字符串

1. 字符串基础

注意事项：
1.对于字符串问题，最好使用char []来存储，不要用string，否则可能会占用大量内存及减低速度
2.strlen(char [])，以及相似方法的复杂度均为O(n)，千万不要用在循环里！
3.map储存string是O(n)比较的，unordered_map会自动hash。
基础概念：
1.**字符集**：字符集 Σ 建立了一个全序关系，对于Σ中的任意两个不同的元素都可以比较大小，其元素称为字符。

2.**字符串**：n个字符顺次排列形成的序列，n为S的长度，记为|S|。第i个字母为S[i]，有些地方为S[i-1]。

3.**子串**：S[i..j]，i≤j，也就是依次排列的S[i],S[i+1],...,S[j]。

4.**子序列**：从S中将若干元素提取出来并不改变相对位置形成的序列，也就是S[p1],S[p2],...,S[pk]，1≤ p1≤p2≤...≤pk≤|S|

5.**前后缀**：前缀为从串首开始到某个位置结束的一个特殊子串，到位置i的前缀记为Preffix(S,i)。后缀为从某个位置开始到整个串末尾结束的一个特殊子串，从位置i开始的后缀记为Suffix(S,i)。真前缀后缀就是去除字符串本身的前缀后缀。

6.**回文串**：正着写和倒着写一样的字符串。

7.**字典序**：以第i个字符作为第i关键字进行大小比较，空字符小于字符集内任何字符。

8.**border**：对字符串S和0≤r<|S|，若S长度为r的前缀和长度为r的后缀相等，就称S长度为r的前缀是S的 border。所有前缀的border组成的函数也就是前缀函数。

9.**字符串的周期**：对字符串S和0<p≤|S|，S[i]=S[i+p]对所有0≤i≤|S|-p-1成立，则p为S的周期

10.**自动机**：竞赛中所说的“自动机”一般都指“确定有限状态自动机”(DFA)。

①**字符集** Σ，该自动机只能输入这些字符。

②**状态集合**Q。如果把一个 DFA 看成一张有向图，那么 DFA 中的状态就相当于图上的顶点。

③**起始状态**$start$，$start \in Q$，是一个特殊的状态。起始状态一般用s表示，为了避免混淆，这里使用$start$。

④**接受状态集合** F，$F \subseteq Q$，是一组特殊的状态。

⑤**转移函数** δ，δ是一个接受两个参数返回一个值的函数，其中第一个参数和返回值都是一个状态，第二个参数是字符集中的一个字符。如果把一个 DFA 看成一张有向图，那么 DFA 中的转移函数就相当于顶点间的边，而每条边上都有一个字符。

2.标准库

因为效率问题，这里只讲标准库而不讲string
1.**strlen(S)**，返回从S[0]到'\0'的长度，不开O_2优化时，时间复杂度为O(n)
2.**strcmp(S1,S2)**，字典序S1>S2返回正数，=返回0，<返回负数，不同平台的正负数返回不同
3.**strcpy(S1,S2)**，复制S2到S1中
4.**strncpy(S1,S2,cnt)**，复制S2 cnt个字符到S1中，S2长度不够补充'\0'
5.**strcat(S1,S2)**，S1后面接上S2
6.**strstr(S1,S2)**，找到S2在S1第一次出现的地址，如果找不到，返回NULL，效率为O(n^2)，不如kmp

3.字符串hash

hash的本质为把字符串通过hash函数映射成值域较小、方便比较的范围。
hash函数的性质：
hash函数值不一样时，字符串一定不一样；
hash函数一样时，字符串大概率一样。
通常用的hash为多项式hash，
$f(s)=\sum_{i=0}^{len-1}s[i]*base^{len-i}\%M$
或者 $f(s)=\sum_{i=0}^{len-1}s[i]*base^{i}\%M$

```
//双模，可取区间
const int bs=233;
const int M[2]={998244353,1000000007};
int bsp[2][N];
void init(int n){
    bsp[0][0]=bsp[1][0]=1;
    for (int k=0;k<2;k++){
        for (int i=1;i<=n;i++){
            bsp[k][i]=bsp[k][i-1]*bs%M[k];
        }
    }
}
int hash(char *s,int k){
    int len=strlen(s);
    int ans=0;
    for (int i=0;i<len;i++){
        ans=(ans+(s[i]-'a')*bsp[k][i])%M[k];
    }
    return ans;
}
struct Hash{
    char s[N];
    int a[2][N];
    void init(){
        int len=strlen(s);
        a[0][0]=a[1][0]=0;
        for (int k=0;k<2;k++){
            for (int i=0;i<len;i++){
                a[k][i]=(a[k][max(i-1,0ll)]+(s[i]-'a')*bsp[k][i])%M[k];
            }
        }
    }
    int qpow(int x,int y,int p){
        int ans=1;
        while (y){
            if (y&1)ans=ans*x%p;
            x=x*x%p;y>>=1;
        }return ans;
    }
    int sub(int l,int r,int k){
        return (a[k][r]-(l>0?a[k][l-1]:0)+M[k])%M[k]*qpow(bsp[k][l],M[k]-2,M[k])%M[k];
    }
}hs;
```

