题目讲解

- 设计一个队列
- 支持：出队，入队，求最大元素
- 要求O(1)
- 均摊分析
- 要点：新开一个辅助队列，维持其单调性

题目讲解

- 给定一个正整数组a，是否能以3个数为边长构成三角形？
- 即是否存在不同的i，j，k，
- 满足 a[i] < a[j] + a[k]
- 并且 a[j] < a[i] + a[k]
- 并且 a[k] < a[i] + a[j]
- 要点：假设a[i] < a[j] < a[k], 不等式如何化简？

题目讲解

Leetcode 152 Maximum Product Subarray

- 要点：
- A*B ~ lg (A*B) = lg(A) + lg(B)
- 积化和，注意正负号

必知必会的数据结构

Ben

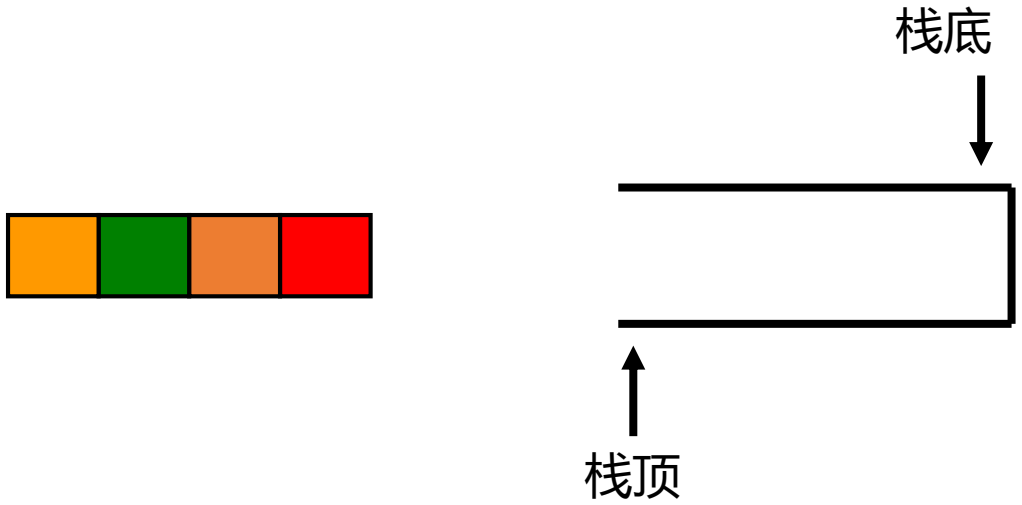
Outline

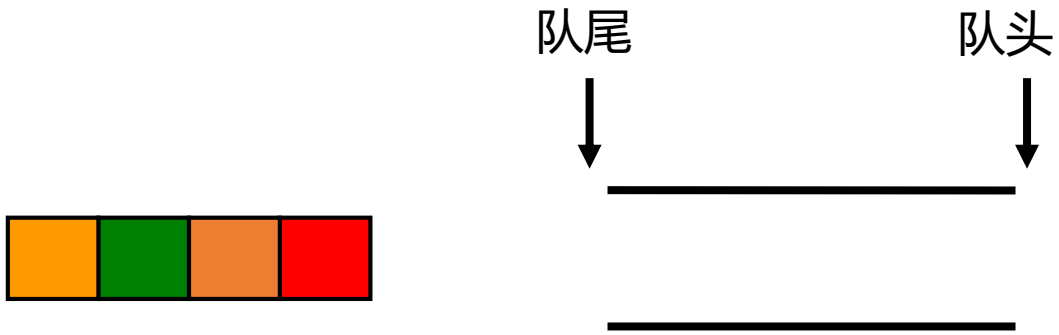
- 栈和队列
- 数据结构——平衡的艺术
  - 哈希表
  - 布隆过滤器

栈和队列：定义

- 存放数据的线性表
- 操作：入栈/队列、出栈/队列、判断满/空
- 空间复杂度：O(n)
- 单次操作时间复杂度：O(1)
- 区别
  - 先进后出（FILO, First In Last Out）
  - 先进先出（FIFO, First In First Out）

