

字符串 2

AC 自动机

算法简介

一种用于多模式串匹配的自动机，可以近似看成 Trie 树上的 KMP 算法。

算法例题

例题一

[洛谷p5357](#)

题意

给你一个文本串 S 和 n 个模式串 T_i 求每个模式串 T_i 在 S 中出现的次数。

题解

建立 AC 自动机后记录每个节点的访问次数，最后拓扑或建立 fail 树统计答案。时间复杂度 $O(|S| + \sum_{i=1}^n |T_i|)$

```
const int MAXN=2e6+5,MAXS=2e5+5;
int idx[MAXN];
struct AC{
    int ch[MAXS][26],val[MAXS],fail[MAXS],cnt[MAXS],deg[MAXS],sz,tot;
    int ans[MAXS];
    int insert(char *s){
        int len=strlen(s),pos=0;
        _for(i,0,len){
            int c=s[i]-'a';
            if(!ch[pos][c]){
                ch[pos][c]=++sz;
                val[sz]=fail[sz]=0;
                mem(ch[sz],0);
            }
            pos=ch[pos][c];
        }
        if(!val[pos])return val[pos]=++tot;
        else return val[pos];
    }
    void getFail(){
```



```

_rep(i,1,n){
    scanf("%s",buf);
    idx[i]=solver.insert(buf);
}
solver.getFail();
scanf("%s",buf);
solver.query(buf);
_rep(i,1,n)
enter(solver.ans[idx[i]]);
return 0;
}

```

例题二

洛谷p2444

题意

给定 n 个 01 串 S_i 问是否存在无限长串不含任何 S_i

题解

建立 AC 自动机标记每个 S_i 的终止节点，然后建 fail 树，显然所有终止节点在 fail 树中的子树节点均不能访问，于是将其标记。

最后 dfs 遍历树，在不经过所有标记节点的前提下判定是否存在可以到达的环。注意为保证时间复杂度 dfs 遍历后也需要标记节点。

总时间复杂度 $O(\sum_{i=1}^n |S_i|)$

```

const int MAXS=3e4+5;
struct AC{
    struct Edge{
        int to,next;
    }edge[MAXS<<1];
    int head[MAXS],edge_cnt;
    int ch[MAXS][2],val[MAXS],fail[MAXS],sz;
    void AddEdge(int u,int v){
        edge[++edge_cnt]=Edge{v,head[u]};
        head[u]=edge_cnt;
    }
    void insert(char *s){
        int len=strlen(s),pos=0;
        _for(i,0,len){
            int c=s[i]-'0';
            if(!ch[pos][c]){
                ch[pos][c]=++sz;
            }
        }
    }
}

```

```
        val[sz]=fail[sz]=0;
        mem(ch[sz],0);
    }
    pos=ch[pos][c];
}
val[pos]=1;
}
void getFail(){
    queue<int> q;
    _for(i,0,2){
        if(ch[0][i])
            q.push(ch[0][i]);
    }
    while(!q.empty()){
        int u=q.front();q.pop();
        _for(i,0,2){
            if(ch[u][i]){
                fail[ch[u][i]]=ch[fail[u]][i];
                q.push(ch[u][i]);
            }
            else ch[u][i]=ch[fail[u]][i];
        }
    }
    _rep(i,1,sz)
    AddEdge(fail[i],i);
}
void dfs(int u,int flag){
    val[u]=flag;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        dfs(v,flag|val[v]);
    }
}
bool dfs2(int u){
    val[u]=-1;
    _for(i,0,2){
        if(val[ch[u][i]]==-1)
            return true;
        else if(!val[ch[u][i]]&&dfs2(ch[u][i]))
            return true;
    }
    val[u]=1;
    return false;
}
bool query(){
    dfs(0,0);
    return dfs2(0);
}
}solver;
char buf[MAXS];
```

```
int main()
{
    int n=read_int();
    _for(i,0,n){
        scanf("%s",buf);
        solver.insert(buf);
    }
    solver.getFail();
    if(solver.query())
        puts("TAK");
    else
        puts("NIE");
    return 0;
}
```