4.Tire

```
struct Trie{
    int ch[N][26],cnt=0;
    int newnode(){
        cnt++;
        for (int i=0;i<26;i++)ch[cnt][i]=0;
        return cnt;
    }
    void init(){
        cnt=0;
        newnode();
    }
    void insert(char *s){
```

```
        int len=strlen(s);
        int rt=1;
        for (int i=0;i<len;i++){
            int t=s[i]-'a';
            if (!ch[rt][t])ch[rt][t]=newnode();
            rt=ch[rt][t];
        }
    }
};
```

1.树上最大异或路径(高位到低位建树，每次插入根到点的异或和，然后询问与该值异或最大的值)

2.01字典树，从低位到高位建树

```
#include<bits/stdc++.h>
using namespace std;
const int N=5e5*23+10;
struct Trie_01{
    const int M=20;
    //xorv[rt]指以rt为根的子树维护的异或和
    //w[rt]指有几个数经过这条rt到它父亲的边
    int ch[N][2],cnt=0,w[N],xorv[N];
    int newnode(){
        ++cnt;
        ch[cnt][0]=ch[cnt][1]=w[cnt]=xorv[cnt]=0;
        return cnt;
    }
    void pushup(int rt){
        xorv[rt]=w[rt]=0;
        if (ch[rt][0]){
            w[rt]+=w[ch[rt][0]];
            xorv[rt]^=xorv[ch[rt][0]]<<1;
        }
        if (ch[rt][1]){
            w[rt]+=w[ch[rt][1]];
            xorv[rt]^=xorv[ch[rt][1]]<<1|(w[ch[rt][1]]&1);
        }
    }
    void insert(int rt,int x,int dep){
        if (dep==M){
            w[rt]++;return;
        }
        if (!ch[rt][x&1])ch[rt][x&1]=newnode();
        insert(ch[rt][x&1],x>>1,dep+1);
        pushup(rt);
    }
    void erase(int rt,int x,int dep){
        if (dep==M){
            w[rt]--;return;
        }
        erase(ch[rt][x&1],x>>1,dep+1);
        pushup(rt);
    }
    void addall(int rt){//全部数+1
        swap(ch[rt][0],ch[rt][1]);
        if (ch[rt][0])addall(ch[rt][0]);
        pushup(rt);
    }
    void suball(int rt){//全部数-1
        swap(ch[rt][0],ch[rt][1]);
        if (ch[rt][1])suball(ch[rt][1]);
        pushup(rt);
    }
    int merge(int a,int b){//复杂度为小的数字集合的复杂度(类似启发式合并的复杂度)，trie合并不仅仅限于01-trie
        if(!a)return b;//用这种合并时因为每个数多一个根，要数组大小+1*N
        if(!b)return a;
        w[a]=w[a]+w[b];
        xorv[a]^=xorv[b];
        for (int i=0;i<2;i++)ch[a][i]=merge(ch[a][i],ch[b][i]);
        return a;
```

```
    }
}tr;
//一棵树，三种操作，1.与x距离1的点权值+1。2.x权值-y。3.询问与x距离1的点权值异或和
int n,m;
vector<int>v[N];
int f[N],rt[N];
int ad[N],a[N];
void dfs(int x,int fa){
    f[x]=fa;
    for (int i:v[x]){
        if (i==fa)continue;
        dfs(i,x);
    }
}
signed main(){
    int op,x,y;
    cin>>n>>m;
    for (int i=1;i<n;i++){
        scanf("%d%d",&x,&y);
        v[x].push_back(y);v[y].push_back(x);
    }
    dfs(1,0);
    for (int i=1;i<=n;i++)scanf("%d",&a[i]);
    for (int i=1;i<=n;i++)rt[i]=tr.newnode();
    for (int i=2;i<=n;i++)tr.insert(rt[f[i]],a[i],0);
    while (m--){
        scanf("%d",&op);
        if (op==1){
            scanf("%d",&x);
            tr.addall(rt[x]);
            ad[x]++;
            if (f[x]!=0){
                int t=f[f[x]];
                if (t)tr.erase(rt[t],a[f[x]]+ad[t],0);
                a[f[x]]++;
                if (t)tr.insert(rt[t],a[f[x]]+ad[t],0);
            }
        }
        else if (op==2){
            scanf("%d%d",&x,&y);
            if (f[x])tr.erase(rt[f[x]],a[x]+ad[f[x]],0);
            a[x]-=y;
            if (f[x])tr.insert(rt[f[x]],a[x]+ad[f[x]],0);
        }
        else{
            scanf("%d",&x);
            printf("%d\n",(tr.xorv[rt[x]]^(a[f[x]]+ad[f[f[x]]])));
        }
    }
    return 0;
}
```

