The National School of Artificial Intelligence
المدرسة الوطنية العليا للذكاء الاصطناعي

Data Structures and Algorithms 2

Chapter 1
Programming : A General Overview

Dr. Fouzia ANGUEL

2nd year / S3

September 2024 – January 2025

Course Outline

- **Introduction**

- **Mathematics Review**
  - Exponent
  - Logarithms
  - Series
  - Proofs
    - Proof By Induction
      - Sum of arithmetic serie
      - Sum of squares
      - Fibonacci numbers
    - Proof By Counter Example
    - Proof By Contradiction

- **Recursion**
  - Recursion vs Iteration

Introduction

What Is an Algorithm?

An algorithm is an explicit sequence of instructions, performed on data, to accomplish a desired objective.

Example

Problem : determine the k^{th} largest number in a group of N numbers.

Selection problem

Introduction

Selection problem

Algorithm 1

- read the N numbers into an array,
- sort the array in decreasing order
- return the element in position k.

Introduction

Selection problem

Algorithm 2

- read the first k elements into an array and sort them (in decreasing order).

- A new element, is ignored or it is placed in its correct spot in the array, bumping one element out of the array.

- The element in the kth position is returned as the answer.

Introduction

- An algorithm is considered **correct** if, for every input instance, it terminates with the correct output.

- We say that a correct algorithm **solves** the given computational problem.

- An **incorrect** algorithm might not terminate at all on some input instances,

- or it might halt with an answer other than the desired one.

Introduction

Selection problem

Simulation

- Using a random file of **30** million elements and **k =15,000,000.**
- It showed that neither algorithm finishes in a reasonable amount of time;
- Each one requires several days of computer processing to terminate (although eventually with a correct answer).
- The proposed algorithms work, BUT they cannot be considered good algorithms.
- In many problems, writing a working program is not good enough.
- If the program is to be run on a large data set then the running time becomes an issue .

Introduction

How to estimate the running time of a program?

How can we tell which algorithm is better?

- ❖ We could implement both algorithms, run them both

  Expensive and error prone

- ❖ Preferably, we should analyze them mathematically

  Algorithm analysis

  - measuring the resources needed for its execution,

    time (how long the algorithm takes to run) and

    space (how much memory the algorithm uses).

  - It helps evaluate the efficiency of an algorithm, especially when the input size becomes large.

Mathematics Review

Exponents

$$X^A X^B = X^{A+B}$$

$$X^{AB} = (X^A)^B = (X^B)^A$$

$$X^N + X^N = 2X^N \neq X^{2N}$$

$$2^N + 2^N = 2^{N+1}$$

Mathematics Review

Logarithms

In computer science, all logarithms are to the base 2 unless specified otherwise :

$X^A = B$ if and only if $\log_X B = A$

$\log_A B = \log_C B / \log_C A$ $\quad A,B,C > 0 , A \neq 0$

$\log(AB) = \log A + \log B$ $\quad A,B > 0$

$\log(A/B) = \log A - \log B$

$\log(A^B) = B \log A$

$a^{\log n} = n^{\log a}$

$\log 1 = 0$ $\quad \log 2 = 1$ $\quad \log 1024 = 10$ $\quad \log 1048\,576 = 10$

Arithmetic series

Each term in an arithmetic series is increased by a constant value (usually 1) :

$$0+1+2+3+\cdots+n=\sum_{k=0}^{n}k=\frac{n(n+1)}{2}$$

Proof 1: write out the series twice and add each column

$$\begin{array}{ccccccccccccccc} & 1 & + & 2 & + & 3 & + \cdots + & n-2 & + & n-1 & + & n \\ + & n & + & n-1 & + & n-2 & + \cdots + & 3 & + & 2 & + & 1 \\ \hline & (n+1) & + & (n+1) & + & (n+1) & + \cdots + & (n+1) & + & (n+1) & + & (n+1) \end{array}$$

$$= n\,(n+1)$$

Since we added the series twice, we must divide the result by 2

Geometric series

The next series we will look at is the geometric series with common ratio r:

$$\sum_{k=0}^{n} r^k = \frac{1 - r^{n+1}}{1 - r}$$

and if $|r| < 1$ then it is also true that

$$\sum_{k=0}^{\infty} r^k = \frac{1}{1 - r}$$

Geometric series

proof : multiply by $$1 = \frac{1-r}{1-r}$$

Telescoping series:
all but the first and last terms cancel

$$\sum_{k=0}^{n} r^k = \frac{(1-r)\sum_{k=0}^{n} r^k}{1-r}$$

$$= \frac{\sum_{k=0}^{n} r^k - r\sum_{k=0}^{n} r^k}{1-r}$$

$$= \frac{(1 + r + r^2 + \cdots + r^n) - (r + r^2 + \cdots + r^n + r^{n+1})}{1-r}$$

