

Algorithm Code Book

Tanvir Hasan Anick

July 16, 2015

Contents

1	Data Structure	3
1.1	Trie	3
1.1.1	Static Trie	3
1.2	RMQ	4
1.2.1	Bit	4
1.2.2	Square Root Decompostion	5
1.2.3	MO's Algorithm	5
1.2.4	Segment Tree	7
1.2.5	Sliding Window RMQ	14
1.2.6	Sparse Table	15
1.3	Heavy Light Decomposition	15
1.4	Ternary Bit Mask	19
2	Graph Theory	20
2.1	DFS	20
2.1.1	Bicoloring	20
2.1.2	Cycle Finding	20
2.2	Topological Sort	20
2.3	Havel Hakimi	21
2.4	Articulation Point/Bridge	22
2.4.1	Find Articulation Point:	22
2.4.2	Find Bridge version 1:	22
2.4.3	Find Bridge version 2:	23
3	Flow networks/ matching	24
3.1	Max Flow	24
4	Dynamic programming	26
4.1	Edit Distance	26
5	Strings	28
5.1	KMP	28
5.2	Aho Corasick	29
5.2.1	Aho Corasick with Dynamic Trie	29
5.2.2	Aho Corasick with Static Trie	30
5.3	Manacher's Algorithm	32
6	Computational geometry	34
7	Math	35
7.1	Reduce Ratio	35
7.2	Floyd's Cycle Finding algorithm	35

8	Number Theory	37
8.1	NCR	37
8.1.1	Lucas Theorem	37

Chapter 1

Data Structure

1.1 Trie

1.1.1 Static Trie

```
1 #define Max 10005
2 int getId(char c)
3 {
4     return c>='a'?c-'a':c-'A'+26;
5 }
6 struct Trie
7 {
8     struct Tree
9     {
10         int Next[52];
11         bool word;
12         void clear()
13         {
14             word=false;
15             memset(Next,-1,sizeof(Next));
16         }
17     }T[Max];
18     int ptr;
19     void clear()
20     {
21         ptr=1;
22         T[0].clear();
23         memset(T[0].Next,0,sizeof(T[0].Next));
24     }
25     void Insert(const char *str)
26     {
27         int p=0;
28         for(int i=0;str[i];i++)
29         {
30             int id=getId(str[i]);
31             if(T[p].Next[id]<=0)
32             {
33                 T[p].Next[id]=ptr;
34                 T[ptr++].clear();
35             }
36             p=T[p].Next[id];
37         }
38         T[p].word=true;
39     }
40     bool Search(const char *str)
41     {
42         int p=0;
43         for(int i=0;str[i];i++)
44         {
45             int id=getId(str[i]);
46             if(T[p].Next[id]>0)
47             {
```

```

48         p=T[p].Next[id];
49     }
50     else return false;
51 }
52 return T[p].word;
53 }
54 };
55 Trie A;

```

1.2 RMQ

1.2.1 Bit

1D Bit

```

1 #define MaxVal 100000
2 int Bit[MaxVal];
3 /**find sum from 1 to idx**/
4 int read(int idx)
5 {
6     int sum = 0;
7     while (idx > 0)
8     {
9         sum += Bit[idx];
10        idx -= (idx & -idx);
11    }
12    return sum;
13 }
14 /**update value ind to MaxVal**/
15 void update(int idx ,int val)
16 {
17     while (idx <= MaxVal)
18     {
19         Bit[idx] += val;
20         idx += (idx & -idx);
21     }
22 }
23
24 /**Find the value of idx**/
25 int readSingle(int idx)
26 {
27     int sum = Bit[idx]; /// sum will be decreased
28     if (idx > 0) /// special case
29     {
30         int z = idx - (idx & -idx); /// make z first
31         idx--; /// idx is no important any more, so instead y, you can use idx
32         while (idx != z) /// at some iteration idx (y) will become z
33         {
34             sum -= Bit[idx]; /// substruct Bit frequency which is between y and "the
35             same path"
36             idx -= (idx & -idx);
37         }
38     }
39     return sum;

```

2D Bit

```

1 void updatey(int x , int y , int val)
2 {
3     while (y <= max_y)
4     {
5         tree[x][y] += val;
6         y += (y & -y);
7     }
8 }
9 void update(int x , int y , int val)
10 {
11     while (x <= max_x)

```

```

12     {
13         updatey(x , y , val); // this function should update array tree[x]
14         x += (x & -x);
15     }
16 }

```

1.2.2 Square Root Decomposition

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int sz=100005;
4  const int inf=(1<<28);
5  template<typename t> t MIN3(t a,t b, t c)
6  {
7      return min(a,min(b,c));
8  }
9  int BLOCK[400];
10 int arr[sz];
11 int getId(int indx,int blockSZ)
12 {
13     return indx/blockSZ;
14 }
15 void init(int sz)
16 {
17     for(int i=0; i<=sz; i++)BLOCK[i]=inf;
18 }
19 void update(int val,int indx,int blockSZ)
20 {
21     int id=getId(indx,blockSZ);
22     BLOCK[id]=min(BLOCK[id],val);
23 }
24 int query(int L,int R,int blockSZ)
25 {
26     int lid=getId(L,blockSZ);
27     int rid=getId(R,blockSZ);
28     if(lid==rid)
29     {
30         int ret=inf;
31         for(int i=L; i<=R; i++)ret=min(ret,arr[i]);
32         return ret;
33     }
34     int m1=inf,m2=inf,m3=inf;
35     for(int i=L; i<(lid+1)*blockSZ; i++)m1=min(m1,arr[i]);
36     for(int i=lid+1; i<rid; i++)m2=min(m2,BLOCK[i]);
37     for(int i=rid*blockSZ; i<=R; i++)m3=min(m3,arr[i]);
38     return MIN3(m1,m2,m3);
39 }
40 int main()
41 {
42     int N,Q;
43     scanf("%d %d",&N,&Q);
44     int blockSZ=sqrt(N);
45     init(blockSZ);
46     for(int i=0; i<N; i++)
47     {
48         int x;
49         scanf("%d",&x);
50         arr[i]=x;
51         update(x,i,blockSZ);
52     }
53     while(Q--)
54     {
55         int x,y;
56         scanf("%d %d",&x,&y);
57         printf("%d\n",query(x,y,blockSZ));
58     }
59     return 0;
60 }

```

1.2.3 MO's Algorithm

```

1  /**
2   MO's Algorithm
3   problem: http://www.spoj.com/problems/DQUERY
4
5   MOs algorithm is just an order in which we process the queries.
6   We were given M queries , we will re-order the queries in a particular order and
7   then process them.
8   Clearly , this is an off-line algorithm . Each query has L and R, we will call
9   them opening and closing .
10  Let us divide the given input array into Sqrt(N) blocks .
11  Each block will be  $N / \text{Sqrt}(N) = \text{Sqrt}(N)$  size .
12  Each opening has to fall in one of these blocks .
13  Each closing has to fall in one of these blocks .
14
15  All the queries are first ordered in ascending order of their block number (
16  block number is the block in which its opening falls) .
17  Ties are ordered in ascending order of their R value .
18
19  */
20 #include<bits/stdc++.h>
21 using namespace std;
22 #define Mx 30005
23 #define MxNum 1000005
24 int BlockSize;
25 int Answer;
26 int Freq[MxNum] ,Num[Mx];
27 struct info
28 {
29     int L,R,qno;
30     info ( int L=0,int R=0,int qno=0):L(L) ,R(R) ,qno(qno) {};
31     bool operator<(const info &a) const
32     {
33         if(L/BlockSize!=a.L/BlockSize) return L/BlockSize<a.L/BlockSize;
34         return R<a.R;
35     }
36 }Query[200005];
37 int StoreAnswer[200005];
38 void Add(int indx)
39 {
40     Freq[Num[indx]]++;
41     if (Freq[Num[indx]]==1) Answer++;
42 }
43 void Remove(int indx)
44 {
45     Freq[Num[indx]]--;
46     if (Freq[Num[indx]]==0) Answer--;
47 }
48 int main()
49 {
50     int N;
51     scanf("%d",&N);
52     BlockSize=sqrt(N);
53     for (int i=0;i<N;i++)
54     {
55         scanf("%d",&Num[i]);
56     }
57     int Q;
58     scanf("%d",&Q);
59     for (int i=0;i<Q;i++)
60     {
61         int x,y;
62         scanf("%d %d",&x,&y);
63         Query[i]=info(x-1,y-1,i);
64     }
65     sort(Query,Query+Q);
66     int currentL=0,currentR=0;
67     Answer=0;
68     for (int i=0;i<Q;i++)
69     {
70         int L=Query[i].L;

```