3.最小异或生成树Xor MST

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5*17+10;
int n;
int a[N];
struct node{
    const int M=30;
    int ch[N][2],cnt=0;
    vector<int>v[N];
    int newnode(){
        ++cnt;ch[cnt][0]=ch[cnt][1]=0;
        return cnt;
    }
    void init(){
        cnt=0;newnode();
        for (int i=1;i<=n;i++)v[i].clear();
```

```
    }
    void insert(int x){
        int rt=1;
        for (int i=M;i>=0;i--){
            int t=(x>>i&1);
            if (!ch[rt][t])ch[rt][t]=newnode();
            rt=ch[rt][t];
            v[rt].push_back(x);
        }
    }
    int query(int rt,int x,int dep){
        int ans=0;
        for (int i=dep;i>=0;i--){
            int t=(x>>i&1);
            if (ch[rt][t])rt=ch[rt][t];
            else rt=ch[rt][t^1],ans+=(1<<i);
        }
        return ans;
    }
    long long solve(int rt,int dep){
        int l=ch[rt][0],r=ch[rt][1];
        long long ans=0;
        if (l&&r){//左右子树连一条最小的边
            ans=1e16;
            if (v[l].size()>v[r].size())swap(l,r);
            for (int i:v[l]){
                ans=min(ans,query(r,i,dep-1)+(1ll<<dep));
            }
        }
        if (l)ans+=solve(l,dep-1);
        if (r)ans+=solve(r,dep-1);
        return ans;
    }
}tr;
signed main(){
    cin>>n;
    for (int i=1;i<=n;i++)scanf("%lld",&a[i]);
    sort(a+1,a+1+n);
    tr.init();
    a[0]=-1;
    for (int i=1;i<=n;i++)if (a[i]!=a[i-1])tr.insert(a[i]);
    cout<<tr.solve(1,30)<<endl;
    return 0;
}
```

5.前缀函数(KMP)

前缀函数(next数组)：next[i]为子串S[0..i]相等的真前缀和真后缀的最长长度,也就是border，和右边字符没关系

1.字符串匹配kmp，查找S2在S1中出现的位置

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
char s1[N],s2[N];
int nx[N];
void get_nx(char *s) {
    int len=strlen(s);
    for (int i=1,j=0;i<len;i++){
        while (j&&s[i]!=s[j])j=nx[j-1];
        if (s[i]==s[j])j++;
        nx[i]=j;
    }
```

```
}
signed main(){
    cin>>s1>>s2;
    get_nx(s2);
    int len1=strlen(s1),len2=strlen(s2);
    int i=0,j=0;
    for(int i=0,j=0;i<len1;i++){
        while(j&&s1[i]!=s2[j])j=nx[j-1];
        if (s1[i]==s2[j])j++;
        if (j==len2){cout<<i-j+1<<endl;j=nx[j-1];}
    }
    return 0;
}
```

2.字符串周期：len-nx[len-1]也就是S的最小周期

3.①统计字符串S每个前缀在S中出现的次数

```
for (int i=0;i<len;i++)if(nx[i])ans[nx[i]-1]++;
for (int i=len-1;i>=0;i--)if(nx[i])ans[nx[i]-1]+=ans[i];
for (int i=0;i<len;i++)ans[i]++;
```

②统计字符串S1每个前缀在S2中出现的次数

构造S1+'#'+S2的字符串,只管nx[p]，$p \geq |S1|+1$

```
scanf("%s%s",s1,s2);
int len1=strlen(s1);
strcat(s1,"#");strcat(s1,s2);
get_nx(s1);
int len=strlen(s1);
for (int i=len1+1;i<len;i++)if(nx[i])ans[nx[i]-1]++;
for (int i=len-1;i>=0;i--)if(nx[i])ans[nx[i]-1]+=ans[i];
```

