Algorithms
2/13/01 Lecture #4 16.070

• Today's lecture is about Design -- cDio

• Definition: An **algorithm** is a sequence of statements to be executed in order, to produce some desired result. Alternately, it is a finite set of rules which gives a sequence of operations for solving a specific type of problem.

• Examples
  ➢ Cookbook Recipe
  ➢ Steps in a surgical procedure
  ➢ Instructions to build a model airplane
Reasons for writing Algorithms

- To define the nature of a program; i.e., what function it performs.
- A summary or guide to writing the program.
- Allows you to concentrate on logic and organization vs how to express the ideas in a computer language.
- After program is coded, programmer can use summary as check for completeness.
- Provides common medium for porting programs from one machine to another.
Features of Algorithms

• Algorithms are written for people to read, not for computers

• An algorithm should not be confused with a Program which is:
  ➢ A translation of an algorithm into a programming language
  ➢ A specific implementation (cdIo) of an algorithm

• Algorithm features (use as a checklist!)
  ➢ Finite - Terminate after a finite number of steps and in finite time
  ➢ Definiteness, Unambiguous - Each step/action is precisely defined
  ➢ Input/Output - Has zero or more inputs, one or more outputs
  ➢ Generality - e.g., convert input number of inches to centimeters, vs printing "One inch is 2.54 centimeters"
  ➢ Effectiveness/Correctness - Operations must be sufficiently basic/correct
Approaches to Representing Algorithms

• **Pseudo-Code** (Section 2.6 in "C" book)
  - English-like representation of logic
  - Easy translation from pseudo-code into program. Pseudo-code can become comments
  - Can be applied to any program

• **Flow Charts** (Section 2.6 in "C" book)
  - Visual representation logic flow showing required operations
  - Emphasizes flow of events. Easier to review by peers
  - Appropriate for designs that follow logical sequence of events

• **Finite State Automata** (Section 5.3 in "R" book)
  - Visual representation showing available states and state transitions
  - Emphasizes states and state changes
  - Used when system reacts to environment/conditions -- e.g., controller code
Example Lander Problem

- Define an algorithm to land a craft on a planet. As the pilot, you have available to you a velocity reading and a distance measurement. You also have control of thrusters to slow your descent. Do not land too hard: landing velocity should be below 2m/s.
Features of the Algorithm

• Finiteness
• Definiteness
• Input
  ➢
  ➢
• Output
  ➢
• Effectiveness
How to Write Pseudo-Code ("C" p. 35)

- Use a separate line for each logical step
- Each line/step is identified by a number
- Each step begins with a summary phrase in brackets which sums up principal content of the step (not rigorously followed in book, but good practice)
- After summary phrase, describe action to be performed or decision to be made
- Steps are executed in order, unless otherwise specified
- Once completed, review and work through the pseudo-code to ensure correctness
Pseudo-Code of the Example Lander Problem

- Write an English-like description of the steps needed to solve the Example Lander problem
  
  1. [Define distance variable] Let \( d \) represent distance between craft and surface of planet
  
  2. [Define velocity variable] Let \( v \) represent velocity of craft approaching the surface
  
  3. [Input] Read distance \( d \).
  
  4. [Check to see if landed] If \( d = 0 \), terminate; craft has landed.
  
  5. [Input] Read velocity \( v \).
  
  6. [Check velocity and slow descent] If \( v > 2 \text{ m/s} \), apply thrust.
  
  7. [Repeat until landed] Go back to step 3.

- Notice format: step numbers, intro phrase in brackets, description in words and symbols of action to be performed or decision to be made
How to Draw Flow Charts

• Explicitly show flow of logic: operations to be performed, conditions to be checked, branches to be taken

• Symbols representing types of operations performed by computers

• Details of statements written inside shapes (e.g., summary phrase from pseudo-code)

• Flow lines show order statements are executed
**Flow Chart Example**

- Draw a Flow Chart of the steps needed to solve the Example Lander Problem
  - Identify Starting point
  - Identify I/O
  - Identify logical flow of events
  - Identify conditions to check -- decision points
  - Identify stopping point(s)

![Flow Chart Diagram]

1. Start
2. Read distance
   - If distance is greater than 0:
     - Read velocity
       - If velocity is greater than 2 m/s:
         - Apply thrust
         - Stop
       - If velocity is less than or equal to 2 m/s:
         - Stop
   - If distance is less than or equal to 0:
     - Stop
Example of Flow Chart from Chandra X-ray Observatory
Finite State Automata (FSA)

