KMP算法

kmp算法又叫"Knuth-Morris-Pratt"算法，是用来在一个字符串中寻找另一个字符串的位置的匹配算法，我们把前者叫做主串（Text String），后者叫做模式串（Pattern String）。

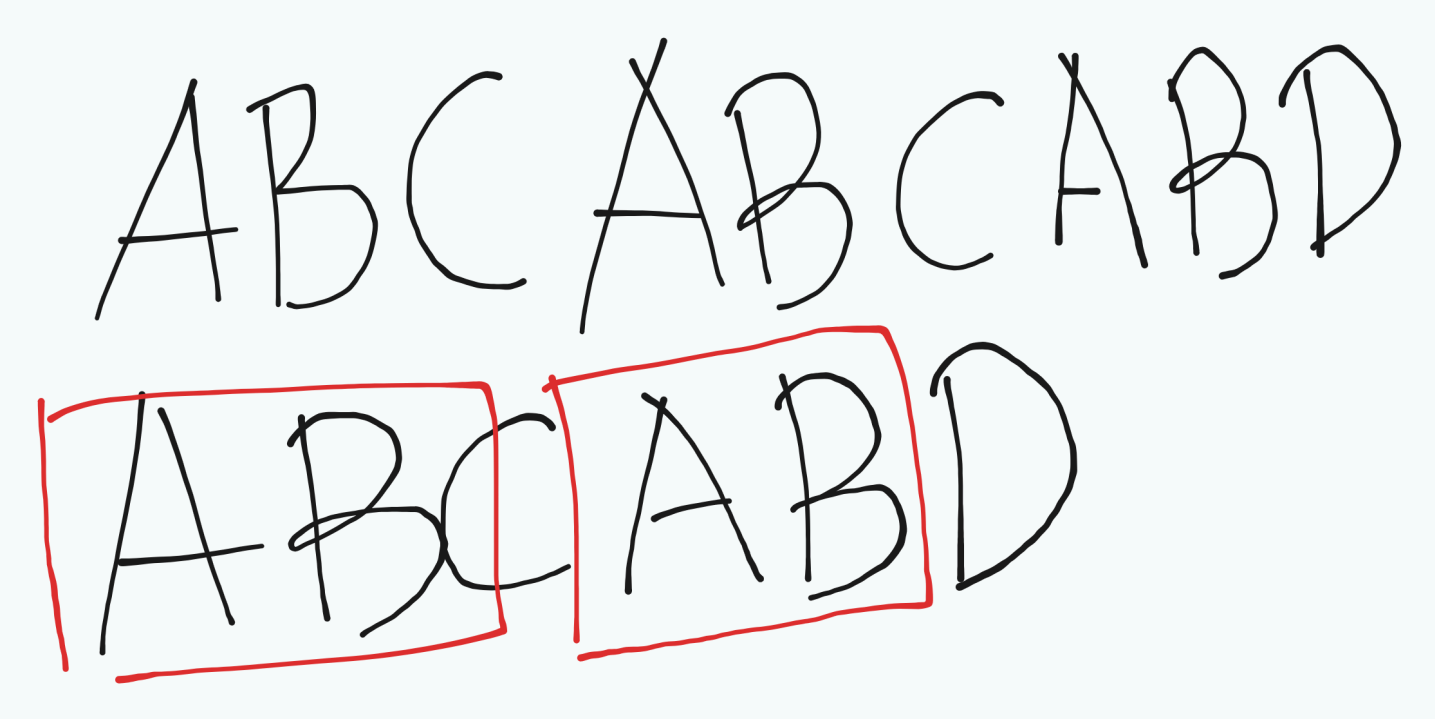
关键是记住一句话：**"利用匹配失败的信息，避免回溯主串，而是滑动模式串。"**

这是什么意思呢？我们来看看一个例子：

ts = "ABCABCABD" pattern = "ABCABD"

我们按简单**双指针**匹配算法来看看，算法如下给出

```
vector<int> matchAlgo(string& ts, string& pattern){
    int pt = 0, pa = 0;
    //双指针，分别在ts和pattern中移动
    vector<int> ans;
    for(int i=0;i<ts.length();i++){
        pt = i;
        pa = 0;
        for(int t = 0;t<pattern.length() && i+t<ts.length();){
            if(ts[pt+t]==pattern[pa+t]){
                t++;
            }
            else break;
        }
        if(t==pattern.length()){
            ans.push_back(i);
        }
        //如果匹配失败了，就会直接将pa归零，pt移动到下一个i处。
    }
}
```



如图所示，我们在i=0的时候，进行匹配，匹配到t=5时，ABCABD的'D'发生了错配，此时按照上面的算法，我们需要之间回溯pt到i=1，pa=0；

但是看我两个圈，我们把两个圈叫做前缀和后缀，可以注意到，我们后缀长度为2,且其必定与主串中相同结尾，长度为2的串相等，而且我们知道我们的前缀是和后缀相等的，那么我们的前缀也就和主串中那个长度为2的串相等了，那么是不是自然地就会有一种想法是，我们将前缀移动到这个串的位置，从而利用了之前错配得到的信息（后缀与主串最后的串相等），这里这个后缀的长度是不一定的，只不过我们取的是前缀后缀相等长度最长的以保证效率。

