11402 Ahoy, Pirates!

In the ancient pirate ages, the Pirate Land was divided into two teams of pirates, namely, the Buccaneer and the Barbary pirates. Each pirate's team was not fixed, sometimes the opponent pirates attacked and he was taken away to the other pirate team. All on a sudden a magician appeared in the Pirate Land, where he was making transition of pirates from their team to other team at his own will. Of course, handy spells were used. The process of changing team was known as mutating.

There were N pirates and all of the pirates have a unique id from 0 to $N-1$. The great magician could mutate a bunch of pirates with consecutive id's to another one.

Suppose there were 100 pirates in the pirate land and all of them were Barbary pirates, then the magician could cast a spell to change pirates with id's from 10 to 33 to Buccaneer pirates. Then the whole pirate land would have 24 Buccaneer and 76 Barbary pirates.

The magician was very fast casting the spell. Once, God started to dislike this. God had favor for the Buccaneer pirates and God asked the magician, "Tell me, how many of the pirates of index from 2 to 30 are Buccaneer pirates?". Now the magician was puzzled as he was only efficient in casting spells, not in counting :-)

Being clever enough, the magician captured a clever man from the Earth Land. And unfortunately its you! Now you have to answer the Gods questions.

Input

The first line of input will contain number of test cases T.

For each test case:

The first part of the description will be of the pirate land. There could be up to N ($1 \leq N \leq 1024000$) pirates. Each pirate is either assigned to Buccaneer or Barbary Pirate. Buccaneer pirates are described by '1' (ONE) and Barbary pirates are described by '0' (ZERO). You have to build a string of the pirates description. Each case starts with an integer M ($M \leq 100$), where M pair lines follow. In each pair of lines (we call it a set), first has an integer T ($T \leq 200$) and next one has a nonempty string **Pirates** (consisting of 0 and 1, 0 for Barbary, 1 for Buccaneer, has maximum length of 50). For each pair concatenate the string **Pirates**, T times. Concatenate all the resulting M sets of strings to build the pirate description. The final concatenated string describes the pirates from index 0 to end ($N-1$ for N pirates).

Now the next part of the input will contain queries. First line of next part has an integer Q describing number of queries. Each subsequence Q ($1 \leq Q \leq 1000$) lines describe each query. Each query has a string `F` or `E` or `I` or `S` and two integers, a and b denoting indexes. The meaning of the query string are follows:

`F` a b, means, mutate the pirates from index a to b to Buccaneer Pirates.

`E` a b, means, mutate the pirates from index a to b to Barbary Pirates.

`I` a b, means, mutate the pirates from index a to b to inverse pirates.

`S` a b, means, "God's query" God is asking a question: "Tell me, how many Buccaneer pirates are there from index a to b?"

($a \leq b$, $0 \leq a < n$, $0 \leq b < n$, index range are inclusive)

Output

For each test print the case number as the sample output suggests. Then for each of God's query, output the query number, colon (`:`) and a space and the answer to the query as the sample suggest.

Explanation:

Case1:

The pirate land is as follows ($N = 18$)
101010101010001000
Before God's first query it was as follows
000000111111111111

Case 2:

The pirate land is as follows ($N = 9$)
111000000

Sample Input

```
2
2
5
10
2
1000
5
F 0 17
I 0 5
S 1 10
E 4 9
S 2 10
3
3
1
4
0
2
0
2
I 0 2
S 0 8
```

