Algorithm Code Book

Tanvir Hasan Anick

July 16, 2015

Contents

Chapter 1

Data Structure

1.1 Trie

1.1.1 Static Trie

```
#define Max 10005
int getId(char c)
{
    return c>='a'?c-'a':c-'A'+26;
}
struct Trie
{
    struct Tree
    {
        int Next[52];
        bool word;
        void clear()
        {
            word=false;
            memset(Next,-1,sizeof(Next));
        }
    }T[Max];
    int ptr;
    void clear()
    {
        ptr=1;
        T[0].clear();
        memset(T[0].Next,0,sizeof(T[0].Next));
    }
    void Insert(const char *str)
    {
        int p=0;
        for(int i=0;str[i];i++)
        {
            int id=getId(str[i]);
            if(T[p].Next[id]<=0)
            {
                T[p].Next[id]=ptr;
                T[ptr++].clear();
            }
            p=T[p].Next[id];
        }
        T[p].word=true;
    }
    bool Search(const char *str)
    {
        int p=0;
        for(int i=0;str[i];i++)
        {
            int id=getId(str[i]);
            if(T[p].Next[id]>0)
            {
                p=T[p].Next[id];
            }
            else return false;
        }
        return T[p].word;
    }
};
Trie A;
```

1.2 RMQ

1.2.1 Bit

1D Bit

```
#define MaxVal 100000
int Bit[MaxVal];
/**find sum from 1 to idx**/
int read(int idx)
{
    int sum = 0;
    while (idx > 0)
    {
        sum += Bit[idx];
        idx -= (idx & -idx);
    }
    return sum;
}
/**update value ind to MaxVal**/
void update(int idx ,int val)
{
    while (idx <= MaxVal)
    {
        Bit[idx] += val;
        idx += (idx & -idx);
    }
}

/**Find the value of idx**/
int readSingle(int idx)
{
    int sum = Bit[idx]; /// sum will be decreased
    if (idx > 0)  /// special case
    {
        int z = idx - (idx & -idx); /// make z first
        idx--; /// idx is no important any more, so instead y, you can use idx
        while (idx != z)  /// at some iteration idx (y) will become z
        {
            sum -= Bit[idx];/// substruct Bit frequency which is between y and ”the same path”
            idx -= (idx & -idx);
        }
    }
    return sum;
}
```

2D Bit

```
void updatey(int x , int y , int val)
{
    while (y <= max_y)
    {
        tree[x][y] += val;
        y += (y & -y);
    }
}
void update(int x , int y , int val)
{
    while (x <= max_x)
    {
        updatey(x , y , val);// this function should update array tree[x]
        x += (x & -x);
    }
}
```

1.2.2 Square Root Decompostion

```
#include<bits/stdc++.h>
using namespace std;
const int sz=100005;
const int inf=(1<<28);
template<typename t> t MIN3(t a,t b, t c)
{
    return min(a,min(b,c));
}
int BLOCK[400];
int arr[sz];
int getId(int indx,int blockSZ)
{
    return indx/blockSZ;
}
void init(int sz)
{
    for(int i=0; i<=sz; i++)BLOCK[i]=inf;
}
void update(int val,int indx,int blockSZ)
{
    int id=getId(indx,blockSZ);
    BLOCK[id]=min(BLOCK[id],val);
}
int query(int L,int R,int blockSZ)
{
    int lid=getId(L,blockSZ);
    int rid=getId(R,blockSZ);
    if(lid==rid)
    {
        int ret=inf;
        for(int i=L; i<=R; i++)ret=min(ret,arr[i]);
        return ret;
    }
    int m1=inf,m2=inf,m3=inf;
    for(int i=L; i<(lid+1)*blockSZ; i++)m1=min(m1,arr[i]);
    for(int i=lid+1; i<rid; i++)m2=min(m2,BLOCK[i]);
    for(int i=rid*blockSZ; i<=R; i++)m3=min(m3,arr[i]);
    return MIN3(m1,m2,m3);
}
int main()
{
    int N,Q;
    scanf("%d %d",&N,&Q);
    int blockSZ=sqrt(N);
    init(blockSZ);
    for(int i=0; i<N; i++)
    {
        int x;
        scanf("%d",&x);
        arr[i]=x;
        update(x,i,blockSZ);
    }
    while(Q--)
    {
        int x,y;
        scanf("%d %d",&x,&y);
        printf("%d\n",query(x,y,blockSZ));
    }
    return 0;
}
```

