**算法笔记**

**数组**

**静态数组**

1. 在内存中分配一块连续、大小固定的内存。静态数组的长度是预先定义好的，在整个程序中，一旦给定大小就无法改变。因此静态数组不适合插入，删除等操作。
2. 静态数组插入或删除的办法为重新申请一块新的内存，将其他元素复制到新内存的过程中进行插入或删除操作，因此耗费时间随数组的大小呈线性变化。

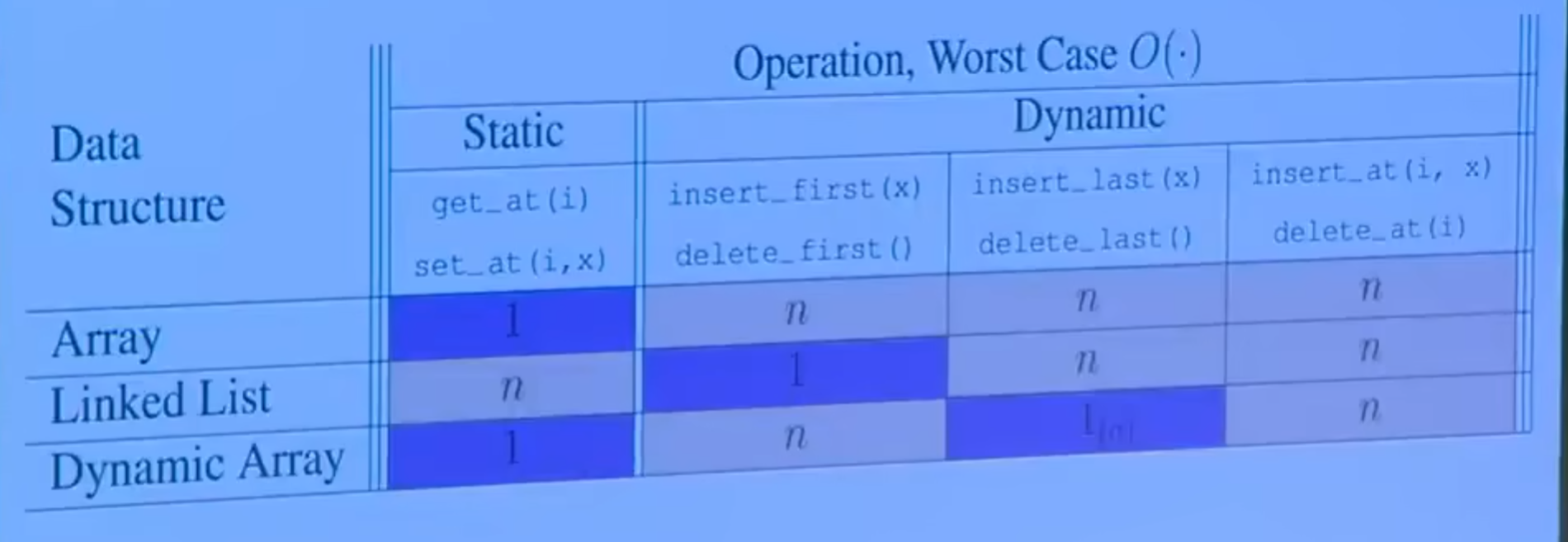
**动态数组**

当元素大小接近容器长度时。将容器扩容为原来的两倍。可能某些操作耗费的时间较多，平均所有操作下来耗费的时间最少。

**链表**

1. 插入与删除十分高效。通过改变节点中指针域所存储的地址快速进行插入与删除，耗费时间几乎为常量。
2. 访问元素无法直接访问，必须通过头指针开始一级一级访问，耗费时间随所访问元素在链表中的位置呈线性变化。
3. 链表一般有头指针以及尾指针，尾指针可以快速访问到链表中最后一个元素。

**链表、静态数组、动态数组的优劣**



**Set**

通过关键字key来查询Set中的数据内容。

1. 无序：想要进行查询只能遍历
2. 有序：将其中的数据排序后通过二分进行查找

**线性排序**

**二叉树**

一棵二叉树是结点的一个有限集合，该集合或者为空，或者是由一个根节点加上两棵别称为左子树和右子树的二叉树组成。  
二叉树的特点：

1. 每个结点最多有两棵子树，即二叉树不存在度大于2的结点。
2. 二叉树的子树有左右之分，其子树的次序不能颠倒。
3. 一个节点的高度是从该节点向下到叶子结点的最长路径的边数

