Lec_25.md

Lecture 25 - Recursion

Nov. 05/2020

Announcements:

Reminder that we have a quiz tomorrow

- 45 minutes (usually they are 30 minutes)
    - More questions?

Recursion

Thinking Recursively

Recall the process for thinking recursively:

1. Find the **Basis**

- Identify the simplest size of the problem you can solve

2. Define the **Recursion**

- For any case larger than the **basis**
    - Think about dividing the problem into parts that look like the original problem, but smaller in size
    - Goal is to reach the basis

3. Write the Code
4. **Trace** the Code to validate

Palindrome

A **palindrome** is a word that reads the same from

- left to right
- right to left

Examples:

- bob
- deed
- level

Non-examples:

- tarek
- hello

Assuming an array of n characters that stores a word, lets write a recursive function that detects if this word is a **palindrome**

1. **Basis**

- Single characters are immediately **palindromes**

2. **Recursive**

- Check if the left and right characters of the single character **basis** are equal (**palindrome** condition)

```
bool isPalindrome (char* seq, int left, int right) {
  if (left == right) return true;
  if (seq[left] == seq[right]) return isPalindrome (seq, left+1, right-1);
  else return (false);
}
```

Notes:

1. `if (left == right) return true;`

- **Basis**
- For a single character, `left == right` evaluates to `true`

2. `if (seq[left] == seq[right]) return isPalindrome (seq, left+1, right-1);`

- If at any point, `seq[left]==seq[right]` evaluates to `false`
  - Then the values on the left and right of the **basis** are not equal
    - So the word is not a **palindrome**

3. Issue

- What if the word has *even* length?
  - Then the **basis** is not *one character*, but rather *two characters*

Fix:

```
bool isPalindrome (char* seq, int left, int right) {
  if (left == right) return true;
  if ( ((left+1) == right) && (seq[left] == seq[right]) ) return true;
  if (seq[left] == seq[right]) return isPalindrome (seq, left+1, right-1);
  else return (false);
}
```

Notes:

1. `if ( ((left+1) == right) && (seq[left] == seq[right]) ) return true;`

- **base case** for when the input word has *even* length

Recursion on a Linked List

As an exercise, we want to print a linked list in reverse order

- Print the **tail** node first
  - And consecutive nodes until we hit the **head** node

1. **Basis**

- The simplest case is *not* when **linked list** has size 1
- Simplest case is when the **linked list** has size 0
  - e.g. `p == NULL` , where p is the current node

2. **Recursive**

- Print the value of the previous node
- Can implement this behaviour using the code:

```
void rprint (listNode * p) {
  if (p == NULL) return;
  rprint(p->next);
  cout << p->data;
}
```

Notes:

1. `if (p == NULL) return;`

- **Basis**
- `p==NULL` evaluates to `true` on the n+1 node
  - n the size of the **linked list**
- Or in other words, on the `next` node for the last node in the **linked list**
  - This node has no data (it is `NULL`, after all)
  - So we `return;`, without doing anything

2. `rprint(p->next)`

- **Recursive** call
- This will **recurse downwards** (traversing through the linked list)
  - Until we hit the **Basis**

3. `cout << p->data;`

- When the **recurse upwards** happens, this will print the list starting with the *last element*
  - Will print the **tail** first, and then every consecutive node until finally printing the **head** node

4. Notice there is no return

- `return;` statement is good to have
  - *Not required* for `void` functions

5. Can we pass `listNode * p` *by reference*?

- Yes, but it doesn't add any extra value

Quicksort

Want to sort an N-element array of integers `A` in ascending order.

`A`:

Index	Element
0	
1	
2	
3	
...	
N-2	

Index	Element
N-1	

Using the **Quicksort** algorithm:

1. Select one element of array

- Identify this element as the **pivot**

2. Determine the final location of the pivot in the sorted array

- Identify that location as **p**

3. Shuffle the elements of the array

- All elements of the array with values *less than or equal* to value of pivot have indices *less than* **p**
- All elements of the array with values *greater than or equal* to value of pivot have indices *greater than* **p**

This lends itself to recursion greatly:

`A` :

Index	Element
0	
1	
2	
3	
...	
p-1	
p	**pivot**, which is sorted
p+1	
...	
N-2	
N-1	

We can now consider two separate arrays

- One with the range `A[0]` to `A[p-1]`
- Another with the range `A[p+1]` to `A[N-1]`

Quicksort the left half, **Quicksort** the right half until both halves are sorted

QuickSort PseudoCode

```
void QuickSort (int *A, int left, int right) {
  int PivotIndex;
  PivotIndex = SelectAndShuffle (A, left, right);
  if (PivotIndex > left) QuickSort (A, left, PivotIndex-1);
  if (PivotIndex < right) QuickSort (A, PivotIndex+1, right);
}
```

Initial call to **Quicksort**: `QuickSort(A,0,N-1);`

Tracing Quicksort

Start with `A`:

Index	Element
0	26
1	33
2	35
3	29
4	19
5	12
6	22

1. Select 26 as **pivot**
2. Place 26 in `A[3]`
3. Shuffle

Index	Element
0	19
1	12
2	22
3	26
4	33
5	35
6	29

For the left side:

Index	Element
0	19
1	12
2	22

1. Select 19 as pivot
2. Place it in `A[1]`
3. Shuffle

Index	Element
0	12
1	19

Index	Element
2	22

For the right side:

Index	Element
0	29
1	33
2	35

1. Select 33 as **pivot**
2. Place 33 in `A[1]`
3. Shuffle

Index	Element
0	29
1	33
2	35

Upwards Traversal

Merge **left side**, **right side**, and original **pivot**

Index	Element
0	12
1	19
2	22
p	26
0	29
1	33
2	35

Which becomes

Index	Element
0	12
1	19
2	22
3	26
4	29
5	33
6	35

And now the array is ***completely sorted!***

SelectAndShuffle Implementation

1. Sweep through the array

- Start at left and end at upper bound of array `N-1`

2. When an element is found that is *smaller* than the **pivot**, place it onto the left side
3. Repeat this until the array is *pseudo-sorted* into two sections

- One section where every value is ***less*** than **pivot**, but *is not necessarily sorted*
- Another section where every value is ***greater*** than **pivot**, but *is not necessarily sorted*

4. Place **pivot** in between these two sections

```
int SelectAndShuffle (int *A, int left, int right) {
  int Lastsmall = left; int temp;
  for (i = left+1 ; i <= right ; ++i) {
    if (A[i] <= A[left]) {
      Lastsmall = Lastsmall + 1;
      temp = A[i];
      A[i] = A[Lastsmall];
      A[Lastsmall] = temp;
    }
  }
  temp = A[left];
  A[left] = A[Lastsmall];
  A[Lastsmall] = temp;
  return (Lastsmall);
}
```

Notes:

1. `temp = A[i]; A[i] = A[lastsmall]; A[lastsmall] = temp;`

- This is code to swap two values in an array

2. `temp = A[left]; A[left] = A[Lastsmall]; A[Lastsmall] = temp;`

- This places the **pivot** in the right location