Sample Output

```
Case 1:
Q1: 5
Q2: 1
Case 2:
Q1: 0
```

```cpp
#include<bits/stdc++.h>
using namespace std;
#define MAXN 1024005
string s;
struct data{
    int one, lazy;
}tree[4*MAXN];
void relaxation(int nd,int b,int e){
    if(tree[nd].lazy==2){
        tree[nd].one = (e-b+1) - tree[nd].one;
    }else{
        tree[nd].one = (e-b+1) * tree[nd].lazy;
    }
}
void pushDown(int nd, int b,int e){
    if(tree[nd].lazy!=-1) {
        relaxation(nd,b,e);
        if(b!=e){
            int l=2*nd, r=2*nd+1, m=(b+e)/2;
            if(tree[nd].lazy==2) {
                if(tree[l].lazy == 0) tree[l].lazy = 1;
                else if(tree[l].lazy == 1) tree[l].lazy = 0;
                else if(tree[l].lazy == 2) tree[l].lazy =-1;
                else if(tree[l].lazy ==-1) tree[l].lazy = 2;

                if(tree[r].lazy == 0) tree[r].lazy = 1;
                else if(tree[r].lazy == 1) tree[r].lazy = 0;
                else if(tree[r].lazy == 2) tree[r].lazy =-1;
                else if(tree[r].lazy ==-1) tree[r].lazy = 2;
            }else {
                tree[l].lazy = tree[nd].lazy;
                tree[r].lazy = tree[nd].lazy;
            }
        }
        tree[nd].lazy = -1;
    }
}
void build(int nd, int b, int e){
    if(b==e){
        tree[nd].one = s[b]-'0';
        tree[nd].lazy = -1;
        return;
    }
    int l=2*nd, r=2*nd+1, m = (b+e)/2;
    build(l,b,m);
    build(r,m+1,e);
    tree[nd].one  = tree[l].one  + tree[r].one;
    tree[nd].lazy = -1;
}
int fun(int nd, int b, int e){
    if(tree[nd].lazy != -1){
        if(tree[nd].lazy == 0) return 0;
        else if(tree[nd].lazy == 1) return (e-b+1);
        else if(tree[nd].lazy == 2) return (e-b+1) - tree[nd].one;
    }else{
        return tree[nd].one;
    }
}
void update(int nd,int b,int e,int x,int y,int c){
    pushDown(nd,b,e);
    int l=2*nd, r=2*nd+1, m = (b+e)/2;
    if(b==x && e==y){
        if(c==0) tree[nd].lazy = 0;
        else if(c==1) tree[nd].lazy = 1;
        else if(c==2) tree[nd].lazy = 2;
        pushDown(nd,b,e);
        return;
    }
    if(y<=m) update(l,b,m,x,y,c);
    else if(x>m)update(r,m+1,e,x,y,c);
    else {
        update(l,b,m,x,m,c);
        update(r,m+1,e,m+1,y,c);
    }
    tree[nd].one  = fun(l,b,m)  + fun(r,m+1,e);
}
```

```cpp
int query(int nd,int b,int e,int x,int y){
    pushDown(nd,b,e);
    if(b==x && e==y)return tree[nd].one;
    int l=2*nd, r=2*nd+1, m = (b+e)/2;
    if(y<=m) return query(l,b,m,x,y);
    else if(x>m) return query(r,m+1,e,x,y);
    else return query(l,b,m,x,m) + query(r,m+1,e,m+1,y);
}
int main(){
    ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    int tt; cin>>tt;
    for(int ks=1; ks<=tt; ks++){
        cout<<"Case "<<ks<<":"<<endl;
        s="";
        int m; cin>>m;
        while(m--){
            int t; cin>>t;
            string h; cin>>h;
            while(t--) s += h;
        }
        int n = s.size();

        build(1,0,n-1);

        int q,k=0; cin>>q;
        while(q--){
            char c; int x,y;
            cin>>c>>x>>y;
            if(c=='F'){
                update(1,0,n-1,x,y,1);
            }else if(c=='E'){
                update(1,0,n-1,x,y,0);
            }else if(c=='I'){
                update(1,0,n-1,x,y,2);
            }else{
                int ans = query(1,0,n-1,x,y);
                cout<<"Q"<<++k<<": "<<ans<<endl;
            }
        }
    }
    return 0;
}
/*
2
2
5
10
2
1000
5
F 0 17
I 0 5
S 1 10
E 4 9
S 2 10
3
3
1
4
0
2
0
2
I 0 2
S 0 8
*/
```

```cpp
#include<bits/stdc++.h>