1.2.3 MO's Algorithm

```cpp
/**
    MO's Algorithm
    problem: http://www.spoj.com/problems/DQUERY

    MOs algorithm is just an order in which we process the queries.
    We were given M queries, we will re-order the queries in a particular order and then process them.
    Clearly, this is an off-line algorithm. Each query has L and R, we will call them opening and closing.
    Let us divide the given input array into Sqrt(N) blocks.
    Each block will be N / Sqrt(N) = Sqrt(N) size.
    Each opening has to fall in one of these blocks.
    Each closing has to fall in one of these blocks.

    All the queries are first ordered in ascending order of their block number (block number is the block in which its opening falls).
    Ties are ordered in ascending order of their R value.

**/
#include<bits/stdc++.h>
using namespace std;
#define Mx 30005
#define MxNum 1000005
int BlockSize;
int Answer;
int Freq[MxNum],Num[Mx];
struct info
{
    int L,R,qno;
    info(int L=0,int R=0,int qno=0):L(L),R(R),qno(qno){};
    bool operator<(const info &a)const
    {
        if(L/BlockSize!=a.L/BlockSize)return L/BlockSize<a.L/BlockSize;
        return R<a.R;
    }
}Query[200005];
int StoreAnswer[200005];
void Add(int indx)
{
    Freq[Num[indx]]++;
    if(Freq[Num[indx]]==1)Answer++;
}
void Remove(int indx)
{
    Freq[Num[indx]]--;
    if(Freq[Num[indx]]==0)Answer--;
}
int main()
{
    int N;
    scanf("%d",&N);
    BlockSize=sqrt(N);
    for(int i=0;i<N;i++)
    {
        scanf("%d",&Num[i]);
    }
    int Q;
    scanf("%d",&Q);
    for(int i=0;i<Q;i++)
    {
        int x,y;
        scanf("%d %d",&x,&y);
        Query[i]=info(x-1,y-1,i);
    }
    sort(Query,Query+Q);
    int currentL=0,currentR=0;
    Answer=0;
    for(int i=0;i<Q;i++)
    {
        int L=Query[i].L;
        int R=Query[i].R;
        while(currentL<L)
        {
            Remove(currentL);
            currentL++;
        }
        while(currentL>L)
        {
            Add(currentL-1);
            currentL--;
        }
        while(currentR<=R)
        {
            Add(currentR);
            currentR++;
        }
        while(currentR>R+1)
        {
            Remove(currentR-1);
            currentR--;
        }
        StoreAnswer[Query[i].qno]=Answer;
    }
    for(int i=0;i<Q;i++)
    {
        printf(”%d\n”,StoreAnswer[i]);
    }
    return 0;
}
```

1.2.4 Segment Tree

Lazy Propagration1

```
/**
**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)
*/
#include<bits/stdc++.h>
using namespace std;
typedef long long LLD;
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++)
    {
        memset(tree,0,sizeof(tree));
        memset(lazy,0,sizeof*lazy);
        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;
}
```

Lazy Propagration2

```
/*
**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'.
*/
#include<bits/stdc++.h>
```

```
using namespace std;
#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;
    scanf("%d",&test);
    for(int t=1;t<=test;t++)
    {
        memset(lazy,0,sizeof(lazy));
        int N,Q;
        scanf("%d %d",&N,&Q);
        build(0,N-1,1);
        printf("Case %d:\n",t);
        for(int i=0;i<Q;i++)
        {
            int d;
            scanf("%d %d %d",&d,&x,&y);
            if(d==0)
            {
                update(0,N-1,1,x,y,1);
            }
            else printf("%d\n",query(0,N-1,1,x,y));
        }
    }
    return 0;
}
```

Segment Tree Variant 1

```
/**
**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
*/
#include<bits/stdc++.h>
using namespace std;
typedef long long LLD;
#define MAX 50005
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;
}
```

Segment Tree Variant 2

```
/**
**You are given a sequence A of N (N <= 50000) integers between -10000 and 10000.
**On this sequence you have to apply M (M <= 50000) operations:
**modify the i-th element in the sequence or for given x y print max{Ai + Ai+1 + ..
    + Aj | x<=i<=j<=y }.
**/
#include<bits/stdc++.h>
using namespace std;
typedef long long LLD;
template<class T> T MAX3(T a,T b,T c) {return max(a,max(b,c));}
LLD Inf=(1ll<<60);
#define MN 50005
struct info
{
    LLD prefixSum;
    LLD suffixSum;
    LLD Total;
    LLD TotalMax;
    info(int pre=-Inf,int suff=-Inf,int total=-Inf,int totalmax=-Inf):prefixSum(pre
    ),suffixSum(suff),Total(total),TotalMax(totalmax){};
};
info marge(const info &a,const info &b)
{
    info ret;
    ret.Total=a.Total+b.Total;
    ret.prefixSum=max(a.prefixSum,a.Total+b.prefixSum);
    ret.suffixSum=max(a.suffixSum+b.Total,b.suffixSum);
    ret.TotalMax=MAX3(a.TotalMax,b.TotalMax,a.suffixSum+b.prefixSum);
    return ret;
}
LLD arr[MN];
info Tree[3*MN];
void build(int L,int R,int indx)
{
    if(L==R)
    {
        Tree[indx]=info(arr[L],arr[L],arr[L],arr[L]);
        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,LLD val)
{
    if(L==R)
    {
        Tree[indx]=info(val,val,val,val);
        return;
    }
    int mid=(L+R)>>1;
    if(x<=mid)update(L,mid,2*indx,x,val);
    else update(mid+1,R,2*indx+1,x,val);
    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 and y==R)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);
    return marge(query(L,mid,2*indx,x,mid),query(mid+1,R,2*indx+1,mid+1,y));
}
int main()
{
    #ifdef _ANICK_
    //f_input;
    #endif // _ANICK_
    int N;
    scanf("%d",&N);
    for(int i=1;i<=N;i++)scanf("%lld",&arr[i]);
    build(1,N,1);
    int Q;
    scanf("%d",&Q);
    while(Q--)
    {
        int t,x,y;
        scanf("%d %d %d",&t,&x,&y);
        if(t)printf("%lld\n",query(1,N,1,x,y).TotalMax);
        else update(1,N,1,x,y);
    }
    return 0;
}
```

Segment Tree Variant 3

```
/**
**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
**/
#include<bits/stdc++.h>
using namespace std;
typedef long long LLD;
```

```
#define MAX 50005
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;
}
```