• Also known as Finite State Machine, Finite State System
• A mathematical model of a system, with discrete states explicitly represented
• The discrete inputs are also explicitly represented, along with the resulting state transition
• The state of the system summarizes the information of past inputs that is needed to determine the behavior of the system on subsequent inputs

FSAs are widely used in the specification of systems that are state-driven
State Transition Diagrams

• A Diagram of a Finite State Automata is often called a State Transition Diagram

• A Moore finite state automaton is a state transition diagram which consists of
  ➢ finite set of states depicted as circles, one initial (marked with input arrow) and one or more final (marked with halo)
  ➢ Alphabet of symbols representing inputs/condition that cause system to transition to a different state
  ➢ Arcs/arrows denote transitions from state to state, and are labeled by the input/condition that causes the transition
Finite State Automata expressed in Tabular form:

<table>
<thead>
<tr>
<th>State:</th>
<th>Input:</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td></td>
<td>q₀</td>
<td>q₁</td>
</tr>
<tr>
<td>q₁</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>
HETE State Transition Diagram

0 Wait
- delay < thresh
  - delay > thresh

1 Detumble
- XZ momentum < thresh
  - XZ momentum > thresh
- deltumble delay < thresh || XZ momentum > thresh

2 Spin Up
- XZ momentum > thresh
  - delay < thresh || Δang_mom > thresh || orbit_day=0
- Δang_mom < thresh && orbit_day=1

3 Reorient
- Δang_mom > t || orbit_day=0
  - orbit_day=0 && body_rate_magn>thresh
  - orb_day=0
  - orb_day=0 && tracking=0

4 Deploy Wheel
- default - sin(el)>thresh && wheel_rate>thresh
- delay < thresh

5 Acquisition
- sin(el)>t
- paddle_deploy && opt tracking
- delay<thresh || sin(az)>thresh

6 Deploy Paddles
- sin(az)>t
- paddle_deploy=1 || delay>thresh || ROM = deploy

7 Orbit_Day
- orbit_day=0 && sin(az)<t
- orbit_day=1 && (sin(az)>thresh || sin(el)>thresh) or orbit_day=1 && tracking=0
- default || delay<thresh

8 Orbit_Night
- default
- orbit_day=1 && (sin(az)>thresh || sin(el)>thresh) or orbit_day=1 && tracking=0
- tracking=1

- deltumble delay < thresh || sin(az)>thresh
  - delay < thresh
- delay > thresh
- default - sin(az)<thresh && wheel_rate>thresh
- default || delay<thresh
State Transition Diagram -- A Sample Problem

Man, wolf, goat, cabbage crossing a river. Boat can carry man plus one of the other three. To cross, man must ferry one at a time.

Forbidden states:

- Cannot leave wolf alone with goat
- Cannot leave goat alone with cabbage

Draw state transition diagram showing legal states and legal transitions
Example Lander Problem - State Transition Diagram

- Draw a State Transition Diagram of the control for landing the craft.
- Hints:
  1. Use "thrust" and "No thrust" as states
  2. Transitions are made due to velocity or distance
  3. Include an ending state!
- Process:
  1. Determine initial and final states
  2. Determine intermediate states
  3. Determine transitions (arrows) and what causes the transition (label on arrow)
v<2m/s
No Thrust
q₀

v>2m/s
Thrust

v<2m/s
d=0

Landed
d=0

v>2m/s
Review

• Today we learned three techniques for representing algorithms

<table>
<thead>
<tr>
<th>Technique</th>
<th>Advantages</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudo-code</td>
<td>Close to programming language</td>
<td>May be ambiguous, error-prone</td>
</tr>
<tr>
<td>Flowchart</td>
<td>Good for single tasks; shows flow of execution</td>
<td>Cannot represent concurrent tasks or time-based behavior; becomes complex as program grows</td>
</tr>
<tr>
<td>Finite State Automata</td>
<td>Automatic code generation possible; shows states and transitions</td>
<td>Cannot represent concurrent tasks or details inside of states; becomes complex as number of states grow</td>
</tr>
</tbody>
</table>

• Important note: In your homework, you may choose whatever method best suits the problem, unless explicitly specified in your problem set
Reading Guidelines and Administrative issues

• For today's and Friday's lecture
  ➢ Read R5 Intro, 5.1-5.3, 5.6
  ➢ Skim 5.4, 5.5, 5.7
  ➢ Do not read 5.8-5.11
  ➢ Read C2.6-2.10

• Problem Set 2 is on the web

• New course outline on the web showing weightings for each problem set

• New TA office set up in 33-112. Problem sets returned here, one week after turn-in.

• Text book issues?