Overview

Definition of an Algorithm

- ***Definition:*** *An algorithm is a finite sequence of instructions, if followed, accomplishes a particular task.*

In addition, all algorithms must satisfy the following criteria:

1) **Input**
   - There are zero or more quantities that are externally supplied. (0개 이상의 input)
2) **Output**
   - At least one quantity is produced.(1개 이상의 ouptput)

Definition of an Algorithm (cont'd)

3) Definiteness (명확성)

- Each instruction is clear and unambiguous. (모든 구조는 명확해야함)

4) **Finiteness (유한성)**

- If we trace out the instructions of an algorithm, then for all cases, the algorithm *terminates* after a finite number of steps.

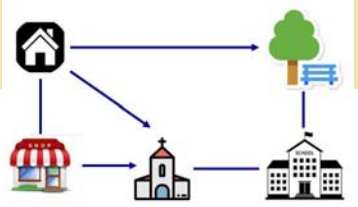
5) **Effectiveness (유효성, 실현가능성)**

- Every instruction must be *basic enough* to be carried out, in principle, by a person using pencil and paper. (It is not enough that each operation be definite as in 3).
  (손으로 쓸 수 있을 정도로 간단해야함)
- It also must be feasible(실현가능)

How to specify an Algorithm?

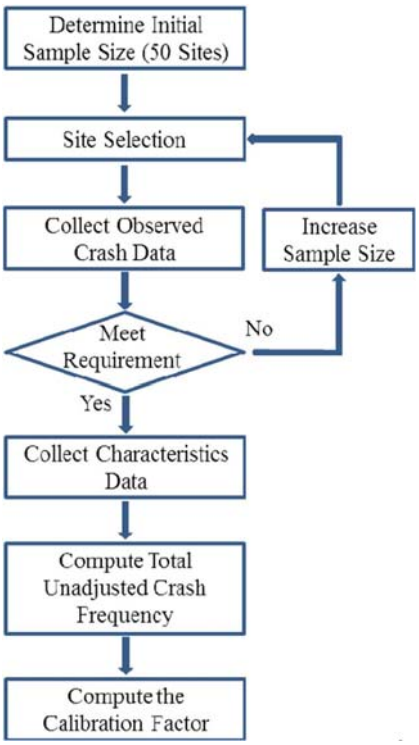
- Use a natural language like English or Korean.
  - Problem?
    - *Hard to make sure the statements are definite.*

- Find places you can go from home.
- From each of those places, find all paths.
- Keep track of the distance you've traveled as you go.
- Repeat this process until you get to school.
- Compare the distance you've traveled.
- Find the shortest path.



How to specify an Algorithm?

- Use graphic representation such as **flowcharts**.
  - Only suitable for small and simple algorithms.



How to specify an Algorithm?

- Use a ***pseudocode*** (or a ***pseudo-language***)
  - A combination of the constructs of a programming language (e.g., Java language) together with informal natural statements.

- We will describe an algorithm in Java, or sometimes in a pseudo-code.

Pseudocode

- High-level description of an algorithm
- More structured than English prose
- Less detailed than a program
- Preferred notation for describing algorithms
- Hides low-level detailed program design issues

Example: find max element of an array

Algorithm ***arrayMax*(*A*, *n*)**
  Input array ***A*** of ***n*** integers
  Output maximum element of ***A***

  currentMax ← *A*[0]
  for ***i*** ← 1 **to** ***n*** – 1 **do**
    if *A*[*i*] > ***currentMax*** **then**
      currentMax ← *A*[*i*]
  return ***currentMax***

Pseudocode Details

- Control flow
  - **if** … **then** … [**else** …]
  - **while** … **do** …
  - **repeat** … **until** …
  - **for** … **do** …
  - Indentation replaces braces
- Method declaration

  Algorithm ***method*** (***arg*** [, ***arg***…])
    Input …
    Output …
- Method call

  var.method (***arg*** [, ***arg***…])
- Return value

  return ***expression***
- Expressions

  ← Assignment (like = in Java)

  = Equality testing (like == in Java)

  n^2 Superscripts and other mathematical formatting allowed

Confusing terminologies

- Data Types
- Abstract Data Types
- Data Structures

What is a data type?

- A data type is a notion used in programming language.
- ***Definition:*** *A data type is a category of data characterized by*
  - *a collection of values(or domain) and*
  - ***a set of operations*** *that act on those values.*
  - It also has a particular representation

Example: **int** data type in Java

- value: {-2^31, ..., -1, 0, 1, 2, 2^31-1}
- operations: +, -, / (division), * (multiplication), <<, >> etc.

