CS1101S Discussion Group Week 10:
Memoization & Object-oriented Programming

Niu Yunpeng

niuyunpeng@u.nus.edu

October 24, 2017

The Real Essence

Never write code.

The Real Essence

Write programs!

Overview

1. Memoization
   - Inspiration
   - To use memoizationn
   - Memoization & tabulation
2. Object-oriented concepts
   - Class, object & instance

Memoization

Inspiration from Fibonacci

fib(5)
fib(4) fib(3)
fib(3) fib(2) fib(2) fib(1)
fib(2) fib(1) fib(1) fib(0) fib(1) fib(0) 1
fib(1) fib(0) 1 1 0 1 0
1 0

Memoization

Why is this version of Fibonacci **bad**?

- Because it **repeat**s solving the same sub-programs.
- A waste of resources both in time and space.

Suggestion

- Solve each sub-problem only once, and use the result repeatedly.

Memoization

A straightforward example

```
function slow_example(x) {
    if (x > 100) {
        return 1;
    } else {
        return slow_example(x + 3) + slow_example(x + 3);
    }
}

slow_example(2);
```

Memoization

A straightforward example

```
function fast_example(x) {
    if (x > 100) {
        return 1;
    } else {
        return fast_example(x + 3) * 2;
    }
}

fast_example(2);
```

Memoization

A straightforward principle

- **DRY**!
- Don't repeat youself!

Significance

The ***DRY*** principle is the underlying reason for:

- abstraction/wishful thinking
- modular design
- memoization/dynamic programming
- ...

Memoization

Memoization

- How can we repeatedly use the results previously been computed?
- Store them and access the data whenever in need.

Problem...

- We need to store a lot of data.
- We need a proper data structure.

Memoization

To choose a proper data structure

- What to store: the results for every value of the function parameter, like `fibo(1)`, `fibo(2)`, `fibo(3)`, etc.
- How to store: store in a linear data structure, like array or table.
- When the function has 1 parameter, use 1D list/array.
- When the function has 2 parameters, use 2D list/array.
- ...

Memoization

List or array?

- List is better if we can store data incrementally, like 1, 2, 3, ...
- If we cannot store them one by one in the incremental order, then it will become meaningless when we access the data using `list_ref(lst, n)`.

Thus...

- We should choose to use array.
- After we solve a new problem, add `arr[n + 1]`.

Memoization

memoize

```
function memoize(func) {
    var arr = [];

    return function (x) {
        if (arr[x] != undefined) {
            return arr[x];
        } else {
            var result = func(x);
            arr[x] = result;

            return result;
        }
    };
}
```

Memoization

Problem here!

- For each element in `arr`, its index is the parameter `n`, the value is the return value `func(n)`.
- But, what if the values of the parameter is not "**positive integers**"?
    - Although JavaScript allows everything to be used as index, that is bad programming practice. It is not supported in other languages as well.

Memoization

Solution

- Create an abstract data structure, called *table* or *dictionary*.
- It has a lot of entries, just like array.
  - Each entry has an index and a value, just like array.
  - In fact, it should be implemented using array!
- The only difference: index does not have to be positive integers!

Memoization

Example

- The possible values of the parameter are -2, -1, 0, 1, 2, ...
  - Table will just use `arr[n + 3]` rather than `arr[n]`
- The possible values are 0.5, 1, 1.5 ...
  - Table will just use `arr[n * 2]` rather than `arr[n]`
- The possible values are ..., -3, -2, -1, 0, 1, 2, ...
  - How?

Memoization

Understanding

- *Table* or *dictionary* is simply a small improvement to array (by using a `map` on index).
- However, it is only helpful on some special cases.

What if possible values are all real numbers?

- *Table* or *dictionary* cannot help as well.

