or solution) mainly in terms of running time but also in terms of other factors (e.g. memory, developer's effort etc.)

What is Running Time Analysis?

- It is a process of determining how processing time increases as the size of the problem (input size) increases.
- Input size is number of elements in the input and depending on the problem type the input may be of different types. In general, we encounter the following types of inputs:
  - Size of an array
  - Polynomial degree
  - Number of bits in binary representation of the input
  - Number of elements in a matrix
  - Vertices and edges in a graph

How to compare Algorithms?

- To compare algorithms, let us define some objective measures.
- Execution times?
  - Not a good measure as execution times are specific to a particular computer
- Number of statements executed?
  - Not a good measure since the number of statements varies with the programming language as well as style of the individual programmer

How to compare Algorithms?

- Ideal Solution?
  - Let us assume that we expressed running time of the given algorithm as a function of the input size n (i.e. f(n)).
- We can compare these different functions corresponding to running times and this kind of comparison is independent of machine time, programming style, etc.

What is Rate of Growth?

- The rate at which the running time increases as a function of input is called rate of growth.
- Let us assume that you went to a shop for buying a car and a cycle. If your friend sees you there and asks what you are buying then in general we say buying a car.
- This is because cost of car is too big compared to cost of cycle
  - Total cost = cost of car + cost of cycle
  - Total cost ≈ cost of car (approximation)

What is Rate of Growth?

- We can represent the cost of car and cost of cycle in terms of function and for a given function we ignore the low order terms that are relatively insignificant (for large value of input size, n)
- As an example in the below case n^4, $2n^3$, 100n, and 500000 are individual costs of some function and we approximate it to n^4. Since, n^4 is the largest rate of growth.

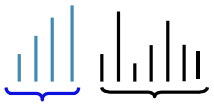
$$n^4+2n^3+100n+500000 \approx n^4$$

The importance of efficient algorithm

- Which algorithm ? : Analyzing the time complexity
- Worst case analysis
  - Sorting algorithm
    - Insertion-Sort : $an^2+bn+c \approx O(n^2)$
    - Merge-Sort : O(n log n)
  - Searching algorithm
    - Sequence Search : O(n)
    - Binary Search : O(log n)

Analyzing the time complexity : Review

- Analyzing the time complexity :Counting the primitive operations
- “แต่ละคำสั่งทำงานกันกี่ ครั้ง ?” (n = size of input)
- Insertion Sort : an example algorithm to solve the sorting problem



1)sorted 2)not yet examined

8 3 9 1 4

key

1)sorted 2)not yet examined

8 3 9 1 4

3 8 9 1 4

3 8 9 1 4

1 3 8 9 4

1 3 4 8 9

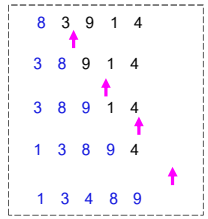
Analyzing the time complexity : Review

Input : an array A

Output : the sorted array A

Insertion_Sort_Algorithm(A)

1.	For j = 2 to length(A)	5 = n
2.	key = A[j]	4 = n-1
3.	//Insert A[j] into the sorted sequence A[1..j-1]	
4.	i = j -1	4 = n-1
5.	while i > 0 and A[i] > key	2 +1+4+3
6.	A[i +1] =A[i]	1 +0+3+2
7.	i = i -1	1 +0+3+2
8.	A[i+1] = key	4 = n-1



8 3 9 1 4

3 8 9 1 4

3 8 9 1 4

1 3 8 9 4

1 3 4 8 9

Analyzing the time complexity : Review

Input : an array A

t_j is the number of times the while loop test

Output : the sorted array A

Insertion_Sort_Algorithm(A)

		Constant	Times
1.	For j = 2 to length(A)	c1	n
2.	key = A[j]	c2	n-1
3.	//Insert A[j] into the sorted sequence A[1..j-1]		
4.	i = j -1	c4	n-1
5.	while i > 0 and A[i] > key	c5	$\sum_{j=2}^{n} t_j$
6.	A[i +1] =A[i]	c6	$\sum_{j=2}^{n} (t_j - 1)$
7.	i = i -1	c7	$\sum_{j=2}^{n} (t_j - 1)$
8.	A[i+1] = key	c8	n-1

Analyzing the time complexity : Review

- Analyzing the time complexity :Counting the primitive operations
- “แต่ละคำสั่งทำงานกันกี่ ครั้ง ?” (n = size of input)
- Running time of the algorithm is the sum of running time of each statement executed ;a statement that takes c_in to total running time.
- Let T(n), the running time of Insertion-Sort

$$T(n)=c_1n+ (c_2+c_4+c_8)(n-1)+ c_5\sum_{j=2}^{n} t_j +(c_6+c_7) \sum_{j=2}^{n}(t_j-1)$$

Types of analysis

- If we have an algorithm for a problem and want to know on what inputs the algorithm is taking less time (performing well) and on what inputs the algorithm is taking huge time.
- To analyze an algorithm we need some kind of syntax and that forms the base for asymptotic analysis/notation.
- Best Case :
  - Defines the input for which the algorithm takes lowest time.
  - Input is the one for which the algorithm runs the fastest.

