Algorithms

Atharv Sule

December 17, 2021

Contents

Part I

Fundamentals

Chapter 1

Summary of topics

- You can run algorithms to study their properties
- You can put them to good use immediately in applications
- Programming constructs(building blocks), software libraries(programmming concepts), and operating systems used to impemet programs make up our peogramming model
- To understand this model let us frist talk about statements
- Here are the different types of statements:
  - Declarations: reate specific type of variables and name w identifiers
  - Assignments: associate data type with variable
  - Conditionals: provide change in execution flow
  - Loops: more profound change in execution flow, repeat block mutiple times
  - Call and returns relate static methods
- arrays store a sequence of values
  - to initalize an array declare array name and type,create the array

  initalxe the values

  - Defalt values are set to zero, you initalze them through a for loop

- Static methods: can bel declared without the name of the method, declare class name
  - Here's an example static method:

    ```
    public static squrt(double c)
    ```

- properties of methods
  - Methods can be overloaded
  - methods have a single return value but can have mulitple return statements
  - A method can have side effects
- Recursion: method will call intself
- External Libraries: imported statements (ex: `java.lang.*`)

1.1 Subheading

Chapter 2

Data Abstraction

- Abstract data type: data type whose representation is hidden from the client
- Abstract data types are important because they support encapsulation.
- You do not need to know the data type implemented in order to be able to use it
- Objects are characterized by three properties: state, identity, behavior
- State: value from its data type
- Identity: distinguishes one object from another.
- Behavior: are an objects predefined functions.
- Reference: means of accessing an object
- To invoke methods, you would put the class name "." and then the method name
- For Example:

  ```
  Counter heads = new Counter("heads");
  heads.increment();
  ```

- The primary purpose of static methods is to implement functions, while non-static methods implement data-type operations
- Aliasing: both variables refer to the same object

- For Example:

```
Counter c1 = new Counter("ones");
c1.increment();
Counter c2 = c1;
c2.increment;
stdOut.println(c1);
```

- you can pass objects as arguments to methods, java passes a copy of the argument value from the calling program to the method(cannot change value of the variable
- All nonprimitve types are objects so in a way arrays are objects
- $e = mc^2$
- Writing code that refers to data abstraction is referred to as object-oriented programming
- This is displalystyle:
$$e = mc^2$$
- (ADT): to simplify client code
- `In` Class allows multiple lines of code
- Constructor-initiates instance variables
- Scope:(parameter: the method, Local: the block statement, Instance: whole class)
- Instace methods: behavior of class
- Sometimes you need to maintain two implementations one for clients and another for other people
- immutable data type: value never changes once cons
- Algorithms is an implementation of an is the implementation of an instance method in an absract data type
- Data absraction is good for algorithms because it is a framework because it specifies what the algorithms need to accomplish and how the client can make use of it

Chapter 3

Bags, Queues, and Stacks

- several fundemental data types involve collection of objects
- bag queu and stack are essiential in unfersanding algorithums
- parameterized types- pass in what type of data that you want to use
- EX:

```
     Stack<String> stack = new Stack<String>();
     stack.push("Test");

     String next = stack.pop();
     #+end_src java
   - Casting a primitve type as a wrapper
   - FIFO queue- first to leave and frit to ender polcy
   - pushdown stack- based on first in frist out
   - Arithmeic: below is an example of how arithmetic is used in java
      #+begin_src java
 import java.util.Stack;
import java.util.*;

public class Evaluate {
    public static void main(String[] args) {
        Stack<String> ops = new  Stack<String>();
        Stack<Double> vals = new  Stack<Double>();

        // if array args length is equal to zero then the length is zero
//        if (args.length == 0) {
```

```
//              System.out.println("Usage:expression");
//              return;
//          }
            String arg1 = args[0];
            int charIndex = 0;
            System.out.println(arg1.length());
            while (charIndex < arg1.length()){
                char stringChar = arg1.toCharArray()[charIndex++];
                String s ="" + stringChar;
                if(s.equals("(")) {

                }else if (s.equals("+")){
                    ops.push(s);
                } else if(s.equals("-")){
                    ops.push(s);
                }else if (s.equals("*")){
                    ops.push(s);
                }else if(s.equals("/")){
                    ops.push(s);
                }else if (s.equals("sqrt")){
                    ops.push(s);
                }else if (s.equals(")")){
                    String op = ops.pop();
                    double v = vals.pop();
                    if(op.equals("+")){
                        v = vals.pop() + v;
                    }else if (op.equals("-")){
                        v = vals.pop() -v;
                    }else if(op.equals("*")){
                        v = vals.pop() *v;
                    }else if (op.equals("/")){
                        v = vals.pop()/ v;
                    }else if(op.equals("sqrt")){
                        v = Math.sqrt(v);
                    }
                    vals.push(v);
                }else{
                    vals.push(Double.parseDouble(s));
                }
            };
            System.out.println(vals.pop());
        }
    }
```