1.2.5 Sliding Window RMQ

```
/**
    every K size window RMQ
    Calculate in O(N+K) time
**/
#include<bits/stdc++.h>
using namespace std;
vector<int>SlidingRMQ(int *A,int N,int k)
{
    /** Create a Double Ended Queue, Qi that will store indexes of array elements
        The queue will store indexes of useful elements in every window and it will
        maintain decreasing order of values from front to rear in Qi, i.e.,
        arr[Qi.front[]] to arr[Qi.rear()] are sorted in increasing order
    **/
    vector<int>MinWindow;
    deque<int>Q;
    int i;
    /* Process first k (or first window) elements of array */
    for(i=0;i<k;i++)
    {
        /// For very element, the previous largest elements are useless so
        /// remove them from Qi
        while(!Q.empty() and A[i]<=A[Q.back()])Q.pop_back();
        Q.push_back(i);
    }
    /// Process rest of the elements, i.e., from arr[k] to arr[n-1]
    while(i<N)
    {
        /// The element at the front of the queue is the smallest element of
        /// previous window, so insert it result
        MinWindow.push_back(A[Q.front()]);

        /// Remove the elements which are out of this window
        while(!Q.empty() and Q.front()<=i-k)Q.pop_front();

        /// Remove all elements larger than the currently
        /// being added element (remove useless elements)
        while(!Q.empty() and A[i]<=A[Q.back()])Q.pop_back();

        /// Add current element at the rear of Qi
        Q.push_back(i);
        i++;
    }
    /// insert the minimum element of last window
    MinWindow.push_back(A[Q.front()]);
    return MinWindow;
}
int main()
{
    int A[]={100,10, -1, 2,-3,-4,10, 1,100,20};
    vector<int>a=SlidingRMQ(A,10,2);
    for(int i=0;i<a.size();i++)cout<<a[i]<<" ";
    return 0;
}
```

1.2.6 Sparse Table

```cpp
/**
    Compute sparse table in O(NlogN)
    query in O(1)
    Ref link: https://www.topcoder.com/community/data-science/data-science-tutorials/range-minimum-query-and-lowest-common-ancestor/
**/
#include<bits/stdc++.h>
using namespace std;
#define Max 10000005
int rmq[24][Max];
int A[Max];
void Compute_ST(int N)
{
    for (int i = 0; i < N; ++i)rmq[0][i] = i;
    for (int k = 1; (1 << k) < N; ++k)
    {
        for (int i = 0; i + (1 << k) <= N; i++)
        {
            int x = rmq[k - 1][i];
            int y = rmq[k - 1][i + (1 << k - 1)];
            rmq[k][i] = A[x] <= A[y] ? x : y;
        }
    }
}

int RMQ(int i, int j)
{
    int k = log2(j-i);
    int x = rmq[k][i];
    int y = rmq[k][j - (1 << k) + 1];
    return A[x] <= A[y] ? x : y;
}

int main()
{

    return 0;
}
```