4.本质不同子串数目

O(n^2)，通过迭代的方式，已知S的本质不同子串数目，再在前面或者后面加上一个字符，求含该字符的前缀在S中出现的次数为1的个数，加之。支持前后删减一个字符。

5.kmp自动机

如下代码，建立nxt$[N][26]$，nxt$[i][c]$代表当前的nx函数为i，下一位为c的话，下一位的nx函数为多少；当然也可以以nxt$[i][c]$代表当前的匹配位为i，下一位为c的话，下一位的nx函数为多少来建立kmp自动机

```
int nx[N];
int nxt[N][26];
void get_nx(char *s) {
    int len=strlen(s);
    for (int i=1,j=0;i<len;i++){
        while (j&&s[i]!=s[j])j=nx[j-1];
        if (s[i]==s[j])j++;
        nx[i]=j;
    }
    for (int i=0;i<len;i++){
        for (int c=0;c<26;c++){
            if (i&&c+'a'!=s[i])nxt[i][c]=nxt[nx[i-1]][c];
            else nxt[i][c]=i+(c+'a'==s[i]);
        }
    }
}
```

6.Z函数(EXKMP)

z[i]为S和S[i..len-1]的最长公共前缀，z[0]通常不定义为0，和右边字符有关系，不能从左往右迭代


```
int z[N];
void get_z(char *s){//z函数在重新计算时要重新清零
    int len=strlen(s);
    for (int i=1,l=0,r=0;i<len;i++){
        if (i<=r)z[i]=min(r-i+1,z[i-l]);//不超过当前长度的已经匹配过的长度
        while(i+z[i]<len&&s[z[i]]==s[i+z[i]])++z[i];
        if (i+z[i]-1>r)l=i,r=i+z[i]-1;
    }
}
```

1.字符串匹配，查找S2在S1中出现的位置

S2+'#'+S1，求z函数，z[i]为len2的位置为匹配位置

2.字符串的周期

从小到大循环，找到i+z[i]-1==len-1的第一个位置(同next数组)，n-z[i]就是其周期

3.本质不同子串

O(n^2)，通过迭代的方式，已知S的本质不同子串数目，再在前面或者后面加上一个字符，把新增的字符通过反转之类的操作放到第一个位置，求一遍z函数，后方没有出现过的字符就是多出的本质不同子串数。

7.AC自动机和fail树

fail指针指向存在的最长后缀处标号

```
struct Trie{
    int ch[N][26],f[N],cnt,ans[N];
    int newnode(){
        cnt++;
        for (int i=0;i<26;i++)ch[cnt][i]=0;
        f[cnt]=1;ans[cnt]=0;
        return cnt;
    }
    void init(){
        cnt=0;
        newnode();
    }
    void insert(char *s){
        int len=strlen(s),rt=1;
        for (int i=0;i<len;i++){
            int t=s[i]-'a';
            if (!ch[rt][t])ch[rt][t]=newnode();
            rt=ch[rt][t];
        }
        ans[rt]++;
    }
    void build(){
        queue<int>q;
        for (int i=0;i<26;i++){//预处理第二层，防止ch[1][i]=0使得f[ch[1][i]]指向自己的节点
            if(ch[1][i])q.push(ch[1][i]);
            else ch[1][i]=1;
        }
        while(!q.empty()){
            int t=q.front();q.pop();
            for (int i=0;i<26;i++){
                if (ch[t][i]){
                    f[ch[t][i]]=ch[f[t]][i];//ch[t][i]的fail为t的fail的i连边
                    q.push(ch[t][i]);
                }
                else ch[t][i]=ch[f[t]][i];//保证ch[t][i]!=0,连fail节点，同时方便后面查询
            }
        }
```

```
    }
    int query(char *s) {
        int rt=1,res=0;
        int len=strlen(s);
        for (int i=0;i<len;i++){
            rt=ch[rt][s[i]-'a'];
            for (int j=rt;j!=1&&ans[j]!=-1;j=f[j]){
                res+=ans[j],ans[j]=-1;
            }
        }
        return res;
    }
}AC;
```

