CS32 - Week 4

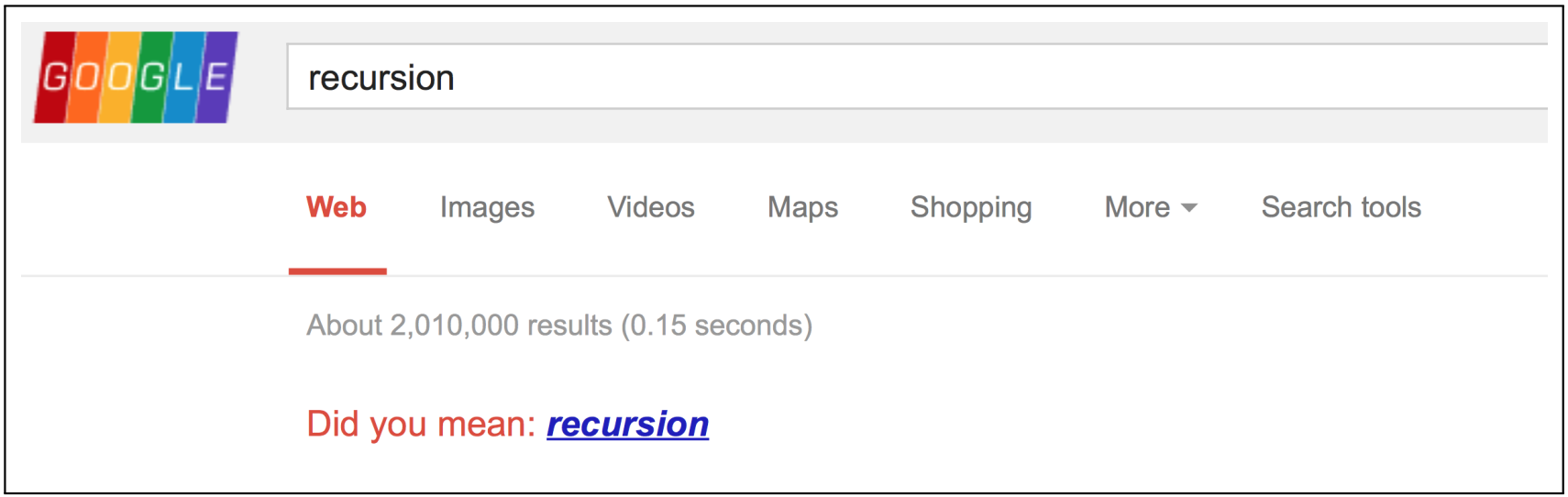
Doga Kisa

July 18, 2014

Recursion

In order to understand recursion, one must first understand recursion.

Let’s see what Google tells about recursion:



Recursion

A way of solving problems:

- If the problem is trivial, return the result.
- If not trivial, break the problem into small sub-problems.
- Solve sub-problems recursively.
- Use solutions of sub-problems to solve the problem.

```
Solve(Problem) {
  if(Problem is trivial) return result
  sol1 = Solve(Smaller Problem);
  sol2 = Solve(Smaller Problem);
  return combine(sol1, sol2);
}
```

Recursion Rules

Two rules of recursion:

- Base step (stopping condition).
- Simplifying step.

You must have those two steps. Otherwise, your recursive functions will run forever.

Recursive Calls

When dealing with recursion:

- Don’t delve into tracing down every level of the recursion.
- Instead, make the recursive leap of faith:
  - Any recursive call to a smaller problem than the one you were given works.

Example #1 ★

Write a recursive function to calculate factorials, that is:

$$Fact(n) = \begin{cases} 1, & \text{if } n = 0 \\ n \times Fact(n-1), & \text{if } n > 0 \end{cases}$$

```
int factorial(int n) {
  //base case
  if(n==0) return 1;

  //simplifying step
  int part1 = n;
  int part2 = factorial(n-1);

  //combine sub-problems
  return part1*part2;
}
```

Example #2 ★

Write a recursive function to compute b^e, for $b > 0$ and $e \geq 0$:

$$Power(b, e) = \begin{cases} 1, & \text{if } e = 0 \\ b \times Power(b, e-1), & \text{if } e > 0 \end{cases}$$

```
int power(int b, int e) {
  //base case
  if(e==0) return 1;

  //simplifying step and combining sub-problems
  return b*power(b,e-1);
}
```

How many times does this function recurse? e times.

Example #2.5 ★★

Can we optimize power(b, e)?

$$Pow(b,e) = \begin{cases} 1, & \text{if } e = 0 \\ Pow(b, e/2) \times Pow(b, e/2), & \text{if } e \text{ is even} \\ b \times Pow(b, \lfloor e/2 \rfloor) \times Pow(b, \lfloor e/2 \rfloor), & \text{if } e \text{ is odd} \end{cases}$$

```
int power(int b, int e) {
  //base case
  if(e==0) return 1;

  //simplifying step and combining sub-problems
  int half = power(b,e/2);
  if((e % 2)==0) return half*half:
  else return b*half*half;
}
```

How many times does this function recurse? $\log(e)$ times.

Example #3 ★★

- Do we always have one stopping condition? No.
- Do we always have one recursive call? No.

