Morgan and a String

Editorial by morze_47

You have to find the lexicographically minimal string, made of symbols on top of two stacks.

Let us be greedy. Just take that symbol, which is less.

Consider A = "JD"

and B = "CM".

So in the result string the first symbol should be 'C' as 'C' is less than 'J'.

After that, your stacks looks like so:

A = "JD",

B = "M".

'J' is less than 'M', so we take 'J'.

And so on.

But what if symbols on tops are same?

Consider A = "CA"

B = "CB"

We have two options: Take 'C' from A or take 'C' from B. If we take 'C' from B our next symbol would be 'B' and if we take 'C' from A our next symbol would be 'A', which is less then 'B'.

Now consider

A = c1 c2 c3 ... ck a1 a2 ... an,

B = c1 c2 c3 ... ck b1 b2 ... bm,

where a1 < b1.

If you take the first symbol from A, you would get

A = a1 a2 ... an,

B = ci ci+1 ... ck b1 b2 ... bm

On the other hand, if you take the first symbol from B, you would get

A = ci ci+1 ... ck a1 a2 ... an

B = b1 b2 ... bm

If you look at these examples, you can see that we chose the lower value character from the tops of the stacks.

But consider A = B = 'AAAA ... AAAA' ('A' 100000 times).

There we need to compare strings of length up to 100000 200000 times.

Here a suffix array comes to help us. If we build the suffix array for string A+'a'+B+'a', we can compare suffixes of strings A and B with O(1) time complexity.

Note If your solution used hashes with base 2^{32}, it should fail at tests > 10.

Set by morze_47

Problem Setter's code:

C++

```
//Author Vitalii
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <memory.h>
using namespace std;
#define FILL(a, val) memset((a), (val), sizeof(a));
namespace SuffixArray
{
    const int MAXSIZE = 200100;
    const int ALPHABET = 128;
    int p[MAXSIZE], c[MAXSIZE], cnt[MAXSIZE];
```

STATISTICS

Difficulty:	Expert
Success Rate:	37.4%
Time Complexity:	O(N log N)
Required Knowledge:	Strings, Suffix Arrays
Publish Date:	May 01 2014

Originally featured in 101 Hack April

Of the 838 contest participants, 123 (14.68%) submitted code for this challenge.

NEED HELP?

View discussions

View top submissions

```
    int pn[MAXSIZE], cn[MAXSIZE];
    vector<int> getSuffixArray(string& s)
    {
        FILL(cnt, 0);
        int n = s.size();
        for (int i = 0; i < n; ++i)
        {
            ++cnt[s[i]];
        }
        for (int i = 1; i < ALPHABET; ++i)
        {
            cnt[i] += cnt[i-1];
        }
        for (int i = 0; i < n; ++i)
        {
            p[--cnt[s[i]]] = i;
        }
        int count = 1;
        c[p[0]] = count-1;
        for (int i = 1; i < n; ++i)
        {
            if (s[p[i]] != s[p[i-1]])
                ++count;
            c[p[i]] = count - 1;
        }
        for (int h = 0; (1<<h) < n; ++h)
        {
            for (int i = 0; i < n; ++i)
            {
                pn[i] = p[i] - (1<<h);
                if (pn[i] < 0)
                    pn[i] += n;
            }

            FILL(cnt, 0);

            for (int i = 0; i < n; ++i)
            {
                ++cnt[c[i]];
            }
            for (int i = 1; i < count; ++i)
            {
                cnt[i] += cnt[i-1];
            }
            for (int i = n-1; i >= 0; --i)
            {
                p[--cnt[c[pn[i]]]] = pn[i];
            }
            count = 1;
            cn[p[0]] = count-1;
            for (int i = 1; i < n; ++i)
            {
                int pos1 = (p[i] + (1<<h))%n;
                int pos2 = (p[i-1] + (1<<h))%n;
                if (c[p[i]] != c[p[i-1]] || c[pos1] != c[pos2])
                    ++count;
                cn[p[i]] = count - 1;
            }
            for (int i = 0; i < n; ++i)
            {
                c[i] = cn[i];
            }
        }
        vector<int> res;
        res.reserve(n);
        for (int i = 0; i < n; ++i)
        {
            res.push_back(c[i]);
        }
        return res;
    }
}
string solve(string& a, string& b)
{
    a.push_back('a');
    b.push_back('b');
    string s = a+b;
    vector<int> suffixArray = SuffixArray::getSuffixArray(s);
    string res = "";
    int pos1=0, pos2=0;
    while (true)
    {
```

```
        if (pos1 >= (a.size()-1) && pos2 >= (b.size()-1))
        {
            break;
        }
        if (pos1 >= (a.size()-1))
        {
            res += b[pos2++];
            continue;
        }
        if (pos2 >= (b.size()-1))
        {
            res += a[pos1++];
            continue;
        }
        if (suffixArray[pos1] < suffixArray[a.size() + pos2])
            res += a[pos1++];
        else
            res += b[pos2++];
    }
    return res;
}
int main()
{
    ios_base::sync_with_stdio(false);
    int T;
    cin >> T;
    for (int t = 0; t < T; ++t)
    {
        string a,b;
        cin >> a >> b;
        cout << solve(a,b) << endl;
    }
    return 0;
}
```

