Data Structures and Algorithms

Chapter 1 Introduction

Dr. Zhiqiang Liu

School of Software and Microelectronics, Northwest Polytechnical University

Outline

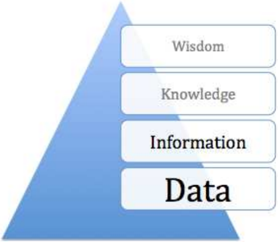
Next Section

1.1 Preliminary understanding

Symbol

1.1 Preliminary understanding

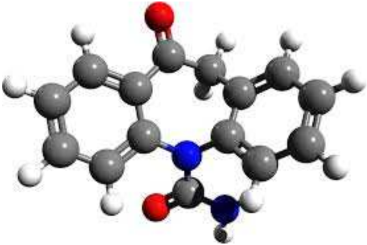
Symbol



1.1 Preliminary understanding

Symbol

Relation



1.1 Preliminary understanding

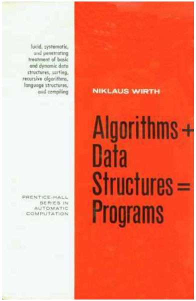
- Program: Find out the max one from n numbers

1.1 Preliminary understanding

- Program: Find out the max one from n numbers
  - How to organize and store n numbers and other temp numbers?
  - What are operations of those numbers for processing the problem?
  - How to use those operations?
  - What are steps to deal with the problem by those operations?

1.1 Preliminary understanding

- Prof. Niklaus Wirth
- http://www.inf.ethz.ch/personal/wirth/
  - Received Turing Award in 1984
  - Father of Pascal
  - Algorithms+Data Structures = Programs
    Prentice-Hall, 1976



Where are Algorithms and Data Structures in follow program

```c
/* Orignal program segment of Tiny HTTPd */
void accept_request(int client){
 char buf[1024];
 int numchars;
 char method[255];
 char url[255];
 size_t i, j;
 struct stat st;
 int cgi = 0;

 numchars = get_line(client, buf, sizeof(buf));
 i = 0; j = 0;
 while (!ISspace(buf[j]) && (i < sizeof(method) - 1))
 {
  method[i] = buf[j]; i++; j++;
 }

 i = 0;
 while (ISspace(buf[j]) && (j < sizeof(buf)))
  j++;
 ... ...
```

Next Section

Next Subsection

1.2.1 The Contents of the course

- We have 12 weeks to learn fundamental data structures and algorithms for organizing and processing information
- Deeply understand the basic structures used in all software

1.2.1 The Contents of the course

- We have 12 weeks to learn fundamental data structures and algorithms for organizing and processing information
- Deeply understand the basic structures used in all software
  - Understand the data structures and their trade-offs
  - Rigorously analyze the algorithms that use them (math!)

1.2.1 The Contents of the course

- We have 12 weeks to learn fundamental data structures and algorithms for organizing and processing information
- Deeply understand the basic structures used in all software
  - Understand the data structures and their trade-offs
  - Rigorously analyze the algorithms that use them (math!)
- You must learn this course, if you would like to find the answer of following questions:
  - How does Google quickly find web pages that contain a search term?
  - What's the fastest way to broadcast a message to a network of computers?
  - How can a subsequence of DNA be quickly found within the genome?
  - How does your operating system track which memory (disk or RAM) is free?
  - In the game Half-Life, how can the computer determine which parts of the scene are visible?
  - ......

Next Subsection

1.2.2 Data Structure Philosophy

- Any organization for a collection of data can be searched, processed in any order, or modified.

1.2.2 Data Structure Philosophy

- Any organization for a collection of data can be searched, processed in any order, or modified.
- Each data structure has costs and benefits.
- Rarely is one data structure better than another in all situations.

1.2.2 Data Structure Philosophy

- Any organization for a collection of data can be searched, processed in any order, or modified.
- Each data structure has costs and benefits.
- Rarely is one data structure better than another in all situations.
- A data structure requires:
  - space for each data item it stores,
  - time to perform each basic operation,
  - programming effort.