Example: **boolean**

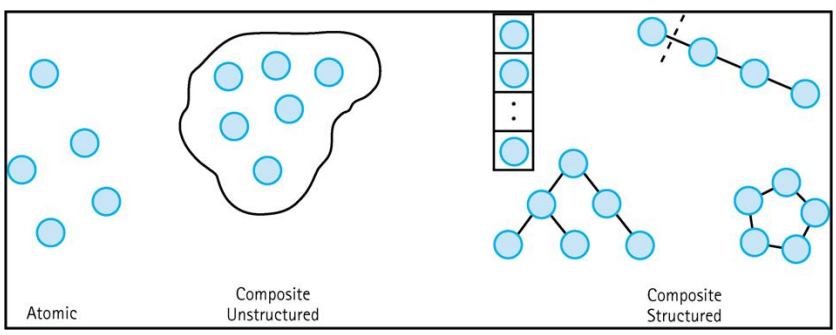
- value: {true, false}
- operations: and, or, not etc.

Simple Data Types vs. Complex Data Types

- Simple types (or atomic types or scala type)
  - Consist of values that are in the most basic form cannot be decomposed into smaller parts.
    - Ex., integer, floating-point numbers, boolean, etc.
- Complex types (or composite types)
  - Composites can be decomposed into a combination of atomic values or other composites.
    - Ex., complex numbers, strings, objects or containers/collections like arrays, lists, dictionaries, etc.

Collection (or Container) Data Types

- Each value represents a collection. The individual values of the collection are known as elements.
- of the container composed of multiple data items.
  - **Unstructured**: A collection of data items that are not organized with respect to one another
  - **Structured**: A collection of data items that are organized with respect to one another



Data Types in Java

- **Primitive (simple or atomic or scalar) types**
  - Predefined data types that represents data
  - boolean, byte, char, short, int, long, float, double
  - 모든 데이터를 객체로 만들면 낭비-> 심플 데이터 타입
- **Reference types**
  - Interfaces and classes
    - User-defined types
  - Arrays

Abstract data types (ADTs)

- Set of values (i.e., domain)
- Set of operations
- invariants

User's point of view

Data Type

Data Structures

- Concrete representations of data

Implementer's point of view

Data Structures

- Concrete organization of a collection of data elements reflecting their relationships
  - Set, One-to-One, One-to-Many, Many-to-Many etc.
- Implementation dependent data structures
  - **Array** and **Linked List**
- Implementation independent data structures
  - Stack, Queue, Tree, Hash Table, Heap, Graph, etc.
- Implementation independent data structures are normally defined first as **Abstract Data Type (ADT)**s.

Abstract Data Type (ADT)

- ***Definition:*** *An ADT is a data type whose properties (domain and operations) are specified independently of any particular implementation.*
  - *Separation of specification from implementation => Encapsulation*
- Java interface and class mechanism provides the means to define ADTs.

Specification of Collection ADT

- A Collection is a data type that is capable of holding a group of other items.
- **Bag** ADT
  - There can be many instances of the same item in the bag.

```
public interface IntBag {
    void add(int item);
    int countOccur(int item);
    boolean remove(int item);
    int size();
}
```

Adds one item to the collection.

Checks how many occurrences of a certain item are in the collection.

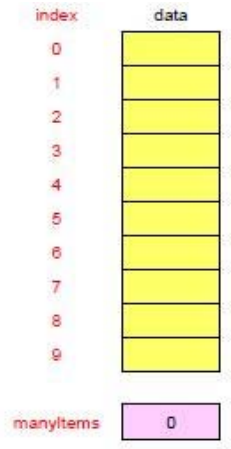
Removes one item from the collection.

checks how many items are in the collection.

Implementation Using Java Array

Representation:

- Use a partially filled array of fixed capacity
- Use one integer variable called manyItems, which stores the number of items currently in the bag
- An empty bag is initialized by a constructor, dynamically creating the array, and setting manyItems = 0.

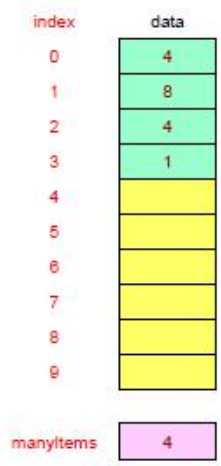



```
public class IntArrayBag implements IntBag {

    public static int INITIAL_CAPACITY = 100;

    private int[] data;
    private int manyItems;

    public IntArrayBag() {
        data = new int[INITIAL_CAPACITY];
        manyItems = 0;
    }
```

Remove

Code:

```
/**
 * Remove one copy of a specified element from this bag.
 * @param target
 *   the element to remove from the bag.
 * @postcondition
 *   If target is in the bag, one copy is removed, returns true.
 *   Otherwise the bag remains unchanged, returns false.
 **/
public boolean remove(int target) {
  int i;                                  // Find target
  for (i = 0; (i < manyItems) && (target != data[i]); i++) ;
  if (i == manyItems) return false;  // Not found.
  data[i] = data[--manyItems];       // Found. So remove.
  return true;
}
```



What is an ADT for?

- **Generalization**
  - ADTs can be used just like primitive types in PL.
- **Encapsulation** (**information hiding**):
  - The definition of an ADT and its operations can be localized to one section of the program.
  - Functions that make use of the ADT can safely ignores its implementation details.
- **Note**: An ADT can be built on some ADTs which can also be build on other ADTs as well.