$$= \frac{1-r^{n+1}}{1-r}$$

Geometric series

A common geometric series will involve the ratios r = ½ and r = 2:

$$\sum_{i=0}^{n}\left(\frac{1}{2}\right)^{i}=\frac{1-\left(\frac{1}{2}\right)^{n+1}}{1-\frac{1}{2}}=2-2^{-n}$$

$$\sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^{i}=2$$

$$\sum_{k=0}^{n}2^{k}=\frac{1-2^{n+1}}{1-2}=2^{n+1}-1$$

Proofs

The two most common ways of proving statements in data-structure analysis are proof by induction and proof by contradiction .

The best way of proving that a theorem is false is by exhibiting a counterexample.

Proof by induction

- Prove that a property holds for input size 1 (base case)
- Assume that the property holds for input size 1,…n (inductive hypothesis).
- Show that the property holds for input size n+1 (next value).

Then, the property holds for all input sizes, n.
(this proves the theorem)

Proof by induction

Example 1: Let's prove by induction the sum of the arithmetic serie :

$$0+1+2+3+\cdots+n=\sum_{k=0}^{n}k=\frac{n(n+1)}{2}$$

The statement is true for $n = 0$:

$$\sum_{i=0}^{0}k=0=\frac{0\cdot 1}{2}=\frac{0(0+1)}{2}$$

Assume that the statement is true for an arbitrary n:

$$\sum_{k=0}^{n}k=\frac{n(n+1)}{2}$$

We must show that

$$\sum_{k=0}^{n+1}k=\frac{(n+1)(n+2)}{2}$$

Proof by induction

Then, for $n + 1$, we have:

$$\sum_{k=0}^{n+1} k = (n+1) + \sum_{i=0}^{n} k$$

By assumption, the second sum is known:

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

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

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

- The statement is true for $n = 0$ and
- the truth of the statement for n implies the truth of the statement for $n + 1$.
- Therefore, by the process of mathematical induction, the statement is true for all values of $n \geq 0$.

Proof by induction

Example 2 : **Sum of Squares**

Now we show that

$$1 + 2^2 + 3^2 + \dots\dots\dots n^2 = \sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$1(1+1)(2+1)/6 = 1$ Thus the property holds for $n = 1$ (base case)

Assume that the property holds for $n=1,..m$, thus

$$1 + 2^2 + 3^2 + \dots\dots\dots m^2 = m(m + 1)(2m + 1)/6$$

and show the property for $m + 1$,

$$1 + 2^2 + 3^2 + \dots\dots\dots m^2 + (m+1)^2 = (m+1)(m + 2)(2m+3)/6 \quad ?$$

$$\begin{aligned} 1 + 2^2 + 3^2 + \dots\dots\dots m^2 + (m+1)^2 &= (m)(m + 1)(2m+1)/6 + (m+1)^2 \\ &= (m+1)[m(2m+1)/6 + m+1] \\ &= (m+1)[2m^2 + m + 6m + 6]/6 \\ &= (m+1)(m + 2)(2m + 3)/6 \end{aligned}$$

Proof by induction

Example 3 : Fibonacci Numbers

Sequence of numbers, F_0 F_1 , F_2 , F_3 ,.......

$F_0 = 1, F_1 = 1$

$F_i = F_{i-1} + F_{i-2}$ (Recurrence relation)

$F_2 = 2$, $F3 = 3$, $F4 = 5$

Prove that $F_n < (5/3)^n$ for $i >= 1$

Base case:

$F_1 = 1 < (5/3)$

$F_2 = 2 < (5/3)^2$

Proof by induction

We assume that the theorem is true for i= 1,…k (inductive hypothesis)

We need to show $F_{k+1} < (5/3)^{k+1}$?

We have :

$$F_{k+1} = F_k + F_{k-1}$$

$$\begin{aligned}
F_{k+1} &< (5/3)^k + (5/3)^{k-1} \\
&< (3/5)(5/3)^{k+1} + (3/5)^2(5/3)^{k+1} \\
&< ((3/5)+ (3/5)^2) (5/3)^{k+1} \\
&< (24/25) (5/3)^{k+1}
\end{aligned}$$

$$F_{k+1} < (5/3)^{k+1}$$

proving the theorem

Proof by Counterexample

- Want to prove something is not true!
- Give an example to show that it does not hold!

