数据结构算法

1. 数组
2. 创建数组 var a =[] 数组长度 array.length
3. Push 添加到数组末尾 unshift 插入到数组首位
4. Pop 删除数组中最靠后元素 shift 删除数组最后一个元素
5. 输出数组为字符串 toString join (分隔符)

栈和队列：数据结构类似于数组，但在添加和删除数组元素时更为可控。、

1. 栈

栈后进先出的有序集合。新添加的或者待删除的都保存在栈的末尾，称作栈顶，另一端是栈顶。

为栈声明方法：

栈例子：一堆书

Push(element(s)) 添加一个或者多个元素到栈顶 pop:移除栈顶元素，同时返回被移除的元素。 Peek：返回栈顶元素 isEmpty 栈内元素是否为空

1. 队列

队列是遵循FIFO( first in first out,先进先出)原则一组有序的项。

声明队列方法：

Enqueuer(element(s)) 向队列尾部添加一个（或多个）新的项。

Dequeue 移除队列的第一项，并返回被移除的项。

Front 返回队列中第一个元素

队列例子：电影院排队

1. 链表 链表存储有序的元素集合，不同于数组，链表中的元素在内存中并不是连续放置的。每个元素由一个存储元素本身的节点和一个指向下一个元素的引用（也称指针或链接）组成。

相对于数组，链表的好处在于，添加或删除元素的时候不需要移动其他元素。

链表例子：寻宝游戏，火车

1. .集合 set

Add(value) 向集合添加一个新的项

Remove(value) 从集合移除一个值

Has(value) 如果值在集合中，返回true.否则返回false

Clear() 移除集合中所有项

Size 返回中的所有项

Vlues 返回一个包含集中所有值的数组

1. 字典和散列表

字典：类似于map.和set的部分方法重合。 散列表 散列函数的表 hashmap .类似于hash的map,.

集合、字典和散列表可以存储不重复的值。在集合中，我们感兴趣的是值本身，并把它作为主要元素，在字典中，我们用[键，值]来存储数据，在散列表中也是一样。

字典也称作映射，字典存储[键，值]对。键名用以查询特定元素。

与Set相似，ES6同样包含了一个Map类的实现，即字典。

映射字典的方法：

Set(key,value)向字典中添加新的元素、

Remove(key) 通过键值从字典中移除键值对应的数据值

Has(key) 如果这个键值存在于这个字典中，返回true,否则false.

Get(key) 通过键值查找特定的数值返回

Clear() 将这个字典中的所有元素全部删除

Size() 返回字典中包含元素的数量。

Keys() 将字典所包含的所有键名以数组形式返回。

Values() 将字典所包含的所有数值以数组形式返回，。

散列表

HashTable类，也叫HashMap类，是Dictionary类的一种散列表实现方式。

散列算法的作用是尽可能快地在数据结构中找到一个值。散列函数的作用是给定一个键值，然后返回值在表中的地址。

基础方法：

Put(key,value) 向散列表中增加一个新的项，也能更新散列表

Remove(key) 根据键值 从散列表中移除值

Get(key) 返回根据键值检索到的特定的值

使用hashtable类：

Var has=new Hashtable();

Hash.put(‘gan’,sedrt);

散列集合：由一个集合构成，但是插入、移除或获取元素时，使用的是散列函数。

处理散列表的冲突：

分离链接: 为散列表的每一个位置创建一个链表并将元素存储在里面。

线性探查 当向表中某个位置加入一个新元素的时候，如果索引为index的位置已经被占据了。就尝试index+1的位置。

双散列法

例子：一个实际的字典和地址簿。

1. 树 树对于存储需要快速查找的数据非常有用

树的相关术语

一个树结构包含一系列存在父子关系的节点，每个节点都有一个父节点（除了顶部的第一个节点）以及0个或多个子节点。

树的深度取决于它的祖先节点的数量。树的高度取决于所有节点深度的最大值。

二叉树和二叉搜索树

二叉树的节点最多有两个，左侧子节点和右侧子节点。

二叉搜索树（BST）是二叉树的一种，但是它只允许你在左侧节点存储（比父节点）小的值，在右侧节点存储（比父节点）大于等于的值。

和链表一样，将通过指针来表示节点之间的关系术语称其为边。对于树，也是用两个指针，一个指向左侧节点，一个指向右侧节点。

在树类实现的方法：

Insert(key) 向树中插入一个新的键。

Search(key) 在树中查找一个键，如果节点存在，则返回true.