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
struct AC{
    int cnt,rt;
    int ans[N],ch[N][26],f[N];
    int newnode(){
        ++cnt;
        for (int i=0;i<26;i++)ch[cnt][i]=0;
        f[cnt]=ans[cnt]=0;
        return cnt;
    }
    void init(){
        cnt=0;
        rt=newnode();
    }
    void insert(string s){
        int l=s.size();
        int x=rt;
        for (int i=0;i<l;i++){
            int t=s[i]-'a';
            if (!ch[x][t])ch[x][t]=newnode();
            x=ch[x][t];
        }
        ans[x]++;
    }
    void build(){
        queue<int>q;
        q.push(rt);
        while (!q.empty()){
            int t=q.front();q.pop();
            for (int i=0;i<26;i++){
                int to=ch[t][i];
                if (!to)continue;
                if (t==rt)f[to]=rt;
                else{
                    int fa=f[t];
                    while (fa!=rt&&!ch[fa][i])fa=f[fa];
                    if (ch[fa][i])f[to]=ch[fa][i];
                    else f[to]=rt;
                }
                q.push(to);
            }
        }
    }
    int query(string s){
        int l=s.size(),x=rt,res=0;
        for(int i=0;i<l;i++){
            int t=s[i]-'a';
            while(x!=rt&&!ch[x][t])x=f[x];
            if (!ch[x][t])continue;
            for (int j=ch[x][t];j!=rt&&ans[j]!=-1;j=f[j]){
                res+=ans[j],ans[j]=-1;
            }
            x=ch[x][t];
        }
```

```
        return res;
    }
}ac;
string s;
signed main(){
    int n;
    cin>>n;
    ac.init();
    for (int i=1;i<=n;i++){
        cin>>s;
        ac.insert(s);
    }
    ac.build();
    cin>>s;
    cout<<ac.query(s)<<endl;
    //system("pause");
    return 0;
}
// 2 a aa aaaaaaaa
```

8.后缀数组

sa[i]为排名为i的后缀的起始位置的下标，rk[i]为后缀i的排名，sa[rk[i]]=i，rk[sa[i]]=i

```
#include<bits/stdc++.h>//O(nlog^2n),特殊数据会卡，例如 a,...,z,a...z,...
using namespace std;
const int N=1e6+10;
char s[N];
int sa[N],rk[N],oldrk[N];
int w;
signed main(){
    scanf("%s",s+1);
    int n=strlen(s+1);
    for (int i=1;i<=n;i++)sa[i]=i,rk[i]=s[i];
    for (w=1;w<n;w<<=1){
        sort(sa+1,sa+n+1,[](int x,int y){
            return rk[x]==rk[y]?rk[x+w]<rk[y+w]:rk[x]<rk[y];
        });
        memcpy(oldrk,rk,sizeof rk);
        int p,i;
        for (p=0,i=1;i<=n;i++){
            if (oldrk[sa[i]]==oldrk[sa[i-1]]&&oldrk[sa[i]+w]==oldrk[sa[i-1]+w]){
                rk[sa[i]]=p;
            }
            else rk[sa[i]]=++p;
        }
        if (p==n)break;
    }
    for (int i=1;i<=n;i++)cout<<sa[i]<<" ";
    return 0;
}
```

```
#include<bits/stdc++.h>//O(nlog^n)
using namespace std;
const int N=1e6+10;
char s[N];
int n;
int sa[N],rk[N],oldrk[N],cnt[N],id[N];
void SA(){
    n=strlen(s+1);
    int m=max(300,n);
    int i,j,w;
    for (i=1;i<=n;i++)++cnt[rk[i]=s[i]];
    for (i=1;i<=m;i++)cnt[i]+=cnt[i-1];
    for (i=n;i>=1;i--)sa[cnt[rk[i]]--]=i;
    for (w=1;w<n;w<<=1){
        for (i=1;i<=m;i++)cnt[i]=0;
        for (i=1;i<=n;i++)id[i]=sa[i];
        for (i=1;i<=n;i++)++cnt[rk[id[i]+w]];
```

```
        for (i=1;i<=m;i++)cnt[i]+=cnt[i-1];
        for (i=n;i>=1;i--)sa[cnt[rk[id[i]+w]]--]=id[i];
        for (i=1;i<=m;i++)cnt[i]=0;
        for (i=1;i<=n;i++)id[i]=sa[i];
        for (i=1;i<=n;i++)++cnt[rk[id[i]]];
        for (i=1;i<=m;i++)cnt[i]+=cnt[i-1];
        for (i=n;i>=1;i--)sa[cnt[rk[id[i]]]--]=id[i];
        memcpy(oldrk,rk,sizeof rk);
        int p,i;
        for (p=0,i=1;i<=n;i++){
            if (oldrk[sa[i]]==oldrk[sa[i-1]]&&oldrk[sa[i]+w]==oldrk[sa[i-1]+w]){
                rk[sa[i]]=p;
            }
            else rk[sa[i]]=++p;
        }
        if (p==n)break;
    }
}
signed main(){
    scanf("%s",s+1);
    SA();
    for (int i=1;i<=n;i++)cout<<sa[i]<<" ";
    return 0;
}
```

1.每次从S中取队首或者队尾的字符加入新队列，使得最后字典序最小

贪心，如果队首字符<队尾字符，插入队首；如果队首字符>队尾字符，插入队尾；如果相等就比较下一个字符，总体而言就是比较字典序

S+'超小值'+S^T，S^T代表反串，求出rk，rk[超小值前面的]为队首的字典序排名，rk[超小值后面的]为队尾的字典序排名