1.3 Heavy Light Decomposition

```cpp
#include<bits/stdc++.h>
using namespace std;
#define pp pair<int,int>
#define pb push_back
const int Max=10000;
struct info
{
    int v,cost;
    info(int v=0,int cost=0):v(v),cost(cost){};
};
vector<pp>edges;
vector<info>Graph[Max+5];
int Tree[5*Max+5],BaseArray[Max+5],SubTreeSize[Max+5];
int ChainHead[Max+5],ChainNum[Max+5],PosInBaseArray[Max+5],ChainNo;
int Level[Max+5],Parent[Max+5],SparseTable[Max+5][16];
int ptr;
void init(int N)
{
    for(int i=0;i<=N;i++)
    {
        Graph[i].clear(),ChainHead[i]=-1;
        for(int j=0;j<=15;j++)SparseTable[i][j]=-1;
    }
    edges.clear();
    ptr=ChainNo=0;
}
```

```
void buildSegmentTree(int l,int r,int indx)
{
    if(l==r)
    {
        Tree[indx]=BaseArray[l];
        return;
    }
    int mid=(l+r)>>1;
    int lindx=indx<<1;
    int rindx=lindx|1;
    buildSegmentTree(l,mid,lindx);
    buildSegmentTree(mid+1,r,rindx);
    Tree[indx]=max(Tree[lindx],Tree[rindx]);
}
void updateSegmentTree(int l,int r,int indx,int update_indx,int value)
{
    if(l==r)
    {
        Tree[indx]=value;
        return;
    }
    int mid=(l+r)>>1;
    int lindx=indx<<1;
    int rindx=lindx|1;
    if(update_indx<=mid)updateSegmentTree(l,mid,lindx,update_indx,value);
    else updateSegmentTree(mid+1,r,rindx,update_indx,value);
    Tree[indx]=max(Tree[lindx],Tree[rindx]);
}
int querySegmentTree(int l,int r,int indx,int x,int y)
{
    if(l>y||r<x)return 0;
    if(x<=l&&y>=r)return Tree[indx];
    int mid=(l+r)>>1;
    int lindx=indx<<1;
    int rindx=lindx|1;
    int c1=0,c2=0;
    if(x<=mid)c1=querySegmentTree(l,mid,lindx,x,y);
    if(y>mid)c2=querySegmentTree(mid+1,r,rindx,x,y);
    return max(c1,c2);
}
void dfs(int from,int u,int depth)
{
    Level[u]=depth;
    Parent[u]=from;
    SubTreeSize[u]=1;
    int sz=Graph[u].size();
    for(int i=0;i<sz;i++)
    {
        int v=Graph[u][i].v;
        if(v==from)continue;
        dfs(u,v,depth+1);
        SubTreeSize[u]+=SubTreeSize[v];
    }
}
void sparseTable(int N)
{
    for(int i=0;i<=N;i++)SparseTable[i][0]=Parent[i];
    for(int j=1;(1<<j)<=N;j++)
    {
        for(int i=0;i<=N;i++)
        {
            if(SparseTable[i][j-1]!=-1)
            {
                int a=SparseTable[i][j-1];
                SparseTable[i][j]=SparseTable[a][j-1];
            }
        }
    }
}
int LCA(int p,int q)
```

```
{
    if(Level[p]<Level[q])swap(p,q);
    int Log=log2(Level[p])+1;
    for(int i=Log;i>=0;i--)
    {
        if((Level[p]-(1<<i))>=Level[q])p=SparseTable[p][i];
    }
    if(p==q)return p;
    for(int i=Log;i>=0;i--)
    {
        if(SparseTable[p][i]!=-1&&SparseTable[p][i]!=SparseTable[q][i])
        {
            p=SparseTable[p][i],q=SparseTable[q][i];
        }
    }
    return Parent[p];
}
/**
 * Actual HL-Decomposition part
 * Initially all entries of chainHead[] are set to -1.
 * So when ever a new chain is started, chain head is correctly assigned.
 * As we add a new node to chain, we will note its position in the baseArray.
 * In the first for loop we find the child node which has maximum sub-tree size.
 * The following if condition is failed for leaf nodes.
 * When the if condition passes, we expand the chain to special child.
 * In the second for loop we recursively call the function on all normal nodes.
 * chainNo++ ensures that we are creating a new chain for each normal child.
 **/
void heavyLightDecompositon(int from,int curNode,int cost)
{
    if(ChainHead[ChainNo]==-1)ChainHead[ChainNo]=curNode; /// Assign chain head
    ChainNum[curNode]=ChainNo;
    PosInBaseArray[curNode]=ptr; /// Position of this node in baseArray which we will use in Segtree
    BaseArray[ptr++]=cost;
    int sc=-1,nextCost;
    int sz=Graph[curNode].size();
    for(int i=0;i<sz;i++)  /// Loop to find special child
    {
        int v=Graph[curNode][i].v;
        if(v==from)continue;
        if(sc==-1||SubTreeSize[sc]<SubTreeSize[v])
        {
            sc=v;
            nextCost=Graph[curNode][i].cost;
        }
    }
    if(sc!=-1)heavyLightDecompositon(curNode,sc,nextCost); /// Expand the chain
    for(int i=0;i<sz;i++)
    {
        int v=Graph[curNode][i].v;
        int cost=Graph[curNode][i].cost;
        if(v==from||sc==v)continue;
        ChainNo++;
        heavyLightDecompositon(curNode,v,cost);
    }
}
void updateTree(int ith,int val)
{
    pp a=edges[ith];
    int u=a.first,v=a.second;
    int indx=PosInBaseArray[u];
    if(Level[u]<Level[v])indx=PosInBaseArray[v];
    updateSegmentTree(0,ptr-1,1,indx,val);
}
/**
 * query_up:
 * It takes two nodes u and v, condition is that v is an ancestor of u
 * We query the chain in which u is present till chain head, then move to next chain up
 * We do that way till u and v are in the same chain, we query for that part of chain and break
 **/
int queryUp(int u,int v)
{
    if(u==v)return 0;
    int uchain,vchain=ChainNum[v],ans=-1;
    while(true)
    {
        uchain=ChainNum[u];
        if(uchain==vchain)
        {
            if(u==v)         /// Both u and v are in the same chain, so we need to query from u to v, update answer and break.
                break;       /// We break because we came from u up till v, we are done
            ans=max(ans,querySegmentTree(0,ptr-1,1,PosInBaseArray[v]+1,PosInBaseArray[u]));
            break;
        }
        int uchainhead=ChainHead[uchain];
        ans=max(ans,querySegmentTree(0,ptr-1,1,PosInBaseArray[uchainhead],PosInBaseArray[u]));
                    /// Above is call to segment tree query function. We do from chainHead of u till u. That is the whole chain from
        u=Parent[uchainhead];
    }
    return ans;
}
int queryTree(int u,int v)
{
    int lca=LCA(u,v);
    return max(queryUp(u,lca),queryUp(v,lca));
}
int main()
{
    int test;
    cin>>test;
    while(test--)
    {
        int N;
        cin>>N;
        init(N);
        for(int i=0;i<N-1;i++)
        {
            int u,v,c;
            cin>>u>>v>>c;
            u--,v--;
            Graph[u].pb(info(v,c));
            Graph[v].pb(info(u,c));
            edges.pb(pp(u,v));
        }
        dfs(-1,0,0);
        sparseTable(N);
        heavyLightDecompositon(-1,0,-1);
        buildSegmentTree(0,ptr-1,1);
        string ch;
        int x,y;
        while(true)
        {
            cin>>ch;
            if(ch[0]=='D')break;
            cin>>x>>y;
            if(ch[0]=='Q')printf("%d\n",queryTree(x-1,y-1));
            else if(ch[0]=='C')updateTree(x-1,y);
        }
    }
    return 0;
}
```

1.4 Ternary Bit Mask

```

int more_bit[10];
int get_bit(int mask , int pos)
{
    return (mask / more_bit[pos]) % 3;
}
int set_bit(int mask, int pos , int bit)
{
    int tmp = (mask / more_bit[pos]) % 3;
    mask -= tmp * more_bit[pos];
    mask += bit * more_bit[pos];
    return mask;
}
void init(void)
{
    more_bit[0] = 3;
    for(int i = 1; i < 10; i++) more_bit[i] = 3 * more_bit[i - 1];
}
```

Chapter 2

Graph Theory

2.1 DFS

2.1.1 Bicoloring

```
///color will be initial with -1
int color[20005];
bool dfs(int u,int c)
{
    if(color[u]==c)return true;
    if(color[u]==(1-c))return false;
    color[u]=c;
    bool ret=true;
    for(auto v:graph[u])ret&=dfs(v,1-c);
    return ret;
}
```

