Week 28: String Algorithms Advanced Applications

Topics: - KMP Algorithm for Pattern Matching - Z-Algorithm for Substring Search - Aho-Corasick Algorithm for Multiple Pattern Matching - Suffix Arrays and Longest Common Prefix (LCP) - Rolling Hash / Rabin-Karp Algorithm - Palindromic Substrings (Manacher's Algorithm)

Weekly Tips: - KMP uses prefix function to avoid redundant comparisons. - Z-Algorithm computes Z-array to find pattern occurrences efficiently. - Aho-Corasick builds a trie with fail links for multiple patterns. - Suffix Arrays sort all suffixes and help in substring search, LCP, and pattern counting. - Rolling hash allows fast substring comparison with modular arithmetic. - Manacher's algorithm finds all palindromic substrings in O(n).

Problem 1: KMP Pattern Matching Link: CSES String Matching **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> prefixFunction(const string &s){
    int n=s.size(); 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;
}
int main(){
    string t,p; cin>>t>>p;
    string s=p+'#'+t;
    vector<int> pi=prefixFunction(s);
    for(int i=0;i<pi.size();i++)
        if(pi[i]==p.size()) cout<<i-2*p.size()<<' '; // occurrences
    cout<<endl;
}
```

Explanation Comments: - Compute prefix function to find the longest border. - Concatenate pattern and text with delimiter. - Occurrences are positions where prefix equals pattern length.

Problem 2: Suffix Array and LCP Link: CSES Distinct Substrings **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```cpp
#include <bits/stdc++.h>
using namespace std;
string s; int n;
vector<int> sa,lcp;
void buildSA(){
    n=s.size();
    vector<int> rank(n),tmp(n);
    sa.resize(n); iota(sa.begin(),sa.end(),0);
    for(int i=0;i<n;i++) rank[i]=s[i];
    for(int k=1;;k*=2){
        auto cmp=[&](int i,int j){
            if(rank[i]!=rank[j]) return rank[i]<rank[j];
            int ri=i+k<n?rank[i+k]:-1;
            int rj=j+k<n?rank[j+k]:-1;
            return ri<rj;
        };
        sort(sa.begin(),sa.end(),cmp);
        tmp[sa[0]]=0;
        for(int i=1;i<n;i++) tmp[sa[i]]=tmp[sa[i-1]]+(cmp(sa[i-1],sa[i])?1:0);
        rank=tmp;
        if(rank[sa[n-1]]==n-1) break;
    }
}
void buildLCP(){
    lcp.assign(n,0);
    vector<int> rank_sa(n);
    for(int i=0;i<n;i++) rank_sa[sa[i]]=i;
    int h=0;
    for(int i=0;i<n;i++){
        if(rank_sa[i]>0){
            int j=sa[rank_sa[i]-1];
            while(i+h<n && j+h<n && s[i+h]==s[j+h]) h++;
            lcp[rank_sa[i]]=h;
            if(h>0) h--;
        }
    }
}
int main(){
    cin>>s;
    buildSA(); buildLCP();
    long long res=0;
    for(int i=0;i<n;i++) res+=n-sa[i]-lcp[i];
    cout<<res<<endl;
}
```

Explanation Comments: - Suffix array sorts all suffixes lexicographically. - LCP stores longest common prefix length between consecutive suffixes. - Number of distinct substrings = sum of (length - LCP) for each suffix.

End of Week 28 - Advanced string algorithms help solve pattern matching, substring counting, and text-processing problems. - Practice KMP, Z-Algorithm, Suffix Arrays, LCP, and Manacher’s algorithm extensively for ACM-ICPC contests.