KMP及扩展

KMP算法主要解决子串在母串中的匹配定位(pattern)问题。

可以在O(n+m)的复杂度下找到匹配。

为什么要用kmp

对于在长度为n的母串搜索长度为m的子串问题，双指针暴力枚举的时间复杂度可以达到$O(nm)$，事实上，指针一格格走会徒增很多没必要的计算。失配时，如果我们能将子串中当前匹配的字符移动到上一次出现的位置，那么就可以减少很多计算。

如何记录需要移动的位置

我们要移动的下一个位置满足这个性质：最前面的k个字符和j之前的最后k个字符是一样的。假设我们在k位置失配，那么移动的位数就和前缀 $=$ 后缀的最大位数有关。

板子

```
//luoguP3375 【模板】KMP字符串匹配
#include<bits/stdc++.h>
using namespace std;
string s1,s2;
int j=0,len1,len2,next[1000007];
signed main(){
    cin>>s1;cin>>s2;
    len1=s1.length();
    len2=s2.length();
    next[0]=0;
    for(int i=1;i<len2;i++){ //获取失配数组
        j=next[i];
        while(j&&s2[j]!=s2[i]) j=next[j];
        next[i+1]=(s2[j]==s2[i])?j+1:0;
```

```
    }
    j=0;
    for(int i=0;i<len1;i++){
        while(j&&s1[i]!=s2[j]) j=next[j];
        if(s1[i]==s2[j]){
            j++;
            if(j==len2) cout<<i-j+2<<endl;
        }
    }
    for(int i=1;i<=len2;i++) cout<<next[i]<<" ";
    return 0;
}
```

前缀函数

```
vector<int> prefix_function(string s) {
  int n = (int)s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j]) j = pi[j - 1];
    if (s[i] == s[j]) j++;
    pi[i] = j;
  }
  return pi;
}
```

Z函数

定义

对于长度为n的字符串s，定义函数z[i]表示s和s[i,n-1](即以s[i]开头的后缀)的最长公共前缀（LCP,longest common prefix）的长度。特别的，z[0]=0。

Z Box

Z Box是字符串str中的一个区间[l,r]满足str[l,r]是str的前缀（不要求最长）。在位置i时，要求[l,r]必须包含i，且r尽可能大，那么就能得到Z函数数组。计算中我们维护右端点最靠右的Z-Box，规定初始时$l=r=0$，保证$l\le i$。

计算过程

$$\text{如果}i\le r\text{，有}s[i,r]=s[i-l,r-l]\text{，因此}z[i]\ge min(z[i-l],r-i+1)\text{。}$$

$$\text{①若}z[i-l]<r-i+1\text{,则}z[i]=z[i-l]$$

$$\text{②若}z[i-l]\ge r-i+1\text{，令}z[i]=r-i+1\text{,然后暴力枚举直到}z[i]\text{不能扩展为止。}$$

$$\text{如果}i>r\text{,根据朴素算法，从}s[i]\text{开始暴力求出}z[i]$$

$$\text{在求出}z[i]\text{后，如果}i+z[i]-1>r\text{，就需要更新}[l,r]\text{，即令}l=i,r=i+z[i]-1\text{。}$$

```
void getz(int n,char *str){ //Z函数模板
    int l=0,r=0;
    z[0]=0;z[1]=n;
    for(int i=2;i<=n;i++){
```

```
        if(i>r){
            while(str[i+z[i]]==str[1+z[i]]) z[i]++;
            l=i,r=i+z[i]-1;
        }else if(z[i-l+1]<r-i+1) z[i]=z[i-l+1];
        else{
            z[i]=r-i;
            while(str[i+z[i]]==str[1+z[i]]) z[i]++;
            l=i,r=i+z[i]-1;
        }
    }
}
```

扩展KMP

问题描述

给定两个字符串str1,str2，求出str1的每一个后缀与str2的最长公共前缀。

Ext Box

当str1==str2时，就相当于求z函数。

当str1和str2不相同时，用ext[i]表示str1第i位开始的后缀与str2的最长公共前缀，只需要修改z函数的推导过程即可。

luoguP5410 【模板】扩展 KMP（Z 函数）

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int N=2e7+7;
int n,m,ans1,ans2;
int z[N],ext[N];
char str1[N],str2[N];

void getz(){ //str2的每一个后缀与str2的最长公共前缀
    z[0]=m;
    int l=0,r=0;
    for(int i=1;i<m;i++){ //和exkmp的求法是一样的，这个写法好像比较简单
        if(i<=r&&z[i-l]<r-i+1) z[i]=z[i-l];
        else{
            z[i]=max(0ll,r-i+1);
            while(i+z[i]<m&&str2[i+z[i]]==str2[z[i]]) z[i]++;
        }
        if(i+z[i]-1>r) l=i,r=i+z[i]-1;
    }
}

void exkmp(){ //str1的每一个后缀与str2的最长公共前缀
    int now=0;
    while(str1[now]==str2[now]&&now<min(n,m)) now++; //暴力
    ext[0]=now;
    int p=0;
    for(int i=1;i<n;i++){
```

```
        if(i+z[i-p]<ext[p]+p) ext[i]=z[i-p];
        else{
            int now=ext[p]+p-i;
            now=max(now,0ll);
            while(str2[now]==str1[i+now]&&now<m&&now+i<n) now++;
            ext[i]=now;
            p=i;
        }
    }
}

signed main(){
    scanf("%s%s",str1,str2);
    n=strlen(str1),m=strlen(str2);
    getz();
    exkmp();
    /*
    for(int i=0;i<m;i++) cout<<z[i]<<" ";
    cout<<endl;
    for(int i=0;i<n;i++) cout<<ext[i]<<" ";
    cout<<endl;
    */
    for(int i=0;i<m;i++) ans1^=(i+1)*(z[i]+1);
    for(int i=0;i<n;i++) ans2^=(i+1)*(ext[i]+1);
    cout<<ans1<<"\n"<<ans2<<"\n";
    return 0;
}
```

参考

OI-wiki

https://www.bilibili.com/video/BV1LK4y1X74N

https://www.luogu.com.cn/blog/pks-LOVING/zi-fu-chuan-xue-xi-bi-ji-qian-xi-kmp-xuan-xue-di-dan-mu-shi-chuan-pi-post