```

68     int R=Query[i].R;
69     while(currentL<L)
70     {
71         Remove(currentL);
72         currentL++;
73     }
74     while(currentL>L)
75     {
76         Add(currentL-1);
77         currentL--;
78     }
79     while(currentR<=R)
80     {
81         Add(currentR);
82         currentR++;
83     }
84     while(currentR>R+1)
85     {
86         Remove(currentR-1);
87         currentR--;
88     }
89     StoreAnswer[Query[i].qno]=Answer;
90 }
91 for(int i=0;i<Q;i++)
92 {
93     printf("%d\n",StoreAnswer[i]);
94 }
95 return 0;
96 }

```

1.2.4 Segment Tree

Lazy Propagation1

```

1  /**
2  **You are given an array of N elements, which are initially all 0. After **that you
   will be given C commands. They are
3  **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.
4  **1 p q – output a line containing a single integer which is the sum of all **the
   array elements between p and q (inclusive)
5  */
6  #include<bits/stdc++.h>
7  using namespace std;
8  typedef long long LLD;
9  LLD tree[3*100005];
10 LLD lazy[3*100005];
11 void update(int left,int right,int index,int x,int y,int value)
12 {
13     if(x<=left&&y>=right)
14     {
15         tree[index]+=(LLD)(right-left+1)*value;
16         lazy[index]+=value;
17         return;
18     }
19     int mid=(left+right)/2;
20     if(lazy[index]!=0)
21     {
22         tree[2*index]+=(LLD)(mid-left+1)*lazy[index];
23         tree[2*index+1]+=(LLD)(right-mid)*lazy[index];
24         lazy[2*index]+=lazy[index];
25         lazy[2*index+1]+=lazy[index];
26         lazy[index]=0;
27     }
28     if(x<=mid)
29     {
30         update(left,mid,2*index,x,y,value);
31     }
32     if(y>mid)
33     {

```



```

34     update(mid+1,right,2*index+1,x,y,value);
35 }
36 tree[index]=tree[2*index]+tree[2*index+1];
37 }
38 LLD query(int left,int right,int index,int x,int y)
39 {
40     LLD a1=0,a2=0;
41     if(x<=left&&right<=y)
42     {
43         return tree[index];
44     }
45     int mid=(left+right)/2;
46     if(lazy[index]!=0)
47     {
48         tree[2*index]+=(LLD)(mid-left+1)*lazy[index];
49         tree[2*index+1]+=(LLD)(right-mid)*lazy[index];
50         lazy[2*index]+=lazy[index];
51         lazy[2*index+1]+=lazy[index];
52         lazy[index]=0;
53     }
54     if(x<=mid)
55     {
56         a1=query(left,mid,2*index,x,y);
57     }
58     if(y>mid)
59     {
60         a2=query(mid+1,right,2*index+1,x,y);
61     }
62     return (a1+a2);
63 }
64 int main()
65 {
66     int test,t;
67     scanf("%d",&test);
68     for(t=1;t<=test;t++)
69     {
70         memset(tree,0,sizeof(tree));
71         memset(lazy,0,sizeof*lazy);
72         int s,q;
73         scanf("%d %d",&s,&q);
74         while(q--)
75         {
76             int x,y,v,dec;
77             scanf("%d",&dec);
78             if(dec)
79             {
80                 scanf("%d %d",&x,&y);
81                 LLD ans=query(0,s-1,1,x-1,y-1);
82                 printf("%lld\n",ans);
83             }
84             else
85             {
86                 scanf("%d %d %d",&x,&y,&v);
87                 update(0,s-1,1,x-1,y-1,v);
88             }
89         }
90     }
91     return 0;
92 }

```

Lazy Propagation2

```

1  /*
2  **You have an array with n elements which is indexed from 0 to n - 1. **Initially
3  all elements are zero. Now you have to deal with two types of **operations
4  **1.Increase the numbers between indices i and j (inclusive) by 1. This **is **
5  represented by the command '0 i j'.
6  **2.Answer how many numbers between indices i and j (inclusive) are **divisible by
7  3. This is represented by the command '1 i j'.
8  */
9  #include<bits/stdc++.h>

```

```

7 using namespace std;
8 #define Max 100010
9 int Tree[8*Max][4];
10 int lazy[8*Max];
11 int temp[4];
12 void build(int left,int right,int indx)
13 {
14     if(left==right)
15     {
16         Tree[indx][0]=1;
17         Tree[indx][1]=Tree[indx][2]=lazy[indx]=0;
18         return;
19     }
20     int mid=(left+right)/2;
21     build(left,mid,2*indx);
22     build(mid+1,right,2*indx+1);
23     for(int i=0;i<3;i++)
24     {
25         Tree[indx][i]=Tree[2*indx][i]+Tree[2*indx+1][i];
26     }
27 }
28 void update(int left,int right,int indx,int x,int y,int add)
29 {
30     if(lazy[indx])
31     {
32         int lazy_val=lazy[indx];
33         lazy[2*indx]=(lazy[2*indx]+lazy_val)%3;
34         lazy[2*indx+1]=(lazy[2*indx+1]+lazy_val)%3;
35         for(int i=0;i<3;i++)temp[(lazy_val+i)%3]=Tree[indx][i];
36         for(int i=0;i<3;i++)Tree[indx][i]=temp[i];
37         lazy[indx]=0;
38     }
39     if(left>y||right<x)return;
40     if(x<=left&&right<=y)
41     {
42         for(int i=0;i<3;i++)
43         {
44             temp[(i+add)%3]=Tree[indx][i];
45         }
46         for(int i=0;i<3;i++)Tree[indx][i]=temp[i];
47         lazy[2*indx]=(lazy[2*indx]+add)%3;
48         lazy[2*indx+1]=(lazy[2*indx+1]+add)%3;
49         return;
50     }
51     int mid=(left+right)/2;
52     update(left,mid,2*indx,x,y,add);
53     update(mid+1,right,2*indx+1,x,y,add);
54     for(int i=0;i<3;i++)
55     {
56         Tree[indx][i]=Tree[2*indx][i]+Tree[2*indx+1][i];
57     }
58 }
59 int query(int left,int right,int indx,int x,int y)
60 {
61     if(lazy[indx])
62     {
63         int lazy_val=lazy[indx];
64         lazy[2*indx]=(lazy[2*indx]+lazy_val)%3;
65         lazy[2*indx+1]=(lazy[2*indx+1]+lazy_val)%3;
66         for(int i=0;i<3;i++)temp[(lazy_val+i)%3]=Tree[indx][i];
67         for(int i=0;i<3;i++)Tree[indx][i]=temp[i];
68         lazy[indx]=0;
69     }
70     if(left>y||right<x)return 0;
71     if(x<=left&&right<=y)return Tree[indx][0];
72     int mid=(left+right)/2;
73     return query(left,mid,2*indx,x,y)+query(mid+1,right,2*indx+1,x,y);
74 }
75 int main()
76 {

```