Memoization

To use table or dictionary

- Use `make_table()` rather than `var arr = []`
- Use `contains()` rather than `XXX !== undefined`
- Use `put()` rather than `arr[?]  = XXX`
- Use `lookup()` rather than `return arr[?]`

Memoization

memoize

```
function memoize(func) {
    var table = make_table();

    return function (x) {
        if (contains(x, table)) {
            return lookup(x, table);
        } else {
            var result = func(x);
            put(x, result, table);

            return result;
        }
    };
}
```

Memoization

memoize_2d

```
function memoize_2d(func) {
    var table = make_2d_table();

    return function (x, y) {
        if (contains(x, y, table)) {
            return lookup(x, y, table);
        } else {
            var result = func(x, y);
            put(x, y, result, table);

            return result;
        }
    };
}
```

Memoization

A few examples using memoization

- Fibonacci
- k-combination
- coin_change
- ...

Memoization

Fibonacci

```
function fibo(n) {
    if (n <= 1) {
        return n;
    } else {
        return fibo(n - 1) + fibo(n - 2);
    }
}
```

Think about it...

- Time/space complexity

Memoization

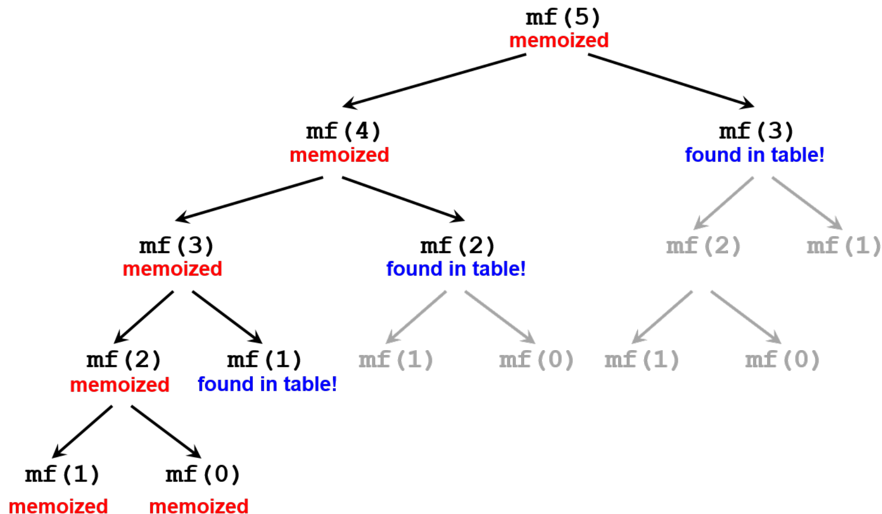
Use `memoize` to improve Fibonacci

```
var memo_fib = memoize(function (n) {
    return n <= 1 ? n : memo_fib(n - 1) + memo_fib(n - 2);
});
```

Reason

- Never solve the same sub-problem again.
- **DRY**!

Memoization



Memoization

Another k-combination

- No need to list all possible k-combinations.
- We only want to count the number of k-combinations.
- After that, we try to use `memoize` to improve it.

Thus...

- We do not care about the actual values for n items in the list.
- We use their indexes $1, 2, ..., n$ to represent them.

Memoization

k-combination

```
function k_combination(n, k) {
    if (k > n) {
        return 0;
    } else if (k === 0) {
        return 1;
    } else {
        return k_combination(n - 1, k - 1) +
               k_combination(n - 1, k);
    }
}
```

Memoization

Use memoize_2d to improve k-combination

```
var memo_k_combination = memoize_2d(function (n, k) {
    if (k > n) {
        return 0;
    } else if (k = 0) {
        return 1;
    } else {
        return memo_k_combination(n - 1, k - 1) +
               memo_k_combination(n - 1, k);
    }
});
```

Memoization

coin_change problem

- Find the number of ways to make changes.
- Still remember?

