/\*\*  
\*\*You are given an array of **N** elements, which are initially all 0. After \*\*that you will be given **C** commands. They are –  
\*\***0 p q v** - you have to add **v** to all numbers in the range \*\*of **p** to **q** (inclusive), where **p** and **q** are two indexes of the array.  
\*\***1 p q** - output a line containing a single integer which is the sum of all \*\*the array elements between **p** and **q**(inclusive)  
\*/

LLD tree[3\*100005];

LLD lazy[3\*100005];

void update(int left,int right,int index,int x,int y,int value)

{

if(x<=left&&y>=right)

{

tree[index]+=(LLD)(right-left+1)\*value;

lazy[index]+=value;

return;

}

int mid=(left+right)/2;

if(lazy[index]!=0)

{

tree[2\*index]+=(LLD)(mid-left+1)\*lazy[index];

tree[2\*index+1]+=(LLD)(right-mid)\*lazy[index];

lazy[2\*index]+=lazy[index];

lazy[2\*index+1]+=lazy[index];

lazy[index]=0;

}

if(x<=mid)

{

update(left,mid,2\*index,x,y,value);

}

if(y>mid)

{

update(mid+1,right,2\*index+1,x,y,value);

}

tree[index]=tree[2\*index]+tree[2\*index+1];

}

LLD query(int left,int right,int index,int x,int y)

{

LLD a1=0,a2=0;

if(x<=left&&y>=right)

{

return tree[index];

}

int mid=(left+right)/2;

if(lazy[index]!=0)

{

tree[2\*index]+=(LLD)(mid-left+1)\*lazy[index];

tree[2\*index+1]+=(LLD)(right-mid)\*lazy[index];

lazy[2\*index]+=lazy[index];

lazy[2\*index+1]+=lazy[index];

lazy[index]=0;

}

if(x<=mid)

{

a1=query(left,mid,2\*index,x,y);

}

if(y>mid)

{

a2=query(mid+1,right,2\*index+1,x,y);

}

return (a1+a2);

}

int main()

{

int test,t;

scanf("%d",&test);

for(t=1;t<=test;t++)

{

mem(tree,0);

mem(lazy,0);

int s,q;

scanf("%d %d",&s,&q);

while(q--)

{

int x,y,v,dec;

scanf("%d",&dec);

if(dec)

{

scanf("%d %d",&x,&y);

LLD ans=query(0,s-1,1,x-1,y-1);

printf("%lld\n",ans);

}

else

{

scanf("%d %d %d",&x,&y,&v);

update(0,s-1,1,x-1,y-1,v);

}

}

}

return 0;

}

/\*  
\*\*You have an array with **n** elements which is indexed from **0** to **n - 1**. \*\*Initially all elements are zero. Now you have to deal with two types of \*\*operations

\*\*1.Increase the numbers between indices **i** and **j** (inclusive) by **1**. This \*\*is \*\*represented by the command **'0 i j'**.

\*\*2.Answer how many numbers between indices **i** and **j** (inclusive) are \*\*divisible by **3**. This is represented by the command **'1 i j'**.

\*/

#define Max 100010

int Tree[8\*Max][4];

int lazy[8\*Max];

int temp[4];

void build(int left,int right,int indx)

{

if(left==right)

{

Tree[indx][0]=1;

Tree[indx][1]=Tree[indx][2]=lazy[indx]=0;

return;

}

int mid=(left+right)/2;

build(left,mid,2\*indx);

build(mid+1,right,2\*indx+1);

for(int i=0;i<3;i++)

{

Tree[indx][i]=Tree[2\*indx][i]+Tree[2\*indx+1][i];

}

}

void update(int left,int right,int indx,int x,int y,int add)

{

if(lazy[indx])

{

int lazy\_val=lazy[indx];

lazy[2\*indx]=(lazy[2\*indx]+lazy\_val)%3;

lazy[2\*indx+1]=(lazy[2\*indx+1]+lazy\_val)%3;

for(int i=0;i<3;i++)temp[(lazy\_val+i)%3]=Tree[indx][i];

for(int i=0;i<3;i++)Tree[indx][i]=temp[i];

lazy[indx]=0;

}

if(left>y||right<x)return;

if(x<=left&&right<=y)

{

for(int i=0;i<3;i++)

{

temp[(i+add)%3]=Tree[indx][i];

}

for(int i=0;i<3;i++)Tree[indx][i]=temp[i];

lazy[2\*indx]=(lazy[2\*indx]+add)%3;

lazy[2\*indx+1]=(lazy[2\*indx+1]+add)%3;

return;

}

int mid=(left+right)/2;

update(left,mid,2\*indx,x,y,add);

update(mid+1,right,2\*indx+1,x,y,add);

for(int i=0;i<3;i++)

{

Tree[indx][i]=Tree[2\*indx][i]+Tree[2\*indx+1][i];

}

}

int query(int left,int right,int indx,int x,int y)

{

if(lazy[indx])

{

int lazy\_val=lazy[indx];

lazy[2\*indx]=(lazy[2\*indx]+lazy\_val)%3;

lazy[2\*indx+1]=(lazy[2\*indx+1]+lazy\_val)%3;

for(int i=0;i<3;i++)temp[(lazy\_val+i)%3]=Tree[indx][i];

for(int i=0;i<3;i++)Tree[indx][i]=temp[i];

lazy[indx]=0;

}

if(left>y||right<x)return 0;

if(x<=left&&right<=y)return Tree[indx][0];

int mid=(left+right)/2;

return query(left,mid,2\*indx,x,y)+query(mid+1,right,2\*indx+1,x,y);