```

77     int x,y;
78     int test;
79     scanf("%d",&test);
80     for(int t=1;t<=test;t++)
81     {
82         memset(lazy,0,sizeof(lazy));
83         int N,Q;
84         scanf("%d %d",&N,&Q);
85         build(0,N-1,1);
86         printf("Case %d:\n",t);
87         for(int i=0;i<Q;i++)
88         {
89             int d;
90             scanf("%d %d %d",&d,&x,&y);
91             if(d==0)
92             {
93                 update(0,N-1,1,x,y,1);
94             }
95             else printf("%d\n",query(0,N-1,1,x,y));
96         }
97     }
98     return 0;
99 }

```

Segment Tree Variant 1

```

1  /**
2  **Give a array Of N numbers. Finding Maximum cumulative number frequency in **the
   range.
3  **input:
4  **10 4
5  **1 1 1 3 3 3 3 2 2 2
6  **1 5
7  **1 6
8  **1 7
9  **Output:
10 **3
11 **3
12 **4
13 **2
14 */
15 #include<bits/stdc++.h>
16 using namespace std;
17 typedef long long LLD;
18 #define MAX 50005
19 struct info
20 {
21     int Lcnt,Rcnt,Max,Lnum,Rnum;
22     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){};
23 };
24 info Tree[3*MAX];
25 int arr[MAX];
26 info marge(const info &L,const info &R)
27 {
28     info ret;
29     if(L.Rnum==R.Lnum)
30     {
31         ret.Max=max(L.Rcnt+R.Lcnt,max(L.Max,R.Max));
32     }
33     else ret.Max=max(L.Max,R.Max);
34     ret.Lnum=L.Lnum;
35     ret.Rnum=R.Rnum;
36     if(L.Lnum==R.Lnum)ret.Lcnt=L.Lcnt+R.Lcnt;
37     else ret.Lcnt=L.Lcnt;
38     if(L.Rnum==R.Rnum)ret.Rcnt=L.Rcnt+R.Rcnt;
39     else ret.Rcnt=R.Rcnt;
40     return ret;
41 }
42 void build(int L,int R,int indx)
43 {

```

```

44     if (L==R)
45     {
46         Tree[indx]=info(1,1,1,arr[L],arr[R]);
47         return;
48     }
49     int mid=(L+R)>>1;
50     build(L,mid,2*indx);
51     build(mid+1,R,2*indx+1);
52     Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
53 }
54 info query(int L,int R,int indx,int x,int y)
55 {
56     if (L>=x&&R<=y) return Tree[indx];
57     int mid=(L+R)>>1;
58     info c1,c2;
59     if (x<=mid) c1=query(L,mid,2*indx,x,y);
60     if (y>mid) c2=query(mid+1,R,2*indx+1,x,y);
61     return marge(c1,c2);
62 }
63 int main()
64 {
65     int test;
66     scanf("%d",&test);
67     for(int t=1;t<=test;t++)
68     {
69         int N,C,Q;
70         scanf("%d %d %d",&N,&C,&Q);
71         for(int i=0;i<N;i++)
72         {
73             int x;
74             scanf("%d",&arr[i+1]);
75         }
76         build(1,N,1);
77         printf("Case %d:\n",t);
78         while(Q--)
79         {
80             int x,y;
81             scanf("%d %d",&x,&y);
82             printf("%d\n",query(1,N,1,x,y).Max);
83         }
84     }
85     return 0;
86 }

```

Segment Tree Variant 2

```

1  /**
2  **You are given a sequence A of N (N <= 50000) integers between -10000 and 10000.
3  **On this sequence you have to apply M (M <= 50000) operations:
4  **modify the i-th element in the sequence or for given x y print max{Ai + Ai+1 + ..
   + Aj | x<=i<=j<=y }.
5  **/
6  #include<bits/stdc++.h>
7  using namespace std;
8  typedef long long LLD;
9  template<class T> T MAX3(T a,T b,T c) {return max(a,max(b,c));}
10 LLD Inf=(1ll<<60);
11 #define MN 50005
12 struct info
13 {
14     LLD prefixSum;
15     LLD suffixSum;
16     LLD Total;
17     LLD TotalMax;
18     info(int pre=-Inf,int suff=-Inf,int total=-Inf,int totalmax=-Inf):prefixSum(pre)
19     ,suffixSum(suff),Total(total),TotalMax(totalmax){};
20 };
21 info marge(const info &a,const info &b)
22 {
23     info ret;
24     ret.Total=a.Total+b.Total;

```

```

24     ret.prefixSum=max(a.prefixSum,a.Total+b.prefixSum);
25     ret.suffixSum=max(a.suffixSum+b.Total,b.suffixSum);
26     ret.TotalMax=MAX3(a.TotalMax,b.TotalMax,a.suffixSum+b.prefixSum);
27     return ret;
28 }
29 LLD arr[MN];
30 info Tree[3*MN];
31 void build(int L,int R,int indx)
32 {
33     if(L==R)
34     {
35         Tree[indx]=info(arr[L],arr[L],arr[L],arr[L]);
36         return;
37     }
38     int mid=(L+R)>>1;
39     build(L,mid,2*indx);
40     build(mid+1,R,2*indx+1);
41     Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
42 }
43 void update(int L,int R,int indx,int x,LLD val)
44 {
45     if(L==R)
46     {
47         Tree[indx]=info(val,val,val,val);
48         return;
49     }
50     int mid=(L+R)>>1;
51     if(x<=mid) update(L,mid,2*indx,x,val);
52     else update(mid+1,R,2*indx+1,x,val);
53     Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
54 }
55 info query(int L,int R,int indx,int x,int y)
56 {
57     if(L==x and y==R) return Tree[indx];
58     int mid=(L+R)>>1;
59     if(y<=mid) return query(L,mid,2*indx,x,y);
60     else if(x>mid) return query(mid+1,R,2*indx+1,x,y);
61     return marge(query(L,mid,2*indx,x,mid),query(mid+1,R,2*indx+1,mid+1,y));
62 }
63 int main()
64 {
65     #ifdef _ANICK_
66     //f_input;
67     #endif // _ANICK_
68     int N;
69     scanf("%d",&N);
70     for(int i=1;i<=N;i++)scanf("%lld",&arr[i]);
71     build(1,N,1);
72     int Q;
73     scanf("%d",&Q);
74     while(Q--)
75     {
76         int t,x,y;
77         scanf("%d %d %d",&t,&x,&y);
78         if(t) printf("%lld\n",query(1,N,1,x,y).TotalMax);
79         else update(1,N,1,x,y);
80     }
81     return 0;
82 }

```

Segment Tree Variant 3

```

1  /**
2  **Given a bracket sequence.
3  ** On a bracket word one can do the following operations:
4  **replacement — changes the i-th bracket into the opposite one
5  **check — if the word is a correct bracket expression
6  **/
7  #include<bits/stdc++.h>
8  using namespace std;
9  typedef long long LLD;

```

