Strings

Hash

```cpp
#include <bits/stdc++.h>
using namespace std;

long long B = 26; // Size of Alphabet
long long M = 512927377; //A  big prime

long long int_mod(long long a, long long b) // Calculates mod even if a < 0
{
	return ( a % b + b) % b;
}
bool Rolling_Hash(string text, string pattern) // Rabin - Karp Algorithm
{
	if( text.size() < pattern.size() ) return false;

	long long Hp = 0, Ht = 0, m = pattern.size();
	for(int i = 0 ; i < m ; i++) // Hash of pattern X First Hash of text
	{
		Hp = int_mod(Hp * B +  pattern[i], M);
		Ht = int_mod(Ht * B + text[i], M);
	}

	if(Hp == Ht) return true;
	for(int i = m; i < text.size(); i++) // Rolling hash
	{
Ht = int_mod( Ht - int_mod(text[i-m] * int_mod( pow(B,m-1), M ), M ),M);
		Ht = int_mod( Ht * B , M);
		Ht = int_mod( Ht + text[i], M);

		if(Ht == Hp) return true;
	}
	return false;
}
int main()
{
	string a,b;
	cin>>a>>b;
	cout<< Rolling_Hash(a,b) << '\n';
	return 0;

}
```

KMP

```cpp
#include <bits/stdc++.h>
using namespace std;
vector <int> Build_Failure(string pattern)
{
	vector <int> F(pattern.size()+1);
	F[0] = F[1] = 0;
	for(int i = 2 ; i <= pattern.size(); i++)
	{
		int j = F[i - 1];
		while(true)
		{
			if(pattern[j] == pattern[i -1]){

				F[i] = j + 1;
				break;
			}
			if(j == 0){
				F[i] = 0;
				break;
			}
			j = F[j];
		}
	}
	return F;
}
```

```cpp
int KMP(string text, string patttern)
{
	vector <int> F = Build_Failure(patttern);

	int count = 0;
	int state = 0;
	int index = 0;
	while(true)
	{
		if(index == text.size()) break;

		if(text[index] == patttern[state] )
		{
			state++;
			index++;
			if(state == patttern.size())
				count++;
		}

		else if( state > 0) state = F[state];

		else index++;
	}
	return count;
}
```

Suffix Array

```
#include <bits/stdc++.h>
using namespace std;
const int maxN = 20;
struct tuple
{
	int indxs[2];
	int pos;
}L[maxN];
int cmp(struct tuple a, struct tuple b){
	return (a.indxs[0] == b.indxs[0]) ? (a.indxs[1] < b.indxs[1]?1:0) : (a.indxs[0] < b.indxs[0]?1:0);
}
int sortIndex[5000][maxN];
int Suffix_Array(string s)
{
	for(int i = 0 ; i < s.size() ; i++)
		sortIndex[0][i] = s[i] -'a';

	int done_till = 1, step = 1;
	int N = s.size();

	for(;done_till < N; step++, done_till*=2)
	{

		for(int i = 0 ; i < N ; i++)
		{
			L[i].indxs[0] = sortIndex[step - 1][i];
			L[i].indxs[1] = ( i + done_till ) < N? sortIndex[step-1][i + done_till]:-1;
			L[i].pos = i;
		}
```

```
            sort(L, L + N, cmp);

            for(int i = 0 ; i < N; i++)
            {
            sortIndex[ step ][ L[i].pos ] = i > 0 && L[i].indxs[0] == L[i - 1].indxs[0] && L[i].indxs[1] == L[i - 1].indxs[1] ?
                                                                                    sortIndex[ step ][ L[i-1].pos ] : i;
            }
        }
        return step;
}
```

Z algorithm

```
vector <int> build_Z(string text){
        int l,r;
        vector <int> Z(text.size());
        Z[0] = 0;
        l = r = 0;
        for(int i = 1; i < text.size(); i++)
        {
                if(i > r){
                        l = r = i;

                while( r < text.size() && text[r - l] == text[r] ) r++;
                        Z[i] = (r - l);
                        r--;
                }
                else{
                        int k = i - l;

                        if(Z[k] < (r - i + 1)) Z[i] = Z[k];
                        else{
                                l = i;
                while( r < text.size() && text[r - l] == text[r] ) r++;
                                Z[i] = (r - l);
                                r--;
                        }
                }

        }
        return Z;
}
```

```
vector <int> search(string text, string pattern)
{
        text = pattern + '&' + text;
        vector<int> Z = build_Z(text);
        vector<int> res;
        for(int i = 0 ; i < Z.size();i++)
                if(Z[i] == pattern.size())
                        res.push_back(i - pattern.size() -1);
                return res;

}
```