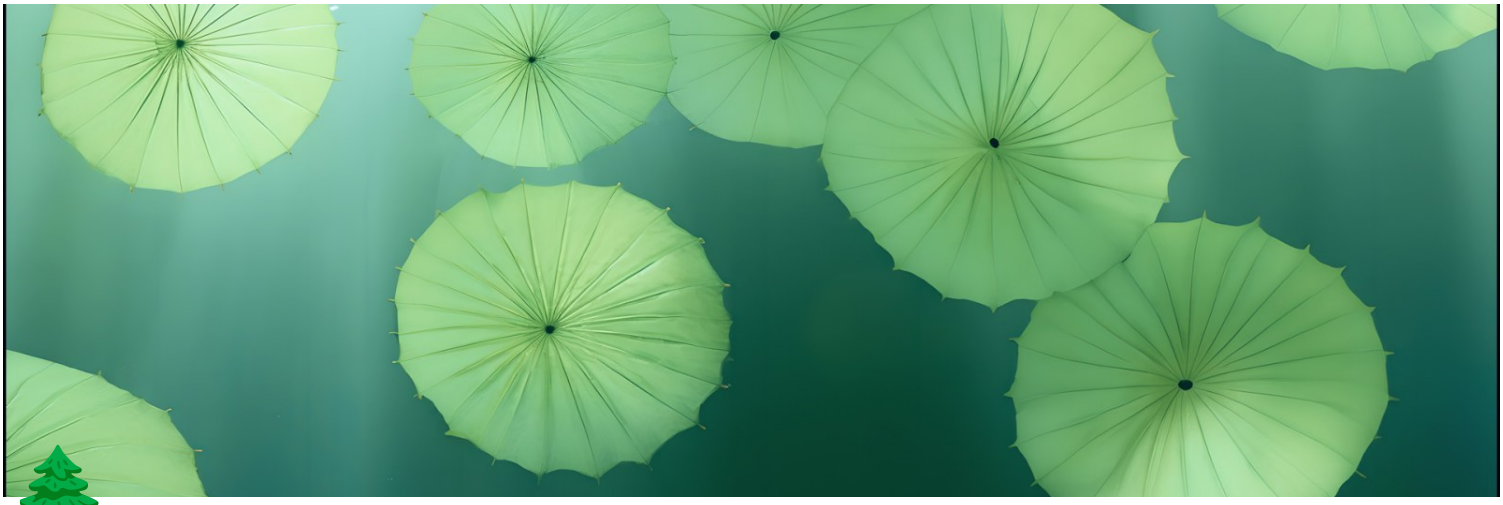

数据结构

一、数组array

- 数组是一种线性数据结构，用于存储同类型的数据集合。
- 掌握数组的声明、初始化、访问、遍历和常见操作（如插入和删除）。

二、栈与队列 stack/queue

`stack` 与 `queue` 被称为 `duque` 的配接器，其底层是以deque为底部架构，通过deque执行具体操作。

`stack`：先入后出

`queue`：先入先出

三、堆与优先队列

`heap`：建立在完全二叉树上，分为两种：大根堆和小根堆。其在STL中做priority_queue的助手

`priority_queue`：其内的元素不是按照被推入的顺序排列，而是自动取元素的权值排列，确省情况下利用一个max-heap完成，后者是以vector一表现的完全二叉树。`priority_queue` 基于大根堆实现，大根堆是一个完全二叉树，其中每个父节点的值都大于或等于其子节点的值。

- 插入和删除操作的时间复杂度是 `O(logn)`

四、vector

`底层实现`：Vector在堆中分配了一段连续的内存空间来存放元素（线性空间），是一种动态的数据结构

三个 `迭代器`：

1. first：指向的是vector中对象的起始字节位置
2. last：指向当前最后一个元素的末尾字节
3. end：指向整个vector容器所占用内存空间的末尾字节

扩容过程：如果集合已满，在新增数据的时候，就要分配一块更大的内存，将原来的数据复制过来，释放之前的内存，在插入新增的元素

两种扩容方式：

1. 固定扩容：空间利用率高，但时间复杂度高。
2. 加倍扩容：时间复杂度低，但空间利用率也低。

在实际应用中，一般采用空间换时间的策略。

相关函数：

- capacity()，在不分配新内存下最多可以保存的元素个数。
- size()，返回当前已经存储数据的个数。对于vector来说，capacity是永远大于等于size的，capacity和size相等时，vector就会扩容。
- resize()：改变当前容器内含有元素的数量(size())，而不是容器的容量。
- reserve()：改变当前容器的最大容量（capacity）。

