

- 马拉车
- kmp求子串
- 求前后重叠的字符串长度
- 求本质不同子序列的数量 (dp)
- 字典树
- Z函数
- 01树

马拉车

```
vector<int> manacher(string res)
{
    string s = "^#";
    for (auto c : res)
    {
        s.push_back(c);
        s.push_back('#');
    }
    int n = s.size();
    s.push_back('$');

    vector<int> zbox(n);
    for (int i = 1, l = 1, r = 1; i < n; i++)
    {
        if (i <= r)
            zbox[i] = min(zbox[r - i + 1], r - i + 1);
        while (s[i - zbox[i]] == s[i + zbox[i]])
            zbox[i]++;
        if (i + zbox[i] - 1 > r)
            l = i - zbox[i] + 1, r = i + zbox[i] - 1;
    }
    return zbox;
}
```

kmp求子串

```
vector<int> prefix_function(string s) { // 求字符串前缀函数
    int n = (int)s.length();
    vector<int> pi(n+1);
    s = ' ' + s;
    for (int i = 2; i <= n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j+1]) j = pi[j];
    }
}
```

```

    if (s[i] == s[j+1]) j++;
    pi[i] = j;
}
return pi;
}

vector<int> find_occurrences(string text, string pattern) {
    string cur = pattern + '#' + text;
    int sz1 = text.size(), sz2 = pattern.size();
    vector<int> v;
    vector<int> lps = prefix_function(cur);
    for (int i = sz2 + 1; i <= sz1 + sz2; i++) {
        if (lps[i] == sz2) v.push_back(i - 2 * sz2);
    }
    return v;
}

```

求前后重叠的字符串长度

```

int getLonestCover(string s, string t)
{
    vector<int> a(t.length(), 0);
    a[0] = t.length();
    int len = 0;
    while (1 + len < t.length() && t[len] == t[1 + len])
        len++;
    a[1] = len;
    int k = 1, mx = k + a[k] - 1;
    for (int i = 2; i < t.length(); i++)
    {
        int w = i - k;
        if (i + a[w] - 1 < mx)
            a[i] = a[w];
        else
        {
            int len = max(0, mx - i + 1);
            while (i + len < t.length() && t[0 + len] == t[i + len])
                len++;
            a[i] = len;
            k = i, mx = i + a[i] - 1;
        }
    }
    vector<int> b(s.length(), 0);
    len = 0;
    while (len < s.length() && len < t.length() && s[len] == t[len])
        len++;
    b[0] = len;
    k = 0, mx = len - 1;
    for (int i = 1; i < s.length(); i++)
    {
        int w = i - k;
        if (i + a[w] - 1 < mx)

```

```

        b[i]=a[w];
    else
    {
        int len=max(0,mx-i+1);
        while(i+len<s.length()&&len<t.length()&&s[i+len]==t[len])
            len++;
        b[i]=len;
        k=i,mx=i+b[i]-1;
    }
}
int ans=0;
for(int i=0;i<s.length();i++)
    if(b[i]==s.length()-i)
    {
        ans=b[i];
        break;
    }
return ans;

```

求本质不同子序列的数量（dp）

```

int distinctSubsqe(string s){
    int n=s.size();
    int M=1e9+7;
    vector<int>dp(n+1);
    dp[0]=0;
    vector<int>last(26,-1);
    for(int i=0;i<n;i++){
        int ch=s[i]-'a';
        dp[i+1]=(2*dp[i]+1)%M;
        if(last[ch]>=0){
            dp[i+1]-=(dp[last[ch]]+1);
        }
        dp[i+1]%M;
        last[ch]=i;
    }
    return dp[n]<0?dp[n]+M:dp[n];
}

```

字典树

```

const int maxn=510005;
int sz[maxn];
ll ans;
int cnt,last[maxn],n;
vector<int>g[maxn];

```

```

struct node{
    int nt[26];
    int tag=0;
};
struct trie{ //trie及所需操作
    node a[maxn];
    int cnt;
    void ins(string s){ //建树
        int x=0;
        for(int i=0;i<s.size();i++){
            if(!a[x].nt[s[i]-'a']) a[x].nt[s[i]-'a']=++cnt;
            x=a[x].nt[s[i]-'a'];
        }
        a[x].tag=1;
    }
    void doit(int x){ //重构树
        if(a[x].tag&&x){
            g[last[x]].push_back(x);
            last[x]=x;
        }
        for(int i=0;i<26;i++)
            if(a[x].nt[i]){
                last[a[x].nt[i]]=last[x];
                doit(a[x].nt[i]);
            }
    }
    bool find(string s){//查找
        int x=0;
        for(int i=0;i<s.size();i++){
            if(!a[x].nt[s[i]-'a'])return 0;
            x=a[x].nt[s[i]-'a'];
        }
        return a[x].tag;
    }
}tr;

```

Z函数

```

vector<int>zbox(string s){
    int n=(int)s.length();
    vector<int>z(n+1);
    for (int i = 1, l = 0, r = 0; i <= n; ++i) {
        if (i <= r && z[i - l] < r - i + 1) {
            z[i] = z[i - l];
        }
        else {
            z[i] = max(0, r - i + 1);
            while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        }
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
    z.insert(z.begin(),0);
}

```

```
    return z;
}
```

01树

```
int son[50*N][2];
int ed[50*N];
int vis[50*N];
int cnt;
int maxn=0;
void insert(int x,int index){
    int p=0;
    for(int i=32;i>=0;i--){
        int idx=(x>>i)&1;
        if(!son[p][idx])son[p][idx]=++cnt;
        p=son[p][idx];
        vis[p]++;
    }
    ed[p]=index;
    maxn=max(maxn,p);
}
void del(int x){
    int p=0;
    for(int i=32;i>=0;i--){
        int idx=(x>>i)&1;
        p=son[p][idx];
        vis[p]--;
    }
    if(vis[p]==0)ed[p]=0;
}
//查询就不写了，查询模仿插入就行
```