栈和队列：定义





栈和队列：实现

- 数组和链表皆可（线性表）
- 指针（辅助变量）
  - 栈顶/底指针
  - 队头/尾指针
- 关键：出入元素的同时移动指针

栈的应用：括号匹配检测

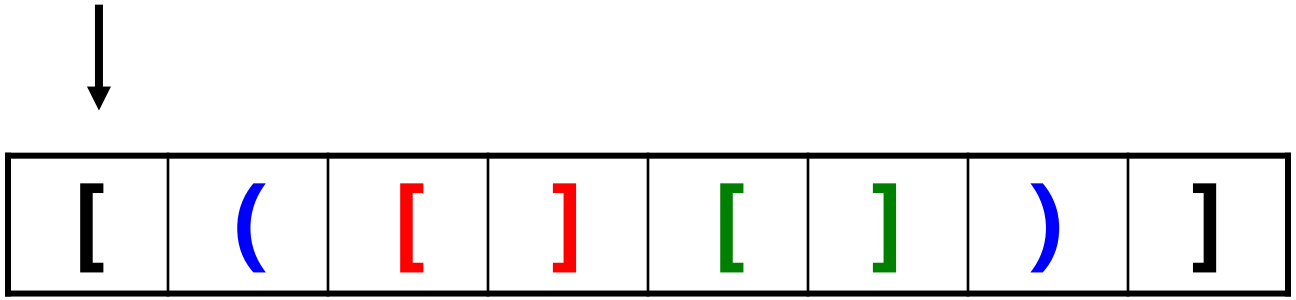
- 括号、引号等符号是成对出现的，必须相互匹配
- 设计一个算法，自动检测输入的字符串中的括号是否匹配
- 比如：
  - {}[([][])]　　匹配
  - [(])　　不匹配
  - (()]　　不匹配

[([ ] [ ])]

栈的应用：括号匹配检测

思考

- 从左向右扫描字符串

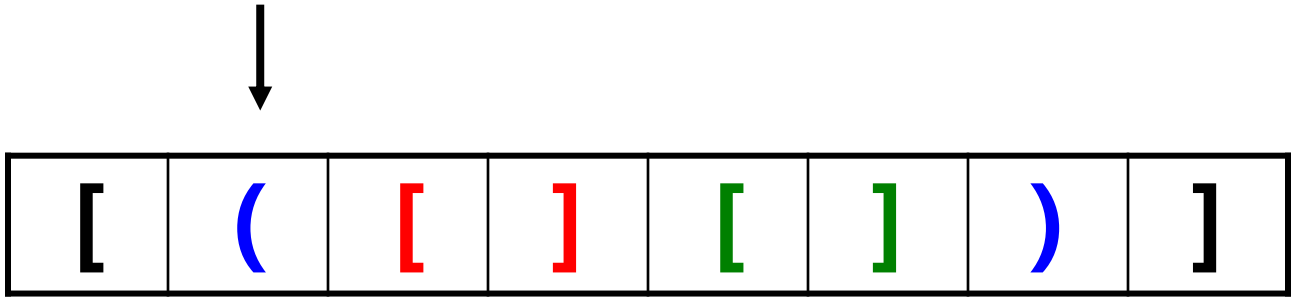


当前是[，期待一个]

