PSA EXTRA: Putting intelligence into your connect 4 game
Optional, but highly encouraged! – DUE WEDNESDAY FOR EVERYONE! (no additional slip day)

Not so much “guidance” for PSA EXTRA, but THERE ARE STILL LAB-HOURS
(if you’re doing PSA Extra!! Or want to practice any other thing!)

Search

About 1,570,000 results (0.20 seconds)

Did you mean: recursion

Recursion - Wikipedia, the free encyclopedia
Recursion is the process of repeating items in a self-similar way. For instance, the surfaces of two mirrors are exactly parallel with each other when recursively defined.

Recursion (computer science) - Wikipedia, the free encyclopedia
Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem; i.e., the method involves repeated application of the same procedure.
1. and 2. Our Tower method will return the result of a “tower” of powers of two, as high as we indicate with the parameter n. See the examples on the left. What is the base case?

```
tower(1) = 2
tower(2) = 2^2
tower(3) = 2^{(2^2)}
tower(4) = 2^{(2^{(2^2)})}
```

<table>
<thead>
<tr>
<th></th>
<th>(1A)</th>
<th>(1B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>B.</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>C.</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>D.</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

// recursive tower
// Math.pow( base, power ) will come in handy here
public static long tower(int n) {
    if ( n == (1-A) ){
        return (1-B) ; // base case
    }
    else{ // recursive step
        ___ (2)________
    }
}
1. and 2. Our Tower method... What would be a good the recursive step?

2) Recursive step: do a little bit of work, using the result of tower on a reduced n.

tower(1) = 2

tower(2) = 2²

tower(3) = 2²(2²)

tower(4) = 2²(2²)

A. return (long) tower(n-1)
B. return (long) Math.pow(2, tower(n-1));
C. (long) Math.pow(2, tower(n-1));
D. (long) tower(Math.pow(2, n-1));

CORRECTION!
JAVA method to obtain any power is pow (base power) !!!!
3. True or False: The recursive step in **selection sort** is to find the smallest number in the sublist and swap it with the first number in that sublist.

A. True

B. False
4. What is the restriction on the elements in the array that we can pass to the binary search algorithm to work (PROPERLY)?

A. The elements in the array need to be sorted

B. The elements in the array need to be binary numbers

C. The elements in the array need to be positive numbers

D. The elements in the array need to be integers

E. Any array can be passed and the algorithm will work
Have you started the PSA Extra?
A. I’m done 😊
B. I’ve started!
C. I haven’t started yet, but I plan to.
D. I’m not planning to do this PSA
PSA-EXTRA: The \textbf{C4Player}

\textbf{ConnectFour}

1) Add AI player “variable”
2) As usual, human-player clicks cell \textcolor{red}{\rightarrow} make the move.
3) Right after... make your AI player “decide” its nextMove \textcolor{red}{\rightarrow} then make that move.

\textbf{C4Player}

\begin{verbatim}
C4Player( char ch, String tbt, int plyIn )
String toString()
char oppCh() // opposite checker than current turn
double scoreBoard(C4Board b)
double[] scoresFor(C4Board b)
int tiebreakMove(double[] scores)
int nextMove(C4Board b) // gives the column index where the computer wants to play given a board b ...}
\end{verbatim}
PSA-EXTRA: The **C4Player**

**DEMO!**

```java
C4Player(char ch, String tbt, int plyIn)

String toString()

double scoreBoard(C4Board b)

double[] scoresFor(C4Board b)

int tiebreakMove(double[] scores)

int nextMove(C4Board b) // gives the column index where the computer wants to play given a board b …
```

**HOW DO I DECIDE THAT!!???!?!**

- **THE GOAL!** Is to make a recursive “exploration” to find the move that gets you the highest score  
- You can START with a “DUMMY” method that decides without much “intelligence”.

**EXPLORE and give a score to the options . . .**
PSA-EXTRA: The C4Player

- **DEADLINE WEDNESDAY** for everyone! (no additional slip, that’s already one slip for everyone that tries this PSA)

- **Replace lowest PSA2 to PSA7 score.** Get up to 3 star points if you don’t get to improve your PSA score (because it’s already so good!)