Tested by devuy11

Problem Tester's code:

C++

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<algorithm>
#include<assert.h>
using namespace std;
#define N 100005
typedef long long int ll;
char str[2*N];
int H=0,Bucket[2*N],nBucket[2*N],index1[2*N];
struct Suffix{
    int idx;
    bool operator <(const Suffix suff) const{
        if(H==0)
            return str[idx]<str[suff.idx];
        else if(Bucket[idx]==Bucket[suff.idx])
            return Bucket[idx+H]<Bucket[suff.idx+H];
        else    return Bucket[idx]<Bucket[suff.idx];
    }
    bool operator ==(const Suffix suff) const{
        return !(*this<suff) && !(suff<*this);
    }
}Pos[2*N];
int UpdateBucket(int l)
{
    int start=0,c=0,id=0;
    for(int i=0;i<l;i++){
        if(i!=0 && !(Pos[i]==Pos[i-1])){
            start=i;
            id++;
        }
        if(i!=start)    c=1;
        nBucket[Pos[i].idx]=id;
    }
    memcpy(Bucket,nBucket,sizeof(Bucket));
    return c;
}
void suffixSort(int l)
```

```
{
    int c,x;
    H=0;
    for(int i=0;i<l;i++)    Pos[i].idx=i;
    sort(Pos,Pos+l);
    c=UpdateBucket(l);
    for(H=1;c;H=H*2){
        sort(Pos,Pos+l);
        c=UpdateBucket(l);
    }
}
void solve()
{
    int l,i,j;
    scanf("%s",str);
    int len1=strlen(str);
    for(i=0;i<len1;i++){
        if(str[i]>='A' && str[i]<='Z')    continue;
        assert(0);
    }
    assert(len1<=100000);
    str[len1]='z';
    scanf("%s",str+len1+1);            //making str1@str2
    int len2=strlen(str+len1+1);
    for(i=0;i<len2;i++){
        if(str[i+len1+1]>='A' && str[i+len1+1]<='Z')  continue;
                assert(0);
    }
    assert(len2<=100000);
    str[len1+len2+1]='z';            //adding $ at the end (as $ is the smallest character)
    str[len1+len2+2]='$';
    str[len1+len2+3]='\0';
    len2++;
    l=len1+len2;
    suffixSort(len1+len2+3);
    for(i=0;i<=l+1;i++){
        index1[Pos[i].idx]=i;
    }
    for(i=0,j=len1+1;i<len1 && j<l;){
        if(index1[i]<index1[j]){
            printf("%c",str[i]);
            i=i+1;
        }
        else{
            printf("%c",str[j]);
            j=j+1;
        }
    }
    while(i<len1){
        printf("%c",str[i]);
        i++;
    }
    while(j<l){
        printf("%c",str[j]);
        j=j+1;
    }
    printf("\n");
}
int main()
{
    int test;
    scanf("%d",&test);
    assert(test>=1);
    while(test--)    solve();
    return 0;
}
```

Feedback

Was this editorial helpful?

Yes No