栈的应用：括号匹配检测

思考

- 从左向右扫描字符串

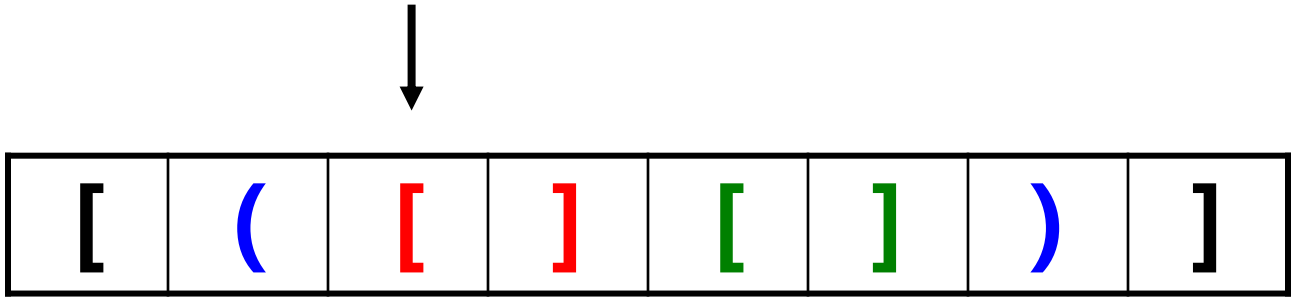


当前是(，和刚才的[不匹配，说明
相匹配的符号还在右边，继续扫描

栈的应用：括号匹配检测

思考

- 从左向右扫描字符串

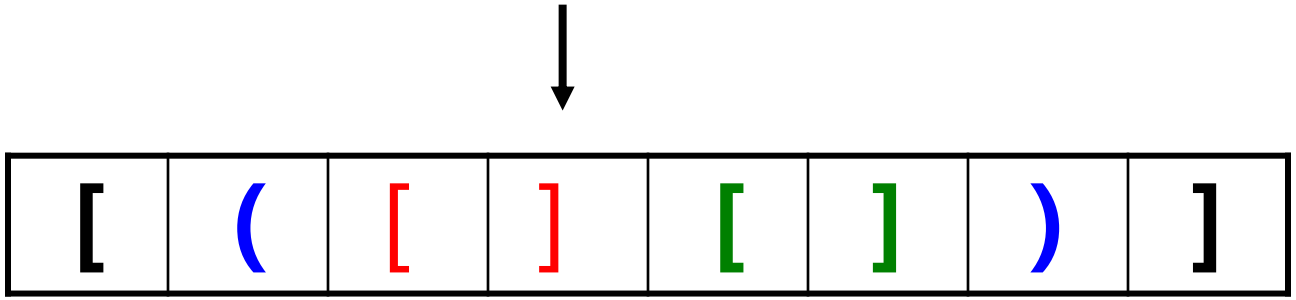


当前是[，和刚才的(不匹配，说明
相匹配的符号还在右边，继续扫描

栈的应用：括号匹配检测

思考

- 从左向右扫描字符串

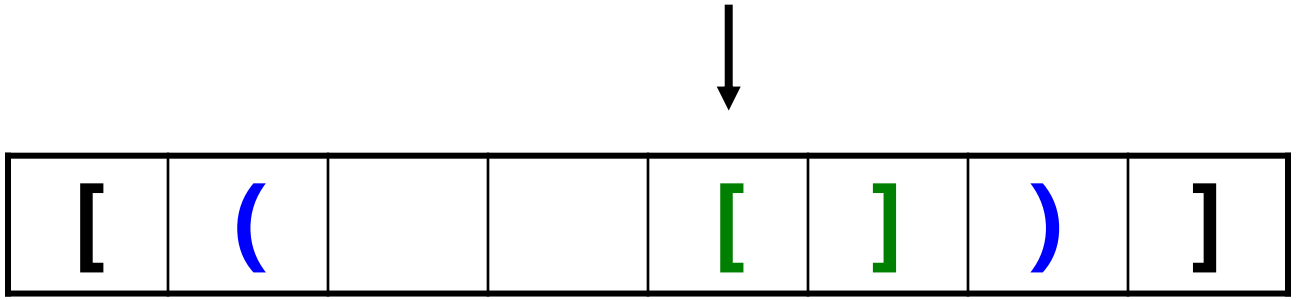


当前是]，和刚才的[正好一对，可以从字符串中“删去”不考虑了

栈的应用：括号匹配检测

思考

- 从左向右扫描字符串

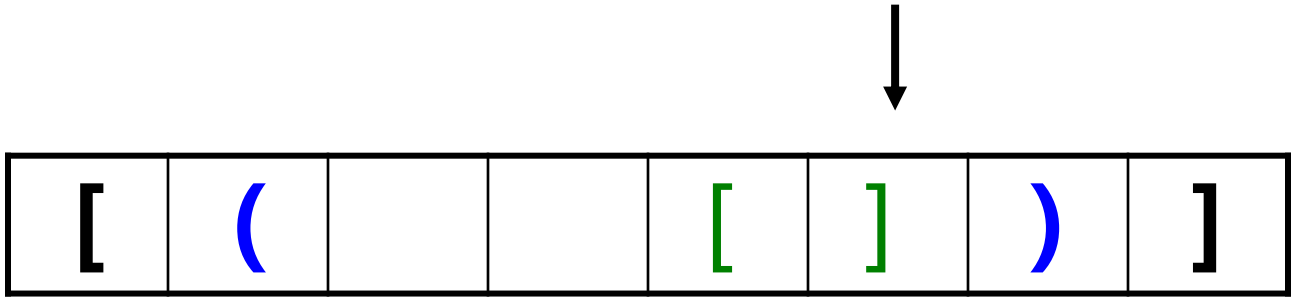