例题三

洛谷p2414

题意

给定一个字符串 S 只包含小写字母和 B, P 两个大写字母。

当前初始串 T 为空串，扫描 S 如果 s_i 为小写字母，则在 T 末尾加入该字母。

如果 s_i 为 B 则删除 T 末尾的一个字母。如果 s_i 为 P 则打印 T

接下来 q 个询问，每次询问第 i 次打印的字符串在第 j 次打印的字符串中出现的次数。

题解

考虑建 fail 树，记第 i 次打印的字符串的结尾结点为 p_i 第 j 次打印的字符串的结尾结点为 p_j

于是该次询问的答案为 Trie 树中根节点到 p_j 的路径与 p_i 在 fail 树中的子树的交集的结点个数。

考虑将所有询问离线到 Trie 然后 dfs 遍历 Trie 树同时处理询问。

可以利用 fail 树上的 dfs 序以及树状数组维护子树，遍历过程中回溯维护链的性质。

时间复杂度 $O((|S|+q)\log |S|)$

```
const int MAXN=1e5+5;
int idx[MAXN],string_cnt;
struct Edge{
    int to,next;
```

```
}edge[MAXN];
int head[MAXN],edge_cnt,dfs_t,L[MAXN],R[MAXN];
void AddEdge(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
void dfs(int u){
    L[u]=++dfs_t;
    for(int i=head[u];i;i=edge[i].next)
        dfs(edge[i].to);
    R[u]=dfs_t;
}
#define lowbit(x) x&(-x)
struct BIT{
    int c[MAXN];
    void add(int pos,int v){
        while(pos<=dfs_t){
            c[pos]+=v;
            pos+=lowbit(pos);
        }
    }
    int query(int pos){
        int ans=0;
        while(pos){
            ans+=c[pos];
            pos-=lowbit(pos);
        }
        return ans;
    }
}tree;
int ans[MAXN];
vector<pair<int,int> >opt[MAXN];
struct AC{
    int ch[MAXN][26],ch2[MAXN][26],p[MAXN],val[MAXN],fail[MAXN],sz;
    void insert(char *s){
        int len=strlen(s),pos=0;
        _for(i,0,len){
            if(islower(s[i])){
                int c=s[i]-'a';
                if(!ch[pos][c]){
                    ch[pos][c]=ch2[pos][c]=++sz;
                    p[sz]=pos;
                }
                pos=ch[pos][c];
            }
            else if(s[i]=='B')
                pos=p[pos];
            else{
                val[pos]=1;
                idx[++string_cnt]=pos;
            }
        }
    }
};
```



```
}
```

例题四

洛谷p4052

题意

给定 n 个模式串 S_i 问有多少个长度为 m 的文本串至少包含一个模式串。(模式串、文本串只含大写字母)

题解

考虑计算出不含任何模式串的文本串，再用总数减去该值即可得到答案。

接下来标记所有模式串终止位置在 fail 树上的子节点，显然转移时不能到达标记结点。

最后设 $\text{dp}(i,j)$ 表示当前位于结点 i 且长度为 j 的方案数，暴力转移即可。总时空复杂度 $O(m \sum_{i=1}^n |S_i|)$

```
const int MAXL=105,MAXS=6005,Mod=1e4+7;
int quick_pow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)ans=ans*a%Mod;
        a=a*a%Mod;
        b>>=1;
    }
    return ans;
}
struct AC{
    int ch[MAXS][26],val[MAXS],fail[MAXS],sz;
    int dp[MAXS][MAXL];
    void insert(char *s){
        int len=strlen(s),pos=0;
        _for(i,0,len){
            int c=s[i]-'A';
            if(!ch[pos][c])
                ch[pos][c]=++sz;
            pos=ch[pos][c];
        }
        val[pos]=1;
    }
    void getFail(){
        queue<int> q;
        _for(i,0,26){
```

```

        if(ch[0][i])
            q.push(ch[0][i]);
    }
    while(!q.empty()){
        int u=q.front();q.pop();
        _for(i,0,26){
            if(ch[u][i]){
                fail[ch[u][i]]=ch[fail[u]][i];
                val[ch[u][i]]|=val[ch[fail[u]][i]];
                q.push(ch[u][i]);
            }
            else ch[u][i]=ch[fail[u]][i];
        }
    }
}
int query(int n){
    dp[0][0]=1;
    _for(i,0,n){
        _rep(j,0,sz){
            _for(k,0,26){
                if(val[ch[j][k]])continue;
                dp[ch[j][k]][i+1]+=dp[j][i];
                dp[ch[j][k]][i+1]%=Mod;
            }
        }
    }
    int ans=0;
    _rep(i,0,sz)ans=(ans+dp[i][n])%Mod;
    return ans;
}
}solver;
char buf[MAXL];
int main()
{
    int n=read_int(),m=read_int();
    _rep(i,1,n){
        scanf("%s",buf);
        solver.insert(buf);
    }
    solver.getFail();
    int del=solver.query(m);
    enter((quick_pow(26,m)-del+Mod)%Mod);
    return 0;
}

```

例题五

NC210775

题意

给定 n 个模式串，每个字符串有一个权值 v 。要求构造一个长度为 L 的字符串，使得其包含的模式串权值和最大。(包含多个相同模式串结果将累加)

