字符串初步

- C++ string类
- STL map
- 字符串的基本操作
- KMP
- Trie树
- *AC自动机
- *字符串Hash（ELFHash...）

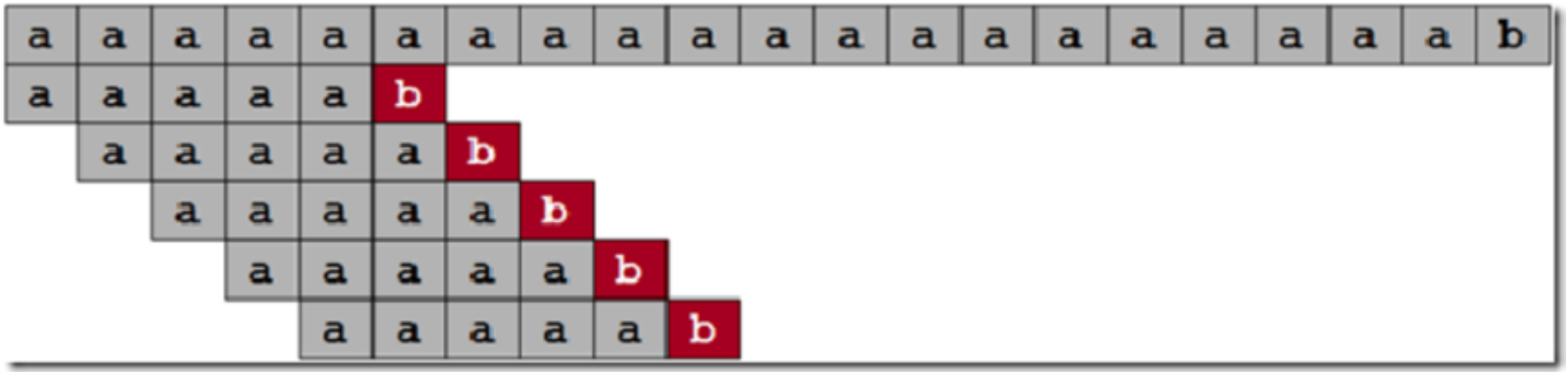
- 子串M长度为m
- 主串S长度为n
- 朴素的模式匹配算法 O(nm)
- 模式匹配算法 KMP
- 多模式匹配算法 AC自动机

- 随机数据下朴素算法快过线性复杂度算法
- 线性时间复杂度的模式匹配算法KMP

- O(n + m)

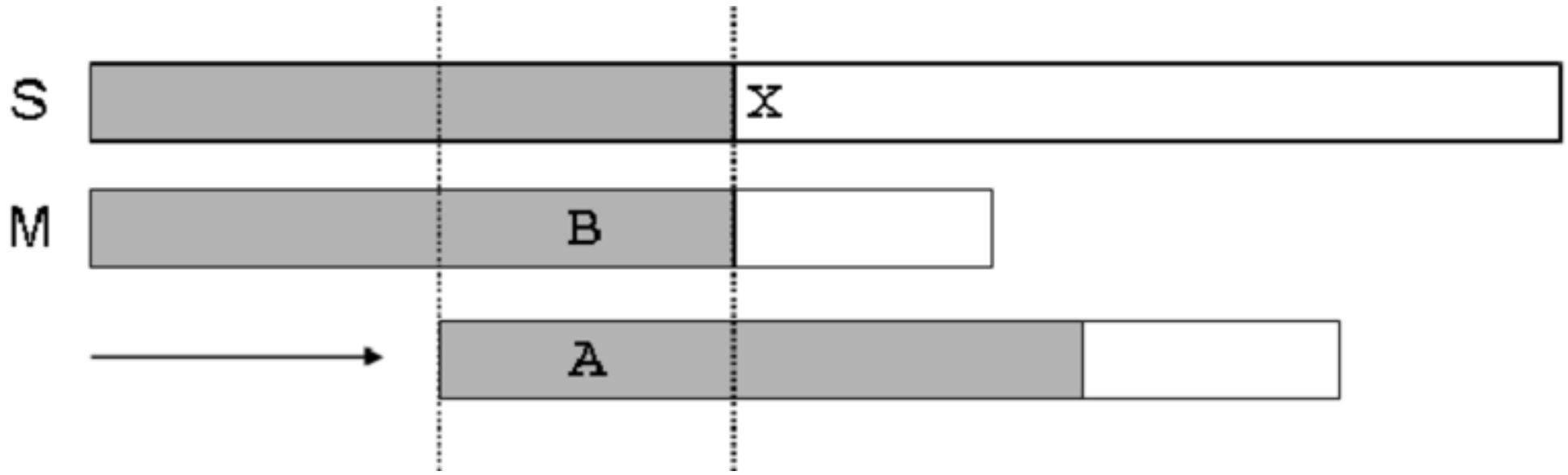
- 如何优化朴素算法?

- 如果失配，朴素算法是从下一个位置重新匹配
- 即为子串向右“滑动”一位



- KMP的优秀之处在于让每次让子串“滑动”尽量多

- 不停将子串向右“滑动”直到再次匹配



- A是灰色字符串的前缀与后缀

- A与B相等

- 为使得算法正确要使A的长度尽量长

- 将S串与M串从第一个字符开始匹配
- 匹配成功则灰色部分不断增加
- 如果失配则将M串向右滑动，使得滑动前灰色部分的后缀与滑动后灰色部分的前缀相等，再进行匹配；如果仍然失配则继续滑动直至匹配成功；若最终仍然失配，则从M串起始位置重新开始。

- 核心在于如何求出每次的“滑动”距离
- next数组
- 即为失配时，S串要与M串的哪个位置开始继续匹配
- 也可以理解为最长的前缀与后缀相等

- 若在M[i]位置失配，则从M[next[i]]位置继续匹配，仍然失配则从M[next[next[i]]]匹配...

- 如何求next数组?

- next数组可以看作自己与自己匹配时求出

- 方法类似

- 时间复杂度分析
- 摊还分析法
- O(n + m)

- Trie树
- 字符串的一种存储结构

- HDU 1251
- 若干个给定的单词
- 统计由某个字符串为前缀的单词的数量

- 多模式匹配算法
- t个子串，长度为m1,m2...mt
- 主串长度为n
- KMP算法复杂度O(sigma(mi) + n * t)

- trie树＋KMP ＝ AC自动机
- 多模式匹配算法
- 线性时间复杂度 O(sigma(mi) + n)