```
cin>>n;
for (int i=1;i<=n;i++)cin>>s[i];s[n+1]='\0';
memcpy(t,s,sizeof s);
reverse(t+1,t+1+n);
strcat(s+1,"0");//此处"0"小于所有其他字符
strcat(s+1,t+1);
n=strlen(s+1);
SA();
int l=1,r=(n-1)>>1,tot=0;
while (l<=r){
    printf("%c",rk[l]<rk[n-r+1]?s[l++]:s[r--]);
    if (++tot%80==0) printf("\n");
}
```

2.height[i]为lcp(sa[i],sa[i-1])，也就是第i名的后缀和第i-1名的后缀的最长公共前缀，height[1]为0。

对于j，k，$rk[i]<rk[k]$，易知具有以下性质1，$lcp(suffix(j),suffix(k))=min(height[rk[j]+1],height[rk[j]+2],\dots,height[rk[k]])$

同时，height数组具有以下性质：$height[rk[i]]\geq height[rk[i-1]]-1$

证明：设$sa[rk[i-1]-1]=k$,即排在后缀i-1前一名的为k

$if\ \ height[rk[i-1]]\leq 1$：显然成立

$else\ \ height[rk[i-1]]$>1：

$\because lcp(suffix(k),suffix(i-1))>1$

$\therefore suffix(k)$，$suffix(i-1)$前缀至少存在一个字母是相同的

$\therefore rk[k+1]<rk[i],lcp(suffix(k+1),suffix(i))=height[rk[i-1]]-1$

$\therefore$由性质1得，$height[rk[i]]\geq height[rk[i-1]]-1$

```
int ht[N];
for (int i=1,k=0;i<=n;i++) {
    if(k)k--;
    while(s[i+k]==s[sa[rk[i]-1]+k])k++;
    ht[rk[i]]=k;
}
```

由性质1+rmq算法或者线段树，可以求得lcp(suffix(j)，suffix(k))

3.字符串子串A=S[a..b],B=S[c..d]大小关系，已知S[a..b],S[c..d]长度可以O(1)判断，

lcp(a,c) $\geq$ min(|A|,|B|)时，A<B $\iff$ |A|<|B|

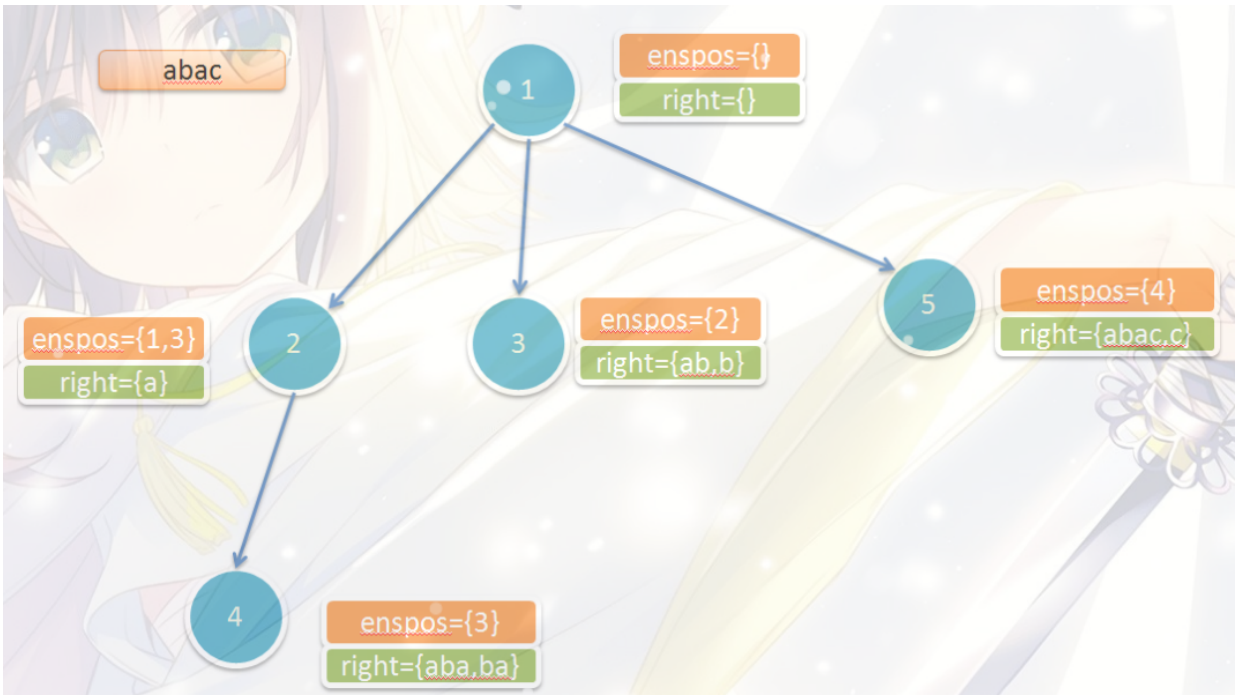
else，A<B $\iff$ rk[a]<rk[b]

4.本质不同子串

所有后缀的所有前缀就是字符串的所有子串

$\frac{n*(n+1)}{2}$-$\sum_1^n height[i]$

9.后缀自动机(SAM)



1.后缀自动机

