字符串

1. KMP

```
int f[N];
void kmp(string s, string p)
{
    p += '@';
    p += s;
    for (int i = 1; i < p.size(); i++)
    {
        int j = f[i - 1];
        while (j && p[j] != p[i])
            j = f[j - 1];
        if (p[j] == p[i])
            f[i] = j + 1;
    }
}
```

2. Manacher

```
int p[N];
string s0, s = "@#";

int main()
{
    cin >> s0;
    for (auto i : s0)
        s += i, s += "#";#include <cstdio>
using namespace std;
using ll = long long;
// !!! N = n * 2, because you need to insert '#' !!!
const int N = 3e7;
#define min(A, B) ((A > B) ? B : A)
// p[i]: range of the palindrome i-centered.
int p[N];
// s: the string.
char s[N] = "@#";
// l: length of s.
int l = 2;

int main()
{
    char tmp = getchar();
    while (tmp > 'z' || tmp < 'a')
        tmp = getchar();
    while (tmp <= 'z' && tmp >= 'a')
        s[l++] = tmp, s[l++] = '#', tmp = getchar();
```

```
    /*<--- input & preparation --->*/
    int m = 0, r = 0;
    ll ans = 0;
    for (int i = 1; i < l; i++)
    {
        // evaluate p[i]
        if (i <= r)
            p[i] = min(p[m * 2 - i], r - i + 1);
        else
            p[i] = 1;
        // brute force!
        while (s[i - p[i]] == s[i + p[i]])
            ++p[i];
        // maintain m, r
        if (i + p[i] > r)
        {
            r = i + p[i] - 1;
            m = i;
        }
        // find the longest p[i]
        if (p[i] > ans)
            ans = p[i];
    }
    printf("%lld", ans - 1);

    return 0;
}
    int m = 0, r = 0;
    ll ans = 0;
    int l = s.size();
    for (int i = 1; i < l; i++)
    {
        if (i <= r)
            p[i] = min(p[m * 2 - i], r - i + 1);
        else
            p[i] = 1;
        while (s[i - p[i]] == s[i + p[i]])
            ++p[i];
        if (i + p[i] > r)
        {
            r = i + p[i] - 1;
            m = i;
        }
        if (p[i] > ans)
            ans = p[i];
    }
}
```

3. hash

随机素数表

42737, 46411, 50101, 52627, 54577, 191677, 194869, 210407, 221831, 241337, 578603, 625409, 713569, 788813, 862481, 2174729, 2326673, 2688877, 2779417, 3133583, 4489747, 6697841, 6791471, 6878533, 7883129, 9124553, 10415371, 11134633, 12214801, 15589333, 17148757, 17997457, 20278487, 27256133, 28678757, 38206199, 41337119, 47422547, 48543479, 52834961, 76993291, 85852231, 95217823, 108755593, 132972461, 171863609, 173629837, 176939899, 207808351, 227218703, 306112619, 311809637, 322711981, 330806107, 345593317, 345887293, 362838523, 373523729, 394207349, 409580177, 437359931, 483577261, 490845269, 512059357, 534387017, 698987533, 764016151, 906097321, 914067307, 954169327

1572869, 3145739, 6291469, 12582917, 25165843, 50331653 （适合哈希的素数）

```
#include <string>
using ull = unsigned long long;
using std::string;
const int mod = 998244353;

int n;
ull Hash[2000200]; // 自然溢出法用unsigned类型
ull RHash[2000200];
ull base[2000200];
void init()
{
    base[0] = 1;
    for (int i = 1; i <= 2000010; i++)
    {
        base[i] = base[i - 1] * 131 % mod;
    }
}
void get_hash(string s)
{
    for (int i = 1; i <= (int)s.size(); i++)
    {
        Hash[i] = Hash[i - 1] * base[1] % mod + s[i - 1];
        Hash[i] %= mod;
    }
}
void get_Rhash(string s)
{
    for (int i = (int)s.size(); i >= 1; i--)
    {
        RHash[s.size() - i + 1] = RHash[s.size() - i] * base[1] % mod + s[i - 1];
        RHash[i] %= mod;
    }
}
ull getR(int l, int r)
{
    if (l > r)
        return 0;
    return (RHash[r] - (RHash[l - 1] * base[r - l + 1]) % mod + mod) % mod;
```

```
}
ull get(int l, int r)
{
    if (l > r)
        return 0;
    return (Hash[r] - (Hash[l - 1] * base[r - l + 1]) % mod + mod) % mod;
}
```