}

int main()

{

int x,y;

int test=buffer\_input();

for(int t=1;t<=test;t++)

{

mem(lazy,0);

int N=buffer\_input();

int Q=buffer\_input();

build(0,N-1,1);

printf("Case %d:\n",t);

for(int i=0;i<Q;i++)

{

int d=buffer\_input();

x=buffer\_input();

y=buffer\_input();

if(d==0)

{

update(0,N-1,1,x,y,1);

}

else printf("%d\n",query(0,N-1,1,x,y));

}

}

return 0;

}

/\*\*

\*\*Give a array Of N numbers. Finding Maximum cumulative number frequency in \*\*the range.  
\*\*input:

\*\*10 4

\*\*1 1 1 3 3 3 3 2 2 2

\*\*1 5

\*\*1 6

\*\*1 7

\*\*Output:

\*\*3

\*\*3

\*\*4

\*\*2

\*/

struct info

{

int Lcnt,Rcnt,Max,Lnum,Rnum;

info(int Lcnt=0,int Rcnt=0,int Max=0,int Lnum=0,int Rnum=0):Lcnt(Lcnt),Rcnt(Rcnt),Max(Max),Lnum(Lnum),Rnum(Rnum){};

};

info Tree[3\*MAX];

int arr[MAX];

info marge(const info &L,const info &R)

{

info ret;

if(L.Rnum==R.Lnum)

{

ret.Max=max(L.Rcnt+R.Lcnt,max(L.Max,R.Max));

}

else ret.Max=max(L.Max,R.Max);

ret.Lnum=L.Lnum;

ret.Rnum=R.Rnum;

if(L.Lnum==R.Lnum)ret.Lcnt=L.Lcnt+R.Lcnt;

else ret.Lcnt=L.Lcnt;

if(L.Rnum==R.Rnum)ret.Rcnt=L.Rcnt+R.Rcnt;

else ret.Rcnt=R.Rcnt;

return ret;

}

void build(int L,int R,int indx)

{

if(L==R)

{

Tree[indx]=info(1,1,1,arr[L],arr[R]);

return;

}

int mid=(L+R)>>1;

build(L,mid,2\*indx);

build(mid+1,R,2\*indx+1);

Tree[indx]=marge(Tree[2\*indx],Tree[2\*indx+1]);

}

info query(int L,int R,int indx,int x,int y)

{

if(L>=x&&R<=y)return Tree[indx];

int mid=(L+R)>>1;

info c1,c2;

if(x<=mid)c1=query(L,mid,2\*indx,x,y);

if(y>mid)c2=query(mid+1,R,2\*indx+1,x,y);

return marge(c1,c2);

}

int main()

{

int test;

scanf("%d",&test);

for(int t=1;t<=test;t++)

{

int N,C,Q;

scanf("%d %d %d",&N,&C,&Q);

for(int i=0;i<N;i++)

{

int x;

scanf("%d",&arr[i+1]);

}

build(1,N,1);

printf("Case %d:\n",t);

while(Q--)

{

int x,y;

scanf("%d %d",&x,&y);

printf("%d\n",query(1,N,1,x,y).Max);

}

}

return 0;

}

/\*\*

\*\*Given a bracket sequence.

\*\* On a bracket word one can do the following operations:  
\*\***replacement** -- changes the i-th bracket into the opposite one  
\*\***check** -- if the word is a correct bracket expression  
\*\*/

struct info

{

int sum,sub;

info(int sum=0,int sub=0):sum(sum),sub(sub){};

};

info Tree[4\*MAX];

char inp[MAX];

info marge(const info &L,const info &R)

{

info ret;

ret.sum= L.sum+R.sum;

ret.sub=L.sub;

ret.sub=min(ret.sub,L.sum+R.sub);

return ret;

}

void build(int L,int R,int indx)

{

if(L==R)

{

int x;

if(inp[L]=='(')x=1;

else x=-1;

Tree[indx]=info(x,x);

return;

}

int mid=(L+R)>>1;

build(L,mid,2\*indx);

build(mid+1,R,2\*indx+1);

Tree[indx]=marge(Tree[2\*indx],Tree[2\*indx+1]);

}

void update(int L,int R,int indx,int x)

{

if(L==R)

{

int x;

if(inp[L]=='(')x=1;

else x=-1;

Tree[indx]=info(x,x);

return;

}

int mid=(L+R)>>1;

if(x<=mid)update(L,mid,2\*indx,x);

else update(mid+1,R,2\*indx+1,x);

Tree[indx]=marge(Tree[2\*indx],Tree[2\*indx+1]);

}

info query(int L,int R,int indx,int x,int y)

{

if(L==x&R==y)return Tree[indx];

int mid=(L+R)>>1;

if(y<=mid)return query(L,mid,2\*indx,x,y);

else if(x>mid)return query(mid+1,R,2\*indx+1,x,y);

else return marge(query(L,mid,2\*indx,x,mid),query(mid+1,R,2\*indx+1,mid+1,y));

}

int main()

{

int N,t=1;

while(scanf("%d",&N)==1)

{

scanf("%s",inp);

build(0,N-1,1);

int Q;

printf("Test %d:\n",t++);

scanf("%d",&Q);

while(Q--)

{

int x;

scanf("%d",&x);

if(x)

{

if(inp[x-1]=='(')inp[x-1]=')';

else inp[x-1]='(';

update(0,N-1,1,x-1);

}

else

{

info y=query(0,N-1,1,0,N-1);

if(y.sum==0&&y.sub>=0)printf("YES\n");

else printf("NO\n");

}

}

}

return 0;

}