1.2.2 Data Structure Philosophy

- Any organization for a collection of data can be searched, processed in any order, or modified.
- Each data structure has costs and benefits.
- Rarely is one data structure better than another in all situations.
- A data structure requires:
  - space for each data item it stores,
  - time to perform each basic operation,
  - programming effort.
- Each problem has constraints on available space and time.
- Only after a careful analysis of problem characteristics can we know the best data structure for the task.

1.2.2 Data Structure Philosophy

How to select a Data structures?

- Analyze the problem to determine the resource constraints a solution must meet.
- Determine the basic operations that must be supported.
- Quantify the resource constraints for each operation.
- Select the data structure that best meets these requirements.

Next Subsection

1.2.3 Purpose of Data Structure and Algorithms

- Data structures organize data => more efficient programs.

1.2.3 Purpose of Data Structure and Algorithms

- Data structures organize data => more efficient programs.
- The choice of data structure and algorithm can make the difference between a program running in a few seconds or many days.

1.2.3 Purpose of Data Structure and Algorithms

- Data structures organize data => more efficient programs.
- The choice of data structure and algorithm can make the difference between a program running in a few seconds or many days.
- For the correct representation and use of data, it is extremely important to pick the right data-structure.

1.2.3 Purpose of Data Structure and Algorithms

- Data structures organize data => more efficient programs.
- The choice of data structure and algorithm can make the difference between a program running in a few seconds or many days.
- For the correct representation and use of data, it is extremely important to pick the right data-structure.
- It is fundamental and important for the computer science people, especially for software engineering people!
- Be able to make good design choices as a developer, project manager, etc.
- Be able to justify and communicate your design decisions

1.2.3 Purpose of Data Structure and Algorithms

- Data structures organize data => more efficient programs.
- The choice of data structure and algorithm can make the difference between a program running in a few seconds or many days.
- For the correct representation and use of data, it is extremely important to pick the right data-structure.
- It is fundamental and important for the computer science people, especially for software engineering people!
- Be able to make good design choices as a developer, project manager, etc.
- Be able to justify and communicate your design decisions
- Must be included in post-graduate entrance examination (computer related)

Next Section

Next Subsection

What is Data Structure?

Preliminary Concepts

Data

- The difference between Data (Raw) and Information (Processed)
- (In computer) Data is carrier of Information, is symbols inputted, stored, processed and outputted by computers

Curriculum	Course name	Period
024020	Data Structure	64
024024	Operating System	48
024026	Database Theory	48

What is Data Structure?

Preliminary Concepts

Data

- The difference between Data (Raw) and Information (Processed)
- (In computer) Data is carrier of Information, is symbols inputted, stored, processed and outputted by computers

Data Element

Data Element is basic unit of Data, a member of Data

Curriculum	Course name	Period
024020	Data Structure	64
024024	Operating System	48
024026	Database Theory	48

What is Data Structure?

Preliminary Concepts

Data

- The difference between Data (Raw) and Information (Processed)
- (In computer) Data is carrier of Information, is symbols inputted, stored, processed and outputted by computers

Data Element

Data Element is basic unit of Data, a member of Data

Data Object

A SET containing the same type of data elements

Curriculum	Course name	Period
024020	Data Structure	64
024024	Operating System	48
024026	Database Theory	48

1.3.1 Data Structure

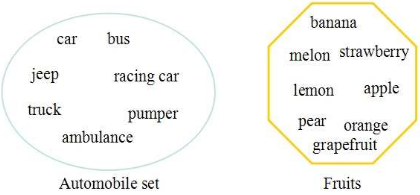
Data Structure

- A data object + specific existing relations between elements
- Set + Relations (Constraint), Ordered pair
- Four basic relations in Data Structure

1.3.1 Data Structure

Data Structure

- A data object + specific existing relations between elements
- Set + Relations (Constraint), Ordered pair
- Four basic relations in Data Structure
  - Sets: Uniform type, No specific relations between elements



