EBook Gratis

APRENDIZAJE Recursion

Free unaffiliated eBook created from
Stack Overflow contributors.

#recursion

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: recursion

It is an unofficial and free Recursion ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Recursion.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con la Recursión

Observaciones

La recursión se refiere a algo que se define en términos de sí mismo. En el contexto de la programación, es la práctica de usar funciones que se llaman a sí mismas o tipos recursivos.

Funciones recursivas

Hay dos partes en una función recursiva:

- Uno o más casos base
- Un paso recursivo

Debido a que una función recursiva se llama a sí misma, la cadena de llamadas de función podría continuar para siempre. Para asegurarse de que la función termina, se deben incluir los casos base. Estos le indicarán a la función cuándo devolver un valor en lugar de realizar otra llamada recursiva. Normalmente, las funciones recursivas verifican sus entradas en el caso base y se devuelven si se ha alcanzado el caso base. Si no se ha llegado al caso base, la función procederá al paso recursivo.

El paso recursivo es cuando una función se llamará a sí misma. Muchas funciones recursivas devolverán algunos cálculos realizados en el resultado de la llamada recursiva. El ejemplo de secuencia de Fibonacci demuestra esto. Esa función recursivamente se llama a sí misma dos veces y luego devuelve la suma de los resultados de esas llamadas

Un uso común de la recursión es navegar y realizar cálculos en grandes estructuras de datos, como árboles y gráficos. Esto funciona bien porque la recursión rompe el problema para resolver problemas más pequeños y se basa en esas soluciones para resolver el problema más grande. Otro uso común para la recursión es realizar operaciones iterativas (bucles) en idiomas (típicamente lenguajes funcionales) que no tienen estructuras de bucles integradas.

Tipos recursivos

Un tipo recursivo es aquel cuyos valores están compuestos de valores del mismo tipo, o expresados de manera diferente: un tipo recursivo se define en términos de sí mismo. Por ejemplo, una lista es un tipo recursivo, ya que cada subconjunto de la lista es en sí misma una lista. Los árboles son otro ejemplo, ya que los nodos se eliminan de un árbol, la construcción restante sigue siendo un árbol.

Formalmente, el conjunto de valores de un tipo recursivo, T , se definirá mediante una ecuación de conjunto recursivo en la forma:

$T = ... T ...$

Una ecuación de conjunto recursiva puede tener muchas soluciones, pero esa ecuación siempre tiene una solución mínima que es un subconjunto de todas las demás soluciones.

Como ejemplo práctico, considere esta definición de Haskell para un tipo de lista,

```
data List a = Nil | Cons a (List a)
```

lo que significa que una lista de `a` 's es una lista vacía o una celda de contras que contiene una' `a` '(la "cabecera" de la lista) y otra lista (la "cola").

Examples

Secuencia De Fibonacci Usando Recursion En Javascript

```
// Returns the nth number in the Fibonacci sequence
function fibonacci(n) {
    if (n === 1 || n === 0) {
        return 1;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

Comprobación de palíndromos con recursión en C ++

```
// Checks a string to see if it is a palindrome
bool function IsPalindrome(string input) {
    if (input.size() <= 1) {
        return true;
    }
    else if (input[0] == input[input.size() - 1]) {
        return IsPalindrome(input.substr(1,input.size() - 2));
    }
    else {
        return false;
    }
}
```

Crear una estructura de jerarquía usando Recursión [JavaScript]

```
var directories = [
  {name: 'users' , parent : null },
  {name: 'distalx' , parent : 'users' },
  {name: 'guest' , parent : 'users' },
  {name: 'shared' , parent : 'users' },
  {name: 'documents' , parent : 'distalx' },
  {name: 'music' , parent : 'distalx' },
  {name: 'desktop' , parent : 'distalx' },
  {name: 'javascript' , parent : 'documents' },
  {name: 'funjs' , parent : 'documents' },
  {name: 'functions' , parent : 'documents' }
]
```