Three types of analysis

- Worst Case :
  - Defines the input for which the algorithm takes huge time.
  - Input is the one for which the algorithm runs the slowest.
- Average Case :
  - Provides a prediction about the running time of the algorithm
  - Assume the input is random:
    Lower Bound <= Average Time <= Upper Bound

Analyzing the time complexity : Review

$T(n)=c_1n+ (c_2+c_4+c_8)(n-1)+ c_5\sum_{j=2}^{n} t_j +(c_6+c_7) \sum_{j=2}^{n}(t_j-1)$

Best Case : data is already sorted $\therefore t_j = 1, \ \forall\ j$

$T(n) = c_1n+ (c_2+c_4) (n-1)+c_5(n-1)+ c_8(n-1)$

$= (c_1+c_2+c_4+c_5+ c_8)n - (c_2+c_4+c_5+ c_8)$

running time -> can be express as an+b

-> a linear function of n -> O(n)

Analyzing the time complexity : Review

$T(n)=c_1n+ (c_2+c_4+c_8)(n-1)+ c_5\sum_{j=2}^{n} t_j +(c_6+c_7) \sum_{j=2}^{n}(t_j-1)$

- Worst Case : reverse sorted order $\therefore t_j = j, \ \forall\ j$

$$\sum_{j=2}^{n} t_j = \frac{n(n+1)}{2} - 1 \quad , \quad \sum_{j=2}^{n} (t_j - 1) = \frac{(n-1)n}{2}$$

$$T(n) = C_1n + (C_2 + C_4)(n-1) + C_5\left[\frac{n(n+1)}{2} - 1\right] + (C_6 + C_7)\left[\frac{n(n-1)}{2}\right] + C_8(n-1)$$

$$= \left[\frac{C_5}{2} + \frac{C_6}{2} + \frac{C_7}{2}\right]n^2 + \left[C_1 + C_2 + C_4 + \frac{C_5}{2} - \frac{C_6}{2} - \frac{C_7}{2} + C_8\right]n - (C_2 + C_4 + C_5 + C_8)$$

- Running time can be express as $an^2+bn+c \approx O(n^2)$

Analyzing the time complexity : Review

- Case : for loop

```
1. for i = 1 to n
2.     A[ i ]=0
3. for i = 1 to n
4.     for j = 1 to n
5.         A[ i ] = A[ i ] + A[ j ]
```

$\sum_{i=1}^{n} 1 = n$

$T(n) = c_1(n+1) + c_2 n + c_3(n+1) + c_4 \sum_{i=1}^{n}(n+1) + c_5 \sum_{i=1}^{n}(n)$

$= c_1 n + c_1 + c_2 n + c_3 n + c_3 + c_4 n^2 + c_4 n + c_5 n^2$

$= (c_4 + c_5)n^2 + (c_1 + c_2 + c_3 + c_4)n + (c_1 + c_3)$

$\therefore T(n) = an^2 + bn + c \approx O(n^2)$

$T(n) = (c_1 + c_2) n + c_3 n + (c_4 + c_5) \sum_{i=1}^{n}(n) = an^2 + bn + c \approx O(n^2)$

Analyzing the time complexity : Review

- Case : while loop : A linear (sequential) search
- Input: Array A[1..n] and key. x
  Output: Position of key (0 , if not found)

linear_search(x, A)

```
1. i = n
2. while ( i > 0 and x ≠ A[ i ] )
3.     i = i - 1
4. return(i)
```

$\sum_{i=n}^{1} 1$ =n รอบ

Best Case: O( ) 1
Worst Case : O( ) n

T(n) = ?

Analyzing the time complexity : Review

- Case : while loop : Binary search
- Input: Sorted array A[1..n] and a searched key, x.
  Output: Position of key (0 , if not found).
- Binary_search(x, A)

```
1. i = 1, j = n
2. while (i < j)
3.     m = ⌊(i + j)/2⌋
4.     if x = A[m] then return (m)
5.     else if x > A[m] then  i = m + 1
6.     else  j = m -1
7. return(0)
```

The number of elements is n
and k is the height of binary tree
- $2^k - 1 = n$
- $\log_2 2^k = \log (n+1)$
- $k = \log_2 (n+1)$

The number of comparison is at most k+1
- Worst Case : O(log n)
- Best Case : O(1)

T(n)= ?

Analyzing the time complexity : Practice and Solution

- 1. An algorithm that finds the maximum element in a finite sequence of array A

Input: data array A[1..n]
Output: the largest element

Find_Max(A)

```
1. //A = {a1, a2, ..., an } and max preserves the largest element
2. max = A[1]
3. for i = 2 to n
4.     if max < a[ i ] then max = A[ i ]
5. return(max)
```

$\sum_{i=1}^{n} 1$ =n รอบ

$T(n) = c_2 + c_3 n + c_4 (n-1) + c_5$

$= (c_3 + c_4) n + (c_2 + c_5 - c_4) = an + b = O(n)$

or ignore for case : n -1

$= c_2 + c_3 n + c_4 n + c_5 = an + b = O(n)$

Analyzing the time complexity : Practice and Solution

- 2. Selection sort algorithm to solve the sorting problem

Selection_Sort(A)

Input: data array A[1..n]

Output: sorted data array A

```
1.  for i=n downto 2
2.    j=1
3.    for (k = 2; k <= i; k++)
4.       if (A[j] < A[k]  ) then j = k;
5.    swap(A,i,j)
```


Analyzing the time complexity : Practice and Solution

```
int Function1(int n)
{   int  sum,i,j,k;
/*1*/  sum=0;
/*2*/  for (i = n; i >0 ; i = i/2)
/*3*/       for (j = 1; j<= n; j++)
/*4*/             sum++;
/*5*/  return sum;
}
```