**Trie树**

概念：字典树（TrieTree），是一种[树形结构](https://so.csdn.net/so/search?q=%E6%A0%91%E5%BD%A2%E7%BB%93%E6%9E%84&spm=1001.2101.3001.7020)，典型应用是用于统计，排序和保存大量的字符串（但不仅限于字符串,如01字典树）。主要思想是利用字符串的**公共前缀**来节约存储空间。很好地利用了串的公共前缀，节约了存储空间。字典树主要包含两种操作，**插入**和**查找**。

**字典树代码实现方法**

**通过二维数组实现**

在用代码实现字典树时，我们主要需要实现两种操作：即录入单词和查找单词.

1.我们用一个二维数组来实现字典树的建树。先把代码写出来，之后再解释。

const int N = 1000050;

int trie[N][26];

int cnt[N];

int id; //id代表字典树中每一个节点的编号，id的大小只与插入字典树的先后顺序有关

void insert(string s)

{

int p = 0;

for (int i = 0; i < s.size(); i++)

{

int x = s[i] - 'a';

if (trie[p][x] == 0) trie[p][x] = ++id;

p = trie[p][x];

}

cnt[p]++;

}

int find(string s)

{

int p = 0;

for (int i = 0; i < s.size(); i++)

{

int x = s[i] - 'a';

if (trie[p][x] == 0)return 0;

p = trie[p][x];

}

return cnt[p];

}

trie[N][26]： 每个trie代表一条边，字典树其中1~N为边上方节点的编号，0代表root节点，1~26为连在i节点下方的26个字母。如果trie[i][x]=0,则代表字典树中目前没有这个点，而trie[i][x]的值代表这个点下方连有的点的编号，例如：trie[i][2]=9代表第i号点和的下方连有一个点‘c’，并且那个点的编号是9，为什么是c呢？因为 ‘c’-‘a’=2

**cnt[N]:** cnt[i]==0代表编号为i的点不是一个单词的结束点，在上面的图中代表这个点不是空点，但是没有标红，cnt[i]！=0代表编号为i的点是一个单词的结束点，即红点。cnt[i]不一定只为0或1，因为有可能多次输入了同一个单词。

p代表查询与插入时的不断变化的当前节点编号，初始化为0，代表初始节点，在函数的循环中，我们首先用x确定接下来要找的字母，再通过变量x确定了接下来我们需要查找当前节点下是否有连接着目标字母的节点。通过每次确定的x，我们通过trie[p][x] 查找连着目标字母的节点的编号，如果目标节点存在，就把p更新成目标节点的编号（p = trie[p][x]）。而如果trie[p][x] == 0，代表字典树中没有这个点，如果是查找就代表没有这个单词，查找失败。而如果是插入函数，我们就用 ++id 来把这个点存进字典树。我们在两个函数的最后用cnt[p]来涂红节点或返回节点值。

原文链接：https://blog.csdn.net/qq\_49688477/article/details/118879270

**通过指针数组实现**

leetcode:208 实现Trie树（前缀树）

class Trie {  
private:  
 vector<Trie\*> children;//通过动态数组对每个节点的位置进行存储  
 bool isEnd;//判断该节点是否为一个单词的结束处  
 Trie\* searchPrefix(string prefix) {  
 Trie\* node = this;  
 for (char ch : prefix) {  
 ch -= 'a';  
 if (node->children[ch] == nullptr) {  
 return nullptr;  
 }  
 node = node->children[ch];  
 }  
 return node;  
 }  
  
public:  
 Trie() : children(26), isEnd(false) {}  
 void insert(string word) {  
 Trie\* node = this;  
 for (char ch : word) {  
 ch -= 'a';  
 if (node->children[ch] == nullptr) {  
 node->children[ch] = new Trie();  
 }  
 node = node->children[ch];  
 }  
 node->isEnd = true;  
 }  
 bool search(string word) {  
 Trie\* node = this->searchPrefix(word);  
 return node != nullptr && node->isEnd;  
 }  
 bool startsWith(string prefix) {  
 return this->searchPrefix(prefix) != nullptr;  
 }  
};  
作者：力扣官方题解  
链接：https://leetcode.cn/problems/implement-trie-prefix-tree/solutions/717239/shi-xian-trie-qian-zhui-shu-by-leetcode-ti500/