```
struct SAM{
    struct node{
        int ch[26];
        int len,fa;
        node(){memset(ch,0,sizeof(ch));len=0;}
    }a[N<<1];
    int las=1,tot=1,ha[N];
    int newnode(){
        ++tot;
        for (int i=0;i<26;i++)a[tot].ch[i]=0;
        a[tot].len=a[tot].len=0;ha[tot]=0;
        return tot;
    }
    void init(){
        las=tot=1;
    }
    void add(int x){
        int p=las,np=las=newnode();ha[tot]=1;//
        a[np].len=a[p].len+1;
        for(;p&&!a[p].ch[x];p=a[p].fa)a[p].ch[x]=np;
        if(!p)a[np].fa=1;//case 1
        else{
            int q=a[p].ch[x];
            if(a[q].len==a[p].len+1)a[np].fa=q;//case 2
            else{
                int nq=newnode();a[nq]=a[q];
                a[nq].len=a[p].len+1;
                a[q].fa=a[np].fa=nq;
                for(;p&&a[p].ch[x]==q;p=a[p].fa)a[p].ch[x]=nq;//case 3
            }
```

```
        }
    }
}sam;
```

2.广义后缀自动机

法一：直接将多个串加分隔符合并成一条串

法二：对于每个串从1节点开始add

```
struct SAM{
    struct node{
        int ch[26];
        int len,fa;
        node(){memset(ch,0,sizeof(ch));len=0;}
    }a[N<<1];
    int las=1,tot=1,ha[N];
    int newnode(){
        ++tot;
        for (int i=0;i<26;i++)a[tot].ch[i]=0;
        a[tot].len=a[tot].len=0;ha[tot]=0;
        return tot;
    }
    void init(){
        las=tot=1;
    }
    int add(int x,int las){
        if(a[las].ch[x]){
            int p=las,t=a[p].ch[x];
            if(a[p].len+1==a[t].len) return t;
            else{
                int y=newnode();
                a[y].len=a[p].len+1;
                memcpy(a[y].ch,a[t].ch,sizeof a[t].ch);
                while(p&&a[p].ch[x]==t)a[p].ch[x]=y,p=a[p].fa;
                a[y].fa=a[t].fa;a[t].fa=y;
                return y;
            }
        }
        int p=las,np=las=newnode();ha[tot]=1;//
        a[np].len=a[p].len+1;
        for(;p&&!a[p].ch[x];p=a[p].fa)a[p].ch[x]=np;
        if(!p)a[np].fa=1;//case 1
        else{
            int q=a[p].ch[x];
            if(a[q].len==a[p].len+1)a[np].fa=q;//case 2
            else{
                int nq=newnode();a[nq].fa=a[q].fa;
                for (int i=0;i<26;i++)a[nq].ch[i]=a[a[q].ch[i]].len?a[q].ch[i]:0;
                a[nq].len=a[p].len+1;
                a[q].fa=a[np].fa=nq;
                for(;p&&a[p].ch[x]==q;p=a[p].fa)a[p].ch[x]=nq;//case 3
            }
        }
        return np;
    }
    void getsum(){//本质不同子串
        long long ans=0;
        for(register int i=2;i<=tot;++i)ans+=a[i].len-a[a[i].fa].len;
        printf("%lld\n",ans);
    }
}sam;
```

法三：先建好字典树再处理其他元素