InOrderTraverse 通过中序遍历方式遍历所有节点

preorderTraverse 通过先序遍历方式遍历所有节点。

PostOrderTraverse 通过后序遍历方式遍历所有节点。

Min 返回树中最小的值。

Max 返回树中最大的值

Remove（key移除某个键

中序遍历 是一种以上行顺序访问BST所有节点的遍历方式，也就是以最小到最大的顺序访问节点。

先序遍历 以优先于后代节点的顺序访问每个节点。先序遍历的一种应用是打印一个结构化的文档。

后序遍历 先访问节点的后代节点，再访问节点本身。

例子：家谱 ，公司架构

1. 非线性数据结构 图

图是网络结构的抽象模型。图是一组由边连接的节点（或顶点）.

一个图 G=(V,E)又以下元素构成。

V:一组顶点

E: 一组边，连接V中的顶点

由一条边连接在一起的顶点称为相邻顶点。 一个顶点的度是其相邻顶点的数量。路径是顶点 v1,v2…..vn的连续序列。

简单路径要求不包含重复的顶点。

图的遍历： 深度优先 广度优先算法

算法 数据结构 描述

深度优先算法 栈 通过将顶点存入栈中，顶点是沿着路径被探索的，存在新的相邻顶点就去访问。

广度优先算法 队列 通过将顶点存入队列中，最先入队列的顶点先被探索。

广度优先搜索：广度优先搜索会从指定的第一个顶点开始遍历图，先访问其所有的相邻点，就像一次访问图的一层。 也就是先宽后深地访问顶点。

从顶点V开始的广度优先搜索算法所遵循的步骤。

1. 创建一个队列Q
2. 将v标注为被发现的（灰色），并将v入队列Q
3. 如果Q非空，则运行以下步骤：
4. 将 u从Q中出队列
5. 将标注U为被发现的（灰色）
6. 将 u所有未被访问过的邻点（白色）入队列
7. 将u标注为已被探索的（黑色）。

深度优先搜索

深度优先搜索算法将会从第一个指定的顶点开始遍历图，沿着路径直到这条路径最后一个顶点被访问了，接着原路退回并探索下一条路径。 先深度后广度

访问顶点v,照如下步骤：

1. 标注v为被发现的（灰色）。
2. 对于v的所有未被访问的邻点w：
3. 访问顶点w.
4. 标注v为已被探索的（黑色）

深度优先搜索的步骤是递归的。

3.排序算法 冒泡排序

冒泡排序： 比较两个相邻的项，如果第一个比第二个大，则交换它们。元素项向上移动至正确地顺序，就好像气泡升至表面一样，冒泡排序因此得名。

this.bubbleSort = function(){

var length = array.length; //{1}

for (var i=0; i<length; i++){ //{2} //假设数组长度为5，则循环5次

for (var j=0; j<length-1; j++ ){ //{3} //假设数组长度为5，则循环4次

if (array[j] > array[j+1]){ //{4} //那么该数组总共循环了5\*4为20次

swap(j, j+1); //{5}

}

}

}

};

var swap = function(index1, index2){

var aux = array[index1];

array[index1] = array[index2];

array[index2] = aux;

};

改进之后。 内循环中减去外循环中已跑过的轮数，避免内循环中所有不必要的比较。比如数组中最大值为5，已经在最后一个位置了，下一次不必对它进行排序

this.modifiedBubbleSort = function(){

var length = array.length;

for (var i=0; i<length; i++){

for (var j=0; j<length-1-i; j++ ){ //{1}

if (array[j] > array[j+1]){

swap(j, j+1);

}

}

}

};

10.1.2 选择排序

选择排序算法是一种原址比较排序算法。选择排序大致的思路就是找到数据结构中的最小值并将其放置在第一位，接着找到第二小的值并将值放在第二位，以此类推。

This.selectionSort=function()[

Var length=array.length;

indexMin;

for(var i=o;i<lenth;i++){

indexMin=I;

for(var j=i;j<lenth-1;j++){

if(array[indexMin]> array[j]){

indexMin=j;

}}

If(i!=indexMin){

Swap(i,indexMin);