1.3.1 Data Structure

Data Structure

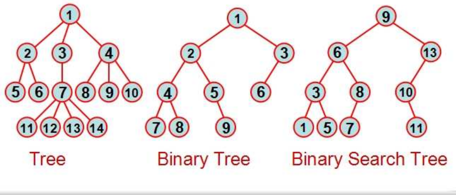
- A data object + specific existing relations between elements
- Set + Relations (Constraint), Ordered pair
- Four basic relations in Data Structure
  - Sets: Uniform type, No specific relations between elements
  - Linear structure: One to one

Curriculum	Course name	Period
024020	Data Structure	64
024024	Operating System	48
024026	Database Theory	48

1.3.1 Data Structure

Data Structure

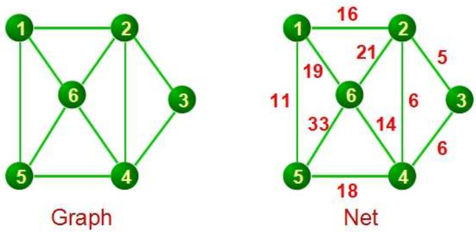
- A data object + specific existing relations between elements
- Set + Relations (Constraint), Ordered pair
- Four basic relations in Data Structure
  - Sets: Uniform type, No specific relations between elements
  - Linear structure: One to one
  - Hierarchical Structure: One to many



1.3.1 Data Structure

Data Structure

- A data object + specific existing relations between elements
- Set + Relations (Constraint), Ordered pair
- Four basic relations in Data Structure
  - Sets: Uniform type, No specific relations between elements
  - Linear structure: One to one
  - Hierarchical Structure: One to many
  - Graph Structure: Many to many



Logical Form and Physical Form

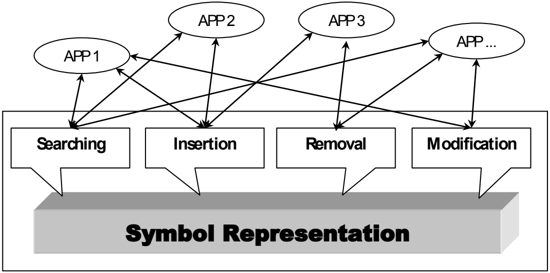
Data structure have

- Logical Form
  - Definition of the data item within an Abstract Data Type (ADT)
  - Problem orient: for example, Integers in mathematical sense: +, -
- Physical Form
  - Implementation of the data item within a data structure.
  - Computer orient: for example, 16/32 bit integers, overflow, Storage Structure in computer
    $(123)_{10} = (1111011)_2$

Next Subsection

1.3.2 Abstract Data Type (ADT)

- Abstract Data Type (ADT): An abstract data type is defined as "class of objects whose logical behavior is defined by a set of values and a set of operations"
  - Encapsulation: Each ADT operation is defined by its inputs and outputs (Interface), Hide implementation details.
  - Abstract: Extract problem essential.
  - Data objects, Relations, Functions (Special relations)



ADT of Natural Number

```
ADT NaturalNumber is
{
Objects: An ordered subset of integers, beginning at 0, ending
    at the maximum (MaxInt).

Relation: C={<i, i+1>| i=0, 1, ..., MaxInt}

Function:  For any x, y belong NaturalNumber;
    False, True belong Boolean, +,-,<,==,= are all available
        properties and methods
Zero( ) :NaturalNumber      return 0
IsZero(x) :Boolean  if (x==0) return True else return False
Add (x, y) :NaturalNumber
      if (x+y<=MaxInt) return x+y  else return MaxInt
Subtract (x, y) :NaturalNumber
      if (x < y) return 0 else return x - y
Equal (x, y) :Boolean
      if (x==y) return True else return False
Successor (x) :NaturalNumber
      if (x==MaxInt) return x else  return x+1
} ADT NaturalNumber
```

ADT of Queue

```
ADT Queue is
{
Objects:

Relation:

Function:


} ADT NaturalNumber
```

Data Type vs ADT

- A data type consists of the values it represents and the operations defined upon it.
- A data type is the physical implementation of an ADT.
  - Each operation associated with the ADT is implemented by one or more subroutines in the implementation
- Data type usually refers to an organization for data in main memory.