- **DO IT FOR THE SAKE OF LEARNING!** Study and run our suggested code for PSA4 and PSA6. Practice with recursion
  Can you make this? It’s very challenging **BUT** fun!

---

C4Player( char ch, String tbt, int plyIn )

String toString()

char oppCh() // opposite checker than current turn
double scoreBoard(C4Board b)
double[] scoresFor(C4Board b)
int tiebreakMove(double[] scores)
int nextMove(C4Board b) // gives the column index where the computer wants to play given a board b ...
public static int find( String[] myList, String toFind, int startIndex )
{
    if ( startIndex >= myList.length ) {
        return -1;
    }
    if ( toFind.equals( myList[startIndex] ) ) {
        return startIndex;
    }
    // Recursive step here . . .

    WHAT ELSE DO WE NEED once we have designed the “base case”?
public static int find( String[] myList, String toFind, int startIndex )
{
    if ( startIndex >= myList.length ) {
        return -1;
    }
    if ( toFind.equals( myList[startIndex] ) ) {
        return startIndex;
    }
    __ what goes here? ______
}

A. return find( myList, toFind, startIndex + 1 );
B. return find( myList, toFind, startIndex - 1 );
C. return startIndex + 1;
D. find( myList, toFind, startIndex + 1 );
E. None of these.

Explain why each answer is wrong or right!
Ever used a telephone book?

- If I want to look up someone’s number and their last name starts with Z, where should I start? Do you really do like “find” method?

- If I want to look up someone’s number and their last name starts with G where should I start?
Binary Search algorithm
(only works for “ordered” data)

Look in the middle of the data list (array)

– If that element is what you are looking for – return its index

– If not

• If what you are looking for “comes before” in the ordering, look in the half on the left

• Else (what you are looking for “comes after” in the ordering), look in the half on the right
Binary search

List a in which data is to be searched

http://mohtashims.files.wordpress.com/2010/07/binary-search.gif
### Binary Search example

```java
public static int binarySearch(int[] theList, int toFind, int low, int high)
```

low will start at 0, high will start at theList.length-1

Assuming the array pictured above is `myArray` these would be some example calls:

```java
> binarySearch( myArray, 4, 0, 8 )
```

```java
> binarySearch( myArray, 5, 0, 8 )
```
Imagine the numbers are arranged like this...

```
  10
 /  \
4    14
 /  \
2    6
   /  \
  12  16
   /  \
  8   18
```
Algorithm “pseudo-code”:
1. calculate the midpoint, mid, between low and high
2. if theList[mid] == toFind, return mid
3. if toFind is larger than theList[mid], recurse on the larger half of the list
4. else if toFind is smaller than theList[mid], recurse on the smaller half of the list

Trace the values of high, low and mid when you call binarySearch with this list above and the value 8. Draw more stack frames if you need them.
Binary Search example

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Algorithm “pseudo-code”:

1. calculate the midpoint, mid, between low and high
2. if theList[mid] == toFind, return mid
3. if toFind is larger than theList[mid], recurse on the larger half of the list
4. else if toFind is smaller than theList[mid], recurse on the smaller half of the list

Trace the values of high, low and mid when you call binarySearch with this list above and the value 8. Draw more stack frames if you need them.

**D. 4**

How many stack frames do we need?

A. 1
B. 2
C. 3
D. 4

Do you have my number? Nope! I have a 10
Binary Search example

<table>
<thead>
<tr>
<th>high</th>
<th>low</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Algorithm “pseudo-code”:
1. calculate the midpoint, \( \text{mid} \), between low and high \( (\text{high} + \text{low}) / 2 \) 
2. if \( \text{theList}[\text{mid}] == \text{toFind} \), return \( \text{mid} \) 
3. if \( \text{toFind} \) is larger than \( \text{theList}[\text{mid}] \), recurse on the larger half of the list 
4. else if \( \text{toFind} \) is smaller than \( \text{theList}[\text{mid}] \), recurse on the smaller half of the list

Trace the values of high, low and mid when you call binarySearch with this list above and the value 8. Draw more stack frames if you need them.