```
struct SAM{
    struct node{
        int ch[26];
        int len,fa;
        node(){memset(ch,0,sizeof(ch));len=0;}
```

```
    }a[N<<1];
    int las=1,tot=1;
    int newnode(){
        ++tot;
        for (int i=0;i<26;i++)a[tot].ch[i]=0;
        a[tot].len=a[tot].fa=0;
        return tot;
    }
    void init(){
        las=tot=1;
    }
    int add(int x,int las){
        int p=las,np=las=a[las].ch[x];
        if(a[np].len)return np;
        a[np].len=a[p].len+1;
        p=a[p].fa;
        for(;p&&!a[p].ch[x];p=a[p].fa)a[p].ch[x]=np;
        if(!p)a[np].fa=1;//case 1
        else{
            int q=a[p].ch[x];
            if(a[q].len==a[p].len+1)a[np].fa=q;//case 2
            else{
                int nq=newnode();a[nq].fa=a[q].fa;
                for (int i=0;i<26;i++)a[nq].ch[i]=a[a[q].ch[i]].len?a[q].ch[i]:0;
                a[nq].len=a[p].len+1;
                a[q].fa=a[np].fa=nq;
                for(;p&&a[p].ch[x]==q;p=a[p].fa)a[p].ch[x]=nq;//case 3
            }
        }
        return np;
    }
    void buildtrie(char *s){
        int len=strlen(s),rt=1;
        for (int i=0;i<len;i++){
            int x=s[i]-'a';
            if(!a[rt].ch[x])a[rt].ch[x]=newnode();
            rt=a[rt].ch[x];
        }
    }
    void buildsam(){
        queue<pair<int,int> >q;
        for (int i=0;i<26;i++)if(a[1].ch[i])q.push({i,1});
        while (!q.empty()){
            auto t=q.front();
            q.pop();
            int las=add(t.first,t.second);
            for(int i=0;i<26;i++)if(a[las].ch[i])q.push({i,las});
        }
    }
    void getsum(){
        long long ans=0;
        for (int i=2;i<=tot;i++)ans+=a[i].len-a[a[i].fa].len;
        printf("%lld\n",ans);
    }
}sam;
```

10.序列自动机

```
//求g[i][j]为[i,n]第一个j出现的位置
const int inf=0x3f3f3f3f;
int nxt[N][27];
void init(char *s){
    int len=strlen(s);
    for(int i=0;i<26;i++) nxt[len][i]=inf;
    for(int i=len-1;i>=0;i--){
        for(int j=0;j<26;j++){
            nxt[i][j]=nxt[i+1][j];
        }
        nxt[i][s[i]-'a']=i;
    }
}
```

11.Manacher

```
int Init(){
    int len=ss.size();
    str[0]='$',str[1]='#';
    int j=1;
    for(int i=0;i<len;++i)
    str[++j]=ss[i],str[++j]='#';
    str[++j]='\0';
    return j;
}
int Manancher(){
    int len=Init(),maxv=0,id=1,mx=1;
    for(int i=1;i<len;++i){
    if(mx>i) p[i]=min(p[2*id-i],mx-i);
        else p[i]=1;
        while(str[i-p[i]]==str[i+p[i]]) p[i]++;
        if(mx<i+p[i]) id=i,mx=i+p[i];
        if (i==p[i])maxv=max(maxv,p[i]-1);
    }
    return maxv;
}
```

12.lyndon分解

```
//给定一个字符串s(sum of s<=2e7),求1-n的前缀每个前缀的最小字典序后缀的位置k*1112^(i-1)，例k=1时,contest的后缀为
contest
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e6+10;
const int mod=1e9+7;
char s[N];
int p[N];//记录每个点的所属lyndon串的左端点
int main() {
    int T;
    cin>>T;
    while(T--){
        scanf("%s",s+1);
        int n=strlen(s+1);
        for(int i=1;i<=n;i++)p[i]=0;
        int i=1;
        while (i<=n){
            int j=i,k=i+1;
            p[i]=i;
            while(j<=n&&s[j]<=s[k]){
                if(s[j]<s[k])j=i,p[k]=i;
                else p[k]=p[j]+k-j,j++;
                k++;
            }
            while(i<=j)i+=k-j;
        }
        ll t=1,ans=0;
        for(int i=1;i<=n;i++)ans=(ans+t*p[i]%mod)%mod,t=t*1112%mod;
```

```
        printf("%lld\n",ans);
    }
    return 0;
}
```