当前是[，目前最近的一个是(，不匹配，继续扫描

栈的应用：括号匹配检测

思考

- 从左向右扫描字符串

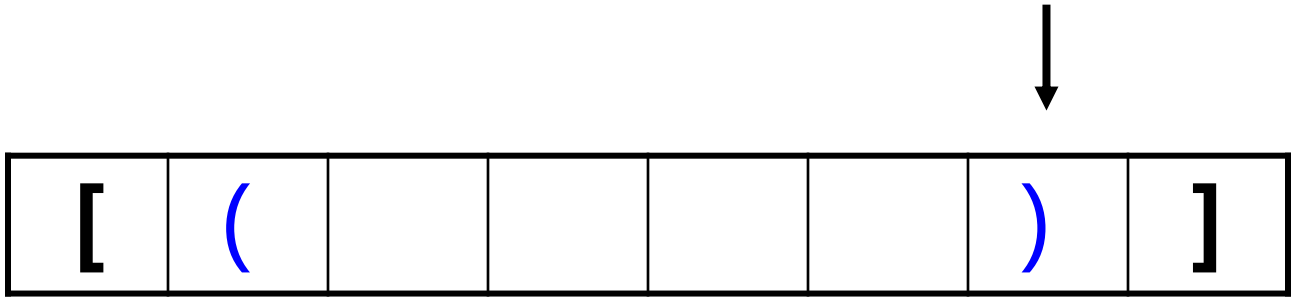


当前是]，和刚才的[正好一对，可以从字符串中“删去”不考虑了

栈的应用：括号匹配检测

思考

- 从左向右扫描字符串

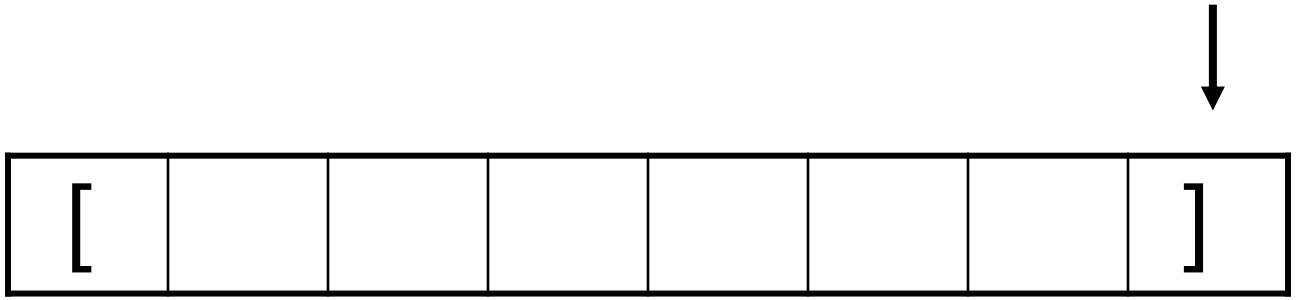


当前是)，目前最近的一个是(，正好一对，可以从字符串中“删去”不考虑了

栈的应用：括号匹配检测

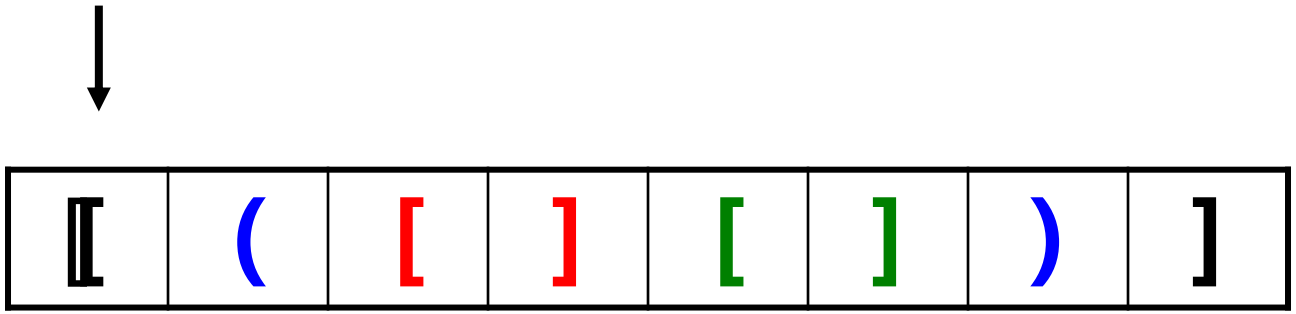
思考

- 从左向右扫描字符串



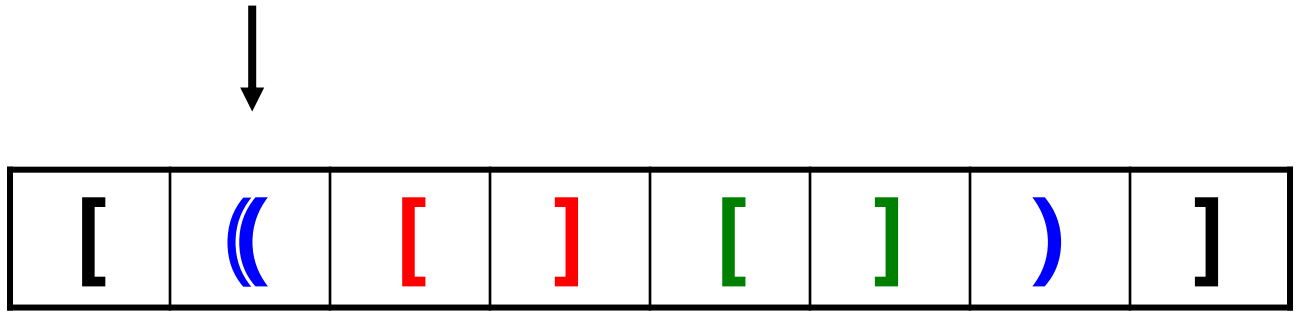
当前是]，目前最近的一个是[，正好一对，可以从字符串中“删去”不考虑了，此时左右的括号都匹配成功