Example: is $F_N < N^2$?

No, F11 = 144 > 121!

- However, if you were to show that $F_N < N^2$ is true

then you would need to show for all N, and not just one number.

Proof by Contradiction

- Suppose you want to prove a theorem .
- Assume that the theorem is false .
- Then show that you arrive at an impossibility and hence the original assumption was erroneous .

Example: The proof that there is an infinite number of prime numbers .

Proof by Contradiction

We assume that the theorem is false,

the number of primes is finite, k. So that The largest prime is P_k

Let $P_1, P_2, \ldots, P_k$ be all the primes

Consider the number N = 1 + P1 * P2* ..... * Pk

N is larger than P_k, thus N is not prime (hypothesis)

So N must be the product of some primes.

However, none of the primes P1, P2 , ... , Pk divide N exactly.

So N is not a product of primes. (contradiction)

because every number is either prime or a product of primes

Introduction to recursion

Let’s consider the following function f valid on nonnegative integers, that satisfies :

$$\left\{ \begin{array}{ll} f(0) = 0 & \\ f(x) = 2f(x - 1) + x^2 & x>0 \end{array} \right\}$$

A function that is defined in terms of itself is called recursive.

the recursive implementation of f

```
int f( int x )
{
 if( x == 0 )
        return 0;
 else
        return 2 * f( x - 1 ) + x * x;
}
```

A recursive function is a subroutine which calls itself, with different parameters.

Introduction to recursion

Basic rules of recursion :

1. Base cases. You must always have some base cases, which can be solved without recursion.

2. Making progress. For the cases that are to be solved recursively, they should progressively move towards the base case.

3. Design rule. Assume that all the recursive calls work.

4. Compound interest rule. Never duplicate work by solving the same instance of a problem in separate recursive calls.

Introduction to recursion

Example 1 : Printing numbers digit by digit

We wish to print out a positive integer, n.
Our routine will have the heading printOut(n).
Assume that the only I/O routine available printDigit(m) will take a single-digit number and outputs it.

```
void printOut( int n ) // Print nonnegative n
{
if( n >= 10 )
        {printOut( n / 10 );}
printDigit( n % 10 );
}
```

Recursive routine to print an integer

Introduction to recursion

Example 1 : Printing numbers digit by digit

Proof :Recursion and induction

Let us prove rigorously that the recursive number printing program works, we'll use a proof by induction.

- First, if n has one digit, then the program is trivially correct,
- Assume then that printOut works for all numbers of k or fewer digits.
- A number of k + 1 digits is expressed by its first k digits (n/10) followed by its least significant digit.
  by the inductive hypothesis, (n/10) is correctly printed,
  and the last digit is n mod 10,

so the program prints out any (k+1)-digit number correctly.

Introduction to recursion

Example 2 : Factorial

To evaluate factorial(n)

factorial(n) = n*(n-1)*...*2* 1

= n * factorial(n-1)

The recursive routine

```
int Factorial(int m)
{
        If     (m == 1 )   return  1
        else    return ( m * Factorial(m-1)) ;
}
```

Introduction to recursion

Example 2 : Factorial

Recursion Versus Iteration

- Factorial of n (n>0) can be iteratively computed as follows:

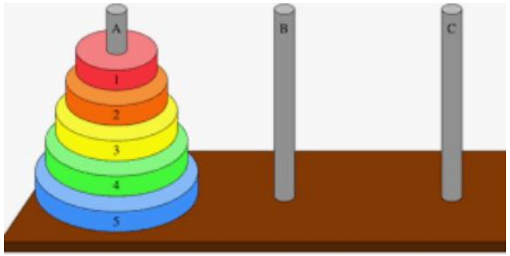
```
factorial = 1
for j=1 to n
        factorial  ←  factorial * j
```

- Compare to the recursive version.

    - In general, iteration is more efficient than recursion because of maintenance of state information, so use it when it simplifies the problem.

Introduction to recursion
Towers of Hanoi

- Source peg, Destination peg, Auxiliary peg



- k disks on the source peg,
- a bigger disk can never be on top of a smaller disk

Need to move all k disks to the destination peg using the auxiliary peg, without ever keeping a bigger disk on the smaller disk.

Introduction to recursion

```
Tower of Hanoi(k, source, auxiliary, destination)
  {
     If k=1 move disk from source to destination; (base case)
     Else,
        {
           Tower of Hanoi(top k-1, source, destination,
              auxiliary);
           Move the kth disk from source to destination;
           Tower of Hanoi(k-1, auxiliary, source, destination);
        }
}
```