题解

标记所有模式串终止位置在 fail 树上的子节点，子树可以累加父节点的权值。

最后设 $\text{dp}(i,j)$ 表示当前位于结点 i 且长度为 j 的最大答案，暴力转移。总时空复杂度 $O(\sum_{i=1}^n |S_i|)$

```
const int MAXL=1e3+5,MAXS=2005,Inf=1e9;
struct AC{
    int ch[MAXS][26],val[MAXS],fail[MAXS],sz;
    int dp[MAXS][MAXL];
    void insert(char *s,int v){
        int len=strlen(s),pos=0;
        _for(i,0,len){
            int c=s[i]-'a';
            if(!ch[pos][c])
                ch[pos][c]=++sz;
            pos=ch[pos][c];
        }
        val[pos]+=v;
    }
    void getFail(){
        queue<int> q;
        _for(i,0,26){
            if(ch[0][i])
                q.push(ch[0][i]);
        }
        while(!q.empty()){
            int u=q.front();q.pop();
            val[u]+=val[fail[u]];
            _for(i,0,26){
                if(ch[u][i]){
                    fail[ch[u][i]]=ch[fail[u]][i];
                    q.push(ch[u][i]);
                }
                else ch[u][i]=ch[fail[u]][i];
            }
        }
    }
    int query(int n){
        _rep(i,0,n)_rep(j,0,sz)dp[j][i]=-Inf;
        dp[0][0]=0;
        _for(i,0,n){
```

```

        _rep(j,0,sz){
            if(dp[j][i]==-Inf)continue;
            _for(k,0,26)
dp[ch[j][k]][i+1]=max(dp[ch[j][k]][i+1],dp[j][i]+val[ch[j][k]]);
        }
    }
    int ans=-Inf;
    _rep(i,0,sz)
    ans=max(ans,dp[i][n]);
    return ans;
}
}solver;
char buf[MAXS];
int main()
{
    int n,m;
    cin>>n>>m;
    _rep(i,1,n){
        scanf("%s",buf);
        int v;
        cin>>v;
        solver.insert(buf,v);
    }
    solver.getFail();
    enter(solver.query(m));
    return 0;
}

```

例题六

UVA11019

题意

给定一个 $n \times m$ 的二维文本串和一个 $x \times y$ 的二维模式串，问模式串在文本串中的出现次数。

题解

考虑将模式串按行拆分为 $t_1, t_2 \dots t_x$ 后插入 AC 自动机，同时进行编号(相同的字符串共用一个编号)。

再将 $t_1, t_2 \dots t_x$ 的编号相连得到一个一维数组 b

同时将文本串按行拆分为 $s_1, s_2 \dots s_n$ 后在 AC 自动机查询并记录匹配位置的模式串编号，得到一个二维数组 a

二维数组中元素 $a_{i,j}$ 的意义是 $s[i][j-y+1, j]$ 与模式串 $t_{a_{i,j}}$ 匹配。

于是对 a 的每列跑 KMP 算法记录 b 的出现数即可。时间复杂度 $O(nm+xy)$

注意上述方法中 \$AC\$ 自动机起到了降维的作用，且可以拓展到任意维字符串匹配。

```
const int MAXN=1e3+5,MAXS=1e4+5;
namespace KMP{
    int f[MAXN];
    int find(int *s1,int n,int *s2,int m){
        int pos=0,cnt=0;
        f[1]=0;
        _rep(i,2,m){
            while(pos&& s2[i]!=s2[pos+1])pos=f[pos];
            if(s2[i]==s2[pos+1])pos++;
            f[i]=pos;
        }
        pos=0;
        _rep(i,1,n){
            while(pos&& s1[i]!=s2[pos+1])pos=f[pos];
            if(s1[i]==s2[pos+1])pos++;
            if(pos==m){
                cnt++;
                pos=f[pos];
            }
        }
        return cnt;
    }
}
struct AC{
    int ch[MAXS][26],val[MAXS],fail[MAXS],sz,cnt;
    void clear(){
        sz=cnt=0;
        mem(ch[0],0);
    }
    int insert(char *s){
        int len=strlen(s),pos=0;
        _for(i,0,len){
            int c=s[i]-'a';
            if(!ch[pos][c]){
                ch[pos][c]=++sz;
                mem(ch[sz],0);
                val[sz]=fail[sz]=0;
            }
            pos=ch[pos][c];
        }
        if(val[pos])return val[pos];
        return val[pos]=++cnt;
    }
    void getFail(){
        queue<int> q;
        _for(i,0,26){
            if(ch[0][i])
```

```

        q.push(ch[0][i]);
    }
    while(!q.empty()){
        int u=q.front();q.pop();
        val[u]+=val[fail[u]];
        _for(i,0,26){
            if(ch[u][i]){
                fail[ch[u][i]]=ch[fail[u]][i];
                q.push(ch[u][i]);
            }
            else ch[u][i]=ch[fail[u]][i];
        }
    }
}
void query(char *s,int *ans,int n){
    int pos=0;
    _for(i,0,n){
        pos=ch[pos][s[i]-'a'];
        ans[i]=val[pos];
    }
}
}solver;
char s1[MAXN][MAXN],s2[MAXN];
int a[MAXN][MAXN],b[MAXN],c[MAXN];
int main()
{
    int T=read_int();
    while(T--){
        solver.clear();
        int n=read_int(),m=read_int(),ans=0;
        _rep(i,1,n)scanf("%s",s1[i]);
        int x=read_int(),y=read_int();
        _rep(i,1,x){
            scanf("%s",s2);
            b[i]=solver.insert(s2);
        }
        solver.getFail();
        _rep(i,1,n)
        solver.query(s1[i],a[i],m);
        _for(i,y-1,m){
            _rep(j,1,n)c[j]=a[j][i];
            ans+=KMP::find(c,n,b,x);
        }
        enter(ans);
    }
    return 0;
}

```