栈的应用：括号匹配检测



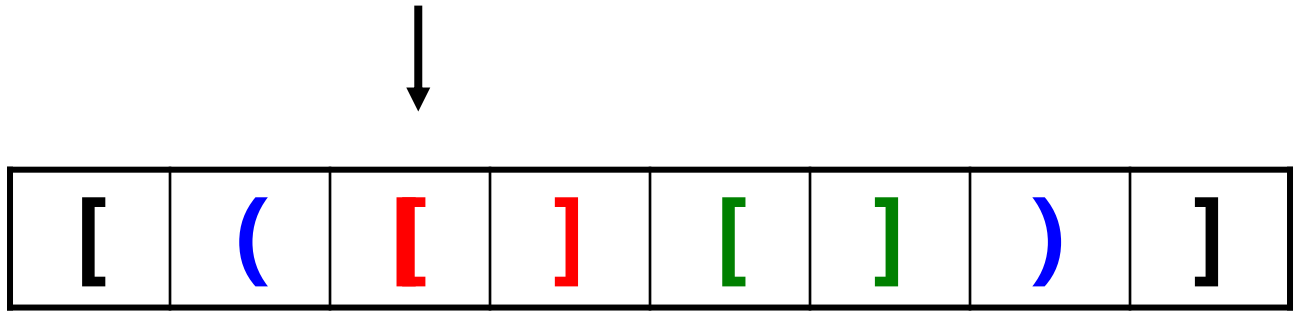
栈为空，
当前字符直接入栈

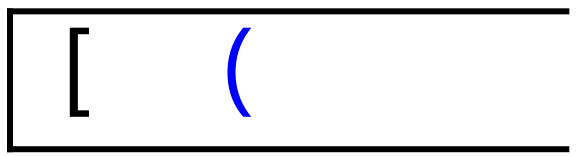
栈的应用：括号匹配检测



栈顶字符和当前字符不匹配，
当前字符入栈

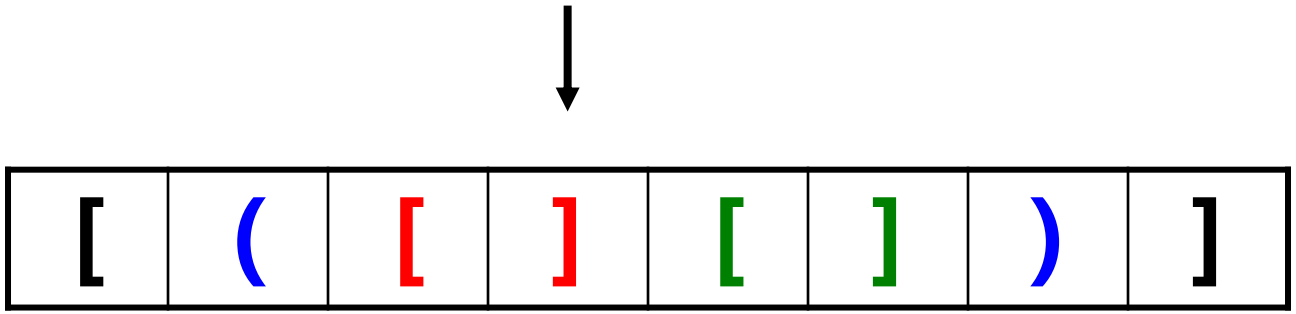
栈的应用：括号匹配检测

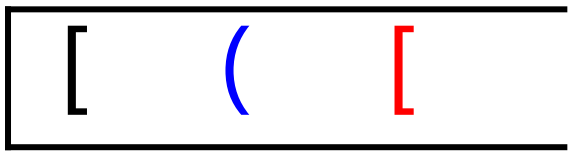




栈顶字符和当前字符不匹配，
当前字符入栈

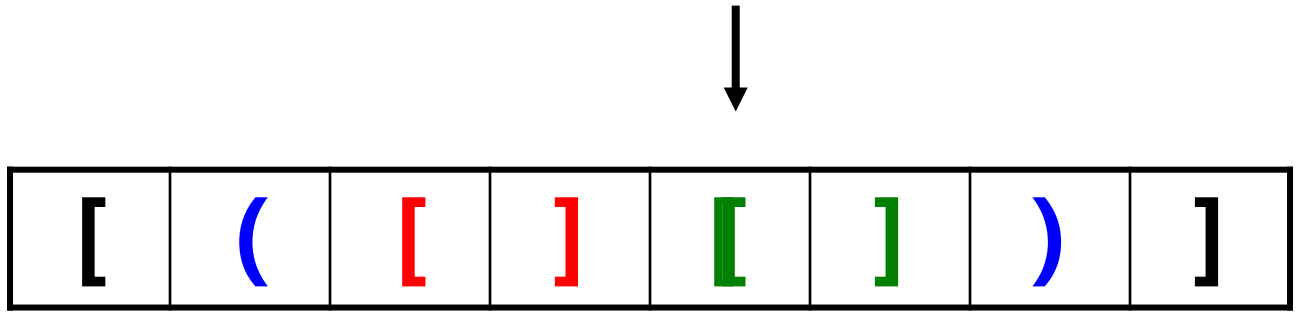
栈的应用：括号匹配检测

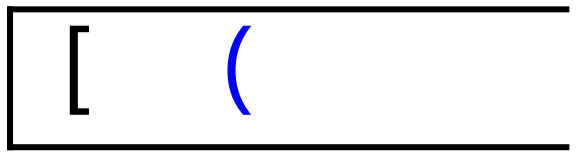




栈顶字符和当前字符匹配，
弹出栈顶字符

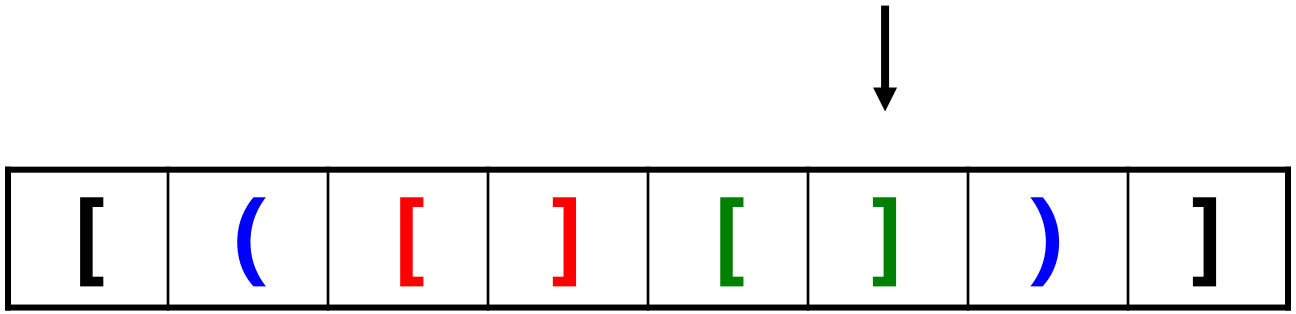
栈的应用：括号匹配检测

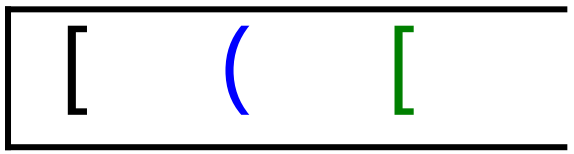




栈顶字符和当前字符不匹配，
当前字符入栈