Next Subsection

What is Algorithm?

Algorithm

- Is n(n>2) prime number?

What is Algorithm?

Algorithm

- Is n(n>2) prime number?
- An algorithm is a finite set of instructions that, if followed, accomplishes a particular task.
- In addition, all algorithms must satisfy the following characteristic:
  - Input: zero, one, or more
  - Output: at least one output
  - Definiteness: each instruction is clear and unambiguous
  - Effectiveness: every instruction must be basic enough to be carried out, in principle, by a person using only pencil and paper
  - Finiteness: terminates after a finite number of steps
- In computational theory
  - Main distinguish between an algorithm and a program is the characteristic Finiteness.

Algorithm Description

- Using a natural language
- Using flow diagram
- Using a programming language
- Combine with the above

Is n(n>2) prime number?

Next Section

Next Subsection

1.4.1 Algorithm Criterion

- There are often many algorithms to solve a problem. How do we choose between them?

1.4.1 Algorithm Criterion

- There are often many algorithms to solve a problem. How do we choose between them?
- Algorithm Criterion
  - Validity （正确性）
  - Usability（可用性）
  - Readability（可读性）
  - Robustness（健壮性）
  - Efficiency（效率）

1.4.1 Algorithm Criterion

- There are often many approaches (algorithms) to solve a problem. How do we choose between them?
- At the heart of computer, program design are two (sometimes conflicting) goals.
  - (1) To design an algorithm that is easy to understand, code, and debug.
  - (2) To design an algorithm that makes efficient use of the computer's resources.

1.4.1 Algorithm Criterion

- There are often many approaches (algorithms) to solve a problem. How do we choose between them?
- At the heart of computer, program design are two (sometimes conflicting) goals.
  - (1) To design an algorithm that is easy to understand, code, and debug.
  - (2) To design an algorithm that makes efficient use of the computer's resources.
- Goal (1) is the concern of Software Engineering
- Goal (2) is the concern of data structures and algorithm analysis.

1.4.1 Algorithm Criterion

- There are often many approaches (algorithms) to solve a problem. How do we choose between them?
- At the heart of computer, program design are two (sometimes conflicting) goals.
  - (1) To design an algorithm that is easy to understand, code, and debug.
  - (2) To design an algorithm that makes efficient use of the computer's resources.
- Goal (1) is the concern of Software Engineering
- Goal (2) is the concern of data structures and algorithm analysis.
- When goal (2) is important, how do we measure an algorithm's cost?

Next Subsection

Algorithm Efficiency: How to Measure the Efficiency?

- (1) Empirical comparison (run programs)

Algorithm Efficiency: How to Measure the Efficiency?

- (1) Empirical comparison (run programs)
- (2) Asymptotic Algorithm Analysis (before run)

Algorithm Efficiency: How to Measure the Efficiency?

- (1) Empirical comparison (run programs)
- (2) Asymptotic Algorithm Analysis (before run)
    - Critical resources;

Algorithm Efficiency: How to Measure the Efficiency?

- (1) Empirical comparison (run programs)
- (2) Asymptotic Algorithm Analysis (before run)
  - Critical resources;
  - Factors affecting running time;
    - Strategy of algorithm
    - Scale of problem
    - Program language
    - Compiler
    - Computer speed

Algorithm Efficiency: How to Measure the Efficiency?

- (1) Empirical comparison (run programs)
- (2) Asymptotic Algorithm Analysis (before run)
    - Critical resources;
    - Factors affecting running time;
        - Strategy of algorithm
        - Scale of problem
        - Program language
        - Compiler
        - Computer speed
    - For most algorithms, running time depends on "size" of the input.
    - Running time is expressed as T(n) for some function T on input size n.