```

10 #define MAX 50005
11 struct info
12 {
13     int sum,sub;
14     info(int sum=0,int sub=0):sum(sum),sub(sub){};
15 };
16 info Tree[4*MAX];
17 char inp[MAX];
18 info marge(const info &L,const info &R)
19 {
20     info ret;
21     ret.sum= L.sum+R.sum;
22     ret.sub=L.sub;
23     ret.sub=min(ret.sub,L.sum+R.sub);
24     return ret;
25 }
26 void build(int L,int R,int indx)
27 {
28     if(L==R)
29     {
30         int x;
31         if(inp[L]=='(')x=1;
32         else x=-1;
33         Tree[indx]=info(x,x);
34         return;
35     }
36     int mid=(L+R)>>1;
37     build(L,mid,2*indx);
38     build(mid+1,R,2*indx+1);
39     Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
40 }
41 void update(int L,int R,int indx,int x)
42 {
43     if(L==R)
44     {
45         int x;
46         if(inp[L]=='(')x=1;
47         else x=-1;
48         Tree[indx]=info(x,x);
49         return;
50     }
51     int mid=(L+R)>>1;
52     if(x<=mid) update(L,mid,2*indx,x);
53     else update(mid+1,R,2*indx+1,x);
54     Tree[indx]=marge(Tree[2*indx],Tree[2*indx+1]);
55 }
56 info query(int L,int R,int indx,int x,int y)
57 {
58     if(L==x&&R==y) return Tree[indx];
59     int mid=(L+R)>>1;
60     if(y<=mid) return query(L,mid,2*indx,x,y);
61     else if(x>mid) return query(mid+1,R,2*indx+1,x,y);
62     else return marge(query(L,mid,2*indx,x,mid),query(mid+1,R,2*indx+1,mid+1,y));
63 }
64 int main()
65 {
66     int N,t=1;
67     while(scanf("%d",&N)==1)
68     {
69         scanf("%s",inp);
70         build(0,N-1,1);
71         int Q;
72         printf("Test %d:\n",t++);
73         scanf("%d",&Q);
74         while(Q--)
75         {
76             int x;
77             scanf("%d",&x);
78             if(x)
79             {

```

```

80         if(inp[x-1]=='(')inp[x-1]=')';
81         else inp[x-1]='(';
82         update(0,N-1,1,x-1);
83     }
84     else
85     {
86         info y=query(0,N-1,1,0,N-1);
87         if(y.sum==0&& y.sub>=0)printf("YES\n");
88         else printf("NO\n");
89     }
90 }
91 }
92 return 0;
93 }

```

1.2.5 Sliding Window RMQ

```

1  /**
2   * every K size window RMQ
3   * Calculate in O(N+K) time
4   */
5  #include<bits/stdc++.h>
6  using namespace std;
7  vector<int>SlidingRMQ(int *A,int N,int k)
8  {
9      /** Create a Double Ended Queue, Qi that will store indexes of array elements
10       * The queue will store indexes of useful elements in every window and it will
11       * maintain decreasing order of values from front to rear in Qi, i.e.,
12       * arr[Qi.front()] to arr[Qi.rear()] are sorted in increasing order
13       */
14      vector<int>MinWindow;
15      deque<int>Q;
16      int i;
17      /* Process first k (or first window) elements of array */
18      for(i=0;i<k;i++)
19      {
20          /// For very element, the previous largest elements are useless so
21          /// remove them from Qi
22          while(!Q.empty() and A[i]<=A[Q.back()])Q.pop_back();
23          Q.push_back(i);
24      }
25      /// Process rest of the elements, i.e., from arr[k] to arr[n-1]
26      while(i<N)
27      {
28          /// The element at the front of the queue is the smallest element of
29          /// previous window, so insert it result
30          MinWindow.push_back(A[Q.front()]);
31
32          /// Remove the elements which are out of this window
33          while(!Q.empty() and Q.front()<=i-k)Q.pop_front();
34
35          /// Remove all elements larger than the currently
36          /// being added element (remove useless elements)
37          while(!Q.empty() and A[i]<=A[Q.back()])Q.pop_back();
38
39          /// Add current element at the rear of Qi
40          Q.push_back(i);
41          i++;
42      }
43      /// insert the minimum element of last window
44      MinWindow.push_back(A[Q.front()]);
45      return MinWindow;
46  }
47  int main()
48  {
49      int A[]={100,10, -1, 2,-3,-4,10, 1,100,20};
50      vector<int>a=SlidingRMQ(A,10,2);
51      for(int i=0;i<a.size();i++)cout<<a[i]<<" ";
52      return 0;
53  }

```

1.2.6 Sparse Table

```
1  /**
2   Compute sparse table in O(NlogN)
3   query in O(1)
4   Ref link: https://www.topcoder.com/community/data-science/data-science-tutorials/range-minimum-query-and-lowest-common-ancestor/
5  */
6  #include<bits/stdc++.h>
7  using namespace std;
8  #define Max 10000005
9  int rmq[24][Max];
10 int A[Max];
11 void Compute_ST(int N)
12 {
13     for (int i = 0; i < N; ++i)rmq[0][i] = i;
14     for (int k = 1; (1 << k) < N; ++k)
15     {
16         for (int i = 0; i + (1 << k) <= N; i++)
17         {
18             int x = rmq[k - 1][i];
19             int y = rmq[k - 1][i + (1 << k - 1)];
20             rmq[k][i] = A[x] <= A[y] ? x : y;
21         }
22     }
23 }
24
25 int RMQ(int i, int j)
26 {
27     int k = log2(j-i);
28     int x = rmq[k][i];
29     int y = rmq[k][j - (1 << k) + 1];
30     return A[x] <= A[y] ? x : y;
31 }
32
33 int main()
34 {
35
36     return 0;
37 }
```

1.3 Heavy Light Decomposition

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  #define pp pair<int,int>
4  #define pb push_back
5  const int Max=10000;
6  struct info
7  {
8      int v, cost;
9      info(int v=0,int cost=0):v(v),cost(cost){};
10 };
11 vector<pp>edges;
12 vector<info>Graph[Max+5];
13 int Tree[5*Max+5],BaseArray[Max+5],SubTreeSize[Max+5];
14 int ChainHead[Max+5],ChainNum[Max+5],PosInBaseArray[Max+5],ChainNo;
15 int Level[Max+5],Parent[Max+5],SparseTable[Max+5][16];
16 int ptr;
17 void init(int N)
18 {
19     for (int i=0;i<=N;i++)
20     {
21         Graph[i].clear(),ChainHead[i]=-1;
22         for (int j=0;j<=15;j++)SparseTable[i][j]=-1;
23     }
24     edges.clear();
25     ptr=ChainNo=0;
26 }
```



```

27 void buildSegmentTree(int l, int r, int indx)
28 {
29     if (l==r)
30     {
31         Tree[indx]=BaseArray[l];
32         return;
33     }
34     int mid=(l+r)>>1;
35     int lindx=indx<<1;
36     int rindx=lindx|1;
37     buildSegmentTree(l, mid, lindx);
38     buildSegmentTree(mid+1, r, rindx);
39     Tree[indx]=max(Tree[lindx], Tree[rindx]);
40 }
41 void updateSegmentTree(int l, int r, int indx, int update_indx, int value)
42 {
43     if (l==r)
44     {
45         Tree[indx]=value;
46         return;
47     }
48     int mid=(l+r)>>1;
49     int lindx=indx<<1;
50     int rindx=lindx|1;
51     if (update_indx<=mid) updateSegmentTree(l, mid, lindx, update_indx, value);
52     else updateSegmentTree(mid+1, r, rindx, update_indx, value);
53     Tree[indx]=max(Tree[lindx], Tree[rindx]);
54 }
55 int querySegmentTree(int l, int r, int indx, int x, int y)
56 {
57     if (l>y || r<x) return 0;
58     if (x<=l&&y>=r) return Tree[indx];
59     int mid=(l+r)>>1;
60     int lindx=indx<<1;
61     int rindx=lindx|1;
62     int c1=0, c2=0;
63     if (x<=mid) c1=querySegmentTree(l, mid, lindx, x, y);
64     if (y>mid) c2=querySegmentTree(mid+1, r, rindx, x, y);
65     return max(c1, c2);
66 }
67 void dfs(int from, int u, int depth)
68 {
69     Level[u]=depth;
70     Parent[u]=from;
71     SubTreeSize[u]=1;
72     int sz=Graph[u].size();
73     for (int i=0; i<sz; i++)
74     {
75         int v=Graph[u][i].v;
76         if (v==from) continue;
77         dfs(u, v, depth+1);
78         SubTreeSize[u]+=SubTreeSize[v];
79     }
80 }
81 void sparseTable(int N)
82 {
83     for (int i=0; i<=N; i++) SparseTable[i][0]=Parent[i];
84     for (int j=1; (1<<j)<=N; j++)
85     {
86         for (int i=0; i<=N; i++)
87         {
88             if (SparseTable[i][j-1]!=-1)
89             {
90                 int a=SparseTable[i][j-1];
91                 SparseTable[i][j]=SparseTable[a][j-1];
92             }
93         }
94     }
95 }
96 int LCA(int p, int q)