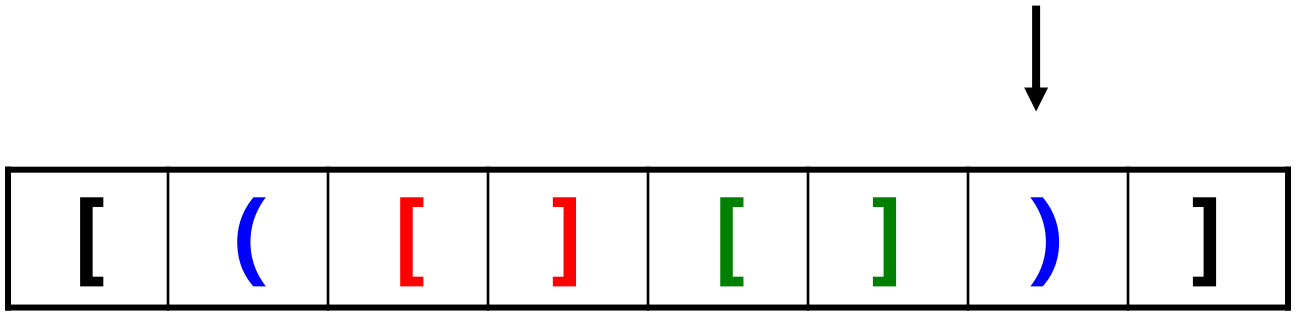
栈的应用：括号匹配检测

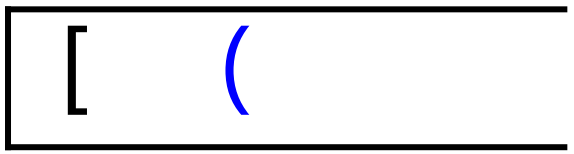




栈顶字符和当前字符匹配，
弹出栈顶字符

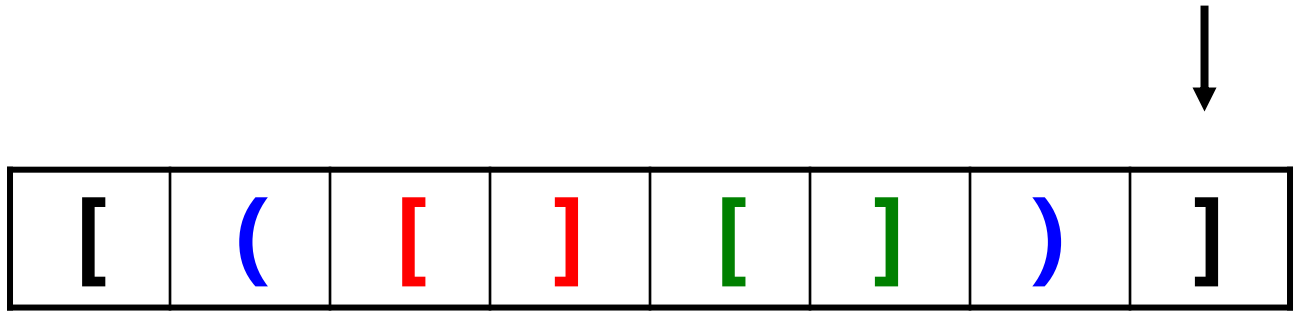
栈的应用：括号匹配检测





栈顶字符和当前字符匹配，
弹出栈顶字符

栈的应用：括号匹配检测



栈顶字符和当前字符匹配，
弹出栈顶字符

栈的应用：作业题

- Leetcode 394 Decode String

栈的应用：模拟系统栈

```
int F(int n) {
        if (n <= 1)
                return 1;
        return n * F(n – 1);
}
```

```
do {
        if (!back) {
                if (n <= 1) {
                        back = true, ret = 1;
                        continue;
                }
                n进栈;
                --n;
        } else { ret *= 出栈； }
} while (栈不为空);
```

栈和队列的应用：作业题

- 设计一个队列/栈
- 支持：出，入，求最大元素
- 要求所有操作O(1)
- 一个例子：
- 3 in, 4 in, 2 in, 5 in, out, out, 6 in, out, out, out
- Leetcode 155. Min Stack

哈希表：定义

- 存放数据的集合
- 操作：根据（Key, Value）进行

  插入，查找，删除（可以没有）
- 空间复杂度：O(m)
- 单次操作时间复杂度：O(1)
- 本质：Key的索引

哈希表：例题

- 给出n个[0, m)范围内的整数，去重
- 快速排序
  - 期望时间复杂度　O(nlogn)
  - 附加空间复杂度　O(1)
- 计数（基数）排序
  - 时间复杂度　　O(n + m)　超越比较排序下限
  - 附加空间复杂度　O(m)

