Introduction to Computer Science
Algorithms

Dr. Antonio Siochi

Outline

- The Concept of an Algorithm
- Algorithm Representation
- Algorithm Discovery
- Iterative Structures
- Recursive Structures
- Efficiency

Definition of an Algorithm

THE FIVE CHARACTERISTICS
1. Finite sequence of instructions
2. Each instruction is unambiguous
3. Each instruction is executable
4. The process terminates for all inputs
5. Some output or result is produced

Example 1

Task: find largest positive integer
step 1: make a list of all the positive integers
step 2: arrange this list in order
step 3: copy the first integer in this list
step 4: stop
NOT an algorithm because
- step 2 is ambiguous: what order?
- step 1 is not executable
  - although it has finite steps, it does not terminate!

Example 2

Task: take exam
step 1: read unanswered question
step 2: if you know the answer, write it down
step 3: go back to step 1 if there are still unanswered questions
step 4: stop

There is a problem with this set of instructions!
Abstract Nature of Algorithms

Distinction between “algorithm” and its “representation”
- algorithm: “underlying concept”
- representation: way of communicating the algorithm

Further distinction:
- program: formal representation of an algorithm
- process: activity of executing the program

Outline

- The Concept of an Algorithm
- Algorithm Representation
  - Algorithm Discovery
  - Iterative Structures
  - Recursive Structures
- Efficiency

Representation

- algorithm is an abstraction of a process
- need to be able to tell other people (or a computer) how to do it
- Write it down!
- syntax – rules for how to write it down
- semantics – rules for determining meaning

Syntax

- In English, the subject comes before the predicate: “You ate lunch.”
- violate the rule: “Ate you lunch”
- syntax can help make semantics clear or confused
- syntax can use words or pictures as primitive elements

Example: Folding a bird
Primitives

- well-defined set of building blocks
- must be unambiguous
- appropriate level of detail
- can be combined with other primitives to form “higher-level” primitives
- machine language primitives will work, however, level of detail is inappropriate for most problems

Origami Primitives

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn paper over as in</td>
<td></td>
</tr>
<tr>
<td>Shade one side of paper as in</td>
<td></td>
</tr>
<tr>
<td>Distinguishes between different sides of paper</td>
<td></td>
</tr>
<tr>
<td>Represents a valley fold so that</td>
<td></td>
</tr>
</tbody>
</table>

Representations

- programming languages, ex. C++ (implement)
- pseudo-code, i.e., English-like statements with a special simple syntax (design)

If it is cold, wear a jacket, otherwise a T-shirt is fine. Depending on the temperature, wear a jacket or not.

```plaintext
if (temp less than 70) then (wear jacket)
else (wear T-shirt)
```

Like writing an outline or taking notes – use it to develop understanding of problem

Pseudocode

- sequential statements (execute statements from top to bottom)
- conditional statements (selection)
  ```plaintext
  if (condition) activity 1
  else activity 2
  ```
- iteration statements (loops)
  ```plaintext
  while (condition) {
  activity
  }
  ```

Example 1

task: take exam

get exam from teacher
write name on exam
sign pledge

```plaintext
while (there is time left) {
  look for an unanswered question
  if (you know answer) write it down
  else
    skip this question
}
```

Example 2

task: convert a decimal to two’s complement
Example 2

Task: Convert a decimal to two's complement

Let X be the decimal.

If (X is positive) {
    write binary equivalent
}
else {
    write binary equivalent
    complement it
    increment it
}
print the result

Modular Design

- Ease of comprehension and modification
- Organize/structure the algorithm into major sections or modules
- Similar in concept to using an outline to write a paper
- Each module can be considered an algorithm as well
- Modules provide different levels of detail

Pseudocode vs. Modules

- Pseudocode is a way of representing algorithms
- Other ways of representing algorithms:
  - Natural language (e.g., English)
  - Pictures, diagrams
  - Programming languages
- Modules are a way of organizing an algorithm

Procedures

- How to represent a module
- Define the module – give a module a name
- Refer to the module – instead of writing out all steps of that module, just give the name of the module
- Pseudocode: procedure nameOfModule

