2020.05.17-2020.05.23 周报

团队训练

团队会议

个人训练 - nikkukun

比赛

学习总结

本周推荐

题目链接

题意

题解

个人训练 - qxforever

比赛

2020.05.17 Educational Codeforces Round 87

题目	A	B	C1	C2	D	E	F	G
通过	√	√	√	√	√	√		
补题								

学习总结

本周推荐

题目链接

个人训练 - Potassium

比赛

学习总结

本周推荐

2015 ACM/ICPC Asia Regional Shanghai Online C Typewriter

题目链接

题意：有一个字符串 s ，给定打印每一种字母的代价。你可以花费单个字母的代价进行一次打字，或花费 $len\times A+2\times B$ 的代价粘贴一段已经打印过的、长度为 len 的字符串。求把这个字符串打印出来的代价最小值。

题解：很明显有 DP ，设 $f(i)$ 表示打印前 i 个字符的代价，则有递推式

$$ f(i)=\min\{f(i-1)+val(s_i),\min_{j<i,s[j+1,i]\subseteq s[1,j]}\{f(j)+(i-j)\times A+2\times B\}\} $$

然后我们发现这个 DP 是 $O(n^2)$ 的，不太行，考虑优化。

设 $left_i$ 表示满足 $j<i,s[j+1,i]\subseteq s[1,j]$ 的最大 j ，很明显 $left_i$ 是单调递增的。那么假设我们求出了 $left_i$ ，很明显由于最后一个字符的加入只会带来他复制与不复制的选择，故 DP 的决策点 j_i 也是不减的，于是我们可以用一个单调增的优先队列优化这个 DP 。

考虑如何求出 $left_i$ ，需要维护 $s[1,j]$ 的子串状态，用一个节点状态 cur 存当前 $s[j+1,i]$ 的状态，并支持随时变为长度更短的后缀，很自然想到用后缀自动机维护。

然后这题就做完了，但有一个细节被遗漏搞了很久：

在随着 j 的变大， cur 变为 $fa[cur]$ 的时候，本以为每次 j 自增只会带来最多一次跳父亲边的情况，故使用了 if 导致错误。当插入新字符时，如果当前节点的父亲被修改，而 endpos 含义也发生变化时，可能会跳多次父边，故需要提前保存父亲节点编号，或者使用 while 跳父边。

```
#include<cstdio>
#include<algorithm>
#include<queue>
#include<map>
#include<cstring>
#include<cmath>
#include<cstdlib>
#include<set>
#include<unordered_map>
#include<vector>
typedef long long ll;
using namespace std;
#define pii pair<int,int>
#define pll pair<ll,ll>
#define pb push_back
#define mp make_pair
#define fi first
#define se second
```

```
#define N 100010
#define P 26
ll val[P];
char s[N];
// SAM
struct SAM{
    int tr[N<<1][P],len[N<<1],fa[N<<1];
    int last,tot;
    void init(){
        tot=0;
        last=tot=newnode();
    }
    int newnode(){
        ++tot;
        memset(tr[tot],0,sizeof(tr[tot]));
        fa[tot]=len[tot]=0;
        return tot;
    }
    void insert(int c){
        int p=last,np=newnode();
        last=np;
        len[np]=len[p]+1;
        while(p&&!tr[p][c])tr[p][c]=np,p=fa[p];
        if(!p)fa[np]=1;
        else{
            int q=tr[p][c];
            if(len[q]==len[p]+1)fa[np]=q;
            else{
                int nq=newnode();
                len[nq]=len[p]+1;
                memcpy(tr[nq],tr[q],sizeof(tr[q]));
                fa[nq]=fa[q];
                fa[q]=fa[np]=nq;
                while(tr[p][c]==q)tr[p][c]=nq,p=fa[p];
            }
        }
    }
}sam;
// monotone queue
ll f[N];
int hd,tl;
pll Q[N];
void insert(int p,ll v){
    while(hd<tl&&Q[tl-1].se>v)tl--;
    Q[tl++]=mp(p,v);
}
ll get(int p){
    while(hd<tl&&Q[hd].fi<p)hd++;
    if(hd>=tl)return 1e18;
    else return Q[hd].se;
}
```

```
int main(){
    int T,i,j,cas=0;
    ll A,B;
    scanf("%d",&T);
    while(T--){
        hd=tl=0;
        scanf("%s",s+1);
        int n=strlen(s+1);
        for(i=0;i<26;i++)scanf("%lld",&val[i]);
        scanf("%lld%lld",&A,&B);
        sam.init();
        j=0;
        int cur=1; // [j+1...i-1]
        for(i=1;i<=n;i++){
            //int F=sam.fa[cur];
            while(j+1<=i-1&&!sam.tr[cur][s[i]-'a']){
                // correct
                if(sam.len[sam.fa[cur]]>=i-1-(j+1))cur=sam.fa[cur];
                sam.insert(s[++j]-'a');
                while(sam.fa[cur]&&sam.len[sam.fa[cur]]>=i-1-
j)cur=sam.fa[cur];
                // correct 2
                /*if(sam.len[F]>=i-1-(j+1))cur=F,F=sam.fa[cur];
                sam.insert(s[++j]-'a');*/
                // wrong
                /*if(sam.len[sam.fa[cur]]>=i-1-(j+1))cur=sam.fa[cur];
                sam.insert(s[++j]-'a');*/
            }
            if(sam.tr[cur][s[i]-'a']){
                cur=sam.tr[cur][s[i]-'a'];
            }else{
                cur=1;
                hd=tl=0;
                sam.insert(s[++j]-'a');
            }
            f[i]=min(f[i-1]+val[s[i]-'a'],get(j)+1LL*i*A);
            insert(i,f[i]-1LL*i*A+2*B);
        }
        printf("Case #%d: %lld\n",++cas,f[n]);
    }
    return 0;
}
```