```

```

97 {
98     if (Level[p] < Level[q]) swap(p, q);
99     int Log = log2(Level[p]) + 1;
100     for (int i = Log; i >= 0; i--)
101     {
102         if ((Level[p] - (1 < i)) >= Level[q]) p = SparseTable[p][i];
103     }
104     if (p == q) return p;
105     for (int i = Log; i >= 0; i--)
106     {
107         if (SparseTable[p][i] != -1 && SparseTable[p][i] != SparseTable[q][i])
108         {
109             p = SparseTable[p][i], q = SparseTable[q][i];
110         }
111     }
112     return Parent[p];
113 }
114 /**
115  * Actual HL-Decomposition part
116  * Initially all entries of chainHead[] are set to -1.
117  * So when ever a new chain is started, chain head is correctly assigned.
118  * As we add a new node to chain, we will note its position in the baseArray.
119  * In the first for loop we find the child node which has maximum sub-tree size.
120  * The following if condition is failed for leaf nodes.
121  * When the if condition passes, we expand the chain to special child.
122  * In the second for loop we recursively call the function on all normal nodes.
123  * chainNo++ ensures that we are creating a new chain for each normal child.
124  */
125 void heavyLightDecompositon(int from, int curNode, int cost)
126 {
127     if (ChainHead[ChainNo] == -1) ChainHead[ChainNo] = curNode; /// Assign chain head
128     ChainNum[curNode] = ChainNo;
129     PosInBaseArray[curNode] = ptr; /// Position of this node in baseArray which we
130     will use in Segtree
131     BaseArray[ptr++] = cost;
132     int sc = -1, nextCost;
133     int sz = Graph[curNode].size();
134     for (int i = 0; i < sz; i++) /// Loop to find special child
135     {
136         int v = Graph[curNode][i].v;
137         if (v == from) continue;
138         if (sc == -1 || SubTreeSize[sc] < SubTreeSize[v])
139         {
140             sc = v;
141             nextCost = Graph[curNode][i].cost;
142         }
143     }
144     if (sc != -1) heavyLightDecompositon(curNode, sc, nextCost); /// Expand the chain
145     for (int i = 0; i < sz; i++)
146     {
147         int v = Graph[curNode][i].v;
148         int cost = Graph[curNode][i].cost;
149         if (v == from || sc == v) continue;
150         ChainNo++;
151         heavyLightDecompositon(curNode, v, cost);
152     }
153 }
154 void updateTree(int ith, int val)
155 {
156     pp a = edges[ith];
157     int u = a.first, v = a.second;
158     int indx = PosInBaseArray[u];
159     if (Level[u] < Level[v]) indx = PosInBaseArray[v];
160     updateSegmentTree(0, ptr - 1, 1, indx, val);
161 }
162 /**
163  * query-up:
164  * It takes two nodes u and v, condition is that v is an ancestor of u
165  * We query the chain in which u is present till chain head, then move to next
166  * chain up

```

```

165  * We do that way till u and v are in the same chain, we query for that part of
166  chain and break
167  */
168  int queryUp(int u, int v)
169  {
170      if(u==v) return 0;
171      int uchain, vchain=ChainNum[v], ans=-1;
172      while(true)
173      {
174          uchain=ChainNum[u];
175          if(uchain==vchain)
176          {
177              if(u==v)          /// Both u and v are in the same chain, so we need to
178              query from u to v, update answer and break.
179              break;          /// We break because we came from u up till v, we are
180              done
181              ans=max(ans, querySegmentTree(0, ptr-1, 1, PosInBaseArray[v]+1,
182              PosInBaseArray[u]));
183              break;
184          }
185          int uchainhead=ChainHead[uchain];
186          ans=max(ans, querySegmentTree(0, ptr-1, 1, PosInBaseArray[uchainhead],
187          PosInBaseArray[u]));
188          /// Above is call to segment tree query function. We do from
189          chainHead of u till u. That is the whole chain from
190          u=Parent[uchainhead];
191      }
192      return ans;
193  }
194  int queryTree(int u, int v)
195  {
196      int lca=LCA(u, v);
197      return max(queryUp(u, lca), queryUp(v, lca));
198  }
199  int main()
200  {
201      int test;
202      cin>>test;
203      while(test--)
204      {
205          int N;
206          cin>>N;
207          init(N);
208          for(int i=0; i<N-1; i++)
209          {
210              int u, v, c;
211              cin>>u>>v>>c;
212              u--, v--;
213              Graph[u].pb(info(v, c));
214              Graph[v].pb(info(u, c));
215              edges.pb(pp(u, v));
216          }
217          dfs(-1, 0, 0);
218          sparseTable(N);
219          heavyLightDecompositon(-1, 0, -1);
220          buildSegmentTree(0, ptr-1, 1);
221          string ch;
222          int x, y;
223          while(true)
224          {
225              cin>>ch;
226              if(ch[0]=='D') break;
227              cin>>x>>y;
228              if(ch[0]=='Q') printf("%d\n", queryTree(x-1, y-1));
229              else if(ch[0]=='C') updateTree(x-1, y);
230          }
231      }
232      return 0;
233  }

```

1.4 Ternary Bit Mask

```
1
2 int more_bit[10];
3 int get_bit(int mask , int pos)
4 {
5     return (mask / more_bit[pos]) % 3;
6 }
7 int set_bit(int mask, int pos , int bit)
8 {
9     int tmp = (mask / more_bit[pos]) % 3;
10    mask -= tmp * more_bit[pos];
11    mask += bit * more_bit[pos];
12    return mask;
13 }
14 void init(void)
15 {
16     more_bit[0] = 3;
17     for(int i = 1; i < 10; i++) more_bit[i] = 3 * more_bit[i - 1];
18 }
```

Chapter 2

Graph Theory

2.1 DFS

2.1.1 Bicoloring

```
1  ///color will be initial with -1
2  int color[20005];
3  bool dfs(int u, int c)
4  {
5      if (color[u]==c) return true;
6      if (color[u]==(1-c)) return false;
7      color[u]=c;
8      bool ret=true;
9      for (auto v: graph[u]) ret&=dfs(v, 1-c);
10     return ret;
11 }
```

2.1.2 Cycle Finding

```
1  int color[20005];
2  bool dfs(int u)
3  {
4      color[u]=GREY;
5      bool no_cycle=true;
6      for (auto v: graph[u])
7      {
8          if (color[v]==WHITE)
9          {
10             no_cycle=dfs(v);
11          }
12          else if (color[v]==GREY) return false;
13      }
14      color[u]=BLACK;
15      return no_cycle;
16 }
```

2.2 Topological Sort

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  #define WHITE 0
4  #define GREY 1
5  #define BLACK 2
6  vector<int> graph[100005];
7  vector<int> ans;
8  int visit[100005];
9  bool dfs(int u)
10 {
11     visit[u]=GREY;
12     bool no_cycle=true;
```