Example

- Refer to module brushTeeth from 2 other modules

  procedure getReadyForSchool
  wakeUp
  brushTeeth
  eatBreakfast

  procedure getReadyForBed
  eatDinner
  brushTeeth
  turn off lights

Definition of module

procedure brushTeeth
  go to bathroom
  grab toothbrush
  open toothpaste tube
  put brush bristles under tube opening
  while (not enough paste on brush)
    squeeze tube
  put brush in mouth
  replace tube cap
  while (2 minutes not past)
    move brush up and down, side to side
  rinse mouth
### Considerations

- Two things you can do to a module:
  - use it
  - define it
- You can use a module before it’s defined, but the algorithm is NOT COMPLETE until the module is defined
- Modules can be used several times without having to define them over and over.

### Parameters

- modules perform actions upon certain objects
  - example: brushTeeth
  - action is brushing
  - IMPLIED object is your teeth
- we can specify exactly WHAT object receives the action
  - example:
    - brush your teeth = brushTeeth(your)
    - brush John’s teeth = brushTeeth(John’s)
- a parameter is the object that receives the action

### Parameters for procedure eat

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>food</td>
<td>eat(food)</td>
</tr>
<tr>
<td>dinner</td>
<td>eat(dinner)</td>
</tr>
<tr>
<td>you</td>
<td>eat(you)</td>
</tr>
<tr>
<td>when</td>
<td>eat(at9)</td>
</tr>
<tr>
<td>eat(you, bigmac, at9)</td>
<td>Procedure: eat(who, what, when) { who will eat what when }</td>
</tr>
</tbody>
</table>

### Examples

- define a procedure to print the sum of two numbers
  ```plaintext
  procedure printSum(number1, number2){
    print(number1 + number2)
  }
  ```
- use it with different actual parameters
  - HOW DO WE DO THIS ???

### How to define a procedure

- Name of procedure (abstraction of body)
- Parameters
- Body
  ```plaintext
  procedure printSum(number1, number2){
    print(number1 + number2)
  }
  ```
How to write pseudocode?

- what are our primitives in pseudocode?
  - anything we need. we just make it up.
- so I can write “brushHair” and that’s my algorithm?
  - only if brushHair is executable on your target machine.
  - if it’s not, you need to treat brushHair like a procedure, and define it in more specific terms.

Outline

- The Concept of an Algorithm
- Algorithm Representation
- Algorithm Discovery
- Iterative Structures
- Recursive Structures
- Efficiency

The Art of Problem Solving

- an algorithm is a precise specification of how to solve a problem, but how do I solve problems?
- problem solving is a skill that is developed and learned
- there is NO step by step recipe for solving problems
- the key is PRACTICE !!!

Polya’s Phases

- Understand the problem !!!
- Get an idea about what kind of algorithm is needed
- Create the algorithm and implement it (program)
- Evaluate the program for accuracy and general use

Phases, NOT steps

- not carried out in sequence typically
- depend on our experience, creativity, and imagination
- to understand a problem, sometimes we need to implement an algorithm (prototype)
- problem solving is part of software engineering

Problem 1

Person A is charged with the task of determining the ages of person B’s three children. B tells A that the product of the children’s ages is 36. After considering this clue, A replies that another clue is required, so B tell A the sum of the children’s ages. Again, A replies that another clue is needed, so B tells A that the oldest child plays the piano. After hearing this clue, A tells B the ages of the three children.

How old are the three children?
"Getting a Foot in the Door"

- the desk drawer approach
- solve only part of a problem first
- work backwards from a solution
- study solutions to similar problems
- solve specific cases first, then generalize
- stepwise refinement (modular approach)

Stepwise Refinement

- natural fit with modular representation approach
- allows team approach
- actually a means of organizing information
- appropriate in well-established domains (e.g., data processing)

Problem 2

Before A, B, C, and D ran a race they made the following predictions:

A predicted that B would win.
B predicted that D would be last.
C predicted that A would be third.
D predicted that A’s prediction would be correct.

Only one of these predictions was true, and this was the prediction made by the winner.

In what order did A, B, C, and D finish the race?