Memoization

coin_change problem

```
function coin_change(amount, kind) {
    if (amount === 0) {
        return 1;
    } else if (amount < 0 || kind === 0) {
        return 0;
    } else {
        return coin_change(amount, kind - 1) +
               coin_change(amount - value(kind), kind);
    }
}
```

Memoization

Use memoize_2d to improve coin_change

```
var memo_coin_change = memoize_2d(function (amount, kind) {
    if (amount === 0) {
        return 1;
    } else if (amound < 0 || kind === 0) {
        return 0;
    } else {
        return memo_coin_change(amount, kind - 1) +
               memo_coin_change(amount - value(kind), kind);
    }
});
```

Memoization

An interesting fact

- "memoization" is a domain-specific word.
- If you look it up in the dictionary, you cannot find it.
- A similar word is "memoris(z)ation". But we didn't misspell it.
- "memoization" is only used in Computer Science.

Domain-specific language (DSL)

- In CS, DSL is actually a kind of programming languages.
- *Google* this term and you will find some interesting things.

Memoization

Review: two approaches

- *Iteration*: the buttom-up approach;
- *Recursion*: the top-down approach.

Recall: why do we use array/table rather than list?

- We may not traverse in the incremental order $1, 2, ..., n$.
- Using `list_ref(lst, n)` is meaningless.

Memoization

Think about memoization again

- Is it the buttom-up approach or top-down approach?

Look at it...

```
var memo_fib = memoize(function (n) {
    return n <= 1 ? n : memo_fib(n - 1) + memo_fib(n - 2);
});
```

Memoization

Memoization & tabulation

- Memoization: top-down approach;
- Tabulation: buttom-up approach.

Data structure

- Memoization: table;
- Tabulation: table or list (array).

Memoization

To use tabulation

- To use tabulation, we will start from the smallest sub-problems.
- Then, we will solve larger and larger sub-problems until the whole problem has been solved.

Example

- If we use tabulation for Fibonacci, we will solve sub-problems in the incremental order, like `fibo(1)`, `fibo(2)`, `fibo(3)`, ...
- Due to the incremental order, we can also use list.

Memoization

Practical usage of memoization

- CPU cache
- SQL execution plan caching
- ...

Practical usage of tabulation

- Constant library
- ...

Memoization

Dynamic programming

- **Dynamic programming** (DP) is a technique for solving problems recursively and is applicable when the computations of the subproblems overlap.
- **Memoization** and **tabulation** are two approaches for DP.

Overview

1. Memoization
   - Inspiration
   - To use memoizationn
   - Memoization & tabulation
2. Object-oriented concepts
   - Class, object & instance

Object-oriented Programming

Our world...

- Our world is only a collection of objects.
- They have various states and behaviours.
- They belong to their own class.
- Objects in the same class are similar.
- ...

Object-oriented Programming

Object-oriented Programming

Terminology

- Class
- Object
- Instance
- Field
- Attribute
- Method
- Constructor
- Inheritance
- Polymorphism
- Override
- ...

Object-oriented Programming

Object in JavaScript

- Object in JavaScript is just a more generic version of *array*.
- It looks like

```
var obj = {"aa": 4,
           "bb": true,
           "cc": function(x) { return x * x; } };
```

Object-oriented Programming

Object in JavaScript

- Using object is really similar to using array.
- It looks like

```
obj["aa"];
obj["bb"];
obj["cc"](5);    // returns 25
```

Object-oriented Programming

Dot operator in JavaScript

- Dot operator is a shortcut for object accessor.
- Thus, it looks like

```
obj.aa;
obj.bb;
obj.cc(5);  // returns 25
```

Objects can become similar

- See these two objects

```
var smith = {
    "name": "Smith",
    "age": 35
}

var marc = {
    "name": "Marc",
    "age": 26
}
```

Constructor in JavaScript

- Constructor is a shortcut for building objects.
- Especially useful for building objects with similar structure.

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

var this_person = new Person("Smith", 35);
var that_person = new Person("Marc", 26);
```

Discussion Group Problems

Let's discuss them now.

The End