五、链表list

底层实现：每个元素都是放在一块内存中，他的内存空间可以是不连续的，通过指针来进行数据的访问。

特点：在哪里添加删除元素性能都很高，不需要移动内存，也不需要对每个元素都进行构造与析构了，所以常用来做随机插入和删除操作容器。

list随机访问性能差，插入删除性能好；vector相反

list属于双向链表，其结点与list本身是分开设计的。

七、双端队列deque

底层实现：deque是一个双端开口的连续线性空间，其内部为分段连续的空间组成，随时可以增加一段新的空间并链接。其采用一块map作为主控，其中每个元素都是指针，指向另一片连续的线性空间，称之为缓存区，这个区里面才是存储数据的。

- 优点：没有vector容器的“开辟内存、复制、释放”等操作，整体连续，并且提供随机访问的接口；提供两端操作，灵活性高。
- 缺点：迭代器需要处理内部跳转，比较复杂。

八、map/set/unordered_

map和set底层都是基于红黑树，但map存储的是键值对，set只存储键。

- 优点：查找某一个数的时间为O(logn)；遍历时采用iterator，效果不错。
- 缺点：每次插入值的时候，都需要调整红黑树，效率有一定影响。

unordered_map和unordered_set底层则是基于哈希表实现的，因此其元素的排列顺序是杂乱无序的。如果你需要有序性或者对性能的稳定性有较高要求，map可能是更好的选择。如果你不关心顺序，并且对平均情况下的性能有较高要求，unordered_map可能更适合。

set：用来判断某一个元素是不是在一个组里面。

map的优缺点：

- 优点：有序性，这是map结构最大的优点。元素的有序性在很多应用中都会简化很多的操作，例如map的查找、删除、增加等一系列操作时间复杂度稳定，都为 `O(logn)`。
- 缺点：查找、删除、增加等操作平均时间复杂度较慢，与n相关。

`unordered_map` 的优缺点

- 查找、删除、添加的速度快，时间复杂度平均仅为 `O(1)`；
- 因为unordered_map内部基于哈希表，以 `（key,value）` 对的形式存储，因此空间占用率高。

九、树Tree

1、平衡二叉树AVL

是一种特殊的二叉排序树，其左右子树都是平衡二叉树，且左右子树高度之差不超过1。

2、红黑树

是一种二叉查找树，但是在每个节点上增加一个存储位记录节点的颜色，非黑即红。通过节点颜色的限制，红黑树保证从根到叶子的最长路径不超过最短路径的2倍，因此它是一种弱平衡二叉树。

相比于真正的AVL树，红黑树旋转次数更少，因此查找、插入、删除性能更高。

`特点`：

1. 节点非黑即红；
2. 根节点和每个叶节点都是黑的；
3. 若一个节点是红色的，则其子节点必然是黑色的；
4. 任意节点到叶子节点NULL指针的每条路径都包含相同数目的黑色节点。

3、字典树

`特点`：

1. 根节点不保存字符，其他节点保存一个字符；
2. 根节点到某一节点，经过的字符连接起来就是该节点存储的字符串；
3. 每个节点的子节点包含的字符都不同。

`优点`：利用公共字符前缀减少查找时间，减小无谓的字符串比较，适合统计、排序和保存大量字符串。

- 二叉树的遍历：

```
void dfs(TreeNode* root, vector<int>& v)//前序遍历
{
	if(root==nullptr)	return;
	v.push_back(root->val);
	calcuTree(root->left, v);
	calcuTree(root->right, v);
}
void dfs(TreeNode* root, vector<int>& v)//中序遍历
{
	if(root==nullptr)	return;
	calcuTree(root->left, v);
	v.push_back(root->val);
```

```
        calcuTree(root->right, v);
}
void dfs(TreeNode* root, vector<int>& v)//后序遍历
{
        if(root==nullptr)   return;
        calcuTree(root->left, v);
        calcuTree(root->right, v);
        v.push_back(root->val);
}
vector<int> postorderTraversal(TreeNode* root) {
    vector<int> v;
    dfs(root, v);
    return v;
}
```

4、二叉搜索树

- 中序遍历：节点递增；
- 不存在相同值的节点。

十、图graph

- 图由节点和边的集合组成，用于表示实体之间的关系。
- 掌握图的表示方法，如邻接矩阵和邻接表。

十一、哈希表Hash Table

哈希函数是一种映射关系，根据关键词Key，经过一定的函数关系计算找到元素的位置。

`哈希冲突/哈希碰撞`：不同的key通过相同的哈希函数计算后，得到相同的哈希地址。

- 哈希表通过哈希函数将键映射到表中的位置。
- 掌握哈希表的基本操作，如添加、删除和查找。