CS1007 lecture #21 notes

thu 21 nov 2002

- news
- vectors
- linear searching
- reading: ch 12.1-12.6

vectors (1).

- Java has a nice class which handles arrays dynamically: `java.util.Vector`
- the elements of a `Vector` can be any type of Java `Object`
- note that when you fetch an element from a vector, you have to cast it from a generic object to the specific class type the object should be (see example below)
- some methods:
  - constructor: `Vector();`
  - `public void addElement( Object obj );`
  - `public void insertElementAt( Object obj, int index );`
  - `public void removeElementAt( int index );`
  - `public void removeAllElements();`
  - `public void setElementAt( Object obj, int index );`
  - `public Object elementAt( int index );`
  - `public int size();`

vectors – example (1).

```
import java.util.*;

public class Dice {

  private Random random;
  private int    value;

  public Dice( Random r ) {
    random = r;
    roll();
  } // end of Dice() constructor

  public void roll() {
    value = Math.abs( random.nextInt() % 6 ) + 1;
  } // end of roll() method

  public int getValue() {
    return( value );
  } // end of getValue() method

  public String toString() {
    return( String.valueOf( value ));
  } // end of toString() method

} // end of Dice class
```

vectors – example (2).

```
import java.util.*;
import java.io.*;

public class ex21a {

  public static void main( String[] args ) {

    Vector dice;
    Random random = new Random();
    int    ndice = Integer.parseInt( args[0] );

    dice = new Vector( ndice );
    for ( int i=0; i<ndice; i++ ) {
      dice.addElement( new Dice( random ));
    }

    for ( int i=0; i<ndice; i++ ) {
      Dice tmp = (Dice)dice.elementAt( i );
      System.out.print( tmp + " " );
    }
    System.out.println();

  } // end of main()

} // end of class ex21a
```

vectors – example (3).

- notice that we instantiate twice...
- notice that we instantiate in the call to `dice.addElement()`
- notice that we *cast* the return from `dice.elementAt()`

searching.

- Often, when you have data stored in an array, you need to locate an element within that array.
- This is called searching.
- Typically, you search for a *key* value (simply the value you are looking for) and return its *index* (the location of the value in the array)
- As with sorting, there are many searching algorithms.
- We'll study the following:
  - linear search
    - standard linear search, on sorted or unsorted data
    - modified linear search, on sorted data only
  - binary search
    - iterative binary search, on sorted data only
    - recursive binary search, on sorted data only (next time)

linear search on UNSORTED DATA.

- Linear search simply looks through all the elements in the array, one at a time, and stops when it finds the key value.
- This is inefficient, but if the array you are searching is not sorted, then it may be the only practical method.

```
public int linearSearch( int key ) {
  for ( int i=0; i<dice.length; i++ ) {
    if ( key == dice[i].getValue() ) {
      return( i );
    }
  } // end for i
  return( -1 );
} // end of linearSearch() method
```

linear search on UNSORTED DATA, using a Vector.

```
public int linearSearchV( int key ) {
  for ( int i=0; i<dice.size(); i++ ) {
    Dice d = (Dice)dice.elementAt( i );
    if ( key == d.getValue() ) {
      return( i );
    }
  } // end for i
  return( -1 );
} // end of linearSearchV() method
```

linear search on SORTED data.

- If the array you are searching IS sorted, then you can modify the linear search to stop searching if you have looked past the place where the key would be stored if it were in the array.
- This only helps shorten the run time if the key is not in the array...

```
public int modifiedLinearSearch( int key ) {
  for ( int i=0; i<dice.length; i++ ) {
    if ( key == dice[i].getValue() ) {
      return( i );
    }
    else if ( key < dice[i].getValue() ) {
      return( -1 );
    }
  } // end for i
  return( -1 );
} // end of modifiedLinearSearch() method
```

linear search on SORTED data, using a Vector.

```
public int modifiedLinearSearchV( int key ) {
  for ( int i=0; i<dice.size(); i++ ) {
    Dice d = (Dice)dice.elementAt( i );
    if ( key == d.getValue() ) {
      return( i );
    }
    else if ( key < d.getValue() ) {
      return( -1 );
    }
  } // end for i
  return( -1 );
} // end of modifiedLinearSearchV() method
```

binary search (1).

- Binary search is much more efficient than linear search, ON A SORTED ARRAY. (It CANNOT be used on an unsorted array!)
- It takes the strategy of continually dividing the search space into two halves, hence the name *binary*. Say you are searching something very large, like the phone book. If you are looking for one name (e.g., “Gilligan”), it is extremely slow and inefficient to start with the A’s and look at each name one at a time, stopping only when you find “Gilligan”. But this is what linear search does. Binary search acts much like you’d act if you were looking up “Gilligan” in the phone book.
  - You’d open the book somewhere in the middle, then determine if “Gilligan” appears before or after the page you have opened to.
  - If “Gilligan” appears after the page you’ve selected, then you’d open the book to a later page.
  - If “Gilligan” appears before the page you’ve selected, then you’d open the book to an earlier page.
  - You’d repeat this process until you found the entry you are looking for.

binary search (2).

```
public int binarySearch( int key ) {
  int lo = 0, hi = dice.length-1, mid;
  while ( lo <= hi ) {
    mid = ( lo + hi ) / 2;
    if ( key == dice[mid].getValue() ) {
      return( mid );
    }
    else if ( key < dice[mid].getValue() ) {
      hi = mid - 1;
    }
    else {
      lo = mid + 1;
    }
  } // end while
  return( -1 );
} // end of binarySearch() method
```

binary search (3).

```
public int binarySearchV( int key ) {
  int lo = 0, hi = dice.size()-1, mid;
  while ( lo <= hi ) {
    mid = ( lo + hi ) / 2;
    Dice d = (Dice)dice.elementAt( mid );
    if ( key == d.getValue() ) {
      return( mid );
    }
    else if ( key < d.getValue() ) {
      hi = mid - 1;
    }
    else {
      lo = mid + 1;
    }
  } // end while
  return( -1 );
} // end of binarySearchV() method
```