1.4.2 Algorithm Efficiency

- Algorithm = control structure + meta operation

```
int largest(int array[],int n)
{
    int currlarge=0; //Largest value seen
    for (int i=1; i<n; i++) //For each val
    {
        if(array[currlarge]<array[i])
            currlarge=i;//Remember pos
    }
    return currlarge;//Return largest
}
```

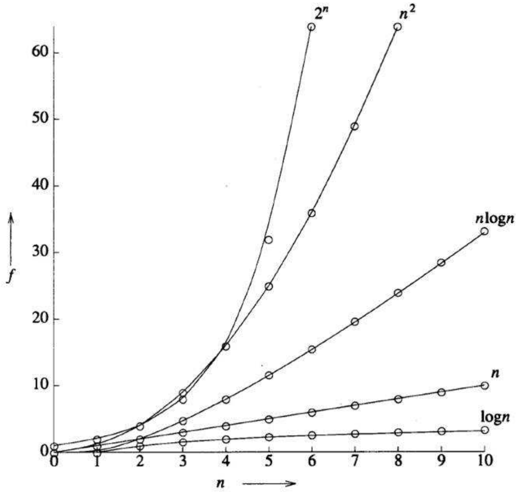
$T(n) = C_1 n + C_2$ steps

compute n squared

```
//compute n squared
int n2(int n)
{
    sum=0;
    for(i=1;i<=n;i++)
        for(j=1;j<n;j++)
            sum++;
    return sum;
}
```

$$T(n) = C_1 n^2 + C_2$$

1.4.2 Algorithm Efficiency



Grow Rate of Functions

1.4.2 Algorithm Efficiency

Growth Rates Compared

	n=1	n=2	n=4	n=8	n=16	n=32
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296

$$f(n) = n^2 + 100n + \log_2 n + 1000$$

Asymptotic Analysis: Big-O

- Running time of the algorithm takes to execute depends on the value of N
  asymptotic complexity
  Time Complexity $T(n) = O(f(n))$
  Space Complexity $S(n) = O(f(n))$

Asymptotic Analysis: Big-O

- Running time of the algorithm takes to execute depends on the value of N asymptotic complexity
  Time Complexity $T(n) = O(f(n))$
  Space Complexity $S(n) = O(f(n))$

- Definition:
  f(n) is O(g(n)) if there exist positive numbers c and N such that $f(n) \leq cg(n)$ for all $n > N$

- Usage:
  The algorithm is in $O(n^2)$ in **[best, average, worst]** case.

Best, Worst, Average Cases

Not all inputs of a given size take the same time to run.

- Sequential search for K in an array of n integers:
- Begin at first element in array and look at each element in turn until K is found
  - Best case: 1 time of comparison
  - Worst case: n times of comparison
  - Average case: $(n+1)/2$

```c
int lookup(int k, int array[], int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        if (k == array[i])
        {   return k; }
    }
    return -1;
}
```

1.4.2 Algorithm Efficiency

Which is fairest? Which is most important? it may be difficult to determine!

A Common Misunderstanding

- "The best case for my algorithm is $n = 1$ because that is the fastest." **WRONG!**
- Big-O refers to a growth rate as n grows to ∞.
- Best case is defined as which input of size n is cheapest among all inputs of size n.

Next Subsection

1.4.3 Big-O Examples

Example 1:

```
a = b;
```

This assignment takes constant time, so it is $T(n) = c$. We say this is in $O(1)$.

1.4.3 Big-O Examples

Example 1:

```
a = b;
```

This assignment takes constant time, so it is T(n) = c. We say this is in O(1).

Example 2: $T(n) = c_1 n^2 + c_2 n$ in average case.

$c_1 n^2 + c_2 n \leq c_1 n^2 + c_2 n^2 \leq (c_1 + c_2) n^2$ for all $n > 1$.

$T(n) \leq cn^2$ for $c = c_1 + c_2$ and $n_0 = 1$.

Therefore, T(n) is in $O(n^2)$ by the definition.