哈希表：思考

- 若n << m，计数排序的大量空间被浪费
- 只需判断是否出现过，优化？
- 将Key区间[0, m) 映射到 [0, p)
- H(key) = key mod p
- 若m > p, 多对一的映射方式

哈希表：实现

- 处理冲突（Key, Value）
  - 开放地址法（数组）
  - 拉链法　　（数组+链表）
- 负载率 = 已有元素大小 / 存储散列大小
- 最坏情况？
- 哈希函数设计

哈希表应用：字符串匹配

- 设字符串A= ‘12314123’
- 求‘123’在A中出现的次数
- 不会写KMP又想要O(n)肿么办？
- Key(‘123’) = ‘1’ * 10^2 + ‘2’ * 10 + ‘3’ * 1

  = 123
- A’ = [123, 231, 314, …, 123]

哈希表应用：字符串匹配

- Key相等时Value有可能不同
- 每次比较Value也是不小的开销，特别是Value可能很大
- 不考虑Value将产生错误率（错误率换时间）
- 多重哈希（降低错误率）

哈希表：麻烦的删除

- 能否直接删除哈希表中的元素？
- 考虑两种不同的实现方式
- 硬删除 vs 软删除

哈希表：思考题

- 设计一个动态平衡的哈希表
- 动态平衡
  - 负载率高 → 增大哈希表空间
  - 负载率低 → 减小哈希表空间
- 正确性测试+压力测试

哈希表：作业题

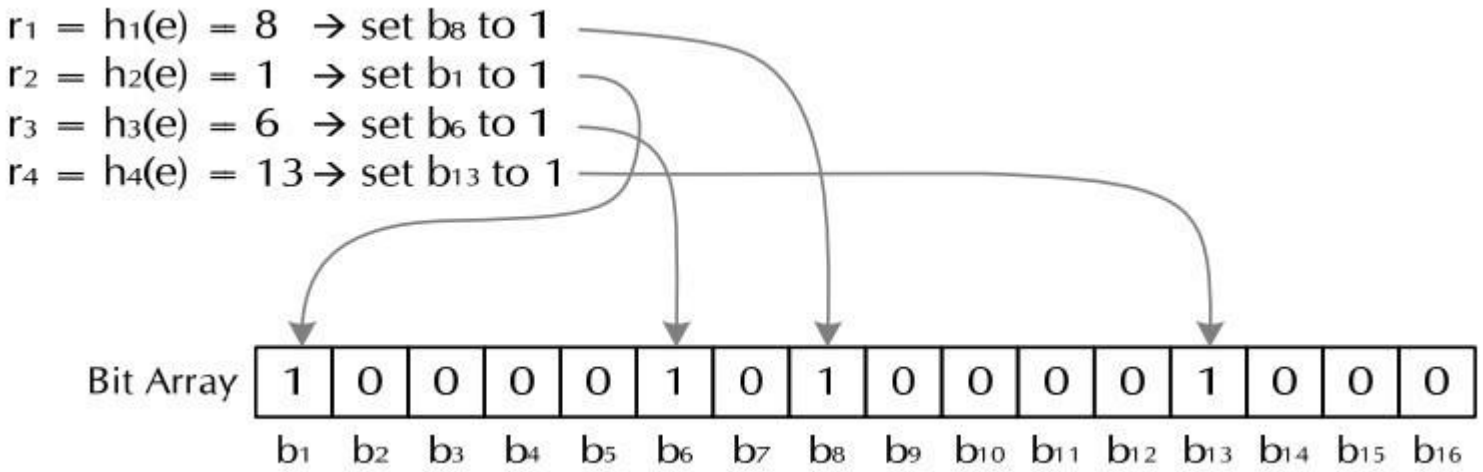
- Leetcode 128. Longest Consecutive Sequence

布隆过滤器：定义

- 判断一个字符串是否出现过的数据结构
- 假设有1亿个字符串，需要多少空间建立Hash索引？
- 1亿 * 8 / 负载率 =
- 哈希表 → 空间换时间
- 布隆过滤器 → 错误率换空间

布隆过滤器：实现

- 由01的数字序列构成
- 插入：多个不同hash函数计算Key，置1
- 查找：有一个为0不可能存在，全为1可能存在
- 空间？



布隆过滤器：优缺点

- 优点
  - 时间和空间
  - 多个hash函数可并行
  - 交差并（位运算）
- 缺点
  - 错误率随着负载率上升而上升
  - 无法删除

布隆过滤器：思考题

- 错误率推导
- 如何进一步优化空间？（位操作）