```

13     int sz=graph[u].size();
14     for(int i=0;i<sz;i++)
15     {
16         int v=graph[u][i];
17         if(visit[v]==WHITE)
18         {
19             no_cycle=dfs(v);
20         }
21         else if(visit[v]==GREY) return false;
22     }
23     visit[u]=BLACK;
24     ans.push_back(u);
25     return no_cycle;
26 }
27 bool topsort(int N)
28 {
29     ans.clear();
30     memset(visit, false, sizeof(visit));
31     int no_cycle=true;
32     for(int i=0;i<N;i++)
33     {
34         if(visit[i]==WHITE) no_cycle&=dfs(i);
35     }
36     return no_cycle;
37 }
38 int main()
39 {
40     return 0;
41 }

```

2.3 Havel Hakimi

```

1  /**
2   * Given N degree d1,d2,d3.....dn. Is it possible to make a graph which have no
3   * cycle and
4   * different two node will be connected with one Edge?
5   */
6  #include<stdio.h>
7  #include<queue>
8  #include<vector>
9  using namespace std;
10 int main()
11 {
12     int N;
13     while(scanf("%d",&N) and N)
14     {
15         priority_queue<int> Q;
16         bool Ok=true;
17         int Odd_Node=0;
18         for(int i=0;i<N;i++)
19         {
20             int x;
21             scanf("%d",&x);
22             if(x>=N or x<0) Ok=false;
23             Odd_Node+=(x%2);
24             Q.push(x);
25         }
26         Ok&=(Odd_Node%2==0); //Handshaking Theorem
27         for(int i=0;i<N and Ok;i++)
28         {
29             int k=Q.top();
30             Q.pop();
31             vector<int> v;
32             for(int j=0;j<k and Ok;j++)
33             {
34                 int x=Q.top();
35                 Q.pop();
36                 x--;
37                 Ok&=(x>=0);

```

```

38         v.push_back(x);
39     }
40     for(int j=0;j<k and Ok;j++)
41     {
42         Q.push(v[j]);
43     }
44 }
45 if(Ok)printf("Possible\n");
46 else printf("Not possible\n");
47 }
48 return 0;
49 }

```

2.4 Articulation Point/Bridge

2.4.1 Find Articulation Point:

```

1  vector<int>Graph[10000];
2  bool visit[10000];
3  int arti[100000];
4  int discover[100000],Back[100000];
5  int predfn;
6  int source;
7  int child_of_root;
8  int cnt=0;
9  void reset()
10 {
11     memset(visit,false,sizeof(visit));
12     memset(arti,false,sizeof(arti));
13     predfn=child_of_root=0;
14 }
15 void articulation(int v)
16 {
17     visit[v]=true;
18     predfn++;
19     discover[v]=Back[v]=predfn;
20     for(int i=0;i<Graph[v].size();i++)
21     {
22         int w=Graph[v][i];
23         if(!visit[w])
24         {
25             articulation(w);
26             Back[v]=min(Back[v],Back[w]);
27             if(Back[w]>=discover[v]&&v!=source)
28             {
29                 arti[v]=true;
30             }
31             else if(v==source)
32             {
33                 child_of_root++;
34                 if(child_of_root==2)
35                 {
36                     arti[v]=true;
37                 }
38             }
39         }
40         else
41         {
42             Back[v]=min(Back[v],discover[w]);
43         }
44     }
45 }

```

2.4.2 Find Bridge version 1:

```

1  vector<int>Graph[200];
2  int Back[205],Discover[205];
3  bool visit[205];
4  bool bridge[205][205];

```

```

5 int brcount;
6 void reset(int n)
7 {
8     for(int i=0;i<=n;i++)Graph[i].clear();
9     memset(visit,false,sizeof(visit));
10    memset(bridge,false,sizeof(false));
11    brcount=0;
12 }
13 void find_bridge(int u, int parent, int depth)
14 {
15     visit[u] = true;
16     Discover[u] = Back[u] = depth;
17
18     for (int i=0 ; i<Graph[u].size() ; i++)
19     {
20         int v = Graph[u][i];
21
22         if (visit[v] && v!=parent)
23         {
24             Back[u] = min(Back[u],Discover[v]);
25         }
26         if (!visit[v])
27         {
28             find_bridge(v, u, depth+1);
29             Back[u] = min(Back[u],Back[v]);
30             if (Back[v]>Discover[u])
31             {
32                 brcount++;
33                 bridge[u][v] = bridge[v][u] = true;
34             }
35         }
36     }
37 }
38 }

```

2.4.3 Find Bridge version 2:

```

1 void find_bridge(int node, int parent)
2 {
3     discovery_time[node] = bedge[node] = ++T;
4     int to, i, connected = adj[node].size();
5     for(i = 0; i < connected; i++)
6     {
7         to = adj[node][i];
8         if(to == parent) continue;
9         if(!discovery_time[to])
10        {
11            printf("%d %d\n", node, to);
12            find_bridge(to, node);
13            bedge[node] = min(bedge[node], bedge[to]);
14            if(bedge[to] > discovery_time[node]) printf("%d %d\n", to, node);
15        }
16        else if(discovery_time[node] > discovery_time[to])
17        {
18            printf("%d %d\n", node, to);
19            bedge[node] = min(bedge[node], discovery_time[to]);
20        }
21    }
22 }

```


Chapter 3

Flow networks/ matching

3.1 Max Flow

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 #define pb push_back
4 #define MN 1000
5 typedef vector< vector<int> > vint2D;
6 const int inf=(1<<29);
7 vint2D graph;
8 int Cost[MN][MN];
9 int parent[MN+5];
10 int flow;
11 void init(int N)
12 {
13     graph=vint2D(N);
14     memset(Cost,0,sizeof(Cost));
15 }
16 void AddEdge(int u,int v,int cost)
17 {
18     graph[u].pb(v);
19     graph[v].pb(u);
20     Cost[u][v]+=cost;
21     Cost[v][u]+=cost;
22 }
23 bool augmenting_path(int source,int sink)
24 {
25     memset(parent,-1,sizeof(parent));
26     queue<int>Q;
27     Q.push(source);
28     while(!Q.empty())
29     {
30         int u=Q.front();
31         Q.pop();
32         int sz=graph[u].size();
33         for(int i=0;i<sz;i++)
34         {
35             int v=graph[u][i];
36             if(parent[v]==-1 and Cost[u][v]>0)
37             {
38                 parent[v]=u;
39                 Q.push(v);
40                 if(v==sink) return true;
41             }
42         }
43     }
44     return false;
45 }
46 void path(int v,int source)
47 {
48     int u=parent[v];
49     flow=min(flow, Cost[u][v]);
```

```

50     if (source!=u) path(u, source);
51     Cost[u][v]-=flow;
52     Cost[v][u]+=flow;
53     return;
54 }
55 int max_flow(int source, int sink)
56 {
57     int ret=0;
58     while(augmenting_path(source, sink))
59     {
60         flow=inf;
61         path(sink, source);
62         ret+=flow;
63     }
64     return ret;
65 }
66 int main()
67 {
68     int test;
69     scanf("%d",&test);
70     while(test--)
71     {
72         int P,S,C,M;
73         scanf("%d %d %d %d",&P,&S,&C,&M);
74         init(P+S+5);
75         int superSource=0, SuperSikn=P+S+1;
76         for(int i=1; i<=P; i++) AddEdge(superSource, i, 1);
77         for(int i=1; i<=S; i++) AddEdge(P+1, SuperSikn, C);
78         for(int i=0; i<M; i++)
79         {
80             int x,y;
81             scanf("%d %d",&x,&y);
82             AddEdge(x, P+y, (1<30));
83         }
84         printf("%d\n", max_flow(superSource, SuperSikn));
85     }
86     return 0;
87 }

