Data Structures

Lecture 02: Analysis of Algorithms

Data Structures

What is a Problem

- A problem is
  - Something we want to solve,
  - Or a question we want to answer
- A computational problem is a general question to be answered
  - Generally has inputs, whose values are left unspecified, and the answers are the outputs

Stating a problem

- Define the inputs, and what properties they must satisfy
- Define a relation between inputs and outputs, giving the properties a solution must satisfy

Example

- Searching a sequence of integers
  - Input, sequence of integers, s and search value (an integer)
  - Output, index i such that s[i]=x, if one exists, otherwise i=-1
- Searching an ordered sequence of integers
  - Input, sequence of integers, s, in ascending order and search value x (an integer), known to be in the sequence s
  - Output, index I such that s[i]=x

Problem Instances

- An instance is obtained by taking a problem and defining particular values for the inputs
- Example: sorting integers
  - Input s: {3,1,4,1,5,9,2,7}
  - Output s' = {1,1,2,3,4,5,7,9}

Algorithms

- An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as outputs
- An algorithm is thus a sequence of computational steps that transform the input into the output

An algorithm solve problems

- An algorithm may solve
  - A computational problem: To search for x in sequence s, try each index i between 1 and |s| in turn; if you find s[i]=x, output i, otherwise output -1
  - An instance of a problem: To sort the sequence {3,1,4,1,5,,9,2,7} output the new sequence {1,1,2,3,4,5,7,9}

Algorithm Properties

- **Inputs/Outputs:** There must be some inputs from the standard sets of inputs and an algorithm's execution must produce outputs
- **Definiteness:** Each must be clear and unambiguous
- **Finiteness:** Algorithms must terminate after finite time or steps
- **Effectiveness:** Every instruction must be basic enough to be carried out, in principle, by a person using only pencil and paper

Correctness

- An algorithm is correct with respect to a problem if for every instances of the inputs
  - The algorithm halts
  - The outputs produced satisfy the input/output relation or produces correct output
- Correctness must be proven
  - An incorrect algorithm might not halt at all on some input instances, or it might halt with an answer other than the desired one

Example

- ```
  Factorial (int n)
  if n = 2 then
      return 2
  return factorial(n-1)*n
  ```
- We note that even though we may check the correctness of this algorithm for an infinity of values > 2, the algorithm is still incorrect as it does not halt on the case n = 1
  ```

Expressing Algorithms

- An algorithm can be specified
  - In English (eg enrollment guide)
  - By diagrams (eg instructions for assembling a kitset furniture)
  - In a computer programming language
  - With pseudo code
  - In some other more or less formal language (eg knitting pattern)
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed

Choice of Algorithms (1/2)

- On what basis, we should choose an algorithm
- Two contradictory goals
  - We would like an algorithm that is easy to understand, code and debug
  - We would like an algorithm that makes use efficient use of the computer resources, especially, one that runs as fast as possible

Choice of Algorithms (2/2)

- Goal 1 is important when we are writing a program to be used once or a few times
  - Cost of the programmers' time will most likely exceed by far the cost of running the program
- Goal 2 is important when presented with a problem whose solution is to be used many times
  - Cost of running the program may far exceed the cost of writing it

Analyzing Algorithms

- Analyzing an algorithm generally means predicting the resources that the algorithm requires
- Some of the resource that are of concern are: memory, communication bandwidth, or computer hardware
- Most important is computational time that we want to measure
- By analyzing several candidate algorithms for a problem, a most efficient one can be easily identified

Performance Analysis

- We can analyze algorithms by determining the time and storage requirements of the algorithm
- The space complexity of a program is the amount of memory it needs to completion
- The time complexity of a program is the amount of computer time it needs to run to completion

Space Complexity

- The space needed by a program is generally seen to be the sum of the following components
  - A fixed part that is independent of the characteristics (e.g., number, size) of the inputs and outputs
  - A variable part that consists of the space needed by component variables whose size is dependent on the particular problem instance being solved
- The space requirement S(P) of any program P is therefore be written as $c + S_P$(instance characteristics) where c is a constant

Example 1

- ```
  float abc(float a, float b, float c)
  {
      return a+b+b*c+(a+b-c)/(a+b)+4.0;
  }
  ```
- Problem instance is characterized by the specific values of a, b, and c.
- S_P(instance characteristics) = 0
  - No matter the values of a, b and c, they always take same amount of memory
  ```

Example 2

```
long int factorial(int n)
{
    int i;
    long int prod=1;
    for (i=2; i<=n; i++)
        prod *= i;
    return prod;
}
```

- S_P(instance characteristics) = 0

Time Complexity

- The time complexity, T(P), by a program P is the sum of the compile time and the run time.
- The compile time does not depend on the instance characteristics
- We concern ourselves with just the run time of a program

Running Time

- The running time of an algorithm on a particular input is the number of primitive operations or steps executed
- Except for time of executing a function call most statements roughly require all the same amount of time
  - Y=mx+c
  - C=5/9*(f-32)
  - Z=f(x)+g(y)
- But we can be more exact if we need be

Factors Effecting Running Time

- The running time of a program depends on factors such as:
  - The input to the program,
  - The quality of code generated by the compiler used to create the object program,
  - The nature and speed of the instructions on the machine used to execute the program, and
  - The time complexity of the algorithm underlying the program

Program Step

- A program step is loosely defined as a syntactically or semantically meaningful segment of a program that has an execution time that is independent of the instance characteristics
- For example, the entire statement
  - return a+b+b*c+(a+b-c)/(a+b)+4.0;
  - Could be regarded as a step since its execution time is independent of instance characteristics
- Any statement that doesn't depend on instance characteristics can be regarded as a single step

Input Size

- The time taken by an algorithm to solve a particular problem depends on the input
- The algorithm can take different amounts of time to solve the problem for two different input instances.
- In general, the time taken by an algorithm grows with the size of the input
- So we describe the running time of a program as a function of the size of its input
  - Time α f(n)
  - Or equivalently,
  - Time = k * f(n) for some K that depends on the implementation

Characteristics of Input Size

- How we characterize input size depends on the problem studied
  - Sorting: number of input items
  - Multiplication: total number of bits
  - Graph algorithms: number of nodes and edges

Best, Worst and Average Case

- Best case complexity gives lower bound on the running time of the algorithm for any instance of inputs. This indicates that the algorithm can never have lower running time than best case for particular class of problems
- Worse case complexity gives upper bound on the running time of the algorithm for all instances of the inputs. This ensures that no input can overcome the running time limit posed by worst case complexity
- Average case complexity gives average number of steps required on any instance of the inputs random (equally likely) inputs, real-life inputs.

The End