using namespace std;
#define MAXN 1024005
string s;
struct data{
    int one, lazy;
}tree[4*MAXN];
void relaxation(int nd,int b,int e){
    if(tree[nd].lazy==2){
        tree[nd].one = (e-b+1) - tree[nd].one;
    }else{
        tree[nd].one = (e-b+1) * tree[nd].lazy;
    }
}
void pushDown(int nd, int b,int e){
    if(tree[nd].lazy!=-1) {
        relaxation(nd,b,e);
        if(b!=e){
            int l=2*nd, r=2*nd+1, m=(b+e)/2;
            if(tree[nd].lazy==2) {
                if(tree[l].lazy == 0) tree[l].lazy = 1;
                else if(tree[l].lazy == 1) tree[l].lazy = 0;
                else if(tree[l].lazy == 2) tree[l].lazy =-1;
                else if(tree[l].lazy ==-1) tree[l].lazy = 2;

                if(tree[r].lazy == 0) tree[r].lazy = 1;
                else if(tree[r].lazy == 1) tree[r].lazy = 0;
                else if(tree[r].lazy == 2) tree[r].lazy =-1;
                else if(tree[r].lazy ==-1) tree[r].lazy = 2;
            }else {
                tree[l].lazy = tree[nd].lazy;
                tree[r].lazy = tree[nd].lazy;
            }
        }
        tree[nd].lazy = -1;
    }
}
void build(int nd, int b, int e){
    if(b==e){
        tree[nd].one = s[b]-'0';
        tree[nd].lazy = -1;
        return;
    }
    int l=2*nd, r=2*nd+1, m = (b+e)/2;
    build(l,b,m);
    build(r,m+1,e);
    tree[nd].one  = tree[l].one  + tree[r].one;
    tree[nd].lazy = -1;
}
void update(int nd,int b,int e,int x,int y,int c){
    pushDown(nd,b,e);
    int l=2*nd, r=2*nd+1, m = (b+e)/2;
    if(b>y || e<x) return;
    if(b>=x && e<=y){
        if(c==2) tree[nd].one = (e-b+1) - tree[nd].one;
        else tree[nd].one = (e-b+1) * c;

        if(c==2) {
            if(tree[l].lazy == 0) tree[l].lazy = 1;
            else if(tree[l].lazy == 1) tree[l].lazy = 0;
            else if(tree[l].lazy == 2) tree[l].lazy =-1;
            else if(tree[l].lazy ==-1) tree[l].lazy = 2;

            if(tree[r].lazy == 0) tree[r].lazy = 1;
            else if(tree[r].lazy == 1) tree[r].lazy = 0;
            else if(tree[r].lazy == 2) tree[r].lazy =-1;
            else if(tree[r].lazy ==-1) tree[r].lazy = 2;
        }else {
            tree[l].lazy = c;
            tree[r].lazy = c;
        }
        tree[nd].lazy = -1;
        return;
    }

    update(l,b,m,x,y,c);
    update(r,m+1,e,x,y,c);
    tree[nd].one  = tree[l].one  + tree[r].one;
}
```

```cpp
int query(int nd,int b,int e,int x,int y){
    pushDown(nd,b,e);
    if(b>y || e<x) return 0;
    if(b>=x && e<=y)return tree[nd].one;
    int l=2*nd, r=2*nd+1, m = (b+e)/2;
    return query(l,b,m,x,y) + query(r,m+1,e,x,y);
}
int main(){
    ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    int tt; cin>>tt;
    for(int ks=1; ks<=tt; ks++){
        cout<<"Case "<<ks<<":"<<endl;
        s="";
        int m; cin>>m;
        while(m--){
            int t; cin>>t;
            string h; cin>>h;
            while(t--) s += h;
        }
        int n = s.size();

        build(1,0,n-1);

        int q,k=0; cin>>q;
        while(q--){
            char c; int x,y;
            cin>>c>>x>>y;
            if(c=='F'){
                update(1,0,n-1,x,y,1);
            }else if(c=='E'){
                update(1,0,n-1,x,y,0);
            }else if(c=='I'){
                update(1,0,n-1,x,y,2);
            }else{
                int ans = query(1,0,n-1,x,y);
                cout<<"Q"<<++k<<": "<<ans<<endl;
            }
        }
    }
    return 0;
}
/*
2
2
5
10
2
1000
5
F 0 17
I 0 5
S 1 10
E 4 9
S 2 10
3
3
1
4
0
2
0
2
I 0 2
S 0 8
*/
```