```
var sortDirectories= function(directories, parent){
  let node = [];
  directories
  .filter(function(d){ return d.parent === parent})
  .forEach(function(d){
    var cd = d;
    cd.child = sortDirectories(directories, d.name);
    return node.push(cd);
  })
  return node;
}

var results = sortDirectories(directories, null);
JSON.stringify(results, null, ' ');
```

salida

```
[{
  "name": "users",
  "parent": null,
  "child": [
      { "name": "distalx",
        "parent": "users",
        "child": [
            { "name": "documents",
              "parent": "distalx",
              "child": [
                  { "name": "javascript",
                    "parent": "documents",
                    "child": []
                  },
                  { "name": "funjs",
                    "parent": "documents",
                    "child": []
                  },
                  { "name": "functions",
                    "parent": "documents",
                    "child": []
                  }
              ]
            },
            { "name": "music",
              "parent": "distalx",
              "child": []
            },
            { "name": "desktop",
              "parent": "distalx",
              "child": []
            }
        ]
      },
      {
        "name": "guest",
        "parent": "users",
        "child": []
```

```
      },
      {
        "name": "shared",
        "parent": "users",
        "child": []
      }
  ]
}]
```

Buscando un valor en un árbol de búsqueda binario con recursión en C / C ++

Estructura para nodos de árbol de búsqueda binario (BST):

```
struct node {
    int data;
    node * left;
    node * right;
}
```

Algoritmo de búsqueda

```
// Check if a value exists in the tree
bool BSTSearch(node * current, int value)
{
    if (current->data == value)
    {
        return true;
    }
    else if (current->data < value)
    {
        if (current->left == NULL)
        {
            return false;
        }
        else
        {
            return BSTSearch(current->left, value);
        }
    }
    else if (current->data > value)
    {
        if (current->right == NULL)
        {
            return false;
        }
        else
        {
            return BSTSearch(current->right, value);
        }
    }
}
```

Obtener todas las combinaciones de elementos de matrices en Javascript

```
function getCombinations(params, combinationsResults){
    if(params.length == 0) return combinationsResults;
```

```
    var head = params[0];
    var tail = params.slice(1);
    var combinationsResultsCurrent = [];
    if(Array.isArray(head)){
        _.uniq(head).forEach(function(item){
            if(combinationsResults.length == 0){
                combinationsResultsCurrent.push(item);
            } else {
                combinationsResults.forEach(function(previousResultItem){
                    combinationsResultsCurrent.push(previousResultItem.concat([item]));
                });
            }
        });
    } else {
        if(combinationsResults.length == 0){
            combinationsResultsCurrent.push(head);
        } else {
            combinationsResults.forEach(function(previousResultItem){
                combinationsResultsCurrent.push([previousResultItem].concat([head]));
            });
        }
    }
    return getCombinations(tail, combinationsResultsCurrent);
```

ejemplo:

Supongamos que tenemos una consulta con caluses IN:

```
SELECT * FROM custom_table WHERE user_id = 'user1' AND location IN ('home', 'work', AND date
IN ('2017-01-10', '2017-01-11'))
```

Y quisiera obtener todas las combinaciones de parámetros para generar consultas sin condiciones IN:

```
SELECT * FROM custom_table WHERE user_id = [value for user_id] AND location = [value for
possible location] AND date = [value for possible date]
```

Para obtener todas las combinaciones posibles de parámetros para consultas equivalentes, podemos ejecutar la función anterior:

```
var params = ['user1', ['home', 'work'], ['2017-01-10', '2017-01-11']];
```

Lea Empezando con la Recursión en línea:
https://riptutorial.com/es/recursion/topic/2682/empezando-con-la-recursion

Creditos

S. No	Capítulos	Contributors
1	Empezando con la Recursión	Ayan, Community, distalx, Filip Allberg, Honza Brabec, Niko Gamulin, Rainbacon