2.1.2 Cycle Finding

```
int color[20005];
bool dfs(int u)
{
    color[u]=GREY;
    bool no_cycle=true;
    for(auto v:graph[u])
    {
        if(color[v]==WHITE)
        {
            no_cycle=dfs(v);
        }
        else if(color[v]==GREY)return false;
    }
    color[u]=BLACK;
    return no_cycle;
}
```

2.2 Topological Sort

```
#include<bits/stdc++.h>
using namespace std;
#define WHITE 0
#define GREY 1
#define BLACK 2
vector<int> graph[100005];
vector<int> ans;
int visit[100005];
bool dfs(int u)
{
    visit[u]=GREY;
    bool no_cycle=true;
    int sz=graph[u].size();
    for(int i=0;i<sz;i++)
    {
        int v=graph[u][i];
        if(visit[v]==WHITE)
        {
            no_cycle=dfs(v);
        }
        else if(visit[v]==GREY)return false;
    }
    visit[u]=BLACK;
    ans.push_back(u);
    return no_cycle;
}
bool topsort(int N)
{
    ans.clear();
    memset(visit,false,sizeof(visit));
    int no_cycle=true;
    for(int i=0;i<N;i++)
    {
        if(visit[i]==WHITE)no_cycle&=dfs(i);
    }
    return no_cycle;
}
int main()
{
    return 0;
}
```

2.3 Havel Hakimi

```
/**
    Given N degree d1,d2,d3.....dn. Is it possible to make a graph which have no cycle and
    different two node will be connected with one Edge?

**/
#include<stdio.h>
#include<queue>
#include<vector>
using namespace std;
int main()
{
    int N;
    while(scanf("%d",&N) and N)
    {
        priority_queue<int>Q;
        bool Ok=true;
        int Odd_Node=0;
        for(int i=0;i<N;i++)
        {
            int x;
            scanf("%d",&x);
            if(x>=N or x<0)Ok&=false;
            Odd_Node+=(x%2);
            Q.push(x);
        }
        Ok&=(Odd_Node%2==0); ///Handshaking Theorem
        for(int i=0;i<N and Ok;i++)
        {
            int k=Q.top();
            Q.pop();
            vector<int> v;
            for(int j=0;j<k and Ok;j++)
            {
                int x=Q.top();
                Q.pop();
                x--;
                Ok&=(x>=0);
                v.push_back(x);
            }
            for(int j=0;j<k and Ok;j++)
            {
                Q.push(v[j]);
            }
        }
        if(Ok)printf(”Possible\n”);
        else printf(”Not possible\n”);
    }
    return 0;
}
```

2.4 Articulation Point/Bridge

2.4.1 Find Articulation Point:

```
vector<int>Graph[10000];
bool visit[10000];
int arti[100000];
int discover[100000],Back[100000];
int predfn;
int source;
int child_of_root;
int cnt=0;
void reset()
{
    memset(visit,false,sizeof(visit));
    memset(arti,false,sizeof(arti));
    predfn=child_of_root=0;
}
void articulation(int v)
{
    visit[v]=true;
    predfn++;
    discover[v]=Back[v]=predfn;
    for(int i=0;i<Graph[v].size();i++)
    {
        int w=Graph[v][i];
        if(!visit[w])
        {
            articulation(w);
            Back[v]=min(Back[v],Back[w]);
            if(Back[w]>=discover[v]&&v!=source)
            {
                arti[v]=true;
            }
            else if(v==source)
            {
                child_of_root++;
                if(child_of_root==2)
                {
                    arti[v]=true;
                }
            }
        }
        else
        {
            Back[v]=min(Back[v],discover[w]);
        }
    }
}
```

2.4.2 Find Bridge version 1:

```
vector<int>Graph[200];
int Back[205],Discover[205];
bool visit[205];
bool bridge[205][205];
```

```
int brcount;
void reset(int n)
{
    for(int i=0;i<=n;i++)Graph[i].clear();
    memset(visit,false,sizeof(visit));
    memset(bridge,false,sizeof(false));
    brcount=0;
}
void find_bridge(int u, int parent, int depth)
{
    visit[u] = true;
    Discover[u] = Back[u] = depth;

    for (int i=0 ; i<Graph[u].size() ; i++)
    {
        int v = Graph[u][i];

        if (visit[v] && v!=parent)
        {
            Back[u] = min(Back[u],Discover[v]);
        }
        if (!visit[v])
        {
            find_bridge(v, u, depth+1);
            Back[u] = min(Back[u],Back[v]);
            if (Back[v]>Discover[u])
            {
                brcount++;
                bridge[u][v] = bridge[v][u] = true;
            }
        }
    }

}
```

2.4.3 Find Bridge version 2:

```
void find_bridge(int node, int parent)
{
    discovery_time[node] = bedge[node] = ++T;
    int to, i, connected = adj[node].size();
    for(i = 0; i < connected; i++)
    {
        to = adj[node][i];
        if(to == parent) continue;
        if(!discovery_time[to])
        {
            printf("%d %d\n", node, to);
            find_bridge(to, node);
            bedge[node] = min(bedge[node], bedge[to]);
            if(bedge[to] > discovery_time[node]) printf("%d %d\n", to, node);
        }
        else if(discovery_time[node] > discovery_time[to])
        {
            printf("%d %d\n", node, to);
            bedge[node] = min(bedge[node], discovery_time[to]);
        }
    }
}
```