Outline

- The Concept of an Algorithm
- Algorithm Representation
- Algorithm Discovery
- Iterative Structures
- Recursive Structures
- Efficiency

Iteration: Search

- iteration involves repeating some action until a set of conditions has occurred
- searching is a process of examining objects from a list of objects, until you find the one that you are looking for
- iteration is applicable to the search process

Sequential Search

- problem: given a list of objects, and an object to search for, determine if the named object is in the list of objects (this is the WHAT)
- one method is to examine each object in the list sequentially, until either we find the object we're looking for, or we look through everything and see that the object is not there (this is the HOW)
**Pseudocode**

Given: a list and an object  
select the first element in the list  
while (selected element is not the object)  
  select the next element in the list

- problem: does this terminate?  
- what happens if we run out of elements?  
- how do we handle the case where the object is NOT in the list?

**Refinement**

procedure sequentialSearch(object, list) {
  select first object in list  
  while ((there are unselected objects in list) AND  
    (selected element is not the object) )  
    select the next element in the list
  
  if (we found object)  
    print("found it")  
  else  
    print("not there")
}

**Refinement 2**

procedure sequentialSearch(object, list) {
  item = firstObjectIn(list)  
  while (! isLast(item) && !(item == object) )  
    item = nextObjectIn(list)  
  
  if (item == object)  
    print("found it")  
  else  
    print("not there")
}

- what if the object were the last item?

**Refinement 3**

procedure sequentialSearch(thing, list) {
  // item is local to this procedure  
  item = list.myFirstThing()  
  while (! item.isLast() && !(item == thing) )  
    item = list.myNextThing()  
  
  if (item == thing)  
    print("found it")  
  else  
    print("not there")
}

- Object syntax

**Components of Loop**

- initialize: set up the conditions, which the loop tests  
- test: compare current loop condition to the stopping condition and terminate if they match  
- modify: change the conditions so that the loop will eventually terminate

**while loop**

Test condition
Condition true  
Condition false  
while(condition)  
Activity
Outline

- The Concept of an Algorithm
- Algorithm Representation
- Algorithm Discovery
- Iterative Structures
- Recursive Structures
- Efficiency

Binary Search

A Binary Search algorithm searches a sorted list by examining the middle item. If the item is not found, the appropriate half of the list is searched again (using the Binary Search algorithm). Each time only half of the remaining list is searched, until the item is found.

Binary Search Algorithm

```
procedure search (List, TargetValue) {
    if (List isEmpty)
        report that TargetValue is not in list
    else {
        TestEntry = middle item of List
        do the appropriate case below:
        case: TargetValue == TestEntry
            report successful find
        case: TargetValue < TestEntry
            search portion of List BEFORE TestEntry
        case: TargetValue > TestEntry
            search portion of List AFTER TestEntry
    }
}
```

Binary Search Example

Searching for "John"

Algorithm Efficiency

If an algorithm does not process information efficiently:
- CPU resources are wasted
- Poor response time may result
- Possibly the problem cannot be solved in your lifetime
Sequential Search

For the sequential search (previously discussed), the time to find an item in the list is proportional to the size of the list divided by 2. This makes sense because some items (close to the front of the list) are found quickly but other items near the end of the list take more time to find. So on average:

\[ \text{time} \sim \frac{N}{2} \]

where \( N \) is the number of items in the list.

---

Binary Search

A Binary Search algorithm searches a sorted list by examining the middle item. If the item is not found, the appropriate half of the list is searched again (using the Binary Search algorithm). Each time only half of the remaining list is searched, until the item is found.

The time to perform a binary search is proportional to the logarithm of the size of the list:

\[ \text{time} \sim \log_2(N) \]

where \( N \) is the number of items in the list.

---

Growth Rates

Importance of Efficiency

Towers of Hanoi

64 Golden Rings on 3 diamond pegs. Priests in India must move the rings from one peg to another, but they can move only one ring at a time and they cannot put a larger ring on top of a smaller one. Legend has it, when all the disks have been moved, the world will end. This problem has a solution that grows exponentially:

\[ \text{time} \sim 2^N \]

---

When will the world end?

Assuming the priest can move one ring per second. It will take:

\[ 2^{64} = 18,446,744,073,709,551,616 \text{ seconds} \]

or approximately,

585 Billion years.

For perspective, the universe is only 15 Billion years old.