}

首先声明一些算法中使用的变量，接着外循环迭代数组，并控制迭代轮次（数组中的第n个值—下一个最小值）。我们假设本迭代轮次的第一个值为数组最小值，然后，从当前i的值开始至数组结束，我们比较位置j的值比当前最小值小，如果是，则改变最小值至新最小值，当内循环结束，将得出数组第n小的值，最后，如果该最小值和原最小值不同，则交换其值。

10.1.3 插入排序

插入排序每次比较一个数组项，以此方式构建最后的数组。

This.insertionSort=function(){

Var length=array.length;

j,temp;

for(var i=1;i<length;i++){

j=I;

temp=array[i];

while(j>0&&array[j-1]>temp){

array[j]=array[j-1];

j--;

}

array[j]=temp;

}

};

10.1.4 归并排序

归并排序是一个可以被实际使用的排序算法，前三个排序算法性能不好，归并排序性能不错，其复杂度为O（nlog n）

归并排序是一种分治算法，其思想是将原始数组切分成较小的数组，直至每个小数组只有一个位置，接着讲小数组归并成较大的数组，直到最后一个只有一个排序完毕的大数组。

由于是分治法，归并排序也是递归的：

This.mergeSort=function(){

Array=mergeSortRec(array);

};

Var mergeSortRec=function(array){

Var length=array.length;

If(length===1){

Return array;

}

Var mid=Math.floor(length/2),

Left=array.slice(0,mid),

Right=array.slice(mid,length);

Return merge

}

}

搜索

10.2.1 顺序搜索

顺序或线性搜索是最基本的搜索算法。它的机制是，将每一个数据结构中的元素和我们要找的元素做比较。顺序搜索是最低效的一种搜索算法。

This.sequentialSearch=function(item){

For(var i=0;i<array.length;i++){

if(item===array[i])

return I;

}

Return -1;

};

2.2二分搜索

二分搜索算法的原理和猜数字游戏类似。

这个算法要求被搜索的数据结构已排序。以下是算法步骤：

1. 选择数组的中间值
2. 如果选中值是带搜索值，则算法执行完毕
3. 如果待搜索值比选中值要小，则返回步骤1并在选中值左边的子数组中寻找。
4. 如果待搜索值比选中值要大，则返回步骤1并在选中值右边的子数组中寻找。

This.binarySearch=function(item){

This.QuickSort();

Var low=0;

High=array.length-1;

mid,element;

While(low<=high){

Mid=Mith.floor(low+high)/2);

Element=array[mid];

If(element<item)

{ low=mid+1;}

Else if(elemnt>item)

high=mid-1;

}else{return mid; }

}}

Return -1;

}10 算法补充知识

递归

递归是一种解决问题的方法，它解决问题的各小部分，直至最终解决问题的大部分。递归通常涉及函数调用自身。

如果忘记加上用以停止函数递归调用的边界条件，递归并不会无条件地执行下去；浏览器会抛出错误，也就是所谓的栈溢出错误。

斐波那契数列

斐波那契数列： 1和 2的数是1.

N(n>2的斐波那契数是（n-1的斐波那契数加上（n-2）的斐波那契数列。

递归更易理解。

大O表示法，描述算法的性能和复杂程度

如何衡量算法的效率？ 通常是用资源，例如CPU(时间)占用、内存占用、硬盘占用、网络占用。

1. O(1)

Function add(num){ return ++num}

O(1) 执行效率与参数无关 常数

1. O(n)

顺序算法为例

This.sequentialSearch=function(item){

For(var i=0;i<array.length;i++){

if(item===array[i])

return I;

}

Return -1;

};

时间复杂度为O(n),n是输入数组的大小。

1. O(n2)

用冒泡排序作为例子

this.modifiedBubbleSort = function(){

var length = array.length;

for (var i=0; i<length; i++){

for (var j=0; j<length-1-i; j++ ){ //{1}

if (array[j] > array[j+1]){

swap(j, j+1);

}

}

}

};

如果用大小为10的数组执行bubbleSort,开销是100（10的平方），如果大小为100，开销是10000（100的平方）。时间复杂度为 0 (n 2)

如果三层嵌套循环，那么时间复杂度为O（n 三次方）

1. 时间复杂度比较

O(log(n)) 对数 O(log(n)c) 对数多项式 O(N) 现行的 O(N2) 二次的 O(N C) 多项式