4. 字典树

```
#include <string>
using std::string;
/* Last modified: 23/07/03 */
// Trie for string and prefix
class Trie
{

    static const int trie_tot_size = 1e5;
    // trie_node_size: modify if get() is modified.
    static const int trie_node_size = 64;
    int tot = 0;
    // end: reserved for count
    const int end = 63;
    int (*nxt)[trie_node_size];

  public:
    Trie()
    {
        nxt = new (int[trie_tot_size][trie_node_size]);
    }
    int get(char x)
    {
        // modify if x is in certain range, assuming 0-9 or a-z.
        if (x >= 'A' && x <= 'Z')
            return x - 'A';
        else if (x >= 'a' && x <= 'z')
            return x - 'a' + 26;
        else
            return x - '0' + 52;
    }
    int find(string s)
    {
        int cnt = 0;
        for (auto i : s)
        {
            cnt = nxt[cnt][get(i)];
            if (!cnt)
                return 0;
        }
        return cnt;
```

```
    }
    void insert(string s)
    {
        int cnt = 0;
        for (auto i : s)
        {
            auto j = get(i);
            // count how many strings went by
            nxt[cnt][end]++;
            if (nxt[cnt][j] > 0)
                // character i already exists.
                cnt = nxt[cnt][j];
            else
            {
                // doesn't exist, new node.
                nxt[cnt][j] = ++tot;
                cnt = tot;
            }
        }
        nxt[cnt][end]++;
    }
    int count(string s)
    {
        int cnt = find(s);
        if (!cnt)
            return 0;
        return nxt[cnt][end];
    }
    void clear()
    {
        for (int i = 0; i <= tot; i++)
            for (int j = 0; j <= end; j++)
                nxt[i][j] = 0;
        tot = 0;
    }
};
```

5.AC自动机

```
#include <queue>
#include <string.h>
#include <string>
using std::string, std::queue;

struct AC_automaton
{
    static const int _N = 1e6;

    int (*trie)[27];
    int tot = 0;
    int *fail;
```

```
    int *e;
    AC_automaton()
    {
        trie = new int[_N][27];
        fail = new int[_N];
        e = new int[_N];
        memset(trie, 0, sizeof(trie));
        memset(fail, 0, sizeof(fail));
        memset(e, 0, sizeof(e));
    }

    void insert(string s)
    {
        int now = 0;
        for (auto i : s)
            if (trie[now][i - 'a'])
                now = trie[now][i - 'a'];
            else
                trie[now][i - 'a'] = ++tot, now = tot;
        e[now]++;
    }

    void build()
    {
        queue<int> q;
        for (int i = 0; i < 26; i++)
            if (trie[0][i])
                q.push(trie[0][i]);
        while (q.size())
        {
            int u = q.front();
            q.pop();
            for (int i = 0; i < 26; i++)
                if (trie[u][i])
                    fail[trie[u][i]] = trie[fail[u]][i], q.push(trie[u][i]);
                else
                    trie[u][i] = trie[fail[u]][i];
        }
    }

    int query(string t)
    {
        int u = 0, res = 0;
        for (auto c : t)
        {
            u = trie[u][c - 'a'];
            for (int j = u; j && e[j] != -1; j = fail[j])
                res += e[j], e[j] = -1;
        }
        return res;
    }
};
```

心态崩了

- `(int)v.size()`
- `1LL << k`
- 递归函数用全局或者 static 变量要小心
- 预处理组合数注意上限
- 想清楚到底是要 `multiset` 还是 `set`
- 提交之前看一下数据范围，测一下边界
- 数据结构注意数组大小 （2倍，4倍）
- 字符串注意字符集
- 如果函数中使用了默认参数的话，注意调用时的参数个数。
- 注意要读完
- 构造参数无法使用自己
- 树链剖分/dfs 序，初始化或者询问不要忘记 idx, ridx
- 排序时注意结构体的所有属性是不是考虑了
- 不要把 while 写成 if
- 不要把 int 开成 char
- 清零的时候全部用 0~n+1。
- 模意义下不要用除法
- 哈希不要自然溢出
- 最短路不要 SPFA，乖乖写 Dijkstra
- 上取整以及 GCD 小心负数
- mid 用 `l + (r - l) / 2` 可以避免溢出和负数的问题
- 小心模板自带的意料之外的隐式类型转换
- 求最优解时不要忘记更新当前最优解
- 图论问题一定要注意图不连通的问题
- 处理强制在线的时候 lastans 负数也要记得矫正
- 不要觉得编译器什么都能优化
- 分块一定要特判在同一块中的情况