```

Chapter 4

Dynamic programming

4.1 Edit Distance

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int dp[88][88];
4 int N,M,step;
5 char S1[88],S2[88];
6 int solve(int i,int j)
7 {
8     if(i==N and j==M) return 0;
9     if(i==N) return M-j;
10    if(j==M) return N-i;
11    int &ret=dp[i][j];
12    if(ret!=-1) return ret;
13    ret=(1<<28);
14    if(S1[i]==S2[j]) ret=solve(i+1,j+1);
15    else
16    {
17        ret=min(ret,solve(i,j+1)+1);
18        ret=min(ret,solve(i+1,j)+1);
19        ret=min(ret,solve(i+1,j+1)+1);
20    }
21    return ret;
22 }
23 void pathPrint(int i,int j,int del,int ins,int st)
24 {
25     if(i==N&&j==M) return ;
26     if(i==N)
27     {
28         for(int k=j;k<M;k++,i++)
29         {
30             printf("%d Insert %d,%c\n",st++,i-del+1+ins,S2[k]);
31         }
32         return ;
33     }
34     if(j==M)
35     {
36         for(;i<N;i++)
37         {
38             printf("%d Delete %d\n",st++,i-del+1+ins);
39             del++;
40         }
41         return ;
42     }
43     int ret = solve(i,j);
44     int tmp;
45     if(S1[i]==S2[j])
46     {
47         tmp=solve(i+1,j+1);
48         if(ret==tmp)
49         {
```

```

50         pathPrint(i+1,j+1,del,ins,st);
51         return ;
52     }
53 }
54 tmp=solve(i,j+1)+1;
55 if(tmp==ret)
56 {
57     printf("%d Insert %d,%c\n",st,i-del+1+ins,S2[j]);
58     pathPrint(i,j+1,del,ins+1,st+1);
59     return ;
60 }
61 tmp=solve(i+1,j)+1;
62 if(tmp==ret)
63 {
64     printf("%d Delete %d\n",st,i-del+1+ins);
65     pathPrint(i+1,j,del+1,ins,st+1);
66     return ;
67 }
68 tmp=solve(i+1,j+1)+1;
69 if(tmp==ret)
70 {
71     printf("%d Replace %d,%c\n",st,i-del+1+ins,S2[j]);
72     pathPrint(i+1,j+1,del,ins,st+1);
73     return ;
74 }
75 return ;
76 }
77 int main()
78 {
79     bool New=false;
80     while(gets(S1))
81     {
82         gets(S2);
83         if(New) printf("\n");
84         New=true;
85         N=strlen(S1);
86         M=strlen(S2);
87         memset(dp,-1,sizeof(dp));
88         step=solve(0,0);
89         printf("%d\n",step);
90         pathPrint(0,0,0,0,1);
91     }
92     return 0;
93 }

```

Chapter 5

Strings

5.1 KMP

Tutorial

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 char TXT[10000000], ptr[10000000];
4 vector<int> compute_prefix(const char *p)
5 {
6     int m=strlen(p+1);
7     vector<int> prefix(m+1);
8     prefix[1]=0;
9     int k=0;
10    for(int i=2; i<=m; i++)
11    {
12        while(k>0 and p[k+1]!=p[i]) k=prefix[k];
13        if(p[k+1]==p[i]) k=k+1;
14        prefix[i]=k;
15    }
16    return prefix;
17 }
18 vector<int> KMP_match(const char *txt, const char *ptrn)
19 {
20     int n=strlen(txt+1);
21     int m=strlen(ptrn+1);
22     vector<int> Prefix=compute_prefix(ptrn);
23     vector<int> Match_position;
24     int q=0;
25     for(int i=1; i<=n; i++)
26     {
27         while(q>0 and ptrn[q+1]!=txt[i]) q=Prefix[q];
28         if(ptrn[q+1]==txt[i]) q=q+1;
29         if(q==m)
30         {
31             Match_position.push_back(i-m);
32             q=Prefix[q];
33         }
34     }
35     return Match_position;
36 }
37 int main()
38 {
39     scanf("%s %s", TXT+1, ptr+1);
40     vector<int> Match_position=KMP_match(TXT, ptr);
41     for(int i=0; i<Match_position.size(); i++)
42     {
43         if(!i) printf("%d", Match_position[i]);
44         else printf(" %d", Match_position[i]);
45     }
46     return 0;
47 }
```

5.2 Aho Corasick

5.2.1 Aho Corasick with Dynamic Trie

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 #define Max 26
4 int getID(char c)
5 {
6     return c>='a'?c-'a':c-'A';
7 }
8 char inp[1000005];
9 char text[1000005];
10 int ans[5000];
11 map<string,int>Map;
12 vector<int>v;
13 struct Trie
14 {
15     Trie *next[26],*fail;
16     int stringMap;
17     Trie()
18     {
19         stringMap=0;
20         for(int i=0;i<Max;i++)next[i]=NULL;
21         fail=NULL;
22     }
23 };
24 Trie *root;
25 void Insert(const char *str,int M)
26 {
27     Trie *p=root;
28     for(int i=0;str[i];i++)
29     {
30         int id=getID(str[i]);
31         if(p->next[id]==NULL)p->next[id]=new Trie();
32         p=p->next[id];
33     }
34     p->stringMap=M;
35 }
36 void computeFailure()
37 {
38     Trie *u,*prefix;
39     queue<Trie*>Q;
40     Q.push(root);
41     while(!Q.empty())
42     {
43         u=Q.front(); //Take a new node
44         Q.pop();
45         for(int i=0;i<Max;i++)
46         {
47             if(u->next[i]!=NULL) //select fail position of ith node of parent u
48             {
49                 prefix=u->fail; // Going to u node fail position/ prefix position
50                 while(prefix!=NULL)
51                 {
52                     if(prefix->next[i]!=NULL) //if match found
53                     {
54                         u->next[i]->fail=prefix->next[i];
55                         break;
56                     }
57                     prefix=prefix->fail; // match not found, going to upper child
58                     prefix position
59                 }
60                 if(prefix==NULL)u->next[i]->fail=root;
61                 Q.push(u->next[i]);
62             }
63         }
64     }
65 void AhoCorasick(const char *str)
```

```

66 {
67     Trie *p=root;
68     int cnt=0;
69     for (int i=0;str[i];i++)
70     {
71         int id=getID(str[i]);
72         while(p->next[id]==NULL&&p!=root)p=p->fail, cnt++;
73         if(p->next[id]!=NULL)p=p->next[id];
74         Trie *tp=p;
75         while(tp!=root)
76         {
77             cnt++;
78             if(tp->stringMap>0)ans[tp->stringMap]++;
79             tp=tp->fail;
80         }
81     }
82 }
83 void Delete(Trie *u)
84 {
85     if(u==NULL) return;
86     for (int i=0;i<Max;i++)Delete(u->next[i]);
87     delete u;
88 }
89
90 int main()
91 {
92     int test;
93     scanf("%d",&test);
94     for (int t=1;t<=test;t++)
95     {
96         Map.clear();
97         v.clear();
98         memset(ans,0,sizeof(ans));
99         root=new Trie();
100         int N;
101         scanf("%d",&N);
102         scanf("%s",text);
103         int cnt=1;
104         for (int i=0;i<N;i++)
105         {
106             scanf("%s",inp);
107             if(Map.find(inp)==Map.end())Map[inp]=cnt++;
108             Insert(inp,Map[inp]);
109             v.push_back(Map[inp]);
110         }
111         computeFailure();
112         AhoCorasick(text);
113         printf("Case %d:\n",t);
114         for (int i=0;i<N;i++)
115         {
116             printf("%d\n",ans[v[i]]);
117         }
118         Delete(root);
119     }
120     return 0;
121 }

```

5.2.2 Aho Corasick with Static Trie

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define root 0
4 #define NuLL -1
5 #define Max 248878
6 #define MC 26
7 int ans[10000];
8 char text[1000005];
9 char inp[100000];
10 map<string,int>Map;
11 vector<int> v;