Chapter 3

Flow networks/ matching

3.1 Max Flow

```
#include<bits/stdc++.h>
using namespace std;
#define pb push_back
#define MN 1000
typedef vector< vector<int> > vint2D;
const int inf=(1<<29);
vint2D graph;
int Cost[MN][MN];
int parent[MN+5];
int flow;
void init(int N)
{
    graph=vint2D(N);
    memset(Cost,0,sizeof(Cost));
}
void AddEdge(int u,int v,int cost)
{
    graph[u].pb(v);
    graph[v].pb(u);
    Cost[u][v]+=cost;
    Cost[v][u]+=cost;
}
bool augmenting_path(int source,int sink)
{
    memset(parent,-1,sizeof(parent));
    queue<int>Q;
    Q.push(source);
    while(!Q.empty())
    {
        int u=Q.front();
        Q.pop();
        int sz=graph[u].size();
        for(int i=0;i<sz;i++)
        {
            int v=graph[u][i];
            if(parent[v]==-1 and Cost[u][v]>0)
            {
                parent[v]=u;
                Q.push(v);
                if(v==sink)return true;
            }
        }
    }
    return false;
}
void path(int v,int source)
{
    int u=parent[v];
    flow=min(flow,Cost[u][v]);
    if(source!=u)path(u,source);
    Cost[u][v]-=flow;
    Cost[v][u]+=flow;
    return;
}
int max_flow(int source,int sink)
{
    int ret=0;
    while(augmenting_path(source,sink))
    {
        flow=inf;
        path(sink,source);
        ret+=flow;
    }
    return ret;
}
int main()
{
    int test;
    scanf("%d",&test);
    while(test--)
    {
        int P,S,C,M;
        scanf("%d %d %d %d",&P,&S,&C,&M);
        init(P+S+5);
        int superSource=0,SuperSikn=P+S+1;
        for(int i=1;i<=P;i++)AddEdge(superSource,i,1);
        for(int i=1;i<=S;i++)AddEdge(P+1,SuperSikn,C);
        for(int i=0;i<M;i++)
        {
            int x,y;
            scanf("%d %d",&x,&y);
            AddEdge(x,P+y,(1<<30));
        }
        printf("%d\n",max_flow(superSource,SuperSikn));
    }
    return 0;
}
```

Chapter 4

Dynamic programming

4.1 Edit Distance

```
#include<bits/stdc++.h>
using namespace std;
int dp[88][88];
int N,M,step;
char S1[88],S2[88];
int solve(int i,int j)
{
    if(i==N and j==M)return 0;
    if(i==N)return M-j;
    if(j==M)return N-i;
    int &ret=dp[i][j];
    if(ret!=-1)return ret;
    ret=(1<<28);
    if(S1[i]==S2[j])ret=solve(i+1,j+1);
    else
    {
        ret=min(ret,solve(i,j+1)+1);
        ret=min(ret,solve(i+1,j)+1);
        ret=min(ret,solve(i+1,j+1)+1);
    }
    return ret;
}
void pathPrint(int i,int j,int del,int ins,int st)
{
    if(i==N&&j==M) return ;
    if(i==N)
    {
        for(int k=j;k<M;k++,i++)
        {
            printf("%d Insert %d,%c\n",st++,i-del+1+ins,S2[k]);
        }
        return ;
    }
    if(j==M)
    {
        for(;i<N;i++)
        {
            printf("%d Delete %d\n",st++,i-del+1+ins);
            del++;
        }
        return ;
    }
    int ret = solve(i,j);
    int tmp;
    if(S1[i]==S2[j])
    {
        tmp=solve(i+1,j+1);
        if(ret==tmp)
        {
            pathPrint(i+1,j+1,del,ins,st);
            return ;
        }
    }
    tmp=solve(i,j+1)+1;
    if(tmp==ret)
    {
        printf("%d Insert %d,%c\n",st,i-del+1+ins,S2[j]);
        pathPrint(i,j+1,del,ins+1,st+1);
        return ;
    }
    tmp=solve(i+1,j)+1;
    if(tmp==ret)
    {
        printf("%d Delete %d\n",st,i-del+1+ins);
        pathPrint(i+1,j,del+1,ins,st+1);
        return ;
    }
    tmp=solve(i+1,j+1)+1;
    if(tmp==ret)
    {
        printf("%d Replace %d,%c\n",st,i-del+1+ins,S2[j]);
        pathPrint(i+1,j+1,del,ins,st+1);
        return ;
    }
    return ;
}
int main()
{
    bool New=false;
    while(gets(S1))
    {
        gets(S2);
        if(New) printf("\n");
        New=true;
        N=strlen(S1);
        M=strlen(S2);
        memset(dp,-1,sizeof(dp));
        step=solve(0,0);
        printf("%d\n",step);
        pathPrint(0,0,0,0,1);
    }
    return 0;
}
```

Chapter 5

Strings

5.1 KMP

Tutorial

```cpp
#include<bits/stdc++.h>
using namespace std;
char TXT[10000000],ptr[10000000];
vector<int> compute_prefix(const char *p)
{
    int m=strlen(p+1);
    vector<int> prefix(m+1);
    prefix[1]=0;
    int k=0;
    for(int i=2; i<=m; i++)
    {
        while(k>0 and p[k+1]!=p[i])k=prefix[k];
        if(p[k+1]==p[i])k=k+1;
        prefix[i]=k;
    }
    return prefix;
}
vector<int> KMP_match(const char *txt,const char *ptrn)
{
    int n=strlen(txt+1);
    int m=strlen(ptrn+1);
    vector<int> Prefix=compute_prefix(ptrn);
    vector<int>Match_position;
    int q=0;
    for(int i=1; i<=n; i++)
    {
        while(q>0 and ptrn[q+1]!=txt[i])q=Prefix[q];
        if(ptrn[q+1]==txt[i])q=q+1;
        if(q==m)
        {
            Match_position.push_back(i-m);
            q=Prefix[q];
        }
    }
    return Match_position;
}
int main()
{
    scanf("%s %s",TXT+1,ptr+1);
    vector<int> Match_position=KMP_match(TXT,ptr);
    for(int i=0; i<Match_position.size(); i++)
    {
        if(!i)printf("%d",Match_position[i]);
        else printf(" %d",Match_position[i]);
    }
    return 0;
}
```