Simplifying Rules

- If $f(n)$ is in $O(g(n))$ and $g(n)$ is in $O(h(n))$, then $f(n)$ is in $O(h(n))$.
- If $f(n)$ is in $O(kg(n))$ for any constant $k > 0$, then $f(n)$ is in $O(g(n))$.
- If $f1(n)$ is in $O(g1(n))$ and $f2(n)$ is in $O(g2(n))$, then $(f1 + f2)(n)$ is in $O(max(g1(n), g2(n)))$.
- If $f1(n)$ is in $O(g1(n))$ and $f2(n)$ is in $O(g2(n))$ then $f1(n)f2(n)$ is in $O(g1(n)g2(n))$.

1.4.3 Big-O Examples

Example 3:

```
sum = 0;
for (i=1; i<=n; i++)
  sum += n;
```

1.4.3 Big-O Examples

Example 3:

```
sum = 0;
for (i=1; i<=n; i++)
  sum += n;
```

$O(n)$.

1.4.3 Big-O Examples

Example 4:

```
sum = 0;
for (j=1; j<=n; j++)
  for (i=1; i<=j; i++)
    sum++;
for (k=0; k<n; k++)
  A[k] = k;
```

1.4.3 Big-O Examples

Example 4:

```
sum = 0;
for (j=1; j<=n; j++)
  for (i=1; i<=j; i++)
    sum++;
for (k=0; k<n; k++)
  A[k] = k;
```

$O(n^2)$.

1.4.3 Big-O Examples

Example 5:

```
sum1 = 0;
for (i=1; i<=n; i++)
  for (j=1; j<=n; j++)
    sum1++;

sum2 = 0;
for (i=1; i<=n; i++)
  for (j=1; j<=i; j++)
    sum2++;
```

1.4.3 Big-O Examples

Example 5:

```
sum1 = 0;
for (i=1; i<=n; i++)
  for (j=1; j<=n; j++)
    sum1++;

sum2 = 0;
for (i=1; i<=n; i++)
  for (j=1; j<=i; j++)
    sum2++;
```

$O(n^2)$.

1.4.3 Big-O Examples

Example 5:

```
sum1 = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=n; j++)
    sum1++;

sum2 = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=k; j++)
    sum2++;
```

1.4.3 Big-O Examples

Example 5:

```
sum1 = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=n; j++)
    sum1++;

sum2 = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=k; j++)
    sum2++;
```

First loop is Σn for $k = 1$ to $\log n, O(n \log n)$.
Second loop is $\Sigma 2^k$ for $k = 0$ to $\log(n-1), O(n)$.

1.4.3 Big-O Examples

Example 5:

```
sum1 = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=n; j++)
    sum1++;

sum2 = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=k; j++)
    sum2++;
```

First loop is Σn for $k = 1$ to $\log n, O(n \log n)$.

Second loop is $\Sigma 2^k$ for $k = 0$ to $\log(n-1), O(n)$.

$O(n \log_2 n)$.

Space Bounds

Space bounds can also be analyzed with asymptotic complexity analysis.

Time: Algorithm

Space: Data Structure

Next Subsection

1.4.4 Space and Time Tradeoff Principle

- One can often reduce time if one is willing to sacrifice space, or vice versa
  - Table lookup
    Factorials
    Cryptography

1.4.4 Space and Time Tradeoff Principle

- One can often reduce time if one is willing to sacrifice space, or vice versa
  - Table lookup
    Factorials
    Cryptography
- Disk-based Space/Time Tradeoff Principle:
  The smaller you make the disk storage requirements, the faster your program will run.

Next Section

1.5 Brief Summary

1. Preliminary understanding
2. Introduction of the course
   - The Contents of the course
   - Data Structure Philosophy
   - Purpose of Data Structure and Algorithms
3. Related Concepts
   - Data Structure
   - Abstract Data Type (ADT)
   - Algorithm
4. Algorithm Efficiency and Analysis
   - Algorithm Criterion
   - Algorithm Efficiency
   - Big-O Examples
   - Space and Time Tradeoff Principle
5. Brief Summary