```

```

12 int getID(const char c)
13 {
14     return c>='a'?c-'a':c-'A';
15 }
16 struct Trie
17 {
18     struct node
19     {
20         int Next[26], fail;
21         int stringMap;
22         void clear()
23         {
24             memset(Next, -1, sizeof(Next));
25             fail=-1;
26             stringMap=0;
27         }
28     }T[Max];
29     int ptr;
30     void clear()
31     {
32         ptr=1;
33         T[0].clear();
34     }
35     void Insert(char *str, int M)
36     {
37         int p=0;
38         for(int i=0; str[i]; i++)
39         {
40             int id=getID(str[i]);
41             if(T[p].Next[id]==-1)
42             {
43                 T[p].Next[id]=ptr;
44                 T[ptr++].clear();
45             }
46             int q=p;
47             p=T[p].Next[id];
48             if(p<0)
49             {
50                 while(1);
51             }
52         }
53         T[p].stringMap=M;
54     }
55     void ComputeFailure()
56     {
57         queue<int>Q;
58         Q.push(root);
59         int u, prefix;
60         int cnt=0, cnt2=0;
61         while(!Q.empty())
62         {
63             u=Q.front();
64             Q.pop();
65             for(int i=0; i<MC; i++)
66             {
67                 if(T[u].Next[i]!=NULL)
68                 {
69                     int now=T[u].Next[i];
70                     prefix=T[u].fail;
71                     while(prefix!=NULL)
72                     {
73                         cnt2++;
74                         if(T[prefix].Next[i]!=NULL)
75                         {
76                             T[now].fail=T[prefix].Next[i];
77                             break;
78                         }
79                         prefix=T[prefix].fail;
80                     }
81                     if(prefix==NULL)T[now].fail=root;

```



```

82         Q.push(now);
83     }
84 }
85 }
86 }
87 };
88 void AhoCorasick(const Trie &A, const char *str)
89 {
90     int p=root;
91     int cnt1=0,cnt2=0;
92     for(int i=0;str[i];i++)
93     {
94         int id=getID(str[i]);
95         while(A.T[p].Next[id]==NULL&&p!=root)p=A.T[p].fail;
96         if(p!=NULL&&A.T[p].Next[id]!=NULL)p=A.T[p].Next[id];
97         int tp=p;
98         while(tp!=root)
99         {
100             if(A.T[tp].stringMap>0)ans[A.T[tp].stringMap]++;
101             tp=A.T[tp].fail;
102         }
103     }
104 }
105 Trie A;
106 int main()
107 {
108     #ifdef _ANICK_
109         freopen("input.txt","r",stdin);
110     #endif // _ANICK_
111     int test;
112     scanf("%d",&test);
113     for(int t=1;t<=test;t++)
114     {
115         Map.clear();
116         v.clear();
117         memset(ans,0,sizeof(ans));
118         A.clear();
119         int N;
120         scanf("%d",&N);
121         scanf("%s",text);
122         int cnt=1;
123         for(int i=0;i<N;i++)
124         {
125             scanf("%s",inp);
126             if(Map.find(inp)==Map.end())Map[inp]=cnt++;
127             A.Insert(inp,Map[inp]);
128             v.push_back(Map[inp]);
129         }
130         A.ComputeFailure();
131         AhoCorasick(A,text);
132         printf("Case %d:\n",t);
133         for(int i=0;i<N;i++)
134         {
135             printf("%d\n",ans[v[i]]);
136         }
137     }
138     return 0;
139 }

```

5.3 Manacher's Algorithm

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 string s, t;
4 char str[1000005];
5 void prepare_string()
6 {
7     int i;
8     t="#";
9     for(i=0;i<s.size();i++)

```

```

10     t+=s[i],t+=">";
11     t+=">";
12 }
13
14 int manacher()
15 {
16     prepare_string();
17     int P[t.size()],c=0,r=0,i,i_mirror,n=t.size()-1;
18     for(i=1;i<n;i++)
19     {
20         i_mirror=(2*c)-i;
21         P[i]=r>i?min(r-i,P[i_mirror]):0;
22         while(t[i+1+P[i]]==t[i-1-P[i]]) P[i]++;
23         if(i+P[i]>r)
24         {
25             c=i;
26             r=i+P[i];
27         }
28     }
29     return *max_element(P+1,P+n);
30 }
31 int main()
32 {
33     int kase=1;
34     while(scanf("%s",str)&&str[0]!='E')
35     {
36         s=str;
37         printf("Case %d: %d\n",kase++,manacher());
38     }
39     return 0;
40 }

```

Chapter 6

Computational geometry

Chapter 7

Math

7.1 Reduce Ratio

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

```
1 int main()
2 {
3     int A,B,x,y;
4     cin>>A>>B>>x>>y;
5     int g=__gcd(x,y);
6     x/=g,y/=g;
7     int t=min(A/x,B/y);
8     cout<<x*t<<" "<<y*t<<endl;
9     return 0;
10 }
```

7.2 Floyd's Cycle Finding algorithm

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 #define pp pair<int,int>
4 int Z,L,M,I;
5 int f(int L)
6 {
7     return (Z*L+I)%M;
8 }
9 pp CycleFinding()
10 {
11     ///L here initial seed
12     int hare,tortoise,lambda,meu;
13     bool cyclefind=false;
14     hare=tortoise=L;
15     while(!cyclefind)
16     {
17         tortoise=f(tortoise);
18         hare=f(hare);
19         hare=f(hare);
20         if(hare==tortoise) cyclefind=true;
21     }
22     hare=L;
23     meu=0;
24     while(hare!=tortoise)
25     {
26         meu++;
27         hare=f(hare);
28         tortoise=f(tortoise);
29     }
30     int i=0;
31     hare=L;
```

```

32     while (i<=meu)
33     {
34         i++;
35         hare=f(hare);
36     }
37     tortoise=f(hare);
38     lambda=1;
39     while (hare!=tortoise)
40     {
41         tortoise=f(tortoise);
42         lambda++;
43     }
44     return {meu,lambda}; ///meu is starting index and lambda is cycle length
45 }
46 int main()
47 {
48     int t=1;
49     while (scanf("%d %d %d %d",&Z,&I,&M,&L) and (Z or I or M or L))
50     {
51         pp a=CycleFinding();
52         cout<<"Cycle starts from index "<<a.first<<"\nCycle length is "<<a.second<<
53         endl;
54     }
55     return 0;
56 }

```

Chapter 8

Number Theory

8.1 NCR

8.1.1 Lucas Theorem

```
1  /**
2   * Fine NCR % M when N C M are large number.
3   * using Lucas theorem.
4   */
5  #include <bits/stdc++.h>
6  using namespace std;
7  typedef long long LLD;
8  LLD mod=1000003;
9  LLD big_mod(LLD n,LLD p,LLD m)
10 {
11     if (p==0) return (LLD) 1;
12     LLD x=big_mod(n,p/2,m);
13     x=(x*x)%m;
14     if (p&1)x=(x*n)%m;
15     return x;
16 }
17 LLD inverse_modulo(LLD t,LLD m)
18 {
19     return big_mod(t,m-2,m);
20 }
21 LLD combi(LLD n, LLD k,LLD m)
22 {
23     if (n<k)
24         return 0;
25     if (n-k<k)
26         return combi(n,n-k,m);
27     LLD i,p=1,t=1;
28     for (i=n-k+1; i<=n; i++)
29         p=(p*i)%m;
30     for (i=1; i<=k; i++)
31         t=(t*i)%m;
32     return (p*inverse_modulo(t,m))%m;
33 }
34 LLD lucas(LLD n, LLD k, LLD m)
35 {
36     if (n<k)
37         return 0;
38     if (k==0 || n==k)
39         return 1;
40     return (lucas(n/m,k/m,m)*combi(n%m,k%m,m))%m;
41 }
42 int main()
43 {
44     return 0;
45 }
```