5.2 Aho Corasick

5.2.1 Aho Corasick with Dynamic Trie

```
#include<bits/stdc++.h>
using namespace std;
#define Max 26
int getID(char c)
{
    return c>='a'?c-'a':c-'A';
}
char inp[1000005];
char text[1000005];
int ans[5000];
map<string,int>Map;
vector<int>v;
struct Trie
{
    Trie *next[26],*fail;
    int stringMap;
    Trie()
    {
        stringMap=0;
        for(int i=0;i<Max;i++)next[i]=NULL;
        fail=NULL;
    }
};
Trie *root;
void Insert(const char *str,int M)
{
    Trie *p=root;
    for(int i=0;str[i];i++)
    {
        int id=getID(str[i]);
        if(p->next[id]==NULL)p->next[id]=new Trie();
        p=p->next[id];
    }
    p->stringMap=M;
}
void computeFailure()
{
    Trie *u,*prefix;
    queue<Trie*>Q;
    Q.push(root);
    while(!Q.empty())
    {
        u=Q.front(); ///Take a new node
        Q.pop();
        for(int i=0;i<Max;i++)
        {
            if(u->next[i]!=NULL) ///select fail position of ith node of parent u
            {
                prefix=u->fail; /// Going to u node fail position/ prefix position
                while(prefix!=NULL)
                {
                    if(prefix->next[i]!=NULL) ///if match found
                    {
                        u->next[i]->fail=prefix->next[i];
                        break;
                    }
                    prefix=prefix->fail; /// match not found, going to upper child prefix position
                }
                if(prefix==NULL)u->next[i]->fail=root;
                Q.push(u->next[i]);
            }
        }
    }
}
void AhoCorasick(const char *str)
```

```
{
    Trie *p=root;
    int cnt=0;
    for(int i=0;str[i];i++)
    {
        int id=getID(str[i]);
        while(p->next[id]==NULL&&p!=root)p=p->fail,cnt++;
        if(p->next[id]!=NULL)p=p->next[id];
        Trie *tp=p;
        while(tp!=root)
        {
            cnt++;
            if(tp->stringMap>0)ans[tp->stringMap]++;
            tp=tp->fail;
        }
    }
}
void Delete(Trie *u)
{
    if(u==NULL)return;
    for(int i=0;i<Max;i++)Delete(u->next[i]);
    delete u;
}

int main()
{
    int test;
    scanf("%d",&test);
    for(int t=1;t<=test;t++)
    {
        Map.clear();
        v.clear();
        memset(ans,0,sizeof(ans));
        root=new Trie();
        int N;
        scanf("%d",&N);
        scanf("%s",text);
        int cnt=1;
        for(int i=0;i<N;i++)
        {
            scanf("%s",inp);
            if(Map.find(inp)==Map.end())Map[inp]=cnt++;
            Insert(inp,Map[inp]);
            v.push_back(Map[inp]);
        }
        computeFailure();
        AhoCorasick(text);
        printf("Case %d:\n",t);
        for(int i=0;i<N;i++)
        {
            printf("%d\n",ans[v[i]]);
        }
        Delete(root);
    }
    return 0;
}
```

5.2.2 Aho Corasick with Static Trie

```
#include<bits/stdc++.h>
using namespace std;
#define root 0
#define NuLL -1
#define Max 248878
#define MC 26
int ans[10000];
char text[1000005];
char inp[100000];
map<string,int>Map;
vector<int> v;
```

```
int getID(const char c)
{
    return c>='a'?c-'a':c-'A';
}
struct Trie
{
    struct node
    {
        int Next[26],fail;
        int stringMap;
        void clear()
        {
            memset(Next,-1,sizeof(Next));
            fail=-1;
            stringMap=0;
        }
    }T[Max];
    int ptr;
    void clear()
    {
        ptr=1;
        T[0].clear();
    }
    void Insert(char *str,int M)
    {
        int p=0;
        for(int i=0;str[i];i++)
        {
            int id=getID(str[i]);
            if(T[p].Next[id]==-1)
            {
                T[p].Next[id]=ptr;
                T[ptr++].clear();
            }
            int q=p;
            p=T[p].Next[id];
            if(p<0)
            {
                while(1);
            }
        }
        T[p].stringMap=M;
    }
    void ComputeFailure()
    {
        queue<int>Q;
        Q.push(root);
        int u,prefix;
        int cnt=0,cnt2=0;
        while(!Q.empty())
        {
            u=Q.front();
            Q.pop();
            for(int i=0;i<MC;i++)
            {
                if(T[u].Next[i]!=NuLL)
                {
                    int now=T[u].Next[i];
                    prefix=T[u].fail;
                    while(prefix!=NuLL)
                    {
                        cnt2++;
                        if(T[prefix].Next[i]!=NuLL)
                        {
                            T[now].fail=T[prefix].Next[i];
                            break;
                        }
                        prefix=T[prefix].fail;
                    }
                    if(prefix==NuLL)T[now].fail=root;
                    Q.push(now);
                }
            }
        }
    }
};
void AhoCorasick(const Trie &A, const char *str)
{
    int p=root;
    int cnt1=0,cnt2=0;
    for(int i=0;str[i];i++)
    {
        int id=getID(str[i]);
        while(A.T[p].Next[id]==NuLL&&p!=root)p=A.T[p].fail;
        if(p!=NuLL&&A.T[p].Next[id]!=NuLL)p=A.T[p].Next[id];
        int tp=p;
        while(tp!=root)
        {
            if(A.T[tp].stringMap>0)ans[A.T[tp].stringMap]++;
            tp=A.T[tp].fail;
        }
    }
}
Trie A;
int main()
{
    #ifdef _ANICK_
        freopen("input.txt","r",stdin);
    #endif // _ANICK_
    int test;
    scanf("%d",&test);
    for(int t=1;t<=test;t++)
    {
        Map.clear();
        v.clear();
        memset(ans,0,sizeof(ans));
        A.clear();
        int N;
        scanf("%d",&N);
        scanf("%s",text);
        int cnt=1;
        for(int i=0;i<N;i++)
        {
            scanf("%s",inp);
            if(Map.find(inp)==Map.end())Map[inp]=cnt++;
            A.Insert(inp,Map[inp]);
            v.push_back(Map[inp]);
        }
        A.ComputeFailure();
        AhoCorasick(A,text);
        printf("Case %d:\n",t);
        for(int i=0;i<N;i++)
        {
            printf("%d\n",ans[v[i]]);
        }
    }
    return 0;
}
```