- abstact data type is a fixed capacity stack
- fixed capacity stack only works for strings
- it requires a cleint to spe
- The pproblem with fixed stack is that it only uses strings to do this we to devolp another class w similar code
- It is possible to iterate through a Stack
- Linked list is recsive data structure that is either empy or a refernce to a node having a generic item and refernce to a node having generic item and a refernce to a linked list
- Ex:

  ```
  private class Node
  {
     Item item;
     Node next;
  }
  ```

- A note has two instance variables: An item and a node
- You define a node in a class and make it private becuse it is not for use by clients
- we Use `new Node()`, results in a new node object with its initial values being null
- you refer to node instance variables by saying: `first.item`, & `first.next` this is known as records
- Below explans how you would build a linked list:

  ```
  // you declare your values like this
  Node first = new Node();
  Node second = new Node();
  Node third = new Node();
  // you iniialise the values like this, they can take up any data value
  ```

```
first.item = "to";
second.item = "be";
third.item = "or";

// Then you will set the next feilds to
first.next = second;
second.next = third
// Third remains null becuse there is no node after it
```

- A liked list represents a sequence of items
- use rectangle system to see each object Do as follows: `[to/ ]-> [be/ ] -> [or/(null)]`
- If you want to insert a new node in the list the best place to do so is at the beggining of the list
- Here is how

  ```
  //
   Node oldFirst = first;
   First = new Node();
   first.item = "not";
   first.next = oldfirst;
  ```

  - to remove nodes from the list you can assign the value `first` to `first.next`
  - like this:

    ```
    first = first.next;
    ```

    * this assing the first value to the value that comes after it elimating the original value
    * To insert at the end all you have to do is esablish a link to the last node in the list.
    * Stack, Queue, and Bag implentations for linked lists:

- Queue implementation basics: push & pop: (add at the begining)

  ```
  public void enquene(Item item){

     // adds item to end of the list
     Node oldlast = last;
  ```

```
   last = new Node();
   last.item = item;
   // next element after last is set to null
   last.next = null;
   // is empty checks if frist element is empty
   if(isEmpty()){

   // sets frist equal to last

   first = last


   }else{
     // next value after oldlast is set to last
     oldlast.next = last

   }


 public Item pop()
{

 // merges with item before it
 Item item = first.item;
 first = first.next;
 N--;
 if(is.Empty()) last = null;
 return item;
}




}
```

  – Stack implementation basics: push: (add item to the top of the stack)

```
public void push(Item item){

Node oldFirst = first;
first = new Node();
```

```
          // sets the old note equal to an item
          first.item = item;
          // sets nextNode equal to oldnote
          first.next = oldfirst;


        }
```

- pop:(remove from the top of the stack)

```
public Item pop()
{
  Item item = first.item;
  // sets the current first equal to the next item
  first = first.next;
  N--;
  return item
}
```

- Here is what you can do
- Here is how:

  ```
  Node oldlast = last;
  last = new Node();
  last.item = "not";
  oldlast.next = last;
  ```

- two ways to rpresnt collection of objects are arrays and linked lists Arrays are built into java, linked lists are easy to build with sandard java records
- linked lists are the fundemental alternative to arrays when structureing data
- To loop through a list you would do this:

  ```
  for(Node x=first; x != null; x = x.next){ // Process x.item}
  ```

Chapter 4

Analysis of Algorithms

Chapter 5

Case Study: Union-Find

Part II

Sorting

Chapter 6

Elementary Sorts

Chapter 7

Merge Sort