从而我们得到了我们的目标行为，在pattern[T]发生错配时，根据pattern[0...T-1]的最长相同前后缀来滑动我们的模式串。

这里就可以用一个数组记录pattern中每个元素应该对应的滑动位置，正常我们是叫做next数组，但是这个名字只有体现作用，没有体现由来，我们还可以取个名字，叫做在 最长相同前后缀长度 。是便于理解的，不过后面主要还是叫next数组吧。

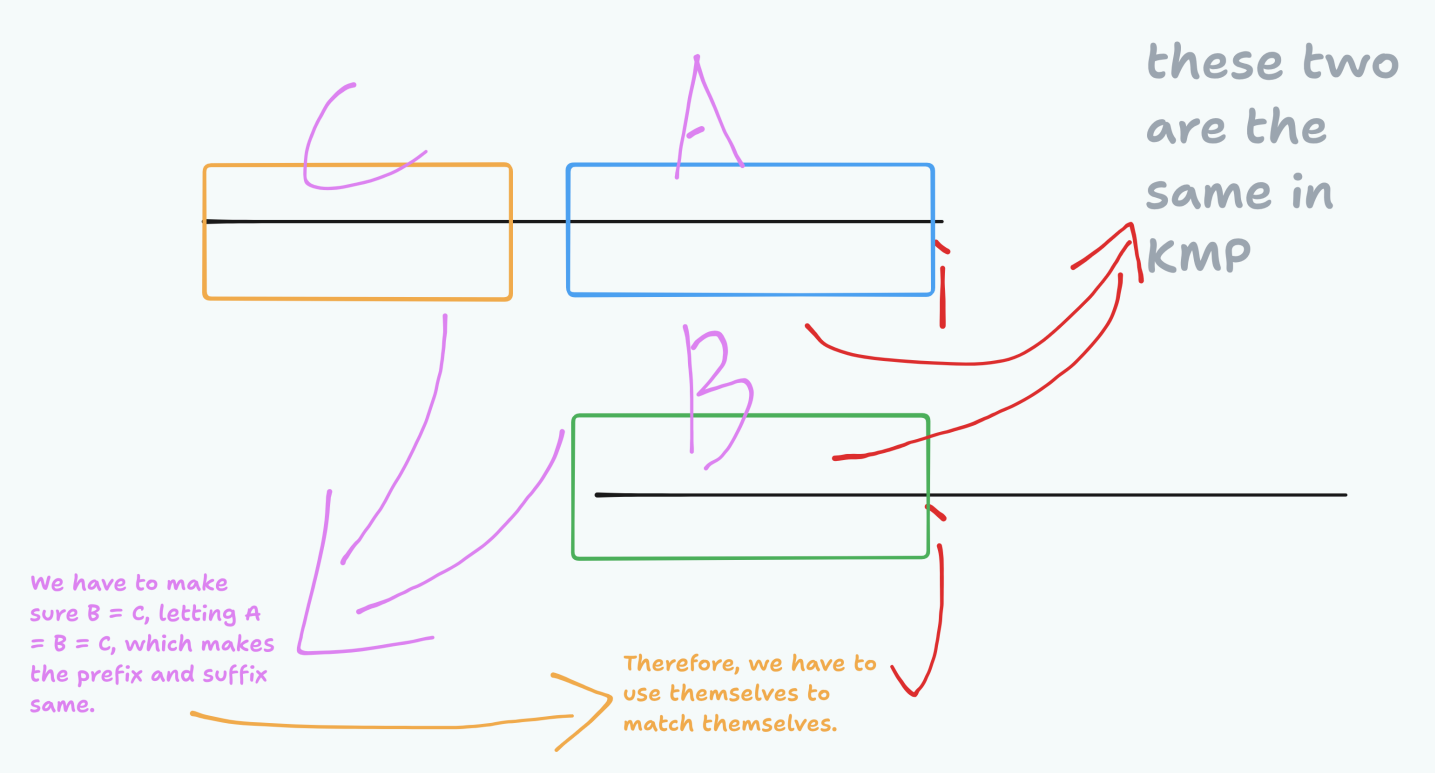
接下来看看计算next数组的函数：

```
vector<int> computeNextArray(stirng&& pattern){
    int m = P.length();
    int next[m];
    int i = 0, j = 0;
    //i为后缀末尾，j为前缀末尾，双指针
    next[0] = 0;//第一个必定没有相同前后缀
    i++;//从第二个
    while(i<m){
        if(pattern[i]==pattern[j]){
            next[i++] == ++j;
            //以i为止的后缀
            //可以匹配以j结尾的前缀，其中j同时也是长度
        }
        else{
            if(j!=0) j = next[j-1];//j=0时回溯没意义。
            else next[i++] = 0;
        }
    }
    return next;
}
```

但是为什么呢?

为什么是用自己匹配自己呢？注意这里的j是前缀结尾指针，i是后缀结尾指针。

我们看图



KMP匹配的过程中，我们用两根直线来方便理解，两根直线错开放置，AB段为已经匹配的段落，我们

能确定A = B，那么我们是要求最长的相同前缀和后缀的，那么也就是使C = A，那么就只要C = B = A就可以了，为了到达这个效果于是我们就用自己匹配自己。

此时如果Pi==Pj的话，就能把A，B变长一格，如果不等的话，就右滑下面那根，缩短范围，更好匹配。