5.3 Manacher's Algorithm

```
#include<bits/stdc++.h>
using namespace std;
string s, t;
char str[1000005];
void prepare_string()
{
    int i;
    t="^#";
    for(i=0;i<s.size();i++)
```

```
        t+=s[i],t+="#";
    t+="$";
}

int manacher()
{
    prepare_string();
    int P[t.size()],c=0,r=0,i,i_mirror,n=t.size()-1;
    for(i=1;i<n;i++)
    {
        i_mirror=(2*c)-i;
        P[i]=r>i?min(r-i,P[i_mirror]):0;
        while(t[i+1+P[i]]==t[i-1-P[i]]) P[i]++;
        if(i+P[i]>r)
        {
            c=i;
            r=i+P[i];
        }
    }
    return *max_element(P+1,P+n);
}
int main()
{
    int kase=1;
    while(scanf("%s", str)&&str[0]!= 'E')
    {
        s=str;
        printf("Case %d: %d\n", kase++, manacher());
    }
    return 0;
}
```

Chapter 6

Computational geometry

Chapter 7

Math

7.1 Reduce Ratio

$\left(\frac{A}{B}\right)$ ratio reduce to $\left(\frac{x}{y}\right)$

```
int main()
{
    int A,B,x,y;
    cin>>A>>B>>x>>y;
    int g=__gcd(x,y);
    x/=g,y/=g;
    int t=min(A/x,B/y);
    cout<<x*t<<" "<<y*t<<endl;
    return 0;
}
```

7.2 Floyd's Cycle Finding algorithm

```
#include<bits/stdc++.h>
using namespace std;
#define pp pair<int,int>
int Z,L,M,I;
int f(int L)
{
    return (Z*L+I)%M;
}
pp CycleFinding()
{
    ///L here initial seed
    int hare,tortoise,lambda,meu;
    bool cyclefind=false;
    hare=tortoise=L;
    while(!cyclefind)
    {
        tortoise=f(tortoise);
        hare=f(hare);
        hare=f(hare);
        if(hare==tortoise)cyclefind=true;
    }
    hare=L;
    meu=0;
    while(hare!=tortoise)
    {
        meu++;
        hare=f(hare);
        tortoise=f(tortoise);
    }
    int i=0;
    hare=L;
    while(i<=meu)
    {
        i++;
        hare=f(hare);
    }
    tortoise=f(hare);
    lambda=1;
    while(hare!=tortoise)
    {
        tortoise=f(tortoise);
        lambda++;
    }
    return {meu,lambda}; ///meu is starting index and lambda is cycle length
}
int main()
{
    int t=1;
    while(scanf("%d %d %d %d",&Z,&I,&M,&L) and (Z or I or M or L))
    {
        pp a=CycleFinding();
        cout<<"Cycle starts from index "<<a.first<<"\nCycle length is "<<a.second<<endl;
    }
    return 0;
}
```

Chapter 8

Number Theory

8.1 NCR

8.1.1 Lucas Theorem

```
/**
    Fine NCR % M when N C M are large number.
    using Lucas theorem.
**/
#include<bits/stdc++.h>
using namespace std;
typedef long long LLD;
LLD mod=1000003;
LLD big_mod(LLD n,LLD p,LLD m)
{
    if(p==0)return (LLD)1;
    LLD x=big_mod(n,p/2,m);
    x=(x*x)%m;
    if(p&1)x=(x*n)%m;
    return x;
}
LLD inverse_modulo(LLD t,LLD m)
{
    return big_mod(t,m-2,m);
}
LLD combi(LLD n, LLD k,LLD m)
{
    if(n<k)
        return 0;
    if(n-k<k)
        return combi(n,n-k,m);
    LLD i,p=1,t=1;
    for(i=n-k+1; i<=n; i++)
        p=(p*i)%m;
    for(i=1; i<=k; i++)
        t=(t*i)%m;
    return (p*inverse_modulo(t,m))%m;
}
LLD lucas(LLD n, LLD k, LLD m)
{
    if(n<k)
        return 0;
    if(k==0 || n==k)
        return 1;
    return (lucas(n/m,k/m,m)*combi(n%m,k%m,m))%m;
}
int main()
{
    return 0;
}
```