Write a recursive function to calculate Fibonacci sequence:

$$Fib(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ Fib(n-1) + Fib(n-2), & \text{if } n > 1 \end{cases}$$

```
int fib(int n) {
  //base case
  if(n==0) return 0;
  if(n==1) return 1;

  //simplifying step and combining sub-problems
  return fib(n-1) + fib(n-2);
}
```

Example #4 ★

How to recurse on arrays?

- By passing indices as arguments.

Write a recursive function to find the max value in an array:

```
int max(int[] arr, int start, int end) {
  //base case
  if(start==end) return arr[start];

  //simplifying step and combining sub-problems
  int first = arr[start];
  int rest = max(arr,start+1,end);
  return ((first>rest)? first:rest);
}
```

For the sake of readability, define a simpler function:

```
int maxValue(int[] arr, int size) {
  return max(arr,0,size-1);
}
```

Example #5 ★★★

Palindrome is a word that reads the same backward or forward, e.g., eye, Bob, sees, kayak, etc.

tattarrattat

Longest one in the Oxford English Dictionary.

Coined by James Joyce in Ulysses for a knock on the door.

Example #5 ★★★

Write a recursive function to decide if a string is palindrome:

Some useful functions:

- `s.length()` returns the length of `s`.
- `s.substr(start,len)` returns a substring of `s` starting from `start` having length of `len`.

```
bool isPalindrome(string s) {
  int n = s.length();
  //base case
  if(n==0) return true;
  if(n==1) return true;

  //simplifying step and combining sub-problems
  if(s[0]==s[n-1] &&
     isPalindrome(s.substr(1,n-2))) return true;
  else return false;
}
```

Example #6 ★★★★★

Write a recursive function to get all permutations of a string.

E.g., given **abc**, we want to get: (in any order)

abc, acb, bac, cab, bca, cba

How to break down the problem into smaller problems?

- What if we have all permutations of **bc**?

  bc, cb

- Isn't it enough to insert **a** into each possible position?

  abc, bac, bca

  acb, cab, cba

Some useful functions:

- `s.length()` returns the length of `s`.
- `s.substr(start,len)` returns a substring of `s` starting from `start` having length of `len`.

Example #6 ★★★★★

`vector` is defined in STL: to represent resizable arrays.

We'll store permutations using this class.

```
1  vector<string> permute(string s) {
2    int n = s.length();
3    //base case
4    if(n==1) return vector<string>(1,s);
5
6    //simplifying step and combining sub-problems
7    char first = s[0];
8    string rest = s.substr(1,n-1);
9    vector<string> rperm = permute(rest);
10   vector<string> results;
11   for(int i=0; i<rperm.size(); i++) {
12     string p = rperm[i]; //p is of size (n-1)
13     for(int j=0; j<n; j++) //n locations to insert first
14       results.push_back(p.substr(0,j)+first+p.substr(j,n-1-j));
15   }
16   return results;
17 }
```

Example #7 ★★★★★

A *k*-way permutation of a string is a permutation of size *k* obtained from the string.
Write a recursive function to get all *k*-way permutations of a string.

E.g., given **abcd** and $k = 2$, we want to get: (in any order)

ab, ac, ad, ba, bc, bd, ca, cb, cd, da, db, dc

How to break down the problem into smaller problems?

- **a** is in *k*-way permutations?
  - We need $(k-1)$-way permutations of **bcd**.
- **a** is not in *k*-way permutations?
  - We need *k*-way permutations of **bcd**.

Example #7 ★★★★★

Assume we can use `permute(string s)`.

```
1  vector<string> kWayPermute(string s, int k) {
2    int n = s.length();
3    //base case
4    if(k==0) return vector<string>(1,"");
5    if(n==k) return permute(s);
6
7    //simplifying step and combining sub-problems
8    char first = s[0];
9    string rest = s.substr(1,n-1);
10   vector<string> results = kWayPermute(rest,k);
11   vector<string> rperm = kWayPermute(rest,k-1);
12   for(int i=0; i<rperm.size(); i++) {
13     string p = rperm[i]; //p is of size (k-1)
14     for(int j=0; j<k; j++) //k locations to insert first
15       results.push_back(p.substr(0,j)+first+p.substr(j,k-1-j));
16   }
17   return results;
18 }
```

Example #7 ★★★★★

Using `permute(string s)` is a bit weird.

```
1  vector<string> kWayPermute(string s, int k) {
2    int n = s.length();
3    //base case
4    if(k==0) return vector<string>(1,"");
5    if(n==1) return vector<string>(1,s);
6
7    //simplifying step and combining sub-problems
8    char first = s[0];
9    string rest = s.substr(1,n-1);
10   vector<string> results;
11   if(n>k) results = kWayPermute(rest,k);
12   vector<string> rperm = kWayPermute(rest,k-1);
13   for(int i=0; i<rperm.size(); i++) {
14     string p = rperm[i]; //p is of size (k-1)
15     for(int j=0; j<k; j++) //k locations to insert first
16       results.push_back(p.substr(0,j)+first+p.substr(j,k-1-j));
17   }
18   return results;
19 }
```

Wrap-up

- Make sure base case is defined.
- Make sure simplifying step is defined.
- Don’t forget to test your function.
  - For each base case.
  - At least one recursive call.

Slides

Slides will be available at

http://www.cs.ucla.edu/~doga/cs32