Do you have my number?
Algorithm “pseudo-code”:
1. calculate the midpoint, mid, between low and high
2. if theList[mid] == toFind, return mid
3. if toFind is larger than theList[mid], recurse on the larger half of the list
4. else if toFind is smaller than theList[mid], recurse on the smaller half of the list

Trace the values of high, low and mid when you call binarySearch with this list above and the value 8. Draw more stack frames if you need them.
Let’s start translating our pseudo-code to actual Java code...

```java
public static int binarySearch( ArrayList<Integer> list, int toFind, int low, int high ) {
    int mid = (low+high) / 2;

    if (toFind < list.get( mid ))
        return ________________________________
    else if (toFind == list.get( mid ))
        return ________________________________
    else // toFind is > list.get(mid)
        return ________________________________
}
```
public int binarySearch( ArrayList<Integer> list,  
    int toFind, int low, int high ) {

    int mid = (low+high) / 2;
    if (toFind < list.get(mid))
        return binarySearch( list, toFind, low, mid-1 );
    else if (toFind == list.get(mid))
        return mid;
    else // toFind is > list.get(mid)
        return binarySearch( list, toFind, mid+1, high );
}

Which is the base case in the method above?
public int binarySearch( ArrayList<Integer> list, int toFind, int low, int high ) {
    int mid = (low+high) / 2;
    if (toFind < list.get( mid ))
        return ________________________________
    else if (toFind == list.get( mid ))
        return ________________________________
    else // toFind is > list.get(mid)
        return ________________________________
}

When will this method not work?
A. When the element you are looking for is the first element in the list
B. When the element you are looking for is the last element in the list
C. When the element you are looking for is not in the list
D. When the element you are looking for is less than 0
E. It will always work
Algorithm “pseudo-code”:
1. calculate the midpoint, mid, between low and high
2. If theList.get(mid) == toFind, return mid
3. if toFind is larger than theList.get(mid), recurse on the larger half of the list
4. else if toFind is smaller than theList.get(mid), recurse on the smaller half of the list

Trace the values of high, low and mid when you call binarySearch with this list above and the value 7. Draw more stack frames if you need them.
Algorithm “pseudo-code”:
1. calculate the midpoint, mid, between low and high
2. if theList[mid] == toFind, return mid
3. if toFind is larger than theList[mid], recurse on the larger half of the list
4. else if toFind is smaller than theList[mid], recurse on the smaller half of the list

Trace the values of high, low and mid when you call binarySearch with this list above and the value 7. Draw more stack frames if you need them.
Algorithm “pseudo-code”:
1. calculate the midpoint, mid, between low and high
2. if theList[mid] == toFind, return mid
3. if toFind is larger than theList[mid], recurse on the larger half of the list
4. else if toFind is smaller than theList[mid], recurse on the smaller half of the list

Trace the values of high, low and mid when you call binarySearch with this list above and the value 7. Draw more stack frames if you need them.
public int binarySearch( ArrayList<Integer> list,
    int toFind, int low, int high ) {
    // base case for not in list

    int mid = (low+high) / 2;
    if (toFind < list.get( mid ))
        return binarySearch( list, toFind, low, mid-1 );
    else if (toFind == list.get( mid ))
        return mid;
    else // toFind is > list.get(mid)
        return binarySearch( list, toFind, mid+1, high );

    Add the missing base case!
public int binarySearch( ArrayList<Integer> list,
        int toFind, int low, int high ) {
    // base case for not in list
    if ( (low == high) && toFind != list.get(mid) ){
        // Other option: if (low > high){
            return -1;
        }

    }

    int mid = (low+high) / 2;

    if (toFind < list.get(mid))
        return binarySearch( list, toFind, low, mid-1 );

    else if (toFind == list.get(mid))
        return mid;

    else // toFind is > list.get(mid)
        return binarySearch( list, toFind, mid+1, high );
}

Add the missing base case
Exam 4 – NEXT TUESDAY

A bit more of programming than exam3, but you’ll get the method list with everything you may need (no need to memorize!!)

• Interfaces, Abstract classes

• Event Listeners

• Recursion
  – Main ideas: base case, recursive case, stack frame
  – Simple examples: mathematical operations, drawing examples